Czech University of Life Sciences Prague

Faculty of Economics and Management

Department of Systems Engineering

Master's Thesis

Vacant taxi routing in Markov Decision Process (MDP)

Bc. Nurbulat Shektbayev

CZECH UNIVERSITY OF LIFE SCIENCES PRAGUE

Faculty of Economics and Management

DIPLOMA THESIS ASSIGNMENT

Bc. Nurbulat Shektbayev

Systems Engineering and Informatics
Informatics

Thesis title

Vacant taxi routing in Markov Decision Process (MDP)

Objectives of thesis

Main objectives of the thesis are:
1) Replicate the vacant taxi routing problem by using Markov Decisions Process (MDP). Set up and define action and state spaces, transition probabilities and reward function (by using real historical data).
2) Compare and evaluate different types of algorithms that are used to solve Markov Decisions Process (MDP) such as: SARSA, value iteration, policy iteration and Q-learning.
3) Evaluate the algorithms by using performance metrics such as: total travel and waiting time, revenue and percentage of taxi driver's productive time.

Methodology

THe first part of the thesis will consist of literature review on the relevant topics and will be based on the monographies and scientific papers.

In the parctical part, the following steps will be carried out:

• Formulate problem and define the key variables and parameters that will be used in the mathematical model.

• Mathematical modeling: Develop a mathematical model of the problem using MDP.

• Data collection: Obtain and format open source data from the different statistics offices or companies around the world.

• Data analysis: Use descriptive statistics to summarize the data (such as mean, median, and standard deviation)

• Simulation: Simulate different scenarios to evaluate the performance of the MDP algorithms by using Python and Python libraries such as: NumPy, Pandas and Matplotlib.

• Algorithm implementation: Implement RL algorithms to solve MDP such as: Value Iteration, Policy Iteration, Q-learning and SARSA.

• Compare the results obtained from different algorithms and evaluate their performance in terms of efficiency and effectiveness.

• Conduct sensitivity analysis to test the robustness of the results.

• Discuss the limitations of the proposed approach and the assumptions made in the modeling process.

The proposed extent of the thesis

80

Keywords

MDP, machine learning, data processing, algorithms, markov decision process, dynamic programming, data

---

Recommended information sources

A Markov Decision Process Approach to Vacant Taxi Routing with E-hailing (Xinlian Yu, Xianbiao Hu, Hyoshin Park) DOI: 10.1016/j.trb.2018.12.013

Constrained Markov Decision Processes (Eitan Altman) ISBN 9780849303821

Markov Decision Processes: Discrete Stochastic Dynamic Programming (Martin L. Puterman) ISBN 978-0471727828

Markov Decision Processes in Practice (Richard J. Boucherie (Editor), Nico M. van Dijk) ISBN 978-3319477640

Reinforcement Learning: An Introduction second edition (Richard S. Sutton and Andrew G. Barto) ISBN 978-0262039246

Expected date of thesis defence

2022/23 SS – FEM

The Diploma Thesis Supervisor

Ing. Robert Hlavatý, Ph.D.

Supervising department

Department of Systems Engineering

Electronic approval: 23. 11. 2023	Electronic approval: 23. 11. 2023
doc. Ing. Tomáš Šubrt, Ph.D.	**doc. Ing. Tomáš Šubrt, Ph.D.**
Head of department	Dean

Prague on 30. 11. 2023

Declaration

I declare that I have worked on my master's thesis titled " Vacant taxi routing in Markov Decision Process (MDP)" by myself and I have used only the sources mentioned at the end of the thesis. As the author of the master's thesis, I declare that the thesis does not break any copyrights.

In Prague on 30.11.2023 ____________________________

Acknowledgement

I would like to say that I am grateful for my advisor Professor Robert Hlavaty for his support, guidance and other useful things and ideas that he has provided. Expertise of Prof. Hlavaty and his insights have been important in process of shaping my ideas and thesis.

I would also like to express my gratitude to the faculty members of the Czech University of Life Science in Prague, for resources and support that they have contributed.

I am also thankful to authors, other individuals and organizations that provided access to the papers and open-source projects data that were used in my research.

Finally, I am grateful to my family, my wife Albina and my son Azim for their constant support and encouragement during process of writing my thesis.

Without the support of above individuals, I could not write and finish my thesis.

Vacant taxi routing in Markov Decision Process (MDP)

Abstract

This research explores approach on how make efficient and profitable taxi routes by implementing a method called Markov Decision Processes (MDP). The work starts with reviewing essential knowledge on reinforcement learning and the MDP in order to build a strong foundation for the future practical part. The main task is to find out how by using the MDP the researcher can improve and get best profitable and efficient taxi routes. This involved analyze of a huge amount of taxi data taken from the New York City. As this allows to understand the different situations a taxi could encounter (states) and the choices a driver could make (actions).

The research then continues with testing four different computer algorithms like Value iteration, Policy iteration, Q-learning and SARSA. These algorithms represent different approaches and strategies that a taxi driver can use to decide where to move from the current location. Each of algorithm was implemented and test by using a programming language Python. Effectiveness was determined by looking at which destinations would be more profitable compare with costs, distance of trips and how much time taxi spent on finishing a ride.

The research can be useful for taxi drivers and taxi companies, public transportation organizations etc. It shows new ways to respond to passenger demand, which could help drivers earn more profit and spend less time waiting for passengers. This thesis also discusses some possible limitations of the approach and suggests ideas for further research and improvements. This work can be beneficial as it can help improve taxi services in cities by making them more efficient and profitable.

Keywords: MDP, machine learning, data processing, algorithms, markov decision process, dynamic programming, data

Trasování prázdného taxi v Markovově rozhodovacím procesu (MDP)

Abstrakt

Tento výzkum zkoumá přístup, jak vytvořit efektivní a ziskové trasy pro taxíky pomocí metody zvané Markovovy rozhodovací procesy (MDP). Práce začíná přehledem základních znalostí o posilovacím učení a MDP, aby byl položen pevný základ pro praktickou část. Hlavním úkolem je zjistit jak lze pomocí MDP zlepšit a získat nejziskovější a nejefektivnější trasy pro taxíky. To zahrnovalo analýzu obrovského množství dat o taxících z New Yorku. To je umožnilo pochopit různé situace, kterým může taxík čelit (stavy) a rozhodnutí, která může řidič učinit (akce).

Výzkum pokračuje testováním čtyř různých počítačových algoritmů jako jsou: Value iteration, Policy iteration, Q-learning a taky SARSA. Tyto algoritmy představují různé přístupy a strategie, které může taxikář použít aby se rozhodl, kam se z aktuálního místa přesune. Každý z algoritmů byl implementován a testován v programovacím jazyce Python. Účinnost byla určena zkoumáním, které cíle by byly ziskovější ve srovnání s náklady, vzdáleností cest a časem, který taxík strávil dokončením jízdy.

Výzkum může být užitečný pro řidiče taxíků a taxi společnosti, organizace veřejné dopravy atd. Ukazuje nové způsoby, jak reagovat na poptávku cestujících, což by mohlo pomoci řidičům vydělávat více peněz a strávit méně času čekáním na pasažéry. Tato práce také diskutuje o možných omezeních přístupu a navrhuje nápady pro další výzkum a zlepšení. Tato práce může být prospěšná, protože může pomoci zlepšit služby taxíků ve městech tím, že je učiní efektivnějšími a ziskovějšími.

Klíčová slova: mdp, strojové učení, zpracování dat, algoritmy, markovův rozhodovací proces, dynamické programování,data

Table of content

1 Introduction

It is hard not to mention changes that happening in today's world where IT, artificial intelligence (AI) and robotics become significant or even crucial in many areas of our lives. Starting with such important one as healthcare where artificial intelligence and robots can assist doctors in making more accurate diagnoses, managing data of patients or performing surgical operations and continuing with manufacturing, finance, agriculture, transportation and so on.

As doing research about these fields, one area dragged the attention and interest. Transportation area is critical for our modern society as it allows us to travel and connect with our families and friends, transport and access goods and services, sustain certain jobs which benefit economy. It is hard to imagine how we would live without transportation industry.

As it has been mentioned before, artificial intelligence is heavily involved in the transportation area where it plays very important role. Artificial intelligence is crucial when we talk about for example autonomous vehicles or optimizing traffic flows in cities. All of it was interesting but there was desire to understand more of background by which artificial intelligence works. This was inspiration to read about Machine Learning (ML), a branch of artificial intelligence that studies how machines can learn and absorb knowledge from data and make decision based on it. Sure, there are many other subfields which are included in AI such as: natural language processing (NLP), computer vision, cognitive computing and so on.

After looking at several topics, it was decided that the master thesis should focus on unique and precise topic. For this reason, it has been chosen Markov Decision Process (MDP) that is one of main components of Reinforcement learning (RL) which is also subfield of ML. Markov Decision Processes (MDPs) offer a strong and powerful framework for analysing and optimizing decision-making problems. The optimization of vacant taxi route is one of the specific and interesting usages of MDP. It allows to learn the best strategies for finding passengers in various locations and at different times of the day. We can do it by modelling the taxi driver's decision-making process as MDP and apply reinforcement learning algorithms to it. This area is very interesting and exciting.

In summary I believe that this research can help and bring something valuable to more efficient transportation systems that could benefit our society and individuals.

Problem statement and research questions

The taxi industry is an important part of our modern urban transportation system; however it faces different types of challenges such as inefficient routing that cause longer travel time periods, high fuel usage and costs (it is more crucial taking into consideration recent energy crises) and negative passengers experience. Even though Markov Decision Process (MDP) can be used to optimize taxi routing, there are not so many research papers that have been done on its application to vacant taxi routing. That's why the master thesis aims to explore MDP and how it can be used to develop an approach and way for vacant taxi route issue and evaluate effectiveness and efficiency through simulation experiments with Python environment.

Overview of the thesis structure

Thesis is mainly consisting of the two parts. The first part is theoretical one and it focuses on introduction to the thesis and theoretical background with related literature review. We will try to describe and concentrate on the theory which stands behind Markov Decision Processes (MDP), vacant taxi routing itself, machine learning (ML) and reinforcement learning (RL). As main topic is Markov Decision Processes (MDP) we will briefly summaries theory for machine learning (ML) and reinforcement learning (RL). As it can give better understanding and overall perception. We will describe MDP components (state and action spaces, reward and value functions, policies and discount factor) and methods of solving MDP (value iteration, policy iteration, Q-learning etc) in more detailed way. We will also briefly go through real life examples or projects which have used MDP in vacant taxi route problem. Such information will help to compare and provide more insight on the topic of my thesis.

The second part is going to be practical one where we will develop methodology, perform experiments in Python environment, make analysis and comparison of outcomes. It will also include conclusion and suggestions. In the practical part we will try to justify chosen methodology and describe the steps of performing the experiment. We will briefly describe data sources and data preprocessing steps, define the MDP model and solutions methods available. After performing the computing part, we will try to evaluate and if possible, improve the model. Analytical part will also include an analysis of possible sensitivity of the results which could take place during experiment and existing or possible limitations.

2 Objectives and Methodology

These are going to be my research questions:

• How exactly can MDP be used to model vacant taxi routing in a dynamic urban transportation environment?

• What is needed to be considered during process of formulating the MDP model?

• How can we be sure that it captures different types of complexities of the modern taxi industry and the urban transportation system?

• How can we use my proposed MDP-based approach to use on large-scale transportation networks and possible real-world scenarios?

• What can be the limitations of the proposed MDP approach with Python implementation? How can be it improved taking into consideration unpredictable factors like traffic jams, fluctuations in demand for taxi and road closures?

Objectives

For the Vacant taxi route problem in Markov Decision Processes we will have these 3 main objectives, such as:

1. Develop Markov Decisions Process (MDP) model for vacant taxi routing problem. We will establish and define the space of actions and states, the transition probabilities as well as determine reward functions for the Markov Decision Process model by using real-world historical data.
2. Develop and relate each Reinforcement Learning (RL) algorithms that are going to be used in the thesis for solving MDP such as: value iteration and policy iteration (Model-based RL) from one side and Q-learning with SARSA (Model-free RL) from another one.
3. Assess algorithms by using established performance metrics such as: revenue gained, travel and idle time.

Additionally, we will try to find answers on these additional objectives:

• How to evaluate the effectiveness of the proposed MDP-based approach in reducing travel and idle time while increasing or keeping the profit on the high level.

• How to investigate the factors that should be considered when formulating the MDP model to ensure it captures the complexities of the taxi industry and the transportation network.

• How to apply the proposed MDP-based approach to handle large-scale transportation networks and real-world scenarios.
• How to improve the proposed approach to account for unpredictable factors like traffic congestion, road closures, and fluctuations in passenger demand.
• How to use research in terms of the potential practical applications and implications of it; additionally we can try to find potential future research directions and areas for further improvement.

By achieving these goals, we aim to develop more effective strategies for taxi drivers (agents) in managing the demand for taxi services. This should lead to increased profits for drivers and a reduction in time and distance traveled without passengers. The plan to process a large dataset through each algorithm to determine the optimal decisions for an agent at the taxi's current and all possible locations. Each method has its own advantages and disadvantages, and we will compare them to identify any similarities or patterns.

Methodology

The introductory section of the thesis will involve an extensive literature review on the relevant topics, including Markov decision processes, and will draw from a variety of credible sources such as monographs, scientific papers, and websites. Through this literature review, the thesis aims to establish a strong foundation for the subsequent research and analysis made in the practical section of the thesis, providing valuable information and good understanding of the research topic.

In the parctical part, the following steps will be carried out:

• We will define the problem which we are trying to solve. The main problem will be optimization of vacant taxi routes using MDP. We will identify necessary variables and parameters that are going to be implemented and used in the mathematical model as: states, actions, rewards, and transition probabilities. This objective will heavliy involve a literature review to identify existing models and approaches which are used, as well as a clear definition of the scope.
• Next we will involve setting up the state and action spaces and reward structure for the routing problem. We are also going to specify the transition probabilities between states based on the selected actions. This objective involves Markov Decision Process (MDP) framework to develop a mathematical model of the problem. We will make assumptions

about the problem in order to simplify the modeling process and will try to justify these assumptions.

• Then we are going to obtain and format open source data from the different statistics offices or companies around the world. Such type of data will include taxi routes (pickup and drop off locations), demand patterns(number of passengers), taxi travel time and other relevant variables. Then we use descriptive statistics to summarize the data (such as mean, median, and standard deviation) In this step we will try to use descriptive statistics to sum up the collected data and include measures as: median, mean and standard deviation. This information will help me to understand the distribution of the data and provide an overview of the data.

• After we will try to simulate different scenarios to evaluate the performance of the MDP algorithms by using Python and Python libraries such as: NumPy, Pandas or Matplotlib. It will allow me to evaluate the performance of the MDP algorithms in various contexts. This step will provide insight into the algorithms' effectiveness in optimizing taxi routes under different condition. Implemention of RL algorithms will also take place in Python enivronment. Value Iteration, Policy Iteration, Q-learning and SARSA algorithms will be used and coded. These algorithms will be used to find the optimal policy for vacant taxi route problem and taking into account important aspect of the trade-off dilema between exploration and exploitation.

• After getting results we will compare and evaluate their performance in terms of efficiency and effectiveness. This step will show the most suitable algorithm for the taxi routing problem and also taking into account factors such as convergence speed, computational complexity and probably a solution quality.

• We are going to make a sensitivity analysis to test the robustness of the results by changing the key variables, parameters and assumptions. This step will assist me in determining the stability of the optimal solutions provided and identifying the potential areas for possible possible improvement.

• At the final step we will discuss limitations of proposed approaches and some of the assumptions that we could made in the modeling process. This might involve discussing the generaliztion of the results, possible limitations of the data which we used in analysis, or the potential impact of different modeling assumptions on

3 Background theories and literature review

Vacant taxi route problem

The vacant taxi route problem is a common issue in the transportation area. It appears at the time when taxi driver needs to pick up passenger from a specific location however, he must travel to that destination empty in order to reach it. It results in spending time, fuel on vain and can be cause of environmental and economic waste. The aim of this issue is to get the optimal route for the taxi that leads to minimum the general empty travel distance, while at the same time to meet expectations of the demand of customers.
Some approaches have been offered as solution to this problem. Lets briefly list them without deep explanation as it would require more resources and pages due to specification and complexity of the issue and offered solutions. These are common approaches that can be used to solve the vacant taxi route problem:

a) Mathematical programming models as integer or mixed-integer programming have offered the solution for the vacant taxi route problem. It formulates a mathematical optimization problem in which the objectives are as follows: to find the optimal routes for vacant taxis, to shrink the total travel time and distance. It has the solution in terms of optimization solvers (linear programming or branch-and-bound algorithms).
b) Other approach is based on using genetic algorithms or simulated annealing that are part of heuristic algorithms. It allows to find close to optimal solution. These algorithms rely on a trial-and-error strategy. This strategy implies that the taxi driver's actions are valued based on the rewards observed and the policy is improved through gained experience.

However not long ago, Reinforcement Learning (RL) methods have been proposed as a solution to the vacant taxi route problem. RL models the problem as a Markov Decision Process (MDP), which is the main focus of this thesis research. In the MDP framework, the state of the environment is defined by the taxi driver's current location, the time snapshot, and the current level of service demand. The taxi driver's actions are represented by the different routes that can be taken to reach the final destination. The transition probabilities in the MDP can be based on historical records of travel time and traffic. MDP, RL and Machine learning (ML) will be described in more details in later chapters of this work.

The vacant taxi route problem is a quite important problem in transportation area with noticeable impact on economy and environment. Solution of this problem by using MDP and RL approaches can lead to more efficient and effective taxi services. It can be beneficial for all stake holders as the taxi drivers can learn how to make more money and be more optimal in their decisions, it will allow them to minimize the empty travel distance and meet the demand. (Zhang, 2017) (Xinlian Yu, 2018)

Machine learning (ML) overview

As it has been mentioned already in the introduction part, Artificial intelligence (AI) is very broad term which consists of many subfields which play important role. Machine Learning (ML) is the crucial one as it allows programs to involve different algorithms and statistical models which can improve performance via experience. To put it simply, ML trains programs on how to learn based on given data, plan future decisions and recognize various patterns.

Machine learning (ML) also can be dived in five main parts:

1. Unsupervised learning (UL). It involves input variables which represent features or attributes that are used to make predictions. In UL a program is left with input variables and no target variables or patterns which it should predict. The program tries to figure it out by algorithms and statistical models with some tries. In such way the program tries to find hidden patters, groups of data and undiscovered structures. UL is used in a detection of anomalies (network traffic, financial transactions etc), it also used widely in e-commerce area (recommendation systems which offer us different products or services based on the search or browser history and purchasing patters).
2. Supervised learning (SL). It involves labeled datasets which have been provided by external supervisor. SL is opposite of Unsupervised learning (UL) where program tries to discover on its own. These datasets have predefined input and output variables. Output variables represent label or so called target variable which a model tries to predict based on the given input variables. SL is used widely nowadays, for example we can find it in image and voice recognitions, email spam and fraud detections etc.
3. Reinforcement learning (RL). This type of machine learning is going to be my main topic and to be more precise Markov Decision Processes (MDP) that is subfield of

it. It is completely different from the above types of learning as it focuses on different aspects and has different objectives. It can be said that RL is similar in some ways to the learning process of humans or animals. It bases on the behavioral psychology theory of reward-based learning. RL algorithms are created in a way to get optimal behaviors by trial-and-error interactions with an environment. It involves getting rewards or punishment. Humans and animals use same trial-error approach but it is important to mention that our and animals brains are more complex and there are many aspects that are still not understood by science even today. (Sutton, 2018)

4. Deep learning (DL) is based on processing and analyzing large amounts of data that pass through neural layers. It allows the machine to learn and recognize various patterns in the data. The neural network consists of so-called interconnected nodes that process data and then they send it to the next layers and so on. Such approach allows for the transformation of data and knowledge in more complex ways. DL has revolutionized the machine learning industry and has created more interest in its present and future usage. It can solve very complex problems and deal with huge amounts of data but it requires more computer power and advanced hardware to perform these tasks. (IBM, n.d.)
5. Transfer learning (TL) stands for its name as it “transfer” knowledge which has been gained from one task to assist other similar one. This brilliant idea of balancing knowledge from one area with a huge amount of labelled data to areas where there is lack of such labeled data. Such approach can improve task performance. TL can be found extensively in a variety of domains, including computer vision, natural language processing, and speech recognition. (Brownlee, 2017)

Reinforcement learning (RL) overview

Famous American psychologist B.F. Skinner conducted experiment in the middle of the 20th century in which he studied some animals and their responses on different kind of stimulus. He used a device known as a Skinner box or operant conditioning chamber to analyze animal behavior in response to various environmental stimuli. He placed the rat inside the Skinner box and presented it with a lever that, when pressed, delivered a food pellet in one of his most famous experiments. The rat first pressed the lever at random, but after learning to identify the lever with the delivery of the food pellet, it pressed the lever more regularly and consistently to obtain the reward. (Saul Mcleod, 2023)

Below you can find famous Skinner Box.

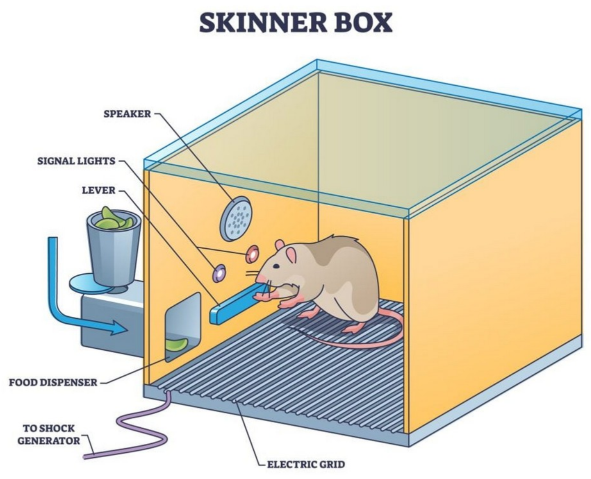


Figure 1 Skinner Box (Saul Mcleod 2023)

This study created the basis for the theory of operant conditioning. This event emphasized the importance of environmental stimuli in shaping behavior.

Reinforcement learning (RL) was heavily inspired by this approach, where an agent learns to take actions based on rewards or punishments in an environment. There is a growing body of research exploring the connection between operant conditioning and reinforcement learning in both psychology and artificial intelligence.

RL focuses on training an agent to interact with an environment in order to maximize a cumulative reward signal. In RL, the agent learns through trial-and-error interactions, similar to how animals learned in Skinner's experiment or how people learn. As it has been already mentioned that RL consists of agent, action, reward and environment but there are more components which are needed to be stated.

These components are:

1) **Agent** is an entity that interacts with an environment in order to achieve some specific goal. By interacting with the environment, the agent learns from it based on either positive or negative feedbacks ("reward" vs "punishment"). The agent uses this information also to decide on future actions and map actions with

feedbacks which he can get. This is called policy. The goal of agent is to maximize the overall reward during the long-term time period. The agent can be a simple software program or more complex robot.

2) **Policy** is a decision-making function that connects and maps the agent's current state to an action. It is a strategy that suggests actual action based on the current state. Policies can be either deterministic or stochastic. The agent should find optimal policy which will lead to maximization of the overall long-term reward.
   Deterministic policy is really useful in a predictable environment as it provides the same action based on the same state. For example, when a person wants to buy snack at the vending machine there are only 2 actions which machine can perform. It either gives snack if enough money has been given or decline transaction if there no or not enough money.
   Stochastic policy is based on the decision-making function that chooses actions in relation to a probability distribution function over available actions that are given at the present state. It means actually that the same state can lead to different actions at different times, based on the probabilities associated with each action. Such type of policy is often used in environments with uncertainty for example AI bots in computer games.
3) **Environment** is the external system in which the agent interacts and from which he gets knowledge (feedbacks). Environment consists of the entities that affect the agent's state, plus other agents, events and objects. Environment can be in various types but most common are deterministic and stochastic.

Deterministic environment states that result of action is determined entirely by the present state and this action. Just to say it simple it means that the result is determined by the action and without any randomness. Good example of such policy can be chess where rewards and states are determined by a player and there are no random factor that could change it (as rules are set up for each move and position of figures).

Stochastic environment has some level of uncertainty and randomness in terms of the reward that the agent can get from the action. It leads to conclusion that even though the agent is going to perform same action there is a chance that he will not get same outcome. Example of this environment can be computer games (like shooters or RPGs games with AI bots) where actions of real player can not be fully predicted.

4) **State** is simply the status of environment at the specific time point. It encompasses all information that the agent needs for taking next action. State can be of two types.
   Discrete state space has finite or limited number of possible states that are different among each other. The agent in such type of discrete state can be in only one of possible finite states. Good example can be again chess with limited number of state spaces represented by rows and columns of the chess board.
   Continuous state space is opposite to the discrete one as it has infinite number of possible states. Continuous state space is more challenging to deal with as it require more advanced storing value methods because traditional table method becomes useless due to possible infinite number of states. Neural networks or function approximators (Gaussian processes, decision trees etc) can be helpful in such situation as they can present policy function as a continuous function which can map state to action and value. Example of it can be the self-navigating rocket Tomahawk. It should take continuously adjust trajectory and other factors (wind, distance to the target and so on).
5) **Action** is a decision of the agent that is intended to achieve a specific objective. Agent choses action based on a set of available actions that are available to the agent in the current state and the most important the policy that he learned.
   Action can be divided also in two same categories as the state (discrete and continuous actions)
   Discrete action are those type of actions which can be picked up only from definite set of actions. Discrete actions can often be found in exercises where the action space is small (again chess or simple games like Mario are good examples). Continuing with Mario example, Mario (the agent) can choose one of the available discrete actions as respond to the present state (go forward or back, jump, crawl or shoot fireball)
   Continuous action is opposite to the one above as the agent can take on any value which is available in a certain range.
   Action can be whatever decision the agent performs (turn left or write, shoot or escape etc)
6) **Reward** is a scalar value that shows if the action of the agent was successful or not. It is used to reward or punish the agent's conduct. Reward can be none, positive or

negative of course based on the action made by the agent. Positive and negative rewards are easy to understand as these rewards that bring the agent closer to the goal or opposite. However zero reward does not bring the agent to the goal nor affects the environment. Example of such 0 reward can be AI bots in game or vacuum cleaner robots that move in an neutral area where they do not perform any action or do not come across any objects.

7) **Value function** is a prediction of the long-term reward that the agent can expect to receive at a specific state or by taking specific action. To say it simply, it is the function that estimates possible long-term reward that the agent should receive in the specific state or action.
   It is used as indicator of how good or bad is quality of the policy with taking into account a possible reward which the agent could receive.
   There are two types of value functions:
   i. **State-value function** estimates the expected total reward that the agent can get from the specific state and the policy. Below is the state-value function where:
      The state-value function is denoted as v_π(s), the policy is denoted as π, the state is denoted as *s,* the expected value of random variable under the condition that the agent follows the policy π is denoted as E_π, the variable *t* denotes any arbitrary time step, the current state is denoted as S_t, γ is denoted as the discount-rate parameter, k is denoted as the number of actions. (Sutton, 2018)

$$v_\pi(s) = E_\pi[¿ \vee St = s] = E_\pi ¿$$

Equation 1 The value function of a state s under a policy p (Sutton, 2018)

   ii. **Action-value function** estimates the expected outcome that the agent can get from taking a particular action at a particular state which is followed by a given policy. Below is function that represent action-value function:

$$q_\pi(s,a) = E_\pi[¿ | St = s, At = a ¿ = E_\pi ¿$$

Equation 2 The action value function of action a in the state s under the policy π (Sutton, 2018)

Here lets quote directly Richard S. Sutton and Andrew G. Barto in order to show what $S_t = s$ and $A_t = a$ represent. *"If the agent is following policy π at time t, then π(a|s) is the probability that* $A_t = a$ *if* $S_t = s$*. Like p, π is an ordinary function; the "|" in the middle of π(a|s) merely reminds that it defines a probability distribution over a at A(s) for each s at S."* (Sutton, 2018)

Where $A_t$is action at time t and $S_t$is state at time t, *a* is action and *s* is state.

Both the state-value and the action-value functions are used in the evaluation of policies and making decisions. The state-value function is useful for forecasting the possible future reward which an agent can get from a given state. The action-value function is good for choosing the best action that the agent can take in a given state.

We came across the discount factor while taking a look at both state and action value functions. For this reason it is important to describe it as well. The discount factor is a parameter that determines the importance of future rewards relative to immediate rewards. γ is denoted as the discount rate. It has value between 0 or 1. When the discount factor approaches 0 the agent become greedier and takes immediate reward and opposite when it approaches 1. In such case the agent takes into consideration the long-term reward.

To sum it up the discount factor is an important indicator because it defines how much importance is given by the agent to future rewards. The balance between short-term and long-term rewards can be achieved by selecting the correct value for the discount factor.

(Sutton, 2018) (Moore, 1996) (Karunakaran, 2021)

8) **Model** is some sort of abstract of the environment in which agent is operating. The agent can use this model to plan or predict the possible future state and reward while taking into consideration the current state and action. There are two types of models that are used in the Reinforcement learning (RL):
   i. **Reward model** tries to estimate the reward related to the state and action. The reward model is denoted as a function *R(s,a)*, where *s* is the current state and *a* is the action taken. It means expected reward that the

agent can get for taking the action *a* at the state *s*. Reward model is set up by the person, who create environment. He or she assigns values whether they should positive, neutral or negative. Reward model that is correctly set up, can be very important as it influence the agent's performance and can result in either achieving the goal and the success or failure.

ii. **Transition model** tries to predict the probability of a next state of the environment, based on the state and action that have been made by the agent.

It is also defined by the designer and it is crucial to correctly set up the transition model, as it connects the present state and action to the next ones.

Transition model can be divided into two types deterministic and stochastic:

- **Stochastic model** refers to a situation, where the result of an action is not certain and can be probabilistic. At the moment when the agent performs a specific action at a given state we can find there a possibility that it can lead to different subsequent states, with varying probabilities. It means that the agent's behavior can be influenced by a degree of unpredictability or uncertainty in the environment. Stochastic models can be very helpful and preferred, where there is uncertainty in the environment, or for example the agent has incomplete knowledge of the environment. The transition model is required for dynamic programming approaches, such as value or policy iteration. When it is required to calculate optimum policies for the agent, these approaches rely on a comprehensive grasp of the transition model. The model can be represented as a matrix, where each element corresponds to the probability of transitioning from one state to another, when a specific action is taken.
- **Deterministic model** can have the outcome of an action completely predicted. This means that if the agent performs a specific action at a certain state, it will always lead to the same state or the respond from the environment. Because of that it is easier to calculate optimal policy and check the agent's results. Example can be again the chess board where each step have determined rules. It can be used in

some cases but it is important to mention that this approach is limited as our world is more complex.

(Hui, 2018) (Sutton, 2018)

Exploration vs exploitation dilemma in Reinforcement learning (RL)

There is one challenge which exists in RL and does not appear in other types of machine learnings. It is called exploration-exploitation dilemma where the agent tries to find balance between getting new information regarding environment (exploration) and actions that might lead to higher immediate reward (exploitation).

In exploitation, the agent will try choosing actions that it thinks are going to bring the highest expected reward based on gained knowledge. Exploitation is more about using the agent's present knowledge of the environment that maximizes immediate rewards.

However, in exploration the agent chooses actions that might not have the highest expected reward at the current moment. But such actions can provide new information about the environment which in return might lead the agent to better decisions next time. Exploration is about obtaining information that allows improving perception of the environment and allows discovering more advanced long-term strategies. There are several strategies which address this dilemma:

1) Epsilon-greedy strategy is a quite simple but at the same time effective exploration-exploitation trade-off technique. In this method, the agent chooses actions that lead to the highest estimated value (exploitation) with probability 1 - ε and selects a random action (exploration) with probability ε.
   Below are steps of how to set up this strategy:
   i. Setup value of the probability of exploration (ε) between 0 and 1.
   ii. Generate a random number again between 0 and 1 at each point of time.
   iii. In case the random number is less than ε, choose a random action (which means exploration).
   iv. In case the random number is greater than or equal to ε, choose the action with the greatest estimated value (which means exploitation).
   v. Update the estimates of value which are based on the observed rewards.
   vi. Then repeat steps 2 till 5 until the process of learning is complete.

Using graphs in such case can show which strategy is optimal for the agent as we can plot cumulative reward over time (where X-axis: Time steps and Y-axis: Cumulative reward) and average reward per time step (where X-axis: Time steps and Y-axis: Average reward). By using both graphs, we can compare the performance of the epsilon-greedy strategy with different values of ε. The agent during exploring more environment (which means larger ε) can probably have lower rewards but as the agent learns better actions, the rewards will increase. Opposite can be done as well where with lower exploration (smaller ε), the agent can gradually converge to a suboptimal solution and ignore better actions. (A. C. K. C. Chan, 2017)

2) The Upper Confidence Bound (UCB) strategy takes into consideration level of uncertainty in the estimated values of actions. The agent picks up actions which are based on upper confidence bounds on their estimated values. The algorithm proceeds as follows:

   i. Initialize the action-value estimates (*q*) and the count of how often each action has been selected (N). For preventing division by zero, *q* values are commonly initialized to zero at the same time N values are assigned a small positive number.

   ii. Calculate the UCB value for each action *a* by using below formula:

$$UCB(a) = q(a) + c * \sqrt{\frac{\ln t}{N(a)}}$$

Equation 3 The Upper Confidence Bound (UCB)

   Where q*(a)* denotes the current estimated value of action *a*, *N(a)* means the number of times that action *a* has been selected, *t* represents the total number of actions taken so far, *c* is a positive exploration parameter that determines the degree of exploration (the higher values will encourage the more exploration) and *ln* is the natural logarithm of *t*.

   iii. Choose the action *a* with the highest *UCB(a)* value

   iv. Perform action *a* and then check the reward gained and the next state.

   v. Refresh the estimated value of the selected action *a* then increase the number of times action *a* has been chosen:

$$N(a) = N(a) + 1$$

Equation 4 The number of times action a has been selected

$$q(a) = q(a) + \left(\frac{1}{N(a)}\right) * (r - Q(a))$$

Equation 5 The estimated value of the chosen action a

vi. Then repeat processes 2 till 5 until the stop criteria will be met. It can be some performance level or some number of iterations.

Based on (Sutton, 2018) we can slightly rewrite our formula:

$$A_t = argmax_a [Q(a) + c\left(\sqrt{\frac{\ln t}{N_t(a)}}\right)]$$

Equation 6 The UCB formula adjusted (Sutton, 2018)

The value being maximized in this context represents an upper bound on the potential true value of action *a*.

The UCB strategy effectively assists the agent in balancing exploration and exploitation. By slightly adjusting the exploration parameter *c* it is possible to affect the level of exploration. Higher values encourage more exploration, while lower values favor exploitation. The UCB strategy enables actions with higher uncertainty (meaning fewer selections) to be explored more, which increases the chances of discovering better actions over time. It can be applied to different real-life problems for example: (e-commerce, resource allocation etc) (Sutton, 2018) (Peter Auer, 2002)

3) Decaying exploration rate can help to balance exploration-exploitation dilemma by gradually decreasing the level of exploration over time. This method can be very useful in scenarios where the knowledge of the agent will be improved due to it interactions with the environment. Vacant taxi route problem is one of those scenarios.

At the start point the agent usually has very limited knowledge about the environment. This implies that the exploration is crucial for gathering needed information. While the agent gets more knowledge, he will most probably relay more on the exploitation. In such way, the optimal decision strategy is created based on the acquired knowledge. Decaying exploration rates help to get the tradeoff balance by reducing the level of exploration as the agent learns more about the environment.

Below are logical steps for implementing this strategy:

I. Implementation stage in which the decay exploration rates can be implemented within previously mentioned strategies. Such as the epsilon greedy strategy where the agent decreases the epsilon value or reducing the exploration parameter c in the Upper Confidence Bound (UCB)

Let me take as an example the epsilon-greedy strategy. The agent chooses the action with the highest estimated value (exploitation) with a probability of $(1 - \varepsilon)$ and then picks up a random action (exploration) with a probability of ε. After some time ε will be reduced this at the same time will allow the agent to explore less and exploit gained information.

Below is the chart example of how it might look.

It is visible that after some steps the agent's epsilon ε will be decreased to the point when the agent will exploit the gained knowledge.

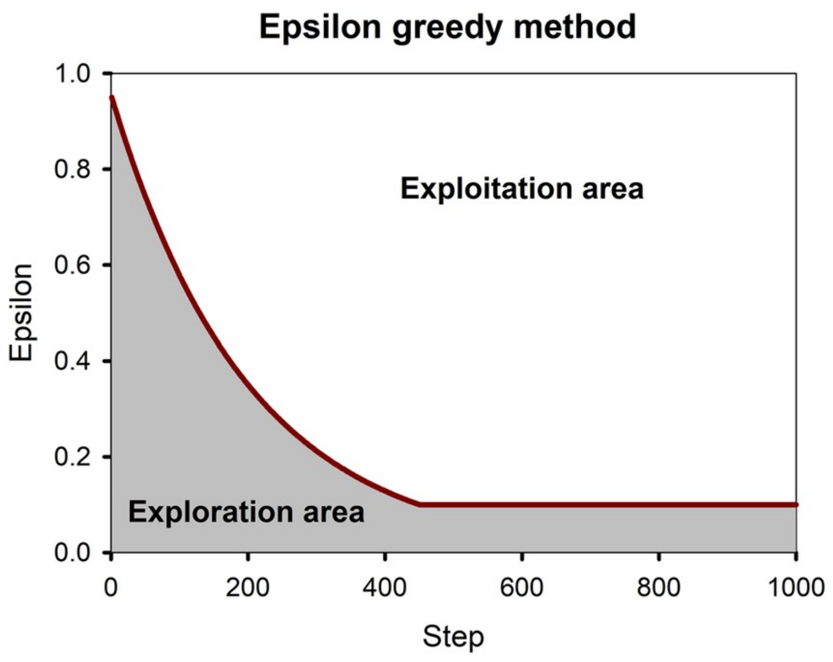


Figure 2 The Epsilon greedy method with reducing the exploration rate ε (Iman Sajedian, 2019)

II. Setting up the decay schedule that is the actual rate of decreasing the exploration. There are several decay strategies which worth to mention:

a) Linear decay method in which the exploration rate *ε* will be decreased linearly over some period. This is very easy to implement approach but it is for sure not the most accurate one as the decrease in exploration is fixed without taking into consideration the learning success of the agent.
The formula for the decreasing exploration rate *ε*:

$$\varepsilon(t)=\varepsilon_0-(\alpha * t)$$

Equation 7 Linearly decreasing exploration rate

Where ε_0 stands for the initial exploration rate, t is time step and α is decreasing rate.

b) Inverse square root decay

This type of decay method decreases the exploration rate *ε* in very slow way over the period. This method is good in cases when the agent needs more time to fully explore the environment before starting the exploitation. Formula for it is down below:

$$\varepsilon(t) = \frac{\varepsilon_0}{\sqrt{t}}$$

Equation 8 Inverse square root decay

Where ε_0 is an initial exploration rate and t is time step and α means decreasing rate.

c) Exponential decay is method where the rate decreases exponentially over period of time (as states its name: exponential). Exponential decay method is very quick and in some sort of way more flexible than two previous ones. It is useful for fast kick start when rapid reduction occurs in the early stages of exploration during the learning process. However, it will decrease slower once the agent gains more experience. Below is the formula:

$$\varepsilon(t) = \varepsilon_0 * (1 - \alpha)^t$$

Equation 9 Exponential decay formula

(SALLOUM, towardsdatascience.com, 2019) (Sutton, 2018) (Weng, 2020) (Kramer, 2010)

III. Application to the vacant taxi route problem can be done by using the decaying exploration rate where the agent can find out different routes and absorb important data about the environment in the first stages of learning. The agent will gradually become more skillful and knowledgeable then the agent can increasingly use(exploit) gained information in order to perform better and optimize selecting process of routes.

IV. Evaluation of decaying exploration rates in the vacant taxi route problem can be done through comparison of the agent's performance by using different decay schedules and exploration strategies available. An average reward per time step or the total cumulative reward can be used in order to evaluate the effectiveness of the decaying exploration rates the exploration-exploitation dilemma.

(SALLOUM, towardsdatascience.com, 2019) (Sutton, 2018) (Weng, 2020) (Kramer, 2010) (The LinkedIn Team, 2023)

4) Posterior sampling or it also can be called Thompson sampling. Posterior sampling is based on the idea of using Bayesian updating so that it maintains a posterior distribution over the expected reward of each: action and the time step of this action. Then its samples from these distributions to choose the action. Thompson sampling balances the exploration-exploitation dilemma by considering both the mean and the uncertainty of the action-value estimates. In order to use Thompson sampling we need to initialize the prior distribution for every pair of action (*a*) and state (*s*), (*s,a*) with estimated values. The most popular selection of distributions is Gaussian or Beta (it can be some other as well). After previous step we need to sample the action-value function *q(s,a)* (for each available action at the present state *s*). Then we need to choose the highest sample value of the action-value function *q(s,a)* and execute this action with the highest value. After the execution we need to record the reward and the next state *s**; then we need to update the posterior distribution function parameters of chosen the action, current and next states and the reward. We are going to continue until the best optimal policy will be generated and the learning process will eventually stop. (Sutton, 2018) (Lihong Li, 2011) (Russo D. J., 2018)

The exploration-exploitation dilemma is very important in Reinforcement learning (RL) as it is basis for finding the optimal policy for the agent. When the agent follows one specific way (the exploitation or the exploration) it can lead to lower reward (in case the agent chooses only exploitation) as the agent will not use opportunities provided by the environment at the current moment or not maximally effective performance (if the agent chooses only exploitation).

To solve this dilemma, the above strategies have been introduced. Each strategy may provide different outcome, it can be due to many factors depending on type of problem it is trying to solve. In my case the problem is finding the vacant taxi route. The performance of each strategy (under such condition) will depend on various factors such as: an initial state of the agent, a customer demand, a traffic condition and following parameters that have been chosen in particular strategy. It is possible and highly suggestible to use more than one or ideally all strategies that can be formulated as Markov Decision Processes (MDP) in the vacant taxi route problem. Some strategies might be more effective in quickly adapting to changing passenger demands, while others might be more robust in the presence of uncertainty or noise in the environment. In this paper we will try to use all of these strategies in order to find out the most suitable for the vacant taxi route problem.

Markov Decision Processes (MDP)

Markov Decision Process (MDP) is a mathematical framework that has been named after Russian mathematician Andrey Andreyevich Markov (1856 - 1922). He worked on the probability theory and developed Markov chains.

Figure 3 Andrey Andreyevich Markov (Robertson, 2006)

Markov chains are a mathematical concept that describes a sequence of events whose probability is controlled only by the state of the previous event. It means that the probability of transitioning to a new state is purely determined by the present state and can't be affected by any previous states. Markov chains are used in a vast range of applications in disciplines (like economics, computer science etc.) (Robertson, 2006)
MDP is the extension of Markov Chains and was developed in the middle of the 20th century by famous American mathematician and computer scientist Richard Bellman (1920-1984). MDP is the mathematical framework for decision-making problems that is used by RL algorithms to find optimal policy. MDP is used to model decision-making problems which results are partly random and under a control of a decision-maker and describe an environment in Reinforcement learning. Reinforcement Learning (RL) is a bigger concept that includes Markov Decision Processes (MDP)
The problem in MDP is formulated as a set of states, actions, state transition probabilities and rewards. The environment (sometimes can be called system) must be in a particular state at any given time and the agent is able to choose an action to take from action space. The agent chose action with regards to the present state and the policy. Taking a new action triggers the system transitions of a new state then the agent receives a reward. The goal of the agent is to maximize the expected overall reward over time. Below chart that has been taken from the book of Sutton, A. B. *"Reinforcement learning: An introduction (2nd ed.)"* clearly shows logic of MDP (where $A_t$is action at time t and S_t is state at time t and R_t is reward at time t)

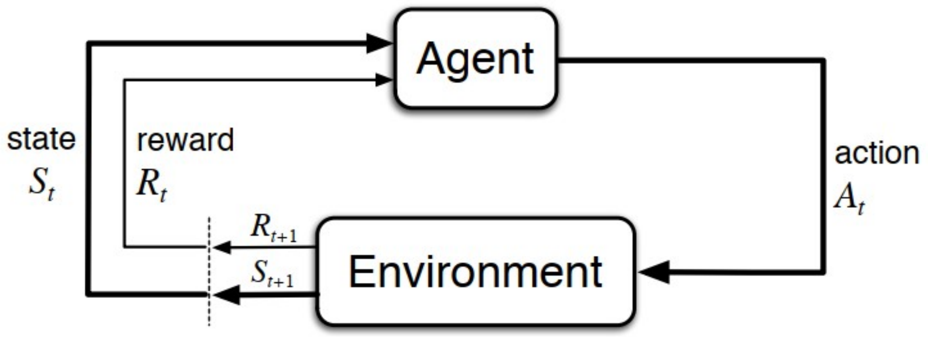


Figure 4 The agent–environment interaction in a Markov decision process (Sutton, 2018)

It is important to mention that there are different types of Markov Decision Processes (MDP) that can be found in the literature. Understanding these variants will help to select the most appropriate one for the vacant taxi route problem:

1) Finite-horizon MDP is limited to some number of time steps which are named horizons. In this case the agent main goal is to maximize the overall cumulative reward during this horizon. Such type of MDP is useful for issue with a final endpoint. Finite-horizon MDPs are particularly useful for problems with a definite endpoint.
   Good example can be a simple inventory management problem where a store owner must decide how many items to order for a three-day period. The store owner aims to maximize profit while minimizing costs associated with overstock and stockouts. The state space is the current inventory level, the action space consists of a quantity of items to order and the reward function calculates a profit based on sales, inventory holding costs plus stockout penalties. In this MDP, the decision-making process is limited to a 3-day period. Where each day, the store owner decides how many items to order then at the end of the day, the inventory warehouse is updated based on sales and deliveries. The objective is to maximize the total profit over this three-day horizon, taking into consideration the costs and penalties associated with inventory management. By formulating the problem as a finite-horizon MDP, the owner of the store can identify the optimal ordering policy for each day that maximize the overall profit for the given horizon.
2) Episodic MDP divides the decision-making process into so called episodes where each of which has its separate terminal state. Once the agent reaches the terminal state, the episode ends then the next episode will begin. The agent is going to learn

continuously across multiple episodes which will allow to refine its policy over time.

Good example can be a financial investment problem where an investor tries to find a way to allocate funds among different assets over 1 year period. The goal of the investor is to maximize the return on investment while at the same time minimizing the risks related to fluctuations in asset prices. This problem can be modeled as the episodic MDP. The state space of this MDP consists of the current portfolio allocation, the historical prices of the assets plus any relevant economic indicators. The action space consists of buying/selling different assets as well as sustaining the current portfolio allocation. The reward function include the returns generated from the portfolio that adjusted for the risk that can be associated with the chosen asset allocation. In this episodic MDP, each episode begins at the start of the investment period and ends at the end of the year period. The objective of the agent is to maximize the total risk-adjusted return on investment during this episode.

3) Infinite-horizon MDP is opposite of the horizon MDP as it does not have a fixed endpoint. The agent operates indefinitely and the objective is to maximize the overall cumulative discounted reward over an infinite number of steps. The discount factor (γ) has a value between 0 and 1, in such way we are ensured that the sum of rewards remains finite despite the infinite time horizon (placing the discount factor in such interval allows prioritizing immediate rewards over distant ones)

   A real-life example of the infinite-horizon MDP can be a power grid management problem. Where operators must decide how to allocate resources to meet the demand for electricity. The state space includes the current power generation levels, the power grid infrastructure and the electricity demand. The action space consists of adjusting power generation levels and investing in the new infrastructure or repairing the existing one. The reward function reflects the profit made from selling electricity and minus the costs of generating power, maintaining infrastructure and possible penalties for power outages.

   The utility company must continuously make decisions to ensure reliable and efficient power supply over an indefinite time period. By finding the optimal policy, the company can effectively balance short-term profits with long-term investments which will result in a sustainable and profitable power grid management strategy. (Sutton, 2018) (Rieder, 2011) (Russo P. D.)

In the case of the vacant taxi route problem that is my topic of the research. The problem can be modeled as the episodic MDP with each episode being a vacant-to-occupied transition for the taxi. The terminal states can be defined as instances when passengers are picked up and the learning process continues with the taxi becoming vacant again after dropping off passengers. However, the infinite-horizon MDP can also be appropriate for modeling the vacant taxi route problem because the process of finding passengers is or can be continuous. It does not have a specific termination point.

By carefully going through these types of MDP, it can be determined that the most appropriate formulation for the vacant taxi route problem should be either the infinite-horizon or the episodic MDP. Because both formulations grasp the continuous nature of the problem and allow the agent to learn and optimize its decision-making process.

Reinforcement learning (RL) frameworks

In order to solve the decision-making problem in RL, it is very crucial to setup the elements of RL that have been described before (such as the agent, the actions and so on) in a step by step and systematic way. Otherwise, neglecting some of its components might lead to totally wrong output and model. RL has some other frameworks beside MDP, which helps to deal with it. For the sake of understanding how vast RL is and which type of the decision-making problems it can solve, it is a good idea to describe them shortly:

1) Multi-Armed Bandit (MAB) represents a simplified but challenging problem in RL. In MAN, the agent interacts with multiple "arms" (actions) where each arm is associated with an unknown probability distribution of rewards. The main objective of the agent is to maximize the cumulative reward by sequentially selecting arms over a limited number of time steps. The main challenge of MAB is to keep balance between the exploration and the exploitation. Algorithms (such as Epsilon-Greedy, the Upper Confidence Bound (UCB) or Thompson sampling) that have been mentioned previously can help to solve MAB. MAB is used in webpages optimization, medical filed (it can help to find best drugs and treatment approach). (Sutton, 2018)
2) Contextual Bandits is more extended version of MAB. It introduces a new element called contextual information (it is often referred as features). The features can assist the agent to make better decisions as it changes the agent's goal. Now the

goal is to obtain a policy that maps the features to the actions and maximize the cumulative reward. (Surmenok, 2017)

3) Inverse Reinforcement Learning (IRL) makes the agent to study an unknown reward function. This function is connected to an expert (can be external action maker for example a human being) action and behavior. The goal is to infer the expert's underlying reward structure and use it to make decisions in the same environment. Good example can be an autonomous car where it learns and observes a human driver and his or her driving decisions. Then agent tries to understand and conclude the implicit reward function that is important in the safe and efficient driving. (Alexander, 2018)
4) Partially Observable Markov Decision Process (POMDP) is the extended version of MDP. It introduces a new element called a belief state. The belief state is simply a probability distribution over the possible states of the environment that considers previous experience (it happens because based on POMDP is partially observable). The goal is slightly altered compare with original MDP, here the goal of the agent is to find the optimal policy that connects the belief system of the agent and the actions that lead to the maximum reward. Such approach can be challenging and require sufficient accuracy. (Michael Hahsler, 2021)
5) Semi-Markov Decision Process (SMDP) involves discrete and continuous states with actions. SMDP is another extension and generalization of the Markov Decision Process (MDP). Compare with classical MDP, SMDP has one major difference and it is time that spent at each state. It can vary and depend on the actions taken by the agent. It implies that the environment can be in one particular state for an unknown period of time. Transitioning to a new state depends now on probabilistic rules. SMDP approach is useful in building complex systems in which actions can cause continuous effects (of course with amount of time to take effect). The objective of the agent in SMDP is to get policy that is going to maximize the expected overall reward and the policy connects the present state and the time that has passed since the last action and to the action that leads to maximum reward. (Baykal-G″ursoy, 2007)

Please note that this might be not full list as there are few more approaches which can be added to the above list based on different experts.

Reinforcement learning (RL) algorithms that solve Markov Decision Processes (MDP)

So based on the previous information regarding setting Markov Decision Processes (MDP) framework it is very important to setup the most important part of solving MDP which is algorithm. RL algorithms allow to update the action-value or the state-value functions one the agent start interacting with the environment. There are many RL algorithms which can be used to solve MDP related tasks, however in this research we will take into consideration only four of them. The rest might be mentioned during comparison or in the conclusion part of the thesis. However, before talking about the algorithms that can solve MDP, it is important to mention some of background in order to understand the field more deeply.

Reinforcement learning (RL) can be split into two main categories such as: Model-based and Model-free. Model-based Reinforcement Learning (MBRL) utilizes a model of the environment to make and plan decisions. The agent aims to learn a model of the environment's dynamics that is also known as the transition model (described before). It predicts the next state under the current state and action. Such model can be either deterministic or stochastic, it will depend on the nature of the environment. The agent will eventually learn the reward model that in return is going to estimate the immediate reward for the state-action pair. Dynamic Programming (DP) algorithms such as Value iteration and Policy iteration can help to solve MBRL problems (such as the vacant taxi route problem in my case) Let me describe these algorithms which are going to be used in the practical part.

1) Value iteration algorithm continuously updates value functions until it approaches the optimal value function. Once we get the optimal value function, we can derive the optimal policy by selecting the action that maximizes the value at the each given state. Value iteration algorithm works by iteratively updating the value function for every state in Markov Decision Processes until it converges to the optimal value function. The value function means the expected total discounted reward that can be gained by following a present policy from given state.

To sum it up, Value iteration algorithm is a dynamic programming (DP) algorithm that iteratively calculates the optimal value function for every state in Markov Decision Processes (MDP) until convergence. After that it extracts the optimal policy based on the optimal value function. It initializes the value function then iteratively update the value function by using famous Bellman equation [1]and after that extracts the optimal policy from the optimal value function. In case of Value iteration algorithm, the optimal policy means the best action that can be taken by the agent in each state that maximizes the expected total discounted reward.

2) Policy iteration algorithm
   Policy iteration is also Dynamic programming (DP) algorithm that is used to solve the MDP problems by repeatedly improving the policy until it becomes the best optimal policy. This optimal policy is again the one that is going to maximize the expected total discounted reward for all possible states. Policy iteration mixes both policy evaluation and improvement steps, such mix allows to iteratively find the best optimal policy. However, this policy iteration is a bit more complex than previous one due to more iteration of the policy evaluation step. Policy iteration tends to be more complex than value iteration.
3) Q-learning Algorithm belongs to model-free reinforcement learning and it is also an off-policy algorithm which means that the agent learns the optimal policy by estimating the optimal Q-values (it can do so even when the current policy is not the optimal one). That's why Q-learning is also called off-policy algorithm.
   Here is it important to define the meaning of Q-value (it can also be called the action-value) Q-value is the expected cumulative reward that the agent can get by performing specific action at the current state and by following a certain policy afterwards. To say it simply, the Q-value is a pair of action and state (*s,a*) and is defined as the expected sum of discounted rewards that can be

[1] Bellman equation is the mathematical relationship used in reinforcement learning. It describes how the agent should update its estimate of the value of a particular state. It also takes into account the expected future rewards of being in that particular state and taking action from that state. The agent can learn to estimate the value of each state and use this information to guide its decision-making.
By iteratively using the Bellman equation, the agent can understand how to estimate the value of each state and use this information to guide its decisions.

gained by taking action *a* in state *s* and by following the optimal policy afterwards. (oreilly.com, n.d.)

$Q(s,a)=E_{\pi p}¿$

Equation 10 State-action value function (Q value function) (oreilly.com, n.d.)

where *s* is the present state of the environment, *a* is the action taken by the agent in that state, r_{t+1+k} is the reward gained by the agent at time step *t+1,*
γ is the discount factor, *E[ ]* means the expected value of all possible results of taking action *a* in state *s* and following some special policy afterwards.
The Q-value is being implemented in many reinforcement learning algorithms such as in my case Q-learning and SARSA. It estimates the value of taking a certain action in specific state. The agent will use Q-value to update policy after that it selects the action with the highest Q-value at the present state.
Later on Q-value can be saved in a lookup table or can be presented as a function approximation (oreilly.com, n.d.)

4) SARSA ((State-Action-Reward-State-Action) is like Q-learning estimates the optimal Q-value for each and every pair action and state. Which represents the expected discounted cumulative reward that the agent can get by perfoming a specific action from a specific state and conducting a particular policy after it. It is important to mention that SARSA is the on-policy algorithm which means that it updates the Q-values table based on the current policy that the is following. (geeksforgeeks.org, 2021)

Below matrix (Figure 5 Reinforcement learning overview) can show how Reinforcement learning can be split in two main parts where: blue color represents Model-free type of RL and green Model-based one. While model-based can also be split in to two parts such as dynamic programming with Markov Decision Processes (MDP) and its algorihms and non-linear dynamics with optimal control algorithm. In the same time model-free in gradient free we can find algorithms which have been specifed above (Q-learning and SARSA). It is also good to mention the unique characteristics of each algorithms and their suitability for the master thesis topic.

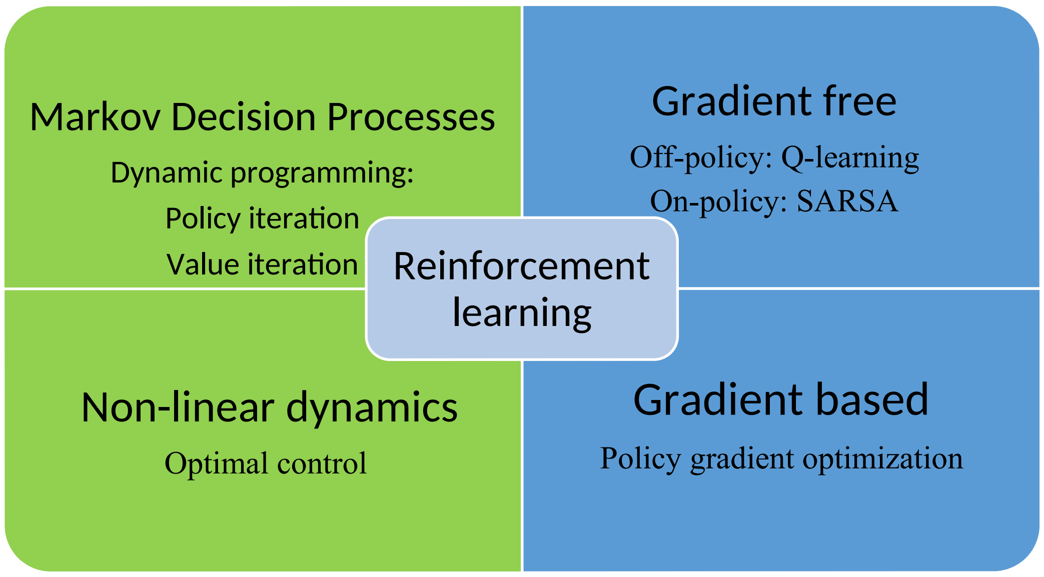


Figure 5 Reinforcement learning overview

3.1.1 Unique characteristics of reinforcement learning algorithm

Value iteration algorithm: The unique characteristic of this algorithm is that it focuse on iteratively updating the value function till the moment when it converges to the optimal value function. Value iteration can be well-suited for the vacant taxi route problem if the state space is not too large (this what also might be tested and confrimed during the running expirement) and the taxi can estimate the optimal value function in order to find the best routes to customers. It is also important to mention that the problem would require knowledge of the MDP's transition probabilities and reward functions. (Sutton, 2018)

Policy iteration algorithm: This algorithm can be split into two-step process: policy evaluation and policy improvement. Again this algorithm can be appropriate choice for the vacant taxi route problem. This method as well as previous one requires full knowledge of the MDP's transition probabilities and reward functions. (Sutton, 2018)

The key requirement for applying MDP based algorithms (Policy iteration and value iteration) is to have a well-defined state space, action space, transition probabilities, and reward function as these components will define the problem accurately which is improtant in solving MDP related problem. In case of the vacant taxi route problem,

The overall success of algorithms would depend on the quality of the data used to estimate the transition probabilities plus reward function (also taking into account possible complexity of the state and action space). However when the problem is well-defined we can say that the algorithm should be able to learn an optimal policy(which will maximize the cumulative reward)

Q-Learning Algorithm: Q-Learning is can learn the optimal policy even if the current policy is suboptimal, additionally it does not require full knowledge of transition probabilities or reward functions (by being part of an off-policy and model-free reinforcement learning algorithm). Such unique characteristics distinguish Q-Learning as suitable choice for the vacant taxi route problem. It also can handle large state spaces and continuous states with function approximation techniques, which makes suitable and scalable for complex type of problems.

SARSA Algorithm: SARSA, like Q-Learning, is a model-free reinforcement learning algorithm. However, it is an on-policy algorithm, meaning it learns the value of the current policy, which allows for better control over exploration and exploitation. This characteristic makes SARSA a suitable choice for the vacant taxi route problem when the environment is uncertain or incomplete information is available, and the learning process aims to balance exploration and exploitation effectively. Like Q-Learning, SARSA can handle large state spaces and continuous states with function approximation techniques.

In summary, the choice of algorithm for the vacant taxi route problem depends on the size of the state space, the availability of information about the MDP, and the desired balance between exploration and exploitation. Model-based methods like Value Iteration and Policy Iteration may be more suitable for smaller state spaces and when full knowledge of the MDP is available, while model-free methods like Q-Learning and SARSA can handle larger state spaces and uncertain environments.

4 Practical Part

Data source overview

The data source plan is to utilize for addressing the empty taxi route issue in MDP is the New York City Taxi and Limousine Commission (TLC) trip record data. This extensive dataset encompasses information on taxi journeys in New York City from 2009 until the present (although 2023 data is currently unavailable at the current moment), featuring details such as pick-up and drop-off date with time, locations and other trip-related aspects. The TLC trip record data is a vast and intricate dataset, comprising millions of rows of data. It includes a variety of crucial variables for later simulation use, such as pick-up and drop-off locations and times, fares, tips or additional ride specifics. The data can be accessible in several formats, like CSV and Parquet, which provide versatility in analysis and processing. The primary purpose of the TLC in collecting this data is to regulate New York City's taxi and for-hire vehicle sector, ensuring passenger safety and tracking industry trends. The data is publicly available for research and analytical purposes and is employed by various groups of stakeholders, including policymakers, researchers and taxi and for-hire vehicle operators. The TLC trip record data, spanning from 2009 to the present, is regularly updated with new information. Consequently, researchers and analysts can leverage this data to observe trends over time and create long-term solutions to problems such as the empty taxi route issue in MDP. However, it is crucial to acknowledge that the data is highly complex and may necessitate substantial processing and cleaning before it can be effectively utilized.

Data analysis

Data was taken from the official website of the City of New-York (nyc.gov (NYC Taxi & Limousine Commission, 2023)) On the website, we can find yellow and green taxi trip records are split by months and years and converted to PARQUET[2] data format. Each record contains data on pick-up and drop-off date and time, pick-up and drop-off locations

[2] PARQUET is a data file format that is open source and optimized for storing and retrieving data in a column-oriented manner. It utilizes advanced compression and encoding techniques to efficiently manage large volumes of complex data. This results in improved performance and faster processing of data. (databricks.com, n.d.)

IDs, ride distances, fares, rate types, payment methods, and passenger numbers. The datasets were gathered and supplied to the NYC Taxi and Limousine Commission (TLC) by authorized technology providers under the Taxicab & Livery Passenger Enhancement Programs (TPEP/LPEP). The TLC did not create the trip data and makes no claims regarding its accuracy.

In the same section we can find For-Hire Vehicle (FHV) trip records which consist of data containing the dispatching base license number, pick-up date and time and taxi zone location IDs (shape file can be found on the same page). These records originate from the FHV trip record submissions made by the bases. It is important to keep in mind that the TLC publishes base trip data as provided by the bases and it actually cannot guarantee or check their accuracy and completeness. For this reason, gathered data might not provide the total volume of trips dispatched by all TLC-licensed bases. The TLC conducts regular reviews of the records and enforces actions when necessary to ensure as far as possible that the information is complete and accurate. (NYC Taxi & Limousine Commission, 2023)

Since the vacant taxi route requires a good quality and nearly complete dataset for better modeling and simulation, the FHV records will not be used as they might not include portions of data. Another reason for excluding the FHV records is the absence of fares in the dataset, which would make it problematic to find the average fare and make calculations of cumulative reward more complicated.

It is also important to mention main differences between green and yellow taxis.

Yellow taxis are licensed by the TLC and can pick up passengers in the entire city (Manhattan and in the outer zones). This taxi type requires to use metered fares which fall under regulations of the TLC. These taxis also follow specific routes and regulations set by the TLC. Yellow taxis became symbol of New York city as they have a distinctive yellow color and roof lights that show availability.

Green taxis (or Boro Taxis) are controlled by the TLC as well but in the same time they are only allowed to pick up passengers in the outer boroughs (Bronx, Brooklyn, Queens, Staten Island) and in certain designated areas of Manhattan (north of West 110th Street and east of 96th Street). Green taxis have a different color and design (green color with a Boro Taxi mark) compared to yellow taxis, and they use metered fares as well. However, their fare rates are way lower than those of yellow taxis and there are some differences in the way they can pick up passengers. These taxis area able to pick up passengers only on streets in special areas outside of Manhattan. For this reason it will also not be included in this dataset of experiment as it does not cover the whole New York city area and has lower fare rates compared to yellow taxis.

I. Data dictionary

Prior to analyzing and processing the raw data, it is important to consult the data dictionary provided by the NYC website to determine which columns will be used or removed.
Below table represents Data Dictionary for Yellow Taxi trip records (nyc.gov, 2922)

Field Name	Description
VendorID	A code indicating the TPEP provider that provided the record. **1= Creative Mobile Technologies, LLC; 2= VeriFone Inc.**
tpep_pickup_datetime	The date and time when the meter was engaged.
tpep_dropoff_datetime	The date and time when the meter was disengaged.
Passenger_count	The number of passengers in the vehicle. This is a driver-entered value.
Trip_distance	The elapsed trip distance in miles reported by the taximeter.
PULocationID	TLC Taxi Zone in which the taximeter was engaged
DOLocationID	TLC Taxi Zone in which the taximeter was disengaged
RateCodeID	The final rate code in effect at the end of the trip. **1= Standard** **rate 2=JFK** **3=Newark** **4=Nassau or** **Westchester** **5=Negotiated fare** **6=Group ride**
Store_and_fwd_flag	This flag indicates whether the trip record was held in vehicle memory before sending to the vendor, aka "store and forward," because the vehicle did not have a connection to the server. **Y= store and forward trip** **N= not a store and forward trip**

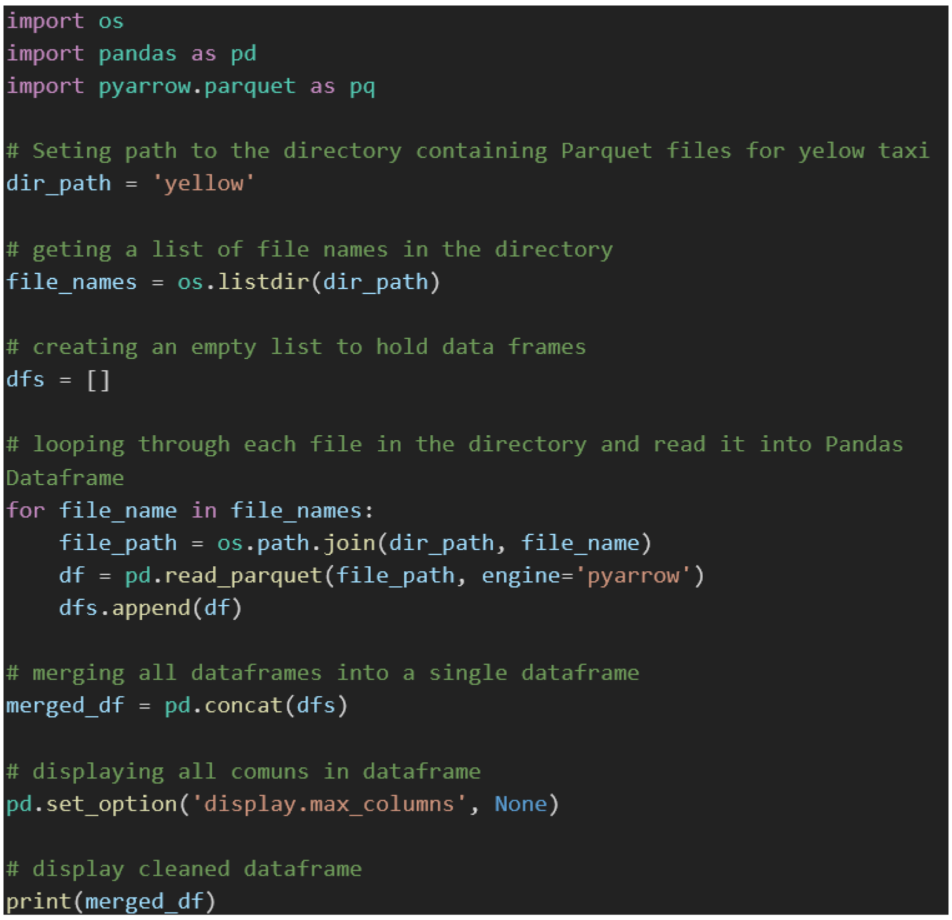
Payment_type	A numeric code signifying how the passenger paid for the trip. **1= Credit card** **2= Cash** **3= No charge** **4= Dispute** **5= Unknown** **6= Voided trip**
Fare_amount	The time-and-distance fare calculated by the meter.
Extra	Miscellaneous extras and surcharges. Currently, this only includes the $0.50 and $1 rush hour and overnight charges.
MTA_tax	$0.50 MTA tax that is automatically triggered based on the metered rate in use.
Improvement_surcharge	$0.30 improvement surcharge assessed trips at the flag drop. The improvement surcharge began being levied in 2015.
Tip_amount	Tip amount – This field is automatically populated for credit card tips. Cash tips are not included.
Tolls_amount	Total amount of all tolls paid in trip.
Total_amount	The total amount charged to passengers. Does not include cash tips.
Congestion_Surcharge	Total amount collected in trip for NYS congestion surcharge.
Airport_fee	$1.25 for pick up only at LaGuardia and John F. Kennedy Airports

Table 1 Data Dictionary – Yellow Taxi Trip Records (nyc.gov, 2922)

For MDP in vacant taxi route, all columns are going to be required except only 3 such as: RateCodeID, Store_and_fwd_flag, Improvement_surcharge are not really needed for the vacant taxi route in MDP.

II. Data processing

Now let's take a look at how many data records we have there and do further analysis, as there might be inconsistent data that needs to be deleted. Because there are 12 separate PARQUET data format files, it will be needed to merge them all into one big dataframe.
For this purpose we are going to use my personal laptop Dell Latitude 7490 - Intel(R) Core(TM) i5-7300U CPU @ 2.60GHz 2.71 GHz and with 16.00 GB RAM. In order to run the code we used Visual Studio Code (which is a free and open-source code editor developed by Microsoft) and imported Pandas, Pyarrow libraries and other libraries. Below Python code is responsible for merging all PARQUET data files into one in order to see how many line records have been made.



```python
import os
import pandas as pd
import pyarrow.parquet as pq

# Seting path to the directory containing Parquet files for yelow taxi
dir_path = 'yellow'

# geting a list of file names in the directory
file_names = os.listdir(dir_path)

# creating an empty list to hold data frames
dfs = []

# looping through each file in the directory and read it into Pandas
Dataframe
for file_name in file_names:
    file_path = os.path.join(dir_path, file_name)
    df = pd.read_parquet(file_path, engine='pyarrow')
    dfs.append(df)

# merging all dataframes into a single dataframe
merged_df = pd.concat(dfs)

# displaying all comuns in dataframe
pd.set_option('display.max_columns', None)

# display cleaned dataframe
print(merged_df)
```

Code 1Python code for PARQUET datafiles merge and display

It provided the result of 39,656,098 rows and 19 columns. Based on the above data dictionary we do not need all 19 columns, additionaly it is going to be necessary to get rid of some inconsistent and biased data. Some lines of code will be created in order to delete rows with inconsistent data such as:

- Number of passengers per ride higher or equal 4 (and lower than 1) according to the *Driver Rule 54-15(g) Chapter 54 - Drivers of Taxicabs and Street Hail Liveries* of the NYC

„The maximum amount of passengers allowed in a yellow taxicab by law is four (4) in a four (4) passenger taxicab or five (5) passengers in a five (5) passenger taxicab, except that an additional passenger must be accepted if such passenger is under the age of seven (7) and is held on the lap of an adult passenger seated in the rear.“ (rule_book_current_chapter_54.pdf, 2016)

- Trip distance which is higher than 100 miles (aprox. 161 km and lower than 0 km). This number is based on the capacity of the full tank of Ford Crown Victoria. This is the most popular car used by the yellow taxi in the New York city. (tankonempty.com, n.d.)

- Amount of fare paid which is going to be higher than 1,000 USD and again lower than 0.
- VendorID will equal 1 or 2 based on the data dictonary (1= Creative Mobile Technologies, LLC; 2= VeriFone Inc.)
- Pickup datetime and dropoff datetime, Congestion surcharge, Airport fee, MTA tax, Tip amount and Extra not equal NaN
- RatecodeID equal of these values:
  (1= Standard rate 2=JFK 3=Newark 4=Nassau or Westchester 5=Negotiated fare or 6=Group ride)
  It is important to mention that eventhough this column is going to be deleted, still we need to be sure that trip records are withing those RatecodeIDs.
- Payment type will equal only 1 = Cash or 2 = Credit. As we are intrested only in trips which had some reward at the end.
- Store and forward flag equal either Y= store and forward trip or N= not a store and forward trip. As well as in prepvous RatecodeID, it will be required to ensure that trips with flags Y or N are presented.
- Drop-off and pickup locations do not equal nothing. It is very important filter as it unselects those rows that might cause problem for Rewards(R) and States(S) and matricies.

Descriptive statistics analysis will be added for below columns:

- PULocationID and DOLocationID – To find the most and the least frequent pickup and dropoff locations based on the LocationIDs, DOLocationIDs and maps provided by the NYC website.
- Median, mean and standard deviation of fare and trip distance
- Most common passengers count and peak and off-peak pickup hours.
- Locations with the highest and the lowest average fares.

To peform all stated above, this code is going to be used: . As it is very long code with many lines, it will be saved in the Appendix chapter.

After running the code, we get results where:

1) Number of rows decreased to 35,382,775 (the code deleted 4,273,323 rows with biased data) and columns were decreased to 16.

2) Fare Median: 10.0 USD. The half of fares in this dataset are less than or equal to 10 USD and in the same time the other half is greater than or equal to 10 USD. This is a measure of central tendency that is not so strong affected by extreme values than the mean (average).

   Fare Mean: 14.570745718502842 USD. It is a measure of central tendency that provides an insight on what can be typical fare amount for a taxi trip in my dataset. But in the same time, it's important to write that the mean can be strongly influenced by extreme values. Thats why it's important to take into account other measures such as the median and the standard deviation.

   Fare Standard Deviation: 13.269421581000914 USD. This measure indicates the dispersion of fare values from the mean. A high standard deviation means that the fare values are more spread out, while a low standard deviation implies that the fare values are closely clustered around the mean. In other words, the high standard deviation indicates a wide range of fares, whereas the low standard deviation suggests a narrow range of fare amounts. In this dataset, the standard deviation is 13.3 USD, which is lower than the average, indicating that fares can significantly deviate from the average fare amount.
3) Peak pickup hour: 18:00

   Off-peak pickup hour: 04:00
4) Most common passenger count: 1.0
5) Distance Median: 1.9 miles. The half of the trip distance miles in this dataset are less than or equal to 1.9 miles (around 3km) and in the same time the other half is greater than or equal to 1.9 miles. This is a measure of central tendency that is not so strong affected by extreme values than the mean (average).

   Distance Mean: 3.5172710735661736 miles (around 5.6 km)

   Distance Standard Deviation: 4.469920397754627 miles. In this dataset, the standard deviation is 4.47 miles (around 7.3 km), which is higher than the mean, indicating that trip distances are more spread out.
6) Popular pickup location: Zone 132 - JFK Airport based on the TLC Taxi zones, zone 132 idicates that most popular borough is Queens, which is the borough of the New York City. JFK(John F. Kennedy International Airport) is in the top 10 of the largest and the busiest airports in the US. (Fubra Limited, n.d.)

Unpopular pickup location: Zone 84 - Eltingville/Annadale/Prince's Bay based on the TLC Taxi zones, zone 84 idicates that least popular pickup borough is Staten Island. This is the least populated part of the New York City. (nyc.gov, 2018)

7) Popular dropoff location: Zone 236 - Upper East Side North Based on the TLC Taxi zones, zone 236 idicates that most popular dropoff borough is Manhattan. This is also the most populated and famous part of the New-York City.

   Unpopular dropoff location: Zone 99 - Freshkills Park based on the TLC Taxi zones, zone 99 idicates that least popular dropoff borough is again Staten Island, the least populated part of New York City. (nyc.gov, 2018)

8) Drop off and pickup locations with the highest average fare: (215 - South Jamaica – Queens and 135 - Kew Gardens Hills - Queens) This indicates the pair of pickup and dropoff locations with the higest average fare.

   Drop off and pickup locations with the lowest average fare: (9 – Auburndal - Queens and 98 - Fresh Meadows - Queens) This indicates the pair of pickup and dropoff locations with the lowest average fare.

Problem definition

After cleaning and processing data, we can finally start with formulating the vacant taxi problem in MDP. For this purpose, it is required firstly to define the goal of the agent. The main objective that the agent (the taxi driver in our case) wants to determine the optimal route that maximizes profits and at the same reducing the overall operational costs for company (This objective aligns with the MDP formulation of the agent's main goal, as discussed in the previous chapter about *Markov Decision Processes (MDP)*)

Markov Decision Process (MDP) components

Now lets define the components of Markov Decisions Processes (MDP):

- States-(S): Depending on the city area, it will be divided into discrete zones. These zones will represent states in the MDP model. We can use drop-off and pickup locations directly as states, such method can increase precision of routing decisions (due to granularity of the model). However, it can also significantly increase the complexity of the MDP model and the overall computation time. For large datasets, the geographical coordinates or K-means [3]clustering algorithm can be used that allow to create zones based on pickup and drop-off locations. It will be decided based on the size of the dataset and its complexity. If there is already setup location IDs or some kind of mapping in a dataset, we will be able to use such opportunity as it assists us in avoiding algorithms mentioned above.

- Actions-(A): It represents set of all possible moves or decisions the agent can make in each state. The action set (A) will be a collection of all possible movements among zones or staying within certain ones. To define the actions, we will need at first to determine the adjacency relationships between the zones. This can be done by using a graph representation where zones are nodes and edges represent connections between neighbouring zones. For each zone, the set of possible actions would include moving to one of the neighbouring zones connected by an edge or staying in the current one.
- State transitions (P) are probabilities of transitioning from one state to another given a specific action. To estimate these probabilities (P) for MDP model by using the reassigned location IDs, it will be required to analyze the taxi trip data to determine the likelihood of transitioning from one zone to another, given a specific action. The transition probabilities can be influenced by various factors such as time of day, day of the week, traffic patterns and other external factors.
- Rewards-(R) represents immediate reward which should be received by the agent for taking the specific action at the present state. Logically speaking for our agent, the goal can be formulated as minimizing the distance or time that the agent spends without the customer or the highest output the agent might get. In such way, the rewards could be designed to encourage reaching high-demand zones quicker or getting to drop-off locations with highest .
- Discount factor-(γ): As it has been mentioned already, the discount factor is a value (between 0 or 1) and it defines the relative importance of future rewards in comparison to the immediate reward. If value goes closer to 1 the agent care more about future rewards instead of immediate ones.

[3] " K-means clustering is a simple unsupervised learning algorithm that is used to solve clustering problems. It follows a simple procedure of classifying a given data set into a number of clusters, defined by the letter "k," which is fixed beforehand. The clusters are then positioned as points and all observations or data points are associated with the nearest cluster, computed, adjusted and then the process starts over using the new adjustments until a desired result is reached.."This definiton was taken from techopedia.com (Rouse, 2016)

I. States(S)

In case of the obtained dataset, we have two ways to express states:

1) PULocationID (TLC Taxi Zone in which the taximeter was engaged) and DOLocationID (TLC Taxi Zone in which the taximeter was engaged) can be used as a state in the MDP model. These locations will represent the discrete states in the model. There is no need to perform further clustering or zone creation since we have the state definitions. These zones can be used as alternative to latitude and longitude coordinates for routing problems.
2) We can use geospatial data that is also available from the NYC OpenData website. GeoJSON file format can be used for defining states. Each zone in the GeoJSON file can be considered a state in the MDP. We can extract these zones and assign them unique identifiers. The GeoJSON file has several columns such as: location_id, zone, borough, and geometry. Each row represents a different taxi zone in New York City. To define zones for a Markov Decision Process (MDP), the location_id can be used as a unique identifier for each zone.

   (City of New York, 2023)

In general, both approaches have their pros and cons, however choice between them depends on the specific requirements and the computational power of a machine. By using pickup and drop-off IDs, the state definition can be simplified, but it might lead to a high-dimensional state space. While using GeoJSON data can offer more flexibility and additional geographical information however it requires preprocessing and may result in a more manageable state space.

As we have large dataset and for this reason it is better to use the original parquet file, as location ids can be more or less by count than in GeoJSON file. Below code will take all unique 'PULocationID'(pickup) and 'DOLocationID'(dropoff) location ids, then calculate and print them out. 'formatted_yellow_taxi_22.parquet' is the merged parquet file which contains data for all 12 months of 2022 year. It was done based on the previous Code 1Python code for PARQUET datafiles merge and display

```
import pandas as pd #import pandas library
#define path to parquet file and read it
file_path = 'formatted_yellow_taxi_22.parquet'
taxi_data = pd.read_parquet(file_path)
def calculate_and_print_counts(taxi_data):
    # Get all unique IDs from PULocationID and DOLocationID
    unique_pickup_locations = taxi_data['PULocationID'].unique()
    unique_dropoff_locations = taxi_data['DOLocationID'].unique()
    #Calculate unique number of states and actions
     num_states  =  len(unique_pickup_locations) #  Number  of  unique
pickup locations
     num_actions  =  len(unique_dropoff_locations) #  Number  of  unique
drop-off locations
    # Print the findings
    print(f"Count of unique PULocationID (states): {num_states}")
    print(f"Count of unique DOLocationID (actions): {num_actions}")
    return num_states, num_actions
#print the result
calculate_and_print_counts(taxi_data)
```
Code 2 States and Actions

Result of the executed code down below:

Count of unique PULocationID (states): 261

Count of unique DOLocationID (actions): 261

So, for all unique IDs in our dataset we have only 261. This list is slightly less, compared with Taxi Zone lookup dictionary, which has been published on the NYC Open data website (City of New York, 2023) This step is important as it allows to filter out rows with inconsistencies which may cause troubles in the future when the MDP model is going to be built.

II. Actions(A)

For 261 States we are going to have 261 Actions, it makes sense because the taxi can move to a close by zone or to completely different zone within the New-York city. This action might be taken as a response to a higher demand or probably better fares in another area. The decision could be based on known patterns of demand, time and day or some special events which might happen in the New-York city. Based on the states already provided and the below for Actions, it has been calculated that total number of states is going to be 261. The total number of states are closely connected with the states. In this case if we take pair of pickup and drop-off locations, it will create 261 actions. This model will reflect more realistic/precise view as the taxi can move within same borough and it will make easier interpretation of policy.

III. State transitions (P)

Empirical data provided by NYC Taxi website can be used in calculation of probabilities. This data will be useful in providing insight into how often taxis move among zones. The data has been already cleaned and prepared before so we can go on with creating Transition Matrix[4].

At first, it will be required to calculate transition counts and normalize those counts to probabilities (we should not forget about importing libraries like numpy and pandas):

```
# Load libraries
import pandas as pd
import numpy as np
# Load the Parquet file
taxi_data                                                          =
pd.read_parquet('yellow/formatted_yellow_taxi_22.parquet')
# Extracting all unique location IDs from the dataframe
unique_locations        =        pd.unique(taxi_data[['PULocationID',
'DOLocationID']].values.ravel('K'))
#  Creates  the  dictionary  with  all  unique  location  IDs  from  the
dataframe
location_to_index   =   {loc_id:   index   for   index,   loc_id   in
enumerate(unique_locations)}
```

Then we can start slowly building the transition matrix. The matrix is going to be 2 dimensional as agreed previously. After setting up dimensions we can continue with counting transitions and grouping.

```
# Set up transition matrix with 2 dimensions
num_locations = len(unique_locations)
# initializes the transition matrix with dimensions num_locations
transition_matrix = np.zeros((num_locations, num_locations))
# Group by PULocationID and DOLocationID and count transitions
transition_counts         =         taxi_data.groupby(['PULocationID',
'DOLocationID']).size().reset_index(name='count')
```

Final steps will be populating transition matrix with data arrays, normalizing in order to get probabilities, replacing NaNs with uniform distribution and finally validating the matrix for stochasticity. NaNs are created by missing outbound transitions that divided by 0. After this 1.0 / num_locations value are assigned to these NaNs values.

```
# Populate the transition matrix
```

[4] In the context of MDP, Transition Matrix is used to describe the probabilities of moving from one state to another on the given action. It is a square matrix where each element stands for probability of transition, while row of matrix represents current state and column represents possible next state. Transition matrix gets probabilistic nature of state transition. (Wong, 2018)

```
for _, row in transition_counts.iterrows():
    pu_index = location_to_index[row['PULocationID']]
    do_index = location_to_index[row['DOLocationID']]
    transition_matrix[pu_index, do_index] = row['count']

# Normalize the matrix to get probabilities by converting counts into
probabilities of transitioning from one location to another and then
insures that each row and then insures that each row sum equals 1
transition_matrix           =           np.divide(transition_matrix,
transition_matrix.sum(axis=1, keepdims=True),
                              out=np.zeros_like(transition_matrix),
                                 where=transition_matrix.sum(axis=1,
keepdims=True) != 0)

# Replace nans with uniform distribution
transition_matrix  =  np.nan_to_num(transition_matrix,  nan=1.0  /
num_locations)

# Validate the matrix and print if it is stochastic or not
if np.allclose(transition_matrix.sum(axis=1), 1):
    print("The transition matrix is stochastic.")
else:
    print("The transition matrix is not stochastic.")
```

Finally, the code validates the matrix and prints if the transition matrix is stochastic or not. We need to be sure that all included transition matrices are stochastic. Stochastic transition probability matrix indicates that type of matrix where each row sums to 1. It means that for each state the probabilities of transitioning to all possible states (by taking certain actions) must sum up to 1. (Gupta, 2018) Full code is in the Appendix chapter Code 4 Transition matrix.

It is very important to check the quality of state transition matrices. The result of the above code is the transition code is stochastic.

As all transition matrices in my model are stochastic (sum up to 1 at every row), it is going to be easy to move to the next steps. Non-stochasticity can have huge impact on the behavior and convergence of the reinforcement learning algorithms (like Value and Policy Iterations and model-free algorithms like Q-Learning and SARSA)

Cleaning data from possible empty rows or anomalies partially help to achieve the stochasticity however the additional checking was required. Anomalies can be caused by underlying issues in the raw data that aren't addressed by applied filters during data preparation. Such as:

- Socioeconomic factors: Level of poverty, crime rate etc. All these factors might make some destinations or pickup locations less desirable with time.
- Geographical factors and unforeseen events: Roads closure, special events, traffic light system failures etc These factors might affect the trip distance, costs etc
- Data collection issues: Errors or limitations during collecting trip data. Here it is important to mention one of the data variables in the dictionary store_and_fwd_flag. It means if the trip data record was kept in the taxi car memory before sending to the vendor, aka "store and forward," due to absence of connection between the vehicle and server. Some data records might never reached server due to connection or other issues. Or the records might reached the server but some of data could be lost during restoring connection between the vehicle and the server.
- COVID-19 pandemic had significant effect not only on the traffic patterns but also on our lives in general. Traffic patterns were changed due to lockdowns and other restrictions during the pandemic. Reduced transportation services provided by taxi and public transport providers. Changes in the travel behavior, prohibitions of mass gathering and complete chaning of leisure and daily livestyles. These and many other facts can be added to the pandemic effect on the traffic.

All above points can have very signifcant impact on the quality of the raw data and these factors are out of someone‘s control. These built-in limitations should be treated accordingly and accepted as uncertainty. For these reason we have implemented the normalization process of transitioning so that each row at the end gives sum of 1.

IV. Rewards(R)

Defining reward structure in MDP is one of crucial parts as it specifies goals of the whole MDP. It indicates what the taxi driver must aim at or avoid in the environment. It is the core of guiding the agent (the taxi driver) in decision-making processes towards achieving the needed outcome. Logically these factors should be included in the matrix:

- Fare amount/total amount – More amount gained means more profit hence reward.
- Trip distance – Length of trips might be focused on longer or less distance, depending on the business model of course.

- Idle time – Decreasing time without any client can be a priority but not often.
- Expenses like charges etc will result in providing clearer overview on the net reward for the agent.

These data fields will be included as they are related to fare, expenses, distance and time. Based on Data dictionary stated in Data Analysis part of this work we will include below data fields:

Field Name	Description
tpep_pickup_datetime	The date and time when the meter was engaged.
tpep_dropoff_datetime	The date and time when the meter was disengaged.
Trip_distance	The elapsed trip distance in miles reported by the taximeter.
Total_amount	The total amount charged to passengers. Does not include cash tips.
Extra	Miscellaneous extras and surcharges. Currently, this only includes the $0.50 and $1 rush hour and overnight charges.
MTA_tax	$0.50 MTA tax that is automatically triggered based on the metered rate in use.
Tip_amount	Tip amount – This field is automatically populated for credit card tips. Cash tips are not included.
Tolls_amount	Total amount of all tolls paid in trip.
Total_amount	The total amount charged to passengers. Does not include cash tips.
Congestion_Surcharge	Total amount collected in trip for NYS congestion surcharge.
Airport_fee	$1.25 for pick up only at LaGuardia and John F. Kennedy Airports

Now, we need to select a reward approach, which is a bit tricky, as we are faced with a trade-off between simplicity and realism, involving the capture and balance of various aspects.

In the case of a simple approach, the reward is directly proportional to the Total_Amount (fare amount + tip). It is a straightforward approach to implement and easy to understand. Taxi drivers receive a reward directly linked to their earnings, aligning with their primary objective. Additionally, taxi drivers can easily grasp how their actions can impact their rewards, leading to more predictable behavior. However, such an approach has its limitations. Firstly, it narrows the focus only to the total fare amount, simultaneously ignoring other factors like customer satisfaction, expenese and efficiency. Furthermore, it can lead to an imbalance in the decision-making process.

A more realistic or balanced approach includes multiple factors, ensuring that no single factor dominates. The balanced approach is more comprehensive as it allows the capture of a broader range of factors that can influence the agent's behavior, such as trip time and distance, additional costs and charges etc. It normalizes factors, preventing a single objective from overshadowing others, ultimately leading to more balanced decision-making by the taxi drivers.

In our case, we are going use the more realistic approach. Also, we do not have data that might somehow provide information on customer satisfaction (Tip_amount has already been included in Total_amount, so it does not make sense and tips do not always indicate customer satisfaction) In the US tipping is very common and clients most often provide tips automatically.

Lets go to set the constant (airport fee) and set airport zone location Ids, then again getting the unique location ids(from pickup and dropoff) with creating dictionary that maps location Ids and the corresponding indices.

```
#import libraries
import numpy as np
import pandas as pd
#define the path
file_path = 'formatted_yellow_taxi_22.parquet'
taxi_data = pd.read_parquet(file_path)
def calculate_rewards_optimized(taxi_data):
    # Setting constants for airport locations and fee
    JFK_AIRPORT_ID = 132
    LAGUARDIA_AIRPORT_ID = 138
    AIRPORT_FEE = 1.25
      #  Get  all  unique  location  IDs  from  both  PULocationID  and
DOLocationID
      all_locations  =  np.union1d(taxi_data['PULocationID'].unique(),
taxi_data['DOLocationID'].unique())
    all_locations.sort()  # Sort for consistent indexing
    # Map location IDs to indices in the matrix
        location_to_index   =   {loc:   idx   for   idx,   loc   in
enumerate(all_locations)}
```

After that we can initialize the reward matrix as the square one (with number of states). Calculation of net rewards will be based on the net reward for each taxi trip by subtracting expenses such as: 'extra', 'mta_tax', and 'congestion_surcharge' charges from the total fare amount total_amount. It will gives us what the taxi gets net after getting rid of all required expenses. For those locations that belong to Airport Ids, we are going to adjust for the airport fee. The code adjusts the net rewards for each trip that is ending at JFK Airport or LaGuardia Airport by subtracting the airport fee.

```
# Initialize the reward matrix
    num_states = len(all_locations)
    reward_matrix = np.zeros((num_states, num_states))
    # Calculate net rewards
         taxi_data['net_reward']    =    taxi_data['total_amount']    -
taxi_data['extra']          -          taxi_data['mta_tax']          -
taxi_data['congestion_surcharge']
    # Adjusting for airport fee
        taxi_data.loc[taxi_data['DOLocationID'].isin([JFK_AIRPORT_ID,
LAGUARDIA_AIRPORT_ID]), 'net_reward'] -= AIRPORT_FEE
```

Next step is adjusting reward by trip distance. ensures that the distance factor is considered when calculating rewards for taxi trips. It provides us more accurate modeling of earnings in case where longer trips are rewarded in a appropriate way in comparison to shorter ones. We calculate the adjusted reward by multiplying the net reward (calculated previously) and 'trip_distance'.

```
   # Adjust the reward by trip distance
        taxi_data['adjusted_reward']   =   taxi_data['net_reward']   *
taxi_data['trip_distance']
```

Later the code aggregates rewards. It is required cause of the purpose to consolidate information about the financial results connected with different routes or state transitions. It can allow us assess pairs of state vs action and which of them are more profitable than others. Aggregating rewards is important in the reinforcement learning, where agents learn to make decisions by maximizing cumulative rewards over some period of time time. The aggregated rewards help us to determine the desirability of different state-action pairs for the agents. The code aggregates the adjusted rewards for each unique state-action pair, where a state-action pair is defined by 'PULocationID' and 'DOLocationID'.

```
# Aggregate rewards for each state-action pair
         aggregated_rewards    =    taxi_data.groupby(['PULocationID',
'DOLocationID'])['adjusted_reward'].sum().reset_index()
```

The rest part of the code iterates through Aggregated rewards and populates the Reward matrix with adjusted rewards for each pair of state and action. Then to be sure that that our reward values are within a reasonable range, the matrix is going to be normalized. It will be divided by the total number of taxi trips. It should be done in order to prevent very large values in the matrix. Then the reward matrix will be printed.

```
# Normalize the reward matrix to avoid extremely large values
```

```
    reward_matrix /= taxi_data.shape[0]
    return reward_matrix
reward_matrix = calculate_rewards_optimized(taxi_data)
# printing  reward matrix
print(reward_matrix)
```

The full code for the reward matrix can be found here: Code 5 Reward matrix

V. Discount factor(γ)

The final step will be adding the discount factor. Once it is done, we can construct the MDP model. Discount factor is very important, especially in this scenario when future rewards need to be weighted differently compared to immediate rewards. The discount factor, denoted as γ (gamma). It can range from 0 to 1 and serves as balance and brings the importance of immediate versus future rewards.

If the discount factor equals 0, the taxi driver will prioritize immediate rewards over future ones. In case the discount factor equals 1, the agent (the taxi driver) will give priority only to future rewards and ignore immediate ones. In the real-world situation, the discount factor should be included in the MDP model. For this reason, we need update the reward matrix so that the discount factor γ (gamma) be added. Quite often γ (gamma) is set up to 0.9 (QUORA, n.d.) However in this model, we are going to use 3 values of the discount factor. It will be equal 0.1, 0.5 (middle between choosing future or immediate rewards) and 0.9 as by using different discount factors we can see if the optimal policy is sensible to the discount factor or not. When the discount factor equals 0.1, it means that the agent strongly neglects the future rewards. Under this condition the taxi driver (the agent) is very focused on getting the immediate reward, it can be useful environments with an high uncertainty as the agent's goals are short-term based and always immediate.

However, when γ (gamma) equals 0.5, it can be viewed as so called balanced approach. As the agent is going to value future rewards more careful compared to immediate rewards. This condition can be often used in environments where we have a certain mix of short-term and long-term factors.

Lastly when gamma equals 0.9 (relatively high), it indicates that the agent strongly emphasis the future reward. In such an environment the agent concern with the long-term consequences and planning. It is suitable for the environment with stability and the long-term planning when the agent plans in the long-term perspective.

It is enough to set up the gamma in the MDP solver itself. This will be shown later in the MDP construction chapter.
(SALLOUM, Basics of Reinforcement Learning, the Easy Way, 2018)

Construction of Markov Decision Process (MDP) for routing taxi

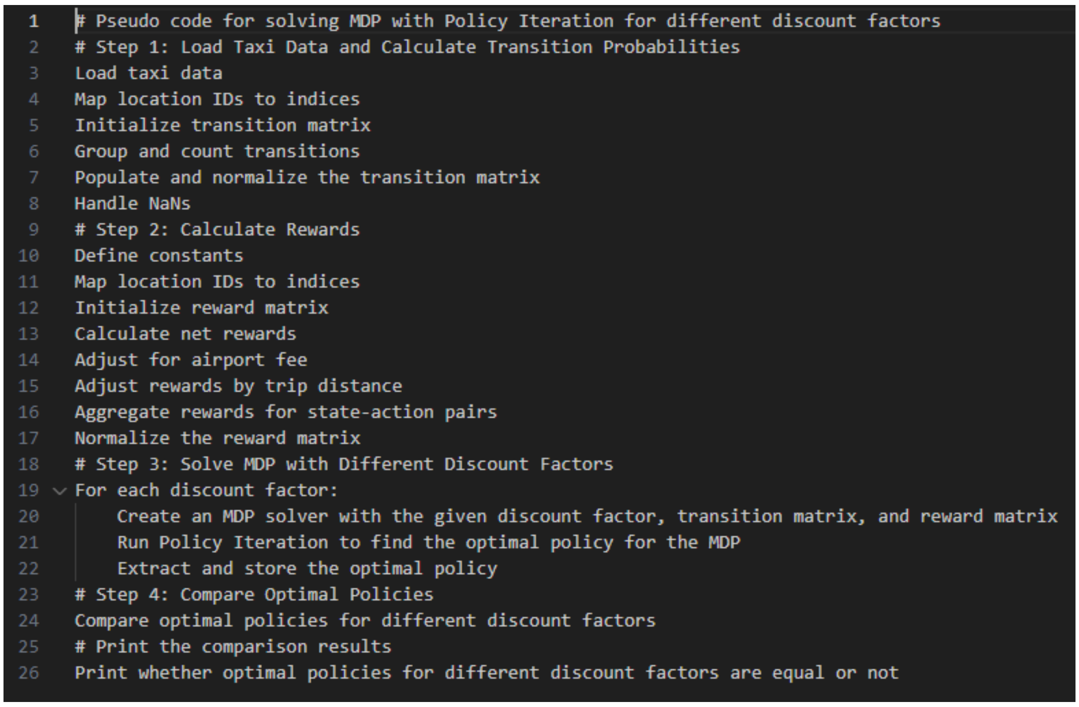
Before constructing the whole MDP code it will be a good idea to create pseudo code to see and understand how MDP components (that have been set up before) are connected and used together. It is also important to highlight that mdptoolbox library in Python can provide all necessary algorithms for solving MDP. To refresh our memory here is the short list of algorithms that this library contains:

- Value Iteration algorithm will iteratively update the value function until it will convergence. Value Iteration computes the optimal value function and then based on this value it is going to derive the policy.
- Policy Iteration algorithm alternates between policy evaluation and policy improvement steps. It tries to find the optimal policy by constantly improving the present policy until its convergence.
- Modified Policy Iteration algorithm is subtype of Policy Iteration which combines elements of both Value Iteration and Policy Iteration to get faster convergence.
- Q-Learning algorithm as has been stated in previous chapters, it learns the optimal action-value (Q-value) function by exploration and exploitation. It is often used for MDPs when we do not know the transition model.
- Relative Value Iteration algorithm is another subtype of Value iteration algorithm which can estimate the state values relative to the best state value. This can be often used in situations when the actual state values are not so important.
- Linear Programming algorithm as stated in its name solves MDP as linear programming problems. By the way this type of algorithm is useful for large-scale MDP.

This library also contains the model-free reinforcement learning algorithms like SARSA (State-Action-Reward-State-Action). (Cordwell, 2015) (sourceforge, 2016)
This library will help in trying different algorithms which have been stated in the beginning of the thesis.

Pseudocode of MDP structure in python code

Below is the pseudocode for the MDP, it should be noted that line 21 (Run Policy Iteration to find the optimal policy for the MDP) does not mean that this pseudocode only for this type of algorithm. Other algorithms will be put there as well.



```
1   # Pseudo code for solving MDP with Policy Iteration for different discount factors
2   # Step 1: Load Taxi Data and Calculate Transition Probabilities
3   Load taxi data
4   Map location IDs to indices
5   Initialize transition matrix
6   Group and count transitions
7   Populate and normalize the transition matrix
8   Handle NaNs
9   # Step 2: Calculate Rewards
10  Define constants
11  Map location IDs to indices
12  Initialize reward matrix
13  Calculate net rewards
14  Adjust for airport fee
15  Adjust rewards by trip distance
16  Aggregate rewards for state-action pairs
17  Normalize the reward matrix
18  # Step 3: Solve MDP with Different Discount Factors
19  For each discount factor:
20      Create an MDP solver with the given discount factor, transition matrix, and reward matrix
21      Run Policy Iteration to find the optimal policy for the MDP
22      Extract and store the optimal policy
23  # Step 4: Compare Optimal Policies
24  Compare optimal policies for different discount factors
25  # Print the comparison results
26  Print whether optimal policies for different discount factors are equal or not
```

Figure 6 Pseudocode for the MDP solver

To use mdptoolbox we need to install pymdptoolbox via Command Prompt (pip install mdptoolbox). After that we can use mdptoolbox in python. This toolbox has already MDP solvers like Policy Iteration, Q-learning and Value Iteration. It can be changed to different name of the solver at the end of the code.

If we look on the overall structure, it will make more sense. Lets review the general structure of the MDP model in python:

- ✓ Import all necessary libraries and load the data
- ✓ State the transition matrix defined before and check if it is stochastic
- ✓ State the reward matrix defined before
- ✓ Use MDP toolbox to execute the algorithms with different discount factors and the above matrices
- ✓ Print and compare the results

I. Value iteration algorithm for MDP

The final code was created and located in the Appendix of this work Code 6 Value Iteration for MDP. However, it is important to mention a couple of details which have been added to the transition matrix part.

Below lines of code were added with the purpose of insuring that the transition matrix is going to be 3 dimensional. It is a requirement for mdptoolbox and originates in the nature of the MDP itself. The nature of the 3D dimensions can be explained by looking on its structure. The first dimension of the 3D matrix equals each possible action while the second and the third dimensions are the transition probabilities from each single state to every other one under a specific action. In matrix below the first dimension is the same as the action due to nature of the environment and the problem.

```
# Initialize 3D transition matrix
num_locations = len(unique_locations)
transition_matrix_3d    =    np.zeros((num_locations,    num_locations,
num_locations))

# Populate the 3D transition matrix
for i in range(num_locations):
    transition_matrix_3d[i, :, :] = transition_matrix
```

In the vacant taxi problem, each location represents a state and driving to another location is a different action. It can happen that transition probabilities are the same for different actions however this format is required to fit the MDP framework where we have actions that are explicitly considered.

Below are results of the code execution:

Optimal Policy discount factor 0.9, 0.5 and 0.1 (they are identical):

(0, 244, 128, 128, 4, 0, 128, 260, 260, 260, 260, 128, 128, 260, 260, 159, 128, 260, 260, 260, 260, 260, 260, 128, 128, 260, 182, 260, 260, 67, 260, 0, 128, 0, 260, 67, 260, 260, 260, 128, 128, 128, 128, 211, 128, 136, 100, 128, 260, 128, 260, 128, 260, 53, 260, 225, 260, 166, 249, 260, 128, 128, 220, 260, 128, 128, 260, 128, 128, 260, 128, 128, 234, 128, 128, 260, 123, 147, 128, 128, 80, 225, 128, 234, 84, 96, 128, 128, 88, 128, 128, 260, 260, 260, 128, 260, 128, 260, 260, 128, 260, 128, 259, 128, 128, 260, 41, 136, 128, 128, 128, 0, 128, 225, 128, 260, 260, 260, 260, 231, 260, 128, 260, 128, 259, 260, 260, 260, 260, 260, 128, 260, 211, 128, 260, 260, 128, 128, 128, 128, 128, 128, 128, 128, 128, 260, 260, 128, 128, 260, 260, 260, 260, 128, 128, 128, 128, 128, 128, 128, 128, 128, 128, 128, 260, 260, 128, 260, 168, 260, 128, 260, 73, 260, 198, 128, 260, 128, 260, 179, 260, 128, 128, 41, 260, 128, 134, 260, 260, 189, 260, 128, 128, 128, 260, 160, 47, 128, 260, 134, 6, 231, 260, 260, 128, 205, 128, 207, 260, 128, 260, 260, 128, 260, 260, 260, 260, 260, 260, 128, 220, 128, 128, 128, 128, 128, 128, 128, 128, 128, 260, 128, 128, 128, 128, 260, 260, 260, 260, 260, 10, 128, 260, 260, 128, 128, 260, 260, 260, 128, 128, 128, 252, 128, 13, 260, 128, 128, 128, 259, 260)

Optimal policies for discount factors 0.9 and 0.5 are equal.

Optimal policies for discount factors 0.9 and 0.1 are equal.

Optimal policies for discount factors 0.5 and 0.1 are equal.

As we can see discount factor did not play a significant role at all. To understand possible reasons and get answers we will need to go to the Results and Discussion

Meanwhile let's make the final code even simpler and save optimal policies for each discount factor and 3 algorithms (Value and Policy iterations and Q-learning). It is important to mention that mdp solver is not designed for SARSA algorithm, for this reason the final code will be edited again (this will be shown later in the next chapters).

II. Policy iteration and Q-learning algorithms for MDP

Below lines of code were added to the main Code 6 Value Iteration for MDP code This will help to save time and effort in analysing optimal policies.

We will need to define 5 functions to run, save, print and compare policies

Below functions define mdp solver by 4 arguments 2 one which are not going to be changed (reward and transition matrices) and save them separately with npy extension. Then each policy will be printed separately

```
# Define function mdp solver by 4 arguments
def    run_mdp_solver(solver,    transition_matrix,    reward_matrix,
discount_factor):
        mdp_solver   =   solver(transition_matrix,   reward_matrix,
discount_factor)
    mdp_solver.run()
    return mdp_solver.policy

# Function to save  the policies
def save_policy(policy, filename):
    np.save(filename, policy)
    print(f"Saved policy to {filename}.npy")

# Function to print the policies
def print_policy(policy, description):
    print(f"{description}: {policy}")
```

Below function compares the policies by iterating through each unique pair using nested loops. It checks if all corresponding elements are equal by using the all() function. After that it stores the results in a dictionary (as True or False). The next function prints out the results (whether optimal policies are equal or not)

```
# Function to compare the policies
def compare_policies(*policies):
    comparison_results = {}
    for i in range(len(policies)):
        for j in range(i + 1, len(policies)):
            comparison_results[(i, j)] = all(p1 == p2 for p1, p2 in
zip(policies[i], policies[j]))
    return comparison_results

# Function to print comparison of the policies
def print_comparison_results(comparison_results):
    for (i, j), result in comparison_results.items():
        if result:
            print(f"Optimal policies for instances {i} and {j} are
equal.")
        else:
            print(f"Optimal policies for instances {i} and {j} are
not equal.")
```

This is final Code 7 Final MDP code to run Value, Policy iterations and Q-learning with above additions and changes.

III. SARSA algorithm for MDP

SARSA algorithm does not exist in the mdp toolbox library, however there are couple of ways how we can build SARSA for MDP. One of the easiest ways is going to be setup it separately in python file and then call it in the final code. As there are transition and reward matrices, we can copy paste them in the final code for SARSA algorithm and define only SARSA itself.

This Code 10 SARSA was designed to calculate optimal policies for multiple gamma and alpha (learning rate) values and with steps per episode that equal 1000.

```
         for _ in range(1000):  # Limit number of steps per episode
till 1000

alphas = [0.1, 0.5, 0.9] # Set up different learning rates
gammas = [0.1, 0.5, 0.9] # Set up different discount factors
episodes = 1000
```

It is important to note that neither the reward matrix nor the transition one being used in Q-learning or SARSA algorithms directly. These matrices are used more like the state-action space setup for further learning process which is going to take place later.

IV. Visualization

It will be nice to have some visual understanding of the outcome. Optimal policies can be shown in couple of ways such as:

1) Table – it is an easy and simple way to present optimal policies with smaller number of actions and state. Unfortunately, it will not be useful in case of 261 records. However usage of pivot table is going to be useful as it can show us which states are predominant.
2) Graph or bar chart can be useful in our case as it is very suitable for routes and locations. It will draw directed edges from state to state represented by its optimal action. This code Code 8 Bar chart for optimal policy will do the representation of the bar chart.
3) Heatmaps can arrange actions and states in a grid-like structure. Each cell in the heatmap will mean a state and the colour represents the preferred action. However, in our case we have 261 action and 261 states, it will be better if clustering takes place. The clustering helps to present data in more sensible way as for number of clusters we can take number of total boroughs in New-York city. Such approach simplifies by significantly reducing the number of unique states to a more manageable level. It also allows to observe the policy's high-level strategies for specific geographic regions. Finally, it simply provides generalization based on the common factor which can be applied to each state. Totally there are 6 boroughs in the city. This code performs heatmapping with clusters Code 9 Heatmapping with clusters

5 Results and Discussion

Possible limitation of the research

Possible limitations of this research
There are number of potential limitations which we might come across while preparing the master's thesis. These restrictions may include, beside others:

• Lack of data as there not so many data resources or benchmark datasets which can be used to validate simulation results.

• Complexity of building MDP as it can require significant calculating resources.

• The simulation environment, the performance metrics may not fully grasp all complexities of the real-world taxi industry and transportation system network.

• Some assumptions and simplifications can be made during the process of building model. This can lead to decreased or limited applicability or accuracy.

• Approaches may not be easily applicable among other transportation areas. This can be due to different road networks, population, some traffic patterns etc.

• Research cannot take into account for some external factors such as: weather, some events or political factors that can potentially affect the performance of the taxi routing system.

Discussion of the results of optimal policies

This Table 7 Policy dictionary has been created to easily navigate among different policies and their comparisons.

5.1.1 Optimal policies for Value iteration

Let's start with discussion of the results of the previous chapter where the MDP model provided identical policies for all 3 discount factors for Value Iteration algorithm. This can be explained by several factors:

- Domination of certain states and actions in couple with the reward structure. The reward structure favors certain transition (more rewarding from taxi's perspective) and makes them the best choice for the agent regardless of the discount factor.
- Complexity of the model or the state space that is characterized by a very large number of states and actions. Switching to the single policy with different discount

factors might indicate that our model is not sensitive to the discount factor. This is due to the rewards and the transitions that are modeled and the inherent characteristics and patterns of our taxi data.

- Imbalance between the short- and long-term rewards where the decision-making is largely driven by immediate profit. The potential rewards do not play a major role in altering the course of action.
- Special characteristics of the data source can make the optimal policy indifferent to the discount factor. Looks like the taxi data has some dominating patterns of certain routes, distances and these patterns could dominate the policy outcomes. It will lead to similar policies.
- Finally, the reason can be the rewards and the transition probabilities. This is more likely to be caused by the rewards and transitions that exhibit a certain uniformity. In our case the variation of rewards is not significant enough to provide different policies with given discounting factors.

5.1.1.1.1 Visualization of Value iteration

After running the code Code 8 Bar chart for optimal policy and Code 9 Heatmapping with clusters in Appendix only for Value iteration algorithm below pictures are available:
Heatmap with 6 clusters Figure 20 Heatmap with 6 clusters for Value iteration located in the Appendix chapter represents the heatmap with 2 diagonals x-axis is labeled as State while y-axis is Cluster. Clusters 1 and 0 has patterns, they overpopulated with Actions in yellow and greenish colors. Let’s look on the bar chart and the pivot table in order to understand which Zone IDs are preferred by the agent (as optimal policy is same for all discount factors lets will keep only one chart and heat map for this case).
Below chart was implemented in excel with file taxi_zone_lookup.csv (taken from the official website of the NYC (City of New York, 2023) By looking on the chart, it is visible that most popular and rewarding borough for the taxi driver is Manhattan and Queens. Queens is going to have 124 destinations by the agent by following this optimal policy. Manhattan is going to have 109 trips which is little bit less but still in the top. EWR - Newark Airport and Staten island boroughs are having the least number of destinations (equaling 5 each). Brooklyn is almost on the same level as two previous boroughs. Bronx is having 11 trips only and holds the 3rd top destination borough.

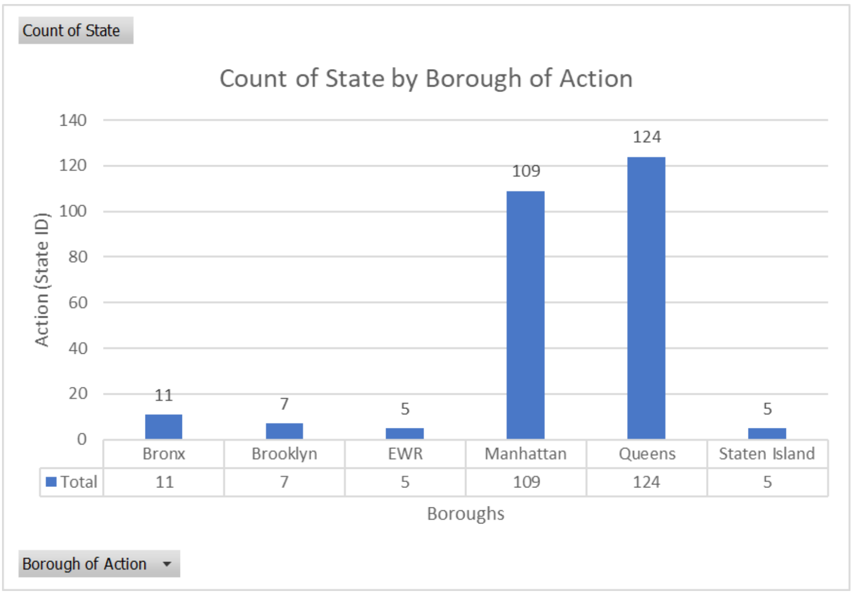


Figure 7 Bar chart of Optimal policy for Value Iteration with gamma 0.1,0,50,9

Within these boroughs exist top favorable locations. In Manhattan borough, LocationID 261 which stands for World Trade Center is having 97 of 109. Queens borough at the same time has LocationID 129 which stands for Jackson Heights that is having 110 out 124 trips. Below bar chart from the Code 8 Bar chart for optimal policy can also show the pattern of choosing optimal action by the agent. Based on this chart we can say that from most of the states (taxi zones), taxi driver will prefer taking those clients whose final destinations are either the World Trade Center or Jackson Heights as these ones provide the highest reward.

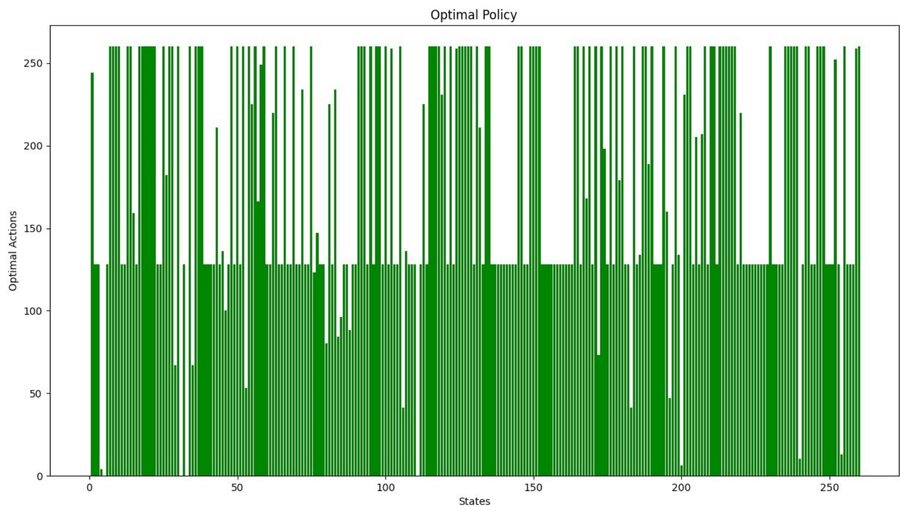
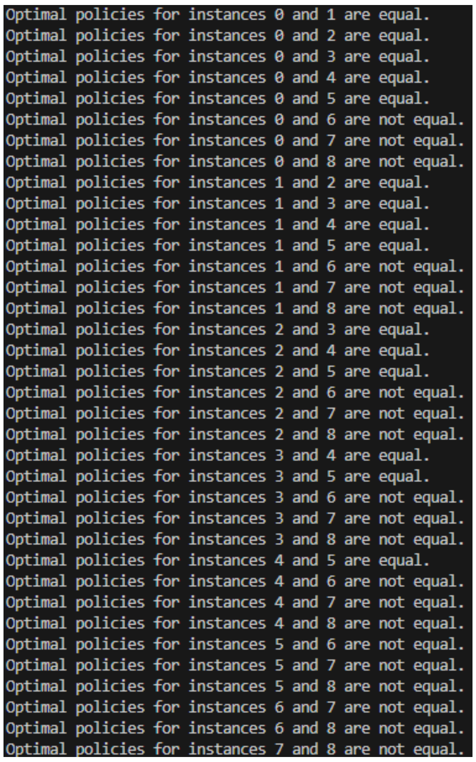


Figure 8 Bar Chart for Value iteration optimal policy with discount factor (0,9, 0,5 0,1) from python code

5.1.2 Optimal policies for Policy iteration

After running Code 7 Final MDP code to run Value, Policy iterations and Q-learning we can get the overall comparison of optimal policies among each other.



```
Optimal policies for instances 0 and 1 are equal.
Optimal policies for instances 0 and 2 are equal.
Optimal policies for instances 0 and 3 are equal.
Optimal policies for instances 0 and 4 are equal.
Optimal policies for instances 0 and 5 are equal.
Optimal policies for instances 0 and 6 are not equal.
Optimal policies for instances 0 and 7 are not equal.
Optimal policies for instances 0 and 8 are not equal.
Optimal policies for instances 1 and 2 are equal.
Optimal policies for instances 1 and 3 are equal.
Optimal policies for instances 1 and 4 are equal.
Optimal policies for instances 1 and 5 are equal.
Optimal policies for instances 1 and 6 are not equal.
Optimal policies for instances 1 and 7 are not equal.
Optimal policies for instances 1 and 8 are not equal.
Optimal policies for instances 2 and 3 are equal.
Optimal policies for instances 2 and 4 are equal.
Optimal policies for instances 2 and 5 are equal.
Optimal policies for instances 2 and 6 are not equal.
Optimal policies for instances 2 and 7 are not equal.
Optimal policies for instances 2 and 8 are not equal.
Optimal policies for instances 3 and 4 are equal.
Optimal policies for instances 3 and 5 are equal.
Optimal policies for instances 3 and 6 are not equal.
Optimal policies for instances 3 and 7 are not equal.
Optimal policies for instances 3 and 8 are not equal.
Optimal policies for instances 4 and 5 are equal.
Optimal policies for instances 4 and 6 are not equal.
Optimal policies for instances 4 and 7 are not equal.
Optimal policies for instances 4 and 8 are not equal.
Optimal policies for instances 5 and 6 are not equal.
Optimal policies for instances 5 and 7 are not equal.
Optimal policies for instances 5 and 8 are not equal.
Optimal policies for instances 6 and 7 are not equal.
Optimal policies for instances 6 and 8 are not equal.
Optimal policies for instances 7 and 8 are not equal.
```

Figure 9 Output of the Code 7 Final MDP code to run Value, Policy iterations and Q-learning

Based on Table 7 Policy dictionary it is visible that regardless of discount factors optimal policies for Value and Policy iterations are identical. This is statement needs to be investigated to understand why these policies are identical and what can be the reason for it. Due policies being identical let's not provide visualization for Policy iteration optimal policies as you can find it in the previous chapter.

5.1.2.1.1 Reasons for identicality of two polices

There can be many causes of policies being equal among each other. However, our Q-learning and SARSA policies are neither identical between each other nor among the rest. This can be explained by the nature of these algorithms. Based on the previous subchapter Unique characteristics of reinforcement learning algorithm where we have discussed unique features of each algorithm, we have also touched the nature and best scenarios of its usage.

Regarding Value iteration we have defined that it is more suitable for medium datasets due to its structure and computationally intensity. Good knowledge of reward structure and transition probabilities is required to build good quality model in both Value and Policy iterations. In our case, we tried to use all possible factors which might have significant impact in the reward structure. If we look on the reward matrix chapter again (Rewards(R)), we will see complexity which have been included in it. We have absorbed different expense and charges which might influence the net reward and then adjusted it by the trip distance to make it more realistic.

Data quality plays not the last role in the defining optimal policies in the MDP model. Based on the previous chapter Data processing, data set has been cleaned out and the difference between original number of records and formatted one equals 4,273,323 rows. Eventhough we have tried to use all filters which make sense and would filter biases or anomalities in the data set. Still the data recording process is not free from technical issues or human factor. Some part of data might be incorrect or missing which can also lead to what we have now. It is also important to mention that this data has been recorded in the middle of COVID-19 pandemic. Certain patterns and external factors could influence the optimal policies.

Another explanation can be the Vacant taxi problem itself. Our state space and actions can be simple for such algorithms and for any systematic approaches that want to find the optimal policy would give the same result. Such this might happen in environments where each state has oblivious optimal action. There is little variance in the value of different actions. Simplicity can also mean that the environment's dynamics and reward structure are proper and well defined and understood (which mean that the optimal policy is robust).

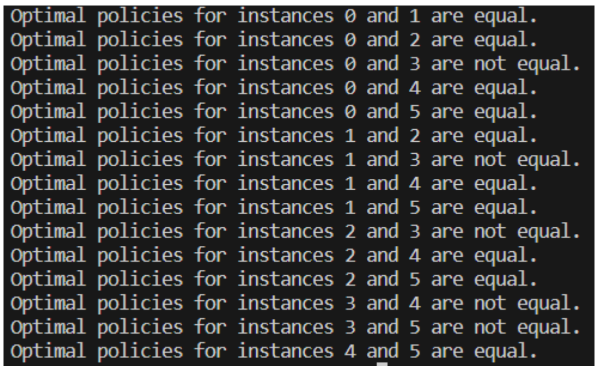
The similarity of these policies can also be served as sort of validation for the correctness of both algorithms. In case when both methods are implemented correctly and converge to the same policy, it might mean more confidence that the solution is correct. When two algorithms agree on the policy, we can imply that the solution is stable. The optimal policy is indifferent to the specifics of the algorithm. Sometimes such stability is desirable.

5.1.2.1.2 Possible solutions to diversify optimal policies for Value and Policy iterations

Discount factor (gamma) is one of the things which freely can be changed. Even though 3 values for discount factors were taken (0.9 0.5 and 0.1), we still can change the factors which tend to 0 or 1. Lets make one discount factor very close to 1 by making it equal to

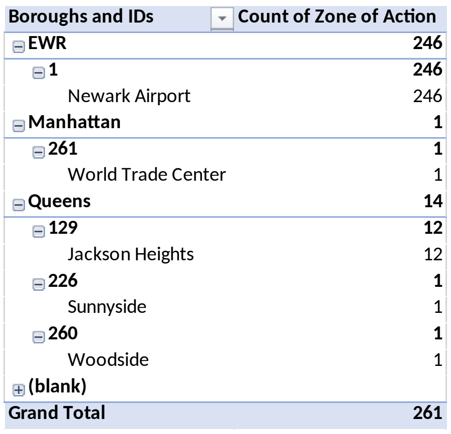
0.9999999999999999999999999999 and the other one very close to 0 by making it 0.0000000000000000000000000001. As in Python and many other programming languages, floating-point numbers always have some finite precision. After changing the discount factors (the 0.5 was kept the same as sort of benchmark) we get below output. There are 5 couples which are not identical. Based on Table 7 Policy dictionary:

1. Value iteration optimal policy with discount factor (0.999..) is not equal Policy iteration optimal policy with the same discount factor.
2. Value iteration with 0.5 factor not equal to Policy iteration with 0.999... factor
3. Value iteration with discount rate 0.0000…1 is not equal to Policy iteration with 0.999...
4. Policy iteration with 0.999... factor is not equal to Policy iteration with 0.5 factor and Policy iteration with 0.000…1 factor.


```
Optimal policies for instances 0 and 1 are equal.
Optimal policies for instances 0 and 2 are equal.
Optimal policies for instances 0 and 3 are not equal.
Optimal policies for instances 0 and 4 are equal.
Optimal policies for instances 0 and 5 are equal.
Optimal policies for instances 1 and 2 are equal.
Optimal policies for instances 1 and 3 are not equal.
Optimal policies for instances 1 and 4 are equal.
Optimal policies for instances 1 and 5 are equal.
Optimal policies for instances 2 and 3 are not equal.
Optimal policies for instances 2 and 4 are equal.
Optimal policies for instances 2 and 5 are equal.
Optimal policies for instances 3 and 4 are not equal.
Optimal policies for instances 3 and 5 are not equal.
Optimal policies for instances 4 and 5 are equal.
```
Figure 10 Output of comparisons of optimal policies for Value and Policy iterations with extremely low and high gamma

Based on the above comparisons, it is clearly visible that optimal policies for Value iterations are identical among each other and only different to the optimal policy of Policy iteration with discount factor close to 1.

Let’s take a quick look at the overview of this optimal policy. The matrix and the pivot table below represent this optimal policy. It is visible that almost all action is going to be with location ID 1 which is Newark Airport with 246 destinations (it is good to mention that due to mapping of States that starts at 0, State 0 will mean LocationID 1). Then Queens borough with only 14 actions which will lead to Jackson Heights in most cases. It is important to mention that Manhattan now has only one spot where the agent will go and it is same the World Trade Center. Such spread is indeed interesting as it indicates that with discount rate almost close to 1 the agent will prioritize future rewards only. Based on given spread LocationID 1 EWR Newark Airport will provide desired reward from most of the states (taxi zones) if EWR is destination of a client.



Boroughs and IDs	Count of Zone of Action
EWR	**246**
1	**246**
Newark Airport	246
Manhattan	**1**
261	**1**
World Trade Center	1
Queens	**14**
129	**12**
Jackson Heights	12
226	**1**
Sunnyside	1
260	**1**
Woodside	1
(blank)	
Grand Total	**261**

Figure 11 Pivot table of optimal policy of Policy iteration with discount rate close to 1

Optimal policy matrix for Policy iteration with the discount factor 0.9999999…: (0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 128, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 128, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 128, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 260, 0, 0, 0, 0, 0, 225, 0, 0, 0, 128, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 128, 0, 128, 0, 0, 0, 0, 0, 128, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 128, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 128, 0, 0, 128, 0, 0, 128, 0, 0, 128, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 259, 0)

Possible explanation of such behavior can be hidden in the approach of the Policy iteration algorithm. In case when we have set the discount factor close to 1, it means that the long-term rewards are considered almost as significantly as immediate rewards. But there is a difference in the way these 2 algorithms work. Value iteration always updates values for all states in each iteration by using the Bellman equation. While Policy iteration alternates between policy evaluation and policy improvement. Very tiny differences in the calculation processes of these algorithms can lead to different policies (especially when we have setup the discount factor so high). It is also good to remember that when setting very high discount factors, the difference in policies is expected behavior.

5.1.3 Optimal policies for Q-learning

Q-learning in contrast has diverse optimal policies for each discount factor. Lets take a look on the optimal policy with the discount factor 0.9

Optimal policy for Q-Learning with discount factor 0.9: (0, 9, 0, 54, 0, 122, 0, 0, 163, 159, 0, 0, 54, 0, 152, 0, 14, 30, 0, 188, 169, 0, 115, 0, 122, 112, 0, 0, 152, 0, 116, 0, 0, 0, 0, 223, 0, 64, 222, 0, 0, 229, 0, 0, 0, 22, 194, 0, 0, 0, 0, 0, 0, 0, 81, 211, 0, 146, 94, 0, 118, 0, 210, 0, 0, 0, 0, 159, 237, 195, 0, 0, 0, 200, 239, 0, 83, 0, 17, 144, 241, 0, 251, 0, 0, 0, 0, 0, 0, 0, 11, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 110, 54, 0, 0, 0, 0, 120, 0, 149, 0, 20, 216, 0, 153, 0, 110, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 98, 177, 0, 243, 0, 0, 179, 0, 0, 0, 186, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 239, 0, 48, 0, 0, 0, 0, 151, 0, 0, 0, 12, 0, 0, 0, 0, 0, 0, 0, 255, 0, 0, 0, 0, 162, 82, 0, 0, 0, 0, 0, 170, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 30, 0, 51, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 30, 164, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0)

From the first glance it is visible that we have kind of similar situation like in previous case of Policy iteration with discount factor close to 1. However, some diversity also can be visible. Figure 21 Heatmap with clusters for Q-learning optimal policy with 0.9 gamma in Appendix can give us some insights on it. Each cluster has state highlighted with different color.

When going through the pivot table as in the previous case, the agent will prioritize future rewards and for this reason the best choice is moving to State 1 – Newark Airport. However, there are also some other boroughs with range of 4 – 19. The highest of which is Brooklyn after which goes Manhattan and then Queens.

Row Labels	**Count of Action**
⊞ **Bronx**	**10**
⊞ **Brooklyn**	**19**
⊟ **EWR**	**196**
⊟ **Newark Airport**	**196**
1	196
⊞ **Manhattan**	**15**
⊞ **Queens**	**17**
⊞ **Staten Island**	**4**
⊞ **(blank)**	
Grand Total	**261**

Table 2 Pivot for Q-learning optimal policy with discount factor 0.9

The below bar chart can also highlight those optimal states and actions. As there are many States with ID 0 (which means LocationID is 1 due to numbering of array that starts at 0)

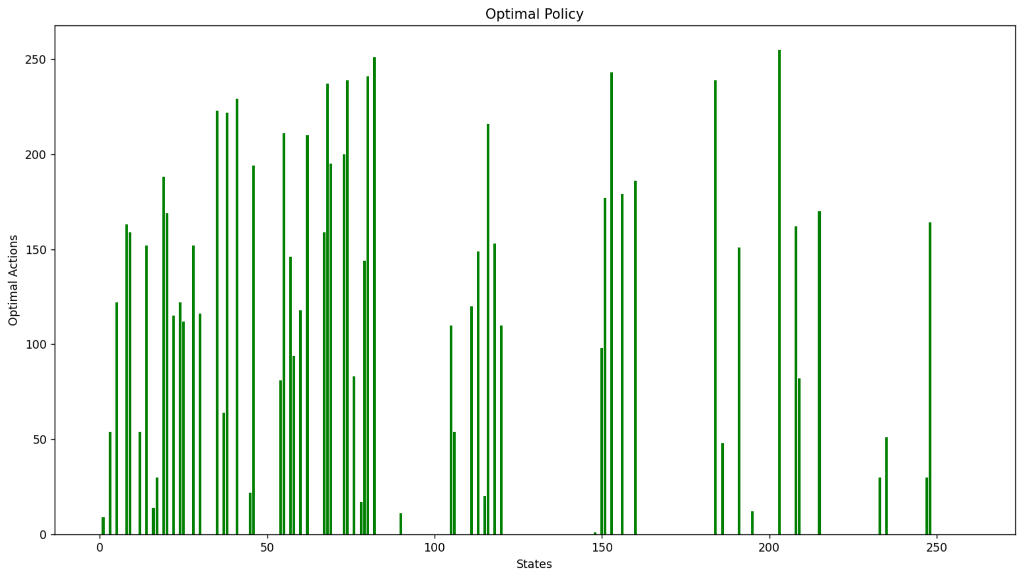


Figure 12 Bar chart of Optimal policy for Q-learning with discount factor 0.9

Let’s continue with Optimal policy for Q-Learning with discount factor 0.5:

(121, 7, 29, 0, 84, 0, 0, 79, 256, 0, 50, 0, 0, 0, 0, 0, 0, 0, 0, 0, 191, 0, 98, 0, 0, 71, 0, 0, 60, 0, 83, 46, 161, 202, 11, 159, 0, 0, 0, 0, 237, 0, 134, 46, 0, 0, 47, 0, 207, 0, 14, 0, 0, 191, 0, 0, 0, 0, 0, 0, 23, 0, 0, 0, 93, 0, 0, 181, 160, 67, 149, 0, 169, 0, 0, 0, 195, 0, 0, 55, 0, 0, 158, 91, 141, 0, 56, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 62, 36, 0, 17, 0, 0, 0, 0, 0, 0, 0, 224, 0, 0, 0, 161, 0, 0, 50, 0, 23, 255, 0, 0, 0, 0, 234, 0, 0, 0, 0, 78, 0, 0, 0, 0, 0, 60, 101, 0, 0, 0, 226, 0, 0, 0, 0, 0, 71, 0, 0, 0, 0, 103, 0, 0, 0, 95, 0, 0, 150, 0, 65, 0, 0, 0, 54, 0, 0, 0, 0, 0, 246, 0, 235, 46, 0, 8, 156, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 163, 0, 0, 44, 0, 0, 0, 0, 0, 0, 0, 31, 0, 0, 74, 0, 0, 0, 0, 0, 249, 0, 0, 0, 0, 0, 0, 0, 54, 0, 0, 0, 0, 0, 0, 67, 0, 0, 0, 38, 0, 74, 0, 0, 0, 0, 0, 0, 0, 0, 181, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0)

Again, by looking at the matrix itself we can say that moving to State 1 (EWR airport) is prevailing. Indeed if compare below pivot table with one above we can that EWR is less by only 6. Even though discount factor being equal to 0.5, the agent still prefers moving to borough EWR, Newark Airport as in case of the long term strategy. Manhattan got little bit better in terms of count of actions but still compare with the main leader this difference is very small. Which means taxi with will prefer client who is heading to Newark Airport in most of New-York locations.

Row Labels	Count of Action ID (location ids)
⊞ **Bronx**	**15**
⊞ **Brooklyn**	**17**
⊟ **EWR**	**190**
Newark Airport	190
⊞ **Manhattan**	**20**
⊞ **Queens**	**17**
⊞ **Staten Island**	**2**
⊞ **(blank)**	
Grand Total	**261**

Table 3 Pivot table for optimal policy of Q-learning with discount rate 0.5

Bar chart of optimal policy of Q-learning looks slightly different in comparison to the Figure 12 Bar chart of Optimal policy for Q-learning with discount factor 0.9 The overall pattern move to the right hand side which can be explained by increased Actions in Manhattan borough and some decreased of Newark Airport's count.

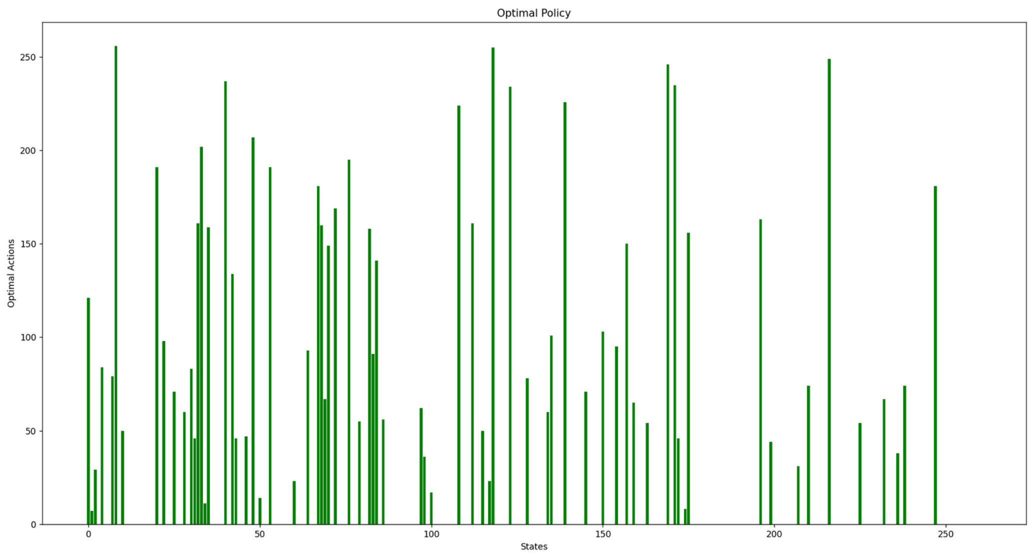


Figure 13 Bar chart of Optimal policy for Q-learning with discount factor 0.5

If we compare the Figure 18 Q learning heatmap of optimal policies with discount factor 0.5 in the Appendix we can see that it has some spread among clusters, especially clusters 1, 2, 3 and 4.

Once analysis and visualisation of the optimal policy with 0.1 gamma for Q-learning will be done, it is going to be possible compare all of them based on the total count of Actions per borough. Such overview can show us the influence of the discount factor. The last optimal policy for Q-Learning with discount factor 0.1: (0, 122, 48, 0, 96, 95, 6, 216, 115, 0, 0, 0, 0, 0, 0, 241, 0, 128, 62, 0, 25, 0, 0, 0, 0, 0, 0, 34, 0, 0, 0, 0, 0, 252, 0, 0, 161, 0, 0, 0, 0, 254, 47, 195, 0, 199, 0, 0, 0, 0, 0, 129, 91, 109, 0, 0, 173, 0, 41, 97, 0, 256, 0, 0, 0, 220, 0, 0, 0, 0, 214, 124, 0, 0, 181, 62, 0, 171, 0, 0, 123, 0, 0, 0, 0, 0, 0, 0, 38, 0, 0, 0, 87, 0, 0, 0, 0, 0, 0, 8, 0, 70, 0, 0, 0, 19, 0, 152, 67, 0, 0, 72, 0, 186, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 177, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 30, 0, 0, 0, 0, 0, 0, 0, 0, 121, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 52, 0, 0, 0, 0, 0, 0, 0, 0, 45, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 59, 0, 0, 0, 0, 0, 0, 0, 0, 0, 172, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 226, 188, 0, 0, 0, 0, 0, 0, 0,0, 0, 0, 0, 0, 0, 0, 0, 151)

Again we can see the same pattern of moving to the State 0 – LocationID 1, Newark Airport. Lets examine the bar chart to see if some movements towards certain states can be detected. It is visible that many of green bars move to the left side which can only mean that with the gamma being equal to 0.1 the agent prefers even more immediate rewarding Actions. In this case it is moving to Newar Airport.

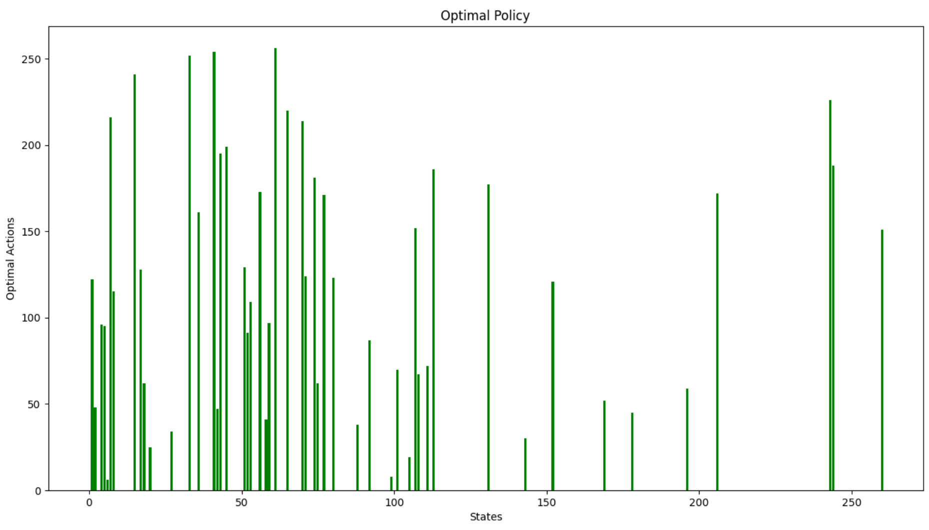
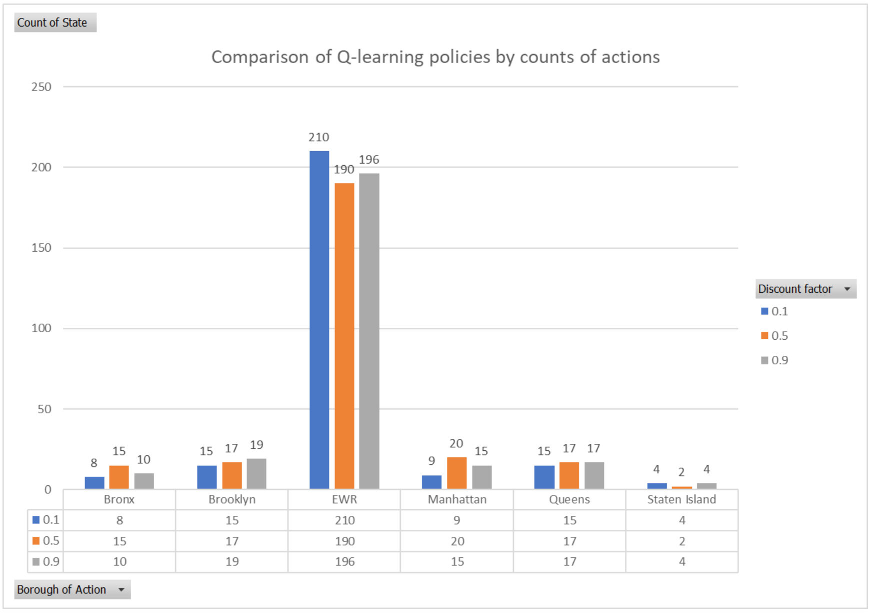


Figure 14Figure 15 Bar chart of Optimal policy for Q-learning with discount factor 0.1

By double checking with the below pivot table, it is clear that the agent is preferring to move to State 0 – LocationID 1 Newark Airport in terms of greedy policy. It is the highest number of Action taken to move to Newark airport.

Table 4 Pivot table for Q-learning optimal policies with discount rate 0.1

Row Labels	Count of Action ID (location ids)
⊞ **Bronx**	**8**
⊞ **Brooklyn**	**15**
⊟ **EWR**	**210**
Newark Airport	210
⊞ **Manhattan**	**9**
⊞ **Queens**	**15**
⊞ **Staten Island**	**4**
⊟ **(blank)**	
(blank)	
Grand Total	**261**



	Bronx	Brooklyn	EWR	Manhattan	Queens	Staten Island
🟦 0.1	8	15	210	9	15	4
🟧 0.5	15	17	190	20	17	2
⬜ 0.9	10	19	196	15	17	4

Figure 15 Comparison of Q-learning optimal policies with 3 discount factors

Now by using boroughs of the New-York city we can cluster results in easy and convenient way. It also allows to see dynamics of each discount factor for each policy. Above bar chart summarizes the total amount of specific action at specific boroughs. We could use taxi zones instead on x-axis but such chart will be too long and unreadable. It is clear that the optimal policy with the discount factor 0.1 strongly focuses on EWR borough

and Newark Airport particularly. The agent that seeks for immediate reward will choose those clients who need travel to the airport. However, when discount factor is increased and set up to 0.5 the overall picture is slightly changed. As this 0.5 gamma stands for more balanced approach between short- and long-term rewards, we can also observe it on the chart. For borough Brooklyn, Staten Island and Queens 0.5 gamma holds middle position and being sort of balance between two other policies. It is good remember that the agent often ends up choosing favourite actions with the highest rewards. Such approach can lead to developing a sort of favourable strategy once it gets more confident in its decisions. Once Q-learning updates its knowledge base we can be sure that it is always going to find the best strategy and stick to its favourable decisions. The gamma γ also play important role in Q-learning but if γ belongs to reasonable and sensible range, the agent will create its own final uniform strategy. Better not to forget the fact that for environments with high level of predictability, Q-learning will most likely come up with similar strategies.
Finally, when there are not so many options or complexities, Q-learning algorithm can instantly define and stick to the best course of actions. This algorithm go thoroughly to check out and understand the possible options.(Kerner, 2023)

5.1.4 Optimal policies for SARSA

In case of SARSA algorithm, diversity and space for manipulation are bigger. Based on previous chapter Unique characteristics of reinforcement learning algorithm, SARSA beside the discount factor (gamma) also has the learning rate (alpha). The learning rate determines how much the Q-value is updated regards to new data. High alpha makes larger updates which at the same time require the process of learning to be more volatile. High learning rate also stands for exploration which leads to more frequent updating of Q-values. While the low alpha leads to smaller updates of Q-values but at same time it is good for a stable learning process. Lower α prefers exploitation than exploration (in case of high alpha). Balance between exploration and exploitation is crucial in complex environments. As the agent must try new actions in order to discover better strategies and at the same time not go away too much from known rewarding structure. Additionally, the interactions between alpha and gamma can be setup in favourable way (the high gamma compensates for the low alpha by emphasizing the long-term rewards of exploration).
Due to high number of mixture of policies with different alpha and gamma it has been decided that SARSA is going to be shown in one pivot and bar chart to grasp all

information it can provide. Lets take the discount factor equal 0.9, 0.5, 0.1 and setup same values for our learning rate (alpha) By running Code 10 SARSA and resaving files in csv format for better representation we get below pivot table with total count of Actions for each borough and each policy.

Table 5 Pivot summary of SARSA optimal policies

Count of Action ID (location ids)	Column Labels						
Policies	**Bronx**	**Brooklyn**	**EWR**	**Manhattan**	**Queens**	**Staten Island**	**Grand Total**
policy_alpha_0.1_gamma_0.1	39	46	10	84	69	13	261
policy_alpha_0.1_gamma_0.5	41	60	6	79	58	17	261
policy_alpha_0.1_gamma_0.9	43	57	10	77	57	17	261
policy_alpha_0.5_gamma_0.1	46	43	10	86	65	11	261
policy_alpha_0.5_gamma_0.5	33	57	8	82	68	13	261
policy_alpha_0.5_gamma_0.9	41	47	10	76	65	22	261
policy_alpha_0.9_gamma_0.1	40	59	10	73	62	17	261
policy_alpha_0.9_gamma_0.5	40	51	8	83	66	13	261
policy_alpha_0.9_gamma_0.9	39	50	8	84	61	19	261
Grand Total	**362**	**470**	**80**	**724**	**571**	**142**	**2349**

Based on the above pivot table each row corresponds to a different policy with labeled name under Policies. Each policy is mixture of different alpha and gamma. While each column represents previously described

Data cells are Action counts in particular borough under special policy. These counts reflect the frequency of certain routes and decisions recommended by the policy for each single location. There is also a grand total per each policy that in sum gives 261 and Grand total for all policies which equals 2349 (all possible actions x all possible states)

The bottom row totals are counts of actions for each borough across all policies. It provides good insight into which boroughs see more overall activity according to SARSA.

Based on the given results we can say that the leader in terms of counts is Manhattan borough with 724 counts of actions, next Queens with 571 counts, then Brooklyn with 470, Bronx 362 counts, Staten Island with 142 counts and Newark Airport with 80 only. Then based on boroughs the LocationIDs are varying, for example for Manhattan m

The action counts for EWR are way lower than those for other boroughs and compared with previous algorithms. It can imply that trips to the airport are less frequent or less prioritized within the SARSA framework.

The policy variations do not dramatically change during overall distribution of actions. It can imply that that the optimal policy is quite robust. This also can mean that data set is limited in terms space of actions and states, so that different SARSA parameter settings do not lead to drastically different strategies.

The spread is relatively balanced, and action counts across policies for most boroughs, with no visible extreme variations. Such balance can mean that the SARSA algorithm is relatively stable across learning and discount settings. So, to speak the distribution of actions across boroughs is also stable and gradually spread. Figure 19 Bar chart of all SARSA optimal policies in Appendix clearly and self-explanatory represents different policies, actions counts and boroughs. We have analyzed the extensive dataset by using different approaches with special attention to the adjustment of the learning rate (alpha) and discount factor (gamma). These parameters have steered the derived policies across New York boroughs with a certain focus on Manhattan. Our analysis indicates that the strategy provided by SARSA remains consistent despite different values of alpha and gamma.

Adjusted reward matrix

Lets implement small change in the reward matrix to see if it is going change the previous patterns and prove models being sensible to different input.

The formula for the adjusted reward was changed to fit more that kind of policy of company where it focuses not only on the profitability but also efficiency (in terms of time spent by the taxi). In adjusted reward, the net_reward by each trip_distance to get a measure that takes into account profitability of the trip, travel distance and time. Division by the trip_duration normalizes the reward by the time taken. Hene the logic is following, the longer the trip takes, the smaller the adjusted reward is going to be. Below parts of code were added to the Reward matrix, the rest is same. Here is the full reward matrix code

Code 11 Adjusted reward matrix with trip duration

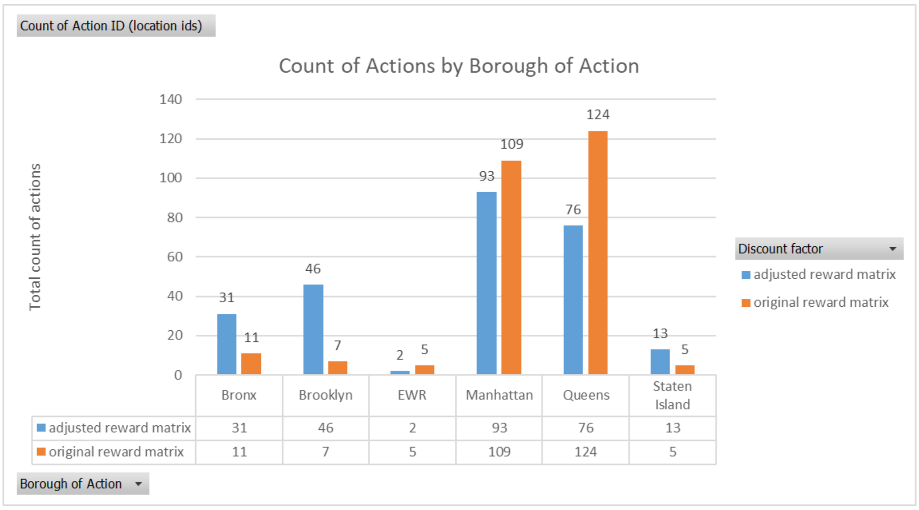
```
# Calculate trip duration in seconds
     taxi_data['trip_duration']  =  (taxi_data['dropoff_datetime']  -
taxi_data['pickup_datetime']).dt.total_seconds()

# Adding very small number to avoid division by zero
    epsilon = 1e-6
        taxi_data['adjusted_reward']   =   taxi_data['net_reward']   *
taxi_data['trip_distance'] / (taxi_data['trip_duration'] + epsilon)
```

To sum it up, the new formula is trying to estimate the efficiency of a trip from an economic standpoint. A high adjusted reward means a trip the taxi earned more money for each unit of time and each mile driven. This is going to be used as prioritization tool for trips in a decision-making process.

Optimal policies for the adjusted reward matrix

After implementing adjustments and running the final code, the results are as following. Optimal policies for all three discount factors for Value and Policy iterations are identical. Optimal policy for Policy and Value iteration with discount factor 0.9, 0.5 and 0.1: (259, 225, 2, 3, 4, 5, 6, 260, 81, 225, 260, 225, 259, 13,28, 67, 260, 31, 32, 33, 34, 36, 36, 260, 260, 39, 40, 41, 159, 211, 128, 109, 46, 47, 48, 49, 50, 24, 52, 259, 54, 55, 71, 72, 73, 259, 259, 76, 259, 78, 79, 80, 81, 128, 234, 84, 85, 86, 87, 88, 67, 90, 91, 92, 93, 94, 259, 96, 260, 260, 112, 113, 114, 115, 260, 259, 260, 119, 260, 121, 122, 259, 259, 125, 126, 259, 260, 129, 231, 92, 132, 259, 134, 135, 1, 260, 260, 151, 260, 128, 154, 81, 128, 157, 259, 159, 160, 161, 162, 163, 260, 165, 195, 167, 168, 169, 170, 260, 73, 1786, 187, 260, 189, 260, 128, 128, 193, 194, 195, 196, 197, 198, 199, 200, 73, 259, 260, 204, 205, 206, 207, 208, 0, 40, , 0, 225, 226, 227, 228, 229, 230, 231, 232, 233, 234, 260, 236, 260, 238, 239, 240, 241, 242, 243, 244, 245, 260, 260,)

Lets run the bar chart and compare 2 optimal policies.



	Bronx	Brooklyn	EWR	Manhattan	Queens	Staten Island
adjusted reward matrix	31	46	2	93	76	13
original reward matrix	11	7	5	109	124	5

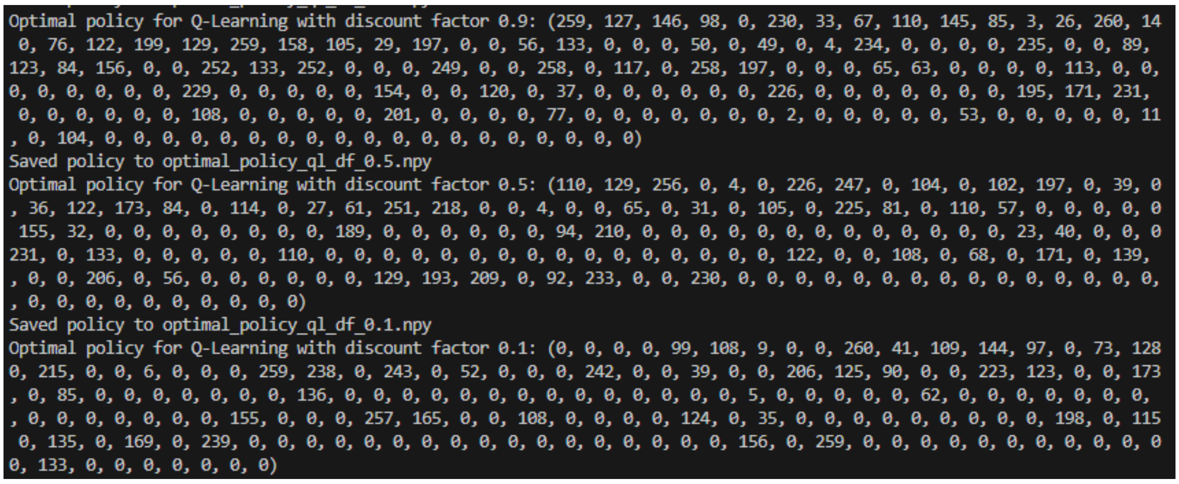
Figure 16 Adjusted matrix comparison

The above visual summary clearly shows changed optimal policy when we have adjusted the reward matrix. Especially it is visible in boroughs like Bronx, Brooklyn, and Queens (Manhattan data was changed slightly). Now the agent significantly changed mind about these boroughs and changed priorities. Even though Manhattan is still at the top of the list and it is not surprisingly (due to being cultural and financial center of the New-York plus popular tourist destination). Such change can be explained by the additional factor that MDP model takes into account (the trip duration). Apparently, Bronx and Brooklyn trips will be better destinations in terms of money earned, time and destination spent than

Queens borough, which makes our taxi driver to choose clients heading those boroughs. If we take a look on Figure 22 Bar chart of optimal policy for Policy and Value iterations with all discount factors and with adjusted reward matrix in Appendix, we will find difference in pattern too.

Adjusted Q-learning

Now let's compare Q-learning policies for the adjusted reward matrix. After execution of code, the below results are given. It is visible that each of optimal policy for Q-learning is different. Visually Action 0 – Moving to ERW airport is still preferable in all 3 policies.



```
Optimal policy for Q-Learning with discount factor 0.9: (259, 127, 146, 98, 0, 230, 33, 67, 110, 145, 85, 3, 26, 260, 14
 0, 76, 122, 199, 129, 259, 158, 105, 29, 197, 0, 0, 56, 133, 0, 0, 0, 50, 0, 49, 0, 4, 234, 0, 0, 0, 0, 235, 0, 0, 89,
123, 84, 156, 0, 0, 252, 133, 252, 0, 0, 0, 249, 0, 0, 258, 0, 117, 0, 258, 197, 0, 0, 0, 65, 63, 0, 0, 0, 0, 113, 0, 0,
0, 0, 0, 0, 0, 0, 229, 0, 0, 0, 0, 0, 154, 0, 0, 120, 0, 37, 0, 0, 0, 0, 0, 0, 226, 0, 0, 0, 0, 0, 0, 0, 195, 171, 231,
 0, 0, 0, 0, 0, 0, 108, 0, 0, 0, 0, 0, 201, 0, 0, 0, 0, 77, 0, 0, 0, 0, 0, 0, 0, 2, 0, 0, 0, 0, 0, 53, 0, 0, 0, 0, 0, 11
, 0, 104, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0)
Saved policy to optimal_policy_ql_df_0.5.npy
Optimal policy for Q-Learning with discount factor 0.5: (110, 129, 256, 0, 4, 0, 226, 247, 0, 104, 0, 102, 197, 0, 39, 0
, 36, 122, 173, 84, 0, 114, 0, 27, 61, 251, 218, 0, 0, 4, 0, 0, 65, 0, 31, 0, 105, 0, 225, 81, 0, 110, 57, 0, 0, 0, 0, 0
 155, 32, 0, 0, 0, 0, 0, 0, 0, 0, 189, 0, 0, 0, 0, 0, 0, 94, 210, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 23, 40, 0, 0, 0
231, 0, 133, 0, 0, 0, 0, 0, 110, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 122, 0, 0, 108, 0, 68, 0, 171, 0, 139,
, 0, 0, 206, 0, 56, 0, 0, 0, 0, 0, 0, 129, 193, 209, 0, 92, 233, 0, 0, 230, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0)
Saved policy to optimal_policy_ql_df_0.1.npy
Optimal policy for Q-Learning with discount factor 0.1: (0, 0, 0, 0, 99, 108, 9, 0, 0, 260, 41, 109, 144, 97, 0, 73, 128
0, 215, 0, 0, 6, 0, 0, 0, 259, 238, 0, 243, 0, 52, 0, 0, 0, 242, 0, 0, 39, 0, 0, 206, 125, 90, 0, 0, 223, 123, 0, 0, 173
, 0, 85, 0, 0, 0, 0, 0, 0, 0, 136, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 5, 0, 0, 0, 0, 0, 62, 0, 0, 0, 0, 0, 0, 0,
, 0, 0, 0, 0, 0, 0, 0, 155, 0, 0, 0, 257, 165, 0, 0, 108, 0, 0, 0, 0, 124, 0, 35, 0, 0, 0, 0, 0, 0, 0, 0, 0, 198, 0, 115
 0, 135, 0, 169, 0, 239, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 156, 0, 259, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
0, 133, 0, 0, 0, 0, 0, 0, 0)
```

Figure 17 Optimal policies of Q-learning with adjusted reward matrix

By analyzing total count of actions based on boroughs from Figure 23 Comparison chart of Original and Adjusted Q-learning optimal policies the following can be concluded:

1) EWR shows difference in the count of actions between the original and adjusted Q-learning models at lower discount factors. It might suggest that the adjusted reward matrix has a significant impact on the optimal policy leading de-emphasizing the importance of EWR zone (by taking into account additional factor in the reward matrix)
2) Manhattan has the count of actions that is relatively high across both models. However, there is a variation with different discount factors. The adjusted model does not consistently increase or decrease the total count which might mean that

indicating the effect of the reward adjustment may be complex and dependent on the discount factor.

3) In Bronx, Brooklyn, Queens and Staten Island, the adjusted Q-learning policies tend to show different patterns with each discount factors when compared to the original Q-learning model. It indicates that the adjustments in the reward matrix can be tailored to capture specific features relevant to these boroughs.
4) The sensitivity of the adjusted Q-learning policies (with gamma) in Queens has the count of actions increasing significantly with the discount factor in the adjusted model. It can mean a strategic shift in the long-term valuation of future rewards.
5) The performance of the adjusted model seems to be more balanced across boroughs in comparison with the original one. EWR and Manhattan have lower counts in some cases we have other boroughs increased. It might be desirable if the goal is to distribute service more evenly across the state.

Adjusted SARSA

Again, for comparison of different optimal policies with various alpha and gamma, the pivot table was chosen. Below is the pivot comparison table with adjusted optimal polices. As we can see the distribution of actions across the boroughs has been shifted.

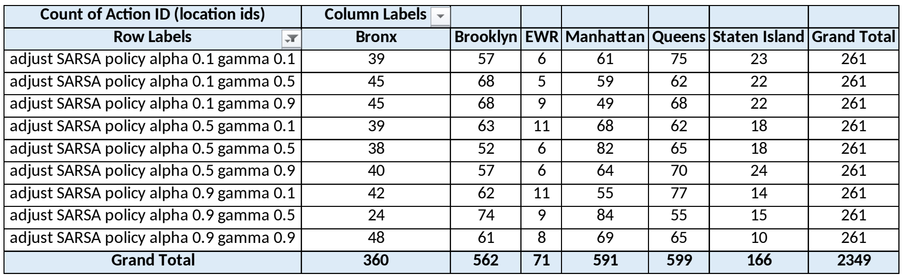
1) There is a slight increase in the number of actions in Bronx in the adjusted policy (from 360 to 362). It means that a major shift did not take place
2) For Brooklyn, the unadjusted policy shows a significant increase in actions (from 470 to 562). This could indicate that the adjusted policy is prioritizing actions in Brooklyn.
3) The count for EWR area decreased from 80 to 71 in the adjusted policy. This can suggest that the adjustments to the reward matrix have made actions associated with EWR less efficient and rewarding.
4) There is significant decrease in actions in Manhattan in the adjusted policy (from 724 to 591). This shift could indicate that the adjusted policy is placing a lower emphasis on Manhattan. It can happen due to higher rewards being assigned to other boroughs.
5) The count in Queens has increased only slightly from 571 to 599. Of course, it is not significant as the changes in Brooklyn or Manhattan however it still implies a

slight prioritizing on Queens in the adjusted policy. Same can be said about Staten Island increase (from 142 to 166) in the adjusted policy

To sum it up, the adjusted policy seems to de-prioritize Manhattan and EWR while other boroughs counts have been increased. It happens due to a reevaluation of the rewards associated with each action based on additional factors such as efficiency. The model can be changed to align with business objectives and to mirror real-world constraints and other conditions.

In general above changes in optimal policies indicate that our MDP models are sensitive to input data and it is important to setup or adjust (in advance) existing settings based on needs and objectives.

Table 6 Pivot summary of SARSA optimal policies with adjusted reward matrix



Count of Action ID (location ids)	Column Labels						
Row Labels	Bronx	Brooklyn	EWR	Manhattan	Queens	Staten Island	Grand Total
adjust SARSA policy alpha 0.1 gamma 0.1	39	57	6	61	75	23	261
adjust SARSA policy alpha 0.1 gamma 0.5	45	68	5	59	62	22	261
adjust SARSA policy alpha 0.1 gamma 0.9	45	68	9	49	68	22	261
adjust SARSA policy alpha 0.5 gamma 0.1	39	63	11	68	62	18	261
adjust SARSA policy alpha 0.5 gamma 0.5	38	52	6	82	65	18	261
adjust SARSA policy alpha 0.5 gamma 0.9	40	57	6	64	70	24	261
adjust SARSA policy alpha 0.9 gamma 0.1	42	62	11	55	77	14	261
adjust SARSA policy alpha 0.9 gamma 0.5	24	74	9	84	55	15	261
adjust SARSA policy alpha 0.9 gamma 0.9	48	61	8	69	65	10	261
Grand Total	**360**	**562**	**71**	**591**	**599**	**166**	**2349**

Possible improvements and future applications

Due to complexity and at the same time the large data set some lines of the code and model structure can be adjusted and improved by using more advanced algorithms, libraries and environments. The data from the NYC open data website will perfectly fit the current setups. It is possible in future use different years and see trends over certain periods or use data from the green taxi. However, it is important to mention that better transition matrix for Policy and Value iteration must be based on number of actual states and actions derived from raw data. In such way the researcher will be sure that he or she has included all possible states and avoid none-stochasticity issue. It is very important to address the challenges encountered with the stochastic nature of matrices derived from raw data. Researchers need to ensure that accurate data representation in stochastic models is particularly vital when dealing with complex and quite often unpredictable urban data sets (also taking into considerations some external factors that are out of researchers control). Up till now researchers preprocessing steps of data has become especially important. These steps include data cleaning, normalization and transformation, they are pivotal in stabilizing the stochastic behavior of the matrices and improving the reliability of the models. Some parts of code should be adjusted to fit intention of researcher like processing and merging data. It is not required to use only parquet format files however this type of format contains more suppressed data than any other data types.

MDP models can be used for determining optimal policies for similar route problems by taking into consideration the data structure and size. It will be useful in determining efficient policies based on different reward factors or transition probabilities.

Testing the models with different data inputs will allow researchers to gain more knowledge about performance of each model under different conditions. Testing is also a good approach in a process of identifying potential weaknesses or limitations of each model. This approach is instrumental in refining the models to better suit real-world applications, such as urban traffic management and public transportation planning.

It is worth mentioning that incorporation of more advanced statistical and probabilistic methods can lead to better handling the stochastic nature of the data. Bayesian inference or Monte Carlo simulations can offer such deeper insights and more reliable outcomes. These approaches allow to grasp more knowledge of more complex urban environments like the one in the New-York city.

Application of this work can be used in the transportation area, for example the optimized models can enhance urban traffic flows, reduce congestion and improve efficiency. It can also be used to assist with the design of more efficient public transportation systems and routes, which are based on detailed analysis of passenger flows and different demand patterns. Additionally, these approaches can bring positive impact in environmental studies by providing insights on sustainable urban policies.

Regarding future research there is a vast space of opportunities which can be explored and combined with current work. Integrating different and more complex machine learning algorithms with the current models can dramatically improve prediction accuracy in dynamic urban environments. The potential heading of the research can be adding more data from other data source providers or even different cities. By enriching models with real-time data potential researchers could make it more responsive and adaptive to changes in the urban landscape. It will open new frontiers in urban transport management and planning.

Conclusion

To summarize the whole thesis, we have explored new ways of how to optimize taxis movements around the New York city that is one the biggest megapolises in the world. We used a method called the Markov Decision Process (MDP) to understand the choices that taxi drivers make daily at their jobs. This involved processing and going through large amount of data from NYC's public records.

We found that the strategies that have been developed for the taxi drivers were quite consistent in their outcomes. This means that our approaches could reliably create effective routes for the taxi drivers in the urban areas. It was also noticed that small changes in the reward structure that taxi drivers get in our models, had great influence on their decisions. It highlights the importance of carefully considering various economic and operational factors when building these models.

Several different computer algorithms were used such as Value Iteration, Policy Iteration, Q-Learning and SARSA that helped to develop our strategies. Each of these methods had a unique way of guiding the taxis from each pickup location to its destination, it showed us the difference between short-term and long-term planning for taxi routes.

Visual tools like heatmaps, pivot tables and bar charts made our findings easier to understand and visualize. These visual images helped us in guiding and locating where taxis were most often needed in the city and how various strategies would work under different conditions.

Except academic research, the thesis can provide a practical guide for city planners, taxi companies and public transport administration. These strategies can help make taxi services more efficient and improve the overall flow of traffic in cities. Which in the same time will eventually lead to a more sustainable and efficient urban transportation system. For sure more research and development will be required to build sophisticated model, but the core base has been created.

There is still room for more research to be done. With advanced machine learning techniques and real-time data future researchers can make these models even more accurate and relevant for always changing city environment. This study can be expanded by including other cities or different types of data which could offer a broader understanding of urban transportation systems.

In summary, the research shows the power of using data to improve how urban transportation can be managed. It highlights the importance of balancing operational efficiency and financial incentives. The gained knowledge and insights could be very useful for future urban planning or smart city projects.

6 References

A. C. K. C. Chan, J. L. (2017). *Reinforcement Learning with Epsilon-Greedy Strategy for Exploration-Exploitation Trade-Off.* Japan: Computing Machinery (ACM).

Alexander, J. (2018). *thegradient*. Retrieved 1 23, 2023, from https://thegradient.pub/learning-from-humans-what-is-inverse-reinforcement-learning/

Baykal-G˝ursoy, M. (2007). *SEMI-MARKOV DECISION PROCESSES.* Piscataway, New Jersey: Rutgers University.

Brownlee, J. (2017, 12 20). *Machinelearningmastery.com*. Retrieved 01 23, 2023, from https://machinelearningmastery.com/transfer-learning-for-deep-learning/

City of New York. (2023). *NYC OpenData*. Retrieved from NYC Taxi Zones: https://data.cityofnewyork.us/Transportation/NYC-Taxi-Zones/d3c5-ddgc

City of New York. (2023). *NYC Taxi Zones*. Retrieved from NYC OpenData: https://data.cityofnewyork.us/Transportation/NYC-Taxi-Zones/d3c5-ddgc

Cordwell, S. A. (2015). *Markov Decision Process (MDP) Toolbox*. Retrieved from pymdptoolbox.readthedocs.io: https://pymdptoolbox.readthedocs.io/en/latest/api/mdptoolbox.html

databricks.com. (n.d.). *Parquet*. Retrieved from databricks.com: https://www.databricks.com/glossary/what-is-parquet#:~:text=What%20is%20Parquet%3F,handle%20complex%20data%20in%20bulk.

Fubra Limited. (n.d.). *US Top 40 Airports*. Retrieved from World Airport Codes: https://www.world-airport-codes.com/us-top-40-airports.html

geeksforgeeks.org. (2021, 6 24). *SARSA Reinforcement Learning*. Retrieved from geeksforgeeks.org: https://www.geeksforgeeks.org/sarsa-reinforcement-learning/

Gupta, S. (2018, 11 21). *Must a transition matrix from a Markov Decision Process be stochastic?* Retrieved from stackoverflow: https://stackoverflow.com/questions/43665797/must-a-transition-matrix-from-a-markov-decision-process-be-stochastic

Hui, J. (2018, 10 14). *jonathan-hui.medium.com*. Retrieved 01 23, 2023, from https://jonathan-hui.medium.com/rl-introduction-to-deep-reinforcement-learning-35c25e04c199#:~:text=The%20transition%20function%20is%20the,discuss

%20Model%2Dbased%20RL%20later.&text=The%20concepts%20in%20RL %20come,fields%20including%20the%20control%20theo

IBM. (n.d.). *What is deep learning?* Retrieved 01 27, 2023, from https://www.ibm.com/topics/deep-learning#:~:text=the%20next%20step-,What %20is%20deep%20learning%3F,from%20large%20amounts%20of%20data.

Iman Sajedian, H. L. (2019). *Double-deep Q-learning to increase the efficiency of metasurface holograms.* Scientific Reports.

Karunakaran, D. (2021, 5 21). *medium.com*. Retrieved 02 1, 2023, from https://medium.com/intro-to-artificial-intelligence/relationship-between-state-v-and-action-q-value-function-in-reinforcement-learning-bb9a988c0127

Kerner, S. M. (2023, 5). *Q-learning*. Retrieved from Machine learning platforms: https://www.techtarget.com/searchenterpriseai/definition/Q-learning#:~:text=Q %2Dlearning%20is%20a%20machine,way%20animals%20or%20children %20learn.

Kramer, A. T. (2010). Acceleration of DBSCAN-Based Clustering with Reduced Neighborhood Evaluations. *SpringerLink, KI 2010*, 195–202.

Lihong Li, O. C. (2011). An Empirical Evaluation of Thompson Sampling. *Yahoo! Research*.

Michael Hahsler, H. K. (2021). *cran.r-project.org*. Retrieved 01 29, 2023, from https://cran.r-project.org/web/packages/pomdp/vignettes/POMDP.html

Moore, L. P. (1996). Reinforcement Learning: A Survey. *JAIR*.

Murphy, K. P. (2012). *Machine Learning: A Probabilistic Perspective.* Massachusetts: The MIT Press.

NYC Taxi & Limousine Commission. (2023). *TLC Trip Record Data*. Retrieved from nyc.gov: https://www.nyc.gov/site/tlc/about/tlc-trip-record-data.page

nyc.gov. (2018, 5 18). *taxi_zone_map_bronx.jpg.* Retrieved from nyc.gov: https://www.nyc.gov/assets/tlc/images/content/pages/about/taxi_zone_map_bronx.jpg

nyc.gov. (2018, 5 18). *taxi_zone_map_manhattan.jpg.* Retrieved from nyc.gov: https://www.nyc.gov/assets/tlc/images/content/pages/about/taxi_zone_map_manhattan.jpg nyc.gov. (2018, 5 18). *taxi_zone_map_queens.jpg.* Retrieved from nyc.gov: https://www.nyc.gov/assets/tlc/images/content/pages/about/taxi_zone_map_queens.jpg

nyc.gov. (2018, 5 18). *taxi_zone_map_staten_island.jpg.* Retrieved from nyc.gov: https://www.nyc.gov/assets/tlc/images/content/pages/about/taxi_zone_map_staten_island.jpg

nyc.gov. (2922, 5 11). *data_dictionary_trip_records_yellow.pdf.* Retrieved from nyc.gov: https://www.nyc.gov/assets/tlc/downloads/pdf/data_dictionary_trip_records_yellow.pdf

oreilly.com. (n.d.). *Hands-On Reinforcement Learning with Python by Sudharsan Ravichandiran*. Retrieved from oreilly.com: https://www.oreilly.com/library/view/hands-on-reinforcement-learning/9781788836524/4a26c40f-9d5f-4fdd-804e-cf66b22005ba.xhtml

Peter Auer, N. C.-B. (2002). *Finite-time Analysis of the Multiarmed Bandit Problem.* Kluwer Academic Publishers.

QUORA. (n.d.). *How should I decide the discount factor in reinforcement learning?* Retrieved from QUORA: https://www.quora.com/How-should-I-decide-the-discount-factor-in-reinforcement-learning

Rieder, U. (2011). *Markov Decision Processes with Applications to Finance.* Springer.

Robertson, J. J. (2006). *MacTutor*. Retrieved 02 3, 2023, from https://mathshistory.st-andrews.ac.uk/Biographies/Markov/

Rouse, M. (2016, 11 30). *K-Means Clustering*. Retrieved from techopedia.com: https://www.techopedia.com/definition/32057/k-means-clustering

rule_book_current_chapter_54.pdf. (2016, 10 25). Retrieved from nyc.gov: https://www.nyc.gov/assets/tlc/downloads/pdf/rule_book_current_chapter_54.pdf

Russo, D. J. (2018). *A Tutorial on Thompson Sampling. Foundations and Trends® in Machine Learning.* 2018.

Russo, P. D. (n.d.). *Lecture 2: Infinite Horizon and Indefinite Horizon MDPs.* New-York: Columbia Business School.

SALLOUM, Z. (2018, 08 29). *Basics of Reinforcement Learning, the Easy Way*. Retrieved from Medium: https://zsalloum.medium.com/basics-of-reinforcement-learning-the-easy-way-fb3a0a44f30e#:~:text=%CE%B3%20is%20a%20discount%20factor,importance%20to%20the%20current%20state.

SALLOUM, Z. (2019). *towardsdatascience.com*. Retrieved 1 23, 2023, from https://towardsdatascience.com/exploration-in-reinforcement-learning-e59ec7eeaa751

Saul Mcleod, P. (2023). *simplypsychology.org*. Retrieved 03 01, 2023, from https://simplypsychology.org/operant-conditioning.html#:~:text=BF%20Skinner%3A%20Operant%20Conditioning,-Skinner%20is%20regarded&text=According%20to%20this%20principle%2C%20behavior,less%20likely%20to%20be%20repeated.

sourceforge. (2016, 3 8). *Documentation*. Retrieved from mdp-toolkit.sourceforge: https://mdp-toolkit.sourceforge.net/documentation.html

Surmenok, P. (2017). *towardsdatascience.com*. Retrieved 01 22, 2023, from https://towardsdatascience.com/contextual-bandits-and-reinforcement-learning-6bdfeaece72a

Sutton, A. B. (2018). *Reinforcement learning: An introduction (2nd ed.).* The MIT Press.

tankonempty.com. (n.d.). *tankonempty.com*. Retrieved from Stats for the Ford Crown Victoria: https://tankonempty.com/viewcar/Ford/Crown%20Victoria

The LinkedIn Team. (2023). *linkedin*. Retrieved 02 3, 2023, from https://www.linkedin.com/advice/0/what-benefits-drawbacks-using-decaying-epsilon

Weng, L. (2020). *github.com*. Retrieved 1 23, 2023, from https://lilianweng.github.io/posts/2020-06-07-exploration-drl/

Wong, R. (2018, 10 2). *Getting Started with Markov Decision Processes: Reinforcement Learning*. Retrieved from towardsdatascience: https://towardsdatascience.com/getting-started-with-markov-decision-processes-reinforcement-learning-ada7b4572ffb

Xinlian Yu, X. H. (2018). *A Markov Decision Process Approach to Vacant Taxi Routing with E-hailing.* Researchgate.com.

Zhang, Y. J. (2017). *Optimal empty vehicle distribution for the taxi industry using a hybrid approach. Transportation Research Part C: Emerging Technologies.* sciencedirect.com.

7 List of pictures, tables, graphs and abbreviations

List of figures

List of tables

List of equations

List of codes

Appendix

```
import pyarrow as pa #import pyarrow library
import pyarrow.parquet as pq
import pandas as pd #import pandas library
import os

# Specify the full path to the directory containing the Parquet files
input_dir = r"C:\Users\Nurbulat\Desktop\MDP\yellow"

# Creating  list of all parquet files in the directory
parquet_files    =    [os.path.join(input_dir,    f)    for    f    in
os.listdir(input_dir) if f.endswith('.parquet')]

for file in parquet_files:
    df = pq.read_table(file).to_pandas()

# processing data loop with dropping rows
dfs = []  # listing to store cleaned DataFrames
for file in parquet_files:
    df = pq.read_table(file).to_pandas()
        # capturing the original row count
    original_row_count = len(df)
     # Additional data cleaning and filtering specified in the Data
Processing chapter
    df = df[(df['PULocationID'] != "") &
        (df['DOLocationID'] != "") &
        (df['tpep_pickup_datetime'].notna()) &
        (df['tpep_dropoff_datetime'].notna()) &
        ((df['VendorID'] == 1) | (df['VendorID'] == 2)) &
           ((df['RatecodeID']  ==  1)  |  (df['RatecodeID']  ==  2)  |
(df['RatecodeID'] == 3) | (df['RatecodeID'] == 4) | (df['RatecodeID']
== 5) | (df['RatecodeID'] == 6)) &
                      ((df['store_and_fwd_flag']     ==     "Y")     |
(df['store_and_fwd_flag'] == "N")) &
        (df['passenger_count'] >= 1) &
        (df['passenger_count'] <= 4) &
        (df['fare_amount'] > 0) &
        (df['fare_amount'] < 1000) &
        (df['trip_distance'] > 0) &
        (df['trip_distance'] < 100) &
        (df['total_amount'] > 0) &
        ((df['payment_type'] == 1) | (df['payment_type'] == 2)) &
        (df['congestion_surcharge'].notna()) &
        (df['airport_fee'].notna()) &
        (df['mta_tax'].notna()) &
        (df['tip_amount'].notna()) &
        (df['extra'].notna())]

    # drop unnecessary columns
          df    =    df.drop(['RatecodeID',    'store_and_fwd_flag',
'improvement_surcharge'], axis=1)

   # Filter out rows with no transition IDs
    unique_pu = set(df['PULocationID'].unique())
    unique_do = set(df['DOLocationID'].unique())
    no_transition_pu_ids = unique_pu - unique_do
    no_transition_do_ids = unique_do - unique_pu
    df = df[~df['PULocationID'].isin(no_transition_pu_ids)]
    df = df[~df['DOLocationID'].isin(no_transition_do_ids)]

    # Capturing the filtered row count
    filtered_row_count = len(df)
    #printing the original row cound and the filtered
    print(f"File: {file}")
    print(f"Original Row Count: {original_row_count}")
    print(f"Filtered Row Count: {filtered_row_count}")

# Extract only the filename and add 'cleaned_' to it
    cleaned_filename = 'cleaned_' + os.path.basename(file)
    cleaned_file_path = os.path.join(input_dir, cleaned_filename)
    pq.write_table(pa.Table.from_pandas(df), cleaned_file_path)
    dfs.append(df)

# concatenate all cleaned dataframes
clean_df = pd.concat(dfs, ignore_index=True)

# setting display options to show all columns
pd.set_option('display.max_columns', None)

# Display cleaned dataframe with all columns
print(clean_df)

# storing cleaned dataframe to a new parque file
pq.write_table(pa.Table.from_pandas(clean_df),
os.path.join(input_dir, 'formatted_yellow_taxi_22.parquet'))

#  calculate  median,  mean,  and  standard  deviation  of  fare_amount
column
fare_median = clean_df['fare_amount'].median()
fare_mean = clean_df['fare_amount'].mean()
fare_std = clean_df['fare_amount'].std()

# analyze additional columns
# analyze most common and least common pickup hours
```

```
pickup_counts                                                        =
clean_df.groupby(clean_df['tpep_pickup_datetime'].dt.hour)
['tpep_pickup_datetime'].count()
max_pickup_hour = pickup_counts.idxmax()
min_pickup_hour = pickup_counts.idxmin()

# analyze the most and the least common passengers count
passenger_counts = clean_df['passenger_count'].value_counts()
most_common_passenger_count = passenger_counts.idxmax()
rare_passenger_count = passenger_counts.idxmin()

# analyze the most popular and the least popular pickup and dropoff
locations
pickup_counts = clean_df['PULocationID'].value_counts()
popular_pickup_locations = pickup_counts.idxmax()
unpopular_pickup_locations = pickup_counts.idxmin()
dropoff_counts = clean_df['DOLocationID'].value_counts()
popular_dropoff_locations = dropoff_counts.idxmax()
unpopular_dropoff_locations = dropoff_counts.idxmin()

# calculate median, mean, and standard deviation of trip_distance
column
distance_median = clean_df['trip_distance'].median()
distance_mean = clean_df['trip_distance'].mean()
distance_std = clean_df['trip_distance'].std()

fare_by_location = clean_df.groupby(['PULocationID', 'DOLocationID'])
['fare_amount'].mean()
max_fare_location = fare_by_location.idxmax()
min_fare_location = fare_by_location.idxmin()

# display the cleaned DataFrame with all columns
pd.set_option('display.max_columns', None)
print(clean_df)

# display the median, mean, and standard deviation of fare_amount
print('Fare Median:', fare_median)
print('Fare Mean:', fare_mean)
print('Fare Standard Deviation:', fare_std)

# display the analysis of additional columns
print('Peak pickup hour:', max_pickup_hour)
print('Off-peak pickup hour:', min_pickup_hour)
print('Most common passenger count:', most_common_passenger_count)
print('Most rare passenger count:', rare_passenger_count)
print('Distance Median:', distance_median)
print('Distance Mean:', distance_mean)
print('Popular pickup locations:', popular_pickup_locations)
print('Unpopular pickup locations:', unpopular_pickup_locations)
print('Popular dropoff locations:', popular_dropoff_locations )
print('Unpopular dropoff locations:', unpopular_dropoff_locations)
print('Distance Standard Deviation:', distance_std)
print('Location with highest average fare:', max_fare_location)
print('Location with lowest average fare:', min_fare_location)
```

Code 3 Clean and prepare data, calculate and analyze columns with statisti

```
# Load libraries
import pandas as pd
import numpy as np

# Load the Parquet file
taxi_data                                                            =
pd.read_parquet('yellow/formatted_yellow_taxi_22.parquet')

# Extracting all unique location IDs from the dataframe
unique_locations        =        pd.unique(taxi_data[['PULocationID',
'DOLocationID']].values.ravel('K'))

#  Creates  the  dictionary  with  all  unique  location  IDs  from  the
dataframe
location_to_index   =   {loc_id:   index   for   index,   loc_id   in
enumerate(unique_locations)}

# Set up transition matrix with 2 dimensions
num_locations = len(unique_locations)

# initializes the transition matrix with dimensions num_locations
transition_matrix = np.zeros((num_locations, num_locations))

# Group by PULocationID and DOLocationID and count transitions
transition_counts         =         taxi_data.groupby(['PULocationID',
'DOLocationID']).size().reset_index(name='count')

# Populate the transition matrix
for _, row in transition_counts.iterrows():
     pu_index = location_to_index[row['PULocationID']] #retrives each
Pickup and dropoff location ids indecies
    do_index = location_to_index[row['DOLocationID']]
    transition_matrix[pu_index, do_index] = row['count']

# Normalize the matrix to get probabilities by converting counts into
probabilities of transitioning from one location to another and then
insures that each row sum equals 1
transition_matrix            =            np.divide(transition_matrix,
transition_matrix.sum(axis=1, keepdims=True),
                              out=np.zeros_like(transition_matrix),
                                                where=transition_matrix.sum(axis=1,
keepdims=True) != 0)

# Replace nans with uniform distribution.
transition_matrix   =   np.nan_to_num(transition_matrix,   nan=1.0   /
num_locations)

# Validate the matrix and print if it is stochastic or not
if np.allclose(transition_matrix.sum(axis=1), 1):
    print("The transition matrix is stochastic.")
else:
    print("The transition matrix is not stochastic.")
```

Code 4 Transition matrix

```
#import libraries
import numpy as np
import pandas as pd

#define the path
file_path = 'formatted_yellow_taxi_22.parquet'
taxi_data = pd.read_parquet(file_path)

def calculate_rewards_optimized(taxi_data):
    # Setting constants for airport locations and fee
    JFK_AIRPORT_ID = 132
    LAGUARDIA_AIRPORT_ID = 138
    AIRPORT_FEE = 1.25

      #  Get  all  unique  location  IDs  from  both  PULocationID  and
DOLocationID
      all_locations  =  np.union1d(taxi_data['PULocationID'].unique(),
taxi_data['DOLocationID'].unique())
    all_locations.sort()  # Sort for consistent indexing

    # Map location IDs to indices in the matrix
         location_to_index   =   {loc:   idx   for   idx,   loc   in
enumerate(all_locations)}

    # Initialize the reward matrix
    num_states = len(all_locations)
    reward_matrix = np.zeros((num_states, num_states))

    # Calculate net rewards
         taxi_data['net_reward']   =   taxi_data['total_amount']   -
taxi_data['extra']         -         taxi_data['mta_tax']         -
taxi_data['congestion_surcharge']
```

```
    # Adjusting for airport fee
        taxi_data.loc[taxi_data['DOLocationID'].isin([JFK_AIRPORT_ID,
LAGUARDIA_AIRPORT_ID]), 'net_reward'] -= AIRPORT_FEE

    # Adjust the reward by trip distance
        taxi_data['adjusted_reward']   =   taxi_data['net_reward']   *
taxi_data['trip_distance']

    # Aggregate rewards for each state-action pair
         aggregated_rewards    =    taxi_data.groupby(['PULocationID',
'DOLocationID'])['adjusted_reward'].sum().reset_index()

    # Populate the reward matrix
    for _, row in aggregated_rewards.iterrows():
        pu_index = location_to_index[row['PULocationID']]
        do_index = location_to_index[row['DOLocationID']]
        reward_matrix[pu_index, do_index] = row['adjusted_reward']

    # Normalize the reward matrix to avoid extremely large values
    reward_matrix /= taxi_data.shape[0]

    return reward_matrix

reward_matrix = calculate_rewards_optimized(taxi_data)

# printing  reward matrix
print(reward_matrix)
```
Code 5 Reward matrix

```
import pandas as pd #import libraries
import numpy as np
import mdptoolbox


# Loading taxi data
file_path = 'formatted_yellow_taxi_22.parquet'
taxi_data = pd.read_parquet(file_path)


# State transition matrix part:

# Extracting all unique location IDs from the dataframe
unique_locations        =        pd.unique(taxi_data[['PULocationID',
'DOLocationID']].values.ravel('K'))

#  Creates  the  dictionary  with  all  unique  location  IDs  from  the
dataframe
```

```
location_to_index   =   {loc_id:   index   for   index,   loc_id   in
enumerate(unique_locations)}

# Set up transition matrix with 2 dimensions
num_locations = len(unique_locations)

# Group by PULocationID and DOLocationID and count transitions
transition_matrix = np.zeros((num_locations, num_locations))

# Group by PULocationID and DOLocationID and count transitions
transition_counts         =         taxi_data.groupby(['PULocationID',
'DOLocationID']).size().reset_index(name='count')

# Populate the transition matrix
for _, row in transition_counts.iterrows():
    pu_index = location_to_index[row['PULocationID']]
    do_index = location_to_index[row['DOLocationID']]
    transition_matrix[pu_index, do_index] = row['count']

# Normalize the matrix to get probabilities
transition_matrix           =           np.divide(transition_matrix,
transition_matrix.sum(axis=1, keepdims=True),
                              out=np.zeros_like(transition_matrix),
                                  where=transition_matrix.sum(axis=1,
keepdims=True) != 0)

# Replace NaNs with a uniform distribution (or self-loop)
transition_matrix   =   np.nan_to_num(transition_matrix,   nan=1.0   /
num_locations)

# Validate the matrix
if np.allclose(transition_matrix.sum(axis=1), 1):
    print("The transition matrix is stochastic.")
else:
    print("The transition matrix is not stochastic.")

# Initialize a 3D transition matrix
num_locations = len(unique_locations)
transition_matrix_3d    =    np.zeros((num_locations,    num_locations,
num_locations))

# Populate the 3D transition matrix
for i in range(num_locations):
    transition_matrix_3d[i, :, :] = transition_matrix

def calculate_rewards_optimized(taxi_data):
    # Constants for airport locations and fee
    JFK_AIRPORT_ID = 132
    LAGUARDIA_AIRPORT_ID = 138
    AIRPORT_FEE = 1.25

      # Get all unique location IDs from both PULocationID and
DOLocationID
      all_locations = np.union1d(taxi_data['PULocationID'].unique(),
taxi_data['DOLocationID'].unique())
    all_locations.sort()  # Sort for consistent indexing

    # Map location IDs to indices in the matrix
        location_to_index = {loc: idx for idx, loc in
enumerate(all_locations)}

    # Initialize the reward matrix
    num_states = len(all_locations)
    reward_matrix = np.zeros((num_states, num_states))

    # Calculate net rewards
         taxi_data['net_reward'] = taxi_data['total_amount'] -
taxi_data['extra'] - taxi_data['mta_tax'] -
taxi_data['congestion_surcharge']

    # Adjust for airport fee
        taxi_data.loc[taxi_data['DOLocationID'].isin([JFK_AIRPORT_ID,
LAGUARDIA_AIRPORT_ID]), 'net_reward'] -= AIRPORT_FEE

    # Adjust reward by trip distance
        taxi_data['adjusted_reward'] = taxi_data['net_reward'] *
taxi_data['trip_distance']

    # Aggregate rewards for each state-action pair
         aggregated_rewards = taxi_data.groupby(['PULocationID',
'DOLocationID'])['adjusted_reward'].sum().reset_index()

    # Populate the reward matrix
    for _, row in aggregated_rewards.iterrows():
        pu_index = location_to_index[row['PULocationID']]
        do_index = location_to_index[row['DOLocationID']]
        reward_matrix[pu_index, do_index] = row['adjusted_reward']

    # Normalize the reward matrix to avoid extremely large values
    reward_matrix /= taxi_data.shape[0]

    return reward_matrix

reward_matrix = calculate_rewards_optimized(taxi_data)
```

```
# Setting up and solving the MDP with discount factor 0.9 by using
mdptoolbox
discount_factor = 0.9
mdp_solver   =   mdptoolbox.mdp.PolicyIteration(transition_matrix_3d,
reward_matrix, discount_factor)
mdp_solver.run()

# Setting up and solving the MDP with discount factor 0.5 by using
mdptoolbox
discount_factor = 0.5
mdp_solver2   =   mdptoolbox.mdp.PolicyIteration(transition_matrix_3d,
reward_matrix, discount_factor)
mdp_solver2.run()

# Setting up and solving the MDP with discount factor 0.1 by using
mdptoolbox
discount_factor = 0.1
mdp_solver3   =   mdptoolbox.mdp.PolicyIteration(transition_matrix_3d,
reward_matrix, discount_factor)
mdp_solver3.run()

# Extracting and displaing optimal policy with discount factor 0.9
optimal_policy = mdp_solver.policy
print("Optimal Policy discount factor 0.9:", optimal_policy)

# Extracting and displaing optimal policy with discount factor 0.5
optimal_policy2 = mdp_solver2.policy
print("Optimal Policy discount factor 0.5:", optimal_policy2)

# Extracting and displaing optimal policy with discount factor 0.1
optimal_policy3 = mdp_solver3.policy
print("Optimal Policy discount factor 0.1:", optimal_policy3)

# Check if policies are equal
policies_equal_0_9_0_5 = all(policy1 == policy2 for policy1, policy2
in zip(mdp_solver.policy, mdp_solver2.policy))
policies_equal_0_9_0_1 = all(policy1 == policy2 for policy1, policy2
in zip(mdp_solver.policy, mdp_solver3.policy))
policies_equal_0_5_0_1 = all(policy1 == policy2 for policy1, policy2
in zip(mdp_solver2.policy, mdp_solver3.policy))

# Print the comparison of results (the optimal policies)
if policies_equal_0_9_0_5:
     print("Optimal  policies  for  discount  factors  0.9  and  0.5  are
equal.")
else:
    print("Optimal policies for discount factors 0.9 and 0.5 are not
equal.")
```

```
if policies_equal_0_9_0_1:
     print("Optimal  policies  for  discount  factors  0.9  and  0.1  are
equal.")
else:
    print("Optimal policies for discount factors 0.9 and 0.1 are not
equal.")

if policies_equal_0_5_0_1:
     print("Optimal  policies  for  discount  factors  0.5  and  0.1  are
equal.")
else:
    print("Optimal policies for discount factors 0.5 and 0.1 are not
equal.")
```
Code 6 Value Iteration for MDP

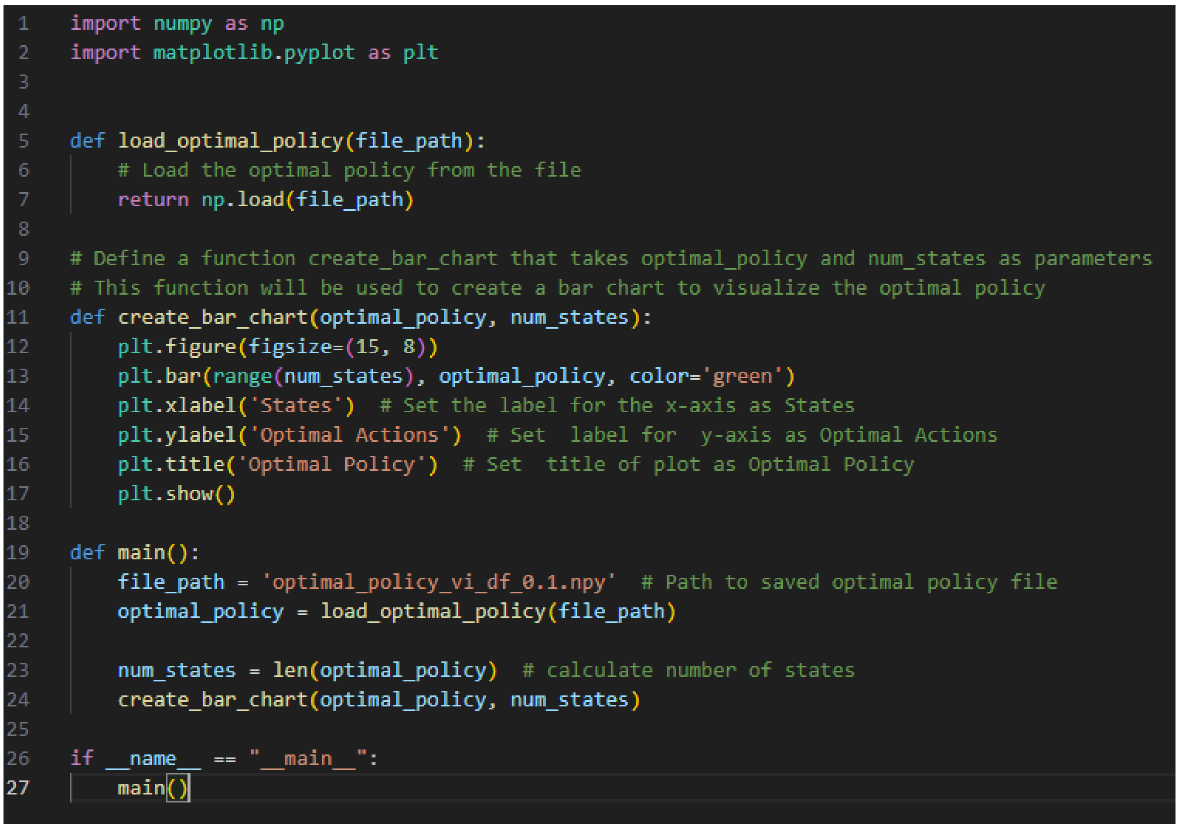
```
#import libraries
import pandas as pd
import numpy as np
import mdptoolbox

# Define function mdp solver by 4 arguments
def    run_mdp_solver(solver,    transition_matrix,    reward_matrix,
discount_factor):
         mdp_solver    =    solver(transition_matrix,    reward_matrix,
discount_factor)
    mdp_solver.run()
    return mdp_solver.policy

# Function to save  the policies
def save_policy(policy, filename):
    np.save(filename, policy)
    print(f"Saved policy to {filename}.npy")

# Function to print the policies
def print_policy(policy, description):
    print(f"{description}: {policy}")

# Function to compare the policies
def compare_policies(*policies):
    comparison_results = {}
    for i in range(len(policies)):
        for j in range(i + 1, len(policies)):
             comparison_results[(i, j)] = all(p1 == p2 for p1, p2 in
zip(policies[i], policies[j]))
    return comparison_results

# Function to print comparison of the policies
```

```
def print_comparison_results(comparison_results):
    for (i, j), result in comparison_results.items():
        if result:
            print(f"Optimal policies for instances {i} and {j} are
equal.")
        else:
            print(f"Optimal policies for instances {i} and {j} are
not equal.")

# Loading taxi data
file_path = 'yellow/formatted_yellow_taxi_22.parquet'
taxi_data = pd.read_parquet(file_path)

# State transition matrix part:

# Extracting all unique location IDs from the dataframe
unique_locations        =        pd.unique(taxi_data[['PULocationID',
'DOLocationID']].values.ravel('K'))

#  Creates  the  dictionary  with  all  unique  location  IDs  from  the
dataframe
location_to_index   =   {loc_id:   index   for   index,   loc_id   in
enumerate(unique_locations)}

# Set up transition matrix with 2 dimensions
num_locations = len(unique_locations)

# initializes the transition matrix with dimensions num_locations
transition_matrix = np.zeros((num_locations, num_locations))

# Group by PULocationID and DOLocationID and count transitions
transition_counts         =         taxi_data.groupby(['PULocationID',
'DOLocationID']).size().reset_index(name='count')

# Populate the transition matrix
for _, row in transition_counts.iterrows():
    pu_index = location_to_index[row['PULocationID']]
    do_index = location_to_index[row['DOLocationID']]
    transition_matrix[pu_index, do_index] = row['count']

# Normalize the matrix to get probabilities
transition_matrix            =            np.divide(transition_matrix,
transition_matrix.sum(axis=1, keepdims=True),
                              out=np.zeros_like(transition_matrix),
                                  where=transition_matrix.sum(axis=1,
keepdims=True) != 0)

# Replace NaNs with a uniform distribution (or self-loop)
```

```
transition_matrix = np.nan_to_num(transition_matrix, nan=1.0 /
num_locations)

# Validate the matrix and print if it is stochastic or not
if np.allclose(transition_matrix.sum(axis=1), 1):
    print("The transition matrix is stochastic.")
else:
    print("The transition matrix is not stochastic.")

# Initialize 3D transition matrix
num_locations = len(unique_locations)
transition_matrix_3d = np.zeros((num_locations, num_locations,
num_locations))

# Populate the 3D transition matrix
for i in range(num_locations):
    transition_matrix_3d[i, :, :] = transition_matrix

# Reward matrix part:

def calculate_rewards_optimized(taxi_data):
    # Setting constants for airport locations and fee
    JFK_AIRPORT_ID = 132
    LAGUARDIA_AIRPORT_ID = 138
    AIRPORT_FEE = 1.25

      # Get all unique location IDs from both PULocationID and
DOLocationID
      all_locations = np.union1d(taxi_data['PULocationID'].unique(),
taxi_data['DOLocationID'].unique())
    all_locations.sort()  # Sort for consistent indexing

    # Map location IDs to indices in the matrix
        location_to_index = {loc: idx for idx, loc in
enumerate(all_locations)}

    # Initialize the reward matrix
    num_states = len(all_locations)
    reward_matrix = np.zeros((num_states, num_states))

    # Calculate net rewards
        taxi_data['net_reward'] = taxi_data['total_amount'] -
taxi_data['extra'] - taxi_data['mta_tax'] -
taxi_data['congestion_surcharge']

    # Adjust for airport fee
        taxi_data.loc[taxi_data['DOLocationID'].isin([JFK_AIRPORT_ID,
LAGUARDIA_AIRPORT_ID]), 'net_reward'] -= AIRPORT_FEE
```

```
    # Adjust reward by trip distance
        taxi_data['adjusted_reward']   =   taxi_data['net_reward']   *
taxi_data['trip_distance']

    # Aggregate rewards for each state-action pair
         aggregated_rewards    =    taxi_data.groupby(['PULocationID',
'DOLocationID'])['adjusted_reward'].sum().reset_index()

    # Populate the reward matrix
    for _, row in aggregated_rewards.iterrows():
        pu_index = location_to_index[row['PULocationID']]
        do_index = location_to_index[row['DOLocationID']]
        reward_matrix[pu_index, do_index] = row['adjusted_reward']

    # Normalize the reward matrix to avoid extremely large values
    reward_matrix /= taxi_data.shape[0]

    return reward_matrix

reward_matrix = calculate_rewards_optimized(taxi_data)


# Set up discount factors
discount_factors = [0.9, 0.5, 0.1]
# Run Value iteration with different discount factors
policies_vi = []
for i, df in enumerate(discount_factors):
          policy    =    run_mdp_solver(mdptoolbox.mdp.ValueIteration,
transition_matrix_3d, reward_matrix, df)
    policies_vi.append(policy)
    save_policy(policy, f"optimal_policy_vi_df_{df}")
     print_policy(policy,  f"Optimal  policy  for  Value  iteration  with
discount factor {df}")

# Run Policy iteration with different discount factors
policies_pi = []
for i, df in enumerate(discount_factors):
         policy    =    run_mdp_solver(mdptoolbox.mdp.PolicyIteration,
transition_matrix_3d, reward_matrix, df)
    policies_pi.append(policy)
    save_policy(policy, f"optimal_policy_pi_df_{df}")
     print_policy(policy,  f"Optimal  policy  for  Policy  iteration  with
discount factor {df}")

# Run Q-Learning with different discount factors
policies_ql = []
for i, df in enumerate(discount_factors):
```

```
            policy     =     run_mdp_solver(mdptoolbox.mdp.QLearning,
transition_matrix_3d, reward_matrix, df)
    policies_ql.append(policy)
    save_policy(policy, f"optimal_policy_ql_df_{df}")
      print_policy(policy,  f"Optimal  policy  for  Q-Learning  with
discount factor {df}")

# Compare all policies
comparison_results  =  compare_policies(*policies_vi,  *policies_pi,
*policies_ql)
print_comparison_results(comparison_results)
```
Code 7 Final MDP code to run Value, Policy iterations and Q-learning


```
import numpy as np
import matplotlib.pyplot as plt


def load_optimal_policy(file_path):
    # Load the optimal policy from the file
    return np.load(file_path)

# Define a function create_bar_chart that takes optimal_policy and num_states as parameters
# This function will be used to create a bar chart to visualize the optimal policy
def create_bar_chart(optimal_policy, num_states):
    plt.figure(figsize=(15, 8))
    plt.bar(range(num_states), optimal_policy, color='green')
    plt.xlabel('States')  # Set the label for the x-axis as States
    plt.ylabel('Optimal Actions')  # Set  label for  y-axis as Optimal Actions
    plt.title('Optimal Policy')  # Set  title of plot as Optimal Policy
    plt.show()

def main():
    file_path = 'optimal_policy_vi_df_0.1.npy'  # Path to saved optimal policy file
    optimal_policy = load_optimal_policy(file_path)

    num_states = len(optimal_policy)  # calculate number of states
    create_bar_chart(optimal_policy, num_states)

if __name__ == "__main__":
    main()
```

Code 8 Bar chart for optimal policy

```
from sklearn.cluster import KMeans
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
import pandas as pd

# loading optimal policy
optimal_policy = np.load('optimal_policy_pi_df_0.1.npy')

# reshaping our policy for clustering
policy_for_clustering = optimal_policy.reshape(-1, 1)
```

```
# start clustering
num_clusters = 6  #based on number of boroughs clusters will be 6
kmeans                    =                    KMeans(n_clusters=num_clusters,
random_state=1).fit(policy_for_clustering)

# Get cluster labels for each state
cluster_labels = kmeans.labels_

# Create dataframe for quick manipulation
clustered_data                    =                    pd.DataFrame({'State':
np.arange(len(optimal_policy)),  'Policy':  optimal_policy,  'Cluster':
cluster_labels})
clustered_data.sort_values(by='Cluster', inplace=True)

# Create 2dimensions array for the heatmap
heatmap_data = np.zeros((num_clusters, len(optimal_policy)))

# Iteration over clusters
for cluster in range(num_clusters):
     # Getting states which belong to current cluster
     states_in_cluster  =  clustered_data[clustered_data['Cluster']  ==
cluster]['State']
    # Iteration over states
    for state in states_in_cluster:
           heatmap_data[cluster,  state]  =  clustered_data.loc[state,
'Policy']

# Plotting the heatmap
plt.figure(figsize=(15, 10))
sns.heatmap(heatmap_data, cmap="viridis")
plt.title("Heatmap of Optimal Policies with Clustering")
plt.xlabel("State")
plt.ylabel("Cluster")
plt.show()
```

Code 9 Heatmapping with clusters

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D

def load_taxi_data(file_path):
    try:
        return pd.read_parquet(file_path)
    except Exception as e:
        print(f"Error loading the file: {e}")
        return None
```

```
# Transition Matrix
def initialize_matrices(taxi_data):
         unique_locations    =    pd.unique(taxi_data[['PULocationID',
'DOLocationID']].values.ravel('K'))
      location_to_index  =  {loc_id:  index  for  index,  loc_id  in
enumerate(unique_locations)}
    num_locations = len(unique_locations)
    transition_matrix = np.zeros((num_locations, num_locations))
          transition_counts    =    taxi_data.groupby(['PULocationID',
'DOLocationID']).size().reset_index(name='count')

    for _, row in transition_counts.iterrows():
        pu_index = location_to_index[row['PULocationID']]
        do_index = location_to_index[row['DOLocationID']]
        transition_matrix[pu_index, do_index] = row['count']

             transition_matrix      =      np.divide(transition_matrix,
transition_matrix.sum(axis=1, keepdims=True),

out=np.zeros_like(transition_matrix),
                                  where=transition_matrix.sum(axis=1,
keepdims=True) != 0)
     transition_matrix  =  np.nan_to_num(transition_matrix,  nan=1.0  /
num_locations)

    if np.allclose(transition_matrix.sum(axis=1), 1):
        print("The transition matrix is stochastic.")
    else:
        print("The transition matrix is not stochastic.")

# Reward Matrix Code
    reward_matrix = calculate_rewards_optimized(taxi_data)

    return transition_matrix, reward_matrix

def calculate_rewards_optimized(taxi_data):
    JFK_AIRPORT_ID = 132
    LAGUARDIA_AIRPORT_ID = 138
    AIRPORT_FEE = 1.25

     all_locations  =  np.union1d(taxi_data['PULocationID'].unique(),
taxi_data['DOLocationID'].unique())
    all_locations.sort()

        location_to_index    =    {loc:    idx    for    idx,    loc    in
enumerate(all_locations)}
```

```
    num_states = len(all_locations)
    reward_matrix = np.zeros((num_states, num_states))

        taxi_data['net_reward']   =   taxi_data['total_amount']   -
taxi_data['extra']         -         taxi_data['mta_tax']         -
taxi_data['congestion_surcharge']
        taxi_data.loc[taxi_data['DOLocationID'].isin([JFK_AIRPORT_ID,
LAGUARDIA_AIRPORT_ID]), 'net_reward'] -= AIRPORT_FEE
        taxi_data['adjusted_reward']   =   taxi_data['net_reward']   *
taxi_data['trip_distance']

        aggregated_rewards   =   taxi_data.groupby(['PULocationID',
'DOLocationID'])['adjusted_reward'].sum().reset_index()

    for _, row in aggregated_rewards.iterrows():
        pu_index = location_to_index[row['PULocationID']]
        do_index = location_to_index[row['DOLocationID']]
        reward_matrix[pu_index, do_index] = row['adjusted_reward']

    reward_matrix /= taxi_data.shape[0]

    return reward_matrix

# Setup  Q-table with zeros it maps state-action pairs to values
def  sarsa_algorithm(transition_matrix,  reward_matrix,  num_states,
num_actions, episodes, alpha, gamma):
    Q = np.zeros((num_states, num_actions))

    for episode in range(episodes):
        # Initialize state randomly for each new episode
        state = np.random.randint(0, num_states)

        # Choose action from state using policy from Q
        action = np.random.randint(0, num_actions)  # Example: random
action

        for _ in range(1000):  # Limit number of steps per episode
till 1000
            # Take action then observe new state and reward
            new_state = np.argmax(transition_matrix[state, action])
            reward = reward_matrix[state, action]

               # Choose new action from new state by using policy
derived from Q
                 new_action  =  np.random.randint(0,  num_actions)   #
Example: random action
```

```
              # Updating Q-value for  current state and action pair
using  SARSA update rule - pair (new_state, new_action).
             Q[state, action] = Q[state, action] + alpha * (reward +
gamma * Q[new_state, new_action] - Q[state, action])

            # Moving to new state and action for  next iteration
            state, action = new_state, new_action

    return Q

# Extract  optimal policy from  Q-table
def extract_policy(Q):
    #for each state choose action with highest Q-value
    return np.argmax(Q, axis=1)

# Plot the extracted policy as a function of states.
def plot_policy(policy, alpha, gamma):
    plt.figure(figsize=(12, 6))
    plt.plot(policy, marker='o')
    plt.title(f'Policy for alpha={alpha}, gamma={gamma}')
    plt.xlabel('States')
    plt.ylabel('Chosen Action')
    plt.grid(True)
    plt.show()

def main():
    file_path = 'yellow/formatted_yellow_taxi_22.parquet'
    taxi_data = load_taxi_data(file_path)

    if taxi_data is not None:
        # Plot the extracted policy as a function of states.
                          transition_matrix,      reward_matrix      =
initialize_matrices(taxi_data)

        alphas = [0.1, 0.5, 0.9] # Set up different learning rates
        gammas = [0.1, 0.5, 0.9] # Set up different discount factors
        episodes = 1000

        # Run SARSA algorithm with actual alpha and gamma values
        for alpha in alphas:
            for gamma in gammas:
                Q = sarsa_algorithm(transition_matrix, reward_matrix,
len(transition_matrix), transition_matrix.shape[1], episodes, alpha,
gamma)
                policy = extract_policy(Q)
                plot_policy(policy, alpha, gamma)

                # Save  policies to the main path
```

```
                                                policy_file_name     =
f"policy_alpha_{alpha}_gamma_{gamma}.npy"
                np.save(policy_file_name, policy)
                print(f"Policy saved as {policy_file_name}")

if __name__ == "__main__":
    main()
```

Code 10 SARSA

Table 7 Policy dictionary

Abbreviations	**Policy name**
Policy 0	Value iteration with 0.9
Policy 1	Value iteration with 0.5
Policy 2	Value iteration with 0.1
Policy 3	Policy iteration with 0.9
Policy 4	Policy iteration with 0.5
Policy 5	Policy iteration with 0.1
Policy 6	Q-learning with 0.9
Policy 7	Q-learning with 0.5
Policy 8	Q-learning with 0.1

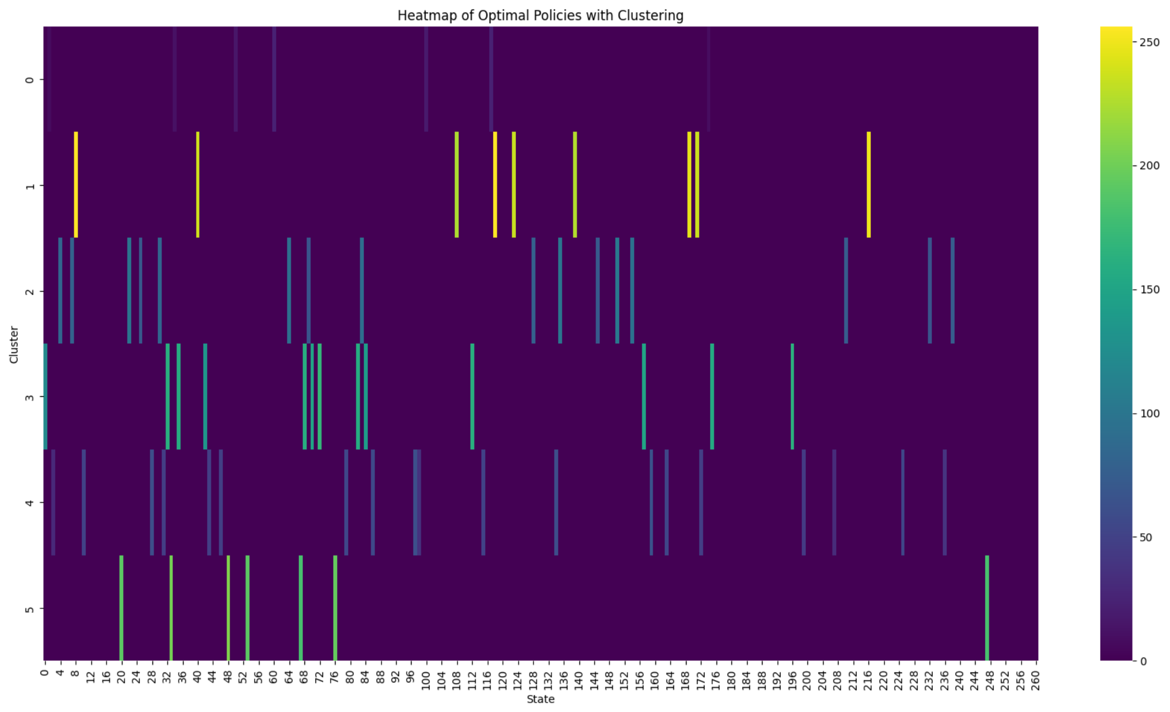
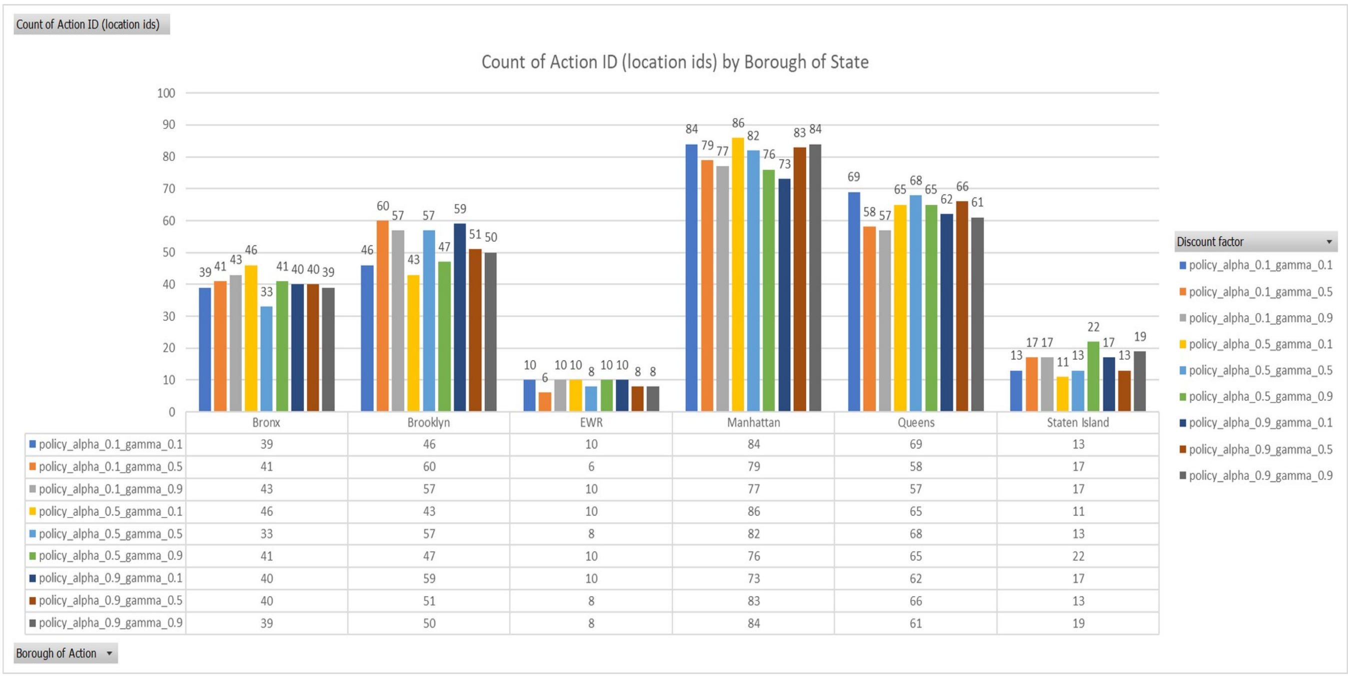


Figure 18 Q learning heatmap of optimal policies with discount factor 0.5



	Bronx	Brooklyn	EWR	Manhattan	Queens	Staten Island
policy_alpha_0.1_gamma_0.1	39	46	10	84	69	13
policy_alpha_0.1_gamma_0.5	41	60	6	79	58	17
policy_alpha_0.1_gamma_0.9	43	57	10	77	57	17
policy_alpha_0.5_gamma_0.1	46	43	10	86	65	11
policy_alpha_0.5_gamma_0.5	33	57	8	82	68	13
policy_alpha_0.5_gamma_0.9	41	47	10	76	65	22
policy_alpha_0.9_gamma_0.1	40	59	10	73	62	17
policy_alpha_0.9_gamma_0.5	40	51	8	83	66	13
policy_alpha_0.9_gamma_0.9	39	50	8	84	61	19

Figure 19 Bar chart of all SARSA optimal policies

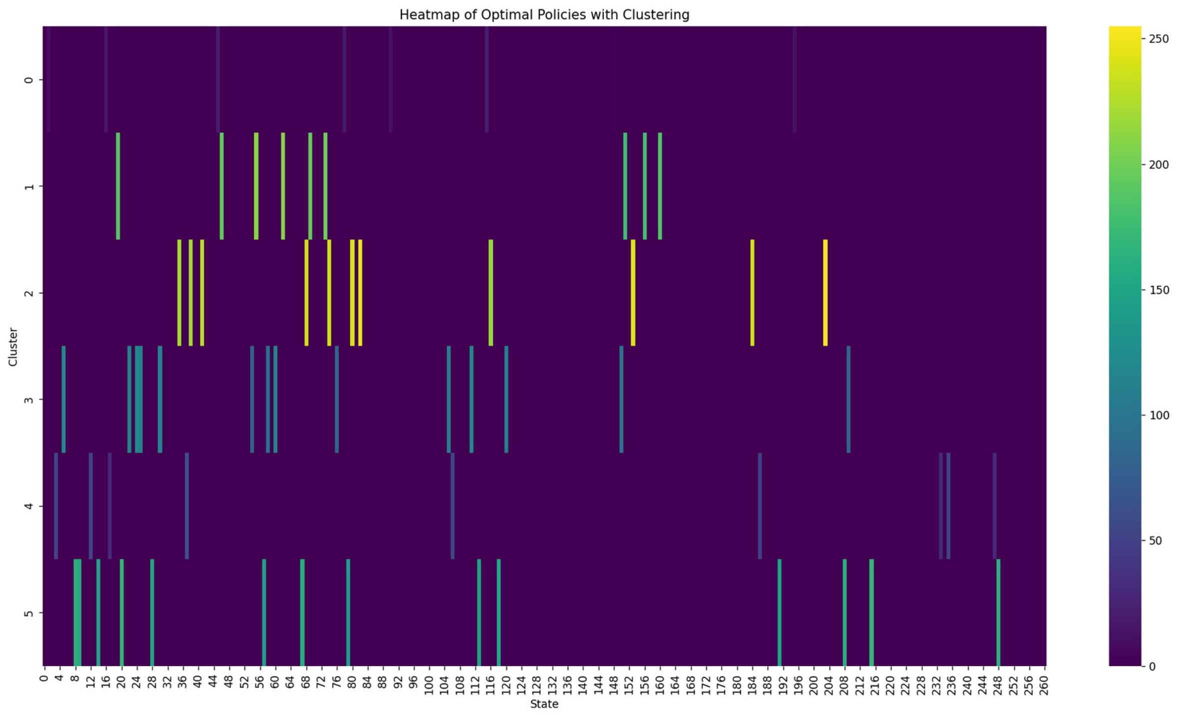

Heatmap of Optimal Policies with Clustering
Cluster
0
1
2
3
4
5
State
0
4
8
12
16
20
24
28
32
36
40
44
48
52
56
60
64
68
72
76
80
84
88
92
96
100
104
108
112
116
120
124
128
132
136
140
144
148
152
156
160
164
168
172
176
180
184
188
192
196
200
204
208
212
216
220
224
228
232
236
240
244
248
252
256
260
250
200
150
100
50
0

Figure 21 Heatmap with clusters for Q-learning optimal policy with 0.9 gamma

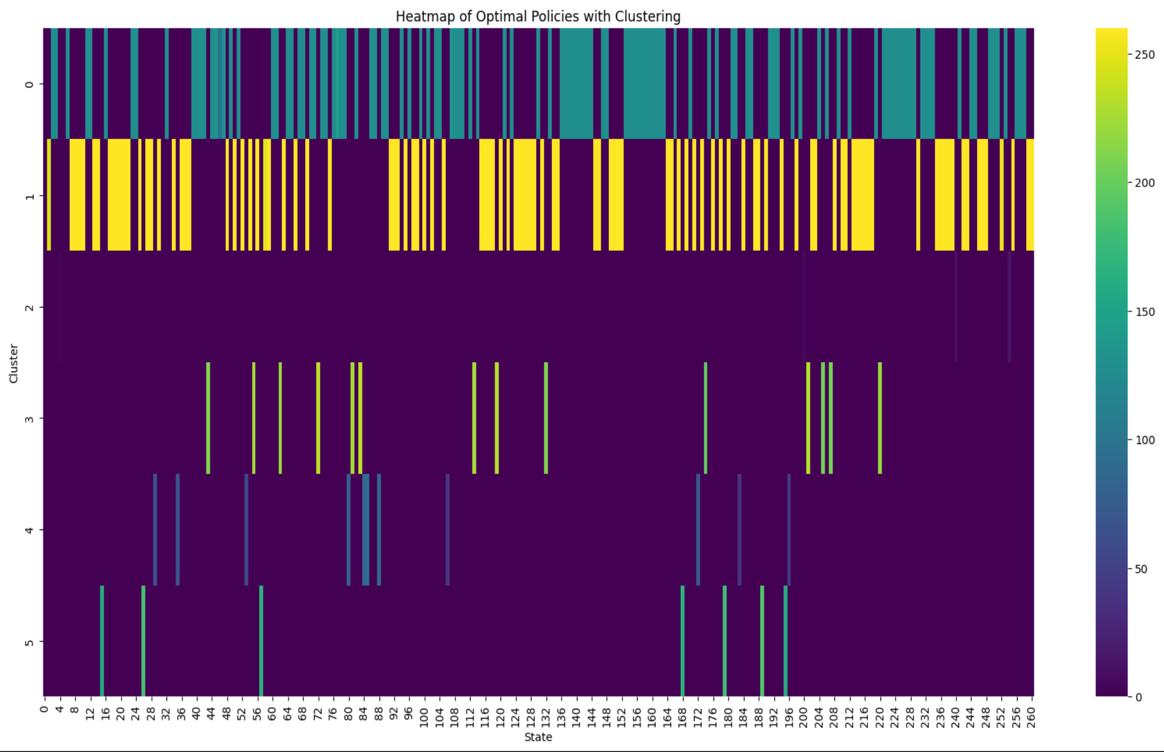


Figure 20 Heatmap with 6 clusters for Value iteration

```
def calculate_rewards_optimized(taxi_data):
     # Constants for airport locations and fee
    JFK_AIRPORT_ID = 132
    LAGUARDIA_AIRPORT_ID = 138
    AIRPORT_FEE = 1.25

    # Convert timestamps to datetime if not already
                            taxi_data['pickup_datetime']            =
pd.to_datetime(taxi_data['tpep_pickup_datetime'])
                           taxi_data['dropoff_datetime']            =
pd.to_datetime(taxi_data['tpep_dropoff_datetime'])

    # Calculate trip duration in seconds
      taxi_data['trip_duration'] = (taxi_data['dropoff_datetime'] -
taxi_data['pickup_datetime']).dt.total_seconds()

    # Unique location IDs from both PULocationID and DOLocationID
      all_locations = np.union1d(taxi_data['PULocationID'].unique(),
taxi_data['DOLocationID'].unique())
    all_locations.sort()  # Sort for consistent indexing

    # Map location IDs to indices in the matrix
        location_to_index = {loc: idx for idx, loc in
enumerate(all_locations)}
```

```
    # Initialize the reward matrix
    num_states = len(all_locations)
    reward_matrix = np.zeros((num_states, num_states))

    # Calculate net rewards including the trip duration
         taxi_data['net_reward']    =    taxi_data['total_amount']    -
taxi_data['extra']           -           taxi_data['mta_tax']           -
taxi_data['congestion_surcharge']

    # Adjust for airport fee
         taxi_data.loc[taxi_data['DOLocationID'].isin([JFK_AIRPORT_ID,
LAGUARDIA_AIRPORT_ID]), 'net_reward'] -= AIRPORT_FEE

     # Small number to avoid division by zero
    epsilon = 1e-6
        taxi_data['adjusted_reward']   =   taxi_data['net_reward']   *
taxi_data['trip_distance'] / (taxi_data['trip_duration'] + epsilon)

    # Aggregate rewards for each state-action pair
         aggregated_rewards    =    taxi_data.groupby(['PULocationID',
'DOLocationID'])['adjusted_reward'].sum().reset_index()

    # Populate the reward matrix
    for _, row in aggregated_rewards.iterrows():
        pu_index = location_to_index[row['PULocationID']]
        do_index = location_to_index[row['DOLocationID']]
        reward_matrix[pu_index, do_index] = row['adjusted_reward']

    # Normalize the reward matrix
    reward_matrix /= taxi_data.shape[0]

       assert   np.all(np.isfinite(reward_matrix)),   "Reward   matrix
contains non-finite values"


    return reward_matrix
```
Code 11 Adjusted reward matrix with trip duration

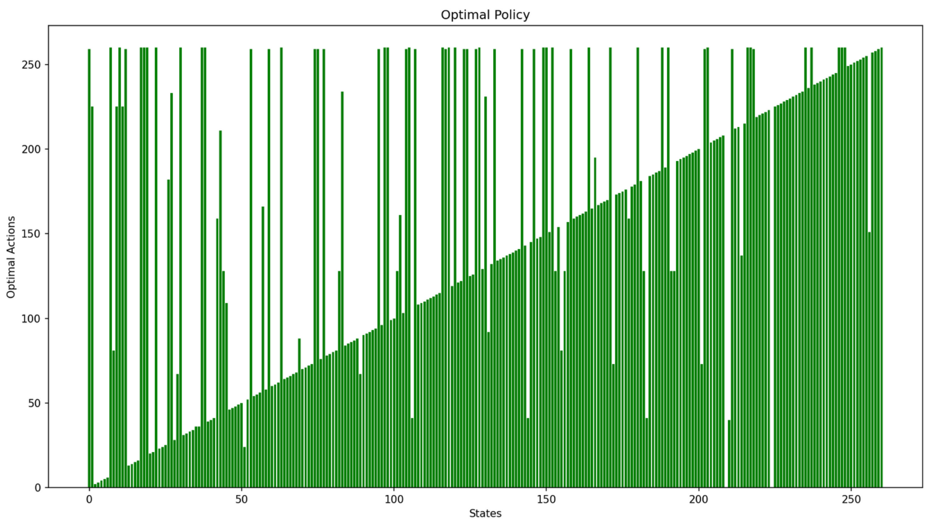
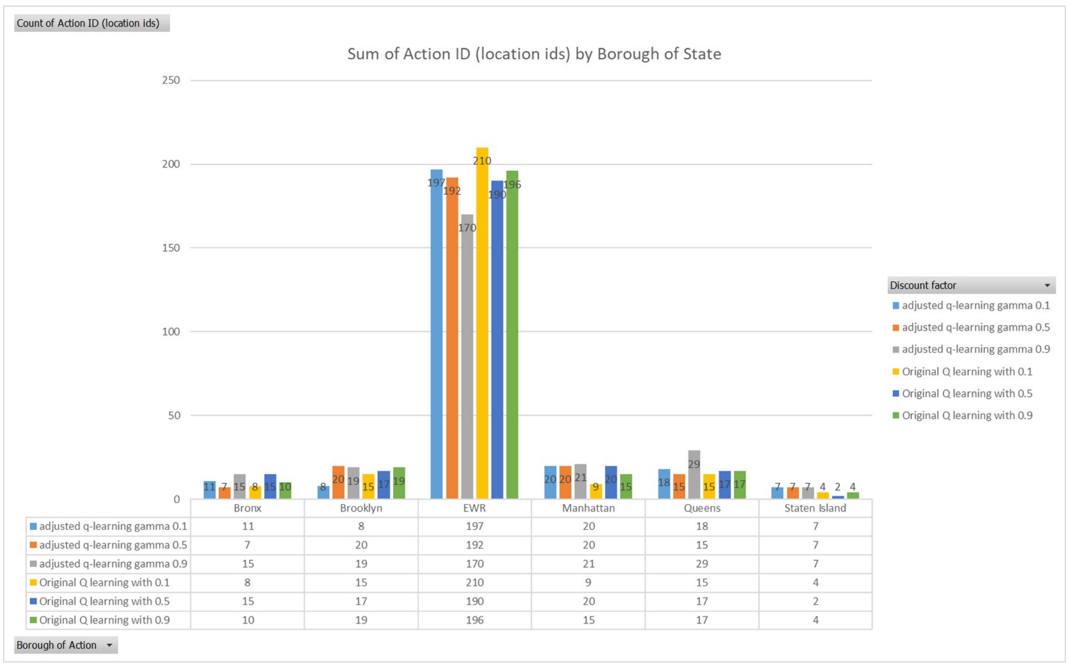


Figure 22 Bar chart of optimal policy for Policy and Value iterations with all discount factors and with adjusted reward matrix



	Bronx	Brooklyn	EWR	Manhattan	Queens	Staten Island
adjusted q-learning gamma 0.1	11	8	197	20	18	7
adjusted q-learning gamma 0.5	7	20	192	20	15	7
adjusted q-learning gamma 0.9	15	19	170	21	29	7
Original Q learning with 0.1	8	15	210	9	15	4
Original Q learning with 0.5	15	17	190	20	17	2
Original Q learning with 0.9	10	19	196	15	17	4

Figure 23 Comparison chart of Original and Adjusted Q-learning optimal policies