FAKULTA MECHATRONIKY,
INFORMATIKY A MEZIOBOROVÝCH
STUDIÍ TUL

Bakalářská práce

Python jako scriptovací jazyk pro .NET runtime prostředí

Studijní program: B0613A140005 Informační technologie
Studijní obor: Aplikovaná informatika

Autor práce: **Jaroslav Bělina**
Vedoucí práce: Ing. Jan Kraus, Ph.D.
Ústav mechatroniky a technické informatiky

Liberec 2024

FAKULTA MECHATRONIKY,
INFORMATIKY A MEZIOBOROVÝCH
STUDIÍ TUL

Zadání bakalářské práce

Python jako scriptovací jazyk pro .NET runtime prostředí

Jméno a příjmení:	**Jaroslav Bělina**
Osobní číslo:	M19000007
Studijní program:	B0613A140005 Informační technologie
Specializace:	Aplikovaná informatika
Zadávající katedra:	Ústav mechatroniky a technické informatiky
Akademický rok:	2023/2024

Zásady pro vypracování:

1. Seznamte se s možnostmí skriptování obecných úloh v runtime prostředí .NET aplikací s využitím jazyka Python či jiného běžně používaného skriptovacího jazyka.
2. Vyberte vhodné kandidáty pro pilotní implementaci jednoduchých ukázkových skriptů – jednoduchých výpočetních úloh nad posloupnostmi měřených hodnot z databáze – a s jejich využitím analyzujte základní vlastnosti a výkonostní parametry jednotlivých variant.
3. Zvolte optimální technologii a implementujte v ní alespoň jednu netriviální analytickou úlohu, která nad daty z/v .NET runtime vhodným způsobem využije knihovnu pro strojové učení ke klasifikaci posloupnosti měřených hodnot.
4. V závěru práce přehlednou formou shrňte svá zjištění a diskutujte další možnosti praktického využití vašich výsledků.

Rozsah grafických prací:	dle potřeby dokumentace
Rozsah pracovní zprávy:	30 až 40 stran
Forma zpracování práce:	tištěná/elektronická
Jazyk práce:	čeština

Seznam odborné literatury:

[1] ROSE, Anthony; GRAHAM, Scott; KRASNOV, Jacob. IronNetInjector: Weaponizing. NET Dynamic Language Runtime Engines. *Digital Threats: Research and Practice*, 2023.

[2] MEIER, Remigius; GROSS, Thomas R. Reflections on the compatibility, performance, and scalability of parallel Python. In: *Proceedings of the 15th ACM SIGPLAN international symposium on dynamic languages*. 2019. p. 91-103.

[3] DE LA TORRE, Fernanda, et al. LLMR: Real-time Prompting of Interactive Worlds using Large Language Models. *arXiv preprint arXiv:2309.12276*, 2023.

[4] HOCKLEY, Devon; WILLIAMSON, Carey. Benchmarking Runtime Scripting Performance in WebAssembly. 2022.

[5] LITVINAVICIUS, Taurius. Introduction to Blazor. In: *Exploring Blazor: Creating Server-side and Client-side Applications in. NET 7*. Berkeley, CA: Apress, 2022. p. 1-5.

Vedoucí práce:	Ing. Jan Kraus, Ph.D. Ústav mechatroniky a technické informatiky
Datum zadání práce:	12. října 2023
Předpokládaný termín odevzdání:	14. května 2024

L.S.

prof. Ing. Zdeněk Plíva, Ph.D.
děkan

doc. Ing. Josef Černohorský, Ph.D.
vedoucí ústavu

V Liberci dne 12. října 2023

Prohlášení

Prohlašuji, že svou bakalářskou práci jsem vypracoval samostatně jako původní dílo s použitím uvedené literatury a na základě konzultací s vedoucím mé bakalářské práce a konzultantem.

Jsem si vědom toho, že na mou bakalářskou práci se plně vztahuje zákon č. 121/2000 Sb., o právu autorském, zejména § 60 – školní dílo.

Beru na vědomí, že Technická univerzita v Liberci nezasahuje do mých autorských práv užitím mé bakalářské práce pro vnitřní potřebu Technické univerzity v Liberci.

Užiji-li bakalářskou práci nebo poskytnu-li licenci k jejímu využití, jsem si vědom povinnosti informovat o této skutečnosti Technickou univerzitu v Liberci; v tomto případě má Technická univerzita v Liberci právo ode mne požadovat úhradu nákladů, které vynaložila na vytvoření díla, až do jejich skutečné výše.

Současně čestně prohlašuji, že text elektronické podoby práce vložený do IS/STAG se shoduje s textem tištěné podoby práce.

Beru na vědomí, že má bakalářská práce bude zveřejněna Technickou univerzitou v Liberci v souladu s § 47b zákona č. 111/1998 Sb., o vysokých školách a o změně a doplnění dalších zákonů (zákon o vysokých školách), ve znění pozdějších předpisů.

Jsem si vědom následků, které podle zákona o vysokých školách mohou vyplývat z porušení tohoto prohlášení.

1. března 2024 Jaroslav Bělina

Python jako scriptovací jazyk pro .NET runtime prostředí

Abstrakt

Tato práce se zaměřuje na problematiku skriptování v prostředí .NET a zkoumá možnosti integrace jazyka Python s tímto prostředím. Nejprve jsou představeny základní koncepty skriptování v .NET Runtime prostředí a výhody tohoto přístupu. Dále je podrobně rozebrán jazyk Python a prostředí .NET, včetně jeho klíčových technologií.

Druhá část je věnována různým možnostem skriptování v prostředí .NET, například použití C#, IronPython a Python.NET. Jsou popsány nástroje jako Csi.exe REPL, IPy.exe REPL, a také jejich integrace pomocí NuGet balíčků.

Praktická část práce se zabývá porovnáním funkcí a vlastností mezi IronPython a Python.NET. Jsou zde analyzovány výsledky zátěžových testů, doba převodu mezi Python a .NET typy nebo tvorba AST při překladu IronPythonu. Práce dále popisuje vytvořené prostředí pro skriptování jazykem Python a ukazuje její aplikaci v praxi pomocí modelu lineární regrese.

Klíčová slova: CLI, CLR, DLR, IronPython, Python, Python.NET, běhové prostředí .NET, strojové učení, skriptování

Python as scripting language for .NET runtime enviroment

Abstract

This thesis focuses on the issue of scripting in the .NET environment and explores the possibilities of integrating Python with this environment. First, the basic concepts of scripting in the .NET Runtime environment and the advantages of this approach are introduced. Next, the Python language and the .NET environment are discussed in detail, including its key technologies.

The second part is devoted to the various options for scripting in the .NET environment, such as using C#, IronPython, and Python.NET. Tools such as Csi.exe REPL, IPy.exe REPL, and their integration using NuGet packages are described.

The practical part of the thesis deals with the comparison of functions and features between IronPython and Python.NET. The results of load tests, conversion time between Python and .NET types, or AST generation when compiling IronPython are analyzed. The paper also describes the Python scripting environment created and shows its application in practice using a linear regression model.

Keywords: CLI, CLR, DLR, IronPython, Python, Python.NET, .NET runtime, machine learning, scripting

Poděkování

Rád bych poděkoval svému vedoucímu práce Ing. Janu Krausovi, Ph.D. za cenné rady a podněty, které mi během vypracovávání práce poskytl.

Obsah

Seznam zkratek

OOP	Objektově orientované programování
CLI	Common Language Infrastructure
CLR	Common Language Runtime
DLR	Dynamic Language Runtime
IL	Intermediate Language
AI	Artificial Intelligence
ML	Machine Learning
VES	Virtual Execution System
AST	Abstract syntax tree
REPL	Read–eval–print loop
API	Application Programming Interface

Seznam obrázků

Seznam tabulek

Úvod

V moderní době je integrace různých programovacích jazyků a technologických ekosystémů neodmyslitelnou součástí vývoje softwarových aplikací. Jedním z klíčových aspektů této integrace je propojení skriptovacích jazyků s běhovými prostředími, což umožňuje využít výhody obou přístupů – rychlosti a jednoduchosti skriptování spolu s robustností a výkonem běhového prostředí. V tomto kontextu zaujímá skriptovací jazyk Python významné místo díky své oblibě, rozšiřitelnosti a bohatému ekosystému knihoven. Python zároveň umožňuje snadnou integraci s různými technologiemi a prostředími, což ho činí vhodným nástrojem pro vývoj moderních softwarových aplikací, které vyžadují flexibilitu a efektivitu.

Tato práce se zaměřuje na analýzu použití programovacího jazyka Python a jeho integrace s prostředím .NET, které je známé svou schopností podporovat různé jazyky a technologie. Hlavním cílem této práce je porovnat dvě základní implementace této integrace: projekt IronPython, vyvíjený společností Microsoft a projekt Python.NET. Hlavním přínosem integrace Python do prostředí .NET je možnost využít knihovny napsané pro jazyk Python nad daty, které pochází z prostředí .NET, a tím vytvářet komplexní a efektivní aplikace spojující sílu obou ekosystémů.

IronPython je implementace programovacího jazyka Python pro prostředí .NET. Funguje na prostředím DLR, které se stará o překlad všech dynamických jazyků podporovaných prostředím .NET na mezi-jazyk, nejedná se tedy o přímou kopii funkce jazyka Python.

Python.NET je knihovna umožňující interakci mezi Python kódem a CLR. Na rozdíl od IronPython se nejedná o implementaci jazyka pro DLR, ale interakci mezi prostředím .NET a interpretrem CPython.

1 Skriptovaní

Skriptování je typ programování, který zahrnuje psaní kódu ve skriptovacím jazyce za účelem automatizace konkrétních úloh nebo přidání dynamických funkcí do aplikace. Skriptovací jazyky jsou obvykle interpretované, což znamená, že jsou překládány do strojového kódu řádek po řádku v průběhu provádění kódu, místo aby byly předem zkompilovány do spustitelného souboru. [1][2]

V kontextu běhového prostředí .NET lze skriptování využít k dynamické interakci s aplikacemi .NET a k manipulaci s nimi. To umožňuje provádět úpravy v reálném čase bez nutnosti překompilovat celou aplikaci. Skriptování v prostředí .NET runtime může být obzvláště užitečné pro úlohy, jako je automatizace procesů, úprava konfigurací za běhu nebo integrace dalších funkcí do běžící aplikace .NET. [3]

1.0.1 Výhody skriptování v prostředí .NET Runtime

Skriptování v prostředí .NET runtime nabízí několik výhod, mezi které patří:

- Dynamické úpravy: Skripty mohou měnit chování aplikace .NET, aniž by vyžadovaly úplnou rekompilaci.[3]
- Flexibilita: Skripty lze psát v různých jazycích, včetně jazyků Python, C# a F#, což vývojářům umožňuje vybrat si jazyk, který je pro jejich úkol nejvhodnější.[2]
- Snadné použití: Skriptovací jazyky jsou často navrženy tak, aby byly jednodušší a přístupnější než ostatní programovací jazyky, což je ideální pro rychlý vývoj a tvorbu prototypů.[2]

1.1 Python

Python je více-paradigmatický programovací jazyk. Plně podporuje objektově orientované a strukturované programování a mnoho jeho prvků, také umožňuje funkcionální a aspektově orientované programování.[4]

Hlavní výhody jazyka Python:

- Multiplatformní - Python běží na různých operačních systémech, jako jsou Windows, macOS a Linux.
- Čitelnost kódu - Syntaktický design Pythonu klade důraz na čitelnost kódu a umožňuje psát kód, který je snadno srozumitelný.
- Bohatá základní knihovna - Python přichází s rozsáhlou standardní knihovnou, která poskytuje mnoho funkcí pro různé potřeby, od práce se soubory a síťovou komunikací až po zpracování textu a webové služby.
- Aktivní komunita a rozšiřitelnost - Python má velkou a aktivní komunitu, která přispívá k rozvoji jazyka a tvorbě rozšiřujících knihoven, jako jsou NumPy, SciPy, TensorFlow, a SciKit-learn, což z něj činí výkonný nástroj pro vědecké výpočty a strojové učení.
- Jednoduchá integrace s ostatními programy - Python snadno spolupracuje s jinými jazyky a nástroji díky rozhraním.
- Silná podpora pro datovou vědu a analýzu dat - Python je oblíbeným jazykem pro datovou vědu díky knihovnám jako Pandas, Matplotlib, a seaborn, které umožňují efektivní manipulaci a vizualizaci dat.

Díky své flexibilitě se Python úspěšně začlenil do různých oblastí, od webového vývoje a vědeckého počítání, po umělou inteligenci a analýzu dat. Jeho interpretovaná povaha umožňuje rychlé prototypování a testování, což podporuje rychlý vývoj. Objektově orientovaný přístup Pythonu podporuje modularitu a zapouzdření, což usnadňuje opětovné použití a udržovatelnost kódu. V roce 2023 se Python umístil jako 3. nejpopulárnější programovací jazyk. [5]

1.2 .NET

Prostředí .NET je softwarová platforma poskytující široké spektrum možností pro vývoj a provoz aplikací pro různé operační systémy. Platforma .NET nabízí řadu prostředků pro vytváření webových aplikací, desktopových programů, mobilních aplikací, webových služeb a mnoho dalšího. Vývojové prostředí .NET bylo navrženo s cíli:

- Poskytnout jednotný přístup k vývoji objektově orientovaného programování (OOP) nezávisle na místě výkonu programu a typu programu (web, desktop).
- Poskytnout prostředí pro běh programu:
  - Minimalizující nasazení programu a konflikty verzí
  - Se zabezpečeným výkonem algoritmu i od neověřeného zdroje
  - Eliminující výkonnostní problémy skriptovacích a interpretovaných jazyků
- Poskytnout standardizovanou komunikační vrstvu pro komunikaci s ostatními aplikacemi

Prostředí .NET v základní verzi podporuje vývoj v jazycích C#, F# a Visual Basic. Další jazyky jsou podporovány po přidání rozšiřujícího balíčku. Mezi tyto jazyky patří IronPython, IronRuby, C++.NET a další. Podle článku Microsoft [6], kód napsaný například v C# se kompilátorem C# přeloží do „mezi-jazyka“ (IL). Toto je možné díky standardu Common Language Infrastructure (CLI). [7]

1.2.1 CLI

Common Language Infrastructure je standard definovaný společností Microsoft pro spouštění vysokoúrovňových jazyků na odlišných platformách bez nutnosti přepisovat kód aplikace. Aplikací implementující CLI je prostředí .NET, dříve .NET framework nebo Mono. CLI-kompatibilní kód je přeložen do mezi-jazyka, který následně VES přeloží do strojového kódu. [8]

Standart CLI specifikuje 5 vlastností:

- Common Type System (CTS) - Sdílené datové typy a operace
- Metadata - Informace o struktuře programu ve formátu nezávislém na architektuře
- Standard Libraries - Sdílené knihovny zajišťující obecně používané funkce jako načítání/ukládání souborů
- Common Lanuage Specification (CLS) - Pravidla a zásady pro vývoj jazyků splňující CLI
- Virtual Execution System (VES) - Načítá a spouští programy, které jsou CLI--kompatibilní za využití modelu CTS a metadat. Stará se o spolupráci mezi moduly aplikace

1.2.2 CLR

Common Language Runtime (CLR) je implementací VES pro prostředí .NET. Jedná se o virtuální stroj, který spravuje běh .NET aplikací. CLR poskytuje klíčové funkce jako správu paměti, řízení výjimek a garbage collection.

Jednou z hlavních funkcí CLR je překlad kódu z mezi-jazyka do strojového kódu a jeho následné spuštění a vykonávání. Všechny jazyky v prostředí .NET jsou spouštěny v CLR, což zajišťuje jednotný přístup k vývoji aplikací a interoperabilitu mezi různými jazyky.

CLR podporuje jen statické jazyky jako C#, ale pro zpracování kódu vytvořeného dynamickými jazyky, jako je Python nebo Ruby, je zapotřebí rozšiřujících knihoven Dynamic Language Runtime (DLR).

[9]

1.2.3 DLR

Dynamic Language Runtime rozšiřuje funkčnost CLR a zajišťuje podporu dynamických jazyků v CLR. Za tímto účelem DLR implementuje jednotný způsob ukládání instrukcí do dynamického binárního stromu a knihovnu operací a převodů nad dynamickými typy.

Dynamic Language Runtime představuje sbor knihoven pro podporu dynamických jazyků v rámci CLR. Účelem DLR je rozšířit funkcionalitu CLR tak, aby umožnil plynulou integraci a vykonávání kódu napsaného v dynamických jazycích, jako jsou Python, Ruby nebo JavaScript, v prostředí .NET.

DLR nabízí knihovnu operací a převodů nad dynamickými typy, což je klíčový prvek pro manipulaci s daty a typy v dynamických jazycích. Tato knihovna umožňuje provádět mnohé operace, jako je dynamické přetypování, manipulace s objekty a vyhodnocování výrazů, a to vše s ohledem na dynamickou povahu těchto jazyků. [10]

Klíčovou vlastností DLR je implementace jednotného mechanismu ukládání instrukcí ze zdrojového kódu do abstraktního syntaktického stromu.

1.2.4 AST

Při zpracování dynamických jazyků v prostředí .NET vytváří DLR abstraktní syntaktické stromy(AST). Tyto AST jsou hierarchickou strukturou uzlů, kde každý uzel reprezentuje určitou část zdrojového kódu, jako jsou operace, proměnné, volání funkcí nebo výrazy. Každý uzel AST obsahuje informace o typu operace nebo výrazu, a pokud je to relevantní, také o jeho operandech či podvýrazech.

Při výkonu kódu je pak tato interní reprezentace využívána k překladu do mezi-jazyka, který je následně interpretován pomocí CLR.

Díky této abstraktní reprezentaci kódu mohou dynamické jazyky využívat výhod CLR a platformy .NET, jako jsou například správa paměti, výjimky, paralelismus a interoperabilita s dalšími jazyky a knihovnami. AST poskytují jednotný způsob manipulace a analýzy zdrojového kódu bez ohledu na konkrétní dynamický jazyk, což usnadňuje vývoj nástrojů pro práci s těmito jazyky v prostředí .NET. [11]

2 Možnosti jak skriptovat v .NET

Skriptovacího rozhraní pro framework .NET lze dosáhnout například vlastní implementací rozhraní REPL pomocí nativního jazyka C# nebo za použití jednoho z mnoha projektů. Některé z těchto projektů zahrnují: csi.exe, CS-Script, Roslyn Scripting API, IronPython, Python.NET, NLua, JavaScript API, IronRuby, IKVM.NET nebo JNBridge.

Tento seznam poskytuje přehled některých možností, které jsou k dispozici pro vytváření skriptovacího rozhraní nebo přímé skriptování v prostředí .NET. Každá z těchto možností má své vlastní výhody a použití závisí na konkrétních potřebách a požadavcích projektu. V rámci této bakalářské práce jsem se zaměřil na IronPython a Python.NET a jejich srovnání s nativním skriptováním v jazyce C#.

2.1 C#

V prostředí .NET lze skriptovat bez zavádění jiného programovacího jazyka, přímo v jazyce C#. Výhodou řešení je, že není potřeba znát žádný další jazyk, ale jak je dále zmíněno, zavedení jiného jazyka v některých případech vede na rozšíření dostupných knihoven.[3]

Pro C# existují projekty jako NumPy.NET a Pandas.NET, které poskytují C# interface pro a zabalují odpovídající CPython knihovnu společně s NuGet odkazy na Python.NET a CPython runtime. Pro skriptování a tedy pro další práci ale C# interface není potřeba, protože Python.NET lze také použít ke spuštění vlastního Python kódu, jak je vysvětleno v sekci Python.NET.

Pro C# existují projekty jako NumPy.NET a Pandas.NET, které zabalují odpovídající CPython knihovnu a zároveň k této knihovně poskitují C# rozhraní, společně s NuGet odkazy na Python.NET a CPython runtime. Pro skriptování a tedy pro další práci ale C# interface není potřeba, protože Python.NET lze také použít ke spuštění vlastního Python kódu, jak je vysvětleno v sekci Python.NET.[12][13]

2.1.1 Csi.exe REPL

Jednou variantou skriptování v C# je samostatný spustitelný soubor csi.exe, resp. jen csi na linuxových systémech. Program je přibalen společně s programem csc.exe, tj. překladačem C#. Spuštění programu z CLI otevře REPL rozhraní, kde je možné psát jednotlivé řádky kódu a živě sledovat jejich výstup. Soubory s .NET assemblies lze s příkazové řádky předat přepínačem `/lib:<path>` a jmenné prostory importovat přepínačem `/u:<namespace>`. [14]

2.1.2 Roslyn skriptovací API

Pro poskytnutí schopnosti skriptování z vlastní .NET aplikace lze použít skriptovací API ze sady nástrojů Roslyn dostupnou jako NuGet balík. Protože jde o C# kód zkompilovaný v kontextu běžícího assembly, je potřeba jen importovat jmenné prostory funkcí `WithImports`. Pro sdílení stavu programu se skriptem existuje parametr funkce spouštějící skript, který lze nastavit na libovolný objekt a jehož atributy jsou dostupné jako globální proměnné v kontextu skriptu. [15]

2.2 IronPython

IronPython jakožto vlastní implementace Python překladače nad .NET umožňuje využití Python knihoven a .NET knihoven, ale ne CPython knihoven jako NumPy, Pandas nebo SciKit. Zejména nedostupnost knihovny NumPy má velký vliv na skriptování, jak se ukázalo už při implementaci srovnávacích testů. Naopak je velice snadné IronPython distribuovat v rámci .NET projektu, NuGet balík IronPython a balík se standardní knihovnou obsahují vše potřebné. [16]

2.2.1 IPy.exe REPL

Jedna možnost využití projektu IronPython pro .NET skriptování je pomocí souboru IPy.exe, který lze spustit z příkazové řádky. Prostředí se chová stejně jako Python REPL, jen s tím rozdílem, že na pozadí běží implementace Pythonu jako jazyka .NET DLR. K .NET assemblies se přistupuje s využitím jmenného prostoru clr, který IPy.exe přibaluje ke standardní knihovně. [16]

2.2.2 NuGet

Kód pro IronPython je také možné spustit NuGet balíkem v .NET projektu, kde samotný projekt musí zajistit získání a předání kódu. Pro skriptování to potom znamená získání kódu od uživatele v reálném čase nějakým řetězcovým vstupem, například z GUI vstupního textového boxu nebo ze standardního vstupu CLI. V tomto kódu je k dispozici opět jmenný prostor clr, který lze využít pro přístup k .NET assemblies, a to včetně assembly samotného .NET projektu. [16]

2.3 Python.NET

Python.NET propojuje .NET s plnohodnotným prostředím CPython, z čehož plyne schopnost využít .NET assemblies i CPython knihovny. Prostředí CPython s chtěnými knihovnami je ale nutné mít nainstalované zvlášť. Skriptovat je možné přímo v Python REPL za využití pip balíku Python.NET nebo z .NET projektu se stejnojmenným NuGet balíkem. [17]

Existuje také NuGet balík Python.Included, který poskytuje nástroje pro zařazení CPython engine do distribuce .NET projektu a instalaci Python i CPython balíků z přiložených `*.whl` souborů nebo pomocí balíčkovacího systému pip.[18]

2.3.1 Python REPL

Skriptovat v jazyce Python s Python.NET pro načítání .NET assemblies lze v libovolném prostředím s balíčkovacím systémem, ve kterém je dostupný Python.NET (pip, Anaconda). Jednou možností je výchozí Python CLI REPL, další variantou je pak IPython shell (interactive python), který lze také spustit ze systémového shellu. IPython poskytuje další nástroje jako prefix `!` pro spuštění příkazu systémového shellu bez nutnosti prostředí opustit, suffix `??` pro zobrazení informací a dokumentace a prefixy `%` a `%%` pro pomocné příkazy. IPython je zároveň součástí prostředí Jupyter Notebook, ve kterém je kromě textového výstupu možné i vykreslit HTML výstup nebo interaktivní widgety. [17]

2.3.2 NuGet

Vkládání jazyka Python do prostředí .NET lze provést obdobně jako v případě IronPython, dále existuje možnost manipulovat s Python objekty přímo v .NET s využitím dynamic a PyObject a přistupovat ke globálním proměnným Python engine přes objekt Scope, jak pro čtení tak i zápis. Nad objektem scope lze i importovat jmenné prostory nebo knihovny - včetně .NET jmenných prostorů s využitím modulu clr - a zároveň je přiřadit do proměnné typu dynamic pro přímé použití z .NET. Stav objektu Scope je trvalý napříč více instancemi spouštění .NET kódu. I z těchto důvodů byl NuGet balík zvolen pro další práci oproti Python REPL s pip balíkem. [17]

2.4 Lua

Spustit Lua kód v .NET projektu umožňuje například projekt NLua, dostupný jako NuGet balík, nebo DynamicLua wrapper pro NLua dostupný ze stejného zdroje. V obou případech je Lua bind inicializován vytvořením stavového objektu. Ten v projektu NLua poskytuje metodu pro vykonání řetězce s Lua kódem, metodu pro načtení assemblies a indexer pro přístup k proměnným pro čtení i zápis. DynamicLua stavový objekt zabaluje do dynamického typu a nahrazuje indexer dynamickými atributy, čímž odpadá i nutnost přetypovat. [19]

2.5 JavaScript

Podobně jako pro Python, i pro JavaScript existují implementace dvou přístupů ke spouštění JS kódu z .NET prostředí. Projekt Jint je vlastní implementace JS interpreteru v .NET, která poskytuje objekt Engine jako stav JS části programu. S tímto stavem lze manipulovat z JS i z .NET, a to včetně přiřazení .NET objektů proměnným nebo přístupu k .NET assemblies z JS. Naopak Javascript.NodeJS propojuje .NET s externě nainstalovaným NodeJS a poskytuje rozhraní pro spouštění JS funkcí v NodeJS runtime. Objekty .NET lze předat jako parametru těchto funkcí za předpokladu, že jsou JSON-serializovatelné.[20][21]

2.6 Požadavky IronPython a Python.NET

Tabulka 2.1 byla sestavena s cílem usnadnit rychlé a efektivní porovnání mezi projekty IronPython a Python.NET. Kritéria byla pečlivě vybrána tak, aby co nejlépe ukázala klady obou projektů.

Kritérium	**IronPython**	**Python.NET**
Provedení	Vlastní implementace nad .NET DLR	Integrace CPython engine a .NET CLR
Python verze	2.7, 3.4	3.7 až 3.12
.NET verze	.NET 5-8, .NET Standard 2.0+, .NET Framework 4.6.1+	.NET 5.0-8.0, .NET Standard 2.0+, .NET Framework 4.6.1+
Další požadavky	-	Nainstalovaný CPython interpreter
Použití .NET v Python	vlastní Python interpreter	běžný interpreter
Použití Python v .NET	DLR skriptovací engine, Python kód	spuštění CPython engine, Python nebo .NET kód
Knihovny	.NET	.NET a CPython
Výjimky	Odpovídající Python a .NET výjimky sloučené	Python a .NET výjimky jsou zvlášť

Tabulka 2.1: Srovnání technologií projektů IronPython a Python.NET

3 Analýza výkonnostních parametrů IronPython a Python.NET

Pro porovnání výkonnostních parametrů zkoumaných nástrojů pro integraci jazyka Python do běhového prostředí .NET bylo potřeba navrhnout testy tak, aby pokrývali možné případy použití při analýze časových řad s prostředky jazyka Python. Zároveň navržené testy pokrývají i elementární úlohy, které jsou opakovaně využívány při zpracovávání dat v kontextu strojového učení.

3.1 Metody navržených testů

Metoda `iterate`: Python.NET implementace využívá vlastnost NumPy broadcasting - jedno číslo je přičteno ke všem hodnotám napříč rozměrem. IronPython implementace využívá pro přechod přes prvky generátorovou notaci uvnitř list comprehension k sesbírání výsledků.

Metoda `find_max`: Python.NET implementace využívá NumPy agregační metody `max()`. Protože se jedná o jednorozměrný vektor, proběhne funkce nad všemi prvky. IronPython implementace využívá vestavenou funkci `max()` vracející nejvyšší prvek.

Metoda `find_elements`: Python.NET implementace využívá logické indexování s podmínkou k výběru prvků splňujících podmínku. IronPython implementace využívá k výběru prvků generátorovou notaci s podmínkou uvnitř list comprehension.

Metoda `add_arrays`: Python.NET implementace využívá vektorizovaný součet - vektory jsou jednoduše sečteny, NumPy převezme paralelizaci. IronPython implementace využívá generátorové notace nad součinem dvojic vytvořených funkcí zip v list comprehension.

Metoda `sample`: Python.NET i IronPython implementace využívají slicing - vybírá prvky a nevytváří nové pole.

Metoda `windows`: Python.NET implementace využívá přetvarování vektoru na matici a následně agregační funkci sum, která proběhne nad jedním rozměrem. IronPython implementace sčítá sumu z podseznamů načtených for cyklem.

Metoda `vector_products`: Python.NET implementace využívá operátor maticového násobení, který v NumPy nad vektory počítá skalární součin. IronPython implementace využívá sumu generátorové notace nad součinem dvojicemi vytvořenými funkcí zip v list comprehension.

Metoda `sort_array` quicksort algoritmem vzestupně seřadí prvky vstupního pole. Python.NET i IronPython implementace využívají vestavenou metodu `sort()`.

Metoda `calculate_rms`: Python.NET implementace využívá NumPy broadcasting a agregační metodu `mean()`. IronPython implementace využívá sumu generátoru, který umocňuje jednotlivé prvky.

Metoda `moving_average`: Python.NET implementace využívá NumPy funkci convolve s jednotkovým polem jako jádrem. IronPython implementace využívá slicing pole ve for cyklu, který sečte a zprůměruje.

Metoda `manipualate_floats` převede pole floatů na pole řetězců, ke každému řetězci připojí řetězec a vrátí pole floatů. Python.NET implementace i IronPython implementace využívají list comprehension nad generátorem, který provede stringovou operaci nad prvky a pole stringů převedou na pole floatů.

Jednotlivé úlohy byly zpracovány za použití vhodných prostředků dostupných pro dané prostředí.

Každá dílčí úloha byla navržena s nástroji dostupnými pro každou zkoumanou integraci. Kromě toho byly úlohy zpracovány způsobem, který je typický pro jednotlivé nástroje, a s ohledem na nižší složitost kódu potřebného pro praktické použití při skriptování v jazyce Python na konzoli. V důsledku byly úlohy pro IronPython implementování za použití cyklů a Python.NET šlo využít vektorizované programování poskytované knihovnou NumPy. V Python.NETu bylo navíc potřeba převést data do NumPy objektů. Časová náročnost převodu byla měřena zvlášť a zahrnuta v grafu 3.7.

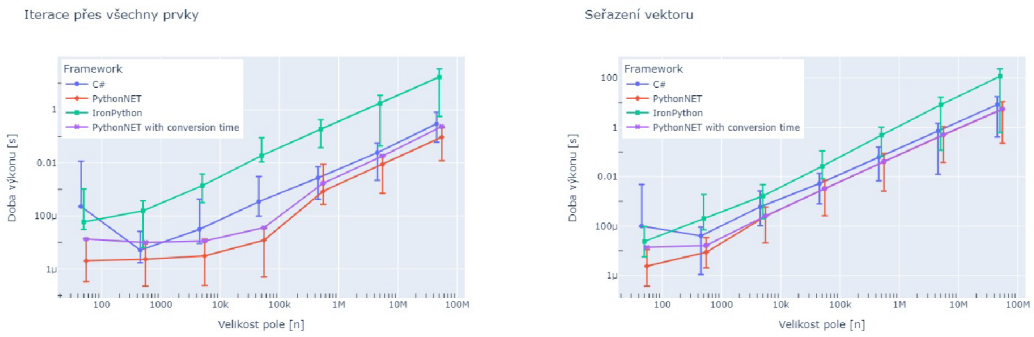
Každá dílčí úloha byla srovnávána za použití souborů dat v rozsahu od 50 až po 50 miliónů floatů. Použití tohoto rozsahu zajistilo, že byly zohledněny jak režijní náklady, tak škálováni

3.2 Porovnání výsledků

Výsledky zátěžových testů mezi kódem napsaném v C#, IronPythonu a Python.NETu zobrazují dobu potřebnou k vykonání úlohy v již běžícím prostředí. Výsledné doby výkonu byly změřeny vestavěnou knihovnou jazyka Python timeit. Grafy pak spojitě ukazují průměrný čas potřebný k vykonání úlohy a svislé chybové úsečky zobrazují odchylku nejdelšího a nejkratšího průběhu úlohy.

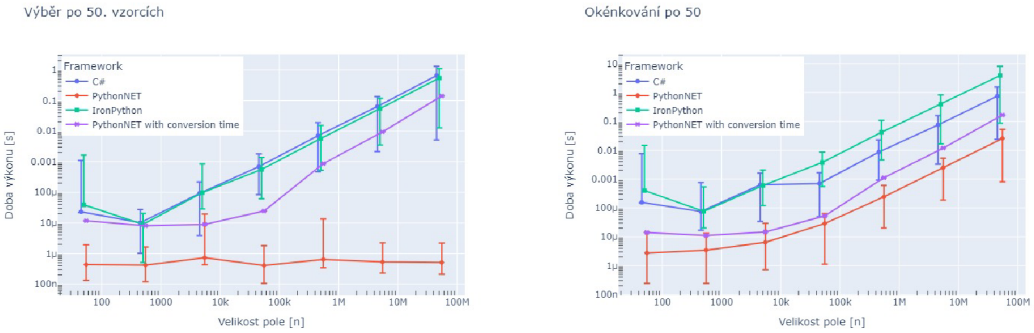
3.2.1 Výsledky zátěžových testů

V obou úlohách se zpracování dat v Python.NETu jeví jako rychlejší varianta s konstantním faktorem zrychlení oproti IronPythonu přibližně 100 pro úlohu ”Iterace přes všechny prvky” a přibližně 10-20 pro úlohu ”Seřazení vektoru”. Rozdíl v koeficientech způsobuje jednoduchost vektorového zpracování první úlohy. V případě zpracování úlohy přímo v jazyce C# lze pro malé vzorky dosáhnout podobné doby výkonu jako v Python.NET.



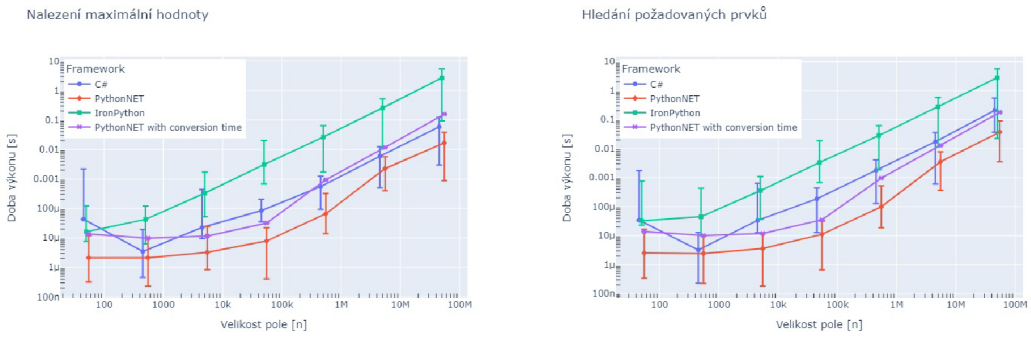
Obrázek 3.1: Operace nad jednotlivými prvky

Na grafech 3.2, stejně jako v případě iterace přes prvky, je u .NET řešení velmi výrazně vidět vliv režijních nákladů pro nejmenší z datových souborů a osmdesátkrát rychlejší okénkování dat s použitím projektu Python.NET. Zároveň je v úloze převzorkování vidět konstantní čas výkonu v případě Python.NET při zanedbání doby převodu dat. Knihovna NumPy poskytuje indexování s krokem, které zabírá konstantní čas bez ohledu na velikost dat, na rozdíl od iterace přes data.



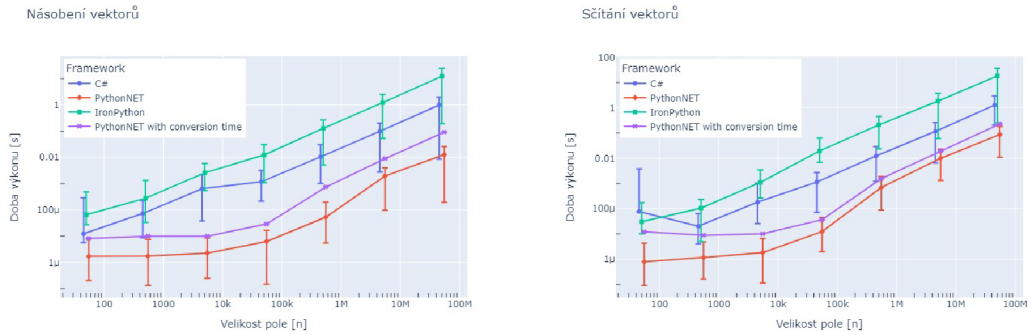
Obrázek 3.2: Okénkování a podvzorkování

Úlohy 3.3 vyhledávání v datech vykazují opožděný nástup růstu časové náročnosti zpracování úlohy při použití knihovny NumPy, oproti řešením IronPython a C#. V případě práce s daty v řádech tisíců je řešení napsané v jazyce C# stejně rychlé nebo i rychlejší než řešení Python.NET, v závislosti na tom, jestli počítáme s převodem dat. U zpracování většího objemu dat je Python.NET s NumPy opět rychlejší variantou.



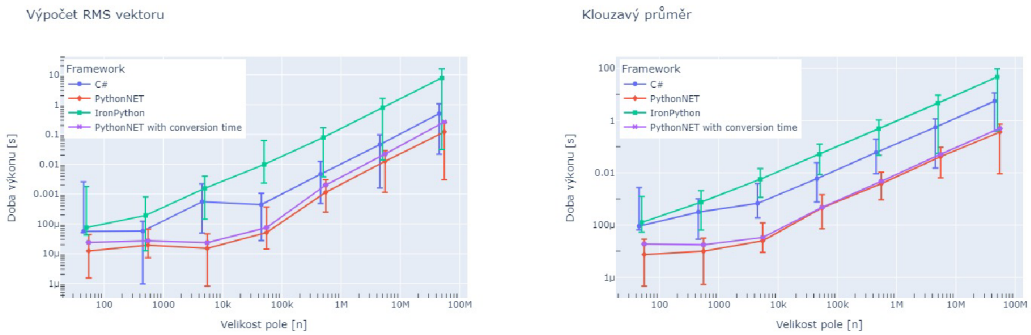
Obrázek 3.3: Vyhledáváni v datech

Výsledky úloh 3.4 se chovají v souladu s doposud pozorovanými jevy. S více daty se snižuje poměr mezi režijními náklady rozdělení práce na jádra a jejího převodu do AVX instrukcí, oproti vlastním výpočtům. Vysoký faktor zrychlení u skalárního součinu u Python.NET, přibližně 300, lze vysvětlit schopností aplikovat fused multiply–add (FMA) instrukce.



Obrázek 3.4: Vektorové operace

V úloze 3.5 klouzavého průměru se růst časové náročnosti u všech řešení chová stejně, i když je využita NumPy funkce `convolve` a koeficient zrychlení Python.NET oproti C# a IronPython je nad 5000 prvků konstantě 10 a 100 v tomto pořadí. Při výpočtu kvadratického průměru je naopak opět vidět vlastnost NumPy broadcasting a u dat nad 50 000 prvků roste časová náročnost řešení Python.NET pomaleji.

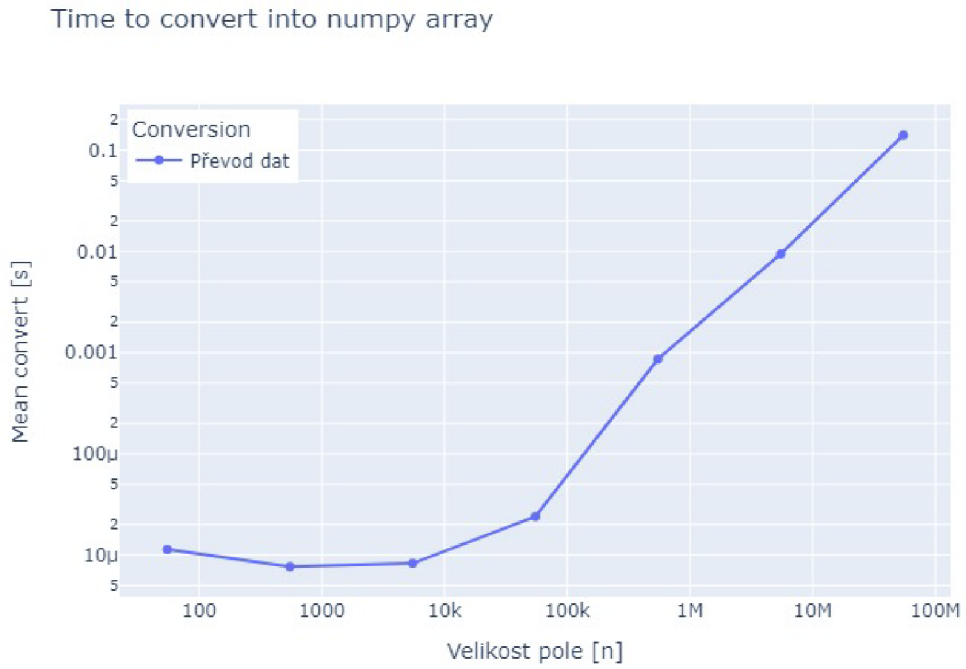


Obrázek 3.5: Analytické operace

Rozhodl jsem se nakonec zaměřit na úlohu 3.6 manipulace s čísly a řetězci, protože jsem při rešerši často narazil na články, které tvrdily, že C# je mnohem efektivnější při manipulaci s řetězci než Python. Výsledky mých testů potvrdily tuto informaci. Dokonce ani při použití knihovny NumPy se Python.NET nedokázal vyrovnat výkonu řešení napsaného v C#. Za zmínku stojí, že během zpracování pole o velikosti 50 000 000 byla nejrychlejší implementace IronPython. Tato situace se v ostatních testech nikdy nevyskytla.

Obrázek 3.6: Manipulace s typy

Ve všech zkoumaných případech vyplývá, že Python.NET s knihovnou NumPy exceluje při analýze dat a zvládá rychleji operace, které jsou často používané ve strojovém učení. Výsledky testů naznačují, že Python.NET je vhodnější volbou pro implementaci REPL aplikace podporující analýzu dat a vývoj strojového učení v prostředí .NET.



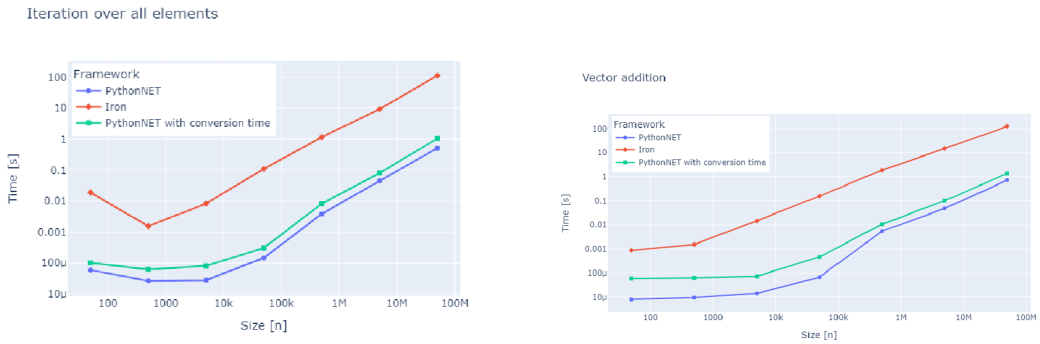
Obrázek 3.7: Doba převodu mezi .NET typy a NumPy typy

Na grafu 3.7 je zobrazena doba potřebná k převodu dat z .NET pole do pole NumPy. Na datech je vidět jasná spojitost mezi dobou potřebnou k převedení dat s rostoucím počtem dat. Na začátku, s menšími poli, doba převodu neroste, ale s postupem času se zvyšuje tempo tohoto růstu. Důležitou informací je, že doba potřebná k převodu jednoho prvku se pohybuje od přibližně 5.18 mikrosekundy pro pole o velikosti 50 až po přibližně 12.66 mikrosekundy pro pole o velikosti 50000000. Tento rozsah dobře ilustruje nárůst časové náročnosti převodu s rostoucí velikostí pole.

3.2.2 Tvorba AST při překladu IronPython

Na většině grafů lze zpozorovat vysokou výchylku časové náročnosti při zpracovávání úlohy o 50 prvcích v případě, že je úloha zpracována kompletně v běhovém prostředí .NET. Tento jev je výrazně potlačen u výpočetně komplexních úloh, jako je seřazení pole, výpočet rms nebo manipulace s datovým typem. Pozorování jevu vedlo k dalším pokusům o nalezení příčiny tohoto jevu. Nakonec se jako příčina ukázala vyhodnocovací strategie odloženého vyhodnocení (lazy evaluation) pro sestavení AST. Strom funkce je sestaven až při prvním volání funkce, které probíhá v měřeném porovnávacím testu.

Strategie odloženého vyhodnocení je vhodná pro rozsáhle projekty, kdy není nutné sestavit stromy pro celý projekt, ale pak zabírá čas při jeho eventuálním sestavení. Sestavení stromu je možné vynutit zavoláním funkce před provedením porovnávacích testů. Pro demonstraci vlivu tohoto jevu na dobu výkonu, ukazuje graf 3.8 vpravo časovou náročnost po jeho eliminaci tímto opatřením.



Obrázek 3.8: Vliv sestrojení stromu

3.2.3 Souběh v Python.NET

Při zkoumání, jak projekt Python.NET zachází se CPythonem, který se odkazuje na části paměti z běhového prostředí .NET, jsem zjistil následovné. Když nastane výkon Python skriptu upravující paměť .NET objektu, následně se vynutí běh .NET garbage collectoru z uvedeného python skriptu a poté se rychle přistoupí k paměti pomocí C#, jsem opakovaně získal přístup k poškozené části paměti. K tomu chování dochází pouze tehdy, pokud jsou splněny všechny tři podmínky a vše se děje v rychlém sledu. Toto chování zmizí při změně byť jen jednoho z těchto kroků nebo zajištěním časového odstupu mezi průběhem garbage collectoru a C# kódem, například uspáním vlákna. V ostatních případech byl projekt Python.NET schopný udržet oba překladače v souladu.

4 Prostředí pro analytické úlohy

Jedním z cílů, které si tato práce klade, je použití vybraného řešení skriptování v .NET pro provedení netriviální úlohy s využitím knihovny pro strojové učení. Pro další postupy byla zvolena vlastní implementace Python REPL s využitím Python.NET, následující kapitola je proto věnována dokumentaci této aplikace a následnému vypracování netriviálního výpočtu.

4.1 Dokumentace aplikace

Aplikace byla navržena tak, aby umožnila poměrně snadné budoucí rozšíření i intuitivní skriptování, dokumentace se tedy zabývá jak praktickým použitím aplikace, tak vnitřní strukturou a postupy pro rozšíření o knihovny i vlastní kód.

4.1.1 Struktura aplikace

Aplikaci tvoří vrstvy MVC (model, view, controller) psané v UI frameworku Avalonia především z důvodu vhodného využití multiplatformnosti, kterou nabízí .NET i Python.

Model aplikace tvoří třída `Variable`, která provazuje Python proměnné s grafickým rozhraním.

Třída `VariablesViewModel` je hlavním ViewModelem aplikace, který obstarává samotný běh jednotlivých řádek Python programu a jejich výstup v okně programu.

Třídy `DataViewModel` a `DataService` demonstrují příkladné připojení existujícího .NET kódu do aplikace jako službu, ze které lze čerpat data.

Samotné okno aplikace je definované v souboru `MainWindow.axaml` typu XAML a se stavem aplikace je svázáno pomocí `Binding` a thread-safe objektů jako `ObservableCollection`.

Inicializaci aplikace, tj. MVC modelu a Python.NET engine, obsluhuje třída `Program`, která zároveň poskytuje ukázku pomocných metod ze strany .NET a dvou způsobu přístupu k nim, `Scope.Set` a `clr.AddReference`.

Aplikace využívá balíčkovací system NuGet pro správu dependencies. Balíčkovacím systémem lze přidat .NET projekty, na které se lze z Python kódu odkázat přes objekt `clr`, především je ale použit pro knihovny Avalonia, Python.NET a Python-included. Deklarace Python-included umožňuje aplikaci nainstalovat si vlastní Python runtime a nainstalovat do něj potřebné Python knihovny ze systému pip a díky tomu je aplikace soběstačná.

4.1.2 Grafické rozhraní

Okno aplikace je rozděleno na panel historie příkazů, panel seznamu proměnných a příkazovou řádku.

V panelu historie příkazů jsou vidět jednotlivé příkazy a jejich výstup, tvoří tedy REPL prostředí. Výstup příkazu lze skrýt použitím středníku na konci.

Proměnné jsou zobrazeny ve formátu název a hodnota, a to v pořadí jejich vytvoření. Protože aplikace zobrazuje hodnotu i pro některé složitější objekty, panel lze scrollovat horizontálně i vertikálně.

Z příkazové řádky lze jednotlivé příkazy spouštět, buď tlačítkem v okně aplikace nebo klávesou enter. Neplatné příkazy nejsou spuštěny, ale ani smazány, aby bylo možné jednoduše opravit například překlep.

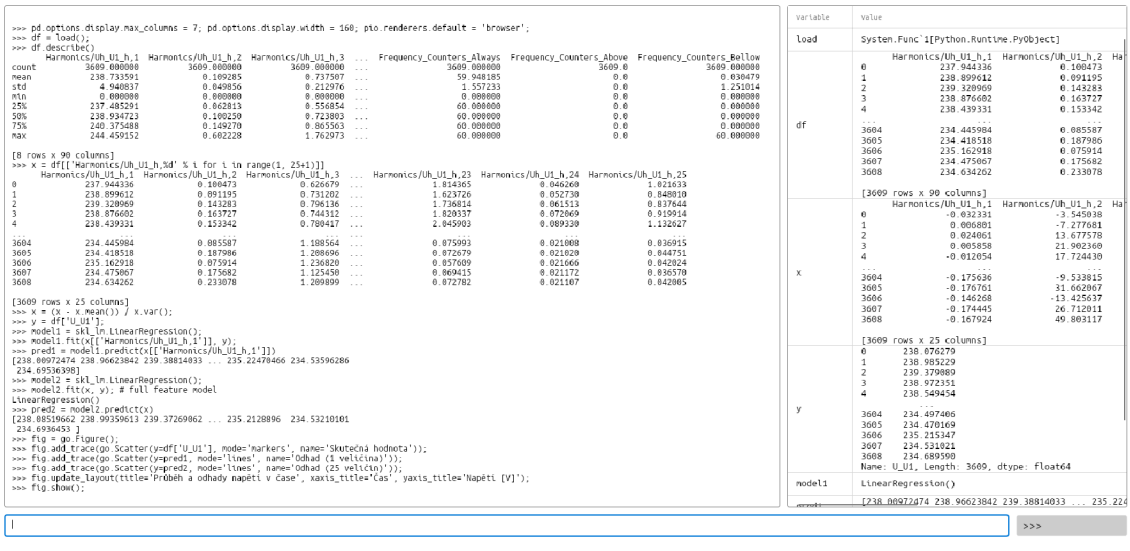
4.1.3 Práce s aplikací

Aplikace je navržena pro REPL workflow a poskytuje okamžitou zpětnou vazbu po spuštění platného příkazu. Výstup je pro přehlednost možné potlačit středníkem na konci řádky (ne příkazu, pokud je na konci řádky komentář, středník přijde až za něj).

Výstup příkazu je možné uložit do proměnné běžnou Python syntaxí. Výstup posledního příkazu je dále uschován v proměnné `_` „podtržítko," nezávisle na tom, zda byl uložen i do nějaké další proměnné.

Bez dalších rozšíření aplikace poskytuje knihovny NumPy, Pandas, Scikit-Learn a Plotly Express. Zároveň je při inicializaci do proměnných přidána funkce `load()`, která vrací ukázková data v podobě `DataFrame`. Právě nad těmito daty a s těmito knihovnami byly zhotoveny demonstrační modely.

Knihovnu Plotly Express je nutné nejdřív nastavit na výstup do prohlížeče, a figury nebudou vykreslovány automaticky ale funkcí `fig.show()`. Pro časté operace jako inicializace, načtení dat nebo vykreslení grafu je možné vytvořit Python soubor a spustit jej funkcí `exec()`.



Obrázek 4.1: Grafické rozhraní aplikace

Pohled do proměnných je obstarán Python funkcí `str()`, kterou Pandas i NumPy implementují tak, že velké tabulky a matice jsou uprostřed ořezány

4.1.4 Rozšíření aplikace

Aplikaci lze rozšířit o Python knihovny, NuGet balíky i vlastní .NET assemblies nebo C# funkce.

Rozšíření o Python knihovny obsluhuje balíčkovací system pip zahrnutý v NuGet balíku Python-Included, který pro něj zároveň poskytuje .NET wrapper, jak je vidět v inicializaci programu ve třídě `Program`. V inicializaci zároveň probíhá jejich zavedení do sdíleného Scope objektu a do .NET proměnné typu `dynamic` pro přístup z Pythonu i z .NET.

NuGet balíky lze instalovat příslušným správcem balíků a přístup k jejich .NET assemblies z Python kódu je obsloužen objektem `clr` opět v inicializační části programu.

Pro zavedení vlastních .NET tříd a metody do Python kódu slouží také objekt `clr`, nicméně aktivní assembly už je v inicializaci do Scope naimportován, takže pouze stačí zajistit, že jde o veřejné třídy nebo metody. Pro usnadnění přístupu k funkcím se dále nabízí možnost jejich přiřazení proměnným Scope objektu, jako je provedeno v inicializaci pro metodu `LoadCeaData()`.

4.2 Ukázková analytická úloha

Následující část je věnována ukázce práce s aplikací zhotovenou v rámci práce pro netriviální výpočty. Data pocházejí z .NET projektu, jsou převedena na Pandas tabulku pomocnou .NET metodou, načtena do Python scope z příkazové řádky a další postupy proběhly pomocí Python příkazů s využitím CPython knihoven Pandas, NumPy, Scikit-learn a Plotly Express.

4.2.1 Model lineární regrese

V ukázce lineárního modelu je srovnána předvídací schopnost dvou modelů. Oba modely předvídají hodnotu napětí z harmonických složek, jeden z jedné složky a jeden z 25. Pro vybrání nezávislých proměnných - harmonických složek - byla využit generátor řetězců zachycený list comprehension pro indexování v Pandas tabulce.

Pro práci s modely lineární regrese byl využit objekt `LinearRegression` z knihovny Scikit-learn. Oba modely byly natrénovány i otestovány na všech datech a skutečná data i predikce obou modelů byly zakresleny do objektu `Figure` knihovny Plotly Express, který byl následně vykreslen do webového prohlížeče.

Z grafu je vidět zlepšení modelu při zavedení více proměnných, i když z důvodu volby ukázkových dat i model jedné složky predikoval skutečné hodnoty poměrně dobře.

Obrázek 4.2: Model linearní regrese

4.2.2 Zdrojový kód aplikace

Kompletní zdrojový kód spolu se souborem `ukázka_použití.txt` je dostupný na webové stránce GitHub. Konkrétní adresa: https://github.com/BelinaJaroslav/pynet_repl_gui.

5 Závěr

Práce si klade za cíl srovnání různých řešení interoperability prvků skriptovacích jazyků s prostředím .NET. Konkrétně jsou v této práci využity dva projekty, které tuto otázku řeší integrací jazyka Python. IronPython integruje prvky Python prostředí, jmenovitě jazyk a standardní knihovnu, v rámci plnohodnotného prostředí .NET. Projekt Python.NET zajišťuje spolupráci souběžně inicializovaného prostředí .NET a interpretu CPython.

V rámci bakalářské práce byla provedena rešerše technologií navržených za účelem podpory takovýchto nástrojů pro .NET prostředí. Následovně bylo v praktické části vyhotoveno jedenáct testovacích úloh. Svým rozsahem pokryly možné případy použití při analýze časových řad s prostředky jazyka Python a zároveň testy pokrývají i elementární úlohy, které jsou opakovaně využívány při zpracovávání dat v kontextu strojového učení. Každá z těchto úloh byla naprogramována pro Python.NET a IronPython zvlášť. Jako základ pro porovnání výsledků testů jsem v grafech také zahrnul doby vykonání metod v jazyce C#. Kód mezi těmito dvěma řešeními není přenosný kvůli jiné verzi jazyka Python a použití CPython knihoven.

Po vyhodnocení výsledků testovacích úloh a rešerše bylo rozhodnuto implementovat konzolovou aplikaci typu REPL v projektu Python.NET. Tento interaktivní nástroj umožňuje dynamické zadávání příkazů jazykem Python, jejich okamžité vyhodnocení a zobrazení výsledků v reálném čase. Uživatelské rozhraní bylo navrženo za použití frameworku Avalonia. Program ve své aktuální konfiguraci podporuje skriptování s knihovnami NumPy, Pandas, Scikit-Learn a Plotly Express. Tyto knihovny byly zvoleny kvůli své oblíbenosti a optimalizovanosti. Díky svému návrhu lze program rozšířit o libovolnou knihovnu napsanou pro CPython nebo .NET.

Z výsledků práce kromě rozšiřitelného REPL programu dále vyplývá vhodnost některých konfigurací pro různé účely. IronPython například umožňuje jednoduše zahrnout Python kód se schopností práce s .NET typy v rámci .NET projektu s příslušnými nástroji. Tento kód lze měnit za běhu nebo mimo běh programu bez nutnosti opětovného sestavení programu. Práce s Python.NET má více požadavků na konfiguraci systému, kde výsledný program bude spuštěn, ale díky těmto požadavkům umožňuje plnohodnotnou souběžnou práci s knihovnami prostředí .NET a CPython.

Použitá literatura

[1] CORALOGIX. *What is scripting* [online]. 2023. [cit. 2024-05-11]. Dostupné z: https://coralogix.com/blog/what-is-scripting/.

[2] DOYLE, Kerry. *Scripting vs. programming languages: Where they differ* [online]. 2023. [cit. 2024-05-11]. Dostupné z: https://www.techtarget.com/searchapparchitecture/tip/Scripting-vs-programming-languages-Where-they-differ.

[3] SHILO, Oleg. *C# Script: The Missing Puzzle Piece* [online]. 2014. [cit. 2024-05-11]. Dostupné z: https://www.codeproject.com/Articles/8656/C-Script-The-Missing-Puzzle-Piece.

[4] FOUNDATION, Python Software. *General Python FAQ* [online]. 2023. [cit. 2023-08-19]. Dostupné z: https://docs.python.org/3/faq/general.html#why-was-python-created-in-the-first-place.

[5] *Stack Overflow Developer Survey 2023* [online]. [B.r.]. [cit. 2023-08-19]. Dostupné z: https://survey.stackoverflow.co/2023/#most-popular-technologies-language.

[6] MICROSOFT. *What is "managed code"?* [online]. 2023. [cit. 2023-08-19]. Dostupné z: https://learn.microsoft.com/en-us/dotnet/standard/managed-code.

[7] MICROSOFT. *Overview of .NET Framework* [online]. 2023. [cit. 2023-08-19]. Dostupné z: https://learn.microsoft.com/en-us/dotnet/framework/get-started/overview.

[8] ECMA INTERNATIONAL. *Standard ECMA-335 - Common Language Infrastructure (CLI)*. 6. vyd. Geneva, Switzerland, 2012. Dostupné také z: https://www.ecma-international.org/wp-content/uploads/ECMA-335_6th_edition_june_2012.pdf.

[9] MICROSOFT. *Common Language Runtime (CLR) overview* [online]. 2023. [cit. 2023-08-19]. Dostupné z: https://learn.microsoft.com/en-us/dotnet/standard/clr.

[10] MICROSOFT. *Dynamic Language Runtime Overview* [online]. 2023. [cit. 2023-08-19]. Dostupné z: https://learn.microsoft.com/en-us/dotnet/framework/reflection-and-codedom/dynamic-language-runtime-overview.

[11] MICROSOFT. *Dynamic Language Runtime Overview* [online]. 2024. [cit. 2024-05-04]. Dostupné z: https://learn.microsoft.com/en-us/archive/msdn-magazine/2007/october/clr-inside-out-ironpython-and-the-dynamic-language-runtime.

[12] NUMPY.NET CONTRIBUTORS. *NumPy.NET* [online]. 2023. [cit. 2024-05-11]. Dostupné z: https://github.com/SciSharp/Numpy.NET.

[13] PANDAS.NET CONTRIBUTORS. *Pandas.NET* [online]. 2024. [cit. 2024-05-11]. Dostupné z: https://github.com/SciSharp/Pandas.NET.

[14] MICROSOFT. *[Essential .NET] C#Scripting* [online]. 2016. [cit. 2024-05-11]. Dostupné z: https://learn.microsoft.com/en-us/archive/msdn-magazine/2016/january/essential-net-csharp-scripting.

[15] ROSLYN CONTRIBUTORS. *Runtime-code-generation-using-Roslyn-compilations-in-.NET-Core-App* [online]. 2021. [cit. 2024-05-11]. Dostupné z: https://github.com/dotnet/roslyn/blob/main/docs/wiki/Runtime-code-generation-using-Roslyn-compilations-in-.NET-Core-App.md.

[16] IRONPYTHON CONTRIBUTORS. *IronPython* [online]. [cit. 2023-08-19]. Dostupné z: https://github.com/IronLanguages/ironpython3.

[17] PYTHON.NET CONTRIBUTORS. *Python.NET* [online]. [cit. 2023-08-19]. Dostupné z: https://github.com/pythonnet/pythonnet.

[18] PYTHON.INCLUDED CONTRIBUTORS. *Python.Included* [online]. 2023. [cit. 2024-05-11]. Dostupné z: https://github.com/henon/Python.Included.

[19] NLUA CONTRIBUTORS. *NLua* [online]. 2024. [cit. 2024-05-11]. Dostupné z: https://github.com/NLua/NLua.

[20] JINT CONTRIBUTORS. *Jint* [online]. 2024. [cit. 2024-05-11]. Dostupné z: https://github.com/sebastienros/jint.

[21] JERING.JAVASCRIPT.NODEJS CONTRIBUTORS. *Jering.Javascript.NodeJS* [online]. 2024. [cit. 2024-05-11]. Dostupné z: https://github.com/JeringTech/Javascript.NodeJS.