Appendix : Churn prediction script

```
# installing the required library
get_ipython().system('conda install --yes scikit-learn')
get_ipython().system('conda install --yes nbconvert')
```

In[2]:

```
# importing required library
import os
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn  as sns
from sklearn.model_selection import train_test_split
```

In[49]:

```
#from sklearn.pipeline import Pipeline
#from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler
#from sklearn.pipeline import FeatureUnion
#from sklearn.base import BaseEstimator, TransformerMixin
from sklearn.linear_model import LogisticRegression
#from sklearn.metrics import classification_report
#from sklearn.metrics import roc_auc_score
#from sklearn.model_selection import StratifiedKFold
#from sklearn.model_selection import cross_val_score
from sklearn.metrics import confusion_matrix
#from sklearn.model_selection import GridSearchCV
from sklearn.ensemble import RandomForestClassifier
#from sklearn.ensemble import GradientBoostingClassifier
from sklearn.svm import SVC
```

In[4]:

```
# Read CSV file into DataFrame df
df = pd.read_csv('data/Telco-Customer-Churn.csv')
```

In[5]:

```
# display the first 5 records
```

```
df.head()
```

2. Data Analysis
#
Let's explore and understand the data and perform Data Cleansing. The goal here is to spot any missing values, and understand the variables. We should also fix wrong datatypes, remove NaN values, check for duplicates. We should also add new columns where we are going to convert yes/no values to 1 and 0.

In[6]:

```
# Printing columns of the data
df.columns
```

In[7]:

```
# shape of data
df.shape
```

Here we can see data is containing 7032 rows and 21 colums

In[8]:

```
# Information about data like name, count of not null data, datatype
df.info()
```

Columns Explanation
#
We have only 3 Numerical Variables and those are:
#
1) Tenure(Number of months the customer has stayed with the company)
#
2) MonthlyCharges(The monthly amount charged to the customer)
#
3) SeniorCitizen(if Customer is senior citizen then 1 else 0)
#
4) TotalCharges(The total amount charged to the customer)
#
TotalCharges is srting but its stores the float value so we will change the datatype of this.
#
All Other Variables are Categorical.
#
Our Target Variable is Churn(Whether the customer churned or not (Yes or No)).

In[9]:

```
# TotalCharges is in String Format, but it should be float
df["TotalCharges"] = pd.to_numeric(df["TotalCharges"],errors='coerce')
```

In[10]:

```
df.info() # After changing type of TotalCharges checking the data info again
```

Now we can see the type of TotalCharges changed from object to float64

In[11]:

```
#Analysis the numberic attributes
df.describe()
```

```
# The above function is showing mathmatical calculation of numberic data.
#
# count - The count of not-empty values.
# mean - The average (mean) value.
# std - The standard deviation.
# min - The minimum value.
# 25% - The 25% percentile.
# 50% - The 50% percentile.
# 75% - The 75% percentile.
# max - The maximum value.
#
# with count we can see few data is null for TotalCharges.
#
# mean is showing what is the average for columns.
#
# With min we can see minimum MonthlyCharges is 18 and maximum is 118. Minimum is
showing lowest value and maximum is showing
# higher value.
```

In[12]:

```
df = df.dropna() # dropping null values. Few values are missing that's why in describe
function count of TotalCharges is
                #7032 and minimum value of tenure is 0. So its not useful to fill the
TotalCharges with some number.
```

In[ ]:

In[13]:

df.describe()

After dropping null values now no null values are there and minimum value of tenure is 1.

In[14]:

df.duplicated().sum() # checking duplicate values if any

Exploring Data Visualizations : To understand how variables are distributed.

In[15]:

```
# Visualize the distribution of dataset
def distr(dist_df):
      fig = plt.subplots(figsize=(12, 13))

      for i, col in enumerate(dist_df.columns):
         plt.subplot(7, 3, i + 1)
         plt.hist(dist_df[col].values, bins=30)
         plt.title(col)
         plt.tight_layout()
```

In[16]:

distr(df)

Description about the Features:
#
CustomerId – Customer ID
#
Gender – Male or Female
#
customerSeniorCitizen – Whether the customer is a senior citizen or not (1, 0)
#

```
# Partner – Whether the customer has a partner or not (Yes, No)
#
# Dependents – Whether the customer has dependents or not (Yes, No)
#
# Tenure – Number of months the customer has stayed with the company
#
# PhoneService – Whether the customer has a phone service or not (Yes, No)
#
# MultipleLines – Whether the customer has multiple lines or not (Yes, No, No phone
service)
#
# InternetService – Customer's internet service provider (DSL, Fiber optic, No)
#
# OnlineSecurity – Whether the customer has online security or not (Yes, No, No internet
service)
#
# OnlineBackup – Whether the customer has online backup or not (Yes, No, No internet
service)
#
# DeviceProtection – Whether the customer has device protection or not (Yes, No, No
internet service)
#
# TechSupport – Whether the customer has tech support or not (Yes, No, No internet service)
#
# StreamingTv – Whether the customer has streaming TV or not (Yes, No, No internet
service)
#
# StreamingMovies – Whether the customer has streaming movies or not (Yes, No, No
internet service)
#
# Contract – The contract term of the customer (Month-to-month, One year, Two year)
#
# PaperlessBilling – Whether the customer has paperless billing or not (Yes, No)
#
# PaymentMethod – The customer's payment method (Electronic check, Mailed check, Bank
transfer (automatic), Credit card (automatic))
#
# MonthlyCharges – The monthly charge amount
#
# TotalCharges – The total amount charged to the customer
#
# Churn – Whether the customer churned or not (Yes or No).(Need to predict)

# In[17]:


Churn = df['Churn'].value_counts() # counting churn values
Churn
```

Here we can see count of churn and non-churn. 0 is representing count of non-churn and 1 is count of churn customer.
#
non-churn : 5163
churn : 1869

In[18]:

```
# plotting the pie chart for churn and non churn count
Churn.plot(kind = 'pie', title = 'Churn', labels = Churn, figsize = (6,6))
plt.legend()
plt.show()
```

To group data by Churn and compute the mean to find out if churners make more Monthly Charge than non-churners:

In[19]:

```
# Group data by 'Churn' and compute the mean
print(df.groupby('Churn')['MonthlyCharges','TotalCharges','tenure'].median())
(df.groupby('Churn')['MonthlyCharges','TotalCharges','tenure'].median()).plot(kind = 'box'
            ,subplots=True, title = 'Avg value of Churn based on Charges and tenure', figsize
= (16,6))

plt.show()
```

Churning customers have higher monthly charges with a median of ca. 80 USD and much lower interquartile range compared to
that of non-churners (median of ca. 65 USD).
#
TotalCharges are the result of tenure and MonthlyCharges, which are more insightful on an individual basis.
#
Churning customers have much lower tenure with a mean of ca. 10 months compared to a median of non-churners of ca. 38 months.
#
Hence! In above result we can see, churners seem to pay more MonthlyCharges than non-churners.

#

Counting Churn customer based on different columns:

In[20]:

```
# Counting Churn data based on SeniorCitizen
print(pd.crosstab(df.SeniorCitizen,df.Churn,margins=True))


# In above result, we can see 5890 customer was not Senior who was using. and out of 5890,
1393 is churn.
# and in opposite site 1142 was SeniorCitien and 476 is churned.

# In[21]:


# Counting Churn data based on SeniorCitizen
print(pd.crosstab(df.gender,df.Churn,margins=True))


# In above result, female and male for both gender count is approx same.

# In[22]:


def bar(x,y):
    result = (pd.crosstab(df[x], df[y], normalize='columns')*100).plot(kind="bar",
                                    title= '% of Churn based on '+ x ,stacked=True, rot=0,figsize =
(9,6))
    for c in result.containers:

        # set the bar label
        result.bar_label(c, label_type='center')


# In[23]:


# % of Churn data based on gender
bar('gender','Churn')


# In[24]:


# % of Churn data based on SeniorCitizen
bar('SeniorCitizen','Churn')


# In above result, non-senior citizens churn % is much lower than senior churn %.

# In[25]:


# % of Churn data based on Partner
```

```
bar('Partner','Churn')


# In above result, Churn rate for customers without partners is higher than Partner

# In[26]:


# % of Churn data based on Dependents
bar('Dependents','Churn')
```

In above result, churn rate for customers without children is very high.

```
# In[27]:


# % of Churn data based on Contract
bar('Contract','Churn')


# In above result, Churn % for month-to-month contracts much higher than with remaining contracts, then one year contract is higher than Two year

# In[28]:


# % of Churn data based on PaymentMethod
bar('PaymentMethod','Churn')


# In above result,electronic check paymentMethod shows much higher churn rate, then mailed check. Bank transfer and credit card is appox same.

# In[29]:


# % of Churn data based on InternetService
bar('InternetService','Churn')


# In above result, Customers with fiber optic InternetService have much higher churn rate.It is approx. 69%

# In[30]:


# Add new col "ExtraService" by summing up the number of remaining services.
df['ExtraService'] = (df[['OnlineSecurity', 'DeviceProtection', 'StreamingMovies',
'TechSupport',
                         'StreamingTV', 'OnlineBackup']] == 'Yes').sum(axis=1)
```

In[31]:

```
# Counting Churn data based on ExtraService
result = pd.crosstab(df.ExtraService,df.Churn).plot(kind="bar",
                                 title= 'Count of Churn based on ExtraService ' ,stacked=True,
rot=0,figsize = (9,6))
for c in result.containers:

    # set the bar label
    result.bar_label(c, label_type='center')
```

In above result, Customer who is using less / no extra services they have higher churn rate as comapre who is using more than 3 extra services.

Cleaning data

In[32]:

```
#droping customerid
df = df.drop(columns = 'customerID')
```

In[33]:

```
def label_encoding(features, df):
   for i in features:
      df[i] = df[i].map({'Yes': 1, 'No': 0})
   return
```

In[34]:

```
#Converting categorical Columns to Numberical using label encoding

label_encoding(['Partner', 'Dependents', 'Churn', 'PhoneService', 'PaperlessBilling'], df)
df['gender'] = df['gender'].map({'Female': 1, 'Male': 0})
df
```

In[35]:

```
# One-Hot-Encoding for identified columns.
OHE_feature = ['MultipleLines', 'InternetService', 'OnlineSecurity', 'OnlineBackup',
'DeviceProtection',
            'TechSupport', 'StreamingTV', 'StreamingMovies', 'Contract', 'PaymentMethod',
'ExtraService']
```

```
df = pd.get_dummies(df, columns=OHE_feature)


# In[36]:


df.info()


# In[37]:


# Min-Max-Scaling for identified columns.
from sklearn.preprocessing import MinMaxScaler

features_mms = ['tenure', 'MonthlyCharges', 'TotalCharges']
df_features_mms = pd.DataFrame(df, columns=features_mms)
df_remaining_features = df.drop(columns=features_mms)

mms = MinMaxScaler()
rescaled_features = mms.fit_transform(df_features_mms)

df_rescaled_features = pd.DataFrame(rescaled_features, columns=features_mms,
index=df_remaining_features.index)

df = pd.concat([df_remaining_features, df_rescaled_features], axis=1)


# In[38]:


#Corelation between all col
plt.figure(figsize = (30,30))
corr_matrix = df.corr() # Corelation Matrix
sns.heatmap(corr_matrix, cmap="coolwarm")
plt.show()


# In[39]:


#Corelation with churn
df.corr()['Churn'].sort_values(ascending=False)


# In[40]:


#To Create Training and Test sets
```

```
X = df.drop("Churn", axis = 1)
y = df.Churn

# Splitting the dataset into the Training and Test sets

X_train, X_test, y_train, y_test = train_test_split(X, y,
                                    test_size = 0.25,
                                    random_state = 0)


# In[41]:


y_test.shape


# In[42]:


y_train.shape


# In[43]:


X_train.shape


# In[44]:


X_test.shape


# In[45]:


# Feature Scaling

sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)


# # Random Forest

# In[52]:


# Instantiate the classifier
```

```
from sklearn.metrics import accuracy_score
clf = RandomForestClassifier()

# Fit to the training data
clf.fit(X_train, y_train)


# In[53]:


# Predict the labels for the test set
y_pred = clf.predict(X_test)
feature_imp = clf.feature_importances_
# Compute accuracy and Evaluating Model Performance
accuracy_score(y_test, y_pred)


# In[ ]:


def imp_features(feature_imp,X):

    print("Important Feature", feature_imp)
    columns = X.columns
    index = np.arange(len(columns))
    font_size = 1
    fontsize_tittle = 25

    plt.figure(figsize=(100, 17))
    plt.bar(index, sorted(feature_imp), width=0.5,
            alpha=0.4,
            color='r',
            label='')

    plt.xlabel('Columns', fontsize=font_size)
    plt.ylabel('Feature Importance', fontsize=font_size)
    plt.title('Feature Importance for each column', fontsize=fontsize_tittle)
    plt.xticks(index, columns)
    plt.show()


# In[ ]:


# Plotting important features
imp_features(feature_imp,X)


# In[ ]:
```

```
#Confusion Matrix
def c_matrix(y_test, y_pred):
    cm = confusion_matrix(y_test, y_pred)
    print("Confusion Matrix" , cm)
    conf_matrix =
pd.DataFrame(data=cm,columns=['Prediction:0','Prediction:1'],index=['Actual:0','Actual:1'])
    plt.figure(figsize=(10,6))
    sns.heatmap(conf_matrix,annot=True,fmt='d',cmap='YlGnBu')


    #Other classification statistics
    TN=cm[0,0] #True Negative
    TP=cm[1,1] #True Positive
    FN=cm[1,0] #False Negative
    FP=cm[0,1] #False Positive
    sensitivity = TP/float(TP+FN)
    specificity = TN/float(TN+FP)

    print(' Accuracy = TP+TN/(TP+TN+FP+FN)= ', (TP+TN)/float(TP+TN+FP+FN) , '\n\n',
        'Missclassification = 1-Accuracy= ', 1- ((TP+TN)/float(TP+TN+FP+FN)),'\n\n',
         'Sensitivity/Recall or True Positive rate = TP/(TP+FN)= ', TP/float(TP+FN), '\n\n',
        'Specificity or True Negative Rate = TN/(TN+FP) = ', TN/float(TN+FP), '\n\n',
        'Precision/Positive Predictiv value = TP/(TP+FP) = ', TP/float(TP+FP), '\n\n',
        'Negative predicted value = TN/(TN+FN) = ', TN/float(TN+FN), '\n\n',
        'Positive Likelihood Ratio = Sensitivity/(1-Specificity) = ',sensitivity/float(1-
specificity), '\n\n',
        'Negative Likelihood Ratio = (1-sensitivity)/specificity = ',float(1-
sensitivity)/specificity)


# In[ ]:


c_matrix(y_test, y_pred)


# # K Nearest Neighbors

# In[ ]:


# Instantiate the classifier

clf= KNeighborsClassifier(n_neighbors=5, metric='minkowski', p=2 )
# Fit to the training data
clf.fit(x_train, y_train)


# In[ ]:
```

```
#Predicting the test set result

y_pred= clf.predict(x_test)
# Compute accuracy and Evaluating Model Performance

accuracy_score(y_test, y_pred)


# In[ ]:


#Confusion Matrix
c_matrix(y_test, y_pred)


# # Logistic Regression

# In[54]:


# Instantiate the classifier
clf = LogisticRegression(max_iter=1000)

# Fit to the training data
clf.fit(X_train, y_train)
print(print(clf.coef_))
a = np.std(X_train, 0)*clf.coef_
importance = clf.coef_[0]
print(dict(zip(X.columns,clf.coef_[0])))
#importance is a list so you can plot it.
feat_importances = pd.Series(importance)
feat_importances.nlargest(20).plot(kind='bar',title = 'Feature Importance')
print(df.columns)


# In[55]:


# Plotting important features
imp_features(importance,X)


# In[ ]:


# Predict the labels for the test set
y_pred = clf.predict(X_test)
```

```
# Compute accuracy and Evaluating Model Performance
accuracy_score(y_test, y_pred)


# In[ ]:




# In[ ]:


#Confusion Matrix
c_matrix(y_test, y_pred)


# # SVM

# In[ ]:


# Instantiate the classifier
clf = SVC(gamma='auto')

# Fit to the training data
clf.fit(X_train, y_train)


# In[ ]:


#Predicting the test set result
y_pred= clf.predict(x_test)
# Compute accuracy and Evaluating Model Performance

accuracy_score(y_test, y_pred)


# In[ ]:


#Confusion Matrix
c_matrix(y_test, y_pred)
```