Table of Contents

Introduction

This work proposes a method that predicts users' gender based on their transactions history information. This dataset is probably best suited for unsupervised ML techniques, but I was curious to see whether there are attributes that can help predict whether a customer is male or female. It's a small dataset (both in terms of features and number of customers), however I think this notebook gives a useful introduction to applying ML algorithms such as Random Forest, SVM, Decision Tree and KNN.

Importing the Packages

Every task must begin with importing the required packages into the respective environment (python in our case). Our primary packages include pandas for working on the data, NumPy for working with the arrays, matplotlib & seaborn for visualization, mplot3d for three-dimensional visualization, and finally scikit-learn for building the K-Means model. Let's import

all the primary packages into our python environment.

```
import numpy as np # for linear algebra
import pandas as pd # for data processing, csv io
from matplotlib import pyplot as plt # data plots
import seaborn as sns # pretty data plots
sns.set()
import re
from sklearn.preprocessing import LabelEncoder # for label normalization
from sklearn.metrics import mean_squared_error, mean_squared_log_error
from pyspark.ml.feature import StringIndexer, OneHotEncoder, VectorAssembler
import seaborn as sb # visualization
from mpl_toolkits.mplot3d import Axes3D # 3d plot
from termcolor import colored as cl # text customization

from sklearn.preprocessing import StandardScaler # data normalization
from sklearn.cluster import KMeans # K-means algorithm

plt.rcParams['figure.figsize'] = (20, 10)
sb.set_style('whitegrid')
```

Now that we have imported all the required primary packages into our python environment. Let's proceed to import the transactions data.

Importing Data

Dataset Description:

- types.csv - reference of transaction types
- codes.csv - reference of transaction codes
- transactions.csv - transactional data on banking operations
- train_set.csv - training set with client gender marking (0/1 - client gender)

We will use the 'read_csv' method provided by the Pandas package to read and import the data into our python environment. We are using the 'read_csv' method because the data we are going to use is in the '.csv' format. If it is an excel sheet, it is recommended to use the 'read_excel' method to read it into the python environment. Now let's import our data in python. Then we joined all datasets into one.

```
df_trans = pd.read_csv('Downloads/Datasets/transactions.csv')
df_types = pd.read_csv('Downloads/Datasets/types.csv', sep='delimiter')
df_codes = pd.read_csv('Downloads/Datasets/codes.csv', sep='delimiter')
df_train = pd.read_csv('Downloads/Datasets/train_set.csv')
df_test = pd.read_csv('Downloads/Datasets/test_set.csv')
```

```
df_trans[['client_id', 'datetime', 'code', 'type', 'sum']] = df_trans['client_id;datetime;code;type;sum'].str.split(
del df_trans['client_id;datetime;code;type;sum']

df_types[['type', 'type_description']] = df_types['type;type_description'].str.split(';', expand=True)
del df_types['type;type_description']

df_codes[['code', 'code_description']] = df_codes['code;code_description'].str.split(';', expand=True)
del df_codes['code;code_description']

df_train[['client_id', 'target']] = df_train['client_id;target'].str.split(';', expand=True)
del df_train['client_id;target']
```

```
join_codes = df_trans.merge(df_codes, on='code', how='left')
```

```
join_types = join_codes.merge(df_types, on='type', how='left')
```

```
df_joined = join_types.merge(df_train, on='client_id', how='left')
```

Now that we have successfully imported our customer segmentation data into our python environment. Let'sexplore and gain some information about the data.

Exploratory Data Analysis

In the data preprocessing part, we will mainly look for

missings and fill them

changing the necessary data types

```
print('Number of missing data:')
print(df_joined.isnull().sum())
df_joined.describe(include='all')
```

```
Number of missing data:
client_id               0
datetime                0
code                    0
type                    0
sum                     0
code_description        0
type_description       41
target              38213
dtype: int64
```

In this part we deleted all null values and work with datatypes

Does gender have an influence on the total sum

```
gender_sum = df_joined.groupby('target', as_index=False).agg({'sum': 'sum'}).sort_values(by='sum', ascending=False)
gender_sum['percentage (%)'] = gender_sum['sum'] / sum(gender_sum['sum']) * 100
gender_sum
```

	target	sum	percentage (%)
0	0.0	-5.037782e+08	32.91084
1	1.0	-1.026958e+09	67.08916

As we can see, gender is affecting on the sum of the transactions

Which transaction amount type is most considerable

```
PaymentMethod_TotalCharges = df.groupby('type_description', as_index=False).agg({'sum': 'sum'}).sort_values(by='sum'
PaymentMethod_TotalCharges.head()
```

	type_description	sum
27	Перевод на карту (с карты) через Мобильный бан...	1.550763e+09
8	Взнос наличных через АТМ (в своем тер.банке)	1.307821e+09
25	Перевод на карту (с карты) через АТМ (в предел...	4.132172e+08
2	Взнос наличных через POS	3.197199e+08
28	Перевод на карту (с карты) через Мобильный бан...	2.378725e+08

Transactions using mobile application is most demanded

It tells us, that we have 1.3e5 records. Also we can get some insights about the amount of transactions. It is obvious, that since the median (50% precintile) is negative, more transactions lead to cash outflows.

Also, we would like to know the number of unique clients, unique codes and unique types.

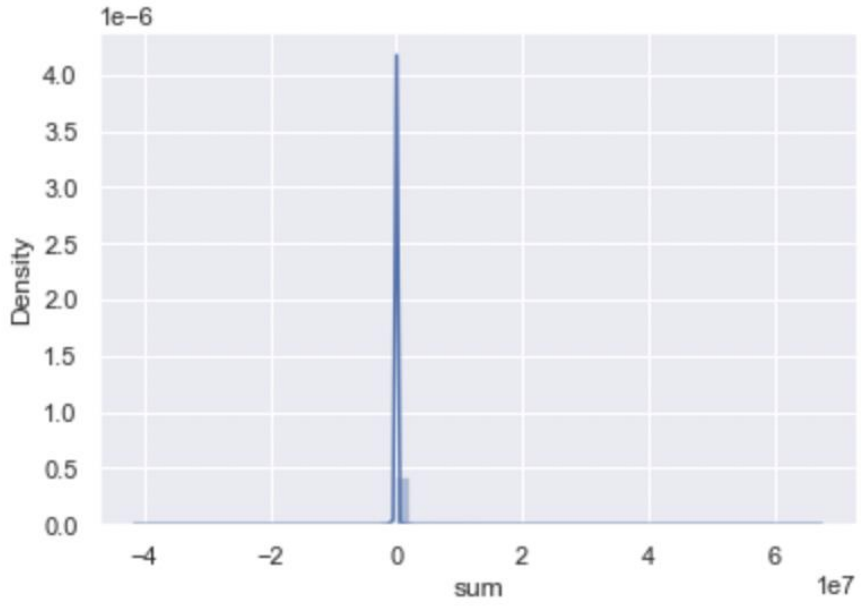
```
print(f"There are {transactions.client_id.unique().shape[0]} unique clients, {transactions.type.unique().shape[0]} unique transaction type
```

```
There are 8656 unique clients, 67 unique transaction types and 175 unique transaction codes
```

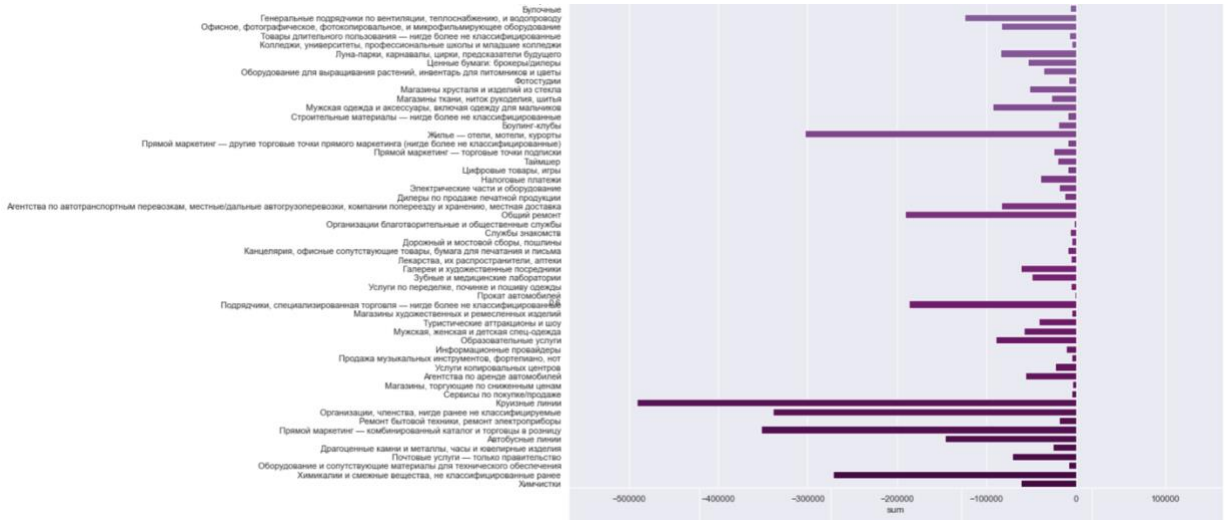
Through the EDA we found some information about features, for example gender is affecting on the sum of the transactions; Transactions using mobile application is most demanded; Most of the transactions were carried out using Financial Institutions - cash withdrawal manually

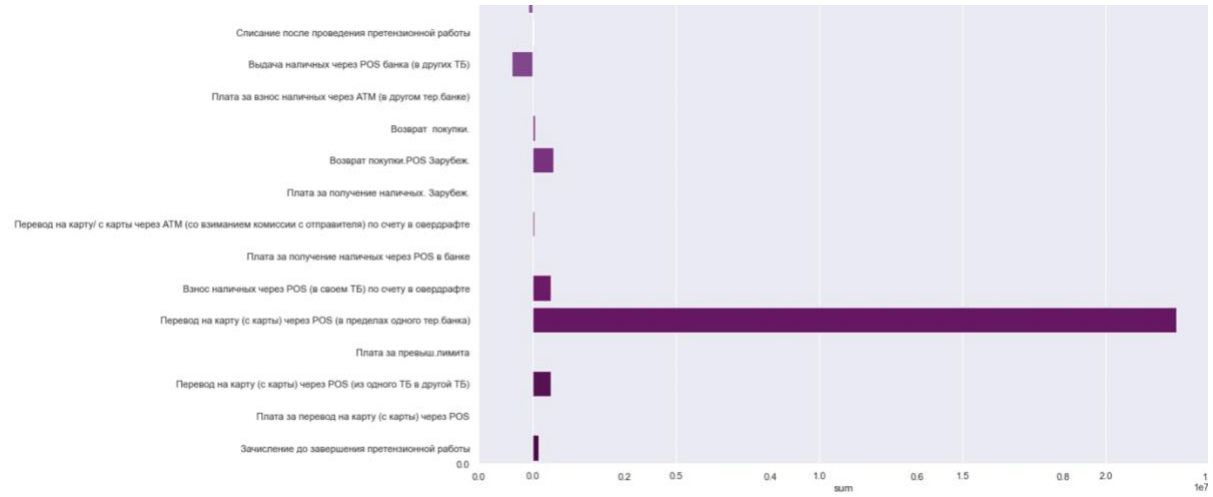
Visualization

Distribution of the sum of transations



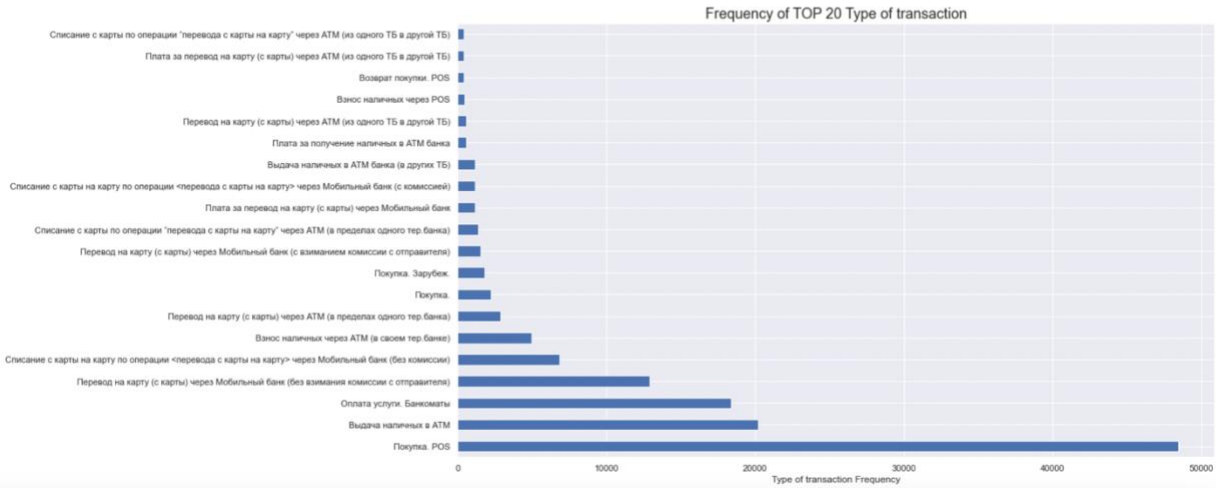
Amount of transactions by code amd type descriptions





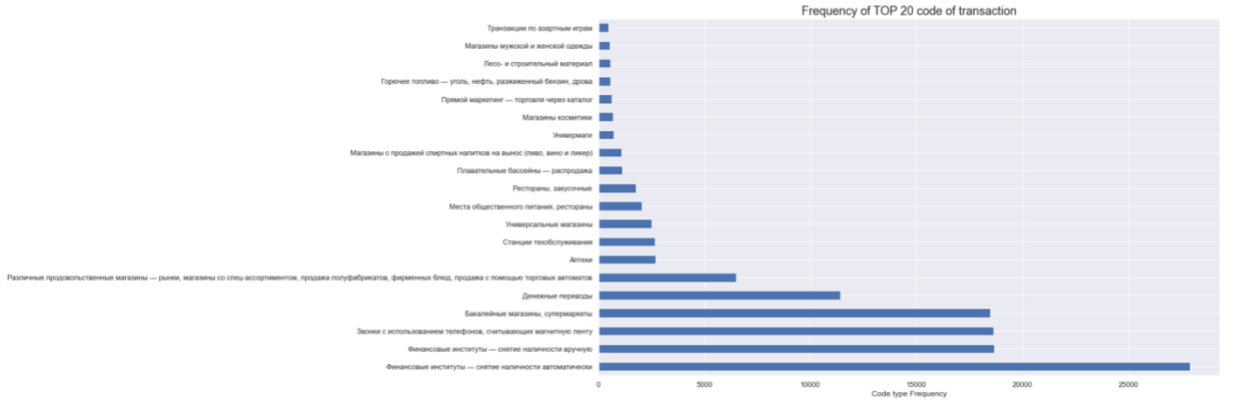
Graph of Frequency of TOP 20 Type of transactions. As we can see from the graph, Покупка POS and выдача наличных в ATM are most popular ones.

```
plt.figure(figsize=(17,10))
df.type_description.value_counts().nlargest(20).plot(kind='barh')
plt.xlabel('Type of transaction Frequency')
plt.title("Frequency of TOP 20 Type of transaction",fontsize=18)
plt.show()
```

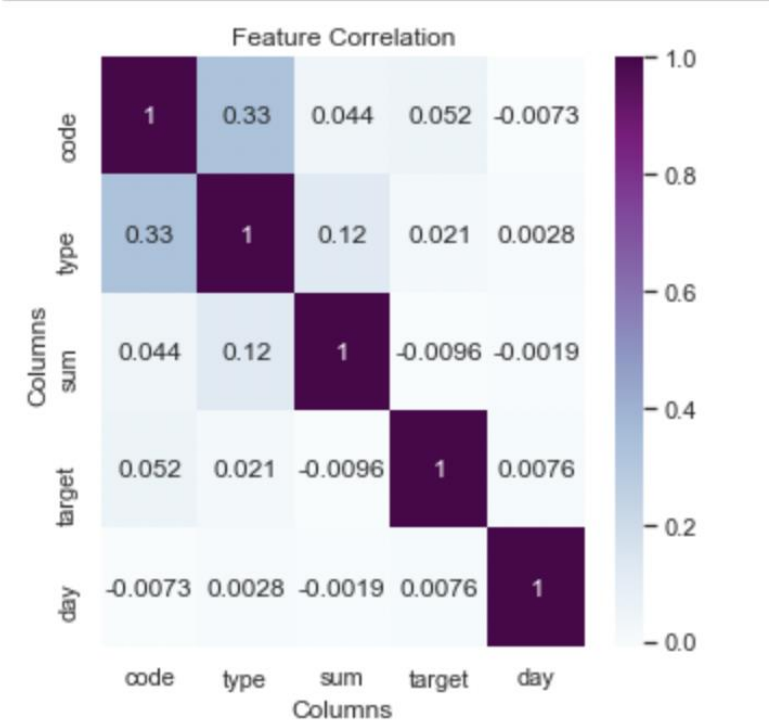


Graph of Frequency of TOP 20 Code of transactions. As we can see from the graph, Финансовые институты — снятие наличности автоматически and Финансовые институты — снятие наличности вручную are most popular ones.

```
plt.figure(figsize=(17,10))
df.code_description.value_counts().nlargest(20).plot(kind='barh')
plt.xlabel('Code type Frequency')
plt.title("Frequency of TOP 20 code of transaction",fontsize=18)
plt.show()
```



Correlation between features



Data Processing

In this step, we are going to normalize the dataset and it is very important to build our model. But what is normalization?

Normalization is a statistical method that helps mathematical-based algorithms to interpret features with different magnitudes and distributions equally

Using the ‘StandardScaler’ function provided by the scikit-learn package, we can feasibly perform normalization over the dataset in python.

```
X = df_joined.values
X = np.nan_to_num(X)

sc = StandardScaler()

cluster_data = sc.fit_transform(X)
print(cl('Cluster data samples : ', attrs = ['bold']), cluster_data[:5])
```

```
Cluster data samples :  [[ 1.57323554  0.6884737  -0.23501079 -0.87417515 -0.98268211  1.36698954]
 [-1.05552774  0.6884737  -0.23501079 -0.04532599 -0.98268211 -1.44433108]
 [ 1.18343625 -1.29943102 -0.66192027  0.02134232 -0.98268211 -0.70693551]
 [-0.71383204 -0.30796977 -0.62707051  0.02422527 -0.98268211  1.19032185]
 [ 1.16668999  0.68681296  1.96923609  0.03582159  1.01762308  1.38235195]]
```

Modeling

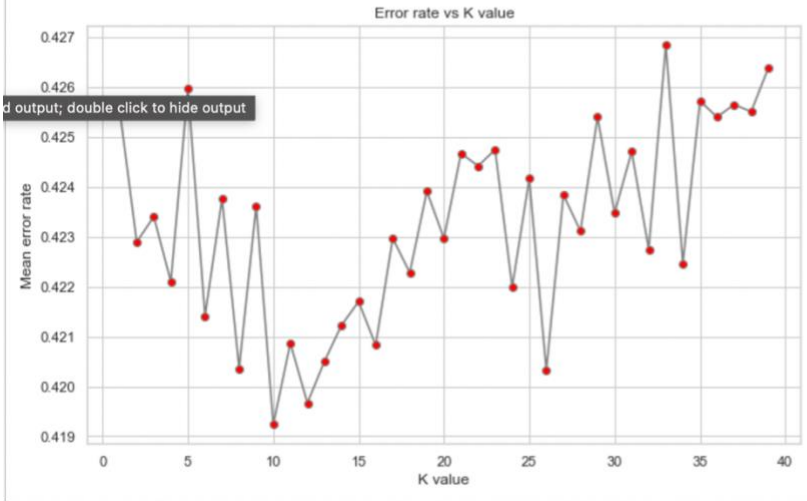
KNN

We were able to classify a couple of more points correctly, but in general, an accuracy score of 0.6 is not good. It looks like we'd need more data (more features or larger dataset) to build a more robust model.

The error rate is what we want to minimize, so we want to know the k that gives the smallest error rate. Let's create a visual representation to make life easier.

```
plt.figure(figsize=(10,6))
plt.plot(range(1,40), error_rate, color='grey', marker='o', markerfacecolor='red')
plt.title('Error rate vs K value')
plt.xlabel('K value')
plt.ylabel('Mean error rate')
```

```
Text(0, 0.5, 'Mean error rate')
```



0.419 is a very high error rate, but it is the best we're able to find. Let's now run the model again with k=13 again instead of k=3.

```
knn = KNeighborsClassifier(n_neighbors=13)
knn.fit(X_train, y_train)
y_pred = knn.predict(X_test)

print(confusion_matrix(y_test, y_pred))
print(classification_report(y_test, y_pred))
print ('Accuracy Score: ' + str(accuracy_score(y_test, y_pred)))
```

```
[[8644 5402]
 [6182 7320]]
              precision    recall  f1-score   support

         0.0       0.58      0.62      0.60     14046
         1.0       0.58      0.54      0.56     13502

    accuracy                           0.58     27548
   macro avg       0.58      0.58      0.58     27548
weighted avg       0.58      0.58      0.58     27548

Accuracy Score: 0.5794976041817917
```

We were able to classify a couple of more points correctly, but in general, an accuracy score of 0.6 is not good. It looks like we'd need more data (more features or larger dataset) to build a more robust model.

From our train and test data, we already know that our test data consisted of 27548 data points. We also notice that there are some actual and predicted values. The actual values are the number of data points that were originally categorized into 0 or 1. The predicted values are the number of data points our KNN model predicted as 0 or 1.

Random Forest

Let's try the Random Forest algorithm instead. We have already scaled data and split into train and test sets.

```
[[9989 4057]
 [4634 8868]]
              precision    recall  f1-score   support

         0.0       0.68      0.71      0.70     14046
         1.0       0.69      0.66      0.67     13502

    accuracy                           0.68     27548
   macro avg       0.68      0.68      0.68     27548
weighted avg       0.68      0.68      0.68     27548

Accuracy Score: 0.6845143023086976
```

Nothing better than a random draw. Let's instead use grid search to find the best parameter values. Parameter tuning is the process to selecting the values for a model's parameters that maximize the accuracy of the model.

We performed Grid search, and results slightly better than previously, but still not noticably different from a random draw.

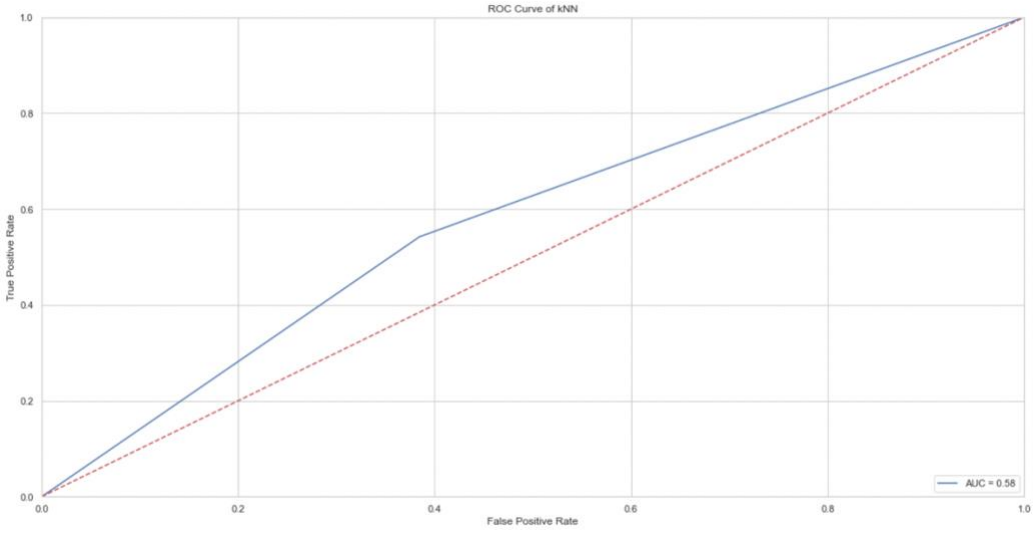
```
[[10139  3907]
 [ 4606  8896]]
              precision    recall  f1-score   support

         0.0       0.69      0.72      0.70     14046
         1.0       0.69      0.66      0.68     13502

    accuracy                           0.69     27548
   macro avg       0.69      0.69      0.69     27548
weighted avg       0.69      0.69      0.69     27548

Accuracy Score: 0.6909757514157108
```

Analyze Models (AUC/ROC, Precision Recall)

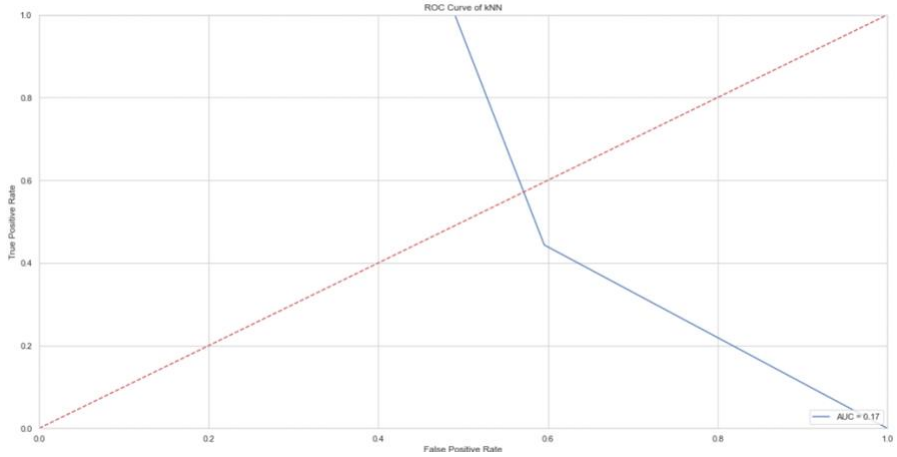
For any machine learning model, we know that achieving a 'good fit' on the model is extremely crucial. This involves achieving the balance between underfitting and overfitting, or in other words, a tradeoff between bias and variance. A useful tool when predicting the probability of a binary outcome is the ROC curve. It is a plot of the false positive rate (x-axis) versus the true positive rate (y-axis) for a number of different candidate threshold values between 0.0 and 1.0. Put another way, it plots the false alarm rate versus the hit rate. The area with the curve and the axes as the boundaries is called the Area Under Curve (AUC). It is this area which is considered as a metric of a good model. With this metric ranging from 0 to 1, we should aim for a high value of AUC. Models with a high AUC are called as models with good skill.

ROC Curve for KNN

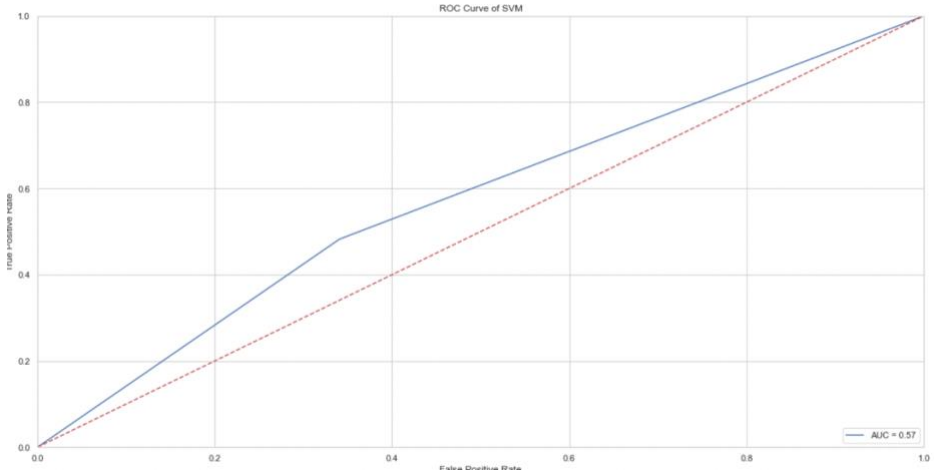
Let us build ROC curve and compute the AUC score of our model and the above plot: 0.58
The diagonal line is a random model with an AUC of 0.5, a model with no skill, which just the same as making a random prediction.



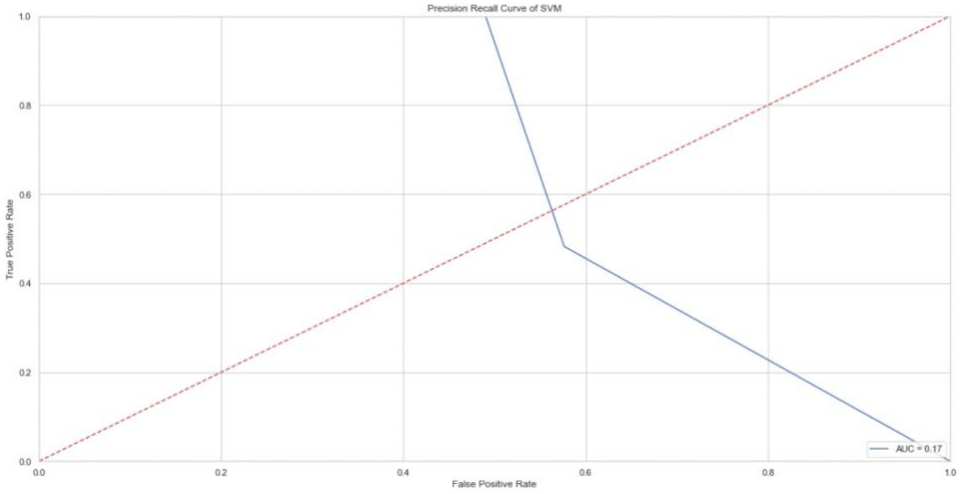
PR Curve for KNN



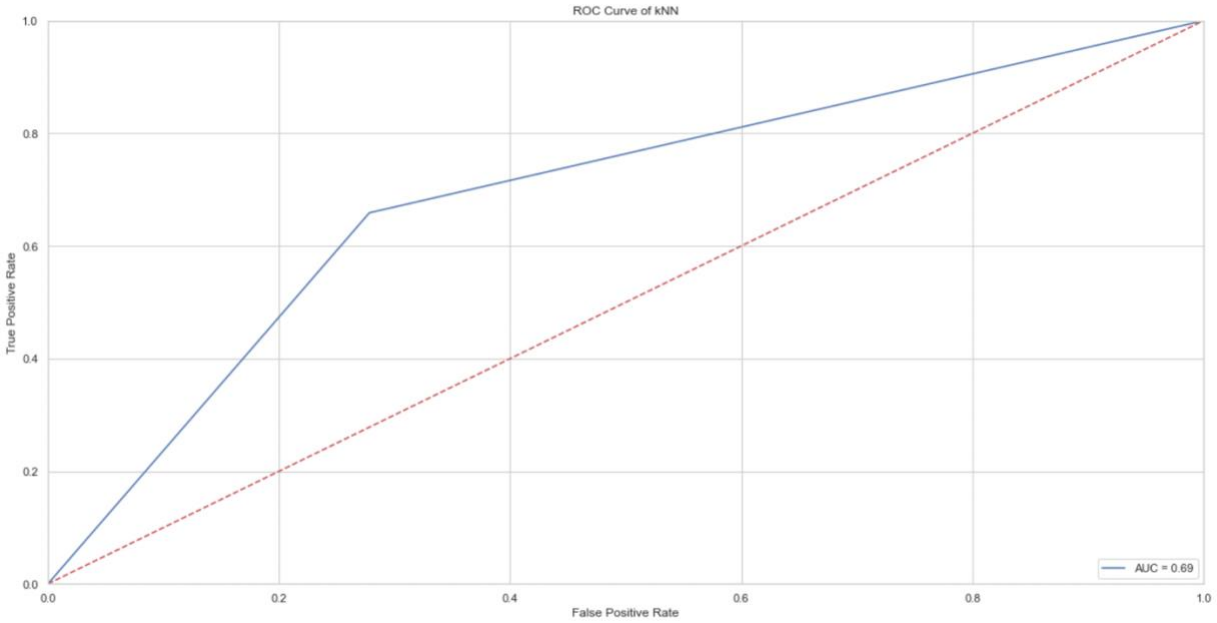
ROC Curve for SVM AUC score of our model and the above plot: 0.57



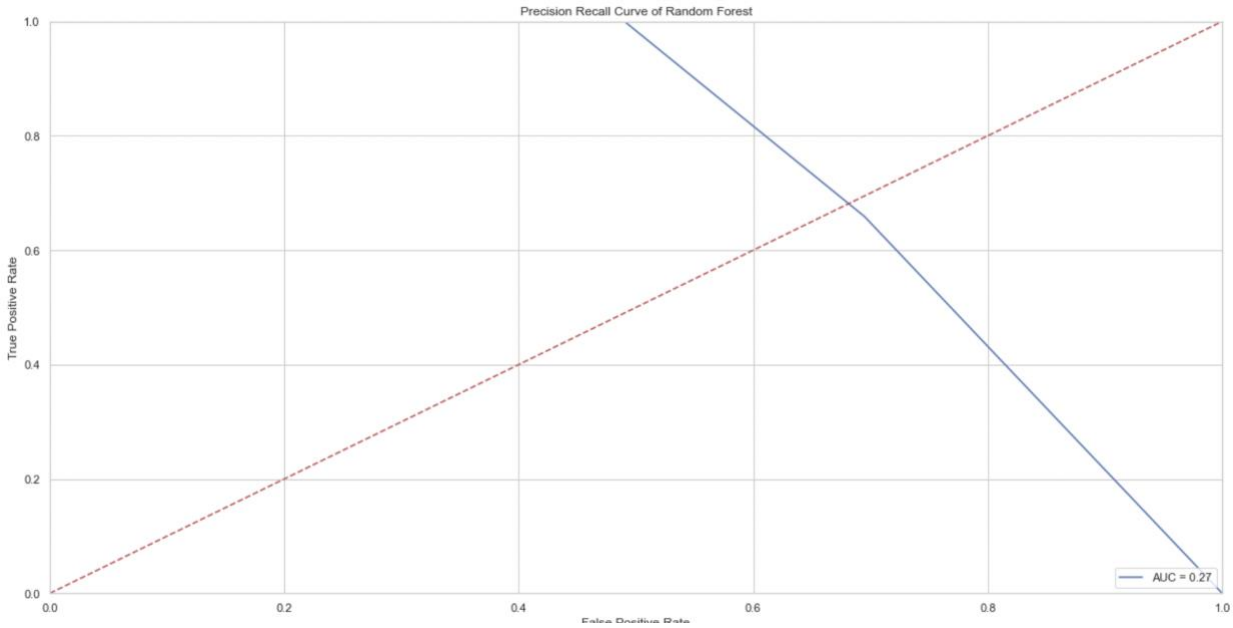
PRC for SVM AUC score of our model and the above plot: 0.17



ROC Curve for Random Forest AUC score of our model and the above plot: 0.69

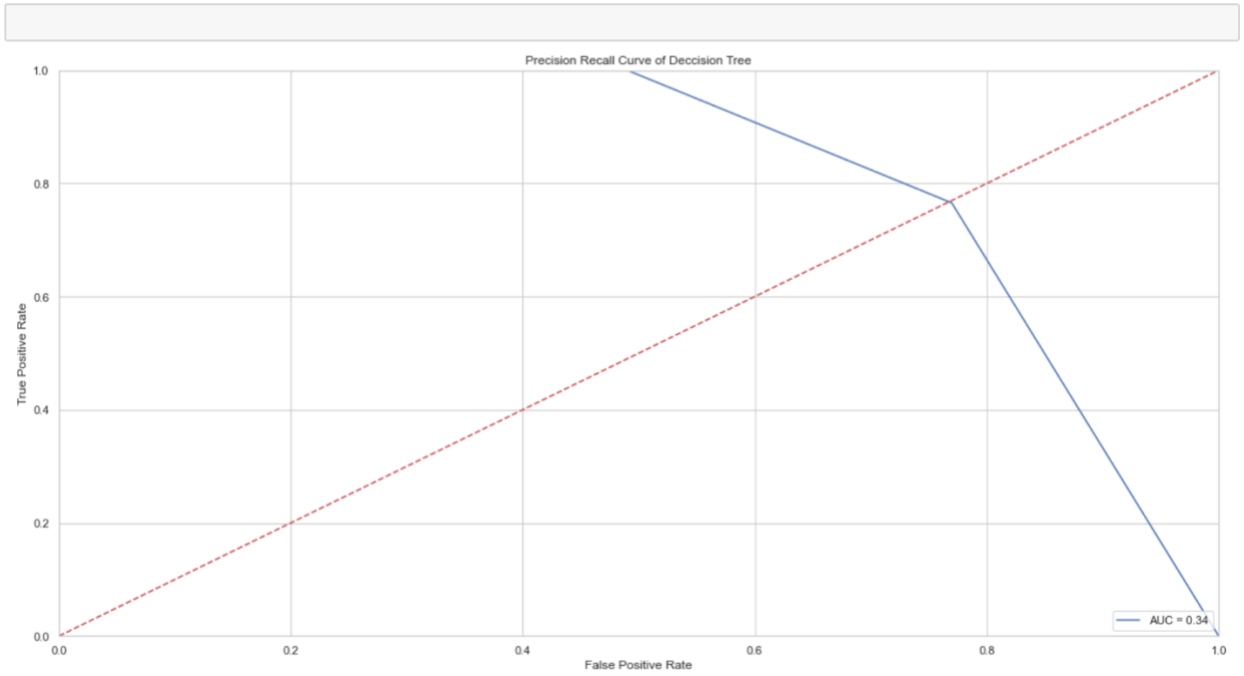


PRC for Random Forest AUC score of our model and the above plot: 0.27



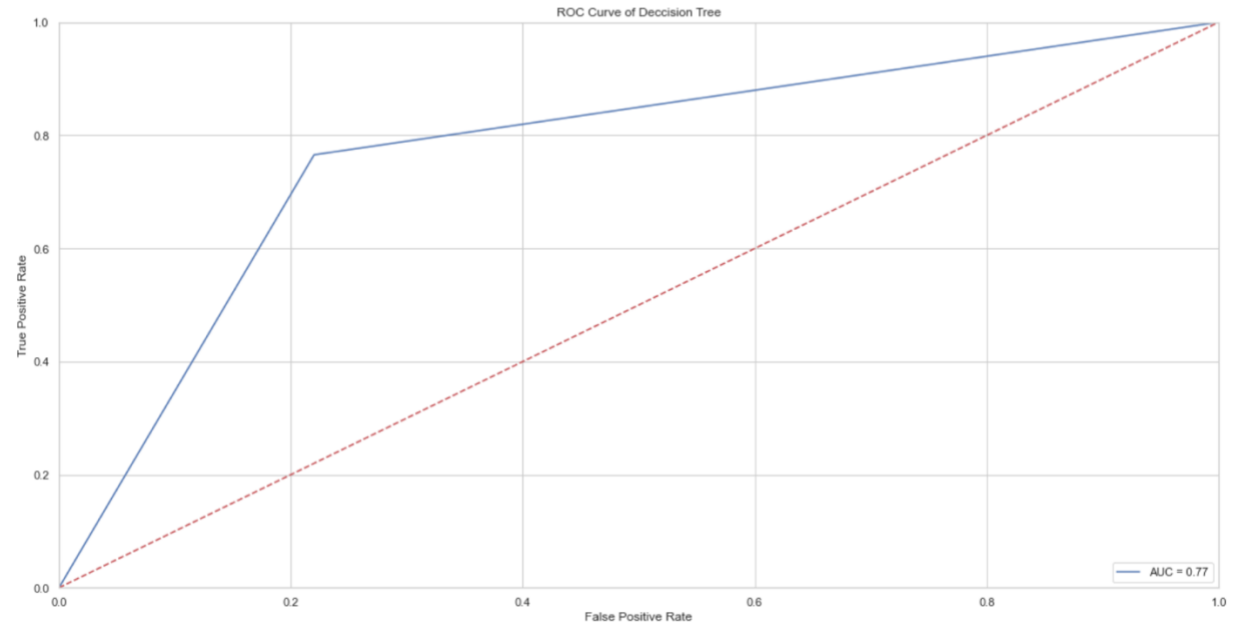
AUC score of our model and the above plot: 0.34

Precision Recall Curve of Deccision Tree



AUC score of our model and the above plot: 0.77

ROC Curve of Deccision Tree



Conclusion

Neither KNN, Random Forest nor the SVM algorithm are very useful in terms of predicting the gender of the customer based on the features Sum, Coded and Type. This indicates that the data does not have prediction capability. This doesn't come as a huge surprise, as we could already see in the EDA that there was little that suggested any major differences between the two genders when it came to these variables. The best accuracy was with Deccision Tree model is 0.77

A high error rate indicates that the model is underfitting and has high bias. The model is not sufficiently complex, so it's simply not capable of representing the relationship between y and the input features. To combat this we could try increasing the number of input features.