Final Project

BD2002

Zhassulan Zhalish
Sanzhar Sailaubek
Aitu Shygys
Sekerbek Dariya

OUTLINE

1. Explore the dataset. Do the descriptive statistics
2. Explanatory data analysis
3. Feature engineering
4. Supervised learning. Build model for prediction the gender of the clients. Decision Trees, KNN, Random Forest. Tune the hyper parameters, grid search, cross validation etc. Visualization of the models etc
5. Analyze models, Result comparison, ROC/AUC, precision and recall curves, deep analyzing.
6. Conclusion.

DESCRIPTIVE STATISTICS

The information contains the number of columns, column labels, column data types, memory usage, range index, and the number of cells in each column (non-null values).

```
transactions.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 130039 entries, 0 to 130038
Data columns (total 5 columns):
 #   Column     Non-Null Count   Dtype
---  ------     --------------   -----
 0   client_id  130039 non-null  int64
 1   datetime   130039 non-null  object
 2   code       130039 non-null  int64
 3   type       130039 non-null  int64
 4   sum        130039 non-null  float64
dtypes: float64(1), int64(3), object(1)
memory usage: 5.0+ MB
```

```
codes.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 184 entries, 0 to 183
Data columns (total 2 columns):
 #   Column            Non-Null Count  Dtype
---  ------            --------------  -----
 0   code              184 non-null    int64
 1   code_description  184 non-null    object
dtypes: int64(1), object(1)
memory usage: 3.0+ KB
```

```
train.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 6000 entries, 0 to 5999
Data columns (total 2 columns):
 #   Column     Non-Null Count  Dtype
---  ------     --------------  -----
 0   client_id  6000 non-null   int64
 1   target     6000 non-null   int64
dtypes: int64(2)
memory usage: 93.9 KB
```

```
types.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 155 entries, 0 to 154
Data columns (total 2 columns):
 #   Column            Non-Null Count  Dtype
---  ------            --------------  -----
 0   type              155 non-null    int64
 1   type_description  155 non-null    object
dtypes: int64(1), object(1)
memory usage: 2.5+ KB
```

DESCRIPTIVE STATISTICS

Transactions transactional data on banking operations

- client_id - the unique id of clients
- datetime - day and time
- code - the code of transaction
- type - the type of transaction
- sum - the income and outcome of clients

	client_id	datetime	code	type	sum
0	96372458	421 06:33:15	6011	2010	-561478.94
1	24567813	377 17:20:40	6011	7010	67377.47
2	21717441	55 13:38:47	6011	2010	-44918.32
3	14331004	263 12:57:08	6011	2010	-3368873.66
4	85302434	151 10:34:12	4814	1030	-3368.87

Training set with client gender marking (0/1 - client gender)

- client_id - the id of client
- targer - gender, where 0/1 stands for women and man

	client_id	target
0	75063019	0
1	86227647	1
2	6506523	0
3	50615998	0
4	95213230	0

DESCRIPTIVE STATISTICS

Types - reference of transaction types

- type - the type of transaction
- type_description - the desription of types-

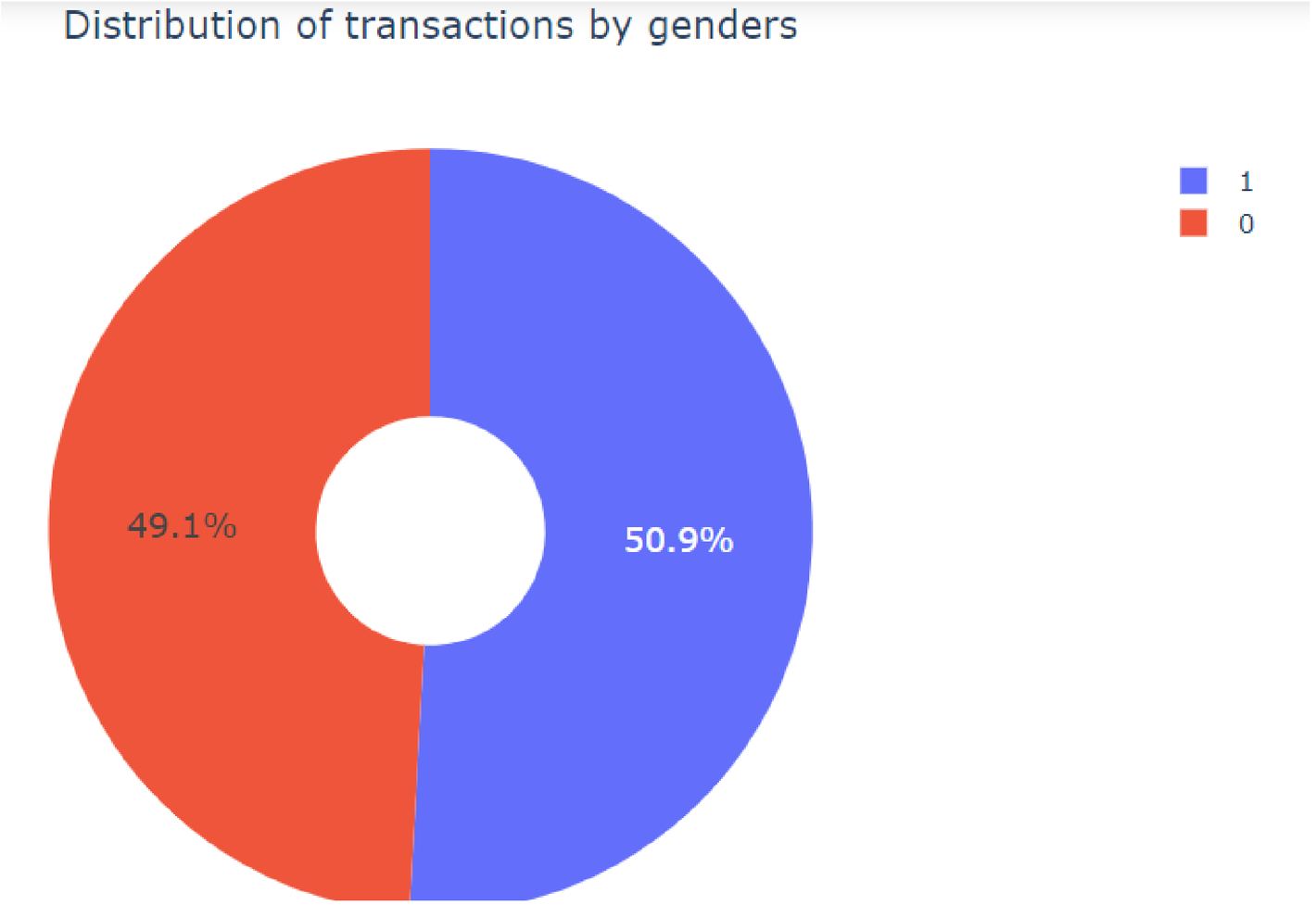
	type	type_description
0	8001	Установление расх. лимита по карте
1	2411	Перевод с карты на счет др.лица в одном тер. б...
2	4035	н/д(нет данных)
3	3001	Комиссия за обслуживание ссудного счета
4	2420	Перевод с карты на счет физ.лица в другом тер....

Codes reference of transaction codes

- code - the type of code
- code_description - the desription of codes

	code	code_description
0	5944	Магазины по продаже часов, ювелирных изделий и...
1	5621	Готовые сумочные изделия
2	5697	Услуги по переделке, починке и пошиву одежды
3	7995	Транзакции по азартным играм
4	5137	Мужская, женская и детская спец-одежда

EXPLANATORY DATA ANALYSIS




```
fig = go.Figure([go.Pie(values=dist_by_gender['client_id'],
                        hole=0.3)])

fig.update_traces( textinfo='percent', textfont_size=15)
fig.update_layout(title="Distribution of transactions by genders",title_x=0.5)
fig.show()
```


Top 5 commonly code description
Count
20k
15k
10k
5k
0
Финансовые институты — снятие наличности автоматически
Финансовые институты — снятие наличности вручную
Звонки с использованием телефонов, считывающих магнитную ленту
Бакалейные магазины, супермаркеты
Денежные переводы
code_description

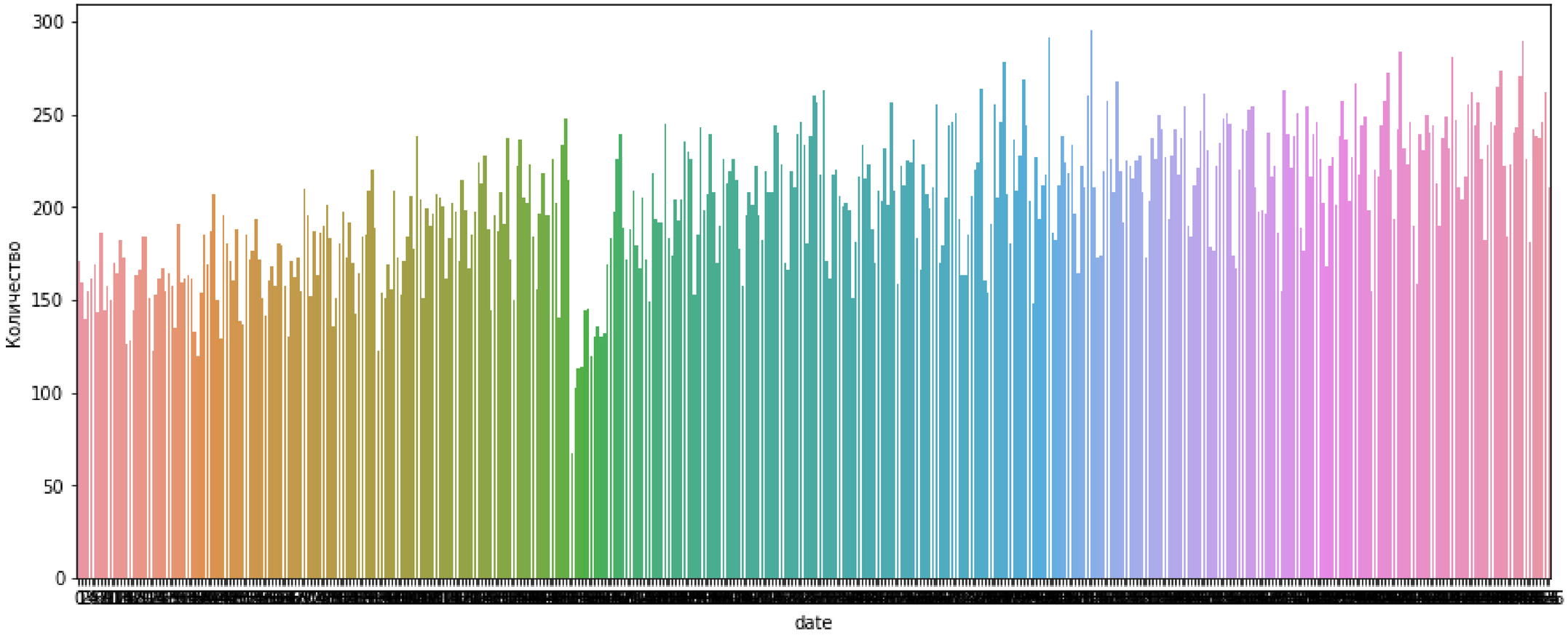
EXPLANATORY DATA ANALYSIS

CORRELATION BETWEEN VARIABLES


```
corrmat = transactions.corr()
corr_features = corrmat.index

plt.figure(figsize=(10,10))
#Plotting heat map
g=sns.heatmap(transactions[corr_features].corr(),annot=True,linewidths=.1)
b, t = plt.ylim() # Finding the values for bottom and top
b += 0.5
t -= 0.5
plt.ylim(b, t)
plt.show()
# showing correlation
```

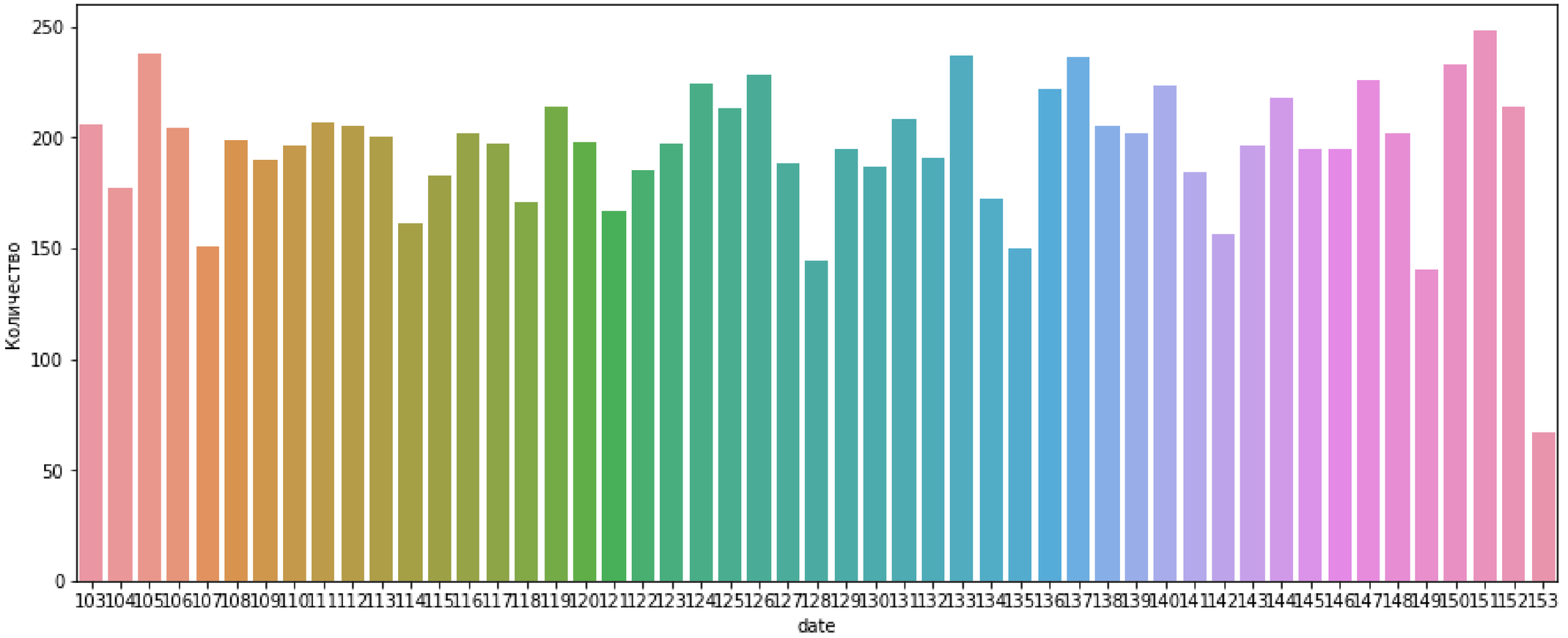
EXPLANATORY DATA ANALYSIS

DISTRIBUTION BY DAYS



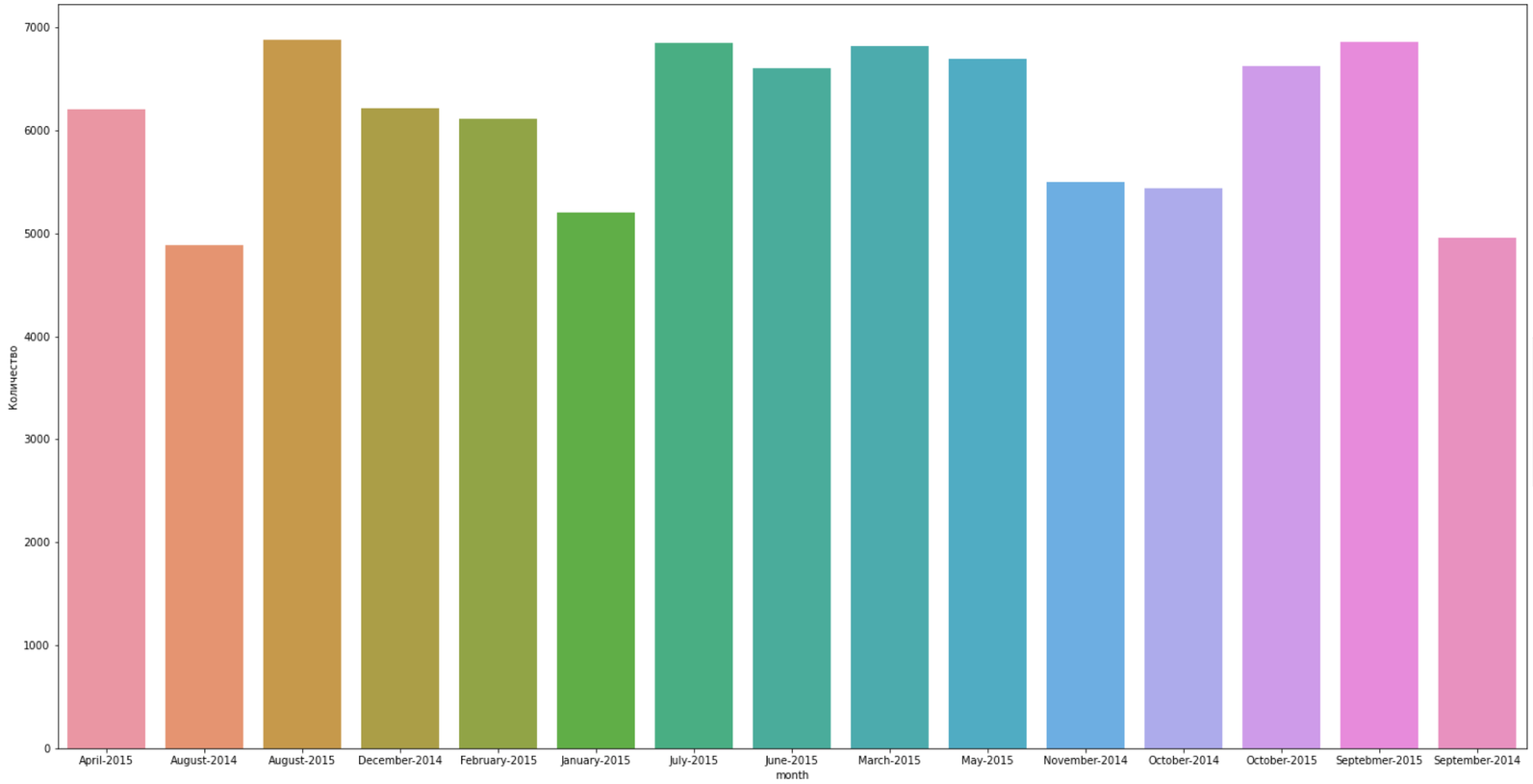
EXPLANATORY DATA ANALYSIS

DISTRIBUTION BY DAYS



EXPLANATORY DATA ANALYSIS

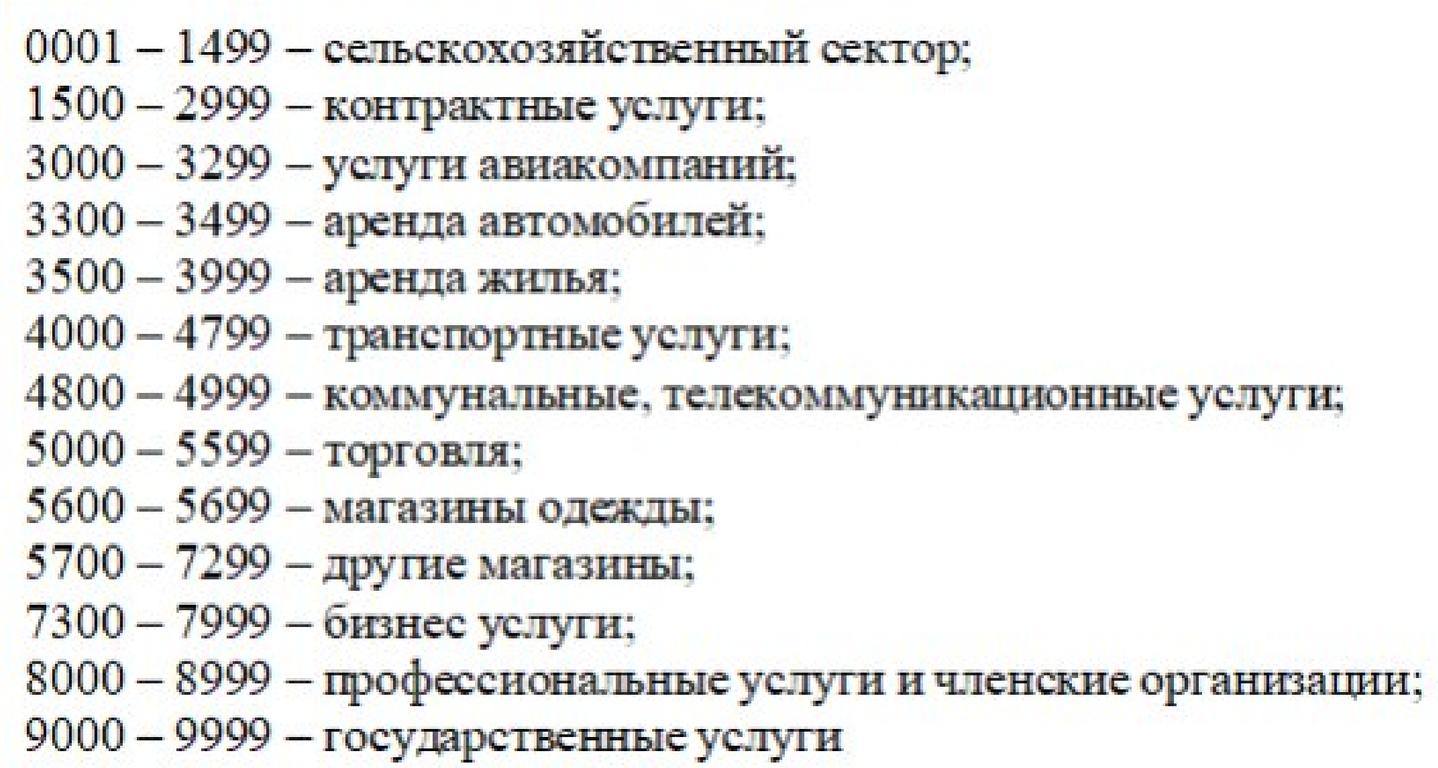
DISTRIBUTION BY MONTH




```
by_month = transactions.groupby('month').count().reset_index()
plt.figure(figsize=(25,13))
sns.barplot(x = by_month['month'], y = by_month['client_id'])
plt.ylabel("Количество")
```

FEATURE ENGINEERING

codes
0001-1499 -> agriculture,
1500-2999 -> contracts,
3000-3299 -> avia_services,
3300-3499 -> car_rent,
3500-3999 -> house_rent,
4000-4799 -> transports,
4800-4999 -> pub_utilities,
5000-5599 -> trade,
5600-5699 -> clothes_shops,
5700-7299 -> other_shops,
7300-7999 -> business_services,
8000-8999 -> prof_services,
9000-9999 -> gov_services.


0001 – 1499 – сельскохозяйственный сектор;
1500 – 2999 – контрактные услуги;
3000 – 3299 – услуги авиакомпаний;
3300 – 3499 – аренда автомобилей;
3500 – 3999 – аренда жилья;
4000 – 4799 – транспортные услуги;
4800 – 4999 – коммунальные, телекоммуникационные услуги;
5000 – 5599 – торговля;
5600 – 5699 – магазины одежды;
5700 – 7299 – другие магазины;
7300 – 7999 – бизнес услуги;
8000 – 8999 – профессиональные услуги и членские организации;
9000 – 9999 – государственные услуги

These are unique codes in the bank world, so we did check and all of them fitted with the conditions. And we created new 13 columns/features from codes column, new 2 columns from sum column

FEATURE ENGINEERING

types
salary_contr -> sum > 0 and type in (7000)
atm_contr -> sum > 0 and type in (7010, 7011, 7014, 7015, 7020, 7021, 7024, 7025, 7060)
refill_acc -> sum > 0 and type in (7050, 7081, 7082, 7084)
refll_card -> sum > 0 and type in (7030, 7031, 7034, 7035, 7040, 7041, 7044, 7045, 7070, 7071, 7074, 7075)
cash_withdr -> sum < 0 and type in (2000, 2001, 2010, 2011, 2020, 2021, 2100, 2110, 2200, 2210, 2900)
transfer_acc -> sum < 0 and type in (2401, 2402, 2412, 2422, 2432)
transfer_card -> sum < 0 and type in (2330, 2331, 2340, 2341, 2370, 2371, 2440, 2446, 2456, 2460)
transfer_leg_ent -> sum < 0 and type in (2320, 2323, 2325) and not (mcc_code eq 4900)
purchase -> sum < 0 and (type in (1000, 1010, 1030, 1100, 1110, 1200, 1210, 1310, 1410, 1510, 2901, 2992, 8035, 8100, 8145, 2992, 8035, 8100, 8146) or code == 4900) and code != -1
So, from types column we created new 9 features dividing by type of transactions.

FEATURE ENGINEERING

```
# working with parts of day
day_part = pd.DataFrame(transactions["time"].str.split(':', expand=True).values,
             columns=['Hour', 'Min', 'Sec'])
transactions['hour'] = (day_part['Hour']).astype(int)

transactions['early_mor'] = ((transactions['hour']>4) & (transactions['hour']<8)).
transactions['morning'] = ((transactions['hour']>=8) & (transactions['hour']<=10))
transactions['late_mor'] = ((transactions['hour']>10) & (transactions['hour']<=11)
transactions['early_aft'] = ((transactions['hour']>12) & (transactions['hour']<=14
transactions['late_aft'] = ((transactions['hour']>14) & (transactions['hour']<=16)
transactions['early_eve'] = ((transactions['hour']>16) & (transactions['hour']<=18
transactions['late_eve'] = ((transactions['hour']>18) & (transactions['hour']<=21)
transactions['night'] = ((transactions['hour']>21) & (transactions['hour']<=4)).as
# transactions.head()
```

Here we worked with time column, with its hour part. We divided it to parts of a day, morning: early morning, morning, late morning; afternoon: early afternoon, late afternoon; evening: early evening late evening; and night.

FEATURE ENGINEERING

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 6000 entries, 0 to 5999
Data columns (total 50 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   client_id          6000 non-null   int64
 1   plus_sum           6000 non-null   float64
 2   minus_sum          6000 non-null   float64
 3   income             6000 non-null   int32
 4   outcome            6000 non-null   int32
 5   agriculture        6000 non-null   int32
 6   contracts          6000 non-null   int32
 7   avia_services      6000 non-null   int32
 8   car_rent           6000 non-null   int32
 9   house_rent         6000 non-null   int32
 10  transports         6000 non-null   int32
 11  pub_utilities      6000 non-null   int32
 12  trade              6000 non-null   int32
 13  clothes_shops      6000 non-null   int32
 14  other_shops        6000 non-null   int32
 15  business_services  6000 non-null   int32
 16  prof_services      6000 non-null   int32
 17  gov_services       6000 non-null   int32
 18  atm contr          6000 non-null   int32
```

So, we generated new 50 columns

FEATURE ENGINEERING

```
# created new features with sum column by ranges
transactions['-50000-0'] = ((transactions['sum']<0) & (transactions['sum']>=-50000)).astype(int)
transactions['-100000--50000'] = ((transactions['sum']<-50000) & (transactions['sum']>=-100000)).astype(int
transactions['-250000--100000'] = ((transactions['sum']<-100000) & (transactions['sum']>=-250000)).astype(
transactions['-500000--250000'] = ((transactions['sum']<-250000) & (transactions['sum']>=-500000)).astype(
transactions['--500000'] = ((transactions['sum']<-500000)).astype(int)
transactions['50000-100000'] = ((transactions['sum']>50000) & (transactions['sum']<=100000)).astype(int)
transactions['100000-250000'] = ((transactions['sum']>100000) & (transactions['sum']<=250000)).astype(int)
transactions['250000-500000'] = ((transactions['sum']>250000) & (transactions['sum']<=500000)).astype(int)
transactions['500000-'] = ((transactions['sum']>500000)).astype(int)
transactions.head()
```

✓ 0.3s Python

	client_id	datetime	code	type	sum	date	time	plus_sum	minus_sum	-50000-0	-100000--50000	-250000--100000
0	96372458	421 06:33:15	6011	2010	-561478.94	421	06:33:15	0.00	-561478.94	0	0	0
1	24567813	377 17:20:40	6011	7010	67377.47	377	17:20:40	67377.47	0.00	0	0	0
2	21717441	55 13:38:47	6011	2010	-44918.32	55	13:38:47	0.00	-44918.32	1	0	0
3	14331004	263	6011	2010	-3368873.66	263	12:57:08	0.00	-3368873.66	0	0	0

We created some new features by using sum column

FEATURE ENGINEERING

```
col_names = transactions.columns.tolist()
# dropping unneeded columns that has more nulls
for i in col_names:
    n = len(transactions[transactions[i]==0])
    if n>5500:
        print(i, ": ", n)
        transactions.drop([i], axis=1, inplace=True)
transactions
```

1] ✓ 0.6s

```
agriculture :  5997
contracts :  5997
avia_services :  5968
car_rent :  5999
house_rent :  5990
transports :  5686
prof_services :  5609
gov_services :  5976
refill_acc :  6000
transfer_acc :  6000
transfer_leg_ent :  6000
```

Deleting unneeded column that have so much nulll values

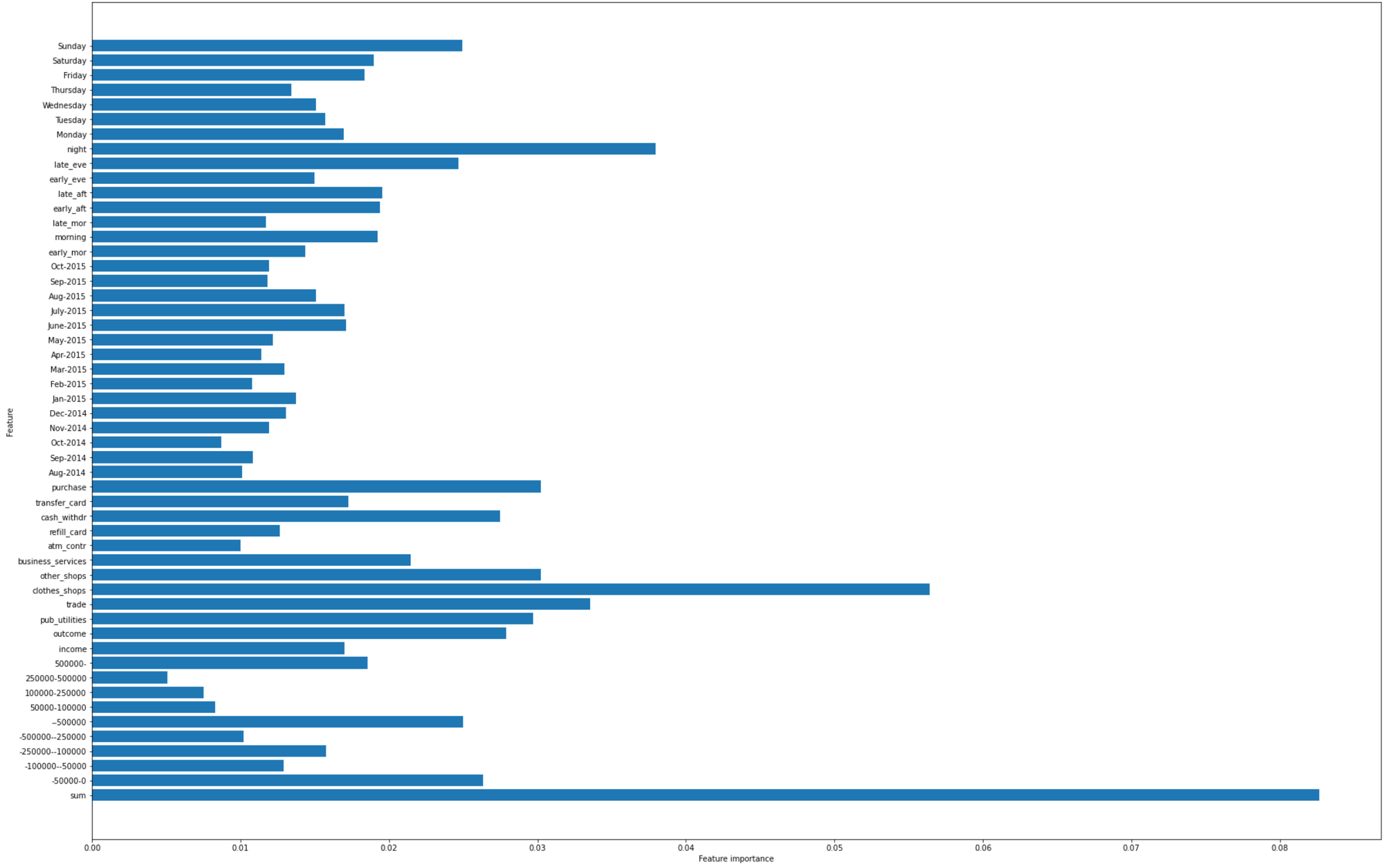
Random forest

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt
import numpy as np
%matplotlib inline
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.25, random_state=20)
forest = RandomForestClassifier(max_depth=10, min_samples_leaf=5, n_estimators=200, random_state=120)
forest.fit(X_train, y_train)

print("Accuracy on training set: {:.3f}".format(forest.score(X_train, y_train)))
print("Accuracy on test set: {:.3f}".format(forest.score(X_test, y_test)))
```

[357] ✓ 3.3s Python

```
Accuracy on training set: 0.782
Accuracy on test set: 0.625
```

Random forest



Random forest

```
rf_best = grid_search.best_estimator_
rf_best
```

[363] ✓ 0.2s Pyth

```
RandomForestClassifier(max_depth=25, min_samples_leaf=5, n_estimators=200,
                       random_state=120)
```

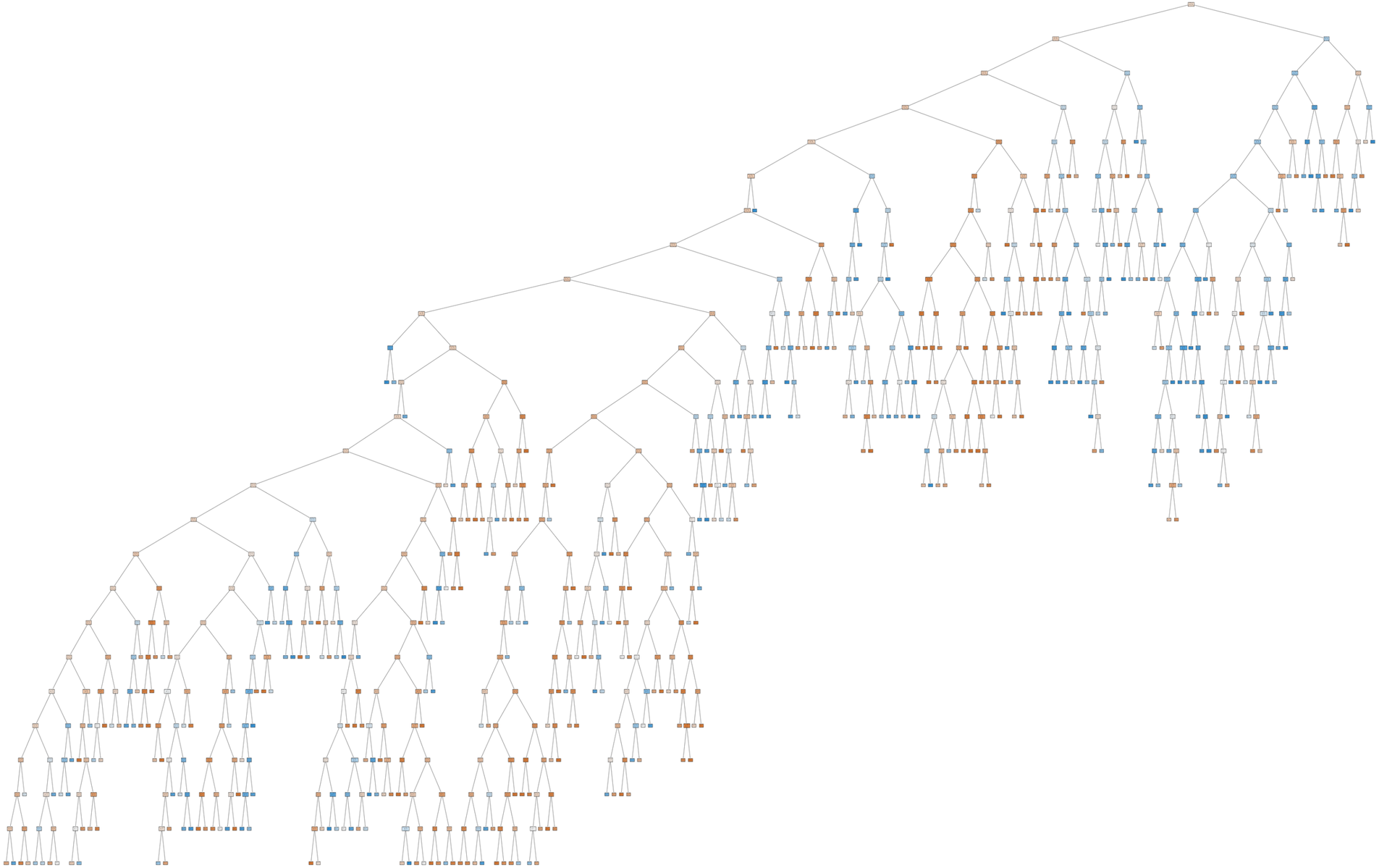
+ Code + Markdown

```
forest = RandomForestClassifier(max_depth=25, min_samples_leaf=5, n_estimators=200,
                                random_state=120)
forest.fit(X_train, y_train)
print("Accuracy on training set: {:.3f}".format(forest.score(X_train, y_train)))
print("Accuracy on test set: {:.3f}".format(forest.score(X_test, y_test)))
```

[364] ✓ 4.6s Pyth

```
Accuracy on training set: 0.946
Accuracy on test set: 0.625
```

Random forest



Random forest

Test Result:

==========

Accuracy Score: 62.33%

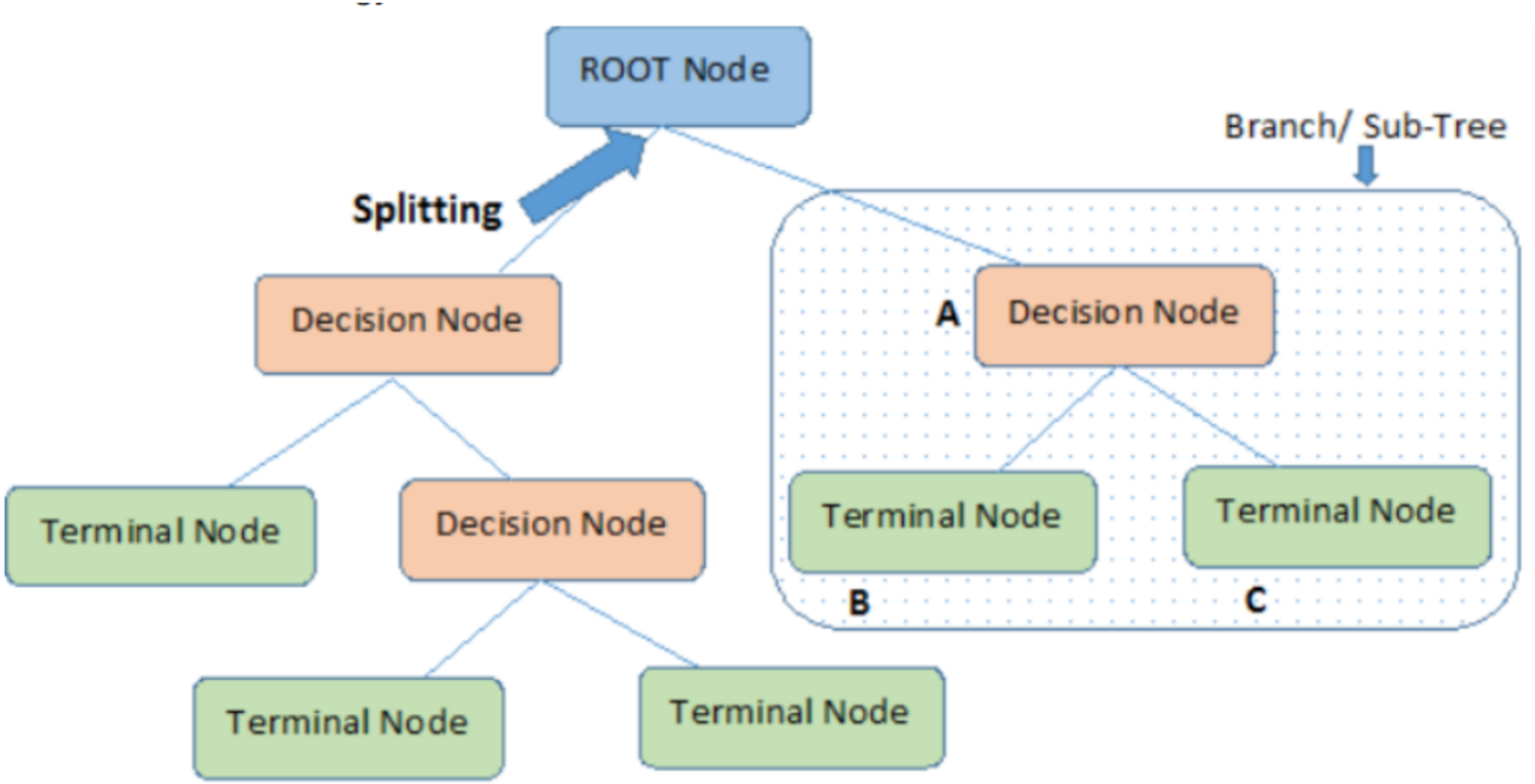
CLASSIFICATION REPORT:

	0	1	accuracy	macro avg	weighted avg
precision	0.640483	0.589744	0.623333	0.615113	0.618293
recall	0.753555	0.455793	0.623333	0.604674	0.623333
f1-score	0.692433	0.514187	0.623333	0.603310	0.614480
support	844.000000	656.000000	0.623333	1500.000000	1500.000000

Confusion Matrix:

```
[[636 208]
[357 299]]
```

Decision Tree



Decision Tree

```
B [527]: new_transactions=transactions.drop(['plus_sum','minus_sum'], axis=1)
```

```
B [531]: X = new_transactions.drop(['target'], axis=1)

         y = new_transactions['target']
```

```
B [532]: # Load libraries
         import pandas as pd
         from sklearn.tree import DecisionTreeClassifier # Import Decision Tree Classifier
         from sklearn.model_selection import train_test_split # Import train_test_split function
         from sklearn import metrics #Import scikit-learn metrics module for accuracy calculation
```

```
B [533]: X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.33, random_state = 42)
```

Entropy

Entropy measures the impurity in the given dataset. In Physics and Mathematics, entropy is referred to as the randomness or uncertainty of a random variable X. In information theory, it refers to the impurity in a group of examples. **Information gain** is the decrease in entropy. Information gain computes the difference between entropy before split and average entropy after split of the dataset based on given attribute values.

Entropy is represented by the following formula:-

$$Entropy = \sum_{i=1}^{C} -p_i * \log_2(p_i)$$

Here, **c** is the number of classes and **pi** is the probability associated with the ith class.

Gini index

$$Gini = 1 - \sum_{i=1}^{C} (p_i)^2$$

Here, again **c** is the number of classes and **pi** is the probability associated with the ith class.

Gini index says, if we randomly select two items from a population, they must be of the same class and probability this is 1 if the population is pure.

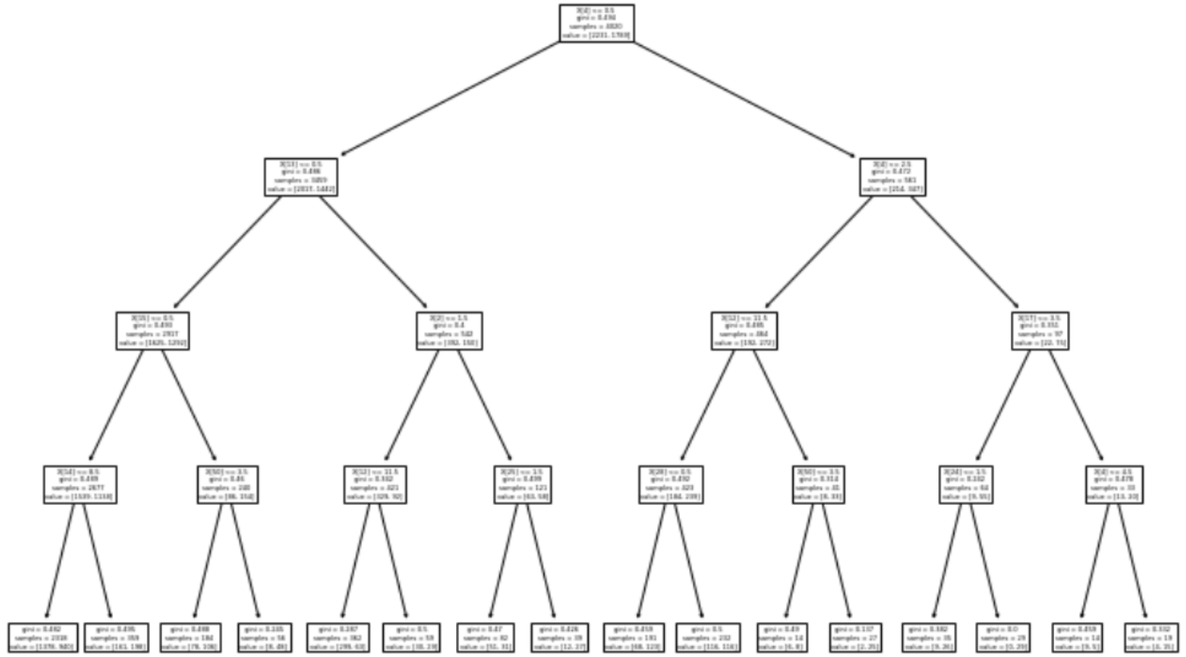
It works with the categorical target variable “Success” or “Failure”. It performs only binary splits. The higher the va of Gini, higher the homogeneity. CART (Classification and Regression Tree) uses the Gini method to create binary splits.

Decision Tree(Gini impurity)

[581]:
```python
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report

def print_score(clf, X_train, y_train, X_test, y_test, train=True):
    if train:
        pred = clf.predict(X_train)
        clf_report = pd.DataFrame(classification_report(y_train, pred, output_dict=True))
        print("Train Result:\n================================================")
        print(f"Accuracy Score: {accuracy_score(y_train, pred) * 100:.2f}%")
        print("_______________________________________________")
        print(f"CLASSIFICATION REPORT:\n{clf_report}")
        print("_______________________________________________")
        print(f"Confusion Matrix: \n {confusion_matrix(y_train, pred)}\n")

    elif train==False:
        pred = clf.predict(X_test)
        clf_report = pd.DataFrame(classification_report(y_test, pred, output_dict=True))
        print("Test Result:\n================================================")
        print(f"Accuracy Score: {accuracy_score(y_test, pred) * 100:.2f}%")
        print("_______________________________________________")
        print(f"CLASSIFICATION REPORT:\n{clf_report}")
        print("_______________________________________________")
        print(f"Confusion Matrix: \n {confusion_matrix(y_test, pred)}\n")
```

```
Train Result:
================================================
Accuracy Score: 63.31%
_______________________________________________
CLASSIFICATION REPORT:
                    0            1  accuracy    macro avg  weighted avg
precision    0.625832     0.654528  0.633085     0.640180      0.638602
recall       0.842671     0.371716  0.633085     0.607194      0.633085
f1-score     0.718243     0.474153  0.633085     0.596198      0.609617
support    2231.000000  1789.000000  0.633085  4020.000000   4020.000000
_______________________________________________
Confusion Matrix:
 [[1880  351]
 [1124  665]]

Test Result:
================================================
Accuracy Score: 58.74%
_______________________________________________
CLASSIFICATION REPORT:
                    0           1  accuracy    macro avg  weighted avg
precision    0.599455    0.552734  0.587374     0.576095      0.578903
recall       0.793508    0.324914  0.587374     0.559211      0.587374
f1-score     0.682965    0.409255  0.587374     0.546110      0.562560
support    1109.000000  871.000000  0.587374  1980.000000   1980.000000
_______________________________________________
Confusion Matrix:
 [[880 229]
 [588 283]]
```

Decision Tree(after hyperparameter tuning)



Accuracy Score: 61.89%

CLASSIFICATION REPORT:

	0	1	accuracy	macro avg	weighted avg
precision	0.613955	0.634837	0.618905	0.624396	0.623248
recall	0.844016	0.338178	0.618905	0.591097	0.618905
f1-score	0.710834	0.441284	0.618905	0.576059	0.590878
support	2231.000000	1789.000000	0.618905	4020.000000	4020.000000

Confusion Matrix:

```
[[1883  348]
 [1184  605]]
```

Test Result:

Accuracy Score: 58.13%

CLASSIFICATION REPORT:

	0	1	accuracy	macro avg	weighted avg
precision	0.591743	0.546256	0.581313	0.568999	0.571733
recall	0.814247	0.284730	0.581313	0.549489	0.581313
f1-score	0.685389	0.374340	0.581313	0.529864	0.548559
support	1109.000000	871.000000	0.581313	1980.000000	1980.000000

Confusion Matrix:

```
[[903 206]
 [623 248]]
```

Best paramters: {'criterion': 'gini', 'max_depth': 4, 'min_samples_leaf': 14, 'min_samples_split': 2})

Train Result:

Decision Tree(Entropy impurity)

B [548]:
```
print('Training set score: {:.4f}'.format(clf_en.score(X_train, y_train)))

print('Test set score: {:.4f}'.format(clf_en.score(X_test, y_test)))
```

```
Training set score: 0.6055
Test set score: 0.5848
```

Confusion matrix

$$\text{Precision} = \frac{True\ Positive}{True\ Positive + False\ Positive}$$

Precision talks about how precise/accurate your model is out of those predicted positive, how many of them are actual positive.

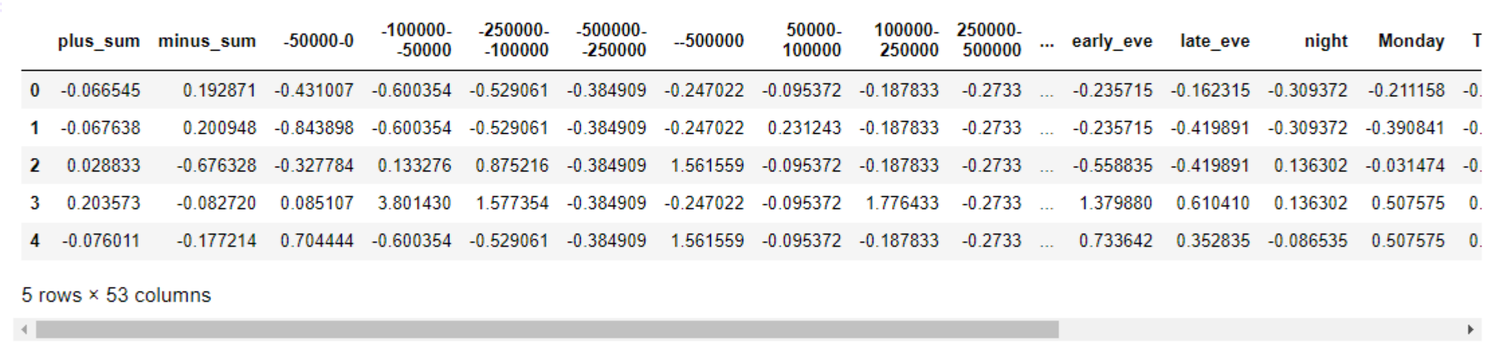
$$\text{Recall} = \frac{True\ Positive}{True\ Positive + False\ Negative}$$

Recall actually calculates how many of the Actual Positives our model capture through labeling it as Positive (True Positive).

$$\text{F1} = 2 \times \frac{Precision * Recall}{Precision + Recall}$$

F1 Score is needed when you want to seek a balance between Precision and Recall.

KNN algorithm

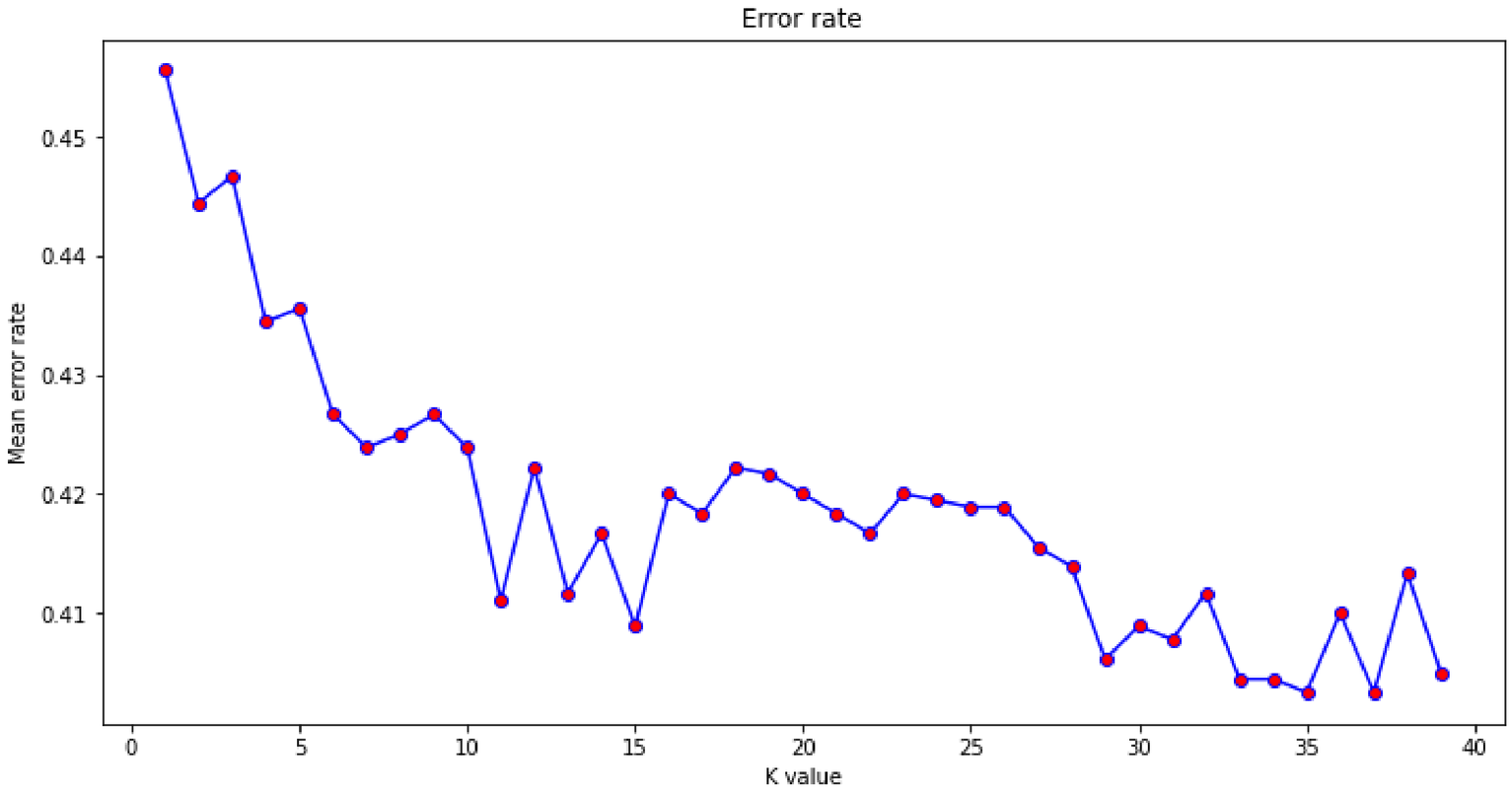


	plus_sum	minus_sum	-50000-0	-100000--50000	-250000--100000	-500000--250000	--500000	50000-100000	100000-250000	250000-500000	...	early_eve	late_eve	night	Monday	T
0	-0.066545	0.192871	-0.431007	-0.600354	-0.529061	-0.384909	-0.247022	-0.095372	-0.187833	-0.2733	...	-0.235715	-0.162315	-0.309372	-0.211158	-0.
1	-0.067638	0.200948	-0.843898	-0.600354	-0.529061	-0.384909	-0.247022	0.231243	-0.187833	-0.2733	...	-0.235715	-0.419891	-0.309372	-0.390841	-0.
2	0.028833	-0.676328	-0.327784	0.133276	0.875216	-0.384909	1.561559	-0.095372	-0.187833	-0.2733	...	-0.558835	-0.419891	0.136302	-0.031474	-0.
3	0.203573	-0.082720	0.085107	3.801430	1.577354	-0.384909	-0.247022	-0.095372	1.776433	-0.2733	...	1.379880	0.610410	0.136302	0.507575	0.
4	-0.076011	-0.177214	0.704444	-0.600354	-0.529061	-0.384909	1.561559	-0.095372	-0.187833	-0.2733	...	0.733642	0.352835	-0.086535	0.507575	0.

5 rows × 53 columns

```
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
scaler.fit(transactions.drop('target', axis=1)) #standarize our data without
the target column
# Transforming the data
scaled_features = scaler.transform(transactions.drop('target', axis=1))
scaled_features

# Use the scaler to create scaler dataframe
# This gives us a standardized version of our data
independent = pd.DataFrame(scaled_features,
columns=transactions.columns[:-1])
independent.head()

#Splitting our data
X = independent
y = transactions['target']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,
random_state=101)
```

KNN algorithm



[0.4555555555555555,
0.4444444444444444,
0.4666666666666666,
0.43444444444444447,
0.4355555555555553,
0.4266666666666667,
0.42388888888888887,
0.425,

```
for i in range(1,40): # Checking every possible k value between 1-40
    knn = KNeighborsClassifier(n_neighbors=i)#go throw loop
    knn.fit(X_train, y_train)#Fit the k-nearest neighbors classifier from
the training datase
    pred_i = knn.predict(X_test)#Predict the class labels for the X_test
    error_rate.append(np.mean(pred_i != y_test))
```

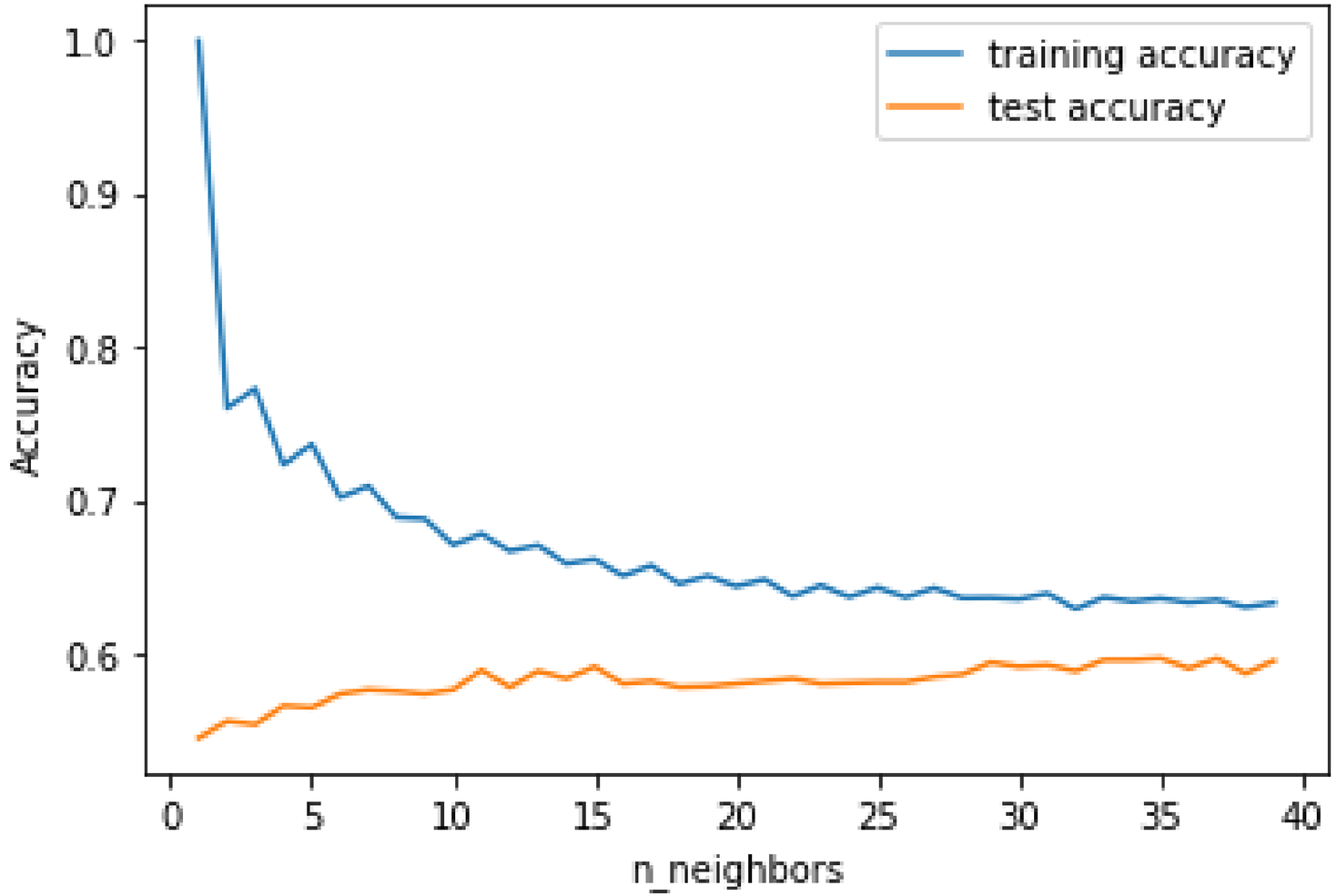
KNN algorithm

```
B [73]: knn = KNeighborsClassifier(n_neighbors=37)# the min error rate at k value 37
knn.fit(X_train, y_train)#Fit the k-nearest neighbors classifier from the training datase
y_pred = knn.predict(X_test)#predict to 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)*100) + '%')
```

```
[[806 176]
 [550 268]]
              precision    recall  f1-score   support

           0       0.59      0.82      0.69       982
           1       0.60      0.33      0.42       818

    accuracy                           0.60      1800
   macro avg       0.60      0.57      0.56      1800
weighted avg       0.60      0.60      0.57      1800

Accuracy Score: 59.66666666666667%
```

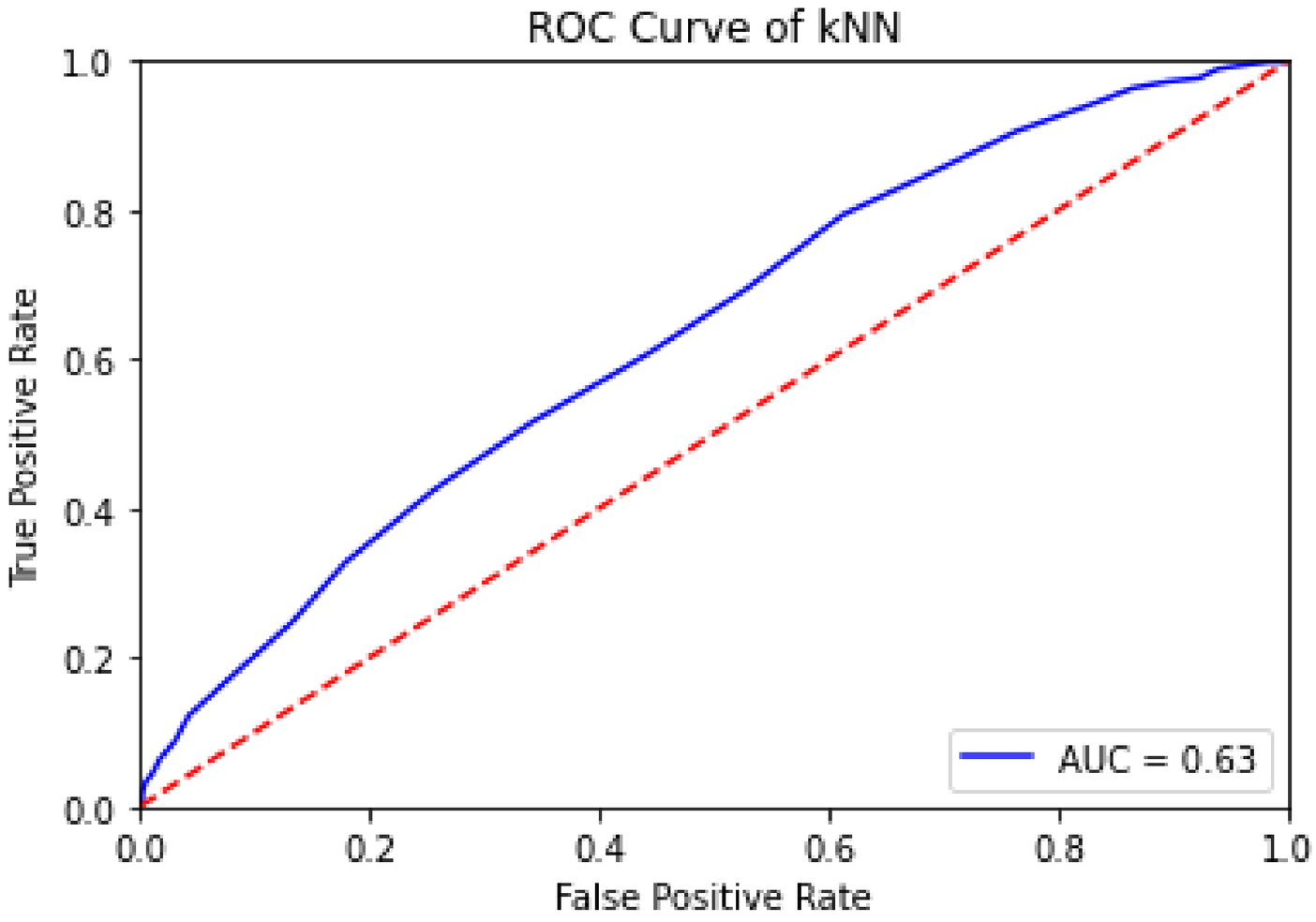
for the class 0:
Precision is the ratio of correctly predicted positive observations to the total predicted positive observations. High precision(точность) relates to the low false positive rate. We have got 0.59 precision which is not bad.
Recall (Sensitivity) - Recall is the ratio of correctly predicted positive observations to the all observations in actual class - yes. We have got recall of 0.82 which is good for this model as it’s above 0.5.
F1 score - F1 Score is the weighted average of Precision and Recall. Therefore, this score takes both false positives and false negatives into account.

KNN algorithm



KNN algorithm

```
from sklearn import metrics
y_scores = knn.predict_proba(X_test)
fpr, tpr, threshold = metrics.roc_curve(y_test, y_scores[:, 1])
roc_auc =  metrics.auc(fpr, tpr)

plt.title('Receiver Operating Characteristic')
plt.plot(fpr, tpr, 'b', label = 'AUC = %0.2f' % roc_auc)
plt.legend(loc = 'lower right')
plt.plot([0, 1], [0, 1],'r--')
plt.xlim([0, 1])
plt.ylim([0, 1])
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.title('ROC Curve of kNN')
plt.show()
```



Result comparison

```
Accuracy on test set: 0.625
```

Random Forest

```
Accuracy Score: 59.666666666666667%
```

KNN

```
Accuracy Score: 58.13%
```

Decision Tree

Conclusion

- Understand the datasets and its features in order to work with it
- And we made a deep statistical analysis with visualizations by using numpy, pandas, matplotlib, seaborn and etc.
- The next step was creating new features from existing columns like code, type, datetime and etc.
- The last step was supervised learning. Making prediction with various models. Compare result of accuracies