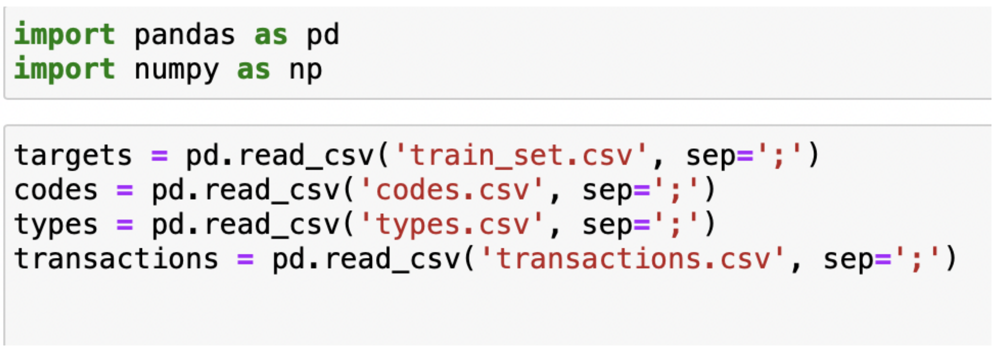
In this assignment we will work with the certain datasets described below (figure 1)



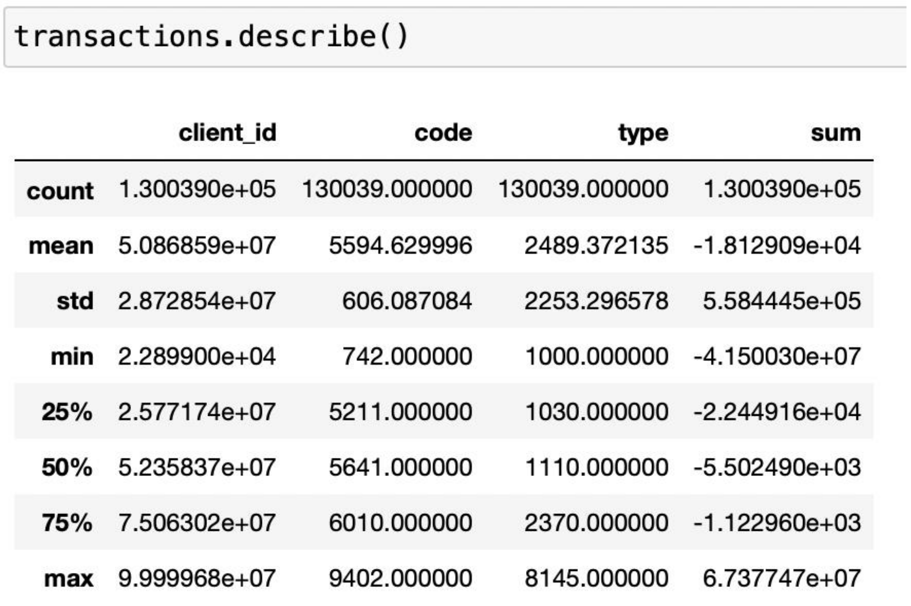
```
import pandas as pd
import numpy as np
```

```
targets = pd.read_csv('train_set.csv', sep=';')
codes = pd.read_csv('codes.csv', sep=';')
types = pd.read_csv('types.csv', sep=';')
transactions = pd.read_csv('transactions.csv', sep=';')
```

Figure 1

Descriptive statistics and analysis:

In order to understand the structure and get an overall picture of the 'transactions' dataset, we used describe() method (figure 2). This method showed us that there are 1.3e5 records. The one thing that we have noticed is the negative value of average sum value, and based on this, we assumed that there is an outflow of money. In next steps we will try to justify our assumptions.



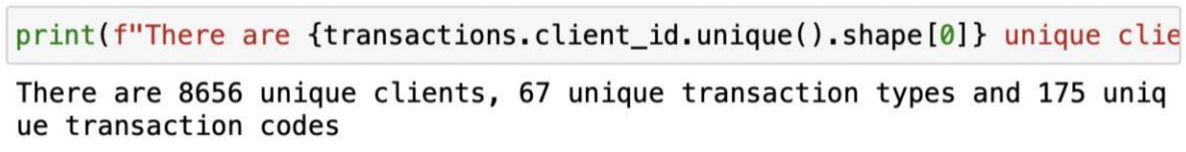
```
transactions.describe()
```

```
          client_id            code            type            sum
count  1.300390e+05   130039.000000   130039.000000   1.300390e+05
 mean  5.086859e+07     5594.629996     2489.372135  -1.812909e+04
  std  2.872854e+07      606.087084     2253.296578   5.584445e+05
  min  2.289900e+04      742.000000     1000.000000  -4.150030e+07
  25%  2.577174e+07     5211.000000     1030.000000  -2.244916e+04
  50%  5.235837e+07     5641.000000     1110.000000  -5.502490e+03
  75%  7.506302e+07     6010.000000     2370.000000  -1.122960e+03
  max  9.999968e+07     9402.000000     8145.000000   6.737747e+07
```

Figure 2

Also, in the 'descriptive statistics' part we found the amount of unique values (figure 3) and compared them with other datasets. So, by comparing 'type' column in 'transaction' dataset with 'types' column in 'type' dataset (figure 4), we identified that the real number of unique types is 63. It means that 4 remained transaction types in transaction dataset are not listed in types dataset (figure 5). We assumed that the reason for this, probably, outdate types.

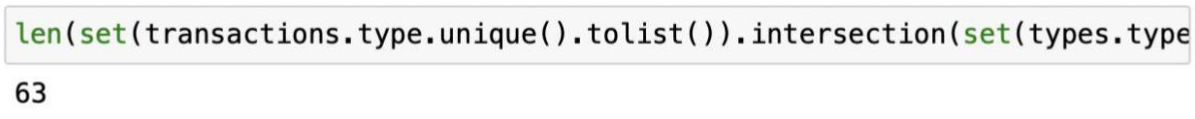
The same procedure was done with ‘codes’ column in ‘transaction’ dataset with ‘code’ column in ‘codes’ dataset, but here the number of unique values is equal (figure 6). So, there are no remained transaction codes. To complete this work intersection() method was used.



```
print(f"There are {transactions.client_id.unique().shape[0]} unique clie
```

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

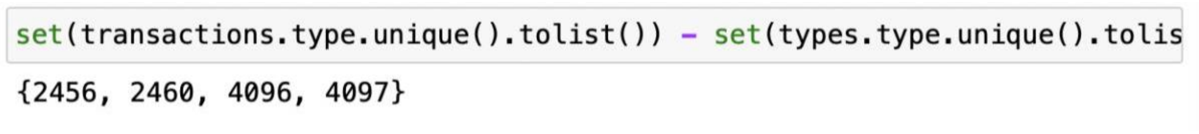
Figure 3



```
len(set(transactions.type.unique().tolist()).intersection(set(types.type
```

```
63
```

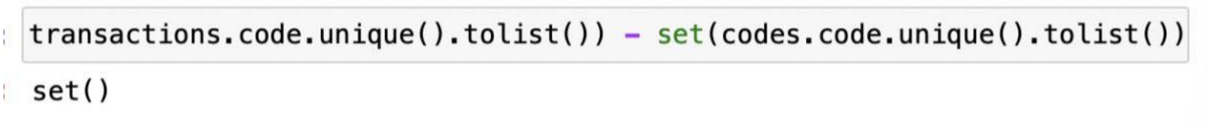
Figure 4



```
set(transactions.type.unique().tolist()) - set(types.type.unique().tolis
```

```
{2456, 2460, 4096, 4097}
```

Figure 5

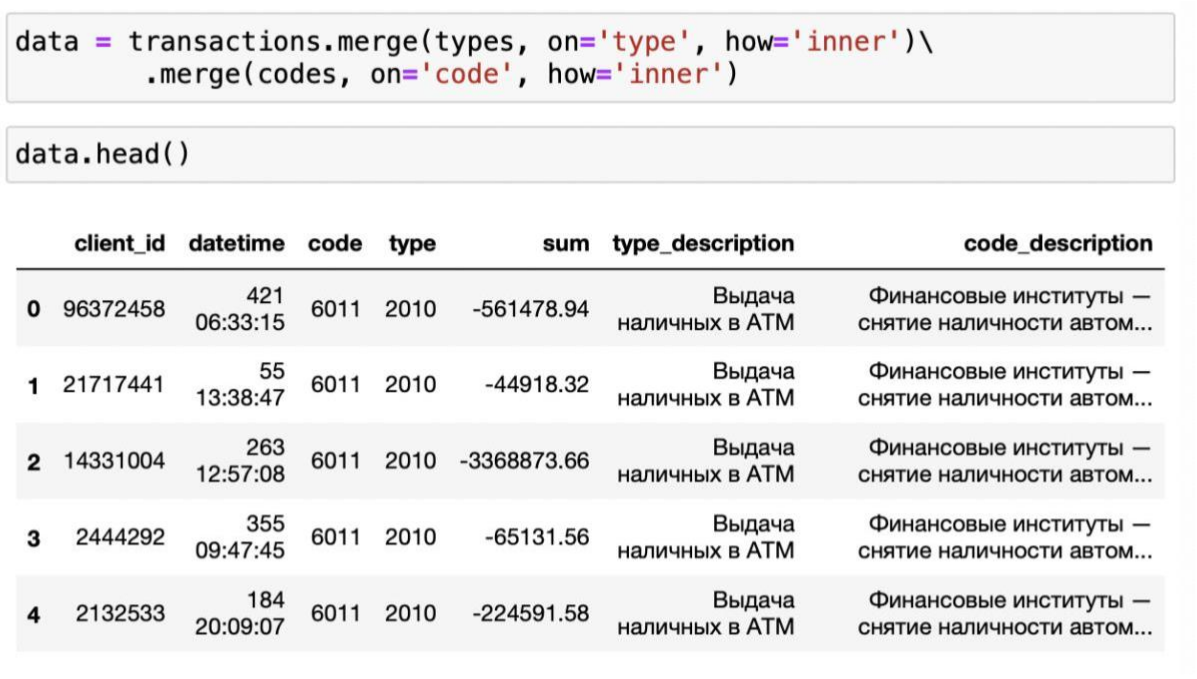


```
transactions.code.unique().tolist()) - set(codes.code.unique().tolist())
```

```
set()
```

Figure 6

The final step in description part is to merge our three datasets: transaction, codes, types using merge() method. In this method the first parameter shows the dataset name, the second ‘on=’ parameter shows a column which will be joined, and the third ‘how=’ parameter shows how it should be joined. In our case inner join is used (figure 7).



```
data = transactions.merge(types, on='type', how='inner')\
        .merge(codes, on='code', how='inner')
```

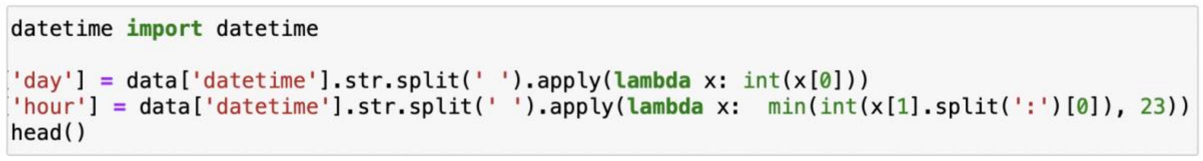
```
data.head()
```

	client_id	datetime	code	type	sum	type_description	code_description
0	96372458	421 06:33:15	6011	2010	-561478.94	Выдача наличных в АТМ	Финансовые институты — снятие наличности автом...
1	21717441	55 13:38:47	6011	2010	-44918.32	Выдача наличных в АТМ	Финансовые институты — снятие наличности автом...
2	14331004	263 12:57:08	6011	2010	-3368873.66	Выдача наличных в АТМ	Финансовые институты — снятие наличности автом...
3	2444292	355 09:47:45	6011	2010	-65131.56	Выдача наличных в АТМ	Финансовые институты — снятие наличности автом...
4	2132533	184 20:09:07	6011	2010	-224591.58	Выдача наличных в АТМ	Финансовые институты — снятие наличности автом...

Figure 7

Explanatory data analysis and few of feature engineering:

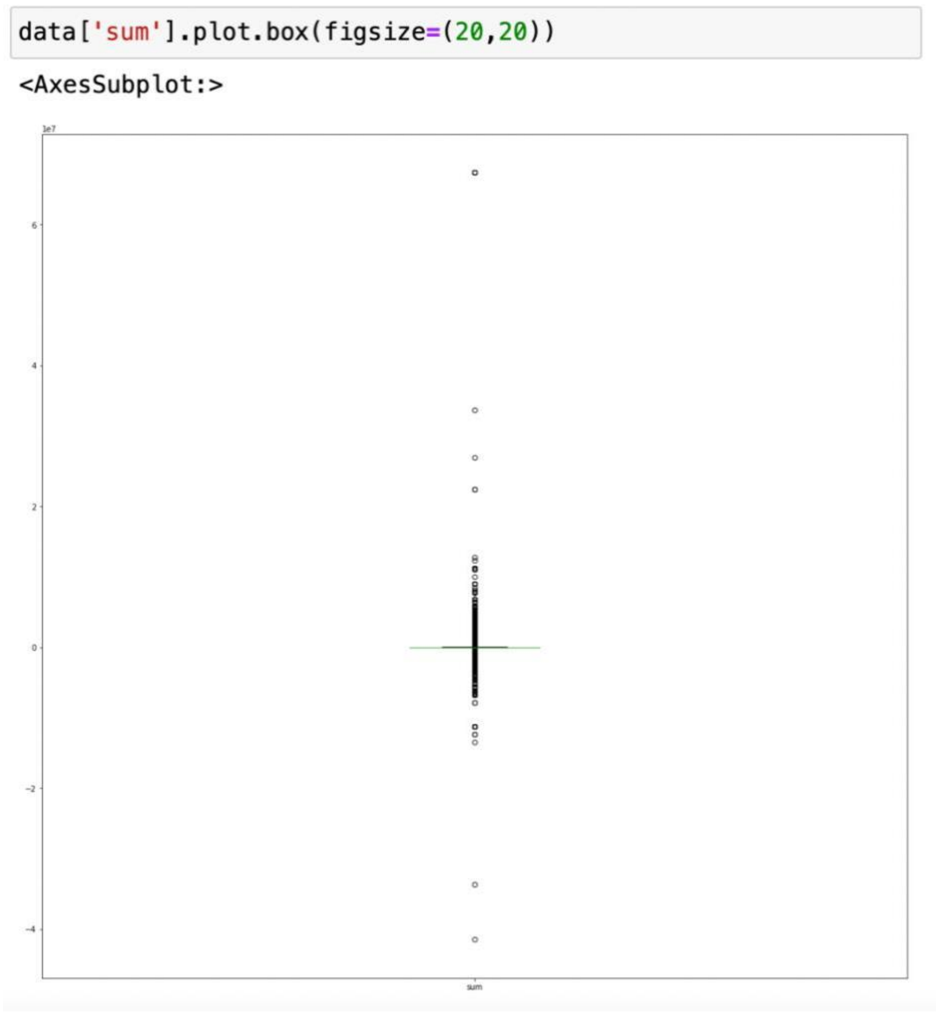
From the figure 7 can be seen that datetime column has a string value. For future engineering it is better to have integer values, so to split them to ‘day’, ‘hour’, ‘minute’, ‘second’ columns we used str.split() method and apply() method with intervals for the values (figure 8).



```
datetime import datetime

'day'] = data['datetime'].str.split(' ').apply(lambda x: int(x[0]))
'hour'] = data['datetime'].str.split(' ').apply(lambda x:  min(int(x[1].split(':')[0]), 23))
head()
```

Figure 8

We have illustrated the values in box plot and it shows a lot of outliers in our data (figure 9).



```
data['sum'].plot.box(figsize=(20,20))
```

```
<AxesSubplot:>
```

Figure 9

data = data.loc[(np.abs(data['sum'] - data['sum'].median()) < 2 * data['sum'].std())]

By using the formula above, we cleaned our data and the box plot now looks like in figure 10

```
data['sum'].plot.box(figsize=(20,20))
```

```
<AxesSubplot:>
```

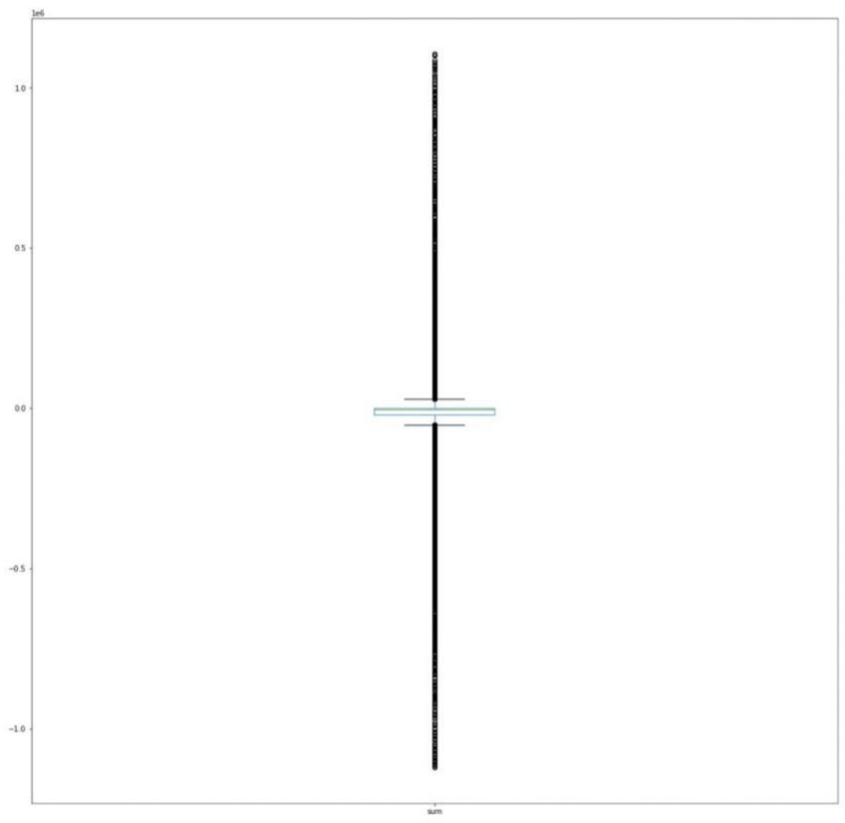
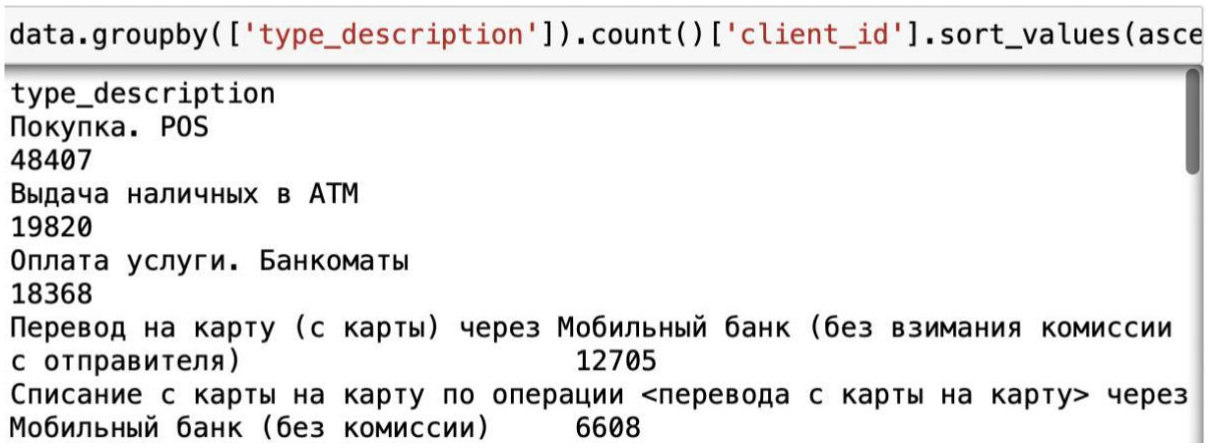


Figure 10

To see the distribution of operations types we counted grouped ‘type description’ column by ‘client id’ and sorted the data in ascending order (figure 11).



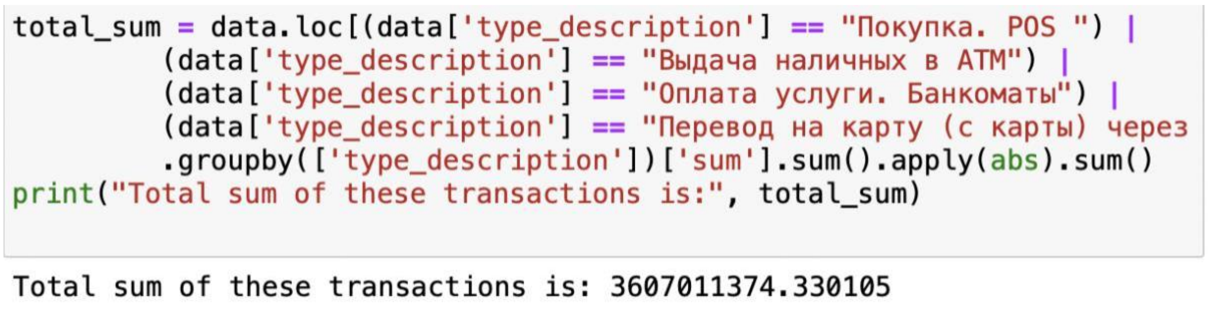
```
data.groupby(['type_description']).count()['client_id'].sort_values(asce
```

```
type_description
Покупка. POS
48407
Выдача наличных в ATM
19820
Оплата услуги. Банкоматы
18368
Перевод на карту (с карты) через Мобильный банк (без взимания комиссии
с отправителя)                         12705
Списание с карты на карту по операции <перевода с карты на карту> через
Мобильный банк (без комиссии)         6608
```

Figure 11

We can see disproportionality in the number of operations. Actually, the dominance of "purhcase via POS" can be easily explained: most stores have POS and people use it to purchase goods really often.

Let us see the cash amount of top-4 (have more than 10K operations) types of operations (figure 12)



```
total_sum = data.loc[(data['type_description'] == "Покупка. POS ") |
         (data['type_description'] == "Выдача наличных в АТМ") |
         (data['type_description'] == "Оплата услуги. Банкоматы") |
         (data['type_description'] == "Перевод на карту (с карты) через
         .groupby(['type_description'])['sum'].sum().apply(abs).sum()
print("Total sum of these transactions is:", total_sum)
```

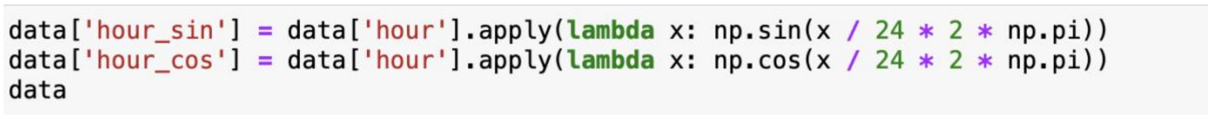
```
Total sum of these transactions is: 3607011374.330105
```

Figure 12

More feature engineering:

We had to make our time encodings cyclic using si and cos trigometric functions. Now instead of hours extending from 0 to 23, we have 6 new features "hour_sin", "hour_cos", which each extend from 0 to 1 and combine to have the nice cyclical characteristics (figure 13).

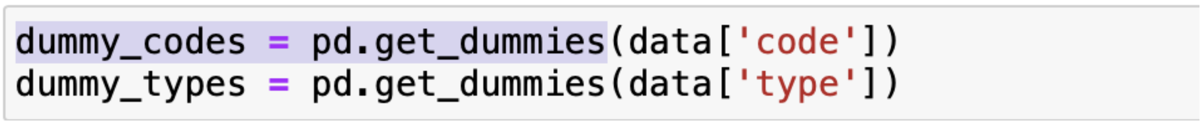
The claim is that using this transform will improve the predictive performance of our models.



```
data['hour_sin'] = data['hour'].apply(lambda x: np.sin(x / 24 * 2 * np.pi))
data['hour_cos'] = data['hour'].apply(lambda x: np.cos(x / 24 * 2 * np.pi))
data
```

Figure 13

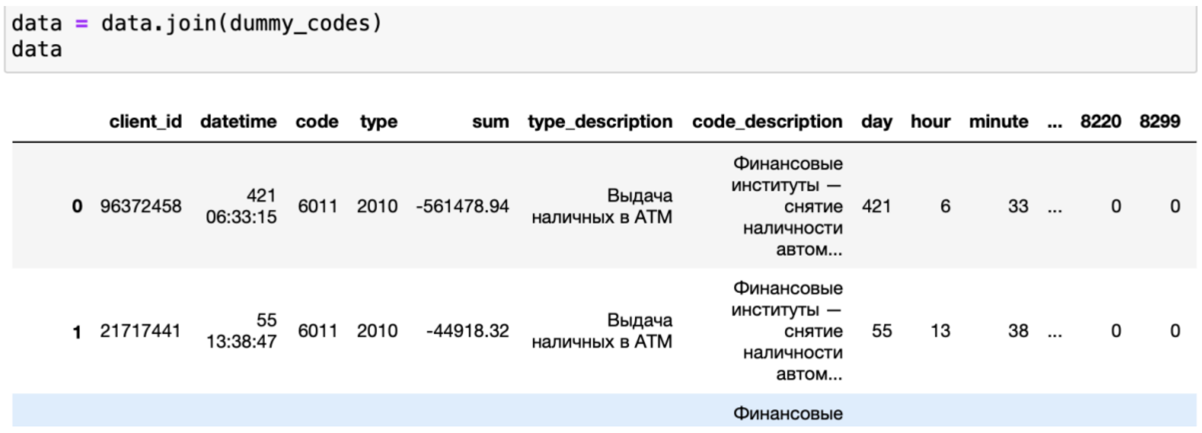
Next, to work with ‘codes’ and ‘types’ datasets we have to convert their categorical values (it is a variable that takes on one of a limited, and most commonly a fixed number of possible values) to dummy values. To convert them we used get_dummies method() (figure 14).



```
dummy_codes = pd.get_dummies(data['code'])
dummy_types = pd.get_dummies(data['type'])
```

Figure 14

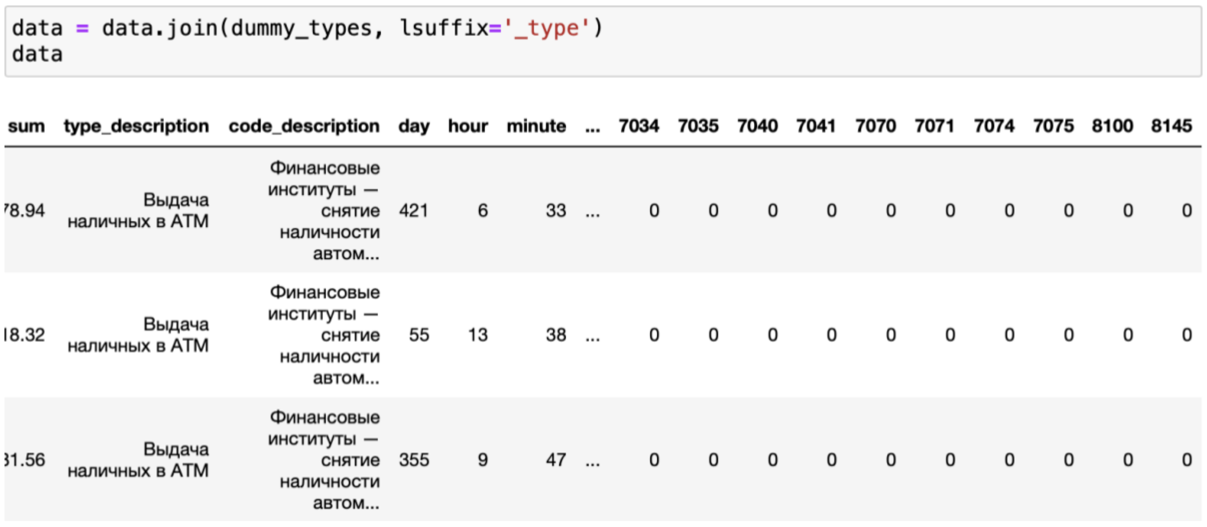
Now we have dummy columns of codes and types. They have a value of one (1) when a categorical event occurs and zero (0) when it doesn’t occur. Finally, we joined these dummy columns to our ‘data’ dataframe (figure 15).



```
data = data.join(dummy_codes)
data
```

	client_id	datetime	code	type	sum	type_description	code_description	day	hour	minute	...	8220	8299
0	96372458	421 06:33:15	6011	2010	-561478.94	Выдача наличных в ATM	Финансовые институты — снятие наличности автом...	421	6	33	...	0	0
1	21717441	55 13:38:47	6011	2010	-44918.32	Выдача наличных в ATM	Финансовые институты — снятие наличности автом...	55	13	38	...	0	0
							Финансовые						

(joining dummy columns of codes)



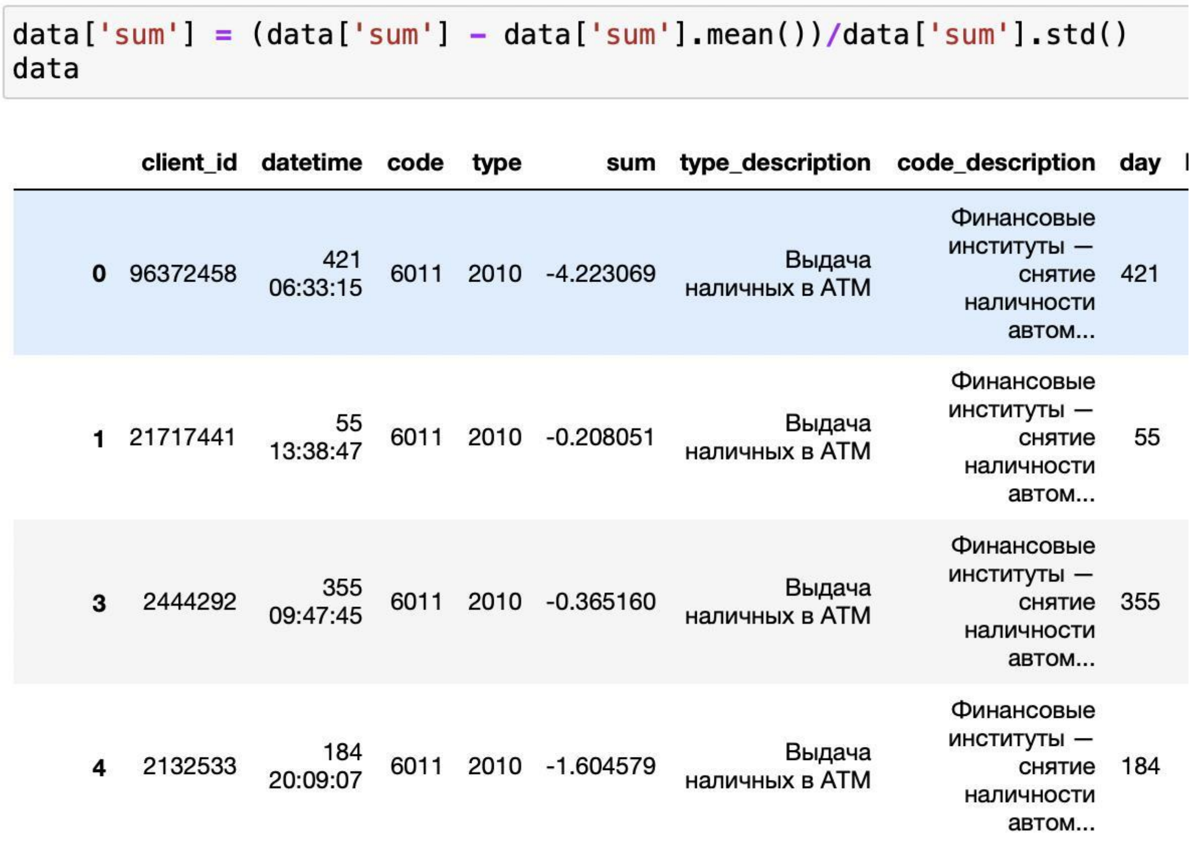
```
data = data.join(dummy_types, lsuffix='_type')
data
```

sum	type_description	code_description	day	hour	minute	...	7034	7035	7040	7041	7070	7071	7074	7075	8100	8145
78.94	Выдача наличных в ATM	Финансовые институты — снятие наличности автом...	421	6	33	...	0	0	0	0	0	0	0	0	0	0
18.32	Выдача наличных в ATM	Финансовые институты — снятие наличности автом...	55	13	38	...	0	0	0	0	0	0	0	0	0	0
31.56	Выдача наличных в ATM	Финансовые институты — снятие наличности автом...	355	9	47	...	0	0	0	0	0	0	0	0	0	0

(joining dummy columns of types with '_type' word)

Figure 15

And the last for future engineering is to normalize our 'sum' column (figure 16). We had to do this because of distorting difference in the range of values. The goal of normalization is to change the values of 'sum' column to a common scale. It will be helpful for further algorithms.



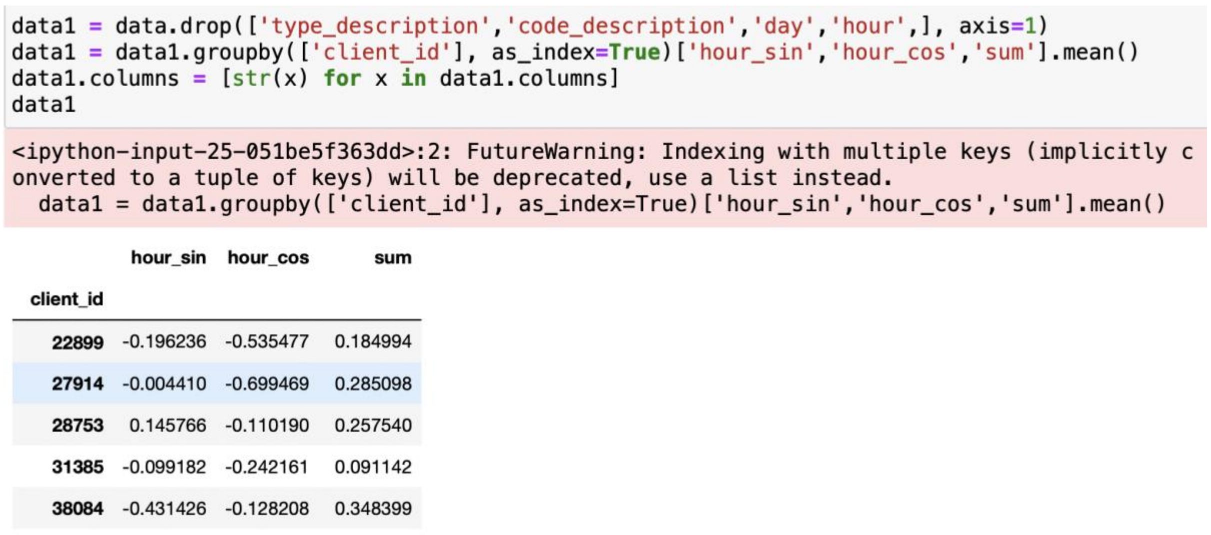
```
data['sum'] = (data['sum'] - data['sum'].mean())/data['sum'].std()
data
```

	client_id	datetime	code	type	sum	type_description	code_description	day
0	96372458	421 06:33:15	6011	2010	-4.223069	Выдача наличных в АТМ	Финансовые институты — снятие наличности автом...	421
1	21717441	55 13:38:47	6011	2010	-0.208051	Выдача наличных в АТМ	Финансовые институты — снятие наличности автом...	55
3	2444292	355 09:47:45	6011	2010	-0.365160	Выдача наличных в АТМ	Финансовые институты — снятие наличности автом...	355
4	2132533	184 20:09:07	6011	2010	-1.604579	Выдача наличных в АТМ	Финансовые институты — снятие наличности автом...	184

Figure 16

Supervised learning:

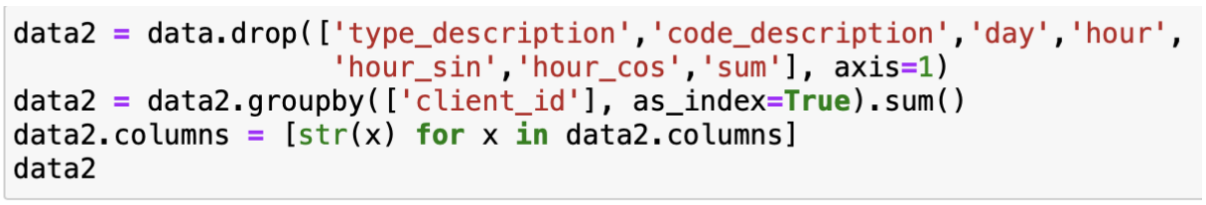
We will keep using our 'data' variable from the previous part, however, now we are going to merge it with train dataset. But we have to prepare data beforehand.



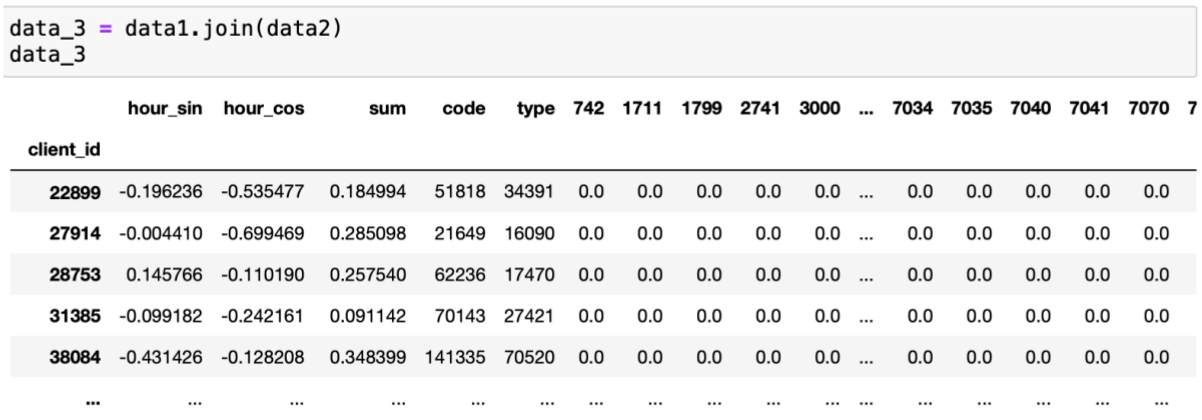
```
data1 = data.drop(['type_description','code_description','day','hour',], axis=1)
data1 = data1.groupby(['client_id'], as_index=True)['hour_sin','hour_cos','sum'].mean()
data1.columns = [str(x) for x in data1.columns]
data1
```

```
<ipython-input-25-051be5f363dd>:2: FutureWarning: Indexing with multiple keys (implicitly c
onverted to a tuple of keys) will be deprecated, use a list instead.
  data1 = data1.groupby(['client_id'], as_index=True)['hour_sin','hour_cos','sum'].mean()
```

	hour_sin	hour_cos	sum
client_id			
22899	-0.196236	-0.535477	0.184994
27914	-0.004410	-0.699469	0.285098
28753	0.145766	-0.110190	0.257540
31385	-0.099182	-0.242161	0.091142
38084	-0.431426	-0.128208	0.348399

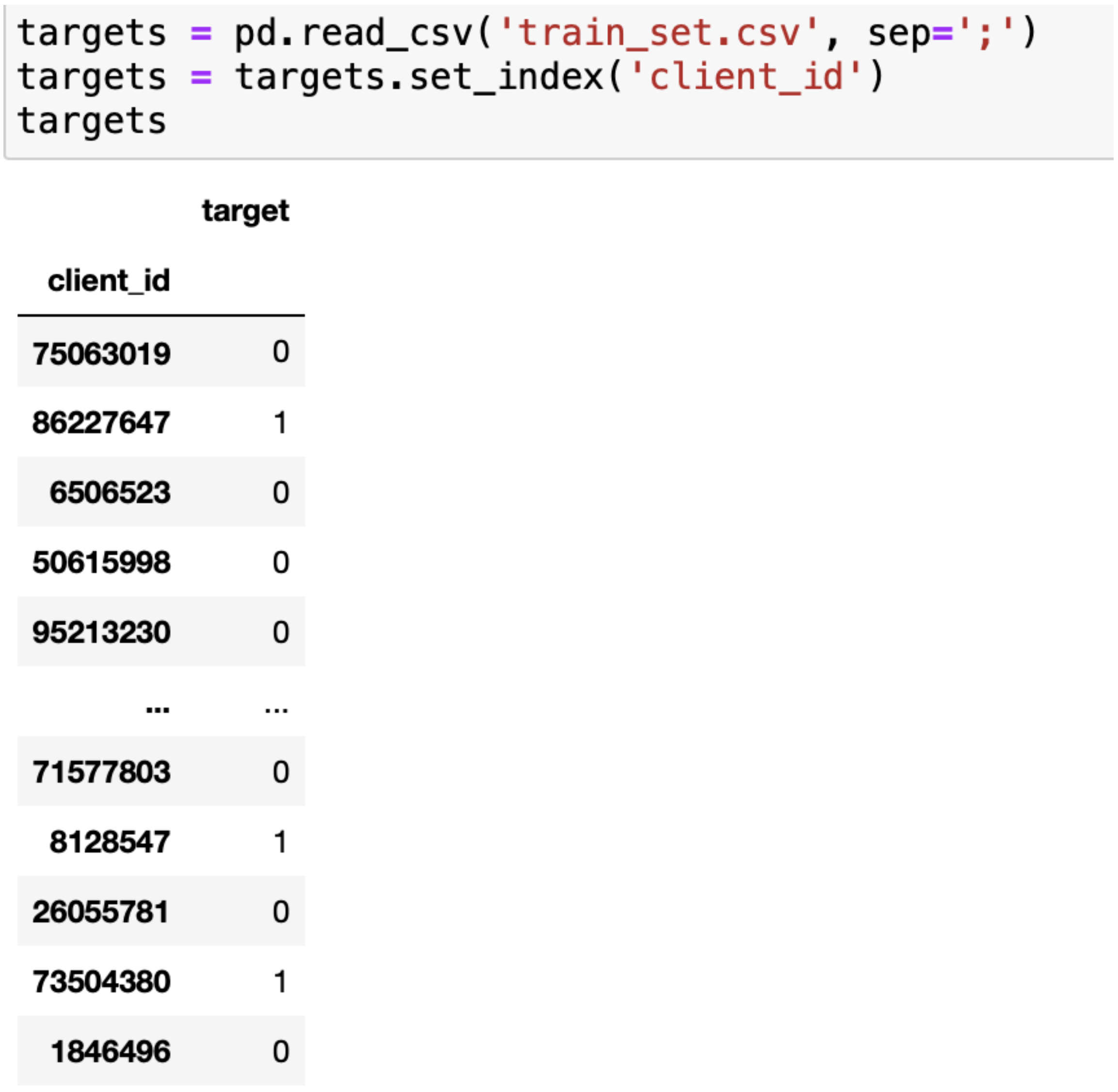


```
data2 = data.drop(['type_description','code_description','day','hour',
                   'hour_sin','hour_cos','sum'], axis=1)
data2 = data2.groupby(['client_id'], as_index=True).sum()
data2.columns = [str(x) for x in data2.columns]
data2
```



```
data_3 = data1.join(data2)
data_3
```

```
            hour_sin  hour_cos       sum    code   type  742  1711  1799  2741  3000  ...  7034  7035  7040  7041  7070  7
client_id
    22899  -0.196236 -0.535477  0.184994   51818  34391  0.0   0.0   0.0   0.0   0.0  ...   0.0   0.0   0.0   0.0   0.0
    27914  -0.004410 -0.699469  0.285098   21649  16090  0.0   0.0   0.0   0.0   0.0  ...   0.0   0.0   0.0   0.0   0.0
    28753   0.145766 -0.110190  0.257540   62236  17470  0.0   0.0   0.0   0.0   0.0  ...   0.0   0.0   0.0   0.0   0.0
    31385  -0.099182 -0.242161  0.091142   70143  27421  0.0   0.0   0.0   0.0   0.0  ...   0.0   0.0   0.0   0.0   0.0
    38084  -0.431426 -0.128208  0.348399  141335  70520  0.0   0.0   0.0   0.0   0.0  ...   0.0   0.0   0.0   0.0   0.0
      ...        ...       ...       ...     ...    ...  ...   ...   ...   ...   ...  ...   ...   ...   ...   ...   ...
```



```
targets = pd.read_csv('train_set.csv', sep=';')
targets = targets.set_index('client_id')
targets
```

```
           target
client_id
 75063019       0
 86227647       1
  6506523       0
 50615998       0
 95213230       0
      ...     ...
 71577803       0
  8128547       1
 26055781       0
 73504380       1
  1846496       0
```

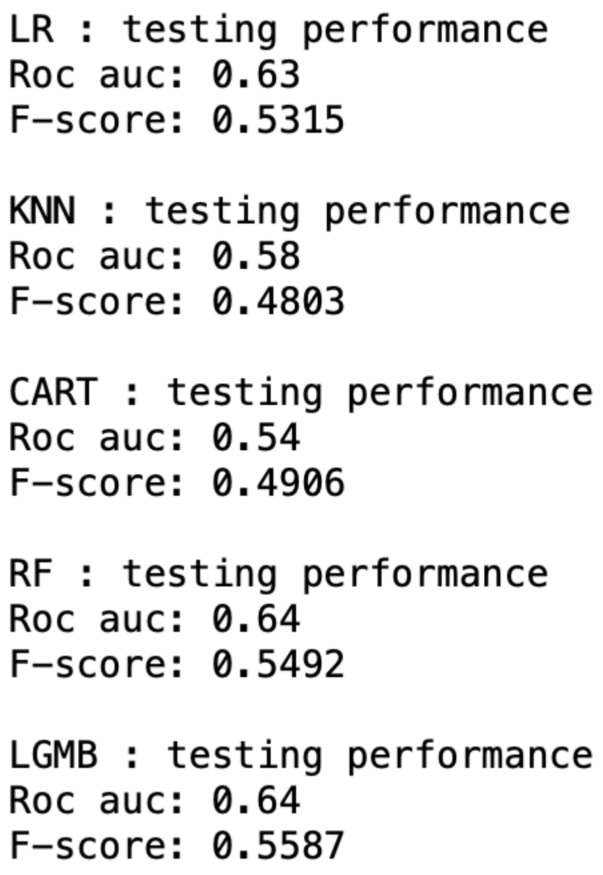
```
data_full = data_3.join(targets)
data_full
```

```
22899  -0.196236  -0.535477  0.184994   51818  34391  0.0  0.0  0.0  0.0  0.0  ...  0.0  0.0  0.0  0.0  1.0
27914  -0.004410  -0.699469  0.285098   21649  16090  0.0  0.0  0.0  0.0  0.0  ...  0.0  0.0  0.0  0.0  0.0
28753   0.145766  -0.110190  0.257540   62236  17470  0.0  0.0  0.0  0.0  0.0  ...  0.0  0.0  0.0  0.0  0.0
31385  -0.099182  -0.242161  0.091142   70143  27421  0.0  0.0  0.0  0.0  0.0  ...  0.0  0.0  0.0  0.0  0.0
38084  -0.431426  -0.128208  0.348399  141335  70520  0.0  0.0  0.0  0.0  0.0  ...  0.0  0.0  0.0  0.0  0.0
  ...        ...        ...       ...     ...    ...  ...  ...  ...  ...  ...  ...  ...  ...  ...  ...  ...
```

```python
#for splitting data arrays into two subsets: for training data and for testing data.
from sklearn.model_selection import train_test_split
X = data_full.drop('target', axis=1)
y = data_full['target']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33, random_state=42)
```

```python
from sklearn.metrics import roc_auc_score, f1_score
from sklearn.preprocessing import PolynomialFeatures, StandardScaler, OneHotEncoder
from sklearn.model_selection import GridSearchCV, KFold, cross_val_score, train_test_split
from sklearn.pipeline import Pipeline
from joblib import load, dump

# import models
!pip install lightgbm
from lightgbm import LGBMClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.neighbors import KNeighborsClassifier
```

```python
models = [('LR',    LogisticRegression()),
          ('KNN',   KNeighborsClassifier()),
          ('CART',  DecisionTreeClassifier()),
          ('RF',  RandomForestClassifier()),
          ('LGMB',  LGBMClassifier())]

pipelines = {}

for name, model in models:
    pipelines[name] = Pipeline([('scale', StandardScaler()),
                                #('polyf', PolynomialFeatures(include_bias=False)),
                                ('model', model)])
    pipe = pipelines[name]
    pipe.fit(X_train, y_train)
    y_hat = pipe.predict(X_test)
    roc_auc = roc_auc_score(y_test, y_hat)
    f = f1_score(y_test, y_hat)
    print(name, ': testing performance')
    print('Roc auc: {:.2f}'.format(roc_auc))
    print('F-score: {:.4f}\n'.format(f))
```

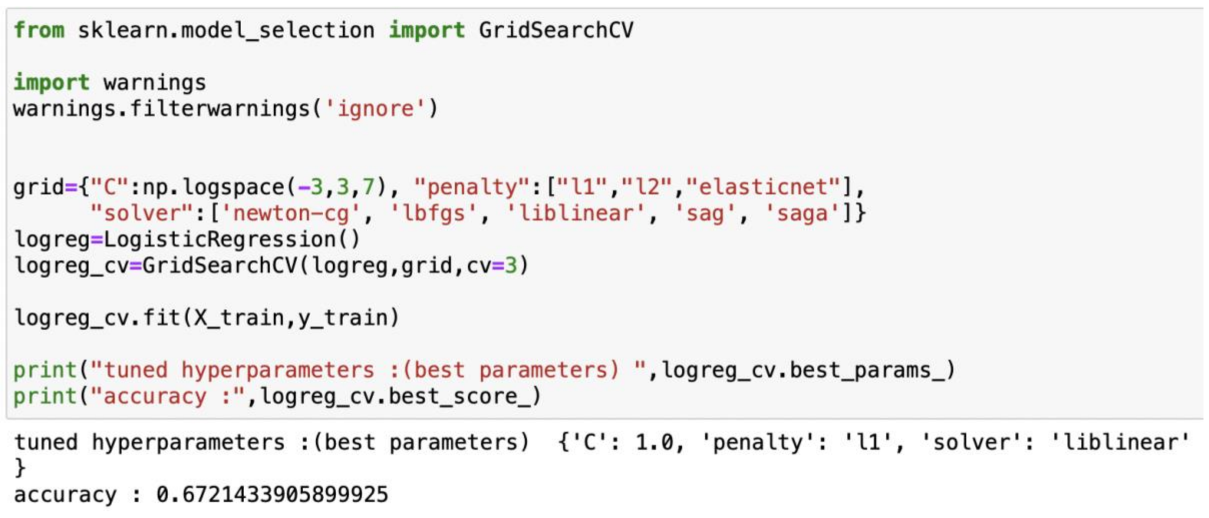


```
LR : testing performance
Roc auc: 0.63
F-score: 0.5315

KNN : testing performance
Roc auc: 0.58
F-score: 0.4803

CART : testing performance
Roc auc: 0.54
F-score: 0.4906

RF : testing performance
Roc auc: 0.64
F-score: 0.5492

LGMB : testing performance
Roc auc: 0.64
F-score: 0.5587
```

We see that both modes: Logistic regression and LGMBoost have shown pretty close results.
Since LGMBoost is out of our classes scope, we will continut with LogisticRegression

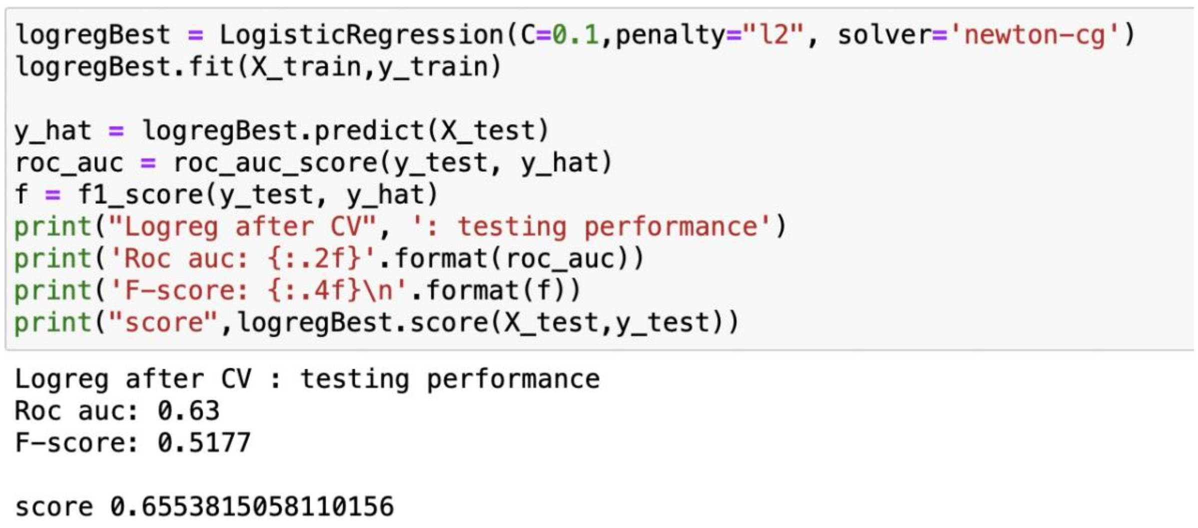
We will do grid search with crossvalidation (on 3 folds) to identify

best parameters of Ligistic Regression for our task



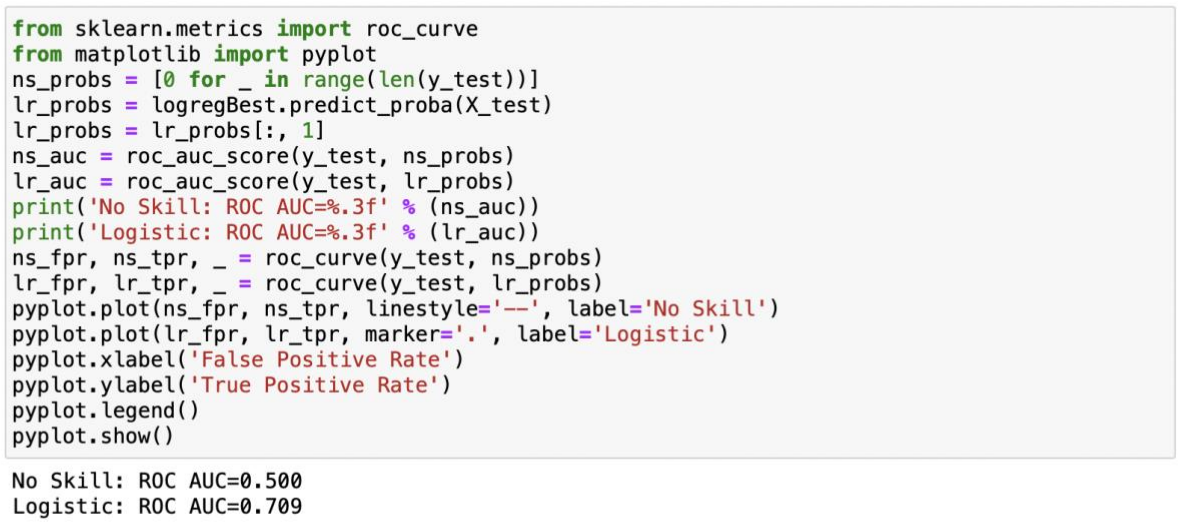
```
from sklearn.model_selection import GridSearchCV

import warnings
warnings.filterwarnings('ignore')


grid={"C":np.logspace(-3,3,7), "penalty":["l1","l2","elasticnet"],
      "solver":['newton-cg', 'lbfgs', 'liblinear', 'sag', 'saga']}
logreg=LogisticRegression()
logreg_cv=GridSearchCV(logreg,grid,cv=3)

logreg_cv.fit(X_train,y_train)

print("tuned hyperparameters :(best parameters) ",logreg_cv.best_params_)
print("accuracy :",logreg_cv.best_score_)
```

```
tuned hyperparameters :(best parameters)  {'C': 1.0, 'penalty': 'l1', 'solver': 'liblinear'
}
accuracy : 0.6721433905899925
```



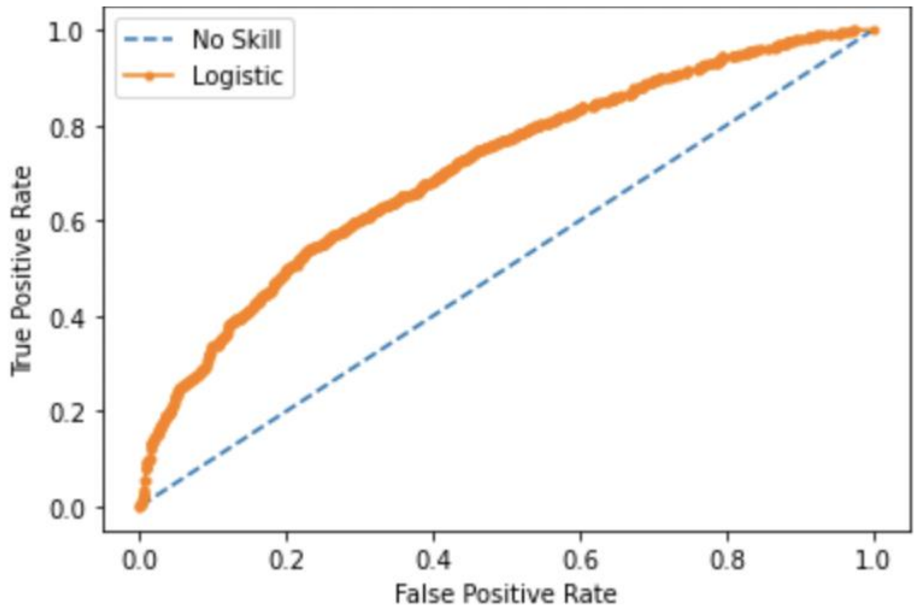
```
logregBest = LogisticRegression(C=0.1,penalty="l2", solver='newton-cg')
logregBest.fit(X_train,y_train)

y_hat = logregBest.predict(X_test)
roc_auc = roc_auc_score(y_test, y_hat)
f = f1_score(y_test, y_hat)
print("Logreg after CV", ': testing performance')
print('Roc auc: {:.2f}'.format(roc_auc))
print('F-score: {:.4f}\n'.format(f))
print("score",logregBest.score(X_test,y_test))
```

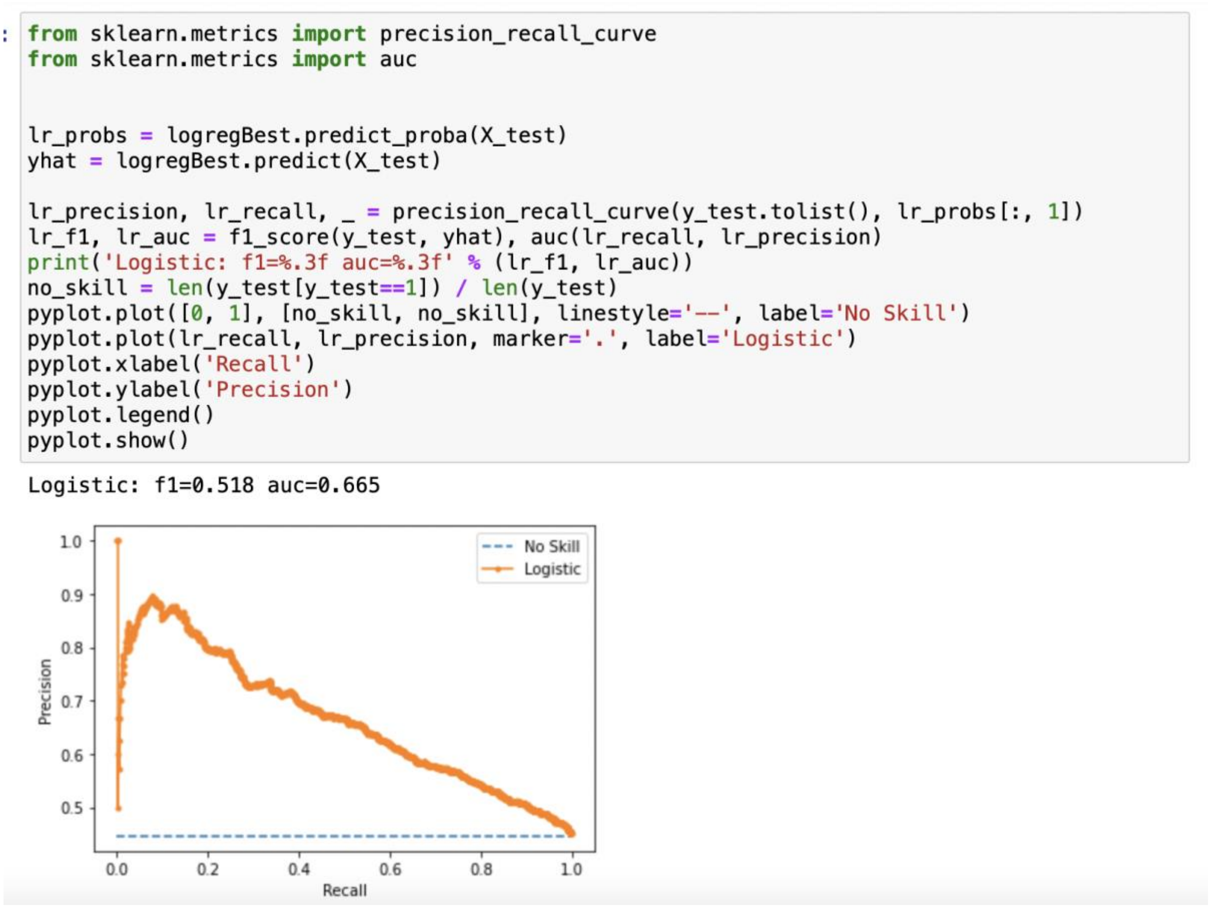
```
Logreg after CV : testing performance
Roc auc: 0.63
F-score: 0.5177

score 0.6553815058110156
```



```
from sklearn.metrics import roc_curve
from matplotlib import pyplot
ns_probs = [0 for _ in range(len(y_test))]
lr_probs = logregBest.predict_proba(X_test)
lr_probs = lr_probs[:, 1]
ns_auc = roc_auc_score(y_test, ns_probs)
lr_auc = roc_auc_score(y_test, lr_probs)
print('No Skill: ROC AUC=%.3f' % (ns_auc))
print('Logistic: ROC AUC=%.3f' % (lr_auc))
ns_fpr, ns_tpr, _ = roc_curve(y_test, ns_probs)
lr_fpr, lr_tpr, _ = roc_curve(y_test, lr_probs)
pyplot.plot(ns_fpr, ns_tpr, linestyle='--', label='No Skill')
pyplot.plot(lr_fpr, lr_tpr, marker='.', label='Logistic')
pyplot.xlabel('False Positive Rate')
pyplot.ylabel('True Positive Rate')
pyplot.legend()
pyplot.show()
```

```
No Skill: ROC AUC=0.500
Logistic: ROC AUC=0.709
```





```
from sklearn.metrics import precision_recall_curve
from sklearn.metrics import auc


lr_probs = logregBest.predict_proba(X_test)
yhat = logregBest.predict(X_test)

lr_precision, lr_recall, _ = precision_recall_curve(y_test.tolist(), lr_probs[:, 1])
lr_f1, lr_auc = f1_score(y_test, yhat), auc(lr_recall, lr_precision)
print('Logistic: f1=%.3f auc=%.3f' % (lr_f1, lr_auc))
no_skill = len(y_test[y_test==1]) / len(y_test)
pyplot.plot([0, 1], [no_skill, no_skill], linestyle='--', label='No Skill')
pyplot.plot(lr_recall, lr_precision, marker='.', label='Logistic')
pyplot.xlabel('Recall')
pyplot.ylabel('Precision')
pyplot.legend()
pyplot.show()
```

```
Logistic: f1=0.518 auc=0.665
```

Analysis and conclusion

We have compared several models, as a result, boosting algorithms (LGMBoost) have shown the best roc-auc score. However, we have tested logistic regression, applied Crossvalidation and grid search to this model and and improved roc-auc score from 0.64 to roughtly (with the best threshold) 0.7. F-score did not change much.

Since we have no imbalance in data (0s and 1s are in the roughtly same proportions), evaluating the model using Roc-Auc curve is more reasonable.

To conclude, our model is good at recognizing gender using transactions history.