# BANKING SCORING MODEL

members: Aldiyar Esenturov
Adil Tyulyutaev
Zhumash Zhandos

01


# Credit scoring system in banks

Credit scoring is a statistical analysis performed by lenders and financial institutions to determine a person's or a small, owner-operated business' creditworthiness. Credit scoring is used by lenders to help decide whether to extend or deny credit

# GOALS

Any scoring system is a risk and you cannot always be sure whether you should approve the credit to some person. Our goals is to minimize this risk so that banks can trust the scoring model.

# About our model and the calculations that it does

# Input parametrs

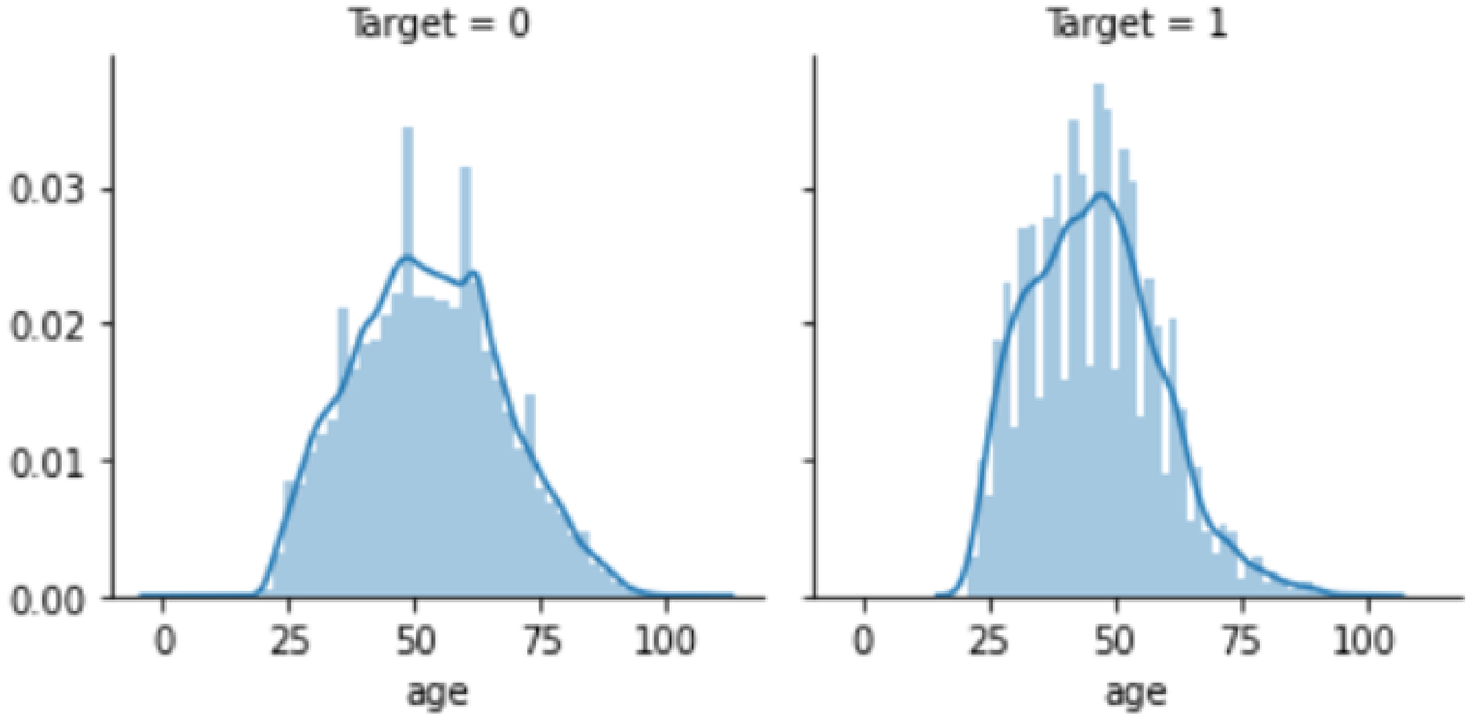
Any scoring model requires input parameters to analyze a person's creditworthiness

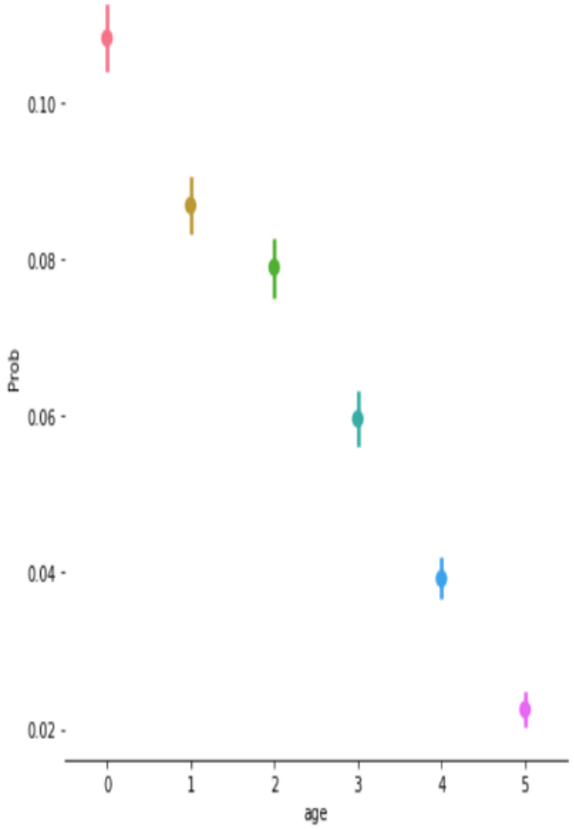
# Parameters in our model

totalAmountOverCr
editLimit
age
Count3059
DebtRatio
Salary
NumberOfOpenCred
its
Count90
NumberOfArrests
Count6089
Dependents
Target – binary value which says will debtor have due more than 90 or not.

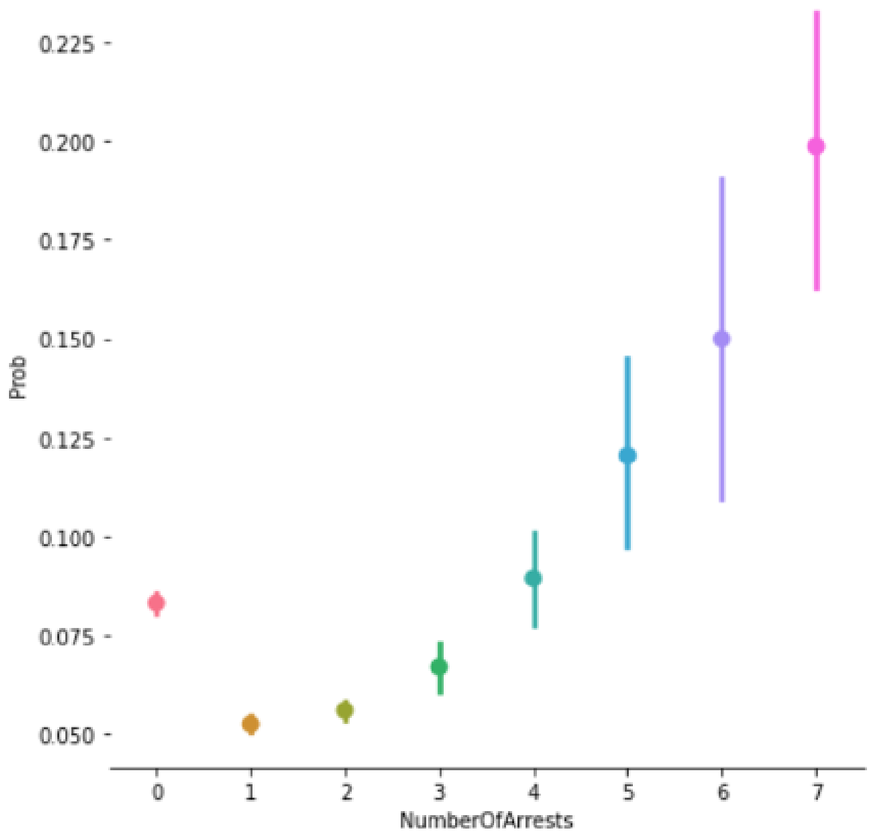
# Here is some graphics

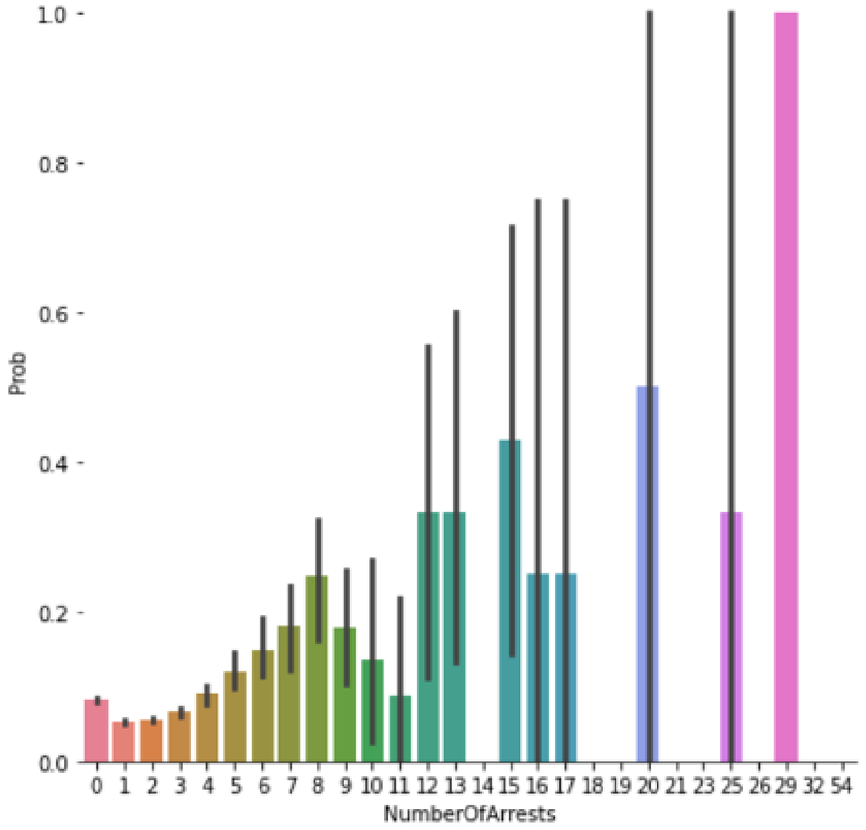
# Age in assessing a person's creditworthiness



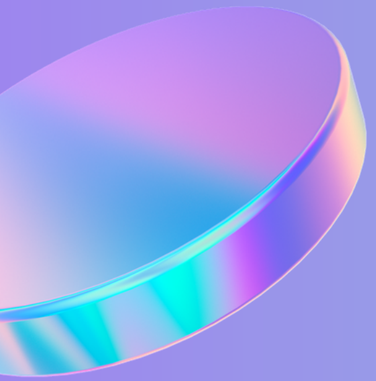
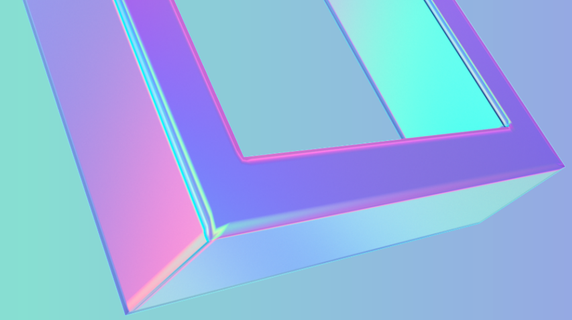
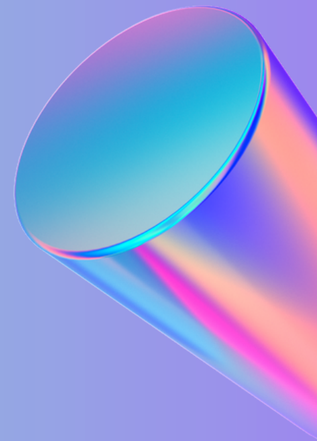
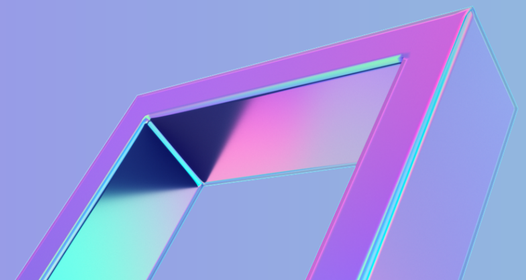



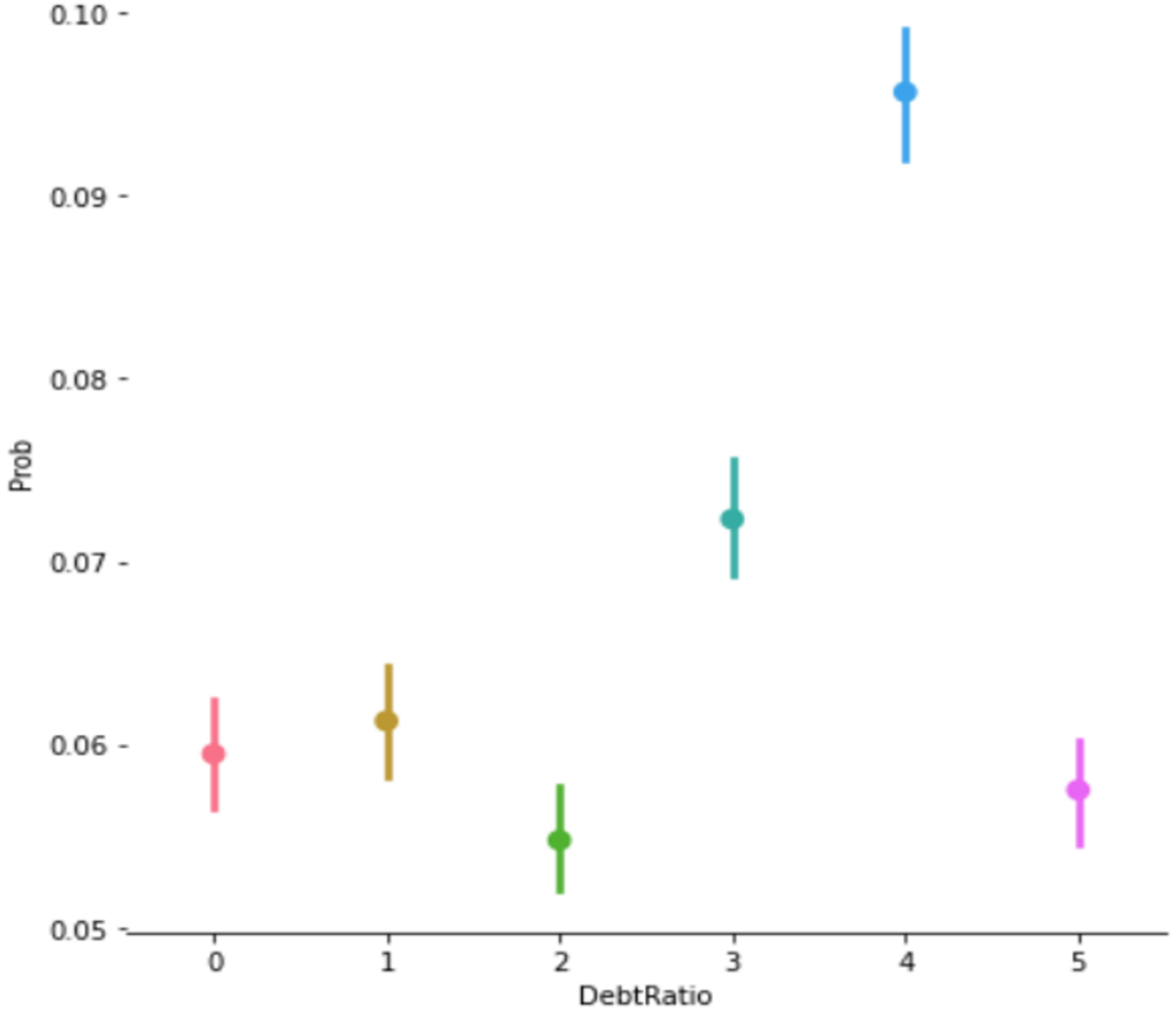
# Number of arrests in assessing a person's creditworthiness

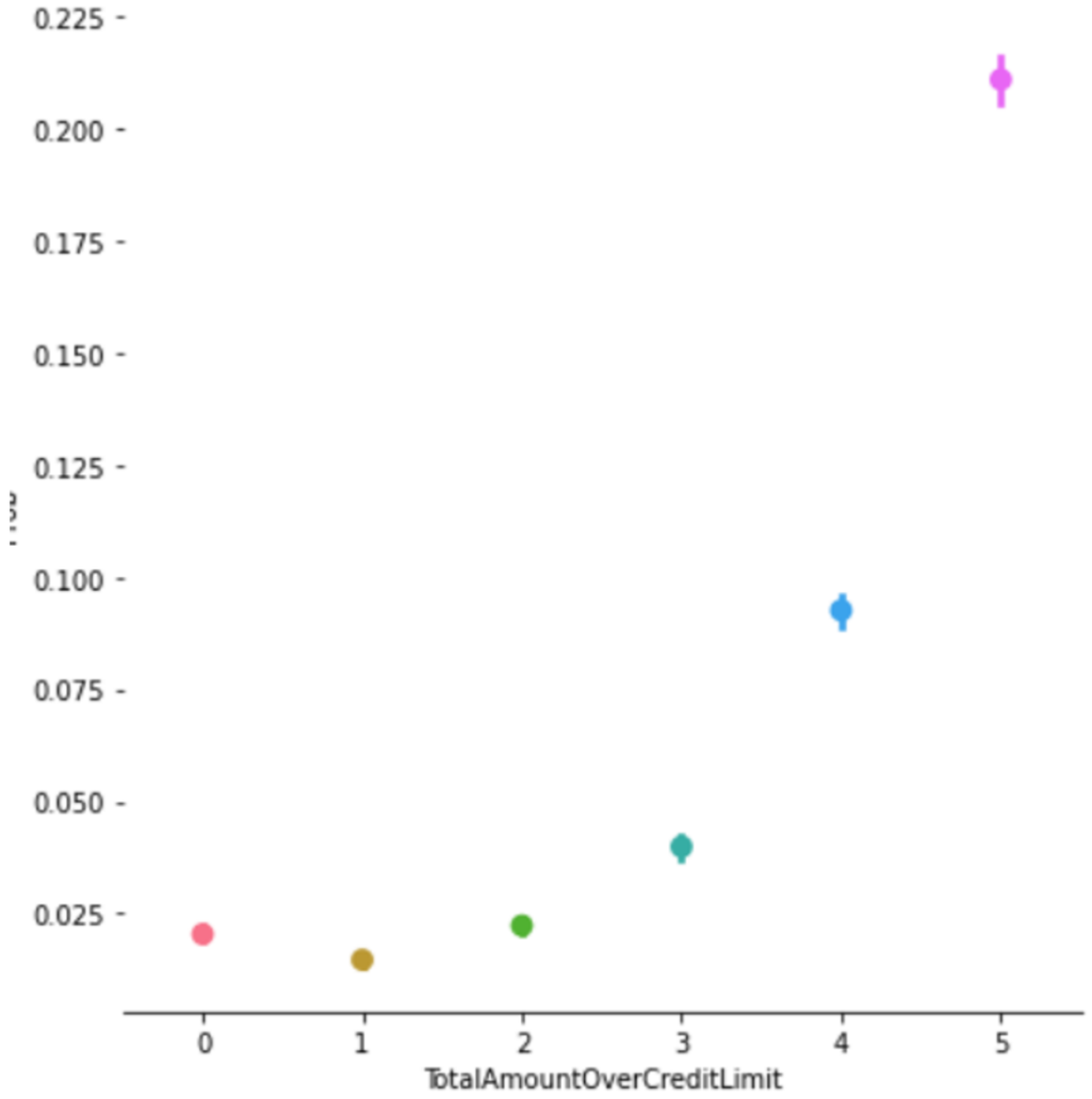


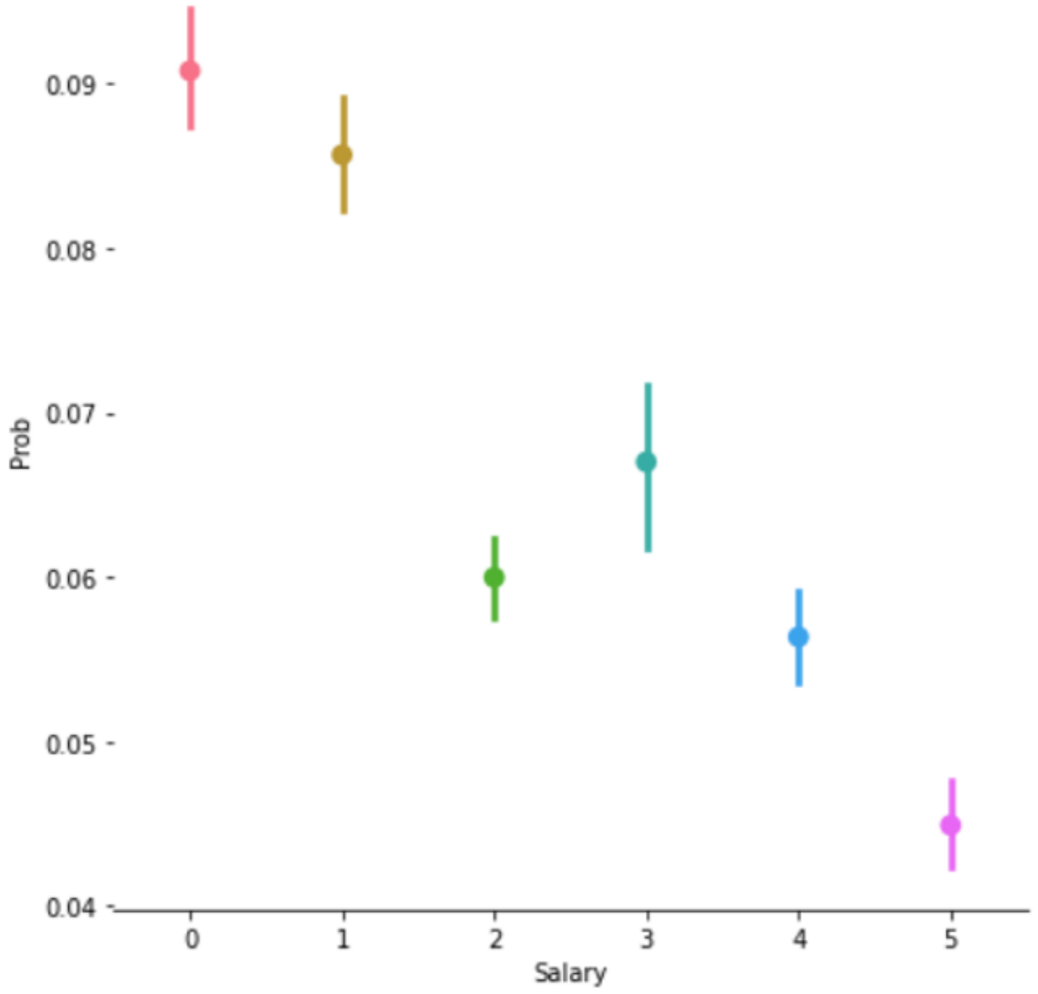



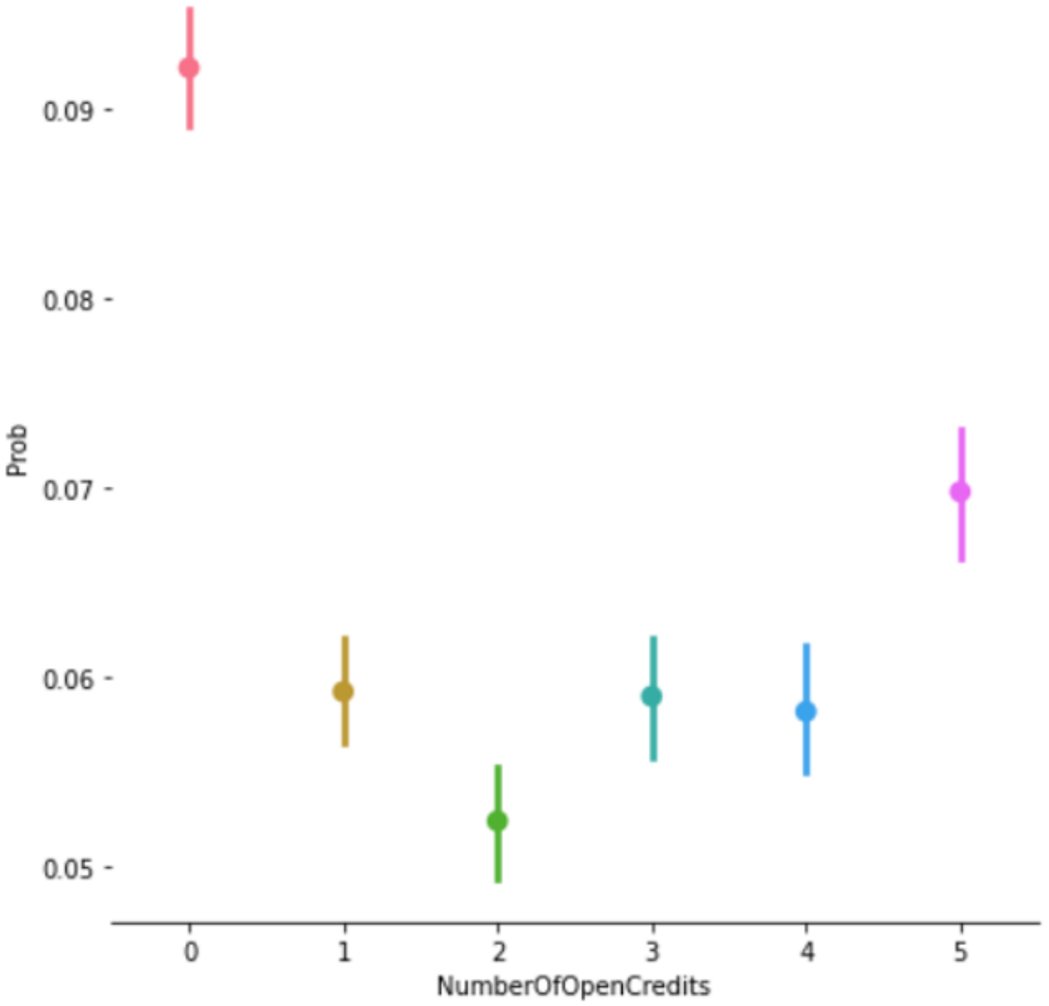
# And other dependencies

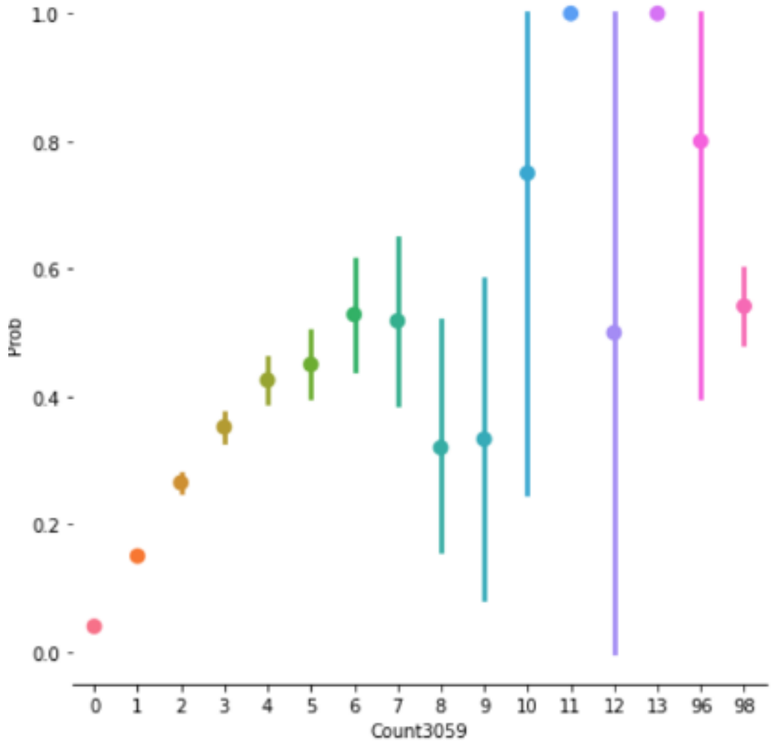





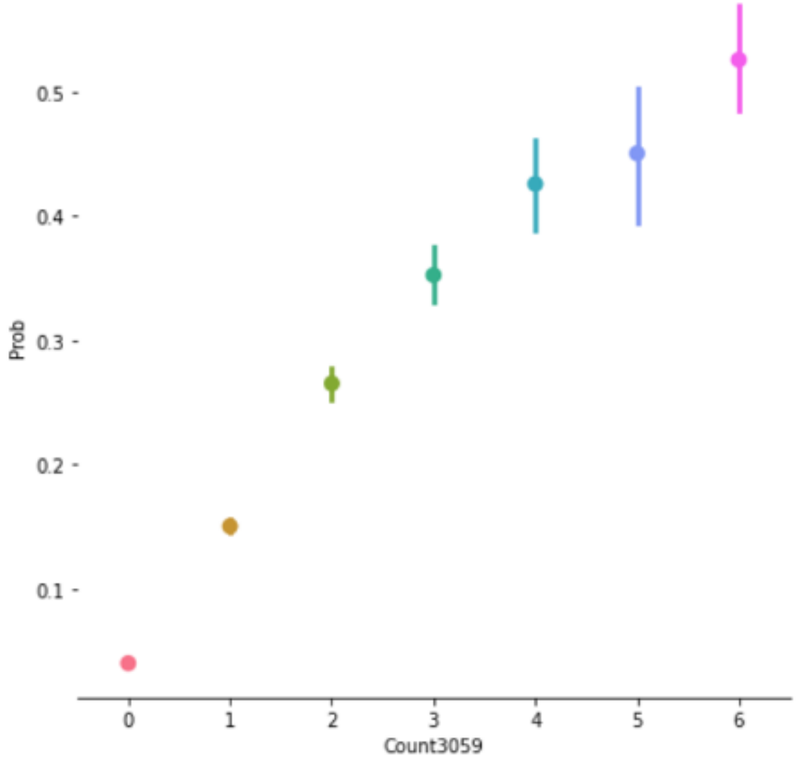

0.10
0.09
0.08
0.07
0.06
0.05
Prob
0
1
2
3
4
5
DebtRatio

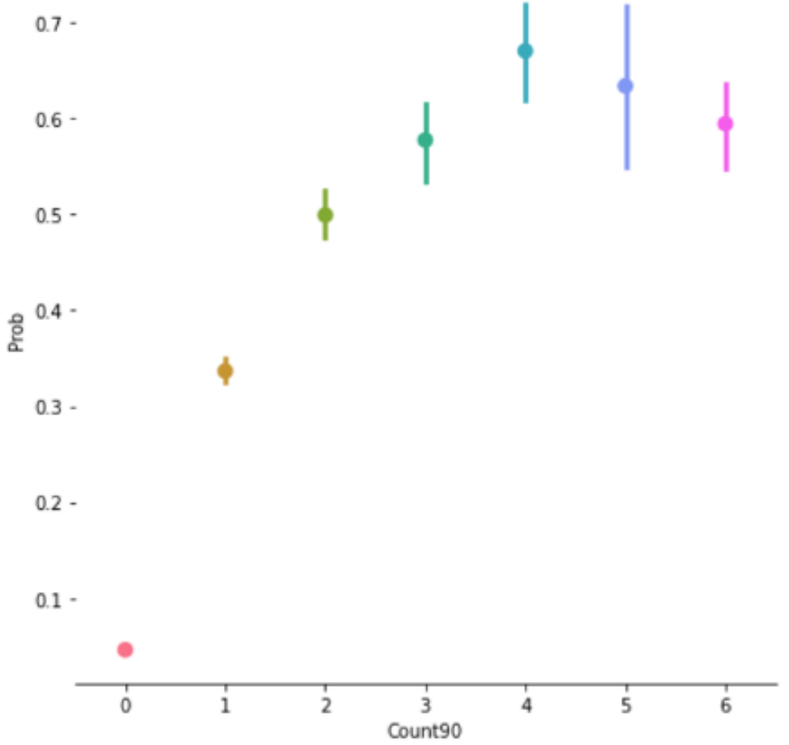

0.225
0.200
0.175
0.150
0.125
0.100
0.075
0.050
0.025
0
1
2
3
4
5
TotalAmountOverCreditLimit


0.09
0.08
0.07
0.06
0.05
0.04
Prob
0
1
2
3
4
5
Salary


0.09
0.08
0.07
0.06
0.05
Prob
0
1
2
3
4
5
NumberOfOpenCredits


Prob
1.0
0.8
0.6
0.4
0.2
0.0
0 1 2 3 4 5 6 7 8 9 10 11 12 13 96 98
Count3059


Prob
0.5
0.4
0.3
0.2
0.1
0 1 2 3 4 5 6
Count3059


Prob
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0 1 2 3 4 5 6
Count90

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 150000 entries, 0 to 149999
Data columns (total 11 columns):
 #   Column                          Non-Null Count   Dtype
---  ------                          --------------   -----
 0   TotalAmountOverCreditLimit  150000 non-null  int8
 1   age                         150000 non-null  int8
 2   Count3059                   150000 non-null  int64
 3   DebtRatio                   150000 non-null  int8
 4   Salary                      150000 non-null  int8
 5   NumberOfOpenCredits         150000 non-null  int8
 6   Count90                     150000 non-null  int64
 7   NumberOfArrests             150000 non-null  int64
 8   Count6089                   150000 non-null  int64
 9   Dependents                  150000 non-null  float64
 10  Target                      150000 non-null  int64
dtypes: float64(1), int64(5), int8(5)
memory usage: 7.6 MB
```

| | TotalAmountOverCreditLimit | age | Count3059 | DebtRatio | Salary | NumberOfOpenCredits | Count90 | NumberOfArrests | Count6089 | Dependents | Target |
|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 4 | 1 | 2 | 4 | 5 | 4 | 0 | 6 | 0 | 2.0 | 1 |
| **1** | 5 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 1.0 | 0 |
| **2** | 4 | 1 | 1 | 0 | 1 | 0 | 1 | 0 | 0 | 0.0 | 0 |
| **3** | 3 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0.0 | 0 |
| **4** | 5 | 2 | 1 | 0 | 5 | 2 | 0 | 1 | 0 | 0.0 | 0 |
| **...** | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... |
| **149995** | 1 | 5 | 0 | 1 | 0 | 0 | 0 | 1 | 0 | 0.0 | 0 |
| **149996** | 3 | 1 | 0 | 4 | 3 | 0 | 0 | 1 | 0 | 2.0 | 0 |
| **149997** | 3 | 3 | 0 | 5 | 2 | 5 | 0 | 1 | 0 | 0.0 | 0 |
| **149998** | 0 | 0 | 0 | 0 | 3 | 0 | 0 | 0 | 0 | 0.0 | 0 |
| **149999** | 5 | 4 | 0 | 2 | 4 | 2 | 0 | 2 | 0 | 0.0 | 0 |

# We added dummies values for each column

```
Y = train['Target']
X = train.drop('Target',axis=1)

X = pd.get_dummies(X, columns = ["TotalAmountOverCreditLimit"], prefix="TotalAmountOverCreditLimit")
X = pd.get_dummies(X, columns = ["age"], prefix="age")
X = pd.get_dummies(X, columns = ["Count3059"], prefix="Count3059")
X = pd.get_dummies(X, columns = ["DebtRatio"], prefix="DebtRatio")
X = pd.get_dummies(X, columns = ["Salary"], prefix="Salary")
X = pd.get_dummies(X, columns = ["NumberOfOpenCredits"], prefix="NumberOfOpenCredits")
X = pd.get_dummies(X, columns = ["Count90"], prefix="Count90")
X = pd.get_dummies(X, columns = ["NumberOfArrests"], prefix="NumberOfArrests")
X = pd.get_dummies(X, columns = ["Count6089"], prefix="Count6089")
X = pd.get_dummies(X, columns = ["Dependents"], prefix="Dependents")

X
```

| | TotalAmountOverCreditLimit_0 | TotalAmountOverCreditLimit_1 | TotalAmountOverCreditLimit_2 | TotalAmountOverCreditLimit_3 | TotalAmountOverCreditLimit_4 | TotalAmountOverCreditLimit_5 | age_0 | age_1 | age_2 | age_3 | ... | Count6089 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | ... | |
| 1 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 0 | 0 | ... | |
| 2 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | ... | |
| 3 | 0 | 0 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | ... | |
| 4 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 1 | 0 | ... | |
| ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... |
| 149995 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | ... | |
| 149996 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | ... | |
| 149997 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 1 | ... | |
| 149998 | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | ... | |
| 149999 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | ... | |

150000 rows × 65 columns

# We added dummies values for each column

```
Y = train['Target']
X = train.drop('Target',axis=1)

X = pd.get_dummies(X, columns = ["TotalAmountOverCreditLimit"], prefix="TotalAmountOverCreditLimit")
X = pd.get_dummies(X, columns = ["age"], prefix="age")
X = pd.get_dummies(X, columns = ["Count3059"], prefix="Count3059")
X = pd.get_dummies(X, columns = ["DebtRatio"], prefix="DebtRatio")
X = pd.get_dummies(X, columns = ["Salary"], prefix="Salary")
X = pd.get_dummies(X, columns = ["NumberOfOpenCredits"], prefix="NumberOfOpenCredits")
X = pd.get_dummies(X, columns = ["Count90"], prefix="Count90")
X = pd.get_dummies(X, columns = ["NumberOfArrests"], prefix="NumberOfArrests")
X = pd.get_dummies(X, columns = ["Count6089"], prefix="Count6089")
X = pd.get_dummies(X, columns = ["Dependents"], prefix="Dependents")

X
```

| | TotalAmountOverCreditLimit_0 | TotalAmountOverCreditLimit_1 | TotalAmountOverCreditLimit_2 | TotalAmountOverCreditLimit_3 | TotalAmountOverCreditLimit_4 | TotalAmountOverCreditLimit_5 | age_0 | age_1 | age_2 | age_3 | ... | Count6089 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | ... | |
| 1 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 0 | 0 | ... | |
| 2 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | ... | |
| 3 | 0 | 0 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | ... | |
| 4 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 1 | 0 | ... | |
| ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... |
| 149995 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | ... | |
| 149996 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | ... | |
| 149997 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 1 | ... | |
| 149998 | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | ... | |
| 149999 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | ... | |

150000 rows × 65 columns

# Model choosing

We decide to use some classifier model ,therefore we used 3 models:

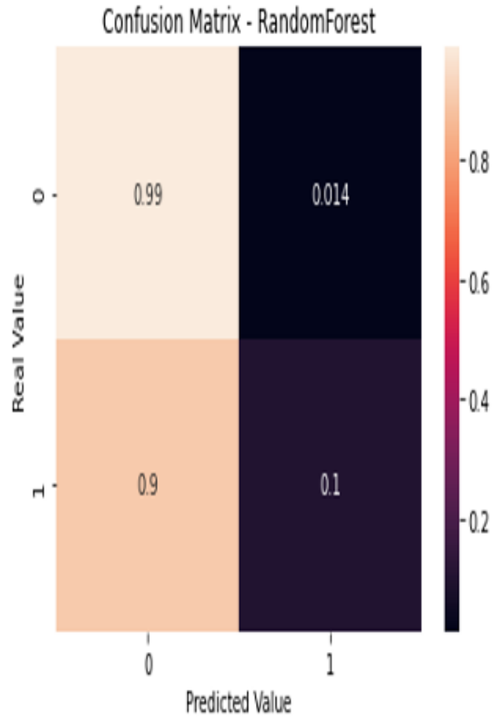
- KNClassifier
- XGBClassifier
- RandomForestClassfier

# RandomForest Classifier

Firstly we used this model for predict scoring of our users.
We faced with problem of imbalanced class.
But our accuracy value was good about 92%, but it was because of our classifier gives a lot of zeros. 2nd type mistakes.
After that we decide to use next metrics: recall, precision and f1-score.

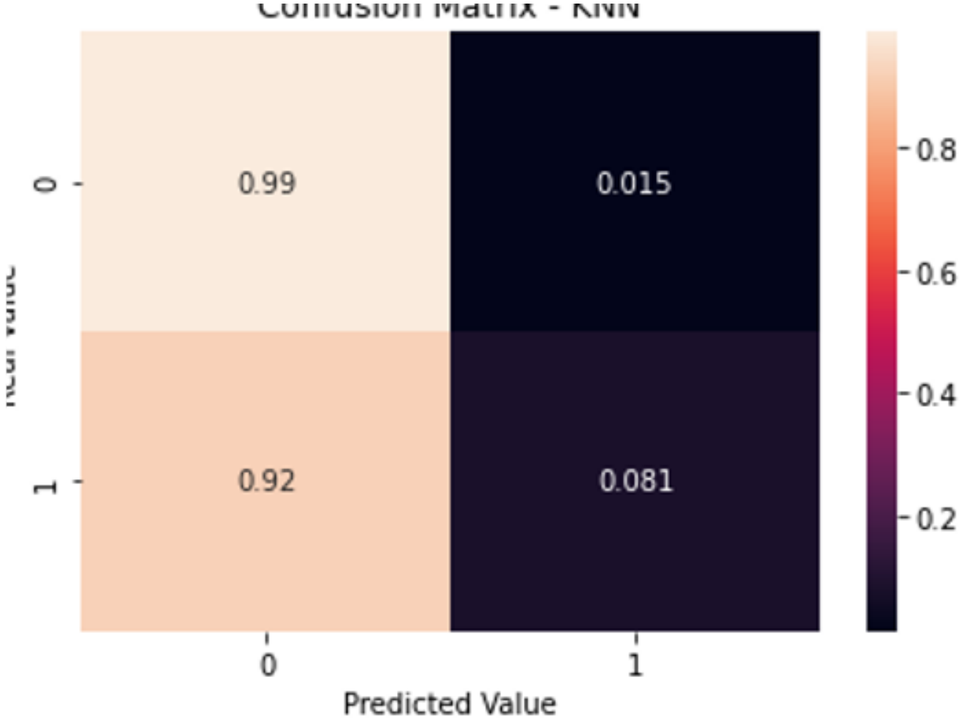
```
y_pred = randomFC.predict(X_test)
import sklearn.metrics as sm
from sklearn.metrics import accuracy_score, f1_score, confusion_matrix, classification_report
print("Mean absolute error =", round(sm.mean_absolute_error(Y_test, y_pred), 2))
print("Mean squared error =", round(sm.mean_squared_error(Y_test, y_pred), 2))
print("Median absolute error =", round(sm.median_absolute_error(Y_test, y_pred), 2))
print("Explain variance score =", round(sm.explained_variance_score(Y_test, y_pred), 2))
print("R2 score =", round(sm.r2_score(Y_test, y_pred), 2))
KNC_pred = y_pred
print("Accuracy score =",round(accuracy_score(Y_test,KNC_pred),5))
print(classification_report(Y_test, KNC_pred))
```

```
Mean absolute error = 0.07
Mean squared error = 0.07
Median absolute error = 0.0
Explain variance score = -0.14
R2 score = -0.16
Accuracy score = 0.92909
              precision    recall  f1-score   support

           0       0.95      0.98      0.96     46266
           1       0.41      0.20      0.27      3234

    accuracy                           0.93     49500
   macro avg       0.68      0.59      0.62     49500
weighted avg       0.91      0.93      0.92     49500
```



# KNeigbours Classifier

KNeigbours Classifier shows the worst results




```
KNC = KNeighborsClassifier(weights='distance',n_neighbors=3)
KNC.fit(X_train,Y_train)
```

```
KNeighborsClassifier(n_neighbors=3, weights='distance')
```

+ Code + Markdown

```
KNC_pred = KNC.predict(X_test)
import sklearn.metrics as sm

print("Mean absolute error =", round(sm.mean_absolute_error(Y_test, KNC_pred), 2))
print("Mean squared error =", round(sm.mean_squared_error(Y_test, KNC_pred), 2))
print("Median absolute error =", round(sm.median_absolute_error(Y_test, KNC_pred), 2))
print("Explain variance score =", round(sm.explained_variance_score(Y_test, KNC_pred), 2))
print("R2 score =", round(sm.r2_score(Y_test, KNC_pred), 2))
print("Accuracy score =",round(accuracy_score(Y_test,KNC_pred),2))
print(classification_report(Y_test, KNC_pred))
```

```
Mean absolute error = 0.08
Mean squared error = 0.08
Median absolute error = 0.0
Explain variance score = -0.23
R2 score = -0.25
Accuracy score = 0.92
              precision    recall  f1-score   support

           0       0.94      0.98      0.96     46266
           1       0.32      0.14      0.20      3234

    accuracy                           0.92     49500
   macro avg       0.63      0.56      0.58     49500
weighted avg       0.90      0.92      0.91     49500
```

# XGB Classifier

- We decide to use XGBClassifier because our sample with imbalanced class
- We used 3 metrics for analyze results of our model recall, precision and f1-score
- After permutations of a lot of variants of parameters we choose next parametrs(max_depth=3, learning_rate=0.01, n_estimators=1000, gamma=1,scale_pos_weight=4)

```
sns.heatmap(confusion_matrix(Y_test, y_pred_xgb, normalize='true'), annot=True, ax=ax)
ax.set_title('Confusion Matrix - XGBoost')
ax.set_xlabel('Predicted Value')
ax.set_ylabel('Real Value')

plt.show()

accuracy = accuracy_score(Y_test, y_pred_xgb)
print("Accuracy: %.2f%%" % (accuracy * 100.0))
```

```
opt/conda/lib/python3.7/site-packages/xgboost/sklearn.py:1146: UserWarning: The use of label encoder in XGBClassifier is deprecated and will be remo
ed in a future release. To remove this warning, do the following: 1) Pass option use_label_encoder=False when constructing XGBClassifier object; and
) Encode your labels (y) as integers starting with 0, i.e. 0, 1, 2, ..., [num_class - 1].
 warnings.warn(label_encoder_deprecation_msg, UserWarning)
08:37:37] WARNING: ../src/learner.cc:1095: Starting in XGBoost 1.3.0, the default evaluation metric used with the objective 'binary:logistic' was ch
nged from 'error' to 'logloss'. Explicitly set eval_metric if you'd like to restore the old behavior.
opt/conda/lib/python3.7/site-packages/xgboost/data.py:114: UserWarning: Use subset (sliced data) of np.ndarray is not recommended because it will ge
erate extra copies and increase memory consumption
 "because it will generate extra copies and increase " +
              precision    recall  f1-score   support

           0       0.96      0.95      0.96     46247
           1       0.40      0.50      0.44      3253

    accuracy                           0.92     49500
   macro avg       0.68      0.72      0.70     49500
eighted avg       0.93      0.92      0.92     49500
```