MARKO NATALIA

natalia.marko.ds@gmail.com

www.linkedin.com/in/natalia-marko-2570a5303

```python
import numpy as np
import pandas as pd
import random as rn
import matplotlib.pyplot as plt
import seaborn as sns
import plotly.express as px
import plotly.subplots as sp
import lightgbm as lgb
import warnings
warnings.filterwarnings("ignore")
import math
from sklearn.model_selection import train_test_split, cross_val_score, cross_val
from sklearn.preprocessing import StandardScaler, MinMaxScaler, RobustScaler
from sklearn.ensemble import (
    RandomForestClassifier, AdaBoostClassifier,
    GradientBoostingClassifier, IsolationForest
)
from sklearn.neighbors import KNeighborsClassifier, LocalOutlierFactor
from sklearn.svm import OneClassSVM
from sklearn.covariance import EllipticEnvelope
from sklearn.metrics import (
    classification_report, confusion_matrix,
    accuracy_score, balanced_accuracy_score, f1_score,
    precision_score, recall_score, roc_auc_score,
    average_precision_score, roc_curve, auc
)
from sklearn.metrics import mean_squared_log_error
from sklearn.linear_model import LinearRegression, Ridge, Lasso, ElasticNet
from sklearn.svm import SVR
from sklearn.neighbors import KNeighborsRegressor
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor, GradientBoostingRegressor, A
from sklearn.metrics import mean_squared_log_error
from xgboost import XGBRegressor
from lightgbm import LGBMRegressor
from sklearn.metrics import mean_squared_error
from sklearn.decomposition import PCA
from sklearn.linear_model import Ridge

from xgboost import XGBClassifier
from sklearn.utils.class_weight import compute_class_weight
from sklearn.ensemble import RandomForestClassifier
from xgboost import XGBClassifier
from sklearn.linear_model import LogisticRegression
from lightgbm import LGBMClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import roc_auc_score, recall_score
```

```
# !wget https://raw.githubusercontent.com/username/repo/branch/myfunctions.py
# import myfunctions


# from importlib import reload
# reload(myfunctions)
```

CHURN model

Model Target: (target)

```
1- Активний абонент через 2 місяці став неактивним
0- Активний абонент через 2 місяці залишився активним
```

Task: Identify subscribers who will leave in two months and churn

Sample size

Train 50% 150K / Test 50% 150K

Model Quality Metrics are defined by indicators AUC, Classification report, Confusion matrix on the Test Sample.

The baseline metric chosen for comparing the quality of student models is AUC with a baseline result (Baseline model) of 0.89 on the Test Sample.

This model will help us better understand how subscribers behave and what factors may influence their decision to leave the service.

Our project requires the use of various data analysis and machine learning methods, including:

- **Feature Exploration**: This means exploring your data and understanding which variables may be important for predicting subscriber churn. We will use visualization and statistical methods for this.
- **Feature Encoding**: Some machine learning algorithms require that all input data be in numerical format. We will try to use various encoding methods to convert categorical data into numerical values.
- **Model Selection and Hyperparameter Tuning**: Choosing the right model and tuning its parameters can affect the quality of our forecast. We will try to use methods such as cross-validation and grid search to find the best model and set of parameters.
- **Feature Importance Evaluation**: After training the model, we will be able to determine which variables were the most important for predicting churn.
- **Model Interpretation with SHAP**: This is a method that helps understand the contribution of each attribute to the model's prediction for a specific observation.

```
from google.colab import drive
drive.mount('/content/drive/')
```

```
Drive already mounted at /content/drive/; to attempt to forcibly remount, c
```

```
churn_train_model_fe = pd.read_pickle('/content/drive/MyDrive/Churn_model_Vodaf
churn_train_model_b_num = pd.read_pickle('/content/drive/MyDrive/Churn_model_Vc
churn_train_model_dpi = pd.read_pickle('/content/drive/MyDrive/Churn_model_Voda
```

```
import sys
import numpy as np  # Ensure numpy is imported with the alias 'np'

# Add the directory containing your .py file to the sys.path
sys.path.append('/content/drive/MyDrive/Churn_model_Vodafone/')

import myfunctions

df_reduced = myfunctions.reduce_mem_usage(churn_train_model_fe)
```

```
Mem. usage decreased to 238.90 Mb (0.0% reduction)
```

```
df_train = myfunctions.reduce_mem_usage(churn_train_model_fe)
```

```
Mem. usage decreased to 238.90 Mb (0.0% reduction)
```

Double-click (or enter) to edit

```
df_train.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 150000 entries, 0 to 149999
Columns: 817 entries, Ama_rchrgmnt_sum_max_mnt1 to abon_id
dtypes: float16(773), float32(1), float64(11), int8(32)
memory usage: 238.9 MB
```

```
df_train.head()
```

	Ama_rchrgmnt_sum_max_mnt1	content_clc_mea_mnt1	content_cnt_max_mnt1	vc
0	0	0.0	13.843750	
1	0	0.0	11.359375	
2	0	0.0	10.265625	
3	0	0.0	9.976562	
4	0	0.0	6.750000	

5 rows × 817 columns

```
df_train.describe().T
```

	count	mean	std	min
Ama_rchrgmnt_sum_max_mnt1	150000.0	0.000000e+00	0.000000e+00	0.000000e+00
content_clc_mea_mnt1	150000.0	0.000000e+00	0.000000e+00	0.000000e+00
content_cnt_max_mnt1	150000.0	NaN	0.000000e+00	0.000000e+00
voice_out_short_part_max_mnt1	150000.0	NaN	0.000000e+00	0.000000e+00
voice_mts_in_nrest_part_std_mnt1	150000.0	NaN	0.000000e+00	0.000000e+00
...	...	...	...	...
MV_DOU_PPM_VF	77634.0	NaN	0.000000e+00	1.480469e+00
MV_DOU_Neg_Bal	113.0	2.970703e+00	1.699219e+00	1.480469e+00
MV_ot_total	108550.0	NaN	0.000000e+00	0.000000e+00
target	150000.0	0.000000e+00	0.000000e+00	0.000000e+00
abon_id	150000.0	7.897443e+07	4.293146e+07	1.545052e+06

817 rows × 8 columns

```
df_train.fillna(-1, inplace=True)
df_train.dtypes
```

```
Ama_rchrgmnt_sum_max_mnt1                int8
content_clc_mea_mnt1                  float16
content_cnt_max_mnt1                  float16
voice_out_short_part_max_mnt1         float16
voice_mts_in_nrest_part_std_mnt1      float16
                                        ...
MV_DOU_PPM_VF                         float16
MV_DOU_Neg_Bal                        float16
MV_ot_total                           float16
target                                float16
abon_id                               float32
Length: 817, dtype: object
```

```
df_train.isnull().sum()
```

```
Ama_rchrgmnt_sum_max_mnt1             0
content_clc_mea_mnt1                  0
content_cnt_max_mnt1                  0
voice_out_short_part_max_mnt1         0
voice_mts_in_nrest_part_std_mnt1      0
                                     ..
MV_DOU_PPM_VF                         0
MV_DOU_Neg_Bal                        0
MV_ot_total                           0
target                                0
abon_id                               0
Length: 817, dtype: int64
```

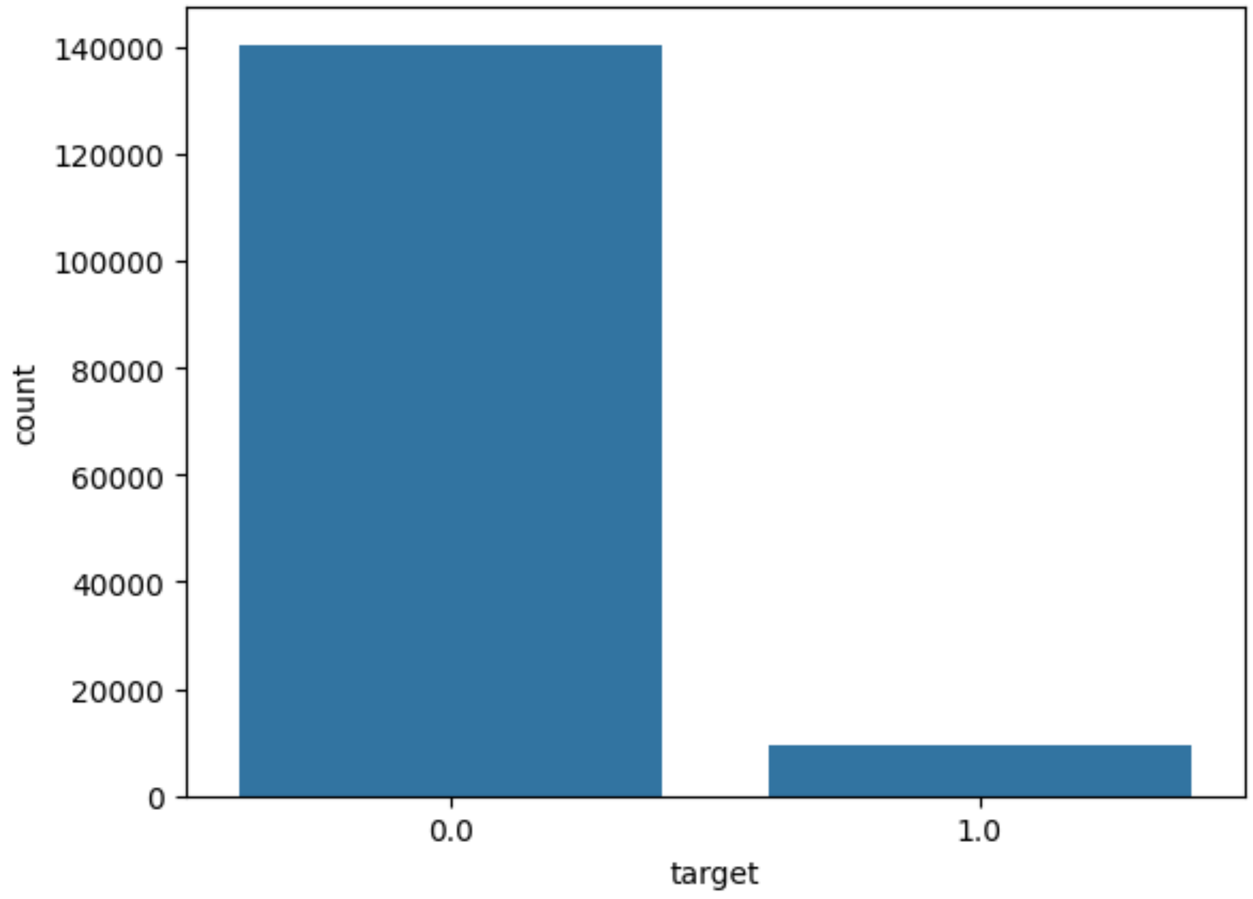
```
# lets use the StandardScaler
scaler = StandardScaler()
df_train_scaled = scaler.fit_transform(df_train)
df_train_scaled = pd.DataFrame(df_train_scaled, index=df_train.index, columns=c
```

```
# target values
df_train['target'].value_counts()
```

```
target
0.0    140414
1.0      9586
Name: count, dtype: int64
```

```
df_train = df_train.astype('float32')
sns.countplot(x='target', data=df_train)
```

```
<Axes: xlabel='target', ylabel='count'>
```



LightGBM is a popular gradient boosting library

that can be effective for classification. It usually shows high performance and is flexible in terms of hyperparameter tuning.

```
y = df_train['target']
X = df_train_scaled.drop(['target', 'abon_id'], axis=1)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, rando
# Splitting the dataset
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_

# Initialize the LGBMClassifier
model_1 = LGBMClassifier(random_state=42)

# Train the model
model_1.fit(X_train, y_train, eval_set=[(X_test, y_test)], callbacks=[lgb.early_
```

```
[LightGBM] [Info] Number of positive: 7651, number of negative: 112349
[LightGBM] [Info] Auto-choosing col-wise multi-threading, the overhead of t
You can set `force_col_wise=true` to remove the overhead.
[LightGBM] [Info] Total Bins 120653
[LightGBM] [Info] Number of data points in the train set: 120000, number of
[LightGBM] [Info] [binary:BoostFromScore]: pavg=0.063758 -> initscore=-2.68
[LightGBM] [Info] Start training from score -2.686774
Training until validation scores don't improve for 5 rounds
Early stopping, best iteration is:
[76]    valid_0's binary_logloss: 0.149994
▾        LGBMClassifier
LGBMClassifier(random_state=42)
```

```
y_pred_1 = model_1.predict(X_test)

# Перетворити ймовірності в бінарні мітки для отримання Classification Report і
y_pred_binary = [1 if prob > 0.5 else 0 for prob in y_pred_1]

# Розрахувати AUC
auc_score_1 = roc_auc_score(y_test, y_pred_1)
print('AUC 1: ', auc_score_1)

# Розрахувати Classification Report
print('Classification Report 1:')
print(classification_report(y_test, y_pred_binary))

# Розрахувати Confusion Matrix
print('Confusion Matrix 1:')
print(confusion_matrix(y_test, y_pred_binary))

# Отримати значення FPR, TPR та трешхолдів
fpr_1, tpr_1, thresholds = roc_curve(y_test, y_pred_1)

# Обчислити площу під ROC-кривою (AUC-ROC)
auc_roc_1 = roc_auc_score(y_test, y_pred_1)
print('AUC-ROC 1: ', auc_roc_1)

# Обчислити площу під ROC-кривою (AUC-ROC)
recall_score_1 = recall_score(y_test, y_pred_1)
print('Recall_score_1: ', recall_score_1)
```

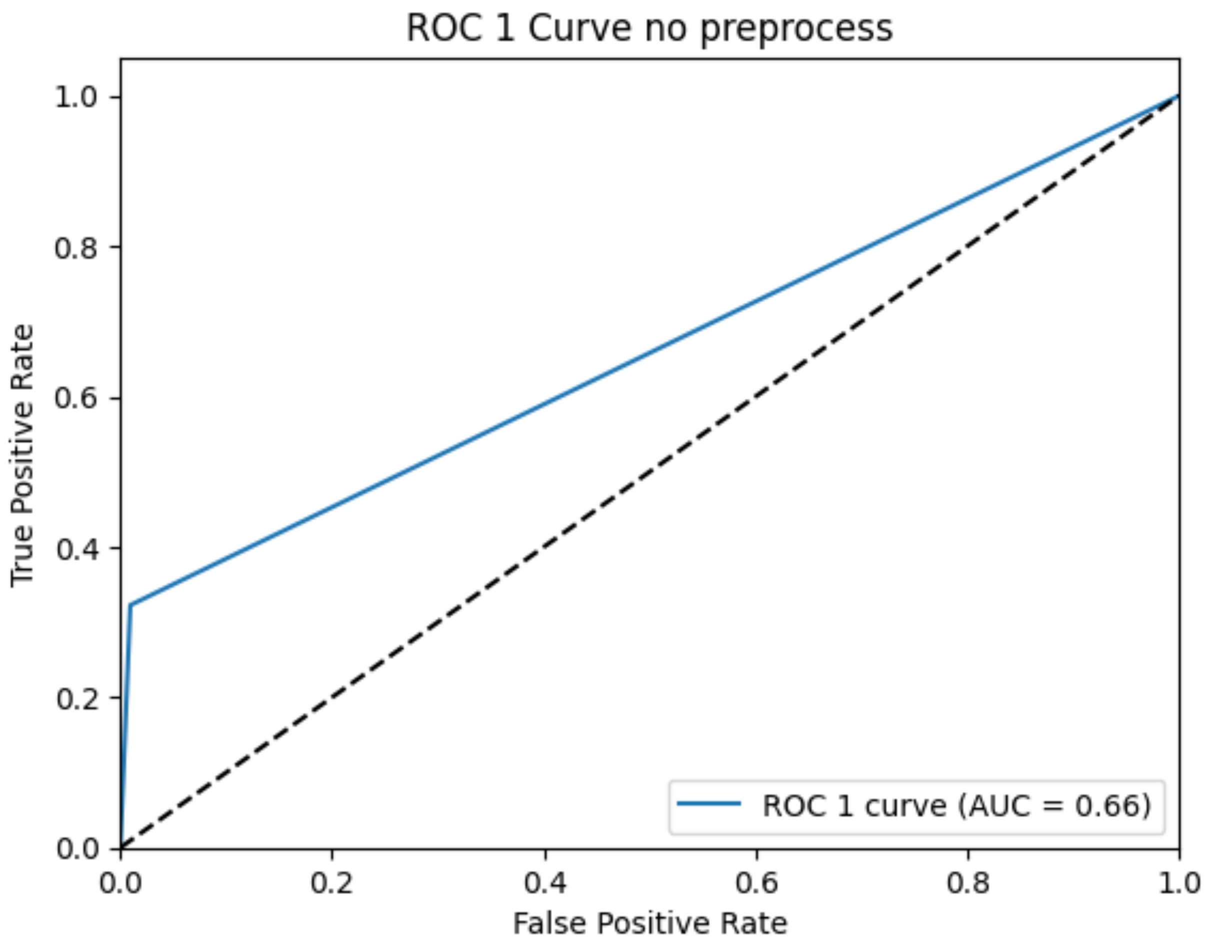
```
AUC 1:  0.6565191271094096
Classification Report 1:
              precision    recall  f1-score   support

         0.0       0.95      0.99      0.97     28065
         1.0       0.70      0.32      0.44      1935

    accuracy                           0.95     30000
   macro avg       0.83      0.66      0.71     30000
weighted avg       0.94      0.95      0.94     30000

Confusion Matrix 1:
[[27800   265]
 [ 1311   624]]
AUC-ROC 1:  0.6565191271094096
Recall_score_1:  0.32248062015503876
```

```
# plot ROC-AUC curve
plt.figure()
plt.plot(fpr_1, tpr_1, label='ROC 1 curve (AUC = %0.2f)' % auc_roc_1)
plt.plot([0, 1], [0, 1], 'k--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC 1 Curve no preprocess')
plt.legend(loc="lower right")
plt.show()
```



First results

AUC-ROC 1: 0.66 Recall 1: 0.33

TP 641, FN 1294 - not very satisfying

The model is not very good at ranking the positive cases. This situation suggests there might be issues with class imbalance. We also can use threshold settings to improve ranking.

```
# створюємо матрицю кореляцій фітчей
df_corr = df_train.drop(['target', 'abon_id'],axis=1).corr(method='spearman')
# створюємо маску щоб виключити самокореляцію
mask = np.ones(df_corr.columns.size) - np.eye(df_corr.columns.size)
df_corr = mask * df_corr
cols_to_drops = []
# loop through each variable
for col in df_corr.columns.values:
    # if we've already determined to drop the current variable, continue
    if np.in1d([col],cols_to_drops):
        continue

    # find all the variables that are highly correlated with the current variat
    # and add them to the drop list
    corr = df_corr[abs(df_corr[col]) > 0.98].index
    cols_to_drops = np.union1d(cols_to_drops, corr)

cols_before_drop = df_train.columns
df_preprocess = df_train.drop(columns=cols_to_drops)

cols_after_drop = df_preprocess.columns
dropped_columns = list(set(cols_before_drop) - set(cols_after_drop))
print("Deleted columns:", len(dropped_columns),'units')
```

```
Deleted columns: 174 units
```

```
df_preprocess.head()
```

	Ama_rchrgmnt_sum_max_mnt1	content_clc_mea_mnt1	content_cnt_max_mnt1	vc
0	0.0	0.0	13.843750	
1	0.0	0.0	11.359375	
2	0.0	0.0	10.265625	
3	0.0	0.0	9.976562	
4	0.0	0.0	6.750000	

5 rows × 643 columns

Train on 643 features

Therefore, our dataset now has 643 features, 174 features were removed because the correlation percentage between them and other features was very high (>98%). This interfered with our ability to conduct quality training and interpretation of the model.

```
# splitting df_preprocess
X = df_preprocess.drop(['target', 'abon_id'],axis=1)
y = df_preprocess['target']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random

# Initialize the LGBMClassifier
model_2 = LGBMClassifier(random_state=42)

# Train the model
model_2.fit(X_train, y_train, eval_set=[(X_test, y_test)], callbacks=[lgb.early
```

```
[LightGBM] [Info] Number of positive: 7651, number of negative: 112349
[LightGBM] [Info] Auto-choosing col-wise multi-threading, the overhead of t
You can set `force_col_wise=true` to remove the overhead.
[LightGBM] [Info] Total Bins 94115
[LightGBM] [Info] Number of data points in the train set: 120000, number of
[LightGBM] [Info] [binary:BoostFromScore]: pavg=0.063758 -> initscore=-2.68
[LightGBM] [Info] Start training from score -2.686774
Training until validation scores don't improve for 5 rounds
Early stopping, best iteration is:
[71]    valid_0's binary_logloss: 0.149876
▾         LGBMClassifier
LGBMClassifier(random_state=42)
```

```
# Робимо прогнози
y_pred_2 = model_2.predict(X_test)

from sklearn.metrics import roc_curve, roc_auc_score, classification_report, cc

# Перетворити ймовірності в бінарні мітки для отримання Classification Report і
y_pred_binary = [1 if prob > 0.5 else 0 for prob in y_pred_2]

# Розрахувати AUC
auc_score_2 = roc_auc_score(y_test, y_pred_2)
print('AUC 2: ', auc_score_2)

# Розрахувати Classification Report
print('Classification Report 2:')
print(classification_report(y_test, y_pred_binary))

# Розрахувати Confusion Matrix
print('Confusion Matrix 2:')
print(confusion_matrix(y_test, y_pred_binary))

# Отримати значення FPR, TPR та трешхолдів
fpr_2, tpr_2, thresholds = roc_curve(y_test, y_pred_2)

# Обчислити площу під ROC-кривою (AUC-ROC)
auc_roc_2 = roc_auc_score(y_test, y_pred_2)
print('AUC-ROC 2: ', auc_roc_2)

# Обчислити площу під ROC-кривою (AUC-ROC)
recall_score_2 = recall_score(y_test, y_pred_2)
print('Recall_score_2: ', recall_score_2)
```

```
AUC 2:  0.6617493719590597
Classification Report 2:
              precision    recall  f1-score   support

         0.0       0.96      0.99      0.97     28065
         1.0       0.72      0.33      0.46      1935

    accuracy                           0.95     30000
   macro avg       0.84      0.66      0.71     30000
weighted avg       0.94      0.95      0.94     30000

Confusion Matrix 2:
[[27818   247]
 [ 1292   643]]
AUC-ROC 2:  0.6617493719590597
Recall_score_2:  0.33229974160206716
```

```
Confusion Matrix 2: - slight better
[[27818   247]
[ 1292   643]]
AUC-ROC 2:  0.66
Recall_score_2:  0.33
```

def preproccess_data

list all transformation steps into one custom function

```
churn_train_model_fe = pd.read_pickle('/content/drive/MyDrive/Churn_model_Vodaf

df_train_pp = myfunctions.preproccess_data(churn_train_model_fe)
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-41-63cae71b7971> in <cell line: 3>()
      1 churn_train_model_fe =
pd.read_pickle('/content/drive/MyDrive/Churn_model_Vodafone/churn_train_mod
      2
----> 3 df_train_pp = myfunctions.preproccess_data(churn_train_model_fe)

/content/drive/MyDrive/Churn_model_Vodafone/myfunctions.py in
preproccess_data(df)
     50     cols_before_drop = df.columns
     51     df_pp = df.drop(columns=cols_to_drops)
---> 52     scaler = StandardScaler()
     53     train_scaled = scaler.fit_transform(df_pp)
     54     df_train_scaled = pd.DataFrame(train_scaled, index=df_pp.index,
columns=df_pp.columns)
```

Lets try to apply popular models to our data

```
pip install catboost
```

```
from catboost import CatBoostClassifier
# Split df_preprocess into X and y
X = df_train_pp
y = df_train['target']

# Split data into training and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random

# Dictionary of models to train
models = {
    "Random Forest": RandomForestClassifier(),
    "XGBoost": XGBClassifier(),
    "CatBoost": CatBoostClassifier(),
    "Logistic Regression": LogisticRegression(),
    "LightGBM": LGBMClassifier()
}

# Train each model and evaluate
results = []

for name, model in models.items():
    # Train the model
    if name == "CatBoost":
        model.fit(X_train, y_train, verbose=False)  # Suppress output for CatBo
    else:
        model.fit(X_train, y_train)

    # Predict probabilities and binary outcomes
    prob_pred = model.predict_proba(X_test)[:, 1]
    binary_pred = model.predict(X_test)

    # Calculate metrics
    auc_score = roc_auc_score(y_test, prob_pred)
    recall = recall_score(y_test, binary_pred)

    # Append results
    results.append({
        "Model": name,
        "AUC-ROC": auc_score,
        "Recall": recall
    })

# Convert results to DataFrame
results_df = pd.DataFrame(results)

# Display the DataFrame
print(results_df)
```

```
results_df


from sklearn.utils import class_weight

# Calculate the scale_pos_weight value
weights = class_weight.compute_class_weight('balanced', classes=np.unique(y_tra
scale_pos_weight = weights[0] / weights[1]
scale_pos_weight


# Dictionary of models to train
models = {
    "Random Forest": RandomForestClassifier(class_weight='balanced'),
    "XGBoost": XGBClassifier(scale_pos_weight=scale_pos_weight),
    "CatBoost": CatBoostClassifier(auto_class_weights='Balanced', verbose=0),
    "Logistic Regression": LogisticRegression(class_weight='balanced'),
    "LightGBM": LGBMClassifier(is_unbalance=True)
}


# Train each model and evaluate
results = []

for name, model in models.items():
    # Train the model
    model.fit(X_train, y_train)

    # Predict probabilities and binary outcomes
    prob_pred = model.predict_proba(X_test)[:, 1]
    binary_pred = model.predict(X_test)

    # Calculate metrics
    auc_score = roc_auc_score(y_test, prob_pred)
    recall = recall_score(y_test, binary_pred)

    # Append results
    results.append({
        "Model": name,
        "AUC-ROC": auc_score,
        "Recall": recall
    })

# Convert results to DataFrame
results_df = pd.DataFrame(results)

# Display the DataFrame
print(results_df)


results_df
```

```
cb_model = CatBoostClassifier(auto_class_weights='Balanced', verbose=0)
log_model = LogisticRegression(class_weight='balanced')
lgb_model = LGBMClassifier(is_unbalance=True)

# Train models
cb_model.fit(X_train, y_train)
log_model.fit(X_train, y_train)
lgb_model.fit(X_train, y_train)


# # Predict probabilities and binary outcomes
prob_pred_cb = cb_model.predict_proba(X_test)[:, 1]
binary_pred_cb = cb_model.predict(X_test)

prob_pred_log = log_model.predict_proba(X_test)[:, 1]
binary_pred_log = log_model.predict(X_test)

prob_pred_lgb = lgb_model.predict_proba(X_test)[:, 1]
binary_pred_lgb = lgb_model.predict(X_test)


# Calculate metrics
auc_score_cb = roc_auc_score(y_test, prob_pred_cb)
recall_cb = recall_score(y_test, binary_pred_cb)

auc_score_log = roc_auc_score(y_test, prob_pred_log)
recall_log = recall_score(y_test, binary_pred_log)

auc_score_lgb = roc_auc_score(y_test, prob_pred_lgb)
recall_lgb = recall_score(y_test, binary_pred_lgb)
```

```
from sklearn.metrics import precision_recall_curve

scores = [prob_pred_cb, prob_pred_log, prob_pred_lgb]
i = 1
optimal_thresholds = []
for x in scores:
    # Calculate precision, recall, and thresholds
    precision, recall, thresholds = precision_recall_curve(y_test, x)

    # Calculate F1 score for each threshold
    f1_scores = 2 * (precision * recall) / (precision + recall)
    # Remove NaN values that could arise from division by zero
    f1_scores = f1_scores[~np.isnan(f1_scores)]

    # Find the threshold where F1 score is maximum
    optimal_idx = np.argmax(f1_scores)
    optimal_threshold = thresholds[optimal_idx]
    optimal_f1 = f1_scores[optimal_idx]

    # Plotting the F1 curve
    plt.plot(thresholds, f1_scores[:-1], label='F1 Curve')
    plt.xlabel('Thresholds')
    plt.ylabel('F1 Score')
    plt.title(f'F1 Score vs. Thresholds {i}')
    plt.axvline(x=optimal_threshold, color='r', linestyle='--', label=f'Optimal
    plt.legend()
    plt.show()
    i += 1
    # Print the optimum threshold with its F1 score
    print(f"Optimum threshold: {optimal_threshold:.4f}, with F1 score: {optimal
    optimal_thresholds.append(optimal_threshold)
```

```
optimal_thresholds_r = [round(x, 2) for x in optimal_thresholds]
optimal_thresholds_r
```

Feature importance

```
lgb_importance = lgb_model.feature_importances_
log_importance = abs(log_model.coef_[0])
importance_cb = cb_model.feature_importances_

# Normalize feature importances for LightGBM
lgb_importance_normalized = lgb_importance / np.sum(lgb_importance)

# Normalize feature importances for Logistic Regression
log_importance_normalized = log_importance / np.sum(log_importance)

# Normalize feature importances for CatBoost
cb_importance_normalized = importance_cb / np.sum(importance_cb)

# Create a DataFrame to hold the feature importances
feature_names = X_train.columns  # Assuming X_train is your training dataset wi
importances_df = pd.DataFrame({
    'Feature': feature_names,
    'LightGBM': lgb_importance_normalized,
    'LogReg': log_importance_normalized,
    'CatBoost': cb_importance_normalized
})



sorted_importances = importances_df.sort_values(by='LightGBM', ascending=False)
sorted_importances


th = 0.004
important_features_df = importances_df[
    (importances_df['LightGBM'] > th) |
    (importances_df['LogReg'] > th) |
    (importances_df['CatBoost'] > th)
]

# Extract the feature names
important_features = important_features_df['Feature']
top_features = important_features.to_list()


important_features


from sklearn.ensemble import VotingClassifier
from sklearn.model_selection import cross_val_score

# Train models
cb_model.fit(X_train[top_features], y_train)
log_model.fit(X_train[top_features], y_train)
lgb_model.fit(X_train[top_features], y_train)
```

```
# # Predict probabilities and binary outcomes
prob_pred_cb = cb_model.predict_proba(X_test[top_features])[:, 1]
binary_pred_cb = cb_model.predict(X_test[top_features])

prob_pred_log = log_model.predict_proba(X_test[top_features])[:, 1]
binary_pred_log = log_model.predict(X_test[top_features])

prob_pred_lgb = lgb_model.predict_proba(X_test[top_features])[:, 1]
binary_pred_lgb = lgb_model.predict(X_test[top_features])

scores = [prob_pred_cb, prob_pred_log, prob_pred_lgb]
i = 1

optimal_thresholds = []
for x in scores:
    # Calculate precision, recall, and thresholds
    precision, recall, thresholds = precision_recall_curve(y_test, x)

    # Calculate F1 score for each threshold
    f1_scores = 2 * (precision * recall) / (precision + recall)
    # Remove NaN values that could arise from division by zero
    f1_scores = f1_scores[~np.isnan(f1_scores)]

    # Find the threshold where F1 score is maximum
    optimal_idx = np.argmax(f1_scores)
    optimal_threshold = thresholds[optimal_idx]
    optimal_f1 = f1_scores[optimal_idx]

    # Plotting the F1 curve
    plt.plot(thresholds, f1_scores[:-1], label='F1 Curve')
    plt.xlabel('Thresholds')
    plt.ylabel('F1 Score')
    plt.title(f'F1 Score vs. Thresholds {i}')
    plt.axvline(x=optimal_threshold, color='r', linestyle='--', label=f'Optimal
    plt.legend()
    plt.show()
    i += 1
    # Print the optimum threshold with its F1 score
    print(f"Optimum threshold: {optimal_threshold:.4f}, with F1 score: {optimal
    optimal_thresholds.append(optimal_threshold)




optimal_thresholds
```

```
# Calculate metrics
auc_score_cb = roc_auc_score(y_test, prob_pred_cb)
recall_cb = recall_score(y_test, binary_pred_cb)

auc_score_log = roc_auc_score(y_test, prob_pred_log)
recall_log = recall_score(y_test, binary_pred_log)

auc_score_lgb = roc_auc_score(y_test, prob_pred_lgb)
recall_lgb = recall_score(y_test, binary_pred_lgb)

print(auc_score_cb, recall_cb)
print(auc_score_log, recall_log)
print(auc_score_lgb, recall_lgb)


# Function to plot confusion matrix
def plot_confusion_matrix(y_true, y_pred, title):
    cm = confusion_matrix(y_true, y_pred)
    plt.figure(figsize=(3, 2))
    sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', cbar=False)
    plt.xlabel('Predicted')
    plt.ylabel('Actual')
    plt.title(title)
    plt.show()


plot_confusion_matrix(y_test, binary_pred_cb, 'catboost')
plot_confusion_matrix(y_test, binary_pred_log, 'logregression')
plot_confusion_matrix(y_test, binary_pred_lgb, 'LightGBM')
```

fine tuning models by Grid Search

```
from sklearn.model_selection import GridSearchCV

# Define the parameter grid for CatBoostClassifier
cb_params = {
    'depth': [4, 6, 10],
    'learning_rate': [0.01, 0.05, 0.1],
    'iterations': [30, 50, 100]
}

# Define the parameter grid for LogisticRegression
log_params = {
    'C': [0.1, 1, 10],
    'solver': ['liblinear', 'saga']
}

# Define the parameter grid for LGBMClassifier
lgb_params = {
    'num_leaves': [31, 41, 51],
    'learning_rate': [0.01, 0.05, 0.1],
    'n_estimators': [20, 40, 60]
}

# Initialize the models with the parameters you want to keep fixed
cb_model = CatBoostClassifier(auto_class_weights='Balanced', verbose=0)
log_model = LogisticRegression(class_weight='balanced')
lgb_model = LGBMClassifier(is_unbalance=True)

# Set up GridSearchCV for each model
cb_grid_search = GridSearchCV(cb_model, cb_params, cv=5, scoring='roc_auc', n_j
log_grid_search = GridSearchCV(log_model, log_params, cv=5, scoring='roc_auc',
lgb_grid_search = GridSearchCV(lgb_model, lgb_params, cv=5, scoring='roc_auc',

# Perform grid search (This may take some time depending on your machine's comp
cb_grid_search.fit(X_train, y_train)
log_grid_search.fit(X_train, y_train)
lgb_grid_search.fit(X_train, y_train)

# Print the best parameters for each model
print("Best parameters for CatBoostClassifier:", cb_grid_search.best_params_)
print("Best parameters for LogisticRegression:", log_grid_search.best_params_)
print("Best parameters for LGBMClassifier:", lgb_grid_search.best_params_)
```

```
Best parameters for CatBoostClassifier: {'depth': 6, 'iterations': 100, 'learn
Best parameters for LogisticRegression: {'C': 0.1, 'solver': 'saga'}
Best parameters for LGBMClassifier: {'learning_rate': 0.1, 'n_estimators': 60,
# Retrieve the best estimators from the grid searches
best_cb_model = cb_grid_search.best_estimator_
best_log_model = log_grid_search.best_estimator_
best_lgb_model = lgb_grid_search.best_estimator_
```

Voting Classifier

```
from sklearn.ensemble import VotingClassifier
# Set up the Voting Classifier
voting_clf = VotingClassifier(estimators=[
    ('cb', best_cb_model),
    ('lr', best_log_model),
    ('lgb', best_lgb_model)
], voting='soft')

# Fit the Voting Classifier with the training data
voting_clf.fit(X_train[top_features], y_train)


# Evaluate the model
scores = cross_val_score(voting_clf, X_train[top_features], y_train, cv=3, scor

print(f'AUC: {scores.mean():.2f} (+/- {scores.std() * 2:.2f})')


# Now you can use voting_clf to make predictions and evaluate the model
voting_pred = voting_clf.predict(X_test[top_features])
voting_proba = voting_clf.predict_proba(X_test[top_features])[:, 1]

# Evaluate the voting classifier
recall = recall_score(y_test, voting_pred)
f1 = f1_score(y_test, voting_pred)
auc = roc_auc_score(y_test, voting_proba)

print(f"Recall on test set: {recall:.4f}")
print(f"F1 Score on test set: {f1:.4f}")
print(f"ROC AUC on test set: {auc:.4f}")
```

Start coding or generate with AI.

```
# Calculate the confusion matrix
cm = confusion_matrix(y_test, voting_pred)

# Plotting the confusion matrix with Seaborn
plt.figure(figsize=(3, 2))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', cbar=False)
plt.xlabel('Predicted labels')
plt.ylabel('True labels')
plt.title('Confusion Matrix')
plt.show()


voting_clf.fit(X[top_features], y)

prob_pred = voting_clf.predict_proba(X[top_features])[:, 1]
binary_pred = voting_clf.predict(X[top_features])

# Calculate metrics
auc = roc_auc_score(y, prob_pred)
recall = recall_score(y, binary_pred)
f1 = f1_score(y, binary_pred)

print(f"Recall on test set: {recall:.4f}")
print(f"F1 Score on test set: {f1:.4f}")
print(f"ROC AUC on test set: {auc:.4f}")




# Calculate the confusion matrix
cm = confusion_matrix(y, binary_pred)

# Plotting the confusion matrix with Seaborn
plt.figure(figsize=(3, 2))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', cbar=False)
plt.xlabel('Predicted labels')
plt.ylabel('True labels')
plt.title('Confusion Matrix')
plt.show()
```

```
# Calculate precision, recall, and thresholds
precision, recall, thresholds = precision_recall_curve(y, prob_pred)

# Calculate F1 score for each threshold
f1_scores = 2 * (precision * recall) / (precision + recall)
# Remove NaN values that could arise from division by zero
f1_scores = f1_scores[~np.isnan(f1_scores)]

# Find the threshold where F1 score is maximum
optimal_idx = np.argmax(f1_scores)
optimal_threshold = thresholds[optimal_idx]
optimal_f1 = f1_scores[optimal_idx]

# Plotting the F1 curve
plt.plot(thresholds, f1_scores[:-1], label='F1 Curve')
plt.xlabel('Thresholds')
plt.ylabel('F1 Score')
plt.title(f'F1 Score vs. Thresholds {i}')
plt.axvline(x=optimal_threshold, color='r', linestyle='--', label=f'Optimal Thr
plt.legend()
plt.show()

# Print the optimum threshold with its F1 score
print(f"Optimum threshold: {optimal_threshold:.4f}, with F1 score: {optimal_f1:


final_predictions = [1 if prob > optimal_threshold else 0 for prob in prob_prec


from sklearn.metrics import recall_score

# Розрахувати Classification Report
print('Classification Report:')
print(classification_report(y, final_predictions))

# Розрахувати Confusion Matrix
print('Confusion Matrix 1:')
print(confusion_matrix(y, final_predictions))

# calc metrics
auc = roc_auc_score(y, prob_pred)
rec = recall_score(y, final_predictions)
f1 = f1_score(y, final_predictions)

print(f"Recall on test set with threshold: {rec:.4f}")
print(f"F1 Score on test set with threshold: {f1:.4f}")
print(f"ROC AUC on test set with threshold: {auc:.4f}")
```

```
# Calculate the confusion matrix
cm = confusion_matrix(y, final_predictions)

# Plotting the confusion matrix with Seaborn
plt.figure(figsize=(3, 2))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', cbar=False)
plt.xlabel('Predicted labels')
plt.ylabel('True labels')
plt.title('Confusion Matrix')
plt.show()
```

saving voting_clf model

```
import pickle

pickle.dump(voting_clf, open('voting_clf.pkl', 'wb'))
```

TEST DATA

```
churn_test_model_fe = pd.read_pickle('/content/drive/MyDrive/Churn_model_Vodafc
```

```
def preproccess_test_data(df):
  df.fillna(-1, inplace=True)
  df.drop(['abon_id'], inplace=True, axis=1)
  df_corr = df.corr(method='spearman')
  mask = np.ones(df_corr.columns.size) - np.eye(df_corr.columns.size)
  df_corr = mask * df_corr
  cols_to_drops = []
  for col in df_corr.columns.values:
      # if we've already determined to drop the current variable, continue
      if np.in1d([col],cols_to_drops):
          continue
      corr = df_corr[abs(df_corr[col]) > 0.98].index
      cols_to_drops = np.union1d(cols_to_drops, corr)
  cols_before_drop = df.columns
  df_pp = df.drop(columns=cols_to_drops)
  scaler = StandardScaler()
  train_scaled = scaler.fit_transform(df_pp)
  df_train_scaled = pd.DataFrame(train_scaled, index=df_pp.index, columns=df_pp
  return df_train_scaled
```

```
top_features
```

```
df_test = preproccess_test_data(churn_test_model_fe)
```

Start coding or generate with AI.

```
# Assuming model_predictions is a dictionary containing the probability predict
# and optimal_thresholds_r is a dictionary with the same keys and optimal thres

final_predictions = {}

for model_name, predictions in model_predictions.items():
    threshold = optimal_thresholds_r[model_name]
    final_predictions[model_name] = [1 if prob > threshold else 0 for prob in p
```