```
#Import + read df

import pandas as pd
import numpy as np
import re
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn import preprocessing
from scipy import stats
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import roc_auc_score
from sklearn.preprocessing import OrdinalEncoder
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import OneHotEncoder
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
from sklearn.metrics import roc_curve, auc


df = pd.read_csv('train.csv')
df_valid = pd.read_csv('valid.csv')

df
```

```
<ipython-input-1-7260918baaa0>:24: DtypeWarning: Columns (51,52,53,54,55,56,57,58,59,60,61,62,63,64,65,66,87,88,89,90,91,92,93,94,95
  df = pd.read_csv('train.csv')
<ipython-input-1-7260918baaa0>:25: DtypeWarning: Columns (51,52,53,54,55,56,57,58,59,60,61,62,63,64,65,66,87,88,89,91,95,96,97,99,13
  df_valid = pd.read_csv('valid.csv')
```

	report_date	client_id	target	col1	col2	col3	col4	col5	col6	col7	...	col2654	col2655	col2656	col2657	col2658
0	2022-11-01	1	0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN
1	2022-11-01	5	0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	7616803.0	7616803.0	7616803.0	NaN	NaN
2	2022-05-01	6	0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN
3	2022-09-01	7	0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN
4	2022-08-01	8	0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	..
14451	2022-07-01	1241	0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN
14452	2022-09-01	1969	1	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN
14453	2022-02-01	7116	1	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN
14454	2021-08-01	7117	0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN
14455	2022-03-01	849	0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN

14456 rows × 2666 columns

Чтобы изменить содержимое ячейки, дважды нажмите на нее (или выберите "Ввод")

```
#Проверка пропусков

plt.figure(figsize=(15, 15))  # Устанавливаем размер графика
sns.heatmap(df.isnull(), cmap='BuPu_r')


#Проверка процентов

missing_values = df.isnull().sum()
missing_values

missing_percent = df.isnull().mean() * 100
missing_percent


#Удаление дубликатов

df = df.drop_duplicates()
df_valid = df_valid.drop_duplicates()
df
```

```
#Удаление колонок где пропусков > 80

threshold = 0.8  # 80% пропусков
df = df.loc[:, df.isnull().mean() < threshold]
df_valid = df_valid.loc[:, df_valid.isnull().mean() < threshold]
df
```

```
#Проверка типов

df.info()
```

```
# Удаление строк с большим количеством пропусков
threshold = 0.5  # 50% пропусков
df = df.dropna(thresh=df.shape[1] * threshold, axis=0)
df_valid = df_valid.dropna(thresh=df_valid.shape[1] * threshold, axis=0)
df
```

```
#Алгоритм для удаления дубликатов

unique_data_columns = []
duplicated_data_columns = set()

for col in df.columns:
    if not any(df[col].equals(df[other_col]) for other_col in unique_data_columns):
        unique_data_columns.append(col)
    else:
        duplicated_data_columns.add(col)
df = df[unique_data_columns]
```

```
unique_data_columns = []
duplicated_data_columns = set()
for col in df_valid.columns:
    if not any(df_valid[col].equals(df_valid[other_col]) for other_col in unique_data_columns):
        unique_data_columns.append(col)
    else:
        duplicated_data_columns.add(col)

# Удаляем дублированные колонки

df_valid = df_valid[unique_data_columns]

# Вывод результата

df_valid
```

```
# удаление колонки с непонятным значением
column_to_drop = 'col1454'
df = df.drop(columns=column_to_drop)
df_valid = df_valid.drop(columns=column_to_drop)
df
```

```
df.info()
```

```
#смотрим что за колонка с объектом

object_columns = df.select_dtypes(include=['object']).columns
object_columns
```

```
# column_to_check = 'client_id'

# # Удаление строк с дублированными значениями в указанной колонке
# df = df.drop_duplicates(subset=[column_to_check], keep=False)
df
```

```
df['col1460'].unique()
```

```
#Проверка на процент пропусков
missing_percent = df.isnull().mean() * 100
missing_percent > 80
```

```
#Удаление одинаковых столбцов
df = df.loc[:, df.nunique() > 1]
```

```
df_valid = df_valid.loc[:, df_valid.nunique() > 1]
df


#Запеняем пустные и неномерные на Nan
df_valid = df_valid.apply(pd.to_numeric, errors='coerce')
df = df.apply(pd.to_numeric, errors='coerce')


#Проверка выбросов и вывод графика

def detect_outliers_iqr(column):
    Q1 = column.quantile(0.25)  # Первый квартиль (25%)
    Q3 = column.quantile(0.75)  # Третий квартиль (75%)
    IQR = Q3 - Q1  # Межквартильный размах
    lower_bound = Q1 - 1.5 * IQR  # Нижняя граница
    upper_bound = Q3 + 1.5 * IQR  # Верхняя граница
    # Выявление выбросов
    outliers = column[(column < lower_bound) | (column > upper_bound)]
    return outliers

# Применение функции к каждой колонке
outliers_dict = {}
for col in df.columns:
    outliers = detect_outliers_iqr(df[col])
    if not outliers.empty:
        outliers_dict[col] = outliers

outliers_dict = {}
for col in df_valid.columns:
    outliers = detect_outliers_iqr(df_valid[col])
    if not outliers.empty:
        outliers_dict[col] = outliers

# Вывод выбросов
# print("Выбросы в датасете:")
# for col, outliers in outliers_dict.items():
#     # print(f"Колонка '{col}': {outliers.tolist()}")

# Визуализация выбросов с использованием boxplot
plt.figure(figsize=(10, 6))
df.boxplot(grid=False)
plt.title('Boxplot для выявления выбросов')
plt.show()


# замена медианой если ненорм распределение, если норм, то средним

import scipy.stats as st

# Identify numerical columns and filter out columns with all NaNs
numerical_cols = df.select_dtypes(include=[np.number]).columns
numerical_cols = [col for col in numerical_cols if df[col].count() > 0]

# Handle string values in numerical columns
for col in numerical_cols:
    df[col] = pd.to_numeric(df[col], errors='coerce')

# Drop columns with all NaNs
df = df.dropna(axis=1, how='all')

# Iterate over each numerical column
for col in numerical_cols:
    if df[col].isnull().sum() > 0:
        data = df[col].dropna()
        if len(data) >= 3:
            stat, p = st.shapiro(data)
            alpha = 0.05
            if p > alpha:
                # Data is approximately normally distributed
                fill_value = data.mean()
                df[col].fillna(fill_value, inplace=True)

            else:
                # Data is not normally distributed
                q1 = data.quantile(0.25)
                q3 = data.quantile(0.75)
                iqr = q3 - q1
                lower_fence = q1 - 1.5 * iqr
                upper_fence = q3 + 1.5 * iqr
                outliers = data[(data < lower_fence) | (data > upper_fence)]
                skew = data.skew()
                if not outliers.empty and abs(skew) > 0.5:
```

```
                    # Data has outliers and is asymmetric
                    fill_value = data.median()
                    df[col].fillna(fill_value, inplace=True)

                else:
                    # Data is not normal but may be symmetric without significant outliers
                    fill_value = data.mean()
                    df[col].fillna(fill_value, inplace=True)

        else:
            # Not enough data to perform Shapiro-Wilk test
            # Fill missing values with median
            fill_value = data.median()
            df[col].fillna(fill_value, inplace=True)


df
```

```
numerical_cols = df_valid.select_dtypes(include=[np.number]).columns
numerical_cols = [col for col in numerical_cols if df_valid[col].count() > 0]

for col in numerical_cols:
    df_valid[col] = pd.to_numeric(df_valid[col], errors='coerce')

df_valid = df_valid.dropna(axis=1, how='all')

for col in numerical_cols:
    if df_valid[col].isnull().sum() > 0:
        data = df_valid[col].dropna()
        if len(data) >= 3:
            stat, p = st.shapiro(data)
            alpha = 0.05
            if p > alpha:

                fill_value = data.mean()
                df_valid[col].fillna(fill_value, inplace=True)

            else:

                q1 = data.quantile(0.25)
                q3 = data.quantile(0.75)
                iqr = q3 - q1
                lower_fence = q1 - 1.5 * iqr
                upper_fence = q3 + 1.5 * iqr
                outliers = data[(data < lower_fence) | (data > upper_fence)]
                skew = data.skew()
                if not outliers.empty and abs(skew) > 0.5:
                    fill_value = data.median()
                    df_valid[col].fillna(fill_value, inplace=True)

                else:

                    fill_value = data.mean()
                    df_valid[col].fillna(fill_value, inplace=True)

        else:

            fill_value = data.median()
            df_valid[col].fillna(fill_value, inplace=True)
df_valid
```

```
# df.describe()
df_valid.describe()
```

```
common_columns = df.columns.intersection(df_valid.columns)

# Выбираем только общие колонки в обоих датафреймах
df = df[common_columns]
df_valid = df_valid[common_columns]

df
```

```
#Проверка пропусков

null_columns = df.columns[df.isnull().any()].tolist()
if null_columns:
```

```
    print(f"Столбцы с пропусками: {null_columns}")
else:
    print("Пропусков нет ни в одном столбце.")


#Вывод хитмапы

plt.figure(figsize=(12, 8))
sns.heatmap(df.isnull(), cmap='plasma')


# дропаем таргет

X = df.drop('target', axis=1)
y = df['target']

x1 = df_valid.drop('target', axis = 1)
y1 = df_valid['target']

df


# X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42) #разделие на тестовую и обучающую выборку


# model = RandomForestClassifier()
# model.fit(X_train, y_train)


# y_probs = model.predict_proba(X_test)[:, 1]  # Вероятности для положительного класса

# # Оценка AUC-ROC
# auc_roc = roc_auc_score(y_test, y_probs)
# print(f'AUC-ROC: {auc_roc}')


# from sklearn.model_selection import GridSearchCV
# param_grid = {
#     'n_estimators': [50, 100, 150],
#     'max_depth': [None, 5, 10, 15]
# }

# grid_search = GridSearchCV(estimator=model, param_grid=param_grid, cv=5, scoring='accuracy')

# # Обучаем модель с GridSearchCV
# grid_search.fit(X_train, y_train)

# # Выводим лучшие параметры и лучший результат
# print("Лучшие параметры:", grid_search.best_params_)
# print("Лучший результат:", grid_search.best_score_)


# better_model = RandomForestClassifier(**grid_search.best_params_)
# better_model.fit(X_train, y_train)

# y_pred = model.predict(X_test)
# y_prob = model.predict_proba(X_test)[:, 1]
# # Оценка AUC-ROC
# auc_roc = roc_auc_score(y_test, y_probs)

# print(f'AUC-ROC: {auc_roc}')
# accuracy = accuracy_score(y_test, y_pred)
# print("Точность:", accuracy)

# print(classification_report(y_test, y_pred))
# conf_matrix = confusion_matrix(y_test, y_pred)


# y.value_counts()


# plt.figure(figsize=(8, 6))
# sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Greens', xticklabels=['Не целевая', 'Целевая'], yticklabels=['Не целевая', 'Целева
# plt.ylabel('Фактический класс')
# plt.xlabel('Предсказанный класс')
# plt.title('Матрица ошибок')
# plt.show()

# from imblearn.over_sampling import SMOTE


# smote = SMOTE(random_state=42)
# X_balanced, y_balanced = smote.fit_resample(X, y)
```

```
# X_train, X_test, y_train, y_test = train_test_split(X_balanced, y_balanced, test_size=0.3, random_state=42)

# #обучение
# model = RandomForestClassifier(random_state=42)
# model.fit(X_train, y_train)

# # Предсказание вероятностей для тестовой выборки
# y_pred_proba = model.predict_proba(X_test)[:, 1]

# # Вычисление ROC-AUC
# fpr, tpr, _ = roc_curve(y_test, y_pred_proba)
# roc_auc = auc(fpr, tpr)


# y_pred_proba = model.predict_proba(X_test)[:, 1]

# # Вычисление ROC-AUC
# fpr, tpr, _ = roc_curve(y_test, y_pred_proba)
# roc_auc = auc(fpr, tpr)

# # Построение ROC-AUC кривой
# plt.figure()
# plt.plot(fpr, tpr, color='darkorange', lw=2, label=f'ROC curve (area = {roc_auc:.2f})')
# plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
# plt.xlabel('False Positive Rate')
# plt.ylabel('True Positive Rate')
# plt.title('Receiver Operating Characteristic (ROC) Curve')
# plt.legend(loc="lower right")
# plt.show()


# fpr, tpr, _ = roc_curve(y_test, y_prob)
# roc_auc = auc(fpr, tpr)
# plt.figure(figsize=(8, 6))
# plt.plot(fpr, tpr, color='green', lw=2, label='ROC-кривая (AUC = {:.2f})'.format(roc_auc))
# plt.plot([0, 1], [0, 1], color='red', lw=2, linestyle='--')
# plt.xlim([0.0, 1.0])
# plt.ylim([0.0, 1.05])
# plt.xlabel('Ложноположительный процент')
# plt.ylabel('Истинноположительный процент')
# plt.title('ROC-кривая')
# plt.legend(loc='lower right')
# plt.show()


# from imblearn.under_sampling import RandomUnderSampler #Случайное понижение выборки

# # Инициализация объекта
# undersampler = RandomUnderSampler(random_state=42)

# # Применение понижения выборки
# X_train_resampled, y_train_resampled = undersampler.fit_resample(X_train, y_train)

# model = RandomForestClassifier(random_state=42)
# model.fit(X_train_resampled, y_train_resampled)

# # Предсказание на тестовой выборке
# y_pred = model.predict(X_test)

# # Оценка производительности модели
# print("Accuracy:", accuracy_score(y_test, y_pred))
# print("Classification Report:")
# print(classification_report(y_test, y_pred))


class_1 = df[df['target'] == 1]  # Все строки с target = 1
class_0 = df[df['target'] == 0]  # Все строки с target = 0

# Выбор случайных 250 строк из класса 0
class_0_sampled = class_0.sample(n=250, random_state=42)  # random_state для воспроизводимости

# Объединение классов 1 и выбранных строк класса 0
balanced_data = pd.concat([class_1, class_0_sampled])

# Перемешивание данных, чтобы строки не шли подряд
balanced_data = balanced_data.sample(frac=1, random_state=42).reset_index(drop=True)

balanced_data




# Разделение данных на признаки (X) и целевую переменную (y)
```

```
X = balanced_data.drop('target', axis=1)  # Признаки
y = balanced_data['target']  # Целевая переменная

# Разделение данных на обучающую и тестовую выборки
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

# Обучение модели RandomForestClassifier
model1 = RandomForestClassifier(random_state=42)
model1.fit(X_train, y_train)

# Предсказание вероятностей для тестовой выборки
y_pred_proba = model1.predict_proba(X_test)[:, 1]

# Вычисление ROC-AUC
fpr, tpr, _ = roc_curve(y_test, y_pred_proba)
roc_auc = auc(fpr, tpr)

# Построение ROC-AUC кривой
plt.figure(figsize=(8, 6))
plt.plot(fpr, tpr, color='darkorange', lw=2, label=f'ROC curve (area = {roc_auc:.2f})')
plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic (ROC) Curve')
plt.legend(loc="lower right")
plt.show()
```

```
from sklearn.metrics import (
    roc_curve, auc, accuracy_score, precision_score, recall_score, f1_score, confusion_matrix
)

# Предсказание для тестовой выборки
y_pred = model1.predict(X_test)
y_pred_proba = model1.predict_proba(X_test)[:, 1]

# Вычисление метрик производительности
accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred)
recall = recall_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred)
conf_matrix = confusion_matrix(y_test, y_pred)

# Вывод метрик
# print(f"Accuracy: {accuracy:.4f}")
# print(f"Precision: {precision:.4f}")
# print(f"Recall: {recall:.4f}")
# print(f"F1-score: {f1:.4f}")

print("Accuracy:", accuracy_score(y_test, y_pred))
print("Classification Report:")
print(classification_report(y_test, y_pred))
```

```
conf_matrix
```

```
from sklearn.tree import DecisionTreeClassifier

model2 = DecisionTreeClassifier(random_state=42)
model2.fit(X_train, y_train)
y_prob = model2.predict_proba(X_test)[:, 1]

fpr, tpr, _ = roc_curve(y_test, y_prob)
roc_auc = auc(fpr, tpr)
plt.figure(figsize=(8, 6))
plt.plot(fpr, tpr, color='turquoise', lw=2, label='ROC-кривая (AUC = {:.2f})'.format(roc_auc))
plt.plot([0, 1], [0, 1], color='purple', lw=2, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('Ложноположительный процент')
plt.ylabel('Истинноположительный процент')
plt.title('ROC-кривая')
plt.legend(loc='lower right')
plt.show()

print("Accuracy:", accuracy_score(y_test, y_pred))
print("Classification Report:")
print(classification_report(y_test, y_pred))
```

```
from sklearn.ensemble import GradientBoostingClassifier
```

```
model3 =  GradientBoostingClassifier(random_state=42)
model3.fit(X_train, y_train)
y_prob = model3.predict_proba(X_test)[:, 1]

fpr, tpr, _ = roc_curve(y_test, y_prob)
roc_auc = auc(fpr, tpr)
plt.figure(figsize=(8, 6))
plt.plot(fpr, tpr, color='red', lw=2, label='ROC-кривая (AUC = {:.2f})'.format(roc_auc))
plt.plot([0, 1], [0, 1], color='purple', lw=2, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('Ложноположительный процент')
plt.ylabel('Истинноположительный процент')
plt.title('ROC-кривая')
plt.legend(loc='lower right')
plt.show()

print("Accuracy:", accuracy_score(y_test, y_pred))
print("Classification Report:")
print(classification_report(y_test, y_pred))



df_valid


df


X_valid = df_valid.drop('target', axis=1)
y_valid = df_valid['target']

# Применить модель model1 к данным
predictions = model1.predict(X_valid)


predictions_df = pd.DataFrame(predictions, columns=['predicted_value'])

# Добавить столбец client_id из df_valid
if 'client_id' in df_valid.columns:
    predictions_df['client_id'] = df_valid['client_id'].values
else:
    print("Внимание: столбец 'client_id' отсутствует в df_valid.")

# Сохранить предсказания в новый CSV-файл
predictions_df.to_csv('predictions.csv', index=False)

# Создать DataFrame с предсказаниями
predictions_df = pd.DataFrame(predictions, columns=['predicted_value'])

print("Accuracy:", accuracy_score(y_valid, predictions))
print("Precision:", precision_score(y_valid, predictions, average='weighted'))
print("Recall:", recall_score(y_valid, predictions, average='weighted'))
print("F1 Score:", f1_score(y_valid, predictions, average='weighted'))

# Сохранить предсказания в новый CSV-файл
predictions_df.to_csv('submissions_valid.csv', index=False)
```