```
!pip install optuna
```

```
Collecting optuna
  Downloading optuna-3.6.1-py3-none-any.whl.metadata (17 kB)
Collecting alembic>=1.5.0 (from optuna)
  Downloading alembic-1.13.2-py3-none-any.whl.metadata (7.4 kB)
Collecting colorlog (from optuna)
  Downloading colorlog-6.8.2-py3-none-any.whl.metadata (10 kB)
Requirement already satisfied: numpy in /usr/local/lib/python3.10/dist-packages (from optuna) (1.25.2)
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.10/dist-packages (from optuna) (24.1)
Requirement already satisfied: sqlalchemy>=1.3.0 in /usr/local/lib/python3.10/dist-packages (from optuna) (2.0.31)
Requirement already satisfied: tqdm in /usr/local/lib/python3.10/dist-packages (from optuna) (4.66.4)
Requirement already satisfied: PyYAML in /usr/local/lib/python3.10/dist-packages (from optuna) (6.0.1)
Collecting Mako (from alembic>=1.5.0->optuna)
  Downloading Mako-1.3.5-py3-none-any.whl.metadata (2.9 kB)
Requirement already satisfied: typing-extensions>=4 in /usr/local/lib/python3.10/dist-packages (from alembic>=1.5.0->optuna) (4.12.2)
Requirement already satisfied: greenlet!=0.4.17 in /usr/local/lib/python3.10/dist-packages (from sqlalchemy>=1.3.0->optuna) (3.0.3)
Requirement already satisfied: MarkupSafe>=0.9.2 in /usr/local/lib/python3.10/dist-packages (from Mako->alembic>=1.5.0->optuna) (2.1.5)
Downloading optuna-3.6.1-py3-none-any.whl (380 kB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 380.1/380.1 kB 6.1 MB/s eta 0:00:00
Downloading alembic-1.13.2-py3-none-any.whl (232 kB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 233.0/233.0 kB 11.3 MB/s eta 0:00:00
Downloading colorlog-6.8.2-py3-none-any.whl (11 kB)
Downloading Mako-1.3.5-py3-none-any.whl (78 kB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 78.6/78.6 kB 3.9 MB/s eta 0:00:00
Installing collected packages: Mako, colorlog, alembic, optuna
Successfully installed Mako-1.3.5 alembic-1.13.2 colorlog-6.8.2 optuna-3.6.1
```

```
import os
import sys
from tempfile import NamedTemporaryFile
from urllib.request import urlopen
from urllib.parse import unquote, urlparse
from urllib.error import HTTPError
from zipfile import ZipFile
import tarfile
import shutil

CHUNK_SIZE = 40960
DATA_SOURCE_MAPPING = 'german-credit-scoring-data:https%3A%2F%2Fstorage.googleapis.com%2Fkaggle-data-sets%2F4316755%2F7419868%2Fbundle%2Farc

KAGGLE_INPUT_PATH='/kaggle/input'
KAGGLE_WORKING_PATH='/kaggle/working'
KAGGLE_SYMLINK='kaggle'

!umount /kaggle/input/ 2> /dev/null
shutil.rmtree('/kaggle/input', ignore_errors=True)
os.makedirs(KAGGLE_INPUT_PATH, 0o777, exist_ok=True)
os.makedirs(KAGGLE_WORKING_PATH, 0o777, exist_ok=True)

try:
  os.symlink(KAGGLE_INPUT_PATH, os.path.join("..", 'input'), target_is_directory=True)
except FileExistsError:
  pass
try:
  os.symlink(KAGGLE_WORKING_PATH, os.path.join("..", 'working'), target_is_directory=True)
except FileExistsError:
  pass

for data_source_mapping in DATA_SOURCE_MAPPING.split(','):
    directory, download_url_encoded = data_source_mapping.split(':')
    download_url = unquote(download_url_encoded)
    filename = urlparse(download_url).path
    destination_path = os.path.join(KAGGLE_INPUT_PATH, directory)
    try:
        with urlopen(download_url) as fileres, NamedTemporaryFile() as tfile:
            total_length = fileres.headers['content-length']
            print(f'Downloading {directory}, {total_length} bytes compressed')
            dl = 0
            data = fileres.read(CHUNK_SIZE)
            while len(data) > 0:
                dl += len(data)
                tfile.write(data)
                done = int(50 * dl / int(total_length))
                sys.stdout.write(f"\r[{'=' * done}{' ' * (50-done)}] {dl} bytes downloaded")
                sys.stdout.flush()
                data = fileres.read(CHUNK_SIZE)
            if filename.endswith('.zip'):
              with ZipFile(tfile) as zfile:
                zfile.extractall(destination_path)
            else:
              with tarfile.open(tfile.name) as tarfile:
                tarfile.extractall(destination_path)
            print(f'\nDownloaded and uncompressed: {directory}')
    except HTTPError as e:
        print(f'Failed to load (likely expired) {download_url} to path {destination_path}')
        continue
    except OSError as e:
        print(f'Failed to load {download_url} to path {destination_path}')
        continue

print('Data source import complete.')
```

```
Downloading german-credit-scoring-data, 17947 bytes compressed
[==================================================] 17947 bytes downloaded
Downloaded and uncompressed: german-credit-scoring-data
Data source import complete.
```

```
import numpy as np
import optuna

import pandas as pd
import matplotlib.pyplot as plt

from sklearn.model_selection import train_test_split
from sklearn.pipeline import Pipeline
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import OneHotEncoder, StandardScaler, RobustScaler,\
                                  PowerTransformer, TargetEncoder, MinMaxScaler
from sklearn.compose import ColumnTransformer

from sklearn.feature_selection import SelectFromModel, SelectFwe, f_classif

from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import cross_val_score
import sklearn.metrics as met

from sklearn.base import BaseEstimator, TransformerMixin

import warnings
warnings.filterwarnings("ignore")
pd.set_option('display.max_columns', None)

import sklearn
sklearn.set_config(transform_output="pandas")

GLOBAL_SEED = 1
```

```
df = pd.read_csv('/kaggle/input/german-credit-scoring-data/german_credit_cleaned.csv')
```

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1000 entries, 0 to 999
Data columns (total 21 columns):
 #   Column                    Non-Null Count  Dtype 
---  ------                    --------------  ----- 
 0   checking_acc_status       1000 non-null   object
 1   duration                  1000 non-null   int64 
 2   cred_hist                 1000 non-null   object
 3   purpose                   1000 non-null   object
 4   loan_amt                  1000 non-null   int64 
 5   saving_acc_bonds          1000 non-null   object
 6   present_employment_since  1000 non-null   object
 7   installment_rate          1000 non-null   int64 
 8   personal_stat_gender      1000 non-null   object
 9   other_debtors_guarantors  1000 non-null   object
 10  present_residence_since   1000 non-null   int64 
 11  property                  1000 non-null   object
 12  age                       1000 non-null   int64 
 13  other_installment_plans   1000 non-null   object
 14  housing                   1000 non-null   object
 15  num_curr_loans            1000 non-null   int64 
 16  job                       1000 non-null   object
 17  num_people_provide_maint  1000 non-null   int64 
 18  telephone                 1000 non-null   object
 19  is_foreign_worker         1000 non-null   object
 20  target                    1000 non-null   object
dtypes: int64(7), object(14)
memory usage: 164.2+ KB
```

All columns are currently assigned the 'object' data type, which occupies more memory space. Numeric columns should be represented as integers or floats, and categorical columns as 'category' types. This optimization would result in more efficient memory storage

```
numeric_features = ['duration', 'loan_amt', 'installment_rate', 'present_residence_since', 'age', 'num_curr_loans',
                    'num_people_provide_maint']
categorical_features = ['checking_acc_status', 'cred_hist', 'purpose', 'saving_acc_bonds', 'present_employment_since',
                        'personal_stat_gender', 'other_debtors_guarantors', 'property', 'other_installment_plans',
                        'housing', 'job', 'telephone', 'is_foreign_worker']

df[numeric_features] = df[numeric_features].apply(pd.to_numeric, errors='coerce', downcast='integer')
df[categorical_features] = df[categorical_features].astype('category')
df['target'] = np.where(df['target']=='good', 1, 0).astype('int8')

df.info()
```

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1000 entries, 0 to 999
Data columns (total 21 columns):
 #   Column                    Non-Null Count  Dtype   
---  ------                    --------------  -----   
 0   checking_acc_status       1000 non-null   category
 1   duration                  1000 non-null   int8    
 2   cred_hist                 1000 non-null   category
 3   purpose                   1000 non-null   category
 4   loan_amt                  1000 non-null   int16   
 5   saving_acc_bonds          1000 non-null   category
 6   present_employment_since  1000 non-null   category
 7   installment_rate          1000 non-null   int8    
 8   personal_stat_gender      1000 non-null   category
 9   other_debtors_guarantors  1000 non-null   category
 10  present_residence_since   1000 non-null   int8    
 11  property                  1000 non-null   category
 12  age                       1000 non-null   int8    
 13  other_installment_plans   1000 non-null   category
 14  housing                   1000 non-null   category
 15  num_curr_loans            1000 non-null   int8    
 16  job                       1000 non-null   category
 17  num_people_provide_maint  1000 non-null   int8    
 18  telephone                 1000 non-null   category
 19  is_foreign_worker         1000 non-null   category
 20  target                    1000 non-null   int8    
dtypes: category(13), int16(1), int8(7)
memory usage: 24.0 KB
```

```
for cat in categorical_features:
    print(df[cat].value_counts(1))
    print('-----------------')
```

```
-----------------
present_employment_since
below_4y      0.339
above_7y      0.253
below_7y      0.174
below_1y      0.172
unemployed    0.062
Name: proportion, dtype: float64
-----------------
personal_stat_gender
male:single                  0.548
female:divorced_or_married   0.310
male:married_or_widowed      0.092
male:divorced                0.050
```

```
no     0.596
yes    0.404
Name: proportion, dtype: float64
-----------------
is_foreign_worker
yes    0.963
no     0.037
Name: proportion, dtype: float64
-----------------
```

```
df.target.value_counts(1)
```

```
target
1    0.7
0    0.3
Name: proportion, dtype: float64
```

```
X = df[numeric_features+categorical_features] #Independent - predictor variables
y = df['target'] # Dependent variable

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25, random_state=GLOBAL_SEED, stratify=y)
```

Train-test split is done to have unseen data in order to check real performance of the model. test_size = 0.25 depicts 25% of dataset will be used as test data, stratify=y ensures proportion of good and bads equal be same in train and test sets

Baseline Model

```
#Encode categorical features
te = TargetEncoder(random_state=GLOBAL_SEED)
X_train_encoded = te.fit_transform(X_train[categorical_features],y_train)
X_test_encoded = te.transform(X_test[categorical_features])#attention
X_train_preprocessed = pd.concat([X_train[numeric_features], X_train_encoded], axis=1)
X_test_preprocessed = pd.concat([X_test[numeric_features], X_test_encoded], axis=1)

model = LogisticRegression(penalty=None, random_state=GLOBAL_SEED)
model.fit(X_train_preprocessed, y_train)

# Predict on the test data
y_train_pred_proba = model.predict_proba(X_train_preprocessed)[:, 1]
y_pred_proba = model.predict_proba(X_test_preprocessed)[:, 1]

# Evaluate the model
train_auc = met.roc_auc_score(y_train, y_train_pred_proba)
test_auc = met.roc_auc_score(y_test, y_pred_proba)
print(f"AUC-ROC on train data: {round(train_auc, 3)}")
print(f"AUC-ROC on test data: {round(test_auc, 3)}")

print(f"Gini score: {round(test_auc*2-1, 3)}")
```

```
AUC-ROC on train data: 0.712
AUC-ROC on test data: 0.705
Gini score: 0.41
```

```
cats = np.concatenate(te.categories_).reshape(-1,1)
ends = np.concatenate(te.encodings_).reshape(-1,1)

features = []
for feature in categorical_features:
    n_cat = df[feature].nunique()
    [features.append(feature) for i in range(n_cat)]

comb = np.concatenate((cats, ends), axis = 1)
te_df = pd.DataFrame(comb, index=features).reset_index()
te_df.columns = ['feature','category', 'encoding_value']
te_df.head()
```

| | feature | category | encoding_value |
|---|---|---|---|
| 0 | checking_acc_status | above:200 | 0.783003 |
| 1 | checking_acc_status | below_0 | 0.517783 |
| 2 | checking_acc_status | below_200 | 0.611332 |
| 3 | checking_acc_status | no_cheking_acc | 0.881219 |
| 4 | cred_hist | curr_loans_paid_duly | 0.676906 |

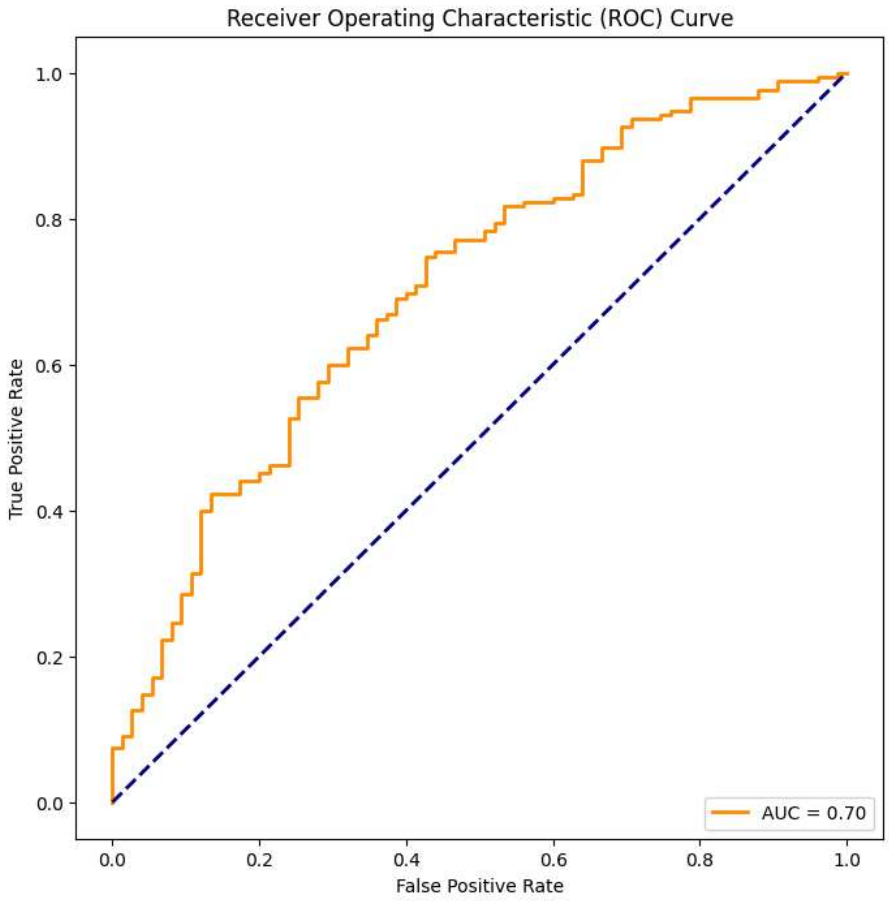
Next steps: Generate code with te_df | View recommended plots | New interactive sheet

Encoding values for purpose and employement

```
te_df[te_df.feature.isin(['present_employment_since', 'purpose'])]
```

| | feature | category | encoding_value |
|---|---|---|---|
| 9 | purpose | business | 0.62774 |
| 10 | purpose | car_new | 0.626822 |
| 11 | purpose | car_used | 0.868949 |
| 12 | purpose | domestic_applience | 0.610256 |
| 13 | purpose | education | 0.580854 |
| 14 | purpose | furniture_equipment | 0.676873 |
| 15 | purpose | others | 0.572246 |
| 16 | purpose | radio_tv | 0.777456 |
| 17 | purpose | repairs | 0.668863 |
| 18 | purpose | retraining | 1.0 |
| 24 | present_employment_since | above_7y | 0.741904 |
| 25 | present_employment_since | below_1y | 0.600839 |
| 26 | present_employment_since | below_4y | 0.705406 |
| 27 | present_employment_since | below_7y | 0.784563 |
| 28 | present_employment_since | unemployed | 0.568561 |

```
# Compute ROC curve and AUC
fpr, tpr, thresholds = met.roc_curve(y_test, y_pred_proba)
roc_auc = met.auc(fpr, tpr)

# Plot ROC curve
plt.figure(figsize=(8, 8))
plt.plot(fpr, tpr, color='darkorange', lw=2, label=f'AUC = {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()
```




```
numeric_features = X.select_dtypes(exclude='category').columns
purpose = ['purpose']
other_categorical_features = X.drop('purpose', axis=1).select_dtypes('category').columns
```

Hyperparameter tuning with Optuna

```
class LogOddsEncoder(BaseEstimator, TransformerMixin):
    def __init__(self, smoothing_factor=1):
        self.feature_log_odds = {}
        self.smoothing_factor = smoothing_factor

    def fit(self, X, y=None):
        for column in X.columns:
            # Calculate log odds for each category in the column
            Xy = pd.concat([X,y], axis=1)
            events = Xy.groupby(column)['target'].mean()
            category_log_odds = np.log((events + self.smoothing_factor) /
                                       (Xy['target'].mean() + 2 * self.smoothing_factor))

            self.feature_log_odds[column] = category_log_odds.to_dict()

        return self

    def transform(self, X, y=None):
        X_encoded = X.copy()
        for column in X.columns:
            # Replace original categories with log odds in the transformed data
            X_encoded[column] = X[column].map(self.feature_log_odds[column])
        return X_encoded

    def get_feature_names_out():
        pass
```

Tuned parameters

```
def objective(trial):

    """ 'objective'that takes hyperparameters as input and returns a score to be minimized """

    params = {
        'class_weight':'balanced',
        'C': trial.suggest_float("C", 1e-3, 1000),
        'l1_ratio': trial.suggest_float("l1_ratio", 0, 1),
        'penalty':'elasticnet',
        'solver': 'saga',
        'random_state': GLOBAL_SEED
    }

    min_frequency = trial.suggest_categorical("min_frequency", [0.01, 0.012, 0.015, 0.020, 0.052])
    smoothing = trial.suggest_float("smoothing", 1e-4, 10) # for
    multiplier = trial.suggest_float("multiplier", 0, 1) # for SelectFromModel

    scaler_options = {
        'StandardScaler': StandardScaler(),
        'RobustScaler': RobustScaler(),
        'MinMaxScaler': MinMaxScaler(feature_range=(-1,1))
    }

    cat_encoding_options = {
        'ohe': OneHotEncoder(sparse_output=False, handle_unknown='ignore', drop='if_binary'),
        'target_encoder': TargetEncoder(random_state=GLOBAL_SEED),
        'lg_encoder': LogOddsEncoder(smoothing)
    }

    purpose_encoding_options = {
        'ohe': OneHotEncoder(min_frequency=min_frequency, sparse_output=False, handle_unknown='ignore'),
        'target_encoder': TargetEncoder(random_state=GLOBAL_SEED),
        'lg_encoder': LogOddsEncoder(smoothing)
    }

    selected_scaler = trial.suggest_categorical("scaler", ['StandardScaler','RobustScaler', 'MinMaxScaler'])
    selected_cat_encoder = trial.suggest_categorical('cat_encoder', ['ohe', 'target_encoder', 'lg_encoder'])
    selected_purpose_encoder = trial.suggest_categorical('purpose_encoder', ['target_encoder', 'lg_encoder'])

    model = LogisticRegression(**params)

    num_p = Pipeline(steps=[('imputer', SimpleImputer(strategy='constant', fill_value=0)),
                                        ('scaler', scaler_options[selected_scaler])])

    if selected_cat_encoder == 'ohe':

        other_cat_p = Pipeline(steps=[('imputer', SimpleImputer(strategy='constant', fill_value='not_given')),
                ('encoder', cat_encoding_options[selected_cat_encoder])])
    else:

        other_cat_p = Pipeline(steps=[('imputer', SimpleImputer(strategy='constant', fill_value='not_given')),
                        ('encoder', cat_encoding_options[selected_cat_encoder]),
                                    ('scaler', scaler_options[selected_scaler])])

    if selected_purpose_encoder == 'ohe':

        purpose_p = Pipeline(steps=[('imputer', SimpleImputer(strategy='constant', fill_value='not_given')),
                        ('encoder', purpose_encoding_options[selected_purpose_encoder])])
    else:

        purpose_p = Pipeline(steps=[('imputer', SimpleImputer(strategy='constant', fill_value='not_given')),
                        ('encoder', purpose_encoding_options[selected_purpose_encoder]),
                                  ('scaler', scaler_options[selected_scaler])])

    transformer = ColumnTransformer(transformers=[
        ('num_p', num_p, numeric_features),
        ('other_cat_p', other_cat_p, other_categorical_features),
        ('purpose_p', purpose_p, purpose)
    ])#.set_output(transform='pandas')


    # Full Pipeline
    pipeline = Pipeline(steps=[
        ('transformer', transformer),
        ('selector', SelectFromModel(model,threshold=f'{multiplier}*median', max_features=None)),
        ('model', model)
    ])#.set_output(transform='pandas')
```

```
    means = cross_val_score(estimator=pipeline,X=X_train,y=y_train, cv=10, scoring='roc_auc')

    return np.mean(means)
```

```
def tune(objective):
    study = optuna.create_study(direction="maximize", sampler=optuna.samplers.TPESampler(
                            seed=GLOBAL_SEED))
    study.optimize(objective, n_trials=100, show_progress_bar=True)

    params = study.best_params
    best_score = study.best_value
    print(f"Best score: {best_score} \nOptimized parameters: {params}")
    return study
```

```
study = tune(objective)
```

```
[I 2024-07-27 09:07:04,618] A new study created in memory with name: no-name-0b4fd711-ec
Best trial: 71. Best value: 0.779739: 100%                    100/100 [02:19<00:00,  1.45s/it]
[I 2024-07-27 09:07:05,660] Trial 0 finished with value: 0.767593378729556 and parameter
[I 2024-07-27 09:07:06,897] Trial 1 finished with value: 0.7658264827871062 and paramete
[I 2024-07-27 09:07:08,945] Trial 2 finished with value: 0.7604900841570241 and paramete
[I 2024-07-27 09:07:10,850] Trial 3 finished with value: 0.7783236861579765 and paramete
[I 2024-07-27 09:07:12,896] Trial 4 finished with value: 0.7565796795493268 and paramete
[I 2024-07-27 09:07:14,158] Trial 5 finished with value: 0.7739242299947795 and paramete
[I 2024-07-27 09:07:15,547] Trial 6 finished with value: 0.7476532696179948 and paramete
[I 2024-07-27 09:07:16,848] Trial 7 finished with value: 0.7653162055335969 and paramete
[I 2024-07-27 09:07:18,166] Trial 8 finished with value: 0.7608245323175593 and paramete
[I 2024-07-27 09:07:19,735] Trial 9 finished with value: 0.7580180074232576 and paramete
[I 2024-07-27 09:07:20,960] Trial 10 finished with value: 0.775037001554639 and paramete
[I 2024-07-27 09:07:22,478] Trial 11 finished with value: 0.775037001554639 and paramete
[I 2024-07-27 09:07:24,313] Trial 12 finished with value: 0.775037001554639 and paramete
[I 2024-07-27 09:07:25,584] Trial 13 finished with value: 0.7713085929656891 and paramet
[I 2024-07-27 09:07:26,803] Trial 14 finished with value: 0.7740444413338917 and paramete
[I 2024-07-27 09:07:28,032] Trial 15 finished with value: 0.7780212199613348 and paramet
[I 2024-07-27 09:07:29,301] Trial 16 finished with value: 0.7736970576593218 and paramet
[I 2024-07-27 09:07:30,300] Trial 17 finished with value: 0.7698014249901043 and paramet
[I 2024-07-27 09:07:31,542] Trial 18 finished with value: 0.7606914414543618 and paramet
[I 2024-07-27 09:07:32,817] Trial 19 finished with value: 0.7779892379974415 and paramet
[I 2024-07-27 09:07:33,826] Trial 20 finished with value: 0.7670526970978161 and paramet
[I 2024-07-27 09:07:35,396] Trial 21 finished with value: 0.7776397310646695 and paramet
[I 2024-07-27 09:07:37,248] Trial 22 finished with value: 0.7795649592409232 and paramet
[I 2024-07-27 09:07:38,650] Trial 23 finished with value: 0.7784072981981104 and paramet
[I 2024-07-27 09:07:39,879] Trial 24 finished with value: 0.7777233431048034 and paramet
[I 2024-07-27 09:07:41,478] Trial 25 finished with value: 0.7540559727393197 and paramet
[I 2024-07-27 09:07:42,703] Trial 26 finished with value: 0.7598297928486608 and paramet
[I 2024-07-27 09:07:43,974] Trial 27 finished with value: 0.7750714216054659 and paramet
[I 2024-07-27 09:07:45,170] Trial 28 finished with value: 0.7623750121904348 and paramet
[I 2024-07-27 09:07:46,496] Trial 29 finished with value: 0.775495935565665 and paramete
[I 2024-07-27 09:07:47,822] Trial 30 finished with value: 0.753397545850376 and paramete
[I 2024-07-27 09:07:49,670] Trial 31 finished with value: 0.7778091063981137 and paramet
[I 2024-07-27 09:07:51,398] Trial 32 finished with value: 0.7774617507185185 and paramet
[I 2024-07-27 09:07:52,657] Trial 33 finished with value: 0.7779956917569715 and paramet
[I 2024-07-27 09:07:53,918] Trial 34 finished with value: 0.7658264827871062 and paramet
[I 2024-07-27 09:07:55,244] Trial 35 finished with value: 0.7778948696914242 and paramet
[I 2024-07-27 09:07:56,525] Trial 36 finished with value: 0.7729958925406013 and paramet
[I 2024-07-27 09:07:57,905] Trial 37 finished with value: 0.7625954439326055 and paramet
[I 2024-07-27 09:07:59,609] Trial 38 finished with value: 0.7459047310359862 and paramet
[I 2024-07-27 09:08:00,934] Trial 39 finished with value: 0.7639848666509865 and paramet
[I 2024-07-27 09:08:02,917] Trial 40 finished with value: 0.7780212199613348 and paramet
[I 2024-07-27 09:08:04,454] Trial 41 finished with value: 0.7750947985566525 and paramet
[I 2024-07-27 09:08:05,759] Trial 42 finished with value: 0.7781048320014686 and paramet
[I 2024-07-27 09:08:07,026] Trial 43 finished with value: 0.774674587102807 and paramete
[I 2024-07-27 09:08:08,285] Trial 44 finished with value: 0.779189637270031 and paramete
[I 2024-07-27 09:08:09,506] Trial 45 finished with value: 0.7674987235897819 and paramet
[I 2024-07-27 09:08:11,208] Trial 46 finished with value: 0.7655710573265948 and paramet
[I 2024-07-27 09:08:12,474] Trial 47 finished with value: 0.7619071863329452 and paramet
[I 2024-07-27 09:08:13,477] Trial 48 finished with value: 0.7649562291686984 and paramet
[I 2024-07-27 09:08:15,294] Trial 49 finished with value: 0.775146428632893 and paramete
[I 2024-07-27 09:08:17,095] Trial 50 finished with value: 0.7732919049777129 and paramet
[I 2024-07-27 09:08:18,335] Trial 51 finished with value: 0.7780233712145115 and paramet
[I 2024-07-27 09:08:19,580] Trial 52 finished with value: 0.7746767383559836 and paramet
[I 2024-07-27 09:08:20,831] Trial 53 finished with value: 0.7781048320014686 and paramet
[I 2024-07-27 09:08:22,065] Trial 54 finished with value: 0.7783535168686931 and paramet
[I 2024-07-27 09:08:23,323] Trial 55 finished with value: 0.7653697000292571 and paramet
[I 2024-07-27 09:08:24,565] Trial 56 finished with value: 0.7784435826683571 and paramet
[I 2024-07-27 09:08:25,777] Trial 57 finished with value: 0.7708047694717097 and paramet
[I 2024-07-27 09:08:27,203] Trial 58 finished with value: 0.7759180114389301 and paramet
[I 2024-07-27 09:08:29,162] Trial 59 finished with value: 0.7594910421817722 and paramet
[I 2024-07-27 09:08:30,645] Trial 60 finished with value: 0.7796507225342337 and paramet
[I 2024-07-27 09:08:31,922] Trial 61 finished with value: 0.7781478570650022 and paramet
[I 2024-07-27 09:08:33,207] Trial 62 finished with value: 0.7706805704549757 and paramet
[I 2024-07-27 09:08:34,450] Trial 63 finished with value: 0.7794813472007893 and paramet
[I 2024-07-27 09:08:35,721] Trial 64 finished with value: 0.7794813472007893 and paramet
[I 2024-07-27 09:08:37,030] Trial 65 finished with value: 0.7759158601857535 and paramet
[I 2024-07-27 09:08:38,302] Trial 66 finished with value: 0.7758107356138529 and paramet
[I 2024-07-27 09:08:39,548] Trial 67 finished with value: 0.7777405531302168 and paramet
[I 2024-07-27 09:08:41,353] Trial 68 finished with value: 0.7779956917569715 and paramet
[I 2024-07-27 09:08:43,119] Trial 69 finished with value: 0.7648145332927943 and paramet
[I 2024-07-27 09:08:44,434] Trial 70 finished with value: 0.7776440335710229 and paramet
[I 2024-07-27 09:08:45,692] Trial 71 finished with value: 0.7797386370807208 and paramet
[I 2024-07-27 09:08:47,016] Trial 72 finished with value: 0.7795649592409232 and paramet
[I 2024-07-27 09:08:48,252] Trial 73 finished with value: 0.7652003246958128 and paramet
[I 2024-07-27 09:08:49,483] Trial 74 finished with value: 0.7761688475593316 and paramet
[I 2024-07-27 09:08:50,760] Trial 75 finished with value: 0.779483498453966 and paramete
[I 2024-07-27 09:08:51,718] Trial 76 finished with value: 0.7629258764205442 and paramet
[I 2024-07-27 09:08:53,130] Trial 77 finished with value: 0.7659316073590068 and paramet
[I 2024-07-27 09:08:55,663] Trial 78 finished with value: 0.7506945679423119 and paramet
[I 2024-07-27 09:08:56,954] Trial 79 finished with value: 0.7780620937716919 and paramet
[I 2024-07-27 09:08:58,225] Trial 80 finished with value: 0.7768807689439355 and paramet
[I 2024-07-27 09:08:59,501] Trial 81 finished with value: 0.7794813472007893 and paramet
```

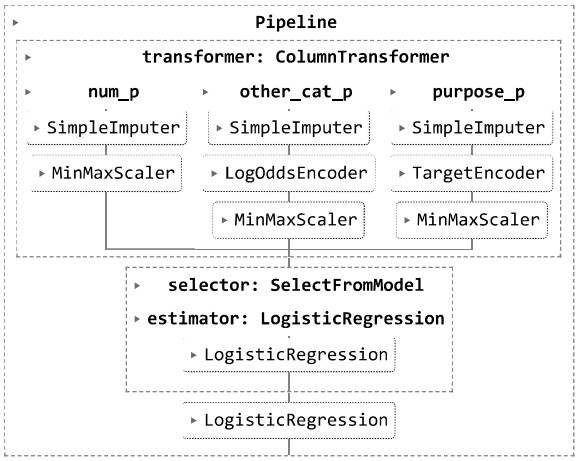
```
[I 2024-07-27 09:08:59,361] Trial 81 finished with value: 0.7794813472007893 and paramet
[I 2024-07-27 09:09:00,768] Trial 82 finished with value: 0.7767541318402681 and paramet
[I 2024-07-27 09:09:02,068] Trial 83 finished with value: 0.7774574482121652 and paramet
[I 2024-07-27 09:09:03,346] Trial 84 finished with value: 0.7795671104940999 and paramet
[I 2024-07-27 09:09:04,607] Trial 85 finished with value: 0.7794813472007893 and paramet
[I 2024-07-27 09:09:05,977] Trial 86 finished with value: 0.7631828794667186 and paramet
[I 2024-07-27 09:09:07,836] Trial 87 finished with value: 0.7773910461974449 and paramet
[I 2024-07-27 09:09:09,352] Trial 88 finished with value: 0.7759158601857534 and paramet
[I 2024-07-27 09:09:10,584] Trial 89 finished with value: 0.779483498453966 and paramete
[I 2024-07-27 09:09:11,574] Trial 90 finished with value: 0.7616160500697007 and paramet
[I 2024-07-27 09:09:12,813] Trial 91 finished with value: 0.7793977351606556 and paramet
[I 2024-07-27 09:09:14,141] Trial 92 finished with value: 0.77650114446669 and parameter
[I 2024-07-27 09:09:15,427] Trial 93 finished with value: 0.7793955839074789 and paramet
[I 2024-07-27 09:09:16,815] Trial 94 finished with value: 0.7794813472007893 and paramet
[I 2024-07-27 09:09:18,072] Trial 95 finished with value: 0.7761688475593316 and paramet
[I 2024-07-27 09:09:19,620] Trial 96 finished with value: 0.7687531910255454 and paramet
[I 2024-07-27 09:09:21,912] Trial 97 finished with value: 0.7517083818560439 and paramet
[I 2024-07-27 09:09:23,195] Trial 98 finished with value: 0.7797364858275441 and paramet
[I 2024-07-27 09:09:24,438] Trial 99 finished with value: 0.779395583907479 and paramete
Best score: 0.7797386370807208
Optimized parameters: {'C': 85.08441326427396, 'l1_ratio': 0.06359193560684012, 'min_fre
```

```
params = study.best_params
params
```

```
{'C': 85.08441326427396,
 'l1_ratio': 0.06359193560684012,
 'min_frequency': 0.02,
 'smoothing': 3.9495914342492124,
 'multiplier': 0.7872054898640072,
 'scaler': 'MinMaxScaler',
 'cat_encoder': 'lg_encoder',
 'purpose_encoder': 'target_encoder'}
```

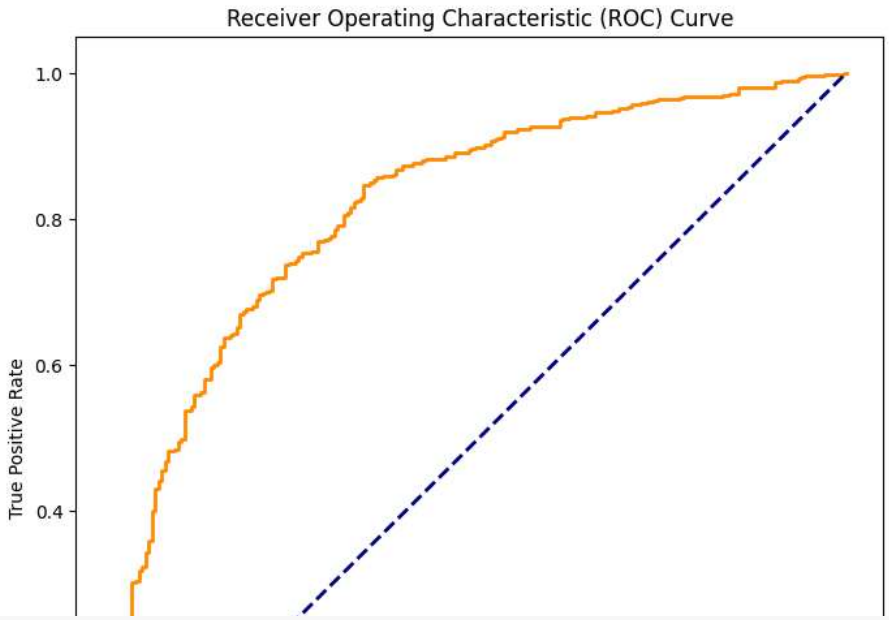
```
model_params = {
    'class_weight': 'balanced',
    'C': params['C'],
    'l1_ratio': params['l1_ratio'],
    'penalty': 'elasticnet',
    'solver': 'saga',
    'random_state': GLOBAL_SEED
}

scaler_options = {
    'StandardScaler': StandardScaler(),
    'RobustScaler': RobustScaler(),
    'MinMaxScaler': MinMaxScaler(feature_range=(-1,1))
}

cat_encoding_options = {
    'ohe': OneHotEncoder(sparse_output=False, handle_unknown='ignore', drop='if_binary'),
    'target_encoder': TargetEncoder(random_state=GLOBAL_SEED),
    'lg_encoder': LogOddsEncoder(params['smoothing'])
}

purpose_encoding_options = {
    'ohe': OneHotEncoder(min_frequency=params['min_frequency'], sparse_output=False, handle_unknown='ignore'),
    'target_encoder': TargetEncoder(random_state=GLOBAL_SEED),
    'lg_encoder': LogOddsEncoder(params['smoothing'])
}


model = LogisticRegression(**model_params)

num_p = Pipeline(steps=[('imputer', SimpleImputer(strategy='constant', fill_value=0)),
                                    ('scaler', scaler_options[params['scaler']])])

if params['cat_encoder'] == 'ohe':
  other_cat_p = Pipeline(steps=[('imputer', SimpleImputer(strategy='constant', fill_value='not_given')),
             ('encoder', cat_encoding_options[selected_cat_encoder])])
else:

    other_cat_p = Pipeline(steps=[('imputer', SimpleImputer(strategy='constant', fill_value='not_given')),
                   ('encoder', cat_encoding_options[params['cat_encoder']]),
                                ('scaler', scaler_options[params['scaler']])])

if params['purpose_encoder'] == 'ohe':
```

```
    purpose_p = Pipeline(steps=[('imputer', SimpleImputer(strategy='constant', fill_value='not_given')),
                    ('encoder', purpose_encoding_options[params['purpose_encoder']])])
else:

    purpose_p = Pipeline(steps=[('imputer', SimpleImputer(strategy='constant', fill_value='not_given')),
                    ('encoder', purpose_encoding_options[params['purpose_encoder']]),
                              ('scaler', scaler_options[params['scaler']])])

transformer = ColumnTransformer(transformers=[
    ('num_p', num_p, numeric_features),
    ('other_cat_p', other_cat_p, other_categorical_features),
    ('purpose_p', purpose_p, purpose)
])


# Full Pipeline
pipeline = Pipeline(steps=[
    ('transformer', transformer),
    ('selector', SelectFromModel(model, threshold=f"{params['multiplier']}*median", max_features=None)),
    ('model', model)
])



# Fit the model
pipeline.fit(X_train, y_train)
```




```
y_train_pred_proba = pipeline.predict_proba(X_train)[:,1]
y_pred_proba = pipeline.predict_proba(X_test)[:,1]
```

```
# Compute ROC curve and AUC
fpr, tpr, thresholds = met.roc_curve(y_train, y_train_pred_proba)
roc_auc = met.auc(fpr, tpr)

# Plot ROC curve
plt.figure(figsize=(8, 8))
plt.plot(fpr, tpr, color='darkorange', lw=2, label=f'AUC = {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()
```




```
# Compute ROC curve and AUC
fpr, tpr, thresholds = met.roc_curve(y_test, y_pred_proba)
roc_auc = met.auc(fpr, tpr)

# Plot ROC curve
plt.figure(figsize=(8, 8))
plt.plot(fpr, tpr, color='darkorange', lw=2, label=f'AUC = {roc_auc:.2f}')
plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
```

Could not connect to the reCAPTCHA service. Please check your internet connection and reload to get a reCAPTCHA challenge.