```
import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
import os
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.preprocessing import MinMaxScaler
from sklearn.model_selection import StratifiedShuffleSplit,
train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import confusion_matrix, accuracy_score,
precision_score, f1_score, roc_auc_score,recall_score
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier,
GradientBoostingClassifier
from sklearn.metrics import precision_recall_fscore_support as score

# Mute the sklearn and IPython warnings
import warnings
warnings.filterwarnings('ignore', module='sklearn')
warnings.filterwarnings('ignore', module='IPython')
```

About Dataset

This dataset is take from **KAGGLE**. It is about Default Payments of Credit Card Clients in Taiwan from 2005

Dataset Information This dataset contains information on default payments, demographic factors, credit data, history of payment, and bill statements of credit card clients in Taiwan from April 2005 to September 2005.

Link of the dataset

https://archive.ics.uci.edu/ml/datasets/default+of+credit+card+clients

Content There are 25 variables:

- ID: ID of each client
- LIMIT_BAL: Amount of given credit in NT dollars (includes individual and family/supplementary credit
- SEX: Gender (1=male, 2=female)
- EDUCATION: (1=graduate school, 2=university, 3=high school, 4=others, 5=unknown, 6=unknown)
- MARRIAGE: Marital status (1=married, 2=single, 3=others)
- AGE: Age in years
- PAY_0: Repayment status in September, 2005 (-1=pay duly, 1=payment delay for one month, 2=payment delay for two months, … 8=payment delay for eight months, 9=payment delay for nine months and above)

- PAY_2: Repayment status in August, 2005 (scale same as above)
- PAY_3: Repayment status in July, 2005 (scale same as above)
- PAY_4: Repayment status in June, 2005 (scale same as above)
- PAY_5: Repayment status in May, 2005 (scale same as above)
- PAY_6: Repayment status in April, 2005 (scale same as above)
- BILL_AMT1: Amount of bill statement in September, 2005 (NT dollar)
- BILL_AMT2: Amount of bill statement in August, 2005 (NT dollar)
- BILL_AMT3: Amount of bill statement in July, 2005 (NT dollar)
- BILL_AMT4: Amount of bill statement in June, 2005 (NT dollar)
- BILL_AMT5: Amount of bill statement in May, 2005 (NT dollar)
- BILL_AMT6: Amount of bill statement in April, 2005 (NT dollar)
- PAY_AMT1: Amount of previous payment in September, 2005 (NT dollar)
- PAY_AMT2: Amount of previous payment in August, 2005 (NT dollar)
- PAY_AMT3: Amount of previous payment in July, 2005 (NT dollar)
- PAY_AMT4: Amount of previous payment in June, 2005 (NT dollar)
- PAY_AMT5: Amount of previous payment in May, 2005 (NT dollar)
- PAY_AMT6: Amount of previous payment in April, 2005 (NT dollar)
- default.payment.next.month: Default payment (1=yes, 0=no)

Main objective

We will predict the payment defaulters on the basis of the dataset provided.Dataset will be splitted into train and test sets.Training dataset will then be trained on different models after that we will find which model will works the best. Best criteria will be based on on accuracy, f1 etc

Stakeholders

By this analysis, our stakeholders will get to know that how our customers are defaulting and customer who have the tendency to default in future payments. It will be very beneficial for our stakeholders to get to know in advance, which customers are going to default beacuse if they know who all have the tendencies to deafault then they can concentreate on creating strategies and robust rules so that customers should default less.

```
data=pd.read_csv('/kaggle/input/uci-credit-card/UCI_Credit_Card.csv')
data.head()
```

Data Cleaning and Exploration

```
data.shape
```

```
data.describe()
data.columns=['ID', 'LIMIT_BAL', 'SEX', 'EDUCATION', 'MARRIAGE', 
'AGE', 'PAY_0',
       'PAY_2', 'PAY_3', 'PAY_4', 'PAY_5', 'PAY_6', 'BILL_AMT1', 
'BILL_AMT2',
       'BILL_AMT3', 'BILL_AMT4', 'BILL_AMT5', 'BILL_AMT6', 'PAY_AMT1',
       'PAY_AMT2', 'PAY_AMT3', 'PAY_AMT4', 'PAY_AMT5', 'PAY_AMT6',
       'default']
data.default.value_counts()

# Finding the unique values in every column
pd.DataFrame([[i, len(data[i].unique())] for i in data.columns],
             columns=['Variable', 'Unique 
Values']).set_index('Variable')

# Check for missing values in dataset  - There are no missing values 
in the dataset
# All the variables are numeric in nature
data.info()

# sns.pairplot(data)

# Check for duplicate customer ids # no duplicate ids found
data.ID.duplicated(keep=False).count()

# All age values are scattered around 29 to 74
data.AGE.hist(grid=False)

sns.pairplot(data)

# Checking the correlation between varibles
plt.figure(figsize = (12, 10))
sns.heatmap(data.corr(), annot = True, 
linewidths=0,fmt='.2f',annot_kws={"size": 8})

# Check the dataset target variables is balanced or unbalanced
print("Defaulters are more in numbers than non defaulters \
n",data.default.value_counts(normalize=True),"")
```

Splitting Data

```
# Get the split indexes
feature_cols = list(data.columns)
feature_cols.remove('default')
stratified_shuf_split = StratifiedShuffleSplit(n_splits=1, 
test_size=0.3, random_state=42)
train, test = next(stratified_shuf_split.split(data[feature_cols], 
data.default))
# Create the dataframes
X_train = data.loc[train, feature_cols]
y_train = data.loc[train, 'default']
```

```
X_test  = data.loc[test, feature_cols]
y_test  = data.loc[test, 'default']
len(X_test), len(X_train)

# effects of stratified shufflesplit
y_train.value_counts(normalize=True),y_test.value_counts(normalize=Tru
e)
```

Model Training and Predictions

Logistic Regression

```
# Standard logistic regression
# Fitting the model
lr = LogisticRegression().fit(X_train, y_train)
# Fitting the model
y_pred_lr = lr.predict(X_test)
# Calculating  the metrices
lr_stats = pd.Series({'precision':precision_score(y_test, y_pred_lr),
                      'recall':recall_score(y_test, y_pred_lr),
                      'accuracy':accuracy_score(y_test, y_pred_lr),
                      'f1score':f1_score(y_test, y_pred_lr),
                      'auc': roc_auc_score(y_test, y_pred_lr)},
                     name='Logistic Regression')

df_lr_final=pd.DataFrame(lr_stats).transpose()
df_lr_final

sns.set_palette(sns.color_palette())
_, ax = plt.subplots(figsize=(4,4))
ax = sns.heatmap(confusion_matrix(y_test, y_pred_lr), annot=True,
fmt='d', annot_kws={"size": 20, "weight": "bold"})
labels = ['False', 'True']
ax.set_xticklabels(labels, fontsize=10);
ax.set_yticklabels(labels[::-1], fontsize=10);
ax.set_ylabel('Prediction', fontsize=15);
ax.set_xlabel('Ground Truth', fontsize=15)
```

K Nearest neighbors

```
# Estimate KNN model and report outcomes
knn = KNeighborsClassifier(n_neighbors=3, weights='distance')

# Fitting the model
knn = knn.fit(X_train, y_train)
# Predicting the model
y_pred_knn = knn.predict(X_test)
```

```
# Calculating  the metrices
knn_stats = pd.Series({'precision':precision_score(y_test,
y_pred_knn),
                      'recall':recall_score(y_test, y_pred_knn),
                      'accuracy':accuracy_score(y_test, y_pred_knn),
                      'f1score':f1_score(y_test, y_pred_knn),
                      'auc': roc_auc_score(y_test, y_pred_knn)
                                      }, name='KNN')

df_knn_final=pd.DataFrame(knn_stats).transpose()
df_knn_final

sns.set_palette(sns.color_palette())
_, ax = plt.subplots(figsize=(4,4))
ax = sns.heatmap(confusion_matrix(y_test, y_pred_knn), annot=True,
fmt='d', annot_kws={"size": 20, "weight": "bold"})
labels = ['False', 'True']
ax.set_xticklabels(labels, fontsize=10);
ax.set_yticklabels(labels[::-1], fontsize=10);
ax.set_ylabel('Prediction', fontsize=15);
ax.set_xlabel('Ground Truth', fontsize=15)
```

Decision Trees

```
dt = DecisionTreeClassifier(random_state=42)
# Fitting the model
dt.fit(X_train, y_train)
# Predicting the model
y_pred_dt = dt.predict(X_test)
# Calculating  the metrices
dt_stats = pd.Series({'precision':precision_score(y_test, y_pred_dt),
                      'recall':recall_score(y_test, y_pred_dt),
                      'accuracy':round(accuracy_score(y_test,
y_pred_dt), 2),
                      'f1score':round(f1_score(y_test, y_pred_dt), 2),
                      'auc': round(roc_auc_score(y_test,
y_pred_dt),2)}, name='Decision Tree')

df_dt_final=pd.DataFrame(dt_stats).transpose()
df_dt_final

sns.set_palette(sns.color_palette())
_, ax = plt.subplots(figsize=(4,4))
ax = sns.heatmap(confusion_matrix(y_test, y_pred_dt), annot=True,
fmt='d', annot_kws={"size": 20, "weight": "bold"})
labels = ['False', 'True']
ax.set_xticklabels(labels, fontsize=10);
ax.set_yticklabels(labels[::-1], fontsize=10);
ax.set_ylabel('Prediction', fontsize=15);
ax.set_xlabel('Ground Truth', fontsize=15)
```

Random Forests

```
# Initialize the random forest estimator
rf = RandomForestClassifier(n_estimators=100, random_state=42,n_jobs=-
1,max_depth=3)
# Fitting the model
rf.fit(X_train, y_train)
# Predicting the model
y_pred_rf = rf.predict(X_test)
# Calculating  the metrices
rf_stats = pd.Series({'precision':precision_score(y_test, y_pred_rf),
                      'recall':recall_score(y_test, y_pred_rf),
                      'accuracy':round(accuracy_score(y_test,
y_pred_rf), 2),
                      'f1score':round(f1_score(y_test, y_pred_rf), 2),
                      'auc': round(roc_auc_score(y_test,
y_pred_rf),2)}, name='Random Forest')

df_rf_final=pd.DataFrame(rf_stats).transpose()
df_rf_final

sns.set_palette(sns.color_palette())
_, ax = plt.subplots(figsize=(4,4))
ax = sns.heatmap(confusion_matrix(y_test, y_pred_rf), annot=True,
fmt='d', annot_kws={"size": 20, "weight": "bold"})
labels = ['False', 'True']
ax.set_xticklabels(labels, fontsize=10);
ax.set_yticklabels(labels[::-1], fontsize=10);
ax.set_ylabel('Prediction', fontsize=15);
ax.set_xlabel('Ground Truth', fontsize=15)
```

Gradient Boosting Classifier

```
# Initialize the random forest estimator
gb = GradientBoostingClassifier(random_state=42,max_depth=3)
# Fiting the model
gb.fit(X_train, y_train)
# Predicting the model
y_pred_gb = gb.predict(X_test)
# Calculating  the metrices
gb_stats = pd.Series({'precision':precision_score(y_test, y_pred_gb),
                      'recall':recall_score(y_test, y_pred_gb),
                      'accuracy':round(accuracy_score(y_test,
y_pred_gb), 2),
                      'f1score':round(f1_score(y_test, y_pred_gb), 2),
                      'auc': round(roc_auc_score(y_test,
y_pred_gb),2)}, name='Gradient boosting')

df_gb_final=pd.DataFrame(gb_stats).transpose()
df_gb_final
```

```
sns.set_palette(sns.color_palette())
_, ax = plt.subplots(figsize=(4,4))
ax = sns.heatmap(confusion_matrix(y_test, y_pred_gb), annot=True,
fmt='d', annot_kws={"size": 20, "weight": "bold"})
labels = ['False', 'True']
ax.set_xticklabels(labels, fontsize=10);
ax.set_yticklabels(labels[::-1], fontsize=10);
ax.set_ylabel('Prediction', fontsize=15);
ax.set_xlabel('Ground Truth', fontsize=15)
```

Summary and key insights

I have created 5 models logistic, KNN, decision tree, random forests and gradient boosting and used five different validation metrices. below is the summary of all provided how they have performed. All the models are trained on same training sets and tested on same test sets. Also, almost all of the models used same parameters. From the **confusion matrix** it is evident that **Logistic Regression** performed very badly with **ZERO precision and recall**. KNN and Decision Tree model gave some edge as precision, recall and f1 scores starts to improve in these two models by decreasing some accuracy I think gradient boosting method have performed very well as compared to other models with highest accuracy,Precision, recall and highest f1-score.

```
final_report=df_lr_final.append(df_knn_final).append(df_dt_final).appe
nd(df_rf_final).append(df_gb_final)
final_report
```

Feature Importance

```
assert len(gb.feature_importances_)==len(X_train.columns)
feature_imp=pd.DataFrame(gb.feature_importances_).transpose()
feature_imp.columns=[X_train.columns]
sns.barplot(feature_imp,orient='h')
```

Suggestions and next steps for revisiting the model

We could further optimize these models

1. Using **GridSearchCV** that will find the best parameters for every model.
2. Using Sampling because data is **unbalanced**, so we can also look from that angle also to increase the accuracy of the model.
3. We could also change our model based on the **inputs received from our stakeholders** about the business.
4. We could also use XGboost model