

# Using ChatGPT and Prompt Engineering: End-to-End Data Science Project - Loan Approval Classifier

In the project, we will be using Loan Data from Kaggle and execute the data science project using ChatGPT and Prompt Engineering.

- ChatGPT does 80 percent of the work
- write codes through prompt engineering to get everything done

## Project Planning

- It is the most important part of the project, where we look at the available resources and goals to come up with an optimal strategy.
- Initiate a new chat at Open AI.
- Use loan dataset and ask ChatGPT to come up with steps to build an end-to-end generic portfolio project.

### Initial Prompt:

"I have a loan dataset consisting of 9500 rows and 14 columns: ['credit.policy', 'purpose', 'int.rate', 'installment', 'log.annual.inc', 'dti', 'fico', 'days.with.cr.line', 'revol.bal', 'revol.util', 'inq.last.6mths', 'delinq.2yrs', 'pub.rec', 'not.fully.paid'].

Can you list the steps I have to follow to develop an end-to-end project for my portfolio."

We did get the list, but we forgot to mention the *class imbalance issue and the project goal*, which is to accurately predict "loan not paid back".

### Updated Prompt:

"Please include class imbalance issues and accurately predict whether a loan will not be paid back rather than if a loan is paid back."

Again, we are not interested in model monitoring, and we want to build a Gradio app and deploy it on Huggingface Spaces.

### Again Updated Prompt:

"We will be creating a web app using Gradio and deploying it on Spaces and we won't be monitoring the model in the production."

We got a list of 9 steps, as shown above, and a detailed explanation of how we should approach each step.

### Here is the list of tasks:

- **Data cleaning and preprocessing:** Handling missing values, converting categorical variables into numeric variables, scaling/normalizing the data, and handling any outliers or anomalies in the data. In addition, you'll want to address the class imbalance issue by either oversampling the minority class (loans that are not fully paid) or undersampling the majority class (loans that are fully paid).
- **Exploratory data analysis (EDA):** Exploring the dataset to gain insights into the data, such as distributions of variables, correlations between variables, and identifying any patterns in the data.
- **Feature engineering:** Creating new features or transforming existing features to improve the predictive power of the model.
- **Model selection:** Try several models, such as decision trees, random forests, logistic regression, or support vector machines (SVM).
- **Model training and evaluation:** Training the selected model on the data and evaluating its performance using various metrics such as accuracy, precision, recall, and F1 score.
- **Hyperparameter tuning:** Fine-tuning the hyperparameters of the selected model to improve its performance.
- **Creating the web app using Gradio:** Once you have selected the best model, you can create a web app using Gradio.
- **Deploying the web app on Spaces:** Once you have created the web app using Gradio, you can deploy it on Spaces.
- **Testing the web app:** Test it to make sure it is working as expected.

## Exploratory Data Analysis (EDA)

Exploratory data analysis is all about data manipulation, statistical analysis, and data visualization. We can complete it by writing a follow-up prompt about writing Python code for exploratory data analysis on our dataset.

### Follow-Up Prompt:

"Please write a Python code to load and perform Exploratory data analysis (EDA) on loan dataset"

The bot will understand the context and come up with Python code with comments and a detailed explanation of how it works.

```
In [1]: import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

# Load the Loan dataset
```

```
df = pd.read_csv("loan_data.csv")

df.head()
```

```
Out[1]:
```

	credit.policy	purpose	int.rate	installment	log.annual.inc	dti	fico	days.with.cr.line	r
0	1	debt_consolidation	0.1189	829.10	11.350407	19.48	737	5639.958333	
1	1	credit_card	0.1071	228.22	11.082143	14.29	707	2760.000000	
2	1	debt_consolidation	0.1357	366.86	10.373491	11.63	682	4710.000000	
3	1	debt_consolidation	0.1008	162.34	11.350407	8.10	712	2699.958333	
4	1	credit_card	0.1426	102.92	11.299732	14.97	667	4066.000000	

```
In [2]: # Display the number of rows and columns in the dataset
print("Number of rows and columns:", df.shape)

# Display summary statistics for numerical variables
# print(df.describe())
df.describe()
```

Number of rows and columns: (9578, 14)

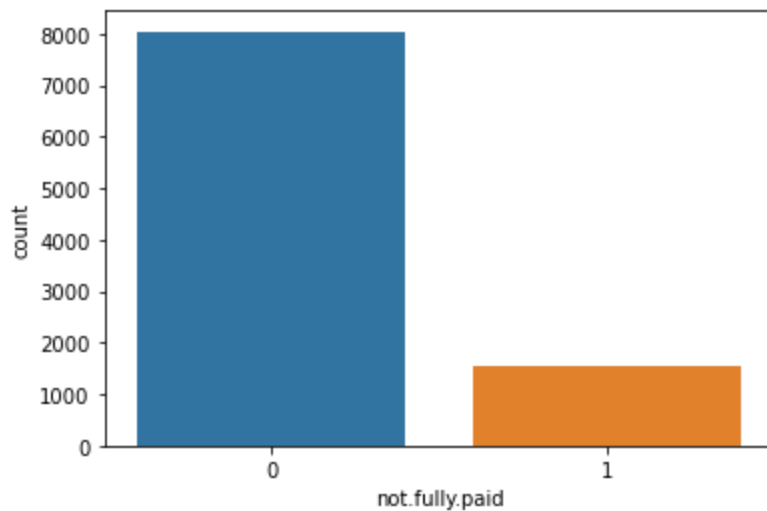
```
Out[2]:
```

	credit.policy	int.rate	installment	log.annual.inc	dti	fico	days.with.cr.li
<b>count</b>	9578.000000	9578.000000	9578.000000	9578.000000	9578.000000	9578.000000	9578.000000
<b>mean</b>	0.804970	0.122640	319.089413	10.932117	12.606679	710.846314	4560.76719
<b>std</b>	0.396245	0.026847	207.071301	0.614813	6.883970	37.970537	2496.93037
<b>min</b>	0.000000	0.060000	15.670000	7.547502	0.000000	612.000000	178.95833
<b>25%</b>	1.000000	0.103900	163.770000	10.558414	7.212500	682.000000	2820.00000
<b>50%</b>	1.000000	0.122100	268.950000	10.928884	12.665000	707.000000	4139.95833
<b>75%</b>	1.000000	0.140700	432.762500	11.291293	17.950000	737.000000	5730.00000
<b>max</b>	1.000000	0.216400	940.140000	14.528354	29.960000	827.000000	17639.95833

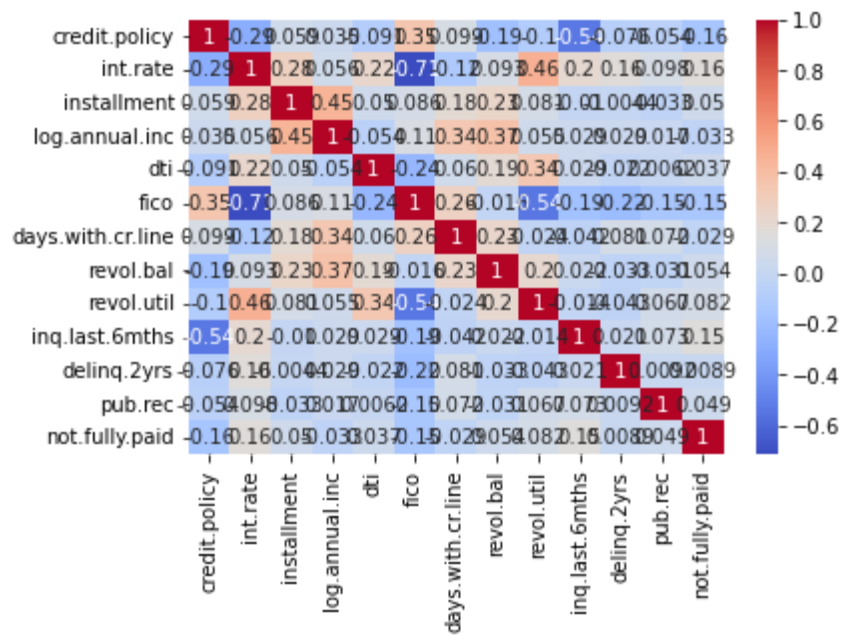
```
In [3]: # Display the number of missing values in each column
df.isnull().sum()
```

```
Out[3]: credit.policy      0
        purpose           0
        int.rate          0
        installment       0
        log.annual.inc    0
        dti               0
        fico              0
        days.with.cr.line 0
        revol.bal         0
        revol.util        0
        inq.last.6mths    0
        delinq.2yrs       0
        pub.rec           0
        not.fully.paid    0
        dtype: int64
```

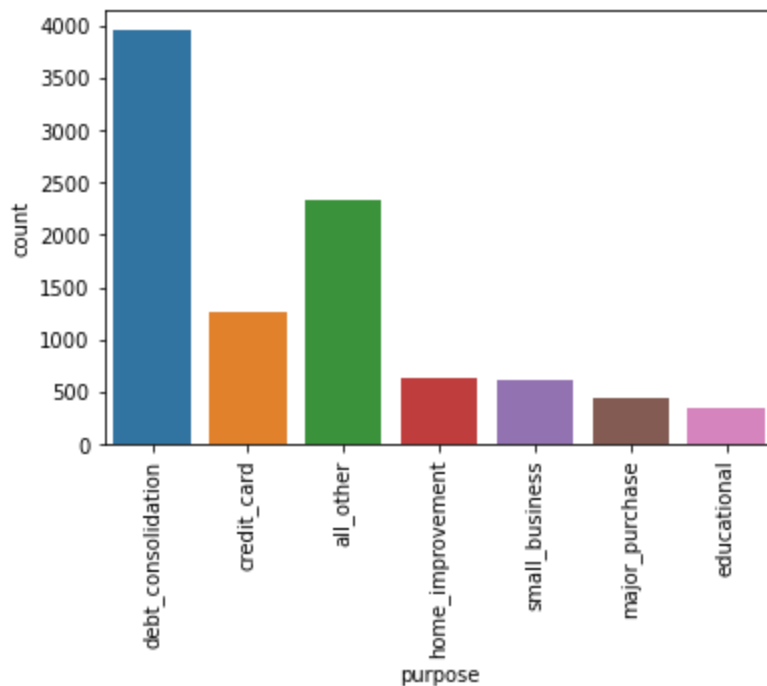
```
In [4]: # Visualize the distribution of the target variable
        sns.countplot(x="not.fully.paid", data=df)
        plt.show()
```



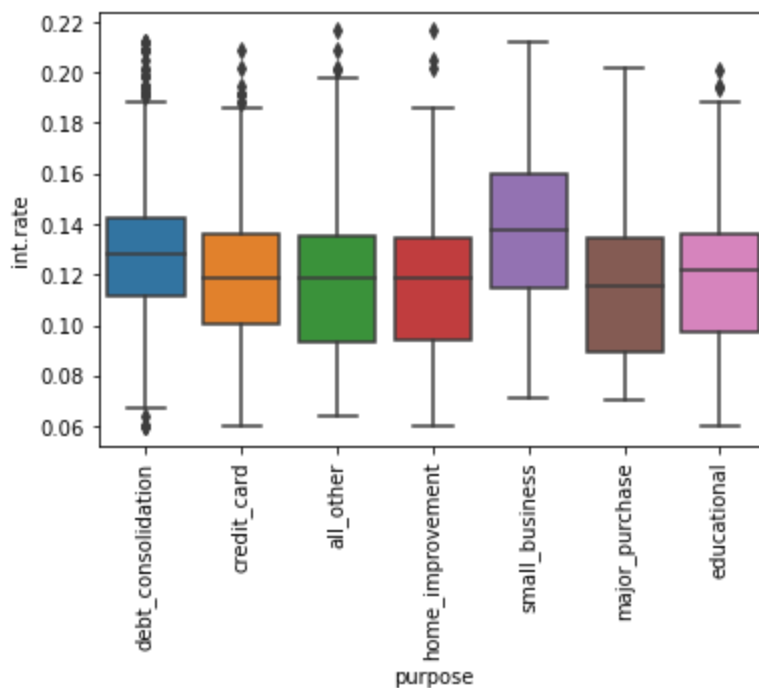
```
In [5]: # Visualize the correlation between variables
        corr = df.corr()
        sns.heatmap(corr, annot=True, cmap="coolwarm")
        plt.show()
```



```
In [6]: # Visualize the distribution of loan purpose
sns.countplot(x="purpose", data=df)
plt.xticks(rotation=90)
plt.show()
```



```
In [7]: # Visualize the distribution of interest rates by loan purpose
sns.boxplot(x="purpose", y="int.rate", data=df)
plt.xticks(rotation=90)
plt.show()
```



## Feature Engineering

Instead of preprocessing and cleaning the data, we will perform feature engineering to simplify the preprocessing task.

Follow-up Prompt: "Write a Python code to perform feature engineering"

```
In [8]: # Perform feature engineering
df["installment_to_income_ratio"] = (
    df["installment"] / df["log.annual.inc"]
)
df["credit_history"] = (df["delinq.2yrs"] + df["pub.rec"]) / df[
    "fico"
]
```

```
In [9]: df.head()
```

```
Out[9]:
```

	credit.policy	purpose	int.rate	installment	log.annual.inc	dti	fico	days.with.cr.line	r
0	1	debt_consolidation	0.1189	829.10	11.350407	19.48	737	5639.958333	
1	1	credit_card	0.1071	228.22	11.082143	14.29	707	2760.000000	
2	1	debt_consolidation	0.1357	366.86	10.373491	11.63	682	4710.000000	
3	1	debt_consolidation	0.1008	162.34	11.350407	8.10	712	2699.958333	
4	1	credit_card	0.1426	102.92	11.299732	14.97	667	4066.000000	

## Preprocessing and Balancing the Data

Again, ChatGPT knows the context of the project. We don't have to explain everything again. So, we will simply ask it to write a Python code to clean and preprocess the data.

```
In [10]: from sklearn.preprocessing import LabelEncoder, StandardScaler
# Drop unnecessary columns
df = df.drop(['credit.policy', 'days.with.cr.line', 'purpose'], axis=1)

# Convert categorical variables to numerical using LabelEncoder
le = LabelEncoder()
df['not.fully.paid'] = le.fit_transform(df['not.fully.paid'])
```

```
In [11]: # Scale the numerical variables using StandardScaler
scaler = StandardScaler()
numerical_cols = ['int.rate', 'installment', 'log.annual.inc', 'dti', 'fico', 'revol.bal']
df[numerical_cols] = scaler.fit_transform(df[numerical_cols])
```

```
In [12]: df.head()
```

```
Out[12]:
```

	int.rate	installment	log.annual.inc	dti	fico	revol.bal	revol.util	inq.last.6mths	de
0	-0.139318	2.463099	0.680388	0.998505	0.688825	0.353732	0.182704	-0.716989	-
1	-0.578868	-0.438854	0.244031	0.244540	-0.101303	0.495018	1.030602	-0.716989	-
2	0.486484	0.230708	-0.908659	-0.141885	-0.759742	-0.397073	-0.730683	-0.262470	-
3	-0.813544	-0.757022	0.680388	-0.654697	0.030385	0.496321	0.909966	-0.262470	-
4	0.743509	-1.043992	0.597961	0.343326	-1.154806	-0.360663	-0.251586	-0.716989	-

```
In [14]: # Handle class imbalance by oversampling the minority class
from imblearn.over_sampling import SMOTE

sm = SMOTE(random_state=42)

X = df.drop('not.fully.paid', axis=1)
y = df['not.fully.paid']

X_resampled, y_resampled = sm.fit_resample(X, y)

df = pd.concat([X_resampled, y_resampled], axis=1)
```

```
In [15]: df.head()
```

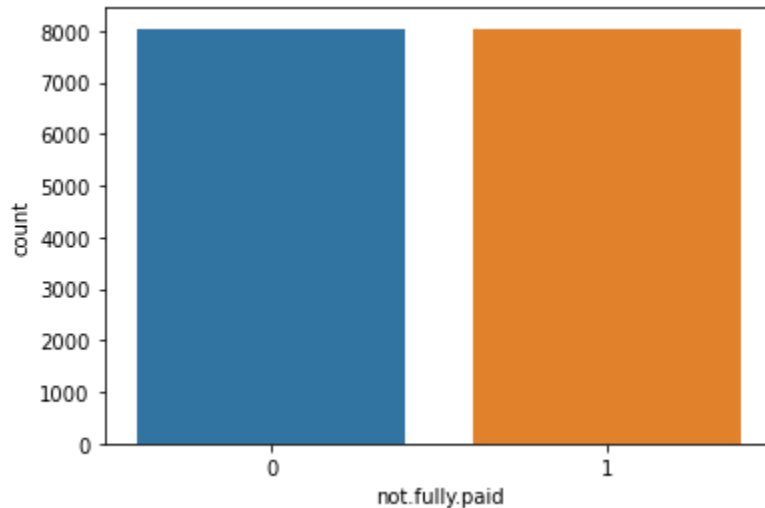
```
Out[15]:
```

	int.rate	installment	log.annual.inc	dti	fico	revol.bal	revol.util	inq.last.6mths	de
0	-0.139318	2.463099	0.680388	0.998505	0.688825	0.353732	0.182704	-0.716989	-
1	-0.578868	-0.438854	0.244031	0.244540	-0.101303	0.495018	1.030602	-0.716989	-
2	0.486484	0.230708	-0.908659	-0.141885	-0.759742	-0.397073	-0.730683	-0.262470	-
3	-0.813544	-0.757022	0.680388	-0.654697	0.030385	0.496321	0.909966	-0.262470	-
4	0.743509	-1.043992	0.597961	0.343326	-1.154806	-0.360663	-0.251586	-0.716989	-

```
In [16]: df['not.fully.paid'].value_counts()
```

```
Out[16]: 0    8045
         1    8045
         Name: not.fully.paid, dtype: int64
```

```
In [17]: # Visualize the distribution of the target variable
sns.countplot(x="not.fully.paid", data=df)
plt.show()
```



## Model Selection

In this part, we have asked ChatGPT to train the data on various models for a model selection task.

Follow-up Prompt: "Write a follow-up python code for model selection. Try decision trees, random forests, logistic regression, or support vector machines (SVM). Note: only include the model selection part."

It has generated the Python script to split the dataset into a training and test and train the dataset on specified models to display accuracy metrics.

```
In [18]: from sklearn.tree import DecisionTreeClassifier
         from sklearn.ensemble import RandomForestClassifier
         from sklearn.linear_model import LogisticRegression
         from sklearn.svm import SVC
         from sklearn.metrics import classification_report, accuracy_score
```

```
In [19]: from sklearn.model_selection import train_test_split

         # Split the dataset into training and testing sets
         X = df.drop('not.fully.paid', axis=1)
         y = df['not.fully.paid']
         X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=
```

```
In [ ]: # from sklearn.metrics import classification_report, accuracy_score

         # # Initialize and train different models
```



```

# # Model Evaluation

# # Evaluate Decision Tree
# y_pred_dt = dt.predict(X_test)
# print("Decision Tree Model Performance:")
# print(classification_report(y_test, y_pred_dt))
# print("Accuracy:", accuracy_score(y_test, y_pred_dt))

# # Evaluate Random Forest
# y_pred_random_forest = rf.predict(X_test)
# print("\nRandom Forest Model Performance:")
# print(classification_report(y_test, y_pred_random_forest))
# print("Accuracy:", accuracy_score(y_test, y_pred_random_forest))

# # Evaluate Logistic Regression
# y_pred_logistic_regression = lr.predict(X_test)
# print("\nLogistic Regression Model Performance:")
# print(classification_report(y_test, y_pred_logistic_regression))
# print("Accuracy:", accuracy_score(y_test, y_pred_logistic_regression))

# # Evaluate Support Vector Machine (SVM)
# y_pred_svm = svm.predict(X_test)
# print("\nSupport Vector Machine (SVM) Model Performance:")
# print(classification_report(y_test, y_pred_svm))
# print("Accuracy:", accuracy_score(y_test, y_pred_svm))

```

```

In [20]: # from sklearn.tree import DecisionTreeClassifier

# Fit and evaluate decision tree classifier
dt = DecisionTreeClassifier(random_state=42)
dt.fit(X_train, y_train)
dt_score = dt.score(X_test, y_test)
print("Decision Tree Classifier Accuracy: {:.2f}%".format(dt_score*100))

print("=====")

# Evaluate Decision Tree
y_pred_dt = dt.predict(X_test)
print("Decision Tree Model Performance:")
print(classification_report(y_test, y_pred_dt))
print("=====")
print("Accuracy:", accuracy_score(y_test, y_pred_dt))

```

```

Decision Tree Classifier Accuracy: 81.21%
=====
Decision Tree Model Performance:

```

	precision	recall	f1-score	support
0	0.83	0.79	0.81	2444
1	0.80	0.83	0.81	2383
accuracy			0.81	4827
macro avg	0.81	0.81	0.81	4827
weighted avg	0.81	0.81	0.81	4827

```

=====
Accuracy: 0.8120986119743112

```

```
In [21]: # from sklearn.ensemble import RandomForestClassifier

# Fit and evaluate random forest classifier
rf = RandomForestClassifier(random_state=42)
rf.fit(X_train, y_train)
rf_score = rf.score(X_test, y_test)
print("Random Forest Classifier Accuracy: {:.2f}%".format(rf_score*100))

print("=====")

# Evaluate Random Forest
y_pred_random_forest = rf.predict(X_test)
print("\nRandom Forest Model Performance:")
print(classification_report(y_test, y_pred_random_forest))
print("=====")
print("Accuracy:", accuracy_score(y_test, y_pred_random_forest))
```

Random Forest Classifier Accuracy: 89.14%

=====

Random Forest Model Performance:

	precision	recall	f1-score	support
0	0.89	0.90	0.89	2444
1	0.90	0.88	0.89	2383
accuracy			0.89	4827
macro avg	0.89	0.89	0.89	4827
weighted avg	0.89	0.89	0.89	4827

=====

Accuracy: 0.8914439610524135

```
In [22]: # from sklearn.linear_model import LogisticRegression

# Fit and evaluate logistic regression classifier
lr = LogisticRegression(random_state=42)
lr.fit(X_train, y_train)
lr_score = lr.score(X_test, y_test)
print("Logistic Regression Classifier Accuracy: {:.2f}%".format(lr_score*100))

print("=====")

# Evaluate Logistic Regression
y_pred_logistic_regression = lr.predict(X_test)
print("\nLogistic Regression Model Performance:")
print(classification_report(y_test, y_pred_logistic_regression))
print("=====")
print("Accuracy:", accuracy_score(y_test, y_pred_logistic_regression))
```

Logistic Regression Classifier Accuracy: 61.32%

```
=====
```

Logistic Regression Model Performance:				
	precision	recall	f1-score	support
0	0.62	0.60	0.61	2444
1	0.60	0.63	0.62	2383
accuracy				0.61 4827
macro avg	0.61	0.61	0.61	4827
weighted avg	0.61	0.61	0.61	4827

```
=====
```

Accuracy: 0.6132173192459084

```
In [23]: # from sklearn.svm import SVC

# Fit and evaluate support vector machine classifier
svm = SVC(random_state=42)
svm.fit(X_train, y_train)
svm_score = svm.score(X_test, y_test)
print("Support Vector Machine Classifier Accuracy: {:.2f}%".format(svm_score*100))

print("=====")

# Evaluate Support Vector Machine (SVM)
y_pred_svm = svm.predict(X_test)
print("\nSupport Vector Machine (SVM) Model Performance:")
print(classification_report(y_test, y_pred_svm))

print("=====")
print("Accuracy:", accuracy_score(y_test, y_pred_svm))
```

Support Vector Machine Classifier Accuracy: 66.54%

```
=====
```

Support Vector Machine (SVM) Model Performance:				
	precision	recall	f1-score	support
0	0.70	0.60	0.64	2444
1	0.64	0.73	0.68	2383
accuracy				0.67 4827
macro avg	0.67	0.67	0.66	4827
weighted avg	0.67	0.67	0.66	4827

```
=====
```

Accuracy: 0.6654236585871142

- **Decision Tree Classifier Accuracy: 81.21%**
- **Random Forest Classifier Accuracy: 89.14%**
- **Logistic Regression Classifier Accuracy: 61.32%**
- **Support Vector Machine Classifier Accuracy: 66.54%**

## Hyperparameter Tuning and Model Evaluation

We will be selecting the better-performing model and performing the model evaluation. We have specified the use of accuracy, precision, recall, and F1 score as metrics and cross-validation to ensure the model stability.

Follow-up Prompt: "Select RandomForestClassifier and write a python code for model evaluation. Use accuracy, precision, recall, and F1 score as metric and cross-validation to ensure the model is not overfitting to the training data."

```
In [24]: from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import cross_val_score, StratifiedKFold
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score

# # Initialize the RandomForestClassifier
# rf_model = RandomForestClassifier(random_state=42)

# Define evaluation metrics
scoring_metrics = ['accuracy', 'precision', 'recall', 'f1']

# Define cross-validation strategy (StratifiedKFold is used for classification)
cv = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)

# Perform cross-validation and evaluate the model
for metric in scoring_metrics:
    # scores = cross_val_score(rf_model, X_train_resampled, y_train_resampled, cv=cv,
    scores = cross_val_score(rf, X_train, y_train, cv=cv, scoring=metric)
    print(f'{metric.capitalize()} (Cross-Validation):')
    print("Scores:", scores)
    print("Mean:", scores.mean())
    print("Standard Deviation:", scores.std())

# Train the final model on the entire training data
rf.fit(X_train, y_train)

# Evaluate the final model on the test data
y_pred_rf = rf.predict(X_test)

print("\nRandom Forest Model Performance (Test Data):")

print("Accuracy:", accuracy_score(y_test, y_pred_rf))

print("=====")

print("Precision:", precision_score(y_test, y_pred_rf))

print("=====")

print("Recall:", recall_score(y_test, y_pred_rf))

print("=====")

print("F1 Score:", f1_score(y_test, y_pred_rf))
```

```

Accuracy (Cross-Validation):
Scores: [0.87039503 0.87527741 0.86906347 0.86856128 0.86856128]
Mean: 0.870371693081023
Standard Deviation: 0.0025429599655303367
Precision (Cross-Validation):
Scores: [0.88899167 0.90037594 0.88080073 0.88419118 0.87522442]
Mean: 0.8859167870294664
Standard Deviation: 0.00850882614826354
Recall (Cross-Validation):
Scores: [0.84819064 0.84554281 0.85512367 0.84982332 0.86130742]
Mean: 0.851997573595227
Standard Deviation: 0.005609453749562348
F1 (Cross-Validation):
Scores: [0.86811201 0.87209832 0.8677723 0.86666667 0.86821015]
Mean: 0.8685718895605952
Standard Deviation: 0.001846469112746746

```

Random Forest Model Performance (Test Data):

Accuracy: 0.8914439610524135

Precision: 0.896037494674052

Recall: 0.8825010490977759

F1 Score: 0.8892177589852008

**We will update the Python to add hyperparameter tuning tasks and save the best-performing model.**

**Updated Prompt:** "also include hyperparameter tuning in the above code and save the best-performing model"

The final version of the code is using GridSearchCV for hyperparameter tuning with five cross-validation splits and uses the f1 metric for evaluation to find the best hyperparameters.

After that, ChatGPT automatically added the code to select the best model for model evaluation on the testing set and displayed the best-performing hyperparameters.

```

In [25]: from sklearn.model_selection import GridSearchCV
# Define the parameter grid for hyperparameter tuning
param_grid = {
    'n_estimators': [100, 200, 300],
    'max_depth': [10, 20, 30, None],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4]
}

# Perform grid search with cross-validation to find the best hyperparameters
grid_search = GridSearchCV(rf, param_grid, cv=5, scoring='f1')
# grid_search.fit(X_train, y_train)

# Get the best model and its hyperparameters
# best_params = grid_search.best_params_
## n_estimators:200, max_depth:20, min_samples_split:10, min_samples_leaf:1

```

```

In [26]: from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score

```

```

best_model = RandomForestClassifier(n_estimators=200, max_depth=20, min_samples_split=
best_model.fit(X_train, y_train)

# Evaluate the best model on the testing set
y_pred = best_model.predict(X_test)
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)

# Print the results
print("Random Forest Classifier Evaluation Results:")
print("Accuracy: {:.2f}%".format(accuracy * 100))
print("Precision: {:.2f}%".format(precision * 100))
print("Recall: {:.2f}%".format(recall * 100))
print("F1 Score: {:.2f}%".format(f1 * 100))

```

```

Random Forest Classifier Evaluation Results:
Accuracy: 86.08%
Precision: 86.89%
Recall: 84.56%
F1 Score: 85.71%

```

## Saving the Model and Scaler

```

In [27]: import joblib
# Save the best model to disk
joblib.dump(best_model, 'loan_classifier.joblib')

```

```

Out[27]: ['loan_classifier.joblib']

```

```

In [28]: # Save the scaler to disk
joblib.dump(scaler, 'std_scaler.bin')

```

```

Out[28]: ['std_scaler.bin']

```

## Conclusion

- Developing a loan approval classifier is one of many examples of using ChatGPT for data science projects.
- We can use it to generate synthetic data, run SQL queries, create data analytics reports, do machine learning research, and much more.
- Generative AI is here to stay, and it will make our lives easier.
- Instead of spending weeks and months on a project, you can develop, test, and deploy data science applications in hours.

In this project, we have used ChatGPT and Prompt Engineering for project planning for;

1. Data analysis,
2. Data cleaning and preprocessing,
3. Model selection,

4. Hyperparameter optimization,
5. Creating and deploying web applications.

In [ ]: