Loan Approval Prediction Notebook Using Logistic Regression, Random Forest, XGBoot

In [ ]:
```
#Incase the XGBoost library is not installed, please uncomment the below line.
#pip install xgboost
```

In [ ]:
```
#importing the necessary libraries
import numpy as np
import pandas as pd
import matplotlib as mpl
import matplotlib.pyplot as plt
import seaborn as sns
import sklearn
import warnings; warnings.filterwarnings(action='once')
```

In [ ]:
```
from sklearn.preprocessing import LabelEncoder
```

Importing the dataset.

In [ ]:
```
#Raw content URL for loan_approval_dataset.csv
url = "https://raw.githubusercontent.com/hastighsh/Credit-Card-Loan-Prediction/main

#Read the data into a DataFrame
df = pd.read_csv(url, delimiter=',')
df
```

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

In [ ]:
```
df.describe()
```

In [ ]:
```
#checking the dataframe information
df.info()
```

Performing Cleaning and Showing 3 Different EDAs.

1- Cleaning the dataset from redundant columns and make the values into integer.

In [ ]:
```
# dropping loan_id from the dataset since it has no value to the training
df.drop(columns="loan_id", inplace=True)

# removing the space before the column names in the rest of the dataset
df.rename(columns=lambda x: x.strip(), inplace=True)
df.info()
```

In [ ]:
```
# # Create a LabelEncoder instance
label_encoder = LabelEncoder()
#
# # Apply label encoding to the 'education' column
df['education'] = label_encoder.fit_transform(df['education'])
```

```
#
# # Apply label encoding to the 'self_employed' column
df['self_employed'] = label_encoder.fit_transform(df['self_employed'])
#
df['loan_status'] = label_encoder.fit_transform(df['loan_status'])
# #changing the data type of the column to int64 from int32
df['education'] = df['education'].astype('int64')
df['self_employed'] = df['self_employed'].astype('int64')
df['loan_status'] = df['loan_status'].astype('int64')
```

In [ ]:
```
df
```

In [ ]:
```
df.info()
```

2- Perform EDA on the dataset to understand the distribution of the data and identify any trends or patterns.

In [ ]:
```
# plotting a histogram of the data using hist()
df.hist(figsize=(20, 16))
plt.show()
```

In [ ]:
```
# Pairwise plot is a favorite in exploratory analysis to understand the relationshi
plt.figure(figsize=(10,8), dpi= 80)
sns.pairplot(df, kind="scatter", plot_kws=dict(s=80, edgecolor="white", linewidth=2
plt.show()
```

In [ ]:
```
# Plotting Correlation Matrix
correlation_matrix = df.corr()
plt.figure(figsize=(12, 10))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', fmt=".2f")
plt.title("Correlation Matrix Heatmap")
plt.show()
```

3- Perform pipelining and preprocessing.

In [ ]:
```
# importing the libraries
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.pipeline import make_pipeline, Pipeline
from sklearn.impute import SimpleImputer
```

In [ ]:
```
# Define the numeric columns
X = df.drop('loan_status',axis=1)
y = df['loan_status']
num_cols = X.select_dtypes(include='number').columns.to_list()
# cat_cols = X.select_dtypes(exclude='number').columns.to_list()
```

In [ ]:
```
# Create pipelines for numeric columns
num_pipeline = make_pipeline(SimpleImputer(strategy='mean'), StandardScaler())
```

In [ ]:
```
#Preprocessing for numerical data
numeric_transformer = Pipeline(steps=[
```

```
    ('imputer', SimpleImputer(strategy='mean')),
    ('scaler', StandardScaler())
])

# Bundle preprocessing for numerical and categorical data
preprocessor = ColumnTransformer(
    transformers=[
        ('num', numeric_transformer, num_cols)],
#         ('cat', categorical_transformer, cat_cols)],
        remainder='passthrough'
)
```

In [ ]:
```
# Create and apply the preprocessing pipeline
data_prepared = preprocessor.fit_transform(X)
feature_names = preprocessor.get_feature_names_out()
data_prepared = pd.DataFrame(data=data_prepared, columns=feature_names)
```

In [ ]:
```
from sklearn.linear_model import Ridge, Lasso
from sklearn.model_selection import cross_val_score
from sklearn.neural_network import MLPClassifier
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
import pandas as pd
import seaborn as sns
import xgboost as xgb
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report
```

This cell below is for feature selection

In [ ]:
```
from sklearn.feature_selection import RFE
from sklearn.ensemble import RandomForestClassifier

#inital the estimator as RandomForestClassifier
y = y.ravel()
estimator = RandomForestClassifier(n_estimators=100, random_state=42)
#use RFE to choose the attribute
selector = RFE(estimator, n_features_to_select=5, step=1)
selector = selector.fit(data_prepared, y)
#get attributes' name
selected_features = data_prepared.columns[selector.support_]
```

The cell below is a generic function to calculate accuracy of our training models to find the best training process

In [ ]:
```
#generic function to calculate accuracy
#inputs: model, model_name, y_test, X_test, X_train, y_train

def calculate_accuracy_classifier(model, model_name, y_test, X_test, X_train, y_tra
    # Make predictions on the test set
    y_pred = model.predict(X_test)

    # Calculate accuracy and other metrics
    accuracy = accuracy_score(y_test, y_pred)
    conf_matrix = confusion_matrix(y_test, y_pred)
```

```
    class_report = classification_report(y_test, y_pred)
    mae = mean_absolute_error(y_test, y_pred)
    mse = mean_squared_error(y_test, y_pred)
    r2 = r2_score(y_test, y_pred)

    # Cross-validation
    scores = cross_val_score(model, X_train, y_train, cv=5, scoring='accuracy')
    mean_score = scores.mean()

    print(f'{model_name} Accuracy: {accuracy}')
    print(f'{model_name} Mean Absolute Error: {mae}')
    print(f'{model_name} Mean Squared Error: {mse}')
    print(f'{model_name} R-squared: {r2}')
    print(f'{model_name} Confusion Matrix:\n{conf_matrix}')
    print(f'{model_name} Classification Report:\n{class_report}')
    print(f'{model_name} Cross-Validation Mean Accuracy: {mean_score}')

    # Plot a heatmap of the confusion matrix
    sns.heatmap(conf_matrix, annot=True, fmt="d", cmap="Blues", cbar=False,
                xticklabels=model.classes_, yticklabels=model.classes_)
    plt.xlabel('Predicted')
    plt.ylabel('True')
    plt.title(f'Confusion Matrix - {model_name}')
    plt.show()
```

The cells below are for 3 different training which are: logistic regression, xgboost, random forest and calculating accuracy of each of them

In [ ]:
```
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split

# Assuming 'data_prepared' is your feature matrix and 'y' is your target variable
# Replace 'selected_features' with the actual features you want to use for predicti
X = data_prepared[selected_features]
y = y.ravel()

# Split the data into training, validation, and testing sets
X_train, X_temp, y_train, y_temp = train_test_split(X, y, test_size=0.4, random_sta
X_val, X_test, y_val, y_test = train_test_split(X_temp, y_temp, test_size=0.5, rand

# Print the shapes of the datasets
print(f"Training set: {X_train.shape}, Validation set: {X_val.shape}, Test set: {X_
```

In [ ]:
```
#Train a logistic regression model and evaluate its performance on validation and t
#inputs: X_train, y_train: Training data and labels | X_val, y_val: Validation data

def logistic_regression(X_train, y_train, X_val, y_val):

    # Initialize the logistic regression model
    logreg_model = LogisticRegression()

    # Train the logistic regression model on the training set
    logreg_model.fit(X_train, y_train)
```

```
    # Calculate accuracy and other metrics on the validation set
    calculate_accuracy_classifier(logreg_model, "Logistic Regression (Validation)",
```

In [ ]:
```
#Train an XGBoost model and evaluate its performance on validation and test sets.
#inputs: X_train, y_train: Training data and labels | X_val, y_val: Validation data

def xgboost(X_train, y_train, X_val, y_val):

    # Initialize the XGBoost model
    xgb_model = xgb.XGBClassifier()

    # Train the XGBoost model on the training set
    xgb_model.fit(X_train, y_train)

    # Calculate accuracy and other metrics on the validation set
    calculate_accuracy_classifier(xgb_model, "XGBoost (Validation)", y_val, X_val,

    return xgb_model
```

In [ ]:
```
#Train a Random Forest model and evaluate its performance on validation and test se
#inputs: X_train, y_train: Training data and labels | X_val, y_val: Validation data

def random_forest(X_train, y_train, X_val, y_val):

    # Initialize the Random Forest model
    rf_model = RandomForestClassifier(n_estimators=100, random_state=42)

    # Train the Random Forest model on the training set
    rf_model.fit(X_train, y_train)

    # Calculate accuracy and other metrics on the validation set
    calculate_accuracy_classifier(rf_model, "Random Forest (Validation)", y_val, X_

    return rf_model
```

In [ ]:
```
logistic_regression(X_train, y_train, X_val, y_val)
```

In [ ]:
```
xg_model = xgboost(X_train, y_train, X_val, y_val)
```

In [ ]:
```
rf_model = random_forest(X_train, y_train, X_val, y_val)
```

Plotting 3 Different Graphs for the Chosen Model (Random Forest).

In [ ]:
```
from sklearn.metrics import roc_curve, auc

# Assuming 'random_forest_model' is your trained Random Forest model
y_scores = rf_model.predict_proba(X_test)[:, 1]
fpr, tpr, thresholds = roc_curve(y_test, y_scores)
roc_auc = auc(fpr, tpr)

# Plot ROC Curve
plt.figure(figsize=(8, 6))
```

```
plt.plot(fpr, tpr, color='darkorange', lw=2, label=f'ROC Curve (AUC = {roc_auc:.2f}
plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--', label='Random Guessing
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve - Random Forest')
plt.legend(loc='lower right')
plt.show()
```

In [ ]:
```
from sklearn.metrics import precision_recall_curve

y_probs = rf_model.predict_proba(X_test)[:, 1]
y_true = y_test

# Create a precision-recall curve
precision, recall, thresholds = precision_recall_curve(y_true, y_probs)

# Plot the precision-recall curve
plt.figure(figsize=(8, 8))
plt.plot(recall, precision, color='darkorange', lw=2, label='Precision-Recall Curve
plt.xlabel('Recall (Sensitivity)')
plt.ylabel('Precision')
plt.title('Random Forest - Precision-Recall Curve')
plt.legend()
plt.show()
```

In [ ]:
```
importances = rf_model.feature_importances_
feature_names = X.columns
print(feature_names)

# Sort feature importances in descending order
indices = np.argsort(importances)[::-1]

plt.figure(figsize=(12, 6))
sns.barplot(x=importances[indices], y=feature_names[indices])
plt.xlabel('Feature Importance Score')
plt.ylabel('Features')
plt.title('Feature Importance - Random Forest')
plt.show()
```