Assignment 2: Individual Assignment

# Remmy Bisimbeko - B26099 - J24M19/011

My GitHub - https://github.com/RemmyBisimbeko/Data-Science

Q1. You are provided with a dataset labelled "arrhythmia-1_data.csv" and a description of the variables within the file "arrhythmia.names.txt".

Transform the dataset and use it to determine the features (variables) that affect the Heart rate.

Q2. The dataset "Loan_Approval_Data.csv" is used by a banking institution to determine if a loan gets approved or not using various features. Transform the dataset prior to attempting the questions. (Loan_Approval_Data.csv )

a) Determine which features affect the "Loan_Status" (10 Marks) b) Using a machine learning model, determine if the Loan_Status of;

i) A female with the following variables (married, zero dependents, a graduate education, self employed, An Applicant Income of 2000 dollars, CoApplicant Income of 5000 dollars, requesting for a Loan_Amount_Term of 365 days, With No credit history=0, and living in a Semiurban area). (5 Marks)

ii) A male with the following variables (single, 2 dependents, graduate education, employed by a private company, an Applicant income of 4800 dollars, CoApplicant Income of zero (0), requesting a Loan_Amount_Term of 480 days, With credit history =1, and living in an Urban area). (5 Marks)

In [ ]:
```
# Import Libraries
import pandas as pd
import numpy as np
```

In [ ]:
```
# Read the Data Set while replacing '?' values with NaN
df = pd.read_csv("Data Sets/arrhythmia-1.data.csv", na_values='?')

# Convert columns to numeric
df = df.apply(pd.to_numeric, errors='coerce')
df.head()
```

Out[ ]:

| | Age | Sex | Height | Weight | QRS Duration | P-R interval | Q-T interval | T interval | P interval | QRS |
|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 75 | 0 | 190 | 80 | 91 | 193 | 371 | 174 | 121 | -16 |
| 1 | 56 | 1 | 165 | 64 | 81 | 174 | 401 | 149 | 39 | 25 |
| 2 | 54 | 0 | 172 | 95 | 138 | 163 | 386 | 185 | 102 | 96 |
| 3 | 55 | 0 | 175 | 94 | 100 | 202 | 380 | 179 | 143 | 28 |
| 4 | 75 | 0 | 190 | 80 | 88 | 181 | 360 | 177 | 103 | -16 |

QN 1. Transform the dataset and use it to determine the features (variables) that affect the Heart rate

We can determine the features that affect heart rate, you can perform a regression analysis, such as linear regression, where heart rate is the dependent variable and the other variables (Age, Sex, Height, Weight, etc.) are independent variables.

Let us import the statsmodels library: statsmodels is a Python module that provides classes and functions for the estimation of many different statistical models, as well as for conducting statistical tests, and statistical data exploration. An extensive list of result statistics are available for each estimator. The results are tested against existing statistical packages to ensure that they are correct. The package is released under the open source Modified BSD (3-clause) license. The online documentation is hosted at statsmodels.org.

In [ ]:
```
# Import statsmodels library
import statsmodels.api as sm
```

In [ ]:
```
# Let us get the relevant variables
X = df[['Age', 'Sex', 'Height', 'Weight', 'QRS Duration', 'P-R interval',
y = df['Heart rate']
```

In [ ]:
```
# Let's add  aconstant term
X = sm.add_constant(X)
```

In [ ]:
```
# Handlinf the  missing values
X.fillna(X.mean(), inplace=True)
y.fillna(y.mean(), inplace=True)
```

In [ ]:
```
# Nown, we fit the regression model
model = sm.OLS(y, X).fit()

# And Print the summary of the regression model
print(model.summary())
```

```
                            OLS Regression Results                            
==========================================================================
====
Dep. Variable:             Heart rate   R-squared:                       
0.529
Model:                            OLS   Adj. R-squared:                  
0.514
Method:                 Least Squares   F-statistic:                     3
5.04
Date:                Sun, 18 Feb 2024   Prob (F-statistic):           1.43
e-62
Time:                        14:18:14   Log-Likelihood:                -16
58.9
No. Observations:                 452   AIC:                             3
348.
Df Residuals:                     437   BIC:                             3
410.
Df Model:                          14                                         
Covariance Type:            nonrobust                                         
==========================================================================
======
                   coef    std err          t      P>|t|      [0.025      
0.975]
--------------------------------------------------------------------------
------
const          146.5380      7.024     20.863      0.000     132.733     1
60.343
Age             -0.0154      0.032     -0.488      0.626      -0.078      
0.047
Sex              5.6785      1.044      5.441      0.000       3.627      
7.730
Height           0.0547      0.013      4.229      0.000       0.029      
0.080
Weight          -0.0514      0.031     -1.633      0.103      -0.113      
0.010
QRS Duration     0.1501      0.036      4.220      0.000       0.080      
0.220
P-R interval    -0.0286      0.014     -2.055      0.040      -0.056      
-0.001
Q-T interval    -0.2847      0.016    -18.169      0.000      -0.315      
-0.254
T interval       0.0450      0.014      3.159      0.002       0.017      
0.073
P interval       0.0813      0.024      3.344      0.001       0.034      
0.129
QRS             -0.0094      0.015     -0.614      0.540      -0.039      
0.021
T                0.0086      0.009      0.942      0.346      -0.009      
0.026
P                0.0010      0.017      0.062      0.950      -0.032      
0.034
QRST             0.0133      0.020      0.673      0.501      -0.026      
0.052
J               -0.0037      0.009     -0.397      0.691      -0.022      
0.015
==========================================================================
====
Omnibus:                       42.653   Durbin-Watson:                   
1.949
Prob(Omnibus):                  0.000   Jarque-Bera (JB):               8
```

```
0.380
Skew:                           0.569   Prob(JB):                     3.51
e-18
Kurtosis:                       4.725   Cond. No.                     7.67
e+03
==========================================================================
====

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is cor
rectly specified.
[2] The condition number is large, 7.67e+03. This might indicate that ther
e are
strong multicollinearity or other numerical problems.
```

QN2. The dataset "Loan_Approval_Data.csv" is used by a banking institution to determine if a loan gets approved or not using various features. Transform the dataset prior to attempting the questions. (Loan_Approval_Data.csv)

Inorder to transform the dataset and prepare it for analysis, we need to address missing values, encode categorical variables, and perform any other necessary data preprocessing steps. Let's go through these steps:

Handle Missing Values, Encode Categorical Variables, Explore Data

In [ ]:
```
# Load the relevant Data Sets
import pandas as pd
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import OneHotEncoder, LabelEncoder
```

In [ ]:
```
# Load the dataset
df = pd.read_csv("Data Sets/Loan_Approval_Data.csv")
df.head()
```

Out[ ]:

| | Loan_ID | Gender | Married | Dependents | Education | Self_Employed | ApplicantIr |
|---|---|---|---|---|---|---|---|
| 0 | LP001002 | Male | No | 0 | Graduate | No | |
| 1 | LP001003 | Male | Yes | 1 | Graduate | No | |
| 2 | LP001005 | Male | Yes | 0 | Graduate | Yes | |
| 3 | LP001006 | Male | Yes | 0 | Not Graduate | No | |
| 4 | LP001008 | Male | No | 0 | Graduate | No | |

In [ ]:
```
# Now, we handle missing values
# For Numerical columns: replace with mean
numerical_cols = ['ApplicantIncome', 'CoapplicantIncome', 'LoanAmount', '
imputer = SimpleImputer(strategy='mean')
df[numerical_cols] = imputer.fit_transform(df[numerical_cols])
```

In [ ]:
```
# For Categorical columns: replace with most frequent value
categorical_cols = ['Gender', 'Married', 'Dependents', 'Education', 'Self
imputer = SimpleImputer(strategy='most_frequent')
df[categorical_cols] = imputer.fit_transform(df[categorical_cols])
```

In [ ]:
```
# We now Encode categorical variables
# Using one-hot encoding for 'Property_Area'
df = pd.get_dummies(df, columns=['Property_Area'], drop_first=True)
```

In [ ]:
```
# I will now use label encoding for other categorical variables
label_encoder = LabelEncoder()
df['Gender'] = label_encoder.fit_transform(df['Gender'])
df['Married'] = label_encoder.fit_transform(df['Married'])
df['Education'] = label_encoder.fit_transform(df['Education'])
df['Self_Employed'] = label_encoder.fit_transform(df['Self_Employed'])
```

In [ ]:
```
# We can now Encode 'Dependents' column
# Since it has ordinal values (0, 1, 2, 3+), we'll map them accordingly
dependents_mapping = {'0': 0, '1': 1, '2': 2, '3+': 3}
df['Dependents'] = df['Dependents'].map(dependents_mapping)
```

In [ ]:
```
# Encode 'Loan_Status' column (target variable)
df['Loan_Status'] = df['Loan_Status'].map({'Y': 1, 'N': 0})
```

In [ ]:
```
# Finally, I will now Save the transformed dataset
df.to_csv("Data Sets/transformed_loan_data.csv", index=False)
```

In [ ]:
```
# Let us check it
df.head()
```

Out[ ]:

| | Loan_ID | Gender | Married | Dependents | Education | Self_Employed | ApplicantIr |
|---|---|---|---|---|---|---|---|
| **0** | LP001002 | 1 | 0 | 0 | 0 | 0 | 5 |
| **1** | LP001003 | 1 | 1 | 1 | 0 | 0 | 4 |
| **2** | LP001005 | 1 | 1 | 0 | 0 | 1 | 3 |
| **3** | LP001006 | 1 | 1 | 0 | 1 | 0 | 2 |
| **4** | LP001008 | 1 | 0 | 0 | 0 | 0 | 6 |

SO, ideally, this is the thought process of the transformation process 1-I Replaced any missing values with appropriate methods such as mean, median, or mode for numerical variables, and with the most frequent category for categorical variables. I used the SimpleImputer from scikit-learn. 2-Encoded Categorical Variables where Conversion of categorical variables was done into numerical format using one-hot encoding for 'Property_Area' or label encoding. 3-Explore Data where I Checkd for any anomalies or inconsistencies in the data and handle them accordingly. The 'Dependents' column was mapped to ordinal values. Finally, the transformed dataset saved to 'Data Sets/transformed_loan_data.csv'.

QN 2 a) Determine which features affect the "Loan_Status" (10 Marks)

To determine which features that affect the "Loan_Status," I will use Logistic Regression, which is a classification algorithm identify the significant predictors

In [ ]:
```
# Impport Libraries
import pandas as pd
```

```
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score
```

In [ ]:
```
# Read the newly saved transformed Data Set
df = pd.read_csv('Data Sets/transformed_loan_data.csv')
df.head()
```

Out[ ]:

| | Loan_ID | Gender | Married | Dependents | Education | Self_Employed | ApplicantIr |
|---|---|---|---|---|---|---|---|
| **0** | LP001002 | 1 | 0 | 0 | 0 | 0 | 5 |
| **1** | LP001003 | 1 | 1 | 1 | 0 | 0 | 4 |
| **2** | LP001005 | 1 | 1 | 0 | 0 | 1 | 3 |
| **3** | LP001006 | 1 | 1 | 0 | 1 | 0 | 2 |
| **4** | LP001008 | 1 | 0 | 0 | 0 | 0 | 6 |

In [ ]:
```
# Lets exclude Loan_ID column
df = df.drop('Loan_ID', axis=1)
```

In [ ]:
```
# Let us Check for any missing values
missing_values = df.isnull().sum()
print("Missing values in the dataset:")
print(missing_values)
```

```
Missing values in the dataset:
Gender                     0
Married                    0
Dependents                 0
Education                  0
Self_Employed              0
ApplicantIncome            0
CoapplicantIncome          0
LoanAmount                 0
Loan_Amount_Term           0
Credit_History             0
Loan_Status                0
Property_Area_Semiurban    0
Property_Area_Urban        0
dtype: int64
```

In [ ]:
```
# Impute missing values in the Dependents column with the mode
imputer = SimpleImputer(strategy='most_frequent')
df['Dependents'] = imputer.fit_transform(df[['Dependents']])
```

In [ ]:
```
# Define features (independent variables) and our target variable
X = df.drop('Loan_Status', axis=1)
y = df['Loan_Status']
```

In [ ]:
```
# I then Split the data into training and testing sets with a test size o
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
```

In [ ]:
```
# then initialize Logistic Regression model
model = LogisticRegression()
```

In [ ]:
```
# Initialize Logistic Regression model specifying max_iter to counter err
model = LogisticRegression(solver='saga', max_iter=2000)
```

In [ ]:
```
# Fit the model on the training data
model.fit(X_train, y_train)
```

Out[ ]:
```
LogisticRegression
LogisticRegression(max_iter=2000, solver='saga')
```

In [ ]:
```
# Let's predict on the testing set
y_pred = model.predict(X_test)
```

In [ ]:
```
# I now calculate the accuracy
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)
```

```
Accuracy: 0.6504065040650406
```

In [ ]:
```
# Finally, I get the coefficients (importance) of each feature
coefficients = pd.DataFrame(model.coef_[0], index=X.columns, columns=['Co
coefficients.sort_values(by='Coefficient', ascending=False, inplace=True)
print("Feature coefficients:")
print(coefficients)
```

```
Feature coefficients:
                         Coefficient
Loan_Amount_Term         2.312585e-03
LoanAmount               4.579856e-04
Credit_History           2.827937e-05
Property_Area_Semiurban  1.165903e-05
Married                  1.139385e-05
Dependents               9.854722e-06
Gender                   7.197956e-06
Self_Employed            1.016146e-06
Property_Area_Urban     -3.934486e-07
Education               -4.935191e-07
ApplicantIncome         -4.575739e-06
CoapplicantIncome       -3.163851e-05
```

QN 2 b) Using a machine learning model, determine if the Loan_Status of; i) A female with the following variables (married, zero dependents, a graduate education, self employed, An Applicant Income of 2000 dollars, CoApplicant Income of 5000 dollars, requesting for a Loan_Amount_Term of 365 days, With No credit history=0, and living in a Semiurban area). (5 Marks)

In [ ]:
```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import OneHotEncoder, StandardScaler
from sklearn.impute import SimpleImputer
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, precision_score, recall_score
```

In [ ]:
```
# let me Load the data
df = pd.read_csv("Data Sets/transformed_loan_data.csv")
```

In [ ]:
```
# Lets exclude Loan_ID column like before
df = df.drop('Loan_ID', axis=1)
```

In [ ]:
```
# Now let me handle missing values (replace with your preferred imputatio
imputer = SimpleImputer(strategy='mean')  # Or 'median', 'mode', and so o
df['ApplicantIncome'] = imputer.fit_transform(df[['ApplicantIncome']])
```

In [ ]:
```
# Convert non-numerical values incase strings are present
df['ApplicantIncome'] = pd.to_numeric(df['ApplicantIncome'], errors='coer
```

In [ ]:
```
# Ensure consistent string-based feature names, this will address the Typ
df.columns = df.columns.astype(str)
```

In [ ]:
```
# `I proceed to Create new features
df['Total_Income'] = df['ApplicantIncome'] + df['CoapplicantIncome']
df['Loan_Amount_to_Income_Ratio'] = df['LoanAmount'] / df['Total_Income']
```

In [ ]:
```
# I then encode categorical variables
encoder = OneHotEncoder()
encoded_df = pd.DataFrame(encoder.fit_transform(df[['Gender', 'Married',
df = pd.concat([df.drop(['Property_Area_Urban', 'Property_Area_Semiurban'
```

In [ ]:
```
# NOw , check the data types
df.dtypes
```

Out[ ]:
```
Gender                 int64
Married                int64
Dependents             int64
Education              int64
Self_Employed          int64
ApplicantIncome      float64
CoapplicantIncome    float64
LoanAmount           float64
Loan_Amount_Term     float64
Credit_History       float64
Loan_Status            int64
0                     object
dtype: object
```

In [ ]:
```
# Remove the last column from your dataset
df = df.drop(columns=[0])
```

In [ ]:
```
# Spliting data
X = df.drop('Loan_Status', axis=1)
y = df['Loan_Status']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
```

In [ ]:
```
# Convert the feature names to strings before scaling
X_train.columns = X_train.columns.astype(str)
X_test.columns = X_test.columns.astype(str)
```

In [ ]:
```
# Removing of any additional or missing columns in the test data
X_test = X_test[X_train.columns]
```

In [ ]:
```
# Scale numerical features
scaler = StandardScaler()
```

```
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

In [ ]:
```
# Choose and train your model
from sklearn.ensemble import HistGradientBoostingClassifier

model = HistGradientBoostingClassifier()  # Adjust hyperparameters as nee
model.fit(X_train, y_train)
```

Out[ ]:
```
▾ HistGradientBoostingClassifier
HistGradientBoostingClassifier()
```

In [ ]:
```
# Evaluating performance
y_pred = model.predict(X_test_scaled)
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(f"Accuracy: {accuracy:.4f}")
print(f"Precision: {precision:.4f}")
print(f"Recall: {recall:.4f}")
print(f"F1-score: {f1:.4f}")
```

```
Accuracy: 0.7724
Precision: 0.7766
Recall: 0.9125
F1-score: 0.8391
```

```
/usr/local/lib/python3.9/site-packages/sklearn/base.py:493: UserWarning: X
does not have valid feature names, but HistGradientBoostingClassifier was
fitted with feature names
  warnings.warn(
```

In [ ]:
```
# New applicant data. let me modify with actual values
applicant_data = {
    'Gender': 1,  # Female
    'Married': 0,  # Not married
    'Dependents': 0,  # Zero dependents
    'Education': 0,  # Graduate education
    'Self_Employed': 1,  # Self-employed
    'ApplicantIncome': 2000,
    'CoapplicantIncome': 5000,
    'LoanAmount': df['LoanAmount'].median(),  # Use median value for loan
    'Loan_Amount_Term': 365,
    'Credit_History': 0,  # No credit history
    'Property_Area_Urban_False': 1,  # Not in urban area
    'Property_Area_Urban_True': 0,
    'Property_Area_Semiurban_False': 0,  # In semiurban area
    'Property_Area_Semiurban_True': 1
}
```

In [ ]:
```
applicant_df = pd.DataFrame([applicant_data])

transformed_applicant = pd.concat([applicant_df, encoded_df], axis=1)  #
transformed_applicant.columns = transformed_applicant.columns.astype(str)

# Remove any additional features not present during fitting
```

```
transformed_applicant = transformed_applicant.drop(columns=['0', 'Propert

transformed_applicant_scaled = scaler.transform(transformed_applicant)  #

prediction = model.predict(transformed_applicant_scaled)[0]
print(f"Predicted Loan Status for new applicant: {prediction}")
```

```
Predicted Loan Status for new applicant: 0
```

```
/usr/local/lib/python3.9/site-packages/sklearn/base.py:493: UserWarning: X
does not have valid feature names, but HistGradientBoostingClassifier was
fitted with feature names
  warnings.warn(
```

In [ ]:
```
QN 2 b)
Using a machine learning model, determine if the Loan_Status of;
ii) A male with the following variables (single, 2 dependents, graduate e
an Applicant income of 4800 dollars, CoApplicant Income of zero (0), requ
With credit history =1, and living in an Urban area). (5 Marks)
```

In [ ]:
```
# Bring in teh Libraries
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.ensemble import HistGradientBoostingClassifier
from sklearn.preprocessing import LabelEncoder
```

In [ ]:
```
# Load data into DataFrame
df = pd.read_csv("Data Sets/transformed_loan_data.csv")
df.head()
```

Out[ ]:

| | Loan_ID | Gender | Married | Dependents | Education | Self_Employed | ApplicantIr |
|---|---|---|---|---|---|---|---|
| **0** | LP001002 | 1 | 0 | 0 | 0 | 0 | 5 |
| **1** | LP001003 | 1 | 1 | 1 | 0 | 0 | 4 |
| **2** | LP001005 | 1 | 1 | 0 | 0 | 1 | 3 |
| **3** | LP001006 | 1 | 1 | 0 | 1 | 0 | 2 |
| **4** | LP001008 | 1 | 0 | 0 | 0 | 0 | 6 |

In [ ]:
```
# Lets exclude Loan_ID column like before
df = df.drop('Loan_ID', axis=1)
```

In [ ]:
```
# Handle missing values using SimpleImputer
imputer = SimpleImputer(strategy='mean')
df['ApplicantIncome'] = imputer.fit_transform(df[['ApplicantIncome']])
```

In [ ]:
```
# Convert categorical variables to numerical using label encoding
label_encoder = LabelEncoder()
categorical_cols = ['Gender', 'Married', 'Dependents', 'Education', 'Self
for col in categorical_cols:
    df[col] = label_encoder.fit_transform(df[col])
```

In [ ]:
```
# Let me create new features
df['Total_Income'] = df['ApplicantIncome'] + df['CoapplicantIncome']
df['Loan_Amount_to_Income_Ratio'] = df['LoanAmount'] / df['Total_Income']
```

In [ ]:
```
# What are their data dtypes
df.dtypes
```

Out[ ]:
```
Gender                      int64
Married                     int64
Dependents                  int64
Education                   int64
Self_Employed               int64
ApplicantIncome           float64
CoapplicantIncome         float64
LoanAmount                float64
Loan_Amount_Term          float64
Credit_History            float64
Loan_Status                 int64
Property_Area_Semiurban     int64
Property_Area_Urban         int64
dtype: object
```

In [ ]:
```
# Remove the last object as before
df = df.drop(columns=[0])
```

In [ ]:
```
# Split data into features (X) and target (y), separate features and targ
X = df.drop(columns=['Loan_Status'])  # Features
y = df['Loan_Status']  # Target variable
```

In [ ]:
```
# Split data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
```

In [ ]:
```
# Convert the feature names to strings before scaling
X_train.columns = X_train.columns.astype(str)
X_test.columns = X_test.columns.astype(str)
```

In [ ]:
```
# Removing of any additional or missing columns in the test data
X_test = X_test[X_train.columns]
```

In [ ]:
```
print(X_test.columns)
```

```
Index(['Gender', 'Married', 'Dependents', 'Education', 'Self_Employed',
       'ApplicantIncome', 'CoapplicantIncome', 'LoanAmount',
       'Loan_Amount_Term', 'Credit_History', 'Total_Income',
       'Loan_Amount_to_Income_Ratio'],
      dtype='object')
```

In [ ]:
```
print(X_train.columns)
```

```
Index(['Gender', 'Married', 'Dependents', 'Education', 'Self_Employed',
       'ApplicantIncome', 'CoapplicantIncome', 'LoanAmount',
       'Loan_Amount_Term', 'Credit_History', 'Total_Income',
       'Loan_Amount_to_Income_Ratio'],
      dtype='object')
```

In [ ]:
```
print(y_test)
```

```
350    1
377    1
163    1
609    1
132    1
      ..
231    1
312    1
248    1
11     1
333    1
Name: Loan_Status, Length: 123, dtype: int64
```

In [ ]:
```
# Choose and train the model
model = HistGradientBoostingClassifier()
model.fit(X_train, y_train)
```

Out[ ]:
```
HistGradientBoostingClassifier

HistGradientBoostingClassifier()
```

In [ ]:
```
# Now ... evaluate performance on the test set
y_pred = 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(f"Accuracy: {accuracy:.4f}")
print(f"Precision: {precision:.4f}")
print(f"Recall: {recall:.4f}")
print(f"F1-score: {f1:.4f}")
```

```
Accuracy: 0.7886
Precision: 0.7812
Recall: 0.9375
F1-score: 0.8523
```

In [ ]:
```
# Define our new applicant data
# new_applicant = {
#     'Gender': [1],  # Male
#     'Married': [0],  # Single
#     'Dependents': [2],  # 2 dependents
#     'Education': [0],  # Graduate
#     'Self_Employed': [0],  # Employed
#     'ApplicantIncome': [4800.0],
#     'CoapplicantIncome': [0.0],  # CoApplicant Income is zero
#     'LoanAmount': [0],  # Unknown
#     'Loan_Amount_Term': [480],  # 480 days
#     'Credit_History': [1],  # Good credit history
#     'Property_Area_Semiurban': [0],  # Not in Semiurban area
#     'Property_Area_Urban': [1]  # In Urban area
# }

new_applicant_data = [[1, 0, 2, 0, 0, 4800.0, 0.0, 0, 480, 1, 0, 1]]  # R
new_applicant = pd.DataFrame(new_applicant_data, columns=X.columns)
```

In [ ]:
```
# Make predictions for the new applicant
prediction = model.predict(new_applicant)
```

```
print("Predicted Loan Status for new applicant:", prediction[0])
```

```
Predicted Loan Status for new applicant: 1
```

Sources: https://www.statsmodels.org/stable/index.html
towardsdatascience.com/build-an-extreme-learning-machine-in-python-91d1e8958599 github.com/icyda17/DPhi_Deploy_model
www.analyticsvidhya.com/blog/2021/06/cross-sell-prediction-solution-to-analytics-vidya/ github.com/Avik1007/ML-Project-Airbnb-New-user-Bookings
github.com/1696648139/bustag github.com/Alchez/scikit-learn-example
github.com/wangjiajen/NCKU_practice_Maggie