Creating a model for Loan Sanction ammount Prediction

data from https://www.kaggle.com/datasets/boss0ayush/loan-sanction-amount-prediction-data/

Importing all the Library Needed

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
from sklearn.model_selection import  GridSearchCV, train_test_split
from sklearn.tree import DecisionTreeRegressor
from sklearn.metrics import mean_absolute_error, mean_absolute_percentage_error, mean_squared
from sklearn.preprocessing import LabelEncoder
from scipy import stats
import math

import gdown
```

Loading the data to colab

you can change this block of code depending on how your loading your data, I'am using Google drive

```
def load_data(link):
    # Extract file ID from the provided link
    file_id = link.split('/')[-2]

    # Generate direct download link
    csv_link = f'https://drive.google.com/uc?id={file_id}'

    # Download the CSV file
    gdown.download(csv_link, f'{file_id}.csv', quiet=False)

    # Read the CSV file
    df = pd.read_csv(f'{file_id}.csv', encoding='utf-8')

    return df

# Use the provided links
test_link = 'https://drive.google.com/file/d/1WVW4P32kTIB6RNbufikoBzdT5K7DkEo_/view?usp=shari

train_link = 'https://drive.google.com/file/d/1WSui92w1LQ-ZLybpXoQ8_wpMV0tPe90E/view?usp=shar

# Load the data
test = load_data(test_link)

train = load_data(train_link)
```

```
Downloading...
From: https://drive.google.com/uc?id=1WVW4P32kTIB6RNbufikoBzdT5K7DkEo_
To: /content/1WVW4P32kTIB6RNbufikoBzdT5K7DkEo_.csv
100%|██████████| 2.87M/2.87M [00:00<00:00, 232MB/s]
Downloading...
From: https://drive.google.com/uc?id=1WSui92w1LQ-ZLybpXoQ8_wpMV0tPe90E
To: /content/1WSui92w1LQ-ZLybpXoQ8_wpMV0tPe90E.csv
100%|██████████| 4.48M/4.48M [00:00<00:00, 232MB/s]
```

```
train.head()
```

	Customer ID	Name	Gender	Age	Income (USD)	Income Stability	Profession	Type of Employment	Location	Loan Amount Request (USD)	...	Credit Score	No. of Defaults	Has Active Credit Card	P
0	C-36995	Frederica Shealy	F	56	1933.05	Low	Working	Sales staff	Semi-Urban	72809.58	...	809.44	0	NaN	
		America							Semi-						

Data Preprocessing Part 1

```
#Check the number of unique value from all of the object datatype
train.select_dtypes(include='object').nunique()
```

```
Customer ID               30000
Name                      30000
Gender                        2
Income Stability              2
Profession                    8
Type of Employment           18
Location                      3
Expense Type 1                2
Expense Type 2                2
Has Active Credit Card        3
Property Location             3
dtype: int64
```

```
# Check the amounnt of missing value
check_missing = train.isnull().sum() * 100 / train.shape[0]
check_missing[check_missing > 0].sort_values(ascending=False)
```

```
Type of Employment            24.233333
Property Age                  16.166667
Income (USD)                  15.253333
Dependents                     8.310000
Credit Score                   5.676667
Income Stability               5.610000
Has Active Credit Card         5.220000
Property Location              1.186667
Loan Sanction Amount (USD)     1.133333
Current Loan Expenses (USD)    0.573333
Gender                         0.176667
dtype: float64
```

```
def preprocess_data(df):
    # Remove identifier columns
    df.drop(columns=['Customer ID', 'Name'], inplace=True)

    # Segment Type of Employment into smaller unique values
    df['Type of Employment'] = df['Type of Employment'].apply(segment_employment_type)



    # Handle missing values
    df.fillna({
        'Property Age': df['Property Age'].median(),
        'Income (USD)': df['Income (USD)'].median(),
        'Dependents': df['Dependents'].median(),
        'Credit Score': df['Credit Score'].median(),
        'Loan Sanction Amount (USD)': df['Loan Sanction Amount (USD)'].median(),
        'Current Loan Expenses (USD)': df['Current Loan Expenses (USD)'].median()
    }, inplace=True)

    df.dropna(subset=['Income Stability', 'Has Active Credit Card', 'Property Location', 'Gender'], inplace=True)

    return df.head(10)
```

Segment Type of Employment into smaller unique value

```
train['Type of Employment'].unique()
```

```
array(['Sales staff', nan, 'High skill tech staff', 'Secretaries',
       'Laborers', 'Managers', 'Cooking staff', 'Core staff', 'Drivers',
```

```
             'Realty agents', 'Security staff', 'Accountants',
             'Private service staff', 'Waiters/barmen staff', 'Medicine staff',
             'Cleaning staff', 'Low-skill Laborers', 'HR staff', 'IT staff'],
            dtype=object)
```

```
def segment_employment_type(value):
    if pd.isna(value):
        return 'Unknown'
    elif 'Sales' in value or 'Realty' in value:
        return 'Sales/Realty'
    elif 'Tech' in value or 'IT' in value:
        return 'Tech/IT'
    elif 'Secretaries' in value or 'HR' in value:
        return 'Secretaries/HR'
    elif 'Laborers' in value or 'Low-skill Laborers' in value:
        return 'Laborers'
    elif 'Managers' in value:
        return 'Managers'
    elif 'Cooking' in value or 'Waiters/barmen' in value:
        return 'Hospitality'
    else:
        return 'Other'
```

```
# Apply the function to create a new column
train['Type of Employment'] = train['Type of Employment'].apply(segment_employment_type)
```

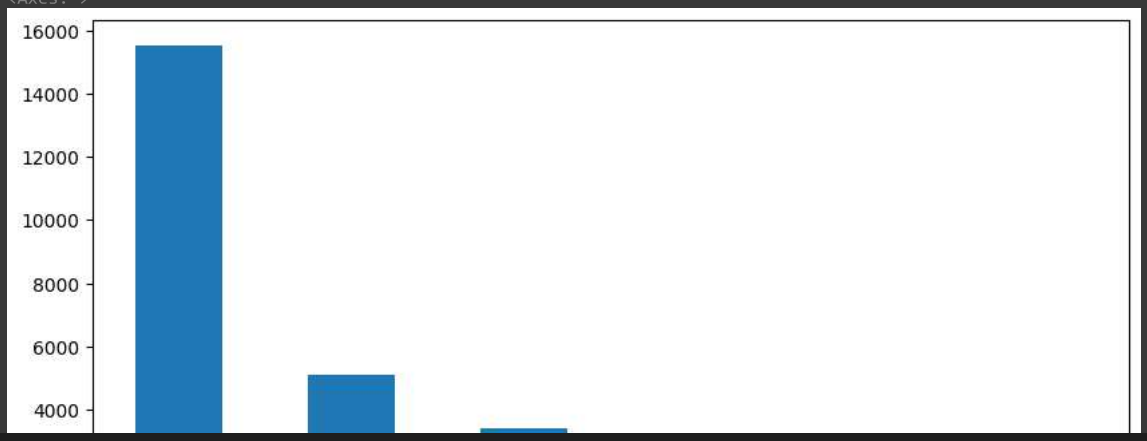
```
preprocess_data(train)
```

	Gender	Age	Income (USD)	Income Stability	Profession	Type of Employment	Location	Loan Amount Request (USD)	Current Loan Expenses (USD)	Expense Type 1	...	Credit Score	No. of Defaults	Has Active Credit Card
1	M	32	4952.910	Low	Working	Other	Semi-Urban	46837.47	495.81	N	...	780.40	0	Unpossessed
2	F	65	988.190	High	Pensioner	Other	Semi-Urban	45593.04	171.95	N	...	833.15	0	Unpossessed
3	F	65	2222.435	High	Pensioner	Other	Rural	80057.92	298.54	N	...	832.70	1	Unpossessed
4	F	31	2614.770	Low	Working	Other	Semi-Urban	113858.89	491.41	N	...	745.55	1	Active
5	F	60	1234.920	Low	State servant	Secretaries/HR	Rural	34434.72	181.48	N	...	684.12	1	Inactive
6	M	43	2361.560	Low	Working	Laborers	Semi-Urban	152561.34	697.67	Y	...	637.29	0	Unpossessed
7	F	45	2222.435	Low	State servant	Managers	Semi-Urban	240311.77	807.64	N	...	812.26	0	Active
8	F	38	1296.070	Low	Working	Other	Rural	35141.99	155.95	N	...	705.29	1	Active
9	M	18	1546.170	Low	Working	Laborers	Rural	42091.29	500.20	N	...	613.24	0	Unpossessed
10	M	18	2416.860	Low	State servant	Other	Semi-Urban	25765.72	140.02	N	...	652.41	0	Active

10 rows × 22 columns

```
# We visualize the column Type of Employment how many times it repeates
plt.figure(figsize=(10,5))
train['Type of Employment'].value_counts().plot(kind='bar')
```

```
<Axes: >
```



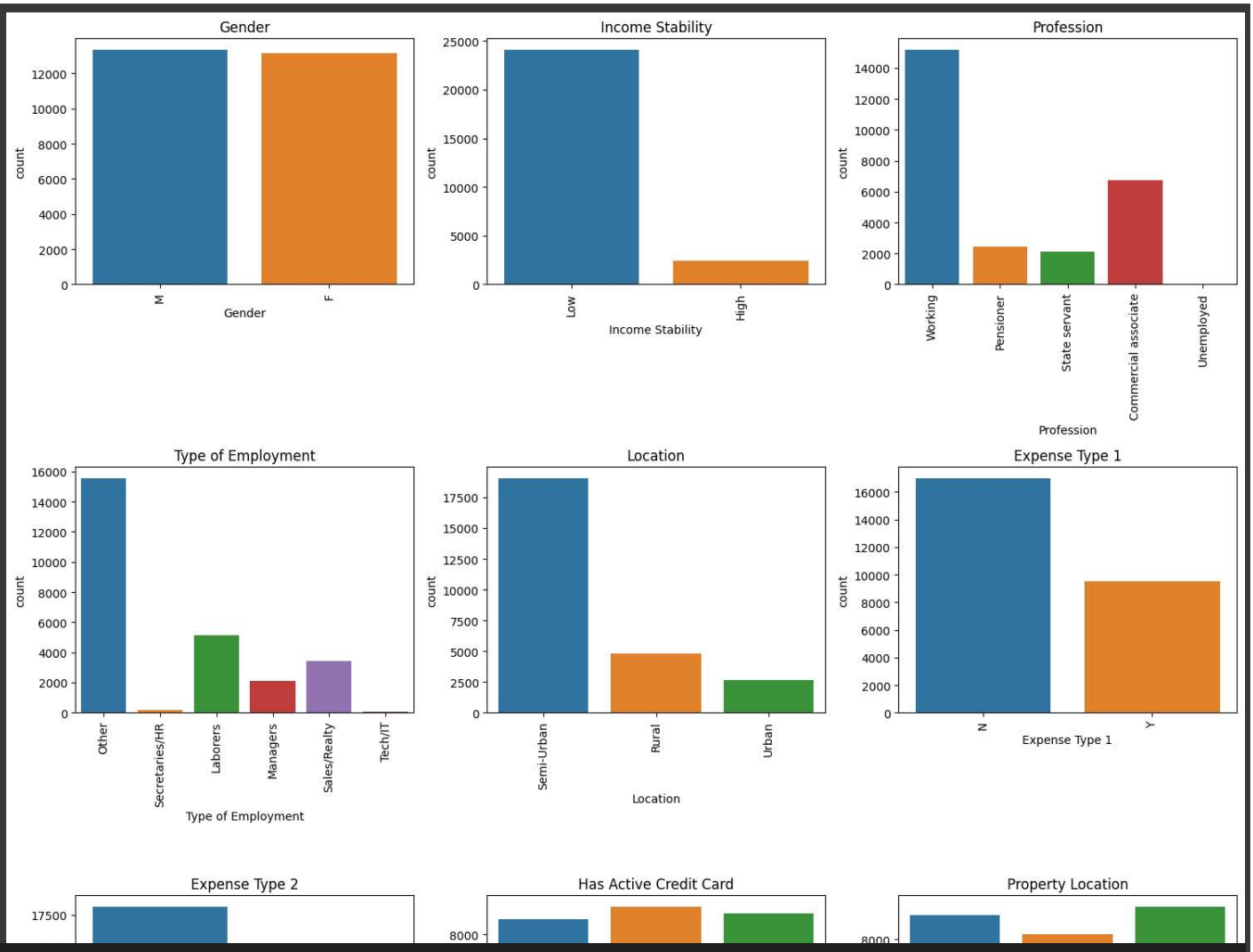

```
# Loop over each column in the DataFrame where dtype is 'object'
for col in train.select_dtypes(include=['object']).columns:

    # Print the column name and the unique values
    print(f"{col}: {train[col].unique()}")
```

```
Gender: ['M' 'F']
Income Stability: ['Low' 'High']
Profession: ['Working' 'Pensioner' 'State servant' 'Commercial associate' 'Unemployed']
Type of Employment: ['Other' 'Secretaries/HR' 'Laborers' 'Managers' 'Sales/Realty' 'Tech/IT']
Location: ['Semi-Urban' 'Rural' 'Urban']
Expense Type 1: ['N' 'Y']
Expense Type 2: ['Y' 'N']
Has Active Credit Card: ['Unpossessed' 'Active' 'Inactive']
Property Location: ['Rural' 'Urban' 'Semi-Urban']
```

Exploratory Data Analysis

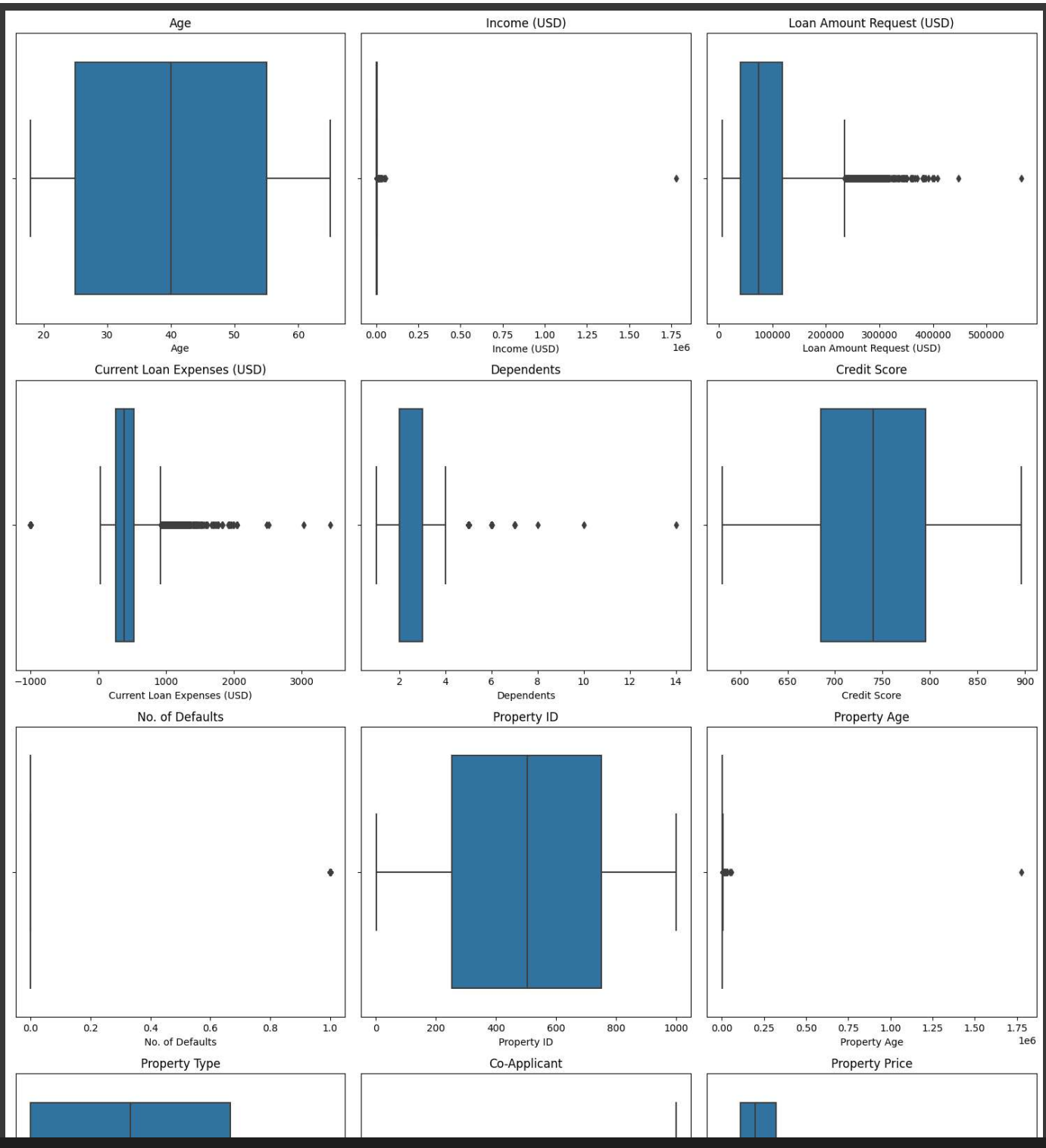
```
def explore_data(df):
  # Get the names of all columns with data type 'object' (categorical columns)
  cat_vars = df.select_dtypes(include='object').columns.tolist()

  # Create a figure with subplots
  num_cols = len(cat_vars)
  num_rows = (num_cols + 2) // 3
  fig, axs = plt.subplots(nrows=num_rows, ncols=3, figsize=(15, 5*num_rows))
  axs = axs.flatten()

  # Create a countplot for the top 6 values of each categorical variable using Seaborn
  for i, var in enumerate(cat_vars):
      top_values = df[var].value_counts().nlargest(6).index
      filtered_tf = df[df[var].isin(top_values)]
      sns.countplot(x=var, data=filtered_tf, ax=axs[i])
      axs[i].set_title(var)
      axs[i].tick_params(axis='x', rotation=90)

  # Remove any extra empty subplots if needed
  if num_cols < len(axs):
      for i in range(num_cols, len(axs)):
          fig.delaxes(axs[i])

  # Adjust spacing between subplots
  fig.tight_layout()

  # Show plot
  plt.show()
```

```
explore_data(train)
```




```
def eplore_2(df):
  # Get the names of all columns with data type 'int' or 'float' except 'cltv' and 'marital_status'
  num_vars = df.select_dtypes(include=['int', 'float']).columns.tolist()
  exclude_vars = ['Loan Sanction Amount (USD)']
  num_vars = [var for var in num_vars if var not in exclude_vars]

  # Create a figure with subplots
  num_cols = len(num_vars)
  num_rows = (num_cols + 2) // 3
  fig, axs = plt.subplots(nrows=num_rows, ncols=3, figsize=(15, 5*num_rows))
  axs = axs.flatten()

  # Create a box plot for each numerical variable using Seaborn
  for i, var in enumerate(num_vars):
      sns.boxplot(x=df[var], ax=axs[i])
      axs[i].set_title(var)

  # Remove any extra empty subplots if needed
  if num_cols < len(axs):
      for i in range(num_cols, len(axs)):
          fig.delaxes(axs[i])

  # Adjust spacing between subplots
  fig.tight_layout()

  # Show plot
  plt.show()
```

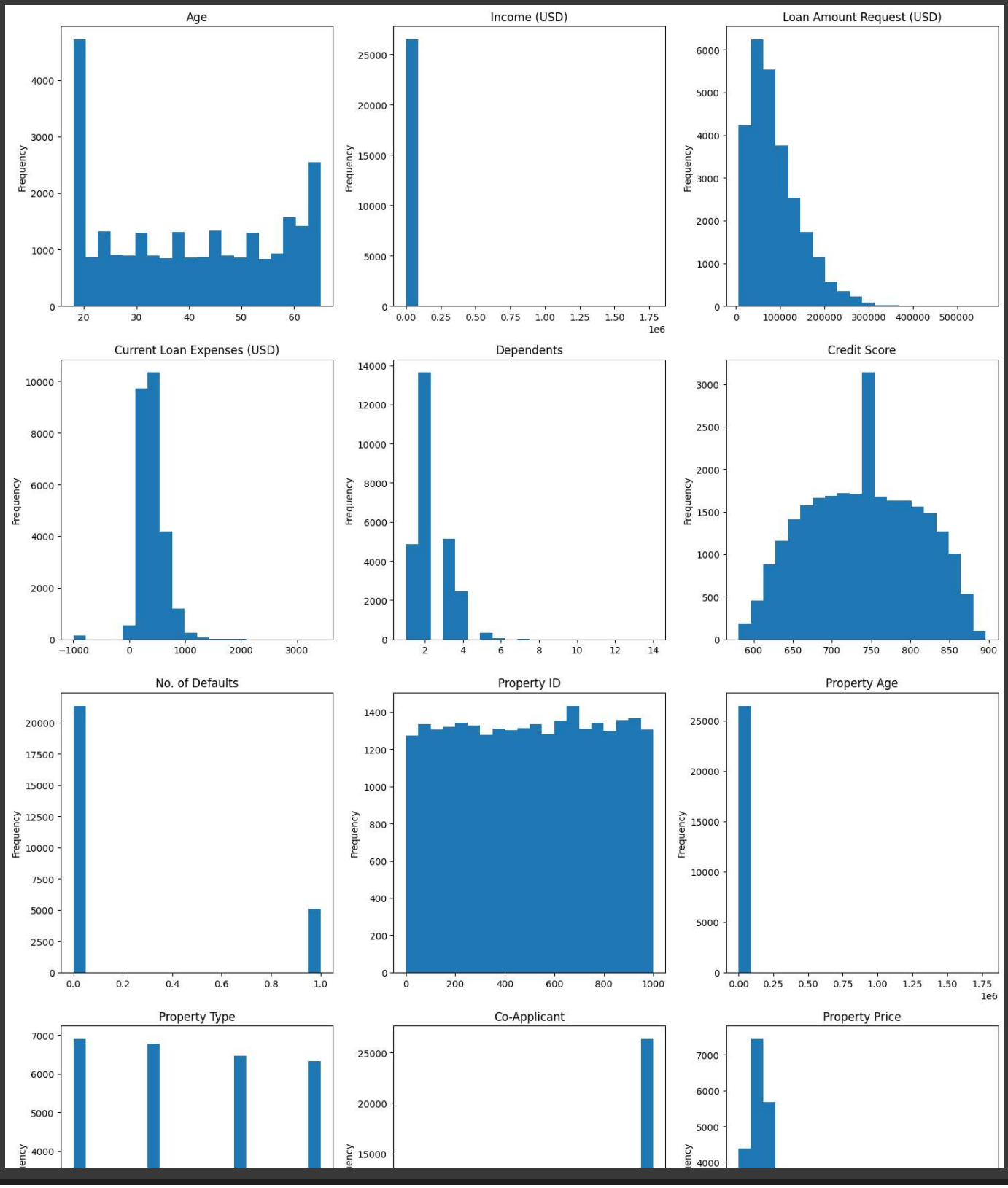
```
eplore_2(train)
```




```
def exprole_3(df):
  # Get the names of all columns with data type 'int' or 'float' except 'marital_status' and 'cltv'
  int_vars = df.select_dtypes(include=['int', 'float']).columns.tolist()
  exclude_vars = ['Loan Sanction Amount (USD)']
  int_vars = [var for var in int_vars if var not in exclude_vars]

  # Create a figure with subplots
  num_cols = len(int_vars)
  num_rows = (num_cols + 2) // 3  # To make sure there are enough rows for the subplots
  fig, axs = plt.subplots(nrows=num_rows, ncols=3, figsize=(15, 5*num_rows))
  axs = axs.flatten()

  # Create a histogram for each integer variable
  for i, var in enumerate(int_vars):
      df[var].plot.hist(ax=axs[i], bins=20)  # You can adjust the number of bins as needed
      axs[i].set_title(var)
```

```
  # Remove any extra empty subplots if needed
  if num_cols < len(axs):
      for i in range(num_cols, len(axs)):
          fig.delaxes(axs[i])

  # Adjust spacing between subplots
  fig.tight_layout()

  # Show plot
  plt.show()
```

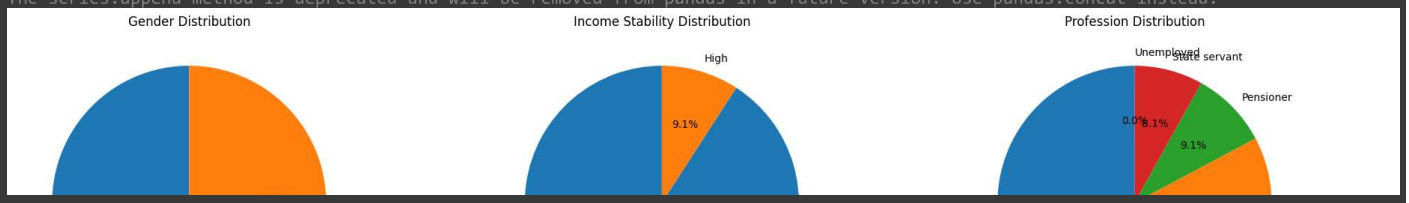
```
exprole_3(train)
```



```
def exprole_4(df):
  # Specify the maximum number of categories to show individually
  max_categories = 5

  # Filter categorical columns with 'object' data type
  cat_cols = [col for col in df.columns if col != 'y' and df[col].dtype == 'object']

  # Create a figure with subplots
  num_cols = len(cat_cols)
  num_rows = (num_cols + 2) // 3
  fig, axs = plt.subplots(nrows=num_rows, ncols=3, figsize=(20, 5*num_rows))

  # Flatten the axs array for easier indexing
  axs = axs.flatten()

  # Create a pie chart for each categorical column
  for i, col in enumerate(cat_cols):
      if i < len(axs):  # Ensure we don't exceed the number of subplots
          # Count the number of occurrences for each category
          cat_counts = df[col].value_counts()

          # Group categories beyond the top max_categories as 'Other'
          if len(cat_counts) > max_categories:
              cat_counts_top = cat_counts[:max_categories]
              cat_counts_other = pd.Series(cat_counts[max_categories:].sum(), index=['Other'])
              cat_counts = cat_counts_top.append(cat_counts_other)

          # Create a pie chart
          axs[i].pie(cat_counts, labels=cat_counts.index, autopct='%1.1f%%', startangle=90)
          axs[i].set_title(f'{col} Distribution')

  # Remove any extra empty subplots if needed
  if num_cols < len(axs):
      for i in range(num_cols, len(axs)):
          fig.delaxes(axs[i])

  # Adjust spacing between subplots
  fig.tight_layout()

  # Show plot
  plt.show()
```

```
exprole_4(train)
```

```
The series.append method is deprecated and will be removed from pandas in a future version. Use pandas.concat instead.
```



Data Preprocessing Part 2

```
# Check the amounnt of missing value
check_missing = train.isnull().sum() * 100 / train.shape[0]
check_missing[check_missing > 0].sort_values(ascending=False)
```

```
Series([], dtype: float64)
```

Label Encoding for Object Datatypes Encode categorical columns

```
## Label Encoding for Object Datatypes Encode categorical columns
def encoder_object(df):
    # Loop over each column in the DataFrame where dtype is 'object'
    for col in df.select_dtypes(include=['object']).columns:
        label_encoder = LabelEncoder()

        # Fit the encoder to the unique values in the column
        label_encoder.fit(df[col].unique())

        # Transform the column using the encoder
        df[col] = label_encoder.transform(df[col])

        # Print the column name and the unique encoded values
        print(f"{col}: {df[col].unique()}")
```

```
# Print the column name and the unique encoded values
encoder_object(train)
```

```
Gender: [1 0]
Income Stability: [1 0]
Profession: [4 1 2 0 3]
Type of Employment: [2 4 0 1 3 5]
Location: [1 0 2]
Expense Type 1: [0 1]
Expense Type 2: [1 0]
Has Active Credit Card: [2 0 1]
Property Location: [0 2 1]
```

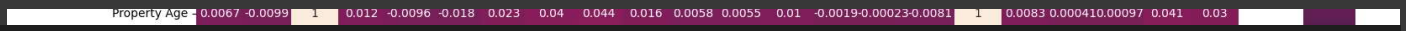
```
train.shape
```

```
(26474, 22)
```

```
# Correlation Heatmap
plt.figure(figsize=(20, 16))
sns.heatmap(train.corr(), fmt='.2g', annot=True)
```

<Axes: >

Remove Outlier from Train Data using Z-Score




```
def remove_outliers(df, columns_to_remove_outliers):
    # Remove outliers using Z-score
    z_scores = np.abs(stats.zscore(df[columns_to_remove_outliers]))
    threshold = 3
    outlier_indices = np.where(z_scores > threshold)[0]
    df = df.drop(df.index[outlier_indices])
    return df.head(10)
```

```
selected_columns = ['Income (USD)', 'Loan Amount Request (USD)', 'Current Loan Expenses (USD)',
                    'Dependents', 'Property Age', 'Co-Applicant', 'Property Price']
remove_outliers(train, selected_columns)
```

	Gender	Age	Income (USD)	Income Stability	Profession	Type of Employment	Location	Loan Amount Request (USD)	Current Loan Expenses (USD)	Expense Type 1	...	Credit Score	No. of Defaults	Has Active Credit Card	Prope
1	1	32	4952.910	1	4	2	1	46837.47	495.81	0	...	780.40	0	2	
2	0	65	988.190	0	1	2	1	45593.04	171.95	0	...	833.15	0	2	
3	0	65	2222.435	0	1	2	0	80057.92	298.54	0	...	832.70	1	2	
4	0	31	2614.770	1	4	2	1	113858.89	491.41	0	...	745.55	1	0	
5	0	60	1234.920	1	2	4	0	34434.72	181.48	0	...	684.12	1	1	
6	1	43	2361.560	1	4	0	1	152561.34	697.67	1	...	637.29	0	2	
7	0	45	2222.435	1	2	1	1	240311.77	807.64	0	...	812.26	0	0	
8	0	38	1296.070	1	4	2	0	35141.99	155.95	0	...	705.29	1	0	
9	1	18	1546.170	1	4	0	0	42091.29	500.20	0	...	613.24	0	2	
10	1	18	2416.860	1	2	2	1	25765.72	140.02	0	...	652.41	0	0	

10 rows × 22 columns

Train Test Split

Build the decision Tree

```
def train_decision_tree(df):
    X = df.drop('Loan Sanction Amount (USD)', axis=1)
    y = df['Loan Sanction Amount (USD)']

    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=0)

    # Create a DecisionTreeRegressor object
    dtree = DecisionTreeRegressor()

    # Define the hyperparameters to tune and their values
    param_grid = {
        'max_depth': [2, 4, 6, 8],
        'min_samples_split': [2, 4, 6, 8],
        'min_samples_leaf': [1, 2, 3, 4],
        'max_features': ['auto', 'sqrt', 'log2'],
        'random_state': [0, 42]
    }

    # Create a GridSearchCV object
    grid_search = GridSearchCV(dtree, param_grid, cv=5, scoring='neg_mean_squared_error')

    # Fit the GridSearchCV object to the data
    grid_search.fit(X_train, y_train)

    # Print the best hyperparameters
    print(grid_search.best_params_)

    #The decission tree

    dtree = DecisionTreeRegressor(random_state=42, max_depth=6, max_features='auto', min_samples_leaf=4, min_samples_split=2)
    dtree.fit(X_train, y_train)

    return dtree, X_test, y_test, X_train,y_train
```

```
dtree, X_test, y_test,X_train,y_train  = train_decision_tree(train)
```

```
`max_features='auto'` has been deprecated in 1.1 and will be removed in 1.3. To keep the past behaviour, explicitly set `max_features
`max_features='auto'` has been deprecated in 1.1 and will be removed in 1.3. To keep the past behaviour, explicitly set `max_features
`max_features='auto'` has been deprecated in 1.1 and will be removed in 1.3. To keep the past behaviour, explicitly set `max_features
`max_features='auto'` has been deprecated in 1.1 and will be removed in 1.3. To keep the past behaviour, explicitly set `max_features
`max_features='auto'` has been deprecated in 1.1 and will be removed in 1.3. To keep the past behaviour, explicitly set `max_features
`max_features='auto'` has been deprecated in 1.1 and will be removed in 1.3. To keep the past behaviour, explicitly set `max_features
`max_features='auto'` has been deprecated in 1.1 and will be removed in 1.3. To keep the past behaviour, explicitly set `max_features
`max_features='auto'` has been deprecated in 1.1 and will be removed in 1.3. To keep the past behaviour, explicitly set `max_features
`max_features='auto'` has been deprecated in 1.1 and will be removed in 1.3. To keep the past behaviour, explicitly set `max_features
`max_features='auto'` has been deprecated in 1.1 and will be removed in 1.3. To keep the past behaviour, explicitly set `max_features
`max_features='auto'` has been deprecated in 1.1 and will be removed in 1.3. To keep the past behaviour, explicitly set `max_features
`max_features='auto'` has been deprecated in 1.1 and will be removed in 1.3. To keep the past behaviour, explicitly set `max_features
`max_features='auto'` has been deprecated in 1.1 and will be removed in 1.3. To keep the past behaviour, explicitly set `max_features
`max_features='auto'` has been deprecated in 1.1 and will be removed in 1.3. To keep the past behaviour, explicitly set `max_features
`max_features='auto'` has been deprecated in 1.1 and will be removed in 1.3. To keep the past behaviour, explicitly set `max_features
{'max_depth': 8, 'max_features': 'auto', 'min_samples_leaf': 3, 'min_samples_split': 8, 'random_state': 0}
```

Evaluate the decision tree

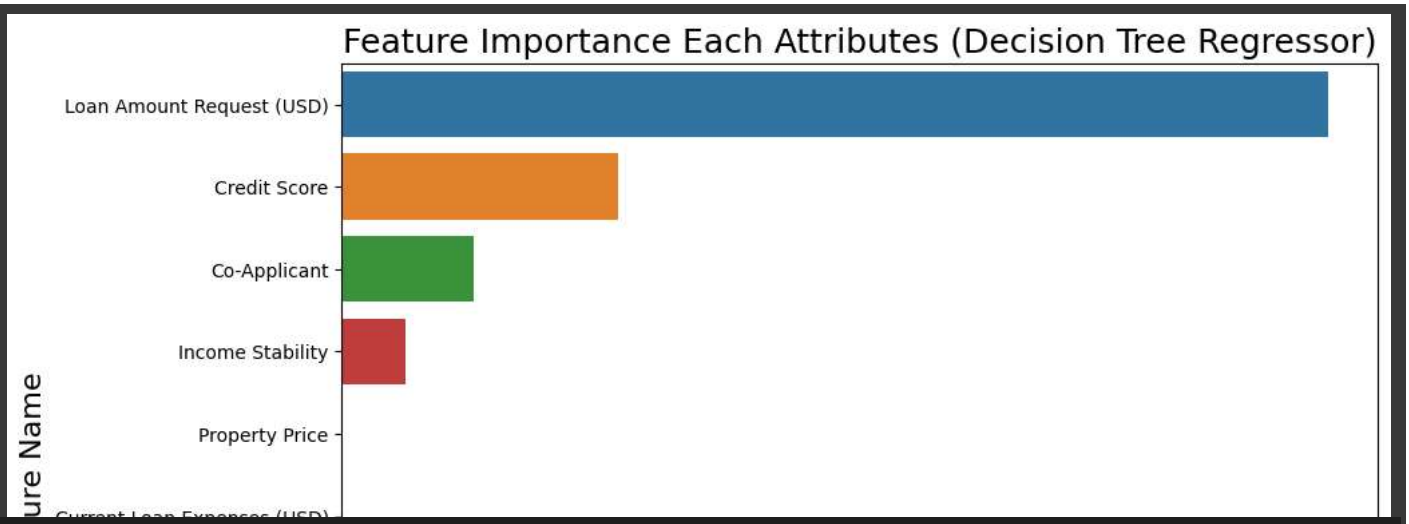
```
def evaluate_decision_tree(model, X_test, y_test):
    y_pred = model.predict(X_test)
    mae = mean_absolute_error(y_test, y_pred)
    mape = mean_absolute_percentage_error(y_test, y_pred)
    mse = mean_squared_error(y_test, y_pred)
    r2 = r2_score(y_test, y_pred)
    rmse = math.sqrt(mse)

    print('MAE is {}'.format(mae))
    print('MAPE is {}'.format(mape))
    print('MSE is {}'.format(mse))
    print('R2 score is {}'.format(r2))
    print('RMSE score is {}'.format(rmse))
```

```
print("Decision Tree Model:")
evaluate_decision_tree(dtree, X_test, y_test)
```

```
Decision Tree Model:
MAE is 12651.787227844097
MAPE is 2.3472735407383396e+19
MSE is 591410680.1464931
R2 score is 0.7426083343470093
RMSE score is 24318.936657397113
```

Plot The Featrue Impontance

```
imp_df = pd.DataFrame({
    "Feature Name": X_train.columns,
    "Importance": dtree.feature_importances_
})
fi = imp_df.sort_values(by="Importance", ascending=False)

fi2 = fi.head(10)
plt.figure(figsize=(10,8))
sns.barplot(data=fi2, x='Importance', y='Feature Name')
plt.title('Feature Importance Each Attributes (Decision Tree Regressor)', fontsize=18)
plt.xlabel ('Importance', fontsize=16)
plt.ylabel ('Feature Name', fontsize=16)
plt.show()
```



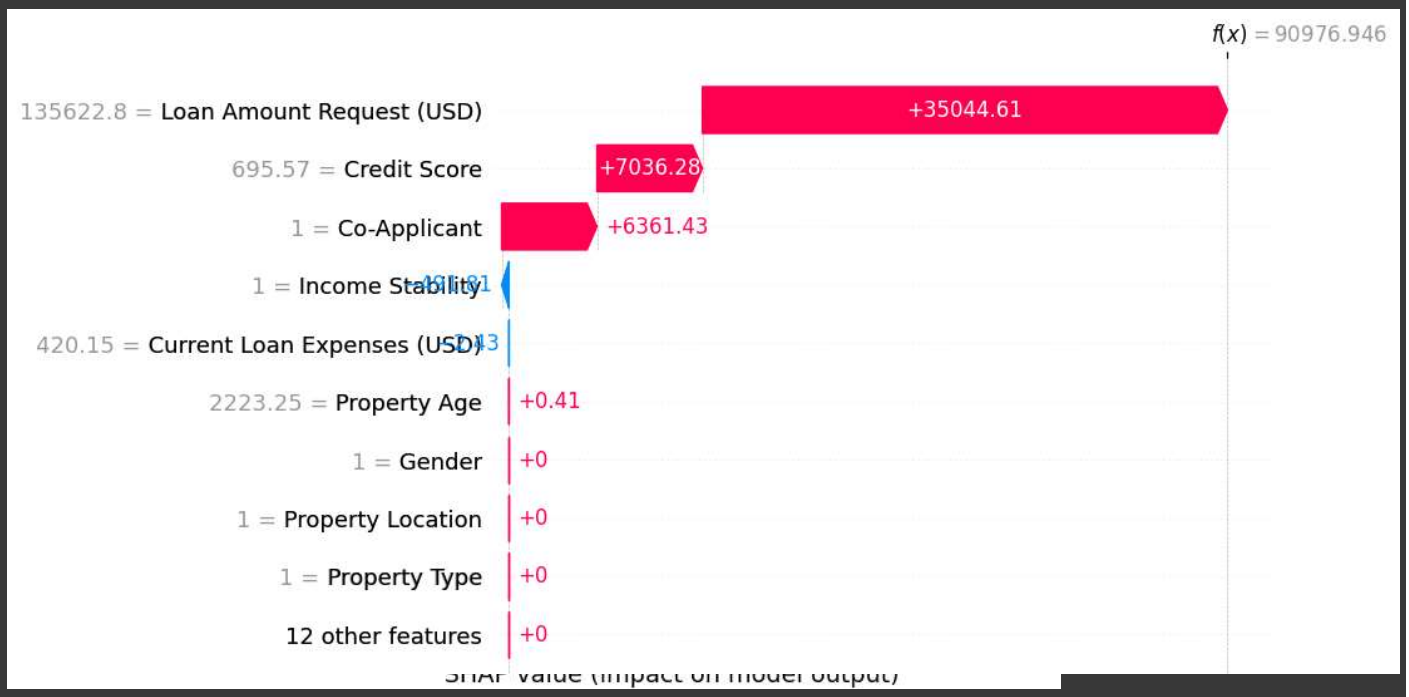

```
!pip install shap
```

```
Requirement already satisfied: shap in /usr/local/lib/python3.10/dist-packages (0.42.1)
Requirement already satisfied: numpy in /usr/local/lib/python3.10/dist-packages (from shap) (1.23.5)
Requirement already satisfied: scipy in /usr/local/lib/python3.10/dist-packages (from shap) (1.11.3)
Requirement already satisfied: scikit-learn in /usr/local/lib/python3.10/dist-packages (from shap) (1.2.2)
Requirement already satisfied: pandas in /usr/local/lib/python3.10/dist-packages (from shap) (1.5.3)
Requirement already satisfied: tqdm>=4.27.0 in /usr/local/lib/python3.10/dist-packages (from shap) (4.66.1)
Requirement already satisfied: packaging>20.9 in /usr/local/lib/python3.10/dist-packages (from shap) (23.2)
Requirement already satisfied: slicer==0.0.7 in /usr/local/lib/python3.10/dist-packages (from shap) (0.0.7)
Requirement already satisfied: numba in /usr/local/lib/python3.10/dist-packages (from shap) (0.56.4)
Requirement already satisfied: cloudpickle in /usr/local/lib/python3.10/dist-packages (from shap) (2.2.1)
Requirement already satisfied: llvmlite<0.40,>=0.39.0dev0 in /usr/local/lib/python3.10/dist-packages (from numba->shap) (0.39.1)
Requirement already satisfied: setuptools in /usr/local/lib/python3.10/dist-packages (from numba->shap) (67.7.2)
Requirement already satisfied: python-dateutil>=2.8.1 in /usr/local/lib/python3.10/dist-packages (from pandas->shap) (2.8.2)
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.10/dist-packages (from pandas->shap) (2023.3.post1)
Requirement already satisfied: joblib>=1.1.1 in /usr/local/lib/python3.10/dist-packages (from scikit-learn->shap) (1.3.2)
Requirement already satisfied: threadpoolctl>=2.0.0 in /usr/local/lib/python3.10/dist-packages (from scikit-learn->shap) (3.2.0)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.10/dist-packages (from python-dateutil>=2.8.1->pandas->shap) (1.16.0)
```

```
import shap
explainer = shap.TreeExplainer(dtree)
shap_values = explainer.shap_values(X_test)
shap.summary_plot(shap_values, X_test)
```

```
No data for colormapping provided via 'c'. Parameters 'vmin', 'vmax' will be ignored
```


```
explainer = shap.Explainer(dtree, X_test)
shap_values = explainer(X_test, check_additivity=False)
shap.plots.waterfall(shap_values[0])
```



Hyperparameter Tuning

```
from sklearn.ensemble import RandomForestRegressor
def tune_random_forest(X_train, y_train):
    # Create a Random Forest Regressor object
    rf = RandomForestRegressor(max_features=1.0)  # Set max_features explicitly

    # Define the hyperparameter grid
    param_grid = {
        'max_depth': [3, 5, 7, 9],
        'min_samples_split': [2, 5, 10],
        'min_samples_leaf': [1, 2, 4],
        'max_features': ['auto', 'sqrt'],
        'random_state': [0, 42]
    }

    # Create a GridSearchCV object
    grid_search = GridSearchCV(rf, param_grid, cv=5, scoring='r2')

    # Fit the GridSearchCV object to the training data
    grid_search.fit(X_train, y_train)

    # Print the best hyperparameters
    best_params = grid_search.best_params_
```

```
    print("Best hyperparameters: ", best_params)

    # Return the best model with tuned hyperparameters
    best_rf_model = grid_search.best_estimator_
    return best_rf_model
```

```
best_rf_model = tune_random_forest(X_train, y_train)
best_rf_model.fit(X_train, y_train)
```

```
`max_features='auto'` has been deprecated in 1.1 and will be removed in 1.3. To keep the past behaviour, explicitly set `max_features=1.
`max_features='auto'` has been deprecated in 1.1 and will be removed in 1.3. To keep the past behaviour, explicitly set `max_features=1.
`max_features='auto'` has been deprecated in 1.1 and will be removed in 1.3. To keep the past behaviour, explicitly set `max_features=1.
`max_features='auto'` has been deprecated in 1.1 and will be removed in 1.3. To keep the past behaviour, explicitly set `max_features=1.
`max_features='auto'` has been deprecated in 1.1 and will be removed in 1.3. To keep the past behaviour, explicitly set `max_features=1.
`max_features='auto'` has been deprecated in 1.1 and will be removed in 1.3. To keep the past behaviour, explicitly set `max_features=1.
`max_features='auto'` has been deprecated in 1.1 and will be removed in 1.3. To keep the past behaviour, explicitly set `max_features=1.
`max_features='auto'` has been deprecated in 1.1 and will be removed in 1.3. To keep the past behaviour, explicitly set `max_features=1.
`max_features='auto'` has been deprecated in 1.1 and will be removed in 1.3. To keep the past behaviour, explicitly set `max_features=1.
`max_features='auto'` has been deprecated in 1.1 and will be removed in 1.3. To keep the past behaviour, explicitly set `max_features=1.
`max_features='auto'` has been deprecated in 1.1 and will be removed in 1.3. To keep the past behaviour, explicitly set `max_features=1.
`max_features='auto'` has been deprecated in 1.1 and will be removed in 1.3. To keep the past behaviour, explicitly set `max_features=1.
`max_features='auto'` has been deprecated in 1.1 and will be removed in 1.3. To keep the past behaviour, explicitly set `max_features=1.
`max_features='auto'` has been deprecated in 1.1 and will be removed in 1.3. To keep the past behaviour, explicitly set `max_features=1.
`max_features='auto'` has been deprecated in 1.1 and will be removed in 1.3. To keep the past behaviour, explicitly set `max_features=1.
`max_features='auto'` has been deprecated in 1.1 and will be removed in 1.3. To keep the past behaviour, explicitly set `max_features=1.
`max_features='auto'` has been deprecated in 1.1 and will be removed in 1.3. To keep the past behaviour, explicitly set `max_features=1.
`max_features='auto'` has been deprecated in 1.1 and will be removed in 1.3. To keep the past behaviour, explicitly set `max_features=1.
`max_features='auto'` has been deprecated in 1.1 and will be removed in 1.3. To keep the past behaviour, explicitly set `max_features=1.
`max_features='auto'` has been deprecated in 1.1 and will be removed in 1.3. To keep the past behaviour, explicitly set `max_features=1.
`max_features='auto'` has been deprecated in 1.1 and will be removed in 1.3. To keep the past behaviour, explicitly set `max_features=1.
`max_features='auto'` has been deprecated in 1.1 and will be removed in 1.3. To keep the past behaviour, explicitly set `max_features=1.
`max_features='auto'` has been deprecated in 1.1 and will be removed in 1.3. To keep the past behaviour, explicitly set `max_features=1.
`max_features='auto'` has been deprecated in 1.1 and will be removed in 1.3. To keep the past behaviour, explicitly set `max_features=1.
`max_features='auto'` has been deprecated in 1.1 and will be removed in 1.3. To keep the past behaviour, explicitly set `max_features=1.
`max_features='auto'` has been deprecated in 1.1 and will be removed in 1.3. To keep the past behaviour, explicitly set `max_features=1.
`max_features='auto'` has been deprecated in 1.1 and will be removed in 1.3. To keep the past behaviour, explicitly set `max_features=1.
`max_features='auto'` has been deprecated in 1.1 and will be removed in 1.3. To keep the past behaviour, explicitly set `max_features=1.
`max_features='auto'` has been deprecated in 1.1 and will be removed in 1.3. To keep the past behaviour, explicitly set `max_features=1.
`max_features='auto'` has been deprecated in 1.1 and will be removed in 1.3. To keep the past behaviour, explicitly set `max_features=1.
`max_features='auto'` has been deprecated in 1.1 and will be removed in 1.3. To keep the past behaviour, explicitly set `max_features=1.
`max_features='auto'` has been deprecated in 1.1 and will be removed in 1.3. To keep the past behaviour, explicitly set `max_features=1.
`max_features='auto'` has been deprecated in 1.1 and will be removed in 1.3. To keep the past behaviour, explicitly set `max_features=1.
`max_features='auto'` has been deprecated in 1.1 and will be removed in 1.3. To keep the past behaviour, explicitly set `max_features=1.
`max_features='auto'` has been deprecated in 1.1 and will be removed in 1.3. To keep the past behaviour, explicitly set `max_features=1.
`max_features='auto'` has been deprecated in 1.1 and will be removed in 1.3. To keep the past behaviour, explicitly set `max_features=1.
`max_features='auto'` has been deprecated in 1.1 and will be removed in 1.3. To keep the past behaviour, explicitly set `max_features=1.
`max_features='auto'` has been deprecated in 1.1 and will be removed in 1.3. To keep the past behaviour, explicitly set `max_features=1.
`max_features='auto'` has been deprecated in 1.1 and will be removed in 1.3. To keep the past behaviour, explicitly set `max_features=1.
`max_features='auto'` has been deprecated in 1.1 and will be removed in 1.3. To keep the past behaviour, explicitly set `max_features=1.
`max_features='auto'` has been deprecated in 1.1 and will be removed in 1.3. To keep the past behaviour, explicitly set `max_features=1.
`max_features='auto'` has been deprecated in 1.1 and will be removed in 1.3. To keep the past behaviour, explicitly set `max_features=1.
`max_features='auto'` has been deprecated in 1.1 and will be removed in 1.3. To keep the past behaviour, explicitly set `max_features=1.
`max_features='auto'` has been deprecated in 1.1 and will be removed in 1.3. To keep the past behaviour, explicitly set `max_features=1.
`max_features='auto'` has been deprecated in 1.1 and will be removed in 1.3. To keep the past behaviour, explicitly set `max_features=1.
`max_features='auto'` has been deprecated in 1.1 and will be removed in 1.3. To keep the past behaviour, explicitly set `max_features=1.
`max_features='auto'` has been deprecated in 1.1 and will be removed in 1.3. To keep the past behaviour, explicitly set `max_features=1.
`max_features='auto'` has been deprecated in 1.1 and will be removed in 1.3. To keep the past behaviour, explicitly set `max_features=1.
`max_features='auto'` has been deprecated in 1.1 and will be removed in 1.3. To keep the past behaviour, explicitly set `max_features=1.
`max_features='auto'` has been deprecated in 1.1 and will be removed in 1.3. To keep the past behaviour, explicitly set `max_features=1.
`max_features='auto'` has been deprecated in 1.1 and will be removed in 1.3. To keep the past behaviour, explicitly set `max_features=1.
`max_features='auto'` has been deprecated in 1.1 and will be removed in 1.3. To keep the past behaviour, explicitly set `max_features=1.
`max_features='auto'` has been deprecated in 1.1 and will be removed in 1.3. To keep the past behaviour, explicitly set `max_features=1.
`max_features='auto'` has been deprecated in 1.1 and will be removed in 1.3. To keep the past behaviour, explicitly set `max_features=1.
`max_features='auto'` has been deprecated in 1.1 and will be removed in 1.3. To keep the past behaviour, explicitly set `max_features=1.
`max_features='auto'` has been deprecated in 1.1 and will be removed in 1.3. To keep the past behaviour, explicitly set `max_features=1.
`max_features='auto'` has been deprecated in 1.1 and will be removed in 1.3. To keep the past behaviour, explicitly set `max_features=1.
`max_features='auto'` has been deprecated in 1.1 and will be removed in 1.3. To keep the past behaviour, explicitly set `max_features=1.
`max_features='auto'` has been deprecated in 1.1 and will be removed in 1.3. To keep the past behaviour, explicitly set `max_features=1.
`max_features='auto'` has been deprecated in 1.1 and will be removed in 1.3. To keep the past behaviour, explicitly set `max_features=1.
`max_features='auto'` has been deprecated in 1.1 and will be removed in 1.3. To keep the past behaviour, explicitly set `max_features=1.
`max_features='auto'` has been deprecated in 1.1 and will be removed in 1.3. To keep the past behaviour, explicitly set `max_features=1.
`max_features='auto'` has been deprecated in 1.1 and will be removed in 1.3. To keep the past behaviour, explicitly set `max_features=1.
`max_features='auto'` has been deprecated in 1.1 and will be removed in 1.3. To keep the past behaviour, explicitly set `max_features=1.
`max_features='auto'` has been deprecated in 1.1 and will be removed in 1.3. To keep the past behaviour, explicitly set `max_features=1.
`max_features='auto'` has been deprecated in 1.1 and will be removed in 1.3. To keep the past behaviour, explicitly set `max_features=1.
`max_features='auto'` has been deprecated in 1.1 and will be removed in 1.3. To keep the past behaviour, explicitly set `max_features=1.
`max_features='auto'` has been deprecated in 1.1 and will be removed in 1.3. To keep the past behaviour, explicitly set `max_features=1.
`max_features='auto'` has been deprecated in 1.1 and will be removed in 1.3. To keep the past behaviour, explicitly set `max_features=1.
`max_features='auto'` has been deprecated in 1.1 and will be removed in 1.3. To keep the past behaviour, explicitly set `max_features=1.
`max_features='auto'` has been deprecated in 1.1 and will be removed in 1.3. To keep the past behaviour, explicitly set `max_features=1.
`max_features='auto'` has been deprecated in 1.1 and will be removed in 1.3. To keep the past behaviour, explicitly set `max_features=1.
`max_features='auto'` has been deprecated in 1.1 and will be removed in 1.3. To keep the past behaviour, explicitly set `max_features=1.
`max_features='auto'` has been deprecated in 1.1 and will be removed in 1.3. To keep the past behaviour, explicitly set `max_features=1.
`max_features='auto'` has been deprecated in 1.1 and will be removed in 1.3. To keep the past behaviour, explicitly set `max_features=1.
`max_features='auto'` has been deprecated in 1.1 and will be removed in 1.3. To keep the past behaviour, explicitly set `max_features=1.
`max_features='auto'` has been deprecated in 1.1 and will be removed in 1.3. To keep the past behaviour, explicitly set `max_features=1.
`max_features='auto'` has been deprecated in 1.1 and will be removed in 1.3. To keep the past behaviour, explicitly set `max_features=1.
`max_features='auto'` has been deprecated in 1.1 and will be removed in 1.3. To keep the past behaviour, explicitly set `max_features=1.
`max_features='auto'` has been deprecated in 1.1 and will be removed in 1.3. To keep the past behaviour, explicitly set `max_features=1.
`max_features='auto'` has been deprecated in 1.1 and will be removed in 1.3. To keep the past behaviour, explicitly set `max_features=1.
`max_features='auto'` has been deprecated in 1.1 and will be removed in 1.3. To keep the past behaviour, explicitly set `max_features=1.
`max_features='auto'` has been deprecated in 1.1 and will be removed in 1.3. To keep the past behaviour, explicitly set `max_features=1.
`max_features='auto'` has been deprecated in 1.1 and will be removed in 1.3. To keep the past behaviour, explicitly set `max_features=1.
```

```
`max_features='auto'` has been deprecated in 1.1 and will be removed in 1.3. To keep the past behaviour, explicitly set `max_features=1.
```

```
`max_features='auto'` has been deprecated in 1.1 and will be removed in 1.3. To keep the past behaviour, explicitly set `max_features=1.
`max_features='auto'` has been deprecated in 1.1 and will be removed in 1.3. To keep the past behaviour, explicitly set `max_features=1.
`max_features='auto'` has been deprecated in 1.1 and will be removed in 1.3. To keep the past behaviour, explicitly set `max_features=1.
`max_features='auto'` has been deprecated in 1.1 and will be removed in 1.3. To keep the past behaviour, explicitly set `max_features=1.
`max_features='auto'` has been deprecated in 1.1 and will be removed in 1.3. To keep the past behaviour, explicitly set `max_features=1.
`max_features='auto'` has been deprecated in 1.1 and will be removed in 1.3. To keep the past behaviour, explicitly set `max_features=1.
`max_features='auto'` has been deprecated in 1.1 and will be removed in 1.3. To keep the past behaviour, explicitly set `max_features=1.
`max_features='auto'` has been deprecated in 1.1 and will be removed in 1.3. To keep the past behaviour, explicitly set `max_features=1.
`max_features='auto'` has been deprecated in 1.1 and will be removed in 1.3. To keep the past behaviour, explicitly set `max_features=1.
`max_features='auto'` has been deprecated in 1.1 and will be removed in 1.3. To keep the past behaviour, explicitly set `max_features=1.
`max_features='auto'` has been deprecated in 1.1 and will be removed in 1.3. To keep the past behaviour, explicitly set `max_features=1.
`max_features='auto'` has been deprecated in 1.1 and will be removed in 1.3. To keep the past behaviour, explicitly set `max_features=1.
`max_features='auto'` has been deprecated in 1.1 and will be removed in 1.3. To keep the past behaviour, explicitly set `max_features=1.
`max_features='auto'` has been deprecated in 1.1 and will be removed in 1.3. To keep the past behaviour, explicitly set `max_features=1.
`max_features='auto'` has been deprecated in 1.1 and will be removed in 1.3. To keep the past behaviour, explicitly set `max_features=1.
`max_features='auto'` has been deprecated in 1.1 and will be removed in 1.3. To keep the past behaviour, explicitly set `max_features=1.
`max_features='auto'` has been deprecated in 1.1 and will be removed in 1.3. To keep the past behaviour, explicitly set `max_features=1.
`max_features='auto'` has been deprecated in 1.1 and will be removed in 1.3. To keep the past behaviour, explicitly set `max_features=1.
`max_features='auto'` has been deprecated in 1.1 and will be removed in 1.3. To keep the past behaviour, explicitly set `max_features=1.
`max_features='auto'` has been deprecated in 1.1 and will be removed in 1.3. To keep the past behaviour, explicitly set `max_features=1.
`max_features='auto'` has been deprecated in 1.1 and will be removed in 1.3. To keep the past behaviour, explicitly set `max_features=1.
`max_features='auto'` has been deprecated in 1.1 and will be removed in 1.3. To keep the past behaviour, explicitly set `max_features=1.
`max_features='auto'` has been deprecated in 1.1 and will be removed in 1.3. To keep the past behaviour, explicitly set `max_features=1.
`max_features='auto'` has been deprecated in 1.1 and will be removed in 1.3. To keep the past behaviour, explicitly set `max_features=1.
`max_features='auto'` has been deprecated in 1.1 and will be removed in 1.3. To keep the past behaviour, explicitly set `max_features=1.
`max_features='auto'` has been deprecated in 1.1 and will be removed in 1.3. To keep the past behaviour, explicitly set `max_features=1.
`max_features='auto'` has been deprecated in 1.1 and will be removed in 1.3. To keep the past behaviour, explicitly set `max_features=1.
`max_features='auto'` has been deprecated in 1.1 and will be removed in 1.3. To keep the past behaviour, explicitly set `max_features=1.
`max_features='auto'` has been deprecated in 1.1 and will be removed in 1.3. To keep the past behaviour, explicitly set `max_features=1.
`max_features='auto'` has been deprecated in 1.1 and will be removed in 1.3. To keep the past behaviour, explicitly set `max_features=1.
`max_features='auto'` has been deprecated in 1.1 and will be removed in 1.3. To keep the past behaviour, explicitly set `max_features=1.
`max_features='auto'` has been deprecated in 1.1 and will be removed in 1.3. To keep the past behaviour, explicitly set `max_features=1.
`max_features='auto'` has been deprecated in 1.1 and will be removed in 1.3. To keep the past behaviour, explicitly set `max_features=1.
`max_features='auto'` has been deprecated in 1.1 and will be removed in 1.3. To keep the past behaviour, explicitly set `max_features=1.
`max_features='auto'` has been deprecated in 1.1 and will be removed in 1.3. To keep the past behaviour, explicitly set `max_features=1.
`max_features='auto'` has been deprecated in 1.1 and will be removed in 1.3. To keep the past behaviour, explicitly set `max_features=1.
`max_features='auto'` has been deprecated in 1.1 and will be removed in 1.3. To keep the past behaviour, explicitly set `max_features=1.
`max_features='auto'` has been deprecated in 1.1 and will be removed in 1.3. To keep the past behaviour, explicitly set `max_features=1.
`max_features='auto'` has been deprecated in 1.1 and will be removed in 1.3. To keep the past behaviour, explicitly set `max_features=1.
`max_features='auto'` has been deprecated in 1.1 and will be removed in 1.3. To keep the past behaviour, explicitly set `max_features=1.
`max_features='auto'` has been deprecated in 1.1 and will be removed in 1.3. To keep the past behaviour, explicitly set `max_features=1.
`max_features='auto'` has been deprecated in 1.1 and will be removed in 1.3. To keep the past behaviour, explicitly set `max_features=1.
`max_features='auto'` has been deprecated in 1.1 and will be removed in 1.3. To keep the past behaviour, explicitly set `max_features=1.
`max_features='auto'` has been deprecated in 1.1 and will be removed in 1.3. To keep the past behaviour, explicitly set `max_features=1.
`max_features='auto'` has been deprecated in 1.1 and will be removed in 1.3. To keep the past behaviour, explicitly set `max_features=1.
`max_features='auto'` has been deprecated in 1.1 and will be removed in 1.3. To keep the past behaviour, explicitly set `max_features=1.
`max_features='auto'` has been deprecated in 1.1 and will be removed in 1.3. To keep the past behaviour, explicitly set `max_features=1.
`max_features='auto'` has been deprecated in 1.1 and will be removed in 1.3. To keep the past behaviour, explicitly set `max_features=1.
`max_features='auto'` has been deprecated in 1.1 and will be removed in 1.3. To keep the past behaviour, explicitly set `max_features=1.
`max_features='auto'` has been deprecated in 1.1 and will be removed in 1.3. To keep the past behaviour, explicitly set `max_features=1.
`max_features='auto'` has been deprecated in 1.1 and will be removed in 1.3. To keep the past behaviour, explicitly set `max_features=1.
`max_features='auto'` has been deprecated in 1.1 and will be removed in 1.3. To keep the past behaviour, explicitly set `max_features=1.
`max_features='auto'` has been deprecated in 1.1 and will be removed in 1.3. To keep the past behaviour, explicitly set `max_features=1.
`max_features='auto'` has been deprecated in 1.1 and will be removed in 1.3. To keep the past behaviour, explicitly set `max_features=1.
`max_features='auto'` has been deprecated in 1.1 and will be removed in 1.3. To keep the past behaviour, explicitly set `max_features=1.
`max_features='auto'` has been deprecated in 1.1 and will be removed in 1.3. To keep the past behaviour, explicitly set `max_features=1.
`max_features='auto'` has been deprecated in 1.1 and will be removed in 1.3. To keep the past behaviour, explicitly set `max_features=1.
`max_features='auto'` has been deprecated in 1.1 and will be removed in 1.3. To keep the past behaviour, explicitly set `max_features=1.
`max_features='auto'` has been deprecated in 1.1 and will be removed in 1.3. To keep the past behaviour, explicitly set `max_features=1.
`max_features='auto'` has been deprecated in 1.1 and will be removed in 1.3. To keep the past behaviour, explicitly set `max_features=1.
`max_features='auto'` has been deprecated in 1.1 and will be removed in 1.3. To keep the past behaviour, explicitly set `max_features=1.
`max_features='auto'` has been deprecated in 1.1 and will be removed in 1.3. To keep the past behaviour, explicitly set `max_features=1.
`max_features='auto'` has been deprecated in 1.1 and will be removed in 1.3. To keep the past behaviour, explicitly set `max_features=1.
`max_features='auto'` has been deprecated in 1.1 and will be removed in 1.3. To keep the past behaviour, explicitly set `max_features=1.
`max_features='auto'` has been deprecated in 1.1 and will be removed in 1.3. To keep the past behaviour, explicitly set `max_features=1.
`max_features='auto'` has been deprecated in 1.1 and will be removed in 1.3. To keep the past behaviour, explicitly set `max_features=1.
`max_features='auto'` has been deprecated in 1.1 and will be removed in 1.3. To keep the past behaviour, explicitly set `max_features=1.
`max_features='auto'` has been deprecated in 1.1 and will be removed in 1.3. To keep the past behaviour, explicitly set `max_features=1.
`max_features='auto'` has been deprecated in 1.1 and will be removed in 1.3. To keep the past behaviour, explicitly set `max_features=1.
`max_features='auto'` has been deprecated in 1.1 and will be removed in 1.3. To keep the past behaviour, explicitly set `max_features=1.
`max_features='auto'` has been deprecated in 1.1 and will be removed in 1.3. To keep the past behaviour, explicitly set `max_features=1.
`max_features='auto'` has been deprecated in 1.1 and will be removed in 1.3. To keep the past behaviour, explicitly set `max_features=1.
`max_features='auto'` has been deprecated in 1.1 and will be removed in 1.3. To keep the past behaviour, explicitly set `max_features=1.
`max_features='auto'` has been deprecated in 1.1 and will be removed in 1.3. To keep the past behaviour, explicitly set `max_features=1.
`max_features='auto'` has been deprecated in 1.1 and will be removed in 1.3. To keep the past behaviour, explicitly set `max_features=1.
`max_features='auto'` has been deprecated in 1.1 and will be removed in 1.3. To keep the past behaviour, explicitly set `max_features=1.
`max_features='auto'` has been deprecated in 1.1 and will be removed in 1.3. To keep the past behaviour, explicitly set `max_features=1.
`max_features='auto'` has been deprecated in 1.1 and will be removed in 1.3. To keep the past behaviour, explicitly set `max_features=1.
`max_features='auto'` has been deprecated in 1.1 and will be removed in 1.3. To keep the past behaviour, explicitly set `max_features=1.
`max_features='auto'` has been deprecated in 1.1 and will be removed in 1.3. To keep the past behaviour, explicitly set `max_features=1.
`max_features='auto'` has been deprecated in 1.1 and will be removed in 1.3. To keep the past behaviour, explicitly set `max_features=1.
`max_features='auto'` has been deprecated in 1.1 and will be removed in 1.3. To keep the past behaviour, explicitly set `max_features=1.
`max_features='auto'` has been deprecated in 1.1 and will be removed in 1.3. To keep the past behaviour, explicitly set `max_features=1.
`max_features='auto'` has been deprecated in 1.1 and will be removed in 1.3. To keep the past behaviour, explicitly set `max_features=1.
`max_features='auto'` has been deprecated in 1.1 and will be removed in 1.3. To keep the past behaviour, explicitly set `max_features=1.
```

```
`max_features='auto'` has been deprecated in 1.1 and will be removed in 1.3. To keep the past behaviour, explicitly set `max_features=1.
```

```
# Model Evaluation
```

Model Evaluation

```
def evaluate_model(model, X_test, y_test):
    # Make predictions
    y_pred = model.predict(X_test)

    # Calculate evaluation metrics
    mae = mean_absolute_error(y_test, y_pred)
    mape = mean_absolute_percentage_error(y_test, y_pred)
    mse = mean_squared_error(y_test, y_pred)
    r2 = r2_score(y_test, y_pred)
    rmse = math.sqrt(mse)

    # Print the evaluation metrics
    print('MAE is {}'.format(mae))
    print('MAPE is {}'.format(mape))
    print('MSE is {}'.format(mse))
    print('R2 score is {}'.format(r2))
    print('RMSE score is {}'.format(rmse))
```

```
print("Random Forest Model:")
evaluate_model(best_rf_model, X_test, y_test)
```

```
Random Forest Model:
MAE is 11690.229893138958
MAPE is 2.1939785754512978e+19
MSE is 556197583.8459972
R2 score is 0.757933653645164
RMSE score is 23583.841583719928
```

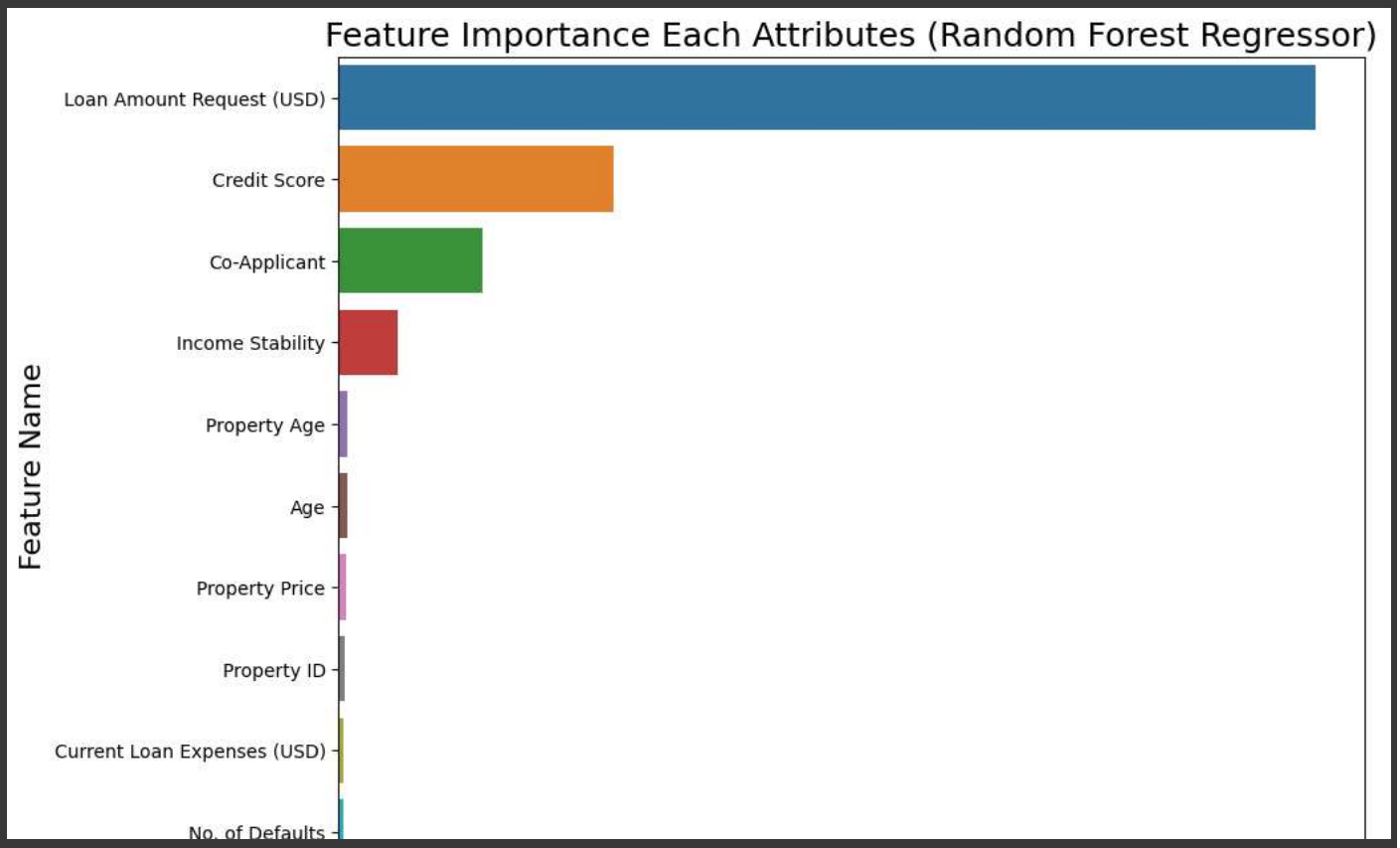
Feature Importance Analysis

```
max_features='auto' has been deprecated in 1.1 and will be removed in 1.3. To keep the past behaviour, explicitly set max_features=1.
```

```
def plot_feature_importance(model, X_train):
    imp_df = pd.DataFrame({
        "Feature Name": X_train.columns,
        "Importance": model.feature_importances_
    })
    fi = imp_df.sort_values(by="Importance", ascending=False)
    fi2 = fi.head(10)

    plt.figure(figsize=(10, 8))
    sns.barplot(data=fi2, x='Importance', y='Feature Name')
    plt.title('Feature Importance Each Attributes (Random Forest Regressor)', fontsize=18)
    plt.xlabel('Importance', fontsize=16)
    plt.ylabel('Feature Name', fontsize=16)
    plt.show()
```

```
max_features='auto' has been deprecated in 1.1 and will be removed in 1.3. To keep the past behaviour, explicitly set max_features=1.
```

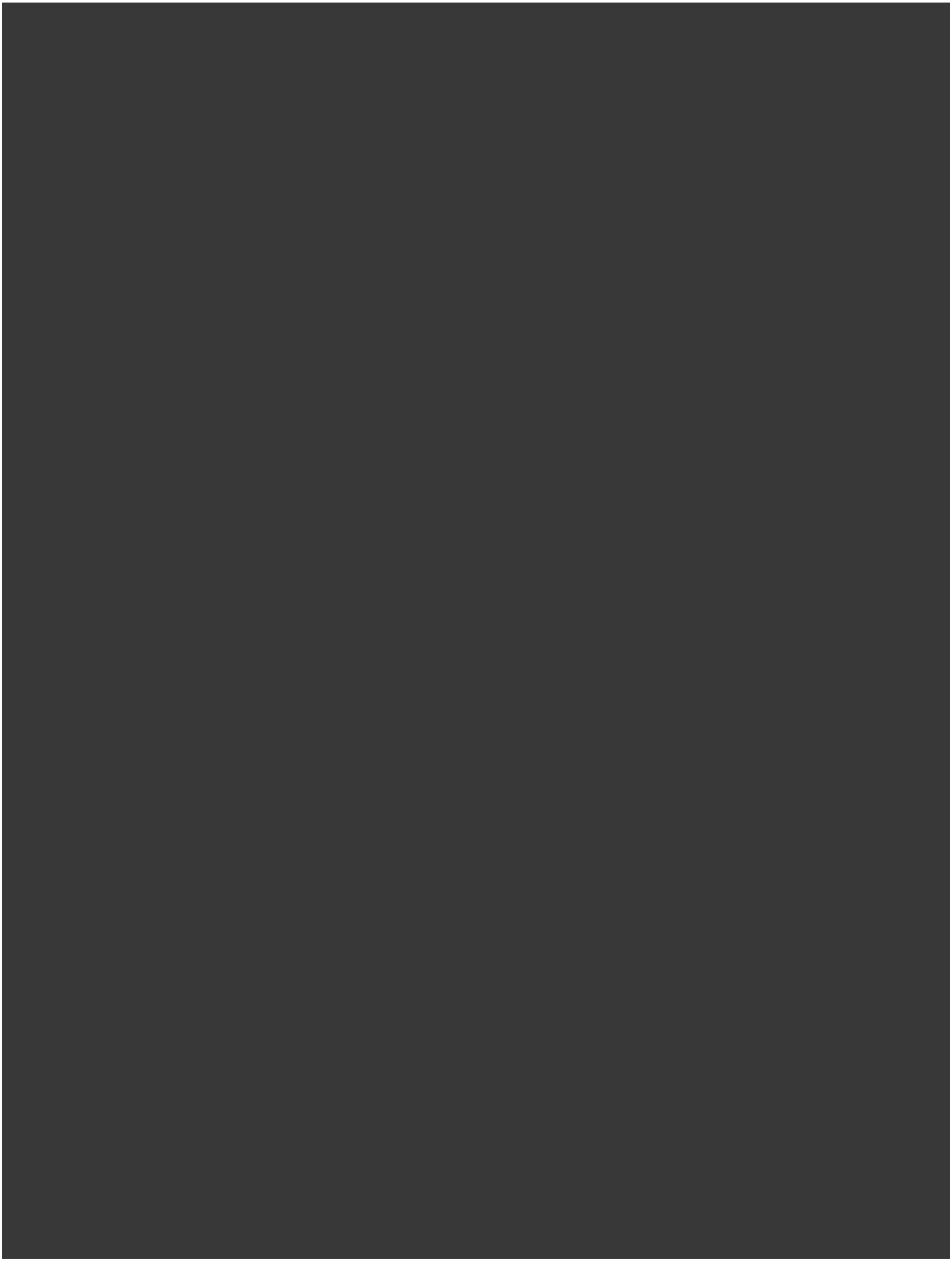
```
plot_feature_importance(best_rf_model, X_train)
```



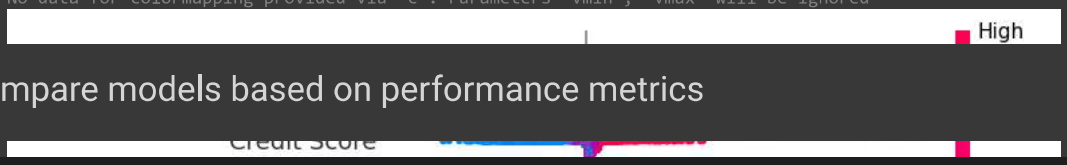
SHAP Analysis

```
def shap_analysis(model, X_test):
    # SHAP Tree Explainer
    explainer = shap.TreeExplainer(model)
    shap_values = explainer.shap_values(X_test)

    # Summary Plot
    print ('Summary Plot')
    shap.summary_plot(shap_values, X_test)

    # Waterfall Plot for a single prediction (you can customize the index)
    print('\n Waterfall Plot for a single prediction ')
    explainer = shap.Explainer(model, X_test, check_additivity=False)
    shap_values = explainer(X_test, check_additivity=False)
    shap.plots.waterfall(shap_values[0])
```

```
shap_analysis(best_rf_model, X_train)
```



```
Summary Plot
No data for colormapping provided via 'c'. Parameters 'vmin', 'vmax' will be ignored
```



Compare models based on performance metrics

```
from sklearn.metrics import mean_absolute_error, mean_squared_error, mean_absolute_percentage_error, r2_score
# Compare models based on performance metrics
def compare_models(model1_metrics, model2_metrics):
    metrics_to_compare = ['MAE', 'MAPE', 'MSE', 'R2', 'RMSE']
    for metric in metrics_to_compare:
        if model1_metrics[metric] < model2_metrics[metric]:
            print(f"{metric}: Decision Tree model performs better")
        elif model1_metrics[metric] > model2_metrics[metric]:
            print(f"{metric}: Random Forest model performs better")
        else:
            print(f"{metric}: Both models perform equally")

# Get performance metrics for Decision Tree model
y_pred_dtree = dtree.predict(X_test)
dtree_metrics = {
    'MAE': mean_absolute_error(y_test, y_pred_dtree),
    'MAPE': mean_absolute_percentage_error(y_test, y_pred_dtree),
    'MSE': mean_squared_error(y_test, y_pred_dtree),
    'R2': r2_score(y_test, y_pred_dtree),
    'RMSE': math.sqrt(mean_squared_error(y_test, y_pred_dtree))
}

# Get performance metrics for Random Forest model
y_pred_rf = best_rf_model.predict(X_test)
rf_metrics = {
    'MAE': mean_absolute_error(y_test, y_pred_rf),
    'MAPE': mean_absolute_percentage_error(y_test, y_pred_rf),
    'MSE': mean_squared_error(y_test, y_pred_rf),
    'R2': r2_score(y_test, y_pred_rf),
    'RMSE': math.sqrt(mean_squared_error(y_test, y_pred_rf))
}

# Compare models
print("\nModel Comparison:")
compare_models(dtree_metrics, rf_metrics)

# Select the best model based on the comparison
if dtree_metrics['MAE'] < rf_metrics['MAE']:
    best_model = dtree
else:
    best_model = best_rf_model

print("\nBest Model Selected:")
print(best_model)
```

```
Model Comparison:
MAE: Random Forest model performs better
MAPE: Random Forest model performs better
MSE: Random Forest model performs better
R2: Decision Tree model performs better
RMSE: Random Forest model performs better

Best Model Selected:
RandomForestRegressor(max_depth=9, max_features='auto', min_samples_leaf=2,
                      min_samples_split=10, random_state=0)
```

Lets See How The Model Predicts The Train Data and by how what percentage

```
# Assuming you have 'new_data' in the same format as your training data
X_new = train.drop('Loan Sanction Amount (USD)', axis=1)
y_new_actual = train['Loan Sanction Amount (USD)']

# Use the model to make predictions on the new data
y_new_pred = best_model.predict(X_new)
```

```
# Evaluate the model on the new data
new_data_metrics = {
    'MAE': mean_absolute_error(y_new_actual, y_new_pred),
    'MAPE': mean_absolute_percentage_error(y_new_actual, y_new_pred),
    'MSE': mean_squared_error(y_new_actual, y_new_pred),
    'R2': r2_score(y_new_actual, y_new_pred),
    'RMSE': math.sqrt(mean_squared_error(y_new_actual, y_new_pred))
}

# Print the evaluation metrics for the new data
print("\nEvaluation Metrics on New Data:")
print(new_data_metrics)
```

```
Evaluation Metrics on New Data:
{'MAE': 10671.90765767297, 'MAPE': 2.047173595112394e+19, 'MSE': 461443783.13640696, 'R2': 0.7929827958328065, 'RMSE': 21481.2425882770
```

MAE (Mean Absolute Error): MAE measures the average absolute difference between the actual values and the predicted values. It gives you an idea of how far off, on average, your predictions are from the actual values. In this case, the MAE is approximately 10,671.91.

MAPE (Mean Absolute Percentage Error): MAPE is a percentage-based error metric that measures the average percentage difference between the actual values and the predicted values. The value you've provided, 2.047173595112394e+19, is extremely high and likely indicates an issue with the model's predictions. Such a high MAPE suggests that the model's predictions are significantly off from the actual values.

MSE (Mean Squared Error): MSE measures the average squared difference between the actual values and the predicted values. It penalizes larger errors more than smaller ones. The MSE in this case is approximately 461,443,783.14.

R2 (R-squared): R-squared is a measure of how well the model explains the variance in the data. It ranges from 0 to 1, where 1 indicates a perfect fit. An R-squared value of 0.7929 suggests that the model explains about 79.29% of the variance in the data, which is relatively good.

RMSE (Root Mean Squared Error): RMSE is the square root of the MSE and is in the same units as the target variable. It provides a measure of the average magnitude of errors. The RMSE in this case is approximately 21,481.24.

In summary, while the R-squared value indicates that the model is explaining a significant portion of the variance in the data, the extremely high MAPE suggests that the model's predictions are not accurate on a percentage basis.

USING The Best Model To Predict.

First we need to prepare the data. in the formart we can use to predict the new data end evaluate the performance

This is evaluation of the best model on new data

in preapraing the data we first transform the data into the form that will be accepted by the model

Steps taken to prepare the testing data.

test.head()

Check the number of unique value from all of the object datatype

test.select_dtypes(include='object').nunique()

Check the amounnt of missing value

check_missing = test.isnull().sum() * 100 / test.shape[0] check_missing[check_missing > 0].sort_values(ascending=False)

check segmet of employment type

test['Type of Employment'].unique()

if not formarted apply this

test['Type of Employment'] = test['Type of Employment'].apply(segment_employment_type)

Processing the new Test Data

preprocess_data(test)

Loop over each column in the DataFrame where dtype is 'object'

for col in test.select_dtypes(include=['object']).columns:

Print the column name and the unique values

____ print(f"{col}: {test[col].unique()}")

exprole the testing data

explore_data(test)

eplore_2(test)

exprole_3(test)

exprole_4(test)

unique encoded values

encoder_object(test)

test.shape

Remove outliers using Z-Score

remove_outliers(test, selected_columns)

```
test.head()
```

	Customer ID	Name	Gender	Age	Income (USD)	Income Stability	Profession	Type of Employment	Location	Loan Amount Request (USD)	...	Dependents	Credit Score	No. of Defaults	H Cr
0	C-26247	Tandra Olszewski	F	47	3472.69	Low	Commercial associate	Managers	Semi-Urban	137088.98	...	2.0	799.14	0	Ur
1	C-35067	Jeannette Cha	F	57	1184.84	Low	Working	Sales staff	Rural	104771.59	...	2.0	833.31	0	Ur
2	C-34590	Keva Godfrey	F	52	1266.27	Low	Working	NaN	Semi-Urban	176684.91	...	3.0	627.44	0	Ur
3	C-16668	Elva Sackett	M	65	1369.72	High	Pensioner	NaN	Rural	97009.18	...	2.0	833.20	0	
4	C-12196	Sade Constable	F	60	1939.23	High	Pensioner	NaN	Urban	109980.00	...	NaN	NaN	0	

5 rows × 23 columns

```
#### Check the number of unique value from all of the object datatype
test.select_dtypes(include='object').nunique()
```

```
Customer ID               20000
Name                      20000
Gender                        2
Income Stability              2
Profession                    7
Type of Employment           18
Location                      3
Expense Type 1                2
Expense Type 2                2
Has Active Credit Card        3
Property Location             3
Co-Applicant                  3
Property Price            19824
dtype: int64
```

```
#### Check the amounnt of missing value
check_missing = test.isnull().sum() * 100 / test.shape[0]
check_missing[check_missing > 0].sort_values(ascending=False)
```

```
Type of Employment            23.445
Dependents                     5.710
Has Active Credit Card         5.380
Property Age                   4.460
Income Stability               4.065
Income (USD)                   3.750
```

```
Credit Score                    3.715
Property Location               0.800
Current Loan Expenses (USD)     0.415
Gender                          0.155
dtype: float64
```

```
#### check segmet of employment type
test['Type of Employment'].unique()
```

```
array(['Managers', 'Sales staff', nan, 'Laborers', 'Core staff',
       'Medicine staff', 'Accountants', 'High skill tech staff',
       'Secretaries', 'Drivers', 'Cooking staff', 'Security staff',
       'Cleaning staff', 'Low-skill Laborers', 'Waiters/barmen staff',
       'Realty agents', 'Private service staff', 'IT staff', 'HR staff'],
      dtype=object)
```

```
#### if segment not formarted apply this

test['Type of Employment'] = test['Type of Employment'].apply(segment_employment_type)

test['Type of Employment'].unique()
```

```
array(['Managers', 'Sales/Realty', 'Unknown', 'Laborers', 'Other',
       'Secretaries/HR', 'Hospitality', 'Tech/IT'], dtype=object)
```

```
def preprocess_data(df):
    # Remove identifier columns
    df.drop(columns=['Customer ID', 'Name'], inplace=True)

    # Segment Type of Employment into smaller unique values
    df['Type of Employment'] = df['Type of Employment'].apply(segment_employment_type)



    # Handle missing values
    df.fillna({
        'Property Age': df['Property Age'].median(),
        'Income (USD)': df['Income (USD)'].median(),
        'Dependents': df['Dependents'].median(),
        'Credit Score': df['Credit Score'].median(),
        'Current Loan Expenses (USD)': df['Current Loan Expenses (USD)'].median()
    }, inplace=True)

    df.dropna(subset=['Income Stability', 'Has Active Credit Card', 'Property Location', 'Gender'], inplace=True)

    return df.head(10)
```

```
#### Processing the new Test Data
preprocess_data(test)
```

	Gender	Age	Income (USD)	Income Stability	Profession	Type of Employment	Location	Loan Amount Request (USD)	Current Loan Expenses (USD)	Expense Type 1	...	Dependents	Credit Score	No. of Defaults	H Cr
0	F	47	3472.69	Low	Commercial associate	Managers	Semi-Urban	137088.98	396.72	N	...	2.0	799.14	0	Ur

```
#### Loop over each column in the DataFrame where dtype is 'object'
for col in test.select_dtypes(include=['object']).columns:

    # Print the column name and the unique values
    print(f"{col}: {test[col].unique()}")
```

```
Gender: ['F' 'M']
Income Stability: ['Low' 'High']
Profession: ['Commercial associate' 'Working' 'Pensioner' 'State servant' 'Unemployed'
 'Maternity leave' 'Student']
Type of Employment: ['Managers' 'Sales/Realty' 'Other' 'Laborers' 'Secretaries/HR' 'Tech/IT']
Location: ['Semi-Urban' 'Rural' 'Urban']
Expense Type 1: ['N' 'Y']
Expense Type 2: ['N' 'Y']
Has Active Credit Card: ['Unpossessed' 'Inactive' 'Active']
Property Location: ['Urban' 'Rural' 'Semi-Urban']
Co-Applicant: ['1' '0' '?']
Property Price: ['236644.5' '142357.3' '300991.24' ... '21566.27' '120281.17' '133425.43']
```

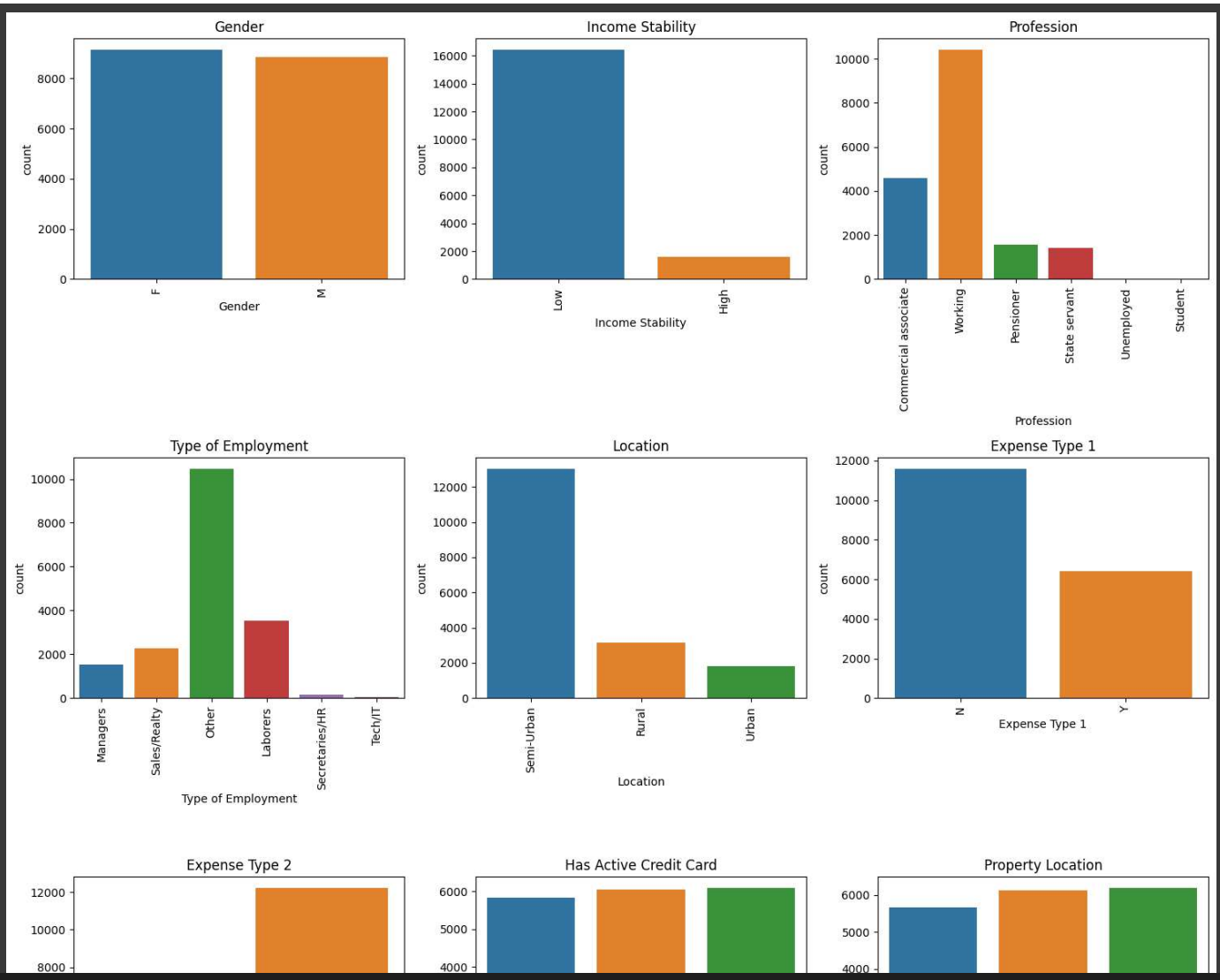
associate

exprole the testing data

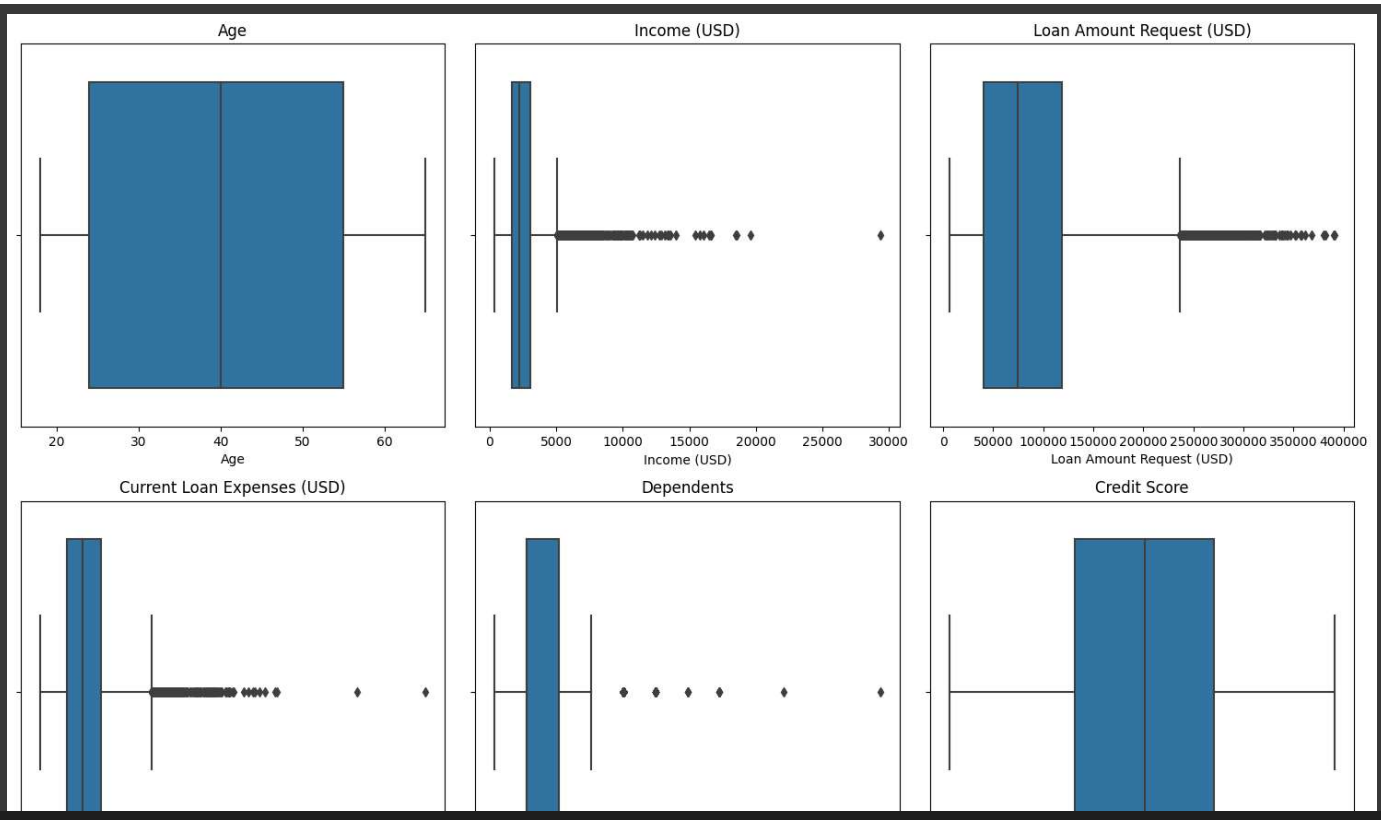
```
test.dtypes
```

```
Gender                         object
Age                             int64
Income (USD)                  float64
Income Stability               object
Profession                     object
Type of Employment             object
Location                       object
Loan Amount Request (USD)     float64
Current Loan Expenses (USD)   float64
Expense Type 1                 object
Expense Type 2                 object
Dependents                    float64
Credit Score                  float64
No. of Defaults                 int64
Has Active Credit Card         object
Property ID                     int64
Property Age                  float64
Property Type                   int64
Property Location              object
Co-Applicant                   object
Property Price                 object
dtype: object
```

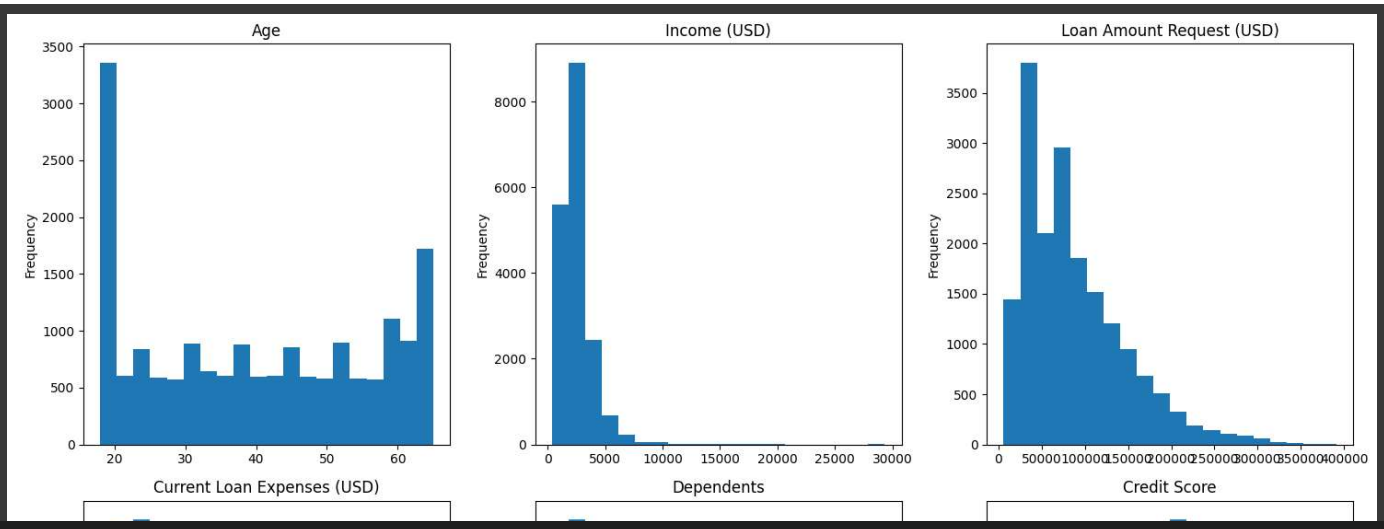
```
explore_data(test)
```




```
eplore_2(test)
```

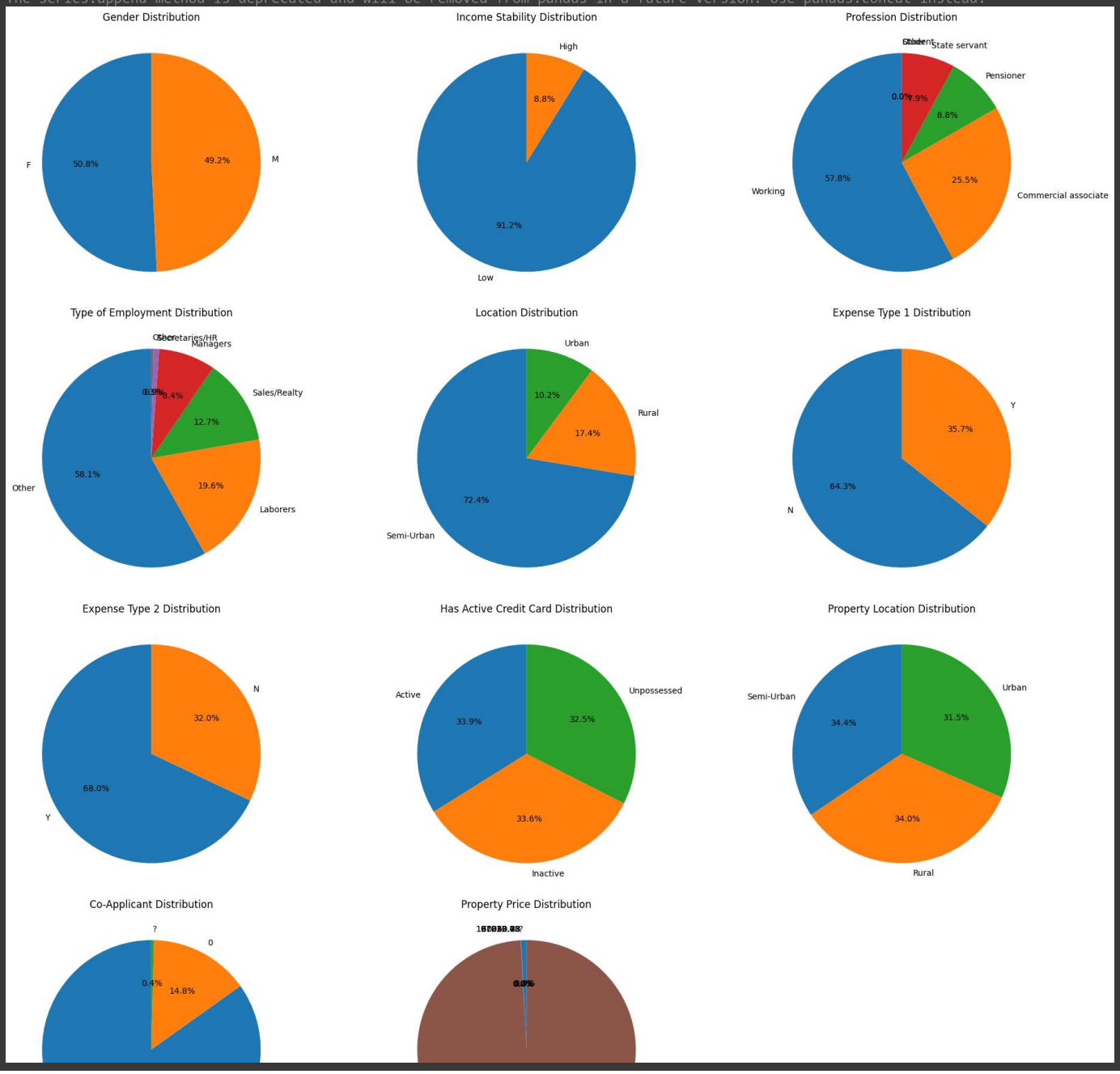



```
exprole_3(test)
```




```
exprole_4(test)
```

```
The series.append method is deprecated and will be removed from pandas in a future version. Use pandas.concat instead.
The series.append method is deprecated and will be removed from pandas in a future version. Use pandas.concat instead.
The series.append method is deprecated and will be removed from pandas in a future version. Use pandas.concat instead.
```



```
#### unique encoded values

encoder_object(test)
```

```
Gender: [0 1]
Income Stability: [1 0]
Profession: [0 6 2 3 5 1 4]
Type of Employment: [1 3 2 0 4 5]
Location: [1 0 2]
Expense Type 1: [0 1]
Expense Type 2: [0 1]
Has Active Credit Card: [2 1 0]
Property Location: [2 0 1]
Co-Applicant: [1 0 2]
Property Price: [7955 3439 9626 ... 7179 1781 2805]
```

```
test.shape
```

```
(17981, 21)
```

```
#### Remove outliers using Z-Score
remove_outliers(test, selected_columns)
```

	Gender	Age	Income (USD)	Income Stability	Profession	Type of Employment	Location	Loan Amount Request (USD)	Current Loan Expenses (USD)	Expense Type 1	...	Dependents	Credit Score	No. of Defaults	Ac Cr
0	0	47	3472.69	1	0	1	1	137088.98	396.72	0	...	2.0	799.14	0	
1	0	57	1184.84	1	6	3	0	104771.59	463.76	1	...	2.0	833.31	0	
2	0	52	1266.27	1	6	2	1	176684.91	493.15	0	...	3.0	627.44	0	
3	1	65	1369.72	0	2	2	0	97009.18	446.15	0	...	2.0	833.20	0	
5	0	59	2944.81	1	6	3	1	31465.78	153.10	1	...	2.0	620.58	0	
6	1	43	1957.31	1	6	3	0	150334.11	433.82	0	...	2.0	731.37	0	
7	1	65	1403.63	0	2	2	1	121029.27	348.18	0	...	1.0	838.66	1	
8	1	64	1604.65	1	6	2	1	39475.31	200.41	0	...	1.0	589.97	0	
10	0	43	2037.14	1	0	2	0	123472.08	350.99	0	...	2.0	739.30	1	
11	1	29	2183.59	1	0	0	1	53651.03	216.60	0	...	3.0	653.00	0	

10 rows × 21 columns

Use the model to make predictions on the new data

Lets first copy the test data to new data so to avoid termparing with the data

```
new_data = test.copy()
```

```
# Assuming 'new_data' doesn't contain the 'Loan Sanction Amount (USD)' column
X_new = new_data  # Replace 'new_data' with your actual new data

# Use the model to make predictions on the new data
y_new_pred = best_model.predict(X_new)

# Create a new column in 'new_data' and populate it with the predicted values
new_data['Predicted Loan Sanction Amount (USD)'] = y_new_pred

# Save the data with the predicted column to a CSV file
test_data_predicted = new_data.to_csv('test_data_with_predictions.csv', index=False)  # Change the filename as needed
```

Lets Predict Loan Sanction For an Individual

```
def predict_loan_sanction_amount():
    # Prompt the user to input values for each feature
    print("Please enter the following information:")
    print('Example First Name')
    client_name = input('Type Your Name: ')
    gender = input("Gender (1-Male/0-Female): ")
    age = int(input("Age: "))
    income_usd = float(input("Income (USD): "))
    income_stability = input("Income Stability (0 - Stable/1- Unstable): ")

    print("4 - 'Working', 1- 'Pensioner', 2-'State servant', 0 - 'Commercial associate', 3- 'Unemployed'")
    profession = input("Profession: ")
    print ("Employment : 7 -'Unknown', 3- 'Other', 5- 'Secretaries/HR', 1-  'Laborers',2- 'Managers', 0- 'Hospitality',4-  'Sales/Realty' ,6-
    type_of_employment = input("Type of Employment: ")

    print("1 - 'Semi-Urban', 0-'Rural', 2- 'Urban'")
    location = input("Location: ")

    loan_amount_requested = float(input("Loan Amount Requested (USD): "))
    current_loan_expenses_usd = float(input("Current Loan Expenses (USD): "))

    expense_type_1 = input("Expense Type 1 (1- Yes, 0 - No): ")

    expense_type_2 = input("Expense Type 2 (1- Yes, 0 - No): ")

    dependents = (input("Dependents(example: 8.9, 2.0..): "))
    credit_score = float(input("Credit Score: "))
    num_of_defaults = int(input("No. of Defaults: "))

    has_active_credit_card = input("Has Active Credit Card (2 - 'Unpossessed', 0- 'Active', 1-  'Inactive'): ")

    property_id = input("Property ID: ")
    property_age = float(input("Property Age: "))

    property_type = input("Property Type (1 - 4): ")

    property_location = input("Property Location (0- Rural, 1- Semiurban, 2- Urban): ")

    co_applicant = input("Co-Applicant (1 - Yes/0 - No): ")

    property_price = float(input("Property Price: "))

    # Create a dictionary with input values
    input_data = {
        'Gender': [gender],
        'Age': [age],
        'Income (USD)': [income_usd],
        'Income Stability': [income_stability],
        'Profession': [profession],
        'Type of Employment': [type_of_employment],
        'Location': [location],
        'Loan Amount Request (USD)': [loan_amount_requested],
        'Current Loan Expenses (USD)': [current_loan_expenses_usd],
        'Expense Type 1': [expense_type_1],
        'Expense Type 2': [expense_type_2],
        'Dependents': [dependents],
        'Credit Score': [credit_score],
        'No. of Defaults': [num_of_defaults],
        'Has Active Credit Card': [has_active_credit_card],
        'Property ID': [property_id],
        'Property Age': [property_age],
        'Property Type': [property_type],
        'Property Location': [property_location],
        'Co-Applicant': [co_applicant],
        'Property Price': [property_price]
    }

    # Create a DataFrame from the input data
    input_df = pd.DataFrame(input_data)

    # Preprocess the input data (similar to training data preprocessing)
    # ... (apply the same preprocessing steps as in your original code)

    # Use the model to make predictions on the preprocessed input data
    predicted_loan_sanction_amount = best_model.predict(input_df)

    # Display the predicted 'Loan Sanction Amount (USD)'
```

```
    print(f"\n Predicted Loan Sanction Amount (USD) for {client_name} : {predicted_loan_sanction_amount[0]:.2f}")
```

```
# Call the function to predict 'Loan Sanction Amount (USD)' for an individual
predict_loan_sanction_amount()
```

```
Please enter the following information:
Example First Name
Type Your Name: Joseph
Gender (1-Male/0-Female): 1
Age: 42
Income (USD): 5000.05
Income Stability (0 - Stable/1- Unstable): 0
4 - 'Working', 1- 'Pensioner', 2-'State servant', 0 - 'Commercial associate', 3- 'Unemployed'
Profession: 4
Employment : 7 -'Unknown', 3- 'Other', 5- 'Secretaries/HR', 1-  'Laborers',2- 'Managers', 0- 'Hospitality',4-  'Sales/Realty' ,6- 'Tech/
Type of Employment: 6
1 - 'Semi-Urban', 0-'Rural', 2- 'Urban'
Location: 1
Loan Amount Requested (USD): 1000000.35
Current Loan Expenses (USD): 200.32
Expense Type 1 (1- Yes, 0 - No): 1
Expense Type 2 (1- Yes, 0 - No): 1
Dependents(example: 8.9, 2.0..): 2.3
Credit Score: 568354.36
No. of Defaults: 0
Has Active Credit Card (2 - 'Unpossessed', 0- 'Active', 1-  'Inactive'): 0
Property ID: 564
Property Age: 564
Property Type (1 - 4): 2
Property Location (0- Rural, 1- Semiurban, 2- Urban): 2
Co-Applicant (1 - Yes/0 - No): 1
Property Price: 25689325.25

 Predicted Loan Sanction Amount (USD) for Joseph : 302677.12
```