Individual Assignment (Programming for Data Analytics)

Submitted By Yaswanthkumar Thirunavukkarasu - 809732

Q1) Descriptive Statistics and Key Trends in Loan Applications Dataset

In [501…

```
import pandas as pd

# Load dataset into a DataFrame
df = pd.read_csv("loanapp.csv")

# Display the first 5 rows
df.head()
```

Out[501…

	married	race	loan_decision	occupancy	loan_amount	applicant_income	num_units	num_dependants	self_employed	monthly_income	purchase_price	liquid_assets	mortage_payment_history	consumer_credit_history	filed_bankruptcy	property_type	gender
0	True	white	reject	1	128	74	1.0	1.0	False	4583	160.0	52.0	2	2	False	2	male
1	False	white	approve	1	128	84	1.0	0.0	False	2666	143.0	37.0	2	2	False	2	male
2	True	white	approve	1	66	36	1.0	0.0	True	3000	110.0	19.0	2	6	True	2	male
3	True	white	approve	1	120	59	1.0	0.0	False	2583	134.0	31.0	2	1	False	1	male
4	False	white	approve	1	111	63	1.0	0.0	False	2208	138.0	169.0	2	6	False	2	male

In [502…

```
df.info()  # Provides information about the dataset, including data types and null values
df.shape  # Displays the number of rows and columns (rows, columns)
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1988 entries, 0 to 1987
Data columns (total 17 columns):
 #   Column                   Non-Null Count  Dtype
---  ------                   --------------  -----
 0   married                  1985 non-null   object
 1   race                     1988 non-null   object
 2   loan_decision            1988 non-null   object
 3   occupancy                1988 non-null   int64
 4   loan_amount              1988 non-null   int64
 5   applicant_income         1988 non-null   int64
 6   num_units                1984 non-null   float64
 7   num_dependants           1985 non-null   float64
 8   self_employed            1988 non-null   bool
 9   monthly_income           1988 non-null   int64
 10  purchase_price           1988 non-null   float64
 11  liquid_assets            1988 non-null   float64
 12  mortage_payment_history  1988 non-null   int64
 13  consumer_credit_history  1988 non-null   int64
 14  filed_bankruptcy         1988 non-null   bool
 15  property_type            1988 non-null   int64
 16  gender                   1974 non-null   object
dtypes: bool(2), float64(4), int64(7), object(4)
memory usage: 237.0+ KB
```

Out[502…

```
(1988, 17)
```

In [503…

```
df.describe()  # Summary statistics for numerical columns
```

Out[503…

	occupancy	loan_amount	applicant_income	num_units	num_dependants	monthly_income	purchase_price	liquid_assets	mortage_payment_history	consumer_credit_history	property_type
count	1988.000000	1988.000000	1988.000000	1984.000000	1985.000000	1988.000000	1988.000000	1988.000000	1988.000000	1988.000000	1988.000000
mean	1.031690	143.272636	84.684105	1.122480	0.771285	5195.220825	196.304088	4620.333873	1.708249	2.110161	1.861167
std	0.191678	80.531470	87.079777	0.437315	1.104464	5270.360946	128.136030	67142.936043	0.555335	1.663256	0.535448
min	1.000000	2.000000	0.000000	1.000000	0.000000	0.000000	25.000000	0.000000	1.000000	1.000000	1.000000
25%	1.000000	100.000000	48.000000	1.000000	0.000000	2875.750000	129.000000	20.000000	1.000000	1.000000	2.000000
50%	1.000000	126.000000	64.000000	1.000000	0.000000	3812.500000	163.000000	38.000000	2.000000	1.000000	2.000000
75%	1.000000	165.000000	88.000000	1.000000	1.000000	5594.500000	225.000000	83.000000	2.000000	2.000000	2.000000
max	3.000000	980.000000	972.000000	4.000000	8.000000	81000.000000	1535.000000	1000000.000000	4.000000	6.000000	3.000000

In [504…

```
df.describe(include=['object'])  # For non-numeric data
```

Out[504…

	married	race	loan_decision	gender
count	1985	1988	1988	1974
unique	2	3	2	2
top	True	white	approve	male
freq	1308	1680	1744	1605

In [505…

```
for col in df.select_dtypes(include=['object']).columns:
    print(df[col].value_counts().to_string())
    print()
```

```
married
True     1308
False     677

race
white     1680
black      197
hispan     111

loan_decision
approve    1744
reject      244

gender
male      1605
female     369
```

In [506…

```
df["self_employed"].value_counts()
```

Out[506…

```
self_employed
False    1731
True      257
Name: count, dtype: int64
```

In [507…

```
df["property_type"].value_counts()
```

Out[507…

```
property_type
2    1380
1     442
3     166
Name: count, dtype: int64
```

General Statistics:

• The dataset contains 1,988 loan applications with various information, including applicants' income, loan amounts, credit history, and other key factors.

• The average loan amount requested is £143,273, with values ranging from £2,000 to £980,000.

• The average applicant income is £84,684, with a standard deviation of £87,080, showing significant income diversity.

• Monthly income averages £5,195, but it varies widely, with some applicants reporting no income (minimum: £0) and others reporting up to £81,000.

• Applicants generally own 1.12 units on average, with some owning up to 4 units.

Loan Approval Trends:

• Of the 1,988 applications, 1,744 were approved (87.7%), while 244 were rejected (12.3%).

• This shows a high approval rate, with the bank approving most of the applications.

Marital Status and Gender Distribution:

• 1,308 applicants (65.8%) are married, and 677 applicants (34.1%) are unmarried.

• 1,605 applicants (80.7%) are male, while 369 applicants (18.6%) are female.

• There is a higher proportion of married and male applicants in the dataset.

Racial Distribution of Applicants:

• 1,680 applicants (84.5%) are White, followed by 197 applicants (9.9%) who are Black, and 111 applicants (5.6%) who are Hispanic.

• The dataset shows that White applicants dominate, with relatively fewer Black and Hispanic applicants.

Self-Employment and Loan Approval:

• 257 applicants (12.9%) are self-employed, while 1,731 applicants (87.1%) are salaried employees.

• The low proportion of self-employed applicants may reflect more stringent loan approval criteria for self-employed individuals.

Property Type Distribution:

• 1,380 applicants (69.4%) have property type "2", 442 applicants (22.2%) have property type "1", and 166 applicants (8.4%) have property type "3".

• Property type "2" is the most common among applicants, with property type "1" being the second most frequent.

Q2. Handling Missing Values:

In [510…

```
# Check for missing values in the dataset
missing_values = df.isnull().sum()

# Display only the columns with missing values (exclude the dtype line)
missing_values[missing_values > 0].to_frame(name='Missing Values')
```

Out[510…

	Missing Values
married	3
num_units	4
num_dependants	3
gender	14

In [511…

```
# Impute missing categorical values with the mode without using inplace=True
df['married'] = df['married'].fillna(df['married'].mode()[0])
df['gender'] = df['gender'].fillna(df['gender'].mode()[0])
```

In [512…

```
# Impute missing numerical values with the median
df['num_units'] = df['num_units'].fillna(df['num_units'].median())
df['num_dependants'] = df['num_dependants'].fillna(df['num_dependants'].median())
```

In [513…

```
# Verify there are no missing values left
missing_values_after_imputation = df.isnull().sum()

# Display columns with missing values (if any)
missing_values_after_imputation[missing_values_after_imputation > 0]
```

Out[513…

```
Series([], dtype: int64)
```

• Missing values were detected in several columns, including 'married' (3 missing), 'num_units' (4 missing), 'num_dependants' (3 missing), and 'gender' (14 missing).

• To handle these missing values, imputation was performed. Given the relatively small number of missing values and the potential impact of removing rows on the analysis, imputation was chosen to preserve data integrity. It is important to note that imputing missing values can introduce bias into the dataset. However, given the small number of missing values, and the need to retain as much data as possible, imputation was deemed the most suitable approach.

• The missing values in the 'married' and 'gender' columns were replaced with the mode (most frequent value) of each respective column. The 'gender' column had the most missing values, therefore, the imputation of this column could have a larger impact on the data.

• The missing values in the 'num_units' and 'num_dependants' columns were replaced with the median of each respective column.

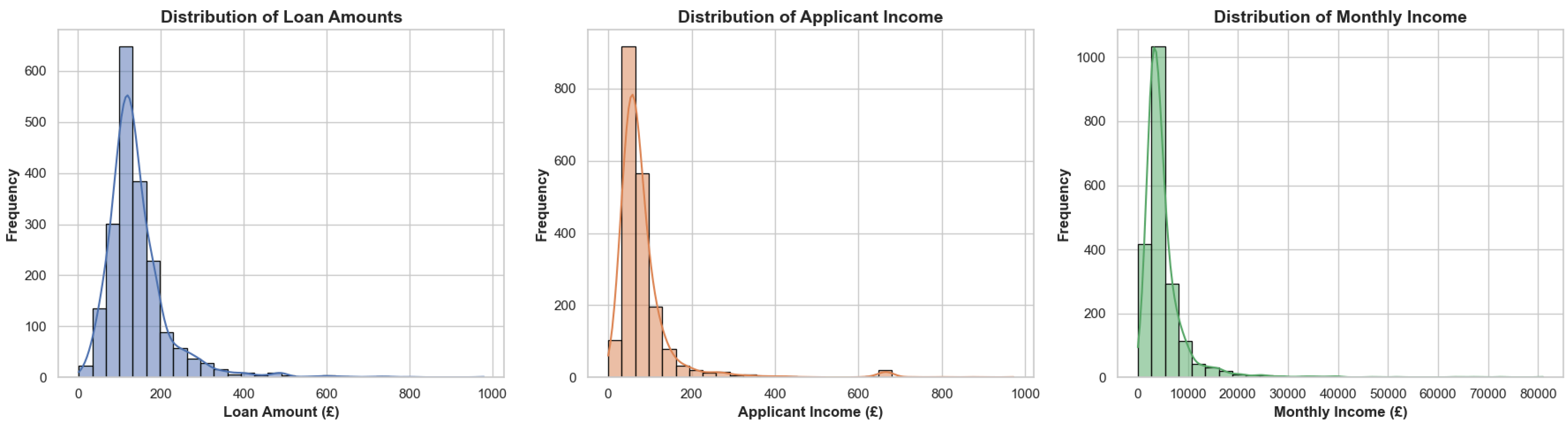
• After imputation, there were no further missing values in the dataset, as verified by rechecking the missing data. This ensures that the dataset is now complete and suitable for analysis.

Q3. Graph Visualization

A. Distribution of One or More Individual Continuous Variables

In [517…

```
import matplotlib.pyplot as plt
import seaborn as sns

# Set modern visualization style
sns.set_theme(style="whitegrid")

# Define color palette
colors = ["#4C72B0", "#DD8452", "#55A868"]

# Create subplots
fig, axes = plt.subplots(1, 3, figsize=(18, 5))

# Loan Amount Distribution
sns.histplot(df["loan_amount"], bins=30, kde=True, color=colors[0], ax=axes[0], edgecolor="black")
axes[0].set_title("Distribution of Loan Amounts", fontsize=14, fontweight='bold')
axes[0].set_xlabel("Loan Amount (£)", fontsize=12, fontweight='bold')
axes[0].set_ylabel("Frequency", fontsize=12, fontweight='bold')

# Applicant Income Distribution
sns.histplot(df["applicant_income"], bins=30, kde=True, color=colors[1], ax=axes[1], edgecolor="black")
axes[1].set_title("Distribution of Applicant Income", fontsize=14, fontweight='bold')
axes[1].set_xlabel("Applicant Income (£)", fontsize=12, fontweight='bold')
axes[1].set_ylabel("Frequency", fontsize=12, fontweight='bold')

# Monthly Income Distribution
sns.histplot(df["monthly_income"], bins=30, kde=True, color=colors[2], ax=axes[2], edgecolor="black")
axes[2].set_title("Distribution of Monthly Income", fontsize=14, fontweight='bold')
axes[2].set_xlabel("Monthly Income (£)", fontsize=12, fontweight='bold')
axes[2].set_ylabel("Frequency", fontsize=12, fontweight='bold')

# Adjust layout
plt.tight_layout()
plt.show()
```



• Loan Amount Distribution: The distribution of loan amounts is right-skewed, indicating that most loans are relatively small, with a few larger loans.

• Applicant Income Distribution: The applicant income also shows a right-skewed distribution, with a concentration of applicants in the lower income range and some high-income outliers.

• Monthly Income Distribution: The monthly income distribution is heavily right-skewed, with the majority of applicants having lower monthly incomes and a few with significantly higher incomes.

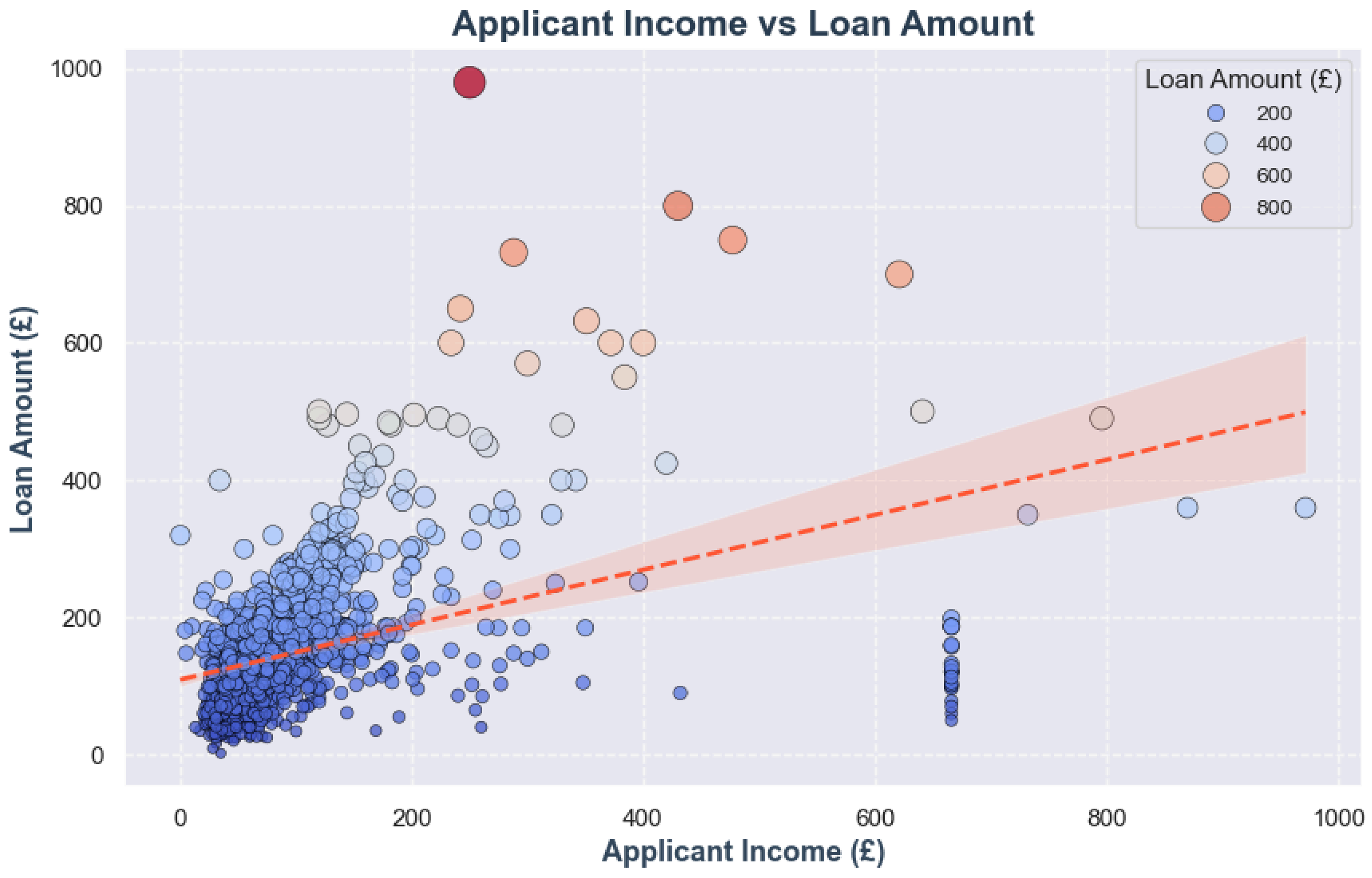
• Outliers: The presence of outliers in the applicant income and monthly income distributions could potentially impact the analysis.

• Potential Transformations: Due to the right-skewness of the data, log transformations could be considered for future analysis.

B. Relationship Between a Pair of Continuous Variables

In [520…

```
# Set modern theme
sns.set_theme(style="darkgrid", palette="muted")

# Create figure
plt.figure(figsize=(10, 6))

# Scatter plot with improved styling
scatter = sns.scatterplot(
    x=df["applicant_income"],
    y=df["loan_amount"],
    hue=df["loan_amount"],  # Gradient color based on loan amount
    palette="coolwarm",  # Modern color scheme
    size=df["loan_amount"],  # Size variation
    sizes=(20, 200),  # Define min/max size
    edgecolor="black",  # Add a black edge for better definition
    alpha=0.75  # Transparency for better clarity
)

# Add regression line
sns.regplot(
    x=df["applicant_income"],
    y=df["loan_amount"],
    scatter=False,
    color="#FF5733",  # Stylish orange line
    line_kws={"linewidth": 2, "linestyle": "--"}
)

# Graph customization
plt.title("Applicant Income vs Loan Amount", fontsize=16, fontweight='bold', color="#2C3E50")
plt.xlabel("Applicant Income (£)", fontsize=13, fontweight='bold', color="#34495E")
plt.ylabel("Loan Amount (£)", fontsize=13, fontweight='bold', color="#34495E")
plt.xticks(fontsize=11)
plt.yticks(fontsize=11)
plt.legend(title="Loan Amount (£)", fontsize=10)
plt.grid(True, linestyle="--", alpha=0.6)

# Display plot
plt.show()
```



• The scatter plot reveals a positive correlation between Applicant Income and Loan Amount, with higher income generally corresponding to larger loan amounts.

• Color Gradient: Darker colors represent larger loan amounts, while lighter colors correspond to smaller loans.

• Size Variation: Data point sizes increase with the loan amount, highlighting larger loans.

• Regression Line: The dashed orange line confirms the upward trend, indicating that income and loan amounts are positively related.

• Overall, the plot shows a general trend of higher income leading to higher loan amounts, with some variation due to other factors. Also, while a positive correlation is visible, the spread of the data indicates that the correlation is not very strong.

• Outliers: There are a few outliers visible on the top left of the graph, that are far from the regression line.

C. Association Between a Categorical Variable and a Continuous One

In [523…

```
# Set the theme
sns.set_theme(style="whitegrid", palette="muted")

# Create figure
plt.figure(figsize=(10, 6))

# Boxplot for Loan Amount vs Property Type, associating palette with the 'property_type'
sns.boxplot(x="property_type", y="loan_amount", data=df, hue="property_type", palette="Set2")

# Graph customization
plt.title("Loan Amount by Property Type", fontsize=16, fontweight='bold', color="#2C3E50")
plt.xlabel("Property Type", fontsize=13, fontweight='bold', color="#34495E")
plt.ylabel("Loan Amount (£)", fontsize=13, fontweight='bold', color="#34495E")
plt.xticks(fontsize=11)
plt.yticks(fontsize=11)

# Display plot
plt.show()
```

Loan Amount Distribution by Property Type:

• Property type 2 has the highest median loan amount and the widest spread of loan amounts, indicating more variability in loan amounts for this property type.

• Property type 1 has the lowest median loan amount and a smaller spread, suggesting that loans for this type are generally lower and less varied.

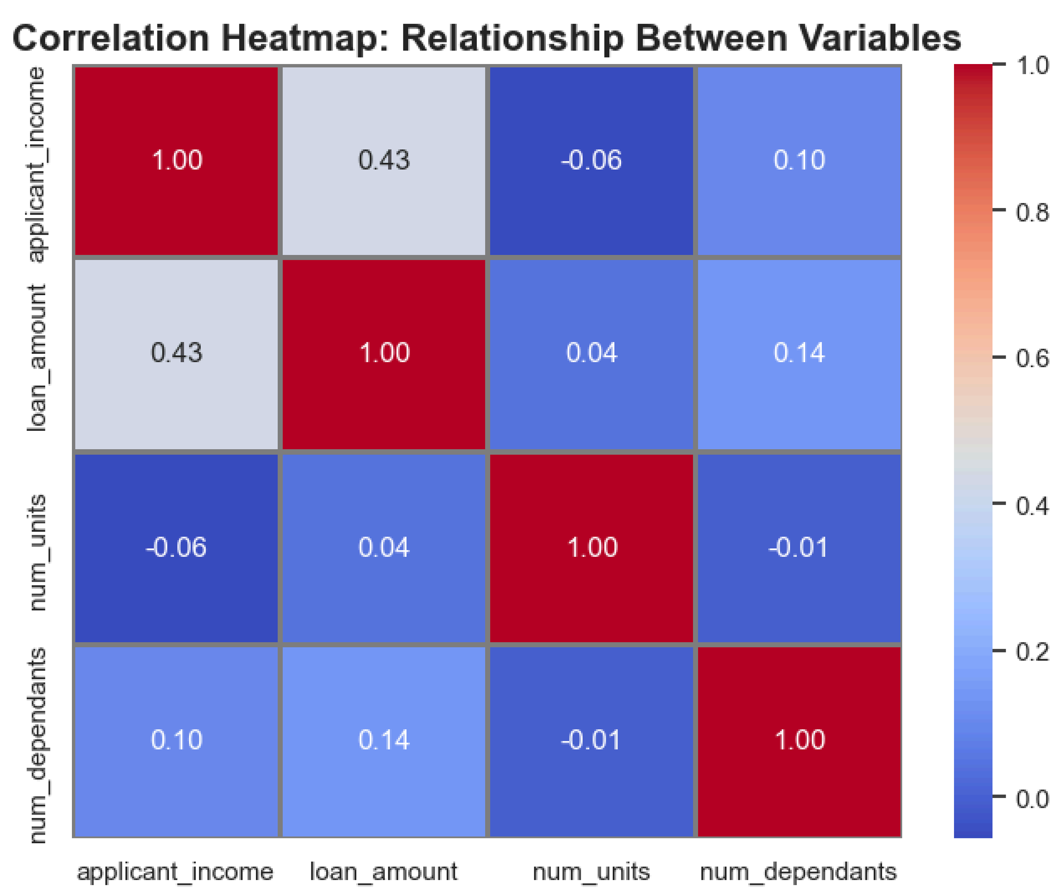
• Property type 3 shows a median loan amount between types 1 and 2, with a moderate spread.

• There are a large number of outliers in property type 2.

D. The Relationship Between More Than Two Variables

In [526…

```
# Compute the correlation matrix
corr_matrix = df[['applicant_income', 'loan_amount', 'num_units', 'num_dependants']].corr()

# Set modern Seaborn theme
sns.set_theme(style="whitegrid", palette="muted")

# Create the heatmap for correlation matrix
plt.figure(figsize=(8, 6))
sns.heatmap(corr_matrix, annot=True, cmap='coolwarm', fmt='.2f', linewidths=1, linecolor='gray', cbar=True)

# Add title and labels
plt.title('Correlation Heatmap: Relationship Between Variables', fontsize=16, fontweight='bold')
plt.show()
```



Correlation Analysis:

• There is a moderate positive correlation (0.43) between 'applicant_income' and 'loan_amount'. This suggests that higher applicant incomes tend to be associated with larger loan amounts.

• The correlation between 'num_units' and the other variables is very weak, close to zero. This indicates that the number of units in the property has little linear relationship with applicant income, loan amount, or the number of dependents.

• The correlation between 'num_dependants' and the other variables are also weak.

• The strongest correlation is between a variable and itself, which is always 1.00.

Q4. Display unique values of a categorical variable and their frequencies

In [529…

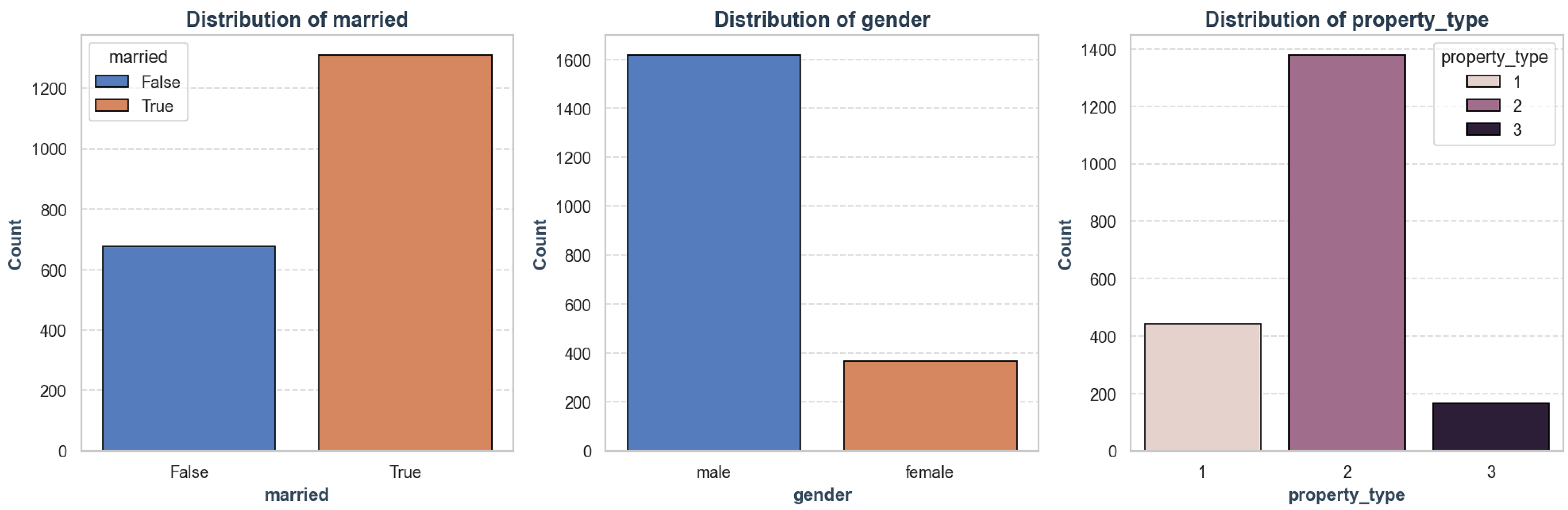
```
# Function to display the unique values and their frequencies in a clean format
def display_value_counts(df, column):
    # Get the value counts of the categorical column
    value_counts = df[column].value_counts()

    # Print a clean and well-formatted output
```

```
    print(f"\nUnique values and their frequencies for '{column}' column:")
    print(f"{'-'*50}")
    print(value_counts.to_string(header=False))
    print(f"{'-'*50}")

# Apply the function to the 'married' column and any other categorical columns
display_value_counts(df, 'married')
display_value_counts(df, 'gender')
display_value_counts(df, 'property_type')
```

```
Unique values and their frequencies for 'married' column:
--------------------------------------------------
True     1311
False     677
--------------------------------------------------

Unique values and their frequencies for 'gender' column:
--------------------------------------------------
male      1619
female     369
--------------------------------------------------

Unique values and their frequencies for 'property_type' column:
--------------------------------------------------
2    1380
1     442
3     166
--------------------------------------------------
```

In [530…

```
# Select categorical columns
categorical_columns = ['married', 'gender', 'property_type']

# Set up figure with three subplots in one row
fig, axes = plt.subplots(1, 3, figsize=(15, 5), dpi=120)

# Loop through categorical columns and create bar plots
for i, column in enumerate(categorical_columns):
    value_counts = df[column].value_counts()

    sns.barplot(
        x=value_counts.index,
        y=value_counts.values,
        hue=value_counts.index,  # Set the hue to x to fix the FutureWarning
        edgecolor="black",
        ax=axes[i]
    )

    # Customize each subplot
    axes[i].set_title(f"Distribution of {column}", fontsize=14, fontweight='bold', color="#2C3E50")
    axes[i].set_xlabel(column, fontsize=12, fontweight='bold', color="#34495E")
    axes[i].set_ylabel("Count", fontsize=12, fontweight='bold', color="#34495E")
    axes[i].tick_params(axis='both', labelsize=11)
    axes[i].grid(axis="y", linestyle="--", alpha=0.6)

# Adjust layout for better spacing
plt.tight_layout()
plt.show()
```



• The 'married' attribute reveals a notable disparity, with married applicants comprising approximately two-thirds of the loan application pool (1311), compared to 677 unmarried applicants. This skew suggests a potential demographic pattern within the applicant base, warranting further investigation into the underlying factors influencing loan applications.

• The 'gender' distribution exhibits a significant imbalance, with male applicants representing a substantial majority (1619) relative to female applicants (369). This observed disproportion may indicate systemic factors influencing loan application behavior, necessitating a deeper analysis into gender-specific trends and potential biases.

• The 'property_type' variable indicates a clear concentration in category '2', with 1380 applications, followed by '1' (442) and '3' (166). This distribution highlights a potential preference or market dominance of property type '2' within the loan applications, suggesting specific market dynamics or applicant preferences that merit further examination.

Q5. Build a contingency table of two potentially related categorical variables. Conduct a statistical test of the independence between them and interpret the results

In [533…

```
import scipy.stats as stats
import warnings

# Suppress FutureWarnings related to downcasting
warnings.simplefilter(action='ignore', category=FutureWarning)

# Check for missing values to confirm
print(df.isnull().sum())  # No need to re-read the CSV since we already have df

# Select two categorical variables for the contingency table
# In this case, using 'married' and 'property_type' as an example
contingency_table = pd.crosstab(df['married'], df['property_type'], margins=True, margins_name="Total")

# Display the contingency table
print("Contingency Table:")
print(contingency_table)

# Perform the Chi-Square Test of Independence
chi2_stat, p_value, dof, expected = stats.chi2_contingency(contingency_table.iloc[:-1, :-1])  # Exclude the total row and column

# Display the Chi-Square test results
print("\nChi-Square Test Result:")
print(f"Chi-Square Statistic: {chi2_stat}")
print(f"P-Value: {p_value}")
print(f"Degrees of Freedom: {dof}")
print(f"Expected Frequencies Table:\n{expected}")

# Interpret the results
alpha = 0.05  # Set significance level
if p_value < alpha:
    print("\nThe null hypothesis is rejected. There is a statistically significant association between 'married' and 'property_type'.")
else:
    print("\nThe null hypothesis is not rejected. There is no statistically significant association between 'married' and 'property_type'.")
```

```
married                    0
race                       0
loan_decision              0
occupancy                  0
loan_amount                0
applicant_income           0
num_units                  0
num_dependants             0
self_employed              0
monthly_income             0
purchase_price             0
liquid_assets              0
mortage_payment_history    0
consumer_credit_history    0
filed_bankruptcy           0
property_type              0
gender                     0
dtype: int64
Contingency Table:
property_type    1     2    3  Total
married
False          251   354   72    677
True           191  1026   94   1311
Total          442  1380  166   1988

Chi-Square Test Result:
Chi-Square Statistic: 151.51394277532896
P-Value: 1.2565082968696851e-33
Degrees of Freedom: 2
Expected Frequencies Table:
[[150.52012072 469.94969819  56.53018109]
 [291.47987928 910.05030181 109.46981891]]

The null hypothesis is rejected. There is a statistically significant association between 'married' and 'property_type'.
```

Chi-Square Test of Independence Results:

• The Chi-Square test of independence was conducted to assess the relationship between marital status ('married') and property type ('property_type').

• Null Hypothesis (H0): Marital status and property type are independent. • Alternative Hypothesis (H1): Marital status and property type are dependent.

• The test yielded a Chi-Square statistic of 151.51, a p-value of 1.26e-33, and 2 degrees of freedom.

• Since the p-value (1.26e-33) is significantly less than the alpha level of 0.05, we reject the null hypothesis. This indicates a statistically significant association between marital status and property type.

• Observed patterns in the contingency table suggest that married applicants are more likely to have property type 2, while unmarried applicants show a more even distribution across property types. This association could be influenced by various socio-economic factors or lending preferences, which could merit further investigation.

Q6. Retrieve one or more subset of rows based on two or more criteria and present descriptive statistics on the subset(s)

In [536…

```python
# Define subsets based on given criteria
subset_married_high_loan = df[(df['married'] == True) & (df['loan_amount'] > 200)]
subset_female_self_employed = df[(df['gender'] == 'female') & (df['self_employed'] == True)]

# Descriptive statistics for the first subset
desc_married_high_loan = subset_married_high_loan.describe()

# Descriptive statistics for the second subset
desc_female_self_employed = subset_female_self_employed.describe()

# Display results
desc_married_high_loan, desc_female_self_employed
```

Out[536…

```
(        occupancy  loan_amount  applicant_income   num_units  num_dependants  \
 count  209.000000   209.000000        209.000000  209.000000      209.000000
 mean     1.028708   305.808612        161.167464    1.071770        1.330144
 std      0.193993   118.935106        127.098371    0.339141        1.256177
 min      1.000000   201.000000         22.000000    1.000000        0.000000
 25%      1.000000   230.000000         99.000000    1.000000        0.000000
 50%      1.000000   267.000000        122.000000    1.000000        1.000000
 75%      1.000000   330.000000        167.000000    1.000000        2.000000
 max      3.000000   980.000000        972.000000    4.000000        7.000000

        monthly_income  purchase_price   liquid_assets  \
 count      209.000000      209.000000      209.000000
 mean     11861.291866      412.524100     5007.760909
 std      11507.828003      218.467248    69157.261274
 min        724.000000      180.000000        3.000000
 25%       6000.000000      283.000000       40.000000
 50%       8334.000000      340.000000       93.400000
 75%      12918.000000      460.000000      183.000000
 max      81000.000000     1535.000000  1000000.000000

        mortage_payment_history  consumer_credit_history  property_type
 count               209.000000               209.000000     209.000000
 mean                  1.478469                 2.066986       1.971292
 std                   0.658353                 1.573614       0.378685
 min                   1.000000                 1.000000       1.000000
 25%                   1.000000                 1.000000       2.000000
 50%                   1.000000                 1.000000       2.000000
 75%                   2.000000                 2.000000       2.000000
 max                   4.000000                 6.000000       3.000000  ,
        occupancy  loan_amount  applicant_income  num_units  num_dependants  \
 count  32.000000     32.00000         32.000000  32.000000       32.000000
 mean    1.062500    170.12500        131.625000   1.156250        0.500000
 std     0.245935    110.48157        153.889938   0.514899        0.803219
 min     1.000000     25.00000         19.000000   1.000000        0.000000
 25%     1.000000    103.00000         51.750000   1.000000        0.000000
 50%     1.000000    144.00000         79.000000   1.000000        0.000000
 75%     1.000000    188.75000        148.500000   1.000000        1.000000
 max     2.000000    600.00000        666.000000   3.000000        3.000000

        monthly_income  purchase_price  liquid_assets  mortage_payment_history  \
 count       32.000000       32.000000      32.000000                32.000000
 mean      8388.562500      276.000000     194.659375                 1.593750
 std       5654.289596      198.223074     278.565058                 0.559918
 min        674.000000       55.000000       3.000000                 1.000000
 25%       4404.000000      154.750000      30.250000                 1.000000
 50%       6593.000000      211.500000      82.500000                 2.000000
 75%      11806.000000      321.750000     212.400000                 2.000000
 max      21666.000000     1000.000000    1100.000000                 3.000000

        consumer_credit_history  property_type
 count                32.000000      32.000000
 mean                  2.156250       1.812500
 std                   1.439296       0.592289
 min                   1.000000       1.000000
 25%                   1.000000       1.000000
 50%                   2.000000       2.000000
 75%                   2.250000       2.000000
 max                   5.000000       3.000000  )
```

Analysis and Interpretation:

Married Applicants with Loan Amount > £200,000:

• This subset consists of 209 married applicants who have requested loan amounts exceeding £200,000 (i.e., loan_amount > 200 in the dataset).

• The average loan amount is £305,810, with a high standard deviation (£118,940), indicating significant variation in loan sizes.

• The average applicant income is £161,170, suggesting this group consists of high-income individuals.

• The average monthly income is £11,861, further reinforcing a strong financial background.

Female Self-Employed Applicants:

• This subset includes 32 female applicants who are self-employed.

• The average loan amount is £170,120, which is lower than that of married high-loan applicants.

• The average applicant income is £131,620, but the high standard deviation (£153,890) suggests a wide range of income levels.

• The average liquid assets are £194,660, indicating moderate financial resources.

Conclusion:

• The two subsets show distinct financial patterns:

• Married high-loan applicants tend to have higher incomes and larger loan requests, possibly indicating greater financial stability.

• Female self-employed applicants have lower loan requests and more income variation, which may suggest different financial needs or risk factors.

Q7. Conduct a statistical test of the significance of the difference between the means of two subsets of the data and interpret the results

In [539…

```python
# Subset 1: Married applicants with loan amount over 200,000
subset_married_high_loan = df[(df['married'] == True) & (df['loan_amount'] > 200)]

# Subset 2: Female self-employed applicants
subset_female_self_employed = df[(df['gender'] == 'female') & (df['self_employed'] == True)]

# Perform t-test to compare the means of loan_amount between the two subsets
t_stat, p_value = stats.ttest_ind(subset_married_high_loan['loan_amount'], subset_female_self_employed['loan_amount'])

# Display the result
print(f"T-statistic: {t_stat}")
print(f"P-value: {p_value}")

# Interpretation based on p-value
if p_value < 0.05:
    print("The difference in loan amounts between the two subsets is statistically significant.")
else:
    print("The difference in loan amounts between the two subsets is not statistically significant.")
```

```
T-statistic: 6.063916895488328
P-value: 5.1495130818247e-09
The difference in loan amounts between the two subsets is statistically significant.
```

Output Interpretation:

T-statistic: 6.06

This value indicates that the difference between the means of the two subsets is quite large compared to the variation within each subset. A t-statistic of 6.06 suggests a significant difference between the two groups.

P-value: 5.15e-09

The very small p-value (close to 0) is much lower than the typical significance threshold of 0.05. This indicates that the difference in loan amounts between the two subsets is statistically significant. In other words, we can reject the null hypothesis that the means are equal.

Conclusion:

Based on the t-test, we can conclude that the difference in loan amounts between married applicants with loan amounts over £200,000 and female self-employed applicants is statistically significant. The two groups' mean loan amounts differ considerably.

Q8. Create one or more tables that group the data by a certain categorical variable and display summarized information for each group (e.g., the mean or sum within the group)

In [542…

```python
# Grouping by 'property_type' and displaying summarized information
grouped_data = df.groupby('property_type').agg({
    'loan_amount': 'mean',
    'applicant_income': 'mean',
    'monthly_income': 'median',  # Using median for monthly income as it might be skewed
    'num_dependants': 'mean',
    'self_employed': 'sum'  # Summing self_employed to get count of self-employed applicants
})

print("Grouped Data by Property Type:")

# Display grouped data in markdown format
print(grouped_data.to_markdown(numalign="left", stralign="left"))
```

```
Grouped Data by Property Type:
| property_type   | loan_amount   | applicant_income   | monthly_income   | num_dependants   | self_employed   |
|:----------------|:--------------|:-------------------|:-----------------|:-----------------|:----------------|
| 1               | 111.181       | 82.4638            | 3700             | 0.359729         | 50              |
| 2               | 152.635       | 87.4601            | 3859             | 0.897101         | 177             |
| 3               | 150.892       | 67.5181            | 3620.5           | 0.807229         | 30              |
```

Interpretation:

Property type 2:

• Has the highest average loan amount and applicant income, indicating that applicants for this property type generally have higher financial profiles.

• Has the highest number of self-employed applicants, which could influence loan risks and opportunities.

Property type 1:

• Has the lowest average loan amount and applicant income, suggesting that applicants in this group may have lower financial profiles.

• Has the lowest number of dependents on average, which could influence the applicant's financial obligations.

Property type 3:

• Has a loan amount and income level between property types 1 and 2, but fewer self-employed applicants compared to property type 2.

Q9. Implement a linear regression model and interpret its output including its accuracy

In [545…

```python
import statsmodels.api as sm
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score

# Select relevant columns (features and target)
# Example: Predict 'loan_amount' based on 'applicant_income' and 'monthly_income'
X = df[['applicant_income', 'monthly_income']]  # Features
y = df['loan_amount']  # Target

# Add a constant to the model (for the intercept)
X = sm.add_constant(X)

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Fit the linear regression model
model = sm.OLS(y_train, X_train).fit()

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

# Predict on the test set
y_pred = model.predict(X_test)

# Evaluate the model's performance
mse = mean_squared_error(y_test, y_pred)
rmse = mean_squared_error(y_test, y_pred, squared=False)
r2 = r2_score(y_test, y_pred)

# Display the evaluation metrics
print(f"\nMean Squared Error (MSE): {mse}")
print(f"Root Mean Squared Error (RMSE): {rmse}")
print(f"R-Squared (R2): {r2}")
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:            loan_amount   R-squared:                       0.348
Model:                            OLS   Adj. R-squared:                  0.347
Method:                 Least Squares   F-statistic:                     423.5
Date:                Thu, 27 Mar 2025   Prob (F-statistic):          4.16e-148
Time:                        02:05:07   Log-Likelihood:                -8871.6
No. Observations:                1590   AIC:                         1.775e+04
Df Residuals:                    1587   BIC:                         1.777e+04
Df Model:                           2
Covariance Type:            nonrobust
====================================================================================
                       coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------------
const               94.8527      2.336     40.604      0.000      90.271      99.435
applicant_income     0.1353      0.021      6.424      0.000       0.094       0.177
monthly_income       0.0068      0.000     19.794      0.000       0.006       0.008
==============================================================================
Omnibus:                      498.893   Durbin-Watson:                   2.006
Prob(Omnibus):                  0.000   Jarque-Bera (JB):             5278.827
Skew:                           1.151   Prob(JB):                         0.00
Kurtosis:                      11.624   Cond. No.                     1.12e+04
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The condition number is large, 1.12e+04. This might indicate that there are
strong multicollinearity or other numerical problems.

Mean Squared Error (MSE): 4790.697277291705
Root Mean Squared Error (RMSE): 69.2148631241275
R-Squared (R2): 0.3325605236155329
```

Model Fit:

• R-squared (0.348): The model explains about 34.8% of the variance in loan amounts using applicant income and monthly income. This suggests that other factors influence loan amounts beyond these two variables.

• Adjusted R-squared (0.347): Similar to R-squared, but adjusted for the number of predictors. Since it is close to R-squared, the predictors contribute meaningfully to the model.

Significance of Predictors:

• Applicant Income (coef = 0.1353, $p < 0.05$): A statistically significant predictor. A unit increase in applicant income is associated with an increase of 0.1353 in loan amount, holding other variables constant.

• Monthly Income (coef = 0.0068, $p < 0.05$): Also statistically significant. A unit increase in monthly income corresponds to a 0.0068 increase in loan amount.

Error Metrics:

• Mean Squared Error (MSE = 4790.70): The average squared difference between actual and predicted loan amounts.

• Root Mean Squared Error (RMSE = 69.21): The typical error in predictions is about 69.21 (in thousands).

• Durbin-Watson (2.006): Indicates no significant autocorrelation in residuals.

Potential Concerns:

• Multicollinearity Warning (Condition Number = 1.12e+04): A high condition number suggests multicollinearity, meaning that applicant income and monthly income might be highly correlated.