EDA Software Analytics Coursework

Candidate Number: 418207

In [118...
```python
# Importing the required libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import statsmodels.api as sm
from scipy import stats
from sklearn.metrics import classification_report
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn import metrics
from sklearn.metrics import confusion_matrix
```

In [119...
```python
# Loading the CSV file into the dataframe and displaying the first 10 entries

df_1= pd.read_csv("loanapp.csv")
df_1.head(10)
```

Out[119...

	married	race	loan_decision	occupancy	loan_amount	applicant_income	num_units	num_dependants	self_employed	monthly_income	pu
0	True	white	reject	1	128	74	1.0	1.0	False	4583	
1	False	white	approve	1	128	84	1.0	0.0	False	2666	
2	True	white	approve	1	66	36	1.0	0.0	True	3000	
3	True	white	approve	1	120	59	1.0	0.0	False	2583	
4	False	white	approve	1	111	63	1.0	0.0	False	2208	
5	False	white	approve	1	141	72	1.0	0.0	False	6000	
6	False	white	approve	1	276	90	1.0	0.0	False	7500	
7	True	white	approve	1	100	72	1.0	1.0	False	3234	
8	True	white	approve	1	267	144	1.0	2.0	True	5417	
9	True	white	approve	1	175	97	1.0	0.0	False	6292	

In [120...
```python
# Displaying the column names and the datatypes in the dataframe
df_1.info()
```

```
<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
```

In [121...
```python
# Displaying the number of rows and columns in the dataframe
df_1.shape
```

Out[121...
```
(1988, 17)
```

1.Display descriptive statistics on the dataset.

For displaying the descriptive statistics of the dataset, we can call the describe function in pandas. It will display the mean, min/max values, standard deviation etc., for each column.

In [122...
```python
# Calling the describe function on the dataframe
df_1.describe()
```

Out[122...

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

1. Check if any records in the data have any missing values; handle the missing data as appropriate.

For checking if a column is null or not we can use isnull() which will return either True (for Null values) or False. To find the sum of all null values we can use the sum() over the isnull(). After finding the number of columns with null values, we can either drop those rows or interpolate values. Using dropna() and giving the columns with Null entries as the parameter we can delete rows with Null values. After dropping those rows, using info() we can check the new shape of the dataframe. The number of rows has decreased from 1988 to 1969 after cleaning the data.

In [123...
```python
# Finding the count of null values using isnull()
df_1.isnull().sum()
```

Out[123...
```
married                     3
race                        0
loan_decision               0
occupancy                   0
loan_amount                 0
applicant_income            0
num_units                   4
num_dependants              3
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                     14
dtype: int64
```

In [124...
```python
# dropping the rows with Null values in the below columns
df_1=df_1.dropna(subset=['num_units','num_dependants','married','gender'])
```

In [125...
```python
# Finding the count of null values using isnull()
df_1.isnull().sum()
```

Out[125...
```
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
```

In [126...
```python
df_1.shape
```

Out[126...
```
(1969, 17)
```

In [127...
```python
df_1.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 1969 entries, 0 to 1987
Data columns (total 17 columns):
 #   Column                   Non-Null Count  Dtype  
---  ------                   --------------  -----  
 0   married                  1969 non-null   object 
 1   race                     1969 non-null   object 
 2   loan_decision            1969 non-null   object 
 3   occupancy                1969 non-null   int64  
 4   loan_amount              1969 non-null   int64  
 5   applicant_income         1969 non-null   int64  
 6   num_units                1969 non-null   float64
 7   num_dependants           1969 non-null   float64
 8   self_employed            1969 non-null   bool   
 9   monthly_income           1969 non-null   int64  
 10  purchase_price           1969 non-null   float64
 11  liquid_assets            1969 non-null   float64
 12  mortage_payment_history  1969 non-null   int64  
 13  consumer_credit_history  1969 non-null   int64  
 14  filed_bankruptcy         1969 non-null   bool   
 15  property_type            1969 non-null   int64  
 16  gender                   1969 non-null   object 
dtypes: bool(2), float64(4), int64(7), object(4)
memory usage: 250.0+ KB
```

In [128...
```python
df_1.describe()
```

Out[128...

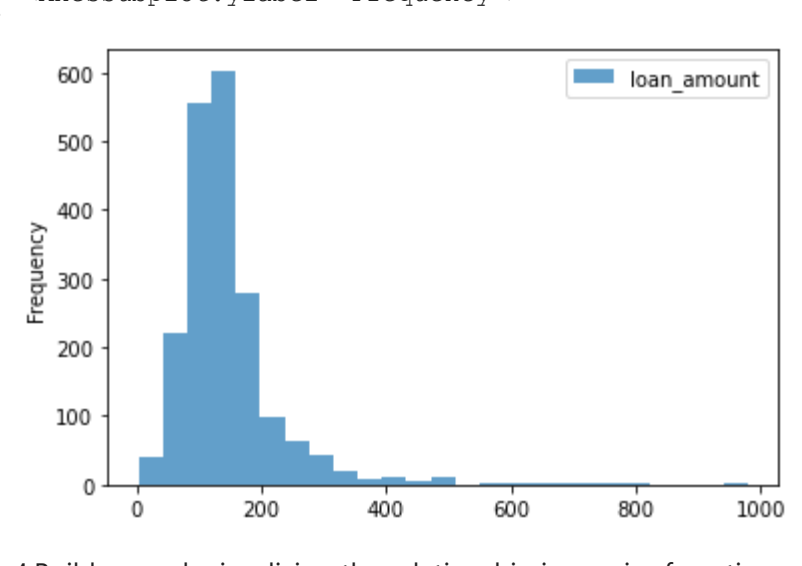
	occupancy	loan_amount	applicant_income	num_units	num_dependants	monthly_income	purchase_price	liquid_assets	mortage_
count	1969.000000	1969.000000	1969.000000	1969.000000	1969.000000	1969.000000	1969.000000	1969.000000	
mean	1.031996	143.505333	84.908075	1.122905	0.770442	5204.838497	196.748566	4664.390041	
std	0.192576	80.802291	87.439115	0.438410	1.103450	5290.834720	128.587218	67464.766307	
min	1.000000	2.000000	0.000000	1.000000	0.000000	0.000000	25.000000	0.000000	
25%	1.000000	100.000000	48.000000	1.000000	0.000000	2876.000000	130.000000	20.000000	
50%	1.000000	126.000000	64.000000	1.000000	0.000000	3812.000000	163.000000	38.000000	
75%	1.000000	165.000000	88.000000	1.000000	1.000000	5600.000000	225.000000	83.000000	
max	3.000000	980.000000	972.000000	4.000000	8.000000	81000.000000	1535.000000	1000000.000000	

1. Build a graph visualizing the distribution of one or more individual continuous variables of the dataset.

To plot graphs in Python, we can use plot over the dataframe variable. To plot a histogram we will call plot over the dataframe df_1 variable 'loan_amount' with hist(for histogram). The bin size can be mentioned as a parameter inside the hist function. From the graph we can see that the maximum loan amount that has been given out is in the 0-200 range. Very less number of people have been given loans greater than 200.

In [129...
```python
# Plotting a histogram based on loan_amount
df_1[['loan_amount']].plot.hist(alpha=0.7,bins=25)
```

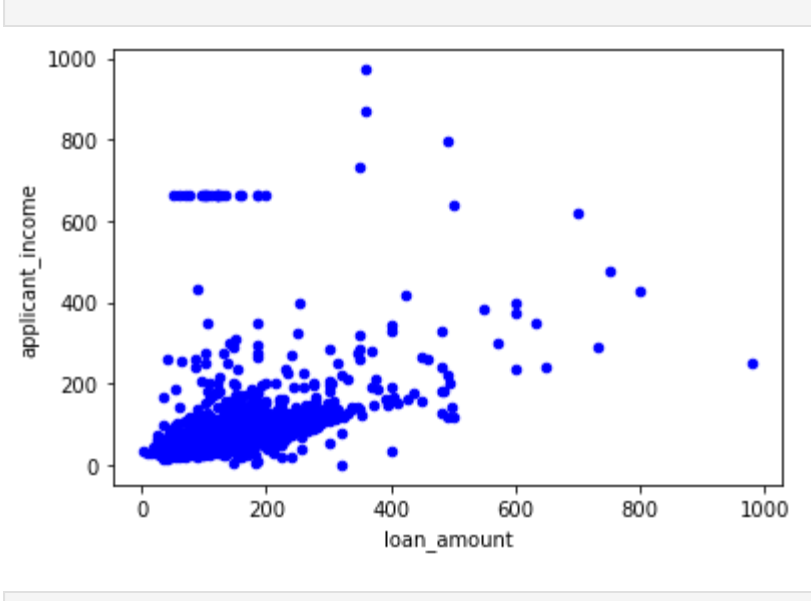
Out[129...
```
<AxesSubplot:ylabel='Frequency'>
```



4.Build a graph visualizing the relationship in a pair of continuous variables. Determine the correlation between them.

For plotting a scatter plot, we can use the plot function again and then mention the plot type as scatter in the parameter kind. We also need to mention the variables to be used for the x and y axis. We can also specify the color for the scatter plot. From the graph we can interpret that there is no proper relationship between loan amount and applicant income. For calculating the correlation between a pair of continous variables, we are using Pearsons correlation. We will call the corr() and mention the columns with which we want to check the correalation and then specify the method as 'pearson' as the parameter for corr().

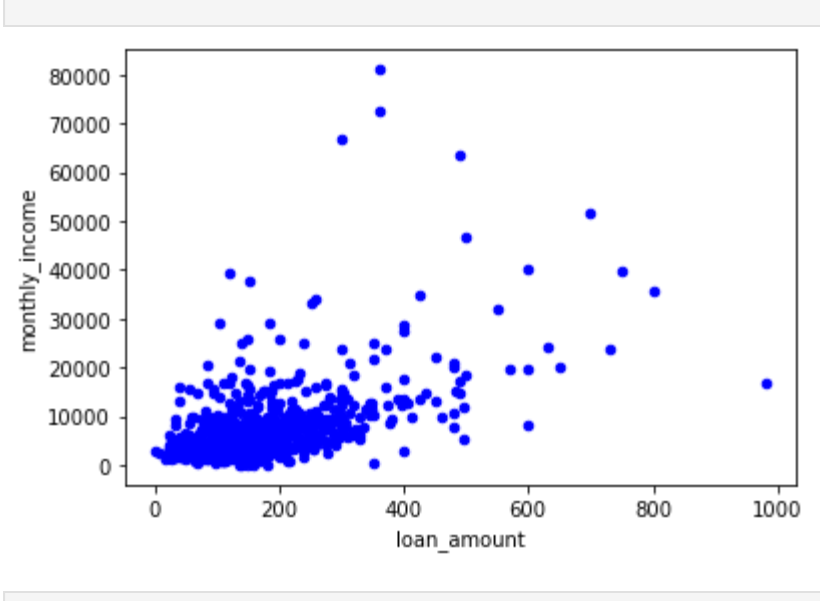
In [130...
```python
# Plotting a scatter plot based on loan_amount and applicant_income
df_1.plot(kind='scatter',x='loan_amount',y='applicant_income',color='blue')
plt.show()
```



In [131...
```python
#Pearsons Correlation using correlation function
df_1['loan_amount'].corr(df_1['applicant_income'],method='pearson')
```

Out[131...
```
0.43271137519859754
```

In [132...
```python
# Plotting a scatter plot based on loan_amount and monthly_income
df_1.plot(kind='scatter',x='loan_amount',y='monthly_income',color='blue')
plt.show()
```



In [133...
```python
#Pearsons Correlation using correlation function
df_1['loan_amount'].corr(df_1['monthly_income'],method='pearson')
```

Out[133...
```
0.5740895502791064
```

5.Display unique values of a categorical variable.

For displaying the unique values for a categorical value we can either use value_counts() on the columns of a dataframe or use the unique() on the dataframe column.

In [134...
```python
#Unique values of column married from the data file
married = df_1['married'].value_counts()
print("\n Unique values of married are:\n",married)

#Unique values of column gender from the data file
race = df_1['race'].value_counts()
print("\n Unique values of race are:\n",race)

#Unique values of column race from the data file
gender = df_1['gender'].value_counts()
print("\n Unique values of gender are:\n",gender)

#Unique values of column race from the data file
loan_decision = df_1['loan_decision'].value_counts()
print("\n Unique values of loan_decision are:\n",loan_decision)

#Unique values of column race from the data file
self_employed = df_1['self_employed'].value_counts()
print("\n Unique values of self_employed are:\n",self_employed)

#Unique values of column race from the data file
filed_bankruptcy = df_1['filed_bankruptcy'].value_counts()
print("\n Unique values of filed_bankruptcy are:\n",filed_bankruptcy)
```

```
 Unique values of married are:
 True     1298
False     671
Name: married, dtype: int64

 Unique values of race are:
 white     1666
black      195
hispan     108
Name: race, dtype: int64

 Unique values of gender are:
 male      1601
female     368
Name: gender, dtype: int64

 Unique values of loan_decision are:
 approve    1727
reject      242
Name: loan_decision, dtype: int64

 Unique values of self_employed are:
 False    1712
True      257
Name: self_employed, dtype: int64

 Unique values of filed_bankruptcy are:
 False    1834
True      135
Name: filed_bankruptcy, dtype: int64
```

6.Build a contingency table of two potentially related categorical variables. Conduct a statistical test of the independence between them.

For building a contingency table we will be using the crosstab function. In the crosstab function we will mention the columns from the dataframe which are needed for building the contingeny table.

After the contingency table is created, we will perform a chi square test on the contingency table to generate the p-value. If the p-value is less than the significane level of 0.05 we will reject the null hypothesis. If its greater than the significance level, we will accept the null hypothesis. The Null hypothesis states that there is no dependence between the columns we took for building the contingency table.

In [135...
```python
#Contingency table with the help of a crosstab function
contingency_table=pd.crosstab(df_1['self_employed'], df_1['loan_decision'])
contingency_table.head()
```

Out[135...

loan_decision	approve	reject
self_employed		
False	1510	202
True	217	40

In [136...
```python
# Running a chi-square test and calculating the p-value
chi2, p_val,dof,expected=stats.chi2_contingency(contingency_table)
print(f"p-value: {p_val}")
```

```
p-value: 0.10688538967672208
```

In [137...
```python
#Contingency table with the help of a crosstab function
contingency_table_2=pd.crosstab(df_1['filed_bankruptcy'], df_1['loan_decision'])
contingency_table_2.head()
```

Out[137...

loan_decision	approve	reject
filed_bankruptcy		
False	1653	181
True	74	61

In [138...
```python
# Running a chi-square test and calculating the p-value
chi2, p_val,dof,expected=stats.chi2_contingency(contingency_table_2)
print(f"p-value: {p_val}")
```

```
p-value: 8.677306949234421e-33
```

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

For retrieving a subset of the data, we will specify the conditions using logical operators or by specifying the condition with which we need the data. Here we have taken 2 subsets of the dataframe based on gender,approve as the loan decision and where the number of dependants are less than 2. For generating the descriptive statistical report of the subset, we can use the describe(). From the data we can see that the mean loan amount for male is greater than that of females.

In [139...
```python
# Creating a subset of the dataset based on gender, loan_decision and number of dependants.
df_1_subset_male=df_1[(df_1['gender']=='male') & (df_1['loan_decision']=='approve') &(df_1['num_dependants']< 2
df_1_subset_male.describe()
```

Out[139...

	occupancy	loan_amount	applicant_income	num_units	num_dependants	monthly_income	purchase_price	liquid_assets	mortage_
count	1013.000000	1013.000000	1013.000000	1013.000000	1013.000000	1013.000000	1013.000000	1013.000000	
mean	1.028628	144.150049	83.465943	1.107601	0.228036	5007.127345	196.387868	6018.981464	
std	0.183752	80.732254	74.259001	0.416165	0.419773	4945.086190	127.778497	76763.678020	
min	1.000000	2.000000	5.000000	1.000000	0.000000	0.000000	25.000000	0.000000	
25%	1.000000	104.000000	51.000000	1.000000	0.000000	2773.000000	132.000000	21.000000	
50%	1.000000	129.000000	66.000000	1.000000	0.000000	3727.000000	165.000000	40.450000	
75%	1.000000	166.000000	89.000000	1.000000	0.000000	5500.000000	225.000000	87.100000	
max	3.000000	980.000000	796.000000	4.000000	1.000000	67000.000000	1535.000000	1000000.000000	

In [140...
```python
# Describing the subset based on the loan_amount
df_1_subset_male['loan_amount'].describe()
```

Out[140...
```
count    1013.000000
mean      144.150049
std        80.732254
min         2.000000
25%       104.000000
50%       129.000000
75%       166.000000
max       980.000000
Name: loan_amount, dtype: float64
```

In [141...
```python
# Creating a subset of the dataset based on gender, loan_decision and number of dependants.
df_1_subset_female=df_1[(df_1['gender']=='female') & (df_1['loan_decision']=='approve') & (df_1['num_dependants
df_1_subset_female.describe()
```

Out[141...

	occupancy	loan_amount	applicant_income	num_units	num_dependants	monthly_income	purchase_price	liquid_assets	mortage_paym
count	290.000000	290.000000	290.000000	290.000000	290.000000	290.000000	290.000000	290.000000	
mean	1.027586	117.534483	68.655172	1.110345	0.158621	4364.134483	159.822517	85.261310	
std	0.183952	60.993055	79.635231	0.417891	0.365954	3220.611697	93.074208	135.104745	
min	1.000000	26.000000	4.000000	1.000000	0.000000	0.000000	29.000000	0.760000	
25%	1.000000	84.250000	40.000000	1.000000	0.000000	2707.250000	110.500000	23.250000	
50%	1.000000	108.000000	52.000000	1.000000	0.000000	3483.000000	140.000000	39.700000	
75%	1.000000	137.000000	68.000000	1.000000	0.000000	4497.250000	177.750000	85.030000	
max	3.000000	600.000000	666.000000	4.000000	1.000000	25750.000000	1000.000000	1000.000000	

In [142...
```python
# Describing the subset based on the loan_amount
df_1_subset_female['loan_amount'].describe()
```

Out[142...
```
count    290.000000
mean     117.534483
std       60.993055
min       26.000000
25%       84.250000
50%      108.000000
75%      137.000000
max      600.000000
Name: loan_amount, dtype: float64
```

8.Conduct a statistical test of the significance of the difference between the means of two subsets of the data.

For conducting a statistical test on the difference between the means of the 2 subsets of the data, we will be performing an independent 2 sample t-test. Here the null hypothesis states that there is no difference beteween the means of the 2 subsets of data. If the p-value is less than the significane level of 0.05, we reject the Null hypothesis. In this case we will be rejecting the Null hypothesis which means that there is significance difference between the means of the 2 subsets.

In [143...
```python
# Performing an independent 2 sample t-test between both the subset
t_value, p_value = stats.ttest_ind(df_1_subset_male['loan_amount'],df_1_subset_female['loan_amount'])
print(f"t-value: {t_value},p-value: {p_value}")
```

```
t-value: 5.204499158870428,p-value: 2.2582328392483397e-07
```

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

For creating a table with certain categorical variable, we can use the groupby function on the dataframe. We need to pass the column name as a parameter and then call the mean() or sum() to calculate the same on the grouped data.

In [144...
```python
# Using groupby function to group the data based on a categorical variable
df_1.groupby(['gender']).mean()
```

Out[144...

	occupancy	loan_amount	applicant_income	num_units	num_dependants	self_employed	monthly_income	purchase_price	liquid_assets
gender									
female	1.021739	120.500000	70.763587	1.144022	0.385870	0.086957	4350.442935	164.040842	106.058190
male	1.034354	148.793254	88.159275	1.118051	0.858838	0.140537	5401.226733	204.266644	5712.151515

In [145...
```python
# Using groupby function to group the data based on a categorical variable
df_1.groupby(['race']).mean()
```

Out[145...

	occupancy	loan_amount	applicant_income	num_units	num_dependants	self_employed	monthly_income	purchase_price	liquid_assets
race									
black	1.015385	136.630769	77.035897	1.302564	0.876923	0.112821	4715.010256	167.421282	60.667436
hispan	1.000000	130.777778	68.712963	1.203704	1.046296	0.101852	4092.388889	155.946759	50.049259
white	1.036014	145.135054	86.879352	1.096639	0.740096	0.134454	5334.286915	202.826247	5502.370060

In [146...
```python
# Using groupby function to group the data based on a categorical variable
df_1.groupby(['loan_decision']).mean()
```

Out[146...

	occupancy	loan_amount	applicant_income	num_units	num_dependants	self_employed	monthly_income	purchase_price	liquid_
loan_decision									
approve	1.029531	143.777649	84.101911	1.103648	0.761436	0.125651	5256.380428	200.106588	4724.9
reject	1.049587	141.561983	90.661157	1.260331	0.834711	0.165289	4837.016529	172.784504	4232.4

10.Implement a linear regression model and interpret its output.

In [147...
```python
#The values of gender and race variables are replaced into o and 1.As string values of this variable may affect
df_1 =df_1.replace({"gender":{"male": 1, "female": 0}})
df_1 =df_1.replace({"race":{"white": 0, "black": 1,"hispanic":2}})
df_1 =df_1.replace({"loan_decision":{"approve": 1, "reject": 0}})
df_1 =df_1.replace({"married":{True: 1, False: 0}})
df_1 =df_1.replace({"self_employed":{True: 1, False: 0}})
df_1 =df_1.replace({"filed_bankruptcy":{True: 1, False: 0}})
df_1.head()
```

Out[147...

	married	race	loan_decision	occupancy	loan_amount	applicant_income	num_units	num_dependants	self_employed	monthly_income	pur
0	1	0	0	1	128	74	1.0	1.0	0	4583	
1	0	0	1	1	128	84	1.0	0.0	0	2666	
2	1	0	1	1	66	36	1.0	0.0	1	3000	
3	1	0	1	1	120	59	1.0	0.0	0	2583	
4	0	0	1	1	111	63	1.0	0.0	0	2208	

In [148...
```python
df_1.corr()
```

Out[148...

	married	loan_decision	occupancy	loan_amount	applicant_income	num_units	num_dependants	self_employed	m
married	1.000000	0.063740	0.013743	0.182586	0.098420	-0.069750	0.358382	0.027294	
loan_decision	0.063740	1.000000	-0.034202	0.009005	-0.024636	-0.117370	-0.021808	-0.038632	
occupancy	0.013743	-0.034202	1.000000	-0.017008	0.029898	0.073769	0.003496	0.084390	
loan_amount	0.182586	0.009005	-0.017008	1.000000	0.432711	0.040274	0.139822	0.145159	
applicant_income	0.098420	-0.024636	0.029898	0.432711	1.000000	-0.055974	0.103319	0.127648	
num_units	-0.069750	-0.117370	0.073769	0.040274	-0.055974	1.000000	-0.006772	0.046137	
num_dependants	0.358382	-0.021808	0.003496	0.139822	0.103319	-0.006772	1.000000	0.042360	
self_employed	0.027294	-0.038632	0.084390	0.145159	0.127648	0.046137	0.042360	1.000000	
monthly_income	0.087152	0.026031	0.085313	0.574090	0.565773	-0.008528	0.145042	0.181424	
purchase_price	0.178697	0.069781	-0.026207	0.834387	0.449911	-0.011270	0.167312	0.149384	
liquid_assets	0.016963	0.002397	0.106203	0.014818	-0.007633	-0.019116	0.007274	0.041041	
mortage_payment_history	-0.147549	-0.117681	-0.073657	-0.153051	-0.057000	0.008702	-0.096291	-0.028365	
consumer_credit_history	0.005126	-0.281428	-0.042771	-0.003109	0.038616	0.067171	0.007983	-0.038398	
filed_bankruptcy	-0.004216	-0.271822	-0.024212	0.025943	0.042242	0.047722	0.063744	0.014195	
property_type	0.167516	-0.070272	0.032889	0.179422	-0.021726	0.542070	0.159857	0.043312	
gender	0.347964	0.018931	0.025542	0.136535	0.077575	-0.023099	0.167134	0.062002	

In [149...
```python
#Using OLS regression to obtain linear regression model
mod = sm.OLS.from_formula('loan_amount ~  applicant_income+purchase_price', df_1)
solution = mod.fit()
#To get detailed information about the model, we can print the results of a call to the .summary() method
solution.summary()
```

Out[149...

OLS Regression Results

Dep. Variable:	loan_amount	R-squared:	0.700
Model:	OLS	Adj. R-squared:	0.700
Method:	Least Squares	F-statistic:	2297.
Date:	Thu, 24 Mar 2022	Prob (F-statistic):	0.00
Time:	23:09:22	Log-Likelihood:	-10255.
No. Observations:	1969	AIC:	2.052e+04
Df Residuals:	1966	BIC:	2.053e+04
Df Model:	2		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
Intercept	38.7057	1.851	20.916	0.000	35.077	42.335
applicant_income	0.0664	0.013	5.198	0.000	0.041	0.091
purchase_price	0.5040	0.009	58.017	0.000	0.487	0.521

Omnibus:	816.040	Durbin-Watson:	1.987
Prob(Omnibus):	0.000	Jarque-Bera (JB):	86363.315
Skew:	-0.966	Prob(JB):	0.00
Kurtosis:	35.387	Cond. No.	472.

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.

The model which we obtained has a good Adjusted R square value of 0.70 with 0 insignificant variables.

1. Implementing a Logistic Regression Model

Below a logistic regression model has been implemented with loan_decision as the dependent variable and all the other variables as independent variables. Training and test sets were created from the original dataset using the test_train_split function and by passing the test_size and the dependent and independent columns. An object of Logistic Regression is created to and using the fit() we are training the data. Using the predict() and score() we can print the accuracy of the classfier. And by importing confusion matrix from sklearn we can print the confusion matrix as well.

In [150...
```python
# Creating 2 different subsets x and y with dependent and independent variables
x=df_1[['applicant_income','purchase_price','gender','married','self_employed','occupancy',
       'num_units','num_dependants','monthly_income','purchase_price','liquid_assets','mortage_payment_history',
'consumer_credit_history','filed_bankruptcy','property_type']]
y=df_1['loan_decision']

# Creating the test and training dataset by splitting the given data
X_train, X_test, y_train, y_test = train_test_split(x, y, test_size=0.3, random_state=101)

# Creating a logistic regression object and fitting it with the training data
lm = LogisticRegression()

lm.fit(X_train, y_train)
```

Out[150...
```
LogisticRegression()
```

In [151...
```python
# Printing the accuracy of the classifier
y_pred = lm.predict(X_test)
print('Accuracy of logistic regression classifier: ',lm.score(X_test, y_test))
```

```
Accuracy of logistic regression classifier:  0.868020304568528
```

In [152...
```python
# Printing the confusion matrix
confusion_matrix = confusion_matrix(y_test, y_pred)
print(confusion_matrix)
```

```
[[  0  78]
 [  0 513]]
```

In [153...
```python
# Printing the classification report
print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.00      0.00      0.00        78
           1       0.87      1.00      0.93       513

    accuracy                           0.87       591
   macro avg       0.43      0.50      0.46       591
weighted avg       0.75      0.87      0.81       591
```

```
C:\Users\prana\anaconda3\lib\site-packages\sklearn\metrics\_classification.py:1248: UndefinedMetricWarning: Precision and F-score are ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
C:\Users\prana\anaconda3\lib\site-packages\sklearn\metrics\_classification.py:1248: UndefinedMetricWarning: Precision and F-score are ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
C:\Users\prana\anaconda3\lib\site-packages\sklearn\metrics\_classification.py:1248: UndefinedMetricWarning: Precision and F-score are ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
```

In [ ]:

In [ ]: