Problem Statement : Dream Housing Finance company deals in all home loans. They have a presence across all urban, semi-urban and rural areas. Customers first apply for a home loan after that company validates the customer's eligibility for a loan. The company wants to automate the loan eligibility process (real-time) based on customer detail provided while filling out the online application form. These details are Gender, Marital Status, Education, Number of Dependents, Income, Loan Amount, Credit History, and others. To automate this process, they have given a problem to identify the customer segments, that are eligible for loan amounts so that they can specifically target these customers.

In [135]:

```
import pandas as pd
import sqlalchemy as sa
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score,roc_auc_score,confusion_matrix,classification_report
import warnings
warnings.filterwarnings("ignore")
```

In [42]:

```
engine=sa.create_engine("mysql+pymysql://root:Payal123@localhost:3307/db_loan")
engine
```

Out[42]:

```
Engine(mysql+pymysql://root:***@localhost:3307/db_loan)
```

In [43]:

```
df=pd.read_sql_table('train',engine)
df.head(2)
```

Out[43]:

	MyUnknownColumn	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amo
0	0	LP001002	Male	No	0.0	Graduate	No	5849	0.0	0.0	
1	1	LP001003	Male	Yes	1.0	Graduate	No	4583	1508.0	128.0	

In [44]:

```
df.shape
```

Out[44]:

```
(614, 14)
```

In [45]:

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 614 entries, 0 to 613
Data columns (total 14 columns):
 #   Column             Non-Null Count  Dtype  
---  ------             --------------  -----  
 0   MyUnknownColumn    614 non-null    int64  
 1   Loan_ID            614 non-null    object 
 2   Gender             614 non-null    object 
 3   Married            614 non-null    object 
 4   Dependents         614 non-null    float64
 5   Education          614 non-null    object 
 6   Self_Employed      614 non-null    object 
 7   ApplicantIncome    614 non-null    int64  
 8   CoapplicantIncome  614 non-null    float64
 9   LoanAmount         614 non-null    float64
 10  Loan_Amount_Term   614 non-null    float64
 11  Credit_History     614 non-null    float64
 12  Property_Area      614 non-null    object 
 13  Loan_Status        614 non-null    object 
dtypes: float64(5), int64(2), object(7)
memory usage: 67.3+ KB
```

EDA

In [46]:

```
df.isna().sum()
```

Out[46]:

```
MyUnknownColumn      0
Loan_ID              0
Gender               0
Married              0
Dependents           0
Education            0
Self_Employed        0
ApplicantIncome      0
CoapplicantIncome    0
LoanAmount           0
Loan_Amount_Term     0
Credit_History       0
Property_Area        0
Loan_Status          0
dtype: int64
```

In [47]:

```
df.select_dtypes("object")
```

Out[47]:

	Loan_ID	Gender	Married	Education	Self_Employed	Property_Area	Loan_Status
0	LP001002	Male	No	Graduate	No	Urban	Y
1	LP001003	Male	Yes	Graduate	No	Rural	N
2	LP001005	Male	Yes	Graduate	Yes	Urban	Y
3	LP001006	Male	Yes	Not Graduate	No	Urban	Y
4	LP001008	Male	No	Graduate	No	Urban	Y
...	...	...	...	...	...	...	...
609	LP002978	Female	No	Graduate	No	Rural	Y
610	LP002979	Male	Yes	Graduate	No	Rural	Y
611	LP002983	Male	Yes	Graduate	No	Urban	Y
612	LP002984	Male	Yes	Graduate	No	Urban	Y
613	LP002990	Female	No	Graduate	Yes	Semiurban	N

614 rows × 7 columns

Changing datatype (OBJECT TO INT)

LOAN_ID

In [48]:

```
df["Loan_ID"] = df["Loan_ID"].str[2:]
```

In [49]:

```
df["Loan_ID"] = df["Loan_ID"].astype(int)
```

GENDER

In [50]:

```
df["Gender"].value_counts().to_dict()
```

Out[50]:

```
{'Male': 489, 'Female': 112, '0': 13}
```

In [51]:

```
df["Gender"] = df["Gender"].replace({'Male': 1, 'Female': 2, '0': 0})
```

MARRIED

In [52]:

```
df["Married"].value_counts().to_dict()
```

Out[52]:

```
{'Yes': 398, 'No': 213, '0': 3}
```

In [53]:

```
df["Married"] = df["Married"].replace({'Yes': 1, 'No': 0, '0': 2})
```

Education

In [56]:

```
df["Education"].value_counts().to_dict()
```

Out[56]:

```
{'Graduate': 480, 'Not Graduate': 134}
```

In [57]:

```
df["Education"] = df["Education"].replace({'Graduate': 1, 'Not Graduate': 0})
```

SELF_EMPLOYED

In [60]:

```
df["Self_Employed"].value_counts().to_dict()
```

Out[60]:

```
{'No': 500, 'Yes': 82, '0': 32}
```

In [61]:

```
df["Self_Employed"] = df["Self_Employed"].replace({'No': 0, 'Yes': 1, '0': 2})
```

Property_Area

In [64]:

```
df["Property_Area"].value_counts().to_dict()
```

Out[64]:

```
{'Semiurban': 233, 'Urban': 202, 'Rural': 179}
```

In [65]:

```
df["Property_Area"] = df["Property_Area"].replace({'Semiurban': 2, 'Urban': 1, 'Rural': 0})
```

Target Column LOAN STATUS

In [67]:

```
df.Loan_Status.value_counts().to_dict()
```

Out[67]:

```
{'Y': 422, 'N': 192}
```

In [68]:

```
df["Loan_Status"] = df["Loan_Status"].replace({'Y': 1, 'N': 0})
```

observation:

- target class is imbalanced.

In [69]:

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 614 entries, 0 to 613
Data columns (total 14 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   MyUnknownColumn    614 non-null    int64
 1   Loan_ID            614 non-null    int32
 2   Gender             614 non-null    int64
 3   Married            614 non-null    int64
 4   Dependents         614 non-null    float64
 5   Education          614 non-null    int64
 6   Self_Employed      614 non-null    int64
 7   ApplicantIncome    614 non-null    int64
 8   CoapplicantIncome  614 non-null    float64
 9   LoanAmount         614 non-null    float64
 10  Loan_Amount_Term   614 non-null    float64
 11  Credit_History     614 non-null    float64
 12  Property_Area      614 non-null    int64
 13  Loan_Status        614 non-null    int64
dtypes: float64(5), int32(1), int64(8)
memory usage: 64.9 KB
```

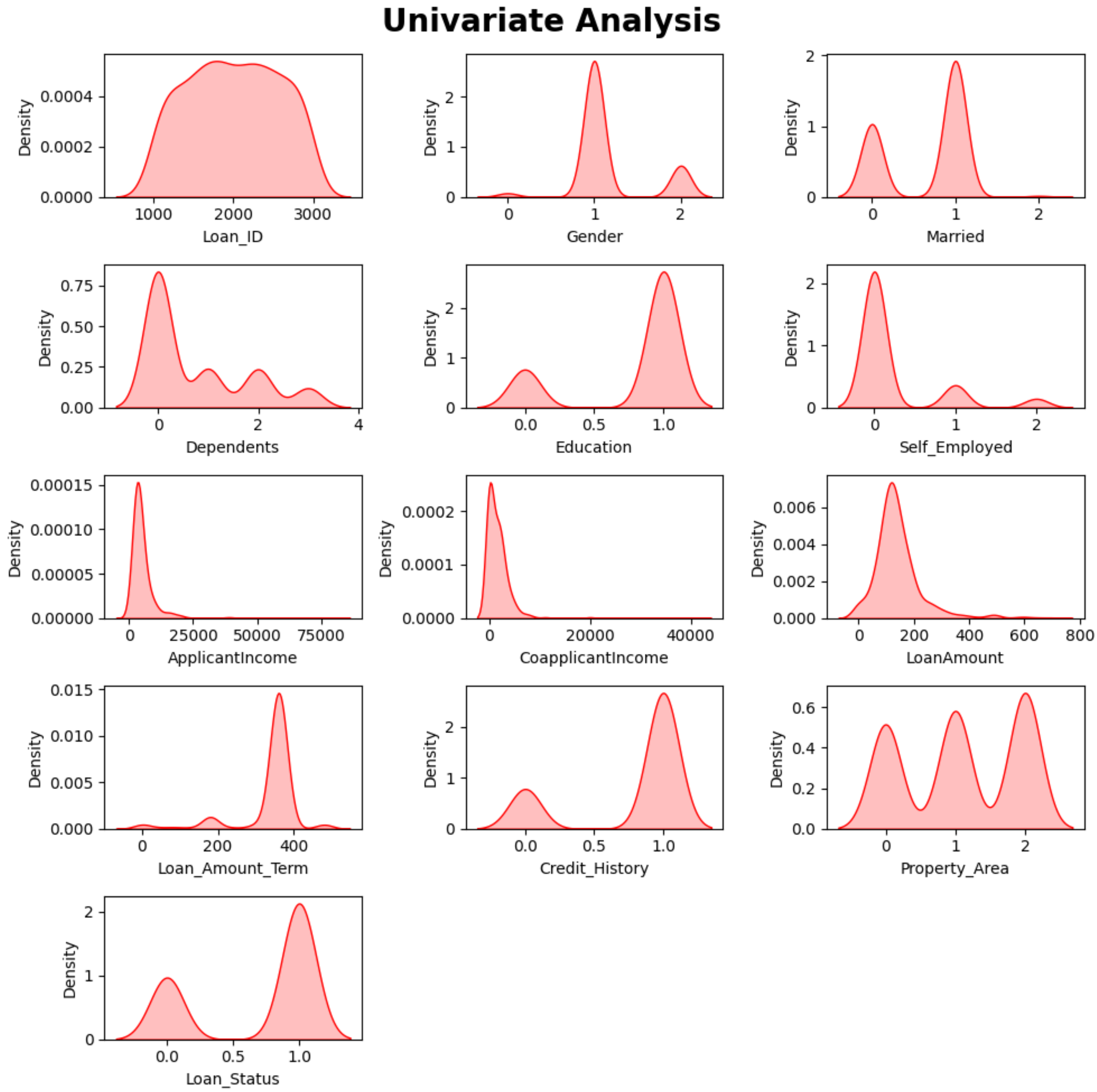
In [72]:

```
df.drop("MyUnknownColumn",axis=1,inplace=True)
```

univariate analysis

In [88]:

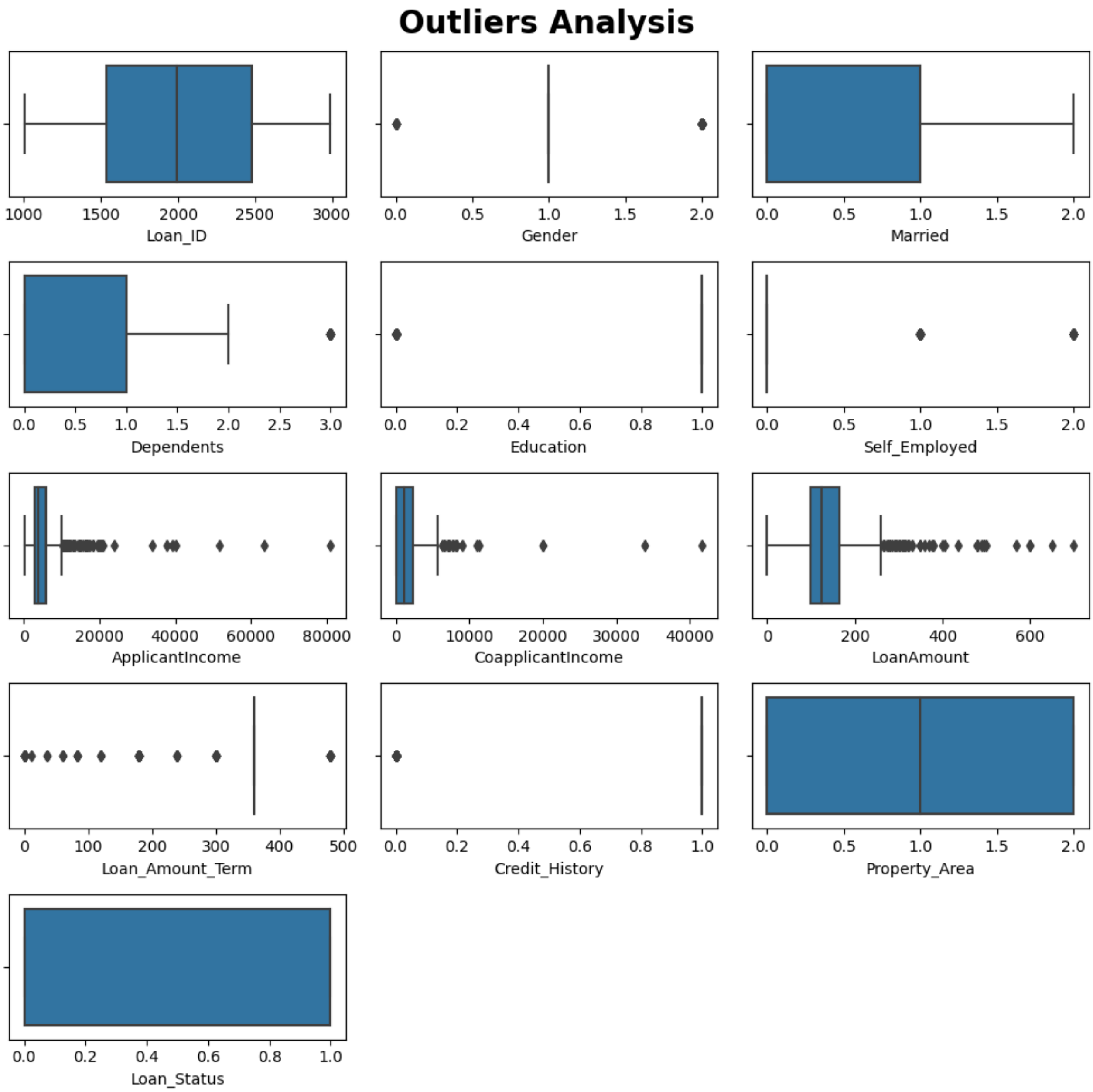
```
plt.figure(figsize=(10,10))
plt.suptitle("Univariate Analysis",fontsize=20 , fontweight="bold")
for i in range(0,len(df.columns)):
            plt.subplot(5,3,i+1)
            sns.kdeplot(x=df[df.columns[i]],shade=True,color="r")
            plt.xlabel(df.columns[i])
            plt.tight_layout()
```



Outliers Analysis

In [90]:

```
plt.figure(figsize=(10,10))
plt.suptitle("Outliers Analysis",fontsize=20 , fontweight="bold")
for i in range(0,len(df.columns)):
               plt.subplot(5,3,i+1)
               sns.boxplot(x=df[df.columns[i]])
               plt.xlabel(df.columns[i])
               plt.tight_layout()
```



In [ ]:

Handling Outliers

Type *Markdown* and LaTeX: α^2

GENDER

In [98]:

```
df["Gender"].describe()
```

Out[98]:

```
count    614.000000
mean       1.161238
std        0.421752
min        0.000000
25%        1.000000
50%        1.000000
75%        1.000000
max        2.000000
Name: Gender, dtype: float64
```

In [99]:

```
print(df['Gender'].quantile(0.10))
print(df['Gender'].quantile(0.90))
```

```
1.0
2.0
```

In [100]:

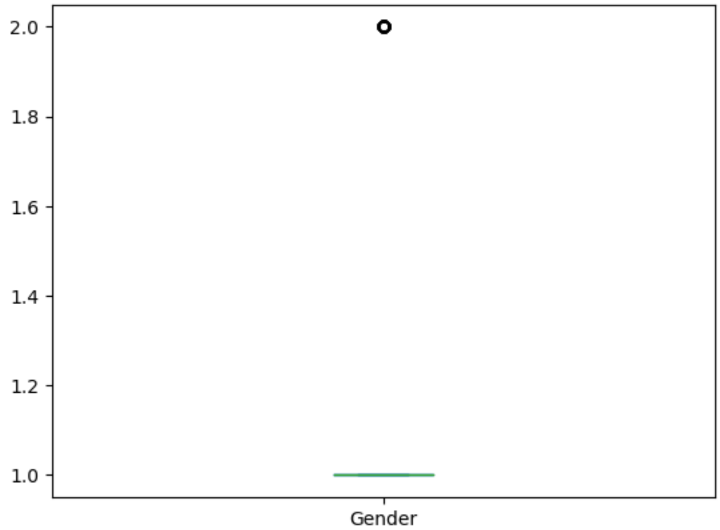
```
df['Gender'] = np.where(df['Gender'] <1, 1,df['Gender'])
df['Gender'] = np.where(df['Gender'] >2, 2,df['Gender'])
```

In [101]:

```
df["Gender"].plot(kind="box")
```

Out[101]:

```
<AxesSubplot: >
```



Self_Employed

In [102]:

```
df["Self_Employed"].describe()
```

Out[102]:

```
count    614.000000
mean       0.237785
std        0.534737
min        0.000000
25%        0.000000
50%        0.000000
75%        0.000000
max        2.000000
Name: Self_Employed, dtype: float64
```

In [103]:

```
print(df['Self_Employed'].quantile(0.10))
print(df['Self_Employed'].quantile(0.90))
```

```
0.0
1.0
```

In [104]:

```
df['Self_Employed'] = np.where(df['Self_Employed'] <0, 0,df['Self_Employed'])
df['Self_Employed'] = np.where(df['Self_Employed'] >1, 1,df['Self_Employed'])
```

In [105]:

```
df["Self_Employed"].plot(kind="box")
```

Out[105]:

```
<AxesSubplot: >
```

ApplicantIncome

In [106]:

```
df["ApplicantIncome"].describe()
```

Out[106]:

```
count      614.000000
mean      5403.459283
std       6109.041673
min        150.000000
25%       2877.500000
50%       3812.500000
75%       5795.000000
max      81000.000000
Name: ApplicantIncome, dtype: float64
```

In [107]:

```
print(df['ApplicantIncome'].quantile(0.10))
print(df['ApplicantIncome'].quantile(0.90))
```

```
2216.1
9459.900000000007
```

In [108]:

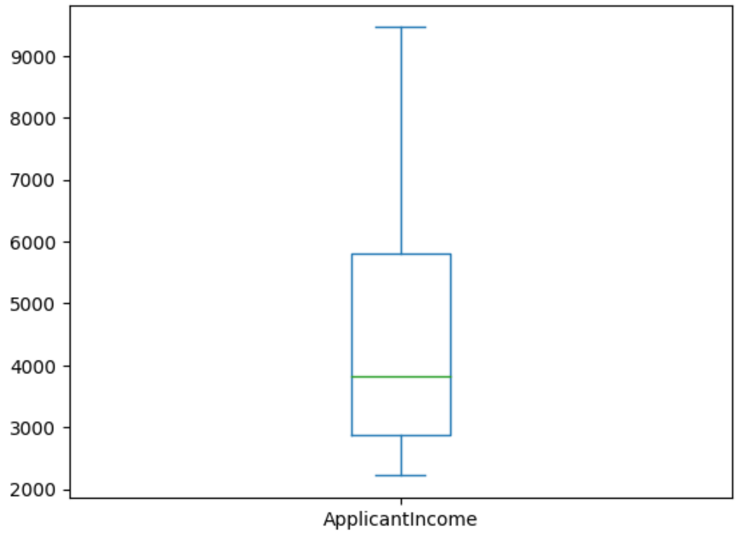
```
df['ApplicantIncome'] = np.where(df['ApplicantIncome'] <2216.1, 2216.1,df['ApplicantIncome'])
df['ApplicantIncome'] = np.where(df['ApplicantIncome'] >9459.900000000007, 9459.900000000007,df['ApplicantIncome'])
```

In [109]:

```
df["ApplicantIncome"].plot(kind="box")
```

Out[109]:

```
<AxesSubplot: >
```



CoapplicantIncome

In [111]:

```
df["CoapplicantIncome"].describe()
```

Out[111]:

```
count      614.000000
mean      1621.245798
std       2926.248369
min          0.000000
25%          0.000000
50%       1188.500000
75%       2297.250000
max      41667.000000
Name: CoapplicantIncome, dtype: float64
```

In [112]:

```
print(df['CoapplicantIncome'].quantile(0.10))
print(df['CoapplicantIncome'].quantile(0.90))
```

```
0.0
3782.200000000002
```

In [113]:

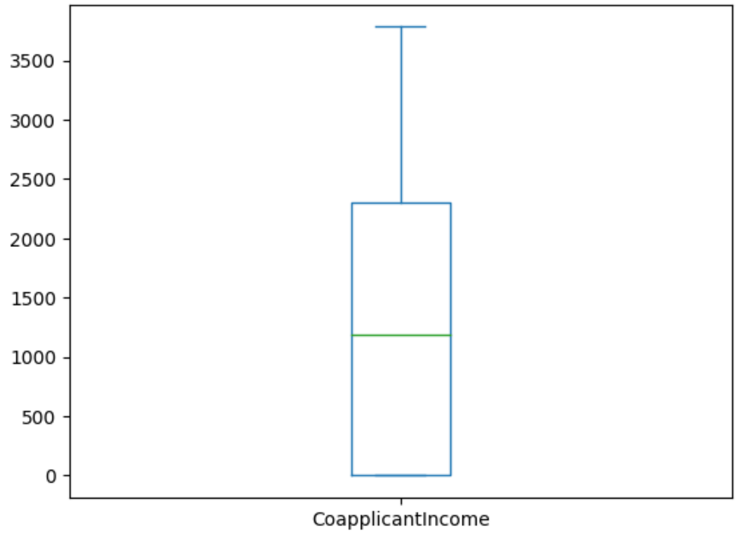
```
df['CoapplicantIncome'] = np.where(df['CoapplicantIncome'] <0.0, 0.0,df['CoapplicantIncome'])
df['CoapplicantIncome'] = np.where(df['CoapplicantIncome'] >3782.200000000002, 3782.200000000002,df['CoapplicantIncome'])
```

In [114]:

```
df['CoapplicantIncome'].plot(kind="box")
```

Out[114]:

```
<AxesSubplot: >
```



LoanAmount

In [115]:

```
df["LoanAmount"].describe()
```

Out[115]:

```
count    614.000000
mean     141.166124
std       88.340630
min        0.000000
25%       98.000000
50%      125.000000
75%      164.750000
max      700.000000
Name: LoanAmount, dtype: float64
```

In [116]:

```
print(df['LoanAmount'].quantile(0.10))
print(df['LoanAmount'].quantile(0.90))
```

```
63.60000000000001
229.4000000000001
```

In [117]:

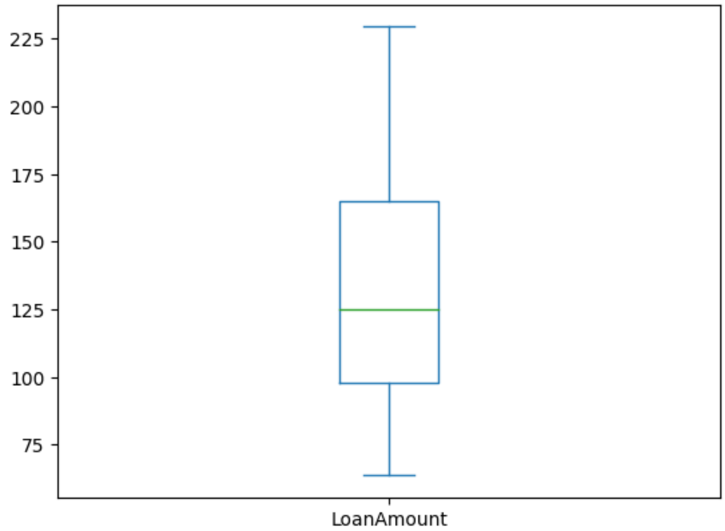
```
df['LoanAmount'] = np.where(df['LoanAmount'] <63.60000000000001, 63.60000000000001,df['LoanAmount'])
df['LoanAmount'] = np.where(df['LoanAmount'] >229.4000000000001, 229.4000000000001,df['LoanAmount'])
```

In [118]:

```
df['LoanAmount'].plot(kind="box")
```

Out[118]:

```
<AxesSubplot: >
```



Loan_Amount_Term

In [119]:

```
df["Loan_Amount_Term"].describe()
```

Out[119]:

```
count    614.000000
mean     334.201954
std       82.183884
min        0.000000
25%      360.000000
50%      360.000000
75%      360.000000
max      480.000000
Name: Loan_Amount_Term, dtype: float64
```

In [120]:

```
print(df['Loan_Amount_Term'].quantile(0.10))
print(df['Loan_Amount_Term'].quantile(0.90))
```

```
180.0
360.0
```

In [121]:

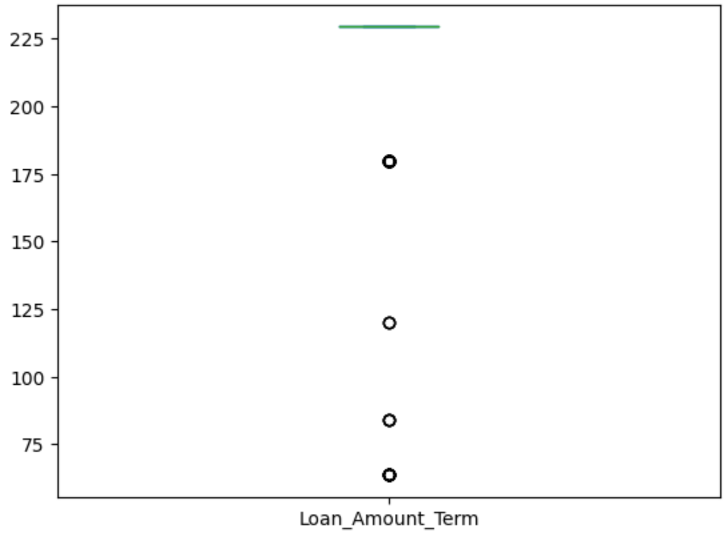
```
df['Loan_Amount_Term'] = np.where(df['Loan_Amount_Term'] <63.60000000000001, 63.60000000000001,df['Loan_Amount_Term'])
df['Loan_Amount_Term'] = np.where(df['Loan_Amount_Term'] >229.4000000000001, 229.4000000000001,df['Loan_Amount_Term'])
```

In [122]:

```
df['Loan_Amount_Term'].plot(kind="box")
```

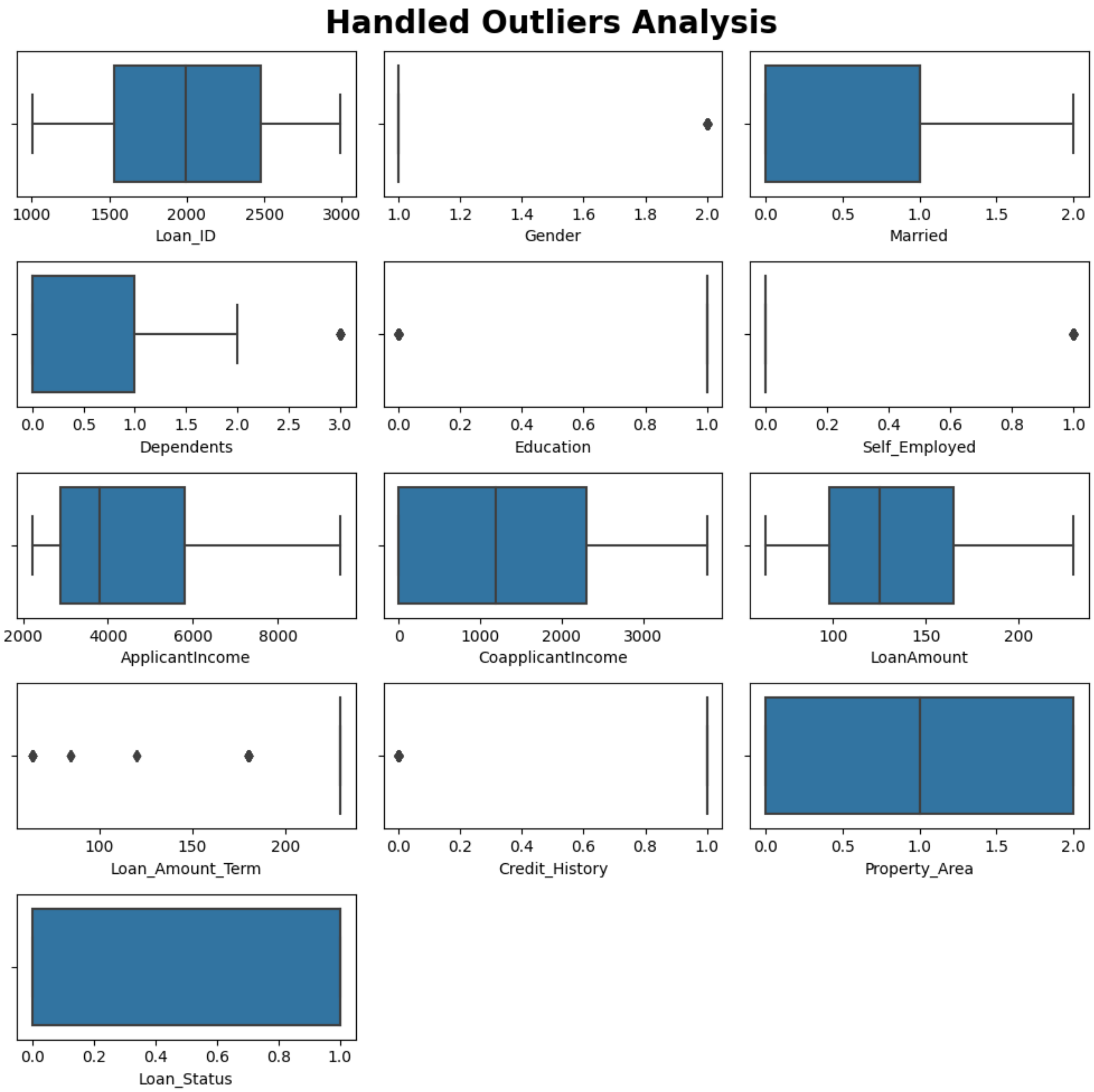
Out[122]:

```
<AxesSubplot: >
```



In [124]:

```
plt.figure(figsize=(10,10))
plt.suptitle(" Handled Outliers Analysis",fontsize=20 , fontweight="bold")
for i in range(0,len(df.columns)):
            plt.subplot(5,3,i+1)
            sns.boxplot(x=df[df.columns[i]])
            plt.xlabel(df.columns[i])
            plt.tight_layout()
```



Saving clean data

In [125]:

```
df.to_csv("cleaned_loan_prediction_data.csv")
```

In [126]:

```
df_new = pd.DataFrame()
```

In [129]:

```
df_new = df
df_new.head(2)
```

Out[129]:

	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amount_Term	Credit_Histo
0	1002	1	0	0.0	1	0	5849.0	0.0	63.6	229.4	
1	1003	1	1	1.0	1	0	4583.0	1508.0	128.0	229.4	

In [130]:

```
df_new.shape
```

Out[130]:

```
(614, 13)
```

Feature Selection

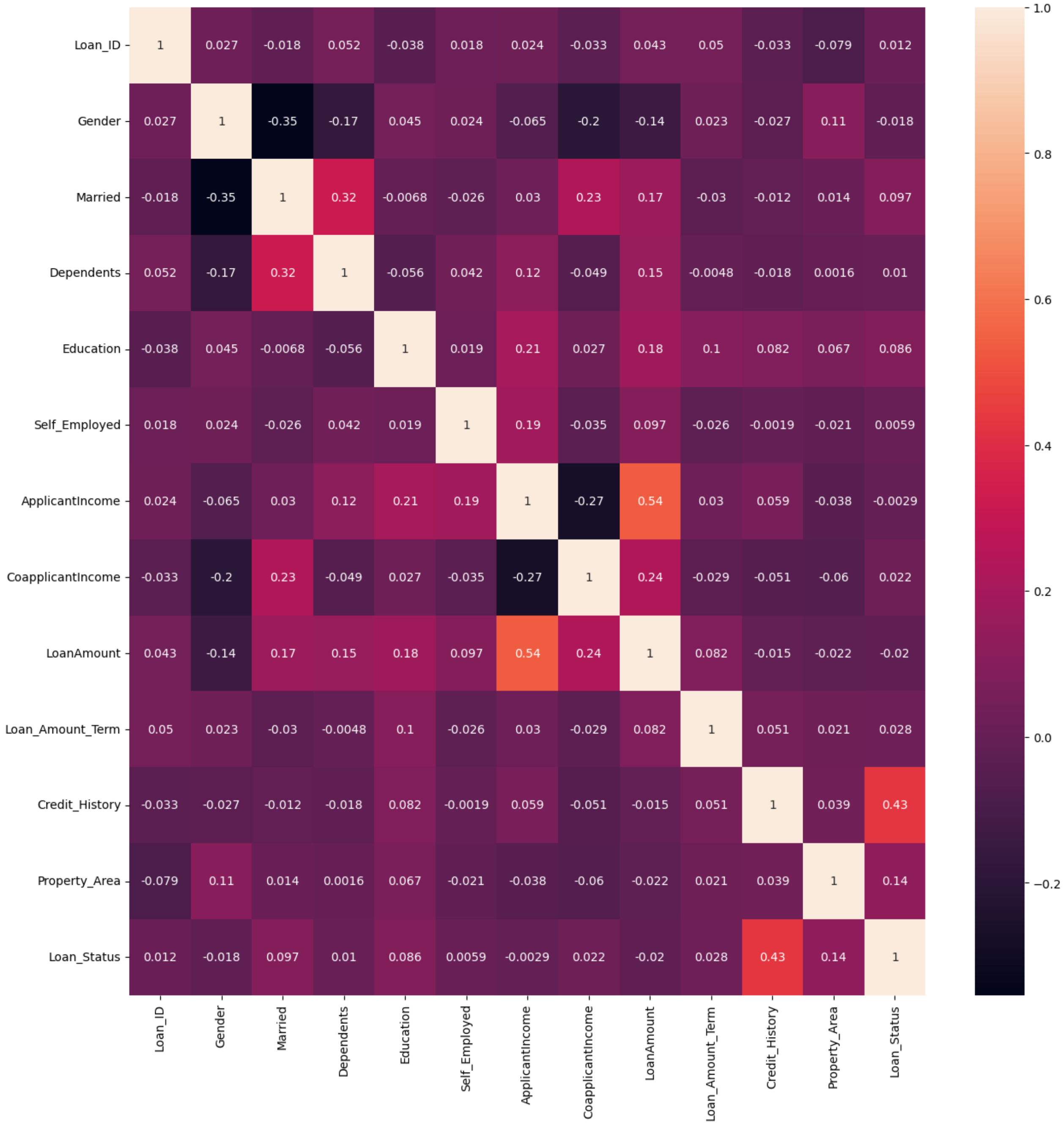
1.pearson's correlation

In [143]:

```
plt.figure(figsize=(15,15))
sns.heatmap(df_new.corr(),annot=True)
```

Out[143]:

<AxesSubplot: >



	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amount_Term	Credit_History	Property_Area	Loan_Status
Loan_ID	1	0.027	-0.018	0.052	-0.038	0.018	0.024	-0.033	0.043	0.05	-0.033	-0.079	0.012
Gender	0.027	1	-0.35	-0.17	0.045	0.024	-0.065	-0.2	-0.14	0.023	-0.027	0.11	-0.018
Married	-0.018	-0.35	1	0.32	-0.0068	-0.026	0.03	0.23	0.17	-0.03	-0.012	0.014	0.097
Dependents	0.052	-0.17	0.32	1	-0.056	0.042	0.12	-0.049	0.15	-0.0048	-0.018	0.0016	0.01
Education	-0.038	0.045	-0.0068	-0.056	1	0.019	0.21	0.027	0.18	0.1	0.082	0.067	0.086
Self_Employed	0.018	0.024	-0.026	0.042	0.019	1	0.19	-0.035	0.097	-0.026	-0.0019	-0.021	0.0059
ApplicantIncome	0.024	-0.065	0.03	0.12	0.21	0.19	1	-0.27	0.54	0.03	0.059	-0.038	-0.0029
CoapplicantIncome	-0.033	-0.2	0.23	-0.049	0.027	-0.035	-0.27	1	0.24	-0.029	-0.051	-0.06	0.022
LoanAmount	0.043	-0.14	0.17	0.15	0.18	0.097	0.54	0.24	1	0.082	-0.015	-0.022	-0.02
Loan_Amount_Term	0.05	0.023	-0.03	-0.0048	0.1	-0.026	0.03	-0.029	0.082	1	0.051	0.021	0.028
Credit_History	-0.033	-0.027	-0.012	-0.018	0.082	-0.0019	0.059	-0.051	-0.015	0.051	1	0.039	0.43
Property_Area	-0.079	0.11	0.014	0.0016	0.067	-0.021	-0.038	-0.06	-0.022	0.021	0.039	1	0.14
Loan_Status	0.012	-0.018	0.097	0.01	0.086	0.0059	-0.0029	0.022	-0.02	0.028	0.43	0.14	1

2. fisher's score

In [144]:

```
from skfeature.function.similarity_based import fisher_score
```

In [145]:

```
x = df_new.drop("Loan_Status",axis=1)
y = df_new["Loan_Status"]
```

In [147]:

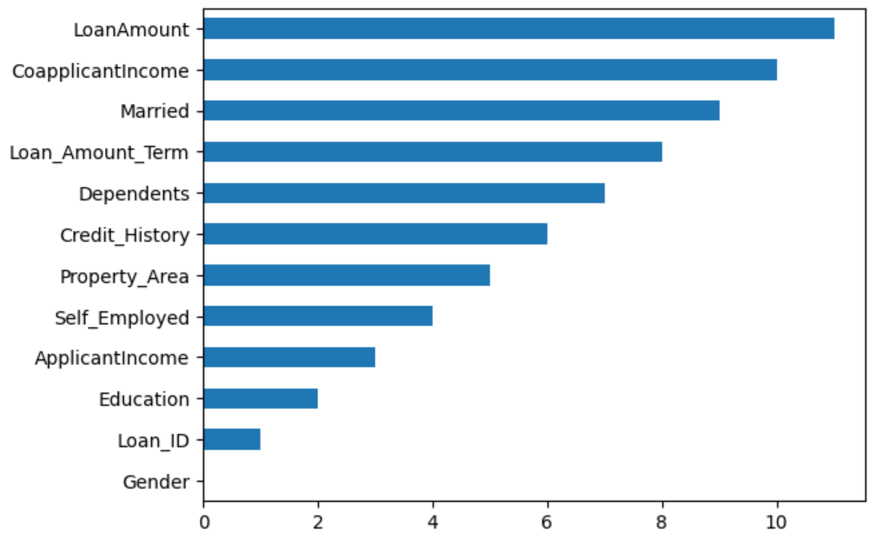
```
fisher_score = fisher_score.fisher_score(x.to_numpy(),y)
```

In [149]:

```
s1 = pd.Series(fisher_score,index=x.columns)
s1.sort_values().plot(kind="barh")
```

Out[149]:

```
<AxesSubplot: >
```



observation:

Acc to Fisher score feature selection technique, Gender column is not much important.

3. Variance Threshold Method

In [150]:

```
from sklearn.feature_selection import VarianceThreshold
```

In [151]:

```
var_th = VarianceThreshold(threshold = 0.3)
var_th.fit_transform(df_new)
var_th.get_support()
```

Out[151]:

```
array([ True, False, False,  True, False, False,  True,  True,  True,
        True, False,  True, False])
```

In [152]:

```
arr = var_th.get_support()
np.where(arr == False)
```

Out[152]:

```
(array([ 1,  2,  4,  5, 10, 12], dtype=int64),)
```

In [153]:

```
df_new.columns[np.where(arr == False)]
```

Out[153]:

```
Index(['Gender', 'Married', 'Education', 'Self_Employed', 'Credit_History',
       'Loan_Status'],
      dtype='object')
```

Observation:

'Gender', 'Married', 'Education', 'Self_Employed', 'Credit_History','Loan_Status' columns has less variance as compared to other columns.

key observation from feature selection:

I have applied correlation , fisher score, variance threshold method techniques to find the best ahe lest imporatant feature. so, in all the techniques "Gender" column is common with less importance. So we can drop that column.

Preprocessing

In [158]:

```
x = df_new.drop(["Loan_Status","Gender"],axis=1)
y = df_new["Loan_Status"]
```

In [159]:

```
x.head(2)
```

Out[159]:

	Loan_ID	Married	Dependents	Education	Self_Employed	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amount_Term	Credit_History	Prop
0	1002	0	0.0	1	0	5849.0	0.0	63.6	229.4	1.0	
1	1003	1	1.0	1	0	4583.0	1508.0	128.0	229.4	1.0	

In [160]:

```
y
```

Out[160]:

```
0      1
1      0
2      1
3      1
4      1
      ..
609    1
610    1
611    1
612    1
613    0
Name: Loan_Status, Length: 614, dtype: int64
```

train_tst_split

In [161]:

```
x_train,x_test,y_train,y_test = train_test_split(x,y,test_size=0.2,random_state=42)
```

In [162]:

```
x_train.shape , x_test.shape
```

Out[162]:

```
((491, 11), (123, 11))
```

In [163]:

```
y_train.shape , y_test.shape
```

Out[163]:

```
((491,), (123,))
```

Logistic Regression

In [164]:

```
lg_model = LogisticRegression()
lg_model.fit(x_train,y_train)
```

Out[164]:

```
▾ LogisticRegression
LogisticRegression()
```

Testing Evaluation

In [165]:

```
y_pred = lg_model.predict(x_test)
```

In [170]:

```
acc_score = accuracy_score(y_test,y_pred)
print("Accuracy Score: ",acc_score)

con_matrix = confusion_matrix(y_test,y_pred)
print("Confusion_Matrix:\n "  ,con_matrix)

clf_report = classification_report(y_test,y_pred)
print("Classification_report: \n" , clf_report )
```

```
Accuracy Score:  0.7479674796747967
Confusion_Matrix:
  [[18 25]
 [ 6 74]]
Classification_report: 
               precision    recall  f1-score   support

           0       0.75      0.42      0.54        43
           1       0.75      0.93      0.83        80

    accuracy                           0.75       123
   macro avg       0.75      0.67      0.68       123
weighted avg       0.75      0.75      0.73       123
```

training Evaluation

In [171]:

```
y_pred_train = lg_model.predict(x_train)
```

In [172]:

```
acc_score = accuracy_score(y_train,y_pred_train)
print("Accuracy Score: ",acc_score)

con_matrix = confusion_matrix(y_train,y_pred_train)
print("Confusion_Matrix:\n "  ,con_matrix)

clf_report = classification_report(y_train,y_pred_train)
print("Classification_report: \n" , clf_report )
```

```
Accuracy Score:  0.7678207739307535
Confusion_Matrix:
  [[ 68  81]
 [ 33 309]]
Classification_report: 
               precision    recall  f1-score   support

           0       0.67      0.46      0.54       149
           1       0.79      0.90      0.84       342

    accuracy                           0.77       491
   macro avg       0.73      0.68      0.69       491
weighted avg       0.76      0.77      0.75       491
```