```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
warnings.filterwarnings('ignore')
```

```
df = pd.read_csv('/content/Training Dataset.csv')
df.head()
```

	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amount_Term	Credit_History	Property_Area	Loan_Status
0	LP001002	Male	No	0	Graduate	No	5849	0.0	NaN	360.0	1.0	Urban	Y
1	LP001003	Male	Yes	1	Graduate	No	4583	1508.0	128.0	360.0	1.0	Rural	N
2	LP001005	Male	Yes	0	Graduate	Yes	3000	0.0	66.0	360.0	1.0	Urban	Y
3	LP001006	Male	Yes	0	Not Graduate	No	2583	2358.0	120.0	360.0	1.0	Urban	Y
4	LP001008	Male	No	0	Graduate	No	6000	0.0	141.0	360.0	1.0	Urban	Y

Next steps: Generate code with df | View recommended plots

```
df.shape
```

```
(614, 13)
```

```
df.columns
```

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

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 614 entries, 0 to 613
Data columns (total 13 columns):
 #   Column             Non-Null Count  Dtype  
---  ------             --------------  -----  
 0   Loan_ID            614 non-null    object 
 1   Gender             601 non-null    object 
 2   Married            611 non-null    object 
 3   Dependents         599 non-null    object 
 4   Education          614 non-null    object 
 5   Self_Employed      582 non-null    object 
 6   ApplicantIncome    614 non-null    int64  
 7   CoapplicantIncome  614 non-null    float64
 8   LoanAmount         592 non-null    float64
 9   Loan_Amount_Term   600 non-null    float64
 10  Credit_History     564 non-null    float64
 11  Property_Area      614 non-null    object 
 12  Loan_Status        614 non-null    object 
dtypes: float64(4), int64(1), object(8)
memory usage: 62.5+ KB
```

```
df.isnull().sum()
```

```
Loan_ID               0
Gender               13
Married               3
Dependents           15
Education             0
Self_Employed        32
ApplicantIncome       0
CoapplicantIncome     0
LoanAmount           22
Loan_Amount_Term     14
Credit_History       50
Property_Area         0
Loan_Status           0
dtype: int64
```

```
df.duplicated().sum() #No Duplicates
```

```
0
```

```
df.drop('Loan_ID',axis=1,inplace=True) #Unwanted Column
```

```
df['Gender'].mode()
```

```
0    Male
Name: Gender, dtype: object
```

```
df['Married'].mode()
```

```
0    Yes
Name: Married, dtype: object
```

```
df['Dependents'].mode()
```

```
0    0
Name: Dependents, dtype: object
```

```
df['Self_Employed'].mode()
```

```
0    No
Name: Self_Employed, dtype: object
```

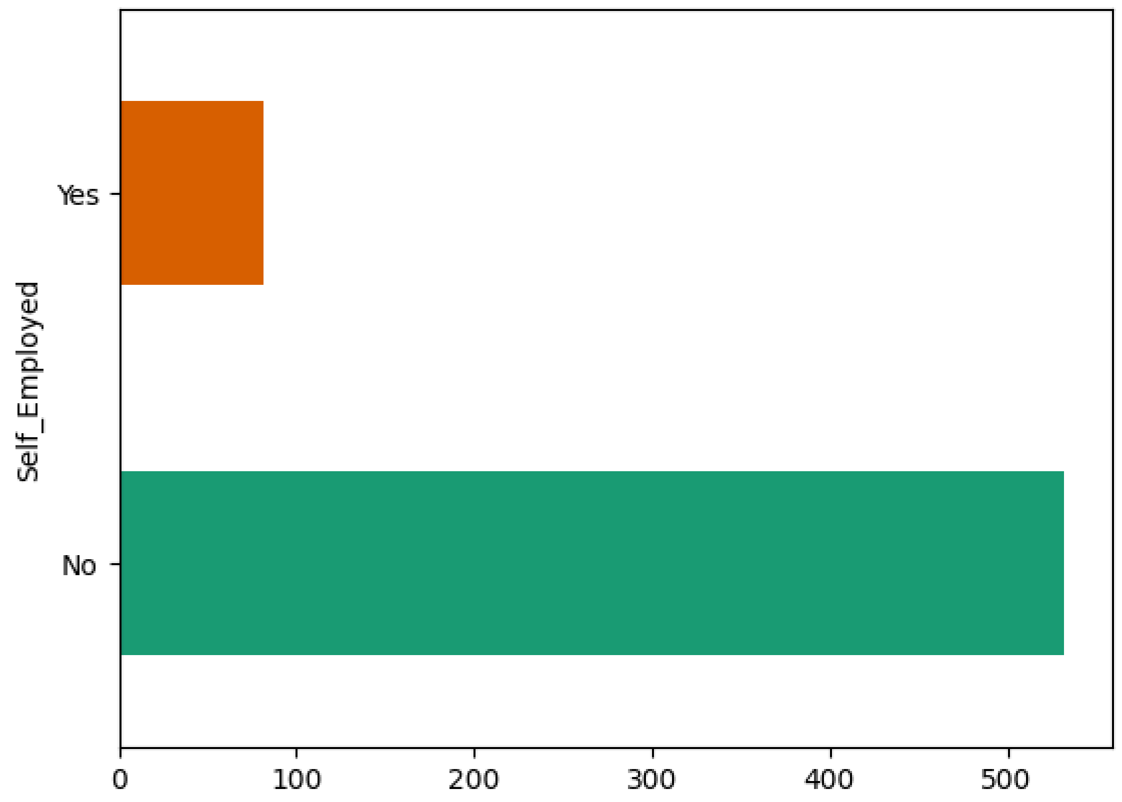
```
df['Gender']= df['Gender'].fillna('Male')
df['Married']= df['Married'].fillna('Yes')
df['Dependents']= df['Dependents'].fillna('0')
df['Self_Employed']= df['Self_Employed'].fillna('No')
df['LoanAmount']= df['LoanAmount'].fillna(df['LoanAmount'].mean())
df['Loan_Amount_Term']= df['Loan_Amount_Term'].fillna(df['Loan_Amount_Term'].mean())
df['Credit_History']= df['Credit_History'].fillna(df['Credit_History'].mean())
```

```
df['Dependents'] = df['Dependents'].replace('3+','3')
```

```
df['Dependents'] = df['Dependents'].astype(int)
```

```
df.groupby('Self_Employed').size().plot(kind='barh', color=sns.palettes.mpl_palette('Dark2'))
```

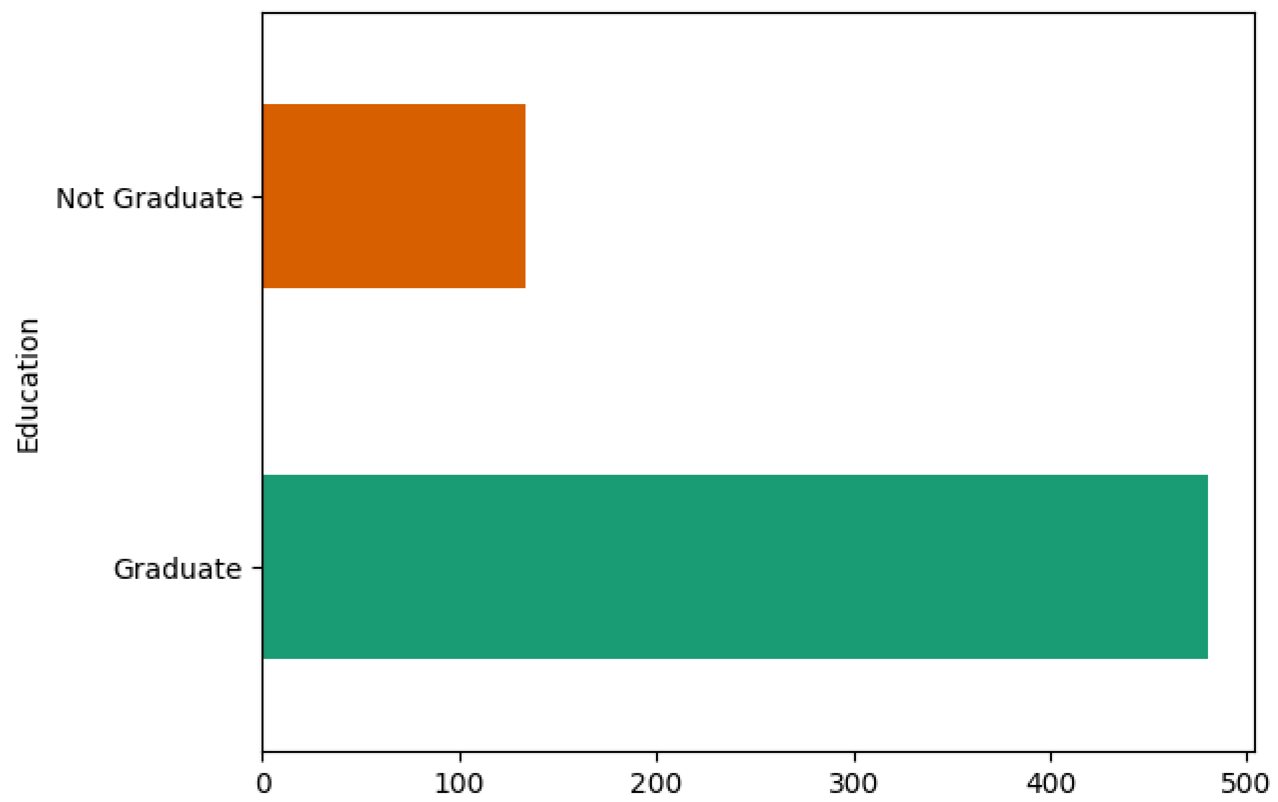
```
<Axes: ylabel='Self_Employed'>
```



Observation: In this dataset, most of them are not self_employed.

```
df.groupby('Education').size().plot(kind='barh', color=sns.palettes.mpl_palette('Dark2'))
```

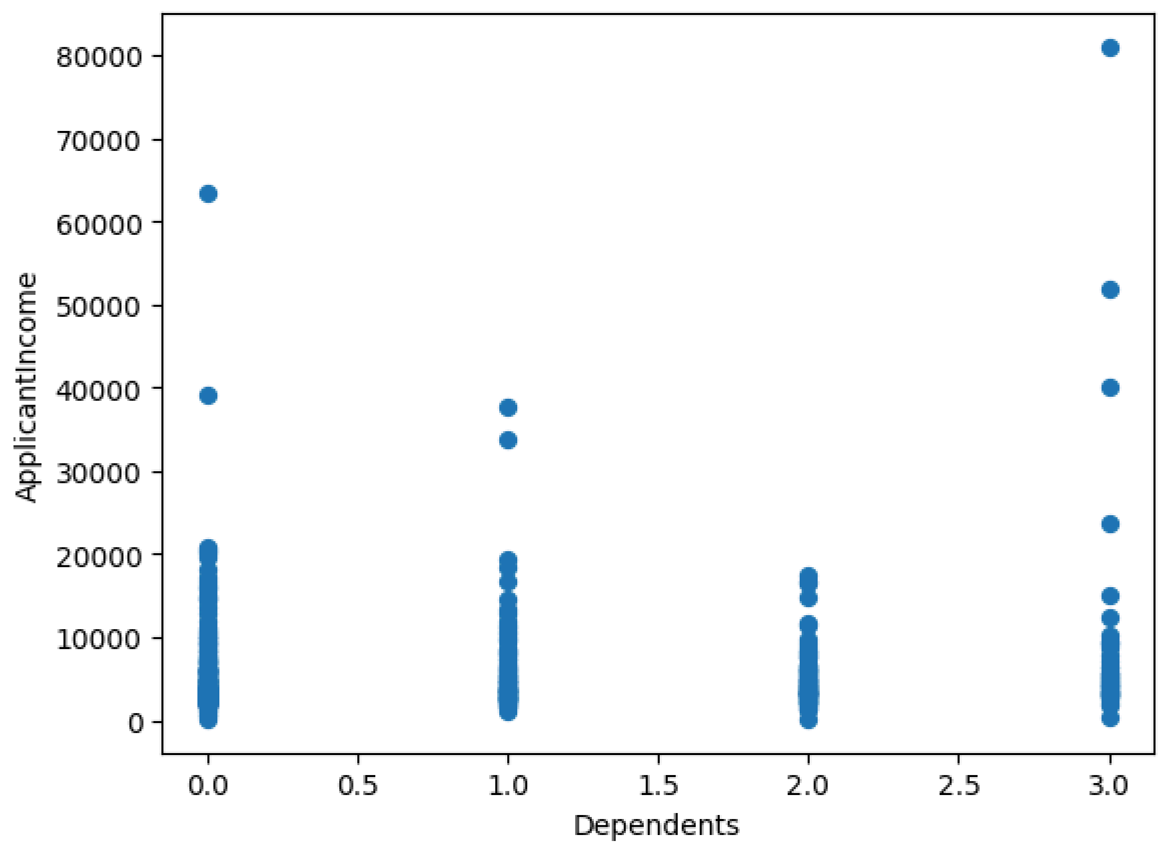
```
<Axes: ylabel='Education'>
```



Observation: In this dataset, most of them are graduates.

```
df.plot(kind='scatter', x='Dependents', y='ApplicantIncome', s=32)
```

```
<Axes: xlabel='Dependents', ylabel='ApplicantIncome'>
```



Observation: The more than 3+ Dependents has highest Income..

```
df['Gender'] = np.where(df['Gender']=='Male',1,0)
df['Married'] = np.where(df['Married']=='Yes',1,0)
df['Self_Employed'] = np.where(df['Self_Employed']=='Yes',1,0)
df['Education'] = np.where(df['Education']=='Graduate',1,0)
df['Loan_Status'] = np.where(df['Loan_Status']=='Y',1,0)
```

```
df.tail()
```

	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amount_Term	Credit_History	Property_Area	Loan_Status
609	0	0	0	1	0	2900	0.0	71.0	360.0	1.0	Rural	1
610	1	1	3	1	0	4106	0.0	40.0	180.0	1.0	Rural	1
611	1	1	1	1	0	8072	240.0	253.0	360.0	1.0	Urban	1
612	1	1	2	1	0	7583	0.0	187.0	360.0	1.0	Urban	1
613	0	0	0	1	1	4583	0.0	133.0	360.0	0.0	Semiurban	0

```
df['Property_Area'].unique()
```

```
array(['Urban', 'Rural', 'Semiurban'], dtype=object)
```

```
from sklearn.preprocessing import OrdinalEncoder
order = ['Rural','Semiurban','Urban'] #Lowest to highest
encoder = OrdinalEncoder(categories=[order])
df['Property_Area']=encoder.fit_transform(df[['Property_Area']])
```

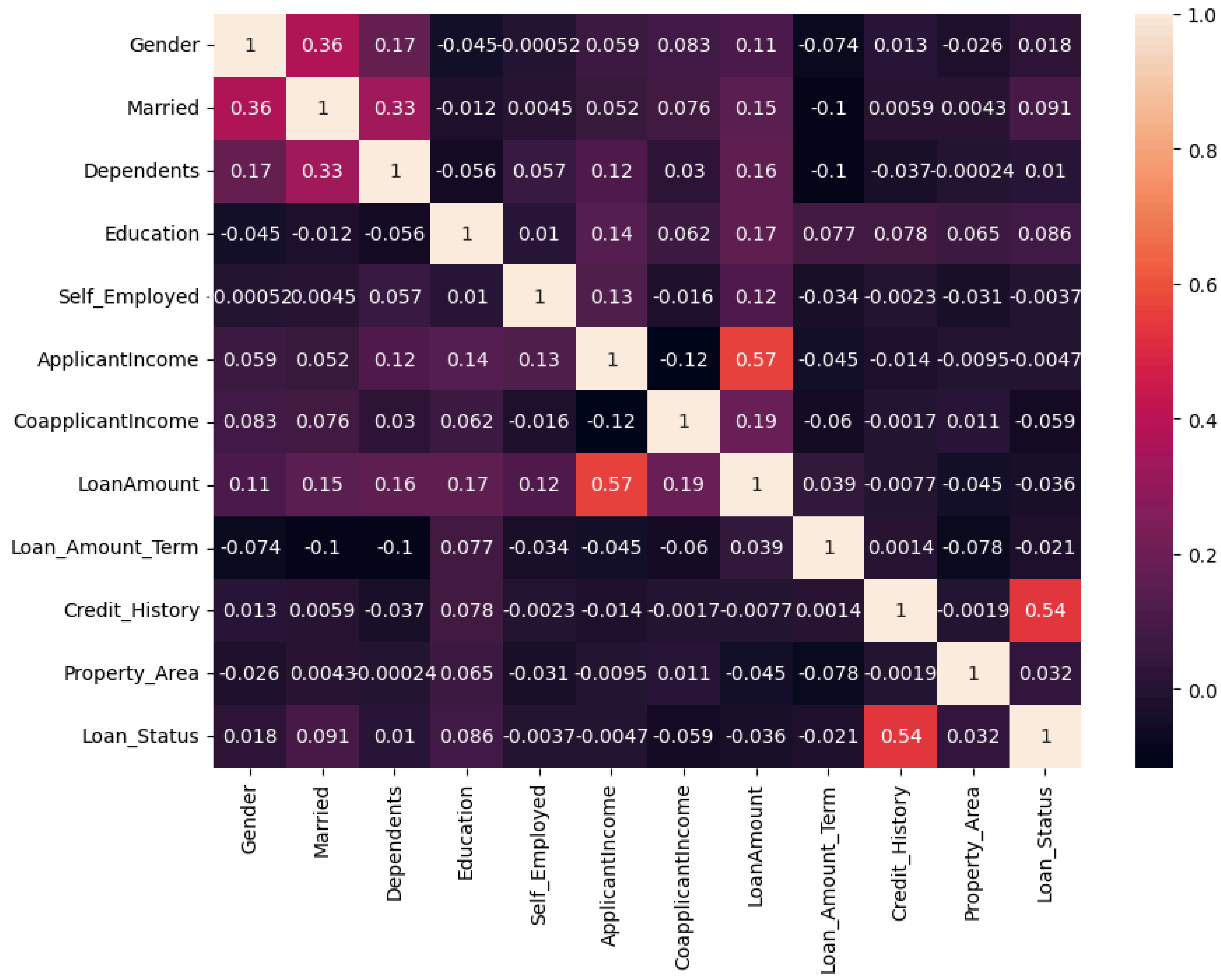
```
details = ['ApplicantIncome','CoapplicantIncome','LoanAmount','Loan_Amount_Term']
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()
df[details] = scaler.fit_transform(df[details])
```

```
df.describe()
```

	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amount_Term	Credit_History	Property_Area	Loan_Status
count	614.000000	614.000000	614.000000	614.000000	614.000000	614.000000	614.000000	614.000000	614.000000	614.000000	614.000000	614.000000
mean	0.817590	0.653094	0.744300	0.781759	0.133550	0.064978	0.038910	0.198860	0.705128	0.842199	1.037459	0.687296
std	0.386497	0.476373	1.009623	0.413389	0.340446	0.075560	0.070229	0.121617	0.137548	0.349681	0.787482	0.463973
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
25%	1.000000	0.000000	0.000000	1.000000	0.000000	0.033735	0.000000	0.132055	0.743590	1.000000	0.000000	0.000000
50%	1.000000	1.000000	0.000000	1.000000	0.000000	0.045300	0.028524	0.173661	0.743590	1.000000	1.000000	1.000000
75%	1.000000	1.000000	1.000000	1.000000	0.000000	0.069821	0.055134	0.225398	0.743590	1.000000	2.000000	1.000000
max	1.000000	1.000000	3.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	2.000000	1.000000

```
plt.figure(figsize=(10,7))
sns.heatmap(df.corr(),annot=True)
```

<Axes: >




```
x=df.iloc[:,:-1] #Independent
y=df.iloc[:,-1] #Dependent
```

```
x
```

	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amount_Term	Credit_History	Property_Area
0	1	0	0	1	0	0.070489	0.000000	0.198860	0.743590	1.0	2.0
1	1	1	1	1	0	0.054830	0.036192	0.172214	0.743590	1.0	0.0
2	1	1	0	1	1	0.035250	0.000000	0.082489	0.743590	1.0	2.0
3	1	1	0	0	0	0.030093	0.056592	0.160637	0.743590	1.0	2.0
4	1	0	0	1	0	0.072356	0.000000	0.191027	0.743590	1.0	2.0
...	...	...	...	...	...	...	...	...	...	...	...
609	0	0	0	1	0	0.034014	0.000000	0.089725	0.743590	1.0	0.0
610	1	1	3	1	0	0.048930	0.000000	0.044863	0.358974	1.0	0.0
611	1	1	1	1	0	0.097984	0.005760	0.353111	0.743590	1.0	2.0
612	1	1	2	1	0	0.091936	0.000000	0.257598	0.743590	1.0	2.0
613	0	0	0	1	1	0.054830	0.000000	0.179450	0.743590	0.0	1.0

614 rows × 11 columns

Next steps: Generate code with x View recommended plots

```
y
```

```
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 Test Split
from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test = train_test_split(x,y,test_size=0.33,random_state=42)
print(x_train.shape)
print(x_test.shape)
print(y_train.shape)
print(y_test.shape)
```

```
(411, 11)
(203, 11)
(411,)
(203,)
```

```
from sklearn.linear_model import LogisticRegression
lr = LogisticRegression()
lr.fit(x_train,y_train)
```

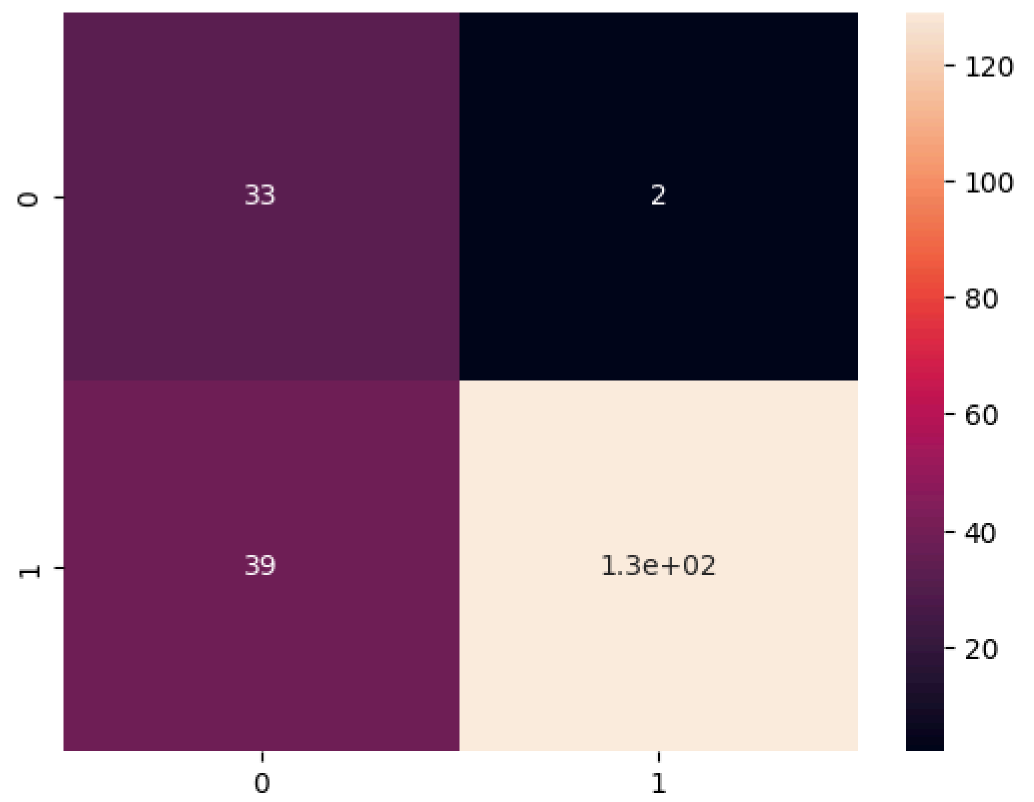
```
▾ LogisticRegression
LogisticRegression()
```

```
from sklearn.metrics import confusion_matrix,accuracy_score,classification_report
y_pred = lr.predict(x_test)
print(accuracy_score(y_pred,y_test))
print(sns.heatmap(confusion_matrix(y_pred,y_test),annot=True))
print(classification_report(y_pred,y_test))
```

```
0.7980295566502463
Axes(0.125,0.11;0.62x0.77)
              precision    recall  f1-score   support

           0       0.46      0.94      0.62        35
           1       0.98      0.77      0.86       168

    accuracy                           0.80       203
   macro avg       0.72      0.86      0.74       203
weighted avg       0.89      0.80      0.82       203
```




```
parameter = {
    'penalty':['l1', 'l2', 'elasticnet'],
    'solver' : ['lbfgs', 'liblinear', 'newton-cg', 'newton-cholesky', 'sag', 'saga']
}
```

```
import warnings
warnings.filterwarnings('ignore')
from sklearn.model_selection import GridSearchCV
lr1 = GridSearchCV(lr,param_grid = parameter,cv=5,scoring='accuracy')
lr1.fit(x_train,y_train)
```

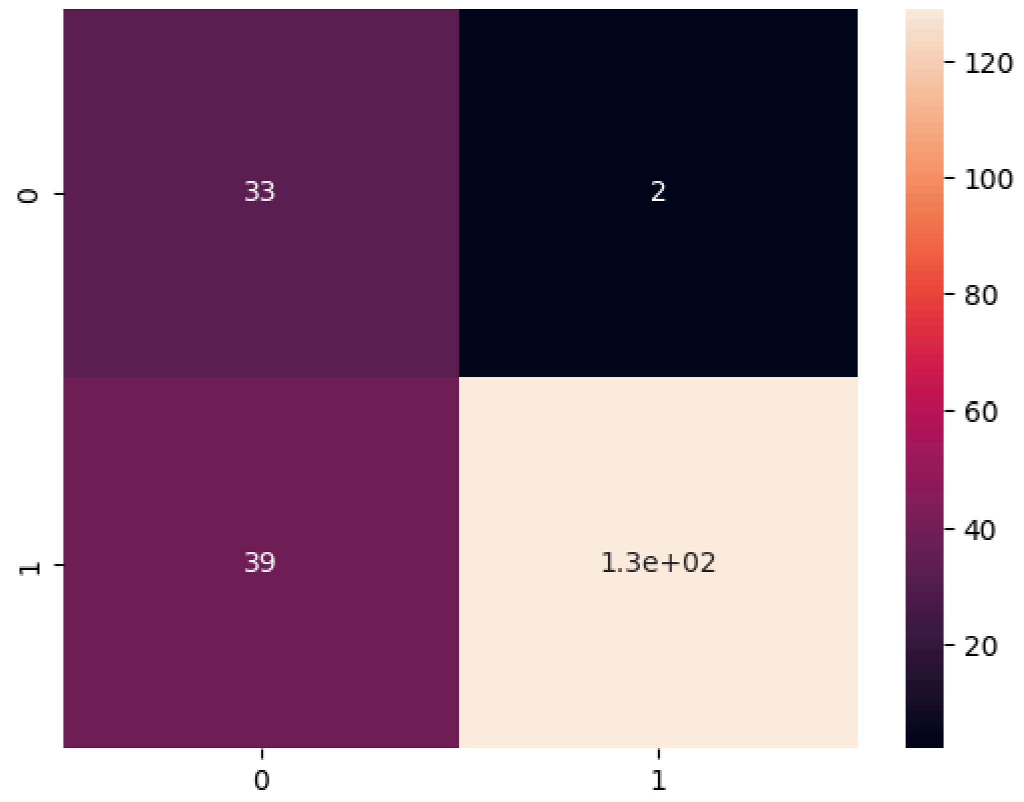
```
GridSearchCV
estimator: LogisticRegression
LogisticRegression
```

```
y_pred = lr1.predict(x_test)
print(accuracy_score(y_pred,y_test))
print(sns.heatmap(confusion_matrix(y_pred,y_test),annot=True))
print(classification_report(y_pred,y_test))
```

```
0.7980295566502463
Axes(0.125,0.11;0.62x0.77)
              precision    recall  f1-score   support

           0       0.46      0.94      0.62        35
           1       0.98      0.77      0.86       168

    accuracy                           0.80       203
   macro avg       0.72      0.86      0.74       203
weighted avg       0.89      0.80      0.82       203
```




```
from sklearn.ensemble import RandomForestClassifier
rfc = RandomForestClassifier()
rfc.fit(x_train,y_train)
```

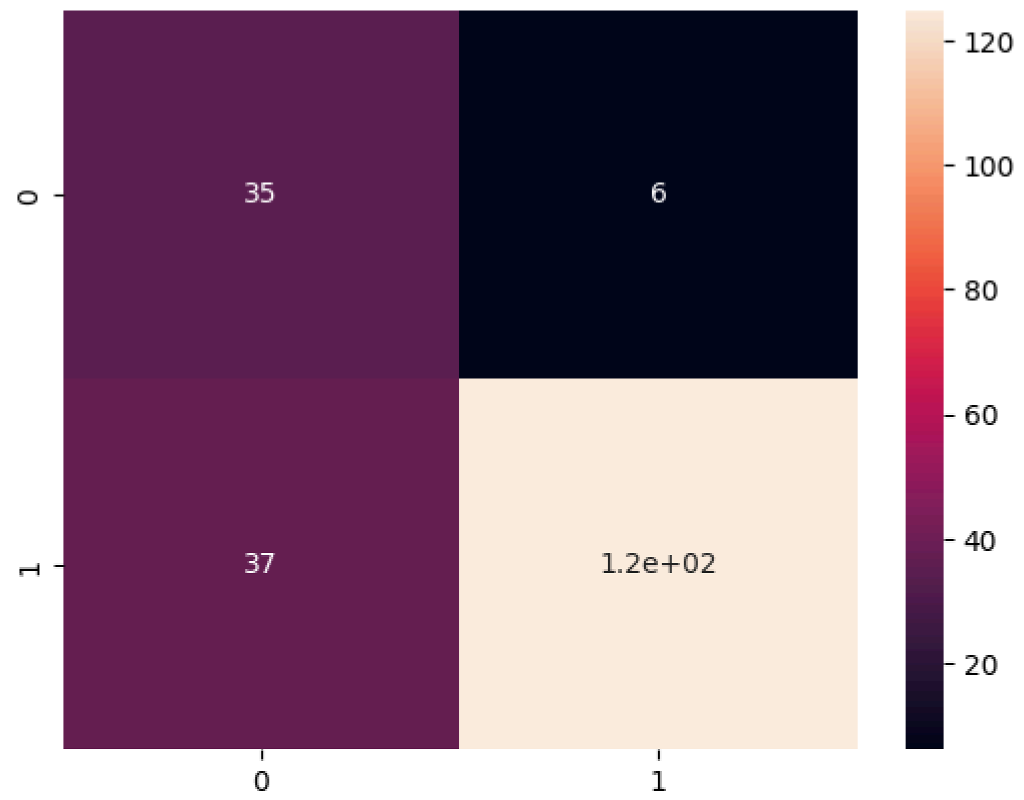
```
RandomForestClassifier
RandomForestClassifier()
```

```
y_pred = rfc.predict(x_test)
print(accuracy_score(y_pred,y_test))
print(sns.heatmap(confusion_matrix(y_pred,y_test),annot=True))
print(classification_report(y_pred,y_test))
```

```
0.7881773399014779
Axes(0.125,0.11;0.62x0.77)
              precision    recall  f1-score   support

           0       0.49      0.85      0.62        41
           1       0.95      0.77      0.85       162

    accuracy                           0.79       203
   macro avg       0.72      0.81      0.74       203
weighted avg       0.86      0.79      0.81       203
```




```
parameters = {
    'criterion' : ['gini', 'entropy', 'log_loss'],
    'max_features': ['sqrt', 'log2'],
    'max_depth' : [1,2,3,4,5,6,7,8,9,10]
}
```

```
rfc1 = GridSearchCV(rfc,param_grid=parameters,cv=5,scoring='accuracy')
rfc1.fit(x_train,y_train)
```

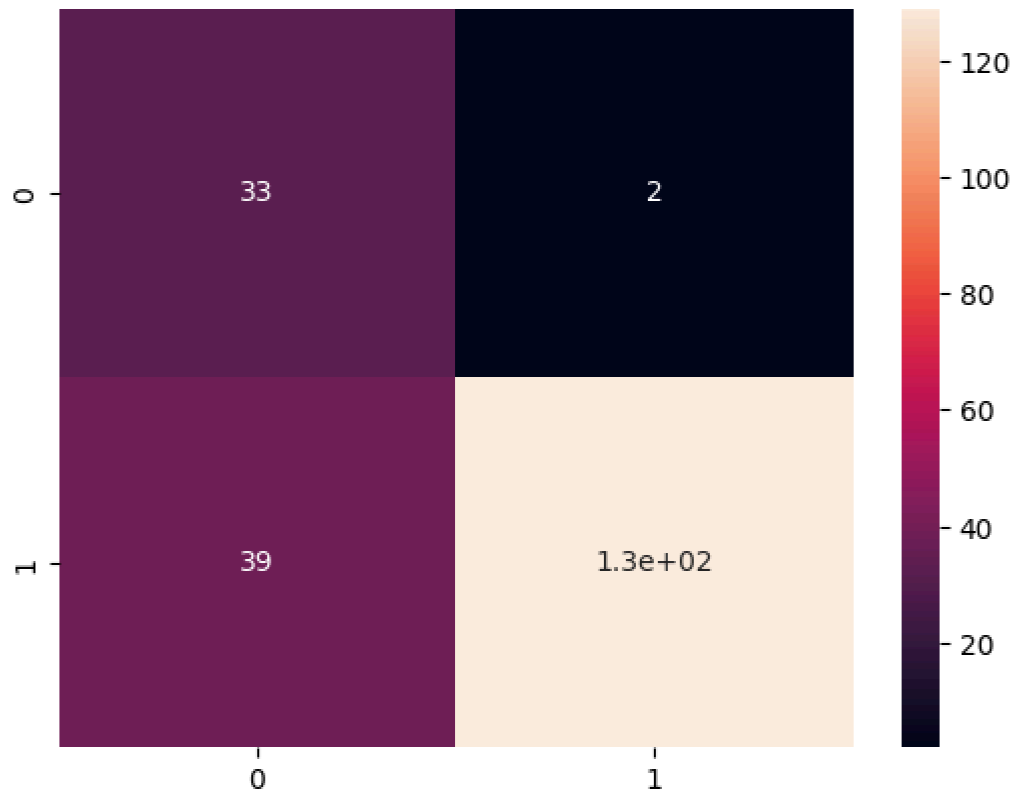
```
GridSearchCV
estimator: RandomForestClassifier
RandomForestClassifier
```

```
y_pred1 = rfc1.predict(x_test)
print(accuracy_score(y_pred1,y_test))
print(sns.heatmap(confusion_matrix(y_pred1,y_test),annot=True))
print(classification_report(y_pred1,y_test))
```

```
0.7980295566502463
Axes(0.125,0.11;0.62x0.77)
              precision    recall  f1-score   support

           0       0.46      0.94      0.62        35
           1       0.98      0.77      0.86       168

    accuracy                           0.80       203
   macro avg       0.72      0.86      0.74       203
weighted avg       0.89      0.80      0.82       203
```




```
import xgboost as xgb
param_grid = {
    'n_estimators': [100, 200, 300],
    'max_depth': [3, 4, 5],
    'learning_rate': [0.01, 0.1, 0.2],
    'subsample': [0.8, 1.0],
    'colsample_bytree': [0.8, 1.0]
}

xgb_clf = xgb.XGBClassifier(use_label_encoder=False, eval_metric='mlogloss')
grid_search = GridSearchCV(estimator=xgb_clf, param_grid=param_grid,
                           cv=3, n_jobs=-1, verbose=2)
grid_search.fit(x_train, y_train)
print(f"Best parameters found: {grid_search.best_params_}")
best_model = grid_search.best_estimator_
y_pred = best_model.predict(x_test)
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy * 100:.2f}%")
```

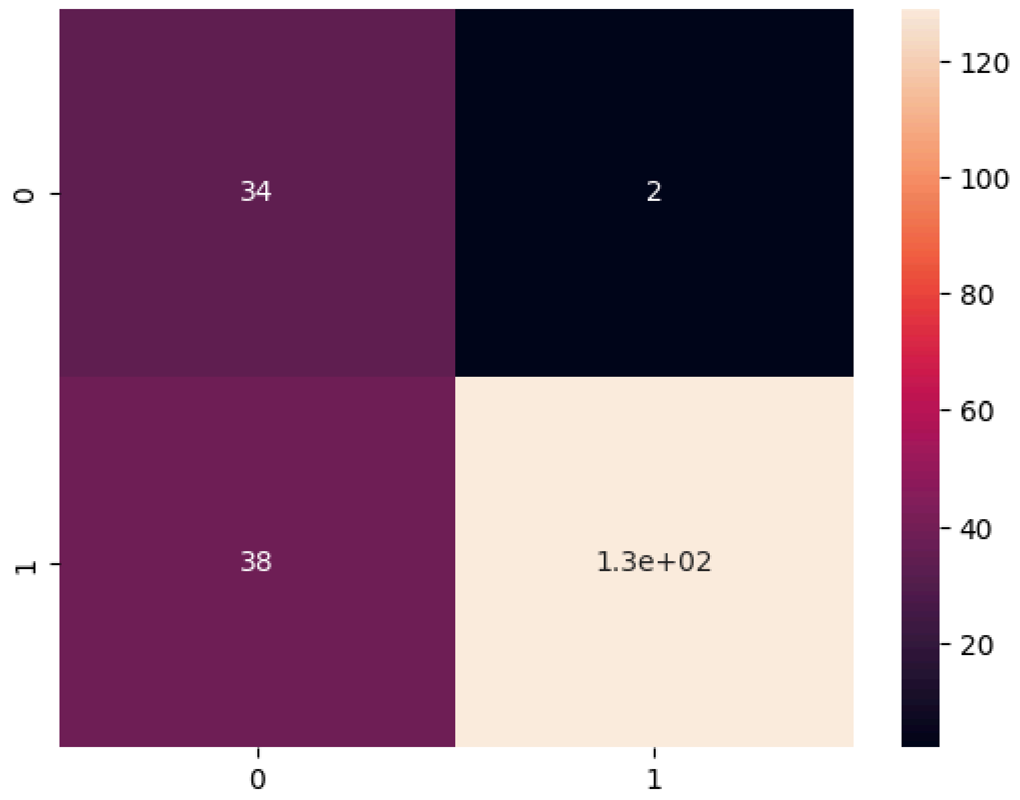
```
Fitting 3 folds for each of 108 candidates, totalling 324 fits
Best parameters found: {'colsample_bytree': 0.8, 'learning_rate': 0.01, 'max_depth': 5, 'n_estimators': 200, 'subsample': 0.8}
Accuracy: 80.30%
```

```
print(accuracy_score(y_pred,y_test))
print(sns.heatmap(confusion_matrix(y_pred,y_test),annot=True))
print(classification_report(y_pred,y_test))
```

```
0.8029556650246306
Axes(0.125,0.11;0.62x0.77)
              precision    recall  f1-score   support

           0       0.47      0.94      0.63        36
           1       0.98      0.77      0.87       167

    accuracy                           0.80       203
   macro avg       0.73      0.86      0.75       203
weighted avg       0.89      0.80      0.82       203
```



Conclusion:

- Logistic Regression : 79.8%
- Randomforest: 79.3%
- Xgboost : 80.3%

We achieved the highest accuracy with the XGBoost model, which is 80.3%. Therefore, we are using this model for our predictions.

```
Start coding or generate with AI.
```

Testing Part:

```python
new = pd.DataFrame([[0,0,0,1,0,2900,0.0,71.0,360.0,1.0,2.0]],columns=x.columns)
new
```

	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amount_Term	Credit_History	Property_Area
0	0	0	0	1	0	2900	0.0	71.0	360.0	1.0	2.0

```python
output = best_model.predict(new)
print("Loan Status : ",output) # Approved
```

```
Loan Status :  [1]
```