In [1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
from scipy import stats
import plotly.express as px
import seaborn as sns
from sklearn.preprocessing import StandardScaler,MinMaxScaler

#Data Processing fns
from sklearn.preprocessing import StandardScaler,MinMaxScaler
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
le=LabelEncoder()


#Classifiesrs
from sklearn.ensemble import AdaBoostClassifier,GradientBoostingClassifier,RandomFo
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from xgboost import XGBClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.svm import SVC
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import RepeatedStratifiedKFold
from sklearn.model_selection import GridSearchCV

#Models Evaluations tools
from sklearn.metrics import classification_report,accuracy_score,confusion_matrix,f
from sklearn.model_selection import cross_val_score
```

In [2]:
```
df=pd.read_csv("https://raw.githubusercontent.com/Mukund94/Datasets/main/loan_predi
```

In [3]:
```
df.head()
```

Out[3]:

	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	Coapplic
0	LP001002	Male	No	0	Graduate	No	5849	
1	LP001003	Male	Yes	1	Graduate	No	4583	
2	LP001005	Male	Yes	0	Graduate	Yes	3000	
3	LP001006	Male	Yes	0	Not Graduate	No	2583	
4	LP001008	Male	No	0	Graduate	No	6000	

In [4]:
```
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
```

In [5]:
```
df.isnull().sum()
```

Out[5]:
```
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
```

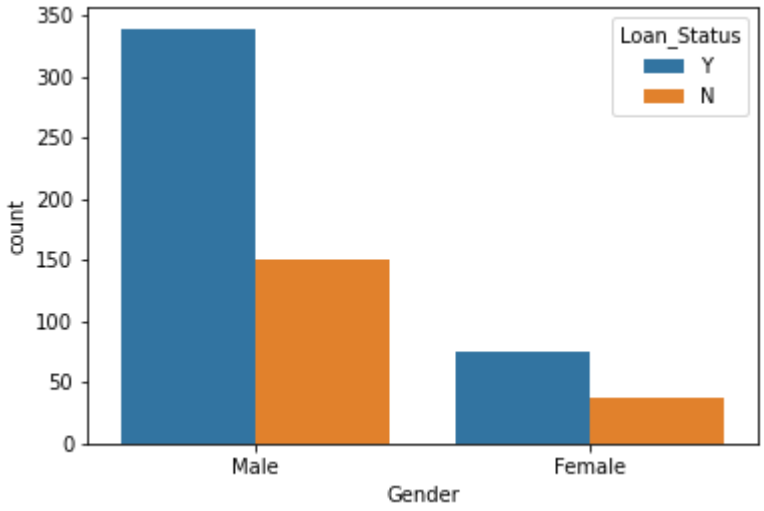
In [6]:
```
df.describe(include='all')
```

Out[6]:

	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	Coa
count	614	601	611	599	614	582	614.000000	
unique	614	2	2	4	2	2	NaN	
top	LP001002	Male	Yes	0	Graduate	No	NaN	
freq	1	489	398	345	480	500	NaN	
mean	NaN	NaN	NaN	NaN	NaN	NaN	5403.459283	
std	NaN	NaN	NaN	NaN	NaN	NaN	6109.041673	
min	NaN	NaN	NaN	NaN	NaN	NaN	150.000000	
25%	NaN	NaN	NaN	NaN	NaN	NaN	2877.500000	
50%	NaN	NaN	NaN	NaN	NaN	NaN	3812.500000	
75%	NaN	NaN	NaN	NaN	NaN	NaN	5795.000000	
max	NaN	NaN	NaN	NaN	NaN	NaN	81000.000000	

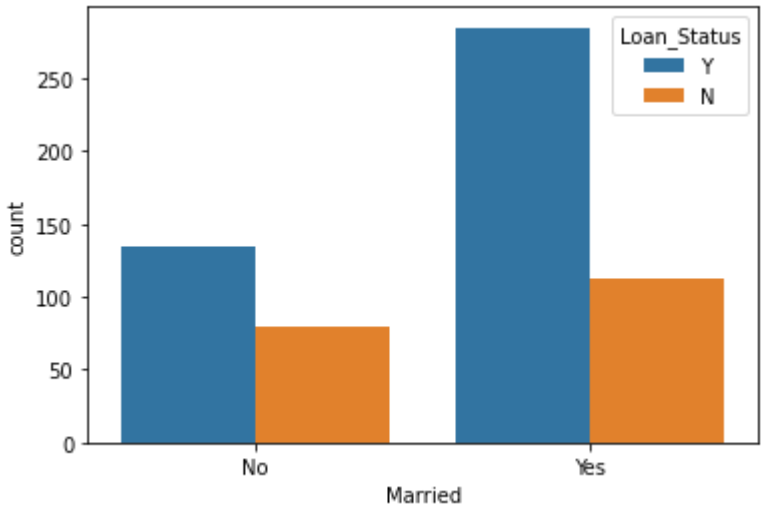
In [7]:
```
sns.countplot(x=df['Gender'],hue=df['Loan_Status'])
```

Out[7]:
```
<AxesSubplot:xlabel='Gender', ylabel='count'>
```



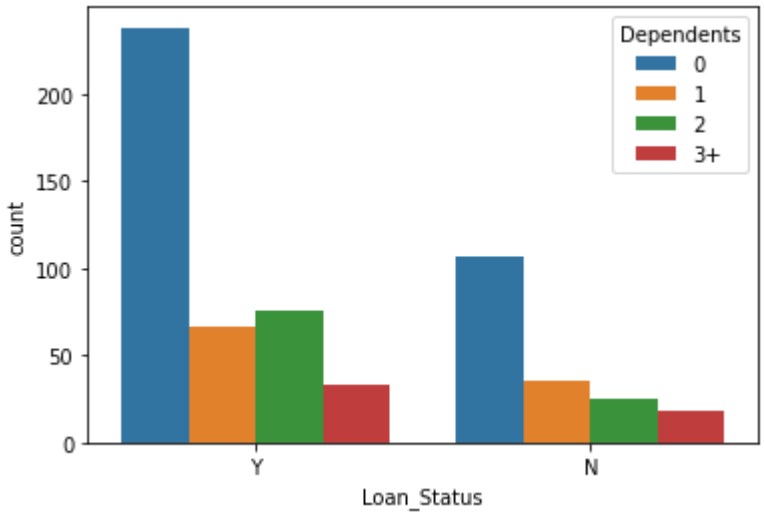
In [8]:
```
sns.countplot(x=df['Married'],hue=df['Loan_Status'])
```

Out[8]:
```
<AxesSubplot:xlabel='Married', ylabel='count'>
```



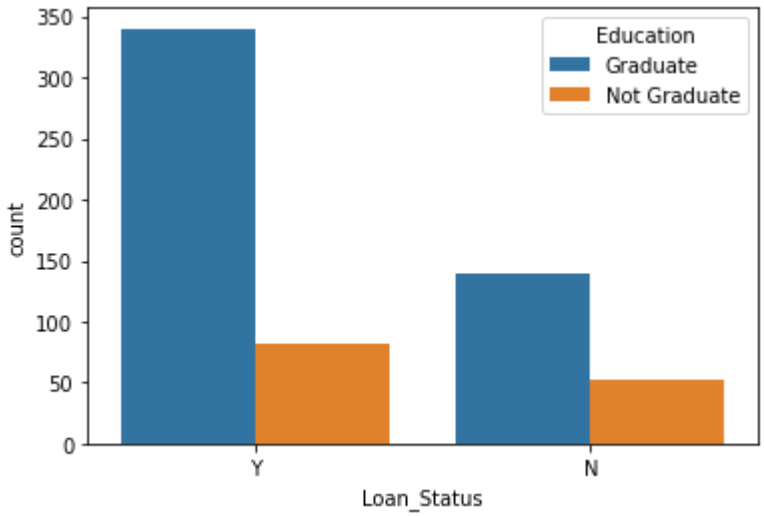
In [9]:
```
sns.countplot(hue=df['Dependents'],x=df['Loan_Status'])
```

Out[9]:
```
<AxesSubplot:xlabel='Loan_Status', ylabel='count'>
```



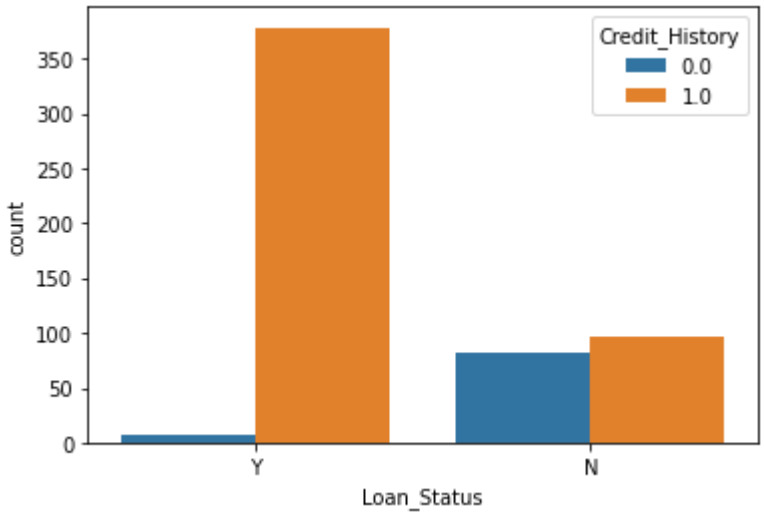
In [10]:
```
sns.countplot(hue=df['Education'],x=df['Loan_Status'])
```

Out[10]:
```
<AxesSubplot:xlabel='Loan_Status', ylabel='count'>
```

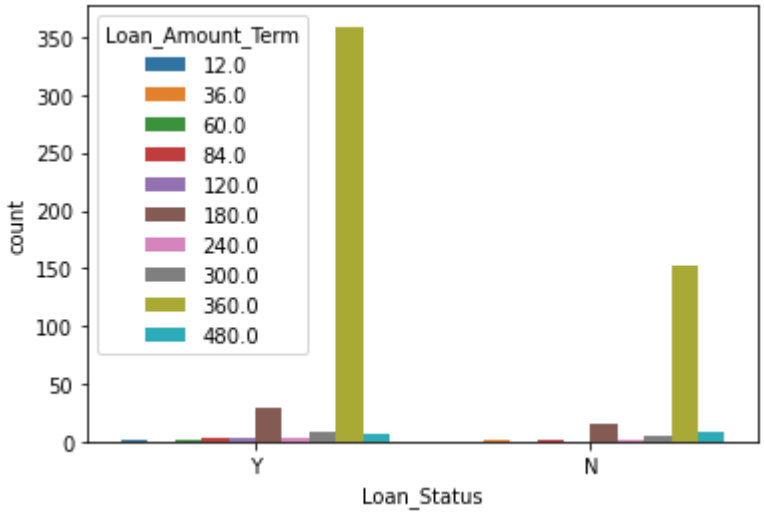


In [11]:
```
sns.countplot(hue=df['Credit_History'],x=df['Loan_Status'])
```

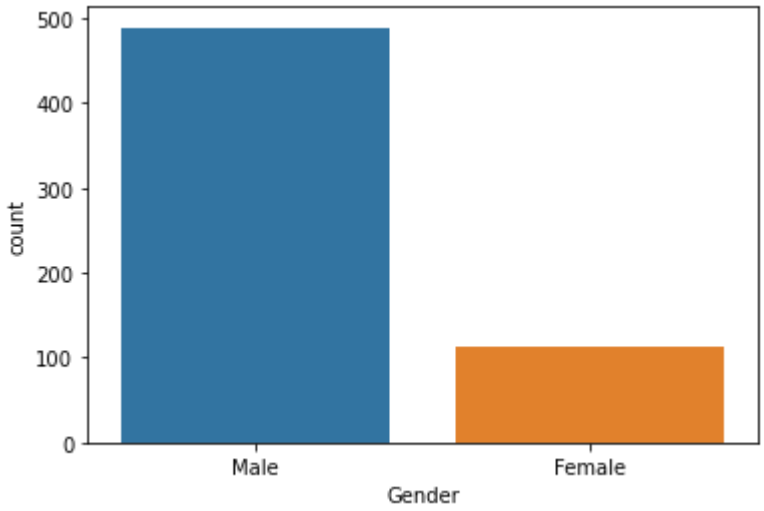
Out[11]:
```
<AxesSubplot:xlabel='Loan_Status', ylabel='count'>
```



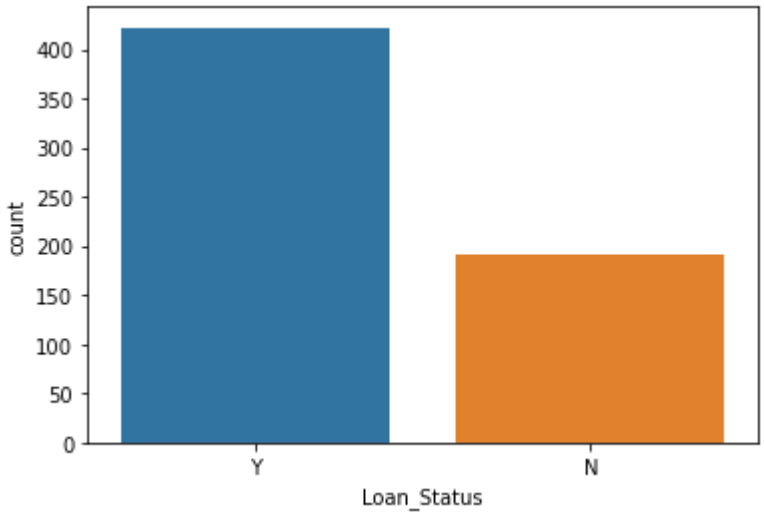
In [12]:
```
sns.countplot(hue=df['Loan_Amount_Term'],x=df['Loan_Status'])
plt.show()
```



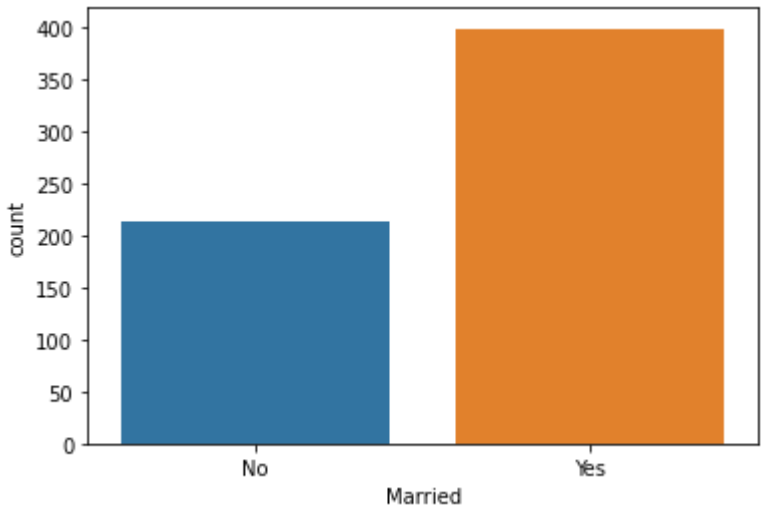
In [13]:
```
sns.countplot(x=df['Gender'])
plt.show()
```



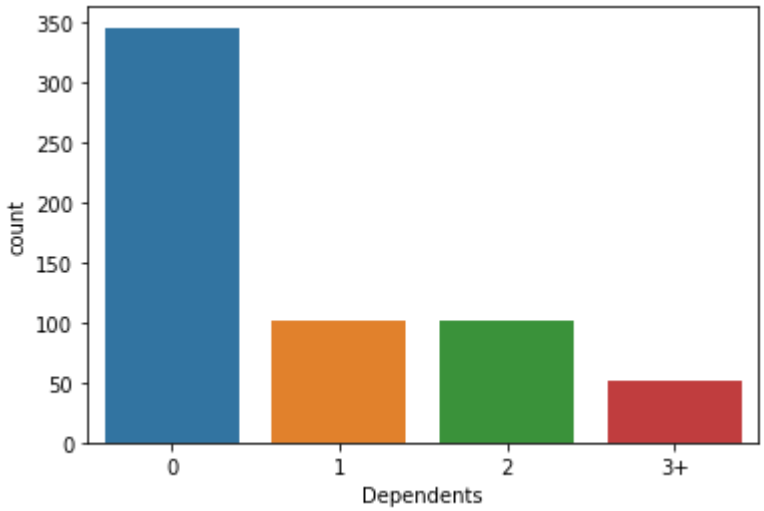
In [14]:
```
sns.countplot(x=df['Loan_Status'])
plt.show()
```



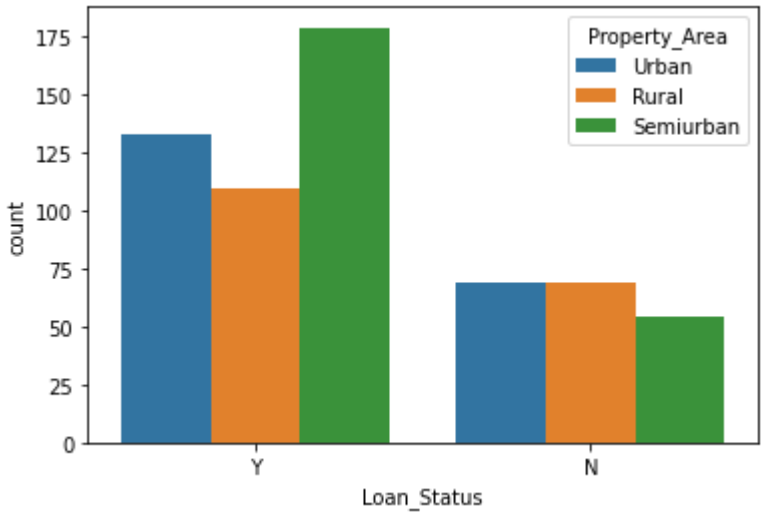
In [15]:
```
sns.countplot(x=df['Married'])
plt.show()
```



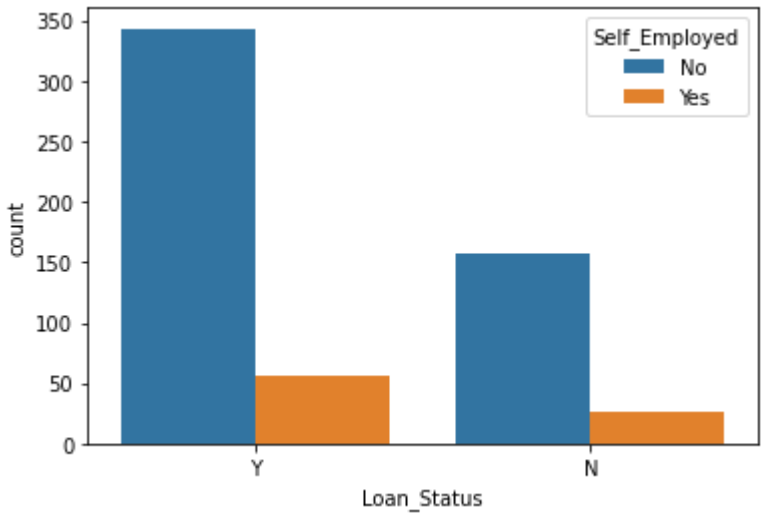
In [16]:
```
sns.countplot(x=df['Dependents'])
plt.show()
```



In [17]:
```
sns.countplot(hue=df['Property_Area'],x=df['Loan_Status'])
plt.show()
```

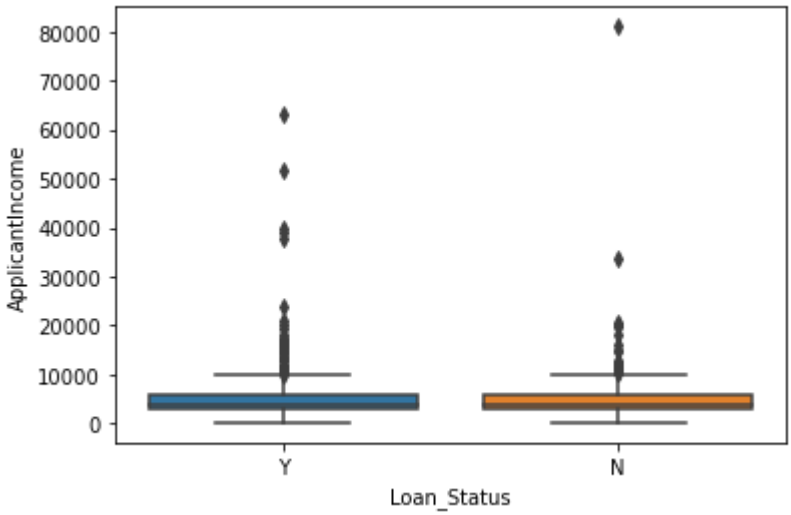


In [18]:
```
sns.countplot(hue=df['Self_Employed'],x=df['Loan_Status'])
plt.show()
```

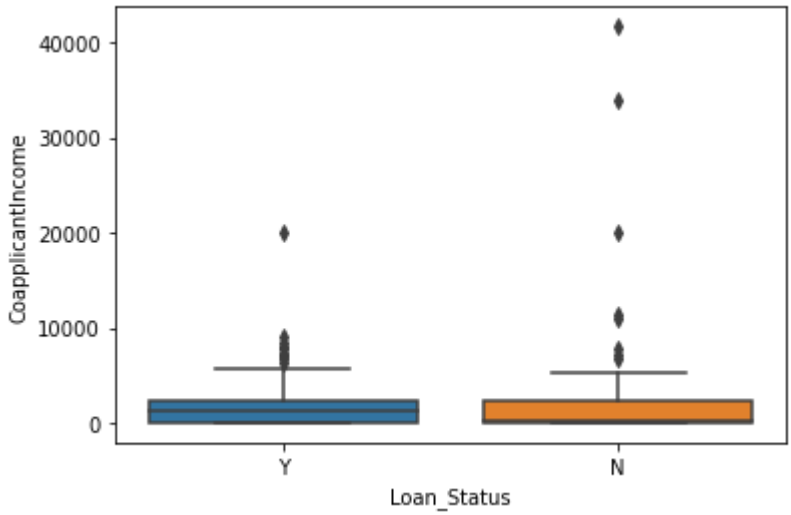


In [19]:
```
# Numeric and Categorical variables
```

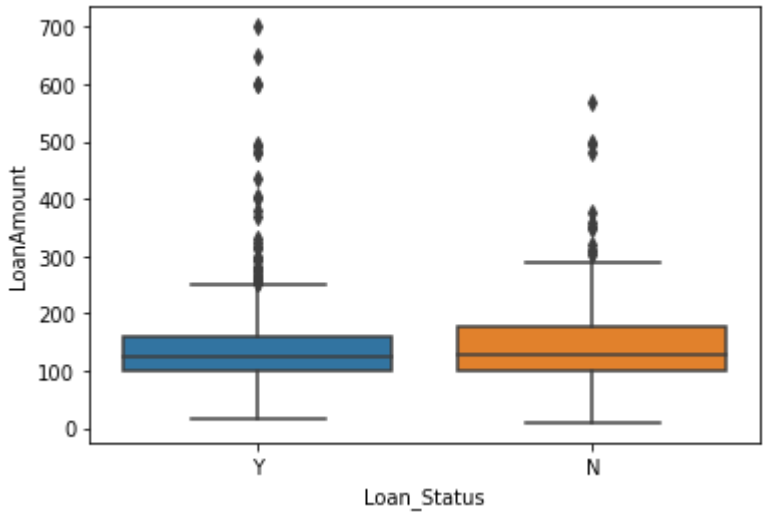
In [20]:
```
sns.boxplot(y=df['ApplicantIncome'],x=df['Loan_Status'],data=df)
plt.show()
```



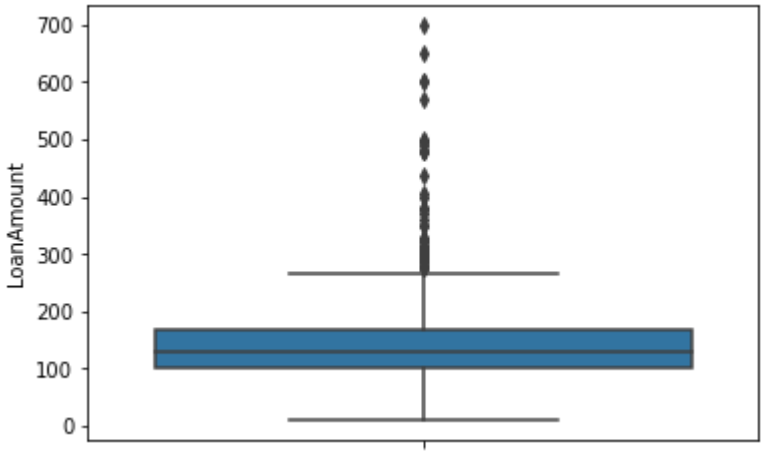
In [21]:
```
sns.boxplot(y=df['CoapplicantIncome'],x=df['Loan_Status'],data=df)
plt.show()
```



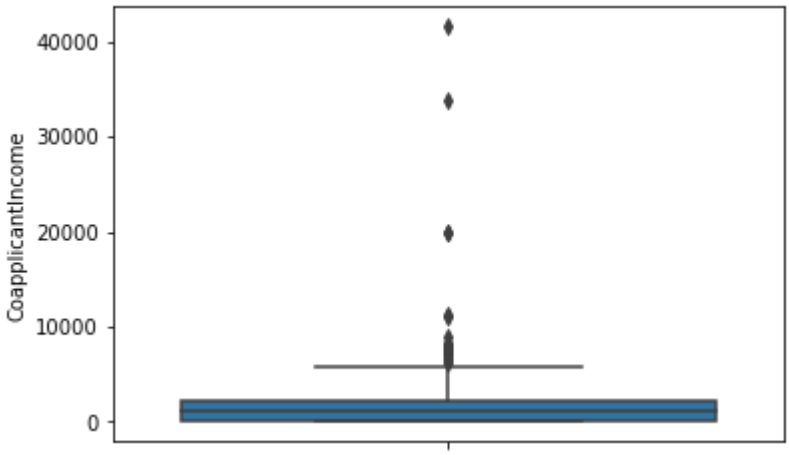
In [22]:
```
sns.boxplot(y=df['LoanAmount'],x=df['Loan_Status'],data=df)
plt.show()
```



In [23]:
```
sns.boxplot(y=df['LoanAmount'],data=df)
plt.show()
```



In [24]:
```
sns.boxplot(y=df['CoapplicantIncome'],data=df)
plt.show()
```



In [25]:
```
df.corr()
```

Out[25]:

	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amount_Term	Credit
ApplicantIncome	1.000000	-0.116605	0.570909	-0.045306	-
CoapplicantIncome	-0.116605	1.000000	0.188619	-0.059878	-
LoanAmount	0.570909	0.188619	1.000000	0.039447	-
Loan_Amount_Term	-0.045306	-0.059878	0.039447	1.000000	
Credit_History	-0.014715	-0.002056	-0.008433	0.001470	

In [26]:
```
#sns.pairplot(data=df,hue='Loan_Status')
```

In [27]:
```
df.skew()
```

```
C:\Users\DELL\AppData\Local\Temp\ipykernel_2280\1665899112.py:1: FutureWarning: Dr
opping of nuisance columns in DataFrame reductions (with 'numeric_only=None') is d
eprecated; in a future version this will raise TypeError.  Select only valid colum
ns before calling the reduction.
  df.skew()
```

Out[27]:
```
ApplicantIncome      6.539513
CoapplicantIncome    7.491531
LoanAmount           2.677552
Loan_Amount_Term    -2.362414
Credit_History      -1.882361
dtype: float64
```

The above shows that data is skewed

In [28]:
```
print(np.mean(df['ApplicantIncome']))
print(np.median(df['ApplicantIncome']))
print(stats.mode(df['ApplicantIncome'])[0])
```

```
5403.459283387622
3812.5
[2500]
```

Mean, Median and Mode are not equl -> so not normally distributed

In [29]:
```
print(np.mean(df['CoapplicantIncome']))
print(np.median(df['CoapplicantIncome']))
print(stats.mode(df['CoapplicantIncome'])[0])
```

```
1621.245798027101
1188.5
[0.]
```

In [30]:
```
print(np.mean(df['LoanAmount']))
print(np.median(df['LoanAmount']))
print(stats.mode(df['LoanAmount'])[0])
```

```
146.41216216216216
nan
[nan]
```

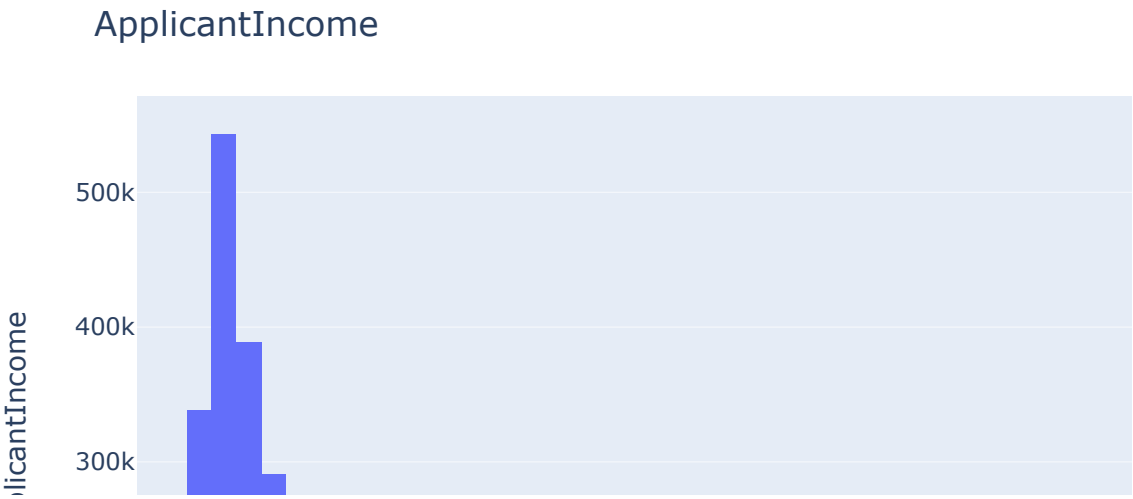
In [31]:
```
print(np.std(df['ApplicantIncome']))
print(np.std(df['CoapplicantIncome']))
print(np.std(df['LoanAmount']))
```

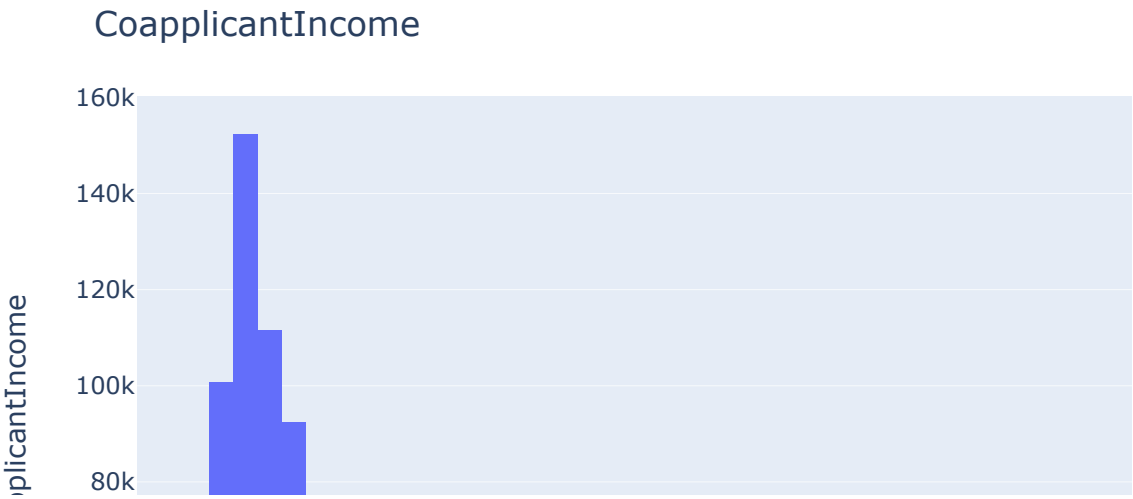
```
6104.064856533888
2923.8644597700627
85.51500809120331
```

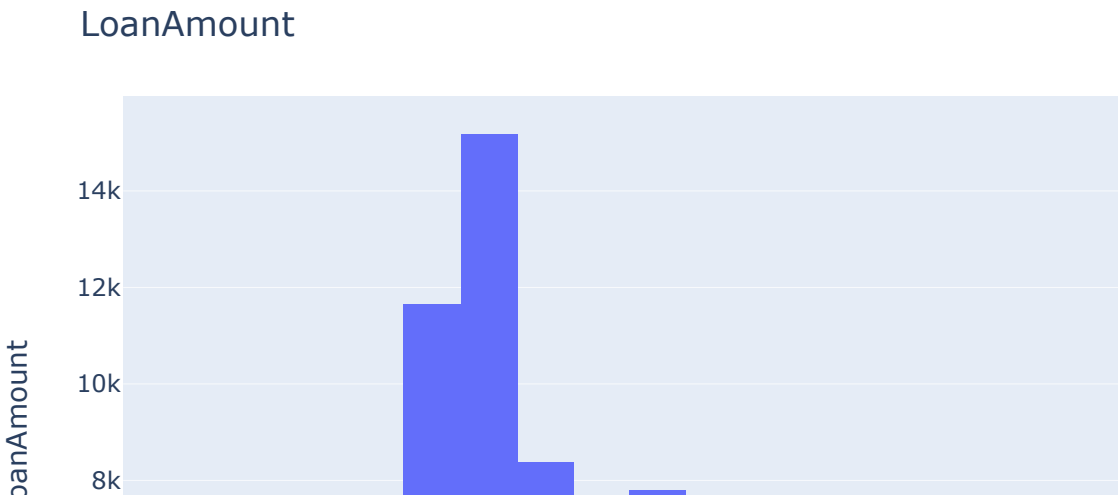
In [32]:
```
fig=px.histogram(df['ApplicantIncome'],x='ApplicantIncome',y='ApplicantIncome')
fig.update_layout(title='ApplicantIncome')
fig.show()

fig=px.histogram(df['CoapplicantIncome'],x='CoapplicantIncome',y='CoapplicantIncome
fig.update_layout(title='CoapplicantIncome')
fig.show()


fig=px.histogram(df['LoanAmount'],x='LoanAmount',y='LoanAmount')
fig.update_layout(title='LoanAmount')
fig.show()
```


ApplicantIncome
500k
400k
300k
licantIncome


CoapplicantIncome
160k
140k
120k
100k
80k
plicantIncome



Missing Values

In [33]:
```
df['Gender'].fillna(df['Gender'].mode()[0],inplace=True)
```

In [34]:
```
df['Married'].fillna(df['Married'].mode()[0],inplace=True)
```

In [35]:
```
df['Dependents'].fillna(df['Dependents'].mode()[0],inplace=True)
```

In [36]:
```
df["Self_Employed"].fillna(df["Self_Employed"].mode()[0],inplace=True)
df["Loan_Amount_Term"].fillna(df["Loan_Amount_Term"].mode()[0],inplace=True)
df["Credit_History"].fillna(df["Credit_History"].mode()[0],inplace=True)
```

In [37]:
```
df["LoanAmount"].fillna(df["LoanAmount"].median(),inplace=True)
```

In [38]:
```
df.dtypes
```

Out[38]:
```
Loan_ID               object
Gender                object
Married               object
Dependents            object
Education             object
Self_Employed         object
ApplicantIncome        int64
CoapplicantIncome    float64
LoanAmount           float64
Loan_Amount_Term     float64
Credit_History       float64
Property_Area         object
Loan_Status           object
dtype: object
```

In [39]:
```
df.Dependents.value_counts()
```

Out[39]:
```
0     360
1     102
2     101
3+     51
Name: Dependents, dtype: int64
```

In [40]:
```
df['Dependents']=df['Dependents'].replace('3+',int(3))
df['Dependents']=df['Dependents'].replace('1',int(1))
df['Dependents']=df['Dependents'].replace('2',int(2))
df['Dependents']=df['Dependents'].replace('0',int(0))
```

In [41]:
```
df.Gender.value_counts()
```

Out[41]:
```
Male      502
Female    112
Name: Gender, dtype: int64
```

In [42]:
```
df['Gender']=le.fit_transform(df['Gender'])
```

In [43]:
```
df.Gender.value_counts()
```

Out[43]:
```
1    502
0    112
Name: Gender, dtype: int64
```

In [44]:
```
df["Married"] = le.fit_transform(df["Married"])
df["Education"] = le.fit_transform(df["Education"])
df["Self_Employed"] = le.fit_transform(df["Self_Employed"])
df["Property_Area"] = le.fit_transform(df["Property_Area"])
df["Loan_Status"] = le.fit_transform(df["Loan_Status"])
```

In [45]:
```
df.dtypes
```

Out[45]:
```
Loan_ID               object
Gender                 int32
Married                int32
Dependents             int64
Education              int32
Self_Employed          int32
ApplicantIncome        int64
CoapplicantIncome    float64
LoanAmount           float64
Loan_Amount_Term     float64
Credit_History       float64
Property_Area          int32
Loan_Status            int32
dtype: object
```

In [46]:
```
x=df.drop(['Loan_ID','Loan_Status'],axis=1)
y=df['Loan_Status']
```

In [47]:
```
x_train,x_test,y_train,y_test=train_test_split(x, y, test_size=0.33, random_state=4
```

All Models

In [48]:
```
Model1=SVC()
```

In [49]:
```
kernel=['linear', 'poly', 'rbf', 'sigmoid', 'precomputed']
c=[50,10,0.1,1.0,0.01]
gamma=["scale","auto"]
```

In [50]:
```
grid=dict(kernel=kernel,C=c,gamma=gamma)
```

In [51]:
```
cv=RepeatedStratifiedKFold(n_splits=10)
```

In [52]:
```
grid_search=GridSearchCV(estimator=Model1,param_grid=grid,cv=cv,scoring="accuracy")
```

In [53]:
```
# grid_result=grid_search.fit(x,y)
```

KNN model

In [54]:
```
model_knn=KNeighborsClassifier()
```

In [55]:
```
leaf_size=list(range(1,50))
n_neighbors=list(range(1,50))
p=[1,2]
hyperparameters=dict(leaf_size=leaf_size,n_neighbors=n_neighbors,p=p)
```

In [56]:
```
clf=GridSearchCV(model_knn,hyperparameters,cv=10)
```

In [57]:
```
grid_knn=clf.fit(x,y)
```

In [58]:
```
print('Best leaf_size:', grid_knn.best_estimator_.get_params()['leaf_size'])
print('Best p:', grid_knn.best_estimator_.get_params()['p'])
print('Best n_neighbors:', grid_knn.best_estimator_.get_params()['n_neighbors'])
LS = grid_knn.best_estimator_.get_params()['leaf_size']
P = grid_knn.best_estimator_.get_params()['p']
Num = grid_knn.best_estimator_.get_params()['n_neighbors']
KNN = KNeighborsClassifier(leaf_size=LS,p=P,n_neighbors=Num)
KNN.fit(x_train,y_train)
y_pred = KNN.predict(x_test)
print(classification_report(y_pred,y_test))
print("KNeighborsClassifier:>",accuracy_score(y_pred,y_test))
```

```
Best leaf_size: 1
Best p: 2
Best n_neighbors: 27
              precision    recall  f1-score   support

           0       0.03      0.50      0.05         4
           1       0.98      0.65      0.78       199

    accuracy                           0.65       203
   macro avg       0.51      0.57      0.42       203
weighted avg       0.97      0.65      0.77       203

KNeighborsClassifier:> 0.645320197044335
```

Decision Tree

In [59]:
```
model_DCT=DecisionTreeClassifier()
```

In [60]:
```
criterion= ["gini", "entropy"]
max_depth=[3,4,5]
hyperparameters=dict(criterion=criterion,max_depth=max_depth)
```

In [61]:
```
clf1=GridSearchCV(model_DCT,hyperparameters,cv=10)
```

In [62]:
```
grid_DCT=clf1.fit(x,y)
```

In [63]:
```
print('Best criterion:', grid_DCT.best_estimator_.get_params()['criterion'])
print('Best max_depth:', grid_DCT.best_estimator_.get_params()['max_depth'])
LS = grid_DCT.best_estimator_.get_params()['criterion']
P = grid_DCT.best_estimator_.get_params()['max_depth']
Num = grid_DCT.best_estimator_.get_params()['criterion']
KNN = DecisionTreeClassifier(criterion=LS,max_depth=P)
KNN.fit(x_train,y_train)
y_pred = KNN.predict(x_test)
print(classification_report(y_pred,y_test))
print("KNeighborsClassifier:>",accuracy_score(y_pred,y_test))
```

```
Best criterion: gini
Best max_depth: 3
              precision    recall  f1-score   support

           0       0.44      0.94      0.60        34
           1       0.98      0.76      0.86       169

    accuracy                           0.79       203
   macro avg       0.71      0.85      0.73       203
weighted avg       0.89      0.79      0.82       203

KNeighborsClassifier:> 0.7931034482758621
```

In [ ]:
```
# Area under the curve

import matplotlib.pyplot as plt
fpr, tpr, thresholds = metrics.roc_curve(y_test, y_pred)
plt.plot(fpr, tpr)
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.0])
plt.title('ROC curve for diabetes classifier')
plt.xlabel('False Positive Rate (1 - Specificity)')
plt.ylabel('True Positive Rate (Sensitivity)')
plt.grid(True)
```