Project By : PRASAD JADHAV

In [1]:
```
import pandas as pd
import numpy as np

import matplotlib.pyplot as plt
import seaborn as sns

from imblearn.over_sampling import SMOTE

from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import cross_val_score
from sklearn.metrics import accuracy_score

from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import GradientBoostingClassifier
from sklearn import svm

from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import RandomizedSearchCV
```

In [2]:
```
import warnings
warnings.filterwarnings('ignore')
```

In [ ]:
```
# Loan_ID : Unique Loan ID

# Gender : Male/ Female

# Married : Applicant married (Y/N)

# Dependents : Number of dependents

# Education : Applicant Education (Graduate/ Under Graduate)
```

```
# Self_Employed : Self employed (Y/N)

# ApplicantIncome : Applicant income

# CoapplicantIncome : Coapplicant income

# LoanAmount : Loan amount in thousands of dollars

# Loan_Amount_Term : Term of loan in months

# Credit_History : Credit history meets guidelines yes or no

# Property_Area : Urban/ Semi Urban/ Rural

# Loan_Status : Loan approved (Y/N) this is the target variable
```

In [13]:
```
dataset = pd.read_csv('loan_prediction.csv')
dataset.shape
```

Out[13]:
```
(614, 13)
```

In [14]:
```
dataset.head()
```

Out[14]:

	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	CoapplicantInco
0	LP001002	Male	No	0	Graduate	No	5849	
1	LP001003	Male	Yes	1	Graduate	No	4583	150
2	LP001005	Male	Yes	0	Graduate	Yes	3000	
3	LP001006	Male	Yes	0	Not Graduate	No	2583	235
4	LP001008	Male	No	0	Graduate	No	6000	

In [15]:
```
dataset.tail()
```

Out[15]:

	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	CoapplicantIn
609	LP002978	Female	No	0	Graduate	No	2900	
610	LP002979	Male	Yes	3+	Graduate	No	4106	
611	LP002983	Male	Yes	1	Graduate	No	8072	
612	LP002984	Male	Yes	2	Graduate	No	7583	
613	LP002990	Female	No	0	Graduate	Yes	4583	

In [16]:
```
print('Number of Rows:',dataset.shape[0])
print('Number of Columns:',dataset.shape[1])
```

```
Number of Rows: 614
Number of Columns: 13
```

In [17]:
```
dataset.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 [18]:
```
dataset.isnull().sum()
```

Out[18]:
```
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 [19]:
```
dataset.duplicated().sum()
```

Out[19]:
```
0
```

In [20]:
```
dataset = dataset.dropna()
```

In [21]:
```
dataset = dataset.drop('Loan_ID',axis=1)
```

In [22]:
```
dataset.describe()
```

Out[22]:

	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amount_Term	Credit_History
count	480.000000	480.000000	480.000000	480.000000	480.000000
mean	5364.231250	1581.093583	144.735417	342.050000	0.854167
std	5668.251251	2617.692267	80.508164	65.212401	0.353307
min	150.000000	0.000000	9.000000	36.000000	0.000000
25%	2898.750000	0.000000	100.000000	360.000000	1.000000
50%	3859.000000	1084.500000	128.000000	360.000000	1.000000
75%	5852.500000	2253.250000	170.000000	360.000000	1.000000
max	81000.000000	33837.000000	600.000000	480.000000	1.000000

In [23]:
```
dataset.corr()
```

Out[23]:

	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amount_Term	Credit_Histor
ApplicantIncome	1.000000	-0.112588	0.495310	-0.010838	-0.05615
CoapplicantIncome	-0.112588	1.000000	0.190740	-0.005775	-0.00869
LoanAmount	0.495310	0.190740	1.000000	0.050867	-0.04077
Loan_Amount_Term	-0.010838	-0.005775	0.050867	1.000000	0.03293
Credit_History	-0.056152	-0.008692	-0.040773	0.032937	1.00000

In [24]:
```
dataset.cov()
```

Out[24]:

	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amount_Term	Credit_His
ApplicantIncome	3.212907e+07	-1.670551e+06	226029.825404	-4006.195303	-112.452
CoapplicantIncome	-1.670551e+06	6.852313e+06	40197.560179	-985.773871	-8.038
LoanAmount	2.260298e+05	4.019756e+04	6481.564505	267.057098	-1.159
Loan_Amount_Term	-4.006195e+03	-9.857739e+02	267.057098	4252.657203	0.758
Credit_History	-1.124526e+02	-8.038516e+00	-1.159751	0.758873	0.124

In [25]:
```
dataset['Loan_Status'].value_counts()
```

Out[25]:
```
Y    332
N    148
Name: Loan_Status, dtype: int64
```

In [26]:
```
dataset['Gender'].value_counts()
```

Out[26]:
```
Male      394
Female     86
Name: Gender, dtype: int64
```

In [27]:
```
dataset['Married'].value_counts()
```

Out[27]:
```
Yes    311
No     169
Name: Married, dtype: int64
```

In [28]:
```
dataset['Education'].value_counts()
```

Out[28]:
```
Graduate        383
Not Graduate     97
Name: Education, dtype: int64
```

In [29]:
```
dataset['Self_Employed'].value_counts()
```

Out[29]:
```
No     414
Yes     66
Name: Self_Employed, dtype: int64
```

In [30]:
```
dataset['Property_Area'].value_counts()
```

Out[30]:
```
Semiurban    191
Urban        150
Rural        139
Name: Property_Area, dtype: int64
```

In [31]:
```
dataset['Credit_History'].value_counts()
```

Out[31]:
```
1.0    410
0.0     70
Name: Credit_History, dtype: int64
```

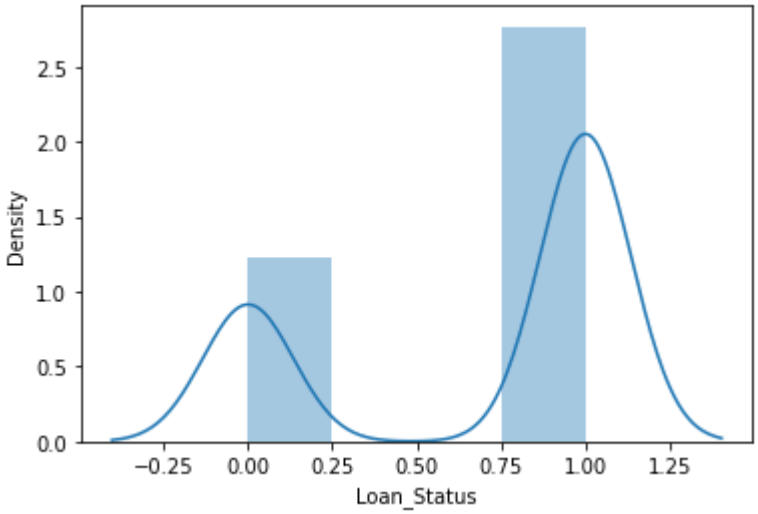
In [32]:
```
dataset['Dependents'] = dataset['Dependents'].replace(to_replace='3+',value='4
```

In [33]:
```
dataset['Dependents'].value_counts()
```

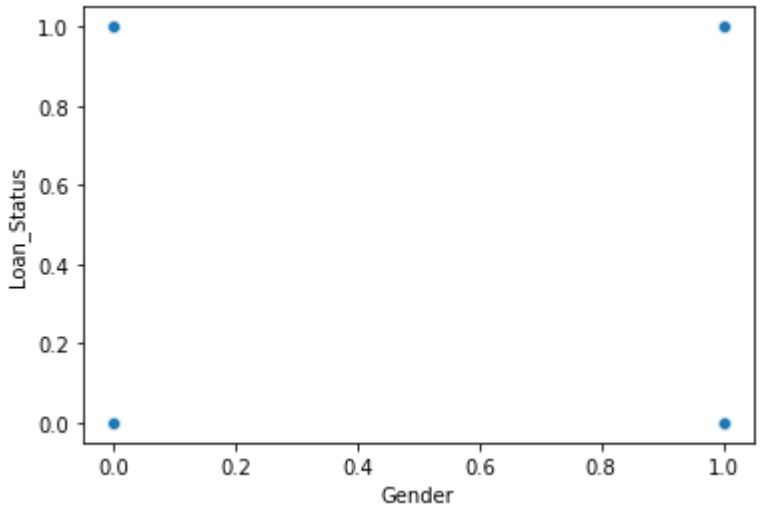
Out[33]:
```
0    274
2     85
1     80
4     41
Name: Dependents, dtype: int64
```

In [34]:
```
dataset['Gender'] = dataset['Gender'].map({'Male':1,'Female':0}).astype('int')
dataset['Married'] = dataset['Married'].map({'Yes':1,'No':0}).astype('int')
dataset['Education'] = dataset['Education'].map({'Graduate':1,'Not Graduate':0
dataset['Self_Employed'] = dataset['Self_Employed'].map({'Yes':1,'No':0}).asty
dataset['Property_Area'] = dataset['Property_Area'].map({'Rural':0,'Urban':1,'
dataset['Loan_Status'] = dataset['Loan_Status'].map({'Y':1,'N':0}).astype('int
```

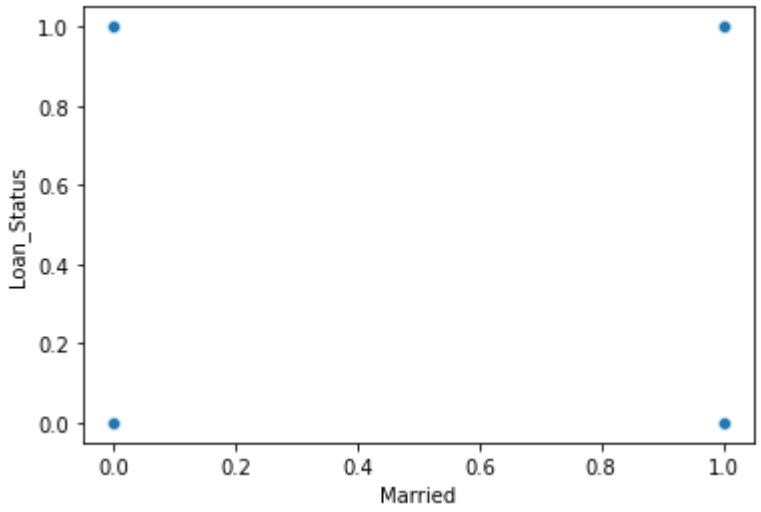
In [35]:
```
sns.distplot(dataset['Loan_Status'])
plt.show()
```



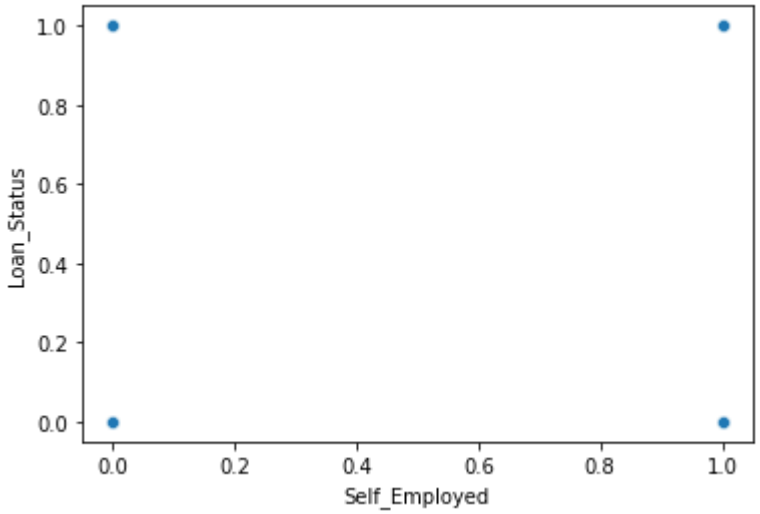
In [36]:
```
sns.scatterplot(dataset['Gender'],dataset['Loan_Status'])
plt.show()
```



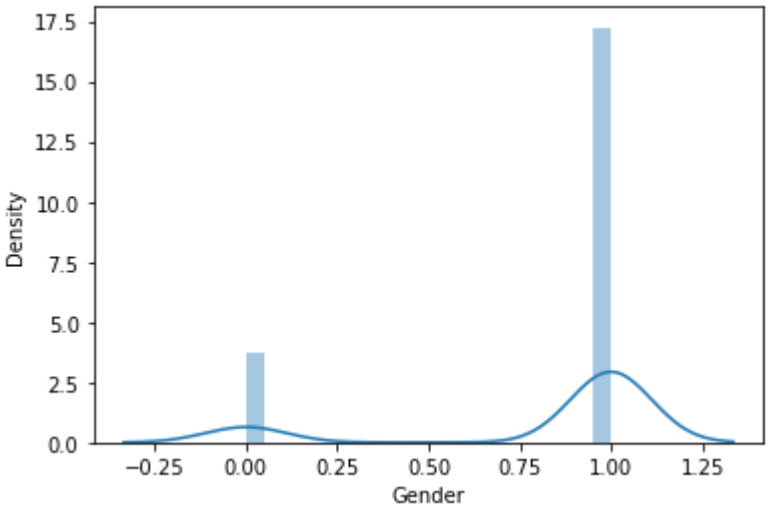
In [37]:
```
sns.scatterplot(dataset['Married'],dataset['Loan_Status'])
plt.show()
```



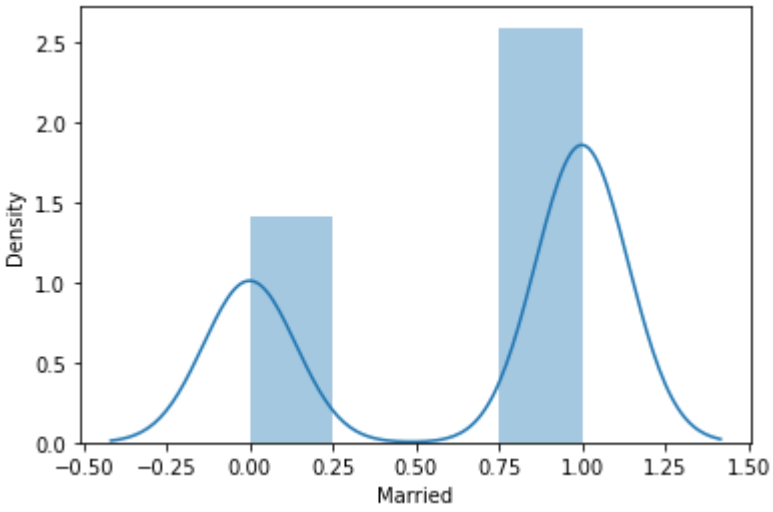
In [28]:
```
sns.scatterplot(dataset['Self_Employed'],dataset['Loan_Status'])
plt.show()
```



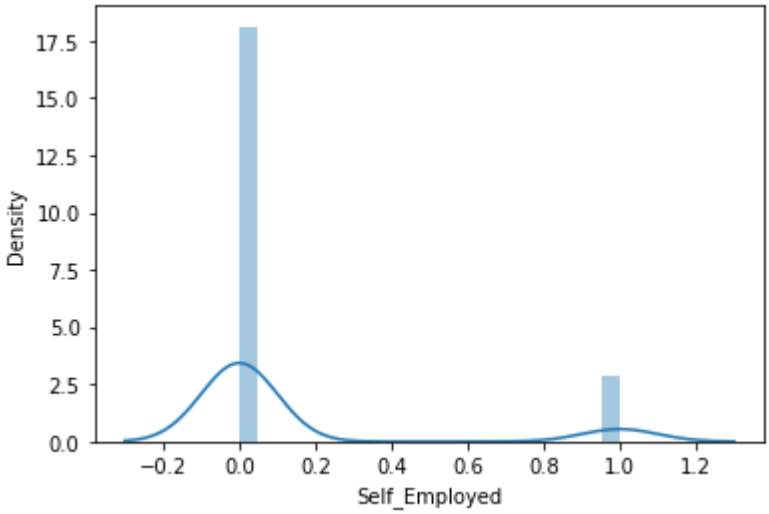
In [29]:
```
sns.distplot(dataset['Gender'])
plt.show()
```



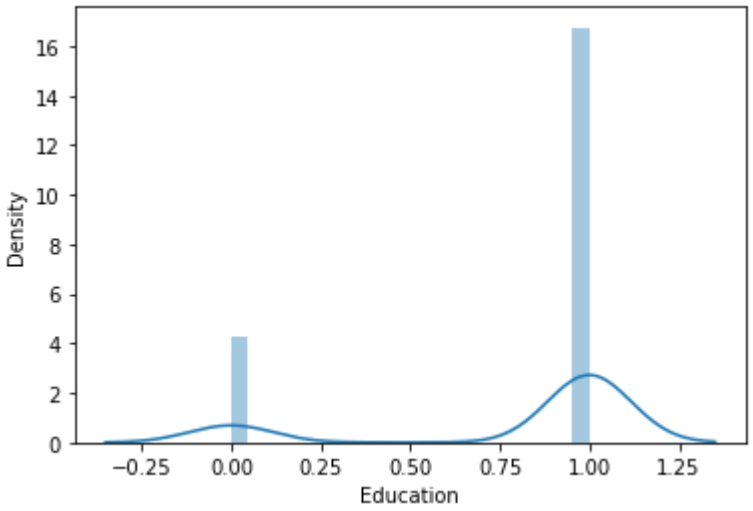
In [30]:
```
sns.distplot(dataset['Married'])
plt.show()
```



In [31]:
```
sns.distplot(dataset['Self_Employed'])
plt.show()
```



In [32]:
```
sns.distplot(dataset['Education'])
plt.show()
```



In [33]:
```
dataset.dtypes
```

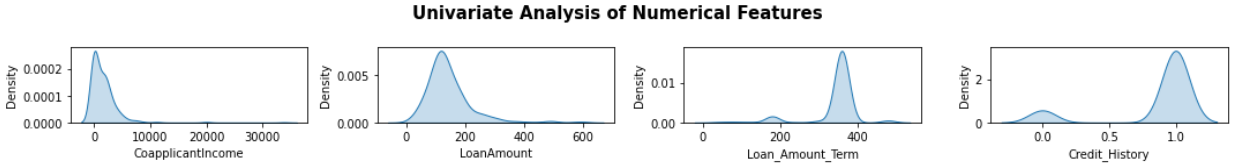
Out[33]:
```
Gender                int32
Married               int32
Dependents           object
Education             int32
Self_Employed         int32
ApplicantIncome       int64
CoapplicantIncome   float64
LoanAmount          float64
Loan_Amount_Term    float64
Credit_History      float64
Property_Area         int32
Loan_Status           int32
dtype: object
```

In [34]:
```
float_cols = [x for x in dataset.columns if dataset[x].dtypes=='float64']
```

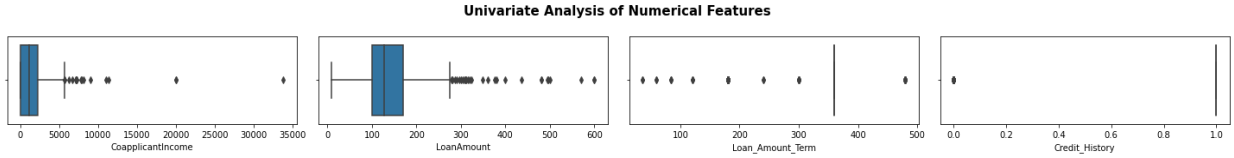
In [35]:
```
object_cols = [x for x in dataset.columns if dataset[x].dtypes=='object']
```

In [36]:
```
int_cols = [x for x in dataset.columns if dataset[x].dtypes=='int64']
```

In [37]:
```
plt.figure(figsize=(15,15))
plt.suptitle('Univariate Analysis of Numerical Features',fontweight='bold',fon

for i in range(0,len(float_cols)):
    plt.subplot(10,4,i+1)
    sns.kdeplot(x=dataset[float_cols[i]],shade=True)
    plt.tight_layout()
```



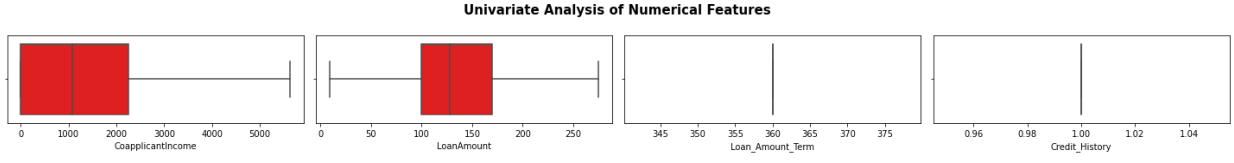
In [38]:
```
plt.figure(figsize=(20,20))
plt.suptitle('Univariate Analysis of Numerical Features',fontweight='bold',fon

for i in range(0,len(float_cols)):
    plt.subplot(10,4,i+1)
    sns.boxplot(data=dataset,x=float_cols[i])
    plt.xlabel(float_cols[i])
    plt.tight_layout()
```



In [39]:
```
def remove_outliers(in_dataset, in_cols):

    first_quartile = in_dataset[in_cols].quantile(0.25)
    third_quartile = in_dataset[in_cols].quantile(0.75)
    iqr = third_quartile - first_quartile
    upper_limit = third_quartile + 1.5 * iqr
    lower_limit = first_quartile - 1.5 * iqr
    in_dataset.loc[(in_dataset[in_cols] > upper_limit), in_cols] = upper_limit
    in_dataset.loc[(in_dataset[in_cols] < lower_limit), in_cols] = lower_limit
    return in_dataset
```

In [40]:
```
for features in float_cols:
    dataset = remove_outliers(dataset,features)
```

In [41]:
```
plt.figure(figsize=(20,20))
plt.suptitle('Univariate Analysis of Numerical Features',fontweight='bold',fon

for i in range(0,len(float_cols)):
    plt.subplot(10,4,i+1)
    sns.boxplot(data=dataset,x=float_cols[i],color='red')
    plt.xlabel(float_cols[i])
    plt.tight_layout()
```



In [38]:
```
dataset['Loan_Status'].value_counts()
```

Out[38]:
```
1    332
0    148
Name: Loan_Status, dtype: int64
```

In [39]:
```
X = dataset.drop('Loan_Status',axis=1)
y = dataset['Loan_Status']
```

In [46]:
```
cols = ['ApplicantIncome','CoapplicantIncome','LoanAmount','Loan_Amount_Term']
```

In [47]:
```
st = StandardScaler()
X[cols]=st.fit_transform(X[cols])
```

In [50]:
```
from sklearn.model_selection import train_test_split
from sklearn.model_selection import cross_val_score
from sklearn.metrics import accuracy_score
import numpy as np
```

In [51]:
```
models_df = {}

def model_val(model,X,y):
    X_train,X_test,y_train,y_test=train_test_split(X,y,
                                                   test_size=0.20,
                                                   random_state=42)
    model.fit(X_train,y_train)
    y_pred=model.predict(X_test)
    print(f"{model} accuracy is {accuracy_score(y_test,y_pred)}")

    score = cross_val_score(model,X,y,cv=5)
    print(f"{model} Avg cross val score is {np.mean(score)}")
    models_df[model]=round(np.mean(score)*100,2)
```

In [58]:
```
models_df
```

Out[58]:
```
{LogisticRegression(): 80.21,
 SVC(): 79.79,
 DecisionTreeClassifier(): 72.5,
 RandomForestClassifier(): 80.0,
 GradientBoostingClassifier(): 78.54}
```

In [53]:
```
from sklearn.linear_model import LogisticRegression
model = LogisticRegression()
model_val(model,X,y)
```

```
LogisticRegression() accuracy is 0.8229166666666666
LogisticRegression() Avg cross val score is 0.8020833333333334
```

In [54]:
```
from sklearn import svm
model = svm.SVC()
model_val(model,X,y)
```

```
SVC() accuracy is 0.8020833333333334
SVC() Avg cross val score is 0.7979166666666667
```

In [55]:
```
from sklearn.tree import DecisionTreeClassifier
model = DecisionTreeClassifier()
model_val(model,X,y)
```

```
DecisionTreeClassifier() accuracy is 0.7395833333333334
DecisionTreeClassifier() Avg cross val score is 0.725
```

In [56]:
```
from sklearn.ensemble import RandomForestClassifier
model =RandomForestClassifier()
model_val(model,X,y)
```

```
RandomForestClassifier() accuracy is 0.8125
RandomForestClassifier() Avg cross val score is 0.8
```

In [57]:
```
from sklearn.ensemble import GradientBoostingClassifier
model =GradientBoostingClassifier()
model_val(model,X,y)
```

```
GradientBoostingClassifier() accuracy is 0.7604166666666666
GradientBoostingClassifier() Avg cross val score is 0.7854166666666667
```

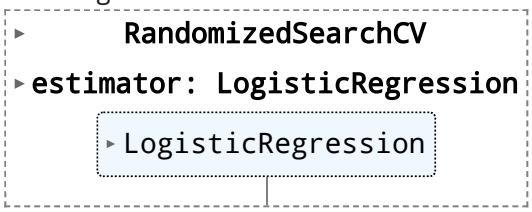
In [59]:
```
log_reg_grid={"C":np.logspace(-4,4,20),
              "solver":['liblinear']}
```

In [60]:
```
rs_log_reg=RandomizedSearchCV(LogisticRegression(),
                   param_distributions=log_reg_grid,
                  n_iter=20,cv=5,verbose=True)
```

In [61]:
```
rs_log_reg.fit(X,y)
```

```
Fitting 5 folds for each of 20 candidates, totalling 100 fits
```

Out[61]:



In [62]:
```
rs_log_reg.best_score_
```

Out[62]:
```
0.8020833333333334
```

In [63]:
```
rs_log_reg.best_params_
```

Out[63]:
```
{'solver': 'liblinear', 'C': 0.23357214690901212}
```

In [64]:
```
svc_grid = {'C':[0.25,0.50,0.75,1],"kernel":["linear"]}
```

In [65]:
```
rs_svc=RandomizedSearchCV(svm.SVC(),
                  param_distributions=svc_grid,
                   cv=5,
                   n_iter=20,
                  verbose=True)
```

In [66]:
```
rs_svc.fit(X,y)
```

```
Fitting 5 folds for each of 4 candidates, totalling 20 fits
```

Out[66]:

```
RandomizedSearchCV
  estimator: SVC
    SVC
```

In [67]:
```
rs_svc.best_score_
```

Out[67]:
```
0.8083333333333333
```

In [68]:
```
rs_svc.best_params_
```

Out[68]:
```
{'kernel': 'linear', 'C': 0.25}
```

In [69]:
```
RandomForestClassifier()
```

Out[69]:

```
RandomForestClassifier
RandomForestClassifier()
```

In [70]:
```
rf_grid={'n_estimators':np.arange(10,1000,10),
  'max_features':['auto','sqrt'],
 'max_depth':[None,3,5,10,20,30],
 'min_samples_split':[2,5,20,50,100],
 'min_samples_leaf':[1,2,5,10]
 }
```

In [71]:
```
rs_rf=RandomizedSearchCV(RandomForestClassifier(),
                  param_distributions=rf_grid,
                   cv=5,
                   n_iter=20,
                  verbose=True)
```

In [72]:
```
rs_rf.fit(X,y)
```

```
Fitting 5 folds for each of 20 candidates, totalling 100 fits
```

Out[72]:

```
RandomizedSearchCV
  estimator: RandomForestClassifier
    RandomForestClassifier
```

In [73]:
```
rs_rf.best_score_
```

Out[73]:
```
0.8083333333333333
```

In [74]:
```
rs_rf.best_params_
```

Out[74]:
```
{'n_estimators': 780,
 'min_samples_split': 20,
 'min_samples_leaf': 1,
 'max_features': 'auto',
 'max_depth': 5}
```

In [78]:
```
X_res,y_res = SMOTE().fit_resample(X,y)
```

In [79]:
```
X_train,X_test,y_train,y_test = train_test_split(X_res,y_res,test_size=0.20,ra
```

In [80]:
```
st = StandardScaler()
X_train = st.fit_transform(X_train)
X_test = st.fit_transform(X_test)
```

In [81]:
```
model_df = {}

def model_val(model,X,y):
    X_train,X_test,y_train,y_test = train_test_split(X_res,y_res,test_size=0.2
    model.fit(X_train,y_train)
    y_pred = model.predict(X_test)
    print(f'{model} Accuracy is {accuracy_score(y_test,y_pred)}')

    score = cross_val_score(model,X,y,cv=5,n_jobs=-1)
    print(f'{model} Average cross val score is {np.mean(score)}')
    model_df[model] = round(np.mean(score)*100,2)
```

In [82]:
```
model = LogisticRegression()
model_val(model,X,y)
```

```
LogisticRegression() Accuracy is 0.7593984962406015
LogisticRegression() Average cross val score is 0.8020833333333334
```

In [49]:
```
model = DecisionTreeClassifier()
model_val(model,X,y)
```

```
DecisionTreeClassifier() Accuracy is 0.706766917293233
DecisionTreeClassifier() Average cross val score is 0.5729166666666667
```

In [50]:
```
model = RandomForestClassifier()
model_val(model,X,y)
```

```
RandomForestClassifier() Accuracy is 0.7368421052631579
RandomForestClassifier() Average cross val score is 0.6375
```

In [51]:
```
model = GradientBoostingClassifier()
model_val(model,X,y)
```

```
GradientBoostingClassifier() Accuracy is 0.7593984962406015
GradientBoostingClassifier() Average cross val score is 0.675
```

In [52]:
```
model_df
```

Out[52]:
```
{LogisticRegression(): 69.17,
 DecisionTreeClassifier(): 57.29,
 RandomForestClassifier(): 63.75,
 GradientBoostingClassifier(): 67.5}
```

In [53]:
```
C = 1.0
svm = svm.SVC(kernel='linear',C=C)
svm.fit(X_train,y_train)
```

Out[53]:

▾ SVC

```
SVC(kernel='linear')
```

In [54]:
```
y_pred = svm.predict(X_test)
```

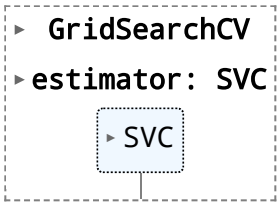
In [55]:
```
accuracy_score(y_test,y_pred)
```

Out[55]:
```
0.7293233082706767
```

In [56]:
```
parameters = [

    {'kernel':['linear'],'C': [1,10]},
    {'kernel':['poly'],'C':[1, 10]},
    {'kernel':['rbf'],'gamma': [1e-3,1e-4],'C': [1,10]}
]
```

In [57]:
```
gr = GridSearchCV(svm,parameters,n_jobs=-1)
gr.fit(X_train,y_train)
```

Out[57]:



In [58]:
```
y_pred_ = gr.predict(X_test)
```

In [59]:
```
accuracy_score(y_test,y_pred_)
```

Out[59]:
```
0.7293233082706767
```

In [60]:
```
from sklearn.model_selection import StratifiedKFold
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.calibration import CalibratedClassifierCV
from sklearn.neural_network import MLPClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
from sklearn.naive_bayes import GaussianNB
from sklearn.ensemble import AdaBoostClassifier
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import ExtraTreesClassifier
from sklearn.linear_model import SGDClassifier
from sklearn.svm import SVC
from xgboost import XGBClassifier
```

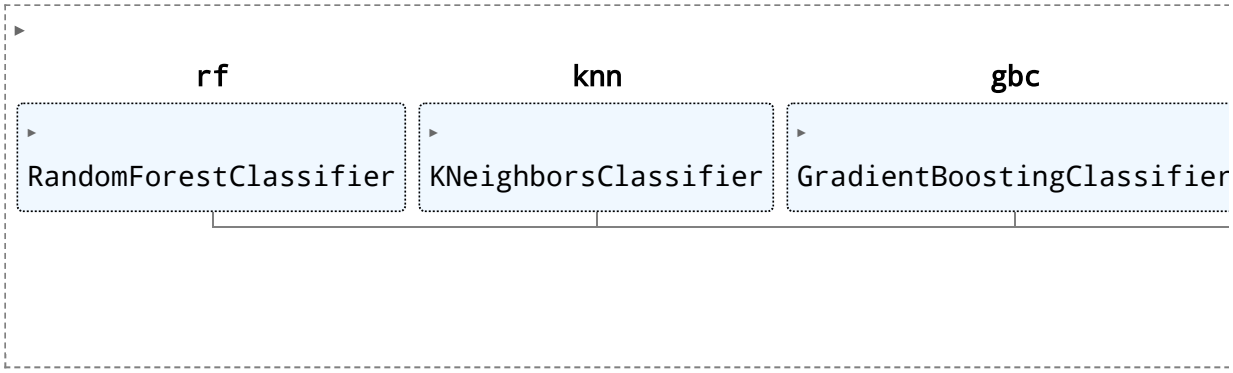
In [61]:
```
estimators = [
              ('rf', RandomForestClassifier(n_estimators = 10, random_state = 4
              ('knn', KNeighborsClassifier(n_neighbors = 10)),
              ('gbc', GradientBoostingClassifier()),
              ('lr', LogisticRegression()),
              ('ccv', CalibratedClassifierCV()),
              ('mlp', MLPClassifier()),
              ('dt', DecisionTreeClassifier()),
              ('lda', LinearDiscriminantAnalysis()),
              ('gnb', GaussianNB()),
```

```
            ('adb', AdaBoostClassifier()),
            ('etc', ExtraTreesClassifier()),
            ('sgd', SGDClassifier()),
            ('svm', SVC()),
            ('xgb', XGBClassifier(n_estimators= 10, random_state = 42))
            ]
```

In [62]:
```
from sklearn.ensemble import StackingClassifier

st = StackingClassifier(
                        estimators = estimators,
                        final_estimator = LogisticRegression(),
                        cv = 10,
                        stack_method='predict',
                        n_jobs=-1
                        )
```

In [63]:
```
st.fit(X_train,y_train)
```

Out[63]:



In [64]:
```
y_pred = st.predict(X_test)
```

In [65]:
```
accuracy_score(y_test,y_pred)
```

Out[65]:
```
0.6992481203007519
```

In [ ]:
```
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import BaggingClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import ExtraTreesClassifier
from sklearn.model_selection import GridSearchCV
from xgboost import XGBClassifier
```

In [66]:
```
dt_parameters = {
 'criterion' : ['gini', 'entropy', 'log_loss'],
 'splitter' : ['best', 'random'],
 'max_depth' : range(1,10,1),
 'min_samples_split' : range(2,10,2),
 'min_samples_leaf' : range(1,5,1),
 'max_features' : ['auto', 'sqrt', 'log2']
}

bg_parameters = {
 'n_estimators' : [5, 10, 15],
 'max_samples' : range(2, 10, 1),
 'max_features' : range(2, 10, 3)
```

```
}

rf_parameters = {
 'criterion' : ['gini', 'entropy', 'log_loss'],
 'max_depth' : range(1, 10, 1),
 'min_samples_split' : range(2, 10, 2),
 'min_samples_leaf' : range(1, 10, 1),
}

et_parameters = {
 'n_estimators' : [10,20,30],
 'criterion' : ['gini', 'entropy', 'log_loss'],
 'max_depth' : range(2,10,1),
 'min_samples_split' : range(2,10,2),
 'min_samples_leaf' : range(1,5,1),
 'max_features' : ['sqrt', 'log2']
}

xgb_parameters = {
 'max_depth' : [2,4,6],
 'min_child_weight' : [2,4,6],
 'gamma' : [i/10 for i in range(4)]
}
```

In [ ]:
```
hyper_1 = GridSearchCV(estimator = DecisionTreeClassifier(), param_grid = dt_pa
hyper_2 = GridSearchCV(estimator = BaggingClassifier(), param_grid = bg_parame
hyper_3 = GridSearchCV(estimator = RandomForestClassifier(), param_grid = rf_pa
hyper_4 = GridSearchCV(estimator = ExtraTreesClassifier(), param_grid = et_para
hyper_5 = GridSearchCV(estimator = XGBClassifier(), param_grid = xgb_parameters
```

In [ ]:
```
hyper_1.fit(X_train,y_train)
hyper_2.fit(X_train,y_train)
hyper_3.fit(X_train,y_train)
hyper_4.fit(X_train,y_train)
hyper_5.fit(X_train,y_train)
```

In [ ]:
```
h_pred_1 = hyper_1.predict(X_test)
h_pred_2 = hyper_2.predict(X_test)
h_pred_3 = hyper_3.predict(X_test)
h_pred_4 = hyper_4.predict(X_test)
h_pred_5 = hyper_5.predict(X_test)
```

In [ ]:
```
print(accuracy_score(y_test,h_pred_1))
print(accuracy_score(y_test,h_pred_2))
print(accuracy_score(y_test,h_pred_3))
print(accuracy_score(y_test,h_pred_4))
print(accuracy_score(y_test,h_pred_5))
```

In [ ]:
```
rf_grid = {'n_estimators':np.arange(10,1000,10),
'max_features':['auto','sqrt'],
'max_depth':[None,3,5,10,20,30],
'min_samples_split':[2,5,20,50,100],
'min_samples_leaf':[1,2,5,10]}
```

In [ ]:
```
rs_rf = RandomizedSearchCV(RandomForestClassifier(),param_distributions=rf_gri
                           cv=5,n_iter=20,verbose=True)
```

In [ ]:
```
rs_rf.fit(X,y)
```

In [ ]:
```
rs_rf.best_score_
```

In [68]:
```
import sklearn
from sklearn.model_selection import train_test_split,GridSearchCV,ShuffleSplit
from sklearn.preprocessing import StandardScaler,OneHotEncoder
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier,AdaBoostClassifier
from sklearn.naive_bayes import BernoulliNB,GaussianNB
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score,precision_score,recall_score,f1_sco

model_list = [('lr',LogisticRegression()),('dt',DecisionTreeClassifier()),('rf
              ,('svm',SVC()),('bnb',BernoulliNB()),('gnb',GaussianNB())]

for model_name,model in model_list:
    m = model.fit(X_train,y_train)
    y_pred = model.predict(X_train)
    print(f'{model_name} : {accuracy_score(y_train,y_pred)}')
```

```
lr : 0.6610169491525424
dt : 1.0
rf : 0.9887005649717514
adb : 0.7664783427495292
svm : 0.7532956685499058
bnb : 0.664783427495292
gnb : 0.6892655367231638
```

In [69]:
```
for model_name,model in model_list:
    m = model.fit(X_train,y_train)
    y_pred = model.predict(X_test)
    print(f'{model_name} : {accuracy_score(y_test,y_pred)}')
```

```
lr : 0.6917293233082706
dt : 0.6541353383458647
rf : 0.7368421052631579
adb : 0.5939849624060151
svm : 0.6691729323308271
bnb : 0.6992481203007519
gnb : 0.7142857142857143
```

In [ ]:
```
algos = {
    'rf':{
        'model':RandomForestClassifier(),
        'params':{
            'n_estimators':[10,20,30,40],
            'criterion':['gini','entropy'],
            'max_depth':[10,20,30],
            'min_samples_split':[2,4,6]
        }
    },
    'dt':{
        'model':DecisionTreeClassifier(),
        'params':{
            'criterion':['gini','entropy'],
            'max_depth':[10,20,30],
            'splitter':['best','random'],
```

```
        }
    },

    'lr':{
        'model':LogisticRegression(),
        'params':{
            'penalty':['l1','l2'],
            'C':[0.1,0.01,1,0.5],
            'solver':['liblinear','lbfgs']
        }

    },

    'svm':{
        'model':SVC(),
        'params':{
            'C':[0.1,0.01,1,0.5],
            'kernel':['linear','poly','rbf']

        }
    },

    'adb':{
        'model':AdaBoostClassifier(),
        'params':{
            'n_estimators':[10,20,30,50],
            'learning_rate':[0.1,1,0.01]
        }
    }

}

scores = []

cv = ShuffleSplit(n_splits=10,test_size=0.2,random_state=42)

for model_name,config in algos.items():
    gd = GridSearchCV(estimator=config['model'],param_grid=config['params'],cv=
    gd.fit(X,y)
    scores.append({'model_name':model_name,'best_score':gd.best_score_,'best_pa

scores = pd.DataFrame(scores)
scores.head()
```

In [84]:
```
X = dataset.drop('Loan_Status',axis=1)
y = dataset['Loan_Status']
```

In [85]:
```
```

In [86]:
```
from sklearn.preprocessing import StandardScaler
st = StandardScaler()
X[cols]=st.fit_transform(X[cols])
```

In [87]:
```
from sklearn.model_selection import train_test_split
from sklearn.model_selection import cross_val_score
from sklearn.metrics import accuracy_score
import numpy as np
```

In [88]:
```
model_df = {}

def model_val(model,X,y):
    X_train,X_test,y_train,y_test=train_test_split(X,y,
                                                   test_size=0.20,
                                                   random_state=42)
    model.fit(X_train,y_train)
    y_pred=model.predict(X_test)
    print(f"{model} accuracy is {accuracy_score(y_test,y_pred)}")

    score = cross_val_score(model,X,y,cv=5)
    print(f"{model} Avg cross val score is {np.mean(score)}")
    model_df[model]=round(np.mean(score)*100,2)
```

In [85]:
```
model_df
```

Out[85]:
```
{LogisticRegression(): 80.21}
```

In [84]:
```
X = dataset.drop('Loan_Status',axis=1)
y = dataset['Loan_Status']
```

In [86]:
```
rf = RandomForestClassifier(n_estimators=270,
 min_samples_split=5,
 min_samples_leaf=5,
 max_features='sqrt',
 max_depth=5)
```

In [87]:
```
rf.fit(X,y)
```

Out[87]:
```
RandomForestClassifier(max_depth=5, min_samples_leaf=5, min_samples_sp
lit=5,
                       n_estimators=270)
```

In [88]:
```
import pickle
import joblib
```

In [92]:
```
pickle.dump(rf,open('loan_status_predict.pkl','wb'))
```

In [91]:
```
model = pickle.load(open('loan_status_predict.pkl','rb'))
```

In [93]:
```
df = pd.DataFrame({
    'Gender':1,
    'Married':1,
    'Dependents':2,
    'Education':0,
    'Self_Employed':0,
    'ApplicantIncome':2889,
    'CoapplicantIncome':0.0,
    'LoanAmount':45,
    'Loan_Amount_Term':180,
    'Credit_History':0,
    'Property_Area':1
},index=[0])
```

In [94]:
```
df
```

Out[94]:

	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	CoapplicantIncome	LoanA
0	1	1	2	0	0	2889	0.0	

In [95]:
```
result = model.predict(df)
```

In [97]:
```
if result==1:
    print('Loan Approved')
else:
    print('Loan Not Approved')
```

```
Loan Not Approved
```

In [104…
```
p = model.predict(df)

prob = model.predict_proba(df)
if p == 1:
    print('Loan Approved!')
    print(f'You will be Loan Approved! with Probability of {prob[0][1]:.2f}')
else:
    print('Not-Loan Approved!')
```

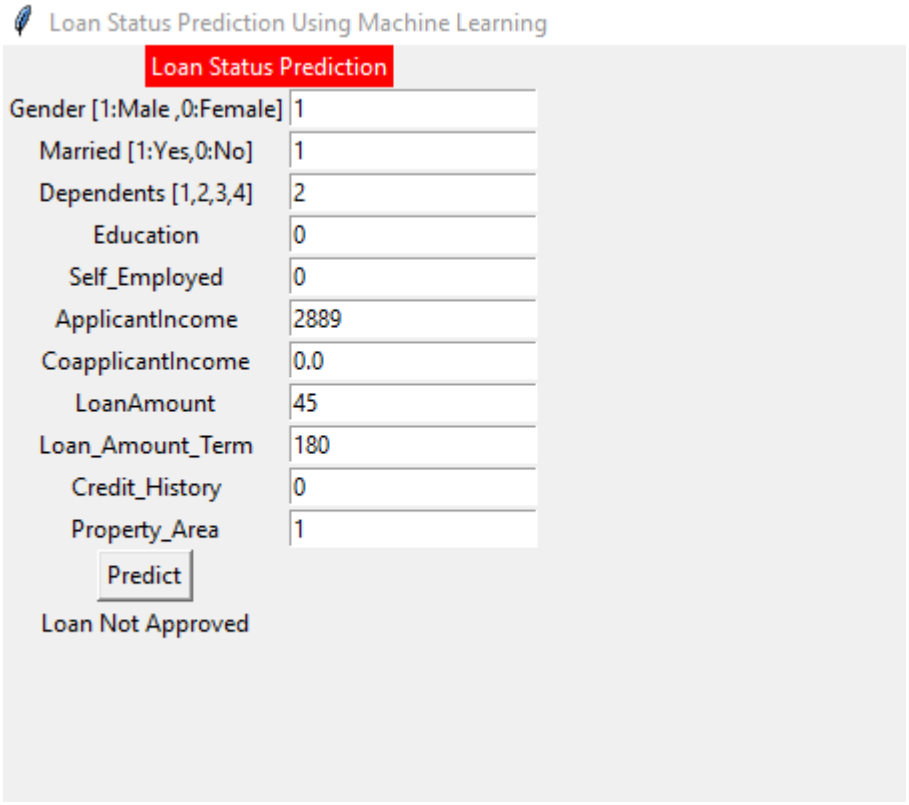
```
Not-Loan Approved!
```

In [98]:
```
from tkinter import *
```

In [107…
```
def show_entry():

    p1 = float(e1.get())
    p2 = float(e2.get())
    p3 = float(e3.get())
    p4 = float(e4.get())
    p5 = float(e5.get())
    p6 = float(e6.get())
    p7 = float(e7.get())
    p8 = float(e8.get())
    p9 = float(e9.get())
    p10 = float(e10.get())
    p11 = float(e11.get())

    model = pickle.load(open('loan_status_predict.pkl','rb'))
    df = pd.DataFrame({
```

```
    'Gender':p1,
    'Married':p2,
    'Dependents':p3,
    'Education':p4,
    'Self_Employed':p5,
    'ApplicantIncome':p6,
    'CoapplicantIncome':p7,
    'LoanAmount':p8,
    'Loan_Amount_Term':p9,
    'Credit_History':p10,
    'Property_Area':p11
},index=[0])
    result = model.predict(df)

    if result == 1:
        Label(master, text="Loan Approved").grid(row=31)
    else:
        Label(master, text="Loan Not Approved").grid(row=31)


master =Tk()
master.title("Loan Status Prediction Using Machine Learning")
label = Label(master,text = "Loan Status Prediction",bg = "red",
              fg = "white").grid(row=0,columnspan=2)

Label(master,text = "Gender [1:Male ,0:Female]").grid(row=1)
Label(master,text = "Married [1:Yes,0:No]").grid(row=2)
Label(master,text = "Dependents [1,2,3,4]").grid(row=3)
Label(master,text = "Education").grid(row=4)
Label(master,text = "Self_Employed").grid(row=5)
Label(master,text = "ApplicantIncome").grid(row=6)
Label(master,text = "CoapplicantIncome").grid(row=7)
Label(master,text = "LoanAmount").grid(row=8)
Label(master,text = "Loan_Amount_Term").grid(row=9)
Label(master,text = "Credit_History").grid(row=10)
Label(master,text = "Property_Area").grid(row=11)


e1 = Entry(master)
e2 = Entry(master)
e3 = Entry(master)
e4 = Entry(master)
e5 = Entry(master)
e6 = Entry(master)
e7 = Entry(master)
e8 = Entry(master)
e9 = Entry(master)
e10 = Entry(master)
e11 = Entry(master)


e1.grid(row=1,column=1)
e2.grid(row=2,column=1)
e3.grid(row=3,column=1)
e4.grid(row=4,column=1)
e5.grid(row=5,column=1)
e6.grid(row=6,column=1)
e7.grid(row=7,column=1)
e8.grid(row=8,column=1)
e9.grid(row=9,column=1)
```

```
e10.grid(row=10,column=1)
e11.grid(row=11,column=1)

Button(master,text="Predict",command=show_entry).grid()

mainloop()
```



CONNECT WITH ME :

LinkedIn GitHub kaggle Medium

PRASADMJADHAV2