MAJOR PROJECT

LOAN PREDICTION WITH MACHINE LEARNING

>>ALOGRITHM :-

- Start the program.
- Import the packages such as pandas, matplotlib.pyplot, seaborn, sklearn.linear_model and sklearn.metrics.
- First create the dataframe from the dataset taken.
- Now do EDA(exploring data analysis) that includes cleaning the data, information of the data, dropping the empty row and manipulating the data.
- Now add the index after dropping some data, and replace the value of 3+ to 4 in Dependents column.
- Next plot the bar graph of Dependents, Education, Married and Self_Employed to visualize the data.
- Now replace the data of columns Married, Gender, Education, Self_Employed, Property_Area, Loan_Status to 1's and 0's.
- Convert the datatype of Dependents to integer.
- Now get the input values as x and output values as y, where the x is a 2D array and y is 1D array

- Split the data to x_1, x_test, y_1, y_text using sklearn.mosel-selection package.
- Now create the model using LogisticRegression and fit the model with x_1, y_1.
- Now predict the output of x_test.
- At last to know the accuracy use the sklearn.metrics and import the accuracy_score class.
- Now print the accuracy.
- Lets check the model with individual prediction and print the predicted value.
- End the program.

SOURCE CODE

NOTE:-"THE BELOW CODE SHOULD BE EXECUTE ONLY IN COLOBRATORY"

```
#importing the necessary packages
#pandas for getting data and creating the data frame
#matploib.pyplot and seaborn for data visualization
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sb
pv = pd.read_csv("loan_prediction.csv")
pv
#Exploring data analysis (EDA) or cleaning the data
pv.info()
pv.shape
pv.size
```

```
#checking for the empty values
pv.isna().sum()
#there are some empty so droping them all
pv = pv.dropna()
pv.isna().sum()
#changing the index
pv.reset_index(inplace=True)
#checking for the uniques values
pv.Dependents.nunique()
#knowing what are the unique values
pv.Dependents.unique()
pv.Dependents.value_counts()
#there are 345 "0's", 102 "1's", 101 "2's", 51 "3+'s"
#so changing the 3+ t0 4
pv.Dependents = pv.Dependents.replace(to_replace='3+', value=4)
```

```
#checking whether it was changed or not
pv.Dependents.value_counts()
#data visualization
sb.countplot(pv.Dependents)
sb.countplot(pv.Education)
sb.countplot(pv.Married)
sb.countplot(pv.Self_Employed)
sb.countplot(x= "Gender", hue= "Loan_Status", data= pv, color="pink")
#currently the data is the collection of various strings so changing them to the int values
#data manipulation
pv.replace({'Married': {
                'Yes': 1,
                'No': 0
            },
```

```
            'Gender': {
                'Male': 1,
                'Female': 0
            },
            'Education': {
                'Graduate': 1,
                'Not Graduate': 0
            },
    'Self_Employed': {
                'Yes': 1,
                'No': 0
            },
            'Property_Area': {
                'Rural': 0,
                "Urban": 1,
                'Semiurban': 2
```

```
            },
            'Loan_Status': {
                'Y': 1,
                'N': 0
            }
        }, inplace=True
)
pv
pv['Dependents'] = pv["Dependents"].astype('int')
#taking input and output values x, y
x = pv.iloc[:, 2:-1].values
y = pv.iloc[:, -1].values
x
y
```

```
#traning the model
from sklearn.model_selection import train_test_split
x_I, x_test, y_I, y_test = train_test_split(x, y, test_size=0.25, random_state=42)
#checking the shape and size of the X_train, x_test, y_train, y_test
#there will be 75% of x_train and y_train of x, y
#and 25% of x_test and by_test of x, y
x_I.shape
x_I.size
x_test.shape
x_test.size
y_I.shape
y_I.size
y_test.shape
y_test.size
```

```
#Run a SUPERVISED-CLASSIFICATION-LogisticRegressor
#first importing the package of sklearn.linear_model
from sklearn.linear_model import LogisticRegression
model = LogisticRegression()
#fitting the model
model.fit(x_I, y_I)
#now the model was fit
#predicting the output
y_pred = model.predict(x_test)
#checking the output predicted by the model
y_pred
#finding the accuracy of the model
from sklearn.metrics import accuracy_score
accuracy_score(y_pred, y_test)*100
```

- #individual prediction
- model.predict([[0, 0, 0, 1, 0, 3000, 500, 100, 360, 1, 1]])
- model.predict([[1, 1, 1, 1, 0, 2000, 1008, 100, 360, 0, 0,]])

COLAB DRIVE LINK:

- https://colab.research.google.com/drive/1LFXZ5z3G8Pfm5Qrf7ToR7RKqEPPdAn3n

EXPLANATION:-

- Creating a DataFrame-
  - ✓ Get the dataset of loan prediction from Kaggle.
  - ✓ Create a DataFrame from the dataset using pandas library.

- EDA OR Data Cleaning-
  - ✓ Frist get the info of the data and change the data according to the data we wanted.
  - ✓ Getting ride of the row if the columns are are empty.
  - ✓ Changing the string data of some columns to 1’s and 0’s.
  - ✓ Changing the value of 3+ to 4 of Dependents column.
  - ✓ Converting the data of dependents from strings to ‘int’.

- Data Visualization-
  - ✓ Visualizing the data by plotting the bar graphs.

- Accuracy-
  - ✓ The accuracy of the model is 75%.

OUTPUT -OF DATAFRAME

```
[5]  pv = pd.read_csv("loan_prediction.csv")
     pv
```

	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
...	...	...	...	...	...	...	...	...	...	...	...	...	...
609	LP002978	Female	No	0	Graduate	No	2900	0.0	71.0	360.0	1.0	Rural	Y
610	LP002979	Female	Yes	3+	Graduate	No	4106	0.0	40.0	180.0	1.0	Rural	Y
611	LP002983	Male	Yes	1	Graduate	No	8072	240.0	253.0	360.0	1.0	Urban	Y
612	LP002984	Male	Yes	2	Graduate	No	7583	0.0	187.0	360.0	1.0	Urban	Y
613	LP002990	Female	No	0	Graduate	Yes	4583	0.0	133.0	360.0	0.0	Semiurban	N

614 rows × 13 columns

OUTPUT – OF EDA

```
[6]  #Exploring data analysis (EDA) or cleaning the data
     pv.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
```

OUTPUT – OF EDA

```
0s   pv.shape #EDA

     (614, 13)
```

```
0s   [8]  pv.size  #EDA

     7982
```

OUTPUT – OF EDA

```
#checking for the empty values
pv.isna().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
```

OUTPUT – OF EDA

```
[11] pv.isna().sum()
```

0s

```
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
```

OUTPUT – OF EDA

```
[12] #changing the index
     pv.reset_index(inplace=True)
```

```
[13] #checking for the uniques values
     pv.Dependents.nunique()
```

```
4
```

```
[14] #knowing what are the unique values
     pv.Dependents.unique()
```

```
array(['1', '0', '2', '3+'], dtype=object)
```

```
     pv.Dependents.value_counts()
```

```
0     269
1      85
2      85
3+     41
Name: Dependents, dtype: int64
```

OUTPUT – OF EDA

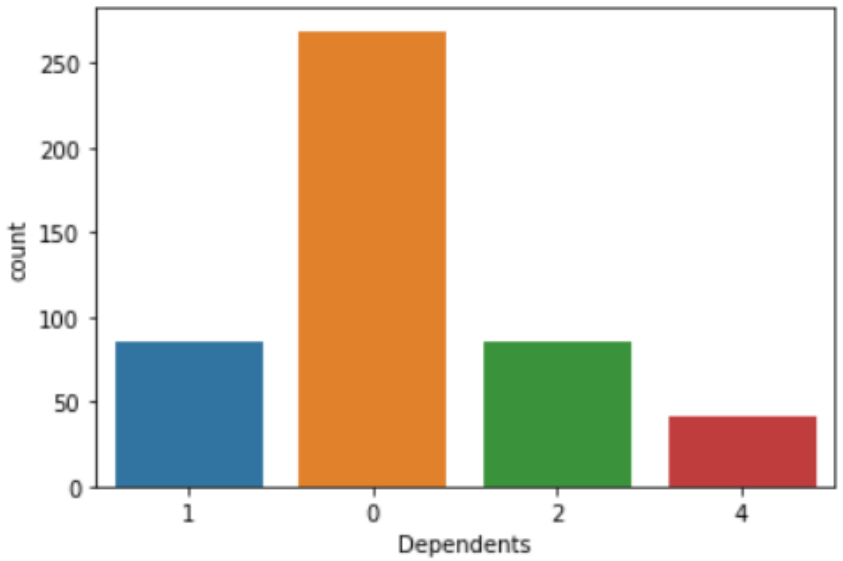
```
[17] #checking whether it was changed or not
     pv.Dependents.value_counts()
```

```
0    269
1     85
2     85
4     41
Name: Dependents, dtype: int64
```

OUTPUT – OF DATA VISUALIZATION

```
#data visualization
sb.countplot(pv.Dependents)
```

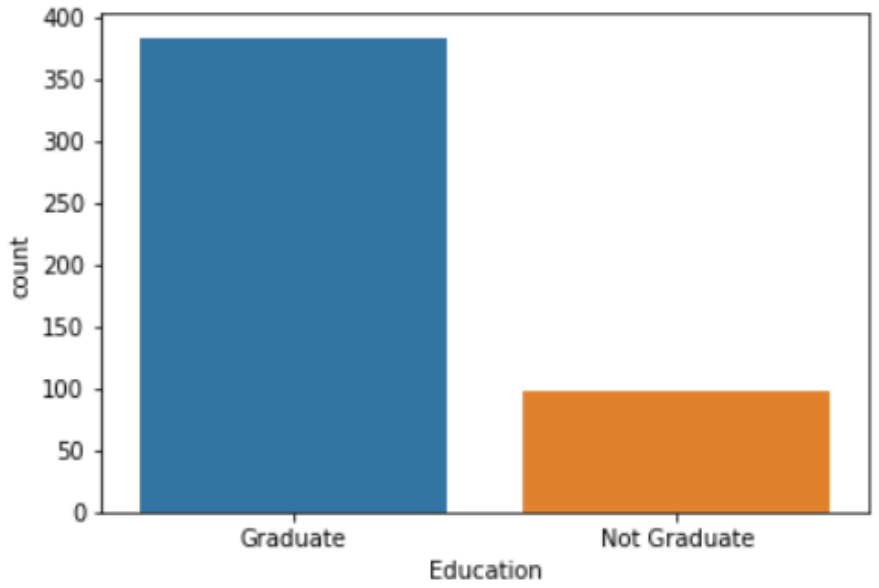
```
/usr/local/lib/python3.7/dist-packages/seaborn/_decorators.py:43:
  FutureWarning
<matplotlib.axes._subplots.AxesSubplot at 0x7fd582744a90>
```



OUTPUT – OF DATA VISUALIZATION

```
sb.countplot(pv.Education)
```

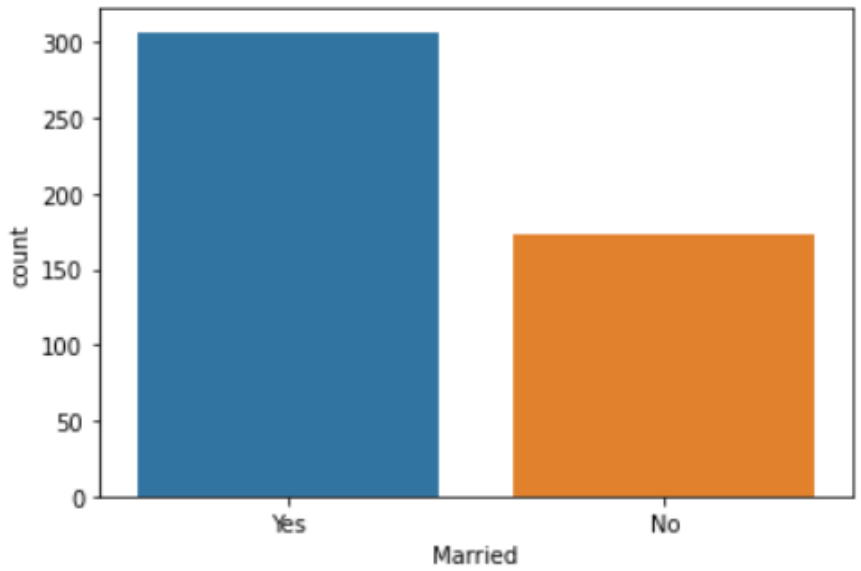
```
/usr/local/lib/python3.7/dist-packages/seaborn/_decorators.py:43: Futur
  FutureWarning
<matplotlib.axes._subplots.AxesSubplot at 0x7fd5824fded0>
```



OUTPUT –OF DATA VISUALIZATION

```
[20] sb.countplot(pv.Married)
```

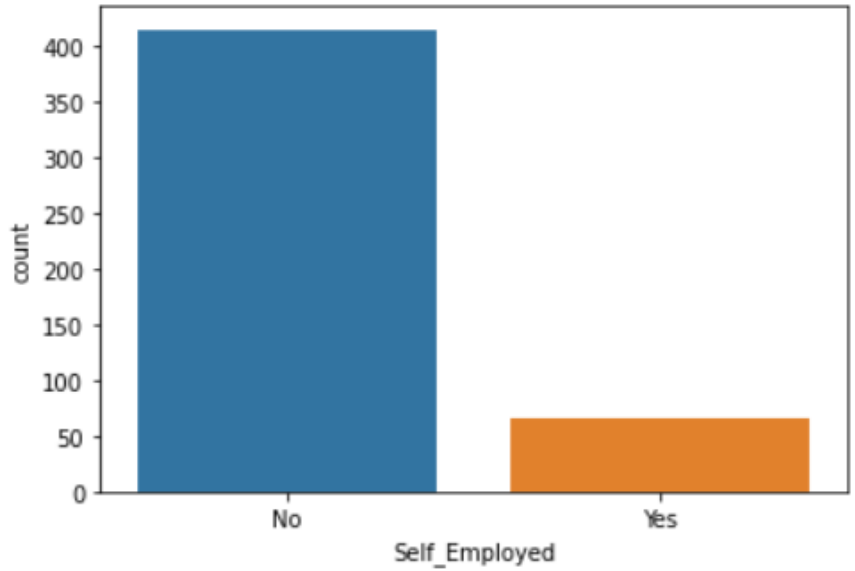
```
/usr/local/lib/python3.7/dist-packages/seaborn/_decorators.py:43: Fu
  FutureWarning
<matplotlib.axes._subplots.AxesSubplot at 0x7fd582029ad0>
```



OUTPUT – OF DATA VISUALIZATION

```
[21] sb.countplot(pv.Self_Employed)
```

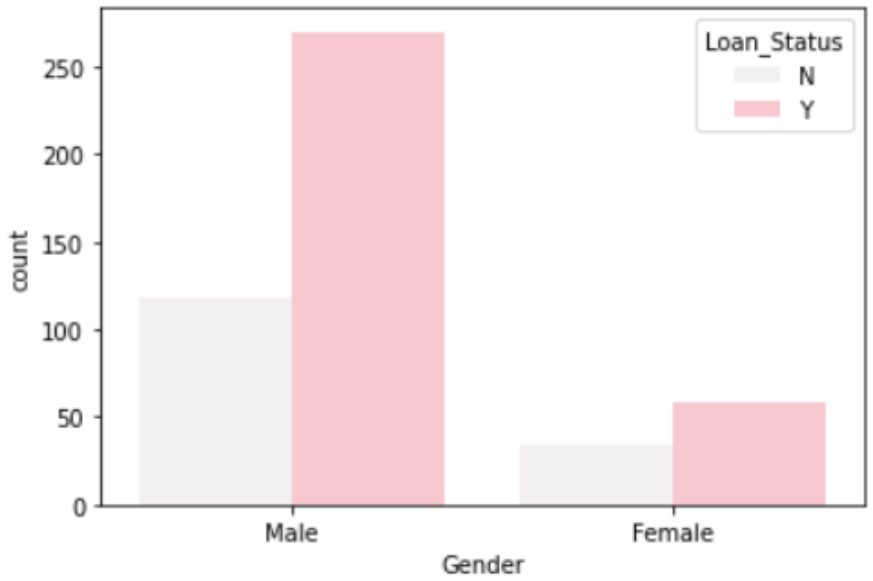
```
/usr/local/lib/python3.7/dist-packages/seaborn/_decorators.py:43: FutureWarni
  FutureWarning
<matplotlib.axes._subplots.AxesSubplot at 0x7fd582025650>
```



OUTPUT – OF DATA VISUALIZATION

```
[22] sb.countplot(x= "Gender", hue= "Loan_Status", data= pv, color="pink")
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fd581f6ea90>
```



OUTPUT – OF INPUT X

```
[29] x #INPUT X

0s

array([[  1.,   1.,   1., ..., 360.,   1.,   0.],
       [  1.,   1.,   0., ..., 360.,   1.,   1.],
       [  1.,   1.,   0., ..., 360.,   1.,   1.],
       ...,
       [  1.,   1.,   1., ..., 360.,   1.,   1.],
       [  1.,   1.,   2., ..., 360.,   1.,   1.],
       [  0.,   0.,   0., ..., 360.,   0.,   2.]])
```

OUTPUT – OF OUTPUT Y

```
[30] y #INPUT Y
```

0s

```
array([0, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 0, 1, 1, 0, 0, 0, 1, 0, 1, 1, 1,
       0, 0, 0, 1, 0, 1, 1, 1, 0, 1, 1, 1, 1, 0, 1, 0, 1, 0, 0, 0, 0, 1,
       0, 1, 1, 1, 1, 0, 1, 0, 0, 1, 1, 0, 0, 1, 1, 1, 0, 0, 0, 0, 1, 0,
       1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1,
       1, 0, 1, 1, 1, 0, 1, 0, 1, 1, 1, 0, 0, 0, 1, 0, 0, 0, 1, 1, 1, 1,
       0, 1, 1, 0, 1, 0, 1, 0, 0, 1, 1, 0, 1, 1, 0, 0, 1, 1, 0, 1, 0, 1,
       0, 0, 0, 1, 1, 0, 1, 0, 0, 1, 0, 1, 1, 0, 1, 1, 0, 0, 1, 1, 0, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 1, 1, 0, 1, 0, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1, 1, 1, 1, 0, 0, 1,
       1, 0, 1, 1, 0, 0, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 0, 0, 0, 1, 1, 1,
       1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 0, 1, 1, 0, 1, 1, 1, 1, 1, 1,
       0, 1, 1, 1, 1, 1, 1, 0, 1, 0, 1, 1, 0, 0, 1, 0, 1, 1, 1, 1, 0, 1,
       0, 1, 1, 1, 0, 1, 1, 0, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 0, 1,
       1, 0, 0, 0, 1, 1, 0, 1, 1, 1, 0, 0, 1, 0, 1, 0, 1, 0, 0, 1, 1, 1,
       1, 1, 1, 0, 1, 1, 0, 1, 0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 0, 0,
       0, 1, 1, 1, 1, 0, 1, 0, 1, 1, 0, 0, 0, 0, 1, 0, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 0, 0, 1, 0, 0, 1, 0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1,
       1, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1,
       1, 1, 0, 1, 1, 0, 1, 1, 0, 1, 0, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1,
       1, 0, 1, 0, 0, 0, 1, 1, 0, 0, 1, 1, 1, 1, 0, 1, 0, 0, 1, 1, 0, 1,
       0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0])
```

OUTPUT – OF ACCURACY & PREDICTION

```
[43] #checking the output predicted by the model
     y_pred
```

```
array([1, 1, 1, 0, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 0, 1, 1, 1, 1,
       1, 0, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 0, 1, 1])
```

```
[44] #finding the accuracy of the model
     from sklearn.metrics import accuracy_score
     accuracy_score(y_pred, y_test)*100
```

```
75.0
```

```
[45] #individual prediction
     model.predict([[0, 0, 0, 1, 0, 3000, 500, 100, 360, 1, 1]])
```

```
array([1])
```

```
[46] model.predict([[1, 1, 1, 1, 0, 2000, 1008, 100, 360, 0, 0,]])
```

```
array([0])
```

THANK YOU