```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
```

```
data = pd.read_csv("/content/train.csv")
data.head()
```

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

Next steps: Generate code with `data` | View recommended plots | New interactive sheet

```
data.tail()
```

	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amount_Term
609	LP002978	Female	No	0	Graduate	No	2900	0.0	71.0	360.0
610	LP002979	Male	Yes	3+	Graduate	No	4106	0.0	40.0	180.0
611	LP002983	Male	Yes	1	Graduate	No	8072	240.0	253.0	360.0
612	LP002984	Male	Yes	2	Graduate	No	7583	0.0	187.0	360.0
613	LP002990	Female	No	0	Graduate	Yes	4583	0.0	133.0	360.0

```
data.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
```

```
data.apply(lambda x: sum(x.isnull()),axis=0)
```

	0
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

```
data['Gender'].value_counts()
```

Gender	count
Male	489
Female	112

```
data.Gender = data.Gender.fillna('Male')
```

```
data['Married'].value_counts()
```

Married	count
Yes	398
No	213

```
data.Married = data.Married.fillna('NO')
```

```
data['Dependents'].value_counts()
```

Dependents	count
0	345
1	102
2	101
3+	51

```
data.replace('3+', 3,inplace=True,limit=None)
data.replace(0, 1,inplace=True,limit=None)
```

```
data['Dependents'] = data['Dependents'].astype(float)
```

```
data.loc[:,'Dependents'].fillna(data['Dependents'].mean() )
```

	Dependents
0	0.0
1	1.0
2	0.0
3	0.0
4	0.0
...	...
609	0.0
610	3.0
611	1.0
612	2.0
613	0.0

614 rows × 1 columns

```
data['Self_Employed'].value_counts()
```

Self_Employed	count
No	500
Yes	82

```
data.Self_Employed = data.Self_Employed.fillna('No')
```

```
data.Self_Employed = data.Self_Employed.fillna('No')
```

```
data['Loan_Amount_Term'].value_counts()
```

Loan_Amount_Term	count
360.0	512
180.0	44
480.0	15
300.0	13
240.0	4
84.0	4
120.0	3
60.0	2
36.0	2
12.0	1

```
data.Loan_Amount_Term = data.Loan_Amount_Term.fillna(360.0)
```

```
data.Credit_History = data.Credit_History.fillna(1.0)
```

```
data.apply(lambda x: sum(x.isnull()),axis=0)
```

	0
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

```
x = data[['Loan_ID','Gender','Married','Dependents','Education','Self_Employed','ApplicantIncome','CoapplicantIncome','LoanAmount','Loan_Amc
y = data['Loan_Status']
```

```
from sklearn.model_selection import train_test_split
```

```
X_train, X_test, y_train, y_test = train_test_split(
    x, y, test_size=0.2, random_state=0)
```

```
X_train
```

	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amount_Term
90	LP001316	Male	Yes	0	Graduate	No	2958	2900.0	131.0	360.0
533	LP002729	Male	No	1	Graduate	No	11250	0.0	196.0	360.0
452	LP002448	Male	Yes	0	Graduate	No	3948	1733.0	149.0	360.0
355	LP002144	Female	No	NaN	Graduate	No	3813	0.0	116.0	180.0
266	LP001877	Male	Yes	2	Graduate	No	4708	1387.0	150.0	360.0
...	...	...	...	...	...	...	...	...	...	...
277	LP001904	Male	Yes	0	Graduate	No	3103	1300.0	80.0	360.0
9	LP001020	Male	Yes	1	Graduate	No	12841	10968.0	349.0	360.0
359	LP002160	Male	Yes	3+	Graduate	No	5167	3167.0	200.0	360.0
192	LP001657	Male	Yes	0	Not Graduate	No	6033	0.0	160.0	360.0
559	LP002804	Female	Yes	0	Graduate	No	4180	2306.0	182.0	360.0

491 rows × 12 columns

Next steps: Generate code with X_train View recommended plots New interactive sheet

```
X_test
```

	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amount_Term
454	LP002453	Male	No	0	Graduate	Yes	7085	0.0	84.0	360.0
52	LP001164	Female	No	0	Graduate	No	4230	0.0	112.0	360.0
536	LP002734	Male	Yes	0	Graduate	No	6133	3906.0	324.0	360.0
469	LP002505	Male	Yes	0	Graduate	No	4333	2451.0	110.0	360.0
55	LP001194	Male	Yes	2	Graduate	No	2708	1167.0	97.0	360.0
...	...	...	...	...	...	...	...	...	...	...
337	LP002112	Male	Yes	2	Graduate	Yes	2500	4600.0	176.0	360.0
376	LP002219	Male	Yes	3+	Graduate	No	8750	4996.0	130.0	360.0
278	LP001907	Male	Yes	0	Graduate	No	14583	0.0	436.0	360.0
466	LP002500	Male	Yes	3+	Not Graduate	No	2947	1664.0	70.0	180.0
303	LP001977	Male	Yes	1	Graduate	No	1625	1803.0	96.0	360.0

123 rows × 12 columns

Next steps: Generate code with `X_test` View recommended plots New interactive sheet

```
from sklearn.preprocessing import LabelEncoder
labelencoder_X = LabelEncoder()
```

```
from sklearn.preprocessing import LabelEncoder
labelencoder_X = LabelEncoder()

for i in range(0, 5):
    # Access column 'i' using .iloc for integer-based indexing
   X_train.iloc[:, i] = labelencoder_X.fit_transform(X_train.iloc[:, i])

#Access column 10 using .iloc
X_train.iloc[:, 10] = labelencoder_X.fit_transform(X_train.iloc[:, 10])


labelencoder_y = LabelEncoder()
y_train = labelencoder_y.fit_transform(y_train)
```

```
X_train
```

	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amount_Term	Cr
90	67	1	1	0	0	No	2958	2900.0	131.0	360.0	
533	426	1	0	1	0	No	11250	0.0	196.0	360.0	
452	360	1	1	0	0	No	3948	1733.0	149.0	360.0	
355	287	0	0	4	0	No	3813	0.0	116.0	180.0	
266	210	1	1	2	0	No	4708	1387.0	150.0	360.0	
...	...	...	...	...	...	...	...	...	...	...	
277	220	1	1	0	0	No	3103	1300.0	80.0	360.0	
9	6	1	1	1	0	No	12841	10968.0	349.0	360.0	
359	289	1	1	3	0	No	5167	3167.0	200.0	360.0	
192	156	1	1	0	1	No	6033	0.0	160.0	360.0	
559	445	0	1	0	0	No	4180	2306.0	182.0	360.0	

491 rows × 12 columns

Next steps: Generate code with `X_train` View recommended plots New interactive sheet

```
y_train
```

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

```
from sklearn.preprocessing import LabelEncoder, OneHotEncoder
labelencoder_X = LabelEncoder()
for i in range(0, 5):
    # Use .iloc for integer-based location selection in DataFrames
     X_test.iloc[:,i] = labelencoder_X.fit_transform(X_test.iloc[:,i])
X_test.iloc[:,10] = labelencoder_X.fit_transform(X_test.iloc[:,10])
#Encoding the Dependent Variable
labelencoder_y = LabelEncoder()
y_test = labelencoder_y.fit_transform(y_test)
```

```
X_test
```

	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amount_Term	Cr
454	93	1	0	0	0	Yes	7085	0.0	84.0	360.0	
52	15	0	0	0	0	No	4230	0.0	112.0	360.0	
536	108	1	1	0	0	No	6133	3906.0	324.0	360.0	
469	96	1	1	0	0	No	4333	2451.0	110.0	360.0	
55	16	1	1	2	0	No	2708	1167.0	97.0	360.0	
...	...	...	...	...	...	...	...	...	...	...	
337	62	1	1	2	0	Yes	2500	4600.0	176.0	360.0	
376	74	1	1	3	0	No	8750	4996.0	130.0	360.0	
278	57	1	1	0	0	No	14583	0.0	436.0	360.0	
466	95	1	1	3	1	No	2947	1664.0	70.0	180.0	
303	59	1	1	1	0	No	1625	1803.0	96.0	360.0	

123 rows × 12 columns

```
from sklearn.preprocessing import LabelEncoder, OneHotEncoder, StandardScaler
import pandas as pd
import numpy as np

# Assuming X_train has categorical columns at indices 0, 1, 2, 3, 4 and 10
# Correct this if 10 is not a valid column index for your data
categorical_cols = [0, 1, 2, 3, 4]  # Removed 10 as it's causing the error

# Convert X_train and X_test back to DataFrames if they are NumPy arrays
X_train = pd.DataFrame(X_train)
X_test = pd.DataFrame(X_test)

# Create a LabelEncoder object
labelencoder = LabelEncoder()

# Apply Label Encoding to categorical columns in X_train and handle unknown values in X_test
```

```
for col in categorical_cols:
    # Fit on the combined unique values from both train and test sets
    all_values = pd.concat([X_train.iloc[:, col], X_test.iloc[:, col]], ignore_index=True).astype(str).unique()
    labelencoder.fit(all_values)

    # Transform the columns
    X_train.iloc[:, col] = labelencoder.transform(X_train.iloc[:, col].astype(str))
    X_test.iloc[:, col] = labelencoder.transform(X_test.iloc[:, col].astype(str))

# Convert all columns to numeric, coercing errors to NaN
# This will replace any remaining string values that couldn't be converted with NaN
for col in X_train.columns:
    X_train[col] = pd.to_numeric(X_train[col], errors='coerce')
for col in X_test.columns:
    X_test[col] = pd.to_numeric(X_test[col], errors='coerce')

# Impute NaN values if any (you can choose your preferred imputation strategy)
from sklearn.impute import SimpleImputer
imputer = SimpleImputer(strategy='mean')  # Or use 'median', 'most_frequent', etc.
X_train = imputer.fit_transform(X_train)
X_test = imputer.transform(X_test)

# Now apply StandardScaler
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test) # Use transform, not fit_transform, for X_test
```

LOGISTIC REGRESSION ALGORITHM

```
from sklearn.linear_model import LogisticRegression
classifier = LogisticRegression(random_state = 0)
classifier.fit(X_train, y_train)
```

```
LogisticRegression
LogisticRegression(random_state=0)
```

```
y_pred = classifier.predict(X_test)
```

```
y_pred
```

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

```
from sklearn import metrics
print('The accuracy of Logistic Regression is: ', metrics.accuracy_score(y_pred, y_test))
```

```
The accuracy of Logistic Regression is:  0.8373983739837398
```

```
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, y_pred)
```

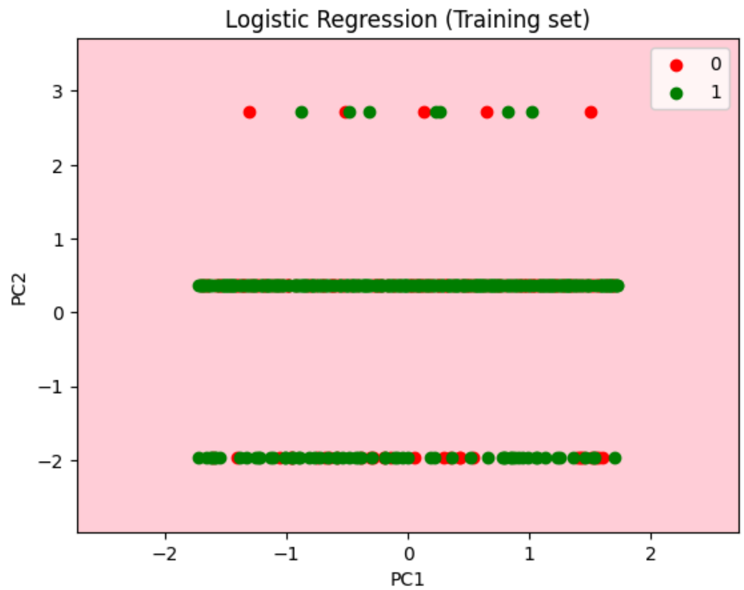
```
cm
```

```
array([[15, 18],
       [ 2, 88]])
```

```
# Visualising the Training set results
from matplotlib.colors import ListedColormap
X_set, y_set = X_train, y_train

# Assuming X_train has 10 features, and you want to visualize the decision boundary
# based on the first two features, you should select those features:
X1, X2 = np.meshgrid(np.arange(start = X_set[:, 0].min() - 1, stop = X_set[:, 0].max() + 1, step = 0.01),
                     np.arange(start = X_set[:, 1].min() - 1, stop = X_set[:, 1].max() + 1, step = 0.01))

# Create input data for prediction with the correct number of features (10)
```

```
# We will use X1 and X2 for the first two features and fill the rest with zeros
# Adjust the number of zeros (8 in this case) to match the remaining features
num_features = 10  # Assuming your model was trained on 10 features
input_data = np.zeros((X1.ravel().shape[0], num_features))
input_data[:, 0] = X1.ravel()
input_data[:, 1] = X2.ravel()

plt.contourf(X1, X2, classifier.predict(input_data).reshape(X1.shape),
             alpha = 0.75, cmap = ListedColormap(('pink', 'lightgreen')))
plt.xlim(X1.min(), X1.max())
plt.ylim(X2.min(), X2.max())
for i, j in enumerate(np.unique(y_set)):
    plt.scatter(X_set[y_set == j, 0], X_set[y_set == j, 1],
                c = ListedColormap(('red', 'green'))(i), label = j)
plt.title('Logistic Regression (Training set)')
plt.xlabel('PC1')
plt.ylabel('PC2')
plt.legend()
plt.show()
```

```
<ipython-input-74-f084ff516800>:23: UserWarning: *c* argument looks like a single numeric RGB or RGBA sequence, which should be avoided
  plt.scatter(X_set[y_set == j, 0], X_set[y_set == j, 1],
```




```
from matplotlib.colors import ListedColormap
X_set, y_set = X_test, y_test
X1, X2 = np.meshgrid(np.arange(start = X_set[:, 0].min() - 1, stop = X_set[:, 0].max() + 1, step = 0.01),
                     np.arange(start = X_set[:, 1].min() - 1, stop = X_set[:, 1].max() + 1, step = 0.01))

# Assuming your model was trained on 10 features
num_features = 10
# Create input data for prediction with the correct number of features (10)
input_data = np.zeros((X1.ravel().shape[0], num_features))
input_data[:, 0] = X1.ravel()
input_data[:, 1] = X2.ravel()

# Now use input_data for prediction:
plt.contourf(X1, X2, classifier.predict(input_data).reshape(X1.shape),
             alpha = 0.75, cmap = ListedColormap(('pink', 'lightgreen')))
plt.xlim(X1.min(), X1.max())
plt.ylim(X2.min(), X2.max())
for i, j in enumerate(np.unique(y_set)):
    plt.scatter(X_set[y_set == j, 0], X_set[y_set == j, 1],
                c = ListedColormap(('red', 'green'))(i), label = j)
plt.title('Logistic Regression (Test set)')
plt.xlabel('PC1')
plt.ylabel('PC2')
plt.legend()
plt.show()
```

```
<ipython-input-77-c97a0b4a767b>:19: UserWarning: *c* argument looks like a single numeric RGB or RGBA sequence, which should be avoided
  plt.scatter(X_set[y_set == j, 0], X_set[y_set == j, 1],
```