You can import your own data into Colab notebooks from your Google Drive account, including from spreadsheets, as well as from Github and many other sources. To learn more about importing data, and how Colab can be used for data science, see the links below under Working with Data.

Business Case: LoanTap Logistic Regression

Problem Statement: Given a set of attributes for an Individual, determine if a credit line should be extended to them. If so, what should the repayment terms be in business recommendations?

```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
```

```
data = pd.read_csv("https://d2beiqkhq929f0.cloudfront.net/public_assets/assets/000/003/549/original/logistic_regression.csv?1651045921")
```

```
data.head()
```

	loan_amnt	term	int_rate	installment	grade	sub_grade	emp_title	emp_length	home_ownership	annual_inc	...	open_acc	pub
0	10000.0	36 months	11.44	329.48	B	B4	Marketing	10+ years	RENT	117000.0	...	16.0	
1	8000.0	36 months	11.99	265.68	B	B5	Credit analyst	4 years	MORTGAGE	65000.0	...	17.0	
2	15600.0	36 months	10.49	506.97	B	B3	Statistician	< 1 year	RENT	43057.0	...	13.0	
3	7200.0	36 months	6.49	220.65	A	A2	Client Advocate	6 years	RENT	54000.0	...	6.0	
4	24375.0	60 months	17.27	609.33	C	C5	Destiny Management Inc.	9 years	MORTGAGE	55000.0	...	13.0	

5 rows × 27 columns

```
# features & label
data.columns
```

```
Index(['loan_amnt', 'term', 'int_rate', 'installment', 'grade', 'sub_grade',
       'emp_title', 'emp_length', 'home_ownership', 'annual_inc',
       'verification_status', 'issue_d', 'loan_status', 'purpose', 'title',
       'dti', 'earliest_cr_line', 'open_acc', 'pub_rec', 'revol_bal',
       'revol_util', 'total_acc', 'initial_list_status', 'application_type',
       'mort_acc', 'pub_rec_bankruptcies', 'address'],
      dtype='object')
```

```
# size of data
data.shape
```

```
(396030, 27)
```

```
# Type of data
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 396030 entries, 0 to 396029
Data columns (total 27 columns):
 #   Column                Non-Null Count   Dtype  
---  ------                --------------   -----  
 0   loan_amnt             396030 non-null  float64
 1   term                  396030 non-null  object 
 2   int_rate              396030 non-null  float64
 3   installment           396030 non-null  float64
 4   grade                 396030 non-null  object 
 5   sub_grade             396030 non-null  object 
 6   emp_title             373103 non-null  object 
 7   emp_length            377729 non-null  object 
 8   home_ownership        396030 non-null  object 
 9   annual_inc            396030 non-null  float64
 10  verification_status   396030 non-null  object 
 11  issue_d               396030 non-null  object 
```

```
 12  loan_status           396030 non-null  object
 13  purpose               396030 non-null  object
 14  title                 394275 non-null  object
 15  dti                   396030 non-null  float64
 16  earliest_cr_line      396030 non-null  object
 17  open_acc              396030 non-null  float64
 18  pub_rec               396030 non-null  float64
 19  revol_bal             396030 non-null  float64
 20  revol_util            395754 non-null  float64
 21  total_acc             396030 non-null  float64
 22  initial_list_status   396030 non-null  object
 23  application_type      396030 non-null  object
 24  mort_acc              358235 non-null  float64
 25  pub_rec_bankruptcies  395495 non-null  float64
 26  address               396030 non-null  object
dtypes: float64(12), object(15)
memory usage: 81.6+ MB
```

```
# Checking null values

data.isnull().sum()
```

```
loan_amnt                   0
term                        0
int_rate                    0
installment                 0
grade                       0
sub_grade                   0
emp_title               22927
emp_length              18301
home_ownership              0
annual_inc                  0
verification_status         0
issue_d                     0
loan_status                 0
purpose                     0
title                    1755
dti                         0
earliest_cr_line            0
open_acc                    0
pub_rec                     0
revol_bal                   0
revol_util                276
total_acc                   0
initial_list_status         0
application_type            0
mort_acc                37795
pub_rec_bankruptcies      535
address                     0
dtype: int64
```

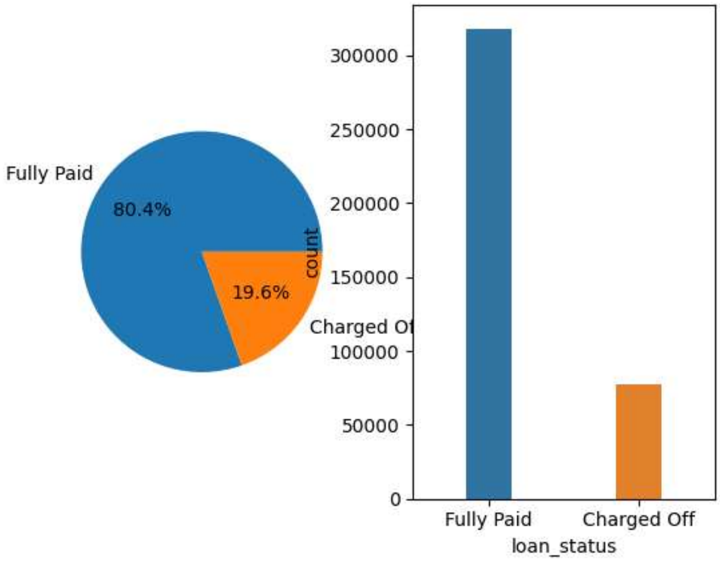
```
# Data cleaning
#----> We’ll remove columns with only one unique value because their variance will be 0 and they won’t help us anticipate anything.

features_nuniques = {}
for i in data.columns:
  features_nuniques[i] = data[i].nunique()

print(features_nuniques)
```

```
{'loan_amnt': 1397, 'term': 2, 'int_rate': 566, 'installment': 55706, 'grade': 7, 'sub_grade': 35, 'emp_title': 173105, 'emp_length
```

```
## Since emp_title has null values more than 5%(5.7 %) and it does not influence the target variable, it is removed

data.drop('emp_title', axis = 1, inplace = True)

## Since only 0.06%, 0.4% , 0.1 & 4.6% of data are null values in revol_utile,title, pub_rec_bankruptcies & employee_length respectively,
## Eventhough 9.5% of data are missed in mort_account,it is imputed with it's mean value as it is influence the target data
data['emp_length'].replace({"10+ years":"10 year", "< 1 year":"0 year", None:np.nan}, inplace = True)

data["emp_length"] = data["emp_length"].str.split(" ").str[0]

data["emp_length"] = data["emp_length"].astype('float64')

for i in ["emp_length","revol_util","mort_acc"]:
  data[i].fillna(data[i].mean(), inplace = True)

for j in ["title","pub_rec_bankruptcies"]:

   data[j].fillna(data[j].mode()[0], inplace = True)
```

```
# Exploratory data analysis
#--->  target variable's(loan_status) distribution in the dataset
plt.subplot(1,2,1)
labels = ["Fully Paid","Charged Off"]
plt.pie(data["loan_status"].value_counts(), labels = labels,autopct='%1.1f%%')
plt.subplot(1,2,2)
sns.countplot(data = data, x = "loan_status", width = 0.3)
```

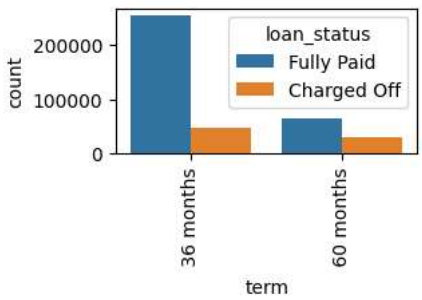
```
<Axes: xlabel='loan_status', ylabel='count'>
```

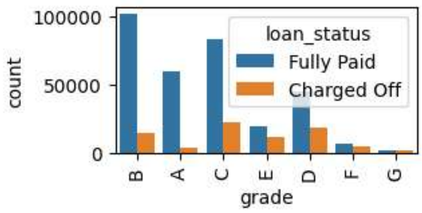


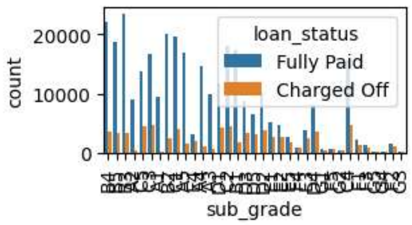

```
# checking correlation between the variables
plt.figure(figsize=(15,8))
sns.heatmap(data.corr(), annot = True, linewidths = 2.5)
```

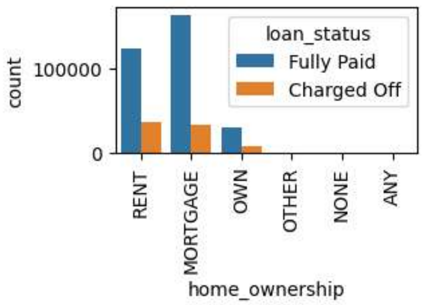
```
<ipython-input-99-490bb3ea7128>:3: FutureWarning: The default value of numeric_only in DataFrame.corr is deprecated. In a future ver
  sns.heatmap(data.corr(), annot = True, linewidths = 2.5)
```

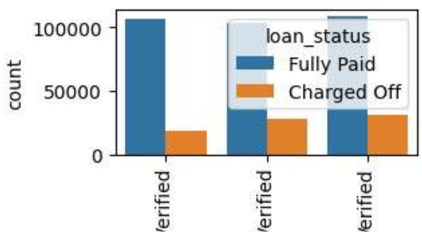
```
# Let’s visualize if there is any relationship between the target variable and other variables
catedgarical_features = ["term", 'grade', 'sub_grade', 'home_ownership', 'verification_status', 'application_type']

for i in range(1, len(catedgarical_features)+1):

  plt.subplot(3,2,i)
  plt.xticks(rotation = 90)
  sns.countplot(data = data, x = catedgarical_features[i-1], hue =  "loan_status")
  plt.show()
```





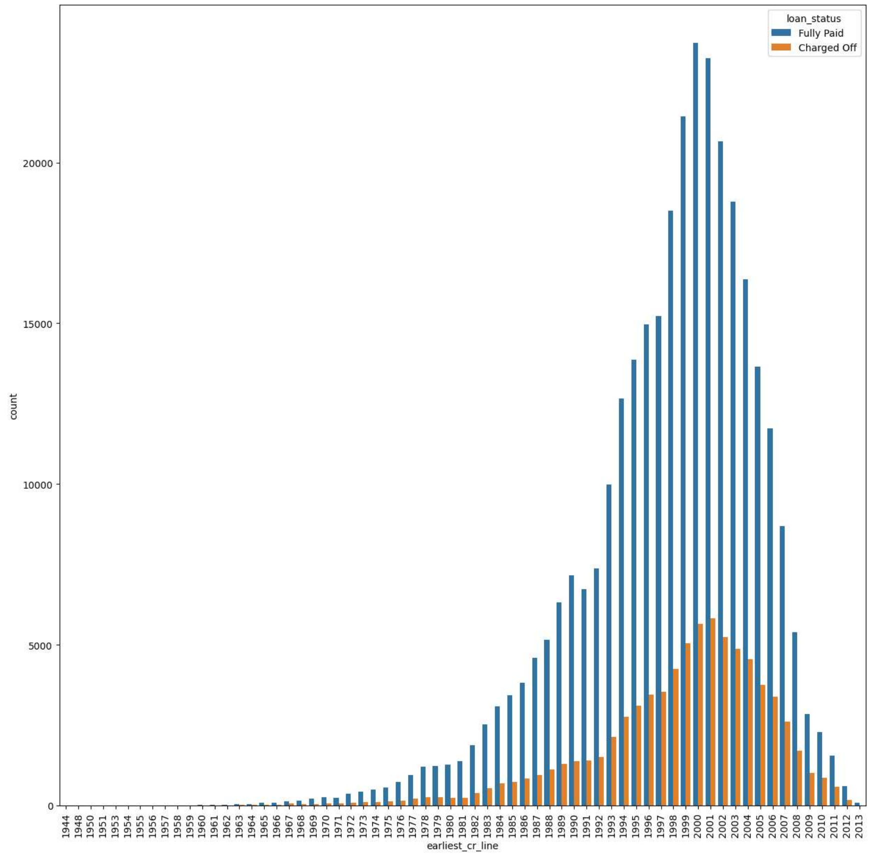






```
# Checkin impact of earliest_cr_line o target variable
data["earliest_cr_line"] = pd.to_datetime(data["earliest_cr_line"])
plt.figure(figsize = (15,15))
plt.xticks(rotation = 90)
sns.countplot(data = data, x = data["earliest_cr_line"].dt.year, hue = "loan_status")
```

<Axes: xlabel='earliest_cr_line', ylabel='count'>




```
# Chisquare test- does earliest_cr_line really influence the loan status
year_loan_status = pd.crosstab(index = data["loan_status"], columns = data["earliest_cr_line"].dt.year)

year_loan_status
```

earliest_cr_line	1944	1948	1950	1951	1953	1954	1955	1956	1957	1958	...	20
loan_status												
Charged Off	1	0	0	1	0	0	3	3	2	4	...	45
Fully Paid	0	1	3	2	2	4	6	4	5	8	...	163

```
from scipy.stats import chi2_contingency

chi_stat, p_value, df, expe_frequecy = chi2_contingency(year_loan_status)
print(f"p_value is {p_value}")
if p_value<0.05:
  print("earliest_cr_line has impact on loan status")
else:
  print("Has no impact")
```

```
p_value is 6.1047661863873966e-198
earliest_cr_line has impact on loan status
```

```
# Feature engineering
#---> Converting categarical target var into numerical

def cat_nume_target(x):
  if x == "Fully Paid":
    return 1
  else:
    return 0
data["loan_status"] = data["loan_status"].apply(cat_nume_target)
```

```
#---> Converting categarical features into numerical
# -----> term is nothing but the number of payments on the loan(36 months or 60 months)
data["term"] = data["term"].apply(lambda x: 36 if x == "36 months" else 60)

# ------> Since grade & subgrade have order, ther are encoded using label encoder # verification type does not influence the target varia
from sklearn.preprocessing import OrdinalEncoder
cols = ["grade","sub_grade","initial_list_status"]
ordinal_encoder = OrdinalEncoder()
data[cols] = ordinal_encoder.fit_transform(data[cols])
#  earliest_cr_line
data["earliest_cr_line"] = data["earliest_cr_line"].dt.year.astype(int)
```

```
#-----> since home_ownership is not ordinal and only 5 categarical values it has, one hot encoding is used to encode  it

from sklearn.preprocessing import OneHotEncoder
col = ["home_ownership"]
ohe = OneHotEncoder(handle_unknown='ignore', sparse = False)

ohe_new = pd.DataFrame(ohe.fit_transform(data[col]))

# One-hot encoding removed index; put it back
ohe_new.index = data.index

# Remove categorical columns (will replace with one-hot encoding)
data = data.drop(col, axis=1)

# Add one-hot encoded columns to numerical features
data = pd.concat([data, ohe_new], axis=1)

# Ensure all columns have string type

data.columns = data.columns.astype(str)
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/preprocessing/_encoders.py:868: FutureWarning: `sparse` was renamed to `sparse_outpu
  warnings.warn(
```

```
# Since verification_status,issue_d, purpose, title ,application_type & address do not have influemce on target variable, hence removed.
data.drop(["verification_status","issue_d", "purpose", "title" ,"application_type", "address"], axis = 1, inplace = True )
```

```
# train test split
x = data.drop("loan_status", axis = 1)
y = data["loan_status"]

from sklearn.model_selection import train_test_split

x_train_cv, x_test, y_train_cv, y_test = train_test_split(x,y, test_size = 0.2, random_state = 1)
```

```
x_train, x_val, y_train, y_val = train_test_split(x_train_cv,y_train_cv, test_size = 0.2, random_state = 1)
```

```
x_train.shape
```

```
(253459, 24)
```

```
# Scaling all features in order to avoid dominanace of any featrures due to it's high range

from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
scaler.fit(x_train)

x_train = scaler.transform(x_train)
x_val = scaler.transform(x_val)
x_test = scaler.transform(x_test)
```

```
# lets train a logistic regression model

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

```
▾ LogisticRegression
LogisticRegression()
```

```
model.coef_
```

```
array([[-5.07647543e-01,  0.00000000e+00,  2.34026322e-01,
         3.91174075e-01, -4.52658264e-04, -7.77469943e-01,
         1.18865997e-02,  2.04303398e-01, -5.33864276e-01,
         1.95553207e-02, -1.04545576e-01, -6.30875796e-02,
         7.12633515e-02, -8.18671391e-02,  1.00579255e-01,
        -1.04558597e-04,  7.29136053e-02,  2.58587365e-02,
         2.29695807e-02,  6.01427298e-02, -2.03460504e-03,
         4.87801806e-03, -7.41563208e-03, -5.71467588e-02]])
```

```
model.intercept_
```

```
array([1.57755842])
```

```
# prediction on train data
model.predict(x_train)
```

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

```
# definng accuracy on train data
def accuracy(y_true, y_pred):
  return np.sum(y_true == y_pred)/(y_true.shape[0])
```

```
# Accuracy on train data
accuracy(y_train, model.predict(x_train))
```

```
0.8053491886261683
```

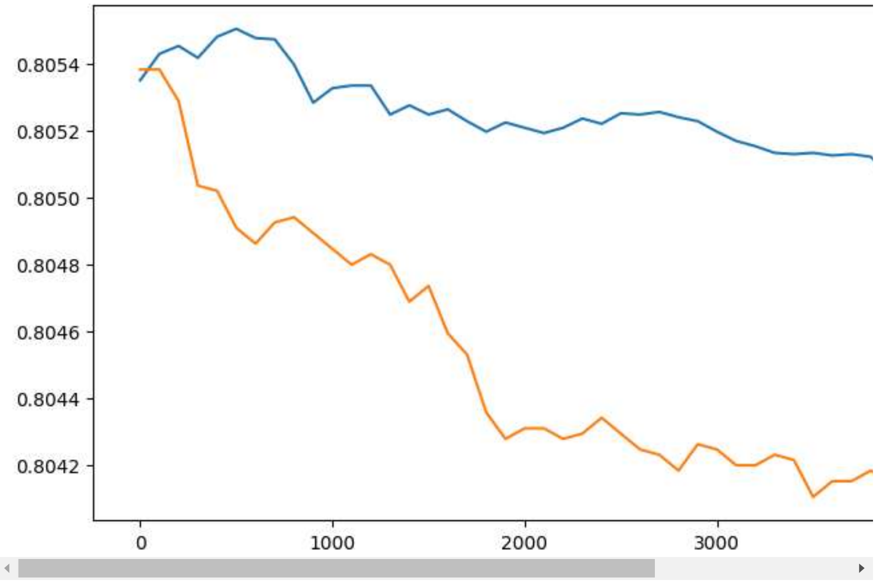
```
# Accuracy on test data
accuracy(y_test, model.predict(x_test))
```

```
0.8037269903795167
```

```
# Hyper parameter tuning(regularization strength)

from sklearn.pipeline import make_pipeline

train_scores = []
val_scores = []

scaler = StandardScaler()
for la in np.arange(0.01, 5000,100):
  scaled_lr = make_pipeline(scaler, LogisticRegression(C = 1/la))
  scaled_lr.fit(x_train, y_train)
  train_score = accuracy(y_train, scaled_lr.predict(x_train))
  val_score = accuracy(y_val, scaled_lr.predict(x_val))
  train_scores.append(train_score)
  val_scores.append(val_score)
```

```
plt.figure(figsize = (10,5))
plt.plot(list(np.arange(0.01, 5000, 100)), train_scores, label= 'train')
plt.plot(list(np.arange(0.01, 5000, 100)), val_scores, label= 'val')
plt.legend(loc = 'lower right')
```

```
<matplotlib.legend.Legend at 0x79295164bf10>
```




```
# As shown above, best regularization strength(lambda) is around 60
model2 = LogisticRegression(C = 1/70)
model2.fit(x_train,y_train)
```

```
LogisticRegression
LogisticRegression(C=0.014285714285714285)
```

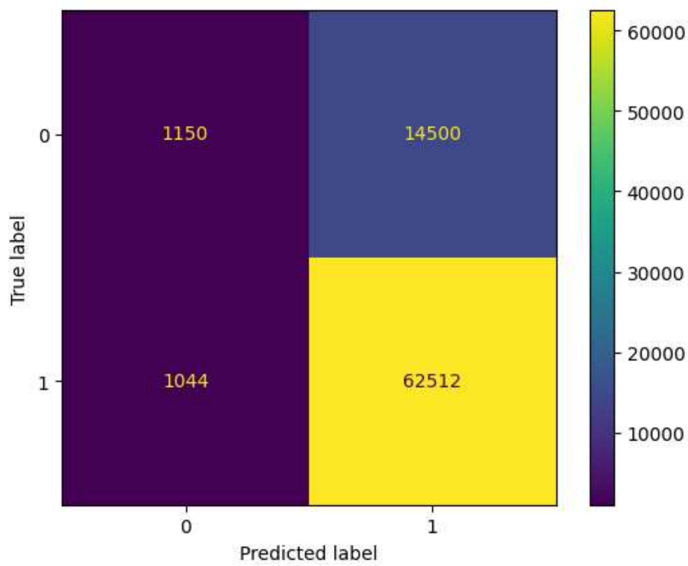
```
accuracy(y_train, model2.predict(x_train))
```

```
0.8054399330858245
```

```
accuracy(y_test, model2.predict(x_test))
```

```
0.8037522409918441
```

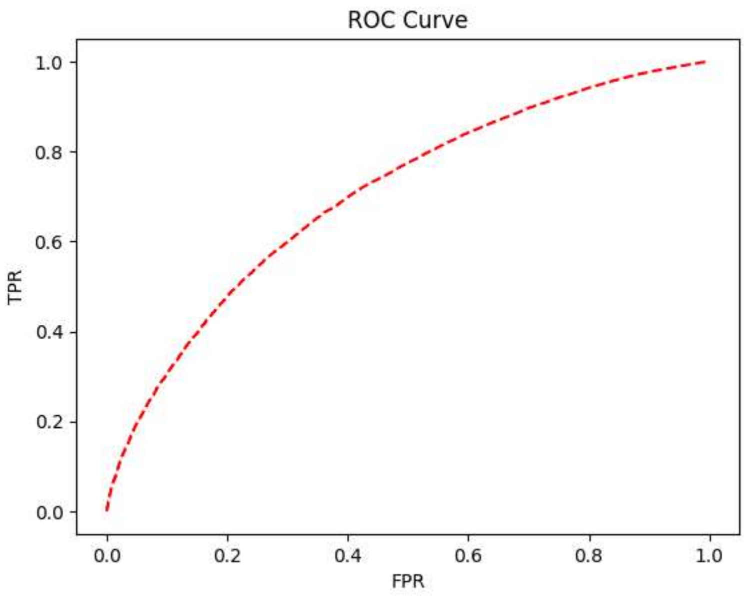
```
# Evaluation of the performance of the model
#---> Confusionmatrix
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay
from matplotlib import pyplot as plt
y_pred = model2.predict(x_test)
conf_matrix = confusion_matrix(y_test,y_pred)
ConfusionMatrixDisplay(conf_matrix).plot()
plt.show()
```



```
from sklearn.metrics import recall_score, precision_score

print(f"recall is {recall_score(y_test, y_pred)}")
print("---------")
print(f"precision is {precision_score(y_test, y_pred)}")
print("----------")
print(f"accuracy is {accuracy(y_test, model2.predict(x_test))}")
```

```
recall is 0.9835735414437661
---------
precision is 0.8117176543915234
----------
accuracy is 0.8037522409918441
```

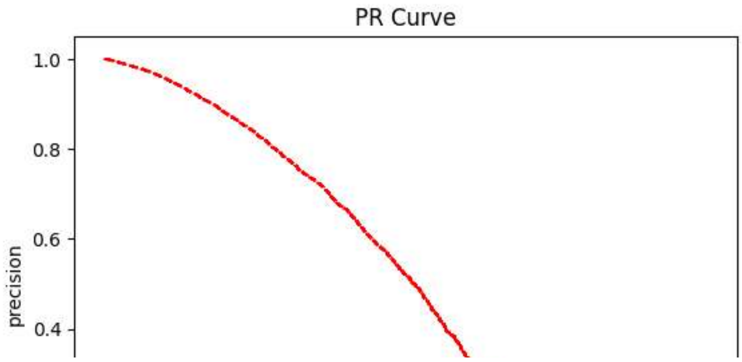
```
# Recall- precision tradeoff--- ROC curve
#----->For this particular case, precision is more important than recall

from sklearn.metrics import roc_curve,roc_auc_score, recall_score, precision_score

prob = model2.predict_proba(x_test)
proba1 = prob[:,1]

fpr, tpr, thresh = roc_curve(y_test, proba1)
plt.plot(fpr,tpr,"--", color = "red")
plt.title("ROC Curve")
plt.xlabel('FPR')
plt.ylabel("TPR")
plt.show()
```




```
# Area under the ROC curve
roc_auc_score(y_test, proba1)
```

```
0.7053091072912581
```

```
# Precision recall curve

from sklearn.metrics import precision_recall_curve

precision, recall, thresh = precision_recall_curve(y_test, proba1)

plt.plot(precision, recall,"--", color = "red")
plt.title("PR Curve")
plt.ylabel('precision')
plt.xlabel("recall")
plt.show()
```




```
# Area under the PR curve curve
from sklearn.metrics import auc

auc(recall, precision)
```

```
0.8986814205625107
```