```
# importing libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
warnings.filterwarnings('ignore')

!gdown 1ZPYj7CZCfxntE8p2Lze_4QO4MyEOy6_d

Downloading...
From: https://drive.google.com/uc?id=1ZPYj7CZCfxntE8p2Lze_4QO4MyEOy6_d
To: /content/logistic_regression.csv
100% 100M/100M [00:00<00:00, 132MB/s]

# read the data
df_main = pd.read_csv('logistic_regression.csv')
df_main.head()

{"type":"dataframe","variable_name":"df_main"}
```

EDA

```
# getting the no of rows and columns
df_main.shape

(396030, 27)

# getting the info of the dataset
df_main.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                 394274 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 the null values in the dataset
df_main.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                    1756
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
```

As we can observe that, some features have huge numbers of null values.

- We will drop the features with huge no of null values

```
df_main.drop(['emp_title', 'emp_length', 'title', 'revol_util',
'mort_acc', 'pub_rec_bankruptcies'], axis=1, inplace=True)

df_main.shape

(396030, 21)

# checking the unique values
df_main.nunique()

loan_amnt                  1397
term                          2
int_rate                    566
installment               55706
grade                         7
sub_grade                    35
home_ownership                6
annual_inc                27197
verification_status           3
issue_d                     115
loan_status                   2
purpose                      14
dti                        4262
earliest_cr_line            684
open_acc                     61
pub_rec                      20
revol_bal                 55622
total_acc                   118
initial_list_status           2
application_type              3
address                  393700
dtype: int64
```

Lets see the number of samples of terget variable

```
# loan_status is the terget variable
df_main['loan_status'].value_counts()

loan_status
Fully Paid     318357
Charged Off     77673
Name: count, dtype: int64
```

Observe

The samples for both the values of y are not the same

- Showing that the data is imbalanced or not have equal number of samples for both the cases

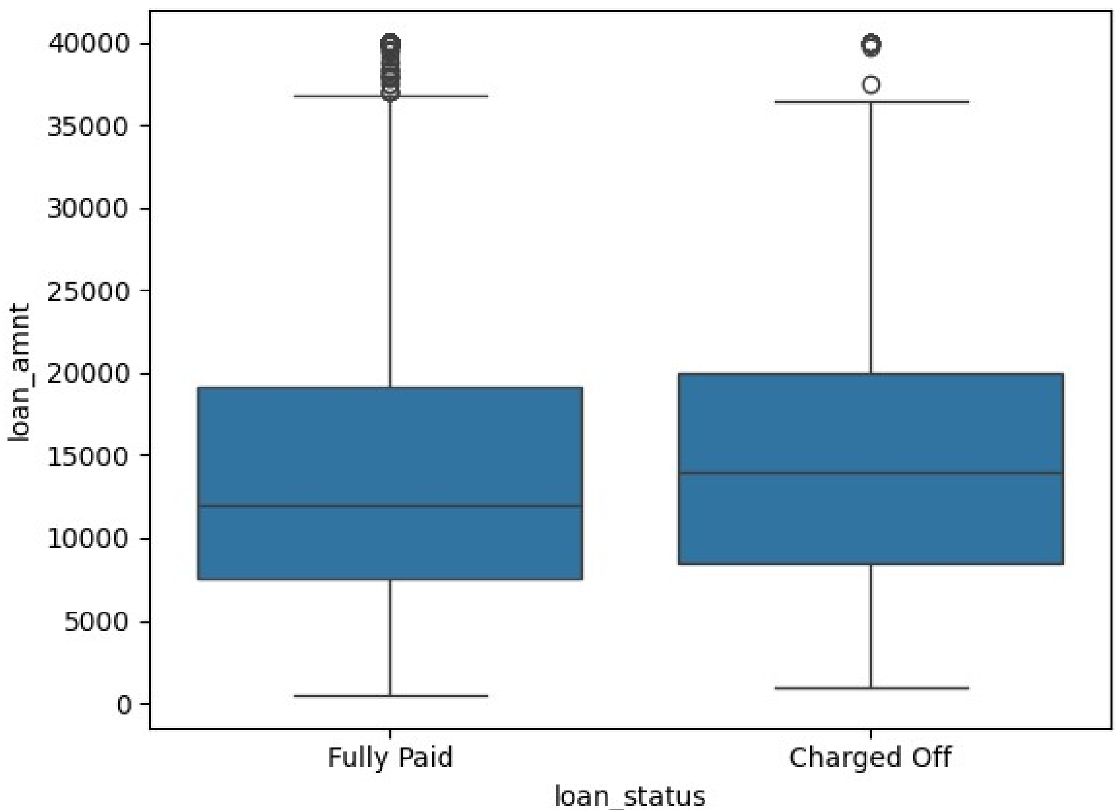
First lets try to find all features ditsributions

```
df_main.columns
```

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

```
sns.boxplot(x='loan_status', y='loan_amnt', data = df_main)
```

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

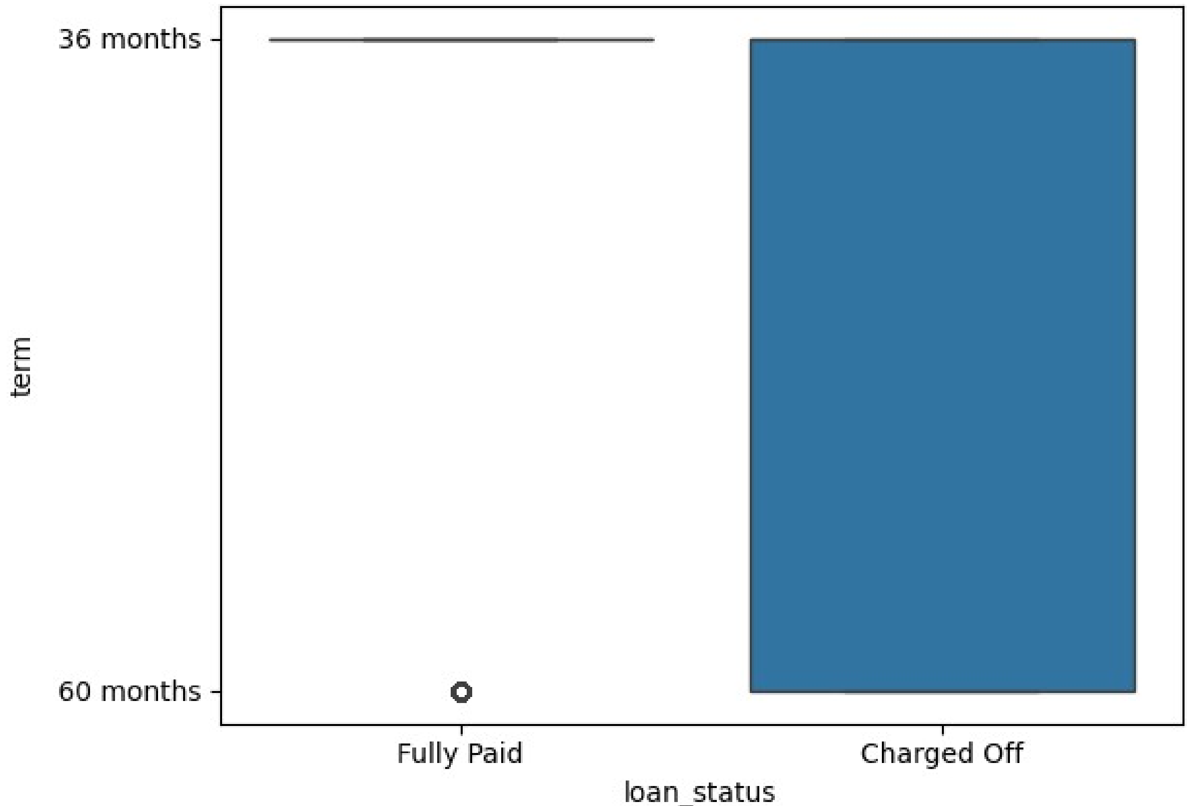


Median: The median loan amount appears not exactly similar for both statuses.
Spread: Both categories have a wide spread, but "Charged Off" might have slightly more variability.
Outliers: Both groups show outliers at higher loan amounts.

'loan_status' is important feature

```
sns.boxplot(x='loan_status', y='term', data = df_main)
```

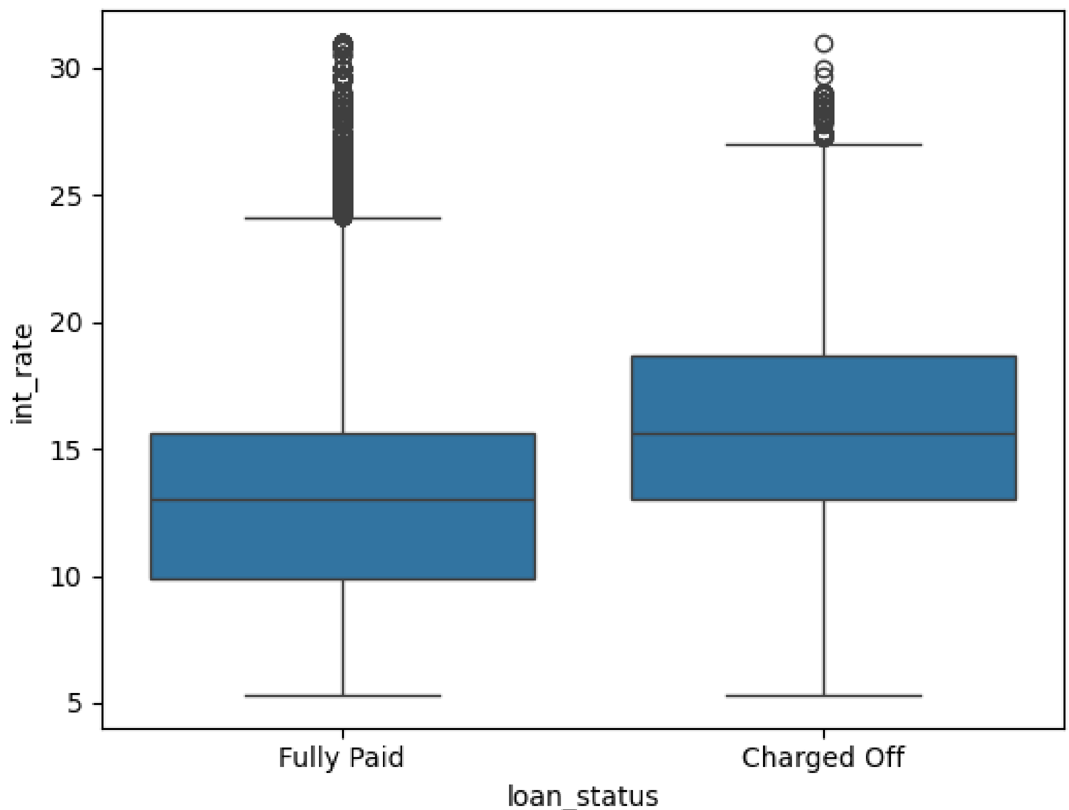
```
<Axes: xlabel='loan_status', ylabel='term'>
```



this feature is completely inclined towards Charged off, hence we will not be taking this feature

```
sns.boxplot(x='loan_status', y='int_rate', data = df_main)
```

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

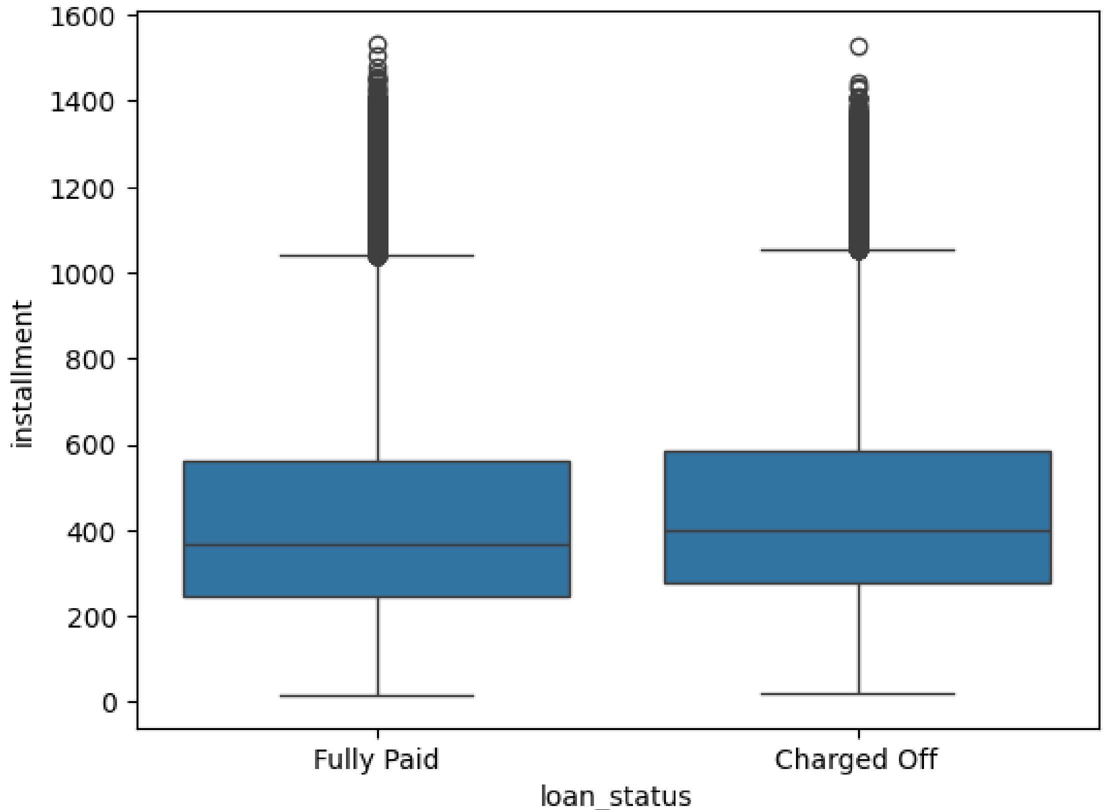


The charged of status have a higher median for interest rate than fully paid which means:
interest rate tend to charged off

int_rate is an important feature

```
sns.boxplot(x='loan_status', y='installment', data = df_main)
```

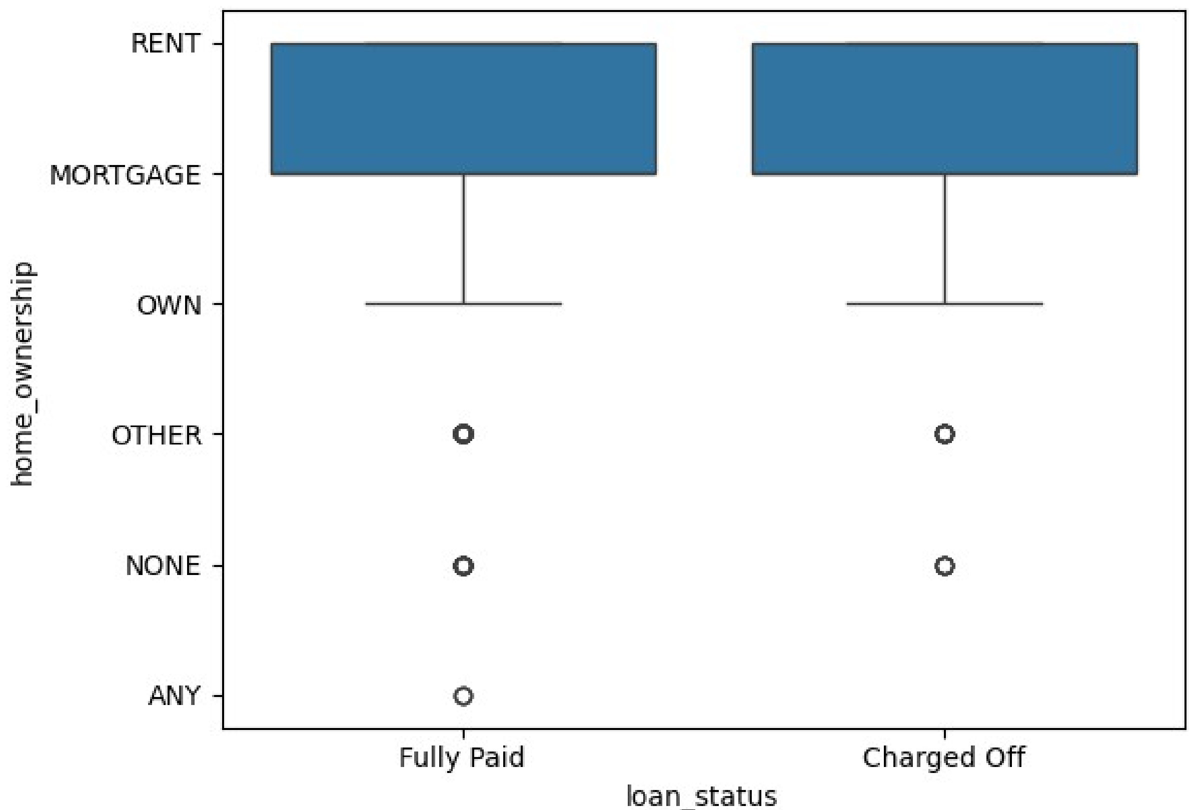
```
<Axes: xlabel='loan_status', ylabel='installment'>
```



If we see median for both, its quite similar Hence this feature does not have any signficance and can be dropped

```
sns.boxplot(x='loan_status', y='home_ownership', data = df_main)
```

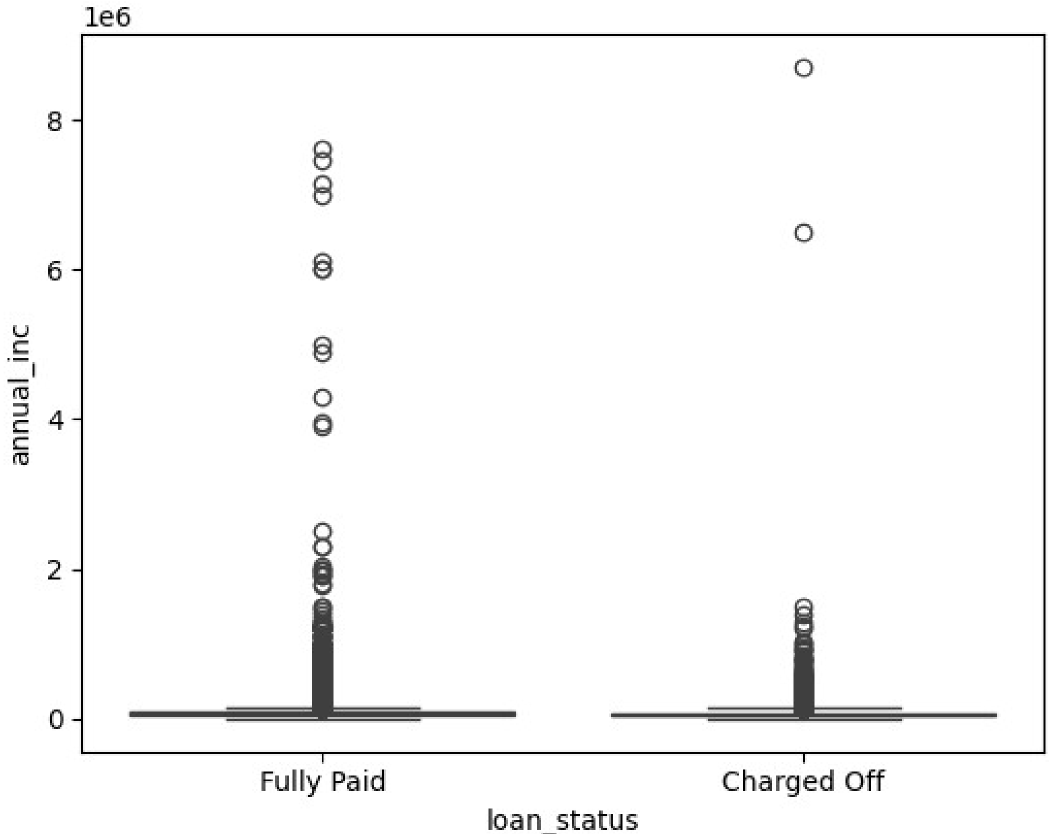
```
<Axes: xlabel='loan_status', ylabel='home_ownership'>
```



If we see median for both, its quite similar Hence this feature does not have any signficance and can be dropped

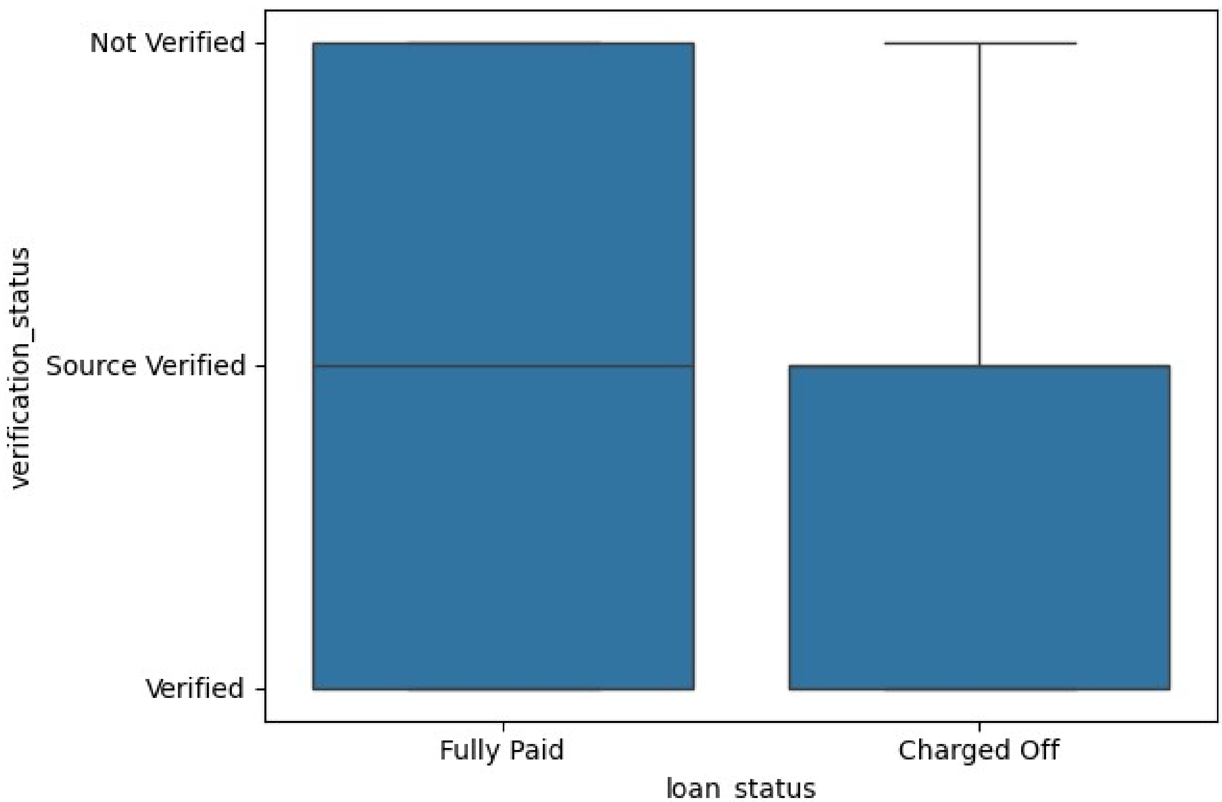
```
sns.boxplot(x='loan_status', y='annual_inc', data = df_main)
```

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




```
sns.boxplot(x='loan_status', y='verification_status', data = df_main)
```

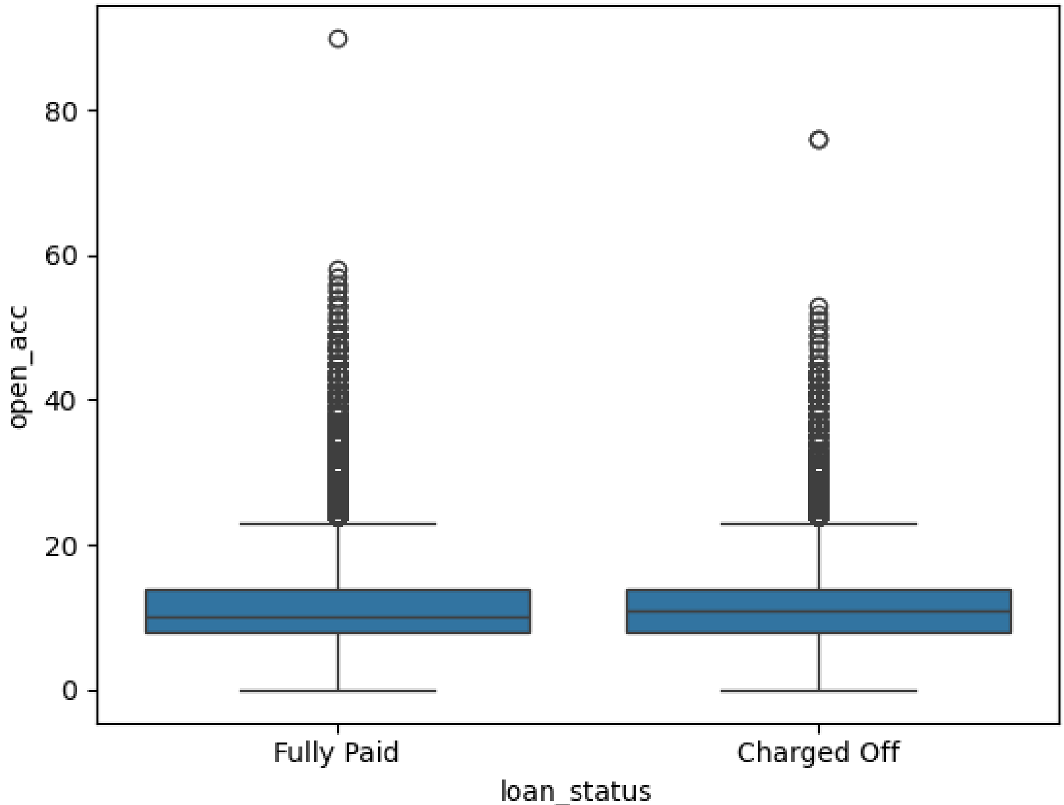
```
<Axes: xlabel='loan_status', ylabel='verification_status'>
```



it is an important feature

```
sns.boxplot(x='loan_status', y='open_acc', data = df_main)
```

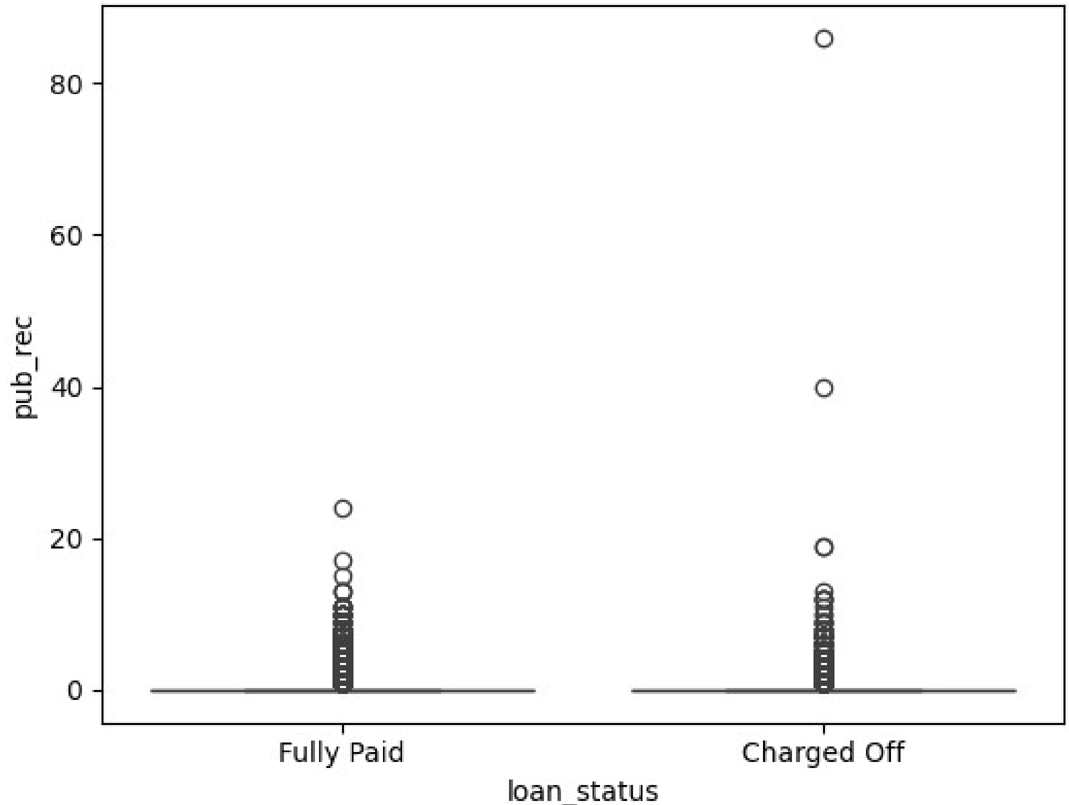
```
<Axes: xlabel='loan_status', ylabel='open_acc'>
```



it can be dropped

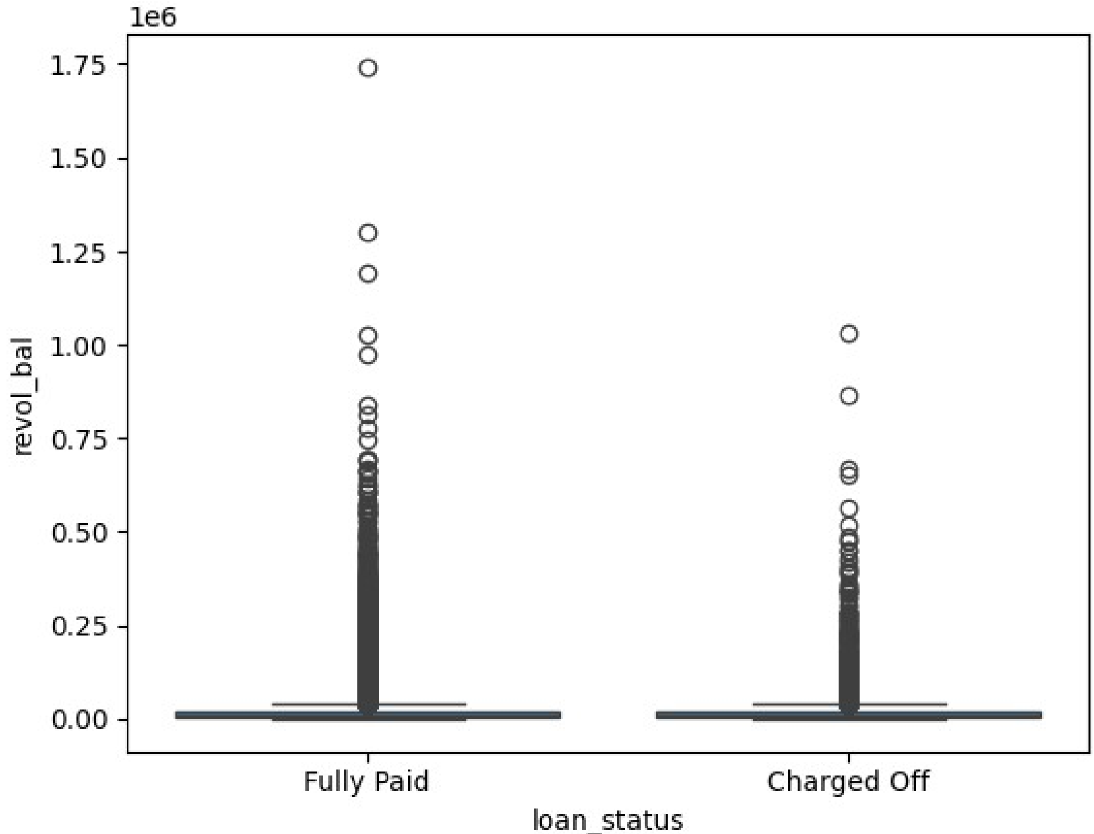
```
sns.boxplot(x='loan_status', y='pub_rec', data = df_main)
```

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



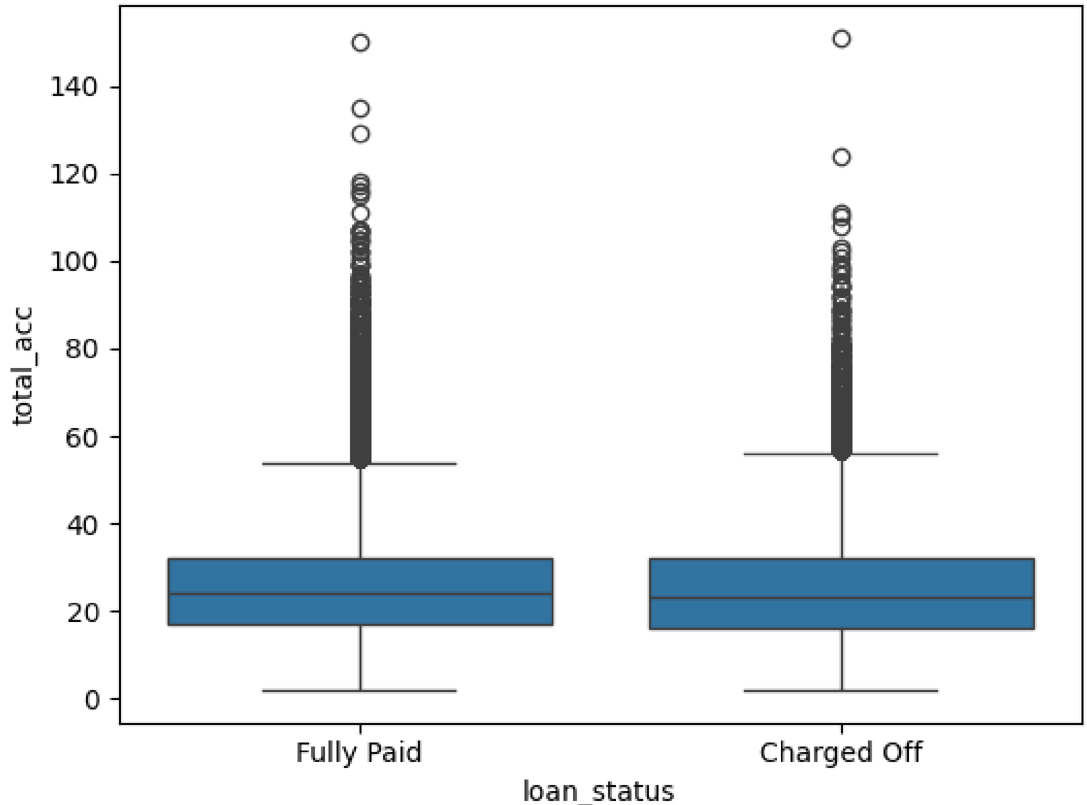

```
sns.boxplot(x='loan_status', y='revol_bal', data = df_main)
```

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



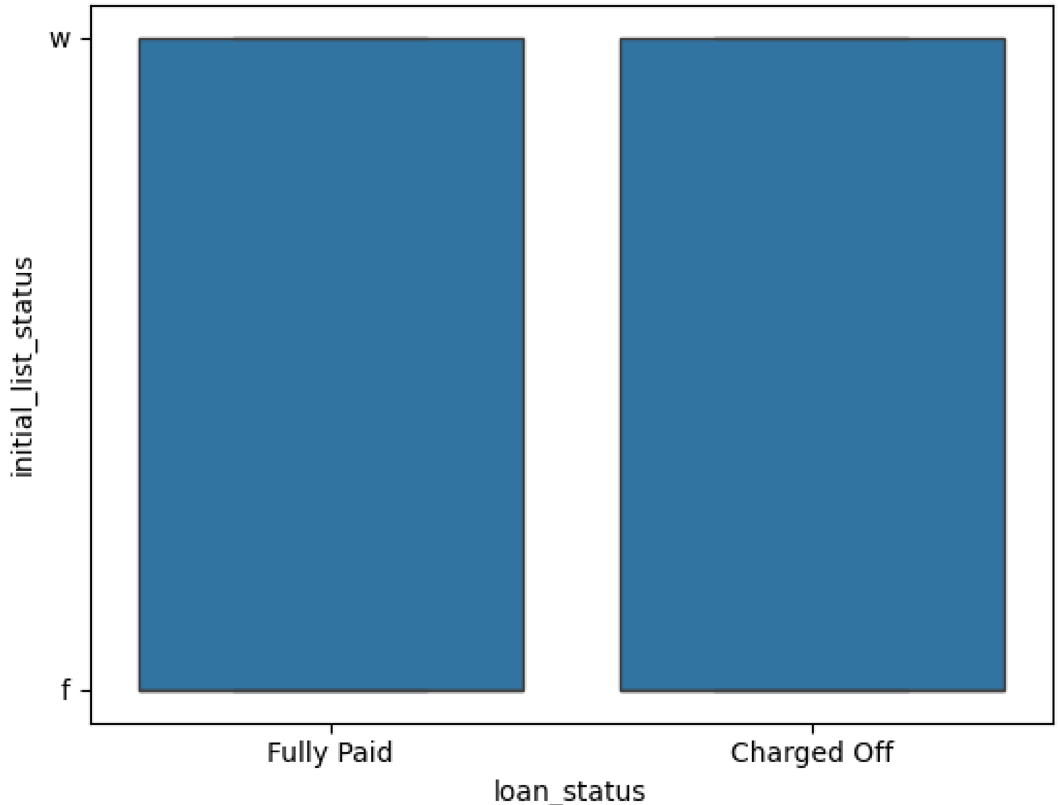

```
sns.boxplot(x='loan_status', y='total_acc', data = df_main)
```

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




```
sns.boxplot(x='loan_status', y='initial_list_status', data = df_main)
```

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



As observe From the above plots, we only take 3 important feature to train a ligistic regression model

```
df = df_main[['loan_amnt','int_rate','annual_inc']]
df
```

{"type":"dataframe","variable_name":"df"}

Lets analyse the target feature now

```
df_main['loan_status']
```

```
0          Fully Paid
1          Fully Paid
2          Fully Paid
3          Fully Paid
4         Charged Off
             ...
396025     Fully Paid
396026     Fully Paid
396027     Fully Paid
396028     Fully Paid
396029     Fully Paid
Name: loan_status, Length: 396030, dtype: object
```

```python
# converting the target variable into numeric categories
from sklearn.preprocessing import LabelEncoder

# Initialize encoder
encoder = LabelEncoder()

# Apply encoding
df_main['loan_status'] = encoder.fit_transform(df_main['loan_status'])
```

This will map Fully Paid and Charged Off to 0 or 1

```python
df_main['loan_status'].value_counts()
```

```
loan_status
1    318357
0     77673
Name: count, dtype: int64
```

```python
y = df_main['loan_status']
x = df
# getting the shape of the features and columns
print(y.shape)
print(x.shape)
```

```
(396030,)
(396030, 3)
```

Split the data into train validation and test

```python
from sklearn.model_selection import train_test_split

# train_test_split divides the dataset into 80% (x_tr_cv, y_tr_cv) and 
20% (x_test, y_test).
x_tr_cv, x_test, y_tr_cv, y_test = train_test_split(x, y, 
test_size=0.2, random_state=1)
# From the 80% (x_tr_cv, y_tr_cv), another split allocates 75% for 
x_train, y_train and 25% for x_val, y_val.
x_train, x_val, y_train, y_val = train_test_split(x_tr_cv, y_tr_cv, 
test_size=0.25,random_state=1)

print(x_train.shape)
print(x_val.shape)
print(x_test.shape)
```

```
(237618, 3)
(79206, 3)
(79206, 3)
```

```python
# scalling the weights of the features
from sklearn.preprocessing import StandardScaler
```

```
# initiate the standard scaler
scaler = StandardScaler()
# fit the standard scaler
scaler.fit(x_train)

# transform the data
X_train = scaler.transform(x_train)
X_val = scaler.transform(x_val)
X_test = scaler.transform(x_test)

X_train
```

```
array([[-0.49187952, -0.10549908, -0.42589236],
       [ 2.50121578,  0.00845449,  0.08593395],
       [ 2.50121578,  1.02956781,  0.83853115],
       ...,
       [ 1.18425385,  1.0988337 , -0.28882936],
       [-1.09049858,  0.2251897 , -0.1514605 ],
       [-0.37215571, -1.17130007, -0.44088409]])
```

```
# importing the logistic regression model from scikit learn
from sklearn.linear_model import LogisticRegression
# initiate the model
model = LogisticRegression()
# train the model
model.fit(X_train, y_train)
```

```
LogisticRegression()
```

```
# coeficient of the model or separetor
model.coef_
```

```
array([[-0.1519049 , -0.58675452,  0.38998474]])
```

```
# getting the intercept of the model
model.intercept_
```

```
array([1.54751541])
```

```
# getting the predicted value from the X_train
y_hat = model.predict(X_train)

# getting the accuracy
def accuracy(y_true, y_pred):
  return np.sum(y_true==y_pred)/y_true.shape[0]

accuracy(y_train, y_hat)
```

```
0.8038995362304202
```

```
accuracy(y_val, model.predict(X_val))
```

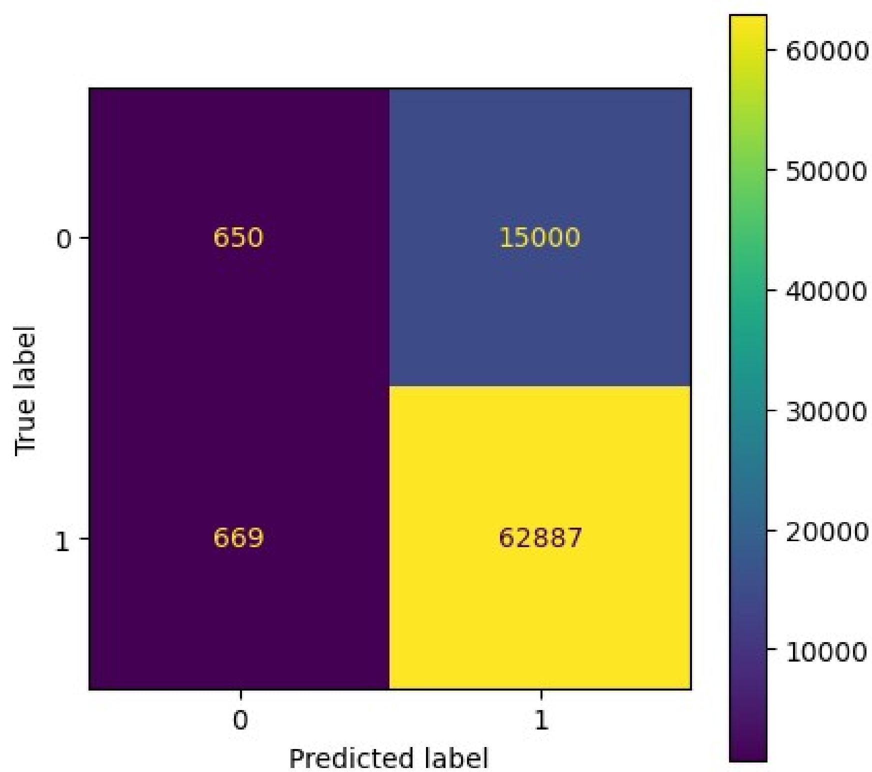
```
0.8021993283337121
```

So our model has a validation accuracy of 80%

Confusion Matrix

Lets use sklearn confusion_matrix function to get the values

```
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay

y_pred = model.predict(X_test)

conf_matrix = confusion_matrix(y_test, y_pred)
conf_matrix # 2D np array

array([[  650, 15000],
       [  669, 62887]])

# ax used here to control the size of confusion matrix
fig, ax = plt.subplots(figsize=(5,5))
ConfusionMatrixDisplay(conf_matrix).plot(ax = ax)

<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at
0x7aa7d167cf10>
```



Finding Accuracy using Confusion Matrix

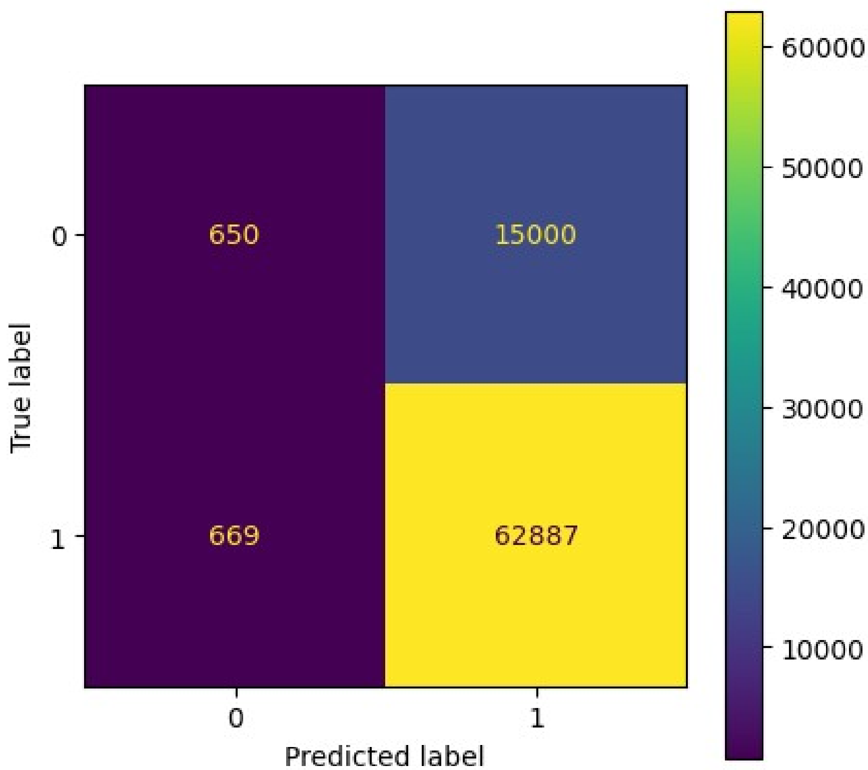
```
np.diag(conf_matrix).sum() / conf_matrix.sum()
```

```
0.8021740777213847
```

Precision

```
fig, ax = plt.subplots(figsize=(5,5))
ConfusionMatrixDisplay(conf_matrix).plot(ax = ax)
```

```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at
0x7aa7d167cdc0>
```




```
from sklearn.metrics import precision_score

precision_score(y_test, y_pred)
```

```
0.8074133038889674
```

observe

Even though the model has a similar precision value than accuracy:

Its still a good model because of its high precision value

Recall

```
from sklearn.metrics import recall_score

recall_score(y_test, y_pred)

0.989473849833218
```

observe

The model's recall value is very higher than accuracy and precision value:
The model captures more true positives (fewer false negatives).
However, it also predicts more false positives, reducing precision.

Insights

1. our model has a validation accuracy of 80%
2. Even though the model has a similar precision value than accuracy:
3. Its still a good model because of its high precision value
4. The model's recall value is very higher than accuracy and precision value:
5. The model captures more true positives (fewer false negatives).
6. However, it also predicts more false positives, reducing precision.