LoanTap
Fast. Flexible. Friendly.

```
import pandas as pd
import numpy as np
import seaborn as sns
from scipy import stats
import matplotlib.pyplot as plt
from sklearn.linear_model import LogisticRegression
from sklearn import metrics
from sklearn.metrics import confusion_matrix
from sklearn.metrics import classification_report
from sklearn.metrics import roc_auc_score
from sklearn.metrics import roc_curve
from sklearn.metrics import precision_recall_curve
from sklearn.model_selection import train_test_split, KFold, cross_val_score
from sklearn.preprocessing import MinMaxScaler
```

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True).
```

```
!ls
```

```
drive  sample_data
```

```
df=pd.read_csv("/content/drive/MyDrive/logistic_regression.csv")
```

```
df.head()
```

	loan_amnt	term	int_rate	installment	grade	sub_grade	emp_title	emp_lengtl
0	10000.0	36 months	11.44	329.48	B	B4	Marketing	10+ years
1	8000.0	36 months	11.99	265.68	B	B5	Credit analyst	4 years
2	15600.0	36 months	10.49	506.97	B	B3	Statistician	< 1 yea
3	7200.0	36 months	6.49	220.65	A	A2	Client Advocate	6 years
4	24375.0	60 months	17.27	609.33	C	C5	Destiny Management Inc.	9 years

5 rows × 27 columns

```
df.shape
```

```
(396030, 27)
```

```
df["loan_status"].value_counts(normalize=True) * 100
```

```
Fully Paid     80.387092
Charged Off    19.612908
Name: loan_status, dtype: float64
```

```
df.describe(include='all')
```

	loan_amnt	term	int_rate	installment	grade	sub_grade	emp_ti
count	396030.000000	396030	396030.000000	396030.000000	396030	396030	373
unique	NaN	2	NaN	NaN	7	35	173
top	NaN	36 months	NaN	NaN	B	B3	Tea
freq	NaN	302005	NaN	NaN	116018	26655	4
mean	14113.888089	NaN	13.639400	431.849698	NaN	NaN	N
std	8357.441341	NaN	4.472157	250.727790	NaN	NaN	N
min	500.000000	NaN	5.320000	16.080000	NaN	NaN	N
25%	8000.000000	NaN	10.490000	250.330000	NaN	NaN	N
50%	12000.000000	NaN	13.330000	375.430000	NaN	NaN	N
75%	20000.000000	NaN	16.490000	567.300000	NaN	NaN	N
max	40000.000000	NaN	30.990000	1533.810000	NaN	NaN	N

11 rows × 27 columns

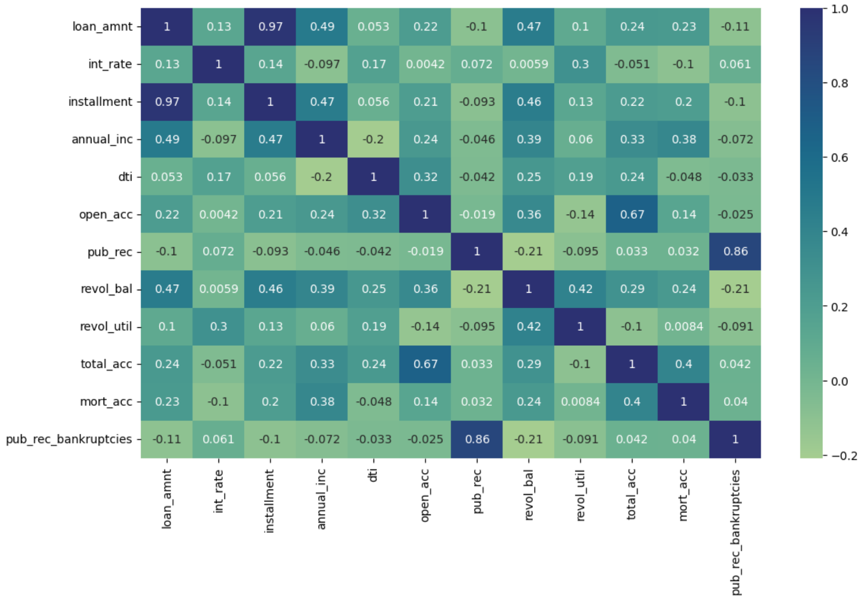
```
df.info()
```

```
<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
```

Correlation between independent features in the data set

```
plt.figure(figsize=(12,7))
sns.heatmap(df.corr(method='spearman'), annot=True, cmap='crest')
```

```
<ipython-input-10-64190c28e1eb>:2: FutureWarning: The default value of numeric_only i
  sns.heatmap(df.corr(method='spearman'), annot=True, cmap='crest')
<Axes: >
```



Dropping "installement" features as it is related to "loan_amnt"

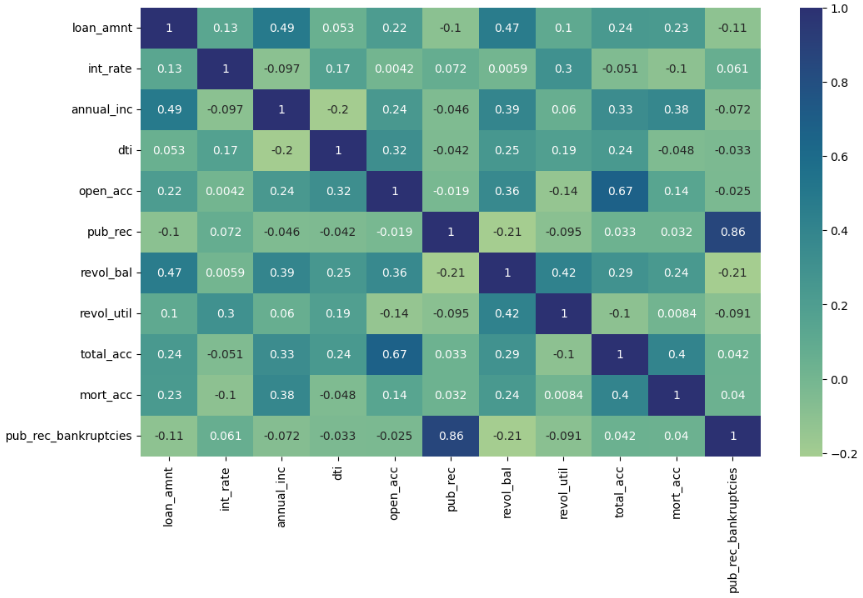
```
df.drop(columns=["installment"], inplace=True)
```

```
df.shape
```

```
(396030, 26)
```

```
plt.figure(figsize=(12,7))
sns.heatmap(df.corr(method='spearman'), annot=True, cmap="crest")
plt.show()
```

```
<ipython-input-13-50821c1d4bc5>:2: FutureWarning: The default value of numeric_only i
  sns.heatmap(df.corr(method='spearman'), annot=True, cmap="crest")
```



```
df['loan_status'].value_counts(normalize=True) * 100
```

```
Fully Paid     80.387092
Charged Off    19.612908
Name: loan_status, dtype: float64
```

```
df.groupby(by='loan_status')['loan_amnt'].describe()
```

	count	mean	std	min	25%	50%	75%	n
loan_status								
Charged Off	77673.0	15126.300967	8505.090557	1000.0	8525.0	14000.0	20000.0	4000
Fully Paid	318357.0	13866.878771	8302.319699	500.0	7500.0	12000.0	19225.0	4000

There is a significant difference between the loan amount taken by charged_off and fully_paid users

```
df['home_ownership'].value_counts()
```

```
MORTGAGE    198348
RENT        159790
OWN          37746
OTHER          112
NONE            31
ANY              3
Name: home_ownership, dtype: int64
```

Most of the people applying for loans have either Mortgages or live in a rented house.

```
df.home_ownership.loc[(df.home_ownership == 'ANY') | (df.home_ownership == 'NONE')] = 'OTHER'
df.home_ownership.value_counts()
```

```
<ipython-input-17-50451bc6ac93>:1: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus
```

```
  df.home_ownership.loc[(df.home_ownership == 'ANY') | (df.home_ownership == 'NONE')] = 'OTHER'
MORTGAGE    198348
RENT        159790
OWN          37746
OTHER          146
Name: home_ownership, dtype: int64
```

Merged home_ownership of ANY, NONE and OTHER into a single group OTHER

```
# coverting issue_d and earliest_cr_line to datetime data type
df["issue_d"]=pd.to_datetime(df["issue_d"])
df["earliest_cr_line"]= pd.to_datetime(df["earliest_cr_line"])
```

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 396030 entries, 0 to 396029
Data columns (total 26 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   grade                 396030 non-null  object
 4   sub_grade             396030 non-null  object
 5   emp_title             373103 non-null  object
 6   emp_length            377729 non-null  object
 7   home_ownership        396030 non-null  object
 8   annual_inc            396030 non-null  float64
 9   verification_status   396030 non-null  object
 10  issue_d               396030 non-null  datetime64[ns]
 11  loan_status           396030 non-null  object
 12  purpose               396030 non-null  object
 13  title                 394275 non-null  object
 14  dti                   396030 non-null  float64
 15  earliest_cr_line      396030 non-null  datetime64[ns]
 16  open_acc              396030 non-null  float64
 17  pub_rec               396030 non-null  float64
 18  revol_bal             396030 non-null  float64
 19  revol_util            395754 non-null  float64
 20  total_acc             396030 non-null  float64
 21  initial_list_status   396030 non-null  object
 22  application_type      396030 non-null  object
 23  mort_acc              358235 non-null  float64
 24  pub_rec_bankruptcies  395495 non-null  float64
 25  address               396030 non-null  object
dtypes: datetime64[ns](2), float64(11), object(13)
memory usage: 78.6+ MB
```

```
df['title']=df["title"].str.lower()
df["title"].value_counts()
```

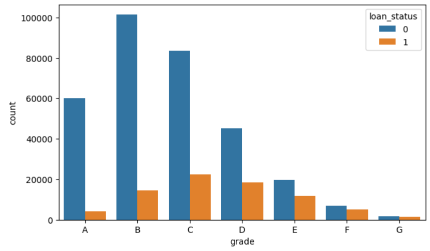
```
debt consolidation                   168108
credit card refinancing               51781
home improvement                      17117
other                                 12993
consolidation                          5583
                                      ...
sweet                                     1
mortgage convertion                       1
debt consolidation and relocation         1
1 payment loan plan                       1
toxic debt payoff                         1
Name: title, Length: 41327, dtype: int64
```

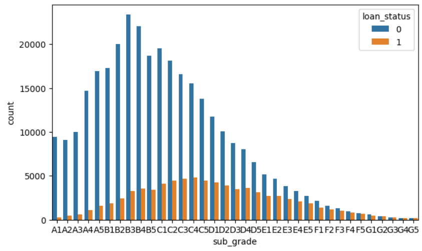
Two important reason for taking loan is :

- debt consolidation
- credit card refinancing

```
def pub_rec(number):
    if number == 0.0:
        return 0
    else:
        return 1

def mort_acc(number):
    if number == 0.0:
        return 0
    else:
        return 1


def pub_rec_bankruptcies(number):
    if number == 0.0:
        return 0
    else:
        return 1


df['pub_rec'] = df.pub_rec.apply(pub_rec)
df['mort_acc'] = df.mort_acc.apply(mort_acc)
df['pub_rec_bankruptcies'] = df.pub_rec_bankruptcies.apply(pub_rec_bankruptcies)
df['loan_status'] = df.loan_status.map({'Fully Paid':0, 'Charged Off':1})


plt.figure(figsize=(17, 10))
plt.subplot(2,2,1)
grade= sorted(df["grade"].unique().tolist())
sns.countplot(data=df, x="grade", hue="loan_status", order=grade)

plt.subplot(2,2,2)
sub_grade= sorted(df["sub_grade"].unique().tolist())
sns.countplot(data=df, x="sub_grade", hue="loan_status", order=sub_grade)
```

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

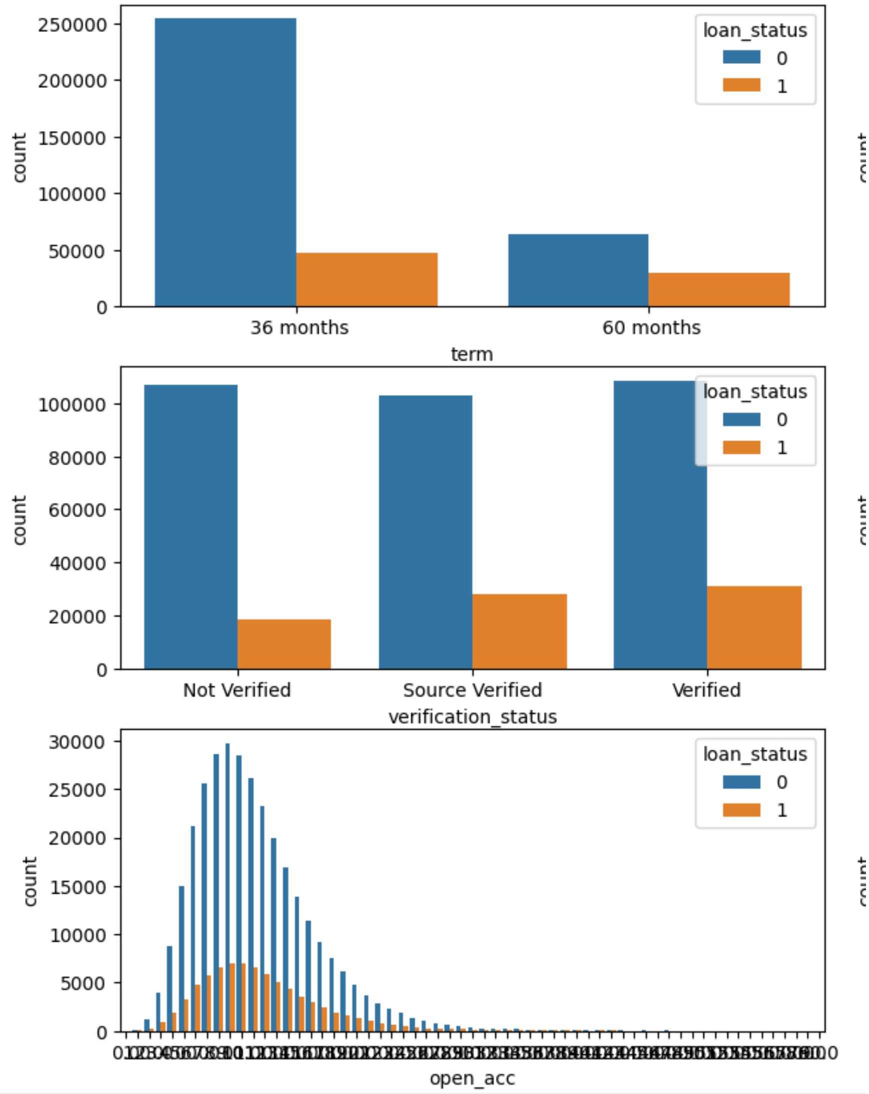




- Most loan belong to grade B and subgrade B3
- Loans with grade C are more likely to be charged off

```
plt.figure(figsize=(15,10))

plt.subplot(3,2,1)
sns.countplot(data=df, x='term', hue='loan_status')

plt.subplot(3,2,2)
sns.countplot(data=df, x='home_ownership', hue='loan_status')

plt.subplot(3,2,3)
sns.countplot(data=df, x='verification_status', hue='loan_status')

plt.subplot(3,2,4)
sns.countplot(data=df, x='mort_acc', hue='loan_status')

plt.subplot(3,2,5)
sns.countplot(data=df, x='open_acc', hue='loan_status')

plt.subplot(3,2,6)
sns.countplot(data=df, x='pub_rec_bankruptcies', hue='loan_status')
```

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




```
df.emp_title.value_counts()[:3]
```

```
Teacher             4389
Manager             4250
Registered Nurse    1856
Name: emp_title, dtype: int64
```

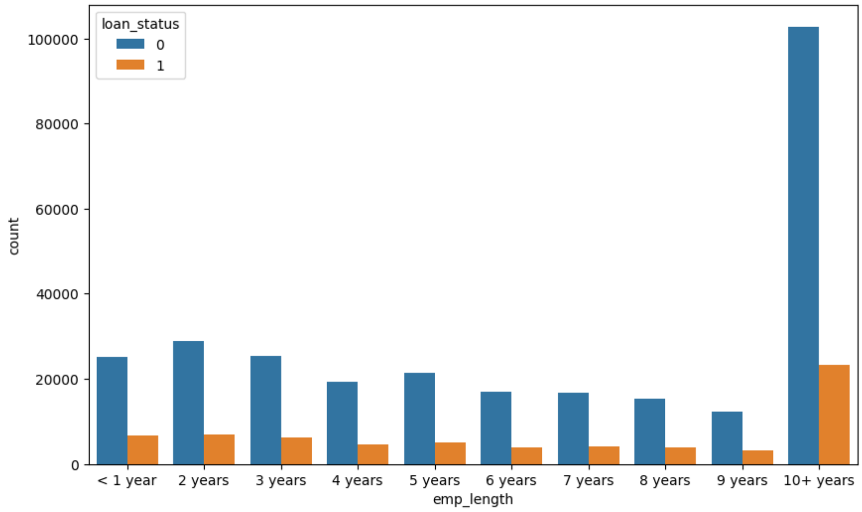
Teachers and Managers are the most common applying for loan

```
df.emp_length.value_counts()
```

```
10+ years    126041
2 years       35827
< 1 year      31725
3 years       31665
5 years       26495
1 year        25882
4 years       23952
6 years       20841
7 years       20819
8 years       19168
9 years       15314
Name: emp_length, dtype: int64
```

```
plt.figure(figsize=(10,6))
order= ["< 1 year", "2 years", "3 years", "4 years", "5 years", "6 years", "7 years", "8 years", "9 years", "10+ years"]
sns.countplot(data=df, x='emp_length', hue="loan_status", order=order)
```

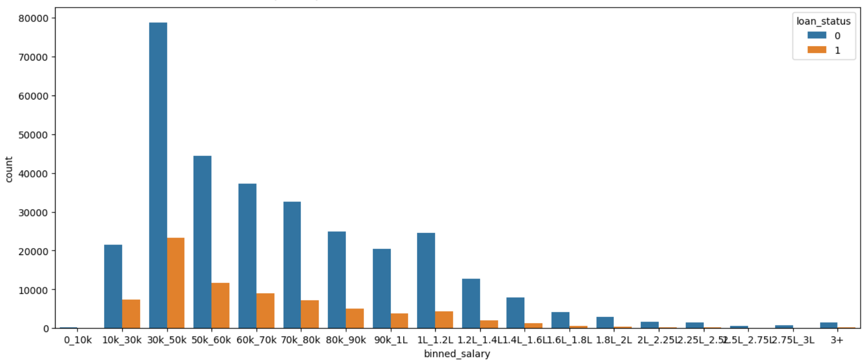
```
<Axes: xlabel='emp_length', ylabel='count'>
```



Higher the tenurity higher the chances of approving loan

```
bins= [0.0, 10000.0, 30000.0, 50000.0, 60000.0, 70000.0, 80000.0, 90000.0, 100000.0,120000.0, 140000.0, 160000.0, 180000.0, 200000.0, 2:
label=["0_10k","10k_30k", "30k_50k","50k_60k","60k_70k", "70k_80k", "80k_90k", "90k_1L", "1L_1.2L", "1.2L_1.4L", "1.4L_1.6L", "1.6L_1.8I
df['binned_salary']= pd.cut(df["annual_inc"], bins=bins, labels=label)

plt.figure(figsize=(15, 6))
sns.countplot(data=df, x="binned_salary", hue="loan_status", order=label)
```

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



Lesser the salary higher the chances of loan to be chared off and most loans applications are from people where salary range is between 10k to 1.2L

Handling missing values

```
(df.isnull().sum()/len(df)) * 100
```

```
loan_amnt               0.000000
term                    0.000000
int_rate                0.000000
grade                   0.000000
sub_grade               0.000000
emp_title               5.789208
emp_length              4.621115
home_ownership          0.000000
annual_inc              0.000000
verification_status     0.000000
issue_d                 0.000000
loan_status             0.000000
purpose                 0.000000
title                   0.443148
dti                     0.000000
earliest_cr_line        0.000000
open_acc                0.000000
pub_rec                 0.000000
revol_bal               0.000000
revol_util              0.069692
total_acc               0.000000
initial_list_status     0.000000
application_type        0.000000
mort_acc                0.000000
pub_rec_bankruptcies    0.000000
address                 0.000000
binned_salary           0.000253
dtype: float64
```

```
total_acc_avg = df.groupby(by='total_acc').mean().mort_acc
def fill_mort_acc(total_acc, mort_acc):
    if np.isnan(mort_acc):
        return total_acc_avg[total_acc].round()
    else:
        return mort_acc
df['mort_acc'] = df.apply(lambda x: fill_mort_acc(x['total_acc'], x['mort_acc']), axis=1)
```

```
<ipython-input-30-5bbf87b41589>:1: FutureWarning: The default value of numeric_only in DataFrameGroupBy.mean is deprecated. In a fu
  total_acc_avg = df.groupby(by='total_acc').mean().mort_acc
```

```
(df.isnull().sum()/len(df)) * 100
```

```
loan_amnt               0.000000
term                    0.000000
int_rate                0.000000
grade                   0.000000
sub_grade               0.000000
emp_title               5.789208
emp_length              4.621115
home_ownership          0.000000
annual_inc              0.000000
verification_status     0.000000
issue_d                 0.000000
loan_status             0.000000
purpose                 0.000000
title                   0.443148
dti                     0.000000
earliest_cr_line        0.000000
open_acc                0.000000
pub_rec                 0.000000
revol_bal               0.000000
revol_util              0.069692
total_acc               0.000000
initial_list_status     0.000000
application_type        0.000000
mort_acc                0.000000
pub_rec_bankruptcies    0.000000
address                 0.000000
binned_salary           0.000253
dtype: float64
```

```
#droppig null values
df.dropna(inplace=True)
df.shape
```

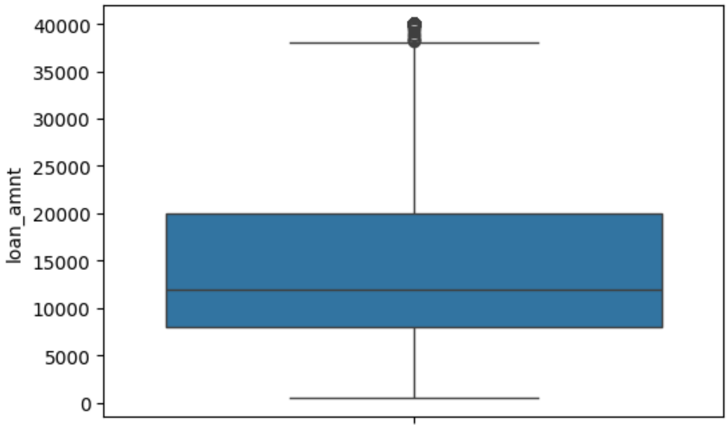
```
(371126, 27)
```

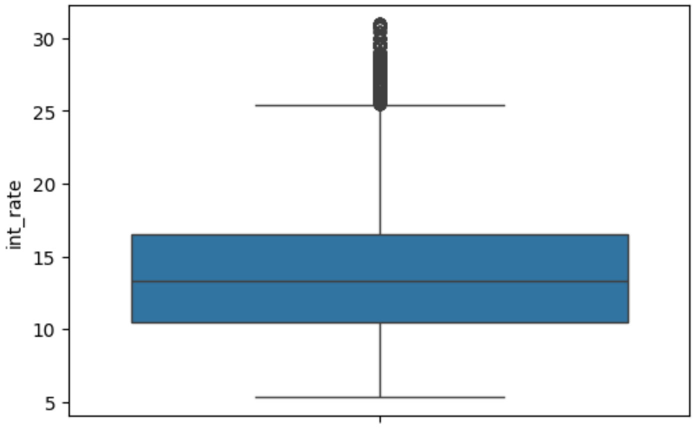
Handling outliers

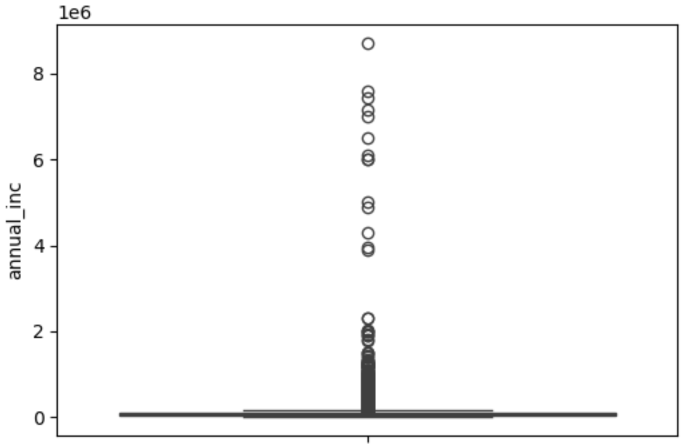
```
num_data= df.select_dtypes(include='number')
num_col= num_data.columns
num_col
```

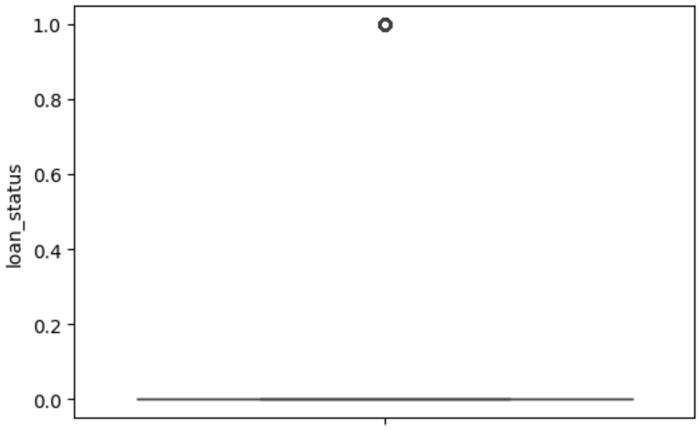
```
Index(['loan_amnt', 'int_rate', 'annual_inc', 'loan_status', 'dti', 'open_acc',
       'pub_rec', 'revol_bal', 'revol_util', 'total_acc', 'mort_acc',
       'pub_rec_bankruptcies'],
      dtype='object')
```

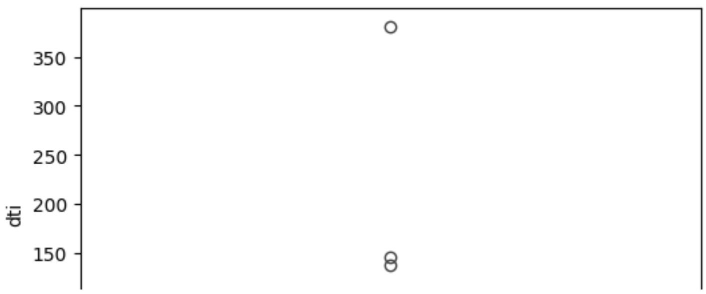
```
for i in num_col:
  plt.figure(figsize=(6,4))
  sns.boxplot(df[i])
  plt.show()
```

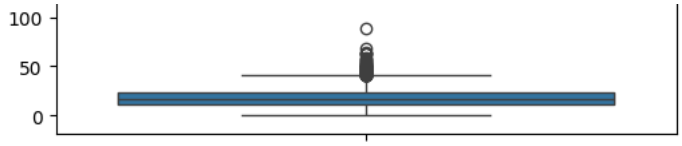

loan_amnt
0
5000
10000
15000
20000
25000
30000
35000
40000

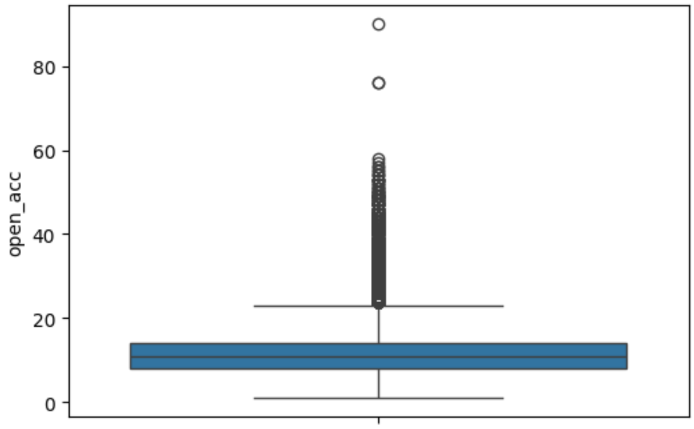

int_rate
5
10
15
20
25
30

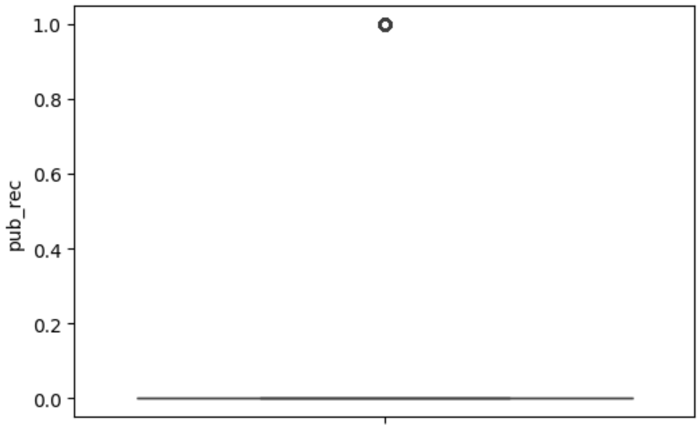

1e6
annual_inc
0
2
4
6
8

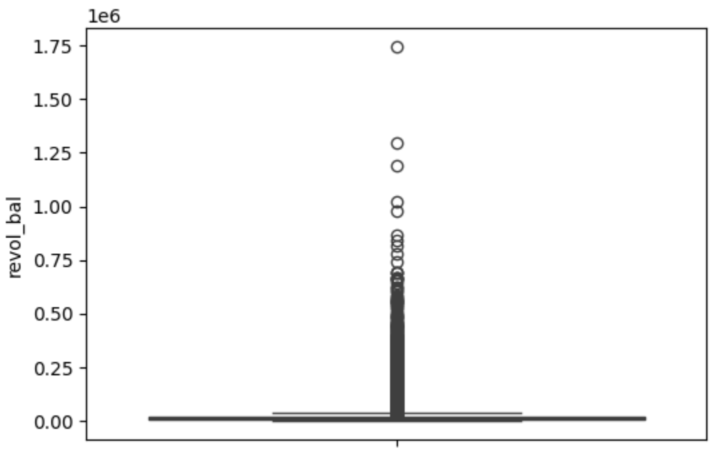

loan_status
0.0
0.2
0.4
0.6
0.8
1.0

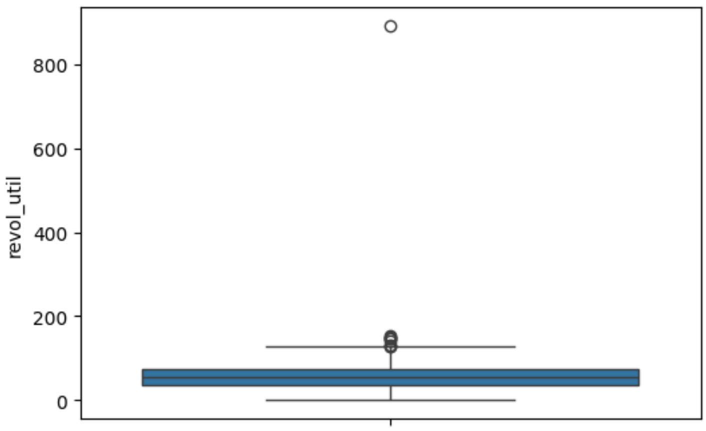

dti
150
200
250
300
350

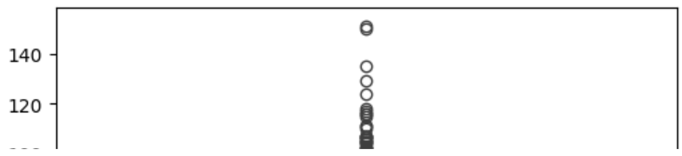

100
50
0

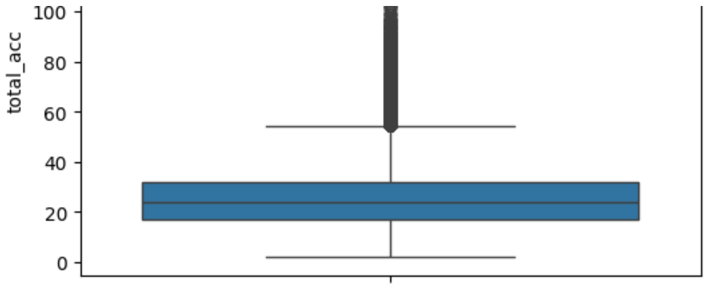

open_acc
80
60
40
20
0

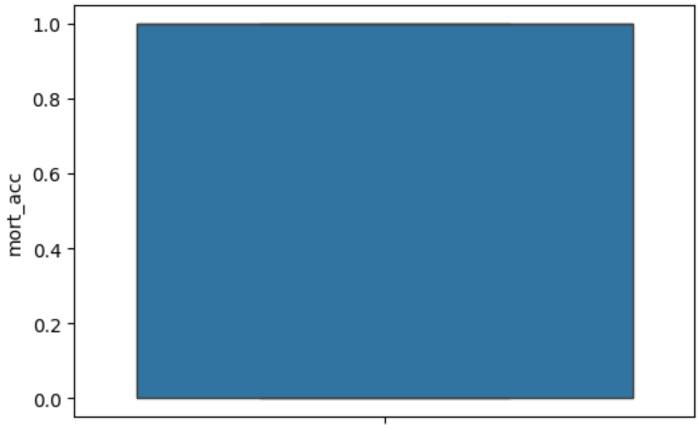

pub_rec
1.0
0.8
0.6
0.4
0.2
0.0

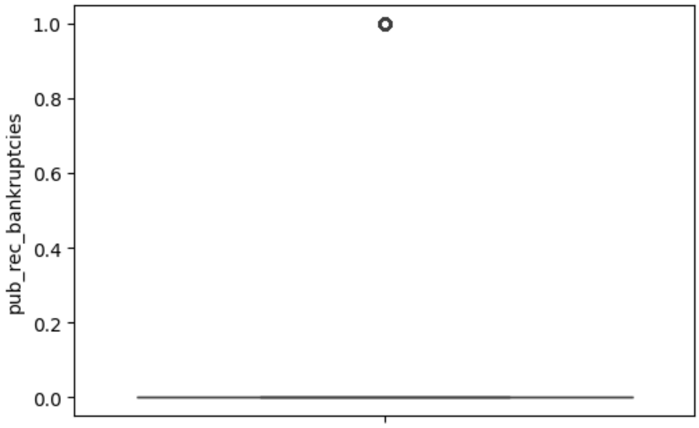

1e6
revol_bal
1.75
1.50
1.25
1.00
0.75
0.50
0.25
0.00


revol_util
800
600
400
200
0


140
120








```
for col in num_col:
    mean = df[col].mean()
    std = df[col].std()

    upper_limit = mean+3*std
    lower_limit = mean-3*std

    df = df[(df[col]<upper_limit) & (df[col]>lower_limit)]
```

```
df.shape
```

```
(355005, 27)
```

DATA PROCESSING

```
df['initial_list_status'].replace({'w': 0,'f': 1 }, inplace=True)
```

```
df['initial_list_status'].value_counts()
```

```
1    214523
0    140482
Name: initial_list_status, dtype: int64
```

```
term_values = {' 36 months': 36, ' 60 months': 60}
df['term'] = df.term.map(term_values)
```

```
df["term"].value_counts()
```

```
36    270167
60     84838
Name: term, dtype: int64
```

```
df['zipcode'] = df.address.apply(lambda x: x[-5:])
```

```
df['zipcode']
```

```
0         22690
1         05113
2         05113
3         00813
4         11650
          ...
396025    30723
396026    05113
396027    70466
396028    29597
396029    48052
Name: zipcode, Length: 355005, dtype: object
```

```
#dropping columns
df.drop(columns=['issue_d', 'emp_title', 'title', 'sub_grade','address', 'earliest_cr_line', 'emp_length', ], inplace=True)
```

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 355005 entries, 0 to 396029
Data columns (total 21 columns):
 #   Column                Non-Null Count   Dtype
---  ------                --------------   -----
 0   loan_amnt             355005 non-null  float64
 1   term                  355005 non-null  int64
 2   int_rate              355005 non-null  float64
 3   grade                 355005 non-null  object
 4   home_ownership        355005 non-null  object
 5   annual_inc            355005 non-null  float64
 6   verification_status   355005 non-null  object
 7   loan_status           355005 non-null  int64
 8   purpose               355005 non-null  object
 9   dti                   355005 non-null  float64
 10  open_acc              355005 non-null  float64
 11  pub_rec               355005 non-null  int64
 12  revol_bal             355005 non-null  float64
 13  revol_util            355005 non-null  float64
 14  total_acc             355005 non-null  float64
 15  initial_list_status   355005 non-null  int64
 16  application_type      355005 non-null  object
 17  mort_acc              355005 non-null  int64
 18  pub_rec_bankruptcies  355005 non-null  int64
 19  binned_salary         355005 non-null  category
 20  zipcode               355005 non-null  object
dtypes: category(1), float64(8), int64(6), object(6)
memory usage: 57.2+ MB
```

```
df.drop(columns=['binned_salary' ], inplace=True)
```

One Hot Encoding

```
dummies= ["grade", "home_ownership","verification_status", "purpose", "application_type", "zipcode"]
df=pd.get_dummies(df, columns=dummies, drop_first=True)
```

```
df.shape
```

```
(355005, 49)
```

Buidling model

```
Y= df["loan_status"]
X= df.drop('loan_status', axis=1)
```

```
X_train, X_test, Y_train, Y_test= train_test_split(X, Y, test_size= 0.3, stratify=Y, random_state=42)
```

```
print(X_train.shape)
print(Y_train.shape)
```

```
(248503, 48)
(248503,)
```

```
#scaling
min_max_scaler= MinMaxScaler()
X_train= min_max_scaler.fit_transform(X_train)
X_test= min_max_scaler.transform(X_test)
```

```
lr=LogisticRegression(max_iter=1000)
lr.fit(X_train, Y_train)
```

```
▾ LogisticRegression
LogisticRegression(max_iter=1000)
```

```
from sklearn.metrics import accuracy_score, recall_score, precision_score, roc_auc_score, f1_score
```

```
print('Accuracy: ', accuracy_score(Y_test, lr.predict(X_test)))
```

```
Accuracy:  0.8907344463014779
```

```
Y_pred= lr.predict(X_test)
Y_pred
```

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

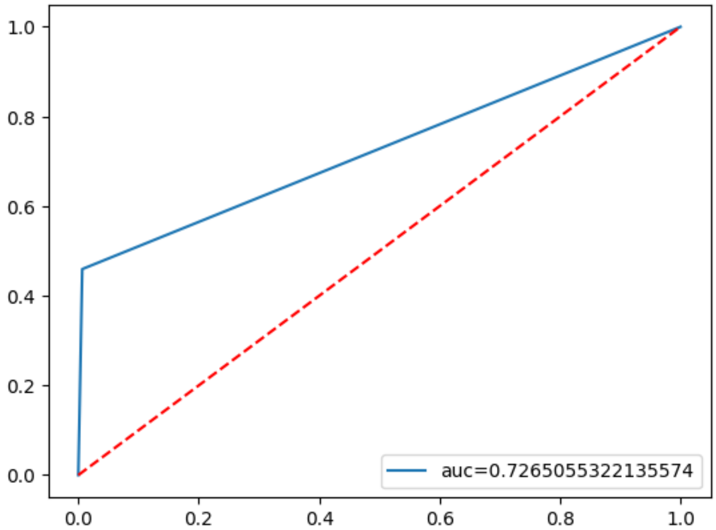
```
confusion_matrixx= confusion_matrix(Y_test, Y_pred)
print(confusion_matrixx)
```

```
[[85448   562]
 [11075  9417]]
```

```
print(classification_report(Y_test, Y_pred))
```

```
              precision    recall  f1-score   support

           0       0.89      0.99      0.94     86010
           1       0.94      0.46      0.62     20492

    accuracy                           0.89    106502
   macro avg       0.91      0.73      0.78    106502
weighted avg       0.90      0.89      0.88    106502
```

```
from sklearn.metrics import roc_curve, roc_auc_score

fpr, tpr, _ = roc_curve(Y_test,   Y_pred)
auc = roc_auc_score(Y_test, Y_pred)
plt.plot(fpr,tpr,label="auc="+str(auc))
plt.plot([0, 1], [0, 1],'r--')
plt.legend(loc=4)
plt.show()
```



Removing features which are multicollinear using Variance Inflation Factor

```
# VIF
from statsmodels.stats.outliers_influence import variance_inflation_factor
def calc_vif(X):
    # Calculating the VIF
    vif = pd.DataFrame()
    vif['Feature'] = X.columns
    vif['VIF'] = [variance_inflation_factor(X.values, i) for i in range(X.shape[1])]
    vif['VIF'] = round(vif['VIF'], 2)
    vif = vif.sort_values(by='VIF', ascending = False)
    return vif
```