LOAN TAP

Loan Analysis Problem Statament

Introduction LoanTap is an online platform committed to delivering customized loan products to millennials. They innovate in an otherwise dull loan segment, to deliver instant, flexible loans on consumer-friendly terms to salaried professionals and businessmen.

The data science team at LoanTap is building an underwriting layer to determine the creditworthiness of MSMEs as well as individuals.

Business Objective

LoanTap deploys formal credit to salaried individuals and businesses with 4 main financial instruments:

1.Personal Loan.

2.EMI Free Loan.

3.Personal Overdraft.

4.Advance Salary Loan.

This case study will focus on the underwriting process behind Personal Loans only

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?

Data Discription

loan_amnt: The listed amount of the loan applied for by the borrower. If at some point in time, the credit department reduces the loan amount, then it will be reflected in this value. term: The number of payments on the loan. Values are in months and can be either 36 or 60. int_rate: Interest Rate on the loan instalment: The monthly payment owed by the borrower if the loan originates. grade: LoanTap assigned loan grade sub_grade : LoanTap assigned loan subgrade emp_title :The job title supplied by the Borrower when applying for the loan.* emp_length : Employment length in years. Possible values are between 0 and 10 where 0 means less than one year and 10 means ten or more years. home_ownership : The home ownership status provided by the borrower during registration or obtained from the credit report. annual_inc : The self-reported annual income provided by the borrower during registration. verification_status: Indicates if income was verified by LoanTap, not verified, or if the income source was verified issue_d : The month which the loan was funded loan_status : Current status of the loan - Target Variable purpose : A category provided by the borrower for the loan request. title : The loan title provided by the borrower dti : A ratio calculated using the borrower’s total monthly debt payments on the total debt obligations, excluding mortgage and the requested LoanTap loan, divided by the borrower's self-reported monthly income. earliest_cr_line :The month the borrower's earliest reported credit line was opened open_acc : The number of open credit lines in the borrower's credit file. pub_rec : Number of derogatory public records revol_bal : Total credit revolving balance revol_util : Revolving line utilization rate, or the amount of credit the borrower is using relative to all available revolving credit. total_acc : The total number of credit lines currently in the borrower's credit file initial_list_status : The initial listing status of the loan. Possible values are – W, F application_type : Indicates whether the loan is an individual application or a joint application with two co-borrowers mort_acc : Number of mortgage accounts. pub_rec_bankruptcies : Number of public record bankruptcies Address: Address of the individual

Libary Imports

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
import warnings
warnings.filterwarnings('ignore')
%matplotlib inline
```

```
pd.set_option('display.float', '{:.2f}'.format)
pd.set_option('display.max_columns', 50)
pd.set_option('display.max_rows', 50)
```

Data Importing

```
df = pd.read_csv("https://d2beiqkhq929f0.cloudfront.net/public_assets/assets/000/003/549/o
```

```
df.sample(10)
```

	loan_amnt	term	int_rate	installment	grade	sub_grade	emp_title	emp_
190787	20000.00	60 months	20.50	535.46	E	E4	Resident Care Coordinator	
114628	25000.00	36 months	10.99	818.35	B	B2	IT Manager	10
107998	10000.00	36 months	10.99	327.36	B	B4	Nor Cal Battery	10
147265	25000.00	60 months	10.99	543.44	B	B3	foreman	
168073	18000.00	60 months	23.40	511.58	E	E5	Teacher	10
145662	4800.00	36 months	14.33	164.83	C	C2	Pld development	
352666	7500.00	36 months	17.56	269.50	D	D1	Geeding construction	
57569	5000.00	60 months	9.99	106.22	B	B4	emanuel medical center	<

DATA SHAPE

```
df.shape
```

```
(396030, 27)
```

Statistical Summary

```
df.describe().T
```

	count	mean	std	min	25%	50%	75
loan_amnt	396030.00	14113.89	8357.44	500.00	8000.00	12000.00	20000.0
int_rate	396030.00	13.64	4.47	5.32	10.49	13.33	16.4
installment	396030.00	431.85	250.73	16.08	250.33	375.43	567.3
annual_inc	396030.00	74203.18	61637.62	0.00	45000.00	64000.00	90000.0
dti	396030.00	17.38	18.02	0.00	11.28	16.91	22.9
open_acc	396030.00	11.31	5.14	0.00	8.00	10.00	14.0
pub_rec	396030.00	0.18	0.53	0.00	0.00	0.00	0.0
revol_bal	396030.00	15844.54	20591.84	0.00	6025.00	11181.00	19620.0
revol_util	395754.00	53.79	24.45	0.00	35.80	54.80	72.9

Exploratory Data Analysis:

OVERALL GOAL:

Get an understanding for which variables are important, view summary statistics, and visualize the data

```
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
```

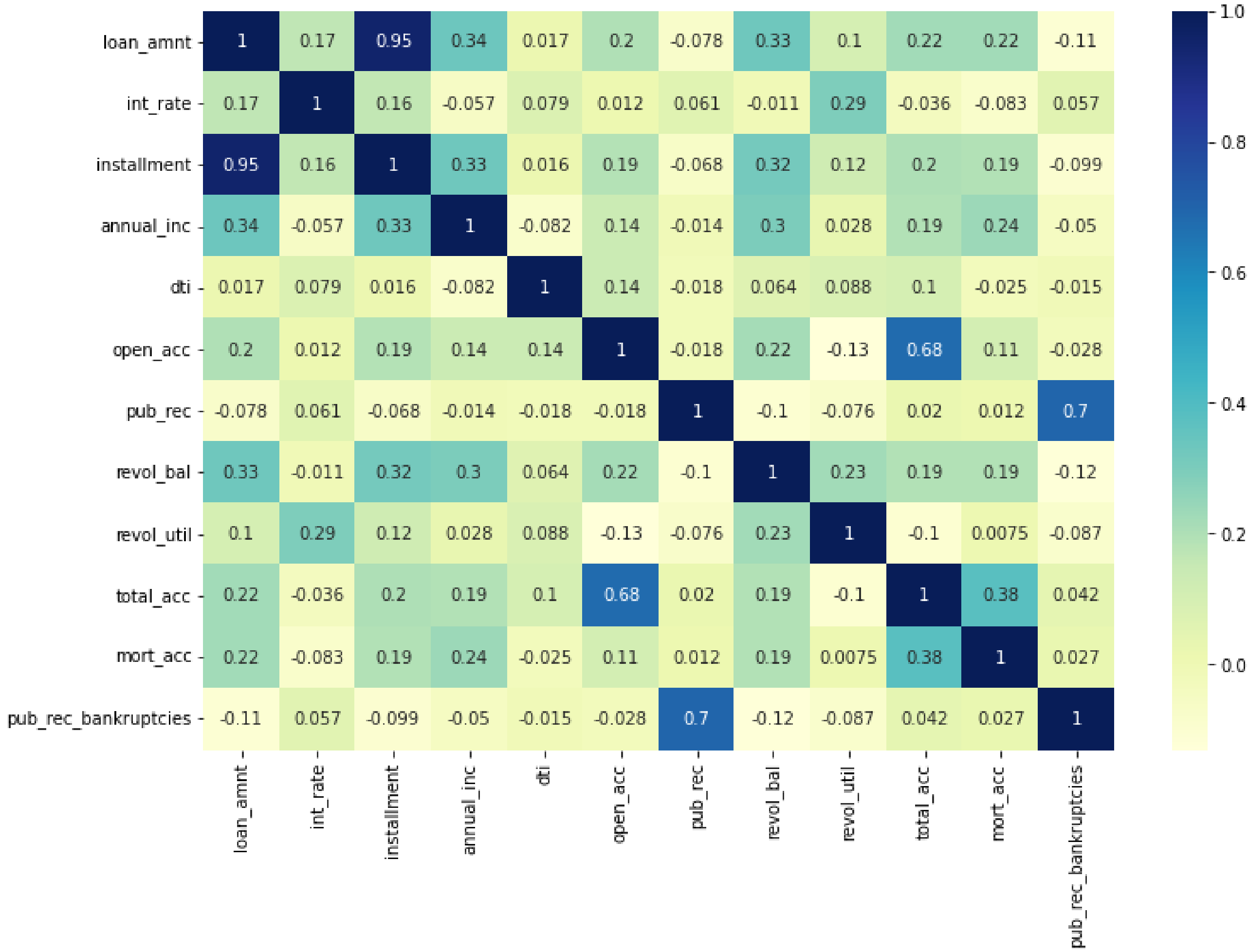
```
 24   mort_acc               358235 non-null  float64
 25   pub_rec_bankruptcies   395495 non-null  float64
 26   address                396030 non-null  object
dtypes: float64(12), object(15)
memory usage: 81.6+ MB
```

So, as we can see, there are no nulls in our dataset, which makes our life easier! 😄 But before moving forward, let's just check if there are any garbage data in our data set. Also, finding the unique values in each feature would give us a sense of the actual data present in our dataset.

HEAT MAP

```
plt.figure(figsize=(12, 8))
sns.heatmap(df.corr(), annot=True, cmap='YlGnBu')
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f85cd25d150>
```



We noticed almost perfect correlation between " `loan_amnt` " the " `installment` " feature. This is an imbalance problem, because we have a lot more entries of people that fully paid their loans then people that did not pay back. We can expect to probably do very well in terms of accuracy but our precision and recall are going to be the true metrics that we will have to evaluate our model based off of. In the loan amount distribution we can see spikes in even ten thousend dollar, so this is indicating that there are certain amounts that are basically standard loans.

```
df.corr().min()
```

```
loan_amnt              -0.11
int_rate               -0.08
installment            -0.10
annual_inc             -0.08
dti                    -0.08
open_acc               -0.13
pub_rec                -0.10
revol_bal              -0.12
revol_util             -0.13
total_acc              -0.10
mort_acc               -0.08
pub_rec_bankruptcies   -0.12
dtype: float64
```

UNIQUE VALUES

```
for column in df:
  print(column, end= ':- ')
  print(df[column].unique())
```

```
 'Dec-1991' 'May-2009' 'Aug-2011' 'Jun-1964' 'Jan-1974' 'May-1981'
 'Jun-1972' 'Jun-1978' 'Sep-1986' 'Jan-1987' 'Jan-1975' 'Feb-1982'
 'Jan-1980' 'Feb-1977' 'Sep-1980' 'Nov-1978' 'Jul-1974' 'Jun-1970'
 'Jan-1984' 'Nov-1980' 'May-1987' 'Sep-1970' 'Jan-1976' 'Feb-1986'
 'Oct-2010' 'Apr-1979' 'Oct-1979' 'Jan-1979' 'Sep-2011' 'Jul-1979'
 'Sep-1975' 'Mar-1981' 'Aug-1971' 'Apr-1980' 'Apr-1977' 'Jan-1965'
 'Nov-1976' 'Nov-1970' 'Nov-2011' 'Nov-1973' 'Sep-1981' 'Jul-1980'
 'Mar-2012' 'Dec-1974' 'Mar-1977' 'Dec-1977' 'May-2012' 'Dec-1979'
 'Jan-2009' 'Jan-1970' 'Dec-2011' 'Feb-1979' 'Mar-1976' 'Jan-1973'
 'Oct-1973' 'Mar-1969' 'Oct-1977' 'Mar-1975' 'Aug-1977' 'Jun-1969'
 'Oct-1963' 'Nov-1960' 'Aug-1970' 'Feb-1975' 'Sep-1974' 'May-1966'
 'Apr-1972' 'Apr-1973' 'Apr-2012' 'May-1975' 'Sep-1966' 'Feb-1969'
 'Feb-2012' 'Jan-1961' 'Aug-1973' 'Feb-1972' 'Apr-1975' 'Jul-1978'
 'Oct-1970' 'Mar-1980' 'Sep-1976' 'Apr-2011' 'Nov-2012' 'Aug-1976'
 'Jun-1975' 'Apr-1981' 'Mar-2009' 'Jun-1977' 'Apr-1971' 'Sep-1969'
 'Jun-2012' 'Apr-1976' 'Feb-1965' 'Jul-1977' 'Jun-1976' 'Mar-1973'
 'Oct-1972' 'Dec-1978' 'Nov-1967' 'Sep-1967' 'Nov-1971' 'Jun-1980'
 'May-1964' 'Feb-1971' 'May-1970' 'Apr-1970' 'Mar-1971' 'Apr-1969'
 'Jan-1963' 'Jun-1974' 'Oct-1974' 'May-1977' 'Dec-1981' 'Jan-1969'
 'Feb-1976' 'Mar-1970' 'Aug-1968' 'Feb-1970' 'Jun-1971' 'Jun-1963'
 'Jun-2013' 'Mar-1972' 'Aug-2012' 'Jan-1967' 'Feb-1968' 'Dec-1969'
 'Jan-1977' 'Jul-1970' 'Feb-1973' 'Mar-1974' 'Feb-1974' 'Dec-1960'
 'Jul-1972' 'Jul-1973' 'Sep-1964' 'Jul-1965' 'Oct-1958' 'Jul-2012'
 'Jun-1973' 'Sep-1978' 'Nov-1975' 'Jul-1963' 'Jan-1964' 'Dec-1968'
 'May-1958' 'Sep-1973' 'May-1971' 'Dec-1972' 'Aug-1965' 'Jul-1976'
 'Oct-2012' 'May-1973' 'Apr-1955' 'Apr-1966' 'Jan-1968' 'Nov-1968'
 'Oct-1969' 'Mar-2013' 'Jan-2013' 'Jul-1967' 'Oct-1965' 'Jan-1966'
 'Aug-1972' 'Jul-1969' 'May-1965' 'Jan-1953' 'Aug-1974' 'May-1968'
 'Aug-1969' 'May-2013' 'Oct-1967' 'Aug-1975' 'Apr-1974' 'Sep-1971'
```

```
 'Apr-1968' 'Jul-1971' 'Jan-1972' 'Nov-1965' 'Dec-1970' 'Dec-1973'
 'Nov-1972' 'Oct-1959' 'Oct-1962' 'Apr-1967' 'Oct-1971' 'Nov-1963'
 'Oct-1968' 'Dec-1962' 'Jun-1960' 'Jan-1960' 'Sep-2013' 'May-1969'
 'Dec-1966' 'Feb-1967' 'Dec-1967' 'Aug-1961' 'Sep-1968' 'Oct-1964'
 'Aug-1966' 'Jul-1966' 'Apr-1964' 'Sep-1962' 'Jul-2013' 'Jun-1967'
 'Apr-1965' 'Jun-1966' 'Jan-1955' 'Jan-1962' 'Feb-1964' 'Aug-1958'
 'Jul-1968' 'May-1967' 'Dec-1959' 'Sep-1963' 'Dec-2012' 'Dec-1963'
 'Jan-1944' 'Jun-1965' 'May-1962' 'Mar-1967' 'Mar-1968' 'Jan-1956'
 'Sep-1965' 'Dec-1951' 'Aug-2013' 'Jun-1968' 'Mar-1965' 'Oct-1957'
 'Nov-1966' 'Dec-1958' 'Feb-1957' 'Feb-1963' 'Mar-1963' 'Jan-1959'
 'May-1955' 'Feb-1966' 'Nov-1950' 'Mar-1964' 'Jan-1958' 'Nov-1964'
 'Sep-1961' 'Apr-1963' 'Jul-1964' 'Nov-1955' 'Jun-1957' 'Dec-1964'
 'Nov-1953' 'Apr-1961' 'Mar-1966' 'Oct-1960' 'Jul-1959' 'Jul-1961'
 'Jan-1954' 'Dec-1956' 'Mar-1962' 'Jul-1960' 'Sep-1959' 'Dec-1950'
 'Oct-1966' 'Apr-1960' 'Jul-1958' 'Nov-1954' 'Nov-1957' 'Jun-1962'
 'May-1963' 'Jul-1955' 'Oct-1950' 'Dec-1961' 'Aug-1951' 'Oct-2013'
 'Aug-1964' 'Apr-1962' 'Jun-1955' 'Jul-1962' 'Jan-1957' 'Nov-1958'
 'Jul-1951' 'Nov-1959' 'Apr-1958' 'Mar-1960' 'Sep-1957' 'Nov-1961'
 'Sep-1960' 'May-1959' 'Jun-1959' 'Feb-1962' 'Sep-1956' 'Aug-1960'
 'Feb-1961' 'Jan-1948' 'Aug-1963' 'Oct-1961' 'Aug-1962' 'Aug-1959']
open_acc:- [16. 17. 13.  6.  8. 11.  5. 30.  9. 15. 12. 10. 18.  7.  4. 14. 20. 19
 21. 23.  3. 26. 42. 22. 25. 28.  2. 34. 24. 27. 31. 32. 33.  1. 29. 36.
 40. 35. 37. 41. 44. 39. 49. 48. 38. 51. 50. 43. 46.  0. 47. 57. 53. 58.
 52. 54. 45. 90. 56. 55. 76.]
pub_rec:- [ 0.  1.  2.  3.  4.  6.  5.  8.  9. 10. 11.  7. 19. 13. 40. 17. 86. 12.
 24. 15.]
revol_bal:- [ 36369.  20131.  11987. ...  34531. 151912.  29244.]
revol_util:- [ 41.8   53.3   92.2  ...   56.26 111.4  128.1 ]
```

VALUE COUNT

```
for column in df:
  print(column, end= ':- ')
  print(df[column].value_counts())
```

```
F2     2766
F3     2286
F4     1787
F5     1397
G1     1058
G2      754
G3      552
G4      374
G5      316
Name: sub_grade, dtype: int64
emp_title:- Teacher                    4389
Manager                     4250
Registered Nurse            1856
RN                          1846
Supervisor                  1830
                            ...
Postman                        1
McCarthy & Holthus, LLC        1
jp flooring                    1
Histology Technologist         1
Gracon Services, Inc           1
Name: emp_title, Length: 173105, dtype: int64
```

```
emp_length:- 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
home_ownership:- MORTGAGE    198348
RENT        159790
OWN          37746
OTHER          112
NONE            31
ANY              3
Name: home_ownership, dtype: int64
annual_inc:- 60000.00    15313
50000.00    13303
65000.00    11333
70000.00    10674
40000.00    10629
            ...
72179.00        1
50416.00        1
46820.80        1
10368.00        1
31789.88        1
Name: annual_inc, Length: 27197, dtype: int64
verification_status:- Verified          139563
Source Verified    131385
Not Verified       125082
Name: verification_status, dtype: int64
issue_d:- Oct-2014    14846
```

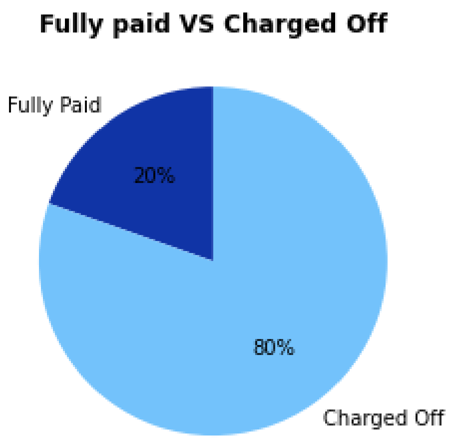
DROP DUPLICATES

```
df.drop_duplicates(inplace= True)
```

So, no bad or duplicate data in our Dataset and no null values either! Now, lets do some basic data analysis, like finding the mean, median, deviation of the data in this data frame

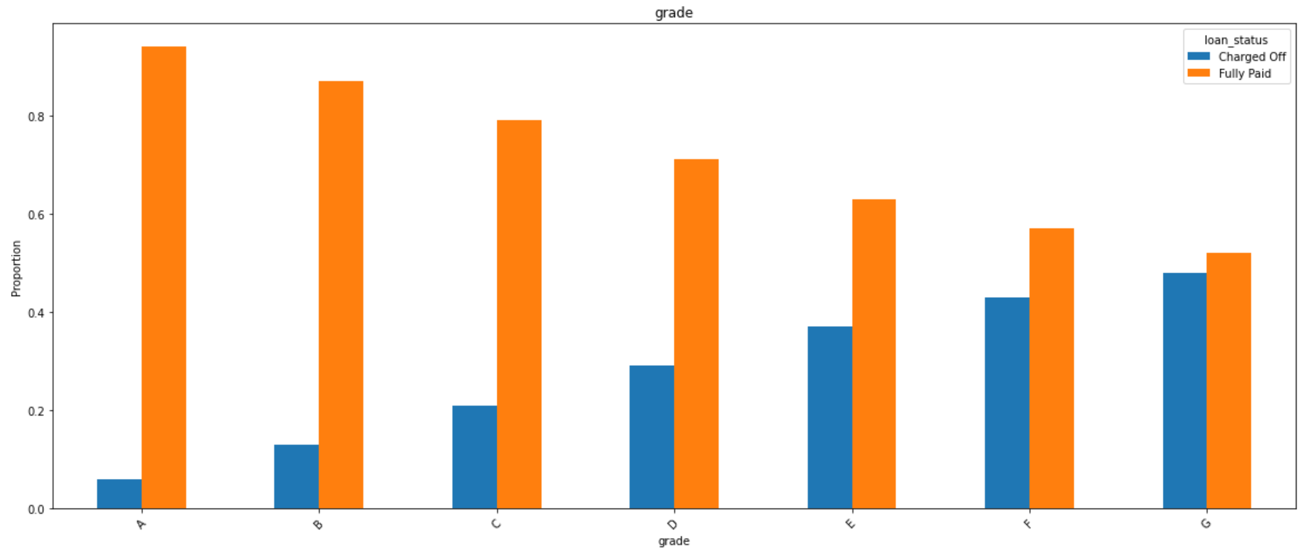
Univariate Analysis

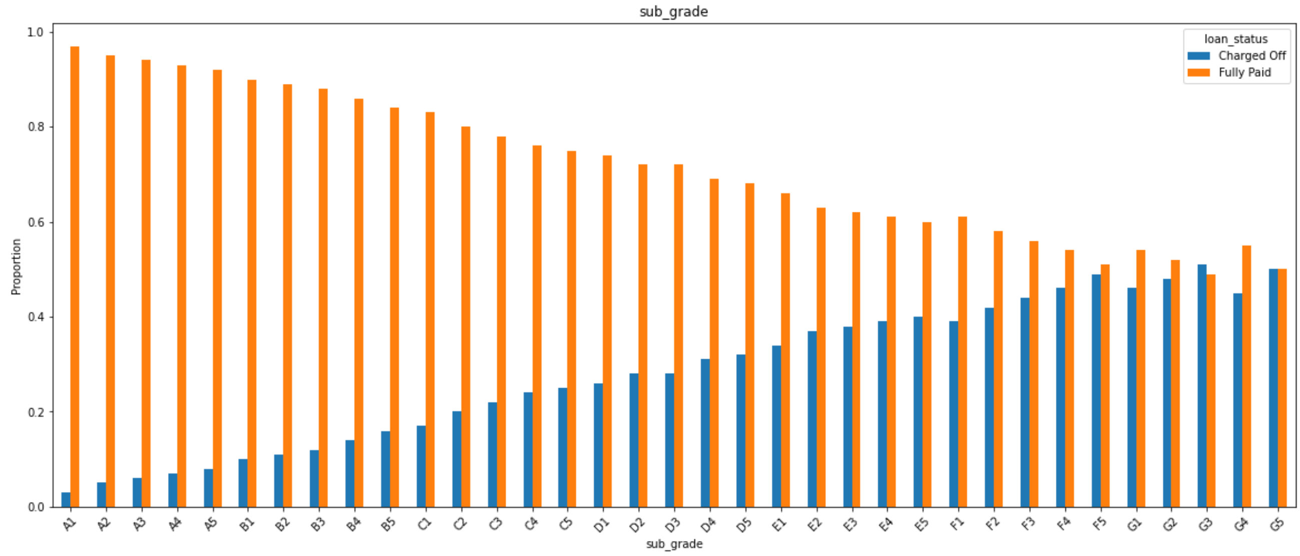
```
products=df.groupby(['loan_status'])['loan_status'].count()
plt.pie(products, labels=['Fully Paid','Charged Off'], colors=['#1034A6','#73C2FB','#74B3C
plt.title('Fully paid VS Charged Off',fontweight="bold")
plt.show()
```

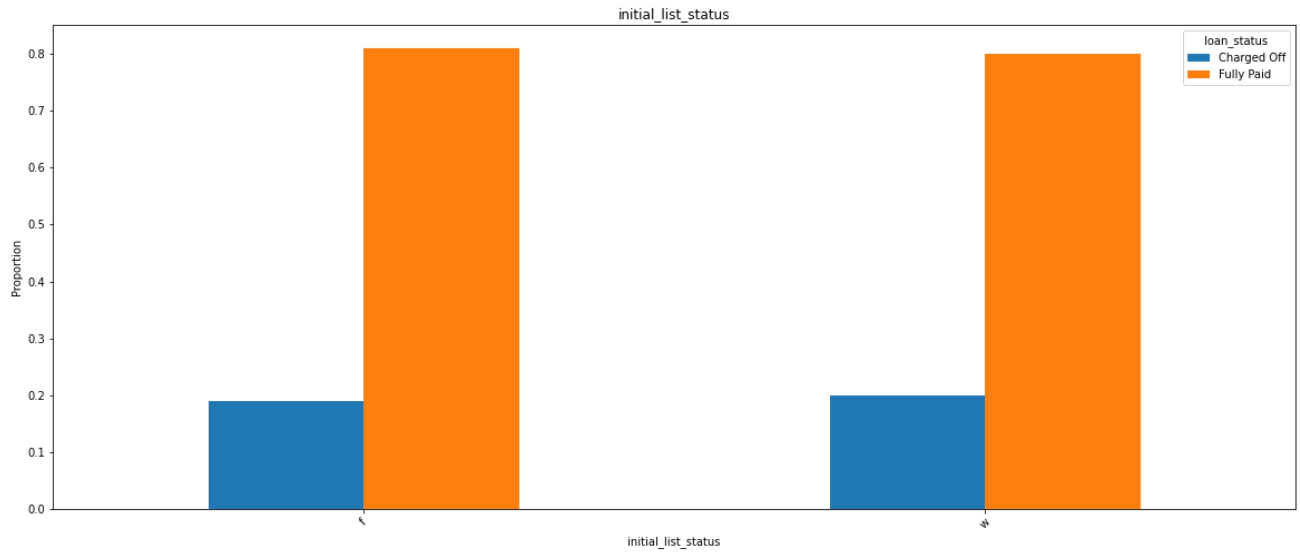


1. 80% of the customers are not paying the loan
2. 20% of the loan are re-paid

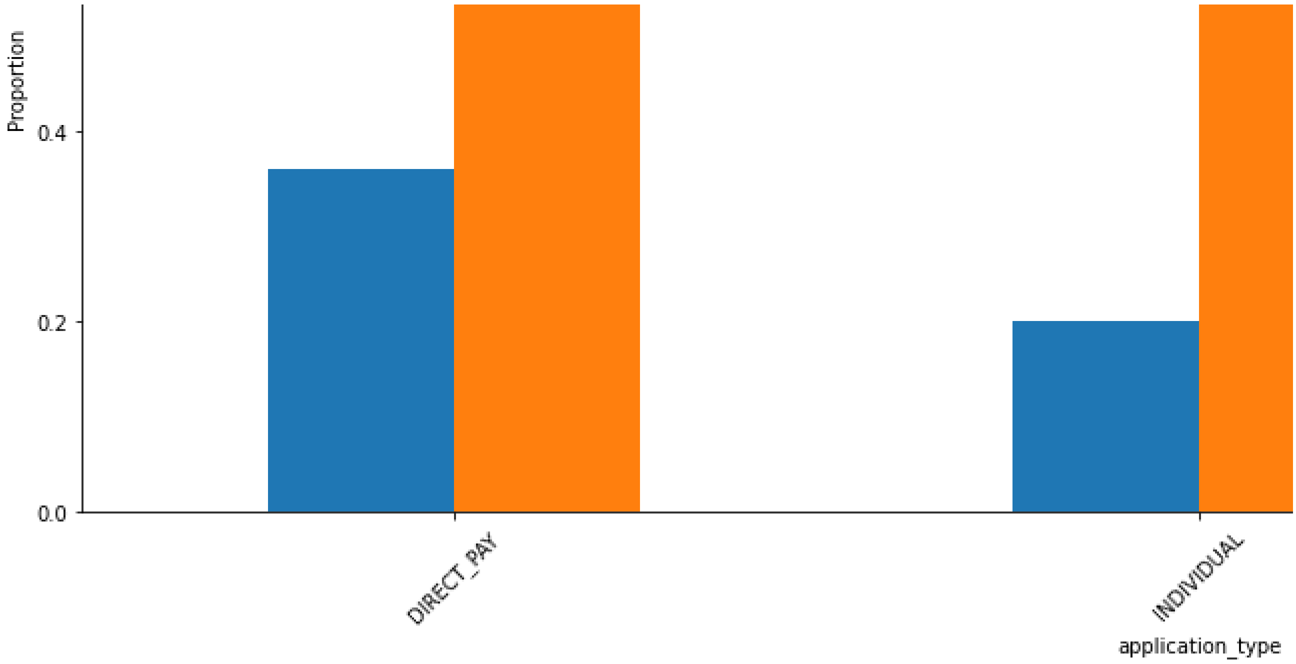
```
cat_cols=['grade', 'sub_grade','initial_list_status', 'application_type','pub_rec_bankrupt
for i in cat_cols:
 other= round(pd.crosstab(df[df[i].notnull()][i], df['loan_status']).\
 div(pd.crosstab(df[df[i].notnull()][i],df['loan_status']).apply(sum,1),0),2)
 ax = other.plot(kind ='bar', title = i, figsize = (20,8))
 ax.set_xlabel(i)
 ax.set_ylabel('Proportion')
 plt.xticks(rotation=45)
 plt.show()
```

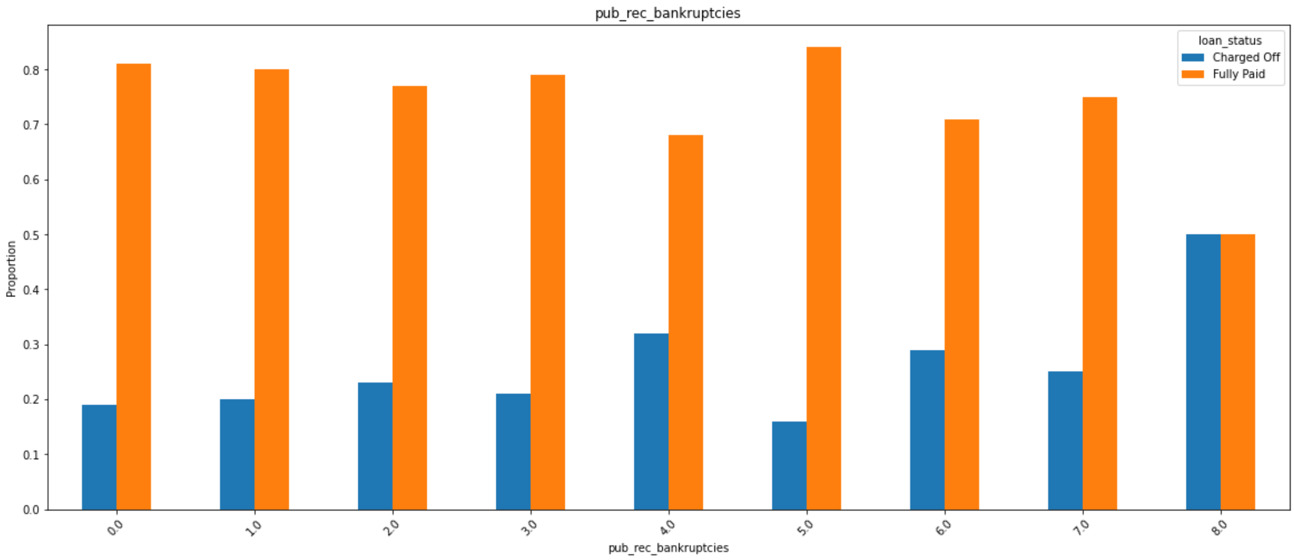

grade
loan_status
Charged Off
Fully Paid
Proportion
0.0
0.2
0.4
0.6
0.8
A
B
C
D
E
F
G
grade

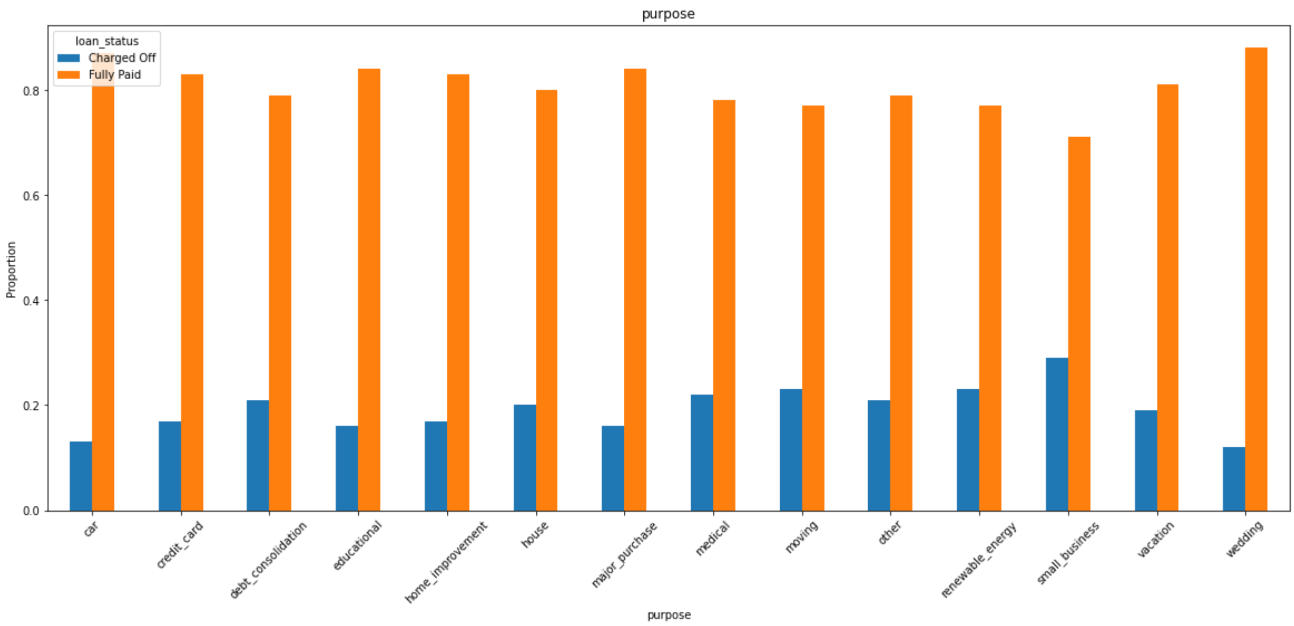

sub_grade
loan_status
Charged Off
Fully Paid
Proportion
0.0
0.2
0.4
0.6
0.8
1.0
A1 A2 A3 A4 A5 B1 B2 B3 B4 B5 C1 C2 C3 C4 C5 D1 D2 D3 D4 D5 E1 E2 E3 E4 E5 F1 F2 F3 F4 F5 G1 G2 G3 G4 G5
sub_grade

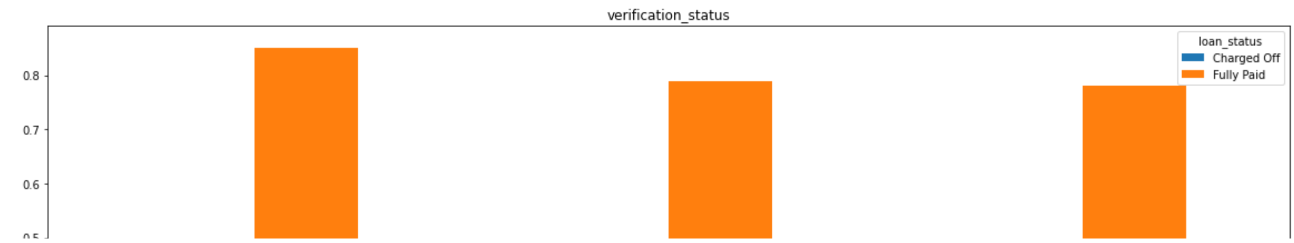

initial_list_status
loan_status
Charged Off
Fully Paid
Proportion
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
f
w
initial_list_status

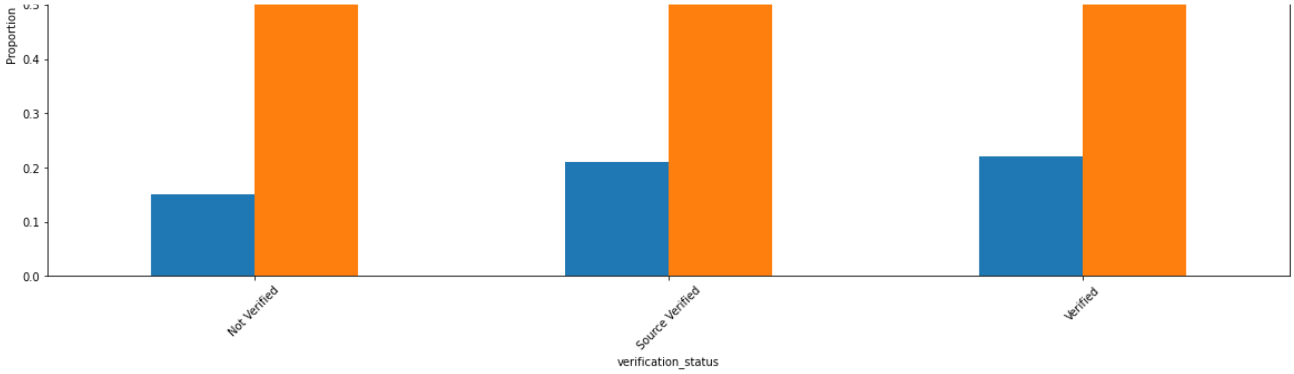

application_type
loan_status
Charged Off
Fully Paid
0.8
0.6

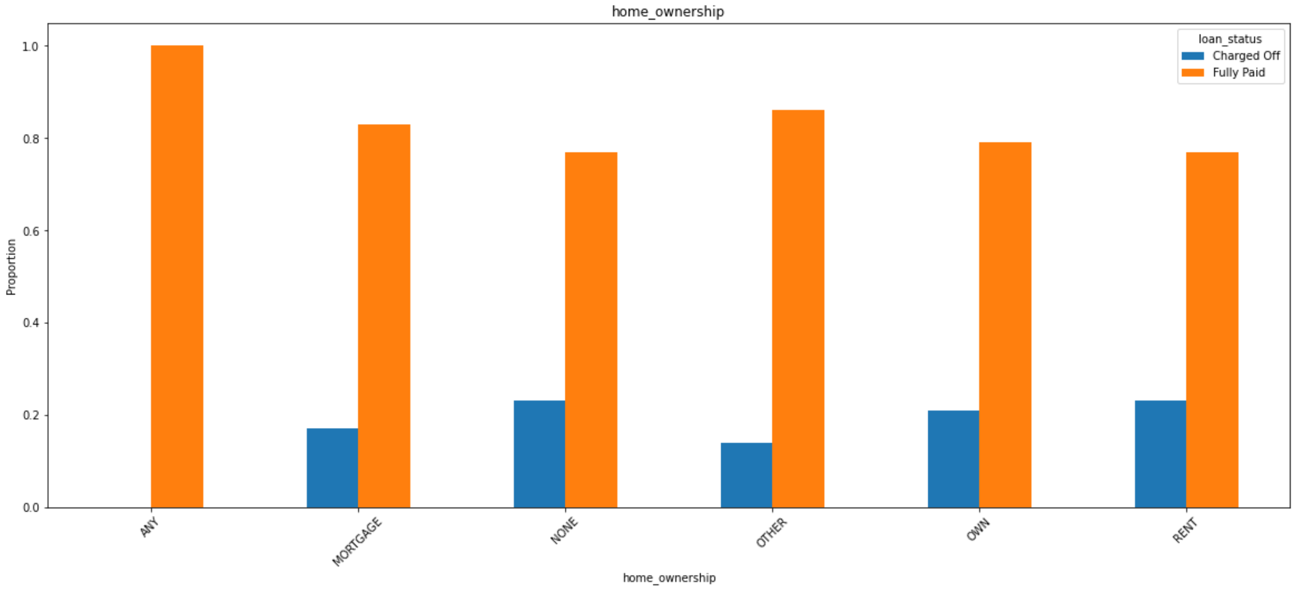

Proportion
0.4
0.2
0.0
DIRECT_PAY
INDIVIDUAL
application_type

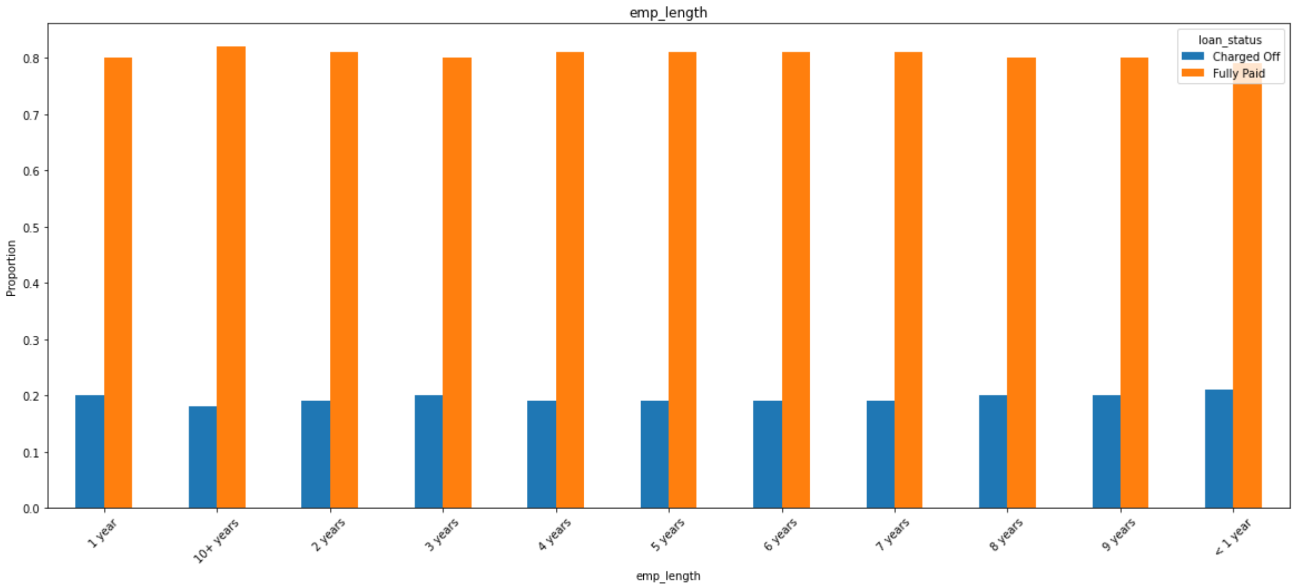

pub_rec_bankruptcies
loan_status
Charged Off
Fully Paid
Proportion
0.0
1.0
2.0
3.0
4.0
5.0
6.0
7.0
8.0
pub_rec_bankruptcies

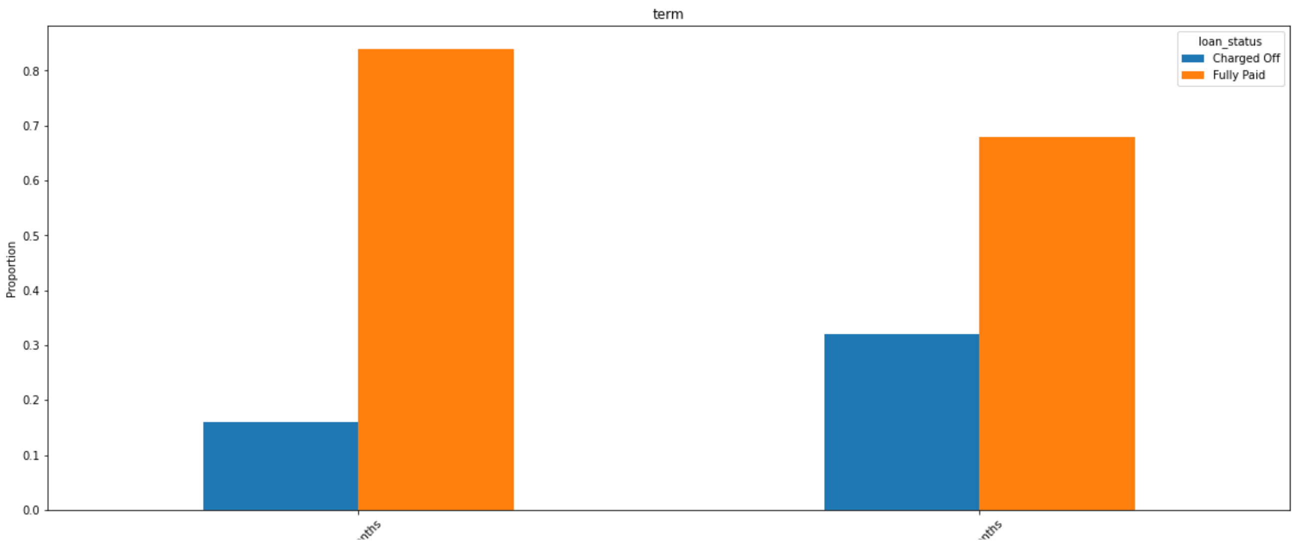

purpose
loan_status
Charged Off
Fully Paid
Proportion
car
credit_card
debt_consolidation
educational
home_improvement
house
major_purchase
medical
moving
other
renewable_energy
small_business
vacation
wedding
purpose


verification_status
loan_status
Charged Off
Fully Paid


Proportion
0.0
0.1
0.2
0.3
0.4
Not Verified
Source Verified
Verified
verification_status


home_ownership
loan_status
Charged Off
Fully Paid
Proportion
0.0
0.2
0.4
0.6
0.8
1.0
ANY
MORTGAGE
NONE
OTHER
OWN
RENT
home_ownership


emp_length
loan_status
Charged Off
Fully Paid
Proportion
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
1 year
10+ years
2 years
3 years
4 years
5 years
6 years
7 years
8 years
9 years
< 1 year
emp_length

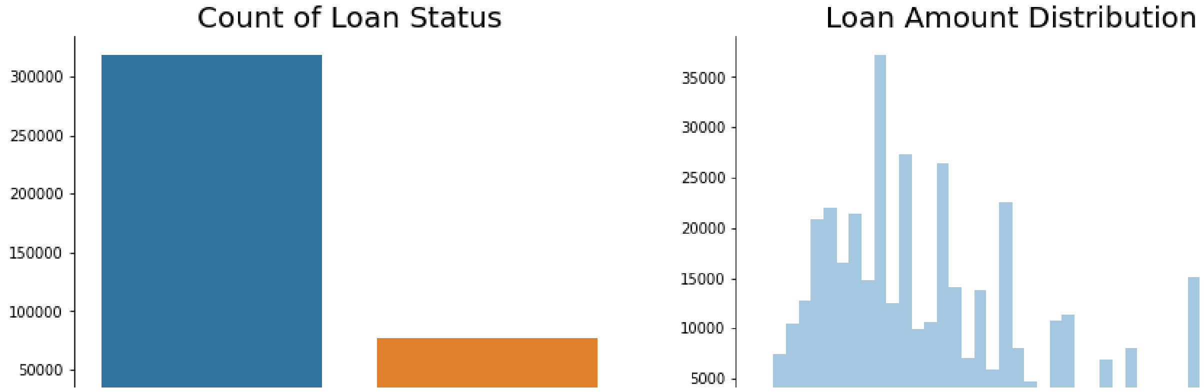

term
loan_status
Charged Off
Fully Paid
Proportion
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8


36 mo
60 mo
term

1. It looks like `F` and `G` subgrades don't get paid back that often. Isloate those and recreate the countplot just for those subgrades.
2. Customer choose 36 months are fully paos compare to 60 months
3. Fully paid customers are choosing loan are major in buying for car and wedding

```
f, axes = plt.subplots(1, 2, figsize=(15,5))
sns.countplot(x='loan_status', data=df, ax=axes[0])
sns.distplot(df['loan_amnt'], kde=False, bins=40, ax=axes[1])
sns.despine()
axes[0].set(xlabel='Status', ylabel='')
axes[0].set_title('Count of Loan Status', size=20)
axes[1].set(xlabel='Loan Amount', ylabel='')
axes[1].set_title('Loan Amount Distribution', size=20)
```

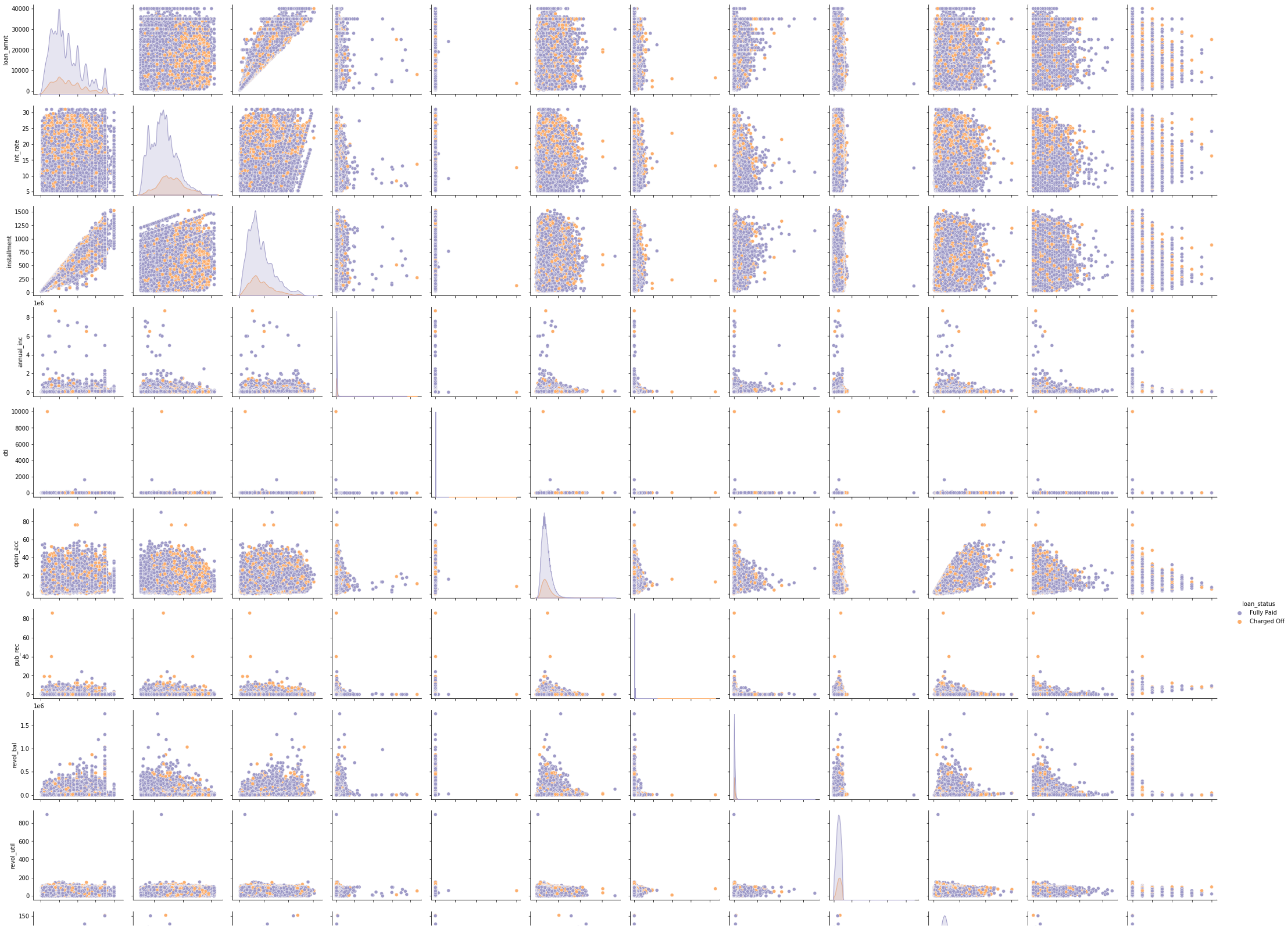
```
Text(0.5, 1.0, 'Loan Amount Distribution')
```




```
plt.figure(figsize=(15, 12))
plt.barh(df.emp_title.value_counts()[:30].index, df.emp_title.value_counts()[:30])
plt.title("The most 30 jobs title afforded a loan")
plt.tight_layout()
```

1. Teacher and Manager are taking more loan
2. Supervisor,Nurse and RN are secondard of taking the loan

```
#scatterplot
sns.pairplot(hue='loan_status',data= df,palette='tab20c_r')
plt.show();
```



Missing Values

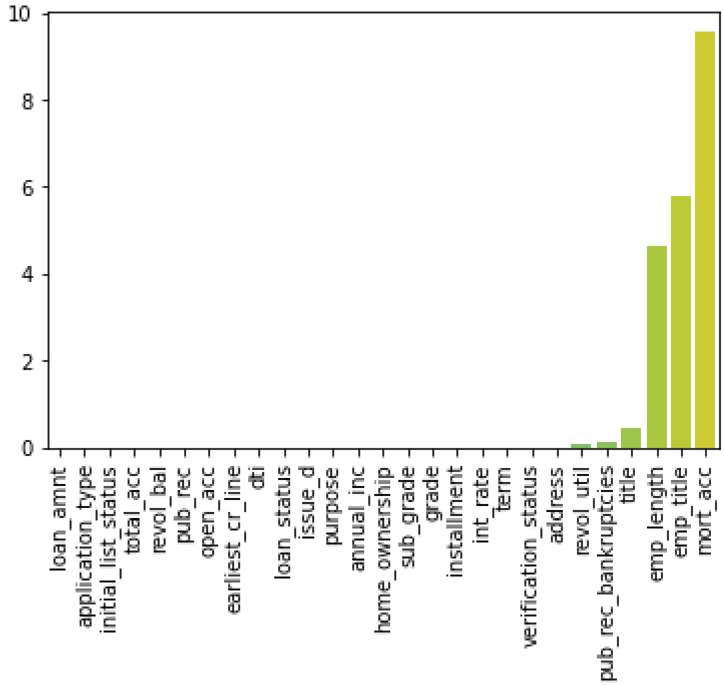
```
# Checking percentage of missing values after removing the missing values
round(100*(df.isnull().sum()/len(df.index)), 2)
```

```
loan_amnt               0.00
term                    0.00
int_rate                0.00
installment             0.00
grade                   0.00
sub_grade               0.00
emp_title               5.79
emp_length              4.62
home_ownership          0.00
annual_inc              0.00
verification_status     0.00
issue_d                 0.00
loan_status             0.00
purpose                 0.00
title                   0.44
dti                     0.00
earliest_cr_line        0.00
open_acc                0.00
pub_rec                 0.00
revol_bal               0.00
revol_util              0.07
total_acc               0.00
initial_list_status     0.00
application_type        0.00
```

```
mort_acc                9.54
pub_rec_bankruptcies    0.14
address                 0.00
dtype: float64
```

We have missing values in emp_title, emp_length, title, revol_util, mort_acc and pub_rec_bankruptcies

```
# y contains the % of missing values od each column and x is the index of the series in ab
sns.barplot(y=((df.isnull().sum()/len(df))*100).sort_values(), x=((df.isnull().sum()/len(d
plt.xticks(rotation=90); # to rotate x-axis labels from horizontal to vertical
```



Missing column values

In the plot we can see how much data is missing as a percentage of the total data. Notice that there is missing almost 10% of mortgage accounts, so we can not drop all those rows. On the other hand, we could drop missing values in revol_util or pub_rec_bankruptcies.

```
#Copy the data frame to further work
data = df.copy()
```

```
#Drop Null Rows
data.dropna(inplace=True)
```

```
# Data shape
data.shape
```

```
(335868, 27)
```

Double-click (or enter) to edit

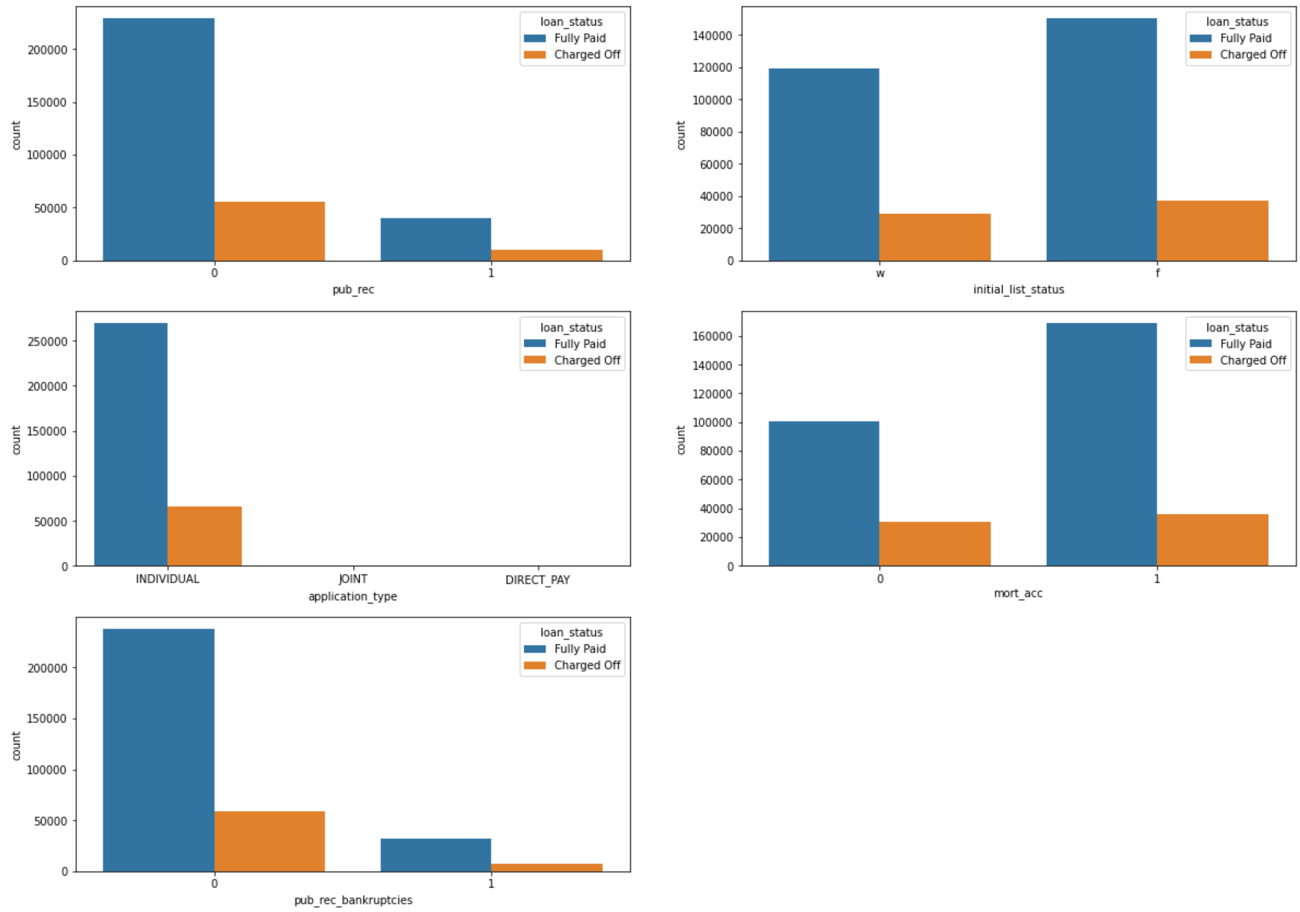
Feature Engineering

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

def mort_acc(number):
    if number == 0.0:
        return 0
    elif number >= 1.0:
        return 1
    else:
        return number

def pub_rec_bankruptcies(number):
    if number == 0.0:
        return 0
    elif number >= 1.0:
        return 1
    else:
        return number
```

```
data['pub_rec'] = data.pub_rec.apply(pub_rec)
data['mort_acc'] = data.mort_acc.apply(mort_acc)
data['pub_rec_bankruptcies'] = data.pub_rec_bankruptcies.apply(pub_rec_bankruptcies)
```

```
plt.figure(figsize=(20, 30))

plt.subplot(6, 2, 1)
sns.countplot(x='pub_rec', data=data, hue='loan_status')

plt.subplot(6, 2, 2)
sns.countplot(x='initial_list_status', data=data, hue='loan_status')

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

plt.subplot(6, 2, 4)
sns.countplot(x='mort_acc', data=data, hue='loan_status')

plt.subplot(6, 2, 5)
sns.countplot(x='pub_rec_bankruptcies', data=data, hue='loan_status')
```

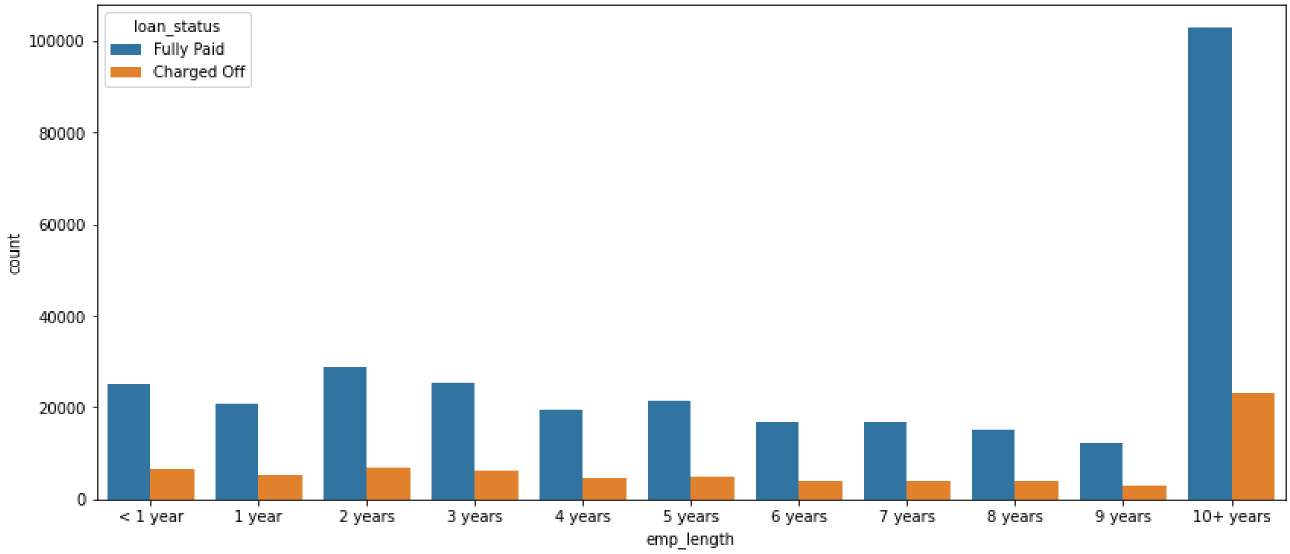
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f85cf136610>
```




```
data['emp_length'].replace(to_replace='10+ years', value='10 years', inplace=True)
data['emp_length'].replace('< 1 year', '0 years', inplace=True)


def emp_length_to_int(s):
    if pd.isnull(s):
        return s
    else:
        return np.int8(s.split()[0])


df_new = data[data['emp_length'].notnull()]
```

```
emp_length_order = [ '< 1 year', '1 year', '2 years', '3 years', '4 years', '5 years', '6
plt.figure(figsize=(14,6))
sns.countplot(x='emp_length',data=df,order=emp_length_order,hue='loan_status')
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f85c3998090>
```



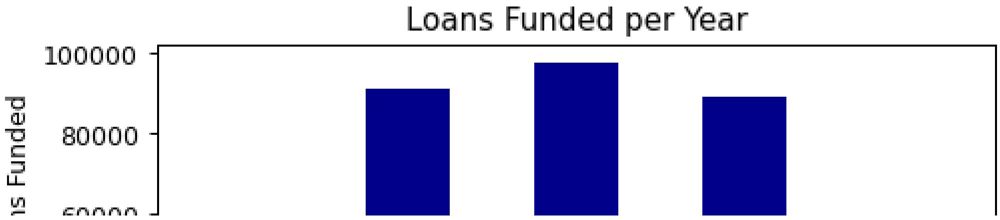

```
data['issue_d'] = pd.to_datetime(data['issue_d'])
```

```
data['issue_d'].describe()
```

```
count                  335868
unique                     58
top       2014-10-01 00:00:00
freq                    14133
first     2012-03-01 00:00:00
last      2016-12-01 00:00:00
Name: issue_d, dtype: object
```

```
plt.figure(figsize=(6,3), dpi=90)
data['issue_d'].dt.year.value_counts().sort_index().plot.bar(color='darkblue')
plt.xlabel('Year')
plt.ylabel('Number of Loans Funded')
plt.title('Loans Funded per Year')
```

```
Text(0.5, 1.0, 'Loans Funded per Year')
```



Data PreProcessing

Section Goals:

> - Remove or fill any missing data.
> - Remove unnecessary or repetitive features.
> - Convert categorical string features to dummy variables.

```
total_acc_avg = data.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
```

```
data['mort_acc'] = data.apply(lambda x: fill_mort_acc(x['total_acc'], x['mort_acc']), axis
```

```
data.drop(inplace=True,axis=1,labels=['emp_title','title','earliest_cr_line','address','gr
```

```
data.term  = data['term'].apply(lambda x:x.split(' ')[1])
```

```
print(f"Data shape: {data.shape}")
```

```
Data shape: (335868, 22)
```

```
data['loan_status'] = data.loan_status.map({'Fully Paid':1, 'Charged Off':0})
```

```
dummies = ['sub_grade', 'verification_status', 'purpose', 'initial_list_status',
           'application_type', 'home_ownership']
data = pd.get_dummies(data, columns=dummies, drop_first=True)
```

```
data.head(3)
```

	loan_amnt	term	int_rate	installment	emp_length	annual_inc	issue_d	loan_stat
0	10000.00	36	11.44	329.48	10 years	117000.00	2015-01-01	
1	8000.00	36	11.99	265.68	4 years	65000.00	2015-01-01	
2	15600.00	36	10.49	506.97	0 years	43057.00	2015-01-01	

Scaling and Test Train split

Double-click (or enter) to edit

```
x = data.drop(labels = ['loan_status','emp_length','issue_d'],axis =1)
```

```
x.shape
```

```
(335868, 70)
```

```
y = data['loan_status'].values.ravel()
```

```
y.shape
```

```
(335868,)
```

```
# machine learning
from sklearn.linear_model import LogisticRegression
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, f1_score, precision_score, recall_score
from statsmodels.stats.outliers_influence import variance_inflation_factor
```

```
X_train,X_test,y_train,y_test=train_test_split(x,y,train_size=0.7,test_size=0.3,random_sta
log_reg = LogisticRegression()
log_reg.fit(X_train,y_train)
y_pred = log_reg.predict(X_test)
```

We will evaluate and compare the following models using a cross-validated AUROC score on the training set:

Logistic regression with SGD training Random forest k-nearest neighbors We'll perform some hyperparameter tuning for each model to choose the most promising model, then more carefully tune the hyperparameters of the best-performing model.

```
from sklearn.pipeline import Pipeline
```

```
from sklearn.preprocessing import StandardScaler
from sklearn.impute import SimpleImputer
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
from sklearn.model_selection import GridSearchCV
```

8.1 Logistic regression with SGD training

Double-click (or enter) to edit

```
from sklearn.linear_model import SGDClassifier
```

```
pipeline_sgdlogreg = Pipeline([
    ('imputer', SimpleImputer(copy=False)), # Mean imputation by default
    ('scaler', StandardScaler(copy=False)),
    ('model', SGDClassifier(loss='log', max_iter=1000, tol=1e-3, random_state=1, warm_star
])
```

```
param_grid_sgdlogreg = {
    'model__alpha': [10**-5, 10**-2, 10**1],
    'model__penalty': ['l1', 'l2']
}
```

```
grid_sgdlogreg = GridSearchCV(estimator=pipeline_sgdlogreg, param_grid=param_grid_sgdlogre
```

Conduct the grid search and train the final model on the whole dataset:

```
grid_sgdlogreg.fit(X_train, y_train)
```

```
Fitting 5 folds for each of 6 candidates, totalling 30 fits
GridSearchCV(cv=5,
             estimator=Pipeline(steps=[('imputer', SimpleImputer(copy=False)),
                                       ('scaler', StandardScaler(copy=False)),
                                       ('model',
                                        SGDClassifier(loss='log',
                                                      random_state=1,
                                                      warm_start=True))]),
             n_jobs=1,
             param_grid={'model__alpha': [1e-05, 0.01, 10],
                         'model__penalty': ['l1', 'l2']},
             pre_dispatch=1, scoring='roc_auc', verbose=1)
```

Mean cross-validated AUROC score of the best model:

```
grid_sgdlogreg.best_score_
```

```
0.711375193558917
```

Best hyperparameters:

```
grid_sgdlogreg.best_params_
```

```
{'model__alpha': 0.01, 'model__penalty': 'l2'}
```

Random forest classifier

Next we train a random forest model. Note that data standardization is not necessary for a random forest.

```
from sklearn.ensemble import RandomForestClassifier
```

```
pipeline_rfc = Pipeline([
    ('imputer', SimpleImputer(copy=False)),
    ('model', RandomForestClassifier(n_jobs=-1, random_state=1))
])
```

The random forest takes very long to train, so we don't test different hyperparameter choices. We'll still use GridSearchCV for the sake of consistency.

```
param_grid_rfc = {
    'model__n_estimators': [50] # The number of randomized trees to build
}
```

The AUROC will always improve (with decreasing gains) as the number of estimators increases, but it's not necessarily worth the extra training time and model complexity.

```
grid_rfc = GridSearchCV(estimator=pipeline_rfc, param_grid=param_grid_rfc, scoring='roc_au
```

```
grid_rfc.fit(X_train, y_train)
```

```
Fitting 5 folds for each of 1 candidates, totalling 5 fits
GridSearchCV(cv=5,
             estimator=Pipeline(steps=[('imputer', SimpleImputer(copy=False)),
                                       ('model',
                                        RandomForestClassifier(n_jobs=-1,
                                                               random_state=1))]),
             n_jobs=1, param_grid={'model__n_estimators': [50]}, pre_dispatch=1,
             scoring='roc_auc', verbose=1)
```

Mean cross-validated AUROC score of the random forest:

```
grid_rfc.best_score_
```

```
0.6950255847657141
```

Not quite as good as logistic regression, at least according to this metric.

```
from sklearn.metrics import roc_auc_score
```

```
y_score = grid_sgdlogreg.predict_proba(X_test)[:,1]
roc_auc_score(y_test, y_score)
```

```
0.711102812752983
```

The test set AUROC score is somewhat lower than the cross-validated score (0.713).

```
# Accuracy
accuracy = accuracy_score(y_test,y_pred)
print("Accuracy:- ",accuracy)
```

```
Accuracy:-  0.8011929218646103
```

```
from sklearn.metrics import confusion_matrix
```

```
# Confusion Matrix
conf_mat = confusion_matrix(y_test,y_pred)
conf_mat
```

```
array([[    5, 19983],
       [   49, 80724]])
```

Confusion Matrix

1. A confusion matrix is a technique for summarizing the performance of a classification algorithm.
2. Classification accuracy alone can be misleading if you have an unequal number of observations in each class, which is our case.
3. We have 308 Type I errors (False Positive) and 8562 Type II errors (False Negative). 7096 True Positive and 63078 True Negative.

```
true_positive = conf_mat[0][0]
false_positive = conf_mat[0][1]
false_negative = conf_mat[1][0]
true_negative = conf_mat[1][1]
# Breaking down the formula for Accuracy (Manual Checking)
Accuracy = (true_positive + true_negative) / (true_positive +false_positive + false_negati
Accuracy
```

```
0.8011929218646103
```

```
# Precison
Precision = true_positive/(true_positive+false_positive)
print("Precision:- ",Precision)
```

```
Precision:-  0.0002501500900540324
```

```
# Recall
Recall = true_positive/(true_positive+false_negative)
print("Recall:- ",Recall)
```

```
Recall:-  0.09259259259259259
```

```
log_reg=LogisticRegression(C=1000,max_iter=10000)
log_reg.fit(X_train,y_train)
print('--------------------------------------------------------------------------')
print('Logistic Regression:')
print('Traning Model accruracy scores: {:.3f}'.format(log_reg.score(X_train,y_train)))
print('Test Model accruracy scores: {:.3f}'.format(log_reg.score(X_test,y_test)))
print('--------------------------------------------------------------------------')
```

```
--------------------------------------------------------------------------
Logistic Regression:
Traning Model accruracy scores: 0.798
Test Model accruracy scores: 0.797
--------------------------------------------------------------------------
```

Feature Selection Using RFE

An ROC curve demonstrates several things:

- It shows the tradeoff between sensitivity and specificity (any increase in sensitivity will be accompanied by a decrease in specificity).
- The closer the curve follows the left-hand border and then the top border of the ROC space, the more accurate the test.
- The closer the curve comes to the 45-degree diagonal of the ROC space, the less accurate the test.

```
from sklearn.metrics import roc_auc_score
```

```
y_score = grid_sgdlogreg.predict_proba(X_test)[:,1]
roc_auc_score(y_test, y_score)
```

```
0.711102812752983
```