Business Case - LoanTap - Logistic Regression

Context

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.

LoanTap deploys formal credit to salaried individuals and businesses 5 main financial instruments:

Personal Loan
EMI Free Loan
Personal Overdraft
Advance Salary Loan

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

Business Problem

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?

Column Profiling

- 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
- installment : 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

In [160]:

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

In [161]:

```
## importing packages for linear regression
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import StandardScaler
from sklearn import linear_model
```

In [162]:

```
df = pd.read_csv("logistic_regression.csv")
```

In [163]:

```
df.head().T
```

Out[163]:

	0	1	2	
loan_amnt	10000.0	8000.0	15600.0	720
term	36 months	36 months	36 months	36 mor
int_rate	11.44	11.99	10.49	6
installment	329.48	265.68	506.97	220
grade	B	B	B	
sub_grade	B4	B5	B3	
emp_title	Marketing	Credit analyst	Statistician	Client Advoc
emp_length	10+ years	4 years	< 1 year	6 ye
home_ownership	RENT	MORTGAGE	RENT	RE
annual_inc	117000.0	65000.0	43057.0	5400
verification_status	Not Verified	Not Verified	Source Verified	Not Veri
issue_d	Jan-2015	Jan-2015	Jan-2015	Nov-2
loan_status	Fully Paid	Fully Paid	Fully Paid	Fully F
purpose	vacation	debt_consolidation	credit_card	credit_c
title	Vacation	Debt consolidation	Credit card refinancing	Credit c refinanc
dti	26.24	22.05	12.79	
earliest_cr_line	Jun-1990	Jul-2004	Aug-2007	Sep-2
open_acc	16.0	17.0	13.0	
pub_rec	0.0	0.0	0.0	
revol_bal	36369.0	20131.0	11987.0	547
revol_util	41.8	53.3	92.2	2
total_acc	25.0	27.0	26.0	1
initial_list_status	w	f	f	
application_type	INDIVIDUAL	INDIVIDUAL	INDIVIDUAL	INDIVIDU
mort_acc	0.0	3.0	0.0	
pub_rec_bankruptcies	0.0	0.0	0.0	
address	0174 Michelle Gateway\r\nMendozaberg, OK 22690	1076 Carney Fort Apt. 347\r\nLoganmouth, SD 05113	87025 Mark Dale Apt. 269\r\nNew Sabrina, WV 05113	823 R Ford\r\nDelacruzs MA 00

Checking Shape and Column Names

In [164]:

```
df.shape
```

Out[164]:

```
(396030, 27)
```

There are 396030 data points and 27 columns of data

In [165]:

```
#checking column names
df.columns
```

Out[165]:

```
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')
```

In [166]:

```
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
```

In [167]:

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

Out[167]:

	count	mean	std	min	25%	50%	7
loan_amnt	396030.0	14113.888089	8357.441341	500.00	8000.00	12000.00	20000
int_rate	396030.0	13.639400	4.472157	5.32	10.49	13.33	16
installment	396030.0	431.849698	250.727790	16.08	250.33	375.43	567
annual_inc	396030.0	74203.175798	61637.621158	0.00	45000.00	64000.00	90000
dti	396030.0	17.379514	18.019092	0.00	11.28	16.91	22
open_acc	396030.0	11.311153	5.137649	0.00	8.00	10.00	14
pub_rec	396030.0	0.178191	0.530671	0.00	0.00	0.00	0
revol_bal	396030.0	15844.539853	20591.836109	0.00	6025.00	11181.00	19620
revol_util	395754.0	53.791749	24.452193	0.00	35.80	54.80	72
total_acc	396030.0	25.414744	11.886991	2.00	17.00	24.00	32
mort_acc	358235.0	1.813991	2.147930	0.00	0.00	1.00	3
pub_rec_bankruptcies	395495.0	0.121648	0.356174	0.00	0.00	0.00	0

In [168]:

```
df.describe(include = "object").T
```

Out[168]:

	count	unique	top	freq
term	396030	2	36 months	302005
grade	396030	7	B	116018
sub_grade	396030	35	B3	26655
emp_title	373103	173105	Teacher	4389
emp_length	377729	11	10+ years	126041
home_ownership	396030	6	MORTGAGE	198348
verification_status	396030	3	Verified	139563
issue_d	396030	115	Oct-2014	14846
loan_status	396030	2	Fully Paid	318357
purpose	396030	14	debt_consolidation	234507
title	394275	48817	Debt consolidation	152472
earliest_cr_line	396030	684	Oct-2000	3017
initial_list_status	396030	2	f	238066
application_type	396030	3	INDIVIDUAL	395319
address	396030	393700	USS Johnson\r\nFPO AE 48052	8

- It can be seen that all the numerical variables have very high maximum values in comparison to 50 and 75 percentile and therefore there is outliers.

In [169]:

```
df.isnull().sum()
```

Out[169]:

```
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
```

In [170]:

```
# Displaying Columns with Missing Values along with Percentage of Record Count as of Entire
df.isna().sum()[df.isna().sum()>0].mul(100)/len(df)
```

Out[170]:

```
emp_title               5.789208
emp_length              4.621115
title                   0.443148
revol_util              0.069692
mort_acc                9.543469
pub_rec_bankruptcies    0.135091
dtype: float64
```

In [171]:

```
df.duplicated().sum()
```

Out[171]:

```
0
```

Observations

- There are missing values in the data - emp_title, emp_length, title, revol_util, mort_acc and pub_rec_bankrupcties
- There are no duplicate rows in the data

Unique Values Check

In [172]:

```
df.nunique()
```

Out[172]:

```
loan_amnt                 1397
term                         2
int_rate                   566
installment              55706
grade                        7
sub_grade                   35
emp_title               173105
emp_length                  11
home_ownership               6
annual_inc               27197
verification_status          3
issue_d                    115
loan_status                  2
purpose                     14
title                    48817
dti                       4262
earliest_cr_line           684
open_acc                    61
pub_rec                     20
revol_bal                55622
revol_util                1226
total_acc                  118
initial_list_status          2
application_type             3
mort_acc                    33
pub_rec_bankruptcies         9
address                 393700
dtype: int64
```

It can be seen that some of the categorical variables like Employment Title, Loan Title, Address has large number of unique values.

Graphical Analysis

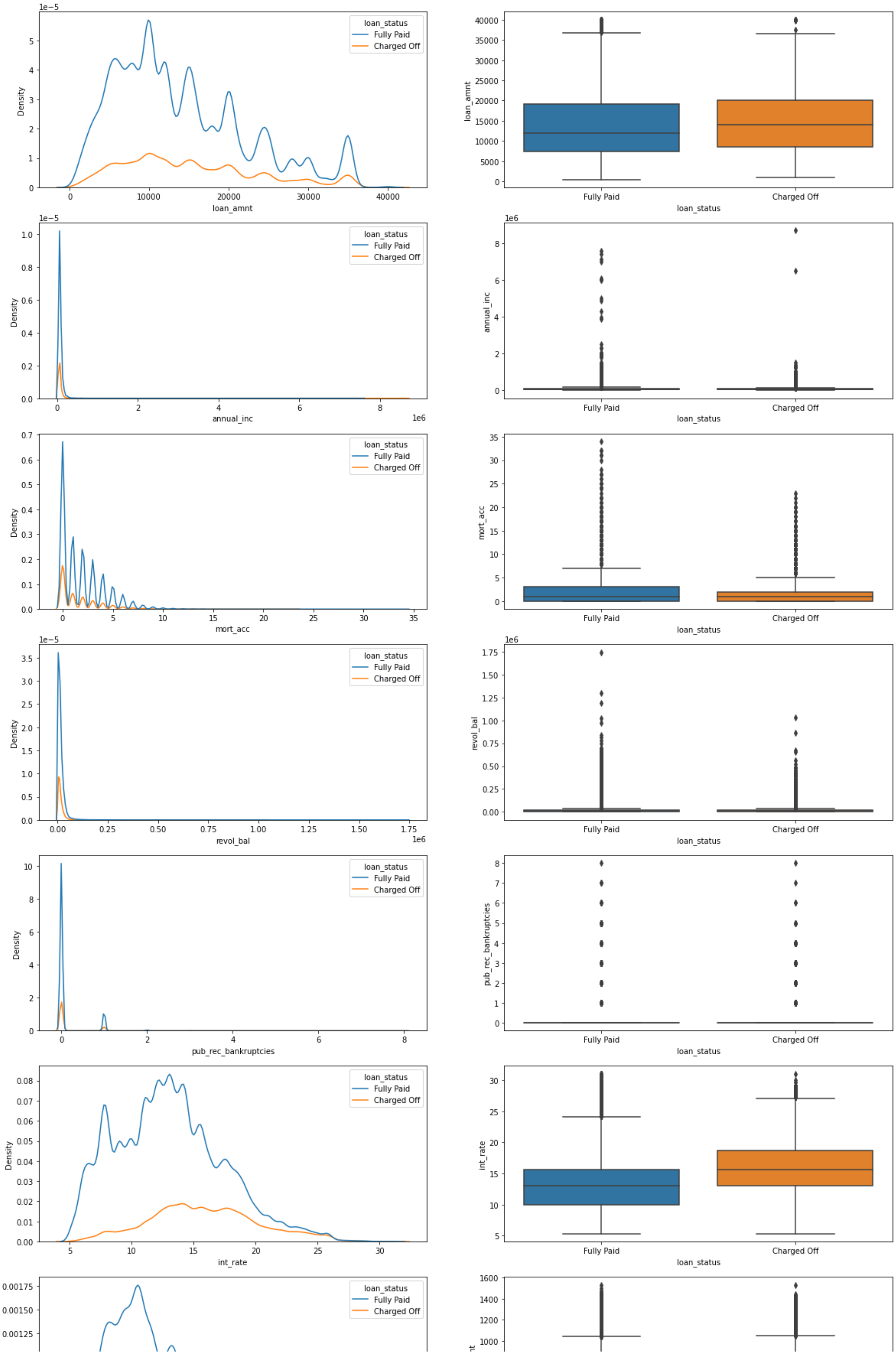
In [173]:

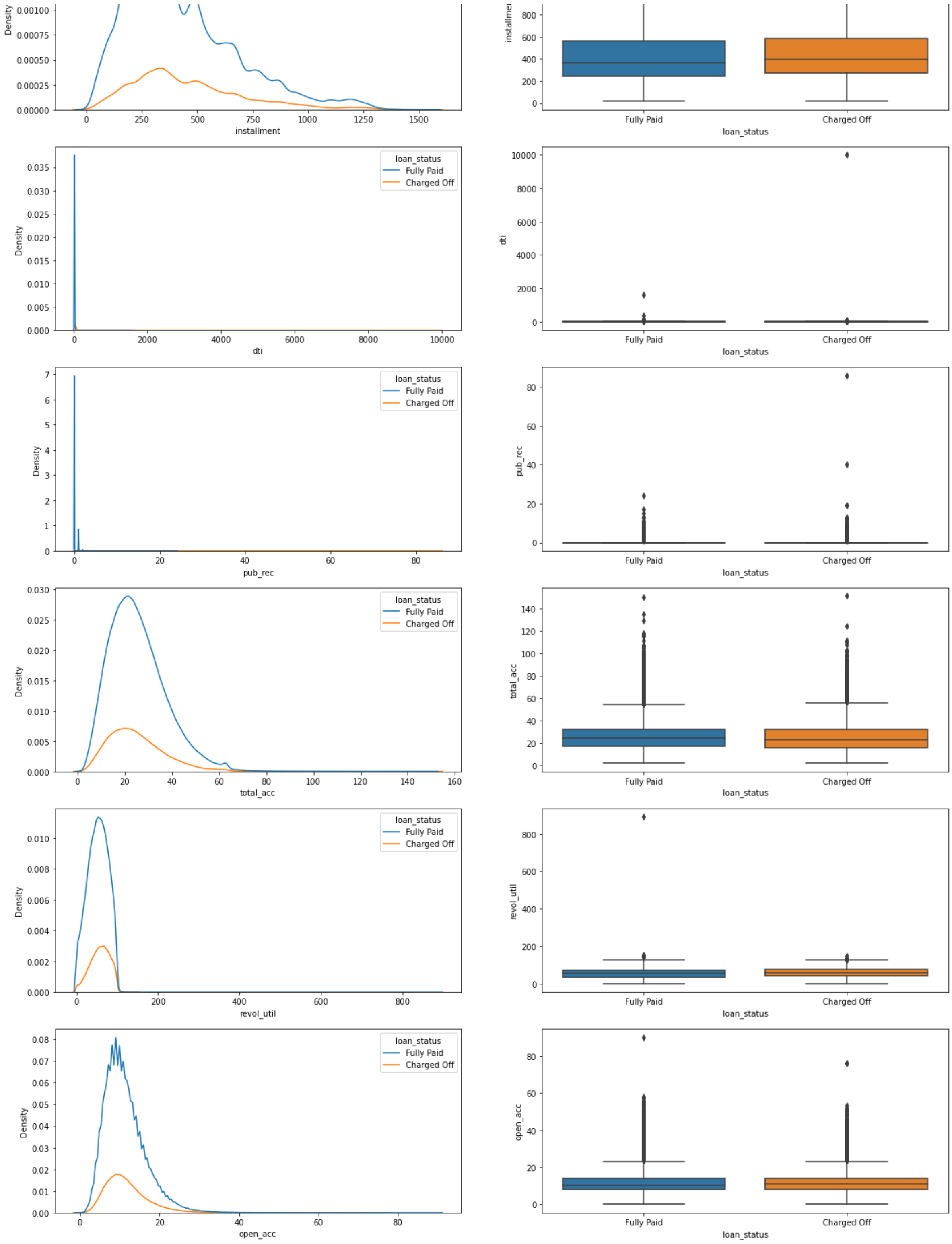
```
# Creating separate sets of numerical and categorical columns
numerical_columns = set(df.select_dtypes(['number']).columns)
categorical_columns = set(df.columns) - numerical_columns
```

Loan Status analysis with Numerical Columns

In [179]:

```python
# Visualizing Distributions & Outliers Through Box Plot
fig, axes = plt.subplots(len(numerical_columns), 2, figsize=(20,len(numerical_columns)*5))
for i, each in enumerate(numerical_columns):
    sns.kdeplot(data=df, x=each, hue = 'loan_status', ax=axes[i][0])
    sns.boxplot(data=df, y=each, x='loan_status', ax=axes[i][1])
plt.show()
```



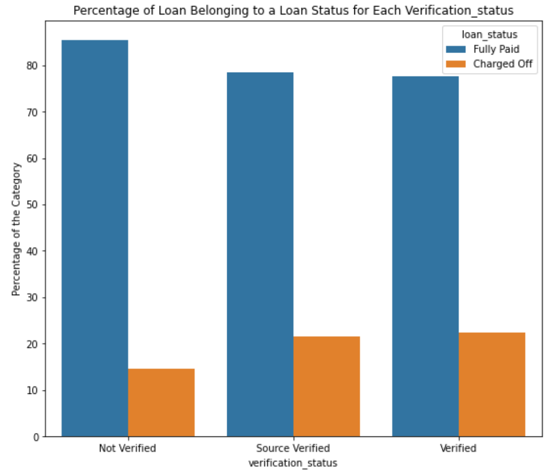


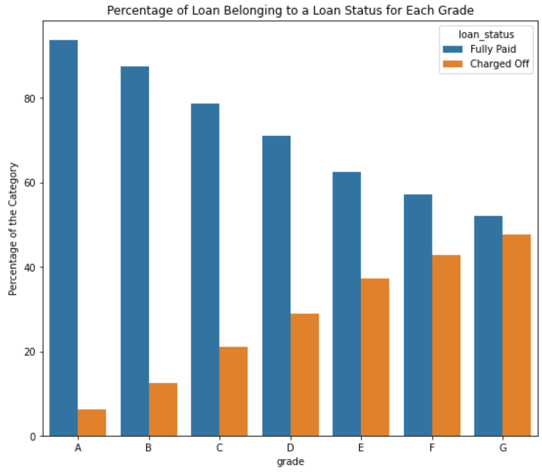
- Columns like Mortgage Account, Public Record, Public Record Bankruptcies can be encoded as 0 (if value is 0) and 1 (if value is greater than 1).
- Variables like Annual Income, Installment, Open Account is skewed and we may use techniques like IQR to remove outliers.

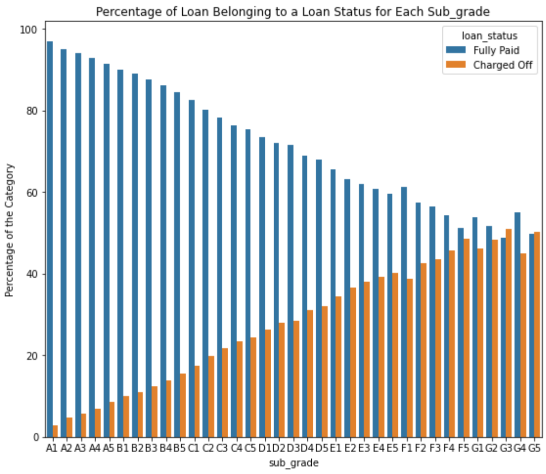
Loan Status analysis with Categorical Columns

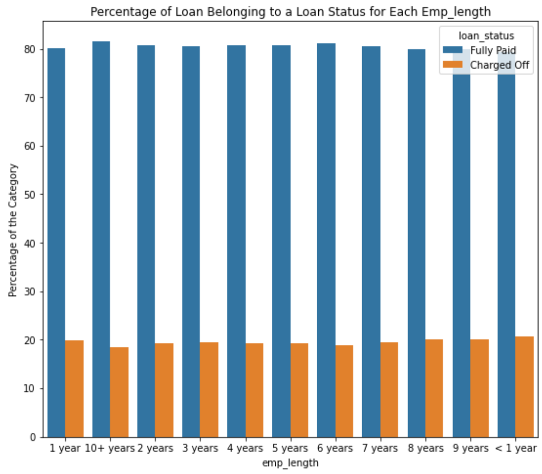
In [175]:

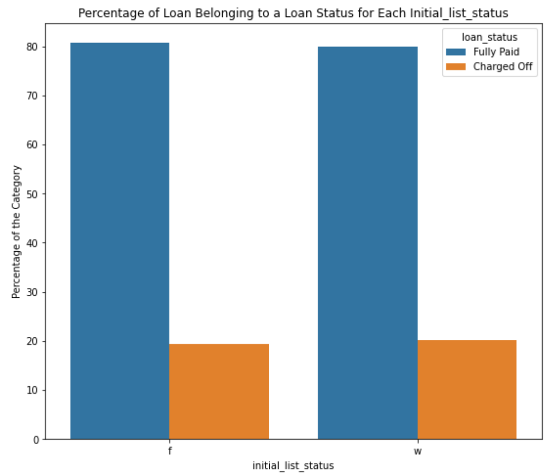
```
cols_to_check_with = ['verification_status', 'grade', 'sub_grade',  'emp_length','initial_l
                      'application_type', 'home_ownership', 'term']
# Plotting Percentage Values for Various Categories Per Loan Status
fig, axes = plt.subplots(len(cols_to_check_with)//2, 2, figsize=(20,len(cols_to_check_with)
percentage_count_dfs = []
for i, each in enumerate(cols_to_check_with):
    percentage_count_dfs.append(pd.DataFrame(df.groupby(by=[each])['loan_status'].value_cou
    percentage_count_dfs[i].columns = ['Percentage of the Category']
    percentage_count_dfs[i].reset_index(inplace=True)
    if i%2 == 1:
        sns.barplot(data=percentage_count_dfs[i],
        x=each,
        y='Percentage of the Category',
        hue='loan_status',
        ax=axes[i//2][1])
        axes[i//2][1].set_title(f'Percentage of Loan Belonging to a Loan Status for Each {e
        plt.grid()
        plt.xticks(rotation=45)
    else:
        sns.barplot(data=percentage_count_dfs[i],
        x=each,
        y='Percentage of the Category',
        hue='loan_status',
        ax=axes[i//2][0])
        axes[i//2][0].set_title(f'Percentage of Loan Belonging to a Loan Status for Each {e
        plt.xticks(rotation=45)
        plt.grid()
plt.show()
```

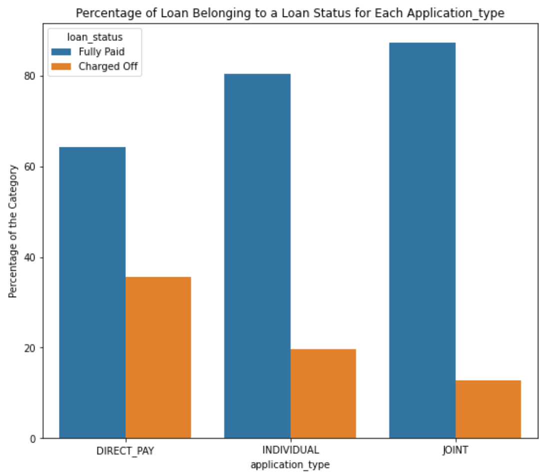

Percentage of Loan Belonging to a Loan Status for Each Verification_status
loan_status
Fully Paid
Charged Off
Percentage of the Category
Not Verified
Source Verified
Verified
verification_status

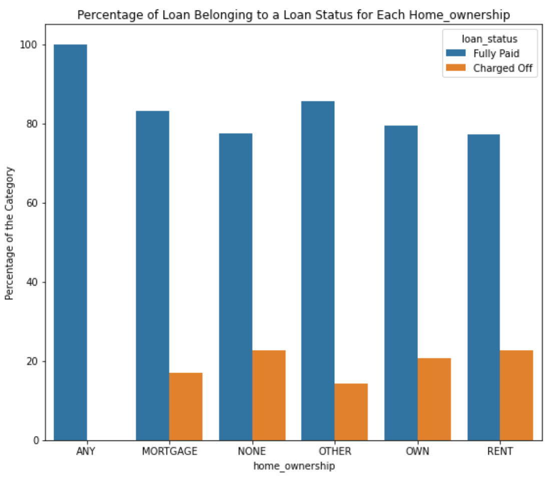

Percentage of Loan Belonging to a Loan Status for Each Grade
loan_status
Fully Paid
Charged Off
Percentage of the Category
A B C D E F G
grade

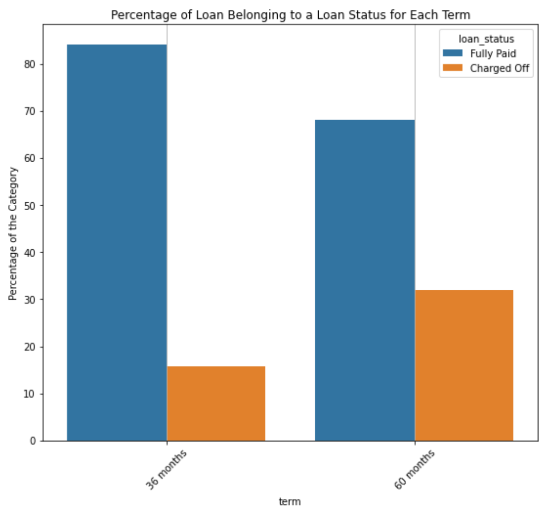

Percentage of Loan Belonging to a Loan Status for Each Sub_grade
loan_status
Fully Paid
Charged Off
Percentage of the Category
A1 A2 A3 A4 A5 B1 B2 B3 B4 B5 C1 C2 C3 C4 C5 D1 D2 D3 D4 D5 E1 E2 E3 E4 E5 F1 F2 F3 F4 F5 G1 G2 G3 G4 G5
sub_grade


Percentage of Loan Belonging to a Loan Status for Each Emp_length
loan_status
Fully Paid
Charged Off
Percentage of the Category
1 year 10+ years 2 years 3 years 4 years 5 years 6 years 7 years 8 years 9 years < 1 year
emp_length


Percentage of Loan Belonging to a Loan Status for Each Initial_list_status
loan_status
Fully Paid
Charged Off
Percentage of the Category
f
w
initial_list_status

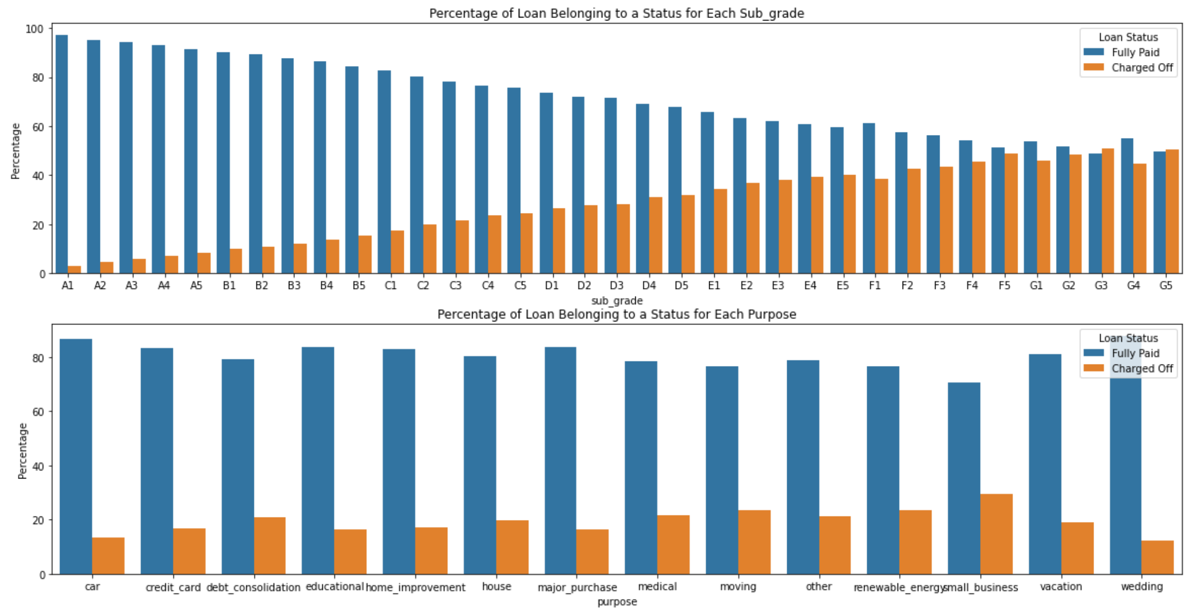

Percentage of Loan Belonging to a Loan Status for Each Application_type
loan_status
Fully Paid
Charged Off
Percentage of the Category
DIRECT_PAY
INDIVIDUAL
JOINT
application_type


Percentage of Loan Belonging to a Loan Status for Each Home_ownership
loan_status
Fully Paid
Charged Off
Percentage of the Category
ANY MORTGAGE NONE OTHER OWN RENT
home_ownership


Percentage of Loan Belonging to a Loan Status for Each Term
loan_status
Fully Paid
Charged Off
Percentage of the Category
36 months
60 months
term

In [177]:

```
cols_to_check_with = ['sub_grade','purpose']
fig, axes = plt.subplots(2, 1, figsize=(20, 10))
percentage_count_dfs = []
for i, each in enumerate(cols_to_check_with):
    percentage_count_dfs.append(pd.DataFrame(df.groupby(by=[each])['loan_status'].value_cou
    percentage_count_dfs[i].columns = ['Percentage']
    percentage_count_dfs[i].reset_index(inplace=True)
    percentage_count_dfs[i].columns = [each, 'Loan Status', 'Percentage']
    sns.barplot(data=percentage_count_dfs[i],

    x=each,
    y='Percentage',
    hue='Loan Status',
    ax=axes[i])

    axes[i].set_title(f'Percentage of Loan Belonging to a Status for Each {each.capitalize(
plt.show()
```



In [90]:

```
# Displaying Absolute Counts of the Loan Extended for Each Home Ownership
df.home_ownership.value_counts()
```

Out[90]:

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

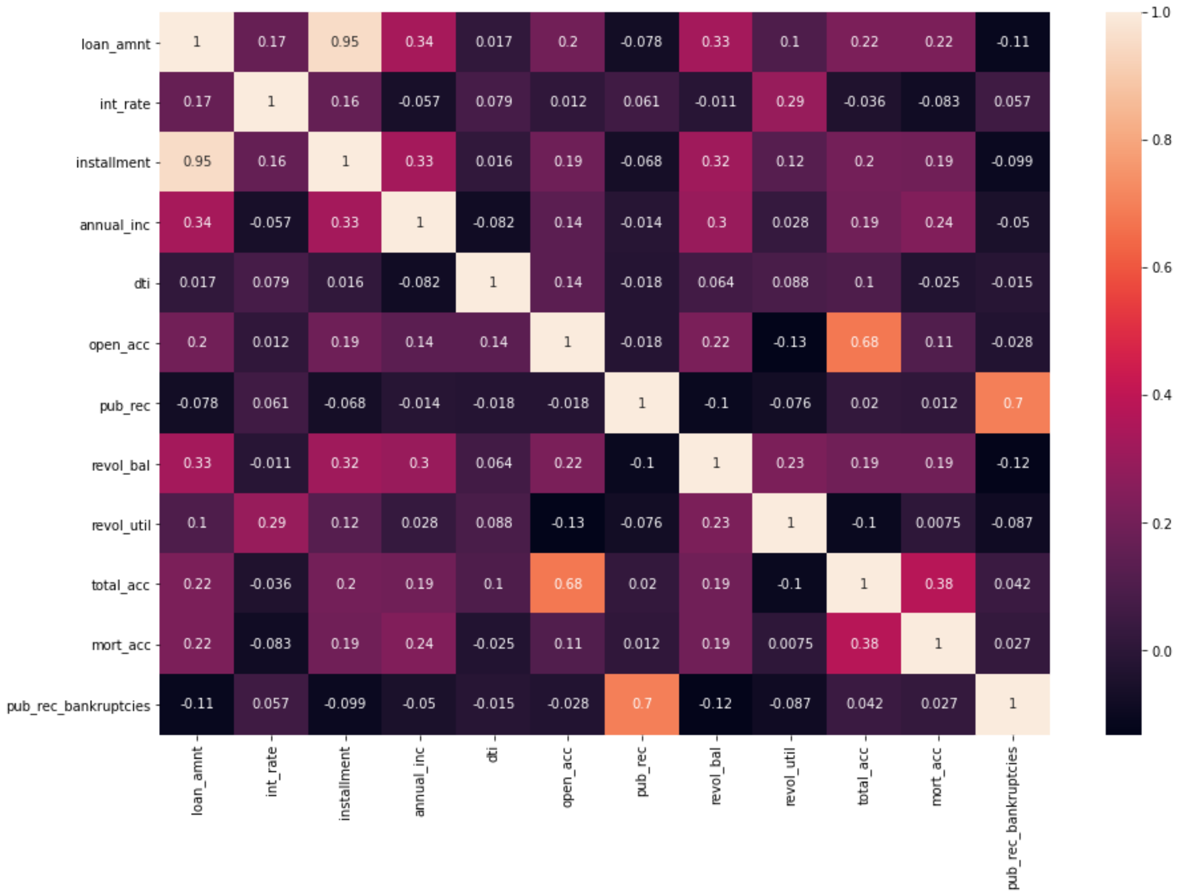
- We see some Loans extended to be charged of across all the categories except Home Ownership = ‘ANY’, in which case there were only 3 credit lines extended, which is not substantial to make any inferences.
- But there is an interesting trend observed for the category Grade, where, as the Grade moves from A to G we see percentage of credit lines extended as charged off increasing from around 5% to 50%.

- Across other categories too we see some high number of charged off Loans such as for Direct Pay Application Type and 60 Months of Tenure as well.

Correlation Check

In [95]:

```
# Plotting Heat Map to Check for Correlation
plt.figure(figsize=(15,10))
sns.heatmap(df.corr(), annot=True)
plt.show()
```



• There is high correlation between Loan Amount and Installment.

Handling of Outliers in data

We can transform the columns Public Record, Public Record Bankruptcies, Mortgage Account as having values 0 or 1 which will also handle outliers.

In [96]:

```
# Columns for which flag transforamtion will be applied
flag_cols = ['pub_rec', 'pub_rec_bankruptcies', 'mort_acc']
```

In [97]:

```
# Defining the function for flag transformation
def transform_flag_columns(x):
    if pd.isna(x):
        return x
    elif int(x)==0:
        return 0
    return 1
# Performing the transformation
for each in flag_cols:
    df[each] = df[each].apply(transform_flag_columns)
    df[each] = df[each].astype(pd.StringDtype())
```

Date Based Columns

In [98]:

```
# Creating a curr_date column with current date value and then store difference in years fo
df['curr_date'] = pd.to_datetime(pd.to_datetime('today').date())
df['issued_years_ago'] = df.curr_date.apply(lambda x:x.year) - pd.to_datetime(df.issue_d).a
df['credit_age_in_years'] = df.curr_date.apply(lambda x:x.year) - pd.to_datetime(df.earlies
# Dropping the unnecessary columns
df.drop(columns=['issue_d', 'earliest_cr_line', 'curr_date'], inplace=True)
```

Cleaning String Based Columns & Employment Length

In [99]:

```
# Defining the function to trim and capitalize each word of the sentence
def transform_title_like(x):
    if pd.isna(x):
        return x
    return x.strip().title()

# Defining the function to transform the employment length

def transform_employment_length(x):
    if pd.isna(x):
        return x
    elif x.strip()=='< 1 year':
        return 0
    elif x.strip()=='10+ years':
        return 10
    return x.strip()[0]

# Initializing the columns for which trimming has to be done

cols_to_clean = ['term', 'grade', 'sub_grade', 'emp_title', 'home_ownership','verification_

# Performing the trimming

for each in cols_to_clean:
    df[each] = df[each].apply(transform_title_like)

# Transforming Employment Length
df['emp_length'] = df['emp_length'].apply(transform_employment_length)
df['emp_length'] = df['emp_length'].astype('Float32')
df['emp_length'] = df['emp_length'].astype(pd.Int8Dtype())
```

Transforming Address

In [ ]:

```
# Defining the function to extract zipcode

def get_zip_code(x):
    if pd.isna(x):
        return x
    return x[-5:]

# Extracting Zipcode

df['zipcode'] = df['address'].apply(get_zip_code)

# Dropping the address column

df.drop(columns='address', inplace=True)
```

Outlier Removal

In [104]:

```
# Creating separate sets of numerical and categorical columns
numerical_columns = set(df.select_dtypes(['number']).columns)
categorical_columns = set(df.columns) - numerical_columns
```

In [105]:

```
# Creating a dictionary to store column wise permitted lower limit and upper limit values
col_limit_val_dict = {}
for each in numerical_columns:
    q1 = df[each].quantile(0.25)
    q3 = df[each].quantile(0.75)
    iqr = q3-q1
    ll = q1 - 1.5*iqr
    ul = q3 + 1.5*iqr
    col_limit_val_dict[each] = (ll,ul)
```

In [106]:

```
# Removing the rows that don't fit the criteria
for key,value in col_limit_val_dict.items():
    df = df[(df[key]>=value[0]) & (df[key]<=value[1])]
```

In [107]:

```
# Displaying the update row count post outlier removal
df.shape
```

Out[107]:

```
(312962, 28)
```

There is about 21% reduction in the row count post outlier removal.

Non-Graphical Analysis

In [108]:

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

Out[108]:

	count	mean	std	min	25%	50%	75
loan_amnt	312962.0	13051.963497	7344.075681	1000.00	7500.00	12000.00	18000.(
int_rate	312962.0	13.547710	4.294021	5.32	10.49	13.33	16.2
installment	312962.0	396.993180	208.978979	19.87	243.36	357.58	521.4
emp_length	312962.0	5.883801	3.628737	0.00	3.00	6.00	10.(
annual_inc	312962.0	65494.998340	28237.885154	4080.00	45000.00	60000.00	81000.(
dti	312962.0	17.235020	7.985544	0.00	11.29	16.79	22.7
open_acc	312962.0	10.662100	4.264979	1.00	8.00	10.00	13.(
revol_bal	312962.0	12285.685620	8652.208885	0.00	5702.00	10253.00	17008.(
revol_util	312962.0	53.718745	24.138235	0.00	36.00	54.60	72.4
total_acc	312962.0	23.779845	10.273687	2.00	16.00	23.00	30.(
issued_years_ago	312962.0	8.366699	1.378224	6.00	7.00	8.00	9.(
credit_age_in_years	312962.0	22.991354	6.039240	9.00	19.00	22.00	27.(

Missing Value check

In [109]:

```
df.isna().sum()[df.isna().sum()>0].mul(100)/len(df)
```

Out[109]:

```
emp_title    1.096938
title        0.392060
mort_acc     8.917057
dtype: float64
```

Earlier we had missing values in six of the columns, removing outliers brought it down to 3 such columns.

Graphical Analysis again

In [110]:

```
numerical_columns
```

Out[110]:

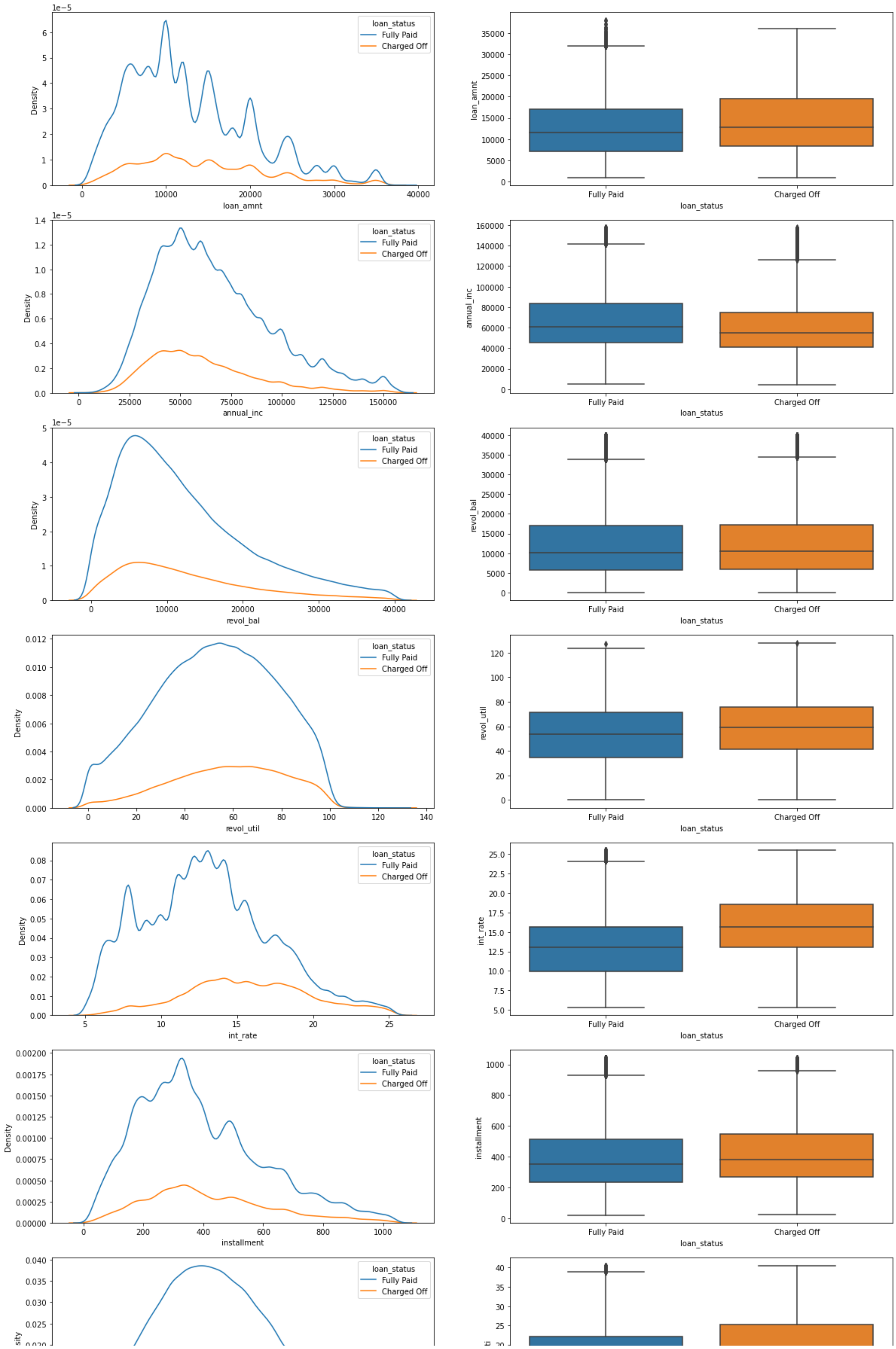
```
{'annual_inc',
 'credit_age_in_years',
 'dti',
 'emp_length',
 'installment',
 'int_rate',
 'issued_years_ago',
 'loan_amnt',
 'open_acc',
 'revol_bal',
 'revol_util',
 'total_acc'}
```

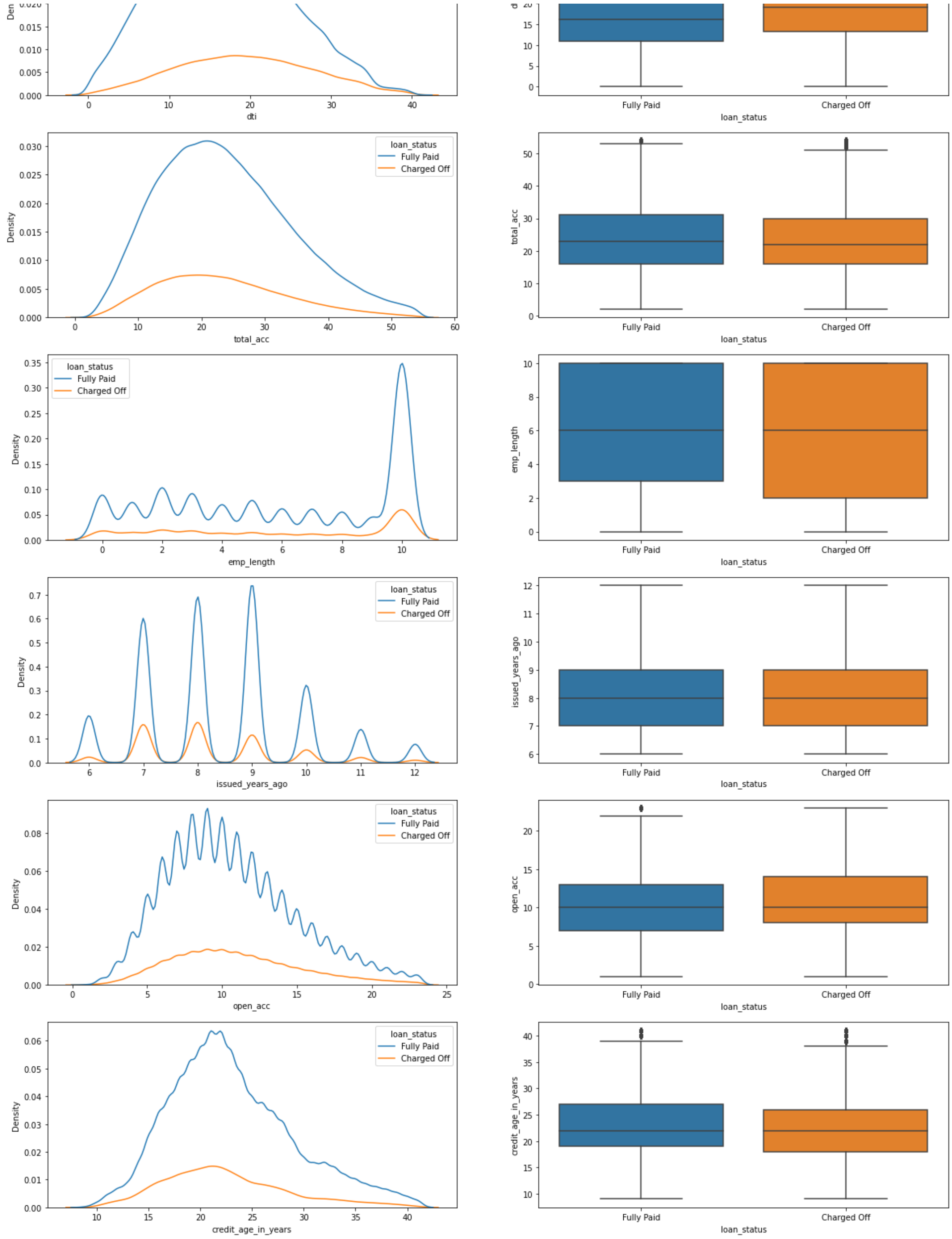
In [119]:

```
df.emp_length=df.emp_length.astype(int)
```

In [121]:

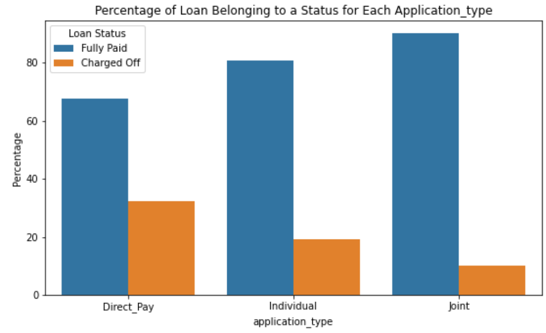
```
# Visualizing Distributions & Outliers Through Box Plot
fig, axes = plt.subplots(len(numerical_columns), 2, figsize=(20,len(numerical_columns)*5))
for i, each in enumerate(numerical_columns):
    sns.kdeplot(data=df, x=each, hue = 'loan_status', ax=axes[i][0])
    sns.boxplot(data=df, y=each, x='loan_status', ax=axes[i][1])
plt.show()
```

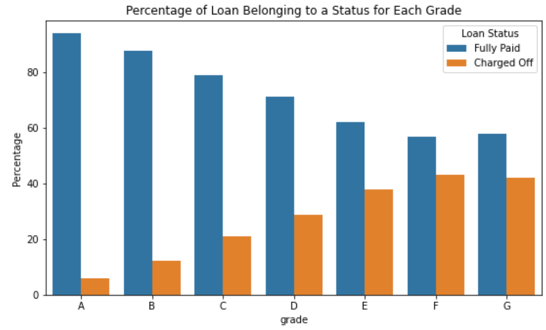


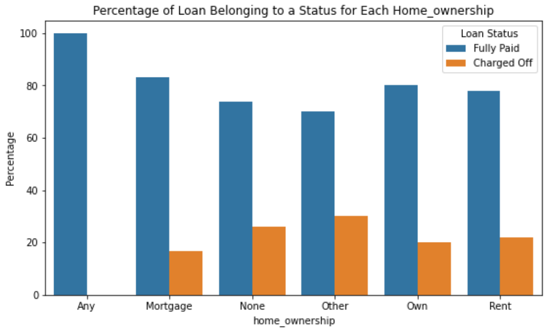

dti
loan_status
total_acc
Density
loan_status
Fully Paid
Charged Off
emp_length
issued_years_ago
open_acc
credit_age_in_years
Fully Paid
Charged Off

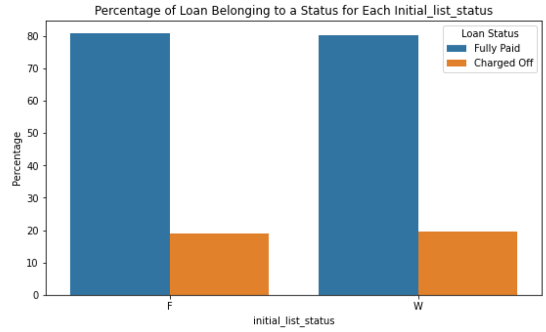
In [130]:

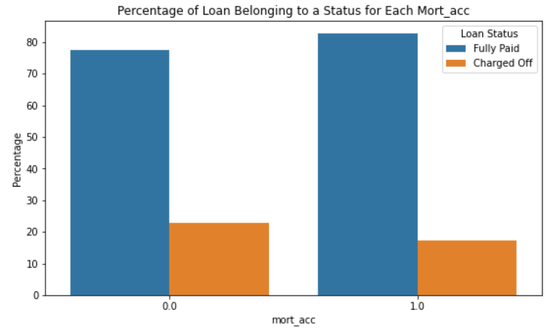
```
cols_to_check_with = ['application_type', 'grade', 'home_ownership','initial_list_status',
                      'pub_rec', 'pub_rec_bankruptcies','term', 'verification_status', 'zip
fig, axes = plt.subplots(len(cols_to_check_with)//2, 2, figsize=(20,len(cols_to_check_with)
percentage_count_dfs = []
for i, each in enumerate(cols_to_check_with):
    percentage_count_dfs.append(pd.DataFrame(df.groupby(by=[each])['loan_status'].value_cou
    percentage_count_dfs[i].columns = ['Percentage']
    percentage_count_dfs[i].reset_index(inplace=True)
    percentage_count_dfs[i].columns = [each, 'Loan Status', 'Percentage']

    if i%2 == 1:
        sns.barplot(data=percentage_count_dfs[i],
        x=each,
        y='Percentage',
        hue='Loan Status',
        ax=axes[i//2][1])
        axes[i//2][1].set_title(f'Percentage of Loan Belonging to a Status for Each {each.c
    else:
        sns.barplot(data=percentage_count_dfs[i],
        x=each,
        y='Percentage',
        hue='Loan Status',
        ax=axes[i//2][0])
        axes[i//2][0].set_title(f'Percentage of Loan Belonging to a Status for Each {each.c
plt.show()
```

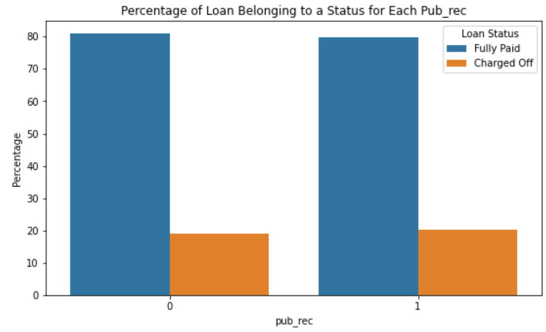

Percentage of Loan Belonging to a Status for Each Application_type
Loan Status
Fully Paid
Charged Off
Percentage
Direct_Pay
Individual
Joint
application_type

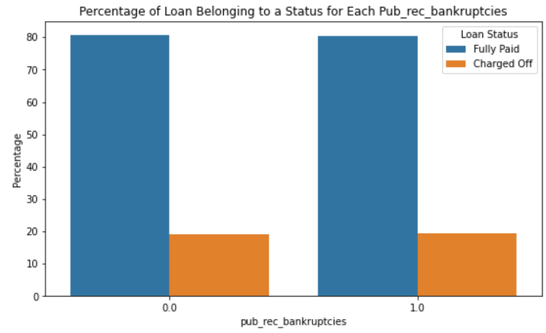

Percentage of Loan Belonging to a Status for Each Grade
Loan Status
Fully Paid
Charged Off
Percentage
A
B
C
D
E
F
G
grade

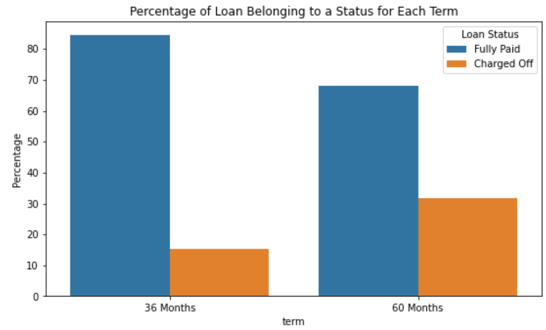

Percentage of Loan Belonging to a Status for Each Home_ownership
Loan Status
Fully Paid
Charged Off
Percentage
Any
Mortgage
None
Other
Own
Rent
home_ownership

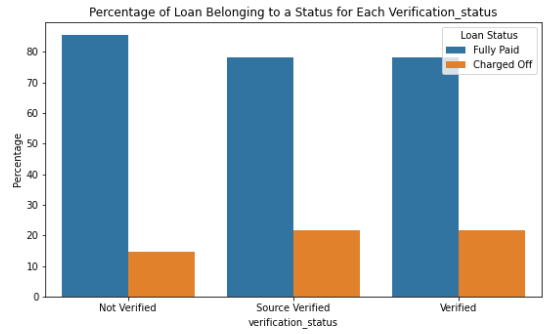

Percentage of Loan Belonging to a Status for Each Initial_list_status
Loan Status
Fully Paid
Charged Off
Percentage
F
W
initial_list_status

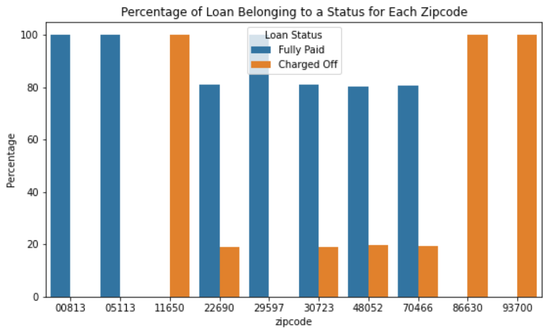

Percentage of Loan Belonging to a Status for Each Mort_acc
Loan Status
Fully Paid
Charged Off
Percentage
0.0
1.0
mort_acc


Percentage of Loan Belonging to a Status for Each Pub_rec
Loan Status
Fully Paid
Charged Off
Percentage
0
1
pub_rec


Percentage of Loan Belonging to a Status for Each Pub_rec_bankruptcies
Loan Status
Fully Paid
Charged Off
Percentage
0.0
1.0
pub_rec_bankruptcies

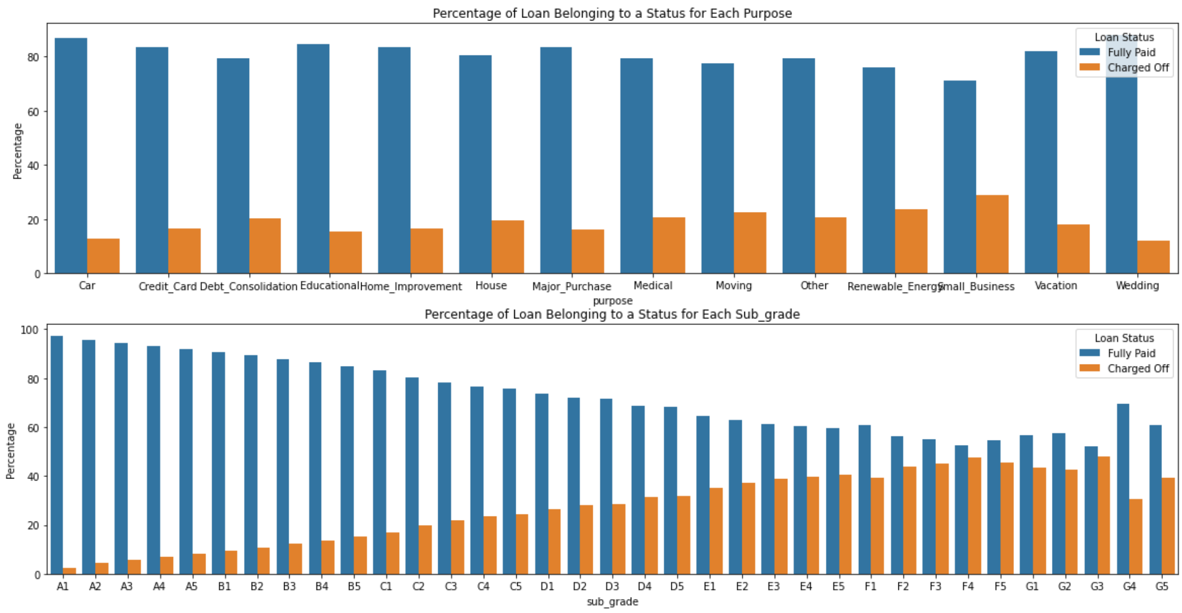

Percentage of Loan Belonging to a Status for Each Term
Loan Status
Fully Paid
Charged Off
Percentage
36 Months
60 Months
term


Percentage of Loan Belonging to a Status for Each Verification_status
Loan Status
Fully Paid
Charged Off
Percentage
Not Verified
Source Verified
Verified
verification_status


Percentage of Loan Belonging to a Status for Each Zipcode
Loan Status
Fully Paid
Charged Off
Percentage
00813
05113
11650
22690
29597
30723
48052
70466
86630
93700
zipcode

In [132]:

```
cols_to_check_with = ['purpose', 'sub_grade']
fig, axes = plt.subplots(2, 1, figsize=(20, 10))
percentage_count_dfs = []
for i, each in enumerate(cols_to_check_with):
    percentage_count_dfs.append(pd.DataFrame(df.groupby(by=[each])['loan_status'].value_cou
    percentage_count_dfs[i].columns = ['Percentage']
    percentage_count_dfs[i].reset_index(inplace=True)
    percentage_count_dfs[i].columns = [each, 'Loan Status', 'Percentage']
    sns.barplot(data=percentage_count_dfs[i],

    x=each,
    y='Percentage',
    hue='Loan Status',
    ax=axes[i])

    axes[i].set_title(f'Percentage of Loan Belonging to a Status for Each {each.capitalize(
plt.show()
```



Based on the the zipcode plot, zipcodes like 11650, 86630, 93700 every loan has been charged off. Also, for certain zipcodes like 00813, 05113, 29597 the entire loan percentage is for fully paid

Data Pre-Processing for Model Building

In [133]:

```
# Printing the percentage of row count per class label
print(df.loan_status.value_counts(normalize=True).mul(100))
# Printing the absolute row count per class label
print(df.loan_status.value_counts())
```

```
Fully Paid     80.743669
Charged Off    19.256331
Name: loan_status, dtype: float64
Fully Paid     252697
Charged Off     60265
Name: loan_status, dtype: int64
```

Train, Test & Validate Split

In [134]:

```
# Importing Necessary Module for splitting teh records
from sklearn.model_selection import train_test_split
```

We will perform a 60-20-20 split.

In [136]:

```
# Creating the feature variable X and target variable y
X = df.drop(columns='loan_status')
y = df['loan_status'].apply(lambda x: 1 if x=='Fully Paid' else 0)
```

In [137]:

```
# Splitting the train, test and validation data into 60:20:20 ratio
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,random_state=3)
X_train, X_val, y_train, y_val = train_test_split(X_train, y_train, test_size=0.25, random_
```

In [138]:

```
# Validating the split by displaying the shape of splitted dataset
print(f'X_train shape: {X_train.shape}, y_train shape: {y_train.shape}')
print(f'X_val shape: {X_val.shape}, y_val shape: {y_val.shape}')
print(f'X_test shape: {X_test.shape}, y_test shape: {y_test.shape}')
```

```
X_train shape: (187776, 27), y_train shape: (187776,)
X_val shape: (62593, 27), y_val shape: (62593,)
X_test shape: (62593, 27), y_test shape: (62593,)
```

In [139]:

```
# Displaying the class label split for train, test and validation data
print(y_train.value_counts(normalize=True).mul(100))
print(y_val.value_counts(normalize=True).mul(100))
print(y_test.value_counts(normalize=True).mul(100))
```

```
1    80.776031
0    19.223969
Name: loan_status, dtype: float64
1    80.520186
0    19.479814
Name: loan_status, dtype: float64
1    80.870065
0    19.129935
Name: loan_status, dtype: float64
```

Missing Value Imputation

We have already seen that we have missing values in three columns and all of them are categorical. We will keep it simple perform a modal imputation here.

In [140]:

```
# Performing modal imputation for the the required columns on train, test and validation da
for each in ['emp_title', 'title', 'mort_acc']:
    train_modal_value = X_train[each].mode()[0]
    val_modal_value = X_val[each].mode()[0]
    test_modal_value = X_test[each].mode()[0]
    X_train[each] = X_train[each].fillna(train_modal_value)
    X_val[each] = X_val[each].fillna(val_modal_value)
    X_test[each] = X_test[each].fillna(test_modal_value)
```

In [141]:

```
# Validating the missing value count for train, test and validate dataset
print(X_train.isna().sum()[X_train.isna().sum()>0].mul(100)/len(df))
print(X_val.isna().sum()[X_val.isna().sum()>0].mul(100)/len(df))
print(X_test.isna().sum()[X_test.isna().sum()>0].mul(100)/len(df))
```

```
Series([], dtype: float64)
Series([], dtype: float64)
Series([], dtype: float64)
```

Encoding

In [145]:

```
# Importing Module for Encoding

import category_encoders as ce
```

In [146]:

```
# Changing the type of flag transformed columns from object to integer
for each in flag_cols:
    X_train[each] = X_train[each].astype(pd.Float32Dtype())
    X_train[each] = X_train[each].astype(pd.Int8Dtype())
    X_val[each] = X_val[each].astype(pd.Float32Dtype())
    X_val[each] = X_val[each].astype(pd.Int8Dtype())
    X_test[each] = X_test[each].astype(pd.Float32Dtype())
    X_test[each] = X_test[each].astype(pd.Int8Dtype())
```

In [147]:

```
# Creating separate lists of column that has to be one hot encoded and target encoded
ohe_cols = ['application_type', 'grade', 'home_ownership','initial_list_status', 'purpose',
            'verification_status', 'zipcode']
target_encoding_cols = ['emp_title', 'region', 'title']
```

In [148]:

```
# Performing One Hot Encoding
ohe = ce.OneHotEncoder(cols=ohe_cols)
X_train = ohe.fit_transform(X_train)
X_val = ohe.transform(X_val)
X_test = ohe.transform(X_test)
```

In [149]:

```
# Performing Target Encoding
te = ce.TargetEncoder(cols=target_encoding_cols)
X_train = te.fit_transform(X_train, y_train)
X_val = te.transform(X_val)
X_test = te.transform(X_test)
```

Model Building

In [150]:

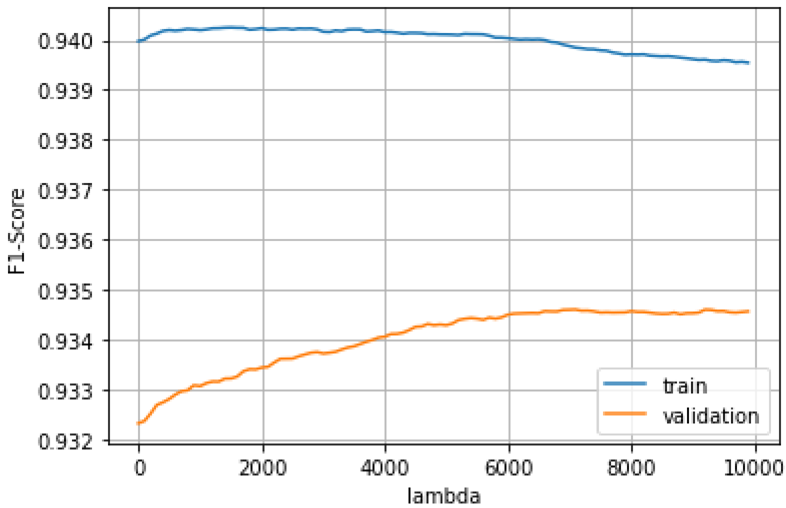
```
# Importing modules for training the model and metrics
import numpy as np
from numpy import argmax
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.pipeline import make_pipeline
from sklearn import metrics
from sklearn.metrics import classification_report, confusion_matrix, f1_score,roc_curve, ro
```

In [152]:

```
# Without Class Weights
# Hyperparameter Tuning for Regularization Value
# Initializing lists to store training and validation F1-Scores for various lambda (regular
nw_train_scores = []
nw_val_scores = []
scaler = StandardScaler()
# Initializing values for iterating over various lambda values
l=0.01
h= 10000.0
d=100.0
for lamda in np.arange(l,h,d):
    nw_std_lr = make_pipeline(scaler, LogisticRegression(C=1/lamda))
    nw_std_lr.fit(X_train, y_train)
    nw_train_y_pred = nw_std_lr.predict(X_train)
    nw_val_y_pred = nw_std_lr.predict(X_val)
    nw_train_scores.append(f1_score(y_train, nw_train_y_pred))
    nw_val_scores.append(f1_score(y_val, nw_val_y_pred))
```

In [153]:

```
# Plotting training and validation F1-Scores for various lambda
plt.figure()
plt.plot(list(np.arange(l,h,d)), nw_train_scores, label="train")
plt.plot(list(np.arange(l,h,d)), nw_val_scores, label="validation")
plt.legend()
plt.xlabel("lambda")
plt.ylabel("F1-Score")
plt.grid()
plt.show()
```



In [154]:

```
# Pick the best lambda value
nw_best_idx = np.argmax(nw_val_scores)
print(f'Best Validation F1-Score: {nw_val_scores[nw_best_idx]}')
nw_lambda_best = l+(d*nw_best_idx)
print(f'Best Lambda: {nw_lambda_best}')
# Test F1-Score by using the best lambda
nwf_std_lr = make_pipeline(scaler, LogisticRegression(C=1/nw_lambda_best))
nwf_std_lr.fit(X_train, y_train)
nw_test_y_pred = nwf_std_lr.predict(X_test)
print(f'Test F1-Score: {f1_score(y_test, nw_test_y_pred)}')
# Print the classification report
nw_classification_report = metrics.classification_report(y_test, nw_test_y_pred)
print(nw_classification_report)
# Print the confusion matrix
print(metrics.confusion_matrix(y_test, nw_test_y_pred))
```

```
Best Validation F1-Score: 0.9346162460803344
Best Lambda: 7100.01
Test F1-Score: 0.9350900699717136
              precision    recall  f1-score   support

           0       0.94      0.45      0.61     11974
           1       0.88      0.99      0.94     50619

    accuracy                           0.89     62593
   macro avg       0.91      0.72      0.77     62593
weighted avg       0.89      0.89      0.87     62593

[[ 5369  6605]
 [  371 50248]]
```

If we look at this model which has overall F1-Score of 0.93 (which is great) is not really great as explained below:

– For class label 1 (Fully Paid) most of the metrics look fine but precision (of total +ve identification how many were actually positive) has suffered, because we have identified more than half of the class label 0 as 1 which contributed more to the total number of predicted positives.

– For class label 0, the recall (0.45) is horrible, because for more than half of the cases we wrongly predicted that the loan would be paid off where as those were charged off.

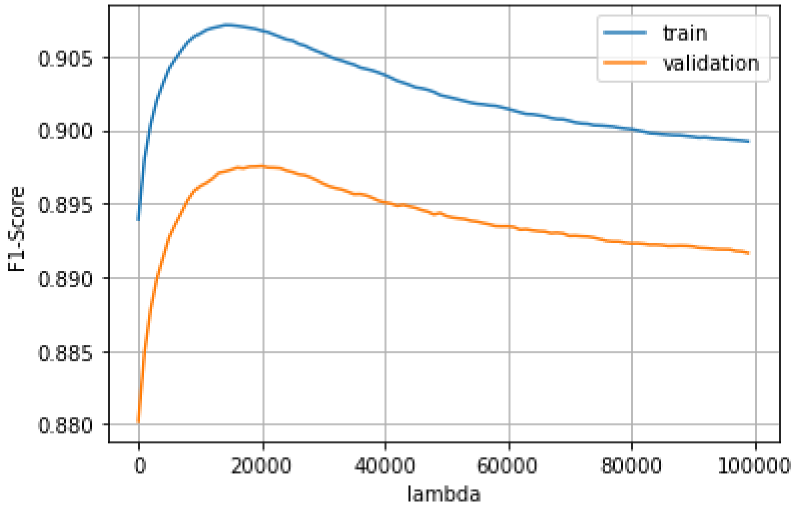
Although our F1-Score is pretty good, but the metrics for class label 0 (Charged Off) is not great. Let's introduce some weight and re-train our model. Since the data consists the class labels in the ratio 4:1 (Fully Paid : Charged Off), let's try this ratio.

In [155]:

```
# With Class Weights
w_train_scores = []
w_val_scores = []
scaler = StandardScaler()
l=0.01
h= 100000.0
d=1000.0
for lamda in np.arange(l,h,d):
    w_std_lr = make_pipeline(scaler, LogisticRegression(C=1/lamda,class_weight={0:0.8, 1:0.
    w_std_lr.fit(X_train, y_train)
    w_train_y_pred = w_std_lr.predict(X_train)
    w_val_y_pred = w_std_lr.predict(X_val)
    w_train_scores.append(f1_score(y_train, w_train_y_pred))
    w_val_scores.append(f1_score(y_val, w_val_y_pred))
```

In [156]:

```
# Plotting F1-Scores
plt.figure()
plt.plot(list(np.arange(l,h,d)), w_train_scores, label="train")
plt.plot(list(np.arange(l,h,d)), w_val_scores, label="validation")
plt.legend()
plt.xlabel("lambda")
plt.ylabel("F1-Score")
plt.grid()
plt.show()
```



In [157]:

```
# Picking Best Lambda and Calculating Test Metrics
w_best_idx = np.argmax(w_val_scores)
print(f'Best Validation F1-Score: {w_val_scores[w_best_idx]}')
w_lambda_best = l+(d*w_best_idx)
print(f'Best Lambda: {w_lambda_best}')
# Test F1-Score
wf_std_lr = make_pipeline(scaler, LogisticRegression(C=1/w_lambda_best,class_weight={0:0.8,
wf_std_lr.fit(X_train, y_train)
w_test_y_pred = wf_std_lr.predict(X_test)
print(f'Test F1-Score: {f1_score(y_test, w_test_y_pred)}')
w_classification_report = metrics.classification_report(y_test, w_test_y_pred)
print(w_classification_report)
print(metrics.confusion_matrix(y_test, w_test_y_pred))
```

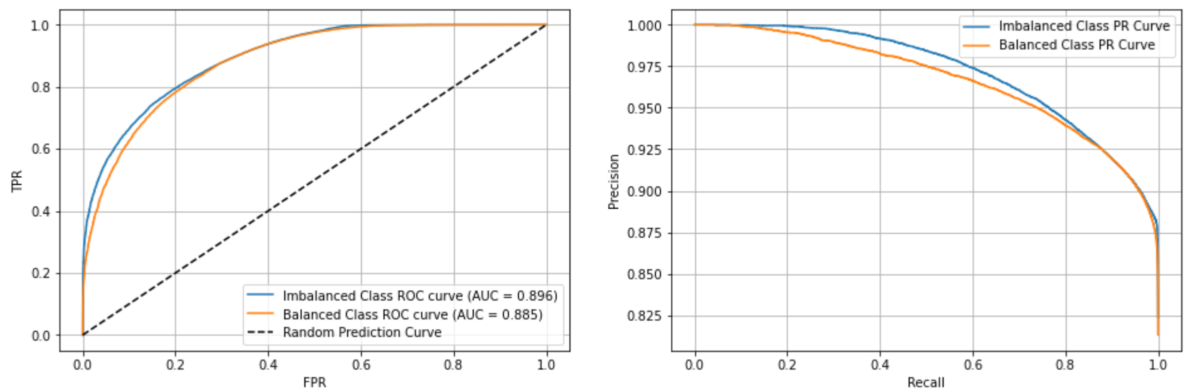
```
Best Validation F1-Score: 0.8975566887051778
Best Lambda: 20000.01
Test F1-Score: 0.8967241045103501
              precision    recall  f1-score   support

           0       0.56      0.71      0.63     11974
           1       0.93      0.87      0.90     50619

    accuracy                           0.84     62593
   macro avg       0.74      0.79      0.76     62593
weighted avg       0.86      0.84      0.85     62593

[[ 8522  3452]
 [ 6671 43948]]
```

Introducing weights to address class imbalance still yielded 0.89 as overall F1-Score, and is better as explained below: - The recall for class label improved from 0.45 to 0.71 (i.e. out of total charged off, how many were identified as charged off). - And the metrics for class label 1 hasn't degraded too much.

ROC & PR Curve

In [158]:

```
nw_test_y_pred_prob = nwf_std_lr.predict_proba(X_test)
nw_test_y_pred_prob = nw_test_y_pred_prob[:,1]
w_test_y_pred_prob = wf_std_lr.predict_proba(X_test)
w_test_y_pred_prob = w_test_y_pred_prob[:,1]
nw_fpr, nw_tpr, _ = metrics.roc_curve(y_test, nw_test_y_pred_prob)
nw_roc_auc_val = metrics.roc_auc_score(y_test, nw_test_y_pred_prob)
w_fpr, w_tpr, _ = metrics.roc_curve(y_test, w_test_y_pred_prob)
w_roc_auc_val = metrics.roc_auc_score(y_test, w_test_y_pred_prob)
nw_precision, nw_recall, _ = precision_recall_curve(y_test, nw_test_y_pred_prob)
w_precision, w_recall, _ = precision_recall_curve(y_test, w_test_y_pred_prob)

fig, axes = plt.subplots(1,2, figsize=(16,5))

axes[0].plot(nw_fpr, nw_tpr, label='Imbalanced Class ROC curve (AUC = %0.3f)' %nw_roc_auc_v
axes[0].plot(w_fpr, w_tpr, label='Balanced Class ROC curve (AUC = %0.3f)' %w_roc_auc_val)
axes[0].plot([0, 1], [0, 1], 'k--', label='Random Prediction Curve')
axes[0].legend(loc='lower right')
axes[0].grid()
axes[0].set_xlabel('FPR')
axes[0].set_ylabel('TPR')

axes[1].plot(nw_recall, nw_precision, label='Imbalanced Class PR Curve')
axes[1].plot(w_recall, w_precision, label='Balanced Class PR Curve')
axes[1].legend()
axes[1].grid()
axes[1].set_xlabel('Recall')
axes[1].set_ylabel('Precision')
plt.show()
```



Considering Charged Off as an equally important class, we should go ahead and pick second model (with weights) even though it has slightly less (not significant) area under ROC curve.

Actionable Insights & Recommendations

• There is an interesting trend for people belonging to different Grades and Sub- grades. As the Grades move from A to G and Subgrades move from A1 to G5,the precentage of charge off loan status rises. So this is a must have data whileaccepting loan application and it should be ensured to have it.

• Another interesting observation is based on Zipcodes, certain zipcodes (mentioned earlier in the analysis) hvae entire loan as charged off and some of them has entire loan as fully paid. So various new loan schemes can be launched for the latter zipcodes since those have very trust worthy applicants and hence better business and ROI or interest generation on the credit line extended.

• Alternatively, two sets of models can used for prediction:

– Stricter (more weightage to the class label ‘Charged Off’) if the bank wants to play safe, let’s say in the regions where charged off percentage is high.

– Bit Leniant (where class imbalance is not addressed) if the bank wants to be aggressive, let’s say in the regions where fully paid percentage is high.

In [ ]: