Defining Problem Statement

- 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.
- 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?

Notes:

1. loan_status : Current status of the loan - Binary value -Target Variable

In [287…

```python
import numpy as np
import pandas as pd
import math
import itertools
import seaborn as sns
import matplotlib.pyplot as plt
import statsmodels.api as sm
```

Data Definition

In [288…

```python
org_data = pd.read_csv("loantap.csv")
org_data.head(8)
```

Out[288]:

	loan_amnt	term	int_rate	installment	grade	sub_grade	emp_title	emp_length	home_ownership	annual_inc	...	open_acc	pub_rec
0	10000.0	36 months	11.44	329.48	B	B4	Marketing	10+ years	RENT	117000.0	...	16.0	0.0
1	8000.0	36 months	11.99	265.68	B	B5	Credit analyst	4 years	MORTGAGE	65000.0	...	17.0	0.0
2	15600.0	36 months	10.49	506.97	B	B3	Statistician	< 1 year	RENT	43057.0	...	13.0	0.0
3	7200.0	36 months	6.49	220.65	A	A2	Client Advocate	6 years	RENT	54000.0	...	6.0	0.0
4	24375.0	60 months	17.27	609.33	C	C5	Destiny Management Inc.	9 years	MORTGAGE	55000.0	...	13.0	0.0
5	20000.0	36 months	13.33	677.07	C	C3	HR Specialist	10+ years	MORTGAGE	86788.0	...	8.0	0.0
6	18000.0	36 months	5.32	542.07	A	A1	Software Development Engineer	2 years	MORTGAGE	125000.0	...	8.0	0.0
7	13000.0	36 months	11.14	426.47	B	B2	Office Depot	10+ years	RENT	46000.0	...	11.0	0.0

8 rows × 27 columns

In [289…

```python
org_data.shape
```

Out[289]:

```
(396030, 27)
```

In [290…

```python
org_data.dtypes
```

Out[290]:

```
loan_amnt               float64
term                     object
int_rate                float64
installment             float64
grade                    object
sub_grade                object
emp_title                object
emp_length               object
home_ownership           object
annual_inc              float64
verification_status      object
issue_d                  object
loan_status              object
purpose                  object
title                    object
dti                     float64
earliest_cr_line         object
open_acc                float64
pub_rec                 float64
revol_bal               float64
revol_util              float64
total_acc               float64
initial_list_status      object
application_type         object
mort_acc                float64
pub_rec_bankruptcies    float64
address                  object
dtype: object
```

In [291…

```python
org_data.describe().T
```

Out[291]:

	count	mean	std	min	25%	50%	75%	max
loan_amnt	396030.0	14113.888089	8357.441341	500.00	8000.00	12000.00	20000.00	40000.00
int_rate	396030.0	13.639400	4.472157	5.32	10.49	13.33	16.49	30.99
installment	396030.0	431.849698	250.727790	16.08	250.33	375.43	567.30	1533.81
annual_inc	396030.0	74203.175798	61637.621158	0.00	45000.00	64000.00	90000.00	8706582.00
dti	396030.0	17.379514	18.019092	0.00	11.28	16.91	22.98	9999.00
open_acc	396030.0	11.311153	5.137649	0.00	8.00	10.00	14.00	90.00
pub_rec	396030.0	0.178191	0.530671	0.00	0.00	0.00	0.00	86.00
revol_bal	396030.0	15844.539853	20591.836109	0.00	6025.00	11181.00	19620.00	1743266.00
revol_util	395754.0	53.791749	24.452193	0.00	35.80	54.80	72.90	892.30
total_acc	396030.0	25.414744	11.886991	2.00	17.00	24.00	32.00	151.00
mort_acc	358235.0	1.813991	2.147930	0.00	0.00	1.00	3.00	34.00
pub_rec_bankruptcies	395495.0	0.121648	0.356174	0.00	0.00	0.00	0.00	8.00

In [292…

```python
org_data.describe(include = "object").T
```

Out[292]:

	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	USCGC Smith\r\nFPO AE 70466	8

In [293…

```python
org_data.isna().sum()
```

Out[293]:

```
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 [294…

```python
org_data[org_data.duplicated() == True].count()
```

Out[294]:

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

In [295…

```python
def value_cnts(df, cols):
    threshold = 14 # will only consider top "n" unique values if unique values in a column exceeds threshold va
    print("--------------XXX-------------")
    print("Value counts of discrete Data")
    print("--------------XXX-------------")
    for column in cols:

        if(df[column].nunique() <= threshold):
            print(f"{column} :\nUnique Values: {df[column].unique()},\nUnique Value Counts: {df[column].nunique
            print("--------------XXX-------------", end = "")
            print()

        else:
            print(f"{column} :\nUnique Values: {df[column].nunique()},\nValue counts of Top-{threshold} unique
            print("--------------XXX-------------", end = "")
            print()
```

In [296…

```python
discrete_cols = org_data.columns[org_data.dtypes == "object"].tolist()
value_cnts(org_data, discrete_cols)
```

```
--------------XXX-------------
Value counts of discrete Data
--------------XXX-------------
term :
Unique Values: [' 36 months' ' 60 months'],
Unique Value Counts: 2,
% of total:
 36 months    76.258112
 60 months    23.741888
Name: term, dtype: float64
--------------XXX-------------
grade :
Unique Values: ['B' 'A' 'C' 'E' 'D' 'F' 'G'],
Unique Value Counts: 7,
% of total:
B    29.295255
C    26.762366
A    16.207611
D    16.040199
E     7.950913
F     2.972502
G     0.771154
Name: grade, dtype: float64
--------------XXX-------------
sub_grade :
Unique Values: 35,
Value counts of Top-14 unique values :,
% of total:
B3    6.730551
B4    6.464409
C1    5.974800
C2    5.701588
B2    5.680125
B5    5.576598
C3    5.358432
C4    5.120824
B1    4.843572
A5    4.677928
C5    4.606722
D1    4.038330
A4    3.986819
D2    3.522713
Name: sub_grade, dtype: float64
--------------XXX-------------
emp_title :
Unique Values: 173105,
Value counts of Top-14 unique values :,
% of total:
NaN                 5.789208
Teacher             1.108249
Manager             1.073151
Registered Nurse    0.468651
RN                  0.466126
Supervisor          0.462086
Sales               0.413605
Project Manager     0.380022
Owner               0.356034
Driver              0.338106
Office Manager      0.307552
manager             0.289120
Director            0.274979
General Manager     0.271192
Name: emp_title, dtype: float64
--------------XXX-------------
emp_length :
Unique Values: ['10+ years' '4 years' '< 1 year' '6 years' '9 years' '2 years' '3 years'
 '8 years' '7 years' '5 years' '1 year' nan],
Unique Value Counts: 11,
% of total:
10+ years    31.826124
2 years       9.046537
< 1 year      8.010757
3 years       7.995606
5 years       6.690150
1 year        6.535363
4 years       6.048027
6 years       5.262480
7 years       5.256925
8 years       4.840037
NaN           4.621115
9 years       3.866879
Name: emp_length, dtype: float64
--------------XXX-------------
home_ownership :
Unique Values: ['RENT' 'MORTGAGE' 'OWN' 'OTHER' 'NONE' 'ANY'],
Unique Value Counts: 6,
% of total:
MORTGAGE    50.084085
RENT        40.347953
OWN          9.531096
OTHER        0.028281
NONE         0.007828
ANY          0.000758
Name: home_ownership, dtype: float64
--------------XXX-------------
verification_status :
Unique Values: ['Not Verified' 'Source Verified' 'Verified'],
Unique Value Counts: 3,
% of total:
Verified           35.240512
Source Verified    33.175517
Not Verified       31.583971
Name: verification_status, dtype: float64
--------------XXX-------------
issue_d :
Unique Values: 115,
Value counts of Top-14 unique values :,
% of total:
Oct-2014    3.748706
Jul-2014    3.183850
Jan-2015    2.955584
Dec-2013    2.681110
Nov-2013    2.650304
Jul-2015    2.593238
Oct-2013    2.536929
Jan-2014    2.450572
Apr-2015    2.391233
Sep-2013    2.317754
Aug-2013    2.300836
Apr-2014    2.277605
Nov-2014    2.236699
May-2014    2.232154
Name: issue_d, dtype: float64
--------------XXX-------------
loan_status :
Unique Values: ['Fully Paid' 'Charged Off'],
Unique Value Counts: 2,
% of total:
Fully Paid     80.387092
Charged Off    19.612908
Name: loan_status, dtype: float64
--------------XXX-------------
purpose :
Unique Values: ['vacation' 'debt_consolidation' 'credit_card' 'home_improvement'
 'small_business' 'major_purchase' 'other' 'medical' 'wedding' 'car'
 'moving' 'house' 'educational' 'renewable_energy'],
Unique Value Counts: 14,
% of total:
debt_consolidation    59.214453
credit_card           20.962806
home_improvement       6.067722
other                  5.349342
major_purchase         2.219529
small_business         1.439537
car                    1.186021
medical                1.059516
moving                 0.720652
vacation               0.619145
house                  0.555766
wedding                0.457541
renewable_energy       0.083075
educational            0.064894
Name: purpose, dtype: float64
--------------XXX-------------
title :
Unique Values: 48817,
Value counts of Top-14 unique values :,
% of total:
Debt consolidation           38.500114
Credit card refinancing      13.000783
Home improvement              3.854253
Other                         3.264904
Debt Consolidation            2.931091
Major purchase                1.204202
Consolidation                 0.972654
debt consolidation            0.895639
Business                      0.744641
Debt Consolidation Loan       0.723178
Medical expenses              0.692372
Car financing                 0.540111
Credit Card Consolidation     0.448198
NaN                           0.443148
Name: title, dtype: float64
--------------XXX-------------
earliest_cr_line :
Unique Values: 684,
Value counts of Top-14 unique values :,
% of total:
Oct-2000    0.761811
Aug-2000    0.741105
Oct-2001    0.731258
Aug-2001    0.728228
Nov-2000    0.690857
Oct-1999    0.688332
Nov-1999    0.681767
Sep-2000    0.679494
Oct-2002    0.666616
Aug-2002    0.656263
Sep-2001    0.647678
Aug-1999    0.643386
Sep-1999    0.638840
Sep-2002    0.638840
Name: earliest_cr_line, dtype: float64
--------------XXX-------------
initial_list_status :
Unique Values: ['w' 'f'],
Unique Value Counts: 2,
% of total:
f    60.113123
w    39.886877
Name: initial_list_status, dtype: float64
--------------XXX-------------
application_type :
Unique Values: ['INDIVIDUAL' 'JOINT' 'DIRECT_PAY'],
Unique Value Counts: 3,
% of total:
INDIVIDUAL    99.820468
JOINT          0.107315
DIRECT_PAY     0.072217
Name: application_type, dtype: float64
--------------XXX-------------
address :
Unique Values: 393700,
Value counts of Top-14 unique values :,
% of total:
USCGC Smith\r\nFPO AE 70466     0.002020
USS Johnson\r\nFPO AE 48052     0.002020
USNS Johnson\r\nFPO AE 05113    0.002020
USS Smith\r\nFPO AP 70466       0.002020
USNS Johnson\r\nFPO AP 48052    0.001768
USNV Smith\r\nFPO AA 00813      0.001515
USCGC Smith\r\nFPO AA 70466     0.001515
USCGC Jones\r\nFPO AE 22690     0.001515
USNS Johnson\r\nFPO AA 70466    0.001515
USNV Smith\r\nFPO AE 30723      0.001515
USS Smith\r\nFPO AP 22690       0.001515
USCGC Miller\r\nFPO AA 22690    0.001515
USNV Brown\r\nFPO AA 48052      0.001515
USCGC Smith\r\nFPO AP 48052     0.001263
Name: address, dtype: float64
--------------XXX-------------
```

Initial Feature Engineering Steps

In [297…

```python
org_data["address"].value_counts().head(10)
```

Out[297]:

```
USCGC Smith\r\nFPO AE 70466     8
USS Johnson\r\nFPO AE 48052     8
USNS Johnson\r\nFPO AE 05113    8
USS Smith\r\nFPO AP 70466       8
USNS Johnson\r\nFPO AP 48052    7
USNV Smith\r\nFPO AA 00813      6
USCGC Smith\r\nFPO AA 70466     6
USCGC Jones\r\nFPO AE 22690     6
USNS Johnson\r\nFPO AA 70466    6
USNV Smith\r\nFPO AE 30723      6
Name: address, dtype: int64
```

In [298…

```python
#Creating copy of original data to preserve it for later
data = org_data.copy()

# Extracting pincode from address column -- We observe while doing value_counts that pincodes are repeating mul
data['pincode'] = data['address'].str.split(" ").str[-1].astype('int64')
data['state'] = data['address'].str.split(" ").str[-2]

# Converting term column to integer
data['term'] = data['term'].str.split(" ").str[1].astype('int64')

# Converting loan_status column to numeric and re-ordering columns to make loan_status as last column
data['loan_status'] = data['loan_status'].apply(lambda x: 1 if x == 'Fully Paid' else 0)
data.insert(len(data.columns) - 1, 'loan_status', data.pop('loan_status'))

# Flag creation for Pub_rec, Mort_acc, Pub_rec_bankruptcies. If value greater than 1.0 then 1 else 0
flag_cols = ['pub_rec', 'mort_acc', 'pub_rec_bankruptcies']
for col in flag_cols:
    data[col] = data[col].apply(lambda x: 1 if x>1 else 0)

# Dropping columns:
# Issue_date, earliest_cr_line as they are dates (irrelevant) and have mutliple values which will cause curse o
# title as it is duplicate of purpose
data.drop(columns = ["issue_d", "title", "earliest_cr_line", "address"], inplace = True)
data.head()
```

Out[298]:

	loan_amnt	term	int_rate	installment	grade	sub_grade	emp_title	emp_length	home_ownership	annual_inc	...	revol_bal	revol_util
0	10000.0	36	11.44	329.48	B	B4	Marketing	10+ years	RENT	117000.0	...	36369.0	41.8
1	8000.0	36	11.99	265.68	B	B5	Credit analyst	4 years	MORTGAGE	65000.0	...	20131.0	53.3
2	15600.0	36	10.49	506.97	B	B3	Statistician	< 1 year	RENT	43057.0	...	11987.0	92.2
3	7200.0	36	6.49	220.65	A	A2	Client Advocate	6 years	RENT	54000.0	...	5472.0	21.5
4	24375.0	60	17.27	609.33	C	C5	Destiny Management Inc.	9 years	MORTGAGE	55000.0	...	24584.0	69.8

5 rows × 25 columns

APP, AE, AA represents armed forces : https://kb.blackbaud.com/knowledgebase/Article/49137

In [299…

```python
vc = data['state'].value_counts(normalize = True).reset_index().rename(columns={'index':'State', 'state':'% of
vc["% of total"] = np.round(vc["% of total"], 3) * 100
print("Shape:", vc.shape)
vc.head(5)
```

```
Shape: (54, 2)
```

Out[299]:

	State	% of total
0	AP	3.6
1	AE	3.6
2	AA	3.5
3	NJ	1.8
4	WI	1.8

In [300…

```python
vc = data['pincode'].value_counts(normalize = True).reset_index().rename(columns={'index':'Pincode', 'pincode':
vc["% of total"] = np.round(vc["% of total"], 3) * 100
vc
```

Out[300]:

	Pincode	% of total
0	70466	14.4
1	30723	14.3
2	22690	14.3
3	48052	14.1
4	813	11.6
5	29597	11.5
6	5113	11.5
7	11650	2.8
8	93700	2.8
9	86630	2.8

In [301…

```python
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 396030 entries, 0 to 396029
Data columns (total 25 columns):
 #   Column                Non-Null Count   Dtype
---  ------                --------------   -----
 0   loan_amnt             396030 non-null  float64
 1   term                  396030 non-null  int64
 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  purpose               396030 non-null  object
 12  dti                   396030 non-null  float64
 13  open_acc              396030 non-null  float64
 14  pub_rec               396030 non-null  int64
 15  revol_bal             396030 non-null  float64
 16  revol_util            395754 non-null  float64
 17  total_acc             396030 non-null  float64
 18  initial_list_status   396030 non-null  object
 19  application_type      396030 non-null  object
 20  mort_acc              396030 non-null  int64
 21  pub_rec_bankruptcies  396030 non-null  int64
 22  pincode               396030 non-null  int64
 23  state                 396030 non-null  object
 24  loan_status           396030 non-null  int64
dtypes: float64(9), int64(6), object(10)
memory usage: 75.5+ MB
```

Missing Value Treatment

In [302…

```python
(data.isna().sum()[data.isna().sum() > 0]*100/len(data)).reset_index().rename(columns={'index':'Cols', 0:'% of
```

Out[302]:

	Cols	% of total
0	emp_title	5.789208
1	emp_length	4.621115
2	revol_util	0.069692

In [303…

```python
data[['revol_util']].describe()
```

Out[303]:

	revol_util
count	395754.000000
mean	53.791749
std	24.452193
min	0.000000
25%	35.800000
50%	54.800000
75%	72.900000
max	892.300000

In [304…

```python
# filling revol_util with mean as number of null values are extremely low so wouldn't impact predicition much
data['revol_util'].fillna(np.mean(data['revol_util']), inplace = True)

# imputing employment length with mode since num of categories are quite less
data['emp_length'].fillna(data['emp_length'].mode()[0], inplace = True)

# There are a lot of categories in profession and missing values represent a % much higher than the highest cat
# So its better to create a completely new category for these people. They might even be unemployed, but we are
data['emp_title'].fillna("Unknown Profession", inplace = True)

data.isna().sum()[data.isna().sum() > 0]
```

Out[304]:

```
Series([], dtype: int64)
```

Outlier Detection

In [305…

```python
#IQR Method
def df_printoutl(df, numeric_cols = None):

    if(numeric_cols is None):
        numeric_cols = df.select_dtypes(include=np.number).columns.tolist()

    print("--------------------XXX--------------------")
    print("\t\tOutlier Counts")
    print("--------------------XXX--------------------")

    for col in numeric_cols:
        count_outliers = 0
        q1 = np.quantile(df[col], 0.25)
        q2 = np.quantile(df[col], 0.50)
        q3 = np.quantile(df[col], 0.75)
        IQR = q3 - q1
        min_outlier = q1 - 1.5*(IQR)
        max_outlier = q3 + 1.5*(IQR)

        min_outl_count = df[col][(df[col] <= min_outlier)].count()
        max_outl_count = df[col][(df[col] >= max_outlier)].count()
        total_outliers = df[col][(df[col] >= max_outlier) | (df[col] <= min_outlier)].count()

        print(f"{col}: {total_outliers} ({np.round(total_outliers/len(df), 4)}% of total)")
        if(min_outl_count == 0 and max_outl_count > 0):
            print(f"Outlier range: >= {np.round(max_outlier, 3)}")
            print("--------------------XXX--------------------")
        elif(max_outl_count == 0 and min_outl_count > 0):
            print(f"Outlier range: <= {np.round(min_outlier, 3)}")
            print("--------------------XXX--------------------")
        else:
            print(f"Outlier range: <= {np.round(min_outlier, 3)} and >= {np.round(max_outlier, 3)}")
            print("--------------------XXX--------------------")
```

In [306…

```python
df_printoutl(data)
```

```
--------------------XXX--------------------
                Outlier Counts
--------------------XXX--------------------
loan_amnt: 194 (0.0005% of total)
Outlier range: >= 38000.0
--------------------XXX--------------------
term: 396030 (1.0% of total)
Outlier range: <= 36.0 and >= 36.0
--------------------XXX--------------------
int_rate: 3777 (0.0095% of total)
Outlier range: >= 25.49
--------------------XXX--------------------
installment: 11250 (0.0284% of total)
Outlier range: >= 1042.755
--------------------XXX--------------------
annual_inc: 16714 (0.0422% of total)
Outlier range: >= 157500.0
--------------------XXX--------------------
dti: 275 (0.0007% of total)
Outlier range: >= 40.53
--------------------XXX--------------------
open_acc: 13251 (0.0335% of total)
Outlier range: >= 23.0
--------------------XXX--------------------
pub_rec: 396030 (1.0% of total)
Outlier range: <= 0.0 and >= 0.0
--------------------XXX--------------------
revol_bal: 21259 (0.0537% of total)
Outlier range: >= 40012.5
--------------------XXX--------------------
revol_util: 12 (0.0% of total)
Outlier range: >= 128.4
--------------------XXX--------------------
total_acc: 8499 (0.0215% of total)
Outlier range: >= 54.5
--------------------XXX--------------------
mort_acc: 0 (0.0% of total)
Outlier range: <= -1.5 and >= 2.5
--------------------XXX--------------------
pub_rec_bankruptcies: 396030 (1.0% of total)
Outlier range: <= 0.0 and >= 0.0
--------------------XXX--------------------
pincode: 0 (0.0% of total)
Outlier range: <= -42953.0 and >= 102655.0
--------------------XXX--------------------
loan_status: 396030 (1.0% of total)
Outlier range: <= 1.0 and >= 1.0
--------------------XXX--------------------
```

In [307…

```python
# continuos_cols = org_data.columns[~org_data.columns.isin(discrete_cols)].tolist()
continuos_cols = ['loan_amnt','int_rate','installment','annual_inc','dti','open_acc','revol_bal','revol_util','
df_printoutl(data, continuos_cols)
```

```
--------------------XXX--------------------
                Outlier Counts
--------------------XXX--------------------
loan_amnt: 194 (0.0005% of total)
Outlier range: >= 38000.0
--------------------XXX--------------------
int_rate: 3777 (0.0095% of total)
Outlier range: >= 25.49
--------------------XXX--------------------
installment: 11250 (0.0284% of total)
Outlier range: >= 1042.755
--------------------XXX--------------------
annual_inc: 16714 (0.0422% of total)
Outlier range: >= 157500.0
--------------------XXX--------------------
dti: 275 (0.0007% of total)
Outlier range: >= 40.53
--------------------XXX--------------------
open_acc: 13251 (0.0335% of total)
Outlier range: >= 23.0
--------------------XXX--------------------
revol_bal: 21259 (0.0537% of total)
Outlier range: >= 40012.5
--------------------XXX--------------------
revol_util: 12 (0.0% of total)
Outlier range: >= 128.4
--------------------XXX--------------------
total_acc: 8499 (0.0215% of total)
Outlier range: >= 54.5
--------------------XXX--------------------
```

In [308…

```python
data.annual_inc.describe()/100
```

Out[308]:

```
count     3960.300000
mean       742.031758
std        616.376212
min          0.000000
25%        450.000000
50%        640.000000
75%        900.000000
max      87065.820000
Name: annual_inc, dtype: float64
```

In [310…

```python
def cap_outlier(df, cols_to_cap):
    thresh = 0.997 # 3 Sigma

    for col in cols_to_cap:
        q = np.quantile(df[col], thresh)
        updates = df[col][df[col] > q].count()
        df.loc[df[col] > q, col] = q
        print(f"Capped {updates} values in {col} with limit {q}")
```

In [311…

```python
cols_to_cap = ["annual_inc", "dti", "open_acc", "revol_bal", "revol_util", "total_acc"]
cap_outlier(data, cols_to_cap)
```

```
Capped 1145 values in annual_inc with limit 350000.0
Capped 1180 values in dti with limit 38.95
Capped 1000 values in open_acc with limit 33.0
Capped 1189 values in revol_bal with limit 152010.82300000003
Capped 1167 values in revol_util with limit 100.0
Capped 1131 values in total_acc with limit 67.0
```

In [312…

```python
data.describe().T
```

Out[312]:

	count	mean	std	min	25%	50%	75%	max
loan_amnt	396030.0	14113.888089	8357.441341	500.00	8000.00	12000.00	20000.00	40000.000
term	396030.0	41.698053	10.212038	36.00	36.00	36.00	36.00	60.000
int_rate	396030.0	13.639400	4.472157	5.32	10.49	13.33	16.49	30.990
installment	396030.0	431.849698	250.727790	16.08	250.33	375.43	567.30	1533.810
annual_inc	396030.0	73449.693854	43328.710265	0.00	45000.00	64000.00	90000.00	350000.000
dti	396030.0	17.342061	8.108238	0.00	11.28	16.91	22.98	38.950
open_acc	396030.0	11.298346	5.070883	0.00	8.00	10.00	14.00	33.000
pub_rec	396030.0	0.020248	0.140849	0.00	0.00	0.00	0.00	1.000
revol_bal	396030.0	15556.759565	16503.799126	0.00	6025.00	11181.00	19620.00	152010.823
revol_util	396030.0	53.780573	24.387575	0.00	35.90	54.80	72.90	100.000
total_acc	396030.0	25.387142	11.767639	2.00	17.00	24.00	32.00	67.000
mort_acc	396030.0	0.399066	0.489707	0.00	0.00	0.00	1.00	1.000
pub_rec_bankruptcies	396030.0	0.005871	0.076396	0.00	0.00	0.00	0.00	1.000
pincode	396030.0	33998.447686	25605.865779	813.00	11650.00	29597.00	48052.00	93700.000
loan_status	396030.0	0.803871	0.397068	0.00	1.00	1.00	1.00	1.000

In [313…

```python
data.describe(include = 'object').T
```

Out[313]:

	count	unique	top	freq
grade	396030	7	B	116018
sub_grade	396030	35	B3	26655
emp_title	396030	173106	Unknown Profession	22927
emp_length	396030	11	10+ years	144342
home_ownership	396030	6	MORTGAGE	198348
verification_status	396030	3	Verified	139563
purpose	396030	14	debt_consolidation	234507
initial_list_status	396030	2	f	238066
application_type	396030	3	INDIVIDUAL	395319
state	396030	54	AP	14308

Summary

1. **Observation:** Data has 396030 entries, 25 columns (after initial feature engg). Data had very few null values and they were imputed without losing any data
2. **Observation:** The column address was extremely granular and has multiple repetitions
   Fix: Divided the address field into state and pincode to bring about some homogenity in the data. This will help us check if location is a factor in deciding the loan status
3. **Observation:** Columns such as Pub_rec, Mort_acc, Pub_rec_bankruptcies are considered negative for a person's creditworthiness. Having more of these can raise concerns for lenders.
   Fix: Created flags
4. **Observation:** Our target value (loan status) has significant class imbalance, this will create issues in prediction.
   Fix: We can under techniques like undersampling, oversampling and smote to reduce this
5. **Observation:** Some features such as grade, verification status, initial list status have an approx. even distribution; Whereas, features like purpose, application type and home ownership have skewed data
   Recommendation: LoanTap should notice this skew in some data and can gather more insights about their customers to provide services for their target audience. They can also use this data to try to increase their penetration in other markets they aren't dominant in yet
6. **Observation:** We have very few outliers (as a % of total), but they all represent real valued data. However, some outliers are extremely signifcant (like for very rich income individuals).
   Fix: We will cap the data to reduce the disparity in data. To cap the data we will use 99.7% quantile for upper value and 0.03% quantile for lower value

Exploratory Data Analysis

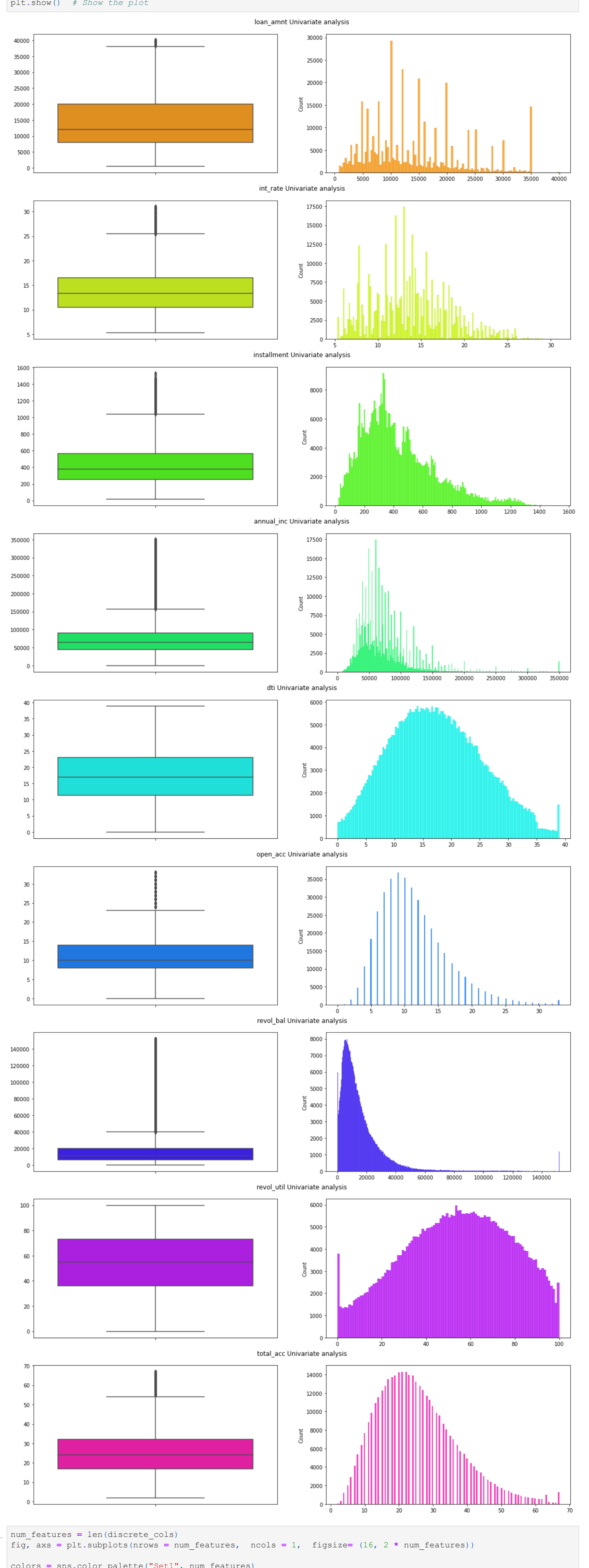
In [314…

```
continuous_cols = ['loan_amnt','int_rate','installment','annual_inc','dti','open_acc','revol_bal','revol_util',
                   'total_acc']
discrete_cols = data.columns[~data.columns.isin(continuous_cols)].tolist()
print(continuous_cols)
print(discrete_cols)
```

```
['loan_amnt', 'int_rate', 'installment', 'annual_inc', 'dti', 'open_acc', 'revol_bal', 'revol_util', 'total_ac
c']
['term', 'grade', 'sub_grade', 'emp_title', 'emp_length', 'home_ownership', 'verification_status', 'purpose',
'pub_rec', 'initial_list_status', 'application_type', 'mort_acc', 'pub_rec_bankruptcies', 'pincode', 'state',
'loan_status']
```

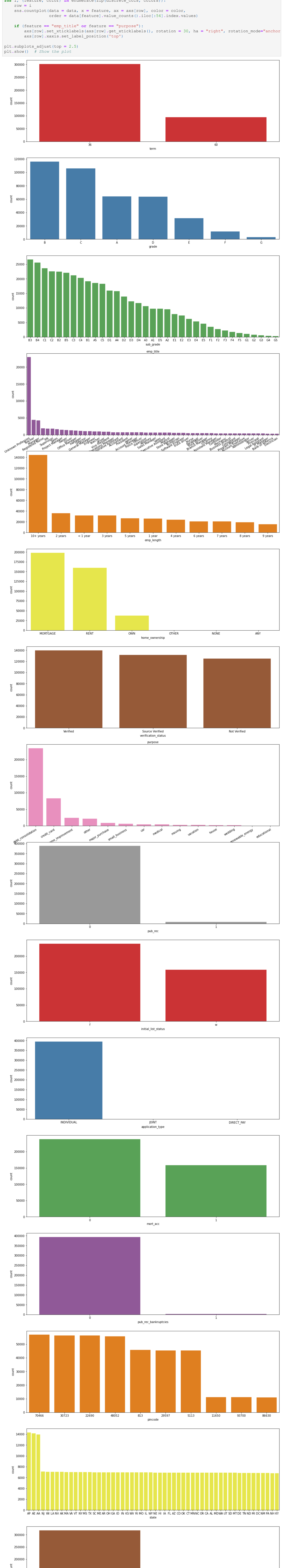
In [315…

```
num_features = len(continuos_cols)
fig, axs = plt.subplots(nrows = num_features,  ncols = 2,  figsize= (5, 5 * num_features))

colors = sns.color_palette("hsv", num_features)

for i, (feature, color) in enumerate(zip(continuos_cols, colors)):
    row = i
    sns.boxplot(data=data, y = feature, ax = axs[row][0], color = color).set(ylabel = "")
    sns.histplot(data=data, x = feature, ax = axs[row][1], bins='auto', color = color).set(xlabel = "")
    axs[row][0].set_title(f"{feature} Univariate analysis", x = 1.1, y = 1.05, fontsize = 12)

plt.subplots_adjust(top = 1, right = 3)
plt.show()   # Show the plot
```



In [316…

```
num_features = len(discrete_cols)
fig, axs = plt.subplots(nrows = num_features,  ncols = 1,  figsize= (16, 2 * num_features))

colors = sns.color_palette("Set1", num_features)

for i, (feature, color) in enumerate(zip(discrete_cols, colors)):
    row = i
    sns.countplot(data = data, x = feature, ax = axs[row], color = color,
                  order = data[feature].value_counts().iloc[:54].index.values)

    if (feature == "emp_title" or feature == "purpose"):
        axs[row].set_xticklabels(axs[row].get_xticklabels(), rotation = 30, ha = "right", rotation_mode="anchor
        axs[row].xaxis.set_label_position('top')

plt.subplots_adjust(top = 2.5)
plt.show()   # Show the plot
```

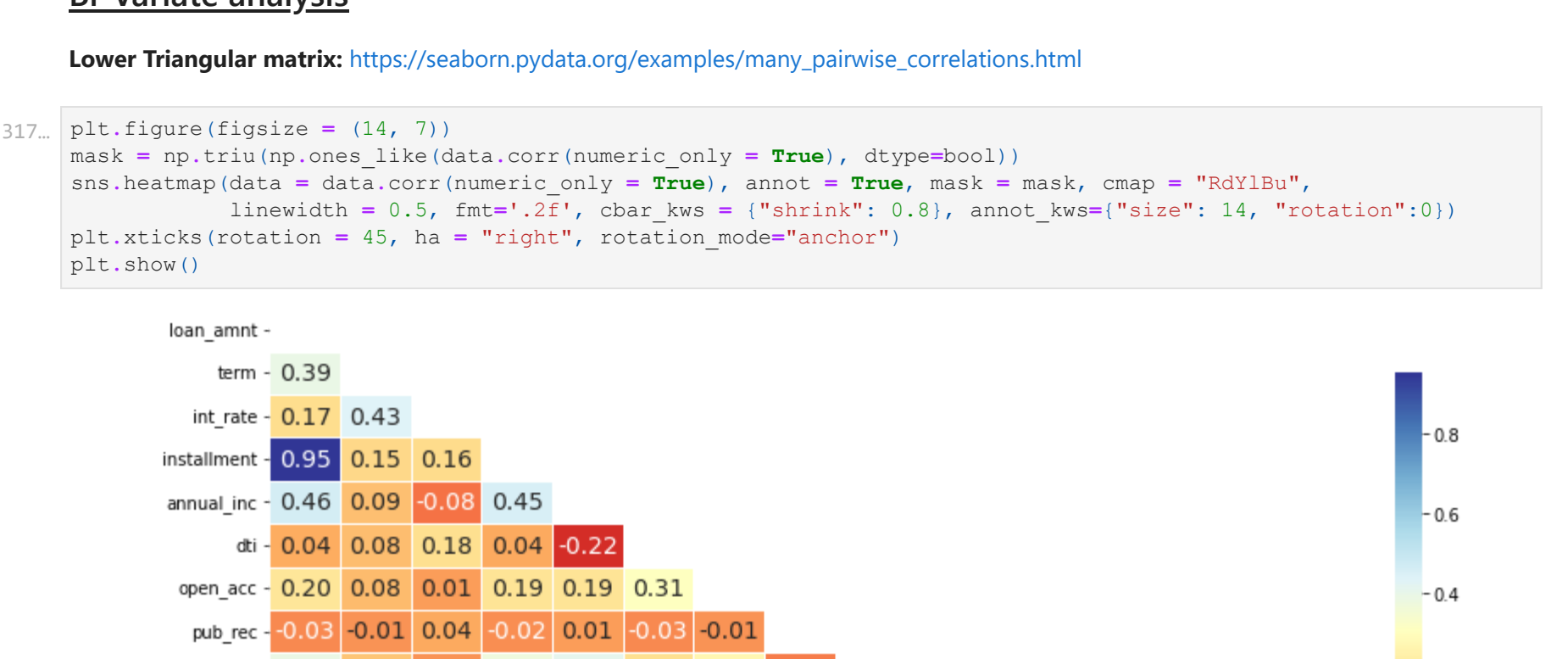


Bi-Variate analysis

Lower Triangular matrix: https://seaborn.pydata.org/examples/many_pairwise_correlations.html

In [317…

```
plt.figure(figsize = (14, 7))
mask = np.triu(np.ones_like(data.corr(numeric_only = True), dtype=bool))
sns.heatmap(data = data.corr(numeric_only = True), annot = True, mask = mask, cmap = "RdYlBu",
            linewidth = 0.5, fmt='.2f', cbar_kws = {"shrink": 0.8}, annot_kws={"size": 14, "rotation":0})
plt.xticks(rotation = 45, ha = "right", rotation_mode="anchor")
plt.show()
```

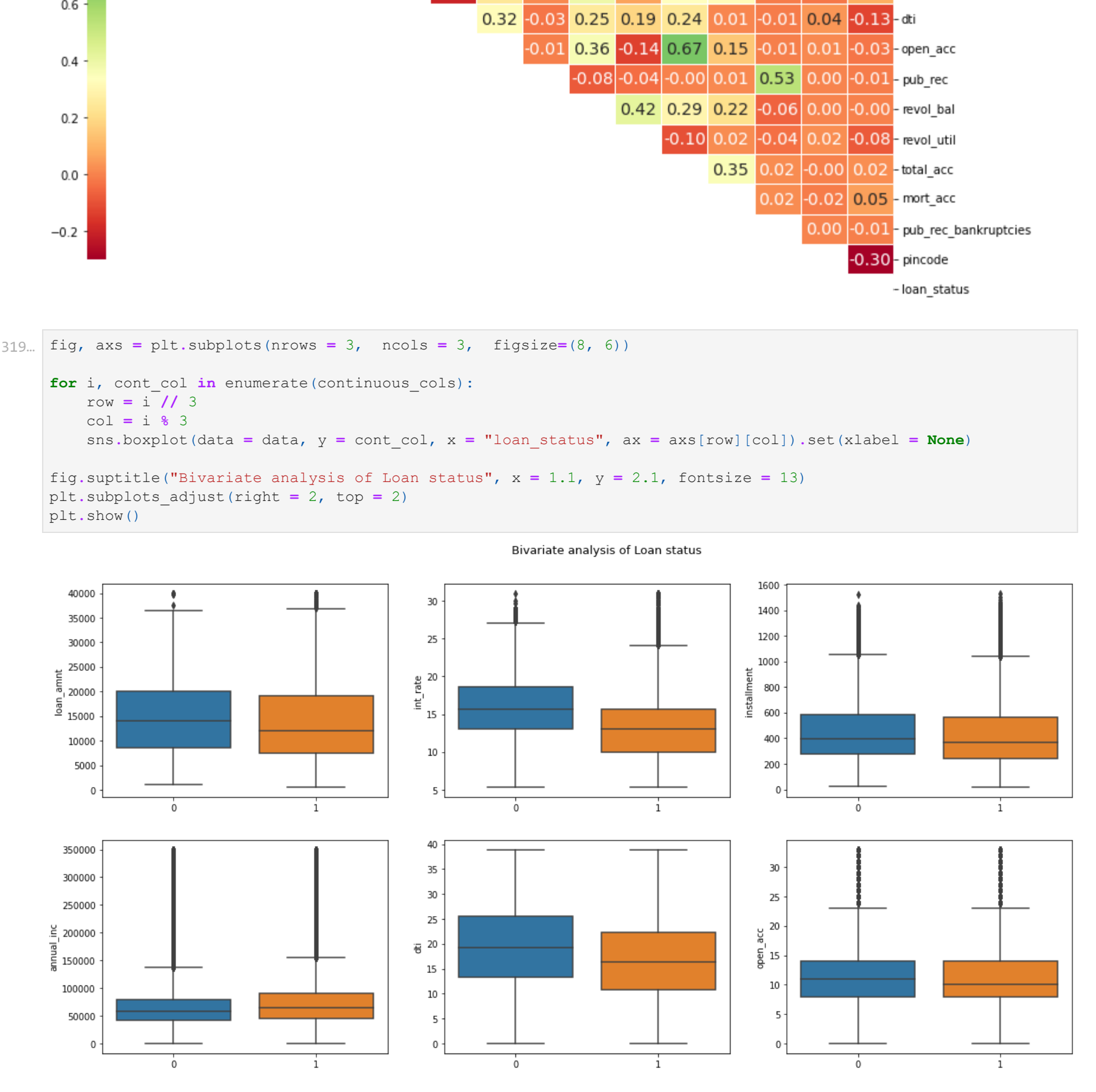


In [318…

```
plt.figure(figsize = (14,7))
mask = np.tril(np.ones_like(data.corr(method='spearman', numeric_only = True), dtype=bool))

sns.heatmap(data = data.corr(method='spearman', numeric_only = True), annot = True, mask = mask, cmap = "RdYlGn
            linewidth = 0.5, fmt='.2f', cbar_kws = {"shrink": 0.8, "location": "left"}, annot_kws={"size": 14})

plt.tick_params(top=True, bottom = False, right = True, left = False,
                labeltop=True, labelbottom=False, labelright=True, labelleft=False)
plt.xticks(rotation = 45, ha = "left", rotation_mode="anchor")
plt.yticks(rotation = 0, va = "center", rotation_mode="anchor")
plt.show()
```

In [319…

```
fig, axs = plt.subplots(nrows = 3,  ncols = 3,  figsize=(8, 6))

for i, cont_col in enumerate(continuous_cols):
    row = i // 3
    col = i % 3
    sns.boxplot(data = data, y = cont_col, x = "loan_status", ax = axs[row][col]).set(xlabel = None)

fig.suptitle("Bivariate analysis of Loan status", x = 1.1, y = 2.1, fontsize = 13)
plt.subplots_adjust(right = 2, top = 2)
plt.show()
```



In [320…

```
discrete_cols_2 = [col for col in discrete_cols if col not in ["loan_status","emp_title", "purpose"]]

for i, disc_col in enumerate(discrete_cols_2):
    temp_df = data.groupby(disc_col)["loan_status"].value_counts(normalize=True)*100
    temp_df = temp_df.rename('percent').reset_index()
    plot = sns.catplot(data = temp_df, x=disc_col, y="percent", hue="loan_status", kind='bar', height=6, aspect

    for p in plot.ax.patches:
        txt = str(p.get_height().round(2)) + '%'
        x = p.get_x() + (p.get_width()/2)
        y = p.get_height() + (0.05 * p.get_height())
        plt.annotate(txt, (x, y), ha='center')

plt.show()
```

Observations

1. Loan amount and installment features have extremely high correlation. This implies that one variable explains the other and thereofore this is a certain case of **Multicollinearity**
2. The features loan_amount, term, annual_income, installment etc are correlated as high income people would take higher loans leading to higher duration of repayment and higher installment values
3. Among the numerical variables, interest rate and pincode have a high correlation with our target variable.
4. Features like **"emp_title", "purpose"** have to be binned based on business intelligence. There are too many columns to directly infer their effect on target and if we encode them it will cause curse of dimentionality, **hence we will drop them**
5. It's clear from the bivariate **boxplot** analysis that:
   - Lower interest rate loans have a higher chance of repayment
   - Lower dti ratio have a higher chance of repayment
   - Lower revol_util has a higher chance of repayment
6. We can infer from the bivariate **countplots** that:
   - Lower term loans have a higher chance of repayment
   - **Grade and sub_grade are extremely important features determining the repayment chances. Higher grade individuals have a great track record or loan repayment**. We will encode these values ordinally
   - Verified loans have higher repayment chance than unverified loans
   - It seems that application type has high correlation with repayment however **we must remember that the data for this feature is extremely skewed**
   - As we saw earlier, geographical location seems to affect loan repayment quite a lot
7. **Sub-grade is a more grannular version of grade so we can keep one and drop the other**

Model Building

Encoding

In [321…
```
df = data.copy()
df = df.drop(columns = ["emp_title", "purpose", "state", "grade"])
df.shape
```

Out[321]:
```
(396030, 21)
```

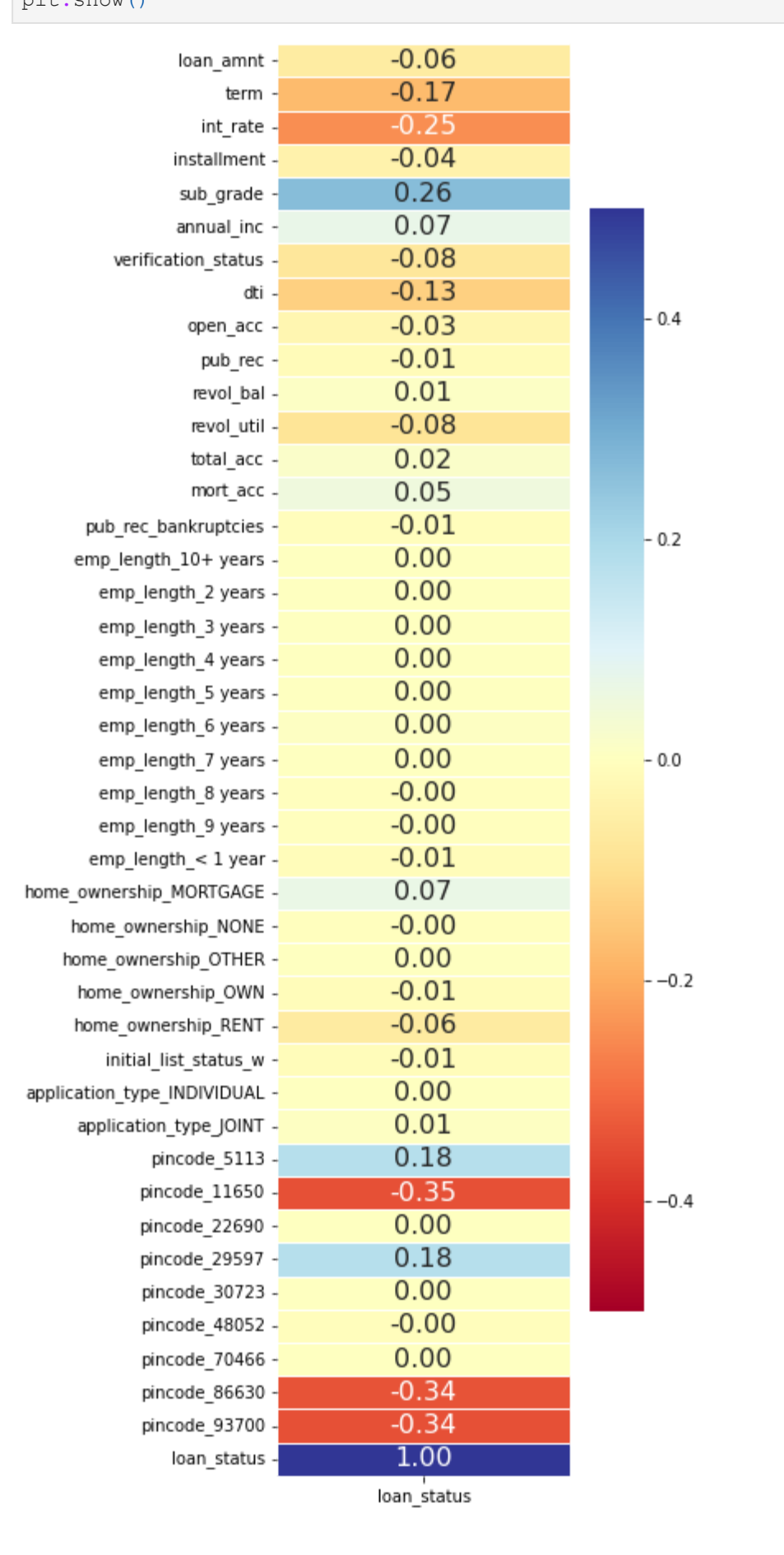
In [334…
```
dummy_columns = ['emp_length','home_ownership','initial_list_status','application_type','pincode']
ord_columns = ['sub_grade','verification_status']

encoded_data = pd.get_dummies(df, columns=dummy_columns, drop_first = True) #Dropping first column to avoid dum

sorted_grades = sorted(df.sub_grade.value_counts().index.values)
encoding_dict = {}
for i in range(0, len(sorted_grades)):
    encoding_dict[sorted_grades[i]] = len(sorted_grades)-i
encoded_data['sub_grade'] = encoded_data['sub_grade'].map(encoding_dict)

encoding_dict_2 = {"Verified":2, "Source Verified":1, "Not Verified":0}
encoded_data['verification_status'] = encoded_data['verification_status'].map(encoding_dict_2)

#re-ordering columns again
encoded_data.insert(len(encoded_data.columns) - 1, 'loan_status', encoded_data.pop('loan_status'))
encoded_data.head()
```

Out[334]:

	loan_amnt	term	int_rate	installment	sub_grade	annual_inc	verification_status	dti	open_acc	pub_rec	...	pincode_5113	pincode_1165
0	10000.0	36	11.44	329.48	27	117000.0	0	26.24	16.0	0	...	0	
1	8000.0	36	11.99	265.68	26	65000.0	0	22.05	17.0	0	...	1	
2	15600.0	36	10.49	506.97	28	43057.0	1	12.79	13.0	0	...	1	
3	7200.0	36	6.49	220.65	34	54000.0	0	2.60	6.0	0	...	0	
4	24375.0	60	17.27	609.33	21	55000.0	2	33.95	13.0	0	...	0	

5 rows × 43 columns

In [335…
```
plt.figure(figsize = (4, 16))

# Converting to dataframe as corr doesn't support series data
sns.heatmap(data = encoded_data.corr()[['loan_status']], annot = True, cmap = "RdYlBu", vmax = 0.5, vmin = -0.5
            linewidth = 0.5, fmt='.2f', cbar = True, annot_kws={"size": 16, "rotation":0})

plt.show()
```



Pre-Processing

In [344…
```
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split

X = encoded_data.iloc[:, :-1]
y = encoded_data.iloc[:, -1]

xTrain, xTest, yTrain, yTest = train_test_split(X, y, train_size = 0.75, test_size = 0.25, random_state = 9, sh
print(xTrain.shape, yTrain.shape, xTest.shape, yTest.shape)

scaler = StandardScaler() #Standard Scaler is preffered as it's robust against outliers, and we have a lot of o
xTrain_Scaled = scaler.fit_transform(xTrain)
xTrain_Scaled_df = pd.DataFrame(data = scaler.fit_transform(xTrain), columns = xTrain.columns, index = xTrain.i
xTest_Scaled = scaler.transform(xTest)
xTest_Scaled_df = pd.DataFrame(data = scaler.transform(xTest), columns = xTest.columns, index = xTest.index)
```

```
(297022, 42) (297022,) (99008, 42) (99008,)
```

In [353…
```
from statsmodels.stats.outliers_influence import variance_inflation_factor
from statsmodels.tools.tools import add_constant

def dropVIF(df, col_list = None):
    thresh = 10 #V.I.F of > 10 will be dropped
    temp_df = add_constant(df) #statsmodel requires constant column seperately
    if col_list is None:
        col_list = df.columns.tolist()
    elif any(elem not in df.columns.tolist() for elem in col_list):
        return "One or more element passed in column list aren't present in given dataframe"

    col_list.insert(0, "const")
    temp_df = temp_df[col_list]
    flag = 1

    while(flag == 1): # While any column has V.I.F > 10
        flag = 0 #reset flag
        vif_max = 1 #lowest possible VIF score
        col_to_drop = None

        for i in range(len(temp_df.columns)): # Calculating V.I.F for each column
            vif = variance_inflation_factor(temp_df, i)
            vif = np.round(vif, 0)
            if(vif > thresh and vif > vif_max):
                col_to_drop = temp_df.columns[i]
                vif_max = vif

        print(f"Dropped column {col_to_drop} with V.I.F = {vif_max}")
        #temp_df.drop(columns = df.columns[vif_loc], inplace = True)


def getVIF(df):
    temp_df = add_constant(df) #statsmodel requires constant column seperately
    return pd.DataFrame(data = [variance_inflation_factor(temp_df, i) for i in range(0, temp_df.shape[1])],
            index = temp_df.columns.tolist(),
            columns = ["VIF"])
```

In [356…
```
VIF_df = getVIF(xTrain_Scaled_df)
VIF_df
```

Out[356]:

	VIF
const	1.000000
loan_amnt	58.698951
term	6.646840
int_rate	21.713933
installment	50.900002
sub_grade	21.454373
annual_inc	1.713175
verification_status	1.174336
dti	1.420700
open_acc	2.137311
pub_rec	1.413369
revol_bal	1.556458
revol_util	1.354573
total_acc	2.191897
mort_acc	1.494894
pub_rec_bankruptcies	1.403470
emp_length_10+ years	4.292723
emp_length_2 years	2.163669
emp_length_3 years	2.048969
emp_length_4 years	1.809903
emp_length_5 years	1.893146
emp_length_6 years	1.712220
emp_length_7 years	1.719341
emp_length_8 years	1.663412
emp_length_9 years	1.538372
emp_length_< 1 year	2.052605
home_ownership_MORTGAGE	24756.528022
home_ownership_NONE	8.334163
home_ownership_OTHER	29.663377
home_ownership_OWN	8487.441357
home_ownership_RENT	23846.926783
initial_list_status_w	1.071810
application_type_INDIVIDUAL	2.470954
application_type_JOINT	2.462060
pincode_5113	1.760634
pincode_11650	1.226181
pincode_22690	1.917134
pincode_29597	1.764420
pincode_30723	1.915895
pincode_48052	1.909913
pincode_70466	1.923908
pincode_86630	1.219957
pincode_93700	1.223428

In [358…
```
dropVIF(xTrain_Scaled_df) #encoded_data.columns[0:15].tolist()
```

```
Dropped column home_ownership_MORTGAGE with V.I.F = 24757.0
```

In [364…
```
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report

lr = LogisticRegression(penalty='l2', max_iter = 1000)
lr.fit(xTrain_Scaled, yTrain)
yPred = lr.predict(xTest_Scaled)
report = classification_report(yTest, yPred)
print(report)
```

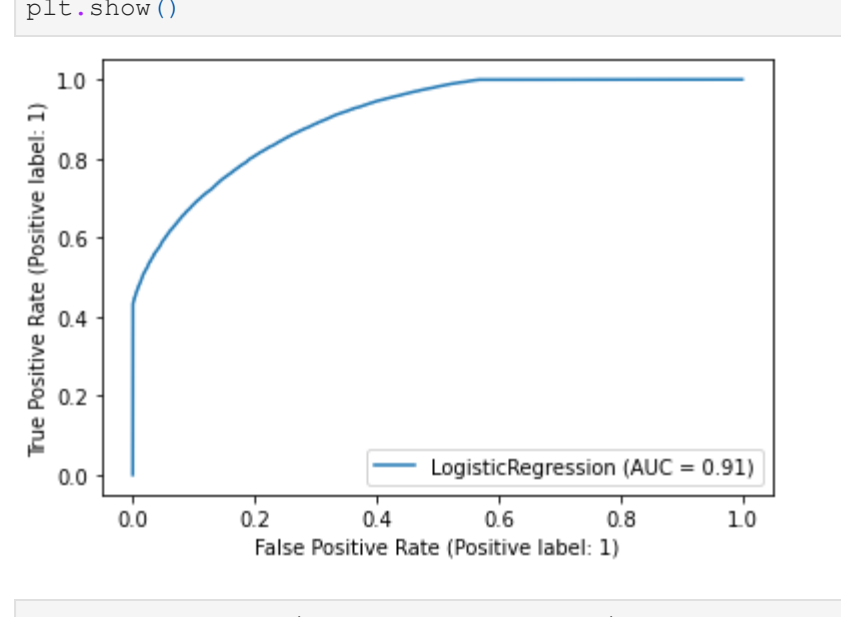
```
              precision    recall  f1-score   support

           0       0.92      0.48      0.63     19575
           1       0.88      0.99      0.93     79433

    accuracy                           0.89     99008
   macro avg       0.90      0.73      0.78     99008
weighted avg       0.89      0.89      0.87     99008
```

In [388…
```
feature_imp = pd.DataFrame(data = np.round(np.insert(lr.coef_, 0, lr.intercept_), 2),
            index = ["Constant"] + xTrain_Scaled_df.columns.tolist(),
            columns = ["Co-efficients"])
feature_imp["Sorted"] = feature_imp["Co-efficients"].abs()
feature_imp.sort_values(by = "Sorted", axis = 0, ascending=False)
```

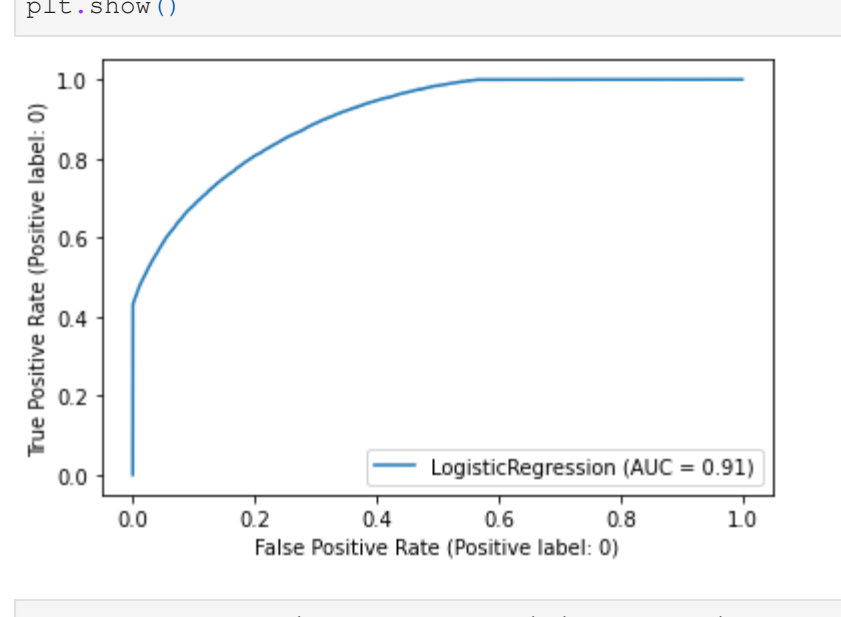
Out[388]:

	Co-efficients	Sorted
Constant	3.87	3.87
pincode_11650	-3.09	3.09
pincode_93700	-3.07	3.07
pincode_86630	-3.04	3.04
pincode_70466	-2.64	2.64
pincode_48052	-2.63	2.63
pincode_22690	-2.62	2.62
pincode_30723	-2.62	2.62
pincode_5113	0.87	0.87
pincode_29597	0.87	0.87
sub_grade	0.75	0.75
int_rate	0.30	0.30
home_ownership_RENT	-0.27	0.27
dti	-0.20	0.20
term	-0.19	0.19
annual_inc	0.18	0.18
home_ownership_MORTGAGE	-0.14	0.14
open_acc	-0.13	0.13
total_acc	0.12	0.12
home_ownership_OWN	-0.12	0.12
revol_util	-0.09	0.09
revol_bal	0.08	0.08
installment	-0.07	0.07
verification_status	-0.04	0.04
pub_rec	-0.03	0.03
loan_amnt	-0.02	0.02
application_type_JOINT	0.02	0.02
mort_acc	0.02	0.02
application_type_INDIVIDUAL	-0.01	0.01
emp_length_8 years	0.01	0.01
emp_length_< 1 year	-0.01	0.01
emp_length_9 years	0.01	0.01
emp_length_7 years	0.01	0.01
emp_length_5 years	0.01	0.01
emp_length_4 years	0.01	0.01
emp_length_3 years	0.01	0.01
emp_length_2 years	0.01	0.01
emp_length_10+ years	-0.01	0.01
emp_length_6 years	0.01	0.01
home_ownership_OTHER	0.00	0.00
initial_list_status_w	-0.00	0.00
home_ownership_NONE	-0.00	0.00
pub_rec_bankruptcies	-0.00	0.00

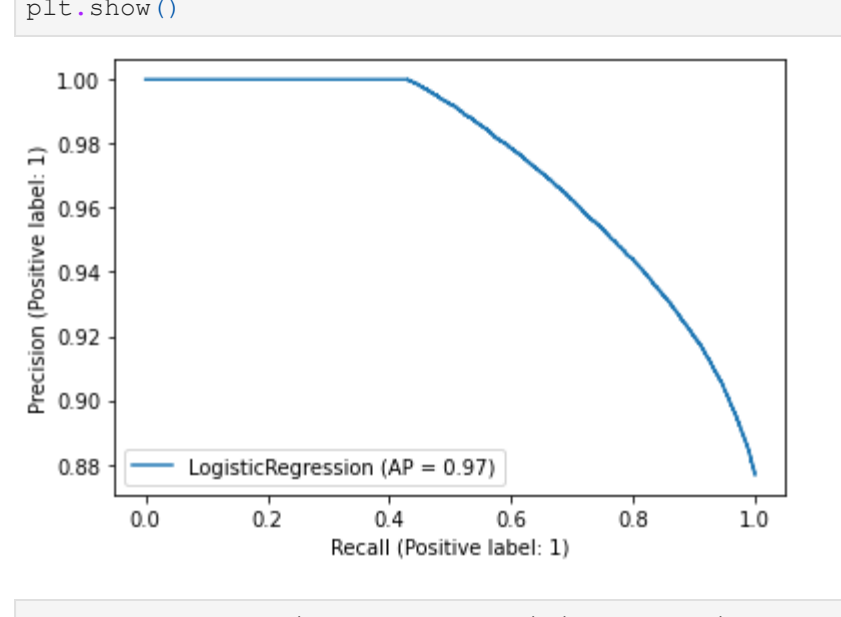
In [389…
```
from sklearn.metrics import RocCurveDisplay

RocCurveDisplay.from_estimator(lr, xTest_Scaled, yTest, pos_label = 1)
plt.show()
```



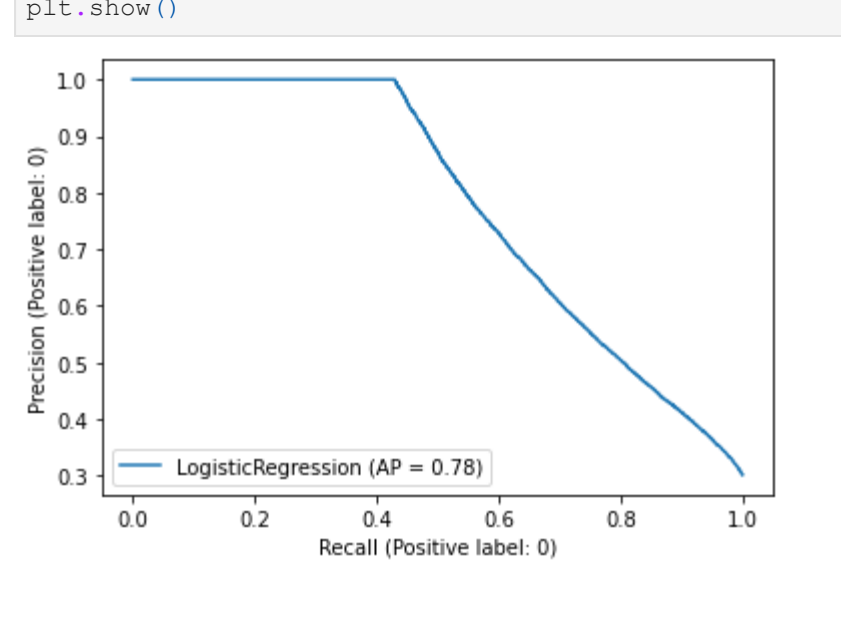
In [390…
```
from sklearn.metrics import RocCurveDisplay

RocCurveDisplay.from_estimator(lr, xTest_Scaled, yTest, pos_label = 0)
plt.show()
```



In [392…
```
from sklearn.metrics import PrecisionRecallDisplay

PrecisionRecallDisplay.from_estimator(lr, xTest_Scaled, yTest, pos_label = 1)
plt.show()
```



In [391…
```
from sklearn.metrics import PrecisionRecallDisplay

PrecisionRecallDisplay.from_estimator(lr, xTest_Scaled, yTest, pos_label = 0)
plt.show()
```



Comments on Results

1. Our data had some variables with extremely high multicollinearity, so reduce it, I have used l2 regularization which will deal with the multicollinearity by constricting the coefficient and by keeping all the variables
2. The most important features in our predictions were **pincode and sub_grade**. I had noticed this clearly when we performed bi-variate feature analysis
3. We can see that while the overall accuracy our our model is good, there exists a clear imbalance in the results achieved for repayed loans and defaulted loans with the former having better results overall (high F1 score). This means our model is **not performing well in detecting the defaulters (class "0"**. This phenonmenon might have multiple reasons but as we had already seen that the classes in our data were **heavily imbalanced**, this makes the model focus more on a specific class. To solve this, **techniques like undersampling, oversampling and SMOTE can be used**
4. We can see through our classification metrics that specifically, the **Recall of target variable is quite bad** for The recall is 0.48, which means that the model is able to identify 48% of the individuals who actually defaulted on their loans. Whereas, since the precision is 0.92, which means that when the model predicts an individual to default on their loan, it is correct 92% of the time. Hence, our **False Negatives for class '0' are quite high, so our model is predicting more people as defaulters than they actually are**
5. The AUC of 0.91 suggests that the model has a **high level of discrimination power** in correctly ranking the probabilities of both defaulting individuals (class 0) and individuals who paid their loans back on time (class 1). It indicates that the model is generally effective in distinguishing between the two classes.

Tradeoff Questions

1. How can we make sure that our model can detect real defaulters and there are less false positives? This is important as we can lose out on an opportunity to finance more individuals and earn interest on it?
   As I have explained we can use undersampling and oversampling techniques for the same. We can also play with the threshold values to get the best threshold using which helps us reduce FN
2. Since NPA (non-performing asset) is a real problem in this industry, it's important we play safe and shouldn't disburse loans to anyone.
   This is true, our model is also giving high precision for defaulters which means it's doing well to save LoanTap money by forewarning the lender before default

Miscellaneous : Ignore

Questionnaire (Answers should present in the text editor along with insights):

1. What percentage of customers have fully paid their Loan Amount?
2. Comment about the correlation between Loan Amount and Installment features.
3. The majority of people have home ownership as ___.
4. People with grades 'A' are more likely to fully pay their loan. (T/F)
5. Name the top 2 afforded job titles.
6. Thinking from a bank's perspective, which metric should our primary focus be on..
7. ROC AUC
8. Precision
9. Recall
10. F1 Score
11. How does the gap in precision and recall affect the bank?
12. Which were the features that heavily affected the outcome?
13. Will the results be affected by geographical location? (Yes/No)

1. should we add_constant while checking V.I.F, if constant is high then what to do?
   https://stackoverflow.com/questions/59694427/what-does-the-high-vif-for-the-constant-term-intercept-indicate
   https://stats.stackexchange.com/questions/332428/regression-model-constant-causes-multicollinearity-warning-but-not-in-standardi
2. PCA + Decision boundary

In [ ]:
```
locs = [df.columns.get_loc(c) for c in col_list if c in df]
```

In [345…
```
temp_df = add_constant(encoded_data)
pd.DataFrame(data = [variance_inflation_factor(temp_df, i) for i in range(0, temp_df.shape[1])],
            index = temp_df.columns.tolist(),
            columns = ["VIF"])
```

Out[345]:

	VIF
const	134564.130215
loan_amnt	59.166840
term	6.683239
int_rate	21.833780
installment	51.296394
sub_grade	21.641137
annual_inc	1.715397
verification_status	1.174971
dti	1.424990
open_acc	2.140934
pub_rec	1.413320
revol_bal	1.556564
revol_util	1.354765
total_acc	2.193000
mort_acc	1.494578
pub_rec_bankruptcies	1.403819
emp_length_10+ years	4.290924
emp_length_2 years	2.169255
emp_length_3 years	2.047375
emp_length_4 years	1.811337
emp_length_5 years	1.893157
emp_length_6 years	1.716832
emp_length_7 years	1.717539
emp_length_8 years	1.664160
emp_length_9 years	1.537277
emp_length_< 1 year	2.048791
home_ownership_MORTGAGE	33007.202027
home_ownership_NONE	11.334034
home_ownership_OTHER	38.327813
home_ownership_OWN	11385.142601
home_ownership_RENT	31777.383815
initial_list_status_w	1.071600
application_type_INDIVIDUAL	2.500057
application_type_JOINT	2.491329
pincode_5113	1.762635
pincode_11650	1.516016
pincode_22690	1.965937
pincode_29597	1.763699
pincode_30723	1.966817
pincode_48052	1.960869
pincode_70466	1.973256
pincode_86630	1.505101
pincode_93700	1.512642
loan_status	1.850755

In [337…

```
from statsmodels.stats.outliers_influence import variance_inflation_factor
from statsmodels.tools.tools import add_constant

pd.DataFrame(data = [variance_inflation_factor(encoded_data, i) for i in range(0, encoded_data.shape[1])],
            index = encoded_data.columns.tolist(),
            columns = ["VIF"])
```

Out[337]:

	VIF
loan_amnt	227.841468
term	118.036616
int_rate	223.367307
installment	203.417954
sub_grade	303.407590
annual_inc	6.643866
verification_status	3.067987
dti	7.942459
open_acc	12.769271
pub_rec	1.442525
revol_bal	2.939612
revol_util	7.942673
total_acc	12.399727
mort_acc	2.487065
pub_rec_bankruptcies	1.412108
emp_length_10+ years	6.751358
emp_length_2 years	2.384889
emp_length_3 years	2.225101
emp_length_4 years	1.927844
emp_length_5 years	2.028771
emp_length_6 years	1.812169
emp_length_7 years	1.812804
emp_length_8 years	1.748755
emp_length_9 years	1.599071
emp_length_< 1 year	2.226961
home_ownership_MORTGAGE	1227.366159
home_ownership_NONE	1.194951
home_ownership_OTHER	1.683642
home_ownership_OWN	233.773335
home_ownership_RENT	987.158600
initial_list_status_w	1.782007
application_type_INDIVIDUAL	1377.953528
application_type_JOINT	2.478282
pincode_5113	1.990758
pincode_11650	1.560147
pincode_22690	2.293134
pincode_29597	1.992403
pincode_30723	2.294221
pincode_48052	2.283125
pincode_70466	2.304719
pincode_86630	1.547945
pincode_93700	1.556365
loan_status	9.435835

In [ ]:

```
# Binning & converting employment length column to integer
# data['emp_length'] = data['emp_length'].apply(lambda yrs: "0 years" if yrs == "< 1 year" else ("10 years" if
# data['emp_length'] = data['emp_length'].str.split(" ").str[0].astype('float64')
# data['emp_length'] = data['emp_length'].apply(lambda yrs: 0 if yrs < 10 else 1)
```

In [ ]:

```
# Ordinal encoding verification_status :
# 'Verified' --> verified by LoanTap, 'Source verified' --> income source was verified
```