LoanTap Case Study

Gautam Naik (gautamnaik1994@gmail.com)

Github link (https://github.com/gautamnaik1994/LoanTap-ML-CaseStudy)

About LoanTap

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

Business Problem

Loantap aims to develop a machine-learning model to assess whether an individual should qualify for a loan. As data scientists, our task is to analyze a person's attributes and decide whether they should receive a credit line by creating a predictive model. Additionally, we need to provide recommendations and actionable insights.

Metric

1. ROC AUC
2. Precision
3. Recall
4. F1 Score

Data Features

Feature	Description
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 [ ]:
```
from pyspark.sql import SparkSession
from pyspark.sql.functions import split,count,lower, regexp_extract
import pyspark.sql.functions as sf
import matplotlib.pyplot as plt
from pyspark.sql.types import StringType, ArrayType, StructField, StructType, IntegerType, FloatType, DoubleType
from pyspark.ml.feature import StringIndexer, VectorAssembler
# , MinMaxScaler, StandardScaler
import pandas as pd
import seaborn as sns
sns.set_style("whitegrid")
pd.set_option('display.max_columns', None)
from scipy.stats import chi2_contingency
import numpy as np
from IPython.core.interactiveshell import InteractiveShell
InteractiveShell.ast_node_interactivity = "all"

from sklearnex import patch_sklearn
patch_sklearn()
from sklearn.preprocessing import StandardScaler, MinMaxScaler, OneHotEncoder, OrdinalEncoder, TargetEncoder
from sklearn.model_selection import train_test_split
from statsmodels.stats.outliers_influence import variance_inflation_factor
# from matplotlib_inline.backend_inline import set_matplotlib_formats
# set_matplotlib_formats('svg')
```

```
Intel(R) Extension for Scikit-learn* enabled (https://github.com/intel/scikit-learn-intelex)
```

In [ ]:
```
# Create a SparkSession
spark = SparkSession.builder \
    .appName("LoanTap") \
    .config("spark.sql.debug.maxToStringFields", 1000) \
    .config("spark.sql.execution.arrow.pyspark.enabled", "true") \
    .config("spark.sql.shuffle.partitions", 1) \
    .config("spark.network.timeout", "120s") \
    .config("spark.executor.heartbeatInterval", "10s") \
    .getOrCreate()
```

```
Setting default log level to "WARN".
To adjust logging level use sc.setLogLevel(newLevel). For SparkR, use setLogLevel(newLevel).
24/07/31 18:28:25 WARN NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
```

In [ ]:
```
df = spark.read \
    .option("header", "true") \
    .option("inferSchema", "true") \
    .option("multiLine", "true") \
    .option("escape", "\"") \
    .csv("./logistic_regression.csv")
df.cache();
```

In [ ]:
```python
invalid_address_count = 0
def split_address(address):

    address = address.split("\n")[1]
    city,state,zipcode = "Unknown", "Unknown", "Unknown"
    try:
        if("," in address):
            # 1726 Cooper Passage Suite 129\nNorth Deniseberg, DE 30723
            # state = DE
            # city = North Deniseberg
            # zip = 30723
            city = address.split(",")[0]
            state = address.split(",")[1].split(" ")[1]
            zipcode = address.split(",")[1].split(" ")[2]
        else:
            # USCGC Tran\nFPO AP 22690
            city, state, zipcode = address.split(" ")
    except:
        global invalid_address_count
        invalid_address_count += 1
    return (city, state, zipcode)

# split_address_udf = sf.udf(split_address, ArrayType(StringType()))

split_address_udf = sf.udf(split_address, StructType([
    StructField("city", StringType(), True),
    StructField("state", StringType(), True),
    StructField("zipcode", StringType(), True)
]))
```

In [ ]:
```python
to_drop=[]
```

In [ ]:
```python
df.printSchema()
```

```
root
 |-- loan_amnt: double (nullable = true)
 |-- term: string (nullable = true)
 |-- int_rate: double (nullable = true)
 |-- installment: double (nullable = true)
 |-- grade: string (nullable = true)
 |-- sub_grade: string (nullable = true)
 |-- emp_title: string (nullable = true)
 |-- emp_length: string (nullable = true)
 |-- home_ownership: string (nullable = true)
 |-- annual_inc: double (nullable = true)
 |-- verification_status: string (nullable = true)
 |-- issue_d: string (nullable = true)
 |-- loan_status: string (nullable = true)
 |-- purpose: string (nullable = true)
 |-- title: string (nullable = true)
 |-- dti: double (nullable = true)
 |-- earliest_cr_line: string (nullable = true)
 |-- open_acc: double (nullable = true)
 |-- pub_rec: double (nullable = true)
 |-- revol_bal: double (nullable = true)
 |-- revol_util: double (nullable = true)
 |-- total_acc: double (nullable = true)
 |-- initial_list_status: string (nullable = true)
 |-- application_type: string (nullable = true)
 |-- mort_acc: double (nullable = true)
 |-- pub_rec_bankruptcies: double (nullable = true)
 |-- address: string (nullable = true)
```

In [ ]:
```python
df.limit(5).toPandas()
```

Out[ ]:

	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
0	10000.0	36 months	11.44	329.48	B	B4	Marketing	10+ years	RENT	117000.0	Not Verified	Jan-2015	Fully Paid	vacation	Vacation	26.24	Jun-
1	8000.0	36 months	11.99	265.68	B	B5	Credit analyst	4 years	MORTGAGE	65000.0	Not Verified	Jan-2015	Fully Paid	debt_consolidation	Debt consolidation	22.05	Jul-
2	15600.0	36 months	10.49	506.97	B	B3	Statistician	< 1 year	RENT	43057.0	Source Verified	Jan-2015	Fully Paid	credit_card	Credit card refinancing	12.79	Aug-
3	7200.0	36 months	6.49	220.65	A	A2	Client Advocate	6 years	RENT	54000.0	Not Verified	Nov-2014	Fully Paid	credit_card	Credit card refinancing	2.60	Sep-
4	24375.0	60 months	17.27	609.33	C	C5	Destiny Management Inc.	9 years	MORTGAGE	55000.0	Verified	Apr-2013	Charged Off	credit_card	Credit Card Refinance	33.95	Mar-

Data Cleaning

Null values

In [ ]:
```python
for col in df.columns:
    print(col," : ", df.filter(df[col].isNull()).count())
```

```
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
```

In [ ]:
```python
df.select("revol_util").summary().show();
```

```
[Stage 90:>                                                         (0 + 1) / 1]
```

```
+-------+------------------+
|summary|        revol_util|
+-------+------------------+
|  count|            395754|
|   mean| 53.79174863677853|
| stddev|24.452193062711693|
|    min|               0.0|
|    25%|              35.8|
|    50%|              54.8|
|    75%|              72.9|
|    max|             892.3|
+-------+------------------+
```

In [ ]:
```
df.select("mort_acc").summary().show();
```

```
+-------+------------------+
|summary|          mort_acc|
+-------+------------------+
|  count|            358235|
|   mean|1.8139908160844138|
| stddev|2.1479304671233352|
|    min|               0.0|
|    25%|               0.0|
|    50%|               1.0|
|    75%|               3.0|
|    max|              34.0|
+-------+------------------+
```

In [ ]:
```
df.groupby('home_ownership').count().sort("count", ascending=False).show()
```

```
+--------------+------+
|home_ownership| count|
+--------------+------+
|      MORTGAGE|198348|
|          RENT|159790|
|           OWN| 37746|
|         OTHER|   112|
|          NONE|    31|
|           ANY|     3|
+--------------+------+
```

Observations

- Majority of the user have mortgage type home ownership

In [ ]:
```
df = df.withColumn(
        "mort_acc",
        sf.when(
            (sf.col("home_ownership") == "MORTGAGE") & (sf.col("mort_acc").isNull()),
            1
        ).otherwise(sf.col("mort_acc"))
    )
df=df.fillna(0, subset=["mort_acc", "pub_rec_bankruptcies", "revol_util"])
df = df.drop("title")

df=df.withColumn("split_address", split_address_udf("address"))
```

In [ ]:
```
df.select("split_address").show(20, False)
```

```
[Stage 99:>                                                         (0 + 1) / 1]
+---------------------------------+
|split_address                    |
+---------------------------------+
|{Mendozaberg, OK, 22690}         |
|{Loganmouth, SD, 05113}          |
|{New Sabrina, WV, 05113}         |
|{Delacruzside, MA, 00813}        |
|{Greggshire, VA, 11650}          |
|{North Deniseberg, DE, 30723}    |
|{East Stephanie, TX, 22690}      |
|{FPO, AE, 30723}                 |
|{FPO, AP, 22690}                 |
|{Mauricestad, VA, 00813}         |
|{Bartlettfort, NM, 00813}        |
|{South Matthew, MS, 00813}       |
|{West Beckyfort, MS, 70466}      |
|{Shellychester, OR, 29597}       |
|{Lake Andrew, NH, 29597}         |
|{Stevenfort, HI, 30723}          |
|{West Aprilborough, PA, 00813}   |
|{Cummingsshire, NH, 30723}       |
|{Port Kirstenborough, CO, 70466}|
|{DPO, AE, 05113}                 |
+---------------------------------+
only showing top 20 rows
```

In [ ]:
```
df = df.withColumn("city", sf.col("split_address.city")) \
       .withColumn("state", sf.col("split_address.state")) \
       .withColumn("zipcode", sf.col("split_address.zipcode")) \
       .drop("split_address")
df=df.drop("address")
```

In [ ]:
```
df.groupBy('loan_status').count().show()
```

```
+-----------+------+
|loan_status| count|
+-----------+------+
| Fully Paid|318357|
|Charged Off| 77673|
+-----------+------+
```

In [ ]:
```
df.groupBy("term").count().show()
```

```
+----------+------+
|      term| count|
+----------+------+
| 36 months|302005|
| 60 months| 94025|
+----------+------+
```

In [ ]:
```
split_col = split(df['term'], " ", -1)
df= df.withColumn('term', split_col.getItem(1))
```

In [ ]:
```
df.groupBy('emp_title').count().sort('count', ascending=False).show()
```

```
+--------------------+-----+
|           emp_title|count|
+--------------------+-----+
|                NULL|22927|
|             Teacher| 4389|
|             Manager| 4250|
|    Registered Nurse| 1856|
|                  RN| 1846|
|          Supervisor| 1830|
|               Sales| 1638|
|     Project Manager| 1505|
|               Owner| 1410|
|              Driver| 1339|
|      Office Manager| 1218|
|             manager| 1145|
|            Director| 1089|
|     General Manager| 1074|
|            Engineer|  995|
|             teacher|  962|
|              driver|  882|
|      Vice President|  857|
|  Operations Manager|  763|
|Administrative As...|  756|
+--------------------+-----+
only showing top 20 rows
```

In [ ]:
```
df = df.withColumn("emp_title", lower(df["emp_title"]))
# replace Null values with 'unknown'
df = df.fillna('unknown', subset=['emp_title'])
```

In [ ]:
```
df.groupBy('emp_title').count().sort('count', ascending=False).show()
```

```
[Stage 109:>                                                        (0 + 1) / 1]
+------------------+-----+
|         emp_title|count|
+------------------+-----+
|           unknown|22927|
|           manager| 5637|
|           teacher| 5430|
|  registered nurse| 2627|
|        supervisor| 2591|
|             sales| 2382|
|            driver| 2306|
|             owner| 2201|
|                rn| 2074|
|   project manager| 1776|
|    office manager| 1638|
|   general manager| 1461|
|      truck driver| 1288|
|          director| 1192|
|          engineer| 1188|
|    police officer| 1041|
|    vice president|  962|
|operations manager|  961|
|     sales manager|  961|
|     store manager|  941|
+------------------+-----+
only showing top 20 rows
```

In [ ]:
```
df.groupBy('emp_length').count().sort('count', ascending=False).show()
```

```
+----------+------+
|emp_length| count|
+----------+------+
| 10+ years|126041|
|   2 years| 35827|
|  < 1 year| 31725|
|   3 years| 31665|
|   5 years| 26495|
|    1 year| 25882|
|   4 years| 23952|
|   6 years| 20841|
|   7 years| 20819|
|   8 years| 19168|
|      NULL| 18301|
|   9 years| 15314|
+----------+------+
```

In [ ]:
```
df = df.withColumn('emp_length', regexp_extract(df['emp_length'], r'(\d+)', 1))

# df.groupBy('grade').avg('emp_length').show()
avg_emp_length=df.groupBy("grade").agg(sf.avg("emp_length").cast("int").alias("avg_emp_length"))

df=df.join(avg_emp_length, "grade", )

df = df.withColumn(
        "emp_length",
        sf.when(
             sf.col("emp_length").isNull(),
            sf.col("avg_emp_length")
        ).otherwise(sf.col("emp_length"))
    )

df = df.withColumn("emp_length", df["emp_length"].cast(IntegerType()))
df=df.drop("avg_emp_length")
```

In [ ]:
```
df.groupBy("purpose").count().sort('count', ascending=False).show()
```

```
+------------------+------+
|           purpose| count|
+------------------+------+
|debt_consolidation|234507|
|       credit_card| 83019|
|  home_improvement| 24030|
|             other| 21185|
|    major_purchase|  8790|
|    small_business|  5701|
|               car|  4697|
|           medical|  4196|
|            moving|  2854|
|          vacation|  2452|
|             house|  2201|
|           wedding|  1812|
|  renewable_energy|   329|
|       educational|   257|
+------------------+------+
```

In [ ]:
```
df.groupBy("grade").count().sort('count', ascending=False).show()
```

```
+-----+------+
|grade| count|
+-----+------+
|    B|116018|
|    C|105987|
|    A| 64187|
|    D| 63524|
|    E| 31488|
|    F| 11772|
|    G|  3054|
+-----+------+
```

In [ ]:
```
df.groupBy('sub_grade').count().sort('count', ascending=False).show();
```

```
+---------+-----+
|sub_grade|count|
+---------+-----+
|       B3|26655|
|       B4|25601|
|       C1|23662|
|       C2|22580|
|       B2|22495|
|       B5|22085|
|       C3|21221|
|       C4|20280|
|       B1|19182|
|       A5|18526|
|       C5|18244|
|       D1|15993|
|       A4|15789|
|       D2|13951|
|       D3|12223|
|       D4|11657|
|       A3|10576|
|       A1| 9729|
|       D5| 9700|
|       A2| 9567|
+---------+-----+
only showing top 20 rows
```

In [ ]:
```
# convert issue_d havimg format of Jan-2015 to date
df = df.withColumn("issue_d", sf.to_date(df["issue_d"], "MMM-yyyy"))
df = df.withColumn("earliest_cr_line", sf.to_date(df["issue_d"], "MMM-yyyy"))
# extract year from issue_d
df = df.withColumn("issue_year", sf.year(df["issue_d"]))
df = df.withColumn("earliest_cr_line_year", sf.year(df["earliest_cr_line"]))
# extract month in form of integer from issue_d
df = df.withColumn("issue_month", sf.month(df["issue_d"]))
df = df.withColumn("earliest_cr_line_month", sf.month(df["earliest_cr_line"]))
```

In [ ]:
```
# # convert multiple colums to int
# df = df.withColumn("earliest_cr_line_year", df["issue_month"].cast(IntegerType()))
# df = df.withColumn("issue_year", df["issue_year"].cast(IntegerType()))
```

In [ ]:
```
to_drop.extend(['issue_d', 'earliest_cr_line'])
```

In [ ]:
```
# drop duplicates
df = df.dropDuplicates()
```

In [ ]:
```
float_cols = [col for col in df.columns if isinstance(df.schema[col].dataType, (DoubleType, FloatType))]
float_cols
```

Out[ ]:
```
['loan_amnt',
 'int_rate',
 'installment',
 'annual_inc',
 'dti',
 'open_acc',
 'pub_rec',
 'revol_bal',
 'revol_util',
 'total_acc',
 'mort_acc',
 'pub_rec_bankruptcies']
```

In [ ]:
```
# convert to int
df=df.withColumn("annual_inc", df["annual_inc"].cast(IntegerType()))
df=df.withColumn("open_acc", df["open_acc"].cast(IntegerType()))
df=df.withColumn("pub_rec", df["pub_rec"].cast(IntegerType()))
df=df.withColumn("total_acc", df["total_acc"].cast(IntegerType()))
df=df.withColumn("mort_acc", df["mort_acc"].cast(IntegerType()))
df=df.withColumn("pub_rec", df["pub_rec"].cast(IntegerType()))
df=df.withColumn("pub_rec_bankruptcies", df["pub_rec_bankruptcies"].cast(IntegerType()))
df=df.withColumn("revol_bal", df["revol_bal"].cast(IntegerType()))
```

In [ ]:
```
for col in df.columns:
    print(col," : ", df.filter(df[col].isNull()).count())
```

```
grade  :  0
loan_amnt  :  0
term  :  0
int_rate  :  0
installment  :  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
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

city  :  0

state  :  0

zipcode  :  0
issue_year  :  0

earliest_cr_line_year  :  0
issue_month  :  0
earliest_cr_line_month  :  0
```

In [ ]:
```
# save to csv
df.write.csv("logistic_regression_cleaned.csv", header=True, mode='overwrite')
# save as parquet
df.write.parquet("logistic_regression_cleaned.parquet", mode='overwrite')
```

EDA

In [ ]:
```
df = spark.read \
    .option("header", "true") \
    .option("inferSchema", "true") \
    .option("multiLine", "true") \
    .option("escape", "\"") \
    .csv("./logistic_regression_cleaned.csv")
df.cache();
```

```
24/07/31 18:50:43 WARN CacheManager: Asked to cache already cached data.
```

In [ ]:
```
df.createOrReplaceTempView("data")
pdf = df.toPandas();
```

In [ ]:
```
pdf.head().T
```

Out[ ]:

	0	1	2	3	4
grade	B	B	B	A	C
loan_amnt	10000.0	8000.0	15600.0	7200.0	24375.0
term	36	36	36	36	60
int_rate	11.44	11.99	10.49	6.49	17.27
installment	329.48	265.68	506.97	220.65	609.33
sub_grade	B4	B5	B3	A2	C5
emp_title	marketing	credit analyst	statistician	client advocate	destiny management inc.
emp_length	10	4	1	6	9
home_ownership	RENT	MORTGAGE	RENT	RENT	MORTGAGE
annual_inc	117000	65000	43057	54000	55000
verification_status	Not Verified	Not Verified	Source Verified	Not Verified	Verified
issue_d	2015-01-01	2015-01-01	2015-01-01	2014-11-01	2013-04-01
loan_status	Fully Paid	Fully Paid	Fully Paid	Fully Paid	Charged Off
purpose	vacation	debt_consolidation	credit_card	credit_card	credit_card
dti	26.24	22.05	12.79	2.6	33.95
earliest_cr_line	2015-01-01	2015-01-01	2015-01-01	2014-11-01	2013-04-01
open_acc	16	17	13	6	13
pub_rec	0	0	0	0	0
revol_bal	36369	20131	11987	5472	24584
revol_util	41.8	53.3	92.2	21.5	69.8
total_acc	25	27	26	13	43
initial_list_status	w	f	f	f	f
application_type	INDIVIDUAL	INDIVIDUAL	INDIVIDUAL	INDIVIDUAL	INDIVIDUAL
mort_acc	0	3	0	0	1
pub_rec_bankruptcies	0	0	0	0	0
city	Mendozaberg	Loganmouth	New Sabrina	Delacruzside	Greggshire
state	OK	SD	WV	MA	VA
zipcode	22690	5113	5113	813	11650
issue_year	2015	2015	2015	2014	2013
earliest_cr_line_year	2015	2015	2015	2014	2013
issue_month	1	1	1	11	4
earliest_cr_line_month	1	1	1	11	4

In [ ]:
```
pdf.describe().T
```

Out[ ]:

	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
term	396030.0	41.698053	10.212038	36.00	36.00	36.00	36.00	60.00
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
emp_length	396030.0	6.006532	3.437114	1.00	3.00	6.00	10.00	10.00
annual_inc	396030.0	74203.170926	61637.622333	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	396030.0	53.754260	24.484857	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	396030.0	1.682895	2.087995	0.00	0.00	1.00	3.00	34.00
pub_rec_bankruptcies	396030.0	0.121483	0.355962	0.00	0.00	0.00	0.00	8.00
zipcode	396030.0	33998.447686	25605.865779	813.00	11650.00	29597.00	48052.00	93700.00
issue_year	396030.0	2013.629074	1.481725	2007.00	2013.00	2014.00	2015.00	2016.00
earliest_cr_line_year	396030.0	2013.629074	1.481725	2007.00	2013.00	2014.00	2015.00	2016.00
issue_month	396030.0	6.553188	3.426622	1.00	4.00	7.00	10.00	12.00
earliest_cr_line_month	396030.0	6.553188	3.426622	1.00	4.00	7.00	10.00	12.00

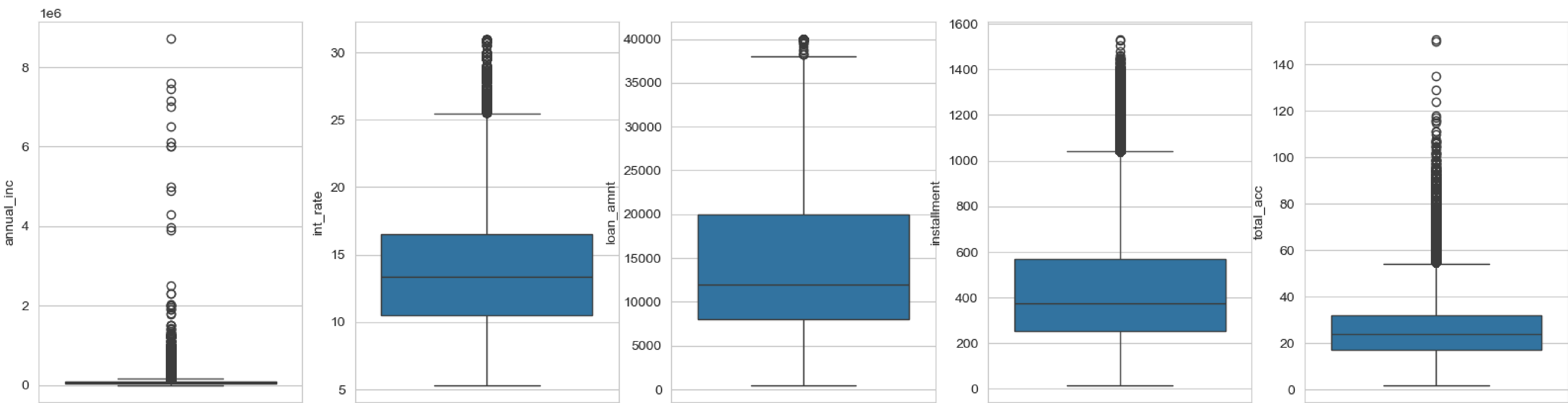
In [ ]:
```
pdf.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 396030 entries, 0 to 396029
Data columns (total 32 columns):
 #   Column                Non-Null Count   Dtype  
---  ------                --------------   -----  
 0   grade                 396030 non-null  object 
 1   loan_amnt             396030 non-null  float64
 2   term                  396030 non-null  int32  
 3   int_rate              396030 non-null  float64
 4   installment           396030 non-null  float64
 5   sub_grade             396030 non-null  object 
 6   emp_title             396030 non-null  object 
 7   emp_length            396030 non-null  int32  
 8   home_ownership        396030 non-null  object 
 9   annual_inc            396030 non-null  int32  
 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  dti                   396030 non-null  float64
 15  earliest_cr_line      396030 non-null  object 
 16  open_acc              396030 non-null  int32  
 17  pub_rec               396030 non-null  int32  
 18  revol_bal             396030 non-null  int32  
 19  revol_util            396030 non-null  float64
 20  total_acc             396030 non-null  int32  
 21  initial_list_status   396030 non-null  object 
 22  application_type      396030 non-null  object 
 23  mort_acc              396030 non-null  int32  
 24  pub_rec_bankruptcies  396030 non-null  int32  
 25  city                  396030 non-null  object 
 26  state                 396030 non-null  object 
 27  zipcode               396030 non-null  int32  
 28  issue_year            396030 non-null  int32  
 29  earliest_cr_line_year 396030 non-null  int32  
 30  issue_month           396030 non-null  int32  
 31  earliest_cr_line_month 396030 non-null  int32  
dtypes: float64(5), int32(14), object(13)
memory usage: 75.5+ MB
```

In [ ]:
```python
cat_cols = ['term', 'grade', 'emp_title', 'emp_length', 'home_ownership', 'verification_status', 'application_type', 'purpose',  'city', 'state', 'zipcode', 'issue_month', 'issue_year', '
int_colums = pdf.select_dtypes(include=['int64', 'int32', "float64"]).columns
```

Outlier Check

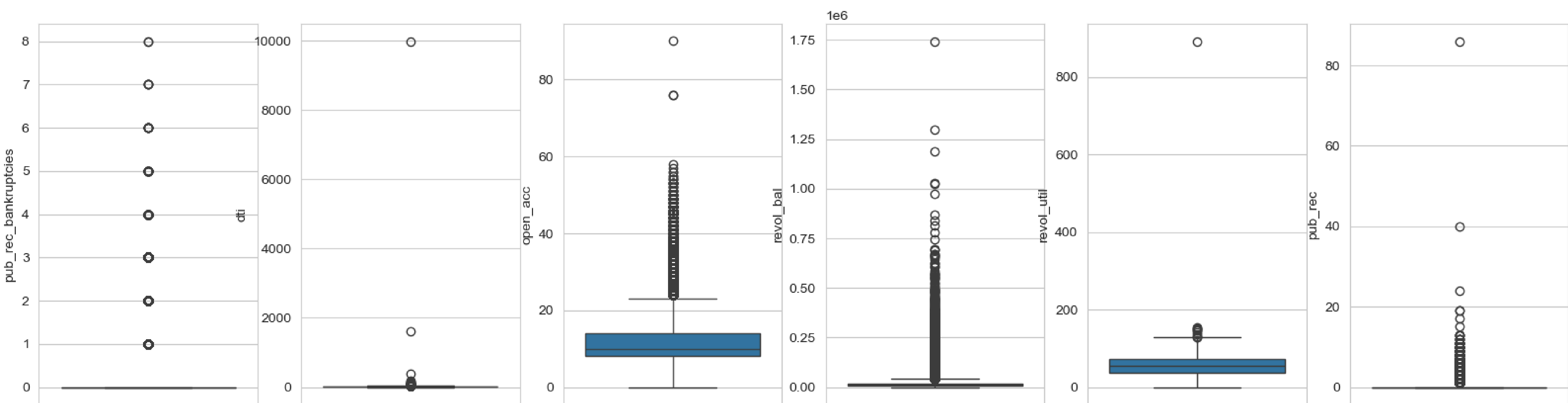
In [ ]:
```python
fig, ax = plt.subplots(1, 5, figsize=(20, 5))
sns.boxplot(y='annual_inc', data=pdf, ax=ax[0]);
sns.boxplot(y='int_rate', data=pdf, ax=ax[1]);
sns.boxplot(y='loan_amnt', data=pdf, ax=ax[2]);
sns.boxplot(y='installment', data=pdf, ax=ax[3]);
sns.boxplot(y='total_acc', data=pdf, ax=ax[4]);
```



Observations

- Annual Income has large number of outliers. This needs to be handled while preparing model

In [ ]:
```python
c=['pub_rec_bankruptcies', 'dti', 'open_acc', 'revol_bal', 'revol_util', 'pub_rec']
fig, ax = plt.subplots(1, 6, figsize=(20, 5));

for i in range(6):
    sns.boxplot(y=c[i], data=pdf, ax=ax[i]);
plt.show();
```

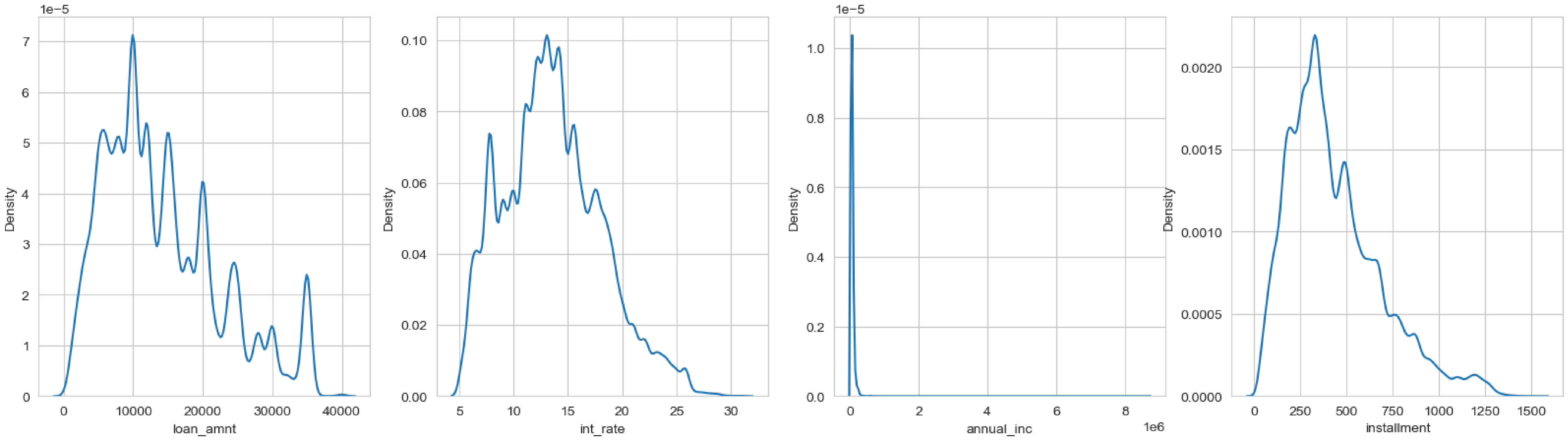


Observations

- Revolve balance have higher number of outliers

Normality Check

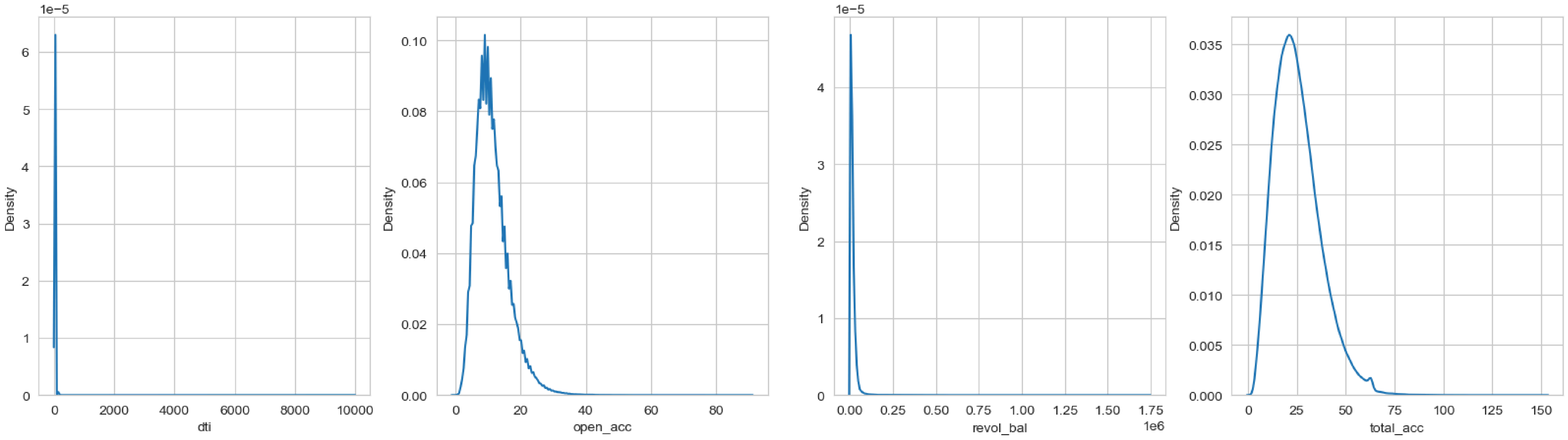
In [ ]:
```python
fig, ax = plt.subplots(1, 4, figsize=(20, 5))
sns.kdeplot(x='loan_amnt', data=pdf, ax=ax[0]);
sns.kdeplot(x='int_rate', data=pdf, ax=ax[1]);
sns.kdeplot(x='annual_inc', data=pdf, ax=ax[2]);
sns.kdeplot(x='installment', data=pdf, ax=ax[3]);
```



Observations

- Above plots show that most of features have right skewed distribution

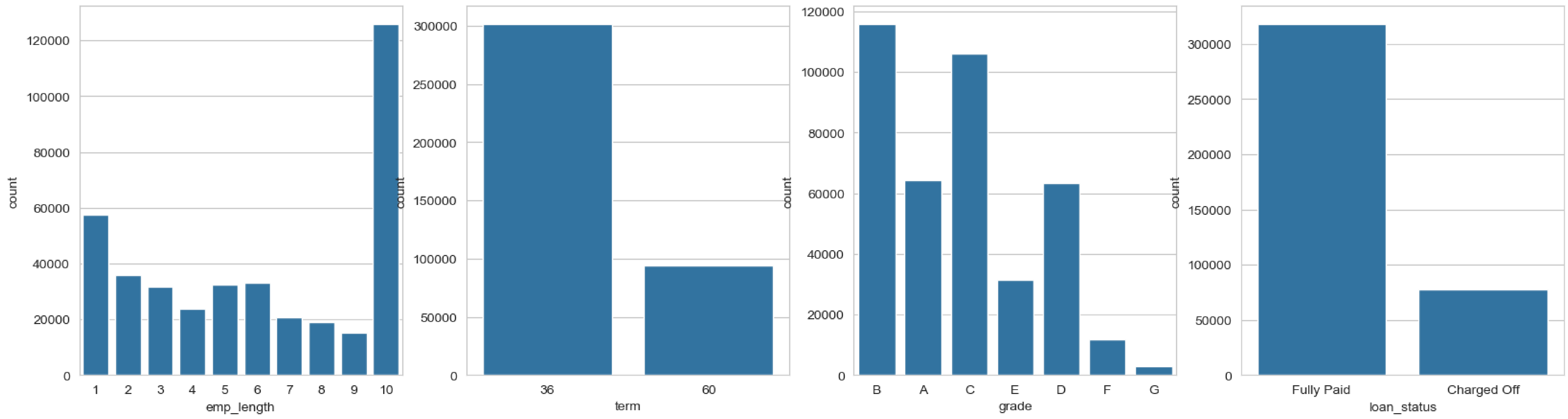
In [ ]:
```python
fig, ax = plt.subplots(1, 4, figsize=(20, 5))
sns.kdeplot(x='dti', data=pdf, ax=ax[0]);
sns.kdeplot(x='open_acc', data=pdf, ax=ax[1]);
sns.kdeplot(x='revol_bal', data=pdf, ax=ax[2]);
sns.kdeplot(x='total_acc', data=pdf, ax=ax[3]);
```



Observations

We can see that annual income, revol_balance and dti is highly right skewed.

In [ ]:
```python
fig, ax = plt.subplots(1, 4, figsize=(20, 5))
sns.countplot(x='emp_length', data=pdf, ax=ax[0]);
sns.countplot(x='term', data=pdf, ax=ax[1]);
sns.countplot(x='grade', data=pdf, ax=ax[2]);
sns.countplot(x='loan_status', data=pdf, ax=ax[3]);
```

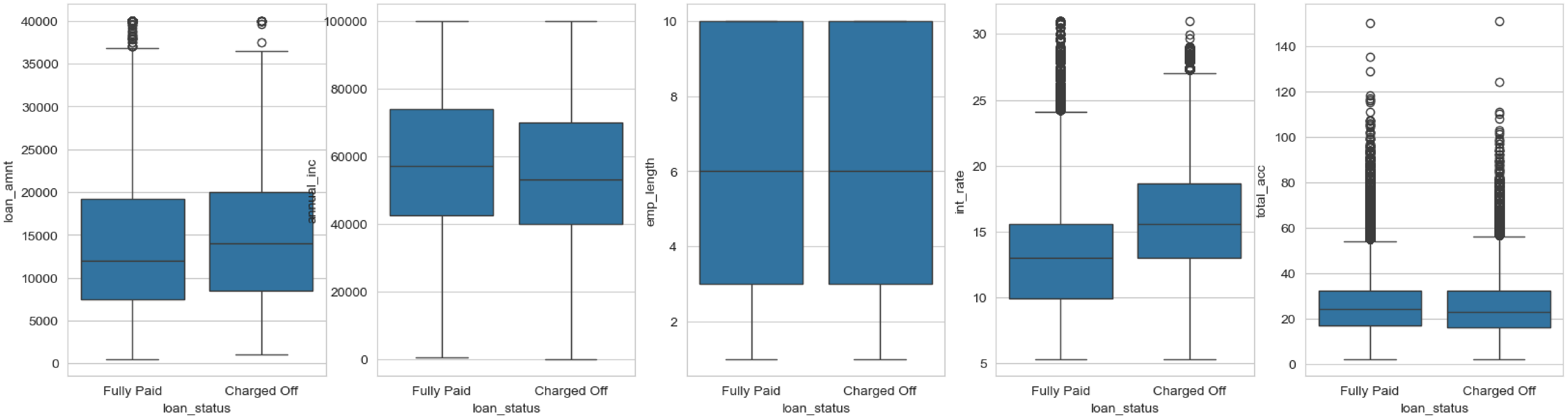


Observations

- From above plot we can see that most loans are taken by users who are younger and decreases as experience increases
- Most of the loans are taken for 36 months
- Most of the loans are taken by B grade users

Bivariate Analysis

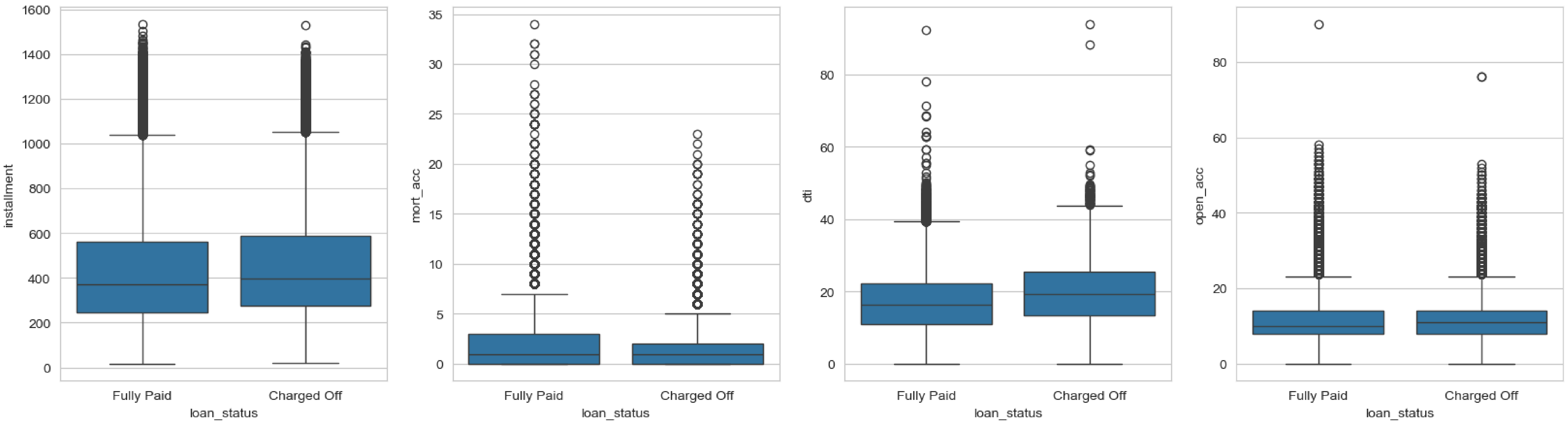
In [ ]:
```python
fig, ax = plt.subplots(1, 5, figsize=(20, 5))
sns.boxplot(x='loan_status', y='loan_amnt', data=pdf, ax=ax[0]);
sns.boxplot(x='loan_status', y='annual_inc', data=pdf[pdf["annual_inc"]<100000], ax=ax[1]); #adjusting for outliers
sns.boxplot(x='loan_status', y='emp_length', data=pdf, ax=ax[2]);
sns.boxplot(x='loan_status', y='int_rate', data=pdf, ax=ax[3]);
sns.boxplot(x='loan_status', y='total_acc', data=pdf, ax=ax[4]);
```



Observations

- From above plot we can see that int_rate can be a useful feature when differentiating users based on loan status

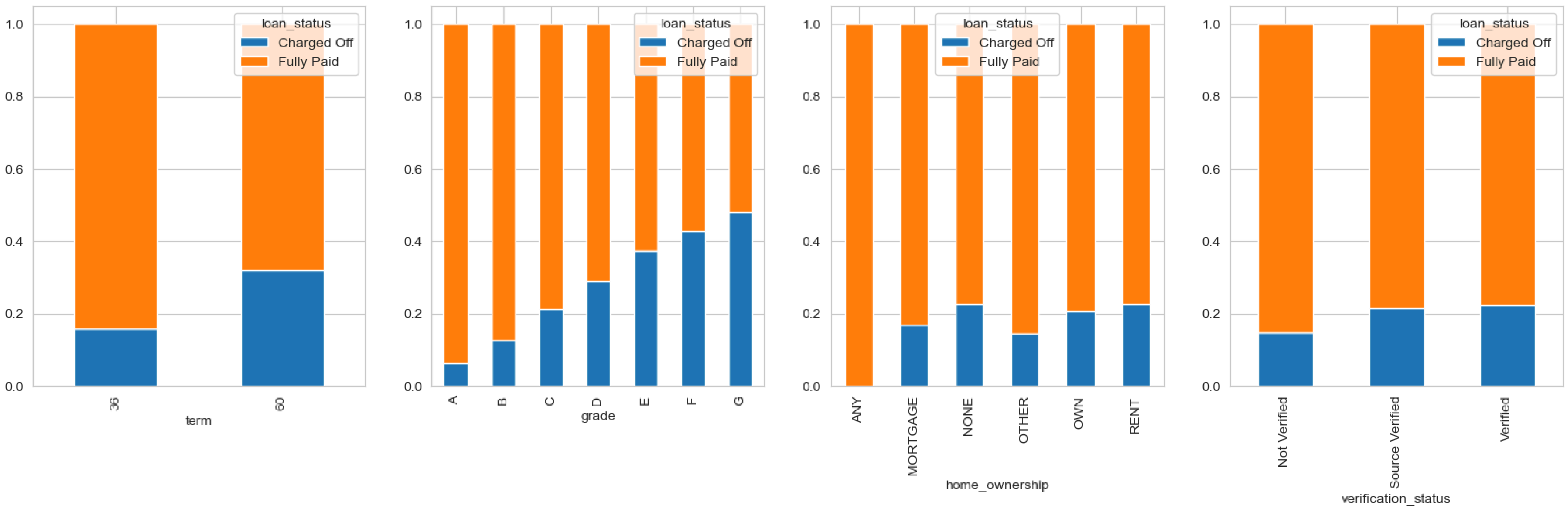
In [ ]:
```
fig, ax = plt.subplots(1, 4, figsize=(20, 5))
sns.boxplot(x='loan_status', y='installment', data=pdf, ax=ax[0]);
sns.boxplot(x='loan_status', y='mort_acc', data=pdf, ax=ax[1]);
sns.boxplot(x='loan_status', y='dti', data=pdf[pdf["dti"]<100], ax=ax[2]);
sns.boxplot(x='loan_status', y='open_acc', data=pdf, ax=ax[3]);
```



Observations

- From above plot we can see that there is not much relation between defaulters and Paid users with above features

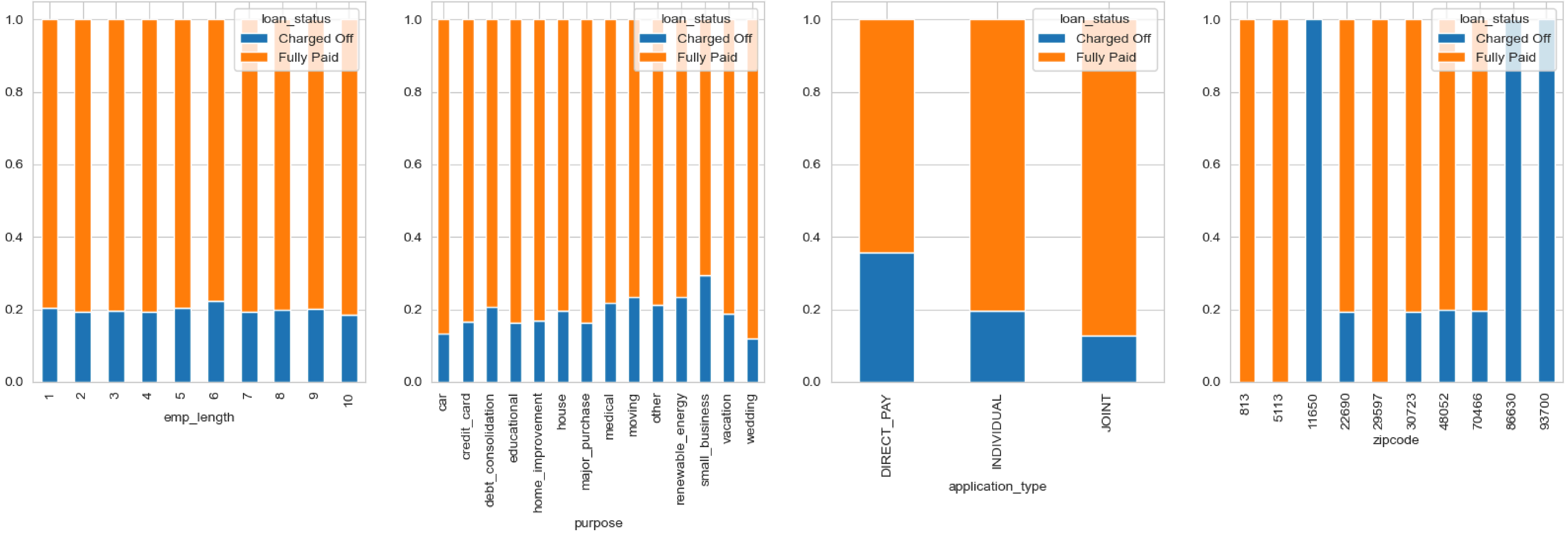
In [ ]:
```
fig, ax = plt.subplots(1, 4, figsize=(20, 5))
pd.crosstab(pdf['term'], pdf['loan_status'],normalize="index").plot(kind='bar', stacked=True, ax=ax[0]);
pd.crosstab(pdf['grade'], pdf['loan_status'], normalize="index").plot(kind='bar', stacked=True, ax=ax[1]);
pd.crosstab(pdf['home_ownership'], pdf['loan_status'], normalize="index").plot(kind='bar', stacked=True, ax=ax[2]);
pd.crosstab(pdf['verification_status'], pdf['loan_status'], normalize="index").plot(kind='bar', stacked=True, ax=ax[3]);
```



Observations

- We can see that more loans with 60 month repayment terms are "Charged off"
- From above plot we can see that Grade A users are those users who have most number of paid loans as compared to Grade G

In [ ]:
```
fig, ax = plt.subplots(1, 4, figsize=(20, 5))
pd.crosstab(pdf['emp_length'], pdf['loan_status'], normalize="index").plot(kind='bar', stacked=True, ax=ax[0]);
pd.crosstab(pdf['purpose'], pdf['loan_status'], normalize="index").plot(kind='bar', stacked=True, ax=ax[1]);
pd.crosstab(pdf['application_type'], pdf['loan_status'], normalize="index").plot(kind='bar', stacked=True, ax=ax[2]);
pd.crosstab(pdf['zipcode'], pdf['loan_status'], normalize="index").plot(kind='bar', stacked=True , ax=ax[3]);
```

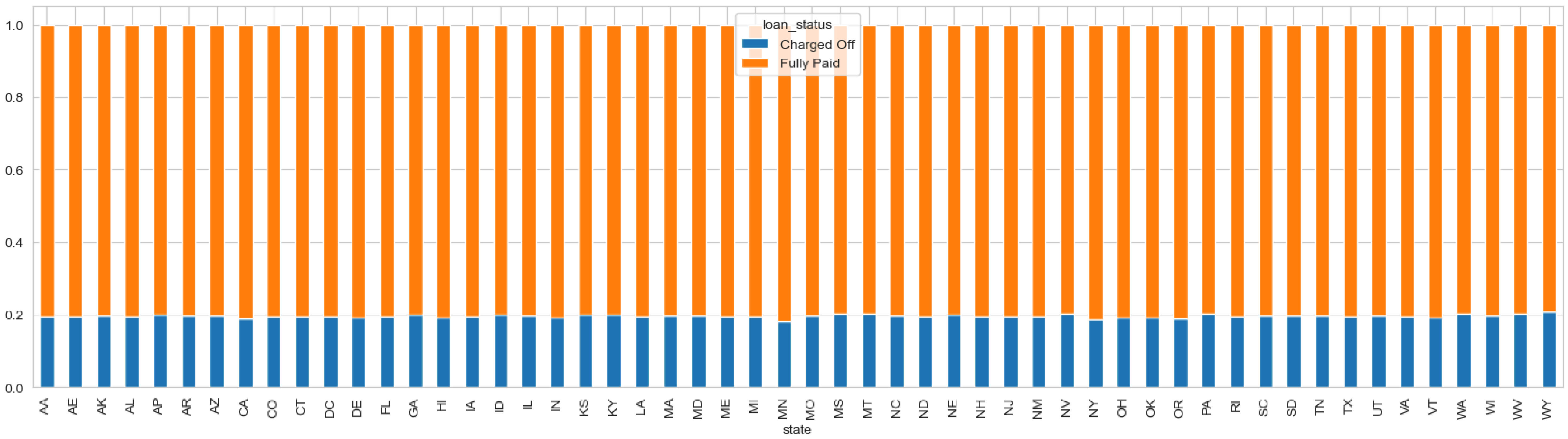


Observations

- We can see that there is equal proportion of loan paid off and charged off for employee experience
- We can see small business have highest percentage of "Charged Off" loans
- Above graph shows that zipcode can be an important feature in separating users on basis of loan status.
- Direct pay application type has higher number of defaulters as compared to joint application type

In [ ]:
```
pd.crosstab(pdf['state'], pdf['loan_status'], normalize="index").plot(kind='bar', stacked=True, figsize=(20, 5));
```

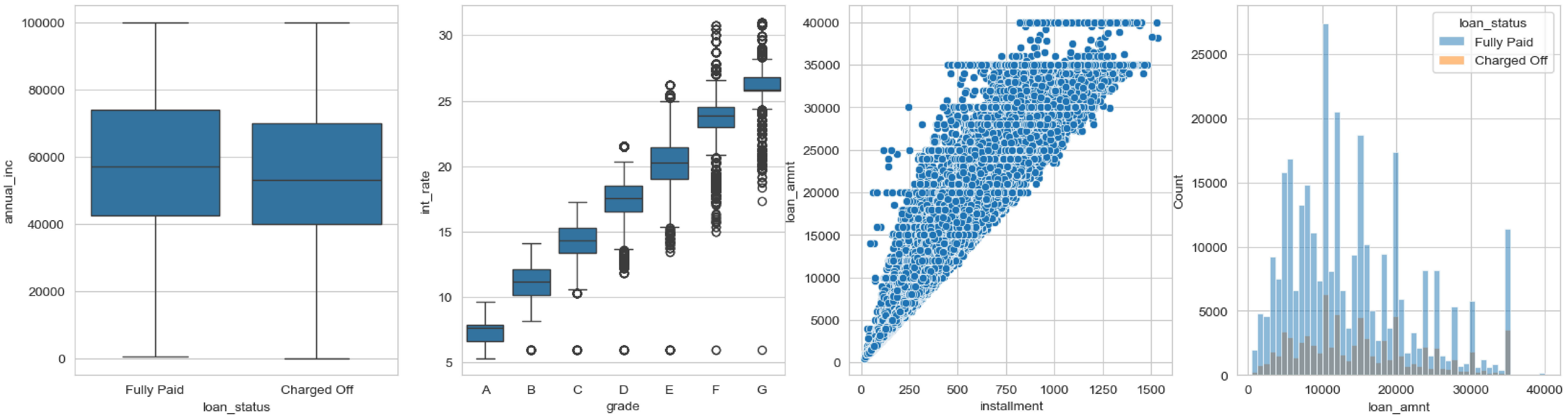
<Figure size 2000x500 with 0 Axes>



Observations

- Above plots show that all states have equal proportion of paid and charged off loans

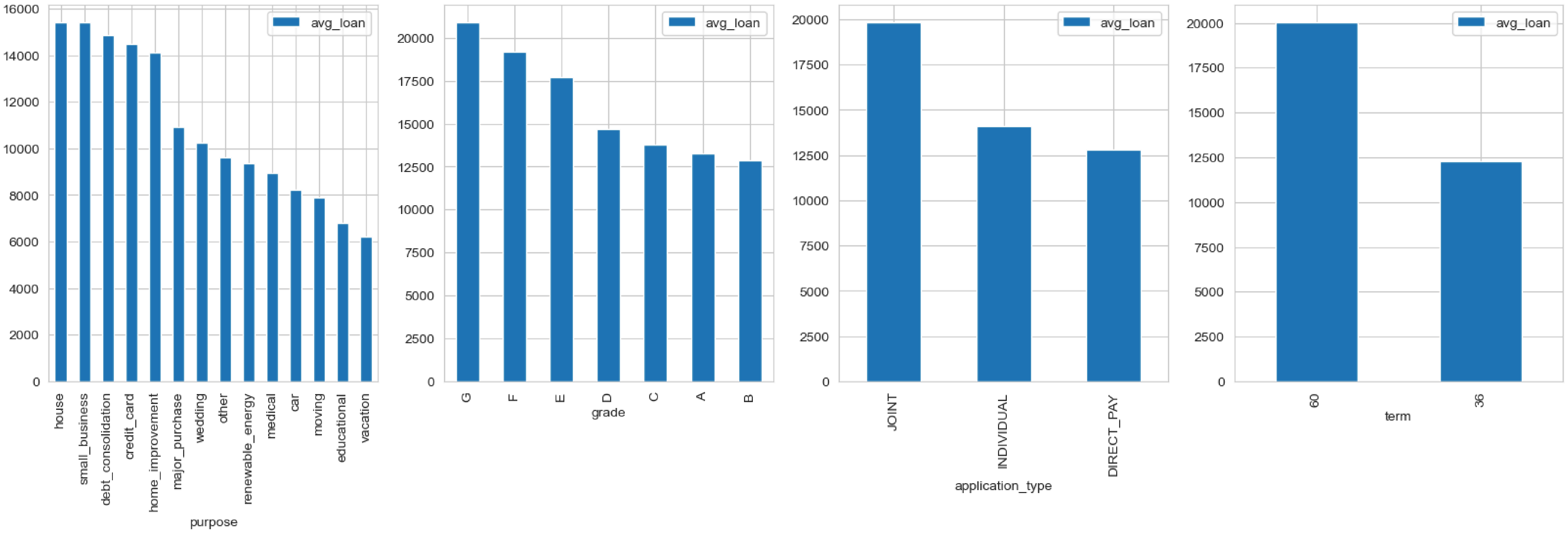
In [ ]:
```
fig, ax = plt.subplots(1, 4, figsize=(20, 5))
sns.boxplot(x='loan_status', y='annual_inc', data=pdf[pdf["annual_inc"]<100000], ax=ax[0]); #adjusting for outliers
sns.boxplot(x='grade', y='int_rate', data=pdf, ax=ax[1], order=sorted(pdf['grade'].unique()));
sns.scatterplot(x='installment', y='loan_amnt', data=pdf, ax=ax[2]);
sns.histplot(x='loan_amnt', data=pdf, bins=50, hue='loan_status', ax=ax[3]);
```



Observations

- From above plot we can see that annual income is not important feature for loan status
- The loan amount is directly proportional to installment
- We can see that interest rate is directly proportinal to grade

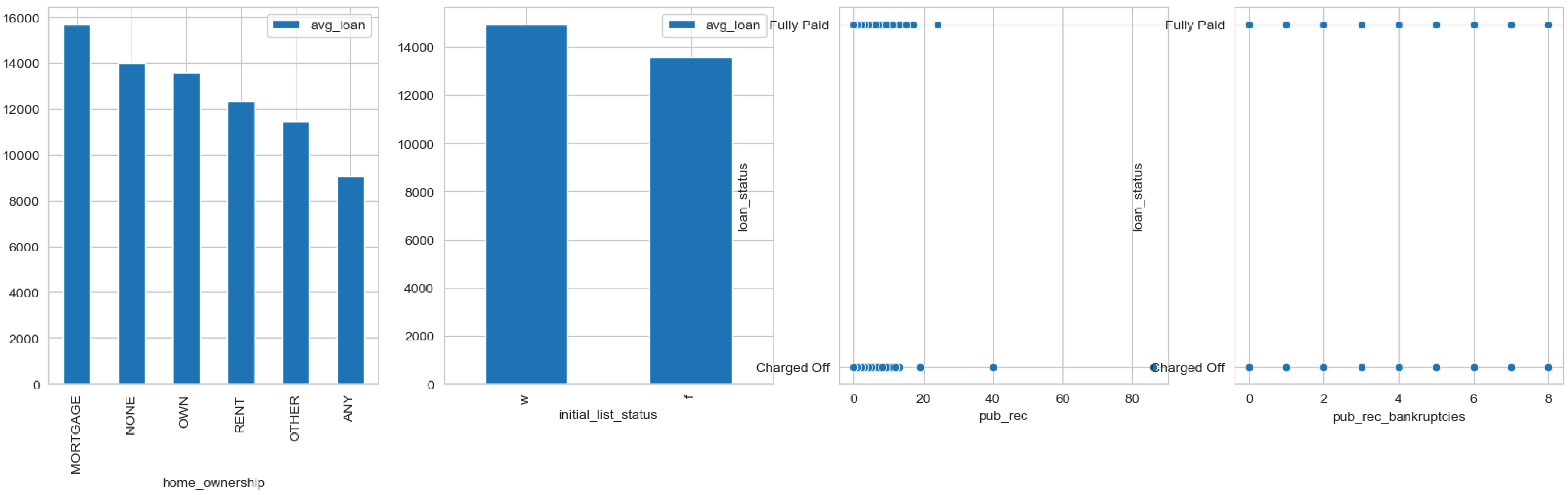
In [ ]:
```
fig , ax = plt.subplots(1, 4, figsize=(20, 5))
spark.sql("""
select round(avg(loan_amnt),2) avg_loan, purpose  from data group by purpose order by avg_loan desc
""").toPandas().plot(kind='bar', x='purpose', y='avg_loan', ax=ax[0]);
spark.sql("""
select round(avg(loan_amnt),2) avg_loan, grade  from data group by grade order by avg_loan desc
""").toPandas().plot(kind='bar', x='grade', y='avg_loan', ax=ax[1]);

spark.sql("""
select round(avg(loan_amnt),2) avg_loan, application_type  from data group by application_type order by avg_loan desc
""").toPandas().plot(kind='bar', x='application_type', y='avg_loan', ax=ax[2]);
spark.sql("""
select round(avg(loan_amnt),2) avg_loan, term  from data group by term order by avg_loan desc
""").toPandas().plot(kind='bar', x='term', y='avg_loan', ax=ax[3]);
```



Observations

- Above plots show average loan amount with different features
- We can see that average loan amount for users is highest for houses and lowest for vacations
- The users with Joint application have higher loan amount
- G graded employees take the highest loan amount while B takes lowest

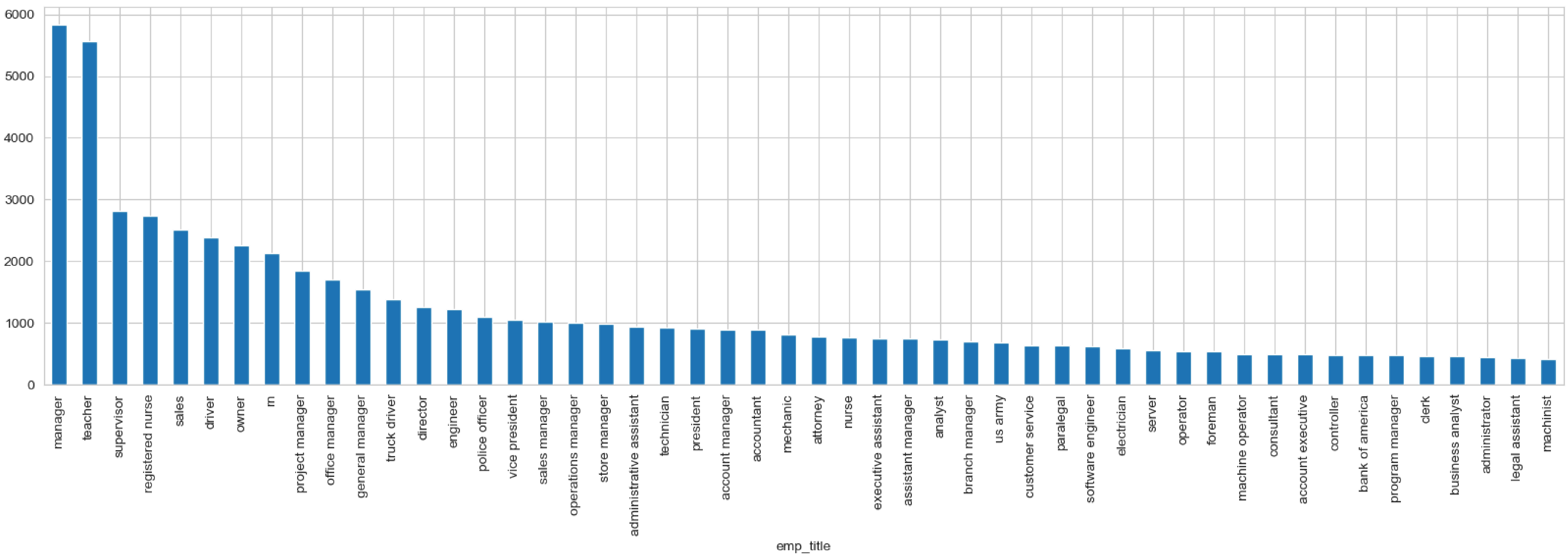
In [ ]:
```
fig , ax = plt.subplots(1, 4, figsize=(20, 5))
spark.sql("""
select round(avg(loan_amnt),2) avg_loan, home_ownership  from data group by home_ownership order by avg_loan desc
""").toPandas().plot(kind='bar', x='home_ownership', y='avg_loan', ax=ax[0]);
spark.sql("""
select round(avg(loan_amnt),2) avg_loan, initial_list_status  from data group by initial_list_status order by avg_loan desc
""").toPandas().plot(kind='bar', x='initial_list_status', y='avg_loan', ax=ax[1]);
sns.scatterplot(x="pub_rec", y="loan_status", data=pdf, ax=ax[2]);
sns.scatterplot(x="pub_rec_bankruptcies", y="loan_status", data=pdf, ax=ax[3]);
```



Observations

- above plots shows the average loan amount for home ownership and initial list status

In [ ]:
```
pdf["emp_title"].value_counts().head(51).iloc[1:].plot(kind='bar', figsize=(20, 5));
```

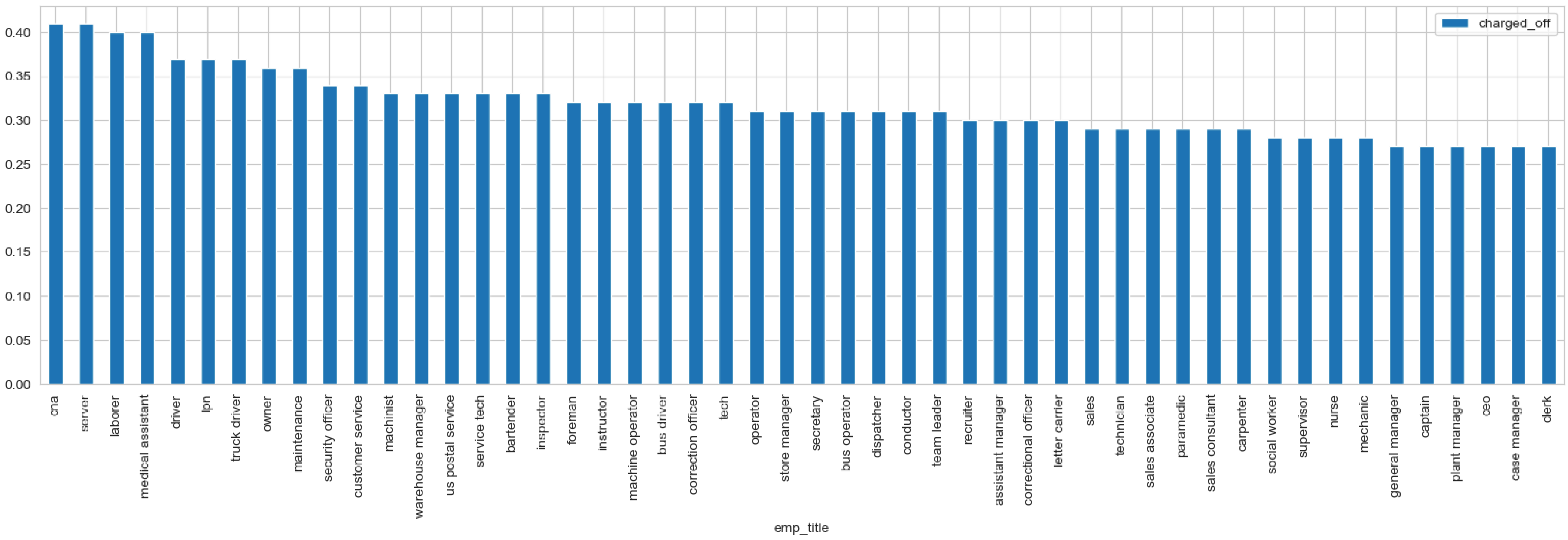


Observations

- Above plot shows top 50 employee titles with highest number of loans
- Above plot shows that employee with Manager and Teacher title, has the highest number of loans

In [ ]:
```
spark.sql("""
select
    emp_title,
    -- sum(case when loan_status = 'Fully Paid' then 1 else 0 end)/count(*) as fully_paid,
   round(sum(case when loan_status = 'Charged Off' then 1 else 0 end)/count(*),2) as charged_off
from data
    where emp_title != 'unknown' and loan_amnt > 10000
    group by emp_title
    having count(*) > 100
```

```
    order by charged_off desc
""").toPandas().head(51).iloc[1:].plot(kind='bar', x='emp_title', y='charged_off', figsize=(20, 5));
```



Observations

Above is a list of employee titles who have higher ratio of charged off loans and have taken more than 100 loans above 10000.

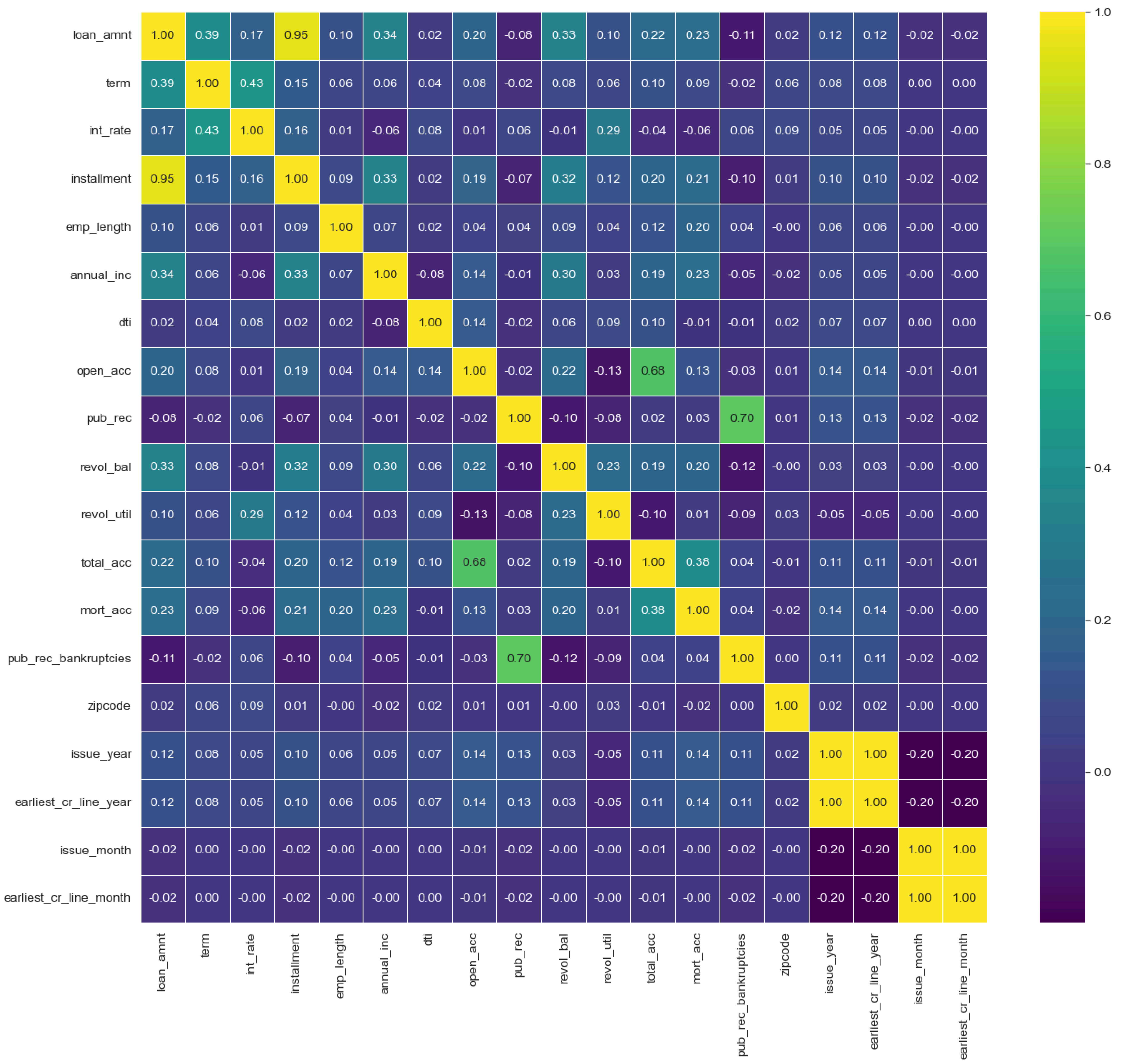
In [ ]:
```
pdf["loan_status"].value_counts(normalize=True)
```

Out[ ]:
```
loan_status
Fully Paid     0.803871
Charged Off    0.196129
Name: proportion, dtype: float64
```

Observations

We can see that 19% users did not pay loans and 20% paid loans

In [ ]:
```
cols=pdf.select_dtypes(include=['int64', 'int32', "float64"]).columns
corr = pdf[cols].corr()
plt.figure(figsize=(15, 13))
sns.heatmap(corr, annot=True, fmt=".2f", linewidths=0.5, cmap='viridis');
```



Observations

- Above plot shows correlation between different columns. We can see that loan_amnt and installment are highly correlated. Similarly, pub_rec and pub_rec_bankruptcies are highly correlated.

Model Building

In [ ]:
```python
df_model = spark.read.parquet("logistic_regression_cleaned.parquet")
df_model.cache();
# https://saturncloud.io/blog/feature-selection-in-pyspark-a-comprehensive-guide-for-data-scientists/
```

In [ ]:
```python
df_model.limit(5).toPandas()
```

Out[ ]:

	grade	loan_amnt	term	int_rate	installment	sub_grade	emp_title	emp_length	home_ownership	annual_inc	verification_status	issue_d	loan_status	purpose	dti	earliest_cr_line	open_acc
0	B	10000.0	36	11.44	329.48	B4	marketing	10	RENT	117000	Not Verified	2015-01-01	Fully Paid	vacation	26.24	2015-01-01	16
1	B	8000.0	36	11.99	265.68	B5	credit analyst	4	MORTGAGE	65000	Not Verified	2015-01-01	Fully Paid	debt_consolidation	22.05	2015-01-01	17
2	B	15600.0	36	10.49	506.97	B3	statistician	1	RENT	43057	Source Verified	2015-01-01	Fully Paid	credit_card	12.79	2015-01-01	13
3	A	7200.0	36	6.49	220.65	A2	client advocate	6	RENT	54000	Not Verified	2014-11-01	Fully Paid	credit_card	2.60	2014-11-01	6
4	C	24375.0	60	17.27	609.33	C5	destiny management inc.	9	MORTGAGE	55000	Verified	2013-04-01	Charged Off	credit_card	33.95	2013-04-01	13

Outlier Treatment

In [ ]:
```python
df_model = df_model.filter("dti < 50")
df_model = df_model.withColumn("revol_bal", sf.power(df_model["revol_bal"], 1/3))
df_model = df_model.withColumn("annual_inc", sf.power(df_model["annual_inc"], 1/3))
# df_model = df_model.withColumn("total_acc", sf.log(df_model["total_acc"]))
df_model = df_model.filter("revol_util < 200")
df_model = df_model.withColumn("loan_amnt", sf.log(df_model["loan_amnt"]))
# df_model = df_model.withColumn("installment", sf.log(df_model["installment"]))
# df_model = df_model.withColumn("open_acc", sf.log1p(df_model["open_acc"]))
```

Drop Columns

In [ ]:
```python
drop_cols=['issue_d', 'emp_title',  'earliest_cr_line', 'emp_title', 'grade','city' ]
df_model = df_model.drop(*drop_cols)
```

Encodings

In [ ]:
```python
sub_grade = df_model.select("sub_grade").distinct().sort("sub_grade")
indexer = StringIndexer(inputCol="sub_grade", outputCol="sub_grade_index")
indexer_model = indexer.fit(sub_grade)
df_encoded = indexer_model.transform(sub_grade)
df_model = indexer_model.transform(df_model)
df_model=df_model.drop("sub_grade")
df_model = df_model.withColumn("sub_grade", df_model["sub_grade_index"].cast(IntegerType()))
df_model = df_model.drop("sub_grade_index")
```

In [ ]:
```python
df_model = df_model.withColumn("loan_status", sf.when(df_model["loan_status"] == "Fully Paid", 0).otherwise(1).cast(IntegerType()))
df_model = df_model.withColumn("term", sf.when(df_model["term"] == 36, 0).otherwise(1).cast(IntegerType()))
# df_model = df_model.withColumn("application_type", sf.when(df_model["application_type"] == "INDIVIDUAL", 0).otherwise(1).cast(IntegerType()))
df_model = df_model.withColumn("initial_list_status", sf.when(df_model["initial_list_status"] == "w", 0).otherwise(1).cast(IntegerType()))
```

In [ ]:
```python
def target_mean_encoding(df, col, target):
    """
    :param df: pyspark.sql.dataframe
        dataframe to apply target mean encoding
    :param col: str list
        list of columns to apply target encoding
    :param target: str
        target column
    :return:
        dataframe with target encoded columns
    """
    target_encoded_columns_list = []
    for c in col:
        means = df.groupby(sf.col(c)).agg(sf.mean(target).alias(f"{c}_mean_encoding"))
        dict_ = means.toPandas().to_dict()
        target_encoded_columns = [sf.when(sf.col(c) == v, encoder)
                                  for v, encoder in zip(dict_[c].values(),dict_[f"{c}_mean_encoding"].values())]
        target_encoded_columns_list.append(sf.coalesce(*target_encoded_columns).alias(f"{c}_mean_encoding"))
    return df.select(*col, *target_encoded_columns_list).distinct()
```

In [ ]:
```python
df_target_encoded = target_mean_encoding(df_model, col=['state'], target='loan_status')
df_model = df_model.join(df_target_encoded, "state")
df_model = df_model.drop("state")
df_model = df_model.withColumnRenamed("state_mean_encoding", "state")
```

In [ ]:
```python
df_model = df_model.withColumn("issue_month_sin", sf.sin((df_model.issue_month-1)*(2.*np.pi/12)))
df_model = df_model.withColumn("issue_month_cos", sf.cos((df_model.issue_month-1)*(2.*np.pi/12)))
df_model = df_model.withColumn("earliest_cr_line_month_sin", sf.sin((df_model.earliest_cr_line_month-1)*(2.*np.pi/12)))
df_model = df_model.withColumn("earliest_cr_line_month_cos", sf.cos((df_model.earliest_cr_line_month-1)*(2.*np.pi/12)))
df_model = df_model.drop("issue_month", "earliest_cr_line_month")
```

In [ ]:
```python
df_model.limit(5).toPandas()
```

Out[ ]:

	loan_amnt	term	int_rate	installment	emp_length	home_ownership	annual_inc	verification_status	loan_status	purpose	dti	open_acc	pub_rec	revol_bal	revol_util	total_acc	initial_list_stat
0	9.210340	0	11.44	329.48	10	RENT	48.909732	Not Verified	0	vacation	26.24	16	0	33.131705	41.8	25	
1	8.987197	0	11.99	265.68	4	MORTGAGE	40.207258	Not Verified	0	debt_consolidation	22.05	17	0	27.203312	53.3	27	
2	9.655026	0	10.49	506.97	1	RENT	35.049454	Source Verified	0	credit_card	12.79	13	0	22.886014	92.2	26	
3	8.881836	0	6.49	220.65	6	RENT	37.797631	Not Verified	0	credit_card	2.60	6	0	17.621736	21.5	13	
4	10.101313	1	17.27	609.33	9	MORTGAGE	38.029525	Verified	1	credit_card	33.95	13	0	29.077084	69.8	43	

In [ ]:
```python
df_model.write.parquet("df_model.parquet", mode='overwrite')
```

In [ ]:
```python
df_model = pd.read_parquet("df_model.parquet")
df_model.head()
```

Out[ ]:

	loan_amnt	term	int_rate	installment	emp_length	home_ownership	annual_inc	verification_status	loan_status	purpose	dti	open_acc	pub_rec	revol_bal	revol_util	total_acc	initial_list_stat
0	9.210340	0	11.44	329.48	10	RENT	48.909732	Not Verified	0	vacation	26.24	16	0	33.131705	41.8	25	
1	8.987197	0	11.99	265.68	4	MORTGAGE	40.207258	Not Verified	0	debt_consolidation	22.05	17	0	27.203312	53.3	27	
2	9.655026	0	10.49	506.97	1	RENT	35.049454	Source Verified	0	credit_card	12.79	13	0	22.886014	92.2	26	
3	8.881836	0	6.49	220.65	6	RENT	37.797631	Not Verified	0	credit_card	2.60	6	0	17.621736	21.5	13	
4	10.101313	1	17.27	609.33	9	MORTGAGE	38.029525	Verified	1	credit_card	33.95	13	0	29.077084	69.8	43	

In [ ]:
```
df_model["was_bankrupt"] = df_model["pub_rec_bankruptcies"].apply(lambda x: 1 if x > 0 else 0)
df_model["has_pub_rec"] = df_model["pub_rec"].apply(lambda x: 1 if x > 0 else 0)
df_model["has_mort_acc"] = df_model["mort_acc"].apply(lambda x: 1 if x > 0 else 0)
df_model = df_model.drop(["pub_rec_bankruptcies", "pub_rec", "mort_acc"], axis=1)
```

In [ ]:
```
df_model.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 395994 entries, 0 to 395993
Data columns (total 29 columns):
 #   Column                    Non-Null Count   Dtype  
---  ------                    --------------   -----  
 0   loan_amnt                 395994 non-null  float64
 1   term                      395994 non-null  int32  
 2   int_rate                  395994 non-null  float64
 3   installment               395994 non-null  float64
 4   emp_length                395994 non-null  int32  
 5   home_ownership            395994 non-null  object 
 6   annual_inc                395994 non-null  float64
 7   verification_status       395994 non-null  object 
 8   loan_status               395994 non-null  int32  
 9   purpose                   395994 non-null  object 
 10  dti                       395994 non-null  float64
 11  open_acc                  395994 non-null  int32  
 12  revol_bal                 395994 non-null  float64
 13  revol_util                395994 non-null  float64
 14  total_acc                 395994 non-null  int32  
 15  initial_list_status       395994 non-null  int32  
 16  application_type          395994 non-null  object 
 17  zipcode                   395994 non-null  object 
 18  issue_year                395994 non-null  int32  
 19  earliest_cr_line_year     395994 non-null  int32  
 20  sub_grade                 395994 non-null  int32  
 21  state                     395994 non-null  float64
 22  issue_month_sin           395994 non-null  float64
 23  issue_month_cos           395994 non-null  float64
 24  earliest_cr_line_month_sin  395994 non-null  float64
 25  earliest_cr_line_month_cos  395994 non-null  float64
 26  was_bankrupt              395994 non-null  int64  
 27  has_pub_rec               395994 non-null  int64  
 28  has_mort_acc              395994 non-null  int64  
dtypes: float64(12), int32(9), int64(3), object(5)
memory usage: 74.0+ MB
```

In [ ]:
```
df_model["loan_status"].value_counts(normalize=True)
```

Out[ ]:
```
loan_status
0    0.803876
1    0.196124
Name: proportion, dtype: float64
```

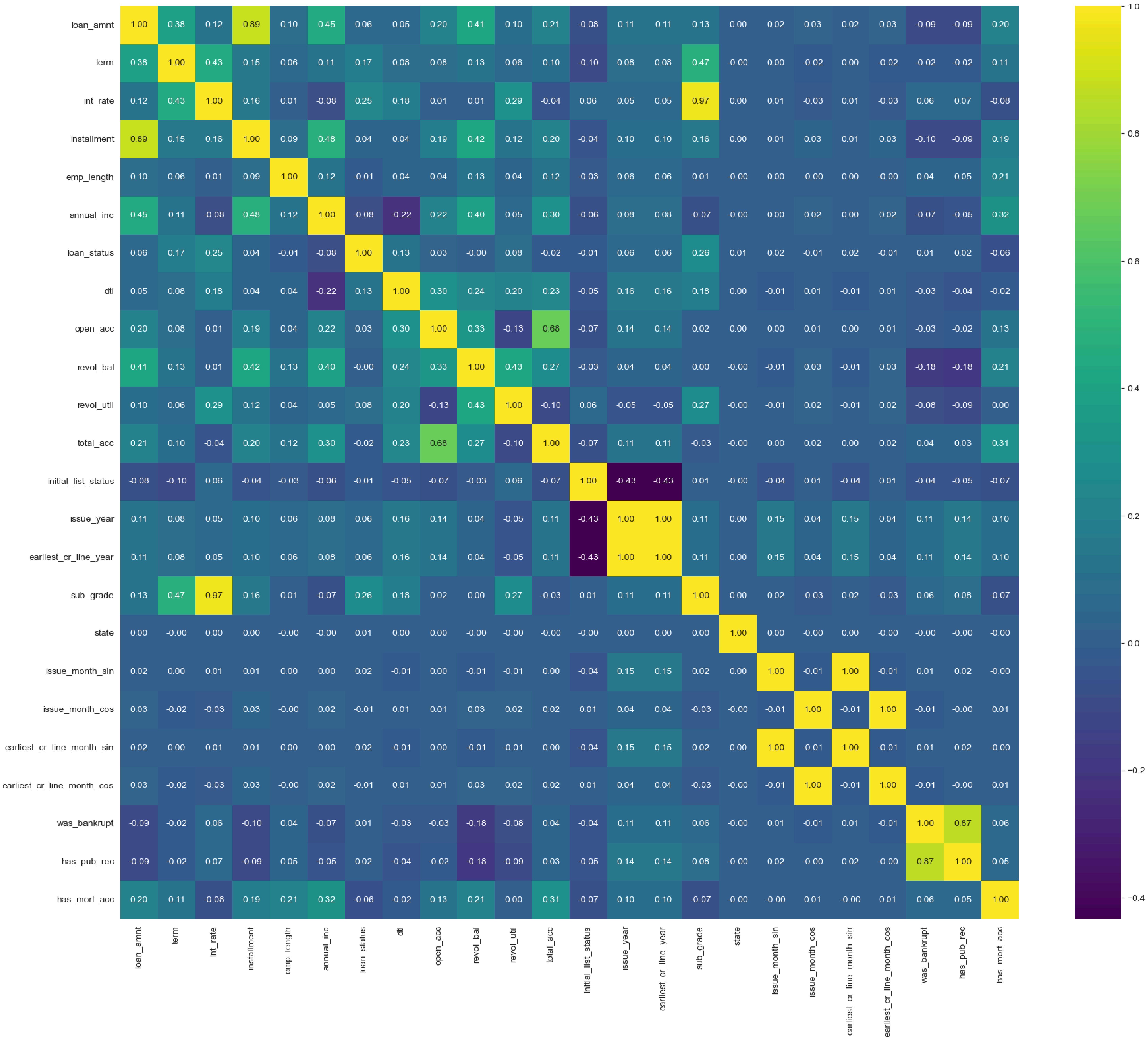
In [ ]:
```
oneHotEncoder = OneHotEncoder()
oneHotEncoder.fit(df_model[["home_ownership", "verification_status", "purpose", "zipcode", "application_type"]])
df_encoded = oneHotEncoder.transform(df_model[["home_ownership", "verification_status", "purpose", "zipcode", "application_type"]])
df_encoded_dataframe = pd.DataFrame(df_encoded.toarray(), columns=oneHotEncoder.get_feature_names_out(["home_ownership", "verification_status", "purpose", "zipcode", "application_type"]))
```

Out[ ]:
```
▾ OneHotEncoder
OneHotEncoder()
```

Finding Correlation

In [ ]:
```
cols=df_model.select_dtypes(include=['int64', 'int32', "float64"]).columns
corr = df_model[cols].corr()
```

In [ ]:
```
plt.figure(figsize=(22, 18))
sns.heatmap(corr, annot=True, fmt=".2f", cmap='viridis');
```



Observations

From above plot we can see that following features are correlated

- Interest Rate and SubGrade
- loan amount and Installement amount
- Earliest credit month sin and issue month sin are correlated

Dropping Correlated Features

In [ ]:
```python
to_drop = ["loan_amnt", "int_rate", "issue_month_cos", "issue_month_sin", "issue_year"]
df_model = df_model.drop(to_drop, axis=1)
```

In [ ]:
```python
df_model = pd.concat([df_model, df_encoded_dataframe], axis=1)
df_model = df_model.drop(["home_ownership", "verification_status", "purpose", "zipcode", "application_type"], axis=1)
```

In [ ]:
```python
df_model
```

Out[ ]:

	term	installment	emp_length	annual_inc	loan_status	dti	open_acc	revol_bal	revol_util	total_acc	initial_list_status	earliest_cr_line_year	sub_grade	state	earliest_cr_line_month_sin	earlie
0	0	329.48	10	48.909732	0	26.24	16	33.131705	41.8	25	0	2015	8	0.192013	0.000000	
1	0	265.68	4	40.207258	0	22.05	17	27.203312	53.3	27	1	2015	9	0.197067	0.000000	
2	0	506.97	1	35.049454	0	12.79	13	22.886014	92.2	26	1	2015	7	0.203976	0.000000	
3	0	220.65	6	37.797631	0	2.60	6	17.621736	21.5	13	1	2014	1	0.195983	-0.866025	
4	1	609.33	9	38.029525	1	33.95	13	29.077084	69.8	43	1	2013	14	0.195129	1.000000	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
395989	1	217.38	2	34.199519	0	15.63	6	12.578177	34.3	23	0	2015	8	0.195557	-1.000000	
395990	0	700.42	5	47.914199	0	21.45	6	35.105261	95.7	8	1	2015	10	0.193690	0.500000	
395991	0	161.32	10	38.372151	0	17.56	15	31.979153	66.9	23	1	2013	5	0.186492	-1.000000	
395992	1	503.02	10	40.000000	0	15.88	9	25.042063	53.8	20	1	2012	11	0.193469	-0.500000	
395993	0	67.98	10	35.032894	0	8.32	3	16.251243	91.3	19	1	2010	11	0.198048	0.500000	

395994 rows × 55 columns

In [ ]:
```python
X_train, X_test, y_train, y_test = train_test_split(df_model.drop("loan_status", axis=1), df_model["loan_status"], test_size=0.2, random_state=25)
```

In [ ]:
```python
min_max_scale_cols = ['earliest_cr_line_year', 'emp_length', 'sub_grade']
standard_scale_cols = ['annual_inc', 'dti',  'installment', 'open_acc',  'revol_bal', 'revol_util', 'total_acc',]
```

In [ ]:
```python
def perform_scaling(scaler, X_train, X_test, cols):
    X_train_scaled = X_train.copy()
    X_test_scaled = X_test.copy()
    X_train_scaled[cols] = scaler.fit_transform(X_train[cols])
    X_test_scaled[cols] = scaler.transform(X_test[cols])
    return X_train_scaled, X_test_scaled
```

In [ ]:
```python
X_train, X_test = perform_scaling(MinMaxScaler(), X_train, X_test, min_max_scale_cols)
X_train, X_test = perform_scaling(StandardScaler(), X_train, X_test, standard_scale_cols)
```

In [ ]:
```python
X_train.head()
```

Out[ ]:

	term	installment	emp_length	annual_inc	dti	open_acc	revol_bal	revol_util	total_acc	initial_list_status	earliest_cr_line_year	sub_grade	state	earliest_cr_line_month_sin	earliest_cr_li
164395	1	-0.435095	1.000000	-0.292487	1.438927	-0.838918	1.122573	1.324278	-1.464612	1	0.666667	0.441176	0.198023	-0.500000	
241040	0	-0.917292	0.000000	-0.940833	-0.861258	-1.227969	-0.836158	1.557287	-0.959992	0	0.777778	0.323529	0.194740	-0.866025	
365230	0	-0.430867	0.777778	-0.202413	-0.110529	-0.644393	-0.611847	-0.654258	-1.380508	1	0.555556	0.058824	0.195210	-0.866025	
26021	0	0.298293	0.000000	-0.109585	-0.542506	-0.255341	-0.937393	-1.161155	0.974386	1	0.555556	0.264706	0.195557	0.866025	
305341	0	-0.925031	0.555556	-0.202413	-1.520915	-0.644393	-0.553120	0.429031	-1.296405	1	0.666667	0.470588	0.196346	0.866025	

VIF

In [ ]:
```python
vif_cols = ['term', 'installment', 'emp_length', 'annual_inc', 'dti', 'open_acc',
       'revol_bal', 'revol_util', 'total_acc', 'initial_list_status',
       'earliest_cr_line_year', 'sub_grade',
       'earliest_cr_line_month_sin', 'earliest_cr_line_month_cos', 'state',
       'was_bankrupt', 'has_pub_rec', 'has_mort_acc']
```

In [ ]:
```python
X_vif = pd.DataFrame(X_train[vif_cols], columns=vif_cols)
vif = pd.DataFrame()

vif['Features'] = X_vif.columns
vif['VIF'] = [variance_inflation_factor(X_vif.values, i) for i in range(X_vif.shape[1])]
vif['VIF'] = round(vif['VIF'], 2)
vif = vif.sort_values(by = "VIF", ascending = False)
vif
```

Out[ ]:

	Features	VIF
14	state	38.71
10	earliest_cr_line_year	28.37
11	sub_grade	6.26
16	has_pub_rec	4.86
15	was_bankrupt	4.64
2	emp_length	3.32
9	initial_list_status	3.14
17	has_mort_acc	3.11
5	open_acc	2.22
8	total_acc	2.16
6	revol_bal	2.06
0	term	1.84
3	annual_inc	1.80
7	revol_util	1.62
1	installment	1.52
4	dti	1.46
12	earliest_cr_line_month_sin	1.03
13	earliest_cr_line_month_cos	1.01

In [ ]:
```python
vif_cols =list(set(vif_cols) - set(["state", "earliest_cr_line_year"]))

X_vif = pd.DataFrame(X_train[vif_cols], columns=vif_cols)
vif = pd.DataFrame()

vif['Features'] = X_vif.columns
vif['VIF'] = [variance_inflation_factor(X_vif.values, i) for i in range(X_vif.shape[1])]
vif['VIF'] = round(vif['VIF'], 2)
vif = vif.sort_values(by = "VIF", ascending = False)
vif
```

Out[ ]:

	Features	VIF
5	has_pub_rec	4.83
13	was_bankrupt	4.64
9	sub_grade	4.18
8	emp_length	2.78
3	has_mort_acc	2.60
12	open_acc	2.21
4	initial_list_status	2.18
14	total_acc	2.14
7	revol_bal	2.05
0	term	1.81
15	annual_inc	1.79
11	revol_util	1.55
1	installment	1.49
6	dti	1.42
2	earliest_cr_line_month_cos	1.00
10	earliest_cr_line_month_sin	1.00

In [ ]:
```python
vif_cols
```

Out[ ]:
```
['term',
 'installment',
 'earliest_cr_line_month_cos',
 'has_mort_acc',
 'initial_list_status',
 'has_pub_rec',
 'dti',
 'revol_bal',
 'emp_length',
 'sub_grade',
 'earliest_cr_line_month_sin',
 'revol_util',
 'open_acc',
 'was_bankrupt',
 'total_acc',
 'annual_inc']
```

In [ ]:
```python
X_train = X_train.drop(["state", "earliest_cr_line_year"], axis=1)
X_test = X_test.drop(["state", "earliest_cr_line_year"], axis=1)
```

In [ ]:
```python
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import GridSearchCV
from sklearn.feature_selection import RFE
from sklearn.metrics import roc_auc_score, roc_curve, confusion_matrix, classification_report
from sklearn.metrics import f1_score, precision_score, recall_score
from sklearn.metrics import precision_recall_curve, auc, ConfusionMatrixDisplay
import pickle
```

Hyperparameter Tuning

In [ ]:
```python
penalty = ['l1', 'l2']
lambdas = [0.01, 0.1, 1, 10, 100, 1000, 10000]
C_values = [1/lambda_val for lambda_val in lambdas]
param_grid = {'C': C_values, 'penalty': penalty}
model = LogisticRegression(max_iter=1000 , penalty=penalty, class_weight='balanced', solver='liblinear', n_jobs=-1)
```

In [ ]:
```python
grid_search = GridSearchCV(model, param_grid, scoring='f1', n_jobs=-1, cv=2, verbose=3)
grid_search.fit(X_train, y_train)
```

```
Fitting 2 folds for each of 14 candidates, totalling 28 fits
[CV 1/2] END ...............C=10.0, penalty=l1;, score=0.625 total time=  12.8s
[CV 1/2] END ..............C=100.0, penalty=l1;, score=0.624 total time=  13.0s
[CV 2/2] END ..............C=100.0, penalty=l1;, score=0.622 total time=  13.0s
[CV 2/2] END ...............C=10.0, penalty=l1;, score=0.622 total time=  13.1s
[CV 2/2] END ................C=0.1, penalty=l2;, score=0.622 total time=  18.9s
[CV 1/2] END ................C=0.1, penalty=l2;, score=0.625 total time=  19.9s
[CV 2/2] END ................C=1.0, penalty=l2;, score=0.622 total time=  23.5s
[CV 1/2] END ................C=1.0, penalty=l2;, score=0.625 total time=  24.3s
[CV 1/2] END ...............C=0.01, penalty=l2;, score=0.625 total time=  12.7s
[CV 2/2] END ...............C=0.01, penalty=l2;, score=0.622 total time=  13.0s
[CV 1/2] END .............C=0.0001, penalty=l1;, score=0.557 total time=   3.0s
[CV 2/2] END .............C=0.0001, penalty=l1;, score=0.559 total time=   3.2s
[CV 1/2] END ..............C=0.001, penalty=l1;, score=0.622 total time=  11.1s
[CV 1/2] END ..............C=0.001, penalty=l2;, score=0.622 total time=   6.9s
[CV 2/2] END ...............C=10.0, penalty=l2;, score=0.622 total time=  31.4s
[CV 2/2] END ..............C=100.0, penalty=l2;, score=0.622 total time=  31.5s
[CV 2/2] END ..............C=0.001, penalty=l2;, score=0.619 total time=   7.0s
[CV 2/2] END ..............C=0.001, penalty=l1;, score=0.619 total time=  11.5s
[CV 1/2] END .............C=0.0001, penalty=l2;, score=0.604 total time=   3.6s
[CV 2/2] END .............C=0.0001, penalty=l2;, score=0.600 total time=   3.5s
[CV 1/2] END ...............C=10.0, penalty=l2;, score=0.625 total time=  33.9s
[CV 1/2] END ..............C=100.0, penalty=l2;, score=0.625 total time=  34.9s
[CV 1/2] END ...............C=0.01, penalty=l1;, score=0.624 total time=  25.7s
[CV 2/2] END ...............C=0.01, penalty=l1;, score=0.622 total time=  26.1s
[CV 2/2] END ................C=1.0, penalty=l1;, score=0.622 total time=  51.2s
[CV 1/2] END ................C=0.1, penalty=l1;, score=0.625 total time= 1.1min
[CV 1/2] END ................C=1.0, penalty=l1;, score=0.625 total time= 1.1min
[CV 2/2] END ................C=0.1, penalty=l1;, score=0.622 total time= 1.2min
```

Out[ ]:
```
▸ GridSearchCV
▸ estimator: LogisticRegression
    ▸ LogisticRegression
```

In [ ]:
```python
grid_search.best_params_
```

Out[ ]:
```
{'C': 0.1, 'penalty': 'l1'}
```

Training Model with Updated values

In [ ]:
```python
model = LogisticRegression(max_iter=1000 , C=0.1,  penalty="l1", class_weight='balanced', solver='liblinear', n_jobs=-1)
model.fit(X_train, y_train)
```

Out[ ]:
```
▾ LogisticRegression
LogisticRegression(C=0.1, class_weight='balanced', max_iter=1000, n_jobs=1,
                   penalty='l1', solver='liblinear')
```

In [ ]:
```python
# with open("logistic_regression_model.pkl", "wb") as f:
#     pickle.dump(model, f)

with open("logistic_regression_model.pkl", "rb") as f:
    model = pickle.load(f)
```

In [ ]:
```python
y_test_pred = model.predict(X_test)
y_train_pred = model.predict(X_train)
y_test_prob = model.predict_proba(X_test)
y_train_prob = model.predict_proba(X_train)
```

Calculate best threshold based on f1 score

In [ ]:
```python
thresholds = np.arange(0.0, 1.0, 0.01)

precisions = []
recalls = []
f1s = []

for threshold in thresholds:
    _y_test_pred = (y_test_prob[:, 1] >= threshold).astype(int)
    precisions.append(precision_score(y_test, _y_test_pred))
    recalls.append(recall_score(y_test, _y_test_pred))
    f1s.append(f1_score(y_test, _y_test_pred))

best_threshold = thresholds[np.argmax(f1s)]

print(f'Best threshold: {best_threshold}')
print(f'Precision at best threshold: {precisions[np.argmax(f1s)]}')
print(f'Recall at best threshold: {recalls[np.argmax(f1s)]}')
print(f'F1 score at best threshold: {f1s[np.argmax(f1s)]}')
```

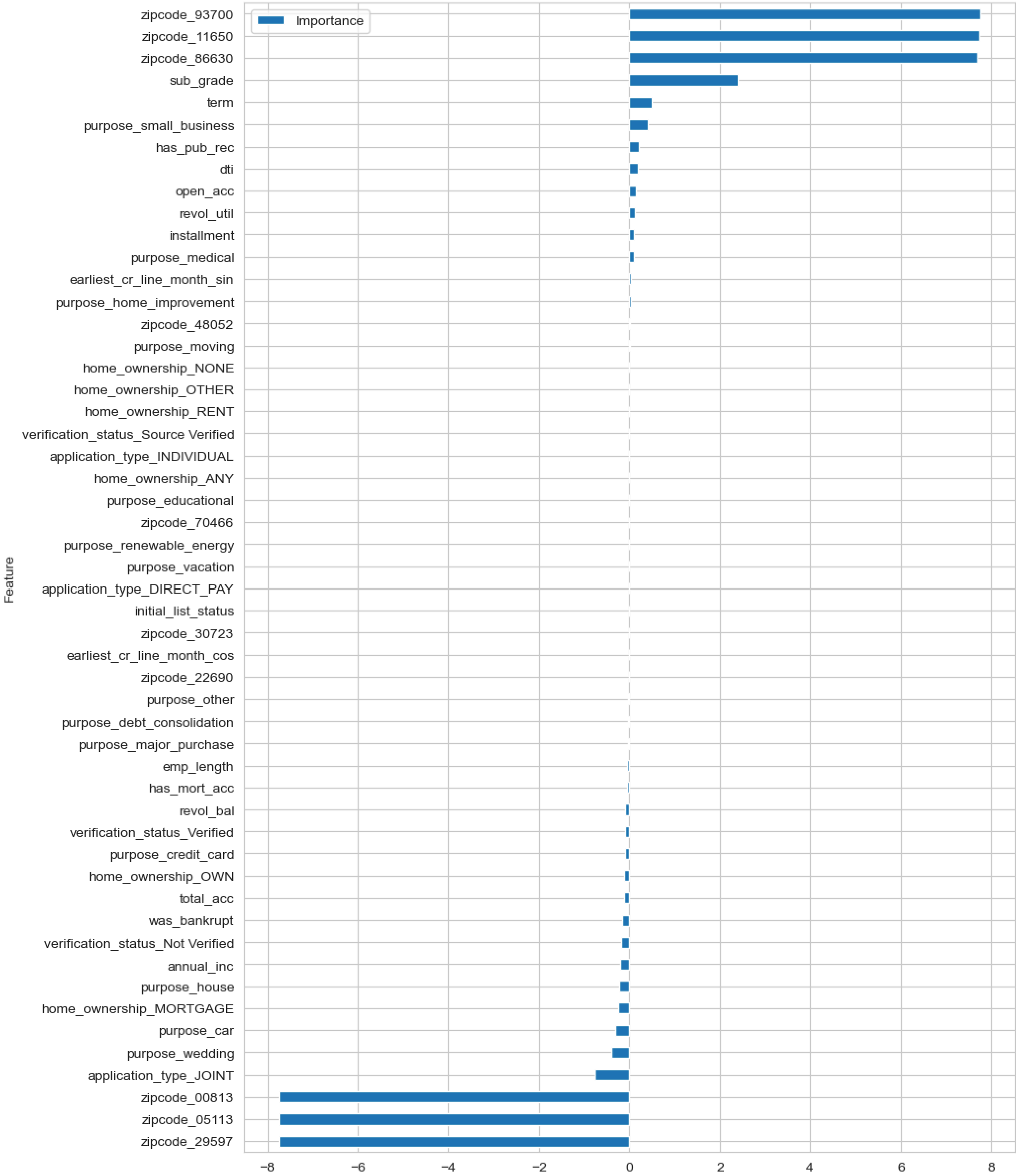
```
Best threshold: 0.67
Precision at best threshold: 0.7145569139646138
Recall at best threshold: 0.6050279688806018
F1 score at best threshold: 0.6552468491052155
```

Use best threshold

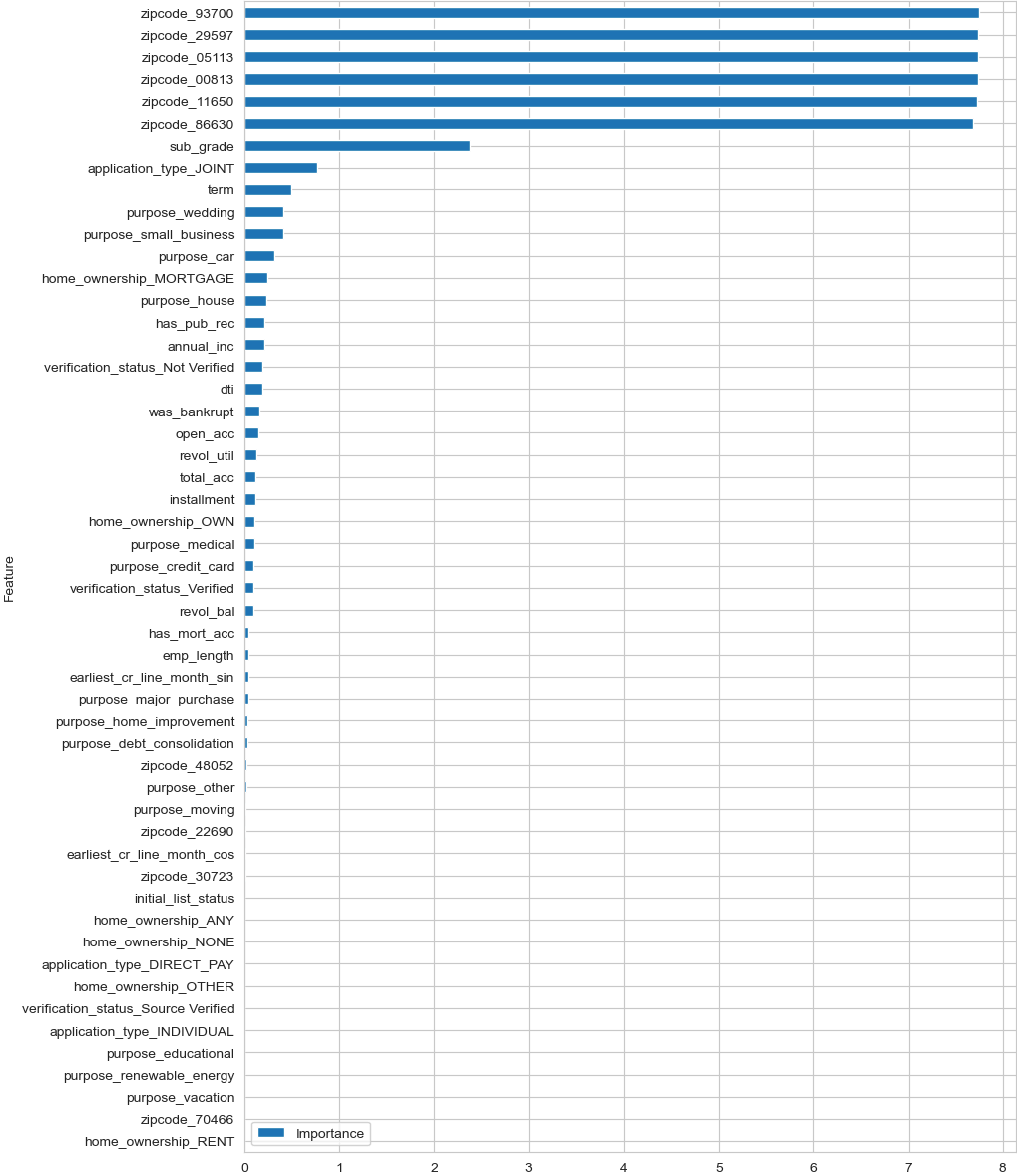
In [ ]:
```
# best_threshold = 0.66
y_test_pred = (y_test_prob[:, 1] >= best_threshold).astype(int)
y_train_pred = (y_train_prob[:, 1] >= best_threshold).astype(int)
```

Feature Importance

In [ ]:
```
coefficients = model.coef_[0]
feature_importance = pd.DataFrame({'Feature': X_train.columns, 'Importance': coefficients})
feature_importance = feature_importance.sort_values('Importance', ascending=True)
feature_importance.plot(x='Feature', y='Importance', kind='barh', figsize=(10, 15));
```



In [ ]:
```
coefficients = model.coef_[0]
feature_importance = pd.DataFrame({'Feature': X_train.columns, 'Importance': np.abs(coefficients)})
feature_importance = feature_importance.sort_values('Importance', ascending=True)
feature_importance.plot(x='Feature', y='Importance', kind='barh', figsize=(10, 15));
```



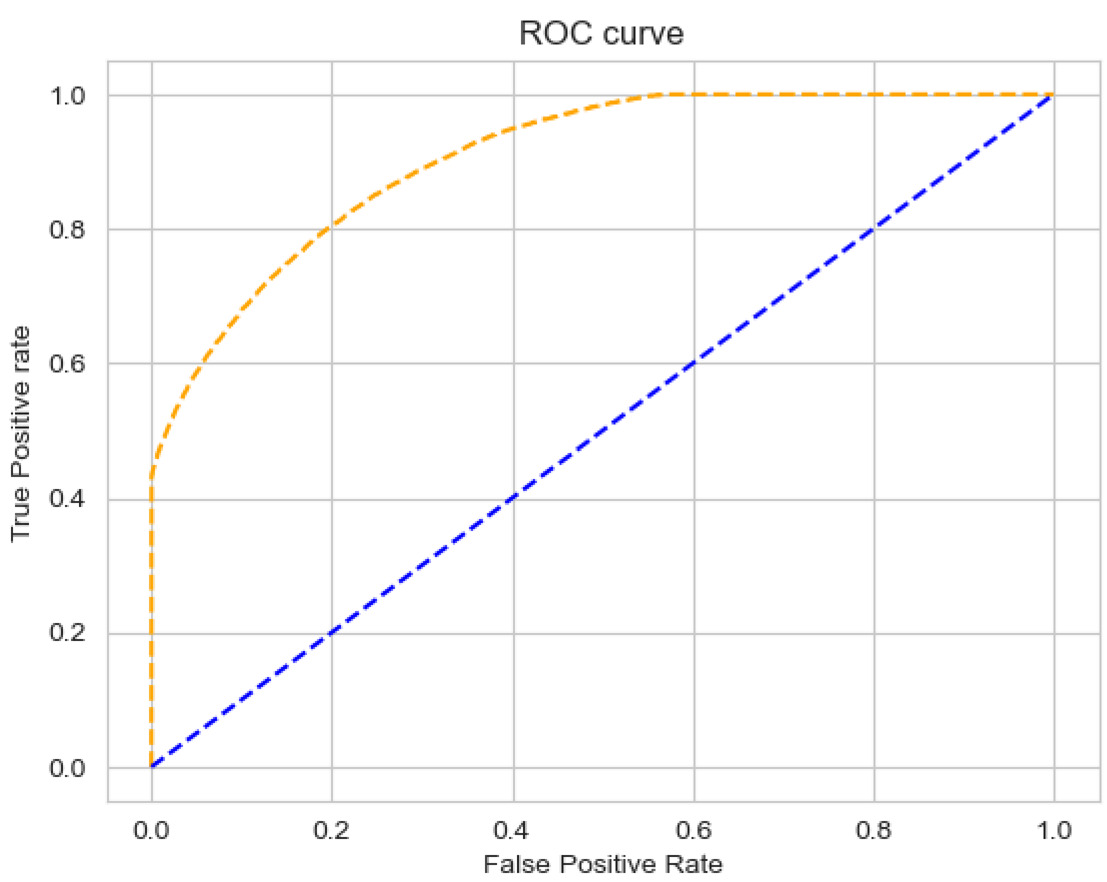
Results Evaluation

ROC Curve

In [ ]:
```
fpr, tpr, threshold = roc_curve(y_test, y_test_prob[:,1], pos_label=1)

random_probs = [0 for i in range(len(y_test))]
p_fpr, p_tpr, _ = roc_curve(y_test, random_probs, pos_label=1)

auc_score = roc_auc_score(y_test, y_test_prob[:,1])
print("AUC Score: ",auc_score)
```

```
AUC Score:  0.9055573414285613
```

In [ ]:
```
plt.plot(fpr, tpr, linestyle='--',color='orange', label='Logistic Regression')
plt.plot(p_fpr, p_tpr, linestyle='--', color='blue')
plt.title('ROC curve')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive rate')
plt.show();
```



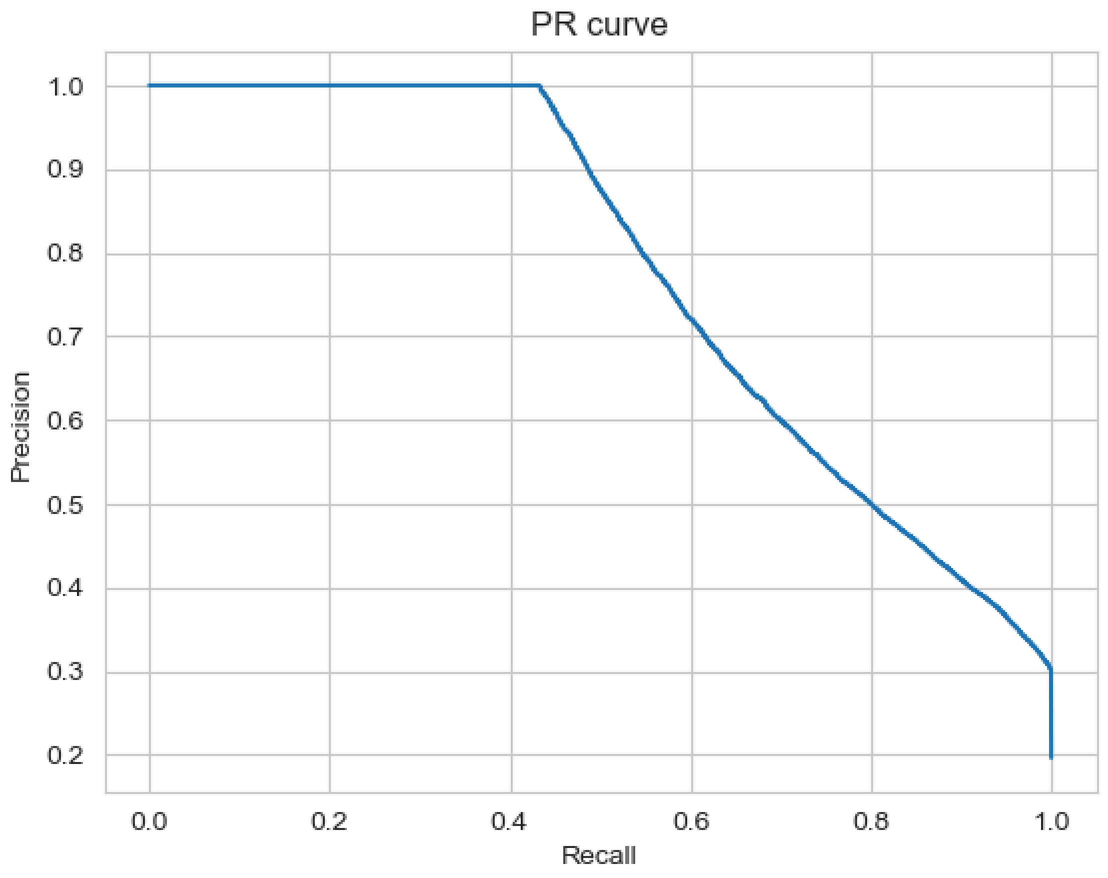
Observations

- From above plot we can see that there is good balance of TPR and FPR

Precision recall Curve

In [ ]:
```
precision, recall, thr = precision_recall_curve(y_test, y_test_prob[:,1])
plt.plot(recall, precision)
plt.xlabel('Recall')
```

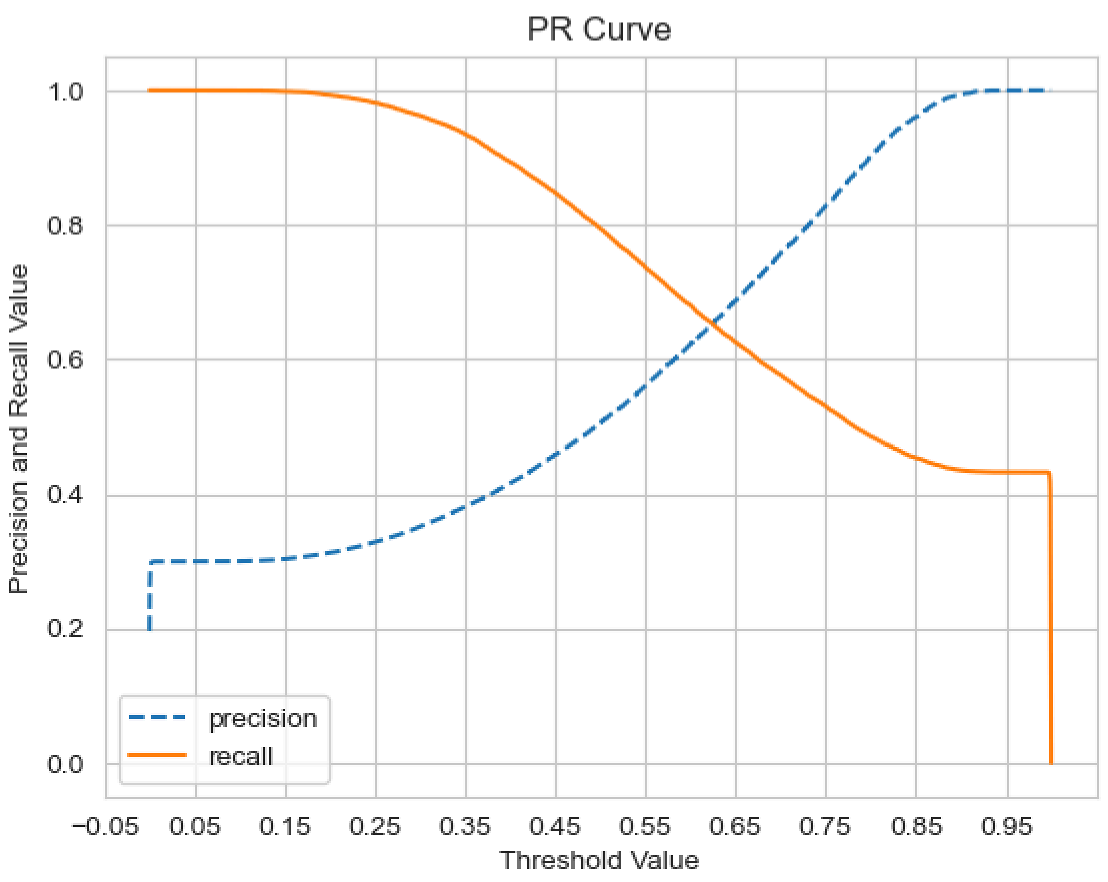
```
plt.ylabel('Precision')
plt.title('PR curve')
plt.show();
```



Observations

Above plot shows value of precision against recall

In [ ]:
```
precision, recall, thr = precision_recall_curve(y_test, y_test_prob[:,1])

threshold_boundary = thr.shape[0]

plt.plot(thr, precision[0:threshold_boundary], linestyle='--', label='precision')
plt.plot(thr, recall[0:threshold_boundary], label='recall')

start, end = plt.xlim()
plt.xticks(np.round(np.arange(start, end, 0.1), 2))

plt.xlabel('Threshold Value')
plt.ylabel('Precision and Recall Value')
plt.legend()
plt.title("PR Curve")
plt.show();
```

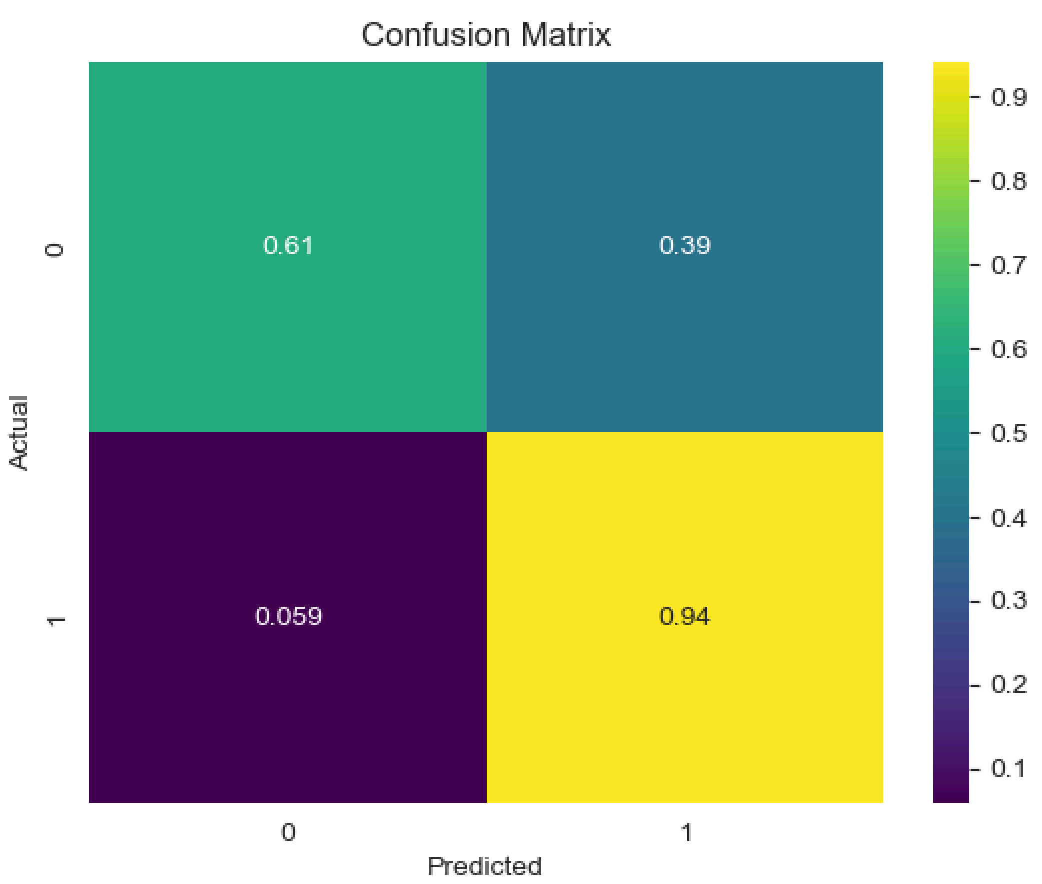


Observations

- From above plot we can see that as threshold value increases, precesion value increases.
- At the same time the value of recall decreases

Confusion Matrix

In [ ]:
```
cm = confusion_matrix(y_test, y_test_pred,  normalize='true', labels=[1, 0])
sns.heatmap(cm, annot=True, cmap='viridis');
plt.title('Confusion Matrix')
plt.xlabel('Predicted')
plt.ylabel('Actual');
```



Observations

- Above is the confusion matrix for the model
- 94% of paid loans were correctly predicted
- 61% of the unpaid loans were correctly predicted
- 39% of unpaid were predicted as paid
- 5% of the paid loans were predicted as not paid

Test Classification Report

In [ ]:
```
print(classification_report(y_test, y_test_pred))
```

```
              precision    recall  f1-score   support

           0       0.91      0.94      0.92     63646
           1       0.71      0.61      0.66     15553

    accuracy                           0.87     79199
   macro avg       0.81      0.77      0.79     79199
weighted avg       0.87      0.87      0.87     79199
```

Train Classification Report

In [ ]:
```
print(classification_report(y_train, y_train_pred))
```

```
              precision    recall  f1-score   support

           0       0.91      0.94      0.92    254684
           1       0.71      0.60      0.66     62111

    accuracy                           0.88    316795
   macro avg       0.81      0.77      0.79    316795
weighted avg       0.87      0.88      0.87    316795
```

Observations

- Class 0 (Defaulted Loans)
    - **Precision (0.91)**: Out of all the loans predicted to be defaulted, 91% were actually defaulted. This means that the model is quite accurate in predicting defaults and has a low false positive rate for this class.
    - **Recall (0.94)**: Out of all the actual defaulted loans, 94% were correctly identified by the model. This indicates that the model is very effective at capturing most of the defaulted loans, with a low false negative rate.
    - **F1-Score (0.92)**: It shows that the model has a good balance between precision and recall for defaulted loans.
- Class 1 (Paid Off Loans)
    - **Precision (0.71)**: Out of all the loans predicted to be paid off, 71% were actually paid off. This means that the model has a higher false positive rate for this class compared to class 0.
    - **Recall (0.60)**: Out of all the actual paid off loans, 60% were correctly identified by the model. This indicates that the model misses a significant number of loans that were actually paid off (higher false negative rate).
    - **F1-Score (0.66)**: The F1-score for paid off loans is 0.66, indicating a moderate balance between precision and recall for this class, but not as strong as for defaulted loans.

In [ ]:
```
scores = {
    "f1":{
        "test": f1_score(y_test, y_test_pred),
        "train": f1_score(y_train, y_train_pred)
    },
    "precision":{
        "test": precision_score(y_test, y_test_pred),
        "train": precision_score(y_train, y_train_pred)
    },
    "recall":{
        "test": recall_score(y_test, y_test_pred),
        "train": recall_score(y_train, y_train_pred)
    },
    "auc":{
        "test": roc_auc_score(y_test, y_test_prob[:,1]),
        "train": roc_auc_score(y_train, y_train_prob[:,1])
    }
}
pd.DataFrame(scores)
```

Out[ ]:

	f1	precision	recall	auc
test	0.655247	0.714557	0.605028	0.905557
train	0.655008	0.714704	0.604514	0.906478

Conclusion

Precision-Recall Tradeoff

Increasing Precision

Precision is the ratio of true positives (correctly predicted positive instances) to the total predicted positives (true positives + false positives). To increase precision, you need to decrease the number of false positives. In the context of loan predictions: A false positive occurs when the model predicts that a loan will be paid off (positive prediction) but it is not paid off Reducing these false positives (incorrectly predicted paid-off loans) will increase precision.

Increasing Recall

Recall is the ratio of true positives to the total actual positives (true positives + false negatives). To increase recall, you need to decrease the number of false negatives. In the context of loan predictions: A false negative occurs when the model predicts that a loan will not be paid off (negative prediction) but it is actually paid off (positive actual). Reducing these false negatives (missed paid-off loans) will increase recall.

To Summarize Precision: Focus on minimizing false positives (loans predicted to be paid off but actually defaulted). Recall: Focus on minimizing false negatives (loans predicted to default but actually paid off). Balancing precision and recall often involves trade-offs, and the choice of threshold can help you achieve the desired balance depending on your specific goals.

Main Metric

According to me the bank should to focus on f1 score as the main metric because the value of f1 score is derived from precision and recall. This will keep both loss due to bad loans and profit due to good loan high.

Predictive Modelling

Model Performance

- The above model has high precision when predicting loan defaulters, but has mediocre precision for predicting whether a loan will be paid or not.
- This model is better for minimizing risk, but will take a hit on maximizing profitability.
- This can be adjusted by changing the threshold value, but trying to increase of decrease precision might affect recall value.

Gap between Precision and Recall

- If the gap between precision and recall increases, this might lead to incorrect prediction. This might lead to losses in business.

Insights and Answers to Questionaire

- Geographical features had big influence on the outcome
- Zipcode feature has the strongest influence on the model.
- Subgrade, application type and term is next 3 important feature.
- Around 80% of the users have paid their loans
- Loan and loan installment have 95% correlation
- People with Grade A have high chances of paying off their loans
- Income of the users have very high right skew. This shows how diverse is the user base
- Managers and Teachers have highest numbers of loans
- We can see that average loan amount for users is highest for houses and lowest for vacations.
- The users with Joint application have higher loan amount.
- G graded employees take the highest loan amount while B takes lowest.

- House and small business purpose have highest number of loans
- Most loans are taken by users who are younger and decreases as experience increases
- Most of the loans are taken for 36 months
- Most of the loans are taken by B grade users
- Majority of the user have mortgage type home ownership
- House and small business have higher average loan amount

Recommendation

- A more powerful model can be used inplace of Logistic regression
- Users with zipcodes 11650, 86630, 93700 must not be given loans
- Users with zipcodes 00813, 05113, 29597 can be given loans
- Users with higher grades have higher default ratio. Other than higher interest rate, additional checking can be used to give loans
- Provide more options for the term period so customers can make payments according to their incomes.
- Model should be continously trained and threshold should be adjusted according to banks profit and losses
- Bank should deeply check Small businesses loans as these have highest charged off rate wrt to other categories
- Bank should encourage more joint pay loans as compared to direct pay and individual type as joint application types have highher repayment rate