HEICODERS ACADEMY

AI200: Applied Machine Learning

Capstone Project: Loan Default Prediction

Dataset Used:

- LendingClub Loan Default Dataset from Kaggle

🏦 Lending Club Credit Default Prediction Capstone Project

📑 Introduction

> `LendingClub` is a US peer-to-peer lending company, headquartered in San Francisco, California. It was the first peer-to-peer lender to register its offerings as securities with the Securities and Exchange Commission (SEC), and to offer loan trading on a secondary market. `LendingClub` is the world's largest peer-to-peer lending platform.

> Solving this case study will give us an idea about how real business problems are solved using EDA and Machine Learning. In this case study, we will also develop a basic understanding of risk analytics in banking and financial services and understand how data is used to minimise the risk of losing money while lending to customers.

📝 Business Understanding

> You work for the `LendingClub` company which specialises in lending various types of loans to urban customers. When the company receives a loan application, the company has to make a decision for loan approval based on the applicant's profile. Two types of risks are associated with the bank's decision:
>
> - If the applicant is likely to repay the loan, then not approving the loan results in a loss of business to the company
> - If the applicant is not likely to repay the loan, i.e. he/she is likely to default, then approving the loan may lead to a financial loss for the company

> The data given contains the information about past loan applicants and whether they 'defaulted' or not. The aim is to identify patterns which indicate if a person is likely to default, which may be used for takin actions such as denying the loan, reducing the amount of loan, lending (to risky applicants) at a higher interest rate, etc.

> When a person applies for a loan, there are two types of decisions that could be taken by the company:
>
> 1. `Loan accepted` : If the company approves the loan, there are 3 possible scenarios described below:
>   - `Fully paid` : Applicant has fully paid the loan (the principal and the interest rate)
>   - `Current` : Applicant is in the process of paying the instalments, i.e. the tenure of the loan is not yet completed. These candidates are not labelled as 'defaulted'.
>   - `Charged-off` : Applicant has not paid the instalments in due time for a long period of time, i.e. he/she has defaulted on the loan
> 2. `Loan rejected` : The company had rejected the loan (because the candidate does not meet their requirements etc.). Since the loan was rejected, there is no transactional history of those applicants with the company and so this data is not available with the company (and thus in this dataset)

🎯 Business Objectives

> - `LendingClub` is the largest online loan marketplace, facilitating personal loans, business loans, and financing of medical procedures. Borrowers can easily access lower interest rate loans through a fast online interface.
> - Like most other lending companies, lending loans to ' `risky` ' applicants is the largest source of financial loss (called `credit loss`). The credit loss is the amount of money lost by the lender when the borrower refuses to pay or runs away with the money owed. In other words, borrowers who defaultcause the largest amount of loss to the lenders. In this case, the customers labelled as ' `charged-off` ' are the ' `defaulters` '.
> - If one is able to identify these risky loan applicants, then such loans can be reduced thereby cutting down the amount of credit loss. Identification of such applicants using EDA and machine learning is the aim of this case study.
> - In other words, the company wants to understand the driving factors (or driver variables) behind loan default, i.e. the variables which are strong indicators of default. The company can utilise this knowledge for its portfolio and risk assessment.
> - To develop your understanding of the domain, you are advised to independently research a little about risk analytics (understanding the types of variables and their significance should be enough).

💾 Data Description

Here is the information on this particular data set:

	LoanStatNew	Description
0	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.
1	term	The number of payments on the loan. Values are in months and can be either 36 or 60.
2	int_rate	Interest Rate on the loan
3	installment	The monthly payment owed by the borrower if the loan originates.
4	grade	LC assigned loan grade
5	sub_grade	LC assigned loan subgrade
6	emp_title	The job title supplied by the Borrower when applying for the loan.*
7	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.
8	home_ownership	The home ownership status provided by the borrower during registration or obtained from the credit report. Our values are: RENT, OWN, MORTGAGE, OTHER

	LoanStatNew	Description
9	annual_inc	The self-reported annual income provided by the borrower during registration.
10	verification_status	Indicates if income was verified by LC, not verified, or if the income source was verified
11	issue_d	The month which the loan was funded
12	loan_status	Current status of the loan
13	purpose	A category provided by the borrower for the loan request.
14	title	The loan title provided by the borrower
15	dti	A ratio calculated using the borrower's total monthly debt payments on the total debt obligations, excluding mortgage and the requested LC loan, divided by the borrower's self-reported monthly income.
16	earliest_cr_line	The month the borrower's earliest reported credit line was opened
17	open_acc	The number of open credit lines in the borrower's credit file.
18	pub_rec	Number of derogatory public records
19	revol_bal	Total credit revolving balance
20	revol_util	Revolving line utilization rate, or the amount of credit the borrower is using relative to all available revolving credit.
21	total_acc	The total number of credit lines currently in the borrower's credit file
22	initial_list_status	The initial listing status of the loan. Possible values are – W, F
23	application_type	Indicates whether the loan is an individual application or a joint application with two co-borrowers
24	mort_acc	Number of mortgage accounts.
25	pub_rec_bankruptcies	Number of public record bankruptcies
26	address	Residential address of the loan applicant

Before we start, we will import all the libraries needed for this Notebook.

In [1]:
```
# import sys

# # Install XGBoost
# !{sys.executable} -m pip install xgboost

# # Install CatBoost
# !{sys.executable} -m pip install catboost

# # Install Plotly
# !{sys.executable} -m pip install Plotly
```

In [2]:
```
import pandas as pd
import numpy as np
from datetime import datetime
import plotly.express as px
import plotly.graph_objects as go

from sklearn.feature_selection import SelectKBest
from sklearn.feature_selection import f_classif

from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MinMaxScaler

from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.model_selection import StratifiedShuffleSplit
import xgboost as xgb
from catboost import CatBoostClassifier

from sklearn import metrics
from sklearn.metrics import classification_report
from sklearn.metrics import roc_curve, auc ,  roc_auc_score
from sklearn.datasets import make_classification
from sklearn.model_selection import RandomizedSearchCV

import warnings
warnings.filterwarnings("ignore")
```

1. 📝 Data Preparation

> **OVERALL GOAL:**
>
> - Ingest the data and relabelling the outcome variable

1a. Import Dataset

We import the dataset using `read_csv()`

In [3]:
```
train_df = pd.read_csv("data/lc_trainingset.csv")
pd.set_option('display.max_columns', None)
train_df
```

Out[3]:

	id	loan_amnt	term	int_rate	installment	grade	sub_grade	emp_title	emp_length	home_ownership	annual_inc	verification_status	issue_d	purpose	title	dti	earliest_cr_line	open_acc	pub_rec	revol_bal	revol_util	total_acc	initial_list_status	application_type	mort_acc	pub_rec_bankruptcies	
0	T0	25000.0	60 months	14.83	592.52	D	D3	NaN	10+ years	RENT	109000.0	Verified	Dec-2010	small_business	LendingClub Startup Loan	13.90	Aug-1996	11.0	0.0	6390.0	41.5	39.0	f	INDIVIDUAL	NaN	0.0	U 0376\nDF
1	T1	9500.0	36 months	12.99	320.05	C	C2	NaN	NaN	MORTGAGE	40000.0	Verified	Jan-2015	medical	Medical expenses	22.29	Oct-1985	6.0	1.0	62512.0	82.6	20.0	f	INDIVIDUAL	2.0	1.0	3275 We 454\nV
2	T2	9000.0	36 months	8.39	283.65	B	B1	Bus Operator	10+ years	MORTGAGE	50000.0	Not Verified	Apr-2016	debt_consolidation	Debt consolidation	15.27	Jul-1996	14.0	0.0	8835.0	23.2	20.0	w	INDIVIDUAL	2.0	0.0	Un Patrickfu
3	T3	16700.0	60 months	22.99	470.69	F	F1	Business Analyst	7 years	MORTGAGE	68000.0	Verified	Mar-2015	debt_consolidation	Debt consolidation	21.92	Sep-2005	13.0	0.0	23489.0	55.3	26.0	f	INDIVIDUAL	2.0	0.0	931 Kimb Suite 749\r
4	T4	2800.0	36 months	15.80	98.17	C	C3	BUDCO	9 years	MORTGAGE	218554.0	Verified	Jun-2013	home_improvement	Home Improvement	14.93	May-1986	20.0	0.0	33014.0	90.7	44.0	w	INDIVIDUAL	7.0	0.0	657 Monro 128\nPort C
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
316819	T316819	20000.0	36 months	7.89	625.72	A	A5	General Manager	1 year	MORTGAGE	70000.0	Verified	Apr-2015	debt_consolidation	Debt consolidation	24.58	Feb-2002	11.0	1.0	7888.0	21.2	26.0	f	INDIVIDUAL	2.0	1.0	5496 Elijah 248\nJ
316820	T316820	12400.0	36 months	13.67	421.82	B	B5	Sales Manager	10+ years	MORTGAGE	60000.0	Verified	Dec-2013	debt_consolidation	Debt consolidation	18.40	Dec-1996	8.0	1.0	16229.0	58.2	13.0	w	INDIVIDUAL	0.0	1.0	39275 Lar T
316821	T316821	30000.0	36 months	12.69	1006.35	C	C2	senior system Engineer	1 year	MORTGAGE	80000.0	Source Verified	Jul-2015	credit_card	Credit card refinancing	9.41	Nov-2000	16.0	1.0	14300.0	66.2	28.0	w	INDIVIDUAL	3.0	0.0	8911 Miller 743\nLake A
316822	T316822	14000.0	36 months	14.31	480.60	C	C4	NaN	NaN	RENT	70000.0	Not Verified	Feb-2015	other	Other	2.66	Jan-2006	5.0	0.0	6444.0	51.1	10.0	w	INDIVIDUAL	0.0	0.0	Square\nPor
316823	T316823	12000.0	60 months	10.75	259.42	B	B4	Digital Marketing Manager	2 years	RENT	80000.0	Not Verified	Feb-2016	debt_consolidation	Debt consolidation	21.72	Jul-2000	6.0	0.0	17573.0	43.7	17.0	w	INDIVIDUAL	1.0	0.0	Common\nJa

316824 rows × 28 columns

1b. How many rows/columns in each dataset?

We understand the dimensionality of the data using `.shape`

In [4]:
```
print(train_df.shape)
print(train_df.info())
```

```
(316824, 28)
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 316824 entries, 0 to 316823
Data columns (total 28 columns):
 #   Column                Non-Null Count   Dtype  
---  ------                --------------   -----  
 0   id                    316824 non-null  object 
 1   loan_amnt             316824 non-null  float64
 2   term                  316824 non-null  object 
 3   int_rate              316824 non-null  float64
 4   installment           316824 non-null  float64
 5   grade                 316824 non-null  object 
 6   sub_grade             316824 non-null  object 
 7   emp_title             298514 non-null  object 
 8   emp_length            302162 non-null  object 
 9   home_ownership        316824 non-null  object 
 10  annual_inc            316824 non-null  float64
 11  verification_status   316824 non-null  object 
 12  issue_d               316824 non-null  object 
 13  purpose               316824 non-null  object 
 14  title                 315423 non-null  object 
 15  dti                   316824 non-null  float64
 16  earliest_cr_line      316824 non-null  object 
 17  open_acc              316824 non-null  float64
 18  pub_rec               316824 non-null  float64
 19  revol_bal             316824 non-null  float64
 20  revol_util            316598 non-null  float64
 21  total_acc             316824 non-null  float64
 22  initial_list_status   316824 non-null  object 
 23  application_type      316824 non-null  object 
 24  mort_acc              286611 non-null  float64
 25  pub_rec_bankruptcies  316388 non-null  float64
 26  address               316824 non-null  object 
 27  loan_status           316824 non-null  object 
dtypes: float64(12), object(16)
memory usage: 67.7+ MB
None
```

1c. Check Out Loan Status

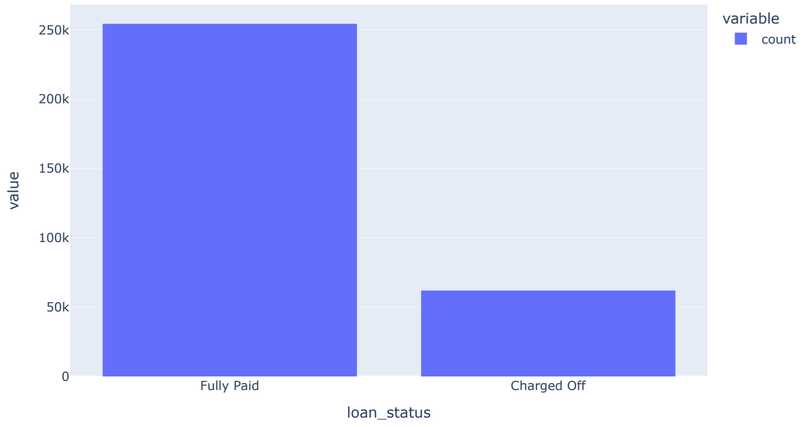
We use `value_counts()` to see the number of instances of each unique status in the loan_status data column

In [5]:
```
train_df['loan_status'].value_counts()
```

Out[5]:
```
loan_status
Fully Paid     254546
Charged Off     62278
Name: count, dtype: int64
```

We visualise the loan status for ease of interpretation

In [6]:
```
fig = px.bar(train_df['loan_status'].value_counts(), width=800, height=500)
fig.show()
```



1d. Relabelling the Loan Status

We are only interested in 2 status i.e. **Defaulted** and **Not Defaulted**. Hence, we will need to add a new variable which will be binary (0s and 1s).

- 0 means Not Defaulted
- 1 means Defaulted

All those loans, whose status is “Fully Paid”, “Current” will be categorized as Not Defaulted and anything else will be categorized as Defaulted. To achieve this we will introduce new variable defaulted.

In [7]:
```
# First we define the function
def change_loan_status(loan_status):
    if loan_status in ['Fully Paid', 'Current']:
        return 0
    else:
        return 1

# Next we apply the function
train_df['loan_status'] = train_df['loan_status'].apply(change_loan_status)
train_df.head()
```

Out[7]:

	id	loan_amnt	term	int_rate	installment	grade	sub_grade	emp_title	emp_length	home_ownership	annual_inc	verification_status	issue_d	purpose	title	dti	earliest_cr_line	open_acc	pub_rec	revol_bal	revol_util	total_acc	initial_list_status	application_type	mort_acc	pub_rec_bankruptcies	address	loan_s
0	T0	25000.0	60 months	14.83	592.52	D	D3	NaN	10+ years	RENT	109000.0	Verified	Dec-2010	small_business	LendingClub Startup Loan	13.90	Aug-1996	11.0	0.0	6390.0	41.5	39.0	f	INDIVIDUAL	NaN	0.0	Unit 8329 Box 0376\nDPO AA 93700	
1	T1	9500.0	36 months	12.99	320.05	C	C2	NaN	NaN	MORTGAGE	40000.0	Verified	Jan-2015	medical	Medical expenses	22.29	Oct-1985	6.0	1.0	62512.0	82.6	20.0	f	INDIVIDUAL	2.0	1.0	3275 West Club Suite 454\nWest Kyle, SC 48052	
2	T2	9000.0	36 months	8.39	283.65	B	B1	Bus Operator	10+ years	MORTGAGE	50000.0	Not Verified	Apr-2016	debt_consolidation	Debt consolidation	15.27	Jul-1996	14.0	0.0	8835.0	23.2	20.0	w	INDIVIDUAL	2.0	0.0	16698 Kline Unions\nNorth Patrickfurt, VA 05113	
3	T3	16700.0	60 months	22.99	470.69	F	F1	Business Analyst	7 years	MORTGAGE	68000.0	Verified	Mar-2015	debt_consolidation	Debt consolidation	21.92	Sep-2005	13.0	0.0	23489.0	55.3	26.0	f	INDIVIDUAL	2.0	0.0	931 Kimberly Passage Suite 749\nDuncanfurt, NM...	
4	T4	2800.0	36 months	15.80	98.17	C	C3	BUDCO	9 years	MORTGAGE	218554.0	Verified	Jun-2013	home_improvement	Home Improvement	14.93	May-1986	20.0	0.0	33014.0	90.7	44.0	w	INDIVIDUAL	7.0	0.0	657 Monroe Row Suite 128\nPort Courtney, ND 00813	

Once again, we apply the `value_counts()` to see the number of instances of each unique status in the loan_status data column. Now we can see that the loan status only has 2 category.

In [8]:
```
train_df['loan_status'].value_counts()
```

Out[8]:
```
loan_status
0    254546
1     62278
Name: count, dtype: int64
```

1d. Data Cleaning

We will start to perform data cleaning. Dropping unwanted features and transforming features to be more relevant.

Using domain knowledge we will drop the columns:

1. "id" as we do not need this
2. "emp_title" and "title" because there are too many different types and this would not be helpful for machine learning
3. "issue_date" as these dates are historical data and future applications that will not be useful for the machine learning model to predict

In [9]:
```
train_drop_df = train_df.drop(columns = ["id","emp_title","issue_d","title"])
train_drop_df.info()
train_modified_df = train_drop_df.copy()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 316824 entries, 0 to 316823
Data columns (total 24 columns):
 #   Column                Non-Null Count   Dtype  
---  ------                --------------   -----  
 0   loan_amnt             316824 non-null  float64
 1   term                  316824 non-null  object 
 2   int_rate              316824 non-null  float64
 3   installment           316824 non-null  float64
 4   grade                 316824 non-null  object 
 5   sub_grade             316824 non-null  object 
 6   emp_length            302162 non-null  object 
 7   home_ownership        316824 non-null  object 
 8   annual_inc            316824 non-null  float64
 9   verification_status   316824 non-null  object 
 10  purpose               316824 non-null  object 
 11  dti                   316824 non-null  float64
 12  earliest_cr_line      316824 non-null  object 
 13  open_acc              316824 non-null  float64
 14  pub_rec               316824 non-null  float64
 15  revol_bal             316824 non-null  float64
 16  revol_util            316598 non-null  float64
 17  total_acc             316824 non-null  float64
 18  initial_list_status   316824 non-null  object 
 19  application_type      316824 non-null  object 
 20  mort_acc              286611 non-null  float64
 21  pub_rec_bankruptcies  316388 non-null  float64
 22  address               316824 non-null  object 
 23  loan_status           316824 non-null  int64  
dtypes: float64(12), int64(1), object(11)
memory usage: 58.0+ MB
```

Next, we will transform the data in 'earliest_cr_line' to date time format and split the month and years to new columns

In [10]:
```
train_modified_df['earliest_cr_line'] = pd.to_datetime(train_modified_df['earliest_cr_line'], format= "%b-%Y")

train_modified_df['year_earliest_cr'] = train_modified_df['earliest_cr_line'].dt.year
train_modified_df['month_earliest_cr'] = train_modified_df['earliest_cr_line'].dt.month

train_modified_df.head()
```

Out[10]:

	loan_amnt	term	int_rate	installment	grade	sub_grade	emp_length	home_ownership	annual_inc	verification_status	purpose	dti	earliest_cr_line	open_acc	pub_rec	revol_bal	revol_util	total_acc	initial_list_status	application_type	mort_acc	pub_rec_bankruptcies	address	loan_status	year_earliest_cr	month_earliest
0	25000.0	60 months	14.83	592.52	D	D3	10+ years	RENT	109000.0	Verified	small_business	13.90	1996-08-01	11.0	0.0	6390.0	41.5	39.0	f	INDIVIDUAL	NaN	0.0	Unit 8329 Box 0376\nDPO AA 93700	1	1996	
1	9500.0	36 months	12.99	320.05	C	C2	NaN	MORTGAGE	40000.0	Verified	medical	22.29	1985-10-01	6.0	1.0	62512.0	82.6	20.0	f	INDIVIDUAL	2.0	1.0	3275 West Club Suite 454\nWest Kyle, SC 48052	1	1985	
2	9000.0	36 months	8.39	283.65	B	B1	10+ years	MORTGAGE	50000.0	Not Verified	debt_consolidation	15.27	1996-07-01	14.0	0.0	8835.0	23.2	20.0	w	INDIVIDUAL	2.0	0.0	16698 Kline Unions\nNorth Patrickfurt, VA 05113	0	1996	
3	16700.0	60 months	22.99	470.69	F	F1	7 years	MORTGAGE	68000.0	Verified	debt_consolidation	21.92	2005-09-01	13.0	0.0	23489.0	55.3	26.0	f	INDIVIDUAL	2.0	0.0	931 Kimberly Passage Suite 749\nDuncanfurt, NM...	0	2005	
4	2800.0	36 months	15.80	98.17	C	C3	9 years	MORTGAGE	218554.0	Verified	home_improvement	14.93	1986-05-01	20.0	0.0	33014.0	90.7	44.0	w	INDIVIDUAL	7.0	0.0	657 Monroe Row Suite 128\nPort Courtney, ND 00813	0	1986	

Lastly, we will clean the address. Since the dataset originates from America, we will need to extract the last 5 digits from the address column to obtain the postal code.

In [11]:
```
train_modified_df['postal_code']= train_modified_df['address'].apply(lambda x: x[-5:])
train_modified_df = train_modified_df.drop(columns = ["address"])
train_modified_df.head()
```

Out[11]:

	loan_amnt	term	int_rate	installment	grade	sub_grade	emp_length	home_ownership	annual_inc	verification_status	purpose	dti	earliest_cr_line	open_acc	pub_rec	revol_bal	revol_util	total_acc	initial_list_status	application_type	mort_acc	pub_rec_bankruptcies	loan_status	year_earliest_cr	month_earliest_cr	postal_code
0	25000.0	60 months	14.83	592.52	D	D3	10+ years	RENT	109000.0	Verified	small_business	13.90	1996-08-01	11.0	0.0	6390.0	41.5	39.0	f	INDIVIDUAL	NaN	0.0	1	1996	8	93700
1	9500.0	36 months	12.99	320.05	C	C2	NaN	MORTGAGE	40000.0	Verified	medical	22.29	1985-10-01	6.0	1.0	62512.0	82.6	20.0	f	INDIVIDUAL	2.0	1.0	1	1985	10	48052
2	9000.0	36 months	8.39	283.65	B	B1	10+ years	MORTGAGE	50000.0	Not Verified	debt_consolidation	15.27	1996-07-01	14.0	0.0	8835.0	23.2	20.0	w	INDIVIDUAL	2.0	0.0	0	1996	7	05113
3	16700.0	60 months	22.99	470.69	F	F1	7 years	MORTGAGE	68000.0	Verified	debt_consolidation	21.92	2005-09-01	13.0	0.0	23489.0	55.3	26.0	f	INDIVIDUAL	2.0	0.0	0	2005	9	00813
4	2800.0	36 months	15.80	98.17	C	C3	9 years	MORTGAGE	218554.0	Verified	home_improvement	14.93	1986-05-01	20.0	0.0	33014.0	90.7	44.0	w	INDIVIDUAL	7.0	0.0	0	1986	5	00813

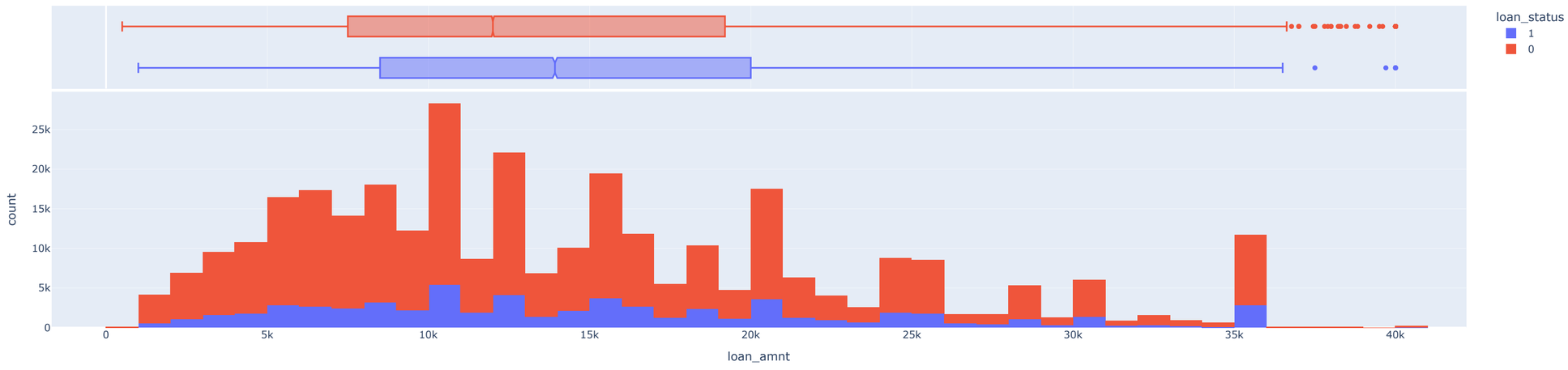
2. 🔍 Exploratory Data Analysis

> **OVERALL GOAL:**
>
> - Get an understanding for which variables are important, view summary statistics, and visualize the data

Numerical features analysis

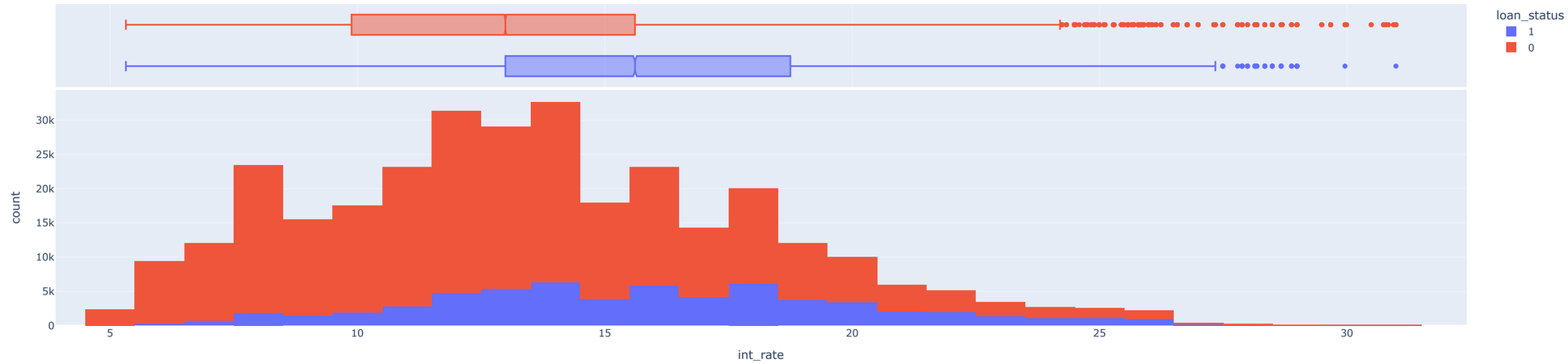
We will first look at the numerical features to gain some understanding on the dataset distribution.

In [12]:
```
fig = px.histogram(train_modified_df, x="loan_amnt", color='loan_status',nbins=50, marginal="box")
fig.show()
```



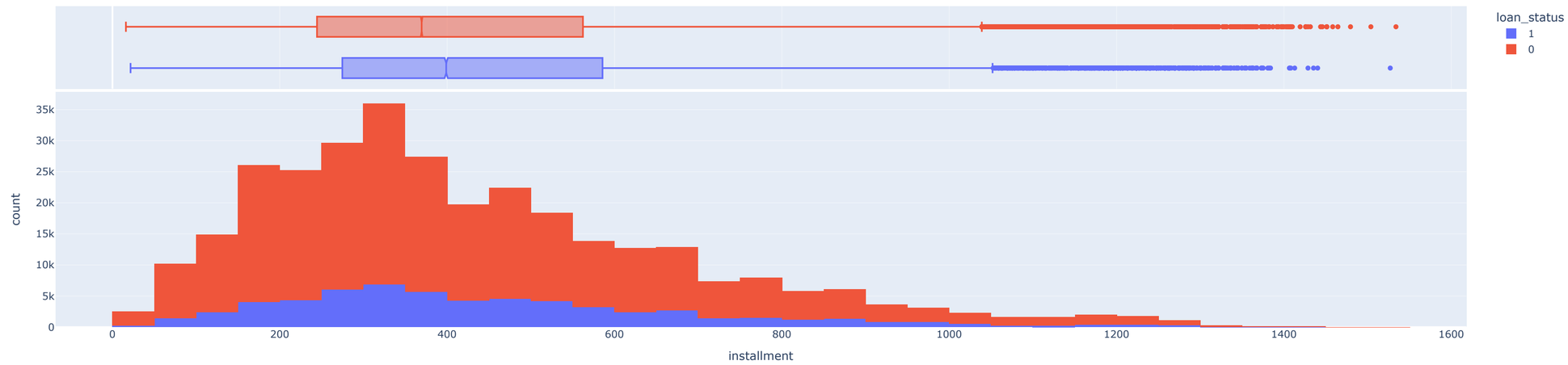
We start observing outliers for loan_amnt around 35k and above.
We also observed a slight right skewed distribution.

In [13]:
```
fig = px.histogram(train_modified_df, x="int_rate", color='loan_status',nbins = 50,marginal="box")
fig.show()
```



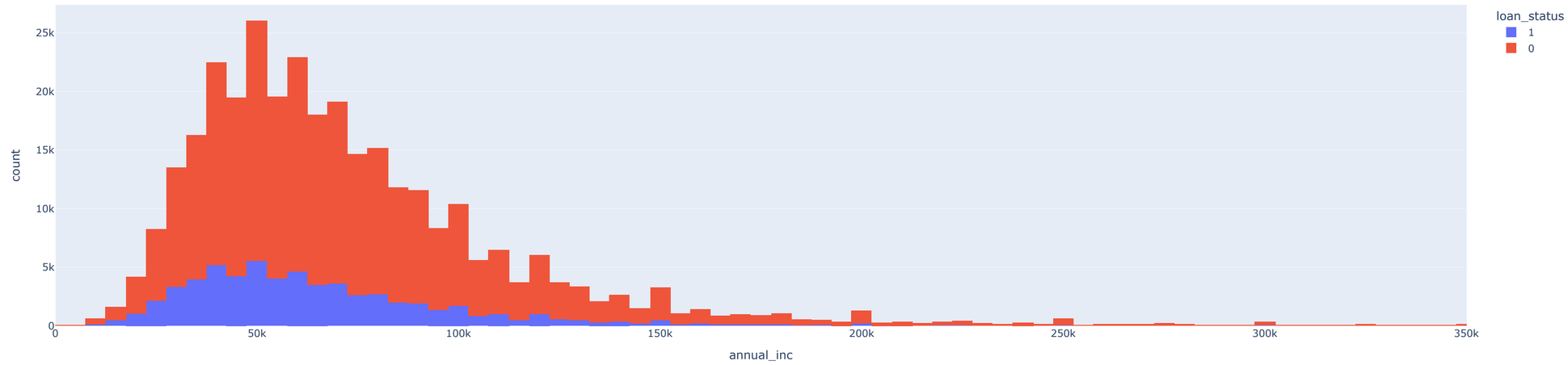
We see a right skewed distribution for loan status.

In [14]:
```
fig = px.histogram(train_modified_df, x="installment", color='loan_status',nbins = 50,marginal="box")
fig.show()
```



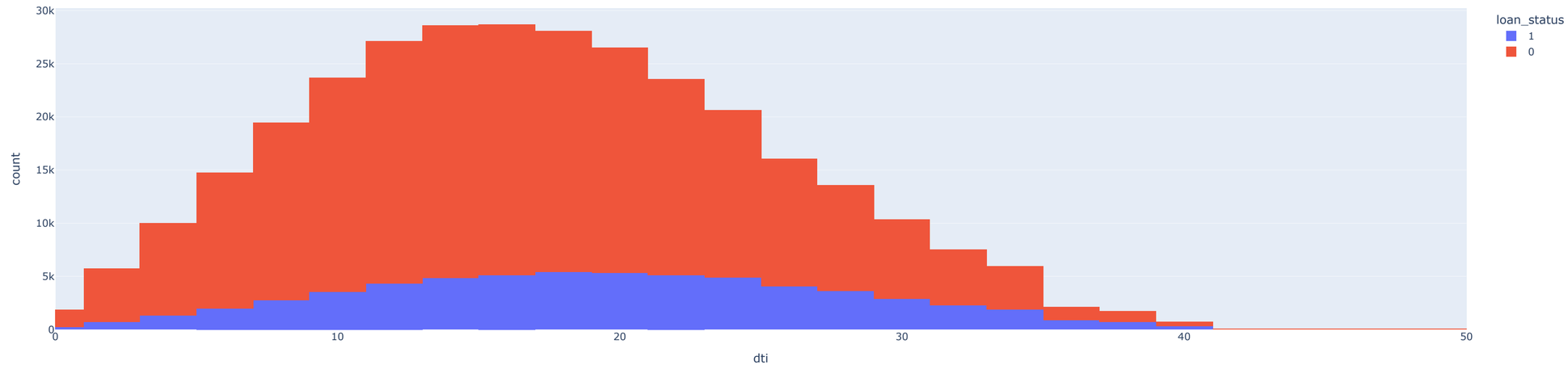
We observed a right skewed distribution for installment.

In [15]:
```
fig = px.histogram(train_modified_df, x="annual_inc", color='loan_status', nbins=2000, range_x=[0,350000])
fig.show()
```



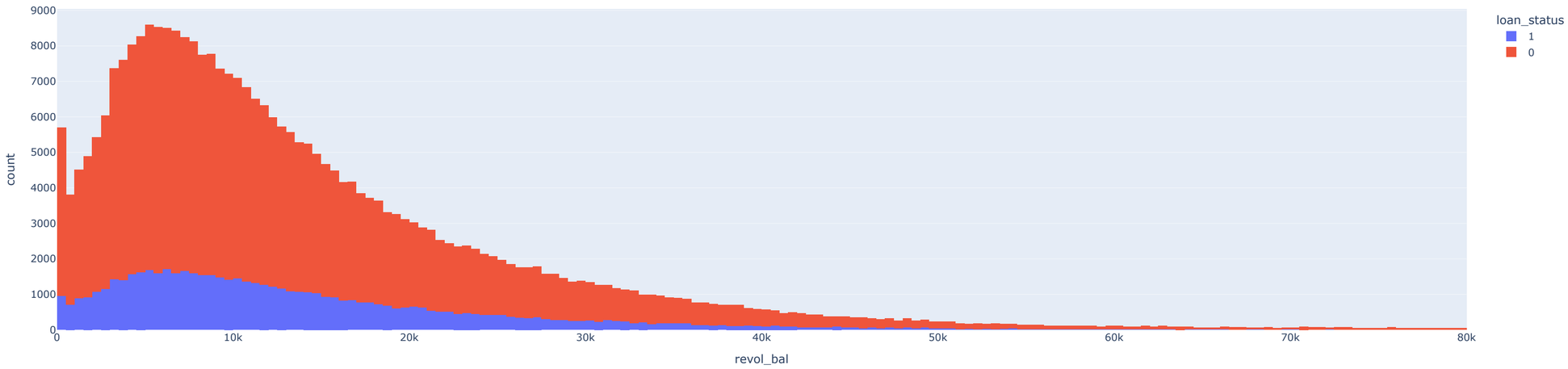
We observed a right skewed distribution for annual income and outliers above 300000.

In [16]:
```
fig = px.histogram(train_modified_df, x="dti", color='loan_status', nbins=5000, range_x=[0,50])
fig.show()
```



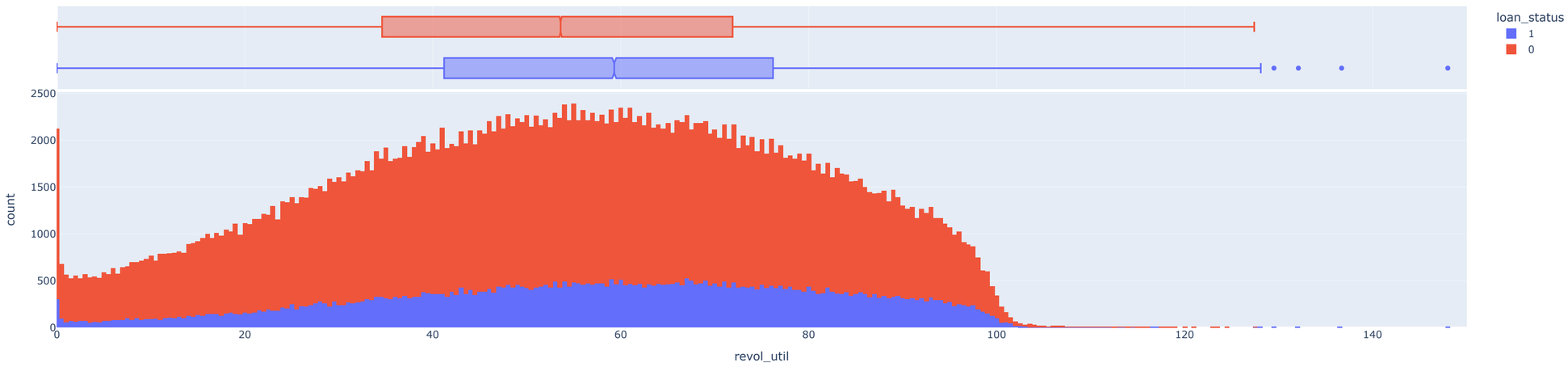
We observed a right skewed distribution for dti and outliers above 50.

In [17]:
```
fig = px.histogram(train_modified_df, x="revol_bal", color='loan_status', range_x=[0,80000])
fig.show()
```

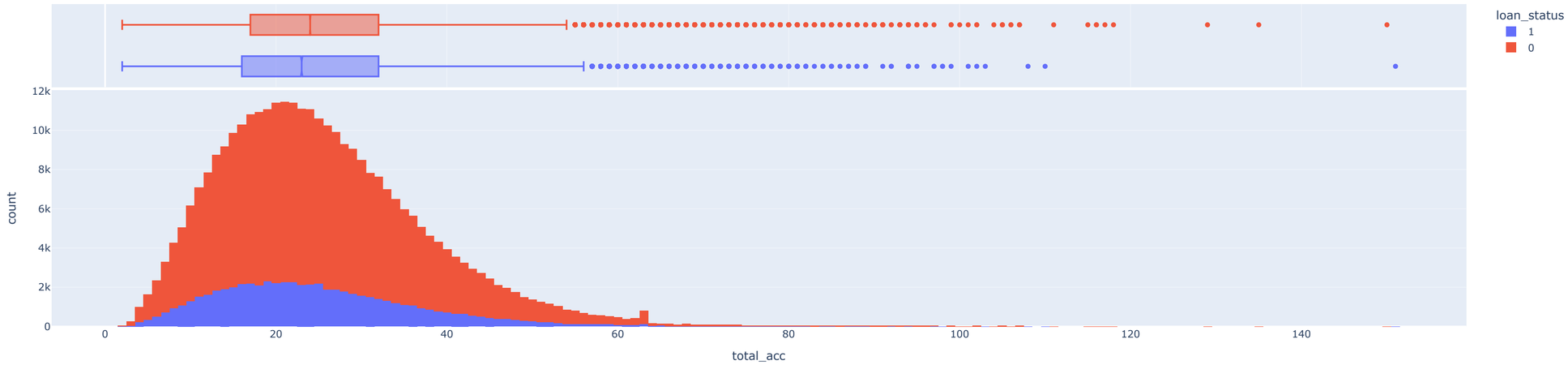


We observed a right skewed distribution for revol_bal.

In [18]:
```
fig = px.histogram(train_modified_df, x="revol_util", color='loan_status', range_x=[0,150], marginal="box")
fig.show()
```

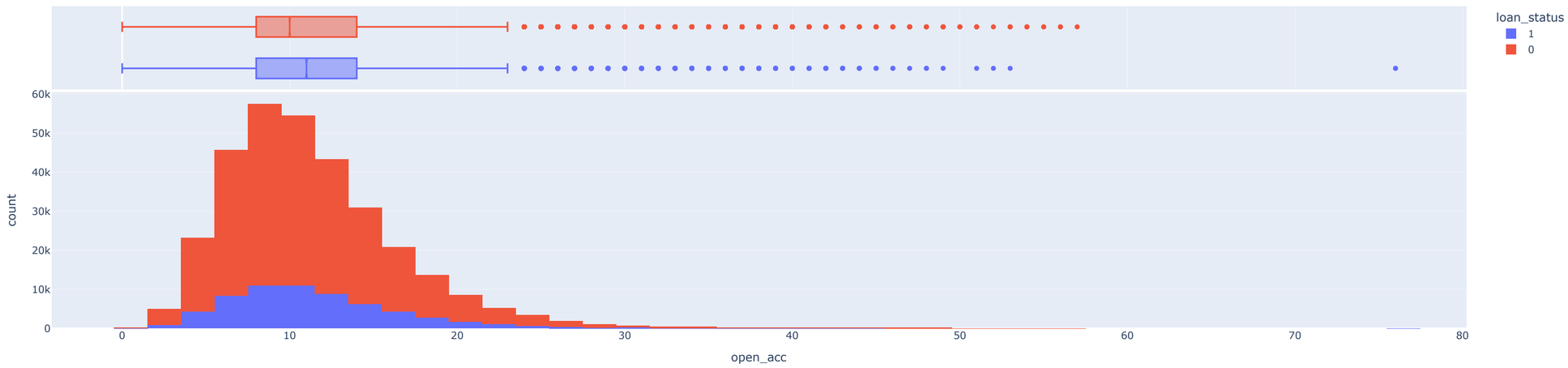


In [19]:
```
fig = px.histogram(train_modified_df, x="total_acc", color='loan_status', marginal='box')
fig.show()
```



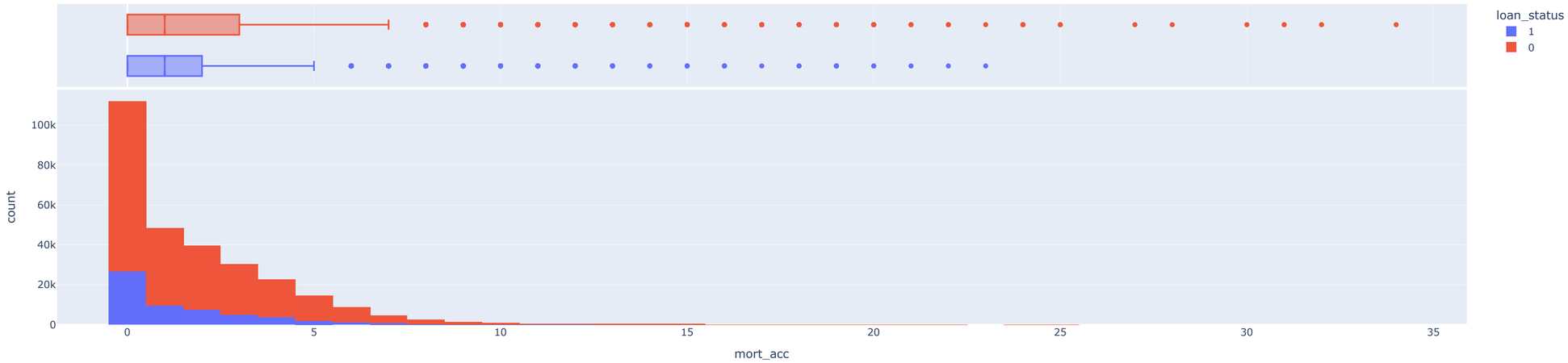
We observed a right skewed distribution for total_acc.

In [20]:
```
fig = px.histogram(train_modified_df, x="open_acc", color='loan_status', nbins=50, marginal="box")
fig.show()
```



We observed a right skewed distribution for open_acc.

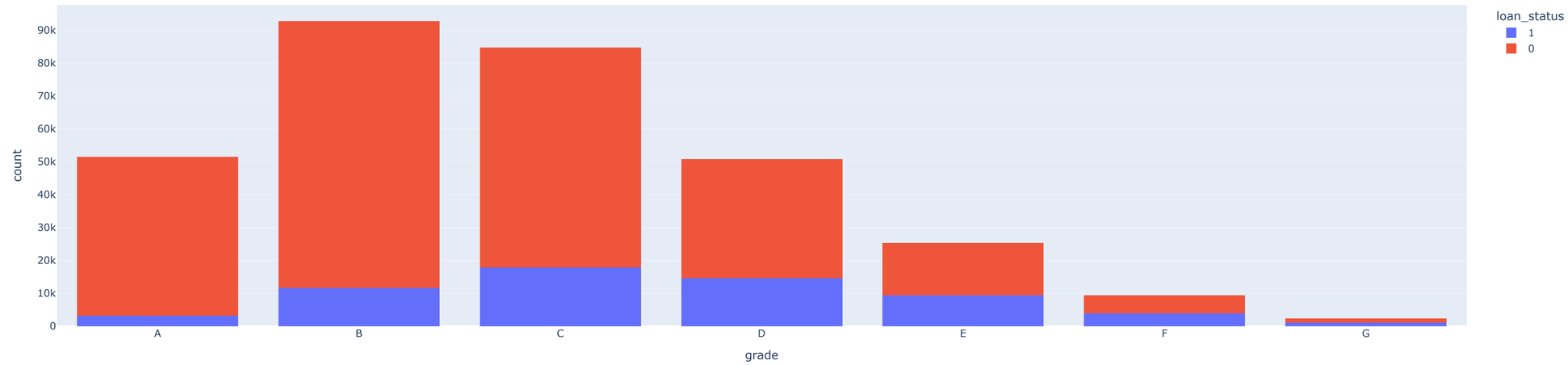
In [21]:
```
fig = px.histogram(train_modified_df, x="mort_acc", color='loan_status', marginal='box')
fig.show()
```



Categorical features analysis

Next, let's look at the categorical features to give us further understanding on the dataset.

In [22]:
```
fig = px.histogram(train_modified_df, x="grade", color='loan_status')
fig.update_xaxes(categoryorder='category ascending')
fig.show()
result=train_modified_df.groupby(['grade','loan_status']).size().unstack()
result.div(result.sum(axis=1),axis=0)
```

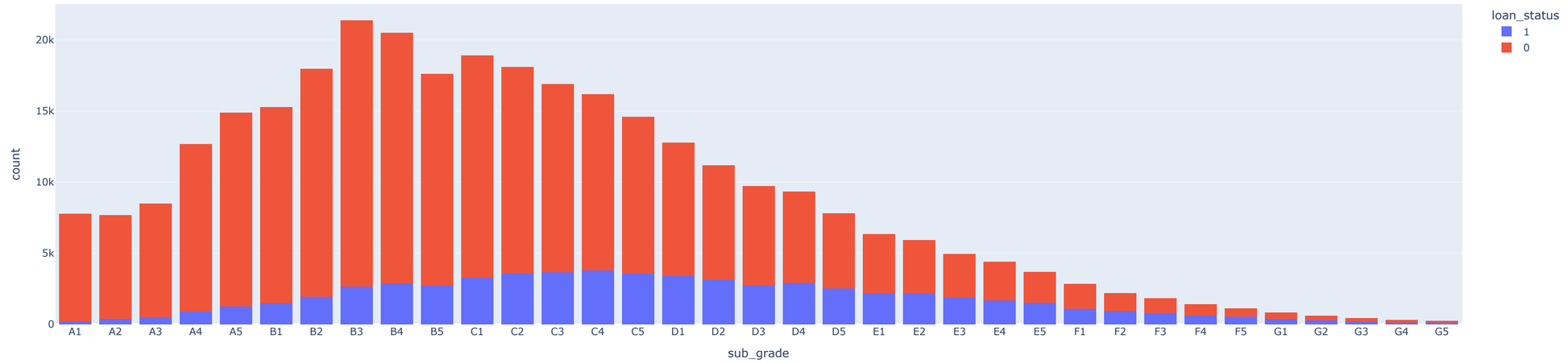


Out[22]:

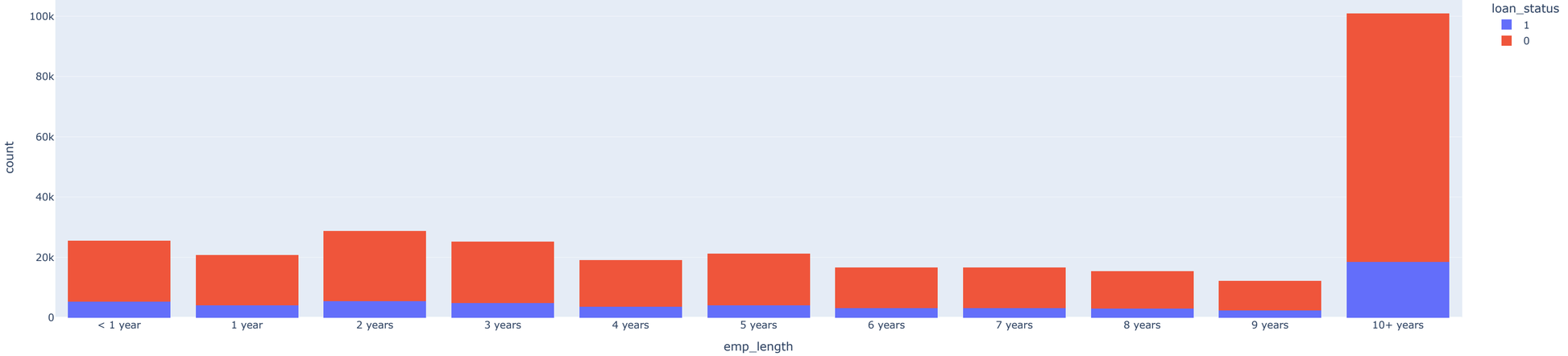
loan_status	0	1
grade		
A	0.937042	0.062958
B	0.873962	0.126038
C	0.788390	0.211610
D	0.709558	0.290442
E	0.625143	0.374857
F	0.572005	0.427995
G	0.521467	0.478533

We observed customers start to default more from grade E onwards.

In [23]:
```
fig = px.histogram(train_modified_df, x="sub_grade", color='loan_status')
fig.update_xaxes(categoryorder='category ascending')
fig.show()
```



In [24]:
```
fig = px.histogram(train_modified_df, x="emp_length", color='loan_status',
                   category_orders={"emp_length": ["< 1 year", "1 year", "2 years", "3 years","4 years","5 years",
                                                   "6 years", "7 years", "8 years", "9 years","10+ years"]})
fig.show()
result=train_modified_df.groupby(['emp_length','loan_status']).size().unstack()
result.div(result.sum(axis=1),axis=0)
```

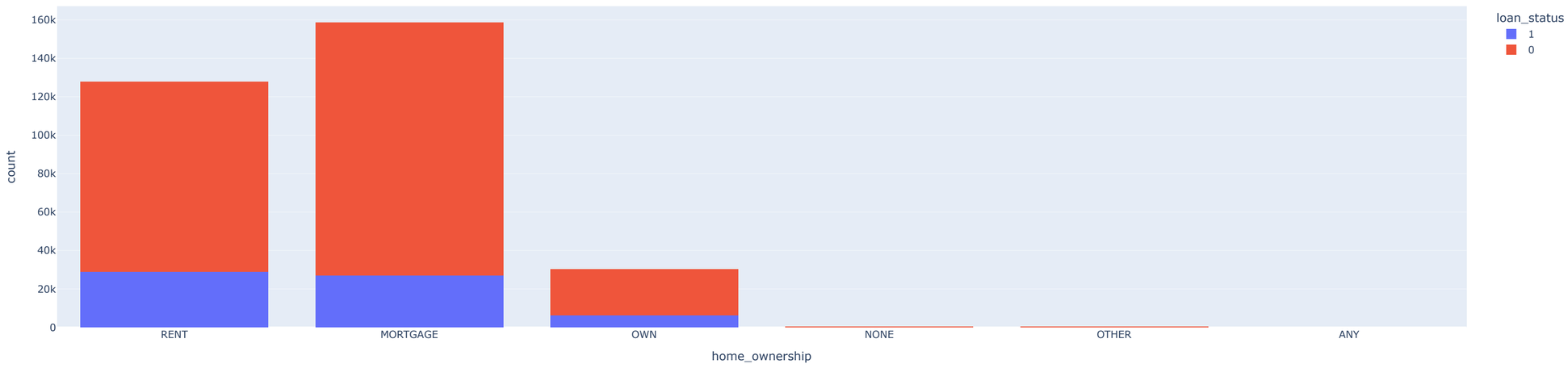


Out[24]:

loan_status	0	1
emp_length		
1 year	0.801701	0.198299
10+ years	0.815671	0.184329
2 years	0.806375	0.193625
3 years	0.804044	0.195956
4 years	0.807365	0.192635
5 years	0.807146	0.192854
6 years	0.809275	0.190725
7 years	0.805602	0.194398
8 years	0.800052	0.199948
9 years	0.798315	0.201685
< 1 year	0.791452	0.208548

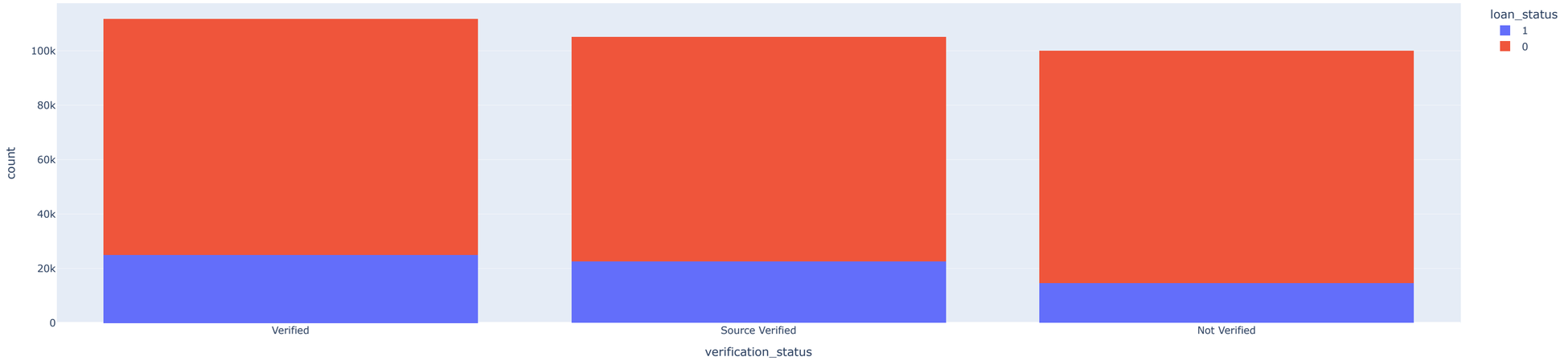
We observed similar distribution for loan defaults across employment length.
We can drop this feature later on.

In [25]:
```
fig = px.histogram(train_modified_df, x="home_ownership", color='loan_status')
fig.show()
```

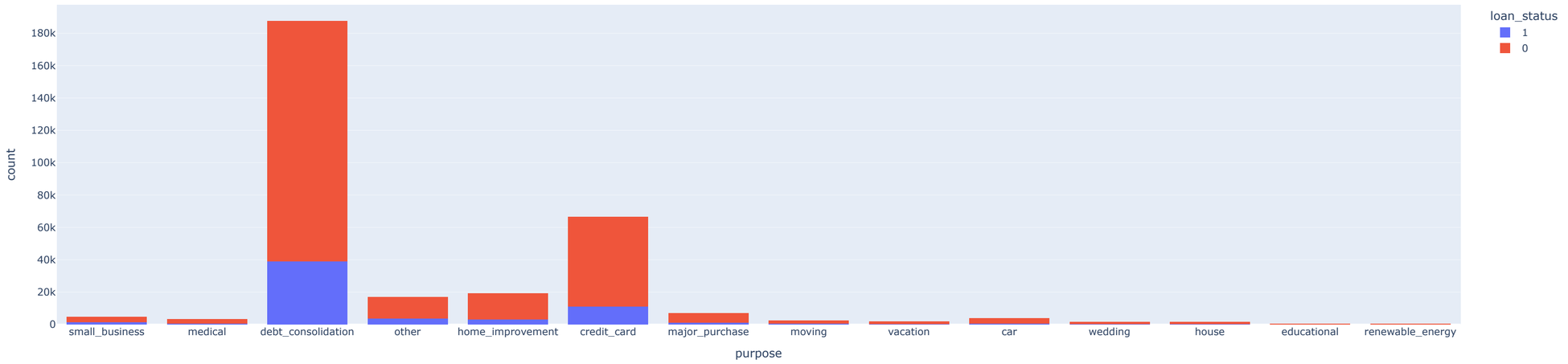


There is limited data for "none", "other" and "any" columns.
We will merge these three categories later on.

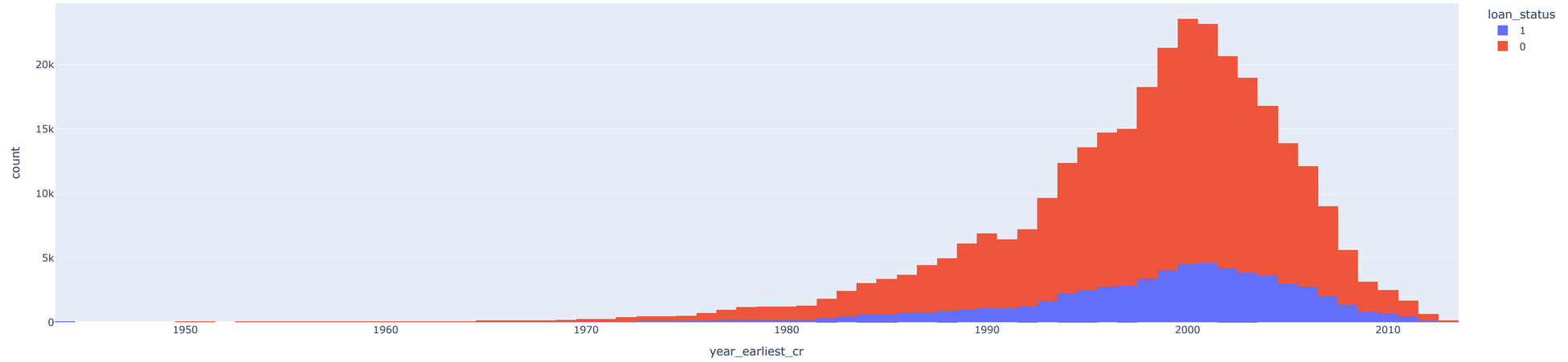
In [26]:
```
fig = px.histogram(train_modified_df, x="verification_status", color='loan_status')
fig.show()
```



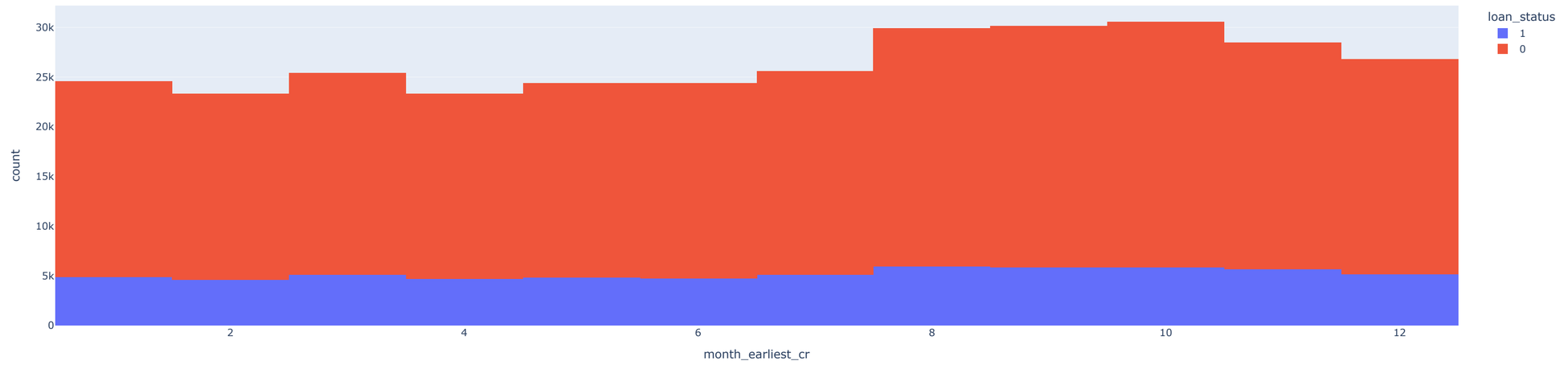
In [27]:
```
fig = px.histogram(train_modified_df, x="purpose", color='loan_status')
fig.show()
```



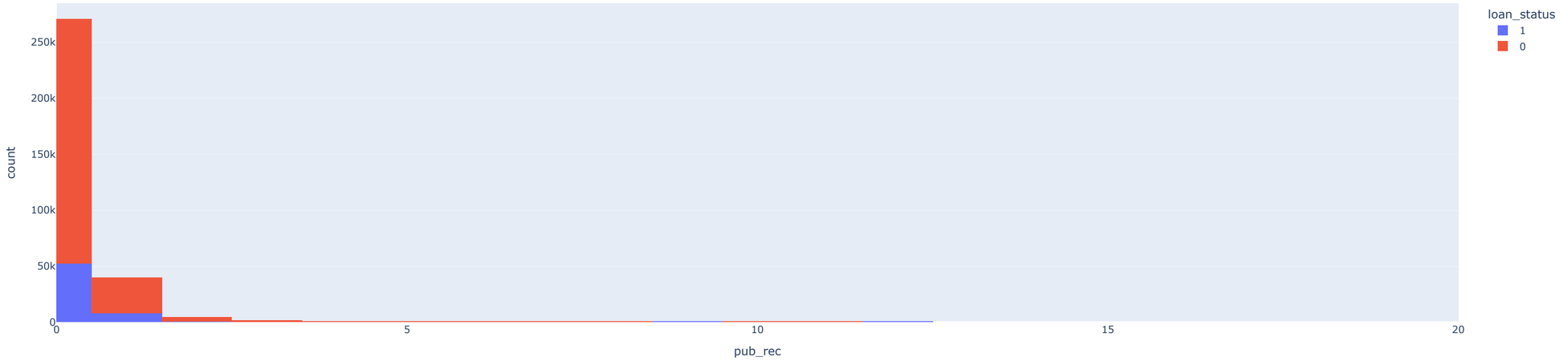
In [28]:
```
fig = px.histogram(train_modified_df, x="year_earliest_cr", color='loan_status')
fig.show()
```



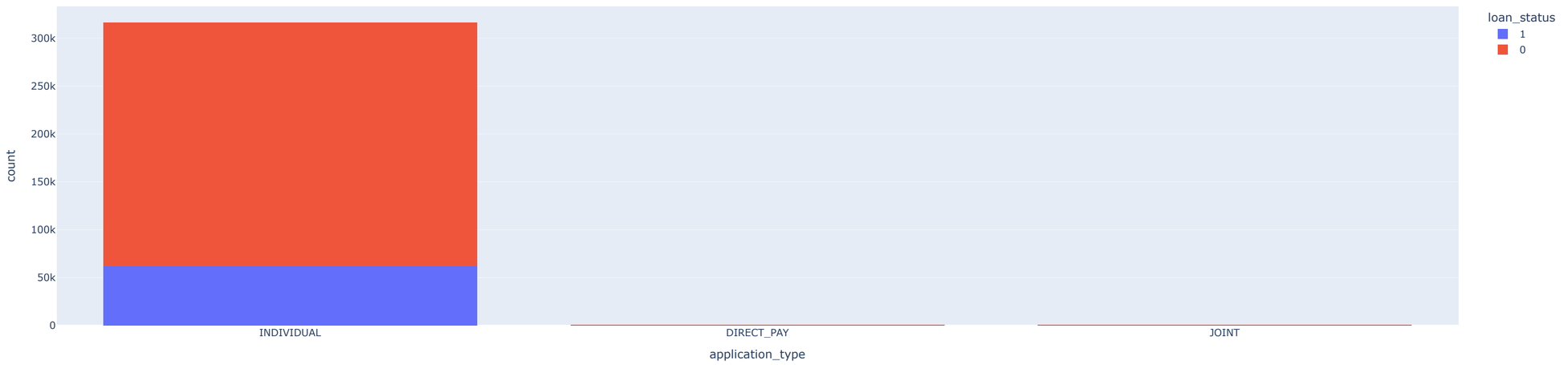
In [29]:
```
fig = px.histogram(train_modified_df, x="month_earliest_cr", color='loan_status')
fig.show()
```



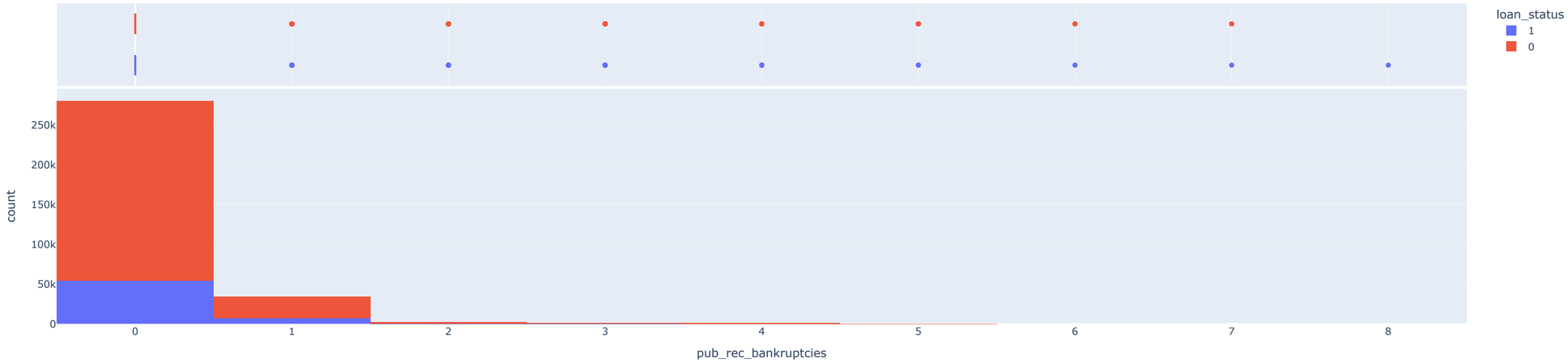
In [30]:
```
fig = px.histogram(train_modified_df, x="pub_rec", color='loan_status', range_x=[0,20])
fig.show()
```



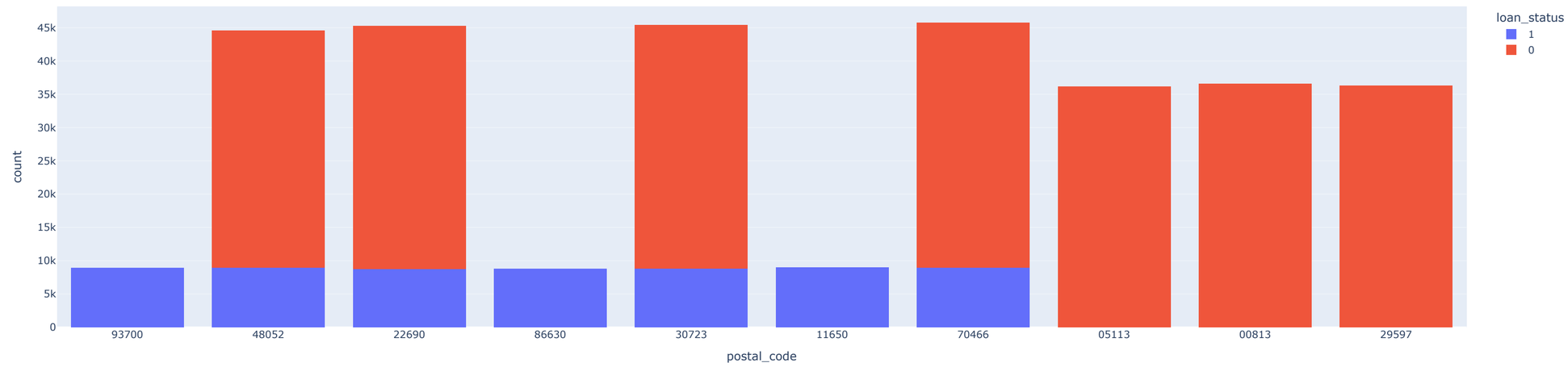
In [31]:
```
fig = px.histogram(train_modified_df, x="application_type", color='loan_status', range_x=[])
fig.show()
```



In [32]:
```
fig = px.histogram(train_modified_df, x="pub_rec_bankruptcies", color='loan_status', marginal='box')
fig.show()
```

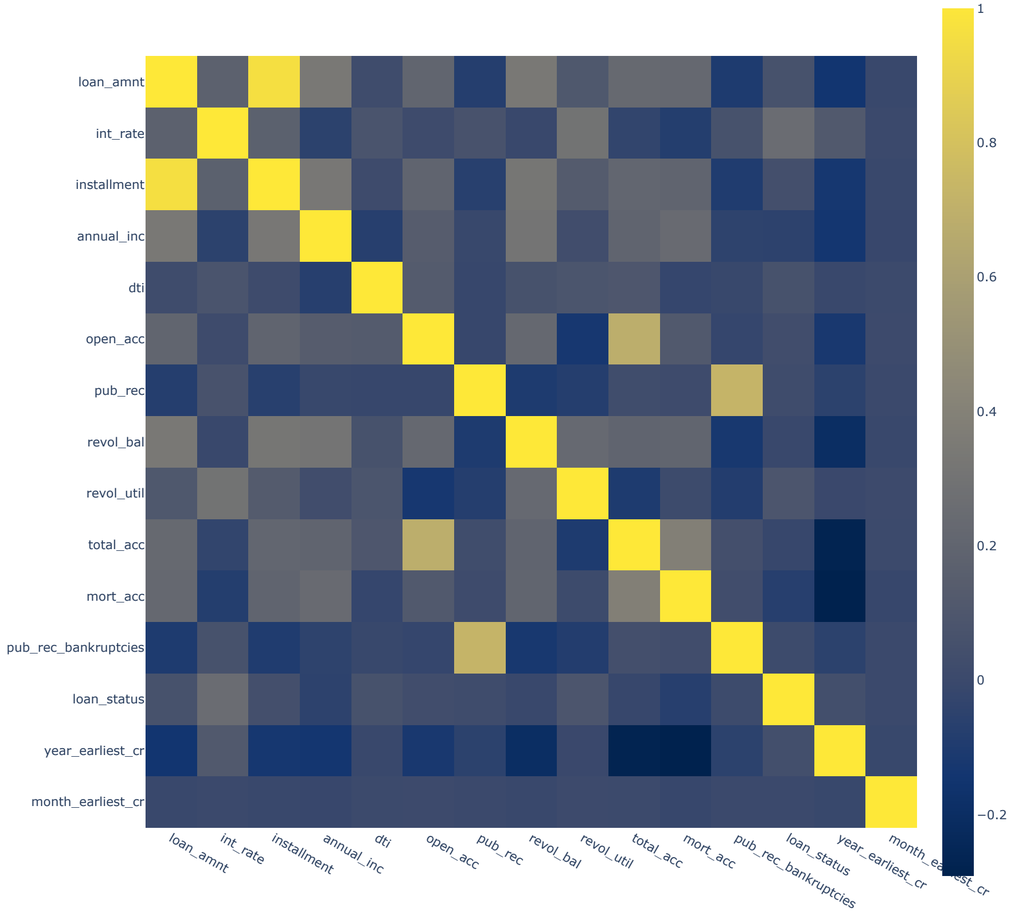


In [33]:
```
fig = px.histogram(train_modified_df, x="postal_code", color='loan_status')
fig.show()
```



Let's take a look at the correlation plot for the numerical features.

In [34]:
```
corr_df = train_modified_df.corr(numeric_only=True)
fig = px.imshow(corr_df,color_continuous_scale='Cividis')
fig.layout.height = 1000
fig.layout.width = 1000
fig.show()
```



We see that loan amount and installment are strongly correlated with one another.

3. 🔄 Feature Engineering

> **OVERALL GOAL:**
>
> - Select relevant features and enhance them to improve the overall performance of the machine learning model

First we convert the text data into useful categorical numerical variable.
We will create a copy of the current dataset so any edit done will be done on the copy dataset and the old dataset will not be modified.

In [35]:
```
prep_data = train_modified_df.copy()
prep_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 316824 entries, 0 to 316823
Data columns (total 26 columns):
 #   Column                Non-Null Count   Dtype
---  ------                --------------   -----
 0   loan_amnt             316824 non-null  float64
 1   term                  316824 non-null  object
 2   int_rate              316824 non-null  float64
 3   installment           316824 non-null  float64
 4   grade                 316824 non-null  object
 5   sub_grade             316824 non-null  object
 6   emp_length            302162 non-null  object
 7   home_ownership        316824 non-null  object
 8   annual_inc            316824 non-null  float64
 9   verification_status   316824 non-null  object
 10  purpose               316824 non-null  object
 11  dti                   316824 non-null  float64
 12  earliest_cr_line      316824 non-null  datetime64[ns]
 13  open_acc              316824 non-null  float64
 14  pub_rec               316824 non-null  float64
 15  revol_bal             316824 non-null  float64
 16  revol_util            316598 non-null  float64
 17  total_acc             316824 non-null  float64
 18  initial_list_status   316824 non-null  object
 19  application_type      316824 non-null  object
 20  mort_acc              286611 non-null  float64
 21  pub_rec_bankruptcies  316388 non-null  float64
 22  loan_status           316824 non-null  int64
 23  year_earliest_cr      316824 non-null  int32
 24  month_earliest_cr     316824 non-null  int32
 25  postal_code           316824 non-null  object
dtypes: datetime64[ns](1), float64(12), int32(2), int64(1), object(10)
memory usage: 60.4+ MB
```

We will convert the columns containing datetime from integer to a string

In [36]:
```
prep_data['year_earliest_cr'] = prep_data['year_earliest_cr'].astype(str)
prep_data['month_earliest_cr'] = prep_data['month_earliest_cr'].astype(str)
prep_data.head()
```

Out[36]:

	loan_amnt	term	int_rate	installment	grade	sub_grade	emp_length	home_ownership	annual_inc	verification_status	purpose	dti	earliest_cr_line	open_acc	pub_rec	revol_bal	revol_util	total_acc	initial_list_status	application_type	mort_acc	pub_rec_bankruptcies	loan_status	year_earliest_cr	month_earliest_cr	postal_code
0	25000.0	60 months	14.83	592.52	D	D3	10+ years	RENT	109000.0	Verified	small_business	13.90	1996-08-01	11.0	0.0	6390.0	41.5	39.0	f	INDIVIDUAL	NaN	0.0	1	1996	8	93700
1	9500.0	36 months	12.99	320.05	C	C2	NaN	MORTGAGE	40000.0	Verified	medical	22.29	1985-10-01	6.0	1.0	62512.0	82.6	20.0	f	INDIVIDUAL	2.0	1.0	1	1985	10	48052
2	9000.0	36 months	8.39	283.65	B	B1	10+ years	MORTGAGE	50000.0	Not Verified	debt_consolidation	15.27	1996-07-01	14.0	0.0	8835.0	23.2	20.0	w	INDIVIDUAL	2.0	0.0	0	1996	7	05113
3	16700.0	60 months	22.99	470.69	F	F1	7 years	MORTGAGE	68000.0	Verified	debt_consolidation	21.92	2005-09-01	13.0	0.0	23489.0	55.3	26.0	f	INDIVIDUAL	2.0	0.0	0	2005	9	00813
4	2800.0	36 months	15.80	98.17	C	C3	9 years	MORTGAGE	218554.0	Verified	home_improvement	14.93	1986-05-01	20.0	0.0	33014.0	90.7	44.0	w	INDIVIDUAL	7.0	0.0	0	1986	5	00813

Next, we will narrow down the catergories in the homeownership column.

In [37]:
```
def simplify_ownership(home_ownership):
    if home_ownership in  ['NONE','ANY', None ]:
        return 'OTHER'
    else:
        return home_ownership

prep_data['home_ownership'] = prep_data['home_ownership'].apply(simplify_ownership)
```

3a. Peforming Feature Engineering on numerical variables

We will:

1. Put a cap on the outliers in our numerical variables
2. Impute the NaNs with either the columns' mean or median
3. Perform log transformation on the variables to ensure a normal distribution

In [38]:
```
numerical_prep_data = prep_data.select_dtypes(exclude = [object])
numerical_prep_data = numerical_prep_data.drop(columns = ['loan_status','earliest_cr_line'])
```

In [39]:
```
def cap_outliers (column):
    std = np.std(column)
    mean = np.mean(column)
    threshold = 3 * std
    lower_bound = mean - threshold
    upper_bound = mean + threshold

    #cap the outliers
    column[column < lower_bound] = lower_bound
    column[column > upper_bound] = upper_bound
    return column

modified_numerical_prep_data = numerical_prep_data.copy()
modified_numerical_prep_data = modified_numerical_prep_data.apply(cap_outliers, axis = 0)
modified_numerical_prep_data.isnull().sum(axis=0)
```

Out[39]:
```
loan_amnt                 0
int_rate                  0
installment               0
annual_inc                0
dti                       0
open_acc                  0
pub_rec                   0
revol_bal                 0
revol_util              226
total_acc                 0
mort_acc              30213
pub_rec_bankruptcies    436
dtype: int64
```

In [40]:
```
imputed_numerical_prep_data = modified_numerical_prep_data.copy()

#Imputing NaN with mean value
imputed_numerical_prep_data['revol_util'] = imputed_numerical_prep_data['revol_util'].fillna(
    imputed_numerical_prep_data['revol_util'].mean())

#Imputing NaN with median value
imputed_numerical_prep_data['mort_acc'] = imputed_numerical_prep_data['mort_acc'].fillna(
    imputed_numerical_prep_data['mort_acc'].median())

imputed_numerical_prep_data['pub_rec_bankruptcies'] = imputed_numerical_prep_data['pub_rec_bankruptcies'].fillna(
    imputed_numerical_prep_data['pub_rec_bankruptcies'].median())

imputed_numerical_prep_data.isnull().sum()
```

Out[40]:
```
loan_amnt             0
int_rate              0
installment           0
annual_inc            0
dti                   0
open_acc              0
pub_rec               0
revol_bal             0
revol_util            0
total_acc             0
mort_acc              0
pub_rec_bankruptcies  0
dtype: int64
```

In [41]:
```
#Log transformation on data
constant = 1
positive_imputed_numerical_prep_data = imputed_numerical_prep_data + constant

log_data = np.log(positive_imputed_numerical_prep_data)
log_data.describe().T
```

Out[41]:

	count	mean	std	min	25%	50%	75%	max
loan_amnt	316824.0	9.354637	0.680420	6.216606	8.987322	9.392745	9.903538	10.576650
int_rate	316824.0	2.635221	0.316446	1.843719	2.441477	2.662355	2.865054	3.334294
installment	316824.0	5.888099	0.640132	2.837908	5.526757	5.930732	6.342315	7.077780
annual_inc	316824.0	11.064682	0.515073	0.000000	10.714440	11.066654	11.407576	12.471835
dti	316824.0	2.784104	0.552588	0.000000	2.508786	2.884801	3.176386	4.351256
open_acc	316824.0	2.426294	0.406999	0.000000	2.197225	2.397895	2.708050	3.322319
pub_rec	316824.0	0.107364	0.262933	0.000000	0.000000	0.000000	0.000000	1.000421
revol_bal	316824.0	9.181609	1.213891	0.000000	8.704834	9.322061	9.883859	11.251848
revol_util	316824.0	3.836211	0.712506	0.000000	3.605498	4.021774	4.302713	4.853286
total_acc	316824.0	3.167915	0.474132	1.098612	2.890372	3.218876	3.496508	4.128487
mort_acc	316824.0	0.763452	0.677489	0.000000	0.000000	0.693147	1.386294	2.224538
pub_rec_bankruptcies	316824.0	0.079455	0.221767	0.000000	0.000000	0.000000	0.000000	0.783311

3b. Peforming Feature Engineering on categorical variables

We will perform one hot encoding on the categorical variables

In [42]:
```
categorical_prep_data = prep_data.select_dtypes(include = [object])
categorical_prep_data.head()
```

Out[42]:

	term	grade	sub_grade	emp_length	home_ownership	verification_status	purpose	initial_list_status	application_type	year_earliest_cr	month_earliest_cr	postal_code
0	60 months	D	D3	10+ years	RENT	Verified	small_business	f	INDIVIDUAL	1996	8	93700
1	36 months	C	C2	NaN	MORTGAGE	Verified	medical	f	INDIVIDUAL	1985	10	48052
2	36 months	B	B1	10+ years	MORTGAGE	Not Verified	debt_consolidation	w	INDIVIDUAL	1996	7	05113
3	60 months	F	F1	7 years	MORTGAGE	Verified	debt_consolidation	f	INDIVIDUAL	2005	9	00813
4	36 months	C	C3	9 years	MORTGAGE	Verified	home_improvement	w	INDIVIDUAL	1986	5	00813

In [43]:
```
#Perform one hot encoding
OH_encoding = pd.get_dummies(categorical_prep_data, columns=['term','home_ownership','verification_status','purpose','initial_list_status',
```

```
                                    'application_type','postal_code',"grade","sub_grade",'year_earliest_cr','month_earliest_cr'])
encoded_data = OH_encoding.drop(columns = ['emp_length','term_ 60 months'])

encoded_data
```

Out[43]:

	term_ 36 months	home_ownership_MORTGAGE	home_ownership_OTHER	home_ownership_OWN	home_ownership_RENT	verification_status_Not Verified	verification_status_Source Verified	verification_status_Verified	purpose_car	purpose_credit_card	purpose_debt_consolidation	purpose_educational	purpose_home_improvement	purpose_house	purpose_major_purch
0	False	False	False	False	True	False	False	True	False	False	False	False	False	False	F
1	True	True	False	False	False	False	False	True	False	False	False	False	False	False	F
2	True	True	False	False	False	True	False	False	False	False	True	False	False	False	F
3	False	True	False	False	False	False	False	True	False	False	True	False	False	False	F
4	True	True	False	False	False	False	False	True	False	False	False	False	True	False	F
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
316819	True	True	False	False	False	False	False	True	False	False	True	False	False	False	F
316820	True	True	False	False	False	False	False	True	False	False	True	False	False	False	F
316821	True	True	False	False	False	False	True	False	False	True	False	False	False	False	F
316822	True	False	False	False	True	True	False	False	False	False	False	False	False	False	F
316823	False	False	False	False	True	True	False	False	False	False	True	False	False	False	F

316824 rows × 155 columns

3c. Concatenate of numerical and categorical dataset

Now we will concatenate the numerical and categorical dataset together to train our machine learning model.

In [44]:
```
print(len(log_data))
print(len(encoded_data))
print(len(prep_data))
concat_data = pd.concat([log_data,encoded_data,prep_data['loan_status']],axis=1)
final_train_df = concat_data
final_train_df.shape
```

```
316824
316824
316824
```

Out[44]: (316824, 168)

We will perform a final check to ensure there are no NaNs in our dataset.

In [45]:
```
final_train_df.isnull().sum(axis = 0)
```

Out[45]:
```
loan_amnt             0
int_rate              0
installment           0
annual_inc            0
dti                   0
                     ..
month_earliest_cr_6   0
month_earliest_cr_7   0
month_earliest_cr_8   0
month_earliest_cr_9   0
loan_status           0
Length: 168, dtype: int64
```

3d. Correlation plot of dataset

We will create a correlation plot for the dataset and drop features that have very low correlation to our loan status.

In [46]:
```
corr_df = final_train_df.corr(method='pearson', numeric_only=True)
corr_df.tail(1)
```

Out[46]:

	loan_amnt	int_rate	installment	annual_inc	dti	open_acc	pub_rec	revol_bal	revol_util	total_acc	mort_acc	pub_rec_bankruptcies	term_ 36 months	home_ownership_MORTGAGE	home_ownership_OTHER	home_ownership_OWN	home_ownership_RENT	verification_status_Not Verified	verification_status_Source Verified	verification_status_Verified	pu
loan_status	0.061003	0.243028	0.050544	-0.085288	0.114579	0.026334	0.018072	0.006444	0.074625	-0.021226	-0.076309	0.007445	-0.172643	-0.068055	-0.00139	0.009006	0.064015	-0.085327	0.033546	0.049965	

From the correlation plot, we observed that data from "earliest_cr_line" has very little correlation, so let's drop that feature out.

In [47]:
```
final_train_df = final_train_df[final_train_df.columns.drop(list(final_train_df.filter(regex='earliest')))]
final_train_df
```

Out[47]:

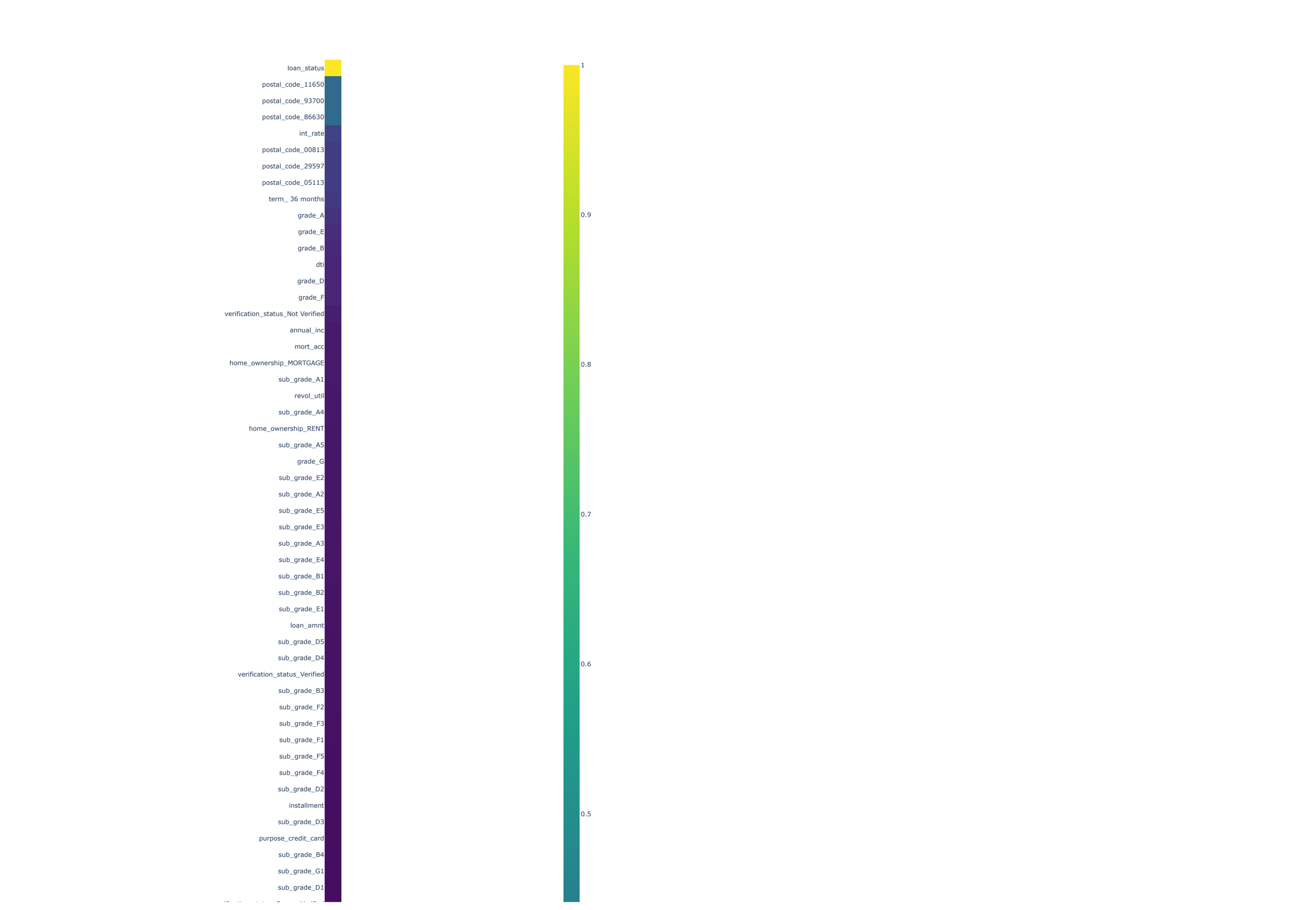
	loan_amnt	int_rate	installment	annual_inc	dti	open_acc	pub_rec	revol_bal	revol_util	total_acc	mort_acc	pub_rec_bankruptcies	term_ 36 months	home_ownership_MORTGAGE	home_ownership_OTHER	home_ownership_OWN	home_ownership_RENT	verification_status_Not Verified	verification_status_Source Verified	verification_status_Verified	purpose_
0	10.126671	2.761907	6.386071	11.599112	2.701361	2.484907	0.000000	8.762646	3.749504	3.688879	0.693147	0.000000	False	False	False	False	True	False	False	True	F
1	9.159152	2.638343	5.771597	10.596660	3.148024	1.945910	0.693147	11.043130	4.426044	3.044522	1.098612	0.693147	True	True	False	False	False	False	False	True	F
2	9.105091	2.239645	5.651260	10.819798	2.789323	2.708050	0.000000	9.086590	3.186353	3.044522	1.098612	0.000000	True	True	False	False	False	True	False	False	F
3	9.723224	3.177637	6.156322	11.127278	3.132010	2.639057	0.000000	10.064330	4.030695	3.295837	1.098612	0.000000	False	True	False	False	False	False	False	True	F
4	7.937732	2.821379	4.596836	12.294793	2.768204	3.044522	0.000000	10.404717	4.518522	3.806662	2.079442	0.000000	True	True	False	False	False	False	False	True	F
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
316819	9.903538	2.184927	6.440500	11.156265	3.241811	2.484907	0.693147	8.973225	3.100092	3.295837	1.098612	0.693147	True	True	False	False	False	False	False	True	F
316820	9.425532	2.685805	6.046947	11.002117	2.965273	2.197225	0.693147	9.694617	4.080922	2.639057	0.000000	0.693147	True	True	False	False	False	False	False	True	F
316821	10.308986	2.616666	6.915078	11.289794	2.342767	2.833213	0.693147	9.568085	4.207673	3.367296	1.386294	0.000000	True	True	False	False	False	False	True	False	F
316822	9.546884	2.728506	6.177114	11.156265	1.297463	1.791759	0.000000	8.771060	3.953165	2.397895	0.000000	0.000000	True	False	False	False	True	True	False	False	F
316823	9.392745	2.463853	5.562296	11.289794	3.123246	1.945910	0.000000	9.774176	3.799974	2.890372	0.693147	0.000000	False	False	False	False	True	True	False	False	F

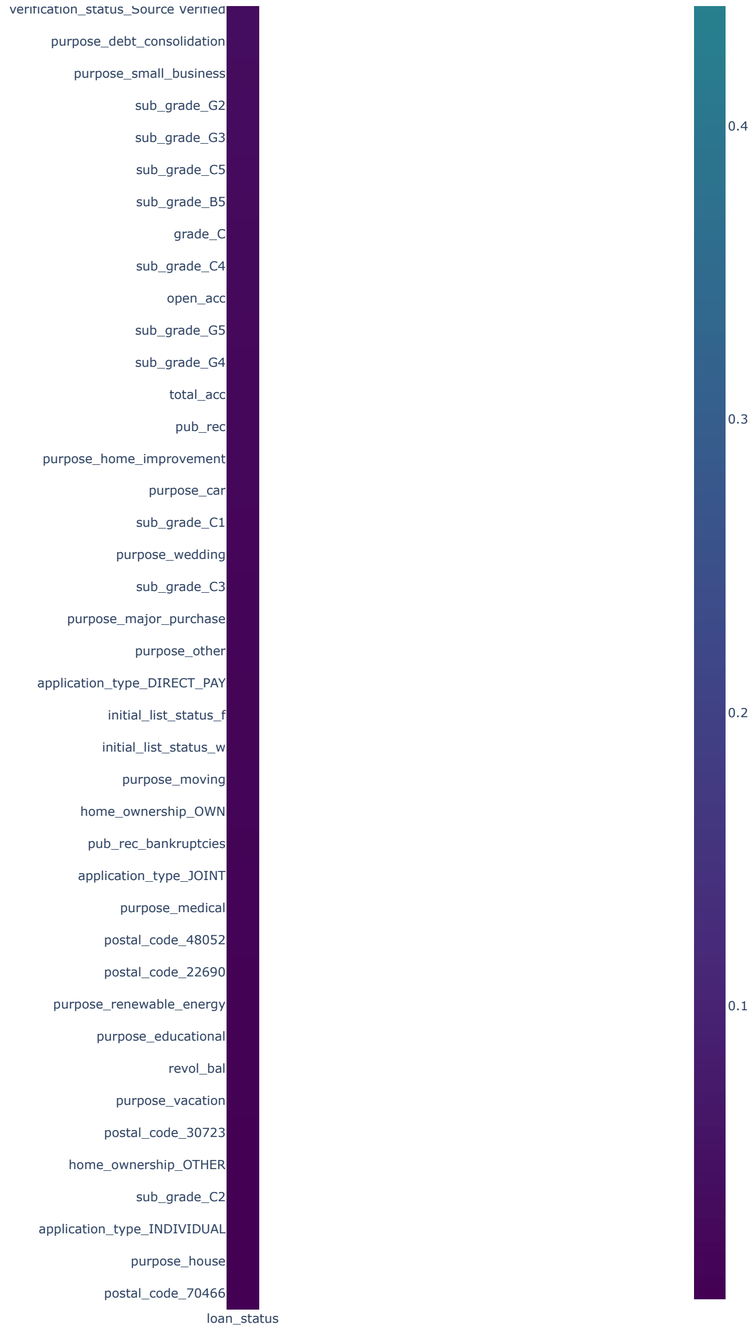
316824 rows × 92 columns

Identifying the top correlated features

We can plot a correlation map and sort the values by highest with respect to loan status to identify the features we can use for teaching our machine learning model.

In [48]:
```
corr = final_train_df.corr(method='kendall')[['loan_status']].abs().sort_values(by='loan_status',ascending=False)
fig = px.imshow(corr,color_continuous_scale='viridis')
fig.layout.height = 3000
fig.layout.width = 1000
fig.show()
```


loan_status
postal_code_11650
postal_code_93700
postal_code_86630
int_rate
postal_code_00813
postal_code_29597
postal_code_05113
term_ 36 months
grade_A
grade_E
grade_B
dti
grade_D
grade_F
verification_status_Not Verified
annual_inc
mort_acc
home_ownership_MORTGAGE
sub_grade_A1
revol_util
sub_grade_A4
home_ownership_RENT
sub_grade_A5
grade_G
sub_grade_E2
sub_grade_A2
sub_grade_E5
sub_grade_E3
sub_grade_A3
sub_grade_E4
sub_grade_B1
sub_grade_B2
sub_grade_E1
loan_amnt
sub_grade_D5
sub_grade_D4
verification_status_Verified
sub_grade_B3
sub_grade_F2
sub_grade_F3
sub_grade_F1
sub_grade_F5
sub_grade_F4
sub_grade_D2
installment
sub_grade_D3
purpose_credit_card
sub_grade_B4
sub_grade_G1
sub_grade_D1
1
0.9
0.8
0.7
0.6
0.5



We can utilize sklearn feature selection to select the top 10 features with respect to our selected label.

In [49]:
```
X = final_train_df.drop('loan_status', axis=1)
y = final_train_df['loan_status']
selector = SelectKBest(score_func=f_classif, k=10)
print("Original feature shape:", X.shape)
new_X = selector.fit_transform(X, y)
selected_indices = selector.get_support(indices=True)
selected_features = X.columns[selected_indices]
print("Transformed feature shape:", new_X.shape)
print(selected_features)
```

```
Original feature shape: (316824, 91)
Transformed feature shape: (316824, 10)
Index(['int_rate', 'term_ 36 months', 'postal_code_00813', 'postal_code_05113',
       'postal_code_11650', 'postal_code_29597', 'postal_code_86630',
       'postal_code_93700', 'grade_A', 'grade_E'],
      dtype='object')
```

From sklearn feature selection, we can see that the top 10 features are: 'int_rate', 'term_ 36 months', 'postal_code_00813', 'postal_code_05113','postal_code_11650', 'postal_code_29597'postal_code_86630','postal_code_93700', 'grade_A', 'grade_E'.

4. Models Building and Comparing Model Perforamance

> **OVERALL GOAL:**
>
> - Building a machine learning model that is capable of generating predictions

4a. Logistic Regression (baseline model without any feature engineering)

Here we start off by creating a baseline model whhich you can use to compare against

Selecting the feature columns
We select the features we want to use in predicting our outcome

In [50]:
```
X = train_df[['loan_amnt','installment','dti']]
y = train_df['loan_status']
```

Import the models of your choice
We select the features we want to use in predicting our outcome

In [51]:
```
logreg_model = LogisticRegression(random_state=0)
```

Train-Test Split We split the data to facilitate the evaluation of the model

In [52]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=0)
```

Evaluate your model
We generate a logistic regression model by fitting the training data using `.fit()`, and thereafter generate predictions using `.predict()`

In [53]:
```
logreg_model = logreg_model.fit(X_train, y_train)
y_pred_proba = logreg_model.predict_proba(X_test)[:,1]
```

We evaluate the model's AUC using `metrics.roc_auc_score()`

In [54]:
```
print('AUC:', metrics.roc_auc_score(y_test, y_pred_proba))
```

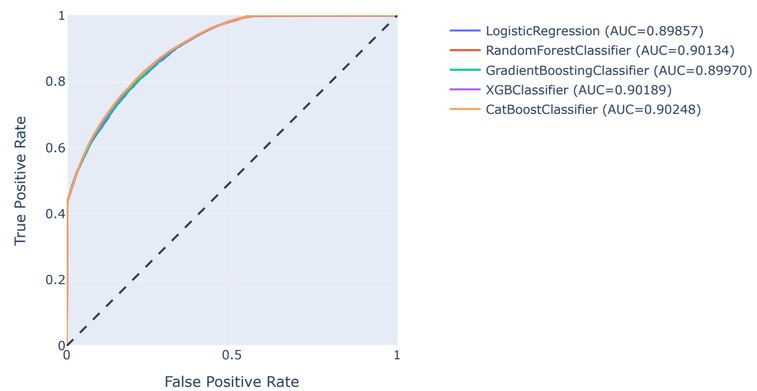
```
AUC: 0.6078044899026738
```

4b. Machine Model Selection (models with feature engineering)

Now we choose the top features we identified previously and use it to train the different learning models which we can use to compare against one another

In [55]:
```
# Features identified to train models
X = final_train_df[['postal_code_11650','postal_code_86630','postal_code_00813','postal_code_29597','postal_code_05113',
                    'postal_code_93700','term_ 36 months',
                    'grade_A','grade_E','int_rate']]
y = final_train_df['loan_status']

# Split data to facilitate the evaluation of models
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=0)

# Convert Features with Scaler
scaler = MinMaxScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Instantiate the classfiers and make a list
models = [LogisticRegression(random_state=5),
          RandomForestClassifier(random_state=5),
          GradientBoostingClassifier(random_state=5),
          xgb.XGBClassifier(random_state=5,verbosity=0),
          CatBoostClassifier(random_seed=5,verbose=0)
         ]

# Plot the ROC curves
fig = go.Figure()
fig.add_shape(
    type='line', line=dict(dash='dash'),
    x0=0, x1=1, y0=0, y1=1
)

#Loop iterates through model lists
for model in models:
    model.fit(X_train_scaled, y_train)
    y_pred = model.predict(X_test_scaled)
    y_pred_proba = model.predict_proba(X_test_scaled)[:,1]

    fpr, tpr, thresholds = roc_curve(y_test, y_pred_proba)
```

```
    auc_score = roc_auc_score(y_test, y_pred_proba)

    model_name = model.__class__.__name__
    name = f"{model_name} (AUC={auc_score:.5f})"
    fig.add_trace(go.Scatter(x=fpr, y=tpr, name=name, mode='lines'))

fig.update_layout(
    xaxis_title='False Positive Rate',
    yaxis_title='True Positive Rate',
    yaxis=dict(scaleanchor="x", scaleratio=1),
    xaxis=dict(constrain='domain'),
    width=800, height=500,
)
fig.update_yaxes(range= [0, 1],constrain='domain')
fig.update_xaxes(range= [0, 1],constrain='domain')
fig.show()
```



Catboostclassifier returned the highest AUC value among the machine learning models.
We will proceed to use the Catboostclassifier for our subsequent training.

In [56]:
```
# Features identified to train models
X = final_train_df[['postal_code_11650','postal_code_86630','postal_code_00813','postal_code_29597','postal_code_05113',
                    'postal_code_93700','term_ 36 months',
                    'grade_A','grade_E','int_rate']]
y = final_train_df['loan_status']

# Split data to facilitate the evaluation of models
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=0)

# Convert Feature/Column with Scaler
scaler = MinMaxScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

#Train the classifier
model = CatBoostClassifier(random_state=5,silent=True)
model.fit(X_train_scaled, y_train)
y_pred = model.predict(X_test_scaled)

print('AUC:',roc_auc_score(y_test, y_pred_proba))
print('Accuracy: ',metrics.accuracy_score(y_test, y_pred))
print('Recall: ',metrics.recall_score(y_test, y_pred, zero_division=1))
print('Precision:',metrics.precision_score(y_test, y_pred, zero_division=1))
print('f1_score:',metrics.f1_score(y_test, y_pred))
print(metrics.classification_report(y_test, y_pred))
```

```
AUC: 0.9024818341011385
Accuracy:  0.8871096709031226
Recall:  0.4419051687313114
Precision: 0.9675005845218612
f1_score: 0.6067003885345649
              precision    recall  f1-score   support

           0       0.88      1.00      0.93     76320
           1       0.97      0.44      0.61     18728

    accuracy                           0.89     95048
   macro avg       0.92      0.72      0.77     95048
weighted avg       0.90      0.89      0.87     95048
```

Since we want to minimise the risk of losing money while lending to customers, this means that the consequences of false negative is high.
In this case, we want to prioritize on the recall score to ensure we have low false negatives.

4C. Hyperparameter tuning

We will start tuning the parameters of the model selected to improve our recall score.

In [57]:
```
# Features identified to train models
X = final_train_df[['postal_code_11650','postal_code_86630','postal_code_00813','postal_code_29597','postal_code_05113',
                    'postal_code_93700','term_ 36 months',
                    'grade_A','grade_E','int_rate']]
y = final_train_df['loan_status']

# Split data to facilitate the evaluation of models
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=0)

# Convert Feature/Column with Scaler
scaler = MinMaxScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Define the hyperparameter distributions
param_dist = {
    'n_estimators': [100, 200, 500],
    'max_depth': [4, 6, 8, 10],
    'learning_rate': [0.001, 0.01, 0.1],
    'l2_leaf_reg': [1, 3, 5],
    'colsample_bylevel': [0.3, 0.5, 0.8, 1],
    'subsample': [0.5, 0.7, 0.8, 1],
    'scale_pos_weight': [1,3, 4, 5,10]
}

# Create the CatBoostClassifier model object
cat_model = CatBoostClassifier(random_state=5,verbose=0)

# Create the RandomizedSearchCV object
random_search = RandomizedSearchCV(estimator = cat_model,
                                   param_distributions=param_dist,
                                   n_iter=15,
                                   cv=5,
                                   scoring=['recall','f1','precision'],
                                   refit='recall',
                                   random_state=5,
                                   n_jobs = -1)

# Fit the RandomizedSearchCV object to the training data
random_search.fit(X_train_scaled, y_train)

# Report
results = random_search.cv_results_
best_index = random_search.best_index_

print("--- Best Model Performance (Optimized for Recall) ---")
print(f"Best Recall:    {results['mean_test_recall'][best_index]:.4f}")
print(f"Associated F1:   {results['mean_test_f1'][best_index]:.4f}")
print(f"Best Params:    {random_search.best_params_}")
```

```
--- Best Model Performance (Optimized for Recall) ---
Best Recall:    0.9705
Associated F1:   0.5020
Best Params:    {'subsample': 1, 'scale_pos_weight': 10, 'n_estimators': 500, 'max_depth': 10, 'learning_rate': 0.01, 'l2_leaf_reg': 5, 'colsample_bylevel': 0.5}
```

We fit the best set of hyperparameters to our machine learning model and view the classification report.

In [58]:
```
# Features identified to train models
X = final_train_df[['postal_code_11650','postal_code_86630','postal_code_00813','postal_code_29597','postal_code_05113',
                    'postal_code_93700','term_ 36 months',
                    'grade_A','grade_E','int_rate']]
y = final_train_df['loan_status']

# Split data to facilitate the evaluation of models
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=0)

# Convert Feature/Column with Scaler
scaler = MinMaxScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Create the CatBoostClassifier model
model = random_search.best_estimator_
y_pred = model.predict(X_test_scaled)
y_pred_proba = model.predict_proba(X_test_scaled)[:, 1]

print('AUC:',roc_auc_score(y_test, y_pred_proba))
print('Accuracy: ',metrics.accuracy_score(y_test, y_pred))
print('Recall: ',metrics.recall_score(y_test, y_pred, zero_division=1))
print('Precision:',metrics.precision_score(y_test, y_pred, zero_division=1))
print('f1_score:',metrics.f1_score(y_test, y_pred))
print(metrics.classification_report(y_test, y_pred))
```

```
AUC: 0.8991110341654822
Accuracy:  0.6242214460062284
Recall:  0.9715933361811192
Precision: 0.340870347127255
f1_score: 0.5046804143726858
              precision    recall  f1-score   support

           0       0.99      0.54      0.70     76320
           1       0.34      0.97      0.50     18728

    accuracy                           0.62     95048
   macro avg       0.66      0.76      0.60     95048
weighted avg       0.86      0.62      0.66     95048
```

In [59]:
```python
# Get the raw probabilities (values between 0 and 1)
y_pred_proba = model.predict_proba(X_test_scaled)[:, 1]

# Manually set a stricter threshold (0.7)
y_pred_stricter = (y_pred_proba >= 0.7).astype(int)

# Print the new report to see the change
from sklearn.metrics import classification_report
print("Performance with 0.7 Threshold:")
print(classification_report(y_test, y_pred_stricter))
```

```
Performance with 0.7 Threshold:
              precision    recall  f1-score   support

           0       0.94      0.77      0.85     76320
           1       0.46      0.81      0.59     18728

    accuracy                           0.78     95048
   macro avg       0.70      0.79      0.72     95048
weighted avg       0.85      0.78      0.80     95048
```

After performing a Randomized Search with 5-fold cross-validation, the CatBoost model was optimized for Recall. On the final unseen test set, the model successfully identified 97% of actual defaults, providing the bank with a robust defensive tool against credit loss.

The model was initially tuned for maximum safety (0.5 threshold), capturing 97% of defaults. To increase loan volume, we 'loosened' the model by adjusting the probability threshold to 0.7. This improved the approval rate of good borrowers from 54% to 77% and increased Precision to 0.46, while maintaining a strong Default Recall of 0.81.

5. 📊 Generate and Export Predictions from Final Model

> **OVERALL GOAL:**
>
> - Export your predictions and deploy it

5a. Re-fit your final model on train.csv

We retrain the chosen model with the optimal parameters, on the chosen basket of features on all the 316824 rows of data we have (instead of just on X_train, y_train).

In [ ]:
```python
import joblib

# Select optimal features to use to predict the Loan_status
X = final_train_df[['postal_code_11650','postal_code_86630','postal_code_00813','postal_code_29597','postal_code_05113',
                    'postal_code_93700','term_ 36 months',
                    'grade_A','grade_E','int_rate']]
y = final_train_df['loan_status']

print(f"Total features rows: {len(X)}")
print(f"Total target rows: {len(y)}")

# Convert Feature/Column with Scaler
scaler = MinMaxScaler()
X_scaled = scaler.fit_transform(X)

# Create the catboost model
final_model = random_search.best_estimator_
final_model.fit(X_scaled, y)

# 3. Save the Scaler and the Model to files
joblib.dump(scaler, 'loan_scaler.pkl')
joblib.dump(final_model, 'loan_prediction_model.pkl')

print("Model and Scaler exported successfully!")
```

```
Total features rows: 316824
Total target rows: 316824
```