Capstone project for AI200: Applied Machine Learning by Heicoders Academy
Porject done by: Aaron Kee

📄 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
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

1. Importing Data

In [1]:

```
import pandas as pd
import numpy as np
import calendar
import plotly.express as px
from datetime import datetime
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split, RandomizedSearchCV, GridSearchCV,
from sklearn import metrics
from sklearn.metrics import mean_squared_error, roc_curve, auc, roc_auc_score
from sklearn.datasets import make_classification
from sklearn.preprocessing import OrdinalEncoder, LabelEncoder
import warnings
warnings.filterwarnings("ignore")

# import data set
train_df = pd.read_csv("data/lc_trainingset.csv")
test_df = pd.read_csv("data/lc_testset.csv")

df = train_df
df = df.drop(df[df["verification_status"] == "Not Verified"].index)
```

In [2]:

```
# change prediction to int
def change_loan_status(loan_status):
    if loan_status in ['Fully Paid', 'Current']:
        return 0
    else:
        return 1

df['loan_status_int'] = df['loan_status'].apply(change_loan_status)
```

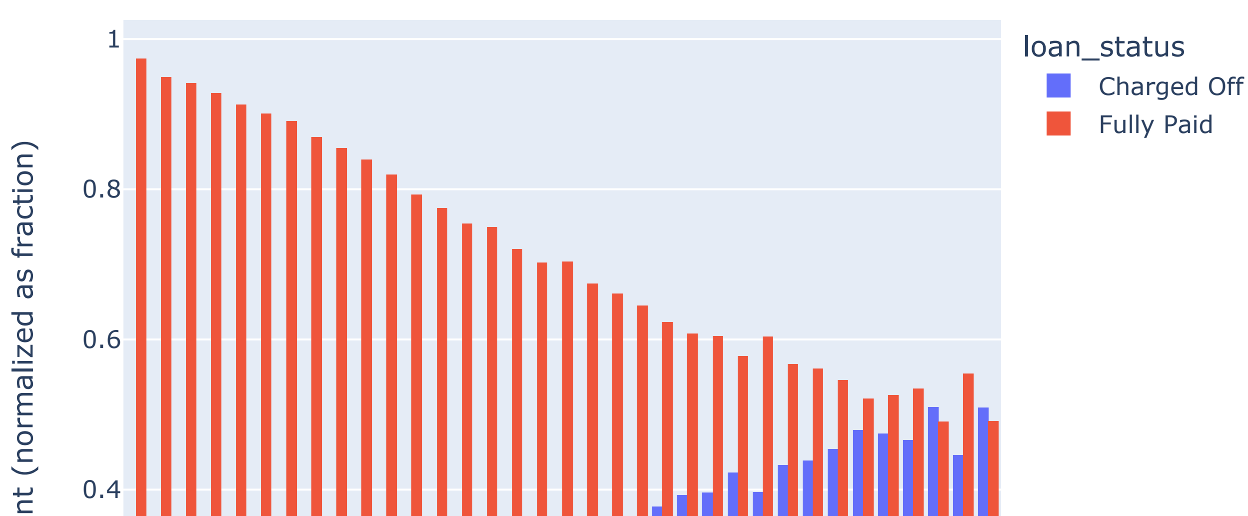
2. Feature Engineering

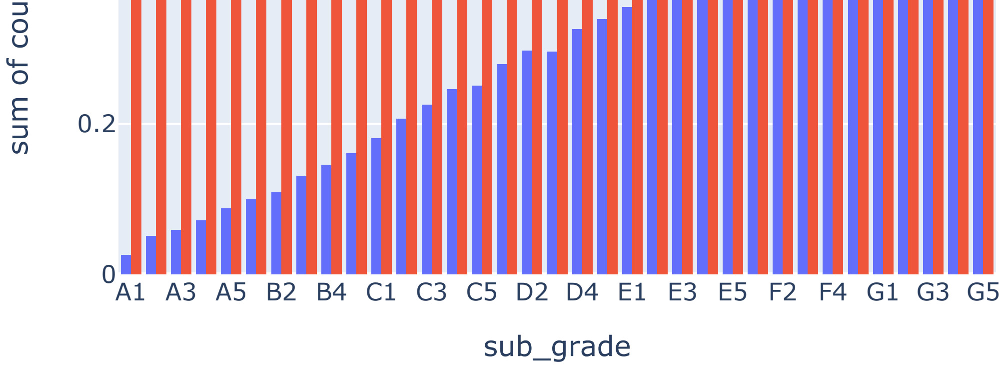
2.1 Grade

From Figure, there is a positive association between grade and probability of defaulting. There are less defaulters at higher grades. However, at the lower grades, there is almost an equal chance of loaners defaulting.

In [3]:
```
df["grade"] = df["sub_grade"]

# change string to int by using ordinal encoder
grade_to_int = ["A1", "A2", "A3", "A4", "A5",
                "B1", "B2", "B3", "B4", "B5",
                "C1", "C2", "C3", "C4", "C5",
                "D1", "D2", "D3", "D4", "D5",
                "E1", "E2", "E3", "E4", "E5",
                "F1", "F2", "F3", "F4", "F5",
                "G1", "G2", "G3", "G4", "G5"]

grade_encoder = OrdinalEncoder(categories=[grade_to_int], dtype=int)

df["grade"] = grade_encoder.fit_transform(df[["grade"]])
```

In [4]:
```
df["count"] = 1

fig = px.histogram(df,
                   y="count",
                   x="sub_grade",
                   color="loan_status",
                  barmode="group",
                  #histnorm="probability",
                  barnorm="fraction")
fig.update_xaxes(categoryorder='array', categoryarray= ["A1", "A2", "A3", "A4", "A5",
                                                        "B1", "B2", "B3", "B4", "B5",
                                                        "C1", "C2", "C3", "C4", "C5",
                                                        "D1", "D2", "D3", "D4", "D5",
                                                        "E1", "E2", "E3", "E4", "E5",
                                                        "F1", "F2", "F3", "F4", "F5",
                                                        "G1", "G2", "G3", "G4", "G5"])
```





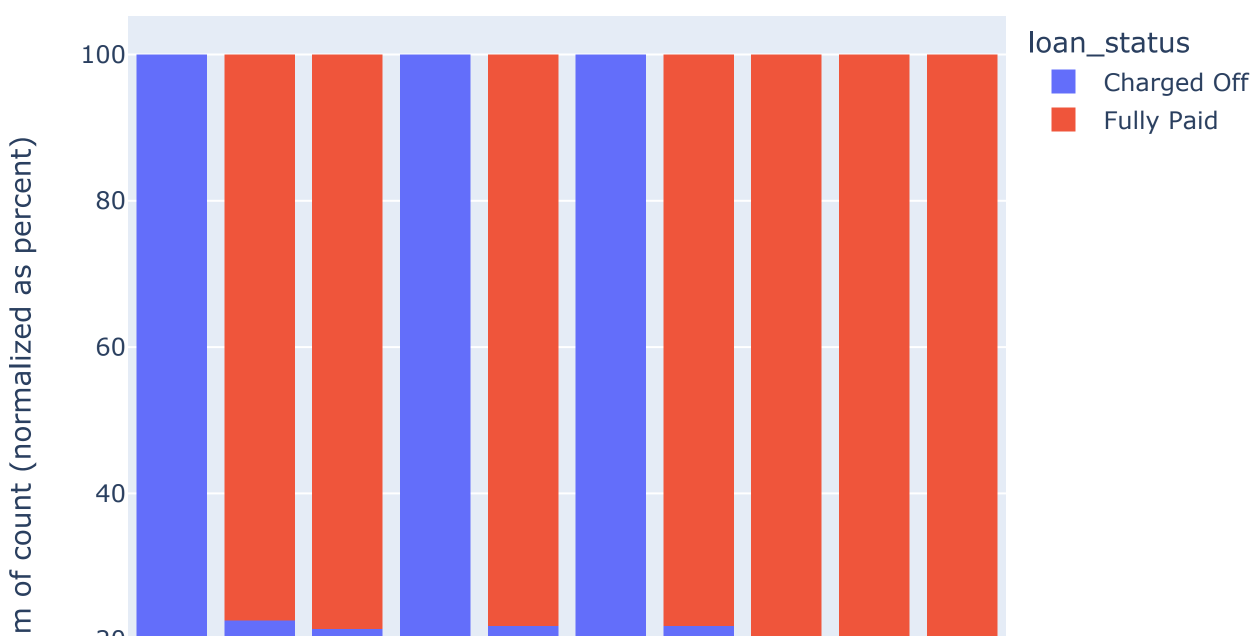
2.2 City area by zipcode

From Fig, some areas has very high (100%) defaulters and some areas has no defaulters and some area has a mix of non-defaulters and defaulters.

In [5]:
```
# get last 5 digits of address (zipcode)
df["area"] = df["address"].apply(lambda x: x[-5:])

area_encoder = LabelEncoder()
df["area_int"] = area_encoder.fit_transform(df[["area"]])
df["area"].unique()
```

Out[5]:
```
array(['93700', '48052', '00813', '22690', '05113', '70466', '30723',
       '86630', '29597', '11650'], dtype=object)
```

In [6]:
```
df["count"] = 1

fig = px.histogram(df,
                   x="area",
                   y="count",
                   color="loan_status",
                   barnorm="percent"
                  )

fig.show()
```



2.3 Loan income ratio

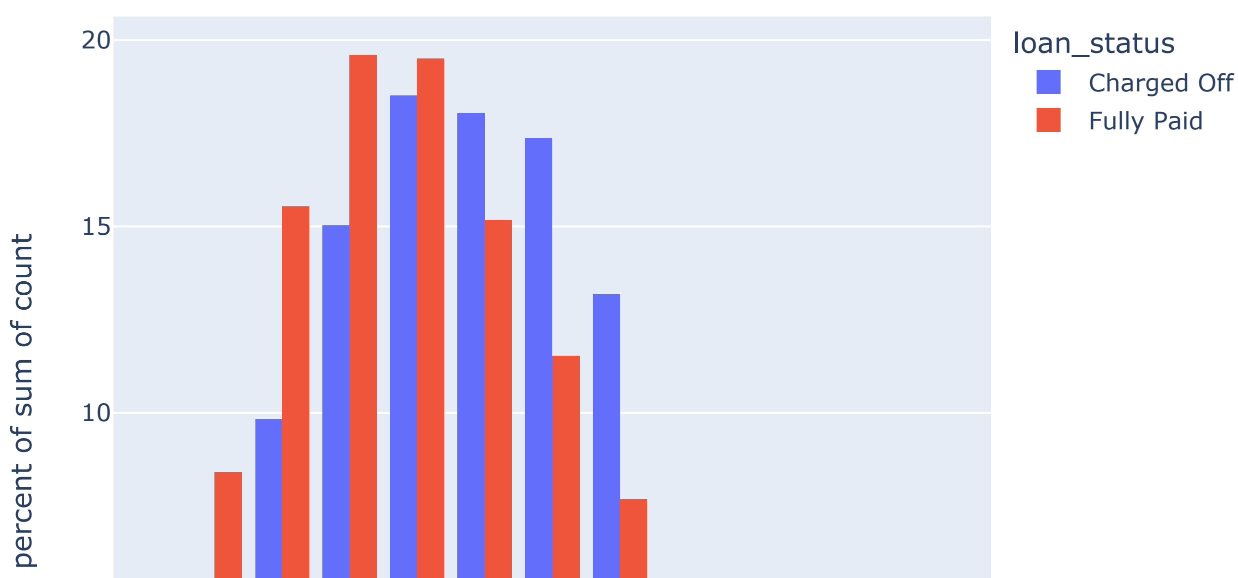
From Fig, there is a slight negative association between loan to income ratio and loan status. At higher loan to income ratio, there is slightly higher amount of defaulters compared to non-defaulters

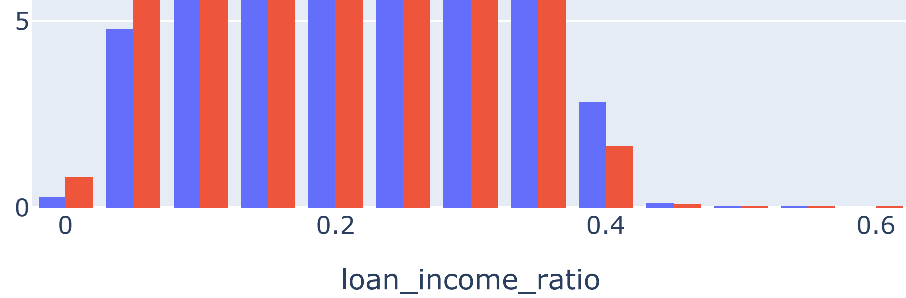
In [7]:

```
# create loan/income ratio
df["loan_income_ratio"] = df["loan_amnt"] / (df["annual_inc"])
df["loan_income_ratio"].fillna(df["loan_income_ratio"].mean())

# log transform annual income
df['annual_inc_log'] = (df['annual_inc']+1).transform(np.log)

# removing outlier with SD
factor = 4
upper_lim_loan_income_ratio = df['loan_income_ratio'].mean () + df['loan_income_ratio'].
df = df[(df['loan_income_ratio'] < upper_lim_loan_income_ratio)]

# log transform
df['loan_income_ratio'] = (df['loan_income_ratio']+1).transform(np.log)

#df[["annual_inc_log", "loan_income_ratio", "annual_inc"]].head(100)
```

In [8]:

```
fig = px.histogram(df,
                   x="loan_income_ratio",
                   y="count",
                   barmode="group",
                   color="loan_status",
                   histnorm="percent",
                   nbins=20)
fig.show()
```





2.4 Closed/opened credit line ratio

From fig, there is a positive association between closed over opened credit line ratio and loan status. At higher close over opened credit line, there are a lower amount of defaulter compared to non-defaulters

In [9]:

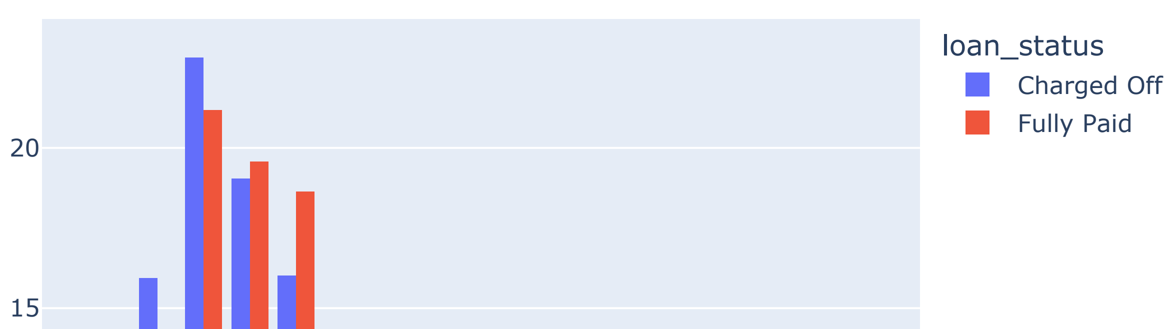
```
# calculating closed credit lines
df["closed_acc"] = df["total_acc"] - df["open_acc"]

# calculating closed over opened credit lines
df["closed_open_ratio"] = df["closed_acc"] / df["open_acc"]

# log transform closed over opened credit lines ratio
df['closed_open_ratio'] = (df['closed_open_ratio']+1).transform(np.log)

# replacing infinity value with max value
max_value = df.loc[df['closed_open_ratio'] != np.inf, 'closed_open_ratio'].max()
df["closed_open_ratio"].replace([np.inf, -np.inf], max_value, inplace=True)

df["closed_open_ratio"].max()
```

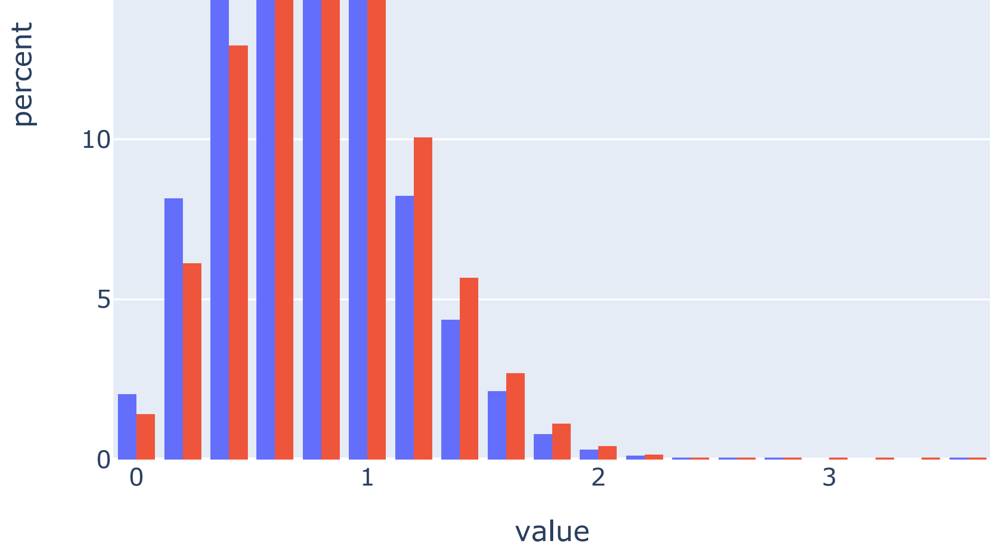
Out[9]:

```
3.5263605246161616
```

In [10]:

```
features = [#"total_acc",
            #"open_acc",
            #"closed_acc",
            "closed_open_ratio"
]

fig = px.histogram(df,
                   x=features,
                   barmode="group",
                   color="loan_status",
                   histnorm="percent",
                   nbins=30
            )
fig.show()
```





3. Model building and comparing Model Performance

3.1 Feature selection

In [11]:
```
features=[
          ### given features ###
          #'loan_amnt',
          #'term',
          #'int_rate',
          #'installment',
          'grade',
          #'emp_title',
          #'emp_length',
          #'home_ownership',
          #'annual_inc',
          #'verification_status',
          #'issue_d',
          #'purpose',
          #'title',
          #'dti',
          #'open_acc',
          #'pub_rec',
          #'revol_bal',
          #'revol_util',
          #'total_acc',
          #'initial_list_status',
          #'application_type',
          #'mort_acc',
          #'pub_rec_bankruptcies',

          ### created features ###
          "area_int",
          #"issue_d_month",
          #"issue_d_year",
          #"pub_rec_binary",
          "loan_income_ratio",
          "closed_open_ratio",
          #"closed_acc",
          #"time_since_cr_line_int",
```

```
        #"earliest_cr_line_minus_issue_d",
        #"issue_d_time",
        ]
```

3.2 Train test split (70/30)

In [12]:
```
X = df[features]
y = df['loan_status_int']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=0)
```

In [13]:
```
X.isna().sum()
```

Out[13]:
```
grade                 0
area_int              0
loan_income_ratio     0
closed_open_ratio     0
dtype: int64
```

In [14]:
```
X.dtypes
```

Out[14]:
```
grade                   int64
area_int                int64
loan_income_ratio     float64
closed_open_ratio     float64
dtype: object
```

In [15]:
```
np.isinf(X).sum()
```

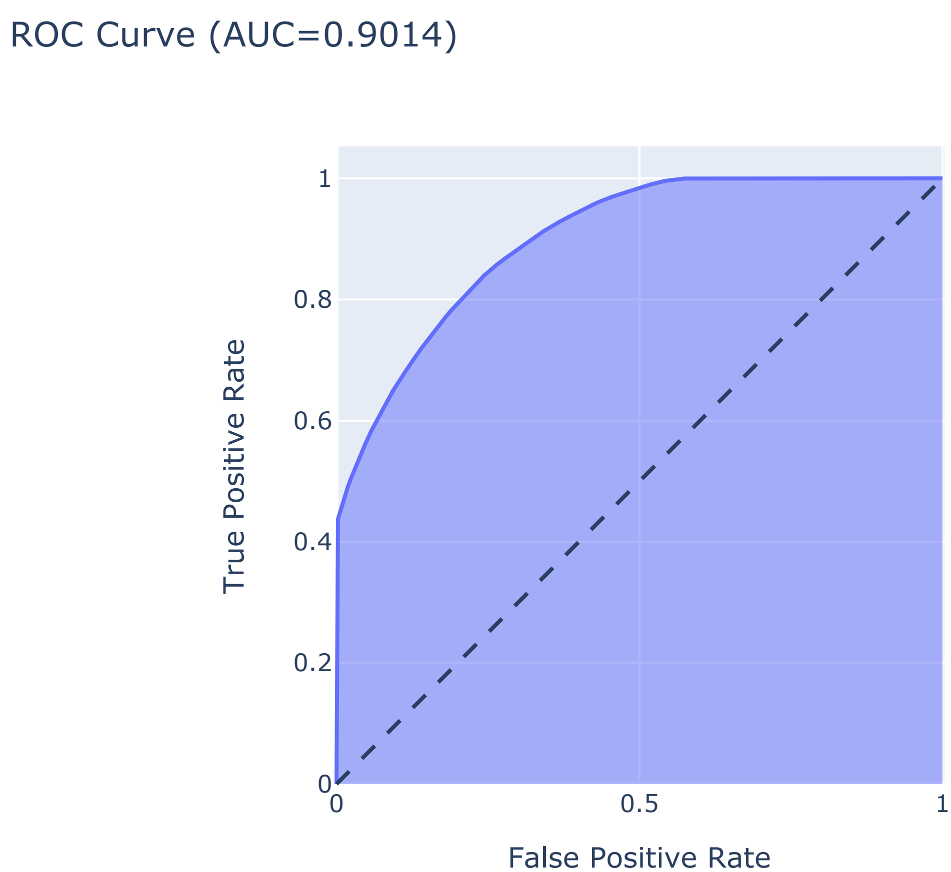
Out[15]:
```
grade                 0
area_int              0
loan_income_ratio     0
closed_open_ratio     0
dtype: int64
```

3.3 XGboost

In [16]:
```
import xgboost as xgb
model_xgboost = xgb.XGBClassifier(learning_rate=0.1,
                                  random_state=5,
                                  max_depth=3,
                                 )

# Fit the model with the training data
model_xgboost.fit(X_train, y_train)

# Predict the target on the test dataset
y_predict = model_xgboost.predict(X_test)
print('\nPrediction on test data', y_predict)

RMSE = mean_squared_error(y_test, y_predict, squared=False)
print('\nRMSE on test dataset: %.4f' % RMSE)

# Accuracy Score on test dataset
accuracy_test = metrics.accuracy_score(y_test, y_predict)
print('\nAccuracy_score on test dataset : ', accuracy_test)

# Generate predictions and format results
y_pred_probabilities = model_xgboost.predict_proba(X)
y_pred_probabilities_formatted = y_pred_probabilities[:, 1]

# Calculate the (1) false positive rate, (2) true positive rate, and (3) thresholds
```

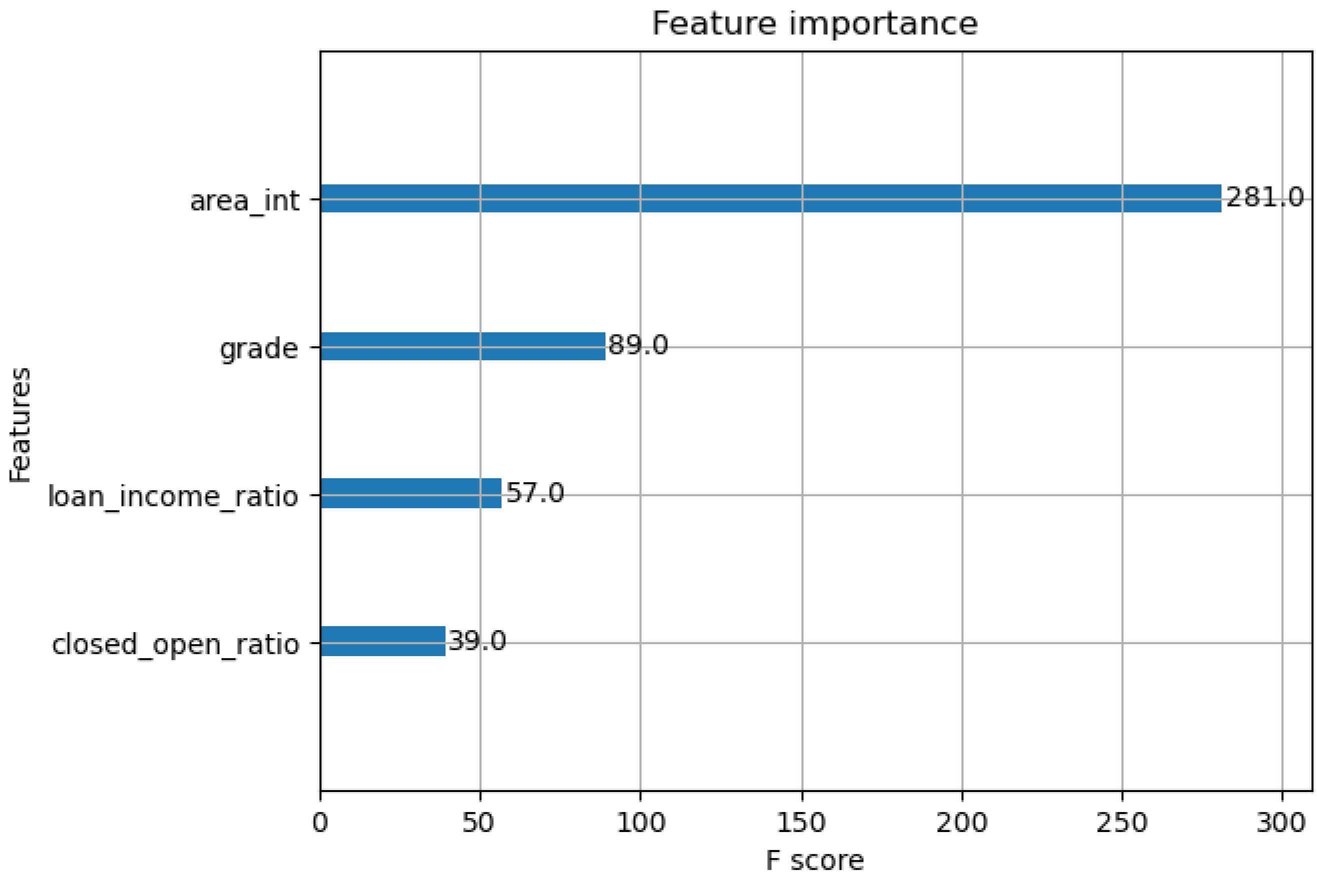
```
y_pred_probabilities_formatted = list(y_pred_probabilities_formatted) # converting to li
fpr, tpr, thresholds = roc_curve(y, y_pred_probabilities_formatted)

# Plotting the chart
fig = px.area(
    x=fpr, y=tpr,
    title=f'ROC Curve (AUC={auc(fpr, tpr):.4f})',
    labels=dict(x='False Positive Rate', y='True Positive Rate'),
    width=700, height=500
)
fig.add_shape(
    type='line', line=dict(dash='dash'),
    x0=0, x1=1, y0=0, y1=1
)
fig.update_yaxes(scaleanchor="x", scaleratio=1)
fig.update_xaxes(constrain='domain')


fig.show()
```

```
Prediction on test data [0 0 0 ... 0 0 1]

RMSE on test dataset: 0.3553

Accuracy_score on test dataset :  0.8737350435237304
```



In [17]:
```
from matplotlib import pyplot
from xgboost import plot_importance
print(model_xgboost.feature_importances_)
plot_importance(model_xgboost)
pyplot.show()
```

```
[0.22039889 0.69735265 0.06191144 0.02033701]
```



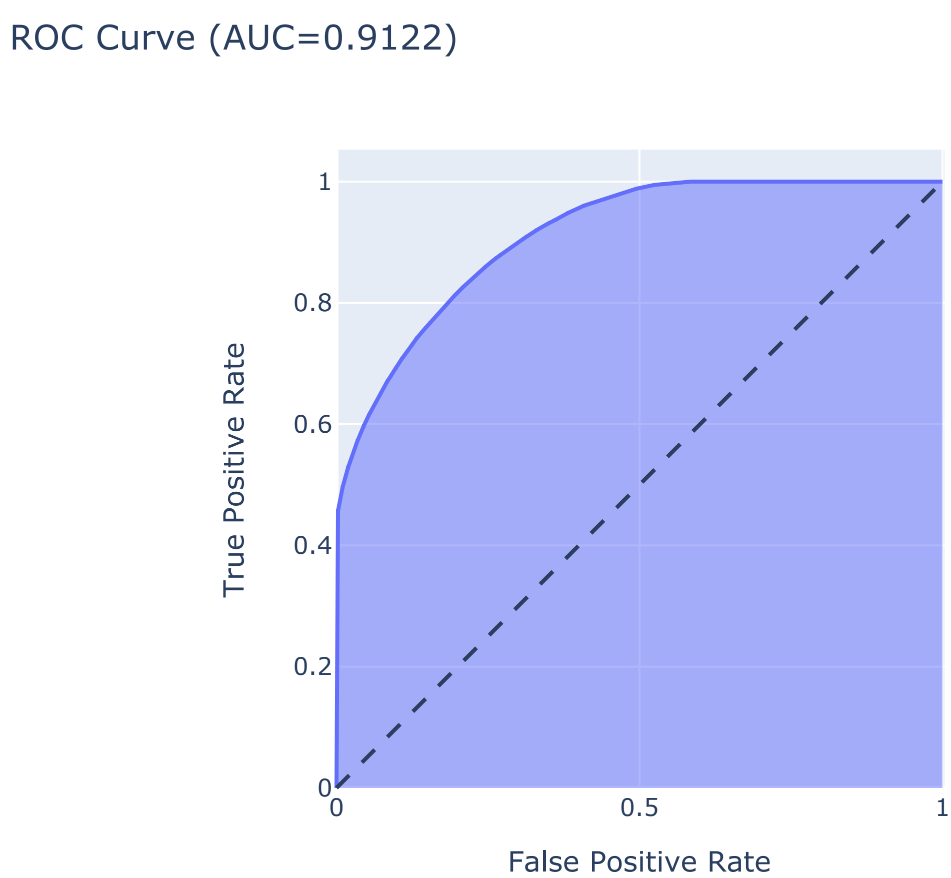
3.4 Catboost

In [18]:

```
from catboost import CatBoostClassifier

# Instantiate the model object
model_CatBoost = CatBoostClassifier(learning_rate=0.1,
                                    random_seed=5,
                                    silent=True
                                   )

# Fit the model with the training data
model_CatBoost.fit(X_train, y_train)  # set verbose=False if you find the logs too long

# Predict the target on the test dataset
y_predict = model_CatBoost.predict(X_test)
print('\nPrediction on test data', y_predict)

# Accuracy Score on test dataset
accuracy_test = metrics.accuracy_score(y_test, y_predict)
print('\nAccuracy_score on test dataset : ', accuracy_test)

# Generate predictions and format results
y_pred_probabilities = model_CatBoost.predict_proba(X)
y_pred_probabilities_formatted = y_pred_probabilities[:, 1]

# Calculate the (1) false positive rate, (2) true positive rate, and (3) thresholds
y_pred_probabilities_formatted = list(y_pred_probabilities_formatted) # converting to li
fpr, tpr, thresholds = roc_curve(y, y_pred_probabilities_formatted)

# Plotting the chart
fig = px.area(
    x=fpr, y=tpr,
    title=f'ROC Curve (AUC={auc(fpr, tpr):.4f})',
    labels=dict(x='False Positive Rate', y='True Positive Rate'),
```

```
    width=700, height=500
)

# This part is just for formatting & adding the dash-line
fig.add_shape(
    type='line', line=dict(dash='dash'),
    x0=0, x1=1, y0=0, y1=1
)
fig.update_yaxes(scaleanchor="x", scaleratio=1)
fig.update_xaxes(constrain='domain')


fig.show()
```

```
Prediction on test data [0 0 0 ... 0 0 1]

Accuracy_score on test dataset :  0.8723816554396974
```



3.5 Grid search using XGboost

Grid search is to find the optimal hyperparameter to achieve the best AUC-ROC
Reference: https://www.datasnips.com/5/tuning-xgboost-with-grid-search/

In [19]:
```
# To set parameters to perform grid search
params = {
        'min_child_weight': [3, 7],
        'gamma': [1, 3],
        'subsample': [0.5, 0.8],
        'colsample_bytree': [0.6, 0.8],
        'max_depth': [3, 5, 7]
        }
```

In [20]:
```
# To initialise XGboostclassifer
estimator = xgb.XGBClassifier(n_estimators=100,
                              n_jobs=-1,
                              eval_metric='auc',
                              early_stopping_rounds=10)
```

In [21]:
```
model = GridSearchCV(estimator=estimator,
                     param_grid=params,
                     cv=3,
                     scoring="neg_log_loss")
```

In [22]:
```
model.fit(X_train,
          y_train,
          eval_set=[(X_train, y_train)],
          verbose=0)
```

Out[22]:
```
▸ GridSearchCV
▸ estimator: XGBClassifier
    ▸ XGBClassifier
```

In [23]:
```
print(model.best_params_)
```

```
{'colsample_bytree': 0.8, 'gamma': 3, 'max_depth': 3, 'min_child_weight': 3, 'subsampl
e': 0.8}
```

4. Refit Final Model and export Predictions

4.1 Train final model using best params for XGboost

In [24]:
```
features

X = df # Select the features you want to use to predict the loan_status
y = df['loan_status_int']
```

In [25]:
```
X = X[features]

final_model_xgboost = xgb.XGBClassifier(learning_rate=0.1,
                                        random_state=5,
                                        colsample_bytree=0.8,
                                        gamma=3,
                                        max_depth=3,
                                        min_child_weight=3,
                                        subsample=0.8,
                                        )
final_model_xgboost.fit(X, y)
```

Out[25]:

```
▾ XGBClassifier
XGBClassifier(base_score=None, booster=None, callbacks=None,
              colsample_bylevel=None, colsample_bynode=None,
              colsample_bytree=0.8, early_stopping_rounds=None,
              enable_categorical=False, eval_metric=None, feature_types=None,
              gamma=3, gpu_id=None, grow_policy=None, importance_type=None,
              interaction_constraints=None, learning_rate=0.1, max_bin=None,
              max_cat_threshold=None, max_cat_to_onehot=None,
              max_delta_step=None, max_depth=3, max_leaves=None,
              min_child_weight=3, missing=nan, monotone_constraints=None,
              n_estimators=100, n_jobs=None, num_parallel_tree=None,
              predictor=None, random_state=5, ...)
```

4.2 Feature engineering for test data set

In [26]:
```
df = test_df

### GRADE ###

df["grade"] = df["sub_grade"]

# change string to int by using ordinal encoder
grade_to_int = ["A1", "A2", "A3", "A4", "A5",
                "B1", "B2", "B3", "B4", "B5",
                "C1", "C2", "C3", "C4", "C5",
                "D1", "D2", "D3", "D4", "D5",
                "E1", "E2", "E3", "E4", "E5",
                "F1", "F2", "F3", "F4", "F5",
                "G1", "G2", "G3", "G4", "G5"]

grade_encoder = OrdinalEncoder(categories=[grade_to_int], dtype=int)

df["grade"] = grade_encoder.fit_transform(df[["grade"]])

### Area ###

# get last 5 digits of address (zipcode)
df["area"] = df["address"].apply(lambda x: x[-5:])

area_encoder = LabelEncoder()
df["area_int"] = area_encoder.fit_transform(df[["area"]])
df["area"].unique()

### Loan to income Ratio ###

# create loan/income ratio
df["loan_income_ratio"] = df["loan_amnt"] / (df["annual_inc"])
df["loan_income_ratio"].fillna(df["loan_income_ratio"].mean())

# log transform annual income
df['annual_inc_log'] = (df['annual_inc']+1).transform(np.log)

# log transform
df['loan_income_ratio'] = (df['loan_income_ratio']+1).transform(np.log)

### Close to opened credit lines ratio ###

# calculating closed credit lines
df["closed_acc"] = df["total_acc"] - df["open_acc"]
```

```
# calculating closed over opened credit lines
df["closed_open_ratio"] = df["closed_acc"] / df["open_acc"]

# log transform closed over opened credit lines ratio
df['closed_open_ratio'] = (df['closed_open_ratio']+1).transform(np.log)

# replacing infinity value with max value
max_value = df.loc[df['closed_open_ratio'] != np.inf, 'closed_open_ratio'].max()
df["closed_open_ratio"].replace([np.inf, -np.inf], max_value, inplace=True)

test_df = df
```

4.3 Export prediction of test data set using final model

In [27]:

```
kaggle_x = test_df[features]

# XGBOOST
probabilities = final_model_xgboost.predict_proba(kaggle_x)
probabilities
```

Out[27]:

```
array([[6.7585158e-01, 3.2414842e-01],
       [7.4820501e-01, 2.5179499e-01],
       [8.7416565e-01, 1.2583433e-01],
       ...,
       [7.1876526e-01, 2.8123477e-01],
       [7.7005982e-01, 2.2994021e-01],
       [9.9953830e-01, 4.6167453e-04]], dtype=float32)
```

In [28]:

```
kaggle_preds = probabilities[:,1]
#kaggle_preds = probabilities
len(kaggle_preds)
#length should be 78237
```

Out[28]:

```
78237
```

In [29]:

```
output_dataframe = pd.DataFrame({
    'id': list(range(len(kaggle_preds))),
    'Predicted': kaggle_preds
})
output_dataframe.to_csv('my_predictions.csv', index=False)
```

5. Features considered but not used due to low correlation

5.1 Time from earliest credit line opened to time when loan is issued

From fig, there is a slight right skewed of the distribution of time from earliest credit lin opened to time when loan is issued among the defaulters comapred to the non-defaulters

In [30]:

```
### Calculates time since earlist credit line opened

# creates a dict of month string to month num
abbr_to_num = {name: num for num, name in enumerate(calendar.month_abbr) if num}

# function for apply. To change month in string format to int format
def month_to_num(s):
```

```
    s = str(s)
    s = s.split("-")
    s[0] = abbr_to_num[s[0]]
    s = "01" + "-" + str(s[0]) + "-" + str(s[1])
    return s

# change string to "%d-%m-%y" format
df['earliest_cr_line'] = df["earliest_cr_line"].apply(month_to_num)

# change from string object to datetime object
df['earliest_cr_line'] = pd.to_datetime(df['earliest_cr_line'], format="%d-%m-%Y")

# current datetime minus min datetime
df['time_since_cr_line'] = df['earliest_cr_line'].apply(lambda x: (x - df["earliest_cr_l
time_since_cr_line_encoder = OrdinalEncoder(categories="auto", dtype=int)
df["time_since_cr_line_int"] = time_since_cr_line_encoder.fit_transform(df[["time_since_
```

In [31]:
```
### Calculates time since loan issued

# creates a dict of month string to month num
abbr_to_num = {name: num for num, name in enumerate(calendar.month_abbr) if num}

# funtion for apply function. To change month in string to int format
def month_to_num_2(s):
    s = str(s)
    s = s.split("-")
    s[0] = abbr_to_num[s[0]]
    s = "01" + "-" + str(s[0]) + "-" + str(s[1])
    return s

# change string to "%d-%m-%y" format
df['issue_d'] = df["issue_d"].apply(month_to_num_2)

# change from string object to datetime object
df['issue_d'] = pd.to_datetime(df['issue_d'], format="%d-%m-%Y")

# current datetime minus min datetime
df['issue_d_time'] = df['issue_d'].apply(lambda x: (x - df["issue_d"].min()).days)/30
issue_d_encoder = OrdinalEncoder(categories="auto", dtype=int)
df["issue_d_time"] = issue_d_encoder.fit_transform(df[["issue_d_time"]])
```

In [32]:
```
# calculates time from earliest credit line opened to time of loan issued
df["earliest_cr_line_minus_issue_d"] = df["time_since_cr_line_int"] - df["issue_d_time"]
```

In [33]:
```
fig = px.histogram(df,
                  x="earliest_cr_line_minus_issue_d",
                    facet_col="loan_status",
                    histnorm="percent",
                    nbins=40
                 )

fig.show()
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
Cell In[33], line 1
----> 1 fig = px.histogram(df,
      2                   x="earliest cr line minus issue_d",
      3                     facet col="loan status",
      4                     histnorm="percent",
      5                     nbins=40
      6                  )
      8 fig.show()
```

```
File ~/anaconda3/lib/python3.10/site-packages/plotly/express/_chart_types.py:477, in his
togram(data_frame, x, y, color, pattern_shape, facet_row, facet_col, facet_col_wrap, fac
et_row_spacing, facet_col_spacing, hover_name, hover_data, animation_frame, animation_gr
oup, category_orders, labels, color_discrete_sequence, color_discrete_map, pattern_shape
_sequence, pattern_shape_map, marginal, opacity, orientation, barmode, barnorm, histnor
m, log_x, log_y, range_x, range_y, histfunc, cumulative, nbins, text_auto, title, templa
te, width, height)
    431 def histogram(
    432     data_frame=None,
    433     x=None,
   (...)
    469     height=None,
    470 ) -> go.Figure:
    471     """
    472     In a histogram, rows of `data_frame` are grouped together into a
    473     rectangular mark to visualize the 1D distribution of an aggregate
    474     function `histfunc` (e.g. the count or sum) of the value `y` (or `x` if
    475     `orientation` is `'h'`).
    476     """
--> 477     return make figure(
    478         args=locals(),
    479         constructor=go.Histogram,
    480         trace patch=dict(
    481             histnorm=histnorm,
    482             histfunc=histfunc,
    483             cumulative=dict(enabled=cumulative),
    484         ),
    485         layout_patch=dict(barmode=barmode, barnorm=barnorm),
    486     )

File ~/anaconda3/lib/python3.10/site-packages/plotly/express/_core.py:1948, in make_figu
re(args, constructor, trace_patch, layout_patch)
   1945 layout_patch = layout_patch or {}
   1946 apply_default cascade(args)
-> 1948 args = build_dataframe(args, constructor)
   1949 if constructor in [go.Treemap, go.Sunburst, go.Icicle] and args["path"] is not N
one:
   1950     args = process_dataframe_hierarchy(args)

File ~/anaconda3/lib/python3.10/site-packages/plotly/express/_core.py:1405, in build_dat
aframe(args, constructor)
   1402     args["color"] = None
   1403 # now that things have been prepped, we do the systematic rewriting of `args`
-> 1405 df output, wide id vars = process args into_dataframe(
   1406     args, wide_mode, var_name, value_name
   1407 )
   1409 # now that `df_output` exists and `args` contains only references, we complete
   1410 # the special-case and wide-mode handling by further rewriting args and/or mutat
ing
   1411 # df_output
   1413 count_name = _escape_col_name(df_output, "count", [var_name, value_name])

File ~/anaconda3/lib/python3.10/site-packages/plotly/express/_core.py:1207, in process_a
rgs_into_dataframe(args, wide_mode, var_name, value_name)
   1205         if argument == "index":
   1206             err_msg += "\n To use the index, pass it in directly as `df.index`."
-> 1207         raise ValueError(err_msg)
   1208 elif length and len(df_input[argument]) != length:
   1209     raise ValueError(
   1210         "All arguments should have the same length. "
   1211         "The length of column argument `df[%s]` is %d, whereas the "
   (...)
   1218         )
   1219     )

ValueError: Value of 'facet_col' is not the name of a column in 'data_frame'. Expected o
```

```
ne of ['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', 'rev
ol_util', 'total_acc', 'initial_list_status', 'application_type', 'mort_acc', 'pub_rec_b
ankruptcies', 'address', 'area', 'area_int', 'loan_income_ratio', 'annual_inc_log', 'clo
sed_acc', 'closed_open_ratio', 'time_since_cr_line', 'time_since_cr_line_int', 'issue_d_
time', 'earliest_cr_line_minus_issue_d'] but received: loan_status
```

5.2 Pub_rec binary

From fig, there is a slightly higher percentage of people defaulting among people with 1 or more public derogatory record.

In [ ]:
```
df["pub_rec_binary"] = df["pub_rec"].apply(lambda x: 0 if x == 0 else 1)
```

In [ ]:
```
df["count"] = 1

fig = px.histogram(df,
                  x="pub_rec_binary",
                  #facet_row="pub_rec_binary",
                  y="count",
                   barnorm="percent",
                  color="loan_status")
fig.show()
```