```
from google.colab import drive
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.
```

```
!pip install sweetviz
```

```
!pip install jupyterthemes
```

Requirements

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

from sklearn.preprocessing import OrdinalEncoder
from sklearn.model_selection import train_test_split, StratifiedKFold, cross_val_score, cr
from sklearn.metrics import accuracy_score
from sklearn.model_selection import train_test_split
from sklearn.metrics import f1_score
from sklearn.model_selection import GridSearchCV
from sklearn.tree import DecisionTreeClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from xgboost import XGBClassifier
from sklearn.metrics import (accuracy_score, confusion_matrix, classification_report, roc_
from sklearn.pipeline import Pipeline

from scipy.stats import boxcox
from scipy import stats
from jupyterthemes import jtplot
```

```
pd.set_option('display.max_columns', 500)
pd.set_option('display.max_rows', 100)
```

Extract

```
df_org = pd.read_csv('/content/drive/MyDrive/Learn ML/LoanStats3a.csv', nrows=39787)
df_org.head()
```

	id	member_id	loan_amnt	funded_amnt	funded_amnt_inv	term	int_rate	insta
0	NaN	NaN	5000.0	5000.0	4975.0	36 months	10.65%	
1	NaN	NaN	2500.0	2500.0	2500.0	60 months	15.27%	
2	NaN	NaN	2400.0	2400.0	2400.0	36 months	15.96%	
3	NaN	NaN	10000.0	10000.0	10000.0	36 months	13.49%	
4	NaN	NaN	3000.0	3000.0	3000.0	60 months	12.69%	

```
df_org.shape
df = df_org
```

```
# Analyzing data

report = sv.analyze(df)

# Generating report
report.show_html('eda_report.html')
```

```
Done! Use 'show' commands to display/save.
Report eda_report.html was generated! NOTEBOOK/COLAB USERS: the web browser MAY not
```

Data Cleaning

Here data are cleaned by removing null values from the dataset.

Comuns with mean null percentage higher than 70% were removed.

```
df = df[df.columns[((df.isnull().sum())/len(df)) < 0.3]]
```

```
report = sv.analyze(df)

# Generating report
report.show_html('eda_report_after_dropping_null.html')
```

```
Done! Use 'show' commands to display/save.
Report eda_report_after_dropping_null.html was generated! NOTEBOOK/COLAB USERS: the
```

```
for col in df.columns:
    if (df[col].nunique()) == 1:
      df.drop(col,inplace=True,axis=1)

print(df.shape)
```

```
(39787, 40)
```

```
report = sv.analyze(df)

# Generating report
report.show_html('eda_report_after_dropping_single_valued_cols.html')
```

```
Done! Use 'show' commands to display/save.
Report eda_report_after_dropping_single_valued_cols.html was generated! NOTEBOOK/COL
```

```
# From the report we can see that the columns Loan amount, funded amount, funded amount ir
# more than 90% correlation. Therefore, we will remove those columns
df.drop(['funded_amnt', 'funded_amnt_inv', 'total_pymnt_inv'], axis=1, inplace=True)
print(df.shape)
```

```
(39787, 37)
```

```
# From reports we can see that there are some columns with almost 1 values. We will remove
df.drop(['delinq_2yrs', 'revol_util', 'total_rec_late_fee', 'recoveries', 'collection_recc
print(df.shape)
```

```
(39787, 31)
```

Removing Unnecessary Symbols and Strings

```
#remove months from term
df['term'] = df['term'].astype(str).map(lambda x: x.lstrip(' ').rstrip('months'))

#Remove percentage mark
df['int_rate'] = df['int_rate'].str.rstrip('%').astype('float')
```

```
df.head()
```

	loan_amnt	term	int_rate	installment	grade	sub_grade	emp_title	emp_length
0	5000.0	36	10.65	162.87	B	B2	NaN	10+ years
1	2500.0	60	15.27	59.83	C	C4	Ryder	< 1 yea
2	2400.0	36	15.96	84.33	C	C5	NaN	10+ years
3	10000.0	36	13.49	339.31	C	C1	AIR RESOURCES BOARD	10+ years
4	3000.0	60	12.69	67.79	B	B5	University Medical Group	1 yea

Converting time data to numcerical values.

```
col = ['issue_d','earliest_cr_line','last_pymnt_d','last_credit_pull_d']
for i in col:
    df[i] = pd.to_datetime(df[i].str.upper(), format='%b-%y', yearfirst=False)
```

```
df['issue_d_year'] = pd.DatetimeIndex(df['issue_d']).year
df['issue_d_month'] = pd.DatetimeIndex(df['issue_d']).month
df['last_pymnt_d_year'] = pd.DatetimeIndex(df['last_pymnt_d']).year
df['last_pymnt_d_month'] = pd.DatetimeIndex(df['last_pymnt_d']).month
df['last_credit_pull_d_year'] = pd.DatetimeIndex(df['last_credit_pull_d']).year
df['last_credit_pull_d_month'] = pd.DatetimeIndex(df['last_credit_pull_d']).month
```

```
df.earliest_cr_line = 2021 - pd.DatetimeIndex(df['earliest_cr_line']).year
df.issue_d_year = 2021 - (df.issue_d_year)

df.last_pymnt_d_year = 2021 - (df.last_pymnt_d_year)
df.last_credit_pull_d_year = 2021 - (df.last_credit_pull_d_year)
```

```
df.drop(['issue_d','last_pymnt_d','last_credit_pull_d'],axis=1,inplace=True)
```

```
df.head()
```

	loan_amnt	term	int_rate	installment	grade	sub_grade	emp_title	emp_length
0	5000.0	36	10.65	162.87	B	B2	NaN	10+ years
1	2500.0	60	15.27	59.83	C	C4	Ryder	< 1 year
2	2400.0	36	15.96	84.33	C	C5	NaN	10+ years
3	10000.0	36	13.49	339.31	C	C1	AIR RESOURCES BOARD	10+ years
4	3000.0	60	12.69	67.79	B	B5	University Medical Group	1 year

Remove rows with null values

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

```
loan_amnt                   1
term                        0
int_rate                    1
installment                 1
grade                       1
sub_grade                   1
emp_title                2468
emp_length               1079
home_ownership              1
annual_inc                  1
verification_status         1
loan_status                 1
purpose                     1
title                      12
zip_code                    1
addr_state                  1
dti                         1
earliest_cr_line            1
inq_last_6mths              1
open_acc                    1
pub_rec                     1
revol_bal                   1
total_acc                   1
total_pymnt                 1
total_rec_prncp             1
total_rec_int               1
last_pymnt_amnt             1
pub_rec_bankruptcies      698
issue_d_year                1
issue_d_month               1
```

```
last_pymnt_d_year            72
last_pymnt_d_month           72
last_credit_pull_d_year       3
last_credit_pull_d_month      3
dtype: int64
```

```
df = df.dropna(axis = 0, how ='any')
```

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

```
loan_amnt                   0
term                        0
int_rate                    0
installment                 0
grade                       0
sub_grade                   0
emp_title                   0
emp_length                  0
home_ownership              0
annual_inc                  0
verification_status         0
loan_status                 0
purpose                     0
title                       0
zip_code                    0
addr_state                  0
dti                         0
earliest_cr_line            0
inq_last_6mths              0
open_acc                    0
pub_rec                     0
revol_bal                   0
total_acc                   0
total_pymnt                 0
total_rec_prncp             0
total_rec_int               0
last_pymnt_amnt             0
pub_rec_bankruptcies        0
issue_d_year                0
issue_d_month               0
last_pymnt_d_year           0
last_pymnt_d_month          0
last_credit_pull_d_year     0
last_credit_pull_d_month    0
dtype: int64
```

```
report = sv.analyze(df)

# Generating report
report.show_html('eda_report_6.html')
```

```
Done! Use 'show' commands to display/save.
Report eda_report_6.html was generated! NOTEBOOK/COLAB USERS: the web browser MAY no
```

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 36536 entries, 1 to 39749
Data columns (total 34 columns):
 #   Column                    Non-Null Count  Dtype  
---  ------                    --------------  -----  
 0   loan_amnt                 36536 non-null  float64
 1   term                      36536 non-null  object 
 2   int_rate                  36536 non-null  float64
 3   installment               36536 non-null  float64
 4   grade                     36536 non-null  object 
 5   sub_grade                 36536 non-null  object 
 6   emp_title                 36536 non-null  object 
 7   emp_length                36536 non-null  object 
 8   home_ownership            36536 non-null  object 
 9   annual_inc                36536 non-null  float64
 10  verification_status       36536 non-null  object 
 11  loan_status               36536 non-null  object 
 12  purpose                   36536 non-null  object 
 13  title                     36536 non-null  object 
 14  zip_code                  36536 non-null  object 
 15  addr_state                36536 non-null  object 
 16  dti                       36536 non-null  float64
 17  earliest_cr_line          36536 non-null  float64
 18  inq_last_6mths            36536 non-null  float64
 19  open_acc                  36536 non-null  float64
 20  pub_rec                   36536 non-null  float64
 21  revol_bal                 36536 non-null  float64
 22  total_acc                 36536 non-null  float64
 23  total_pymnt               36536 non-null  float64
 24  total_rec_prncp           36536 non-null  float64
 25  total_rec_int             36536 non-null  float64
 26  last_pymnt_amnt           36536 non-null  float64
 27  pub_rec_bankruptcies      36536 non-null  float64
 28  issue_d_year              36536 non-null  float64
 29  issue_d_month             36536 non-null  float64
 30  last_pymnt_d_year         36536 non-null  float64
 31  last_pymnt_d_month        36536 non-null  float64
 32  last_credit_pull_d_year   36536 non-null  float64
 33  last_credit_pull_d_month  36536 non-null  float64
dtypes: float64(22), object(12)
memory usage: 9.8+ MB
```

```
report = sv.analyze(df)

# Generating report
report.show_html('eda_report_5.html')
```

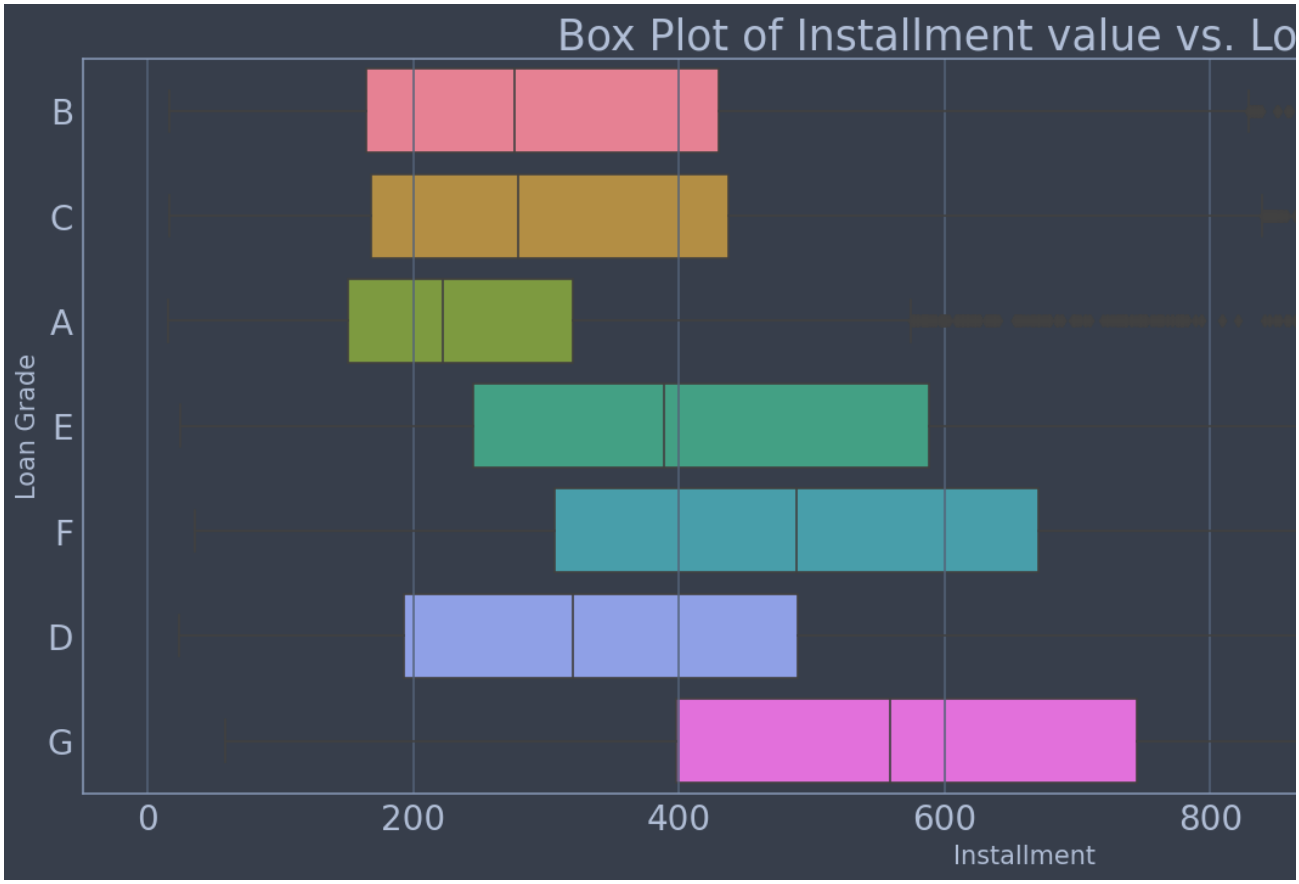
```
Done! Use 'show' commands to display/save.
```

```
Report eda_report_5.html was generated! NOTEBOOK/COLAB USERS: the web browser MAY no
```

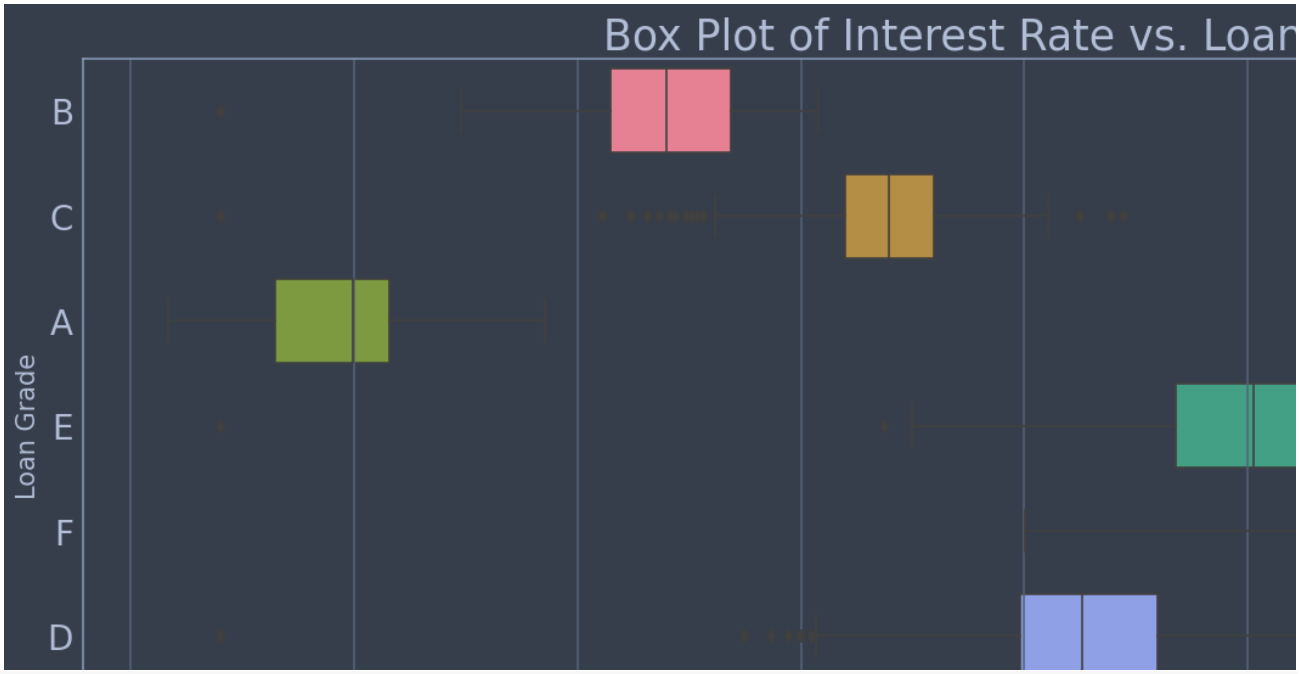
Loan Grades

```
# Plot the borrower's installments compared to load grade
```

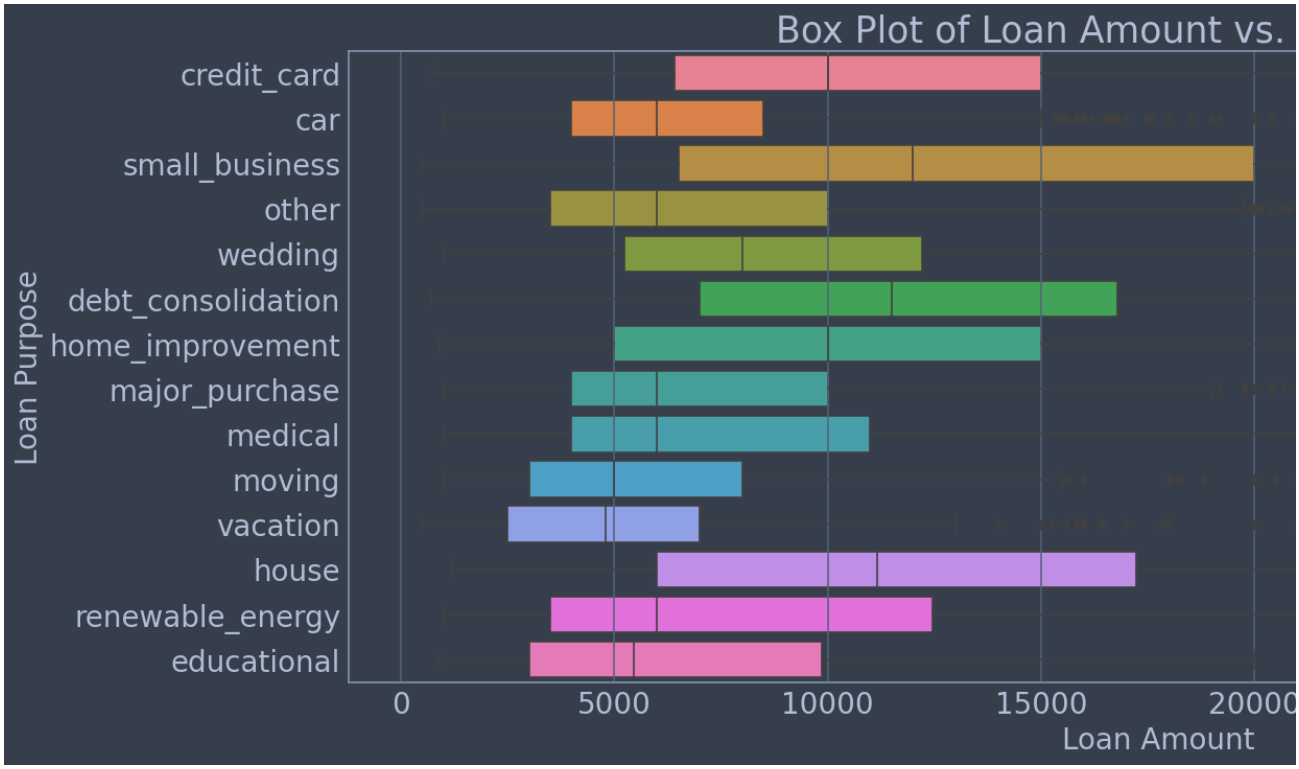
```
plt.figure(figsize=(20,8))
sns.boxplot(x="installment", y="grade", data=df, palette="husl")
plt.title('Box Plot of Installment value vs. Loan Grade', fontsize=25)
plt.xlabel('Installment', fontsize=15)
plt.ylabel('Loan Grade', fontsize=15)
plt.xticks(fontsize=20)
plt.yticks(fontsize=20)
plt.show()
```




```
# Plot the borrower's installments compared to load grade
plt.figure(figsize=(20,8))
sns.boxplot(x="int_rate", y="grade", data=df, palette="husl")
plt.title('Box Plot of Interest Rate vs. Loan Grade', fontsize=25)
plt.xlabel('Installment', fontsize=15)
plt.ylabel('Loan Grade', fontsize=15)
plt.xticks(fontsize=20)
plt.yticks(fontsize=20)
plt.show()
```




```
# Plot the borrower's Loan Amount compared to Loan Purpose
plt.figure(figsize=(20,8))
sns.boxplot(x="loan_amnt", y="purpose", data=df, palette="husl")
plt.title('Box Plot of Loan Amount vs. Loan Purpose', fontsize=25)
plt.xlabel('Loan Amount', fontsize=20)
plt.ylabel('Loan Purpose', fontsize=20)
plt.xticks(fontsize=20)
plt.yticks(fontsize=20)
plt.show()
```



Converting categorical columns to Numerical using Orinal Encoder

```
col_lst_unique = df.columns[df.dtypes == 'object']
col_lst_unique
```

```
Index(['term', 'grade', 'sub_grade', 'emp_title', 'emp_length',
       'home_ownership', 'verification_status', 'loan_status', 'purpose',
       'title', 'zip_code', 'addr_state'],
      dtype='object')
```

```
col_lst_unique = df.columns[df.dtypes == 'object']

ord_enc = OrdinalEncoder()
for col in col_lst_unique:
  df[col] = ord_enc.fit_transform(df[[col]])
```

```
df.head(10)
```

	loan_amnt	term	int_rate	installment	grade	sub_grade	emp_title	emp_length
1	2500.0	1.0	15.27	59.83	2.0	13.0	18641.0	10.0
3	10000.0	0.0	13.49	339.31	2.0	10.0	326.0	1.0
4	3000.0	1.0	12.69	67.79	1.0	9.0	23239.0	0.0
5	5000.0	0.0	7.90	156.46	0.0	3.0	23621.0	3.0
6	7000.0	1.0	15.96	170.08	2.0	14.0	20149.0	8.0
7	3000.0	0.0	18.64	109.43	4.0	20.0	13425.0	9.0
9	5375.0	1.0	12.69	121.45	1.0	9.0	20497.0	10.0
10	6500.0	1.0	14.65	153.45	2.0	12.0	20176.0	5.0
11	12000.0	0.0	12.69	402.54	1.0	9.0	22654.0	1.0
12	9000.0	0.0	13.49	305.38	2.0	10.0	23516.0	10.0

Data Warehousing

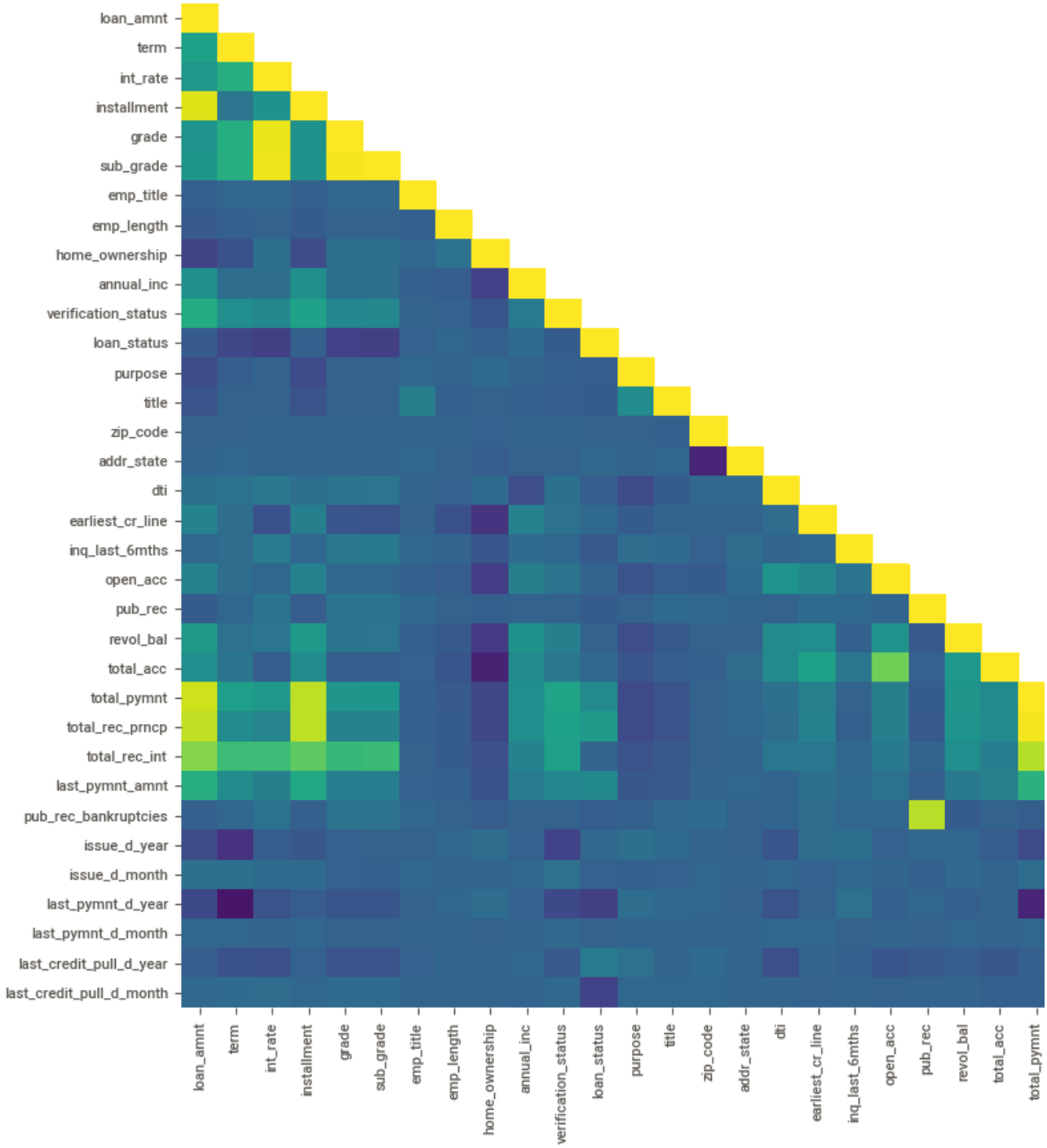
```
df.to_csv('Cleaned_dataset.csv')
```

Heat Map

```
#Heat Map
plt.figure(figsize=(8, 8))

corr_mat = df.corr()
np.tril(np.ones(corr_mat.shape)).astype(np.bool)[0:5,0:5]
```

```
df_lt = corr_mat.where(np.tril(np.ones(corr_mat.shape)).astype(np.bool))
plt.subplots(figsize=(15,10))
sns.heatmap(df_lt, annot=False, cmap="viridis")
plt.show()
```

```
<Figure size 800x800 with 0 Axes>
```



Model Preparation

```
Y = df['loan_status']
X = df.drop('loan_status', axis = 1)

X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.2, random_state=1)
X_train.shape, X_test.shape, y_train.shape, y_test.shape
```

```
((29228, 33), (7308, 33), (29228,), (7308,))
```

```
df.head()
```

total_pymnt	total_rec_prncp	total_rec_int	last_pymnt_amnt	pub_rec_bankruptcies	i
1014.530000	456.46	435.17	119.66	0.0	
12231.890000	10000.00	2214.92	357.48	0.0	
4066.908161	3000.00	1066.91	67.30	0.0	
5632.210000	5000.00	632.21	161.03	0.0	
10137.840010	7000.00	3137.84	1313.76	0.0	

```
#Normalize Numerical Columns
from sklearn.preprocessing import StandardScaler
columns = ['loan_amnt','int_rate', 'installment', 'emp_title', 'emp_length', 'sub_grade',
           'annual_inc', 'title', 'zip_code', 'addr_state', 'dti', 'earliest_cr_line',
           'open_acc','pub_rec','revol_bal', 'total_acc', 'total_pymnt','total_rec_prncp',
           'total_rec_int', 'last_pymnt_amnt', 'issue_d_year', 'issue_d_month', 'last_pymn
           'last_credit_pull_d_month', 'last_credit_pull_d_year']

stan_scaler = StandardScaler()
for col in columns:
  X_train[[col]] = stan_scaler.fit_transform(X_train[[col]])
  X_test[[col]] = stan_scaler.transform(X_test[[col]])
```

```
X_train.head()
```

	loan_amnt	term	int_rate	installment	grade	sub_grade	emp_title	emp_leng

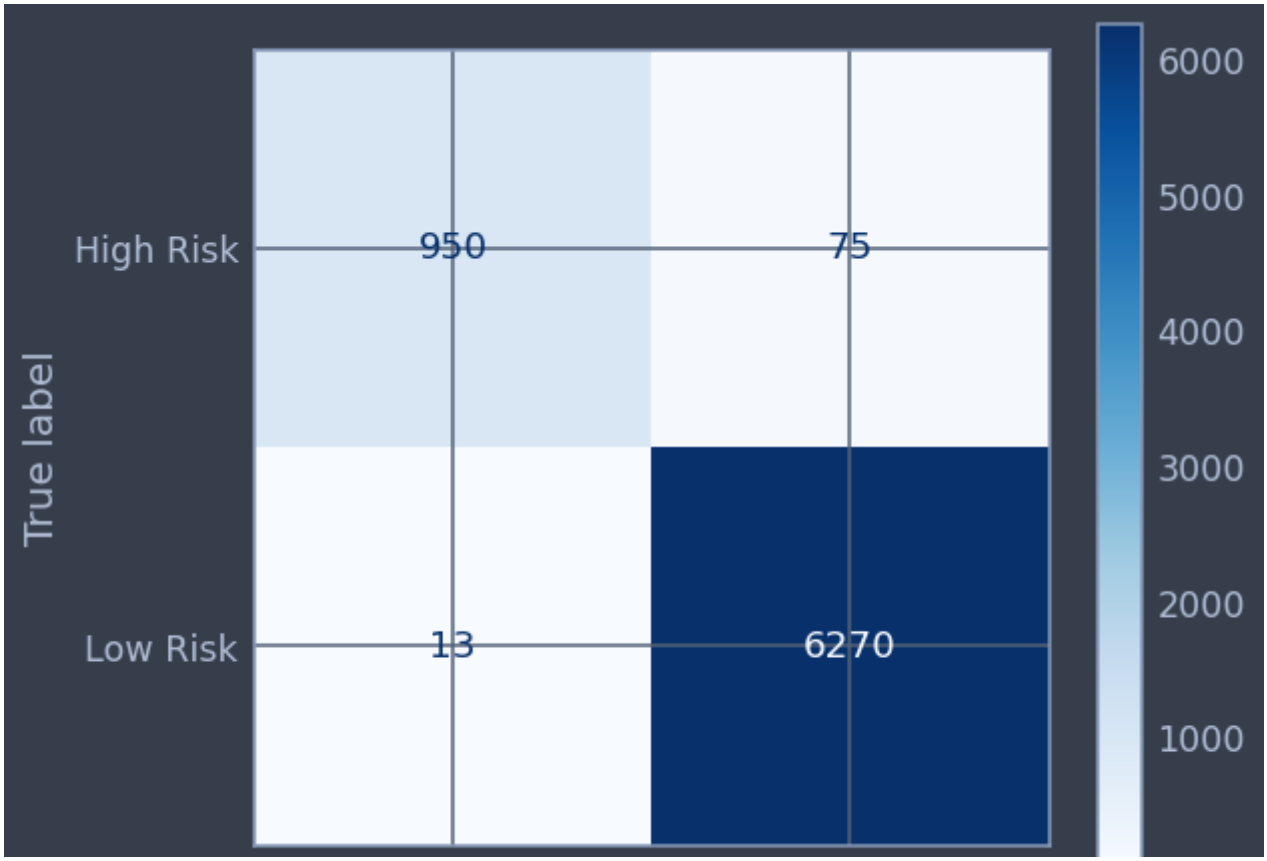
ML Models

Logistic Regression

```
lr = LogisticRegression()
lr.fit(X_train, y_train)
ypred = lr.predict(X_test)
evaluation = f1_score(y_test, ypred)
evaluation
```

```
0.9930313588850175
```

```
disp = plot_confusion_matrix(
    lr, X_test, y_test,
    cmap='Blues', values_format='d',
    display_labels=['High Risk', 'Low Risk']
)

disp = plot_roc_curve(lr, X_test, y_test)
```

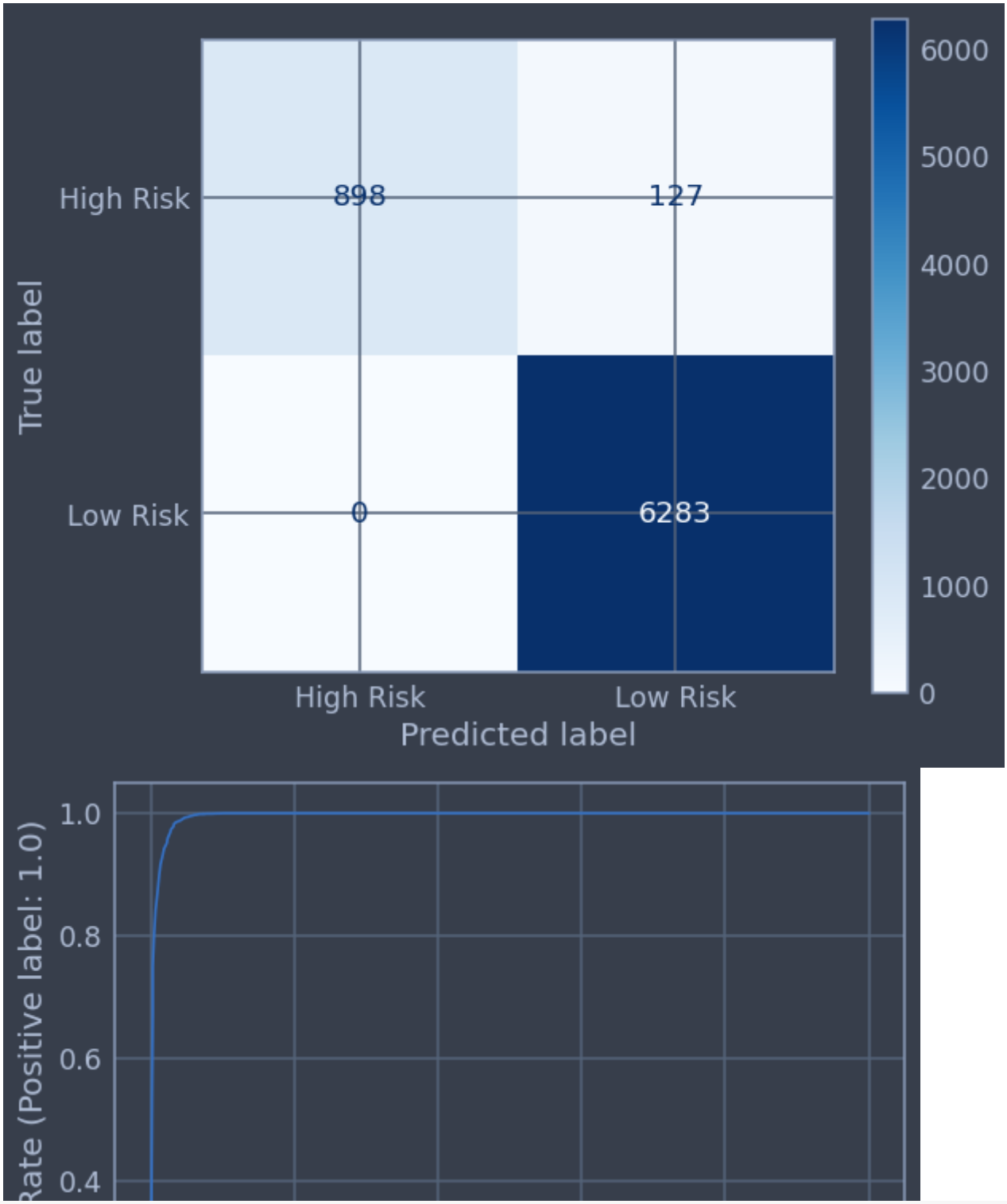


Random Forest

```
rf_clf = RandomForestClassifier()
rf_clf.fit(X_train, y_train)
ypred = rf_clf.predict(X_test)
evaluation = f1_score(y_test, ypred)
evaluation
```

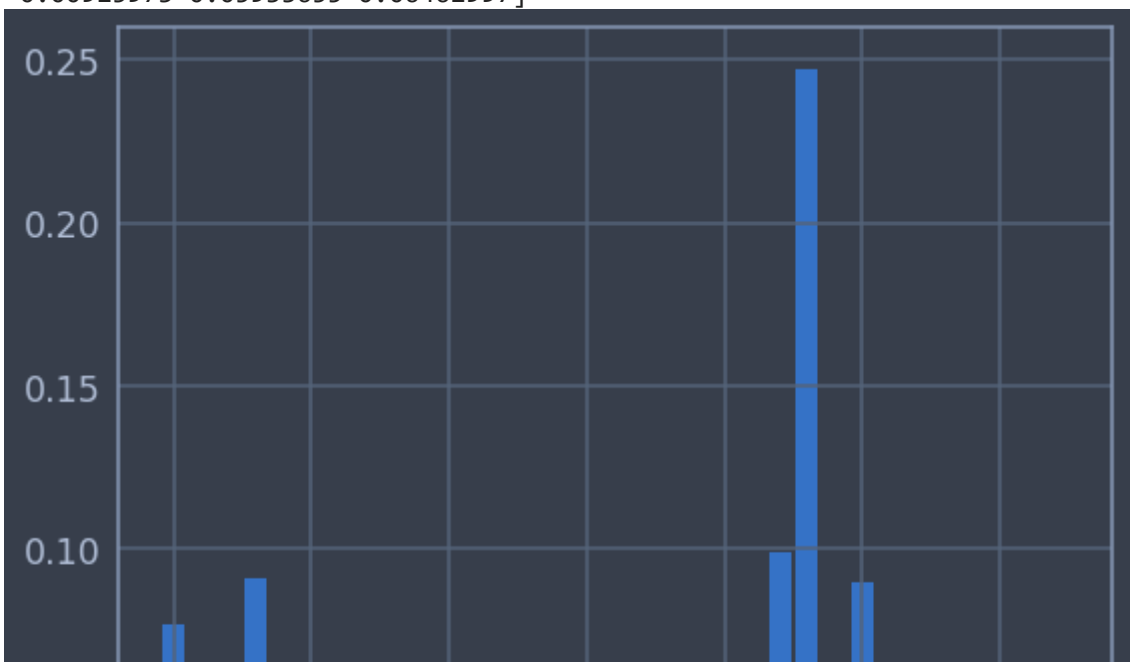
```
0.9904626783321511
```

```
disp = plot_confusion_matrix(
    rf_clf, X_test, y_test,
    cmap='Blues', values_format='d',
    display_labels=['High Risk', 'Low Risk']
)

disp = plot_roc_curve(rf_clf, X_test, y_test)
```




```
# feature importance
print(rf_clf.feature_importances_)
# plot
plt.bar(range(len(rf_clf.feature_importances_)), rf_clf.feature_importances_)
plt.show()
```

```
[0.07703509 0.01405467 0.01801752 0.09108803 0.00795623 0.01258405
 0.00973497 0.0044283  0.00170771 0.0097213  0.00259718 0.00501783
 0.00923641 0.00889334 0.00599039 0.00936921 0.00726185 0.00366224
 0.00606695 0.00084124 0.01006928 0.00772654 0.099202   0.24724599
 0.04079022 0.08953429 0.00078357 0.0106845  0.0066885  0.04878434
 0.00925973 0.05933655 0.06462997]
```

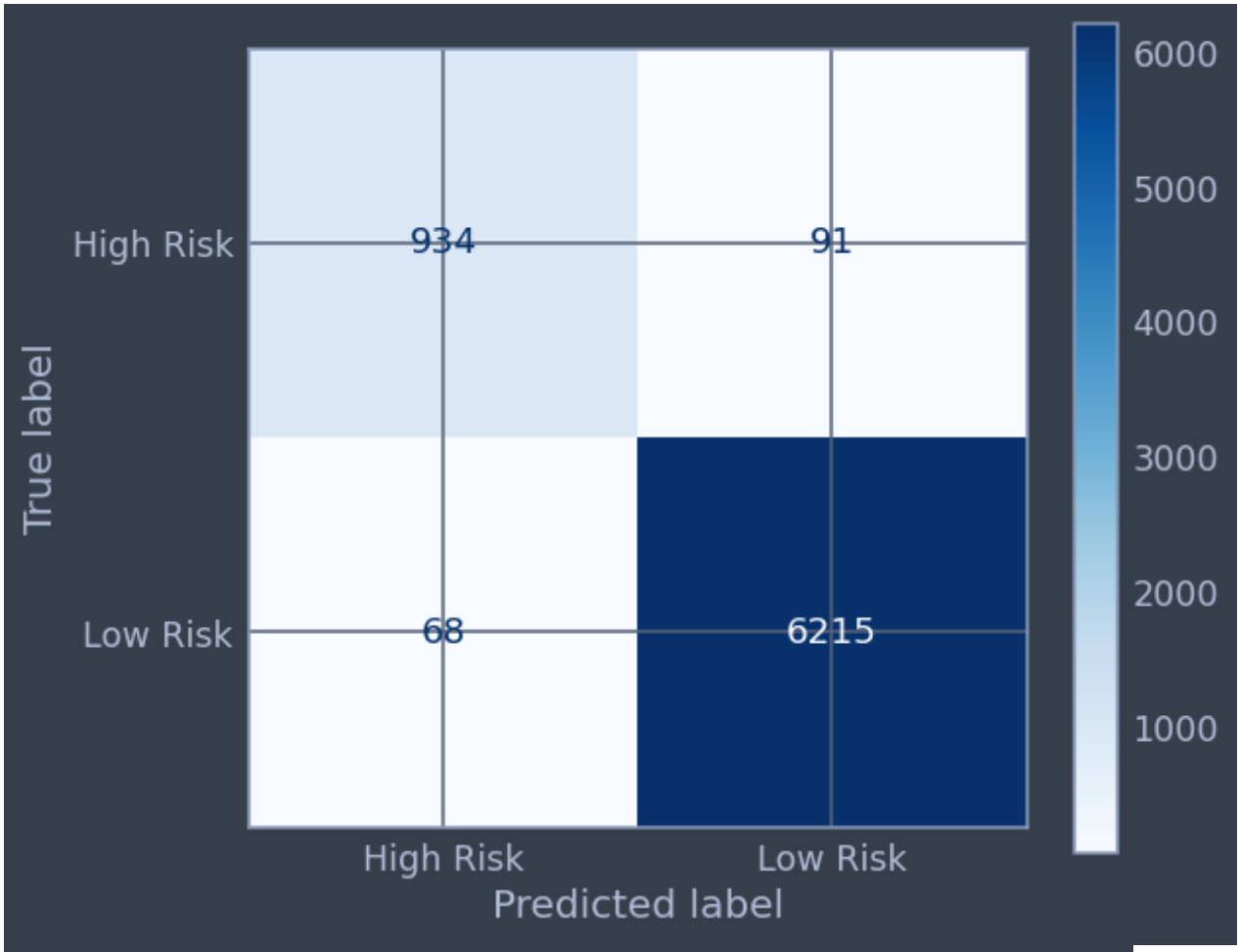


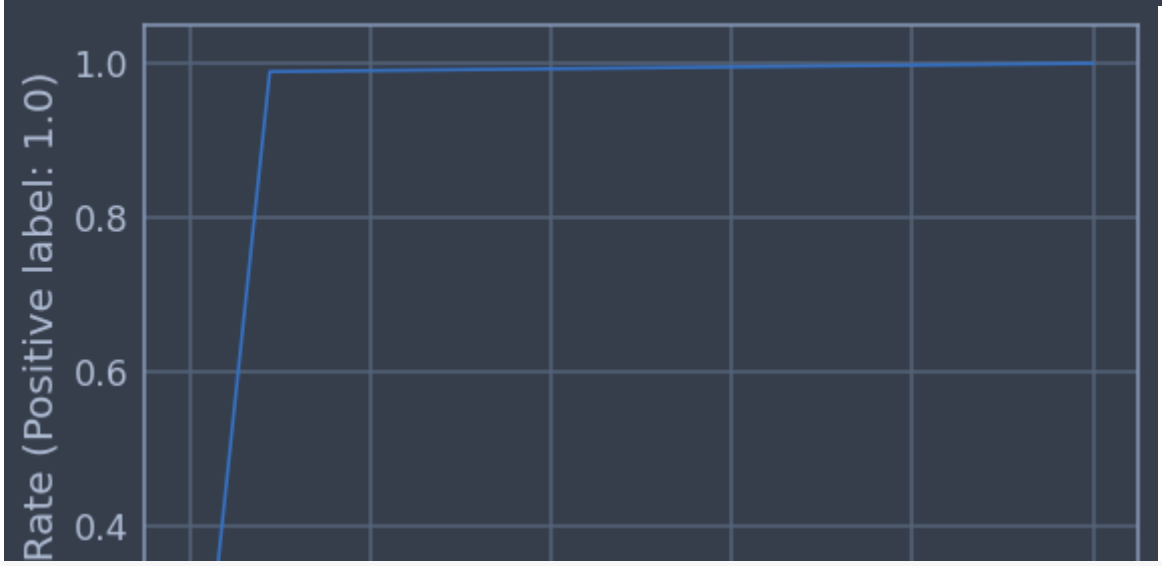
Decision Tree

```
dt = DecisionTreeClassifier()
dt.fit(X_train, y_train)
ypred = dt.predict(X_test)
evaluation = f1_score(y_test, ypred)
evaluation
```

```
0.987369926125983
```

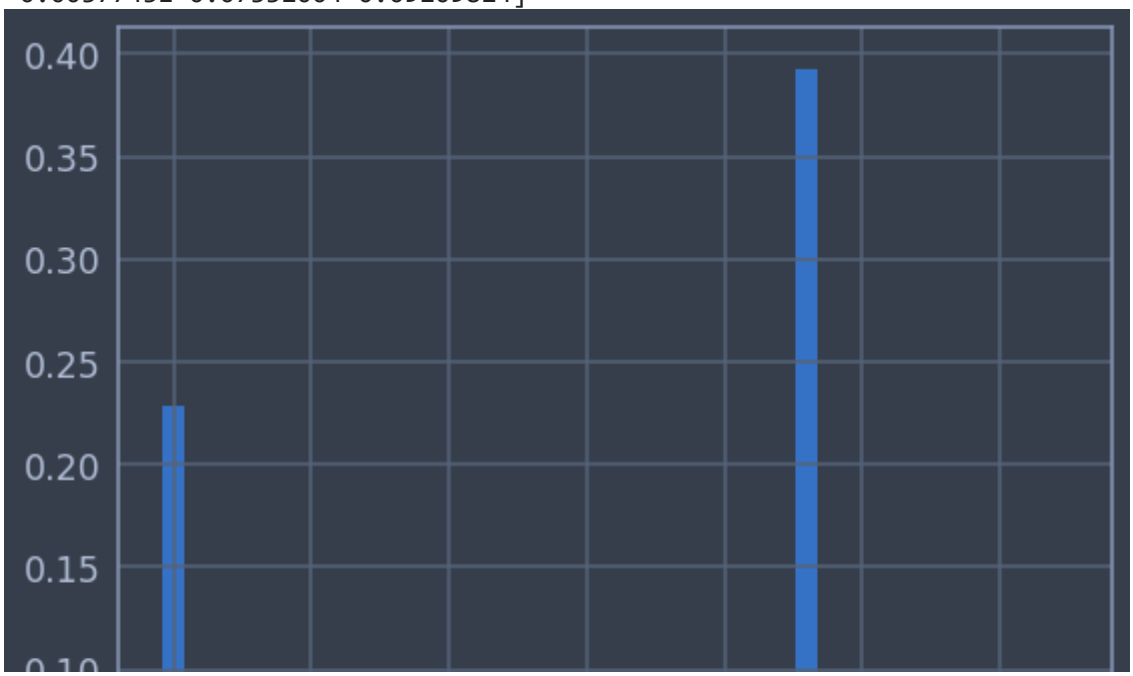
```
disp = plot_confusion_matrix(
    dt, X_test, y_test,
    cmap='Blues', values_format='d',
    display_labels=['High Risk', 'Low Risk']
)

disp = plot_roc_curve(dt, X_test, y_test)
```






```
# feature importance
print(dt.feature_importances_)
# plot
plt.bar(range(len(dt.feature_importances_)), dt.feature_importances_)
plt.show()
```

```
[0.22862789 0.00527173 0.00238766 0.0877383  0.00098576 0.00269363
 0.0025105  0.00154391 0.         0.00419655 0.00078198 0.00266191
 0.00302399 0.0023588  0.00209164 0.00376528 0.00415047 0.00101675
 0.00162067 0.00041293 0.00248835 0.00102917 0.00277892 0.39289863
 0.01777367 0.0362809  0.         0.00104385 0.0020253  0.01664824
 0.00377431 0.07332004 0.09209824]
```

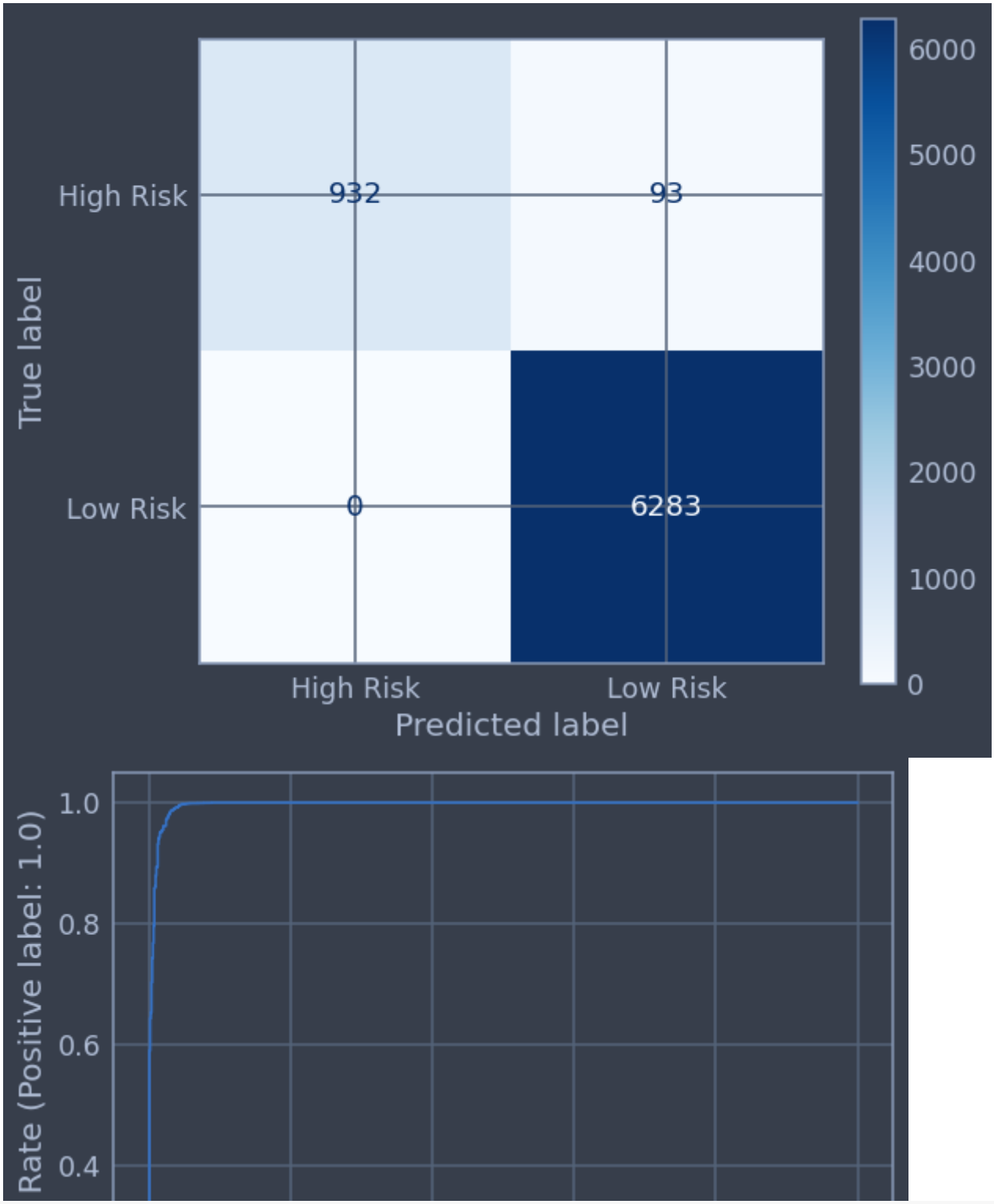


XGBoost

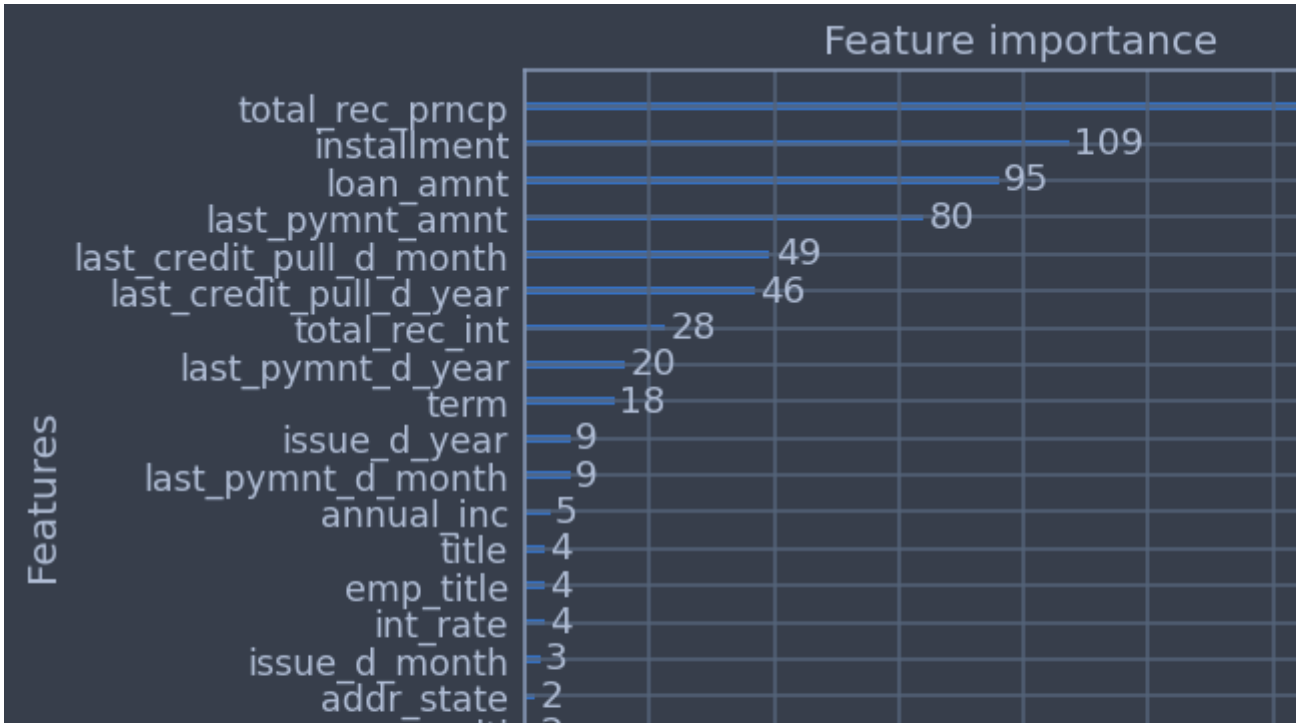
```
xgb_clf = XGBClassifier()
xgb_clf.fit(X_train, y_train)
ypred = xgb_clf.predict(X_test)
evaluation = f1_score(y_test, ypred)
evaluation
```

```
0.9926534481396635
```

```
disp = plot_confusion_matrix(
    xgb_clf, X_test, y_test,
    cmap='Blues', values_format='d',
    display_labels=['High Risk', 'Low Risk']
)

disp = plot_roc_curve(xgb_clf, X_test, y_test)
```




```
from xgboost import plot_importance
# plot feature importance
plot_importance(xgb_clf)
plt.show()
```



Model Deployment

```
from sklearn.compose import ColumnTransformer
pre_process = ColumnTransformer([('scale_data', StandardScaler(),['loan_amnt','funded_amnt
```

```
# define the stages of the pipeline
pipeline = Pipeline(steps= [
                          ('Scaling', pre_process),
                          ('model', LogisticRegression())])

# fit the pipeline model with the training data
pipeline.fit(X_train, y_train)
```

```
/usr/local/lib/python3.7/dist-packages/sklearn/linear_model/_logistic.py:818: Conver
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  extra_warning_msg=_LOGISTIC_SOLVER_CONVERGENCE_MSG,
Pipeline(steps=[('Scaling',
                 ColumnTransformer(transformers=[('scale_data',
                                                  StandardScaler(),
                                                  ['loan_amnt', 'funded_amnt',
                                                   'funded_amnt_inv',
                                                   'int_rate', 'installment',
                                                   'annual_inc', 'dti',
                                                   'delinq_2yrs',
                                                   'inq_last_6mths', 'open_acc',
                                                   'pub_rec', 'revol_bal',
                                                   'revol_util', 'total_acc',
                                                   'total_pymnt',
                                                   'total_pymnt_inv',
                                                   'total_rec_int',
```

```
                                                    'total_rec_late_fee',
                                                    'recoveries',
                                                    'last_pymnt_amnt',
                                                    'pub_rec_bankruptcies'])])),
                ('model', LogisticRegression())])
```

```
pipeline.predict(X_test)
```

```
array([0., 1., 1., ..., 1., 1., 1.])
```

```
# import joblib
from joblib import dump

# dump the pipeline model
dump(pipeline, filename="Identifying_Defaulters.joblib")
```

```
['Identifying_Defaulters.joblib']
```

```
test_file = pd.read_csv('/content/Test_file.csv')
```

```
test_file.head()
```

	loan_amnt	funded_amnt	funded_amnt_inv	term	int_rate	installment	grade	sub_
0	5000	5000	4975	0	10.65	162.87	1	
1	2500	2500	2500	1	15.27	59.83	2	
2	2400	2400	2400	0	15.96	84.33	2	
3	10000	10000	10000	0	13.49	339.31	2	
4	3000	3000	3000	1	12.69	67.79	1	

```
# import joblib
from joblib import load

# load the saved pipleine model
pipeline = load("Identifying_Defaulters.joblib")

# predict on the sample tweet text
pipeline.predict(test_file)
```

```
array([1., 0., 1., 1., 1., 1.])
```

```
!pip install flask-ngrok


from flask import Flask
from flask_ngrok import run_with_ngrok
```

```
app = Flask(__name__)
run_with_ngrok(app)

@app.route("/")
def home():
    return "<h1>GFG is great platform to learn</h1>"

app.run()
```

```
Requirement already satisfied: flask-ngrok in /usr/local/lib/python3.7/dist-packages
Requirement already satisfied: requests in /usr/local/lib/python3.7/dist-packages (f
Requirement already satisfied: Flask>=0.8 in /usr/local/lib/python3.7/dist-packages
Requirement already satisfied: click<8.0,>=5.1 in /usr/local/lib/python3.7/dist-pack
Requirement already satisfied: itsdangerous<2.0,>=0.24 in /usr/local/lib/python3.7/d
Requirement already satisfied: Jinja2<3.0,>=2.10.1 in /usr/local/lib/python3.7/dist-
Requirement already satisfied: Werkzeug<2.0,>=0.15 in /usr/local/lib/python3.7/dist-
Requirement already satisfied: MarkupSafe>=0.23 in /usr/local/lib/python3.7/dist-pac
Requirement already satisfied: chardet<4,>=3.0.2 in /usr/local/lib/python3.7/dist-pa
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.7/dist-p
Requirement already satisfied: idna<3,>=2.5 in /usr/local/lib/python3.7/dist-package
Requirement already satisfied: urllib3!=1.25.0,!=1.25.1,<1.26,>=1.21.1 in /usr/local
 * Serving Flask app "__main__" (lazy loading)
 * Environment: production
   WARNING: This is a development server. Do not use it in a production deployment.
   Use a production WSGI server instead.
 * Debug mode: off
 * Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
 * Running on http://ce99-35-199-40-121.ngrok.io
 * Traffic stats available on http://127.0.0.1:4040
```

✓ 2s completed at 18:15