In [1]:
```
# importing libraries
import numpy as np
import pandas as pd
import matplotlib
import matplotlib.pyplot as plt
import seaborn as sns
```

In [4]:
```
df = pd.read_csv('/home/ec2-user/group15deliverable03.csv')
df.to_csv('group15deliverable03.csv')
```

In [5]:
```
df3=pd.read_csv('./group15deliverable03.csv')
df3
```

Out[5]:

	Unnamed: 0	id	loan_status	loan_amount	funded_amount_by_investors	loa
0	0	1077501	fully paid	5000	4975.0	
1	1	1077430	charged off	2500	2500.0	
2	2	1077175	fully paid	2400	2400.0	
3	3	1076863	fully paid	10000	10000.0	
4	4	1075358	current	3000	3000.0	
...	...	...	...	...	...	
39712	39712	92187	fully paid	2500	1075.0	
39713	39713	90665	fully paid	8500	875.0	
39714	39714	90395	fully paid	5000	1325.0	
39715	39715	90376	fully paid	5000	650.0	
39716	39716	87023	fully paid	7500	800.0	

39717 rows × 22 columns

In [6]:
```
# printing the head of the dataset
df3.head()
```

Out[6]:

	Unnamed: 0	id	loan_status	loan_amount	funded_amount_by_investors	loan_ter
0	0	1077501	fully paid	5000	4975.0	
1	1	1077430	charged off	2500	2500.0	
2	2	1077175	fully paid	2400	2400.0	
3	3	1076863	fully paid	10000	10000.0	
4	4	1075358	current	3000	3000.0	

5 rows × 22 columns

In [7]:
```
# Total number of rows in the dataset
len(df3)
```

Out[7]: 39717

In [8]:
```
# We want to predict if a particular loan was fully paid or charged off
df3['loan_status'].value_counts()
```

Out[8]:
```
fully paid     32950
charged off     5627
current         1140
Name: loan_status, dtype: int64
```

In [9]:
```
# Drop rows which have loan_status other than Fully Paid and Charged Off
df3 = df3[(df3['loan_status'] == 'fully paid') | (df3['loan_status'] == '
```

In [10]:
```
# check number of df after dropping rows. it was 39717 initially
len(df3)
```

Out[10]: 38577

In [11]:
```
df3
```

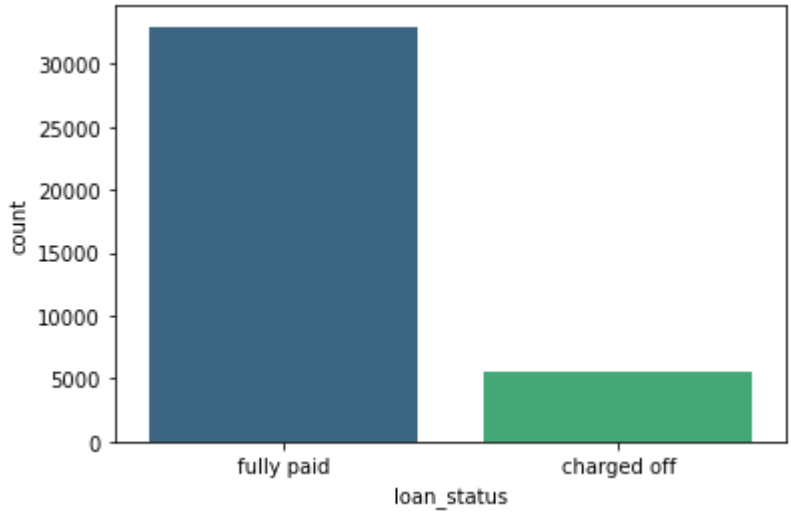
Out[11]:

	Unnamed: 0	id	loan_status	loan_amount	funded_amount_by_investors	loa
0	0	1077501	fully paid	5000	4975.0	
1	1	1077430	charged off	2500	2500.0	
2	2	1077175	fully paid	2400	2400.0	
3	3	1076863	fully paid	10000	10000.0	
5	5	1075269	fully paid	5000	5000.0	
...	...	...	...	...	...	
39712	39712	92187	fully paid	2500	1075.0	
39713	39713	90665	fully paid	8500	875.0	
39714	39714	90395	fully paid	5000	1325.0	
39715	39715	90376	fully paid	5000	650.0	
39716	39716	87023	fully paid	7500	800.0	

38577 rows × 22 columns

In [12]:
```
# check if loan_status now has only fully paid and charged off
sns.countplot(x=df3['loan_status'], data=df3, palette='viridis')
```

Out[12]:
```
<AxesSubplot:xlabel='loan_status', ylabel='count'>
```



Data Cleaning

In [13]:
```
#Let's find the % of null values in each column sorted in descending orde
((df3.isnull().sum()/len(df3))*100).sort_values(ascending=False)
```

Out[13]:
```
employment_length                 2.677761
revolving_line_utilization_rate   0.129611
Unnamed: 0                        0.000000
id                                0.000000
home_ownership                    0.000000
total_credit_lines                0.000000
open_credit_lines                 0.000000
inquiries_last_6_months           0.000000
earliest_credit_line              0.000000
dti                               0.000000
purpose                           0.000000
issued_on                         0.000000
verification_status               0.000000
sub_grade                         0.000000
grade                             0.000000
installment                       0.000000
interest_rate                     0.000000
loan_term                         0.000000
funded_amount_by_investors        0.000000
loan_amount                       0.000000
loan_status                       0.000000
annual_income                     0.000000
dtype: float64
```

In [14]:
```
# visualizing null rows
df4=df3[df3['employment_length'].isnull()]
df4
```

Out[14]:

	Unnamed: 0	id	loan_status	loan_amount	funded_amount_by_investors	loa
168	168	1067644	charged off	2500	2500.00000	
323	323	1058717	fully paid	18000	18000.00000	
394	394	1064582	fully paid	4000	4000.00000	
422	422	1064366	charged off	1000	1000.00000	
439	439	1063912	fully paid	8250	8250.00000	
...	...	...	...	...	...	
32591	32591	480410	fully paid	10000	9975.00000	
32608	32608	480216	fully paid	3700	3700.00000	
32621	32621	479954	charged off	10000	9731.17513	
32631	32631	479836	fully paid	6000	6000.00000	
32665	32665	479468	fully paid	25000	23912.90328	

1033 rows × 22 columns

In [15]:
```
# drop all null rows
df5=df3.dropna()
df5
```

Out[15]:

	Unnamed: 0	id	loan_status	loan_amount	funded_amount_by_investors	loa
0	0	1077501	fully paid	5000	4975.0	
1	1	1077430	charged off	2500	2500.0	
2	2	1077175	fully paid	2400	2400.0	
3	3	1076863	fully paid	10000	10000.0	
5	5	1075269	fully paid	5000	5000.0	
...	...	...	...	...	...	
39712	39712	92187	fully paid	2500	1075.0	
39713	39713	90665	fully paid	8500	875.0	
39714	39714	90395	fully paid	5000	1325.0	
39715	39715	90376	fully paid	5000	650.0	
39716	39716	87023	fully paid	7500	800.0	

37497 rows × 22 columns

In [16]:
```
# there is no null values
((df5.isnull().sum()/len(df5))*100).sort_values(ascending=False)
```

Out[16]:
```
Unnamed: 0                          0.0
id                                  0.0
home_ownership                      0.0
employment_length                   0.0
total_credit_lines                  0.0
revolving_line_utilization_rate     0.0
open_credit_lines                   0.0
inquiries_last_6_months             0.0
earliest_credit_line                0.0
dti                                 0.0
purpose                             0.0
issued_on                           0.0
verification_status                 0.0
sub_grade                           0.0
grade                               0.0
installment                         0.0
interest_rate                       0.0
loan_term                           0.0
funded_amount_by_investors          0.0
loan_amount                         0.0
loan_status                         0.0
annual_income                       0.0
dtype: float64
```

Preprocessing and Exploratory Analysis

In [17]:
```
df5
```

Out[17]:

	Unnamed: 0	id	loan_status	loan_amount	funded_amount_by_investors	loa
0	0	1077501	fully paid	5000	4975.0	
1	1	1077430	charged off	2500	2500.0	
2	2	1077175	fully paid	2400	2400.0	
3	3	1076863	fully paid	10000	10000.0	
5	5	1075269	fully paid	5000	5000.0	
...	...	...	...	...	...	
39712	39712	92187	fully paid	2500	1075.0	
39713	39713	90665	fully paid	8500	875.0	
39714	39714	90395	fully paid	5000	1325.0	
39715	39715	90376	fully paid	5000	650.0	
39716	39716	87023	fully paid	7500	800.0	

37497 rows × 22 columns

In [18]:
```
df5.dtypes.value_counts()
```

Out[18]:
```
object     8
int64      7
float64    7
dtype: int64
```

In [19]:
```
#There are 8 categorical features currently.
```

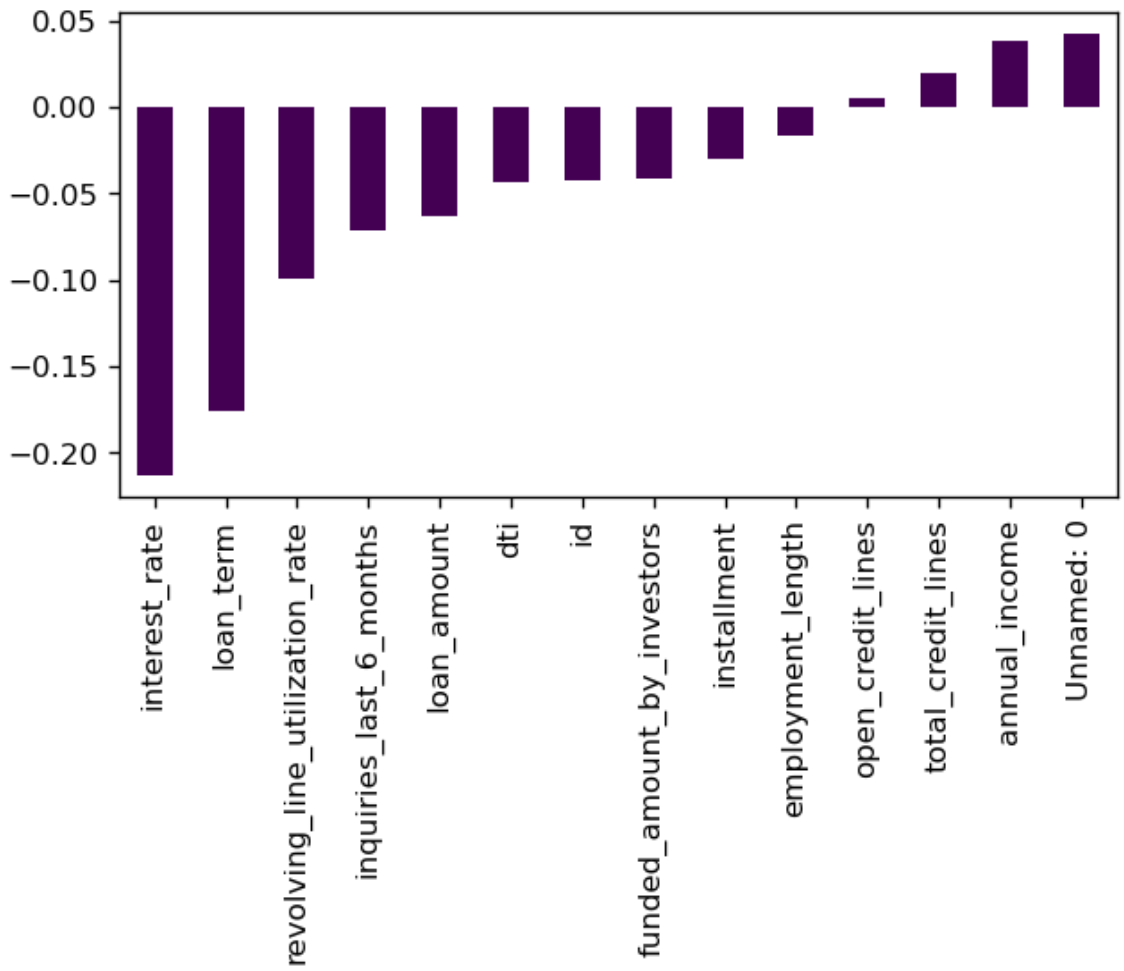
In [20]:
```
df5['loan_status']
```

Out[20]:
```
0         fully paid
1        charged off
2         fully paid
3         fully paid
5         fully paid
            ...
39712     fully paid
39713     fully paid
39714     fully paid
39715     fully paid
39716     fully paid
Name: loan_status, Length: 37497, dtype: object
```

In [21]:
```
# Create dummies for loan_status so that correlation can be calculated wr
#for other continuous features.
```

In [22]:
```
df_temp = df5.copy() # copy so that it does not affect the original data
df_temp['loan_status'] = pd.get_dummies(df_temp['loan_status'], drop_firs
```

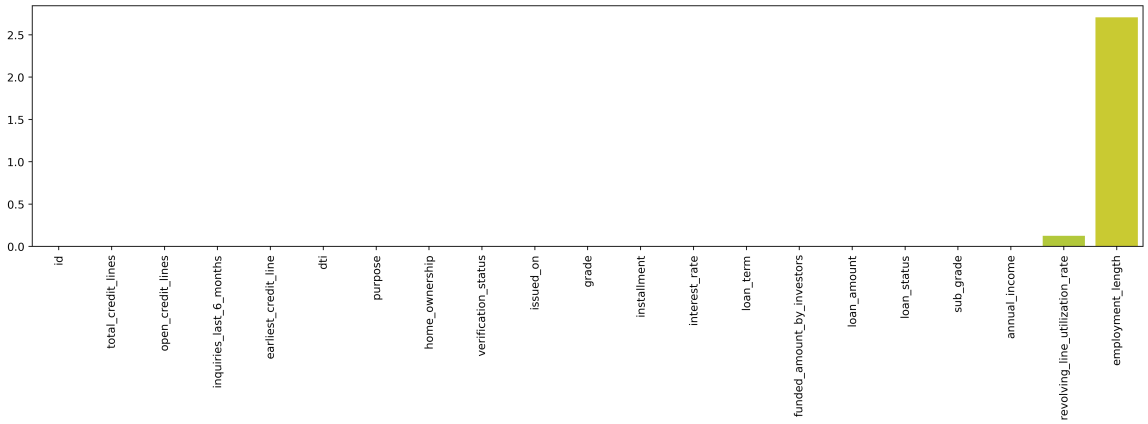
In [23]:
```
plt.figure(figsize=(6,3),dpi=120)
df_temp.corr()['loan_status'].sort_values().drop('loan_status').plot(kind
plt.xticks(rotation=90);
```



In [24]:
```
# interest_rate, loan_term are highly correlated with loan_status compare
```

In [25]:
```
# calculate features with missing values.
```

In [26]:
```
plt.figure(figsize=(18,4),dpi=400)
sns.barplot(y=((df.isnull().sum()/len(df))*100).sort_values(), x=((df.isn
plt.xticks(rotation=90);
```



In [27]:
```
# Columns with missing values and % of missing values.
```

In [28]:
```
df_missing = ((df.isnull().sum()/len(df))*100)[((df.isnull().sum()/len(df
df_missing
```

Out[28]:
```
revolving_line_utilization_rate    0.125891
employment_length                  2.706650
dtype: float64
```

In [31]:
```
df5
```

Out[31]:

	Unnamed: 0	id	loan_status	loan_amount	funded_amount_by_investors	loa
0	0	1077501	fully paid	5000	4975.0	
1	1	1077430	charged off	2500	2500.0	
2	2	1077175	fully paid	2400	2400.0	
3	3	1076863	fully paid	10000	10000.0	
5	5	1075269	fully paid	5000	5000.0	
...	...	...	...	...	...	
39712	39712	92187	fully paid	2500	1075.0	
39713	39713	90665	fully paid	8500	875.0	
39714	39714	90395	fully paid	5000	1325.0	
39715	39715	90376	fully paid	5000	650.0	
39716	39716	87023	fully paid	7500	800.0	

37497 rows × 22 columns

In [32]:
```
df5['loan_status'] = df5['loan_status'].map({'fully paid':1,'charged off'
df5
```

```
/tmp/ipykernel_11002/87717867.py:1: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  df5['loan_status'] = df5['loan_status'].map({'fully paid':1,'charged off':0})
```

Out[32]:

	Unnamed: 0	id	loan_status	loan_amount	funded_amount_by_investors	loa
0	0	1077501	1	5000	4975.0	
1	1	1077430	0	2500	2500.0	
2	2	1077175	1	2400	2400.0	
3	3	1076863	1	10000	10000.0	
5	5	1075269	1	5000	5000.0	
...	...	...	...	...	...	
39712	39712	92187	1	2500	1075.0	
39713	39713	90665	1	8500	875.0	
39714	39714	90395	1	5000	1325.0	
39715	39715	90376	1	5000	650.0	
39716	39716	87023	1	7500	800.0	

37497 rows × 22 columns

In [33]:
```
df6=df5[["loan_status","loan_amount","funded_amount_by_investors","loan_t
df6
```

Out[33]:

	loan_status	loan_amount	funded_amount_by_investors	loan_term	interest_rate
0	1	5000	4975.0	36	10.65
1	0	2500	2500.0	60	15.27
2	1	2400	2400.0	36	15.96
3	1	10000	10000.0	36	13.49
5	1	5000	5000.0	36	7.90
...	...	...	...	...	...
39712	1	2500	1075.0	36	8.07
39713	1	8500	875.0	36	10.28
39714	1	5000	1325.0	36	8.07
39715	1	5000	650.0	36	7.43
39716	1	7500	800.0	36	13.75

37497 rows × 13 columns

Scaling and Test Train split

In [34]:
```
df6.shape
```

Out[34]:
```
(37497, 13)
```

In [35]:
```
X = df6.drop('loan_status', axis=1)
```

In [36]:
```
y = df6['loan_status']
```

In [37]:
```
from sklearn.model_selection import train_test_split
```

In [38]:
```
# keeping 15% for test
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)
```

Scaling

In [39]:
```
from sklearn.preprocessing import MinMaxScaler
```

In [40]:
```
scaler = MinMaxScaler()
```

In [41]:
```
X.head()
```

Out[41]:

	loan_amount	funded_amount_by_investors	loan_term	interest_rate	installment	d
0	5000	4975.0	36	10.65	162.87	27.6
1	2500	2500.0	60	15.27	59.83	1.0
2	2400	2400.0	36	15.96	84.33	8.7
3	10000	10000.0	36	13.49	339.31	20.0
5	5000	5000.0	36	7.90	156.46	11.2

In [42]:
```
y.head()
```

Out[42]:
```
0    1
1    0
2    1
3    1
5    1
Name: loan_status, dtype: int64
```

In [43]:
```
X_train = scaler.fit_transform(X_train)
```

In [44]:
```
X_test = scaler.transform(X_test)
```

Creating Models

In [ ]:
```
# We will try out these models:
#XGBoost
#Random Forests
```

XGBoost

In [46]:
```
pip install xgboost
```

```
Looking in indexes: https://pypi.org/simple, https://pip.repos.neuron.amazonaws.com
Collecting xgboost
  Downloading xgboost-1.7.2-py3-none-manylinux2014_x86_64.whl (193.6 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 193.6/193.6 MB 11.0 MB/s eta 0:00:0000:0100:01
Requirement already satisfied: numpy in /home/ec2-user/anaconda3/envs/python3/lib/python3.8/site-packages (from xgboost) (1.20.3)
Requirement already satisfied: scipy in /home/ec2-user/anaconda3/envs/python3/lib/python3.8/site-packages (from xgboost) (1.5.3)
Installing collected packages: xgboost
Successfully installed xgboost-1.7.2
WARNING: You are using pip version 22.0.4; however, version 22.3.1 is available.
You should consider upgrading via the '/home/ec2-user/anaconda3/envs/python3/bin/python -m pip install --upgrade pip' command.
Note: you may need to restart the kernel to use updated packages.
```

In [47]:
```
from xgboost import XGBClassifier
```

In [48]:
```
# fit model to training data
model = XGBClassifier()
model.fit(X_train, y_train)
```

Out[48]:
```
XGBClassifier(base_score=0.5, booster='gbtree', callbacks=None,
              colsample_bylevel=1, colsample_bynode=1, colsample_bytree=1,
              early_stopping_rounds=None, enable_categorical=False,
              eval_metric=None, feature_types=None, gamma=0, gpu_id=-1,
              grow_policy='depthwise', importance_type=None,
              interaction_constraints='', learning_rate=0.300000012,
              max_bin=256, max_cat_threshold=64, max_cat_to_onehot=4,
              max_delta_step=0, max_depth=6, max_leaves=0, min_child_weight=1,
              missing=nan, monotone_constraints='()', n_estimators=100,
              n_jobs=0, num_parallel_tree=1, predictor='auto', random_state=0, ...)
```

In [49]:
```
preds = model.predict(X_test)
```

In [50]:
```
from sklearn.metrics import accuracy_score,plot_confusion_matrix,classifi
```

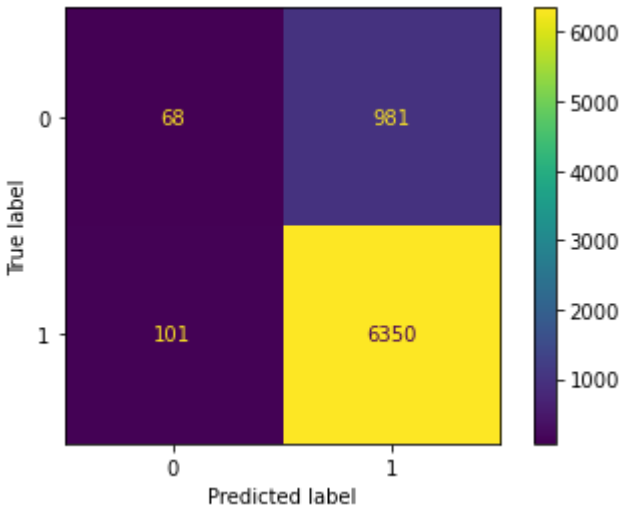
In [51]:
```
print(classification_report(y_test,preds))
```

```
              precision    recall  f1-score   support

           0       0.40      0.06      0.11      1049
           1       0.87      0.98      0.92      6451

    accuracy                           0.86      7500
   macro avg       0.63      0.52      0.52      7500
weighted avg       0.80      0.86      0.81      7500
```

In [52]:
```
plot_confusion_matrix(model,X_test,y_test)
```

```
/home/ec2-user/anaconda3/envs/python3/lib/python3.8/site-packages/sklearn/utils/deprecation.py:87: FutureWarning: Function plot_confusion_matrix is deprecated; Function `plot_confusion_matrix` is deprecated in 1.0 and will be removed in 1.2. Use one of the class methods: ConfusionMatrixDisplay.from_predictions or ConfusionMatrixDisplay.from_estimator.
  warnings.warn(msg, category=FutureWarning)
```

Out[52]:
```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x7f4f5b059280>
```



Random Forests

In [53]:
```
from sklearn.ensemble import RandomForestClassifier
```

In [54]:
```
rf = RandomForestClassifier(n_estimators=100)
```

In [55]:
```
rf.fit(X_train,y_train)
```

Out[55]:
```
RandomForestClassifier()
```

In [56]:
```
preds = rf.predict(X_test)
```

In [57]:
```
print(classification_report(y_test,preds))
```

```
              precision    recall  f1-score   support

           0       0.37      0.03      0.05      1049
           1       0.86      0.99      0.92      6451

    accuracy                           0.86      7500
   macro avg       0.62      0.51      0.49      7500
weighted avg       0.79      0.86      0.80      7500
```

In [58]:
```
plot_confusion_matrix(rf,X_test,y_test)
```

```
/home/ec2-user/anaconda3/envs/python3/lib/python3.8/site-packages/sklear
n/utils/deprecation.py:87: FutureWarning: Function plot_confusion_matrix
is deprecated; Function `plot_confusion_matrix` is deprecated in 1.0 and
will be removed in 1.2. Use one of the class methods: ConfusionMatrixDis
play.from_predictions or ConfusionMatrixDisplay.from_estimator.
  warnings.warn(msg, category=FutureWarning)
```

Out[58]: <sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x7f4f4596ae20>