In [ ]:

```
import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

import scipy.stats as st

import numpy as np

import plotly.express as px

sns.set_context('talk')

plt.style.use('fivethirtyeight')
```

In [ ]:

```
#Define Helper Functions

#outlier detection Tukey's
def tukeys_fences(df,col):

    lower, upper = df[col].quantile([.25,.75]).values

    iqr = upper - lower

    lower_lim = lower - iqr * 2.5

    upper_lim = upper + iqr * 2.5

    mask = (df[col] > upper_lim) | (df[col] < lower_lim)

    print(f'PCT of Outliers Detected in {col}: {mask.sum()/len(df):.3f}%')

    df.loc[mask,col] = np.nan

    return df

#Z-Score Outlier detection

def z_outlier(df,col,thresh=3,append=False):

    from scipy.stats import zscore

    #locate and flag outliers using zscore
    mask = zscore(df[col]).abs() > thresh

    #append zscore of column to dataframe if "append" = True

    if append == True:

        df[f'z_score_of_{col}'] = zscore(df[col])

    return mask
```

In [ ]:

```
from sklearn.model_selection import train_test_split
from sklearn.linear_model import Ridge, ElasticNet, LinearRegression
from sklearn.ensemble import RandomForestRegressor, AdaBoostRegressor
from sklearn.svm import SVR
from xgboost import XGBRegressor
from sklearn.metrics import r2_score, mean_squared_error

#regression function
def train_test_score_regression(df,target = None):

    X = df.drop(target, axis=1)
    y = df[target]

    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=.2,

    models = {'Ridge': Ridge(),
              'XGBoost': XGBRegressor(),
              'SVR': SVR(),
              'ElasticNet': ElasticNet(),
              'RandomForest': RandomForestRegressor(),
              'AdaBoost': AdaBoostRegressor(base_estimator=LinearRegression(

    model_results = {}  # Dictionary to store model results

    for model_name, model in models.items():
        model.fit(X_train, y_train)  # Train models

        # Make Predictions
        y_train_preds = model.predict(X_train)
        y_test_preds = model.predict(X_test)

        # Training Set Performance
        model_r2_train = r2_score(y_train, y_train_preds)
        model_r2_test = r2_score(y_test, y_test_preds)

        model_rmse_train = np.sqrt(mean_squared_error(y_train, y_train_preds
        model_rmse_test = np.sqrt(mean_squared_error(y_test, y_test_preds))

        # Store results in the dictionary
        model_results[model_name] = {
            'R^2 Score (Training)': model_r2_train,
            'RMSE (Training)': model_rmse_train,
            'R^2 Score (Testing)': model_r2_test,
            'RMSE (Testing)': model_rmse_test
        }

    # Sort the models by RMSE on the test set
    sorted_models = {k: v for k, v in sorted(model_results.items(), key=lamb

    return sorted_models


from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier, AdaBoostClassifier
```

```python
from sklearn.svm import SVC
from xgboost import XGBClassifier
from sklearn.metrics import accuracy_score, precision_score, recall_score, f

#classification function
def train_test_score_classification(df, target=None):

    X = df.drop(target, axis=1)
    y = df[target]

    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=.2,

    models = {'Logistic Regression': LogisticRegression(),
              'Random Forest': RandomForestClassifier(),
              'AdaBoost': AdaBoostClassifier(),
             # 'SVM': SVC(),
              'XGBoost': XGBClassifier()}

    model_results = {}  # Dictionary to store model results

    for model_name, model in models.items():
        model.fit(X_train, y_train)  # Train models

        # Make Predictions
        y_train_preds = model.predict(X_train)
        y_test_preds = model.predict(X_test)

        # Training Set Performance
        accuracy_train = accuracy_score(y_train, y_train_preds)
        precision_train = precision_score(y_train, y_train_preds)
        recall_train = recall_score(y_train, y_train_preds)
        f1_train = f1_score(y_train, y_train_preds)

        # Test Set Performance
        accuracy_test = accuracy_score(y_test, y_test_preds)
        precision_test = precision_score(y_test, y_test_preds)
        recall_test = recall_score(y_test, y_test_preds)
        f1_test = f1_score(y_test, y_test_preds)

        # Store results in the dictionary
        model_results[model_name] = {
            'Accuracy (Training)': accuracy_train,
            'Precision (Training)': precision_train,
            'Recall (Training)': recall_train,
            'F1 Score (Training)': f1_train,
            'Accuracy (Testing)': accuracy_test,
            'Precision (Testing)': precision_test,
            'Recall (Testing)': recall_test,
            'F1 Score (Testing)': f1_test
        }

    return model_results
```

Reading in the Data

```
In [ ]: df = pd.read_excel('/Users/jack/Desktop/USITCC_2024_FINAL_DATA.xlsx')
```

```
In [ ]: df.head()

        df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 99903 entries, 0 to 99902
Data columns (total 19 columns):
 #   Column                     Non-Null Count  Dtype
---  ------                     --------------  -----
 0   APPLICATION_ID             99903 non-null  int64
 1   APPLICATION_DT             99903 non-null  datetime64[ns]
 2   PRODUCT                    99903 non-null  object
 3   COSIGNER_IND               99903 non-null  object
 4   ACQUISITION_CHANNEL        99903 non-null  object
 5   PRODUCT_RATE_TYPE_DESC     99903 non-null  object
 6   PRODUCT_TERM_NUM           99903 non-null  int64
 7   INTEREST_RT                99903 non-null  float64
 8   BORROWER_INCOME            99903 non-null  int64
 9   BORROWER_LIABILITY         99903 non-null  int64
 10  CREDIT_SCORE               99903 non-null  int64
 11  REQUESTED_LOAN_AMT         99903 non-null  int64
 12  REASON_FOR_DENIAL          55142 non-null  object
 13  INITIAL_APPROVAL_CNT       99903 non-null  int64
 14  FULL_CREDIT_APPROVAL_CNT   99903 non-null  int64
 15  WITHDRAWN_CANCEL_CLOSED_CNT 99903 non-null  int64
 16  APPROVAL_ACCEPTANCE_CNT    99903 non-null  int64
 17  DISBURSEMENT_STATUS        43008 non-null  object
 18  BOOKED_IND                 99903 non-null  object
dtypes: datetime64[ns](1), float64(1), int64(10), object(7)
memory usage: 14.5+ MB
```

Average Credit Score of all student loans

```
In [ ]: df.CREDIT_SCORE.mean().round(2)
```

```
Out[ ]: 681.97
```

Average credit score of all students with initial approval

```
In [ ]: df.query('INITIAL_APPROVAL_CNT > 0').CREDIT_SCORE.mean().round(2)
```

```
Out[ ]: 748.96
```

```
In [ ]: desc =df[['BORROWER_INCOME']].describe()

        display(desc)

        iqr = desc.loc['75%'] - desc.loc['25%']

        print(iqr)
```

	BORROWER_INCOME
count	99903.00000
mean	82702.52545
std	58393.92286
min	10000.00000
25%	36900.00000
50%	72400.00000
75%	110000.00000
max	300000.00000

```
BORROWER_INCOME    73100.0
dtype: float64
```

Correlation of interest rate and credit score shows strong negative

In [ ]:

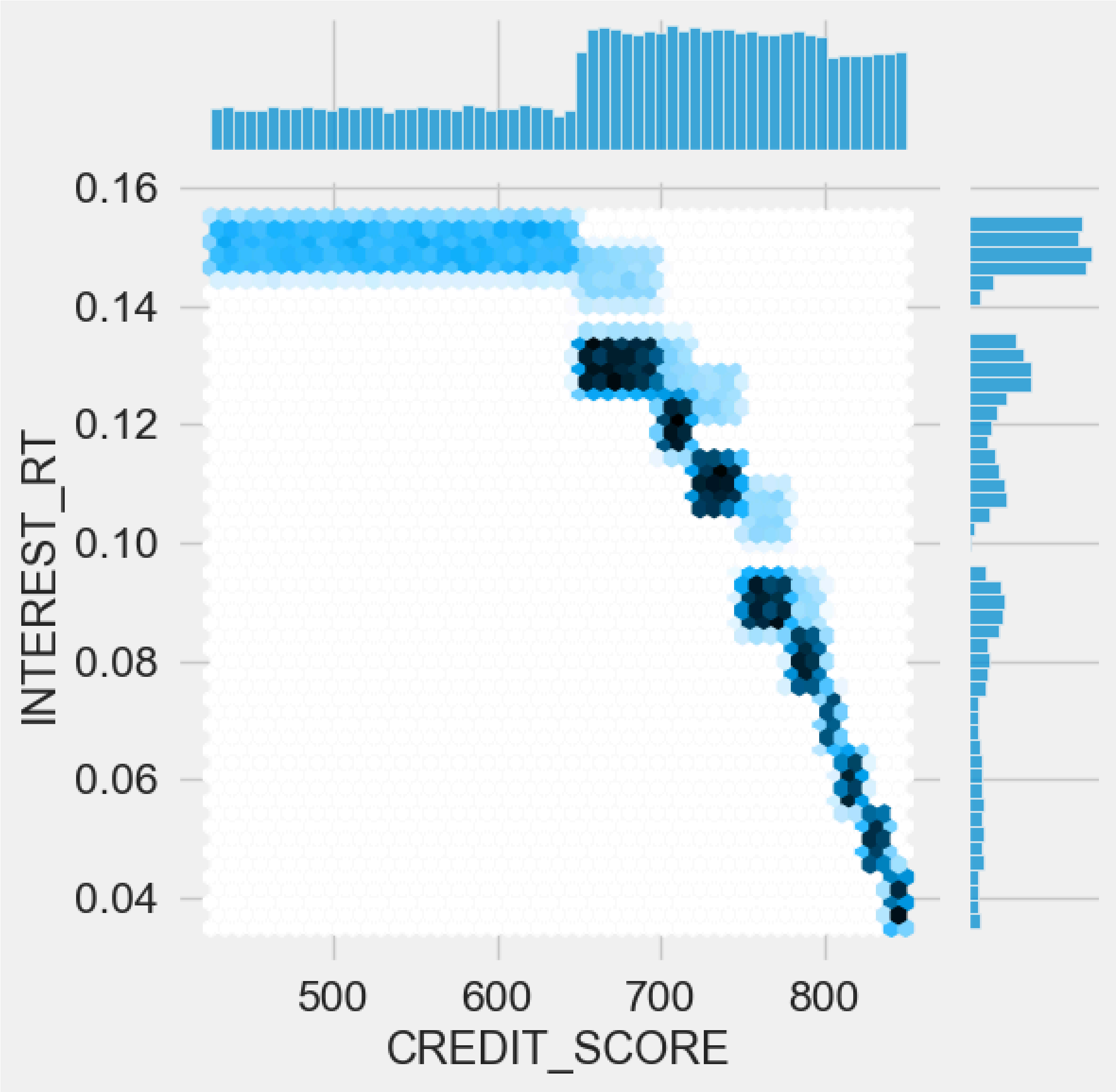
```python
df[['INTEREST_RT','CREDIT_SCORE']].corr(method='pearson').iloc[1,0].round(3)
```

Out[ ]:

```
-0.87
```

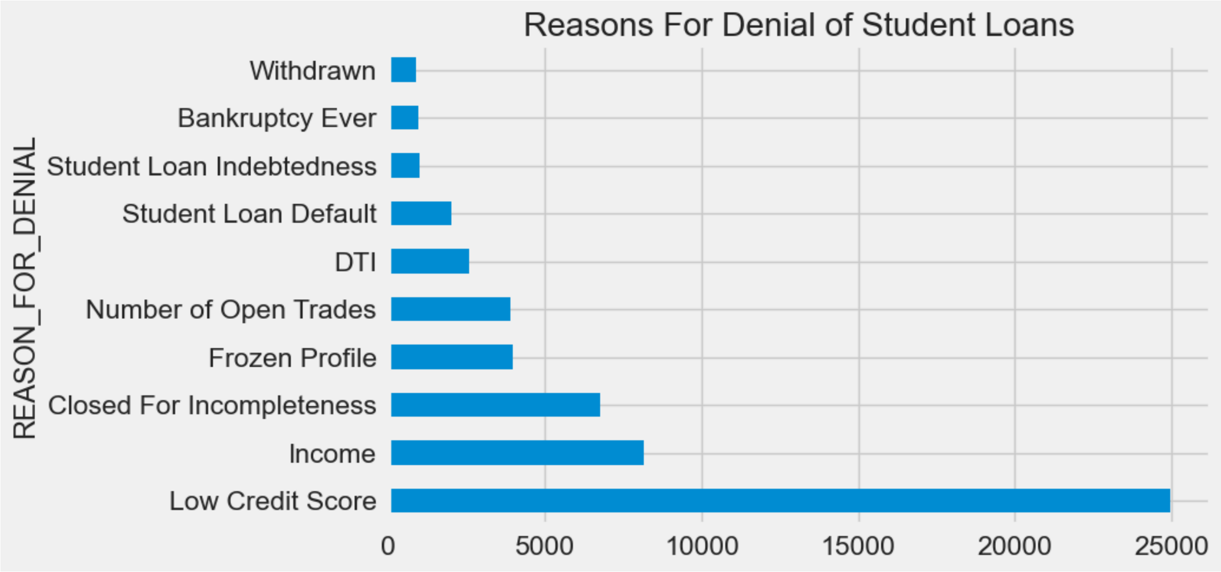
In [ ]:

```python
sns.jointplot(df,x='CREDIT_SCORE',y='INTEREST_RT',kind='hex')

plt.show()
```

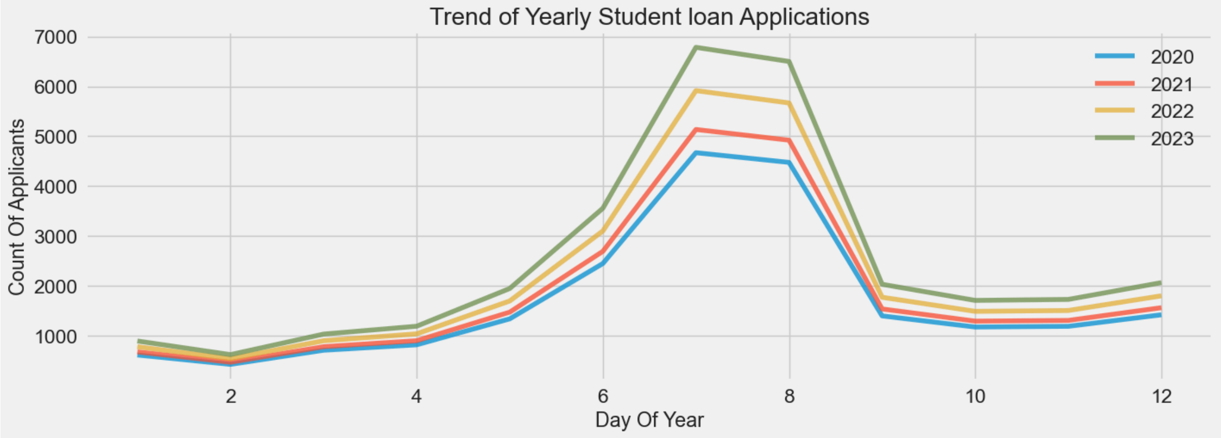


```
In [ ]: df.REASON_FOR_DENIAL.value_counts().plot(kind='barh',
                                                   title='Reasons For Denial of Studer
                                                   figsize=(8,5))
```

```
Out[ ]: <Axes: title={'center': 'Reasons For Denial of Student Loans'}, ylabel='REA
        SON_FOR_DENIAL'>
```



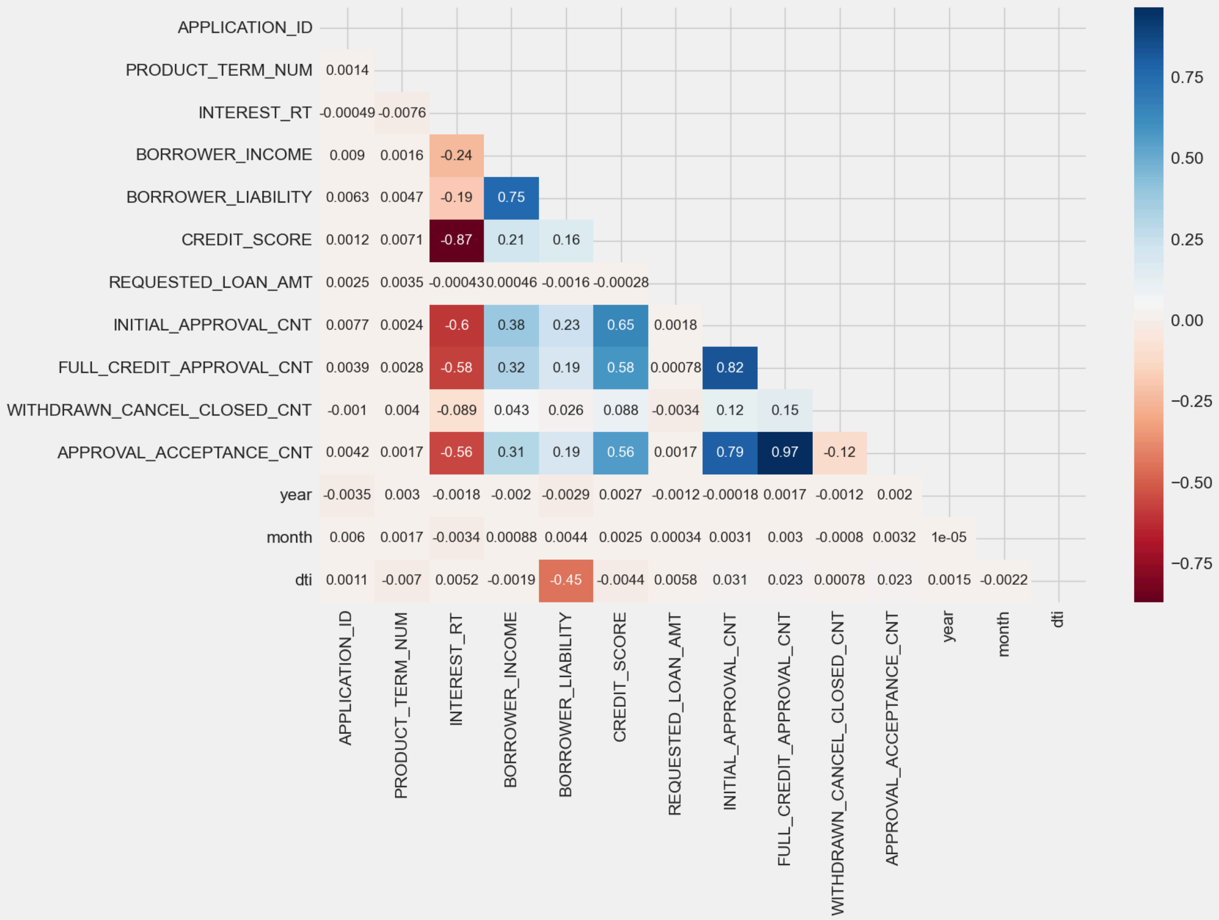
```
In [ ]: df = df.assign(year = df['APPLICATION_DT'].dt.year,
                       month = df['APPLICATION_DT'].dt.month)


        for year_ in df.year.unique():

            yearly = df[df.year == year_]

            yearly.groupby('month')['APPLICATION_ID'].count().plot(alpha=.75,figsize


        plt.title('Trend of Yearly Student loan Applications')

        plt.legend()

        plt.xlabel('Day Of Year')

        plt.ylabel('Count Of Applicants')

        plt.show()
```



```
In [ ]: #creating DTI or Debt to Income Ratio

        df = df.assign(dti = df['BORROWER_INCOME'].div(12) / df['BORROWER_LIABILITY'
```

Check for multi colinearity

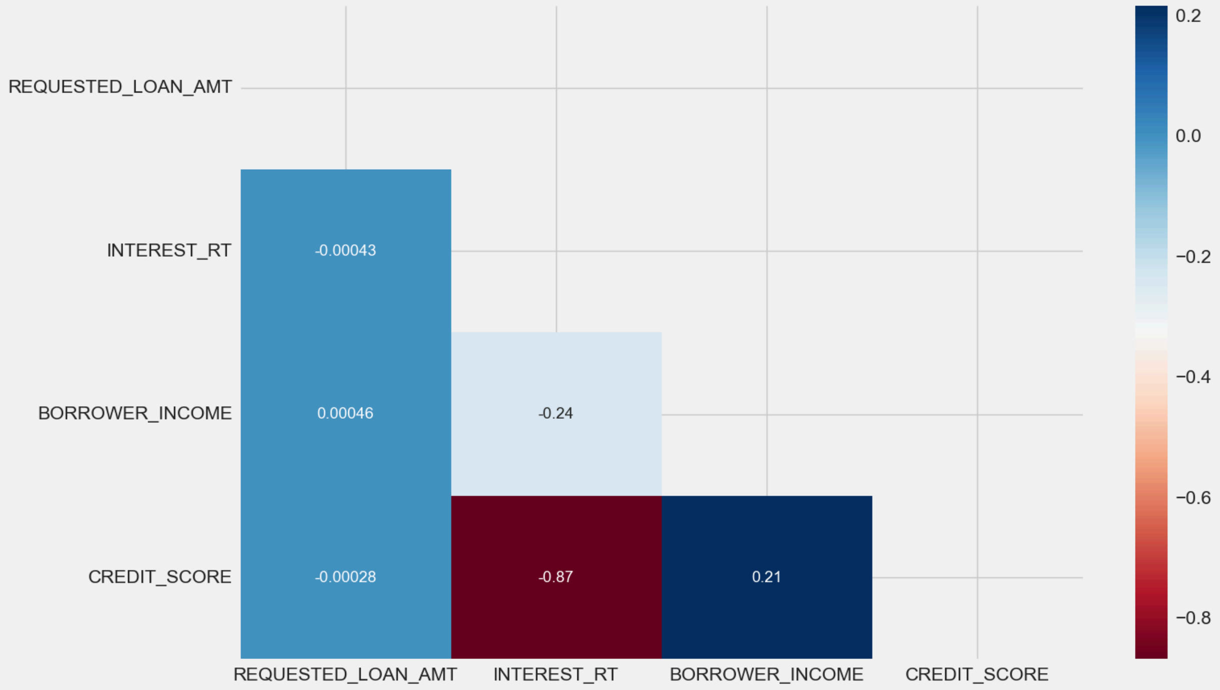
```
In [ ]: corr = df.select_dtypes([int,float]).corr()

        mask = np.triu(np.ones_like(corr))


        plt.figure(figsize=(15,10))

        sns.heatmap(corr,mask=mask, annot=True, cmap='RdBu')

        plt.show()
```



```
In [ ]: #Perform PCA
        from sklearn.preprocessing import StandardScaler
        from sklearn.decomposition import PCA

        scaler = StandardScaler()

        feats = df[['REQUESTED_LOAN_AMT','INTEREST_RT','BORROWER_INCOME','CREDIT_SCO

        feats_scaled = pd.DataFrame(scaler.fit_transform(feats),columns=feats.column
```

```
In [ ]: #creating covariance matrix (a heatmap)
```

```
corr = feats.corr()

mask = np.triu(np.ones_like(corr))

plt.figure(figsize=(15,10))

sns.heatmap(corr,mask=mask, annot=True, cmap='RdBu')

plt.show()
```



interest rate is highly correlated with credit score, no surprise there

fitting pca with 3 components

In [ ]:
```
pca = PCA(n_components=3)

comps = pd.DataFrame(pca.fit_transform(feats),columns=['comp_1','comp_2','co
```

In [ ]:
```
#dropping feats we are using pca on

df = df.drop(feats.columns,axis=1)
```

In [ ]:
```
df.info() #we see that disbursement status and reason for denial are 50% nul

# A good IT team would fill reason for denial with a value if they didn't de
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 99903 entries, 0 to 99902
Data columns (total 18 columns):
 #   Column                     Non-Null Count  Dtype
---  ------                     --------------  -----
 0   APPLICATION_ID             99903 non-null  int64
 1   APPLICATION_DT             99903 non-null  datetime64[ns]
 2   PRODUCT                    99903 non-null  object
 3   COSIGNER_IND               99903 non-null  object
 4   ACQUISITION_CHANNEL        99903 non-null  object
 5   PRODUCT_RATE_TYPE_DESC     99903 non-null  object
 6   PRODUCT_TERM_NUM           99903 non-null  int64
 7   BORROWER_LIABILITY         99903 non-null  int64
 8   REASON_FOR_DENIAL          55142 non-null  object
 9   INITIAL_APPROVAL_CNT       99903 non-null  int64
 10  FULL_CREDIT_APPROVAL_CNT   99903 non-null  int64
 11  WITHDRAWN_CANCEL_CLOSED_CNT  99903 non-null  int64
 12  APPROVAL_ACCEPTANCE_CNT    99903 non-null  int64
 13  DISBURSEMENT_STATUS        43008 non-null  object
 14  BOOKED_IND                 99903 non-null  object
 15  year                       99903 non-null  int32
 16  month                      99903 non-null  int32
 17  dti                        99903 non-null  float64
dtypes: datetime64[ns](1), float64(1), int32(2), int64(7), object(7)
memory usage: 13.0+ MB
```

In [ ]:
```
df = df.drop(['REASON_FOR_DENIAL','DISBURSEMENT_STATUS'],axis=1)
```

In [ ]:
```
df = df.set_index('APPLICATION_ID')
```

creating dummies for modeling, the categories don't appear to be ordinal, meaning their is no inherent order

In [ ]:
```
df = pd.get_dummies(df,columns=['COSIGNER_IND','PRODUCT','ACQUISITION_CHANNE
               dtype=int,
               drop_first=True)\
    .drop(['BORROWER_LIABILITY','APPLICATION_DT',
           #drop these as they leak data
           'APPROVAL_ACCEPTANCE_CNT',
                                                               'FULL_CREDIT_

          ,axis=1) # drop borrower liability due to the same information exi
```

In [ ]:
```
numeric_cols = ['year','month','dti','PRODUCT_TERM_NUM']

scaler = StandardScaler()

df[numeric_cols] = scaler.fit_transform(df[numeric_cols])
```

```
In [ ]: #convert target variable to numerical
        df['BOOKED_IND'] = np.where(df['BOOKED_IND'] == 'Y',1,0)
```

Prepare to model, train-test-split and baseline evaluation

```
In [ ]: from sklearn.model_selection import train_test_split
        from sklearn.linear_model import LogisticRegression
        from sklearn.metrics import classification_report
        target = 'BOOKED_IND'
        X = df.drop(target,axis=1)
        y = df[[target]]
        X_train,X_test, y_train,y_test = train_test_split(X,y,test_size=.7)
```

```
In [ ]: model = LogisticRegression(max_iter=1000)
        model.fit(X_train,y_train)
        pred = model.predict(X_test)
        classification_report(y_test,pred)
```

```
/Users/jack/tensorflow-test/env/lib/python3.8/site-packages/sklearn/utils/va
lidation.py:1184: DataConversionWarning: A column-vector y was passed when a
1d array was expected. Please change the shape of y to (n_samples, ), for ex
ample using ravel().
  y = column_or_1d(y, warn=True)
```

```
Out[ ]: '               precision    recall  f1-score   support\n\n           0
        1.00      0.83      0.91     39967\n           1       0.82      1.00
        0.90     29966\n\n    accuracy                           0.90     69933\n
        macro avg       0.91      0.92      0.90     69933\nweighted avg       0.92
        0.90      0.90     69933\n'
```

```
In [ ]: new_df = pd.DataFrame(np.concatenate([df,comps],axis=1))
        new_df.columns = df.columns.tolist() + comps.columns.tolist()
        new_df.head() #added in PCA components
```

Out[ ]:

	PRODUCT_TERM_NUM	INITIAL_APPROVAL_CNT	WITHDRAWN_CANCEL_CLOSED_CN
0	0.000824	1.0	0.
1	0.000824	1.0	0.
2	0.000824	1.0	0.
3	1.829617	0.0	0.
4	0.000824	0.0	0.

In [ ]:

```
#lets train lots of models at once and eval, lets also join in our pca compo


train_test_score_classification(new_df,target=target)
```

Out[ ]:

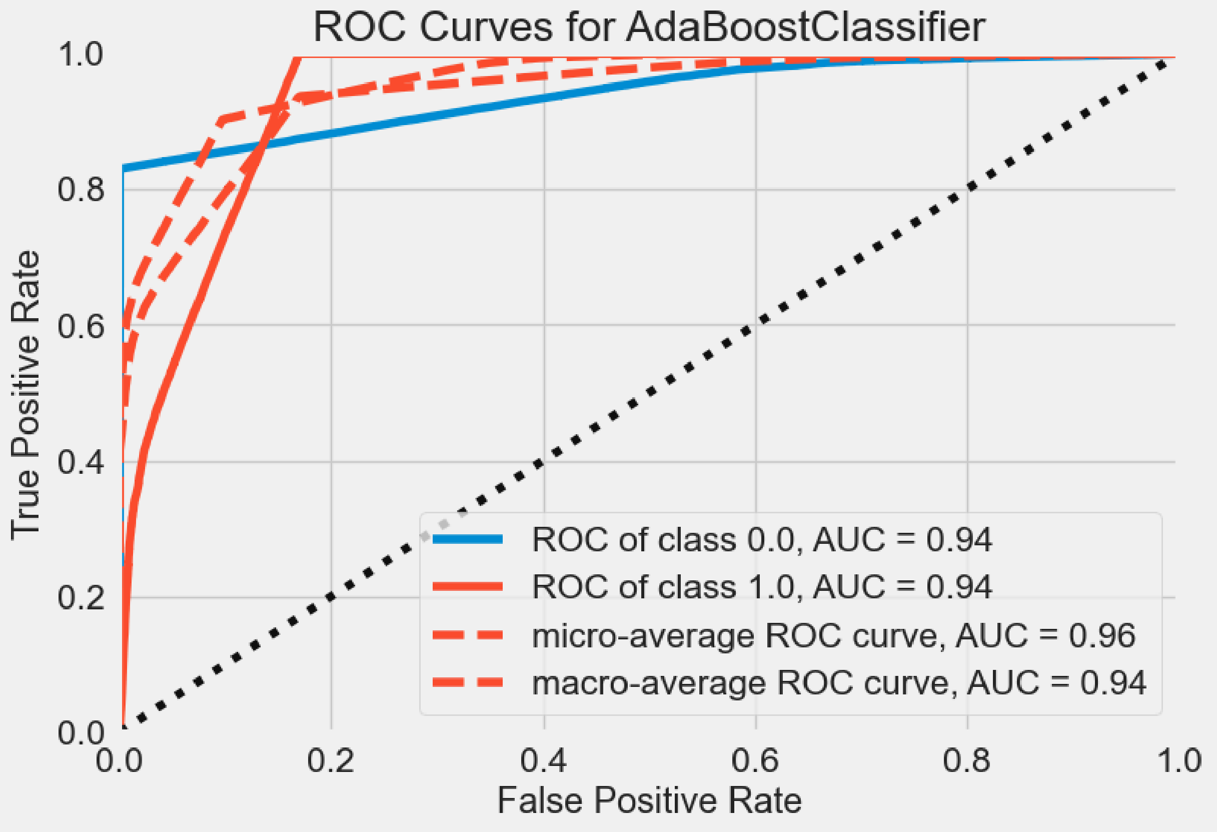
```
{'Logistic Regression': {'Accuracy (Training)': 0.7530842571507219,
  'Precision (Training)': 0.6622526804465568,
  'Recall (Training)': 0.8705140499229942,
  'F1 Score (Training)': 0.752234833266372,
  'Accuracy (Testing)': 0.7533656974125419,
  'Precision (Testing)': 0.6625875676155005,
  'Recall (Testing)': 0.8693426410703897,
  'F1 Score (Testing)': 0.752012882447665},
 'Random Forest': {'Accuracy (Training)': 1.0,
  'Precision (Training)': 1.0,
  'Recall (Training)': 1.0,
  'F1 Score (Training)': 1.0,
  'Accuracy (Testing)': 0.9040588559131174,
  'Precision (Testing)': 0.8201956271576525,
  'Recall (Testing)': 0.9951134380453752,
  'F1 Score (Testing)': 0.8992272512222047},
 'AdaBoost': {'Accuracy (Training)': 0.903030454693326,
  'Precision (Training)': 0.8166702296966674,
  'Recall (Training)': 0.9990701188504344,
  'F1 Score (Training)': 0.8987086992890004,
  'Accuracy (Testing)': 0.904859616635804,
  'Precision (Testing)': 0.8195531793011266,
  'Recall (Testing)': 0.9987201861547411,
  'F1 Score (Testing)': 0.9003094026954743},
 'XGBoost': {'Accuracy (Training)': 0.9079852856535122,
  'Precision (Training)': 0.8241142227439331,
  'Recall (Training)': 0.9996512945689129,
  'F1 Score (Training)': 0.9034350543620988,
  'Accuracy (Testing)': 0.904359141184125,
  'Precision (Testing)': 0.8206062931696086,
  'Recall (Testing)': 0.9952297847585806,
  'F1 Score (Testing)': 0.8995215311004785}}
```

In [ ]:

```
from yellowbrick.classifier import roc_auc

#train ada boost on principal components + old features

X = new_df.drop(target,axis=1)
```

```
y = new_df[[target]]

X_train,X_test, y_train,y_test = train_test_split(X,y,test_size=.7)

classifier = roc_auc(AdaBoostClassifier(),X_train,y_train,X_test,y_test) # A
# with an F1-Score of 89% on the training set and 90% on the test set

model = AdaBoostClassifier()

model.fit(X_train,y_train)

pred = model.predict(X_test)

report = classification_report(y_test,pred,output_dict=True)
```

```
/Users/jack/tensorflow-test/env/lib/python3.8/site-packages/sklearn/utils/va
lidation.py:1184: DataConversionWarning: A column-vector y was passed when a
1d array was expected. Please change the shape of y to (n_samples, ), for ex
ample using ravel().
  y = column_or_1d(y, warn=True)
```



```
/Users/jack/tensorflow-test/env/lib/python3.8/site-packages/sklearn/utils/va
lidation.py:1184: DataConversionWarning: A column-vector y was passed when a
1d array was expected. Please change the shape of y to (n_samples, ), for ex
ample using ravel().
  y = column_or_1d(y, warn=True)
```

In [ ]:

```
summary = pd.DataFrame(report).round(2)# here is the classification report

summary = summary.rename(columns={'0.0':'Negative Class','1.0':'Positive Cla
summary
```

Out[ ]:

	Negative Class	Positive Class	accuracy	macro avg	weighted avg
precision	1.00	0.82	0.9	0.91	0.92
recall	0.83	1.00	0.9	0.91	0.90
f1-score	0.91	0.90	0.9	0.90	0.90
support	39905.00	30028.00	0.9	69933.00	69933.00

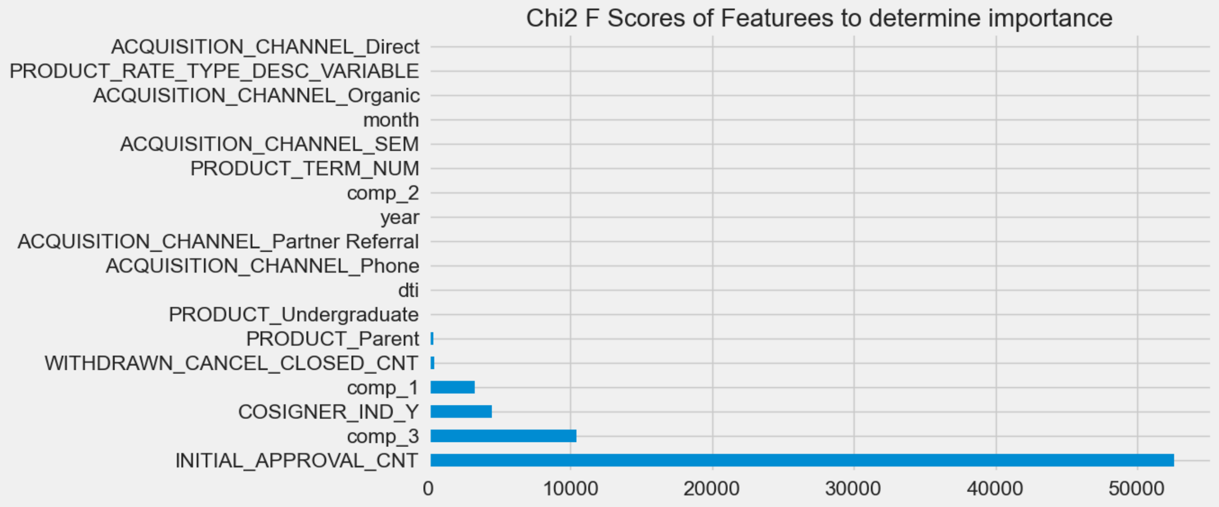
In [ ]:

```
from sklearn.feature_selection import SelectKBest, f_classif

kbest = SelectKBest(score_func=f_classif,k='all')

kbest.fit(X_train,y_train)

feat_importances = pd.Series(kbest.scores_,index=X_train.columns).sort_value


feat_importances.plot(kind='barh',figsize=(10,6),title='Chi2 F Scores of Fea
```

```
/Users/jack/tensorflow-test/env/lib/python3.8/site-packages/sklearn/utils/va
lidation.py:1184: DataConversionWarning: A column-vector y was passed when a
1d array was expected. Please change the shape of y to (n_samples, ), for ex
ample using ravel().
  y = column_or_1d(y, warn=True)
```

Out[ ]:

```
<Axes: title={'center': 'Chi2 F Scores of Featurees to determine importanc
e'}>
```



In [ ]:

```
feat_importances.round(2)
```

Out[ ]:

```
INITIAL_APPROVAL_CNT                    52591.93
comp_3                                  10424.38
COSIGNER_IND_Y                           4494.21
comp_1                                   3240.84
WITHDRAWN_CANCEL_CLOSED_CNT               400.85
PRODUCT_Parent                            352.04
PRODUCT_Undergraduate                     149.64
dti                                        17.76
ACQUISITION_CHANNEL_Phone                   4.59
ACQUISITION_CHANNEL_Partner Referral        1.49
year                                        0.79
comp_2                                      0.65
PRODUCT_TERM_NUM                            0.53
ACQUISITION_CHANNEL_SEM                     0.47
month                                       0.10
ACQUISITION_CHANNEL_Organic                 0.09
PRODUCT_RATE_TYPE_DESC_VARIABLE             0.02
ACQUISITION_CHANNEL_Direct                  0.01
dtype: float64
```

In [ ]:

```
df[['INITIAL_APPROVAL_CNT','BOOKED_IND']].query('BOOKED_IND == 0').describe(


# sanity check to ensure we can include the soft credit check in the model.

# There is still a 20% mean proportion of the soft credit check passing with
```

Out[ ]:

	INITIAL_APPROVAL_CNT	BOOKED_IND
count	56895.000000	56895.0
mean	0.200721	0.0
std	0.400543	0.0
min	0.000000	0.0
25%	0.000000	0.0
50%	0.000000	0.0
75%	0.000000	0.0
max	1.000000	0.0