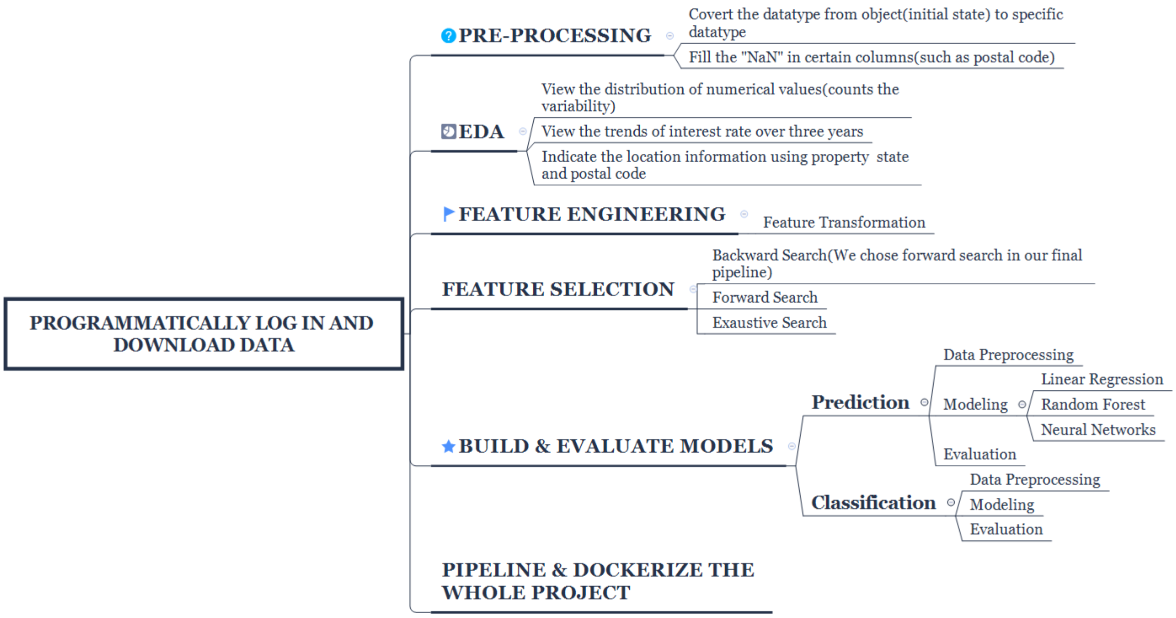
Midterm Case Studies Report

Group 7

Our workflow for the midterm case studies is as follows:



Problem and solution description:

1. Programmatically log in and download the data:

For data wrangling, there are several ways to accomplish the task, different modules such as requests or requests-html are both good approach, here we used both of them, but the basic thoughts are the same.

1) Build a session to store all user sessions and our script will be kept login until we terminate it.

2) Pack up the username and password as post data, send the POST request to ~/auth.php instead of ~/login.php, and in the meanwhile, disable the redirect parameter of auth.php.

3) Pack up another post data and send another post request to the download page, because the website ask the user to review and accept their terms and conditions.

4) After we get two status_code with 200, that means the login is successful

5) We simply used regex to grab all the urls and do a match up, after we get the link according to year and quarter, we can retrieve the file and download it into local system (but actually we download the file to ubunt@aws).

2. Exploratory Data Analysis:

At the very beginning, we use profiling in Pandas to give us a basic view on our dataset.

```
In [3]: pandas_profiling.ProfileReport(df)
```

Number of variables	26
Number of observations	351634
Total Missing (%)	4.4%
Total size in memory	69.8 MiB
Average record size in memory	208.0 B

Numeric	12
Categorical	9
Boolean	0
Date	0
Text (Unique)	1
Rejected	4
Unsupported	0

Warnings

- CREDIT_SCORE is highly skewed (γ1 = 37.028) Skewed
- FIRST_PAYMENT_DATE is highly skewed (γ1 = 38.67) Skewed
- MORTGAGE_INSURANCE_PERCENTAGE is highly skewed (γ1 = 43.525) Skewed
- MORTGAGE_INSURANCE_PERCENTAGE has 302104 / 85.9% zeros Zeros
- MSA has 54684 / 15.6% missing values Missing
- NUMBER_OF_BORROWERS is highly skewed (γ1 = 50.036) Skewed
- NUMBER_OF_UNITS is highly skewed (γ1 = 175.66) Skewed
- ORIGINAL_LOAN_TERM is highly correlated with MATURITY_DATE (ρ = 0.99987) Rejected
- ORIGINAL_LOAN_TO_VALUE is highly correlated with ORIGINAL_COMBINED_LOAN_TO_VALUE (ρ = 0.96867) Rejected

And we finda column called PRODUCT_TYPE has a constant value "FRM", we should remove this column in the future cleaning.

~~PRODUCT_TYPE~~
Constant

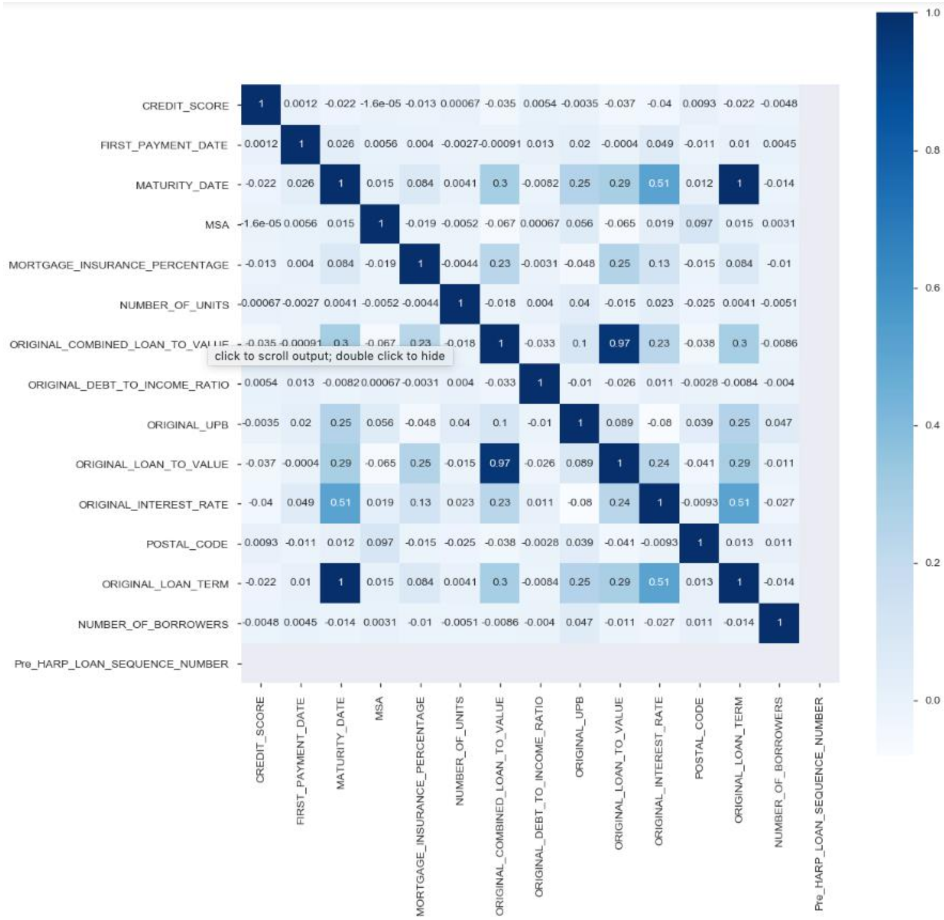
This variable is constant and should be ignored for analysis

Constant value	FRM

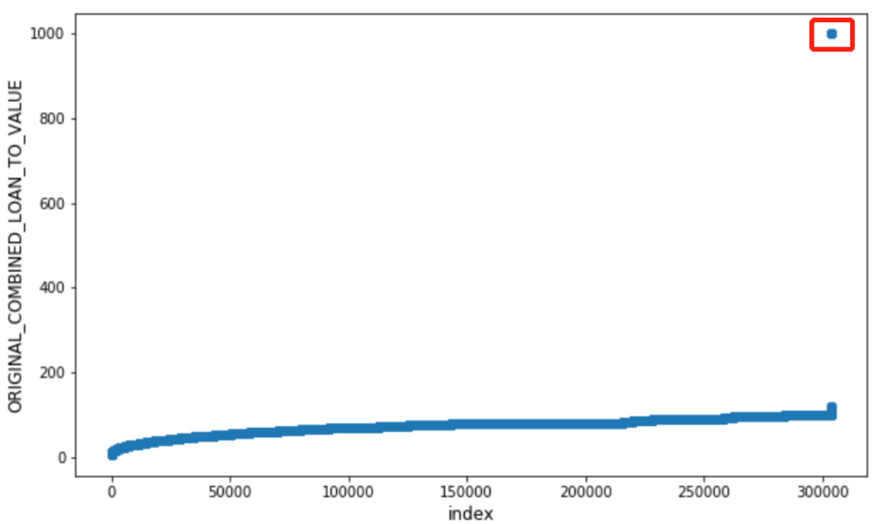
Then, we convert the features in the dataframe to specific datatypes in order to do the EDA part (they are all object in the initial state).

```
plt.figure(figsize = (10,6))
plt.scatter(range(df.shape[0]),np.sort(df.ORIGINAL_COMBINED_LOAN_TO_VALUE.values))
plt.xlabel('index',fontsize=12)
plt.ylabel('ORIGINAL_COMBINED_LOAN_TO_VALUE',fontsize=12)
plt.show()
```

Then we take a look on the correlation between every feature, we find there are some features having a strong postive corrrelation, we can pick one of them as main feature in the future.



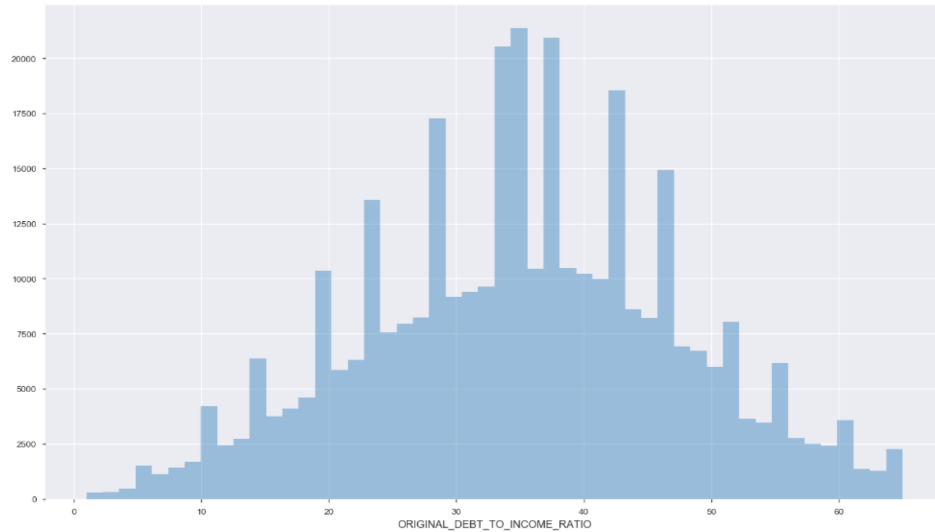
We use the scatter plot to check whether there is outlier in the values of different features. For example, the following figure shows the values of the ORIGINAL_COMBINED_LOAN_TO_VALUE, it indicates that there is one outlier in the values. We can reject these outliers in the preprocessing process to avoid the possible negative impact they may have on our model.



Then, we draw some distribution plots like below to see how values of every feature distribution.

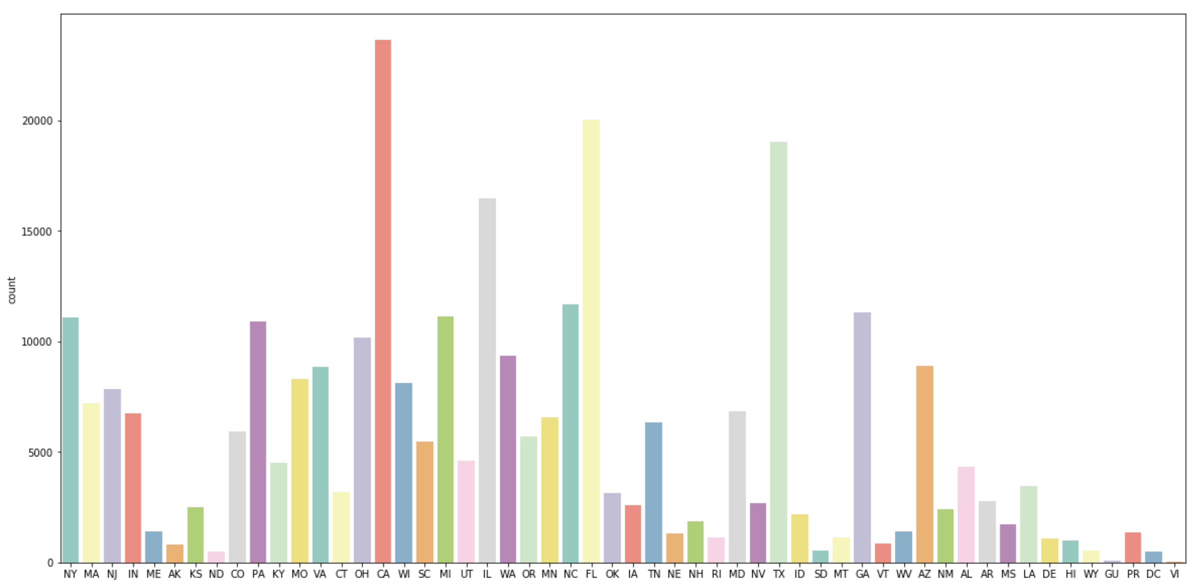
```
In [21]: plt.figure(figsize=(18,10))
         sns.distplot(df.ORIGINAL_DEBT_TO_INCOME_RATIO.values, bins=50, kde=False)
         plt.xlabel('ORIGINAL_DEBT_TO_INCOME_RATIO', fontsize=12)
         plt.show()

/Users/G/anaconda3/lib/python3.6/site-packages/matplotlib/axes/_axes.py:6462: UserWarning: The 'normed' kwarg is depr
ecated, and has been replaced by the 'density' kwarg.
  warnings.warn("The 'normed' kwarg is deprecated, and has been "
```



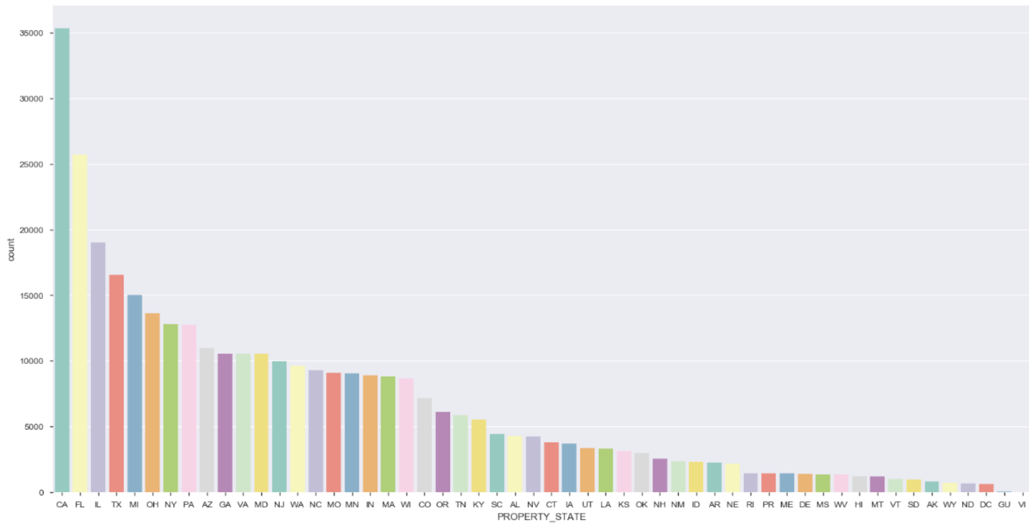
Next, we used countplot and distplot to demonstrate the counts and value distribution of different variables. The distribution of variables can help us get an idea on how to fill in the null values in certain columns.

```
plt.figure(figsize=(20,10))
sns.countplot(x="PROPERTY_STATE", data = df, palette="Set3")
```

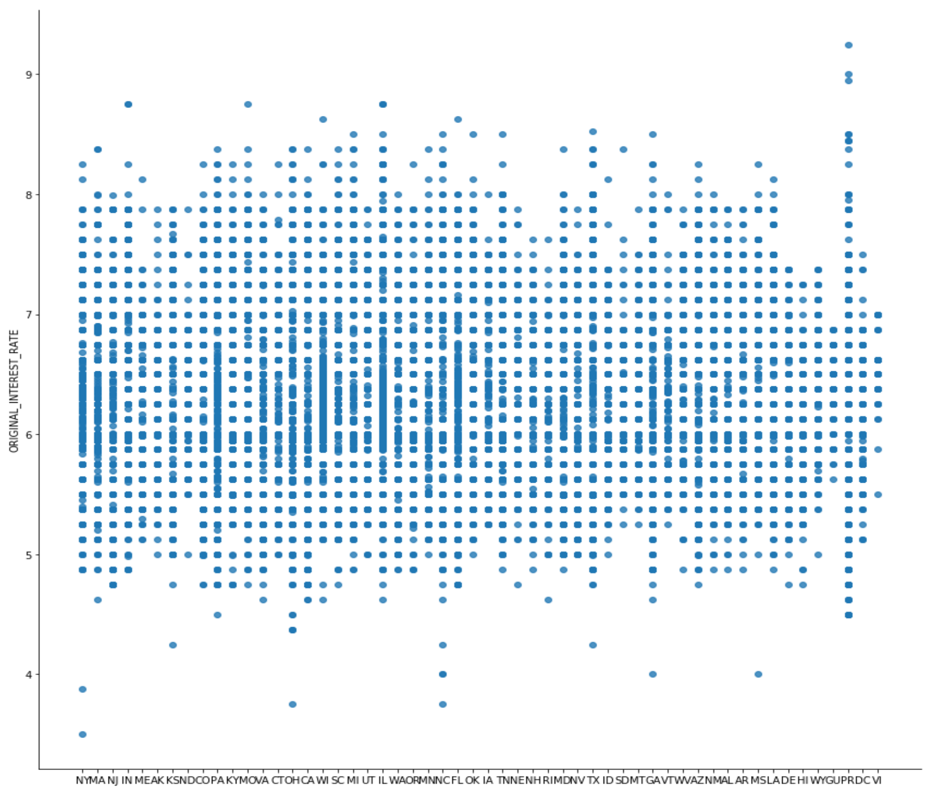


Sort by its amount like below, we can get a clearer view on this feature.

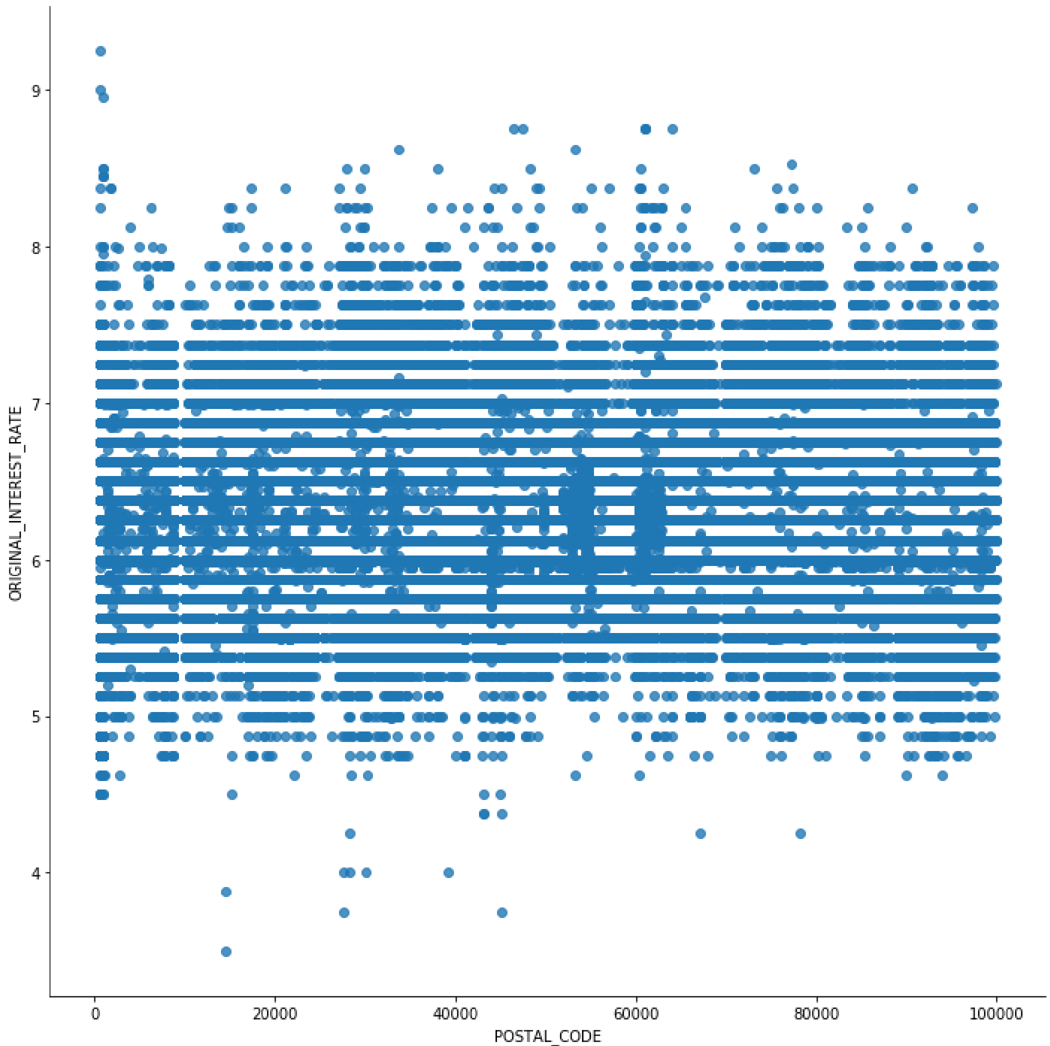
In [30]:
```
plt.figure(figsize=(20,10))
sns.countplot(x="PROPERTY_STATE", data = df, palette="Set3",order = df['PROPERTY_STATE'].value_counts().index)
```
Out[30]: `<matplotlib.axes._subplots.AxesSubplot at 0x1a26a04fd0>`



In our dataset, there are some location related features, such as property state and postal code. We drew a lmplot to see the relation between interest rate and different states. The following plot indicates that the interest rate concentrated on a specific range between different states.



The following plot shows the relation between interest rate and postal code. We can draw similar conclusion with the above plot about the location information. No matter what the distribution looks like, the distribution of the interest rate prensents a certain regularity, that is their values concentrate in a specific range(from 5.8 to 7.0).

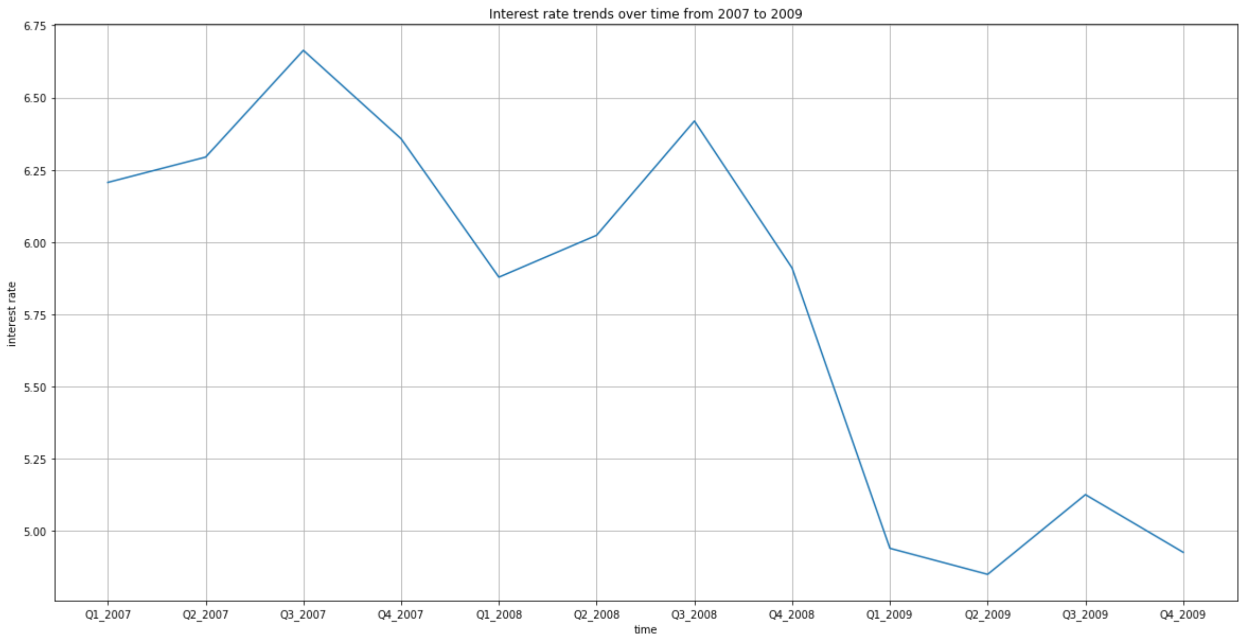


We can also have a look on the distribution on every location though postal code like below:

We can know the area whose postal code is 85200 has the most amount of loan.

After explored some basic information of different features, we wanted to see how the variable that we concerned most, that is, the original interest rate, changed over time. To see the trends, we calculated the mean value of original interest rate of different quarter from 2007 to 2009, 12 values in total. Then, we drew a plot to see the trends of interest rate.

```
# plt.plot(time_list, rate_mean_list)
fig,ax=plt.subplots(figsize=(20,10))
ax.plot(time_list, rate_mean_list)
ax.set(xlabel='time', ylabel='interest rate',
       title='Interest rate trends over time from 2007 to 2009')
ax.grid()
fig.figure
plt.show()
```



The above figure shows the trends of the original interest rate over three years. It demonstrates that the interest rate roughly presents a decreasing trend year by year.

3. Prediction:

As requested, for this part, we use 6 different models to predict the interest rate.
But before we run into the model-trainning part, we should do the feature engineering. In this part, first, we give the year number and quarter number to download the training data automatically and create a function to get testing data automatically as well. Then we add a header for our dataset, after that, drop columns which values has more than 70% NaN, convert date to 2 columns(year and month), replace outlier with NaN, set flag's values as 0 or 1, use OneHotEncoding and LabelEncoding to convert columns which are object data type to numeric, fill all NaN values with their corresponding columns' mean values.
Now we have enough clean data to train our models.

But many models we bulit are on the cloud, so there is no running result record in the jupyter notebook we upload on the github.

1) Linear Regression Model(Run on google colab):

```
X_training = pd.read_csv("drive/My Drive/7390_Assignment_3/X_training.csv").values
y_training = pd.read_csv("drive/My Drive/7390_Assignment_3/y_training.csv",header = None).values
X_testing = pd.read_csv("drive/My Drive/7390_Assignment_3/X_testing.csv").values
y_testing = pd.read_csv("drive/My Drive/7390_Assignment_3/y_testing.csv",header = None).values
```

```
[7]  regr = linear_model.LinearRegression()
```

```
[8]  regr.fit(X_training, y_training)
```

```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=1, normalize=False)
```

```
[9]  y_pred = regr.predict(X_testing)
```

```
[10] print('Coefficients: \n', regr.coef_)
     # The mean squared error
     print("Mean squared error: %.2f" % mean_squared_error(y_testing, y_pred))
     # Explained variance score: 1 is perfect prediction
     print('Variance score: %.2f' % r2_score(y_testing, y_pred))
```

```
Coefficients:
 [[-7.77723193e-04  8.84165122e-04 -1.48268747e-09  4.44346804e-03
   1.80870821e-02  1.41113806e-03 -5.93489984e-04 -7.87285611e-07
   1.28268687e-04  2.02511624e-01 -4.05982107e-06 -3.36148658e-03
  -2.50438222e-02  7.43887758e-02 -1.32776948e-02  7.09157588e-02
   2.50370199e-02  1.62919233e-01 -1.42260527e-01 -1.10137350e-02
   2.11321595e-01  2.14040159e-01  1.11915221e-01 -1.76563743e-01
  -6.93635892e-02 -9.01971377e-02 -2.07574241e-02 -1.73492437e-01
   4.86037721e-01  9.99591839e-02 -5.42760048e-02  9.53321942e-02
   9.97714066e-02  1.70423725e-02  8.74290190e-02 -2.84213107e-02
   5.06807059e-02 -1.80361396e-01 -8.42306356e-02 -1.42596237e-01
   5.64402033e-02 -1.31235410e-02  1.02666997e-01 -7.64291324e-02
  -2.09656597e-02 -1.60130229e-01 -1.24652694e-01  4.81031377e-02
  -1.34456825e-01 -1.09327597e-01  1.86008475e-01  2.07017164e-01
  -9.29014162e-02 -8.57017925e-03  8.22795206e-02  1.49249209e-01
  -1.10301435e-01 -4.62252381e-01 -1.80824483e-01 -1.30070960e-01
  -8.45580539e-02 -8.61282957e-02  1.03255791e-01  1.10412299e-01
  -7.84120618e-02  2.68172953e-01 -1.22867697e-01  1.78576878e-01
   3.65664693e-02 -5.82473093e-02  6.25562211e-02 -4.74224565e-02
  -2.89397844e-02  2.09445160e-01 -2.00218143e-02 -5.38490725e-02
  -5.92120321e-02  2.28391810e-01 -1.30107363e-01 -9.82844476e-02
   1.96045351e-02  3.34322339e-01 -1.37155363e-01 -2.16771511e-01
   2.58256047e-02  1.85930847e-02 -4.44186894e-02]]
Mean squared error: 0.10
Variance score: 0.23
```

As we can see here, grade of linear model is just about 0.23, this is not good enough for us.

2) Random Forest Regression Model:

We used the first quarter data of 2007 to train our random forest regressor, and test the model on the second quarter data of 2007.

```
from sklearn.ensemble import RandomForestRegressor
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
```

```
# split X_train and y_train, X_test and y_test
X_train_df = pd.read_csv('./X_train.csv')
X_test_df = pd.read_csv('./X_test.csv')
y_train_df = pd.read_csv('./y_train.csv', header = None)
y_test_df = pd.read_csv('./y_test.csv',header = None)
```

```
X_train = X_train_df.values
X_test = X_test_df.values
y_train = y_train_df.values
y_test = y_test_df.values
```

```
rf_model = RandomForestRegressor(n_estimators = 100, max_depth = None)
```

```
rf_model.fit(X_train,y_train)
```

```
y_pred = rf_model.predict(X_test)
```

```
from sklearn.model_selection import GridSearchCV
rfr_best = RandomForestRegressor()
params = {'n_estimators': range(50,100,200)}
gs = GridSearchCV(rfr_best, params, cv=10, scoring = 'r2')
gs.fit(X_train, y_train)

print(gs.best_score_)
print(gs.best_params_)
```

```
  estimator.fit(X_train, y_train, **fit_params)
/usr/local/lib/python3.6/dist-packages/sklearn/model_selection/_validation.py:458: DataCon
  estimator.fit(X_train, y_train, **fit_params)
/usr/local/lib/python3.6/dist-packages/sklearn/model_selection/_validation.py:458: DataCon
  estimator.fit(X_train, y_train, **fit_params)
/usr/local/lib/python3.6/dist-packages/sklearn/model_selection/_validation.py:458: DataCon
  estimator.fit(X_train, y_train, **fit_params)
/usr/local/lib/python3.6/dist-packages/sklearn/model_selection/_validation.py:458: DataCon
  estimator.fit(X_train, y_train, **fit_params)
/usr/local/lib/python3.6/dist-packages/sklearn/model_selection/_validation.py:458: DataCon
  estimator.fit(X_train, y_train, **fit_params)
/usr/local/lib/python3.6/dist-packages/sklearn/model_selection/_validation.py:458: DataCon
  estimator.fit(X_train, y_train, **fit_params)
/usr/local/lib/python3.6/dist-packages/sklearn/model_selection/_search.py:740: DataConvers
  self.best_estimator_.fit(X, y, **fit_params)
0.27758371027658896
{'n_estimators': 50}
```

We imported the GridSearchCV module to help us find a best hyperparameter between different choices. In the above example, we input three arguments in the GridSearchCV method. It helped us find that the model gets a highest performance when n_estimators equals 50, and the best score (measured by r2) is 0.277. Honestly, the performance of the random forest model is not good for prediction.

```
[ ]  from sklearn.metrics import mean_squared_error
     from sklearn.metrics import mean_absolute_error
     from sklearn.metrics import r2_score

     mse = mean_squared_error(y_test, y_pred)
     mae = mean_absolute_error(y_test, y_pred)
     r2 = r2_score(y_test, y_pred)

     print(mse, mae, r2)
```

```
0.09486147885798726 0.23660476240366532 0.22822312785562326
```

And then we calculated the MSE, MAE and R2 to see the performance of our random forest regression model. According to its performance, we decided to rule out this option for our final pipeline.

3) Neural Network Model (Built on AWS):

Function to calculate MAE RMS R2 MAPE:

```
def cal_errors(y_testing,y_pred):
    mae = mean_absolute_error(y_testing, y_pred)
    rms = mean_squared_error(y_testing, y_pred)
    r2 = r2_score(y_testing, y_pred)
    mape = mean_absolute_percentage_error(y_testing, y_pred)
    print('MAE = {}, RMS = {}, R2 = {}, MAPE = {}'.format(mae,rms,r2,mape))
```

Scaler dataset in the range of 0-1:

```
def scaler(dataset):
    min_max_scaler = preprocessing.MinMaxScaler(feature_range=( 0, 1))
```

Model:

Here we use degree = 1 to train a model, then we set different polynomial degree for train a new model.

```python
def nn_modeling(X_training, y_training, X_testing, y_testing):
    for hidden_size in hidden:
        reg = MLPRegressor(solver='adam', alpha=1e-5, hidden_layer_sizes=hidden_size, random_state =1)
        reg.fit(X_training, y_training)
        y_pred = reg.predict(X_testing)
        cal_errors(y_testing,y_pred)
```

```python
def make_polynomial_regressor(hidden,dgr = 1):
    return make_pipeline(PolynomialFeatures(degree=dgr),
                         MLPRegressor(solver='adam',
                         alpha=1e-5,
                         hidden_layer_sizes=hidden,
                         random_state =1))
```

```
/usr/local/lib/python3.6/dist-packages/sklearn/neural_network/multilayer_perceptron.py:1306: DataConver
column-vector y was passed when a 1d array was expected. Please change the shape of y to (n_samples, ),
sing ravel().
  y = column_or_1d(y, warn=True)
```

```
MLPRegressor(activation='relu', alpha=1e-05, batch_size='auto', beta_1=0.9,
       beta_2=0.999, early_stopping=False, epsilon=1e-08,
       hidden_layer_sizes=(44,), learning_rate='constant',
       learning_rate_init=0.001, max_iter=200, momentum=0.9,
       nesterovs_momentum=True, power_t=0.5, random_state=1, shuffle=True,
       solver='adam', tol=0.0001, validation_fraction=0.1, verbose=False,
       warm_start=False)
```

```python
def nn_polynomial(X_training, y_training, X_testing, y_testing):
    for hidden_size in hidden:
        for d in range(3):
            reg = make_polynomial_regressor(hidden=hidden_size,dgr=d)
            reg.fit(X_training, y_training)
            y_pred = reg.predict(X_testing)
            cal_errors(y_testing,y_pred)
```

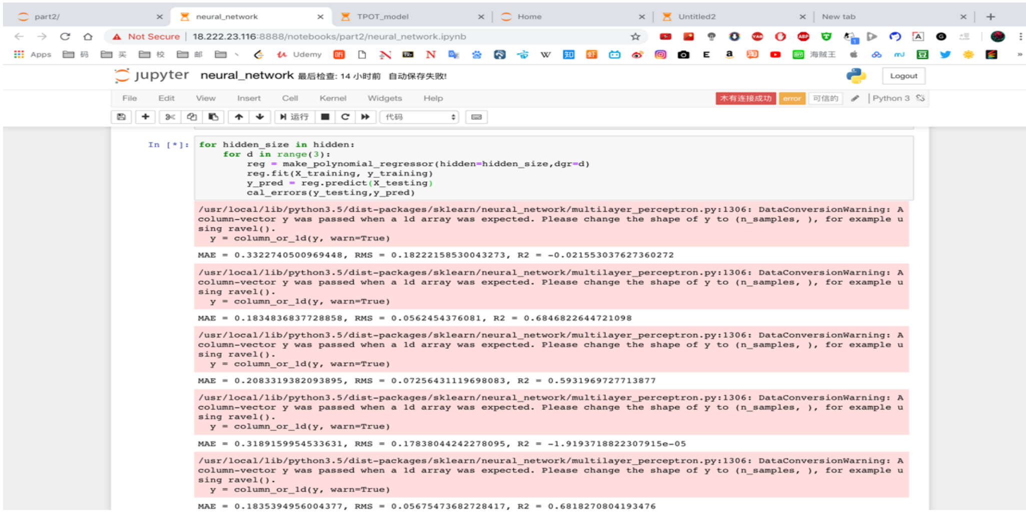
Grades:

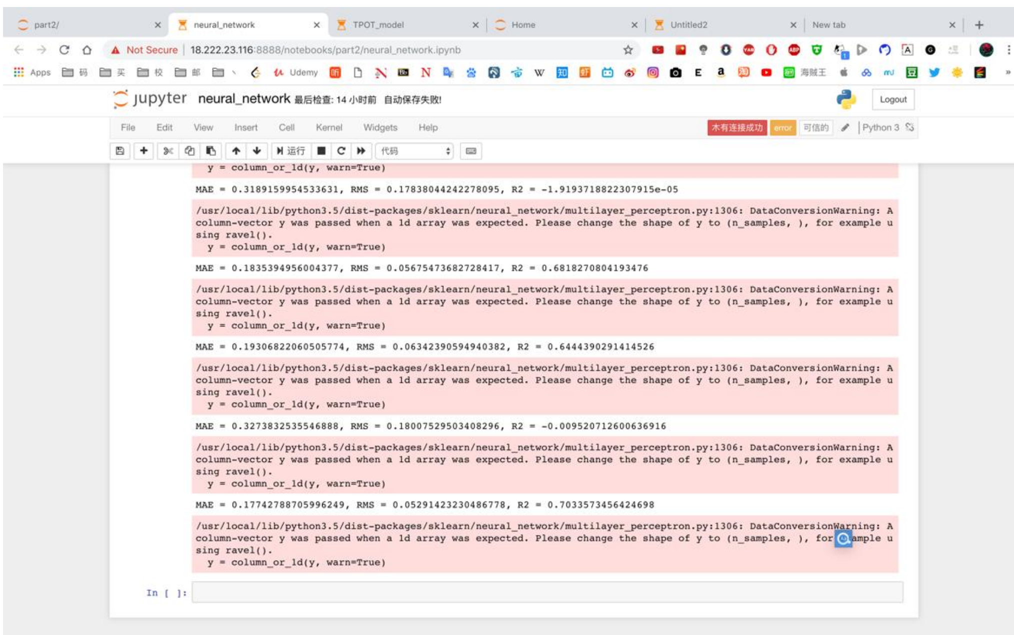
```python
In [*]: for hidden_size in hidden:
            reg = MLPRegressor(solver='adam', alpha=1e-5, hidden_layer_sizes=hidden_size, random_state =1)
            reg.fit(X_training, y_training)
            y_pred = reg.predict(X_testing)
            cal_errors(y_testing,y_pred)
```

```
/usr/local/lib/python3.5/dist-packages/sklearn/neural_network/multilayer_perceptron.py:1306: DataConversionWarning: A
column-vector y was passed when a 1d array was expected. Please change the shape of y to (n_samples, ), for example u
sing ravel().
  y = column_or_1d(y, warn=True)

MAE = 0.18325487547368038, RMS = 0.056112582149205684, R2 = 0.6854270659037025
MAE = 0.17748255440008645, RMS = 0.05286438838637418, R2 = 0.7036367758003209
MAE = 0.17846017667208686, RMS = 0.05379904303813672, R2 = 0.6983970052371005
```

Grades with different degrees:





We can see best grade is about 0.703, neural network has a good performance of training model for this dataset.

4) TPOT Model:

```
In [1]: from tpot import TPOTRegressor
        import pandas as pd
        from sklearn.model_selection import train_test_split

In [2]: X_training = pd.read_csv("./X_training.csv").values
        y_training = pd.read_csv("./y_training.csv", header = None).values
        X_testing = pd.read_csv("./X_testing.csv").values
        y_testing = pd.read_csv("./y_testing.csv", header = None).values

In [*]: tpot = TPOTRegressor(generations=5, population_size=10, verbosity=2)
        tpot.fit(X_training, y_training)
        print(tpot.score(X_testing, y_testing))
        tpot.export('tpot_assignment_3_pipeline.py')

        /usr/local/lib/python2.7/dist-packages/sklearn/utils/validation.py:752: DataConversionWarning: A column-vector y was
        passed when a 1d array was expected. Please change the shape of y to (n_samples, ), for example using ravel().
          y = column_or_1d(y, warn=True)

        Optimization Progress    63pipeline [2:19:24, 95.01s/pipeline]

        Generation 1 - Current best internal CV score: -0.074992296113
        Generation 2 - Current best internal CV score: -0.0748922936093
        Generation 3 - Current best internal CV score: -0.0748922936093

In [ ]:
```

```
In [*]: tpot = TPOTRegressor(generations=5, population_size=20, verbosity=2)
        tpot.fit(X_training, y_training)
        print(tpot.score(X_testing, y_testing))
        tpot.export('tpot_assignment_3_pipeline.py')

        /usr/local/lib/python2.7/dist-packages/sklearn/utils/validation.py:752: DataConversionWarning: A column-vector y was
        passed when a 1d array was expected. Please change the shape of y to (n_samples, ), for example using ravel().
          y = column_or_1d(y, warn=True)

        Optimization Progress    74% 89/120 [3:02:47<59:46, 115.70s/pipeline]

        Generation 1 - Current best internal CV score: -0.0646011301814
        Generation 2 - Current best internal CV score: -0.0585202189246
```

5) AutoML:

```
In [3]: automl = autosklearn.regression.AutoSklearnRegressor(
                per_run_time_limit=99,
                tmp_folder='/tmp/autosklearn_regression_Assignment_3_tmp',
                output_folder='/tmp/autosklearn_regression_Assignment_3_out',
            )

In [4]: automl.fit(X_training, y_training, X_testing, y_testing)

        /usr/local/lib/python3.5/dist-packages/autosklearn/automl.py:854: DataConversionWarning: A column-vector y was pass
        ed when a 1d array was expected. Will change shape via np.ravel().
          y = self._check_y(y)
        /usr/local/lib/python3.5/dist-packages/autosklearn/evaluation/train_evaluator.py:197: RuntimeWarning: Mean of empty
        slice
          Y_train_pred = np.nanmean(Y_train_pred_full, axis=0)

        [WARNING] [2018-11-29 09:21:51,149:EnsembleBuilder(1):290d472de0fc1923dc8697d8f205548a] No models better than rando
        m - using Dummy Score!
        [WARNING] [2018-11-29 09:21:51,554:AutoMLSMBO(1)::290d472de0fc1923dc8697d8f205548a] Could not find meta-data direct
        ory /usr/local/lib/python3.5/dist-packages/autosklearn/metalearning/files/r2_regression_dense
        [WARNING] [2018-11-29 09:21:52,935:EnsembleBuilder(1):290d472de0fc1923dc8697d8f205548a] No models better than rando
        m - using Dummy Score!
        [WARNING] [2018-11-29 09:21:54,941:EnsembleBuilder(1):290d472de0fc1923dc8697d8f205548a] No models better than rando
        m - using Dummy Score!
        [WARNING] [2018-11-29 09:21:56,946:EnsembleBuilder(1):290d472de0fc1923dc8697d8f205548a] No models better than rando
        m - using Dummy Score!
        [WARNING] [2018-11-29 09:21:58,954:EnsembleBuilder(1):290d472de0fc1923dc8697d8f205548a] No models better than rando
        m - using Dummy Score!
        [WARNING] [2018-11-29 09:22:00,959:EnsembleBuilder(1):290d472de0fc1923dc8697d8f205548a] No models better than rando

In [5]: print(automl.show_models())
        predictions = automl.predict(X_testing)
        print("R2 score:", sklearn.metrics.r2_score(y_testing, predictions))

        [(1.000000, MyDummyRegressor(configuration=1, init_params=None, random_state=None)),
        ]
        R2 score: -0.013691494993893771
```

We trained several times for these 2 models, but can’t get a good grade.

6) h2o.AI:

In this model, we can get a good performance when use its cross validation, as you can see grade is about 0.63

```
In [12]: perf = aml2.leader.model_performance(testing)
         perf
```

```
ModelMetricsRegressionGLM: stackedensemble
** Reported on test data. **

MSE: 0.04969705340454395
RMSE: 0.22292835935462305
MAE: 0.16750499838322996
RMSLE: 0.03335339367770484
R^2: 0.629109086591956
Mean Residual Deviance: 0.04969705340454395
Null degrees of freedom: 70133
Residual degrees of freedom: 70130
Null deviance: 9397.803624026541
Residual deviance: 3485.4531434742858
AIC: -11487.246784618035
```

But performance will be not as good as before when we use next quarter data to test it, grade is only 0.26.

```
In [7]: perf = aml.leader.model_performance(test)
        perf
```

```
ModelMetricsRegressionGLM: stackedensemble
** Reported on test data. **

MSE: 0.09062206722153747
RMSE: 0.30103499335050315
MAE: 0.23099075760032312
RMSLE: 0.044337414502309484
R^2: 0.2627142578998969
Mean Residual Deviance: 0.09062206722153747
Null degrees of freedom: 405678
Residual degrees of freedom: 405675
Null deviance: 56732.94213759017
Residual deviance: 36763.4696083661
AIC: 177218.51351249518
```

After all the work finished, we are ready to do feature selection.

We tried 3 methods (backward, forward, Exhaustive Search). But we have no time to get result to say which model after feature selection is the best. But I can say for now, neural network with 1 degree and hidden layer [88,70,50,30,10] has the best performance.

4. Classification:

In classification part, since the data is too large, we start an ec2 ubuntu instance p2.xlarge on AWS. This instance has 60GB ram and a nvidia high performance GPU. Otherwise fitting and modeling are impossible on our local machine.

1) **Data pre-processing**
   During pre- processing, features whose NaN values are more than 70% are dropped, and categorical data are transformed to binary data, datetime are separated into year and date. For Y values, ‘R’ is transformed to ‘-1’ in order to transform them into numerical data.
2) **Modeling**
   We chose different models including logisticregression, mlpclassifier, randomforestclassifier, autosklearn, tpotclassifier, and h2odeeplearningestimator. The training takes more than 4 hours but unfortunately we only got logisticregression model trained.
3) **Evaluation**
   For classification part, we drew roc curve and also confusion metrics instead of simply calculating the scores to evaluate the model, please refer to jupyter notebook for these graphs.

5. Pipeline and Dockerize the whole project:

After finishing all the different parts of work in the jupyter notebook, we made pipeline of the whole project. For the prediction part, we have made pipleline of our regression and classification part and created docker images for them separately. Then, we uploaded them to docker hub.

In the regression pipeline, we chose the forward search(we tried backward, forward and exhaustive search in different jupyter notebooks and finally chose the forward search) to do feature selection, and the best algorithm which is neural networks to act as the prediction model.