-

Advance Data Science
Single-family loan data Mid Term

Team 2

Abhinav Tiwari

Dharit Shah

Nilesh Nerkar

SUMMARY

The report summarizes the analysis performed on Single-family loan data, provided by Freddie Mac(http://www.freddiemac.com/news/finance/sf_loanlevel_dataset.html). The data provided by Freddie Mac consists of the following details:

Mortgages originated from January 1, 1999, through the "Origination Cutoff Date", with monthly loan performance data through the "Performance Cutoff Date," that were sold to Freddie Mac or back Freddie Mac Participation Certificates (PCs).
□ *Fully amortizing 15-, 20-, and 30-year fixed-rate mortgages*
□ *Mortgages categorized as having verified or waived documentation.*

We then build predictive analytics models using the datasets. The problem presented is divided into 2 section:

Section 1: Data wrangling

- Data Download and pre-processing
- Exploratory Data analysis

Section 2: Building and evaluating models

- **Prediction** using Linear Regression, Random Forest, Neural Network KNN Algorithms
- **Classification** using Logistic Regression, Random Forest, Neural Network, SVN Algorithms

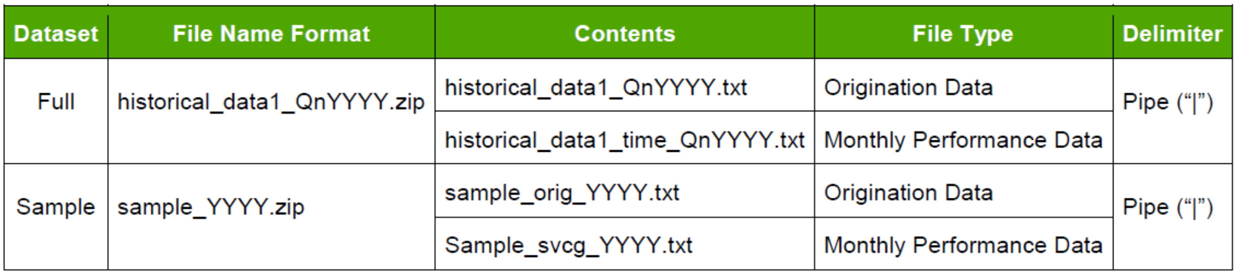
1 PART 1: DATA INGESTION AND WRANGLING

1.1 THE DATA

Single Family Loan-Level Dataset

For each calendar quarter, there is one file containing loan **origination data** and one file containing **monthly performance data** for each loan in the **origination data file**.

Freddie Mac has created a smaller dataset for those who may not require, or have the capability, to download the full Dataset. The sample dataset is a simple random sample6 of 50,000 loans selected from each full vintage year and a proportionate number of loans from each partial vintage year of the full Single Family Loan-Level Dataset. Each vintage year has one origination data file and one corresponding monthly performance data file, containing the same loan-level data fields as those included in the full Dataset. Due to the size of the dataset, the data has been broken up and compressed as detailed below. The files are organized chronologically by year and quarter.



Dataset	File Name Format	Contents	File Type	Delimiter
Full	historical_data1_QnYYYY.zip	historical_data1_QnYYYY.txt	Origination Data	Pipe ("|")
		historical_data1_time_QnYYYY.txt	Monthly Performance Data	
Sample	sample_YYYY.zip	sample_orig_YYYY.txt	Origination Data	Pipe ("|")
		Sample_svcg_YYYY.txt	Monthly Performance Data	

Data Download and pre-processing:

The very first challenge was to programmatically download the data from Freddie Mac website (https://freddiemac.embs.com/FLoan/Data/download.php) and download and preprocess the "SAMPLE" file both for origination and performance data.

To download the file programmatically, first the user should register him/herself by creating username and password. Once logged in, the user can download all the file required for analysis. We have used the python webbot library for this purpose. To store the user credential, we need to store them in the request session so that user didn't redirect back to the login page whenever he/she required to download a file from the Freddie Mac posted dataset.

Auto Navigate to the download page

Webbot - Web automation library for simple and easy web browser automation and offers extensive features but keeping it simple to use and master

```python
from webbot import Browser
from zipfile import ZipFile
import sys
import os
import time
import zipfile
import datetime
import glob
import pandas as pd

# Auto Navigate to the download page
def navigate(user='tiwari.abhi@husky.neu.edu', password='c`PD{0f8'):
    global web
    web = Browser()
    web.go_to("https://freddiemac.embs.com/FLoan/secure/login.php?pagename=download3")
    web.type(user, into='email')
    web.type(password, into='password')
    web.click('Submit Credentials')
    web.click('Yes')

    web.click('Continue')
```

Once, the user is logged in to the website, we will be using request session to drive the further functionality Now we will download from the Freddie Mac Website and download all the "**Sample**" files for our analysis purpose.

```python
def download_all_files(styear=2005, edyear=2006):
    years = list(range(styear, edyear + 1))

    global file_names
    file_names = []

    for year in years:
        web.click('sample_{}.zip'.format(str(year)))
        file_names.append('sample_{}.zip'.format(str(year)))
        print('sample_{}.zip has been downloaded'.format(str(year)))
        time.sleep(60)  # Some delay must be added since site blocks immediate requests in large numbers

    # Path to where files have been downloaded
    global download_path
    download_path = ''
    path_list = os.getcwd().split('\\')[:3]

    for item in path_list:
        download_path = download_path + item + '\\'

    download_path = download_path + 'Downloads'

    print(download_path)
```

```
def assure_path_exists(path):
    if not os.path.exists(path):
        os.makedirs(path)


def extract_zipped_files_to_cwd(path_to_downloaded_files):
    global folder_path_all_files
    folder_path_all_files = os.getcwd() + '\\' + 'Extracted Files'
    zip_ref = zipfile.ZipFile(path_to_downloaded_files, 'r')
    assure_path_exists(folder_path_all_files)
    zip_ref.extractall(folder_path_all_files)
    zip_ref.close()
```

1.2 Data Preprocessing and Cleaning

Since, these files are big in size and consist of huge amount of data, we need to preprocess these files before getting saved in our drive. These Zip file consist of two files:

Origination File:

For origination file, we first analyze the file size for all the year. Using python pandas, we create a data frame where we append all the data from the sample file for all the year. Origination file consist of 26 columns which consist of various details associated with the loan originated in each year.

We have many Null values and spaces (an invalid values) which we need to handle before using these files to compute a summary report. We must make sure that our data is in proper format with same datatype. We have created following functions:

```
def fillNAN_orig(df):
    df['fico'] = df['fico'].fillna(0)
    df['flag_fthb'] = df['flag_fthb'].fillna('X')
    df['cd_msa'] = df['cd_msa'].fillna(0)
    df['mi_pct'] = df['mi_pct'].fillna(0)
    df['cnt_units'] = df['cnt_units'].fillna(0)
    df['occpy_sts'] = df['occpy_sts'].fillna('X')
    df['cltv'] = df['cltv'].fillna(0)
    df['dti'] = df['dti'].fillna(0)
    df['ltv'] = df['ltv'].fillna(0)
    df['channel'] = df['channel'].fillna('X')
    df['ppmt_pnlty'] = df['ppmt_pnlty'].fillna('X')
    df['prop_type'] = df['prop_type'].fillna('XX')
    df['zipcode'] = df['zipcode'].fillna(0)
    df['loan_purpose'] = df['loan_purpose'].fillna('X')
    df['cnt_borr'] = df['cnt_borr'].fillna(0)
    df['flag_sc'] = df['flag_sc'].fillna('N')
    return df


def changedatatype_orig(df):
    # Change the data types for all column
    df[['fico', 'cd_msa', 'mi_pct', 'cnt_borr', 'cnt_units', 'cltv', 'dti', 'orig_upb', 'ltv', 'zipcode',
        'orig_loan_term']] = df[['fico', 'cd_msa', 'mi_pct', 'cnt_borr', 'cnt_units', 'cltv', 'dti',
'orig_upb', 'ltv', 'zipcode','orig_loan_term']].astype('int64')
    df[['flag_sc', 'servicer_name']] = df[['flag_sc', 'servicer_name']].astype('str')
    return df
```

Performance File:

For performance file, we first analyze the file size for all the year. Using python pandas, we create a data frame where we append all the data for all the year. Performance file consist of 23 columns which consist of various information about the loan origination in a year. Since, the size of the file is very large, we decide to summarizes the input file for all the year during its preprocessing. Some of the major columns are defined below:

As we have many empty column values in our origination and preprocessing file, we need to clean those to ensure that we don't have any NAN/NA value in our data. Also, we need to take care of the data type of column. These columns will be required while creating the summary matrices.

```
def fillNA_performance(df):
    df['current_upb'] = df['current_upb'].fillna(0)
    df['delq_sts'] = df['delq_sts'].fillna('XX')
    df['loan_age'] = df['loan_age'].fillna(-1)
    df['mths_remng'] = df['mths_remng'].fillna(-1)
    df['repch_flag'] = df['repch_flag'].fillna('X')
    df['flag_mod'] = df['flag_mod'].fillna('N')
    df['cd_zero_bal'] = df['cd_zero_bal'].fillna(-1)
    df['dt_zero_bal'] = df['dt_zero_bal'].fillna('189901')
    df['current_int_rt'] = df['current_int_rt'].fillna(0)
    df['current_dfr_upb'] = df['current_dfr_upb'].fillna(0)
    df['dt_lst_pi'] = df['dt_lst_pi'].fillna('189901')
    df['mi_recoveries'] = df['mi_recoveries'].fillna(0)
    df['net_sale_proceeds'] = df['net_sale_proceeds'].fillna('U')
    df['non_mi_recoveries'] = df['non_mi_recoveries'].fillna(0)
    df['expenses'] = df['expenses'].fillna(0)
    df['legal_costs'] = df['legal_costs'].fillna(0)
    df['maint_pres_costs'] = df['maint_pres_costs'].fillna(0)
    df['taxes_ins_costs'] = df['taxes_ins_costs'].fillna(0)
    df['misc_costs'] = df['misc_costs'].fillna(0)
    df['actual_loss'] = df['actual_loss'].fillna(0)
    df['modcost'] = df['modcost'].fillna(0)
    df['step_mod_flag'] = df['step_mod_flag'].fillna('X')
    df['def_py_mod'] = df['def_py_mod'].fillna('X')
    df['eltv'] = df['eltv'].fillna(0)
    return df
```

```
def changedatatype_performance(df):
    # Change the data types for all column
    df[['loan_seq', 'delq_sts', 'repch_flag', 'flag_mod', 'net_sale_proceeds', 'step_mod_flag', 'def_py_mod']]
= df[['loan_seq', 'delq_sts', 'repch_flag', 'flag_mod', 'net_sale_proceeds', 'step_mod_flag',
'def_py_mod']].astype('str')
    df[['mth_per', 'loan_age', 'mths_remng', 'dt_zero_bal']] = df[['mth_per', 'loan_age', 'mths_remng',
'dt_zero_bal']].astype('int64')
    df[['current_upb', 'cd_zero_bal', 'current_int_rt', 'current_dfr_upb', 'dt_lst_pi',
'mi_recoveries','non_mi_recoveries', 'expenses', 'legal_costs', 'maint_pres_costs', 'taxes_ins_costs',
'misc_costs','actual_loss', 'modcost']] = df[['current_upb', 'cd_zero_bal', 'current_int_rt',
'current_dfr_upb', 'dt_lst_pi', 'mi_recoveries','non_mi_recoveries', 'expenses', 'legal_costs',
'maint_pres_costs', 'taxes_ins_costs', 'misc_costs','actual_loss', 'modcost']].astype('float32')
    return df
```

Once, we are done with the cleaning of the performance file, we will create a summarized version of the file based on certain column which are important for us. For this step, we created a function which will get the Max/Min/Average value of the columns in our summarized performance file.

```
def get_current_upb(group):
    return {'min_current_upb': group.min(), 'max_current_upb': group.max()}


def get_delq_sts(group):
    return {'min_delq_sts': group.min(), 'max_delq_sts': group.max()}


def get_cd_zero_bal(group):
    return {'min_cd_zero_bal': group.min(), 'max_cd_zero_bal': group.max()}


def get_mi_recoveries(group):
    return {'min_mi_recoveries': group.min(), 'max_mi_recoveries': group.max()}


def get_net_sale_proceeds(group):
    return {'min_net_sale_proceeds': group.min(), 'max_net_sale_proceeds': group.max()}


def get_non_mi_recoveries(group):
    return {'min_non_mi_recoveries': group.min(), 'max_non_mi_recoveries': group.max()}


def get_expenses(group):
    return {'min_expenses': group.min(), 'max_expenses': group.max()}


def get_legal_costs(group):
    return {'min_legal_costs': group.min(), 'max_legal_costs': group.max()}
```

```
def get_maint_pres_costs(group):
    return {'min_maint_pres_costs': group.min(), 'max_maint_pres_costs': group.max()}


def get_taxes_ins_costs(group):
    return {'min_taxes_ins_costs': group.min(), 'max_taxes_ins_costs': group.max()}


def get_misc_costs(group):
    return {'min_misc_costs': group.min(), 'max_misc_costs': group.max()}


def get_actual_loss(group):
    return {'min_actual_loss': group.min(), 'max_actual_loss': group.max()}


def get_modcost(group):
    return {'min_modcost': group.min(), 'max_modcost': group.max()}
```

1.3 Creating Summarized CSV file (Output)

Once the preprocessing and data cleaning steps are performed, we will have created our final output file, one for origination file named ‘OriginationCombinedCode’ and for performance file named ‘PerformanceCombinedSummary’. These final files will be used for our analysis performed in part 2.

We have also created some derived column like ‘Year’ and ‘Quarter’ which will help to create the summary metrics.

```
def createPerformanceCombined(sample_perf_files):
    writeHeader2 = True
    if "sample" in sample_perf_files:
        filename = "Sample Performance Combined Summary.csv"

    abc = glob.glob(sample_perf_files)

    with open(filename, 'w', encoding='utf-8', newline="") as file:
        for f in abc:
            perf_df = pd.read_csv(f, sep='|',
                                  names=['loan_seq', 'mth_per', 'current_upb', 'delq_sts', 'loan_age',
'mths_remng',
                                         'repch_flag', 'flag_mod', 'cd_zero_bal', 'dt_zero_bal',
'current_int_rt',
                                         'current_dfr_upb', 'dt_lst_pi', 'mi_recoveries', 'net_sale_proceeds',
                                         'non_mi_recoveries', 'expenses', 'legal_costs', 'maint_pres_costs',
                                         'taxes_ins_costs', 'misc_costs', 'actual_loss', 'modcost',
'step_mod_flag',
                                         'def_py_mod', 'eltv'], skipinitialspace=False, low_memory=False)

            perf_df = fillNA_performance(perf_df)
            perf_df = changedatatype_performance(perf_df)
            summ_df = pd.DataFrame()
            summ_df['loan_seq'] = perf_df['loan_seq'].drop_duplicates()
            summ_df = summ_df.join(
                (perf_df['current_upb'].groupby(perf_df['loan_seq']).apply(get_current_upb).unstack()),
on='loan_seq')
            summ_df =
summ_df.join((perf_df['delq_sts'].groupby(perf_df['loan_seq']).apply(get_delq_sts).unstack()),
                                   on='loan_seq')
            summ_df = summ_df.join(
                (perf_df['cd_zero_bal'].groupby(perf_df['loan_seq']).apply(get_cd_zero_bal).unstack()),
on='loan_seq')
            summ_df = summ_df.join(

(perf_df['non_mi_recoveries'].groupby(perf_df['loan_seq']).apply(get_non_mi_recoveries).unstack()),
                on='loan_seq')
            summ_df =
summ_df.join((perf_df['expenses'].groupby(perf_df['loan_seq']).apply(get_expenses).unstack()),
                                   on='loan_seq')
            summ_df = summ_df.join(
                (perf_df['legal_costs'].groupby(perf_df['loan_seq']).apply(get_legal_costs).unstack()),
on='loan_seq')
            summ_df = summ_df.join(

(perf_df['maint_pres_costs'].groupby(perf_df['loan_seq']).apply(get_maint_pres_costs).unstack()),
                on='loan_seq')
            summ_df = summ_df.join(
                (perf_df['taxes_ins_costs'].groupby(perf_df['loan_seq']).apply(get_taxes_ins_costs).unstack()),
                on='loan_seq')
            summ_df =
summ_df.join((perf_df['misc_costs'].groupby(perf_df['loan_seq']).apply(get_misc_costs).unstack()),
                                   on='loan_seq')
            summ_df = summ_df.join(
                (perf_df['actual_loss'].groupby(perf_df['loan_seq']).apply(get_actual_loss).unstack()),
on='loan_seq')
            summ_df =
summ_df.join((perf_df['modcost'].groupby(perf_df['loan_seq']).apply(get_modcost).unstack()),
                                   on='loan_seq')

            if writeHeader2 is True:
                summ_df.to_csv(file, mode='a', header=True, index=False)
                writeHeader2 = False
            else:

                summ_df.to_csv(file, mode='a', header=False, index=False)
```

To validate year we have defined function called validate_year(2005-2017)

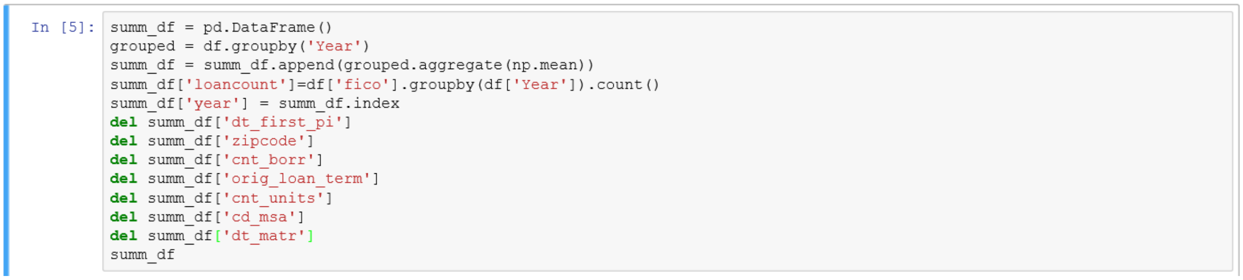
```
def validate_year(year):
    try:
        year = int(year)
    except:
        print('Cannot Parse Year to Integer')
        exit(0)

    if not year in [2005,2006,2007,2008,2009,2010,2011,2012,2013,2014,2015,2016,2017]:
        print('Data unavailabale for the requested year')
        exit(0)
    else:
        return year
```

NOTE: As we are working on two different file, we need to know about the data and the relationship between the two files created. In origination file, we have idloan which is a unique loan sequence number with quarter and year of loan origination attached to it. In performance file, for a given year we have multiple rows associated with a loan number which depicts its performance. We don’t have any loan number duplicated in origination file with respect to year. It is always **Unique**.

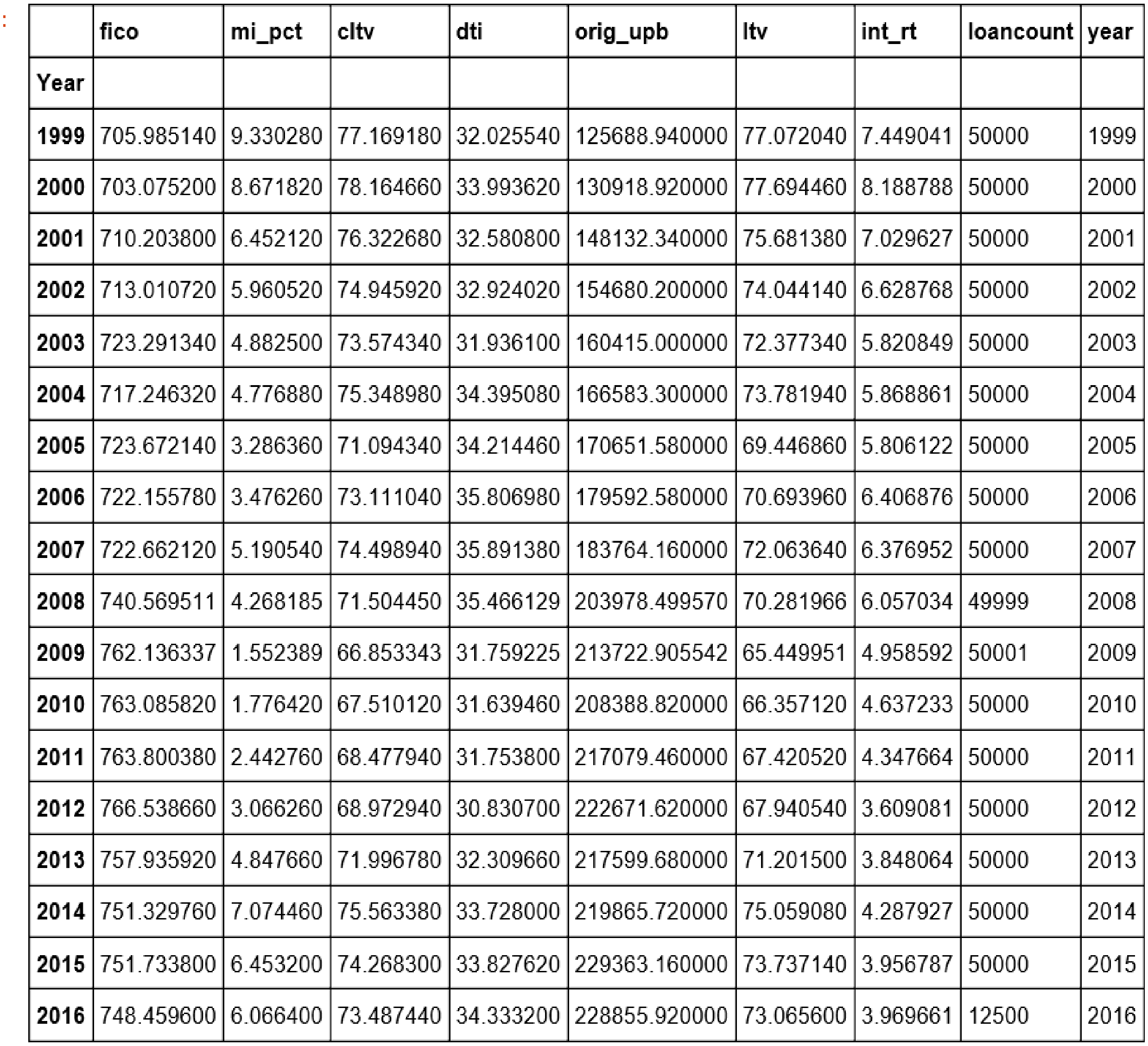
PART 2: Exploratory Data Analysis

In Part 2, we were asked to write Jupyter notebook using Python to graphically represent different summaries of data and summarize our findings in this notebook.

We first create a pandas' data frame for the origination file and group the data on the year.



```
In [5]: summ_df = pd.DataFrame()
        grouped = df.groupby('Year')
        summ_df = summ_df.append(grouped.aggregate(np.mean))
        summ_df['loancount']=df['fico'].groupby(df['Year']).count()
        summ_df['year'] = summ_df.index
        del summ_df['dt_first_pi']
        del summ_df['zipcode']
        del summ_df['cnt_borr']
        del summ_df['orig_loan_term']
        del summ_df['cnt_units']
        del summ_df['cd_msa']
        del summ_df['dt_matr']
        summ_df
```



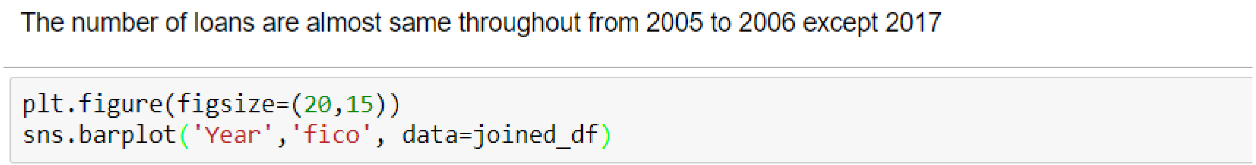
:

	fico	mi_pct	cltv	dti	orig_upb	ltv	int_rt	loancount	year
Year									
1999	705.985140	9.330280	77.169180	32.025540	125688.940000	77.072040	7.449041	50000	1999
2000	703.075200	8.671820	78.164660	33.993620	130918.920000	77.694460	8.188788	50000	2000
2001	710.203800	6.452120	76.322680	32.580800	148132.340000	75.681380	7.029627	50000	2001
2002	713.010720	5.960520	74.945920	32.924020	154680.200000	74.044140	6.628768	50000	2002
2003	723.291340	4.882500	73.574340	31.936100	160415.000000	72.377340	5.820849	50000	2003
2004	717.246320	4.776880	75.348980	34.395080	166583.300000	73.781940	5.868861	50000	2004
2005	723.672140	3.286360	71.094340	34.214460	170651.580000	69.446860	5.806122	50000	2005
2006	722.155780	3.476260	73.111040	35.806980	179592.580000	70.693960	6.406876	50000	2006
2007	722.662120	5.190540	74.498940	35.891380	183764.160000	72.063640	6.376952	50000	2007
2008	740.569511	4.268185	71.504450	35.466129	203978.499570	70.281966	6.057034	49999	2008
2009	762.136337	1.552389	66.853343	31.759225	213722.905542	65.449951	4.958592	50001	2009
2010	763.085820	1.776420	67.510120	31.639460	208388.820000	66.357120	4.637233	50000	2010
2011	763.800380	2.442760	68.477940	31.753800	217079.460000	67.420520	4.347664	50000	2011
2012	766.538660	3.066260	68.972940	30.830700	222671.620000	67.940540	3.609081	50000	2012
2013	757.935920	4.847660	71.996780	32.309660	217599.680000	71.201500	3.848064	50000	2013
2014	751.329760	7.074460	75.563380	33.728000	219865.720000	75.059080	4.287927	50000	2014
2015	751.733800	6.453200	74.268300	33.827620	229363.160000	73.737140	3.956787	50000	2015
2016	748.459600	6.066400	73.487440	34.333200	228855.920000	73.065600	3.969661	12500	2016

Here we show the various details associated with the origination file and the total loan count and mean of various important factor like fico score, interest rate based on year

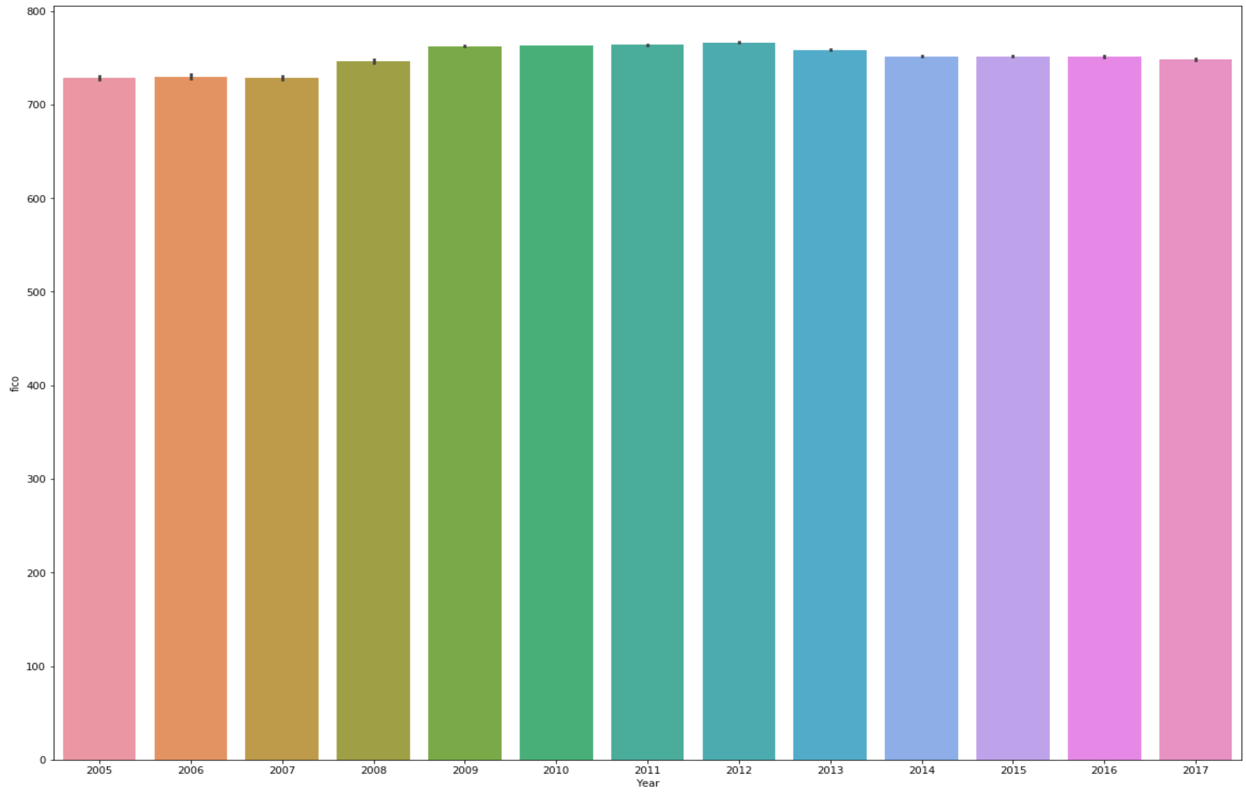
FICO Based on Count

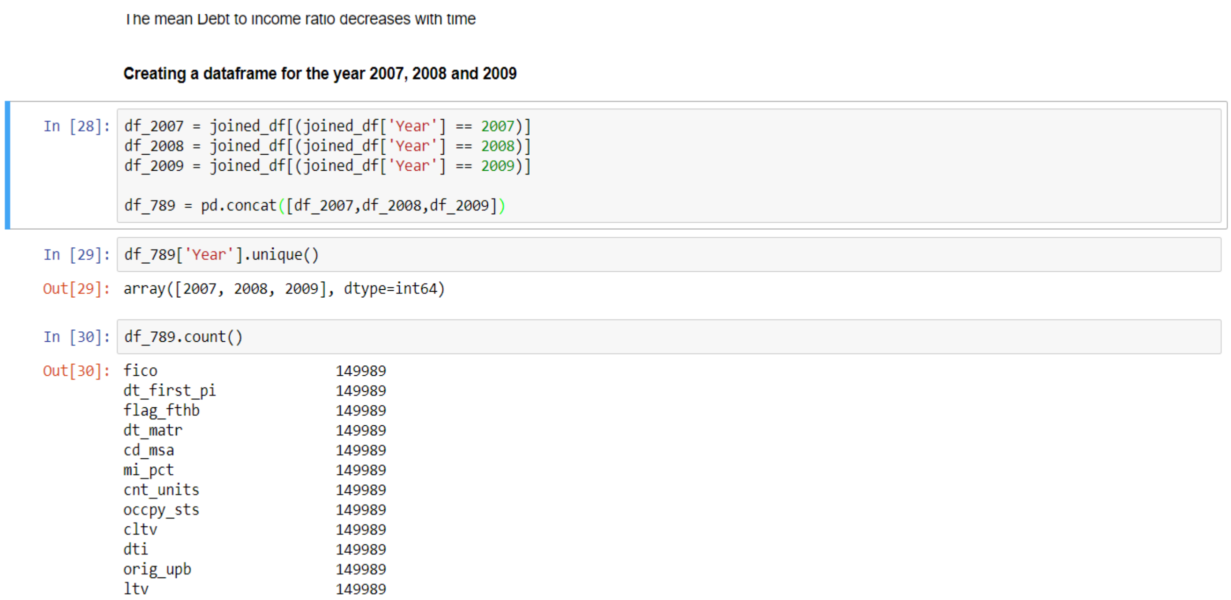
The number of loans are almost same throughout from 2005 to 2006 except 2017



The number of loans are almost same throughout from 2005 to 2006 except 2017

```
plt.figure(figsize=(20,15))
sns.barplot('Year','fico', data=joined_df)
```



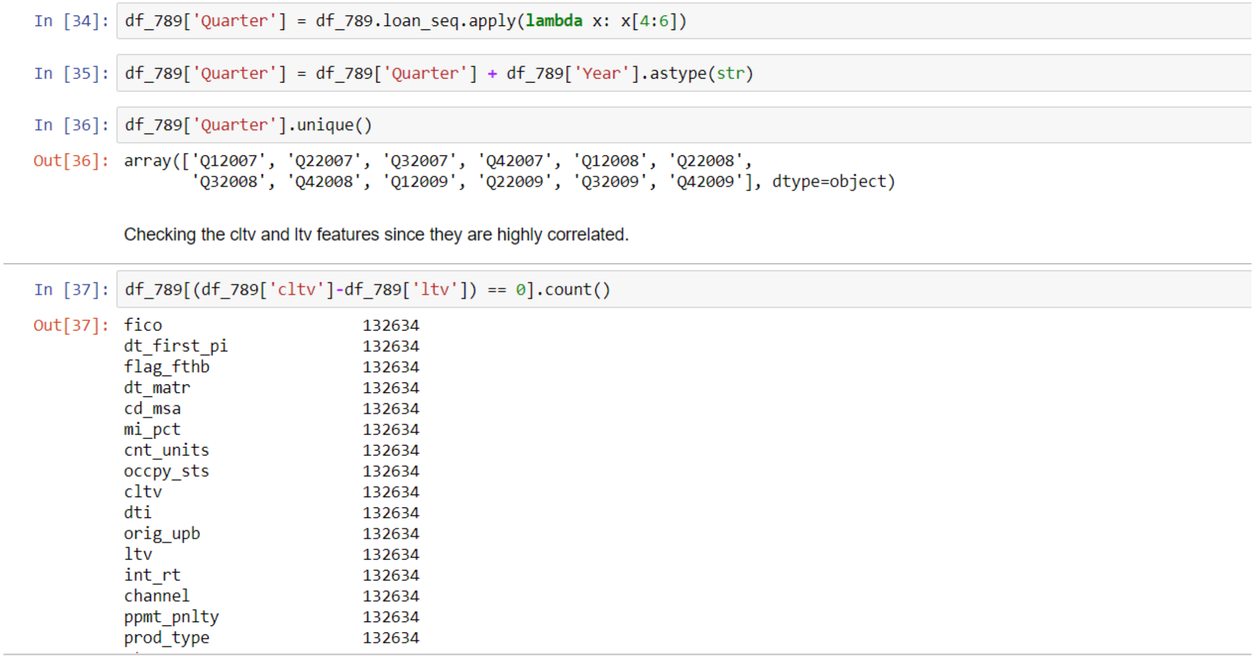


The mean Debt to income ratio decreases with time

Creating a dataframe for the year 2007, 2008 and 2009

```
In [28]: df_2007 = joined_df[(joined_df['Year'] == 2007)]
         df_2008 = joined_df[(joined_df['Year'] == 2008)]
         df_2009 = joined_df[(joined_df['Year'] == 2009)]

         df_789 = pd.concat([df_2007,df_2008,df_2009])

In [29]: df_789['Year'].unique()

Out[29]: array([2007, 2008, 2009], dtype=int64)

In [30]: df_789.count()

Out[30]: fico                     149989
         dt_first_pi              149989
         flag_fthb                149989
         dt_matr                  149989
         cd_msa                   149989
         mi_pct                   149989
         cnt_units                149989
         occpy_sts                149989
         cltv                     149989
         dti                      149989
         orig_upb                 149989
         ltv                      149989
```

Loan Term and date of maturity are correlated, there are many correlation in the later part which are inter related(expenses, actual loss,legal costs, maintaince costs, taxes and misc costs)

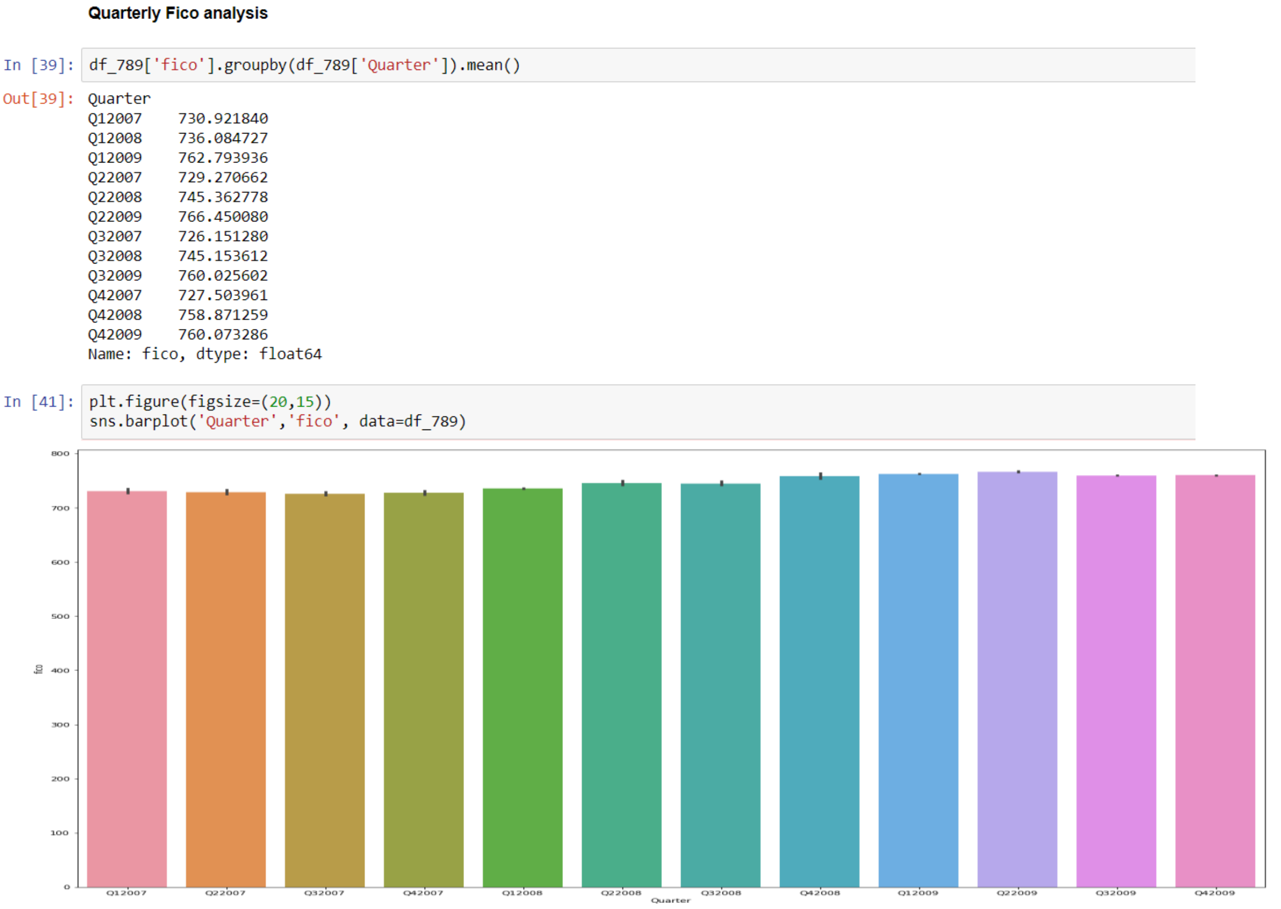


```
In [34]: df_789['Quarter'] = df_789.loan_seq.apply(lambda x: x[4:6])

In [35]: df_789['Quarter'] = df_789['Quarter'] + df_789['Year'].astype(str)

In [36]: df_789['Quarter'].unique()

Out[36]: array(['Q12007', 'Q22007', 'Q32007', 'Q42007', 'Q12008', 'Q22008',
                'Q32008', 'Q42008', 'Q12009', 'Q22009', 'Q32009', 'Q42009'], dtype=object)
```

Checking the cltv and ltv features since they are highly correlated.

```
In [37]: df_789[(df_789['cltv']-df_789['ltv']) == 0].count()

Out[37]: fico                  132634
         dt_first_pi           132634
         flag_fthb             132634
         dt_matr               132634
         cd_msa                132634
         mi_pct                132634
         cnt_units             132634
         occpy_sts             132634
         cltv                  132634
         dti                   132634
         orig_upb              132634
         ltv                   132634
         int_rt                132634
         channel               132634
         ppmt_pnlty            132634
         prod_type             132634
```

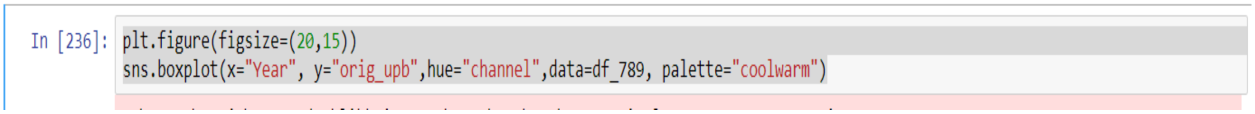
Exploring the quarterly trends of features



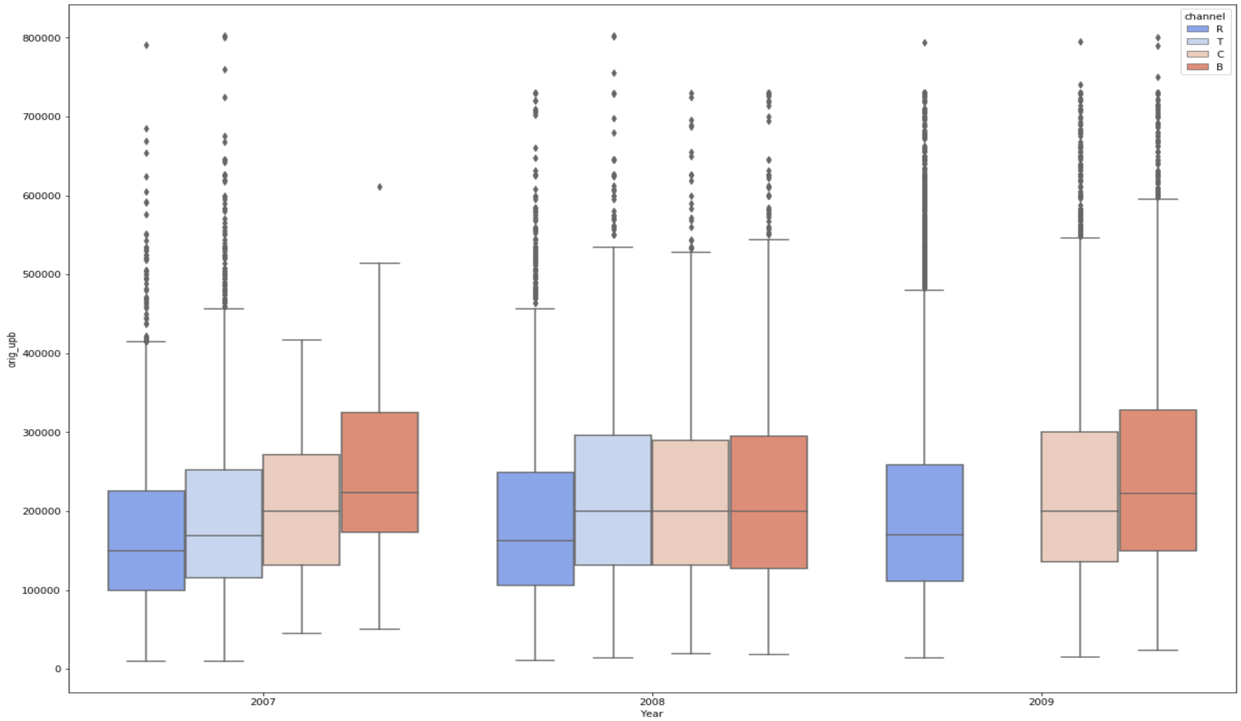
Quarterly Fico analysis

```
In [39]: df_789['fico'].groupby(df_789['Quarter']).mean()

Out[39]: Quarter
         Q12007    730.921840
         Q12008    736.084727
         Q12009    762.793936
         Q22007    729.270662
         Q22008    745.362778
         Q22009    766.450080
         Q32007    726.151280
         Q32008    745.153612
         Q32009    760.025602
         Q42007    727.503961
         Q42008    758.871259
         Q42009    760.073286
         Name: fico, dtype: float64

In [41]: plt.figure(figsize=(20,15))
         sns.barplot('Quarter','fico', data=df_789)
```

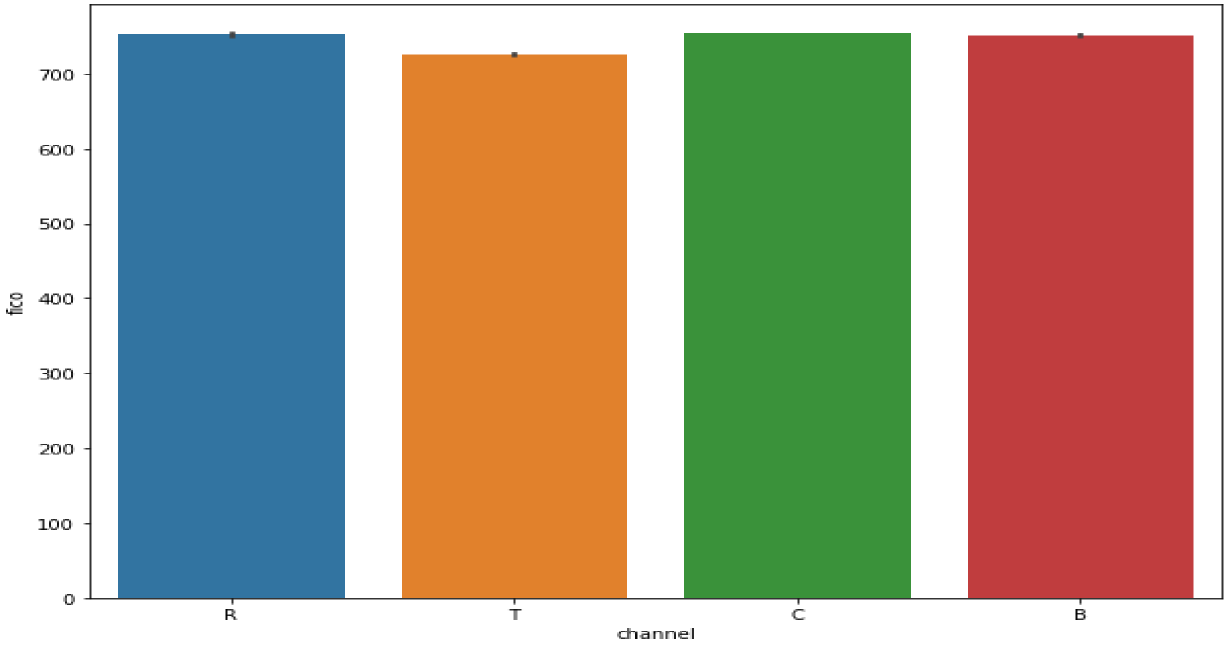
Least number of loans were taken in the quarter 4 of 2007 and the highest was in the first quarter of 2009



```
In [236]: plt.figure(figsize=(20,15))
          sns.boxplot(x="Year", y="orig_upb",hue="channel",data=df_789, palette="coolwarm")
```



Most of loan involved in the occupancy status investement property had less average UPB(unpaid balance) value



2 PART II: Building & Evaluating Models

PREDICTION (Predicting Interest Rates)

Here we are going to create a predictive model based on information from the origination data from the prior quarter using the various regression technique to calculate the following metrics:

MAE (Mean Absolute Error) - In statistics, the mean absolute error (MAE) is a quantity used to measure how close forecasts or predictions are to the eventual outcomes.

RMSE (Root Mean Square Error) - The root-mean-square deviation (RMSD) or root-mean-square error (RMSE) is a frequently used measure of the differences between values (sample and population values) predicted by a model or an estimator and the values observed. The RMSD represents the sample standard deviation of the differences between predicted values and observed values. These individual differences are called residuals when the calculations are performed over the data sample that was used for estimation, and are called prediction errors when computed out-of-sample. The RMSD serves to aggregate the magnitudes of the errors in predictions for various times into a single measure of predictive power. RMSD is a good measure of accuracy, but only to compare forecasting errors of different models for a particular variable and not between variables, as it is scale-dependent

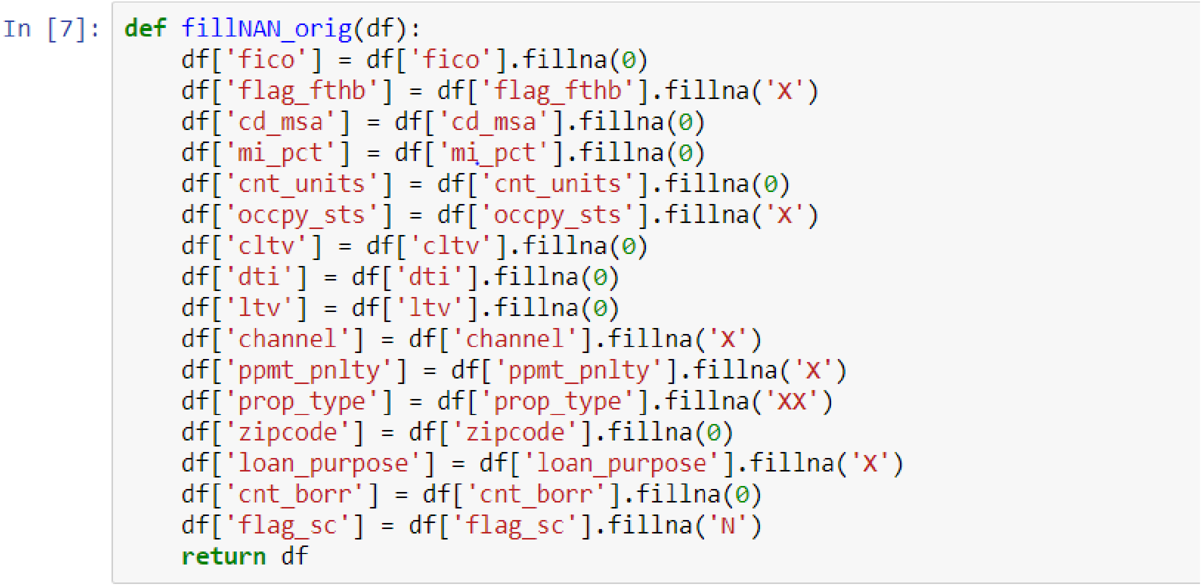
Median Absolute Error- The median_absolute_error is particularly interesting because it is robust to outliers. The loss is calculated by taking the median of all absolute differences between the target and the prediction.

2.1 APPROCH - PREDICTION

2.1.1 CREATING THE FUNCTION AND DOWNLOAD THE INPUT

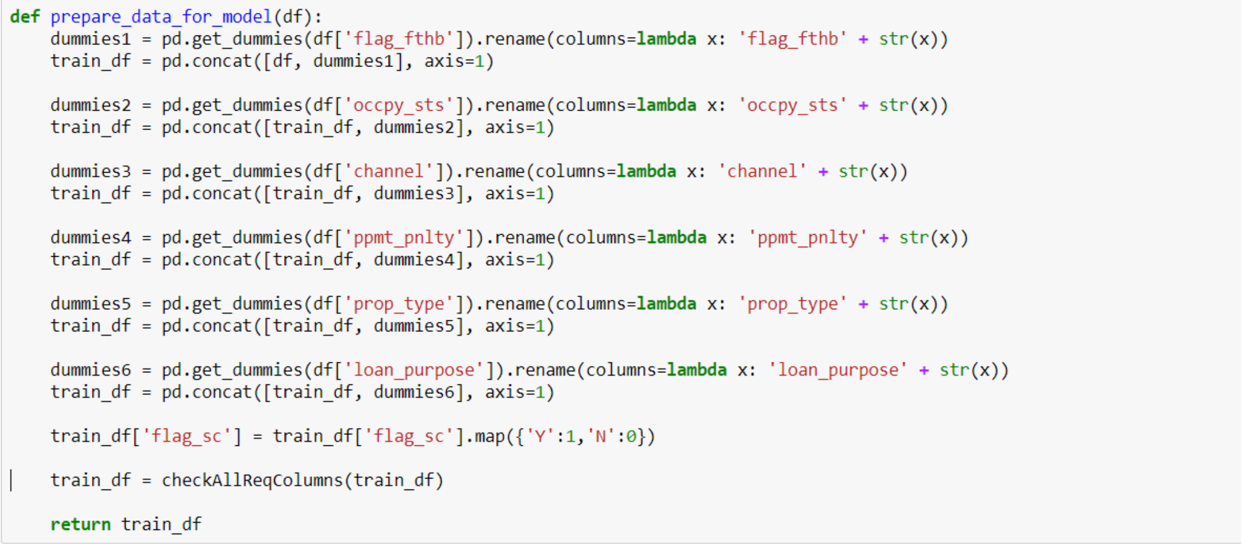
We have created functions that would take hit the Freddie Mac website and download all the Historical file for our purpose. Based on the input Quarter, the file will be imported into python Data Frame. It went through following preprocessing before building model.

Handling the missing values:



```
In [7]: def fillNAN_orig(df):
            df['fico'] = df['fico'].fillna(0)
            df['flag_fthb'] = df['flag_fthb'].fillna('X')
            df['cd_msa'] = df['cd_msa'].fillna(0)
            df['mi_pct'] = df['mi_pct'].fillna(0)
            df['cnt_units'] = df['cnt_units'].fillna(0)
            df['occpy_sts'] = df['occpy_sts'].fillna('X')
            df['cltv'] = df['cltv'].fillna(0)
            df['dti'] = df['dti'].fillna(0)
            df['ltv'] = df['ltv'].fillna(0)
            df['channel'] = df['channel'].fillna('X')
            df['ppmt_pnlty'] = df['ppmt_pnlty'].fillna('X')
            df['prop_type'] = df['prop_type'].fillna('XX')
            df['zipcode'] = df['zipcode'].fillna(0)
            df['loan_purpose'] = df['loan_purpose'].fillna('X')
            df['cnt_borr'] = df['cnt_borr'].fillna(0)
            df['flag_sc'] = df['flag_sc'].fillna('N')
            return df
```

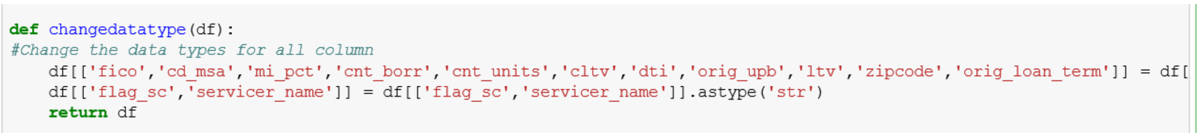
Created dummy columns (1…C):



```
def prepare_data_for_model(df):
    dummies1 = pd.get_dummies(df['flag_fthb']).rename(columns=lambda x: 'flag_fthb' + str(x))
    train_df = pd.concat([df, dummies1], axis=1)

    dummies2 = pd.get_dummies(df['occpy_sts']).rename(columns=lambda x: 'occpy_sts' + str(x))
    train_df = pd.concat([train_df, dummies2], axis=1)

    dummies3 = pd.get_dummies(df['channel']).rename(columns=lambda x: 'channel' + str(x))
    train_df = pd.concat([train_df, dummies3], axis=1)

    dummies4 = pd.get_dummies(df['ppmt_pnlty']).rename(columns=lambda x: 'ppmt_pnlty' + str(x))
    train_df = pd.concat([train_df, dummies4], axis=1)

    dummies5 = pd.get_dummies(df['prop_type']).rename(columns=lambda x: 'prop_type' + str(x))
    train_df = pd.concat([train_df, dummies5], axis=1)

    dummies6 = pd.get_dummies(df['loan_purpose']).rename(columns=lambda x: 'loan_purpose' + str(x))
    train_df = pd.concat([train_df, dummies6], axis=1)

    train_df['flag_sc'] = train_df['flag_sc'].map({'Y':1,'N':0})

    train_df = checkAllReqColumns(train_df)

    return train_df
```

2.1.2 CONVERSION OF DATA TYPE

Data types of the History Origination file is converted as below.



```
def changedatatype(df):
#Change the data types for all column
    df[['fico','cd_msa','mi_pct','cnt_borr','cnt_units','cltv','dti','orig_upb','ltv','zipcode','orig_loan_term']] = df[
    df[['flag_sc','servicer_name']] = df[['flag_sc','servicer_name']].astype('str')
    return df
```

2.1.3 DIFFERENT MACHINE LEARNING ALGORITHMS AND OUTPUT

2.1.3.1 REGRESSION

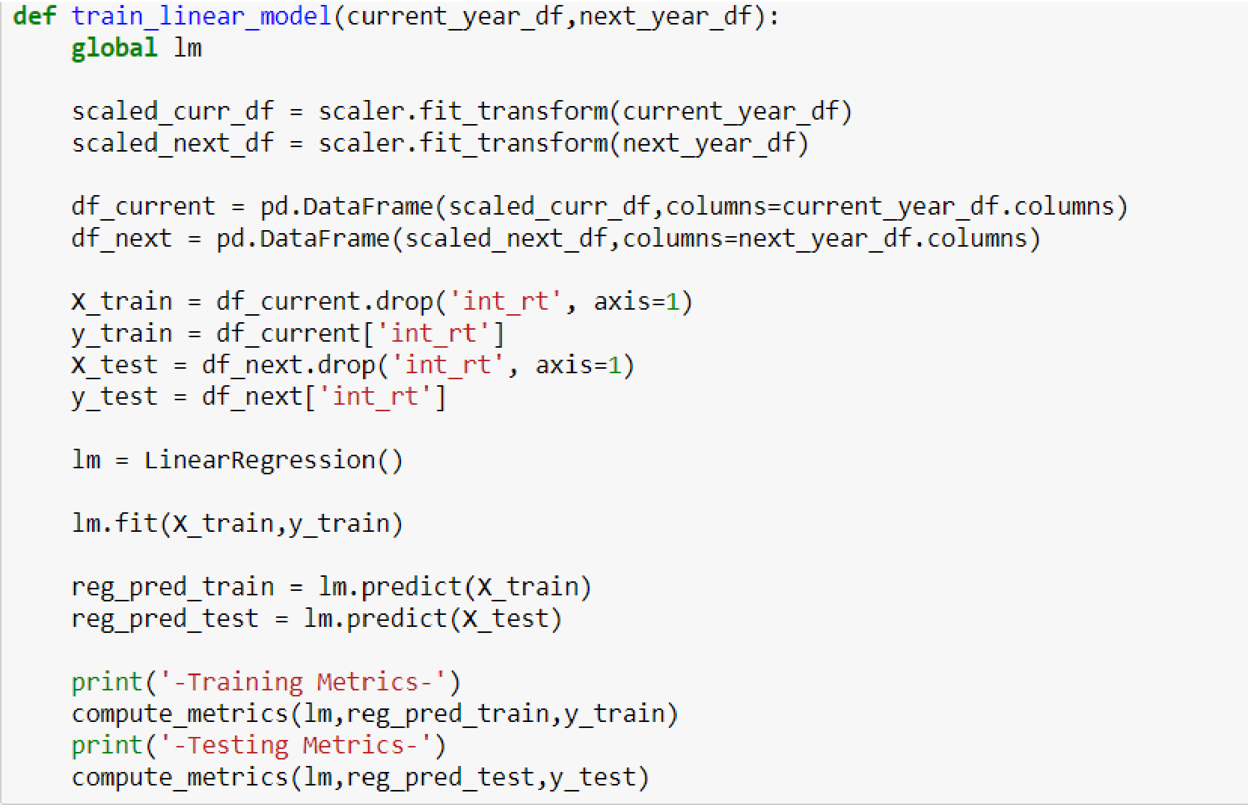
We have used the following regression techniques and compared the R2 and selected Liner Regression.

Regression using Linear Regression:

sklearn.linear_model.LinearRegression : Ordinary least squares Linear Regression.

Implementing Liner Regression

Linear regression is an approach for modeling the relationship between a scalar dependent variable y and one or more explanatory variables.



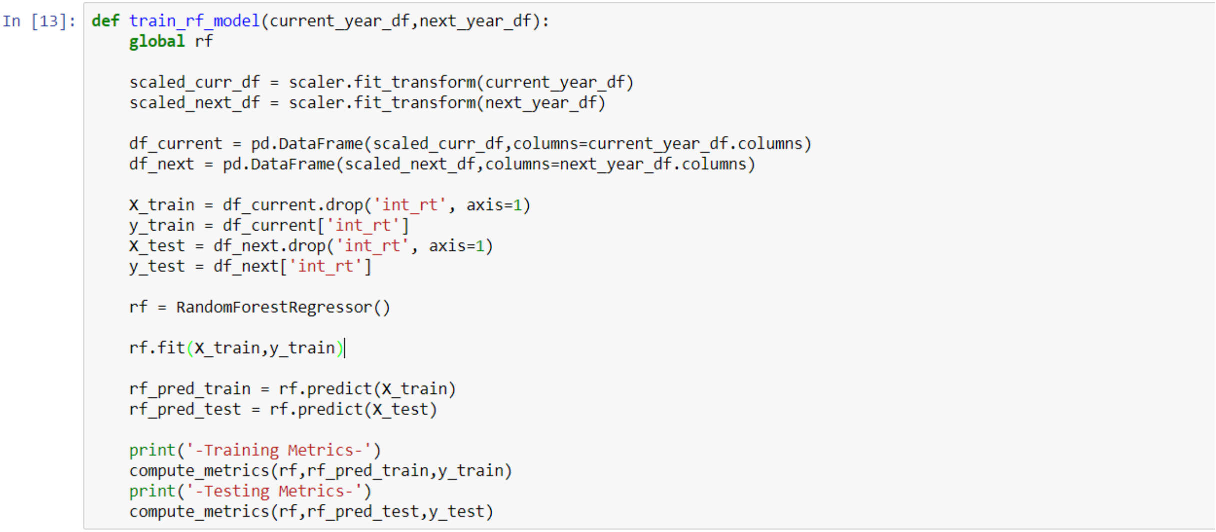
```
def train_linear_model(current_year_df,next_year_df):
    global lm

    scaled_curr_df = scaler.fit_transform(current_year_df)
    scaled_next_df = scaler.fit_transform(next_year_df)

    df_current = pd.DataFrame(scaled_curr_df,columns=current_year_df.columns)
    df_next = pd.DataFrame(scaled_next_df,columns=next_year_df.columns)

    X_train = df_current.drop('int_rt', axis=1)
    y_train = df_current['int_rt']
    X_test = df_next.drop('int_rt', axis=1)
    y_test = df_next['int_rt']

    lm = LinearRegression()

    lm.fit(X_train,y_train)

    reg_pred_train = lm.predict(X_train)
    reg_pred_test = lm.predict(X_test)

    print('-Training Metrics-')
    compute_metrics(lm,reg_pred_train,y_train)
    print('-Testing Metrics-')
    compute_metrics(lm,reg_pred_test,y_test)
```

2.1.4.3 RANDOM FOREST

The random forest starts with a standard machine learning technique called a “decision tree”. This is a type of additive model that makes predictions by combining decisions from a sequence of base models.

sklearn.ensemble.RandomForestRegressor

A random forest is a meta estimator that fits a number of classifying decision trees on various sub-samples of the dataset and use averaging to improve the predictive accuracy and control over-fitting. The sub-sample size is always the same as the original input sample size but the samples are drawn with replacement if bootstrap=True (default).

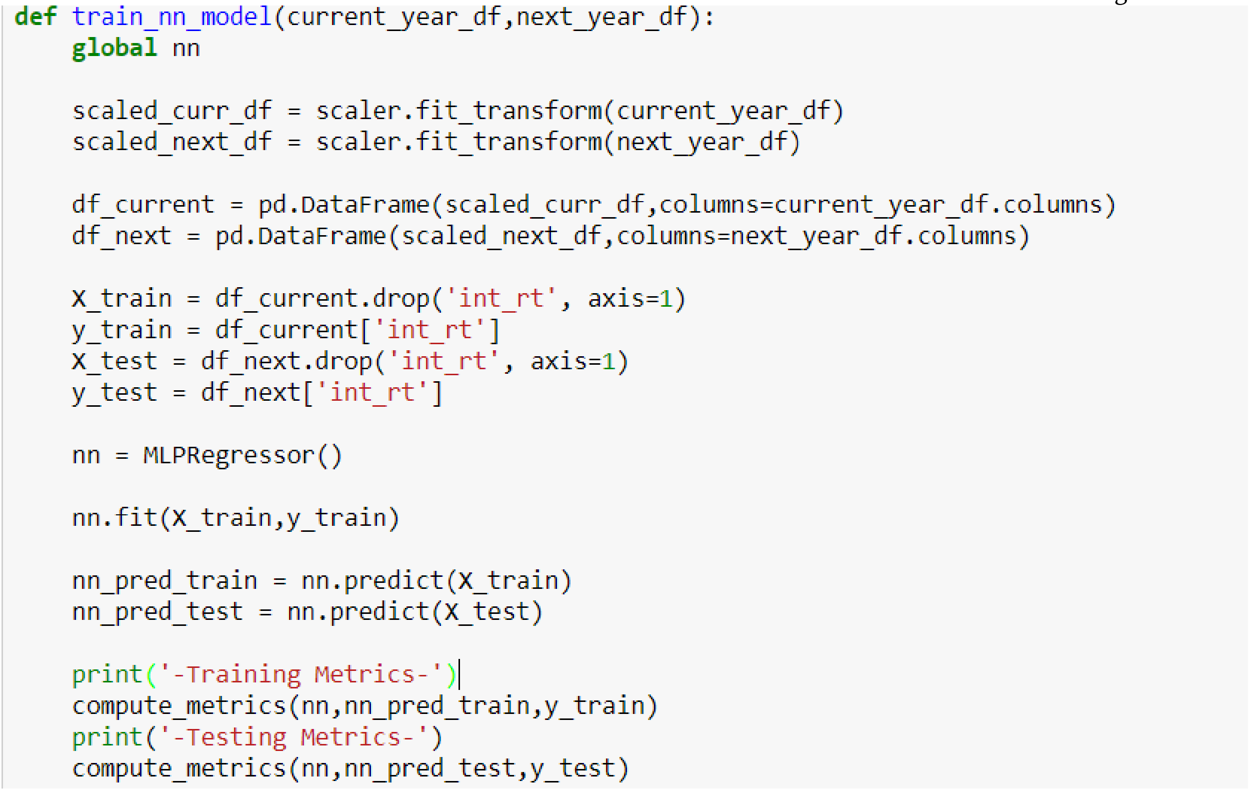


```
In [13]: def train_rf_model(current_year_df,next_year_df):
             global rf

             scaled_curr_df = scaler.fit_transform(current_year_df)
             scaled_next_df = scaler.fit_transform(next_year_df)

             df_current = pd.DataFrame(scaled_curr_df,columns=current_year_df.columns)
             df_next = pd.DataFrame(scaled_next_df,columns=next_year_df.columns)

             X_train = df_current.drop('int_rt', axis=1)
             y_train = df_current['int_rt']
             X_test = df_next.drop('int_rt', axis=1)
             y_test = df_next['int_rt']

             rf = RandomForestRegressor()

             rf.fit(X_train,y_train)

             rf_pred_train = rf.predict(X_train)
             rf_pred_test = rf.predict(X_test)

             print('-Training Metrics-')
             compute_metrics(rf,rf_pred_train,y_train)
             print('-Testing Metrics-')
             compute_metrics(rf,rf_pred_test,y_test)
```

2.1.4.4 NEURAL NETWORK

Neural network terminology is inspired by the biological operations of specialized cells called neurons. A neuron is a cell that has several inputs that can be activated by some outside process.

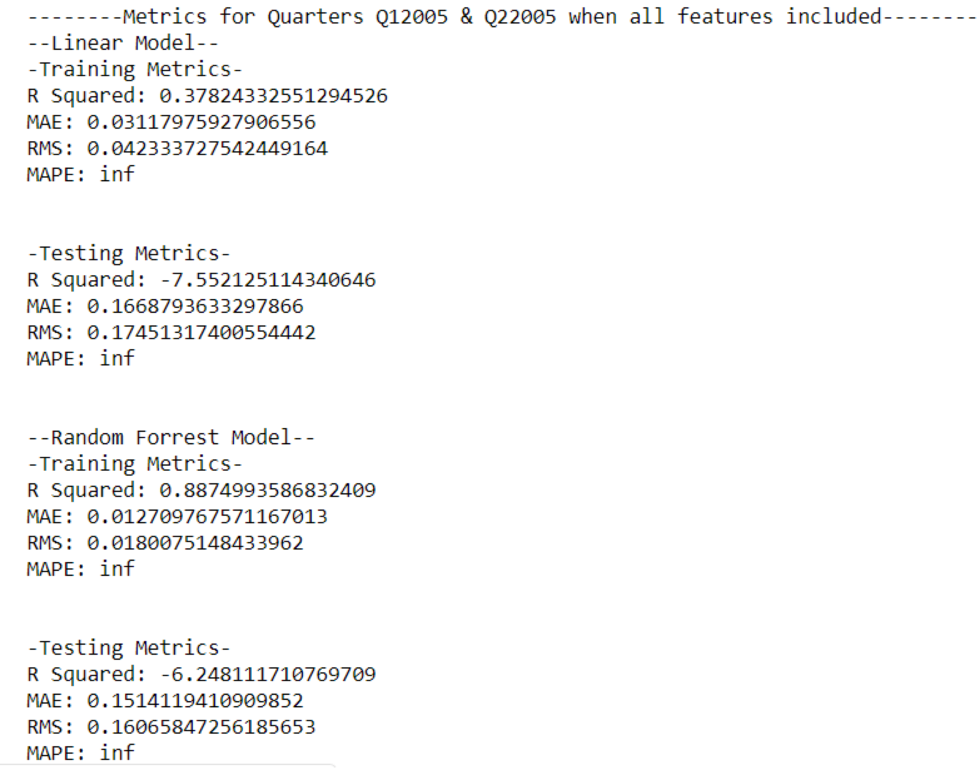
The artificial equivalent of a neuron is a node (also sometimes called neurons, but I will refer to them as nodes to avoid ambiguity) that receives a set of weighted inputs, processes their sum with its activation function, and passes the result of the activation function to nodes further down the graph.



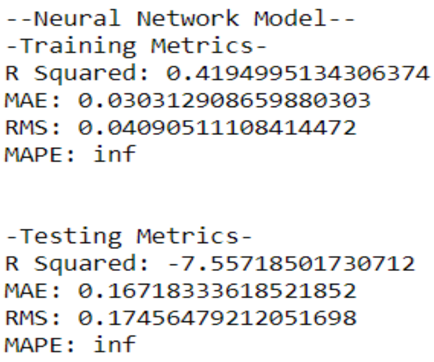
```
def train_nn_model(current_year_df,next_year_df):
    global nn

    scaled_curr_df = scaler.fit_transform(current_year_df)
    scaled_next_df = scaler.fit_transform(next_year_df)

    df_current = pd.DataFrame(scaled_curr_df,columns=current_year_df.columns)
    df_next = pd.DataFrame(scaled_next_df,columns=next_year_df.columns)

    X_train = df_current.drop('int_rt', axis=1)
    y_train = df_current['int_rt']
    X_test = df_next.drop('int_rt', axis=1)
    y_test = df_next['int_rt']

    nn = MLPRegressor()

    nn.fit(X_train,y_train)

    nn_pred_train = nn.predict(X_train)
    nn_pred_test = nn.predict(X_test)

    print('-Training Metrics-')
    compute_metrics(nn,nn_pred_train,y_train)
    print('-Testing Metrics-')
    compute_metrics(nn,nn_pred_test,y_test)
```

2.1.4.5 STORING AND RETURNING THE RESULTS

We are calculating the results from the different machine learning algorithms, where we are capturing following details:
ModelName, RMSE.baseline, MAE.baseline and Median Absolute error.



```
--------Metrics for Quarters Q12005 & Q22005 when all features included--------
--Linear Model--
-Training Metrics-
R Squared: 0.37824332551294526
MAE: 0.03117975927906556
RMS: 0.042333727542449164
MAPE: inf


-Testing Metrics-
R Squared: -7.552125114340646
MAE: 0.1668793633297866
RMS: 0.17451317400554442
MAPE: inf


--Random Forrest Model--
-Training Metrics-
R Squared: 0.8874993586832409
MAE: 0.012709767571167013
RMS: 0.0180075148433962
MAPE: inf


-Testing Metrics-
R Squared: -6.248111710769709
MAE: 0.1514119410909852
RMS: 0.16065847256185653
MAPE: inf
```

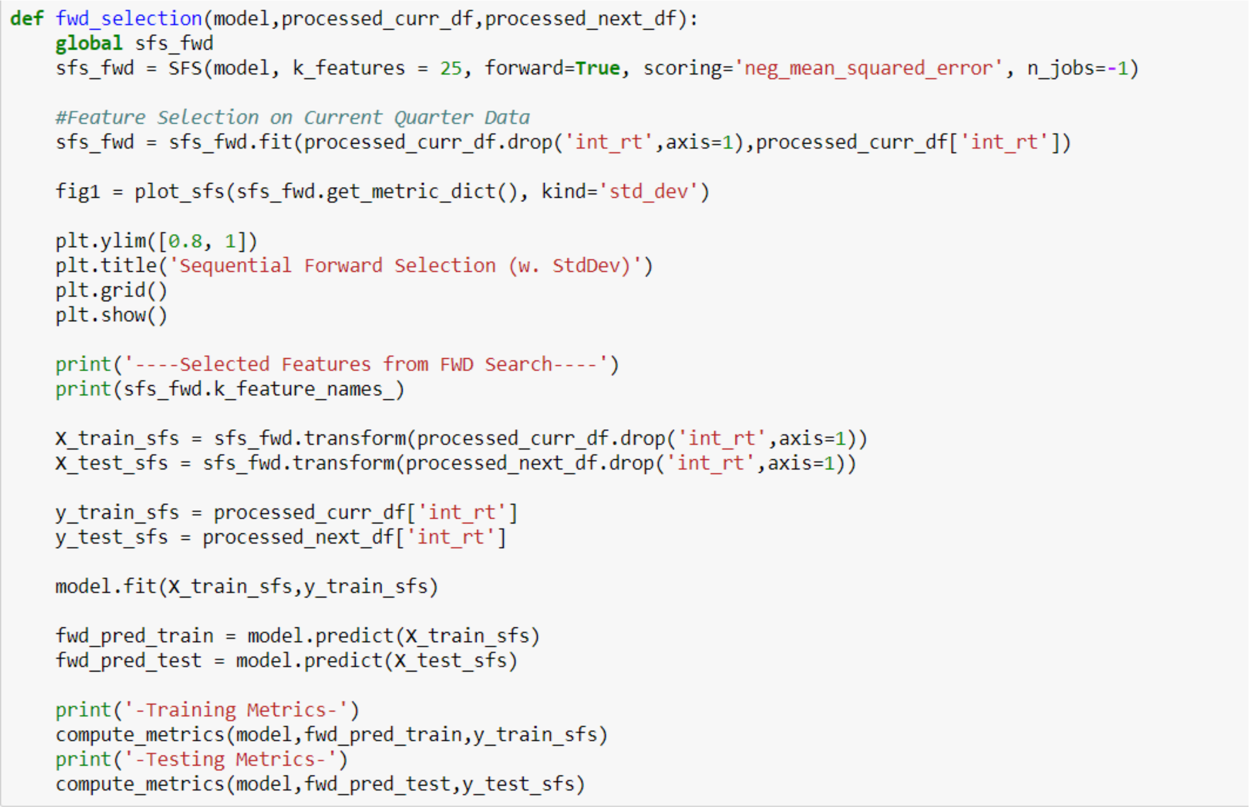


```
--Neural Network Model--
-Training Metrics-
R Squared: 0.4194995134306374
MAE: 0.030312908659880303
RMS: 0.04090511108414472
MAPE: inf


-Testing Metrics-
R Squared: -7.55718501730712
MAE: 0.16718333618521852
RMS: 0.17456479212051698
MAPE: inf
```

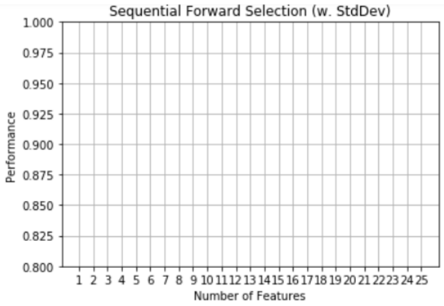
Sequvential Feature Selection

\Sequential feature selection algorithms are a family of greedy search algorithms that are used to reduce an initial *d*-dimensional feature space to a *k*-dimensional feature subspace where *k* < *d*. The motivation behind feature selection algorithms is to automatically select a subset of features that is most relevant to the problem. The goal of feature selection is two-fold: We want to improve the computational efficiency and reduce the generalization error of the model by removing irrelevant features or noise. A wrapper approach such as sequential feature selection is especially useful if embedded feature selection -- for example, a regularization penalty like LASSO -- is not applicable.

Forward Selection

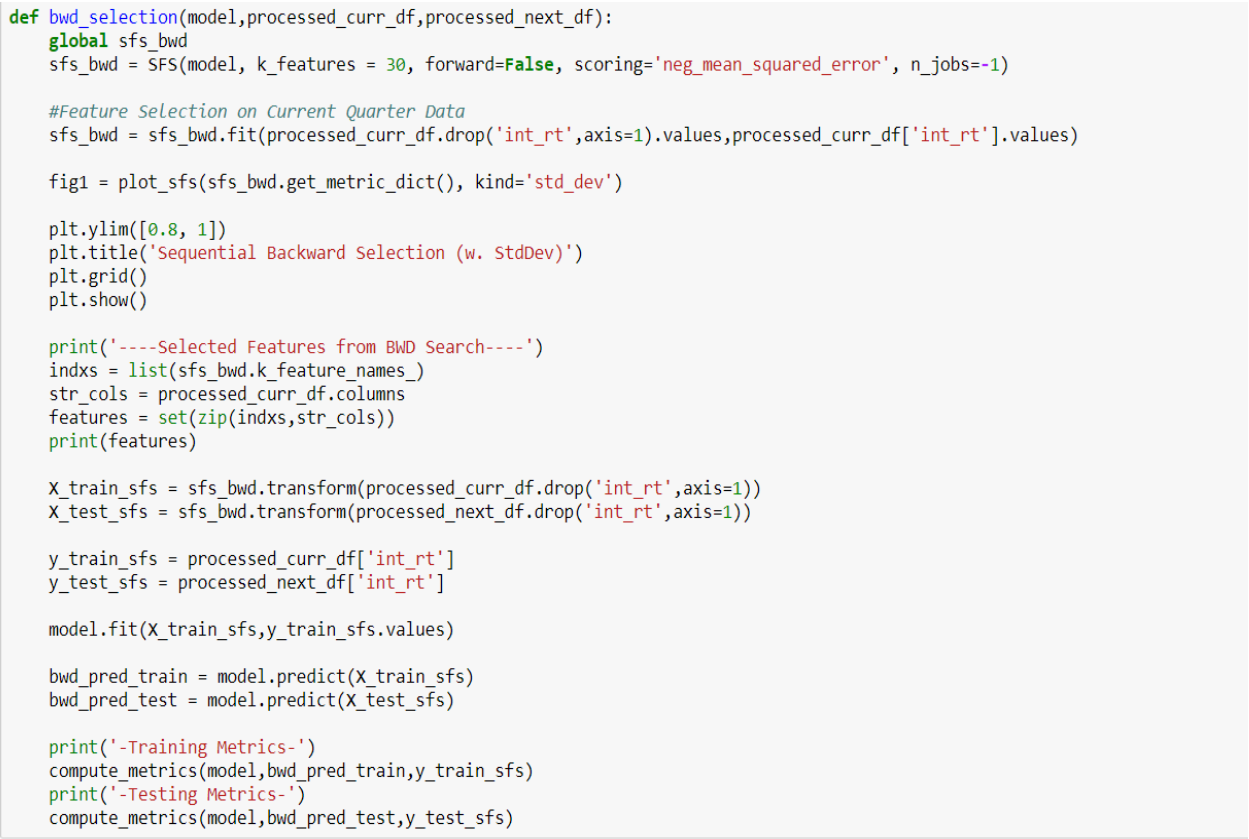


```python
def fwd_selection(model,processed_curr_df,processed_next_df):
    global sfs_fwd
    sfs_fwd = SFS(model, k_features = 25, forward=True, scoring='neg_mean_squared_error', n_jobs=-1)

    #Feature Selection on Current Quarter Data
    sfs_fwd = sfs_fwd.fit(processed_curr_df.drop('int_rt',axis=1),processed_curr_df['int_rt'])

    fig1 = plot_sfs(sfs_fwd.get_metric_dict(), kind='std_dev')

    plt.ylim([0.8, 1])
    plt.title('Sequential Forward Selection (w. StdDev)')
    plt.grid()
    plt.show()

    print('----Selected Features from FWD Search----')
    print(sfs_fwd.k_feature_names_)

    X_train_sfs = sfs_fwd.transform(processed_curr_df.drop('int_rt',axis=1))
    X_test_sfs = sfs_fwd.transform(processed_next_df.drop('int_rt',axis=1))

    y_train_sfs = processed_curr_df['int_rt']
    y_test_sfs = processed_next_df['int_rt']

    model.fit(X_train_sfs,y_train_sfs)

    fwd_pred_train = model.predict(X_train_sfs)
    fwd_pred_test = model.predict(X_test_sfs)

    print('-Training Metrics-')
    compute_metrics(model,fwd_pred_train,y_train_sfs)
    print('-Testing Metrics-')
    compute_metrics(model,fwd_pred_test,y_test_sfs)
```

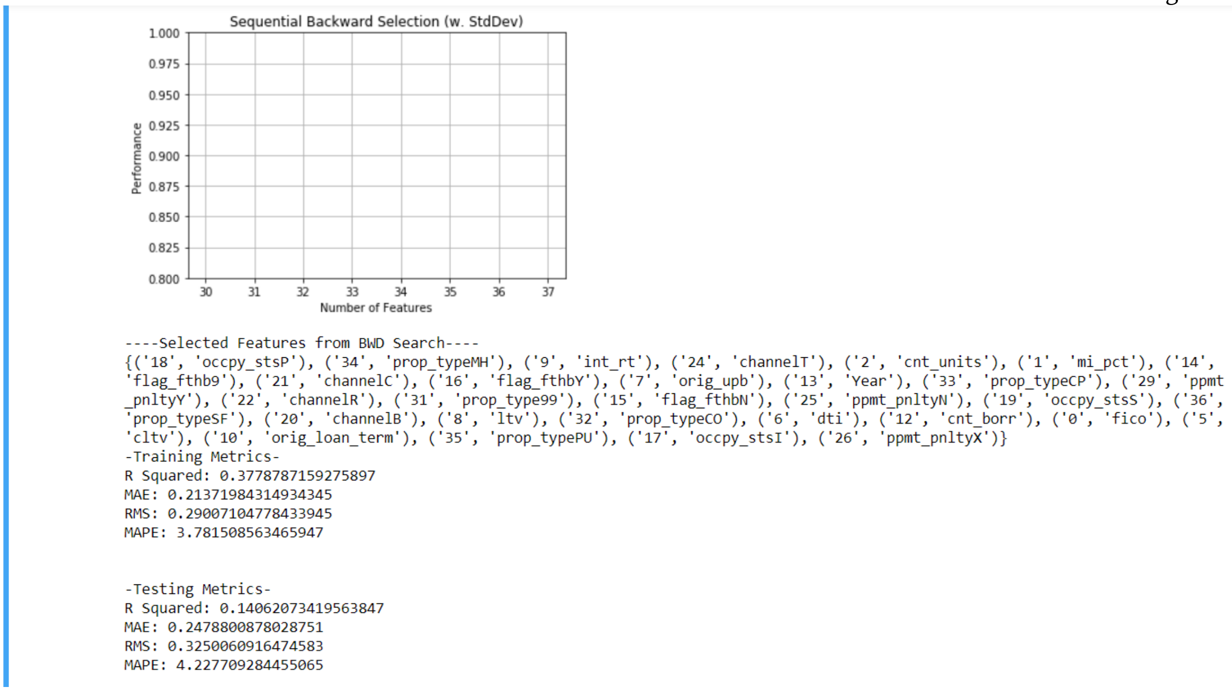


```
----Selected Features from FWD Search----
('fico', 'mi_pct', 'cnt_units', 'orig_upb', 'ltv', 'orig_loan_term', 'cnt_borr', 'Year', 'flag_fthbN', 'flag_fthbY', 'occpy_sts
I', 'occpy_stsP', 'occpy_stsS', 'channelT', 'ppmt_pnltyN', 'prop_typeCO', 'prop_typeCP', 'prop_typeMH', 'loan_purposeC', 'loan_
purposeN', 'loan_purposeP', 'flag_fthbX', 'occpy_stsl', 'occpy_stsO', 'prop_typeLH')
-Training Metrics-
R Squared: 0.37578469953954297
MAE: 0.2137363480849155
RMS: 0.2905588170295989
MAPE: 3.781493586814121


-Testing Metrics-
R Squared: 0.1387088493410591
MAE: 0.2477951753046037
RMS: 0.3253674158239672
MAPE: 4.22543964475919
```

Backward Selection



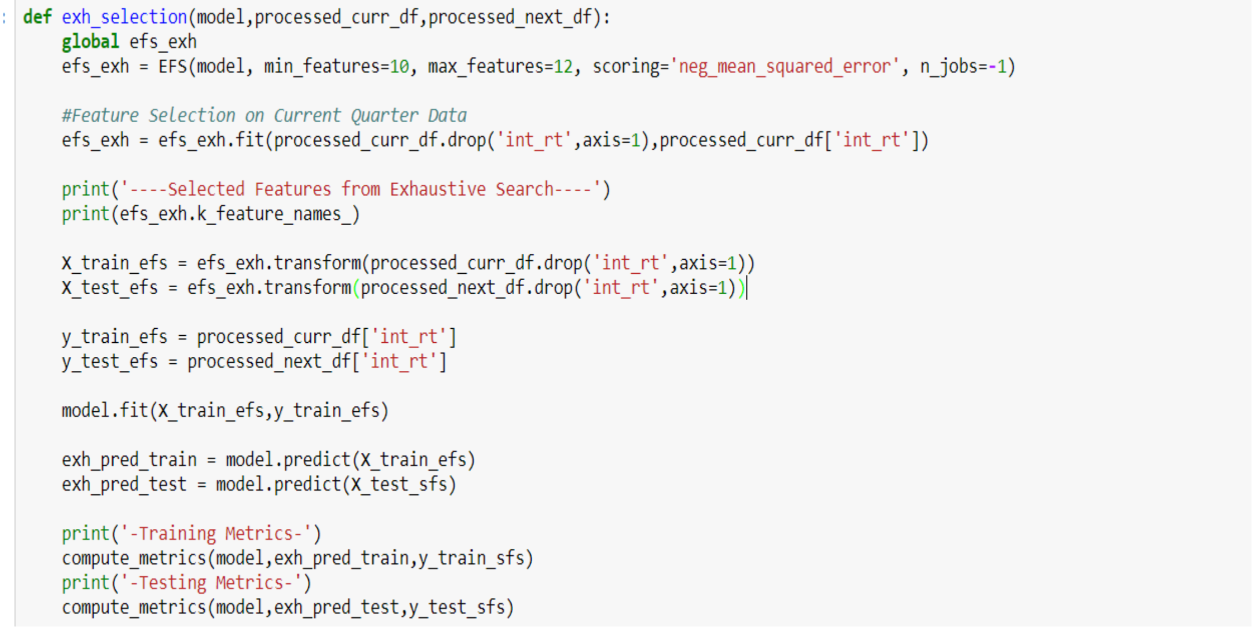
```python
def bwd_selection(model,processed_curr_df,processed_next_df):
    global sfs_bwd
    sfs_bwd = SFS(model, k_features = 30, forward=False, scoring='neg_mean_squared_error', n_jobs=-1)

    #Feature Selection on Current Quarter Data
    sfs_bwd = sfs_bwd.fit(processed_curr_df.drop('int_rt',axis=1).values,processed_curr_df['int_rt'].values)

    fig1 = plot_sfs(sfs_bwd.get_metric_dict(), kind='std_dev')

    plt.ylim([0.8, 1])
    plt.title('Sequential Backward Selection (w. StdDev)')
    plt.grid()
    plt.show()

    print('----Selected Features from BWD Search----')
    indxs = list(sfs_bwd.k_feature_names_)
    str_cols = processed_curr_df.columns
    features = set(zip(indxs,str_cols))
    print(features)

    X_train_sfs = sfs_bwd.transform(processed_curr_df.drop('int_rt',axis=1))
    X_test_sfs = sfs_bwd.transform(processed_next_df.drop('int_rt',axis=1))

    y_train_sfs = processed_curr_df['int_rt']
    y_test_sfs = processed_next_df['int_rt']

    model.fit(X_train_sfs,y_train_sfs.values)

    bwd_pred_train = model.predict(X_train_sfs)
    bwd_pred_test = model.predict(X_test_sfs)

    print('-Training Metrics-')
    compute_metrics(model,bwd_pred_train,y_train_sfs)
    print('-Testing Metrics-')
    compute_metrics(model,bwd_pred_test,y_test_sfs)
```



```
----Selected Features from BWD Search----
{('18', 'occpy_stsP'), ('34', 'prop_typeMH'), ('9', 'int_rt'), ('24', 'channelT'), ('2', 'cnt_units'), ('1', 'mi_pct'), ('14',
'flag_fthb9'), ('21', 'channelC'), ('16', 'flag_fthbY'), ('7', 'orig_upb'), ('13', 'Year'), ('33', 'prop_typeCP'), ('29', 'ppmt
_pnltyY'), ('22', 'channelR'), ('31', 'prop_type99'), ('15', 'flag_fthbN'), ('25', 'ppmt_pnltyN'), ('19', 'occpy_stsS'), ('36',
'prop_typeSF'), ('20', 'channelB'), ('8', 'ltv'), ('32', 'prop_typeCO'), ('6', 'dti'), ('12', 'cnt_borr'), ('0', 'fico'), ('5',
'cltv'), ('10', 'orig_loan_term'), ('35', 'prop_typePU'), ('17', 'occpy_stsI'), ('26', 'ppmt_pnltyX')}
-Training Metrics-
R Squared: 0.3778787159275897
MAE: 0.21371984314934345
RMS: 0.29007104778433945
MAPE: 3.781508563465947


-Testing Metrics-
R Squared: 0.14062073419563847
MAE: 0.2478800878028751
RMS: 0.3250060916474583
MAPE: 4.227709284455065
```

Exhaustive Feature Selection

This exhaustive feature selection algorithm is a wrapper approach for brute-force evaluation of feature subsets; the best subset is selected by optimizing a specified performance metric given an arbitrary regressor or classifier. For instance, if the classifier is a logistic regression and the dataset consists of 4 features, the alogorithm will evaluate all 15 feature combinations (if min_features=1 and max_features=4)

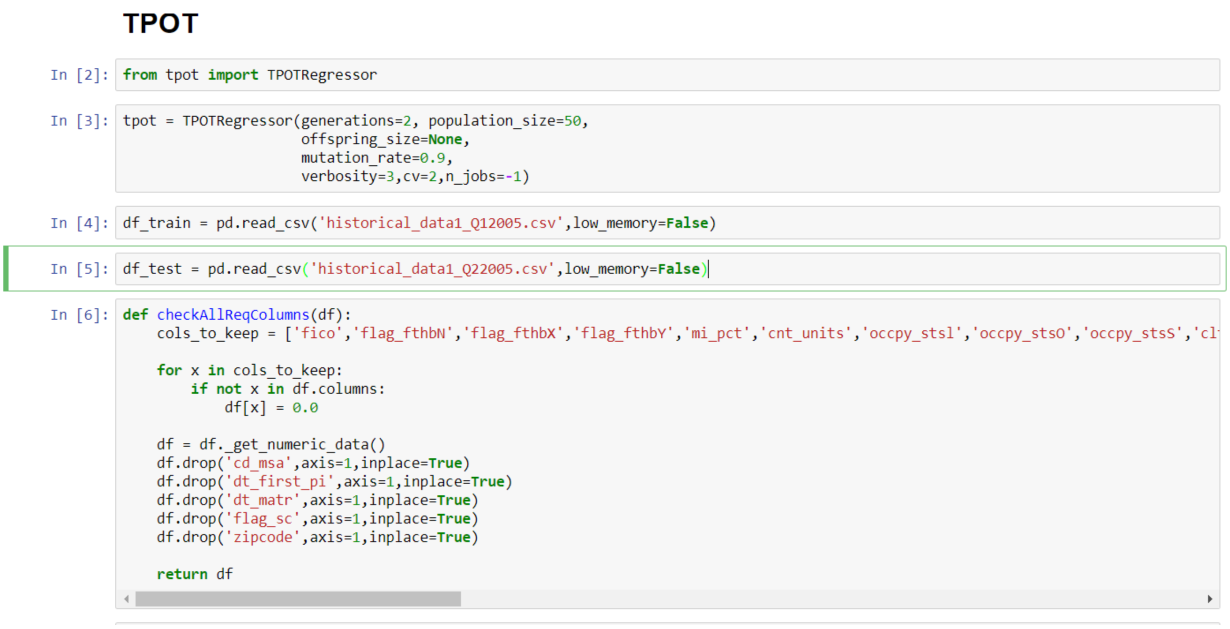


```
def exh_selection(model,processed_curr_df,processed_next_df):
    global efs_exh
    efs_exh = EFS(model, min_features=10, max_features=12, scoring='neg_mean_squared_error', n_jobs=-1)

    #Feature Selection on Current Quarter Data
    efs_exh = efs_exh.fit(processed_curr_df.drop('int_rt',axis=1),processed_curr_df['int_rt'])

    print('----Selected Features from Exhaustive Search----')
    print(efs_exh.k_feature_names_)

    X_train_efs = efs_exh.transform(processed_curr_df.drop('int_rt',axis=1))
    X_test_efs = efs_exh.transform(processed_next_df.drop('int_rt',axis=1))

    y_train_efs = processed_curr_df['int_rt']
    y_test_efs = processed_next_df['int_rt']

    model.fit(X_train_efs,y_train_efs)

    exh_pred_train = model.predict(X_train_efs)
    exh_pred_test = model.predict(X_test_sfs)

    print('-Training Metrics-')
    compute_metrics(model,exh_pred_train,y_train_sfs)
    print('-Testing Metrics-')
    compute_metrics(model,exh_pred_test,y_test_sfs)
```

Try TPOT, H20.Ai and AutoSKLearn Automl algorithms

TPOT

The Tree-Based Pipeline Optimization Tool (TPOT) was one of the very first AutoML methods and open-source software packages developed for the data science community. TPOT was developed by Dr. Randal Olson while a postdoctoral student with Dr. Jason H. Moore at the Computational Genetics Laboratory of the University of Pennsylvania and is still being extended and supported by this team. The goal of TPOT is to automate the building of ML pipelines by combining a flexible expression tree representation of pipelines with stochastic search algorithms such as genetic programming. TPOT makes use of the Python-based scikit-learn library as its ML menu.



TPOT

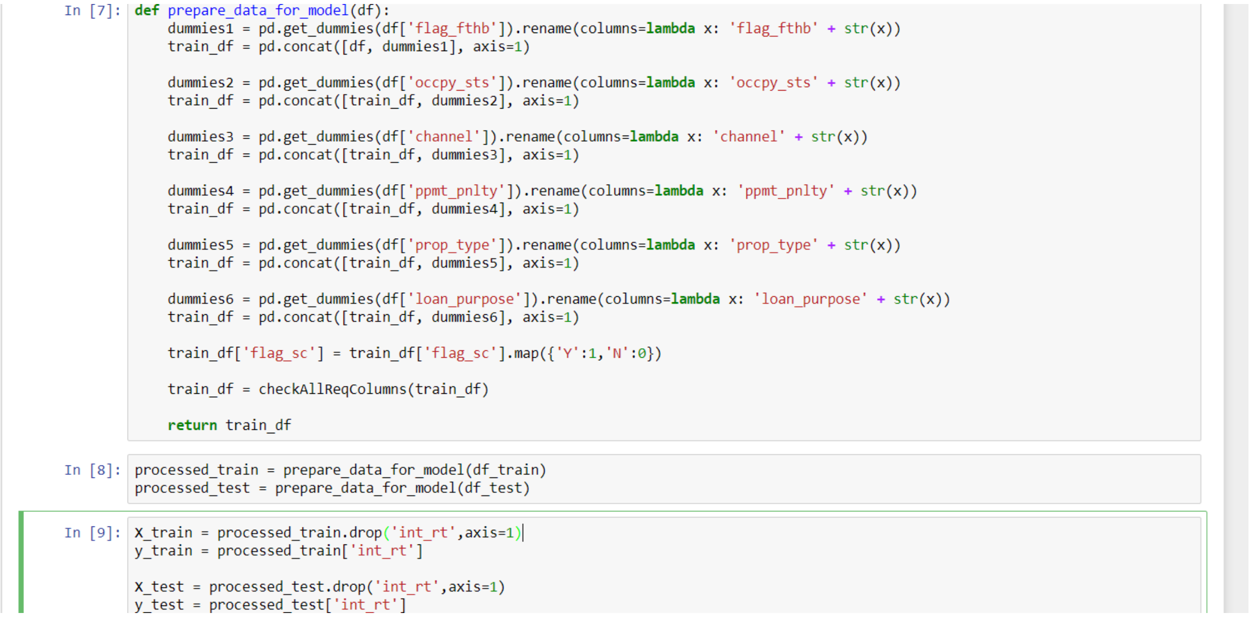
```
In [2]: from tpot import TPOTRegressor
```

```
In [3]: tpot = TPOTRegressor(generations=2, population_size=50,
                             offspring_size=None,
                             mutation_rate=0.9,
                             verbosity=3,cv=2,n_jobs=-1)
```

```
In [4]: df_train = pd.read_csv('historical_data1_Q12005.csv',low_memory=False)
```

```
In [5]: df_test = pd.read_csv('historical_data1_Q22005.csv',low_memory=False)
```

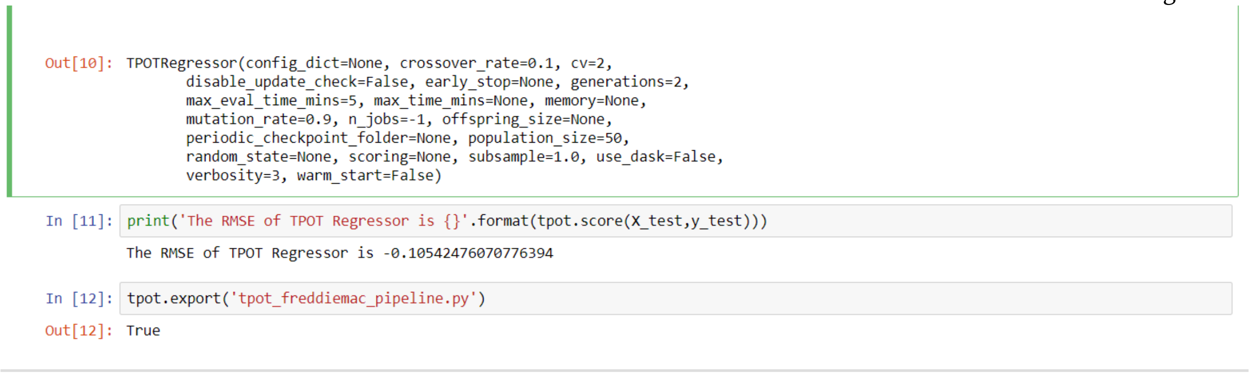
```
In [6]: def checkAllReqColumns(df):
            cols_to_keep = ['fico','flag_fthbN','flag_fthbX','flag_fthbY','mi_pct','cnt_units','occpy_stsl','occpy_stsO','occpy_stsS','cl

            for x in cols_to_keep:
                if not x in df.columns:
                    df[x] = 0.0

            df = df._get_numeric_data()
            df.drop('cd_msa',axis=1,inplace=True)
            df.drop('dt_first_pi',axis=1,inplace=True)
            df.drop('dt_matr',axis=1,inplace=True)
            df.drop('flag_sc',axis=1,inplace=True)
            df.drop('zipcode',axis=1,inplace=True)

            return df
```



```
In [7]: def prepare_data_for_model(df):
            dummies1 = pd.get_dummies(df['flag_fthb']).rename(columns=lambda x: 'flag_fthb' + str(x))
            train_df = pd.concat([df, dummies1], axis=1)

            dummies2 = pd.get_dummies(df['occpy_sts']).rename(columns=lambda x: 'occpy_sts' + str(x))
            train_df = pd.concat([train_df, dummies2], axis=1)

            dummies3 = pd.get_dummies(df['channel']).rename(columns=lambda x: 'channel' + str(x))
            train_df = pd.concat([train_df, dummies3], axis=1)

            dummies4 = pd.get_dummies(df['ppmt_pnlty']).rename(columns=lambda x: 'ppmt_pnlty' + str(x))
            train_df = pd.concat([train_df, dummies4], axis=1)

            dummies5 = pd.get_dummies(df['prop_type']).rename(columns=lambda x: 'prop_type' + str(x))
            train_df = pd.concat([train_df, dummies5], axis=1)

            dummies6 = pd.get_dummies(df['loan_purpose']).rename(columns=lambda x: 'loan_purpose' + str(x))
            train_df = pd.concat([train_df, dummies6], axis=1)

            train_df['flag_sc'] = train_df['flag_sc'].map({'Y':1,'N':0})

            train_df = checkAllReqColumns(train_df)

            return train_df
```

```
In [8]: processed_train = prepare_data_for_model(df_train)
        processed_test = prepare_data_for_model(df_test)
```

```
In [9]: X_train = processed_train.drop('int_rt',axis=1)
        y_train = processed_train['int_rt']

        X_test = processed_test.drop('int_rt',axis=1)
        y_test = processed_test['int_rt']
```

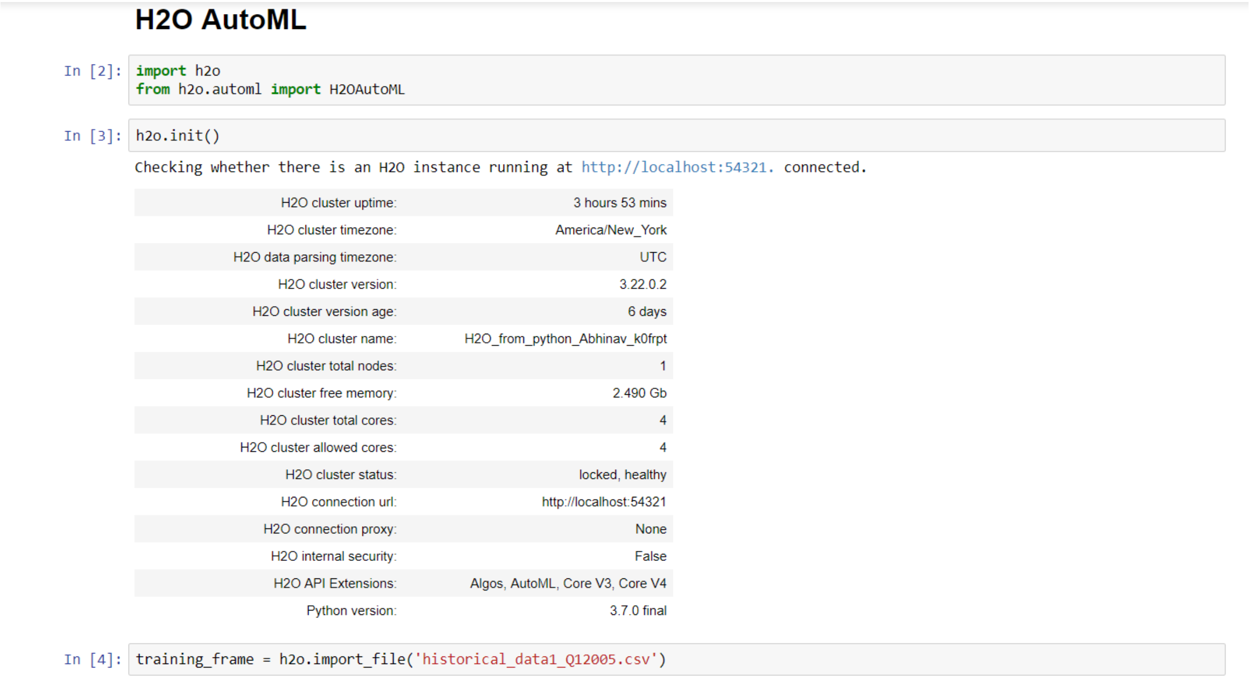


```
Out[10]: TPOTRegressor(config_dict=None, crossover_rate=0.1, cv=2,
                 disable_update_check=False, early_stop=None, generations=2,
                 max_eval_time_mins=5, max_time_mins=None, memory=None,
                 mutation_rate=0.9, n_jobs=-1, offspring_size=None,
                 periodic_checkpoint_folder=None, population_size=50,
                 random_state=None, scoring=None, subsample=1.0, use_dask=False,
                 verbosity=3, warm_start=False)

In [11]: print('The RMSE of TPOT Regressor is {}'.format(tpot.score(X_test,y_test)))

         The RMSE of TPOT Regressor is -0.10542476070776394

In [12]: tpot.export('tpot_freddiemac_pipeline.py')

Out[12]: True
```

H2O AutoML

AutoML is a function in H2O that automates the process of building a large number of models, with the goal of finding the "best" model without any prior knowledge or effort by the Data Scientist.

The current version of AutoML (in H2O 3.16.*) trains and cross-validates a default Random Forest, an Extremely-Randomized Forest, a random grid of Gradient Boosting Machines (GBMs), a random grid of Deep Neural Nets, a fixed grid of GLMs, and then trains two Stacked Ensemble models at the end. One ensemble contains all the models (optimized for model performance), and the second ensemble contains just the best performing model from each algorithm class/family (optimized for production use).

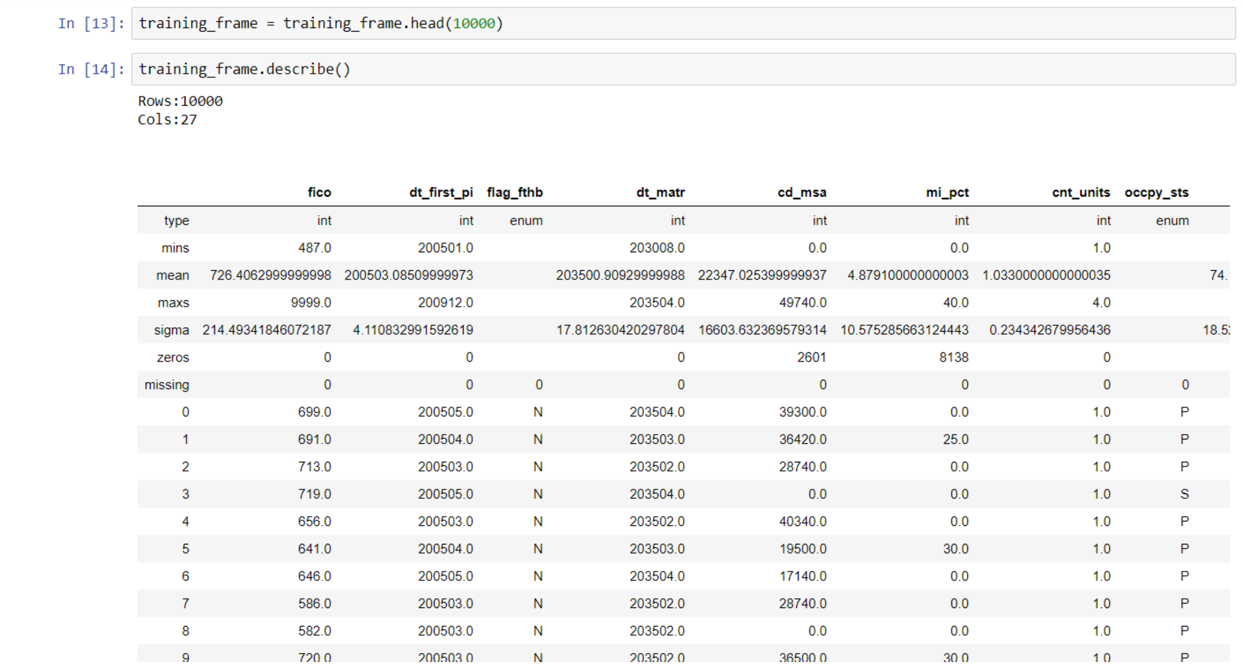


H2O AutoML

```
In [2]: import h2o
        from h2o.automl import H2OAutoML

In [3]: h2o.init()

        Checking whether there is an H2O instance running at http://localhost:54321. connected.
```

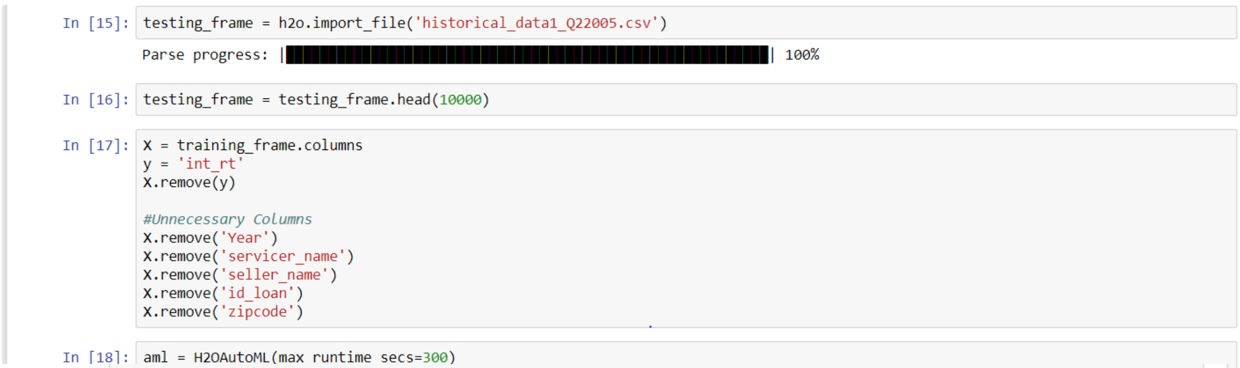
H2O cluster uptime:	3 hours 53 mins
H2O cluster timezone:	America/New_York
H2O data parsing timezone:	UTC
H2O cluster version:	3.22.0.2
H2O cluster version age:	6 days
H2O cluster name:	H2O_from_python_Abhinav_k0frpt
H2O cluster total nodes:	1
H2O cluster free memory:	2.490 Gb
H2O cluster total cores:	4
H2O cluster allowed cores:	4
H2O cluster status:	locked, healthy
H2O connection url:	http://localhost:54321
H2O connection proxy:	None
H2O internal security:	False
H2O API Extensions:	Algos, AutoML, Core V3, Core V4
Python version:	3.7.0 final

```
In [4]: training_frame = h2o.import_file('historical_data1_Q12005.csv')
```

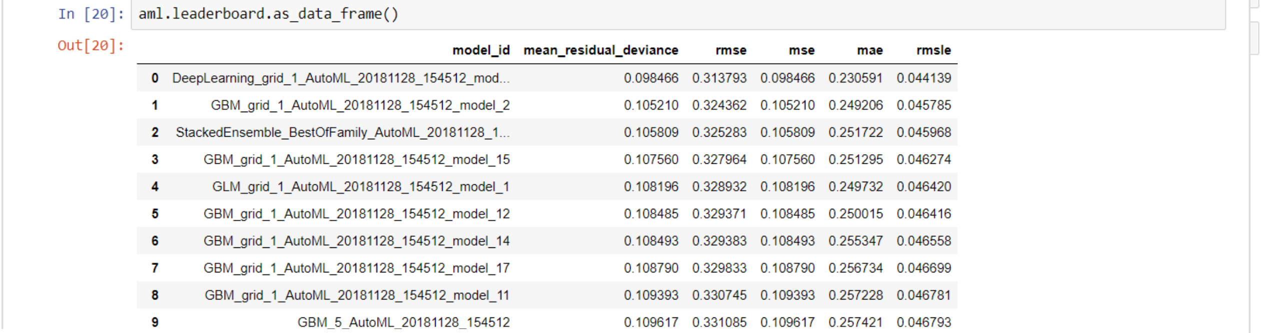


```
In [13]: training_frame = training_frame.head(10000)

In [14]: training_frame.describe()

         Rows:10000
         Cols:27
```

	fico	dt_first_pi	flag_fthb	dt_matr	cd_msa	mi_pct	cnt_units	occpy_sts	
type	int	int	enum	int	int	int	int	enum	
mins	487.0	200501.0		203008.0	0.0	0.0	1.0		
mean	726.4062999999998	200503.08509999973		203500.90929999988	22347.025399999937	4.879100000000003	1.0330000000000035		74.
maxs	9999.0	200912.0		203504.0	49740.0	40.0	4.0		
sigma	214.49341846072187	4.110832991592619		17.812630420297804	16603.632369579314	10.575285663124443	0.234342679956436		18.5
zeros	0	0		0	2601	8138	0		
missing	0	0	0	0	0	0	0	0	
0	699.0	200505.0	N	203504.0	39300.0	0.0	1.0	P	
1	691.0	200504.0	N	203503.0	36420.0	25.0	1.0	P	
2	713.0	200503.0	N	203502.0	28740.0	0.0	1.0	P	
3	719.0	200505.0	N	203504.0	0.0	0.0	1.0	S	
4	656.0	200503.0	N	203502.0	40340.0	0.0	1.0	P	
5	641.0	200504.0	N	203503.0	19500.0	30.0	1.0	P	
6	646.0	200505.0	N	203504.0	17140.0	0.0	1.0	P	
7	586.0	200503.0	N	203502.0	28740.0	0.0	1.0	P	
8	582.0	200503.0	N	203502.0	0.0	0.0	1.0	P	
9	720.0	200503.0	N	203502.0	36500.0	30.0	1.0	P	

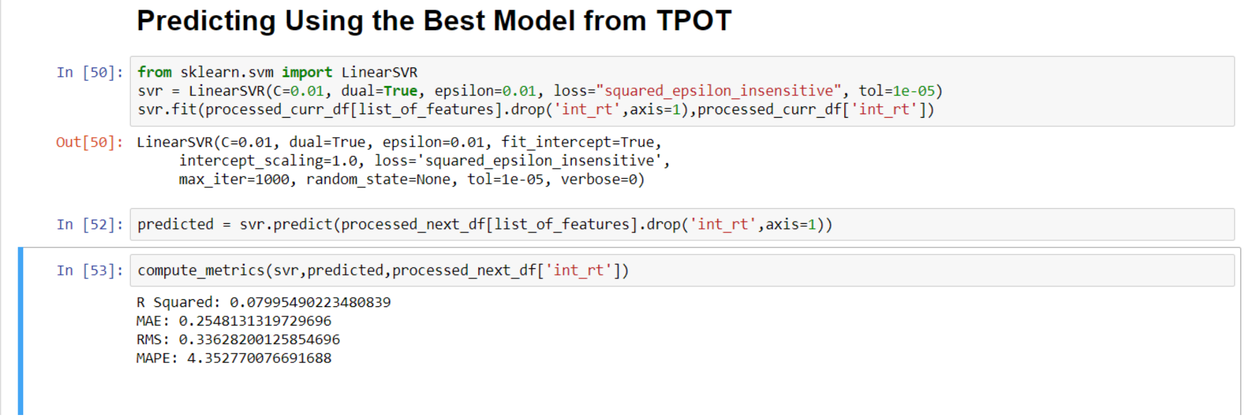


```
In [15]: testing_frame = h2o.import_file('historical_data1_Q22005.csv')

         Parse progress: |██████████████████████████████████████████████████████████| 100%

In [16]: testing_frame = testing_frame.head(10000)

In [17]: X = training_frame.columns
         y = 'int_rt'
         X.remove(y)

         #Unnecessary Columns
         X.remove('Year')
         X.remove('servicer_name')
         X.remove('seller_name')
         X.remove('id_loan')
         X.remove('zipcode')

In [18]: aml = H2OAutoML(max runtime secs=300)
```



```
In [20]: aml.leaderboard.as_data_frame()

Out[20]:
```

	model_id	mean_residual_deviance	rmse	mse	mae	rmsle
0	DeepLearning_grid_1_AutoML_20181128_154512_mod...	0.098466	0.313793	0.098466	0.230591	0.044139
1	GBM_grid_1_AutoML_20181128_154512_model_2	0.105210	0.324362	0.105210	0.249206	0.045785
2	StackedEnsemble_BestOfFamily_AutoML_20181128_1...	0.105809	0.325283	0.105809	0.251722	0.045968
3	GBM_grid_1_AutoML_20181128_154512_model_15	0.107560	0.327964	0.107560	0.251295	0.046274
4	GLM_grid_1_AutoML_20181128_154512_model_1	0.108196	0.328932	0.108196	0.249732	0.046420
5	GBM_grid_1_AutoML_20181128_154512_model_12	0.108485	0.329371	0.108485	0.250015	0.046416
6	GBM_grid_1_AutoML_20181128_154512_model_14	0.108493	0.329383	0.108493	0.255347	0.046558
7	GBM_grid_1_AutoML_20181128_154512_model_17	0.108790	0.329833	0.108790	0.256734	0.046699
8	GBM_grid_1_AutoML_20181128_154512_model_11	0.109393	0.330745	0.109393	0.257228	0.046781
9	GBM_5_AutoML_20181128_154512	0.109617	0.331085	0.109617	0.257421	0.046793



Predicting Using the Best Model from TPOT

```
In [50]: from sklearn.svm import LinearSVR
         svr = LinearSVR(C=0.01, dual=True, epsilon=0.01, loss="squared_epsilon_insensitive", tol=1e-05)
         svr.fit(processed_curr_df[list_of_features].drop('int_rt',axis=1),processed_curr_df['int_rt'])

Out[50]: LinearSVR(C=0.01, dual=True, epsilon=0.01, fit_intercept=True,
              intercept_scaling=1.0, loss='squared_epsilon_insensitive',
              max_iter=1000, random_state=None, tol=1e-05, verbose=0)

In [52]: predicted = svr.predict(processed_next_df[list_of_features].drop('int_rt',axis=1))

In [53]: compute_metrics(svr,predicted,processed_next_df['int_rt'])

         R Squared: 0.07995490223480839
         MAE: 0.2548131319729696
         RMS: 0.33628200125854696
         MAPE: 4.352770076691688
```

2.1.4.6 PREDICTION MODEL EVALUATION: WHICH MODEL TO CHOOSE

We had compared results of all the models decided **Random Forest** gives us much better result.

1. Higher average predictive accuracy
2. Moderate prediction speed
3. Performs well with large number of observations
4. Handles lots of irrelevant features well

WHAT -IF Analysis

Financial Crisis Analysis

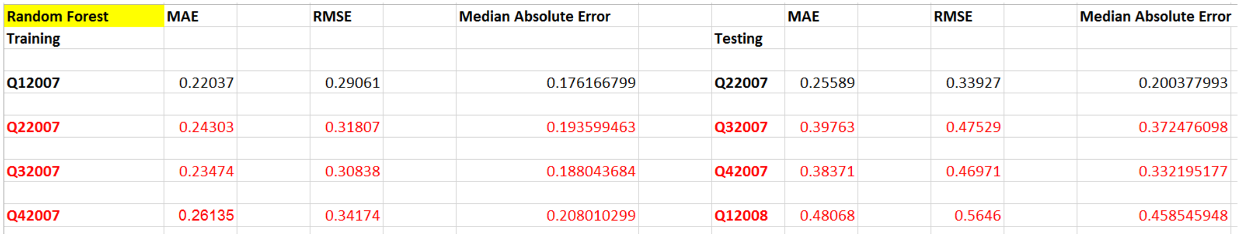
When the housing bubble of 2001-2007 burst, it caused a mortgage security meltdown. This contributed to a general credit crisis, which evolved into a worldwide financial crisis. Many critics have held the United States Congress - and its unwillingness to rein in Fannie Mae and Freddie Mac - responsible for the credit crisis.

In the fall of 2007, Freddie Mac shocked the market by announcing large credit-related loses, fueling the fire for the argument that the two companies pose a tremendous risk to the entire financial system. (http://www.investopedia.com/articles/economics/08/fannie-mae-freddie-mac-credit-crisis.asp)

The Federal Home Loan Mortgage Corporation (Freddie Mac) announced that it will no longer buy the most risky subprime mortgages and mortgage-related securities.

In July 24, 2007 Countrywide Financial Corporation warned of “difficult conditions.” This is evident from the Q32007 Testing measures as the difference between Training and Testing RMSE increased substantially by around 16%.

In November 1, 2007 financial market pressures intensified, reflected in diminished liquidity in interbank funding markets. This is evident in Q42007 Testing measures as the difference between Training and Testing RMSE increased substantially by around 22%.



Random Forest	MAE	RMSE	Median Absolute Error		MAE	RMSE	Median Absolute Error
Training				Testing			
Q12007	0.22037	0.29061	0.176166799	**Q22007**	0.25589	0.33927	0.200377993
Q22007	0.24303	0.31807	0.193599463	**Q32007**	0.39763	0.47529	0.372476098
Q32007	0.23474	0.30838	0.188043684	**Q42007**	0.38371	0.46971	0.332195177
Q42007	0.26135	0.34174	0.208010299	**Q12008**	0.48068	0.5646	0.458545948

Two Years Later (2009):

Financial markets recovered substantially since March 2009 when the financial stress began to ease and market conditions started to improve. (http://www.oecd.org/daf/fin/financial-markets/44563803.pdf)

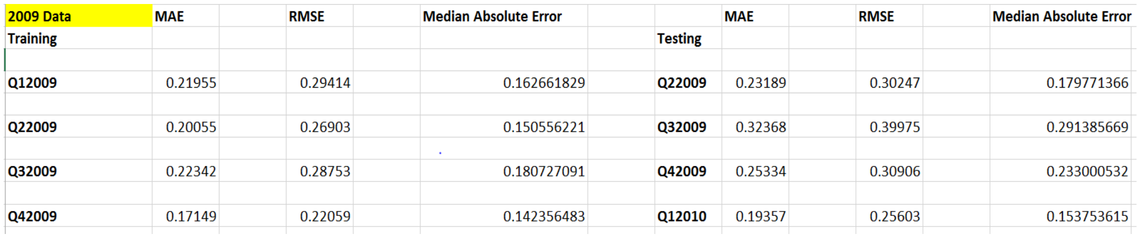
In 2009, Freddie Mac played a critical role in supporting the nation’s housing market by:

- Providing $548.4 billion of liquidity to the mortgage market, helping finance approximately 2.2 million conforming single-family loans and approximately 253,000 units of multifamily rental housing.

- Helping more than 272,000 borrowers stay in their homes or sell their properties through the company's long-standing foreclosure avoidance programs and the Home Affordable Modification program (HAMP), including 129,380 loans that remained in HAMP trial periods as of December 31, 2009 according to information provided by the Making Home Affordable (MHA) program administrator.

- Refinancing approximately $379 billion of single-family loans, creating an estimated $4.5 billion in annual interest savings for borrowers nationwide – this includes approximately 169,000 borrowers whose payments were reduced by an average of $2,000 annually under the Freddie Mac Relief Refinance MortgageSM

(http://www.freddiemac.com/news/archives/investors/2010/2009er-4q09.html)



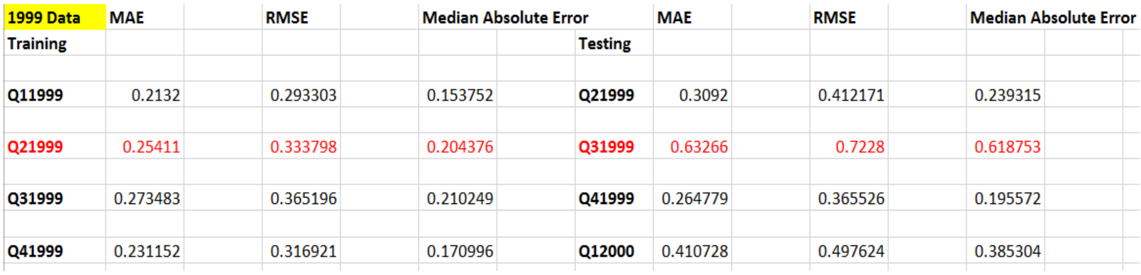
2009 Data	MAE	RMSE	Median Absolute Error		MAE	RMSE	Median Absolute Error
Training				Testing			
Q12009	0.21955	0.29414	0.162661829	Q22009	0.23189	0.30247	0.179771366
Q22009	0.20055	0.26903	0.150556221	Q32009	0.32368	0.39975	0.291385669
Q32009	0.22342	0.28753	0.180727091	Q42009	0.25334	0.30906	0.233000532
Q42009	0.17149	0.22059	0.142356483	Q12010	0.19357	0.25603	0.153753615

As clearly evident from the analysis, Measures are pretty much stable for 2009.

Economic Boom (1999,2013):

1999:

The easing of credit also coincided with spectacular stock market run-ups from 1999 to 2000 Freddie Mac financed homes for more than 2 million families and achieved record earnings per share of $2.96, an increase of 28 percent over 1998.
(http://www.freddiemac.com/investors/pdffiles/annual99.pdf)
(https://en.wikipedia.org/wiki/1990s_United_States_boom)



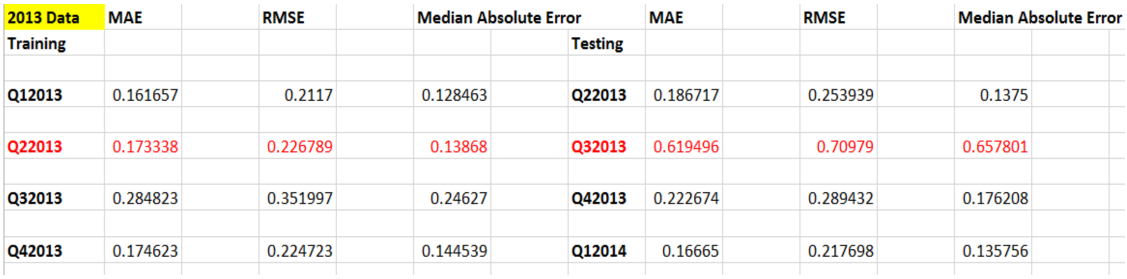
1999 Data	MAE	RMSE	Median Absolute Error		MAE	RMSE	Median Absolute Error
Training				Testing			
Q11999	0.2132	0.293303	0.153752	Q21999	0.3092	0.412171	0.239315
Q21999	0.25411	0.333798	0.204376	Q31999	0.63266	0.7228	0.618753
Q31999	0.273483	0.365196	0.210249	Q41999	0.264779	0.365526	0.195572
Q41999	0.231152	0.316921	0.170996	Q12000	0.410728	0.497624	0.385304

2013:

In 2013, Mortgage rates peaked at 4.6% in August and have held steady since September and several accounting events had significant impacts on the Enterprises' reported financial results. Fannie Mae and Freddie Mac reported levels of 2013 net income are greater than at any prior time in their respective histories. Their historically high net income was driven by reversals of previously accrued losses associated with deferred tax assets (DTA) and their allowance for loan and lease losses (ALLL)—plus revenue from legal settlements of representation and warranties claims and lawsuits regarding private-label securities that the Enterprises purchased as investments. FHFA does not expect benefits of this nature to be repeated in future years and does not expect the 2013 levels of net income to be approached anytime in the foreseeable future. (https://www.fhfa.gov/AboutUs/Reports/ReportDocuments/FHFA_2013_Report_to_Congress.pdf#page=18)

(http://www.foxbusiness.com/features/2013/12/23/housing-market-in-2013-prices-rise-as-lending-remains-tight.html)

Drastic change in Training and Testing measures for the highlighted rows clearly shows the transition in economic trends during Q2 and Q3 around 48%.



2013 Data	MAE	RMSE	Median Absolute Error		MAE	RMSE	Median Absolute Error
Training				Testing			
Q12013	0.161657	0.2117	0.128463	Q22013	0.186717	0.253939	0.1375
Q22013	0.173338	0.226789	0.13868	Q32013	0.619496	0.70979	0.657801
Q32013	0.284823	0.351997	0.24627	Q42013	0.222674	0.289432	0.176208
Q42013	0.174623	0.224723	0.144539	Q12014	0.16665	0.217698	0.135756

Would you recommend using this model for the next quarter? Justify

The proposed model will perform well for the next quarter, if there are not major changes in the data patterns such as financial crisis or economic boom.

CLASSIFICATION (LOAN DELINQUENCY STATUS)

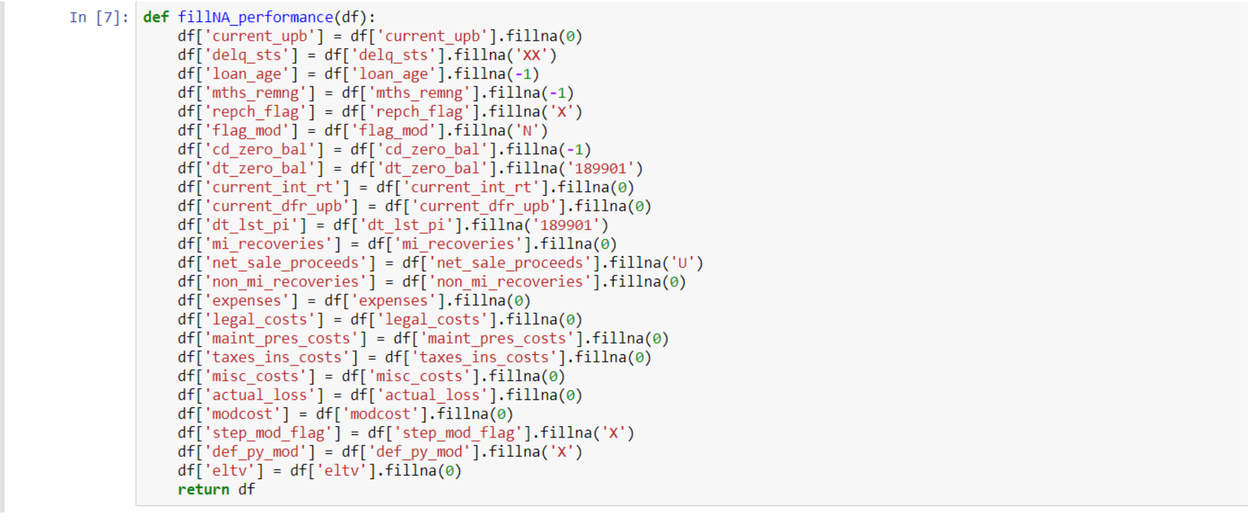
In Loan Performance file, we have a column name delq_sts on which we should predict the Loan Delinquency Status by training the data on the quarter provided and predict the result for the next quarter.

3.3 GENERIC APPROACH: CLASSIFCATION

2.1.1 CREATING THE FUNCTION AND DOWNLOAD THE INPUT

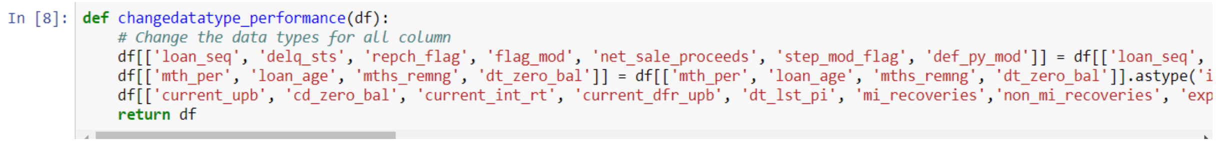
We have created functions that would take hit the Freddie Mac website and download all the Historical file for our purpose. Based on the input Quarter, the file will be imported into python Data Frame. It went through following preprocessing before building model.

Handling the missing values:



```
In [7]: def fillNA_performance(df):
            df['current_upb'] = df['current_upb'].fillna(0)
            df['delq_sts'] = df['delq_sts'].fillna('XX')
            df['loan_age'] = df['loan_age'].fillna(-1)
            df['mths_remng'] = df['mths_remng'].fillna(-1)
            df['repch_flag'] = df['repch_flag'].fillna('X')
            df['flag_mod'] = df['flag_mod'].fillna('N')
            df['cd_zero_bal'] = df['cd_zero_bal'].fillna(-1)
            df['dt_zero_bal'] = df['dt_zero_bal'].fillna('189901')
            df['current_int_rt'] = df['current_int_rt'].fillna(0)
            df['current_dfr_upb'] = df['current_dfr_upb'].fillna(0)
            df['dt_lst_pi'] = df['dt_lst_pi'].fillna('189901')
            df['mi_recoveries'] = df['mi_recoveries'].fillna(0)
            df['net_sale_proceeds'] = df['net_sale_proceeds'].fillna('U')
            df['non_mi_recoveries'] = df['non_mi_recoveries'].fillna(0)
            df['expenses'] = df['expenses'].fillna(0)
            df['legal_costs'] = df['legal_costs'].fillna(0)
            df['maint_pres_costs'] = df['maint_pres_costs'].fillna(0)
            df['taxes_ins_costs'] = df['taxes_ins_costs'].fillna(0)
            df['misc_costs'] = df['misc_costs'].fillna(0)
            df['actual_loss'] = df['actual_loss'].fillna(0)
            df['modcost'] = df['modcost'].fillna(0)
            df['step_mod_flag'] = df['step_mod_flag'].fillna('X')
            df['def_py_mod'] = df['def_py_mod'].fillna('X')
            df['eltv'] = df['eltv'].fillna(0)
            return df
```

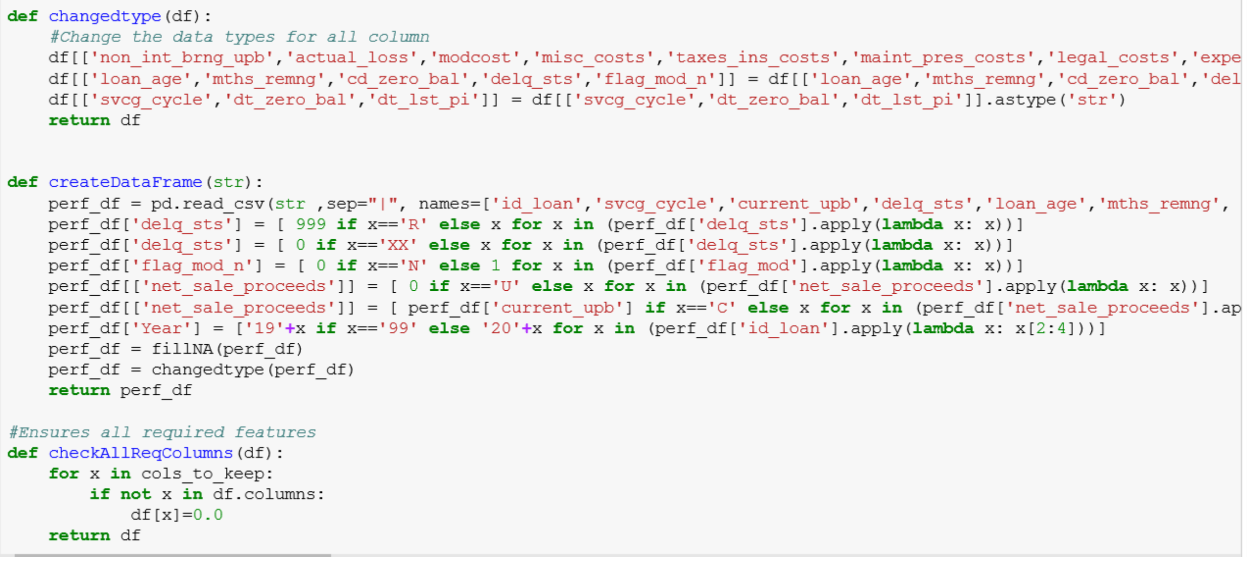
Created dummy columns (1…C):



```
In [8]: def changedatatype_performance(df):
            # Change the data types for all column
            df[['loan_seq', 'delq_sts', 'repch_flag', 'flag_mod', 'net_sale_proceeds', 'step_mod_flag', 'def_py_mod']] = df[['loan_seq',
            df[['mth_per', 'loan_age', 'mths_remng', 'dt_zero_bal']] = df[['mth_per', 'loan_age', 'mths_remng', 'dt_zero_bal']].astype('i
            df[['current_upb', 'cd_zero_bal', 'current_int_rt', 'current_dfr_upb', 'dt_lst_pi', 'mi_recoveries','non_mi_recoveries', 'exp
            return df
```

2.1.2 CONVERSION OF DATA TYPE

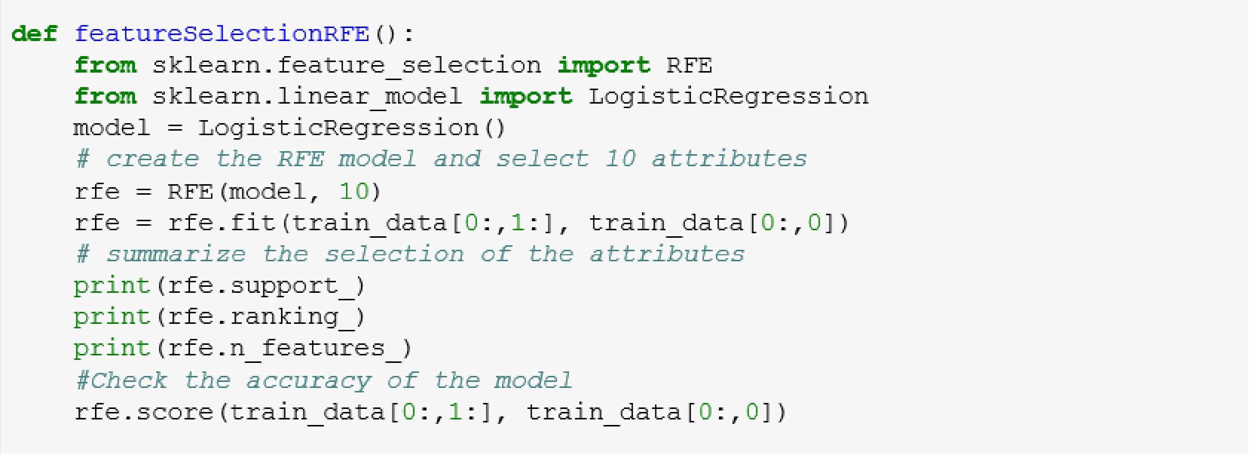
Data types of the History Performance file is converted as below.



```
def changedtype(df):
    #Change the data types for all column
    df[['non_int_brng_upb','actual_loss','modcost','misc_costs','taxes_ins_costs','maint_pres_costs','legal_costs','expe
    df[['loan_age','mths_remng','cd_zero_bal','delq_sts','flag_mod_n']] = df[['loan_age','mths_remng','cd_zero_bal','del
    df[['svcg_cycle','dt_zero_bal','dt_lst_pi']] = df[['svcg_cycle','dt_zero_bal','dt_lst_pi']].astype('str')
    return df


def createDataFrame(str):
    perf_df = pd.read_csv(str ,sep="|", names=['id_loan','svcg_cycle','current_upb','delq_sts','loan_age','mths_remng',
    perf_df['delq_sts'] = [ 999 if x=='R' else x for x in (perf_df['delq_sts'].apply(lambda x: x))]
    perf_df['delq_sts'] = [ 0 if x=='XX' else x for x in (perf_df['delq_sts'].apply(lambda x: x))]
    perf_df['flag_mod_n'] = [ 0 if x=='N' else 1 for x in (perf_df['flag_mod'].apply(lambda x: x))]
    perf_df[['net_sale_proceeds']] = [ 0 if x=='U' else x for x in (perf_df['net_sale_proceeds'].apply(lambda x: x))]
    perf_df[['net_sale_proceeds']] = [ perf_df['current_upb'] if x=='C' else x for x in (perf_df['net_sale_proceeds'].ap
    perf_df['Year'] = ['19'+x if x=='99' else '20'+x for x in (perf_df['id_loan'].apply(lambda x: x[2:4]))]
    perf_df = fillNA(perf_df)
    perf_df = changedtype(perf_df)
    return perf_df

#Ensures all required features
def checkAllReqColumns(df):
    for x in cols_to_keep:
        if not x in df.columns:
            df[x]=0.0
    return df
```

2.1.3 FEATURE SELECTION

Before proceeding with our models, we have done best feature selection using three algorithms. The best features that add to the predictive power of the model and irrelevant features removed from the model. We implemented following feature selection techniques in Python:



```
def featureSelectionRFE():
    from sklearn.feature_selection import RFE
    from sklearn.linear_model import LogisticRegression
    model = LogisticRegression()
    # create the RFE model and select 10 attributes
    rfe = RFE(model, 10)
    rfe = rfe.fit(train_data[0:,1:], train_data[0:,0])
    # summarize the selection of the attributes
    print(rfe.support_)
    print(rfe.ranking_)
    print(rfe.n_features_)
    #Check the accuracy of the model
    rfe.score(train_data[0:,1:], train_data[0:,0])
```

We selected variables as per the **RFE ranking** and used those for further analysis while making sure that all the datasets contain same number of columns. Performing all the feature selection methods we shortlisted below features to best predict or model

We will discuss the algorithm used in the below section for all the Machine Learning algorithm used for classifications

3.3.1 DIFFERENT MACHINE LEARNING ALGORITHMS AND OUTPUT

3.3.1.1 LOGISTIC REGRESSION

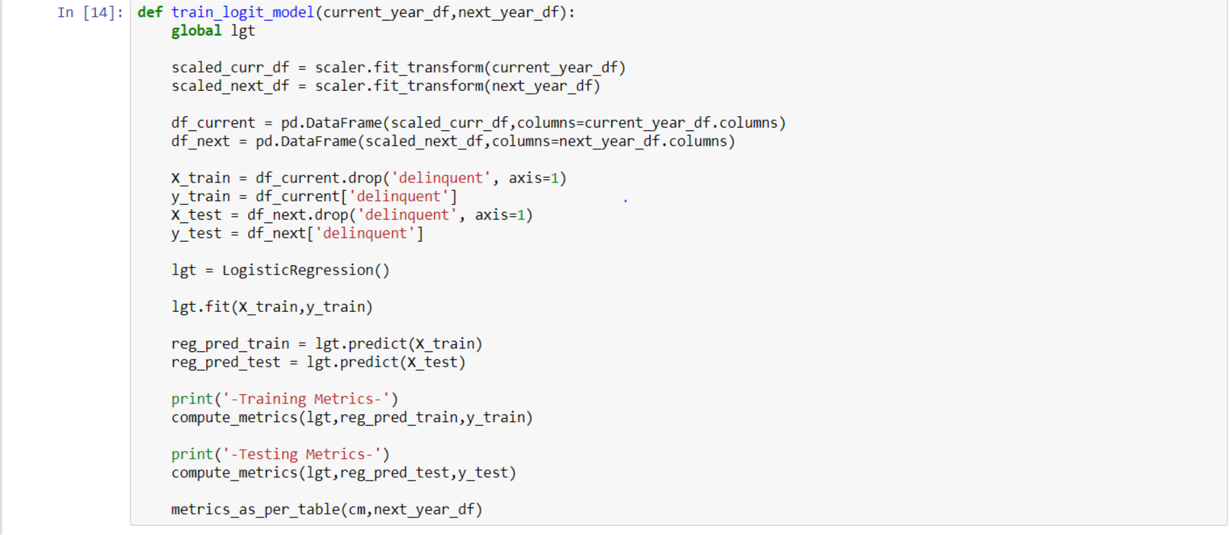
Binary Logistic Regression is a special type of regression where binary response variable is related to a set of explanatory variables, which can be discrete and/or continuous. We are using the logistic regression model for training the model for the quarter supplied and predicting the delinquency status based on the trained model.

Import the following libraries to calculate the logit summary for the logistic regression:

from sklearn.linear_model import LogisticRegression
model = LogisticRegression()

import statsmodels.api as sm
from statsmodels.formula.api import logit, probit, poisson, ols

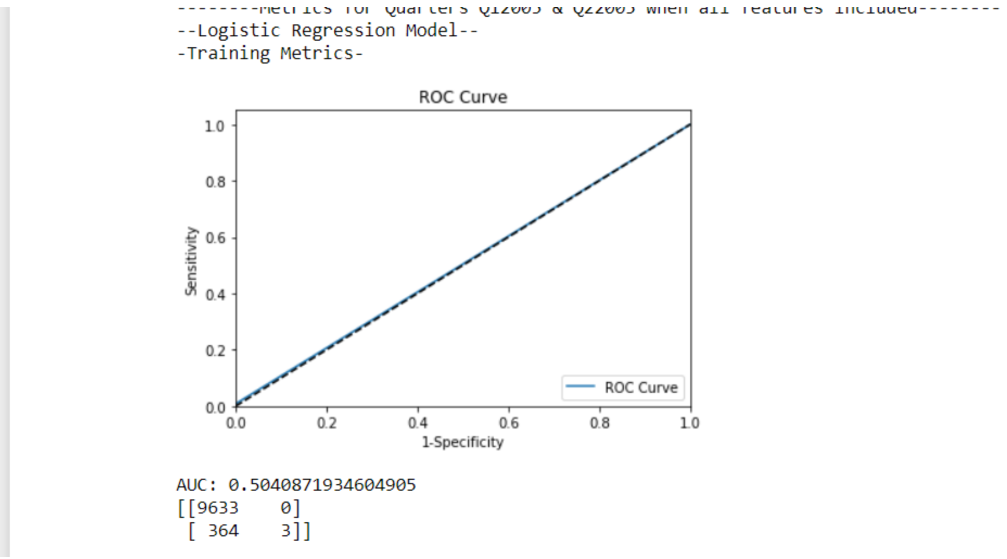
Apply Logit Function:



```
In [14]: def train_logit_model(current_year_df,next_year_df):
             global lgt

             scaled_curr_df = scaler.fit_transform(current_year_df)
             scaled_next_df = scaler.fit_transform(next_year_df)

             df_current = pd.DataFrame(scaled_curr_df,columns=current_year_df.columns)
             df_next = pd.DataFrame(scaled_next_df,columns=next_year_df.columns)

             X_train = df_current.drop('delinquent', axis=1)
             y_train = df_current['delinquent']
             X_test = df_next.drop('delinquent', axis=1)
             y_test = df_next['delinquent']

             lgt = LogisticRegression()

             lgt.fit(X_train,y_train)

             reg_pred_train = lgt.predict(X_train)
             reg_pred_test = lgt.predict(X_test)

             print('-Training Metrics-')
             compute_metrics(lgt,reg_pred_train,y_train)

             print('-Testing Metrics-')
             compute_metrics(lgt,reg_pred_test,y_test)

             metrics_as_per_table(cm,next_year_df)
```

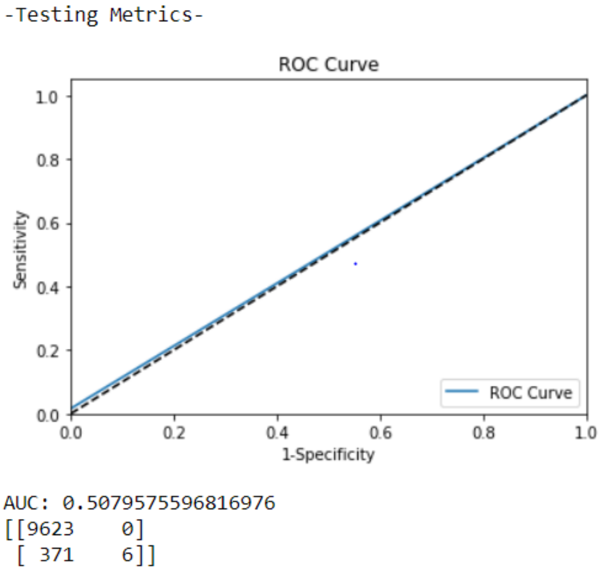
Now predict the delinquency status based on the Test data and generate the accuracy and Classification report.

In this case, our function will result the Confusion matrix as shown below and the ROC curve:



```
--------Metrics for Quarters Q12005 & Q22005 when all features included--------
--Logistic Regression Model--
-Training Metrics-
```

```
AUC: 0.5040871934604905
[[9633    0]
 [ 364    3]]
```

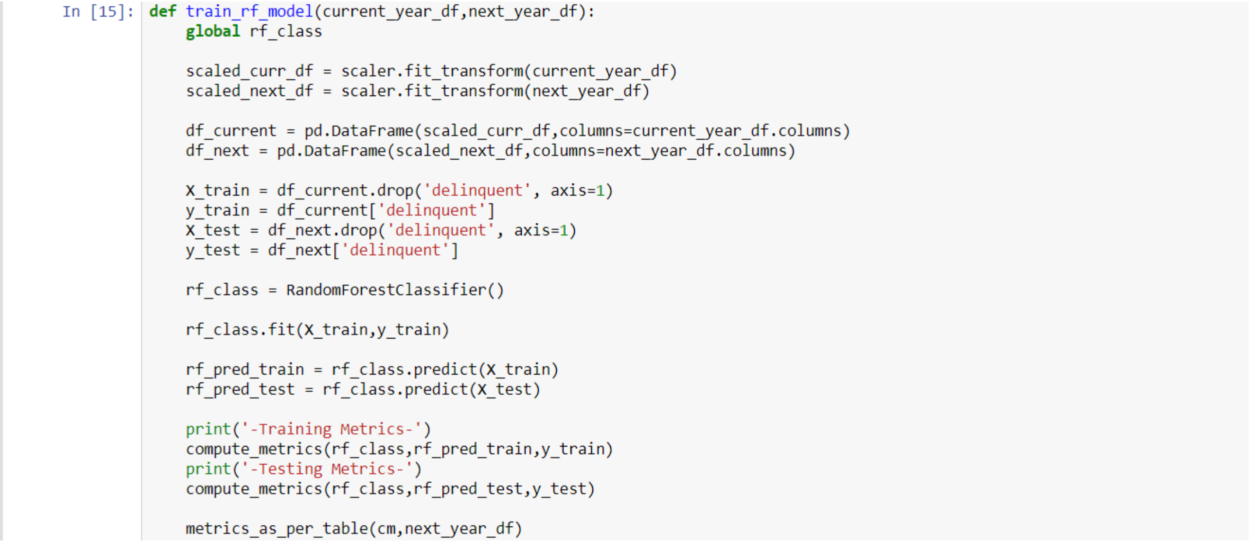


```
-Testing Metrics-
```

```
AUC: 0.5079575596816976
[[9623    0]
 [ 371    6]]
```

3.3.1.3 RANDOM FOREST

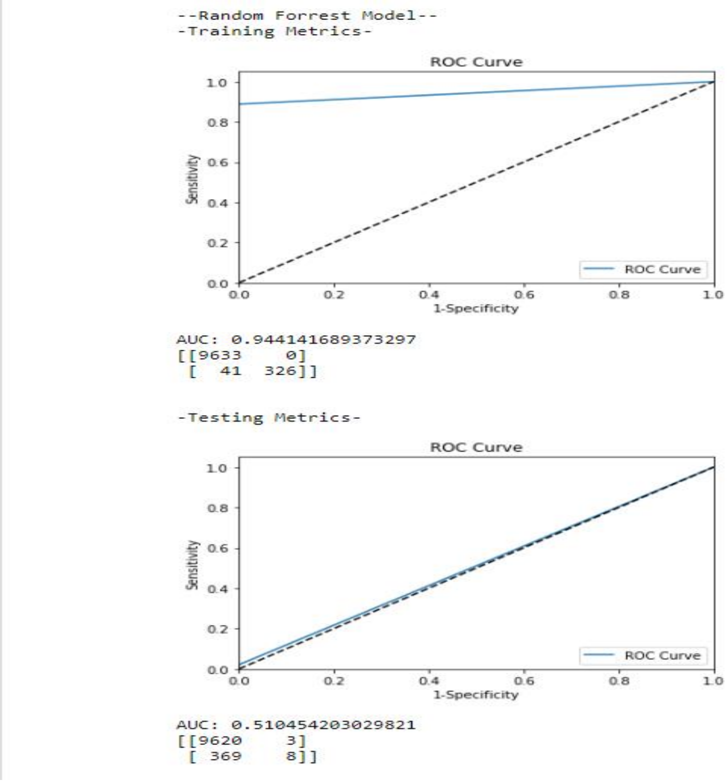
Random Forests grows many classification trees. To classify a new object from an input vector, put the input vector down each of the trees in the forest. Each tree gives a classification, and we say the tree "votes" for that class. The forest chooses the classification having the most votes (over all the trees in the forest).



```
In [15]: def train_rf_model(current_year_df,next_year_df):
             global rf_class

             scaled_curr_df = scaler.fit_transform(current_year_df)
             scaled_next_df = scaler.fit_transform(next_year_df)

             df_current = pd.DataFrame(scaled_curr_df,columns=current_year_df.columns)
             df_next = pd.DataFrame(scaled_next_df,columns=next_year_df.columns)

             X_train = df_current.drop('delinquent', axis=1)
             y_train = df_current['delinquent']
             X_test = df_next.drop('delinquent', axis=1)
             y_test = df_next['delinquent']

             rf_class = RandomForestClassifier()

             rf_class.fit(X_train,y_train)

             rf_pred_train = rf_class.predict(X_train)
             rf_pred_test = rf_class.predict(X_test)

             print('-Training Metrics-')
             compute_metrics(rf_class,rf_pred_train,y_train)
             print('-Testing Metrics-')
             compute_metrics(rf_class,rf_pred_test,y_test)

             metrics_as_per_table(cm,next_year_df)
```

<u>Accuracy Result & Classification Report</u>

In this case, our function will result the Confusion matrix as shown below and the ROC curve:



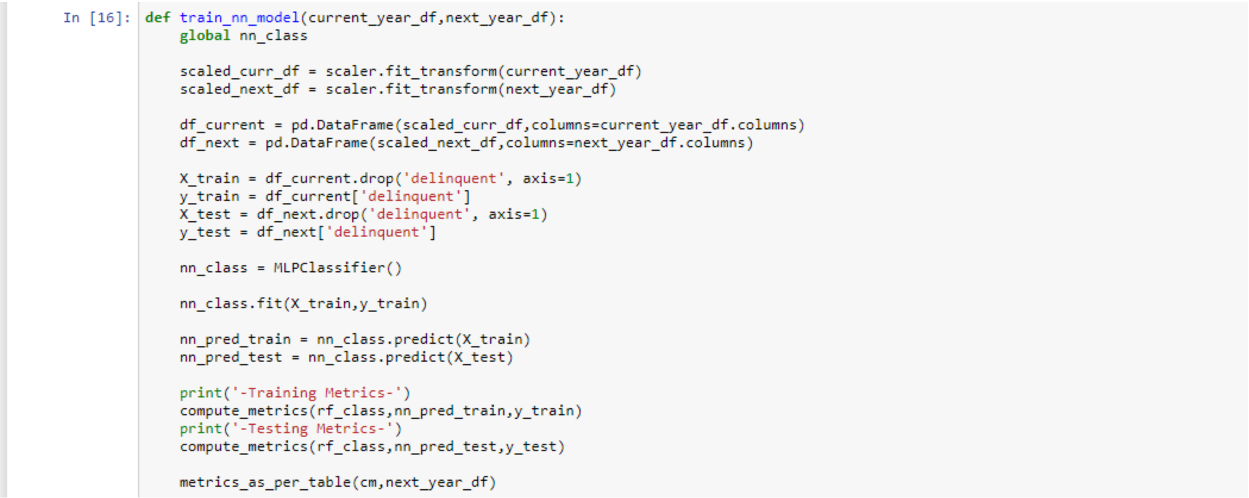
```
--Random Forrest Model--
-Training Metrics-
```

```
AUC: 0.944141689373297
[[9633    0]
 [  41  326]]

-Testing Metrics-
```

```
AUC: 0.510454203029821
[[9620    3]
 [ 369    8]]
```

3.3.1.4 NEURAL NETWORK

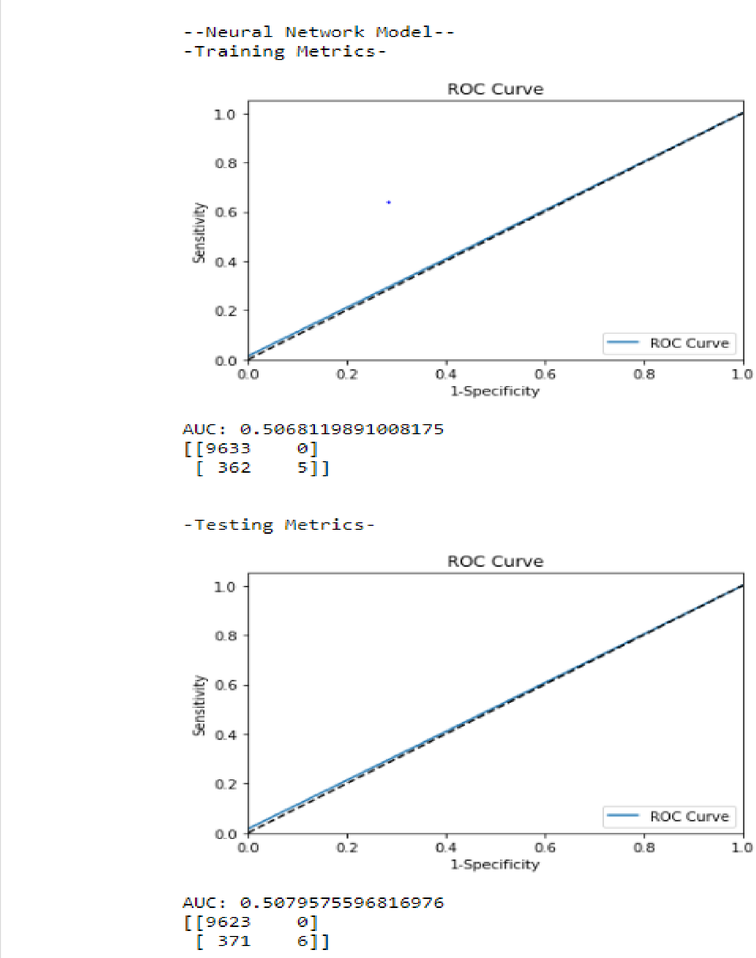
Artificial neural networks are relatively crude electronic networks of neurons based on the neural structure of the brain. They process records one at a time, and learn by comparing their classification of the record (i.e., largely arbitrary) with the known actual classification of the record.



```
In [16]: def train_nn_model(current_year_df,next_year_df):
             global nn_class

             scaled_curr_df = scaler.fit_transform(current_year_df)
             scaled_next_df = scaler.fit_transform(next_year_df)

             df_current = pd.DataFrame(scaled_curr_df,columns=current_year_df.columns)
             df_next = pd.DataFrame(scaled_next_df,columns=next_year_df.columns)

             X_train = df_current.drop('delinquent', axis=1)
             y_train = df_current['delinquent']
             X_test = df_next.drop('delinquent', axis=1)
             y_test = df_next['delinquent']

             nn_class = MLPClassifier()

             nn_class.fit(X_train,y_train)

             nn_pred_train = nn_class.predict(X_train)
             nn_pred_test = nn_class.predict(X_test)

             print('-Training Metrics-')
             compute_metrics(rf_class,nn_pred_train,y_train)
             print('-Testing Metrics-')
             compute_metrics(rf_class,nn_pred_test,y_test)

             metrics_as_per_table(cm,next_year_df)
```

Accuracy Result & Classification Report

In this case, our function will result the Confusion matrix as shown below and the ROC curve:

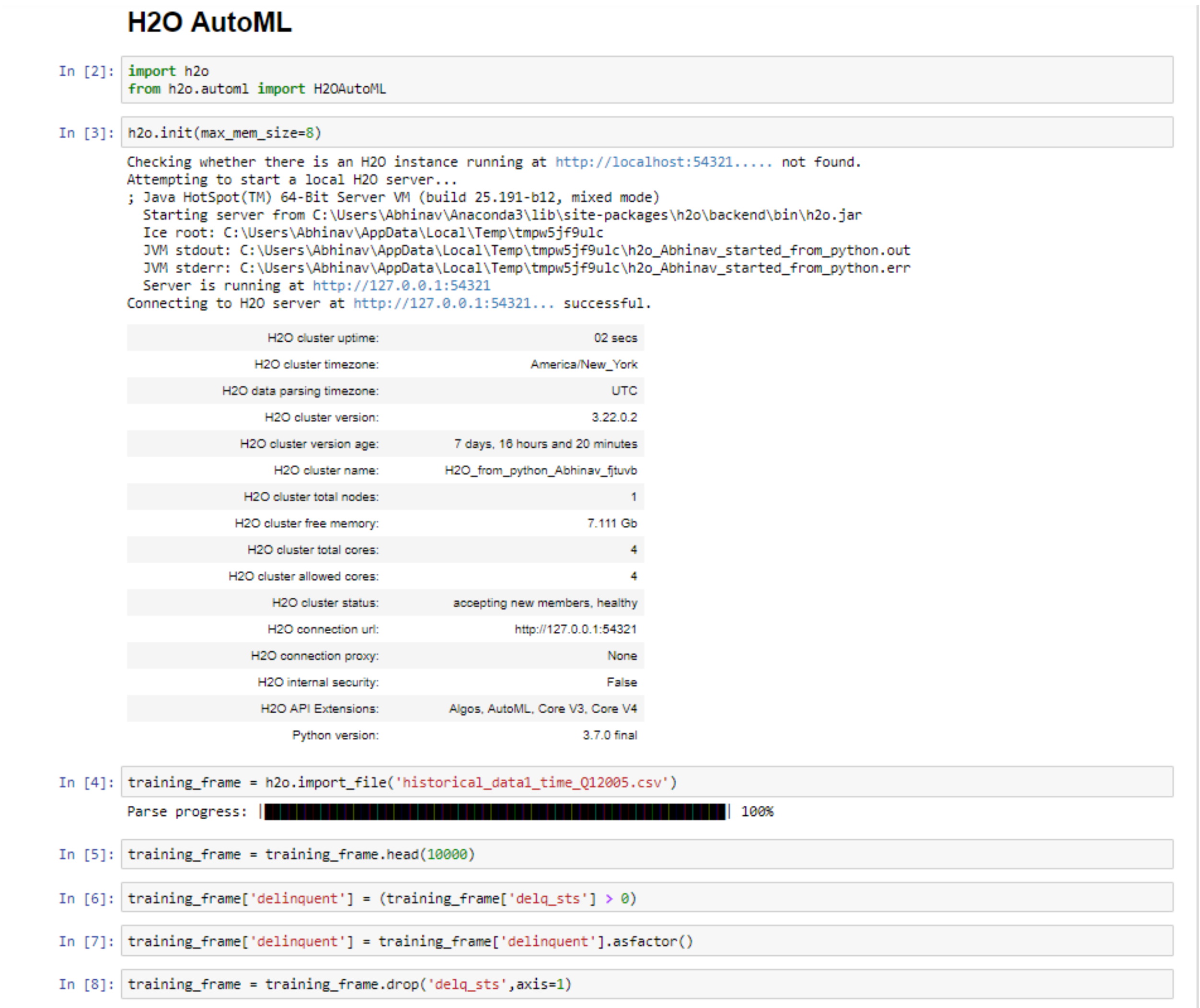


```
--Neural Network Model--
-Training Metrics-
```

```
AUC: 0.5068119891008175
[[9633    0]
 [ 362    5]]

-Testing Metrics-
```

```
AUC: 0.5079575596816976
[[9623    0]
 [ 371    6]]
```

Try TPOT, H20.Ai and AutoSKLearn Automl algorithms



H2O AutoML

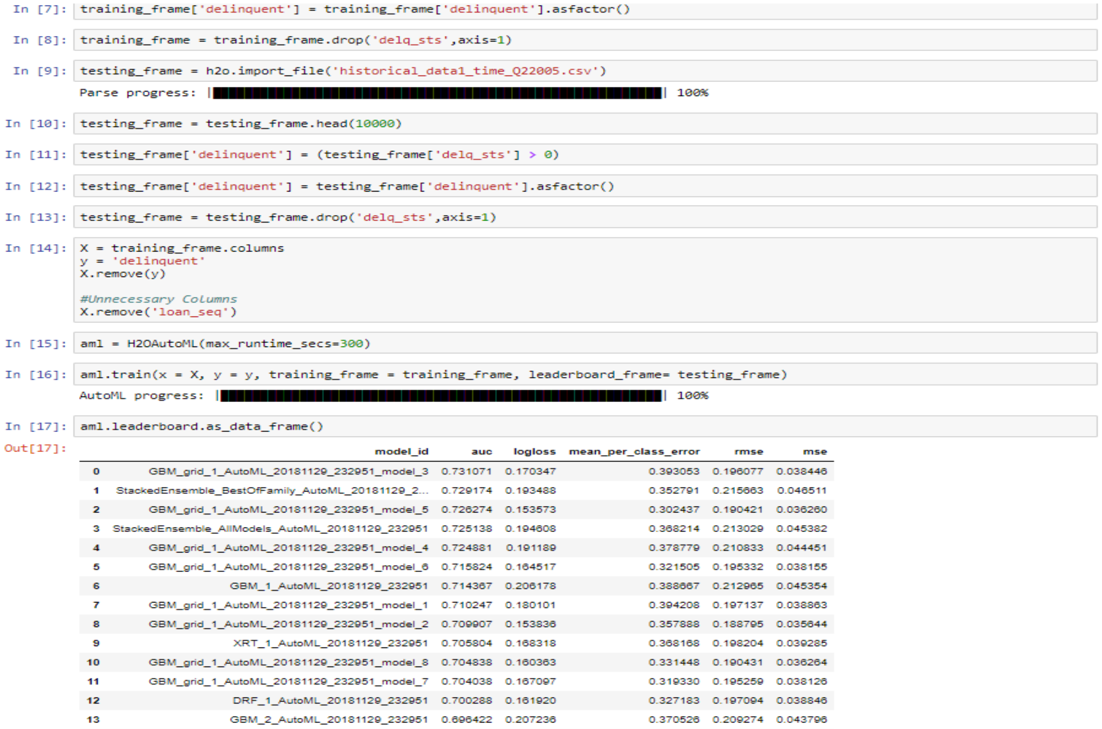
```
In [2]: import h2o
        from h2o.automl import H2OAutoML
```

```
In [3]: h2o.init(max_mem_size=8)
```

```
Checking whether there is an H2O instance running at http://localhost:54321..... not found.
Attempting to start a local H2O server...
; Java HotSpot(TM) 64-Bit Server VM (build 25.191-b12, mixed mode)
  Starting server from C:\Users\Abhinav\Anaconda3\lib\site-packages\h2o\backend\bin\h2o.jar
  Ice root: C:\Users\Abhinav\AppData\Local\Temp\tmpw5jf9ulc
  JVM stdout: C:\Users\Abhinav\AppData\Local\Temp\tmpw5jf9ulc\h2o_Abhinav_started_from_python.out
  JVM stderr: C:\Users\Abhinav\AppData\Local\Temp\tmpw5jf9ulc\h2o_Abhinav_started_from_python.err
  Server is running at http://127.0.0.1:54321
Connecting to H2O server at http://127.0.0.1:54321... successful.
```

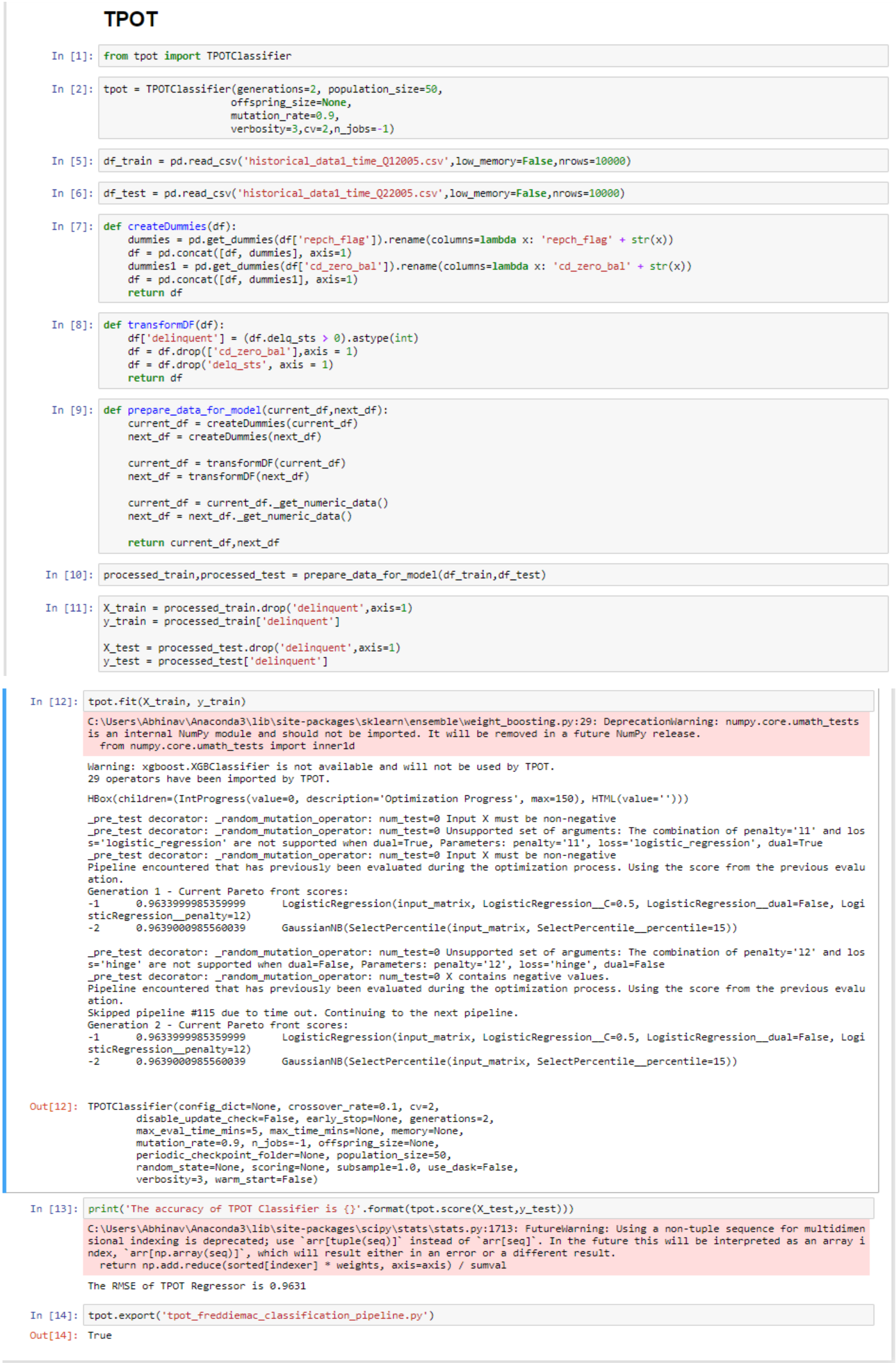
H2O cluster uptime:	02 secs
H2O cluster timezone:	America/New_York
H2O data parsing timezone:	UTC
H2O cluster version:	3.22.0.2
H2O cluster version age:	7 days, 16 hours and 20 minutes
H2O cluster name:	H2O_from_python_Abhinav_fjtuvb
H2O cluster total nodes:	1
H2O cluster free memory:	7.111 Gb
H2O cluster total cores:	4
H2O cluster allowed cores:	4
H2O cluster status:	accepting new members, healthy
H2O connection url:	http://127.0.0.1:54321
H2O connection proxy:	None
H2O internal security:	False
H2O API Extensions:	Algos, AutoML, Core V3, Core V4
Python version:	3.7.0 final

```
In [4]: training_frame = h2o.import_file('historical_data1_time_Q12005.csv')
        Parse progress: |█████████████████████████████████████████████████████████| 100%

In [5]: training_frame = training_frame.head(10000)

In [6]: training_frame['delinquent'] = (training_frame['delq_sts'] > 0)

In [7]: training_frame['delinquent'] = training_frame['delinquent'].asfactor()

In [8]: training_frame = training_frame.drop('delq_sts',axis=1)
```



```
In [7]: training_frame['delinquent'] = training_frame['delinquent'].asfactor()

In [8]: training_frame = training_frame.drop('delq_sts',axis=1)

In [9]: testing_frame = h2o.import_file('historical_data1_time_Q22005.csv')
        Parse progress: |█████████████████████████████████████████████████████████| 100%

In [10]: testing_frame = testing_frame.head(10000)

In [11]: testing_frame['delinquent'] = (testing_frame['delq_sts'] > 0)

In [12]: testing_frame['delinquent'] = testing_frame['delinquent'].asfactor()

In [13]: testing_frame = testing_frame.drop('delq_sts',axis=1)

In [14]: X = training_frame.columns
         y = 'delinquent'
         X.remove(y)

         #Unnecessary Columns
         X.remove('loan_seq')

In [15]: aml = H2OAutoML(max_runtime_secs=300)

In [16]: aml.train(x = X, y = y, training_frame = training_frame, leaderboard_frame= testing_frame)
         AutoML progress: |████████████████████████████████████████████████████████| 100%

In [17]: aml.leaderboard.as_data_frame()

Out[17]:
```

	model_id	auc	logloss	mean_per_class_error	rmse	mse
0	GBM_grid_1_AutoML_20181129_232951_model_3	0.731071	0.170347	0.393053	0.196077	0.038446
1	StackedEnsemble_BestOfFamily_AutoML_20181129_2...	0.729174	0.193488	0.352791	0.215663	0.046511
2	GBM_grid_1_AutoML_20181129_232951_model_5	0.726274	0.153573	0.302437	0.190421	0.036260
3	StackedEnsemble_AllModels_AutoML_20181129_232951	0.725138	0.194608	0.368214	0.213029	0.045382
4	GBM_grid_1_AutoML_20181129_232951_model_4	0.724881	0.191189	0.378779	0.210833	0.044451
5	GBM_grid_1_AutoML_20181129_232951_model_6	0.715824	0.164517	0.321505	0.195332	0.038155
6	GBM_1_AutoML_20181129_232951	0.714367	0.206178	0.388667	0.212965	0.045354
7	GBM_grid_1_AutoML_20181129_232951_model_1	0.710247	0.180101	0.394208	0.197137	0.038863
8	GBM_grid_1_AutoML_20181129_232951_model_2	0.709907	0.153836	0.357888	0.188795	0.035644
9	XRT_1_AutoML_20181129_232951	0.705804	0.168318	0.368168	0.198204	0.039285
10	GBM_grid_1_AutoML_20181129_232951_model_8	0.704838	0.160363	0.331448	0.190431	0.036264
11	GBM_grid_1_AutoML_20181129_232951_model_7	0.704038	0.167097	0.319330	0.195259	0.038126
12	DRF_1_AutoML_20181129_232951	0.700288	0.161920	0.327183	0.197094	0.038846
13	GBM_2_AutoML_20181129_232951	0.696422	0.207236	0.370526	0.209274	0.043796



TPOT

```
In [1]: from tpot import TPOTClassifier

In [2]: tpot = TPOTClassifier(generations=2, population_size=50,
                              offspring_size=None,
                              mutation_rate=0.9,
                              verbosity=3,cv=2,n_jobs=-1)

In [5]: df_train = pd.read_csv('historical_data1_time_Q12005.csv',low_memory=False,nrows=10000)

In [6]: df_test = pd.read_csv('historical_data1_time_Q22005.csv',low_memory=False,nrows=10000)

In [7]: def createDummies(df):
            dummies = pd.get_dummies(df['repch_flag']).rename(columns=lambda x: 'repch_flag' + str(x))
            df = pd.concat([df, dummies], axis=1)
            dummies1 = pd.get_dummies(df['cd_zero_bal']).rename(columns=lambda x: 'cd_zero_bal' + str(x))
            df = pd.concat([df, dummies1], axis=1)
            return df

In [8]: def transformDF(df):
            df['delinquent'] = (df.delq_sts > 0).astype(int)
            df = df.drop(['cd_zero_bal'],axis = 1)
            df = df.drop('delq_sts', axis = 1)
            return df

In [9]: def prepare_data_for_model(current_df,next_df):
            current_df = createDummies(current_df)
            next_df = createDummies(next_df)

            current_df = transformDF(current_df)
            next_df = transformDF(next_df)

            current_df = current_df._get_numeric_data()
            next_df = next_df._get_numeric_data()

            return current_df,next_df

In [10]: processed_train,processed_test = prepare_data_for_model(df_train,df_test)

In [11]: X_train = processed_train.drop('delinquent',axis=1)
         y_train = processed_train['delinquent']

         X_test = processed_test.drop('delinquent',axis=1)
         y_test = processed_test['delinquent']

In [12]: tpot.fit(X_train, y_train)

         C:\Users\Abhinav\Anaconda3\lib\site-packages\sklearn\ensemble\weight_boosting.py:29: DeprecationWarning: numpy.core.umath_tests
         is an internal NumPy module and should not be imported. It will be removed in a future NumPy release.
           from numpy.core.umath_tests import inner1d

         Warning: xgboost.XGBClassifier is not available and will not be used by TPOT.
         29 operators have been imported by TPOT.

         HBox(children=(IntProgress(value=0, description='Optimization Progress', max=150), HTML(value='')))

         _pre_test decorator: _random_mutation_operator: num_test=0 Input X must be non-negative
         _pre_test decorator: _random_mutation_operator: num_test=0 Unsupported set of arguments: The combination of penalty='l1' and los
         s='logistic_regression' are not supported when dual=True, Parameters: penalty='l1', loss='logistic_regression', dual=True
         _pre_test decorator: _random_mutation_operator: num_test=0 Input X must be non-negative
         Pipeline encountered that has previously been evaluated during the optimization process. Using the score from the previous evalu
         ation.
         Generation 1 - Current Pareto front scores:
         -1      0.9633999985359999      LogisticRegression(input_matrix, LogisticRegression__C=0.5, LogisticRegression__dual=False, Logi
         sticRegression__penalty=l2)
         -2      0.9639000985560039      GaussianNB(SelectPercentile(input_matrix, SelectPercentile__percentile=15))

         _pre_test decorator: _random_mutation_operator: num_test=0 Unsupported set of arguments: The combination of penalty='l2' and los
         s='hinge' are not supported when dual=False, Parameters: penalty='l2', loss='hinge', dual=False
         _pre_test decorator: _random_mutation_operator: num_test=0 X contains negative values.
         Pipeline encountered that has previously been evaluated during the optimization process. Using the score from the previous evalu
         ation.
         Skipped pipeline #115 due to time out. Continuing to the next pipeline.
         Generation 2 - Current Pareto front scores:
         -1      0.9633999985359999      LogisticRegression(input_matrix, LogisticRegression__C=0.5, LogisticRegression__dual=False, Logi
         sticRegression__penalty=l2)
         -2      0.9639000985560039      GaussianNB(SelectPercentile(input_matrix, SelectPercentile__percentile=15))


Out[12]: TPOTClassifier(config_dict=None, crossover_rate=0.1, cv=2,
                 disable_update_check=False, early_stop=None, generations=2,
                 max_eval_time_mins=5, max_time_mins=None, memory=None,
                 mutation_rate=0.9, n_jobs=-1, offspring_size=None,
                 periodic_checkpoint_folder=None, population_size=50,
                 random_state=None, scoring=None, subsample=1.0, use_dask=False,
                 verbosity=3, warm_start=False)

In [13]: print('The accuracy of TPOT Classifier is {}'.format(tpot.score(X_test,y_test)))

         C:\Users\Abhinav\Anaconda3\lib\site-packages\scipy\stats\stats.py:1713: FutureWarning: Using a non-tuple sequence for multidimen
         sional indexing is deprecated; use `arr[tuple(seq)]` instead of `arr[seq]`. In the future this will be interpreted as an array i
         ndex, `arr[np.array(seq)]`, which will result either in an error or a different result.
           return np.add.reduce(sorted[indexer] * weights, axis=axis) / sumval

         The RMSE of TPOT Regressor is 0.9631

In [14]: tpot.export('tpot_freddiemac_classification_pipeline.py')

Out[14]: True
```

Comment on the quality of the model and it's outputs. What can you do to do better? Would you recommend using this model to predict delinquents in the next quarter? Justify your answers

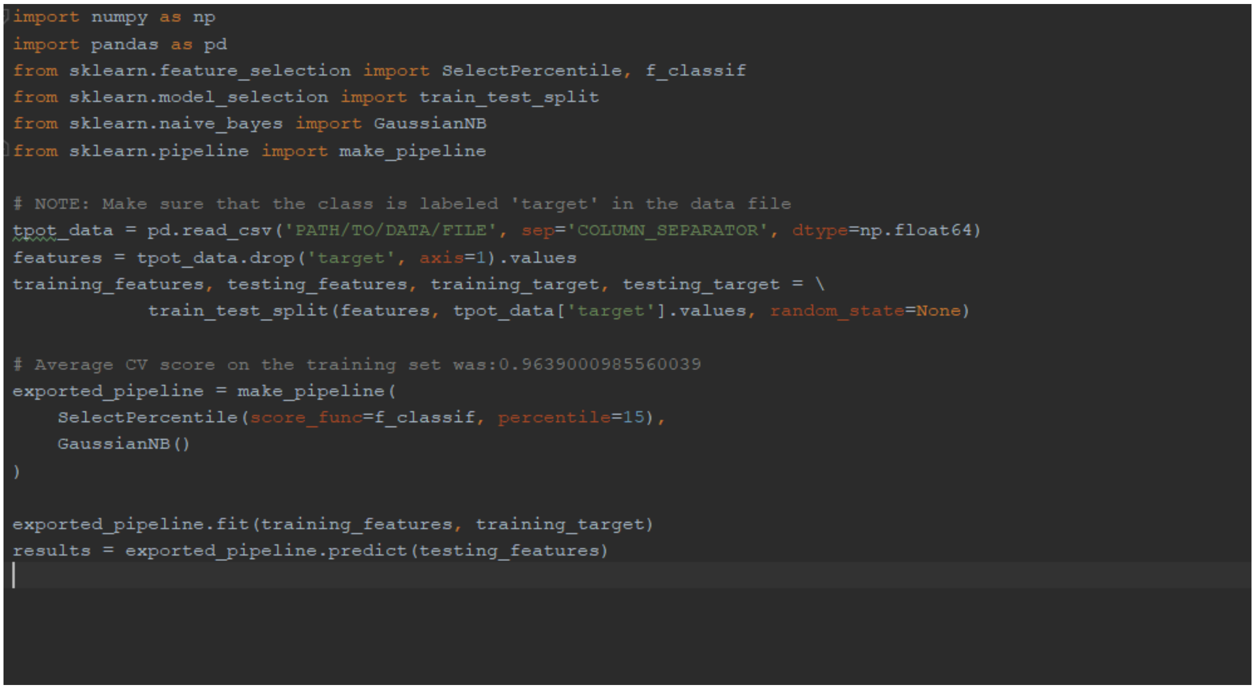
Random forest (or random forests) is an ensemble classifier that consists of many decision trees and outputs the class that is the mode of the class's output by individual trees

Why – Random Forest?

- Rand Forest is fast to build. Even faster to predict!
- Practically speaking, not requiring cross-validation alone for model selection significantly speeds training by 10x-100x or more.
- Automatic predictor selection from large number of candidates
- Resistance to over training
- Ability to handle data without preprocessing
- data does not need to be rescaled, transformed, or modified resistant to outliers
- automatic handling of missing values
- Cluster identification can be used to generate tree-based clusters through sample proximity

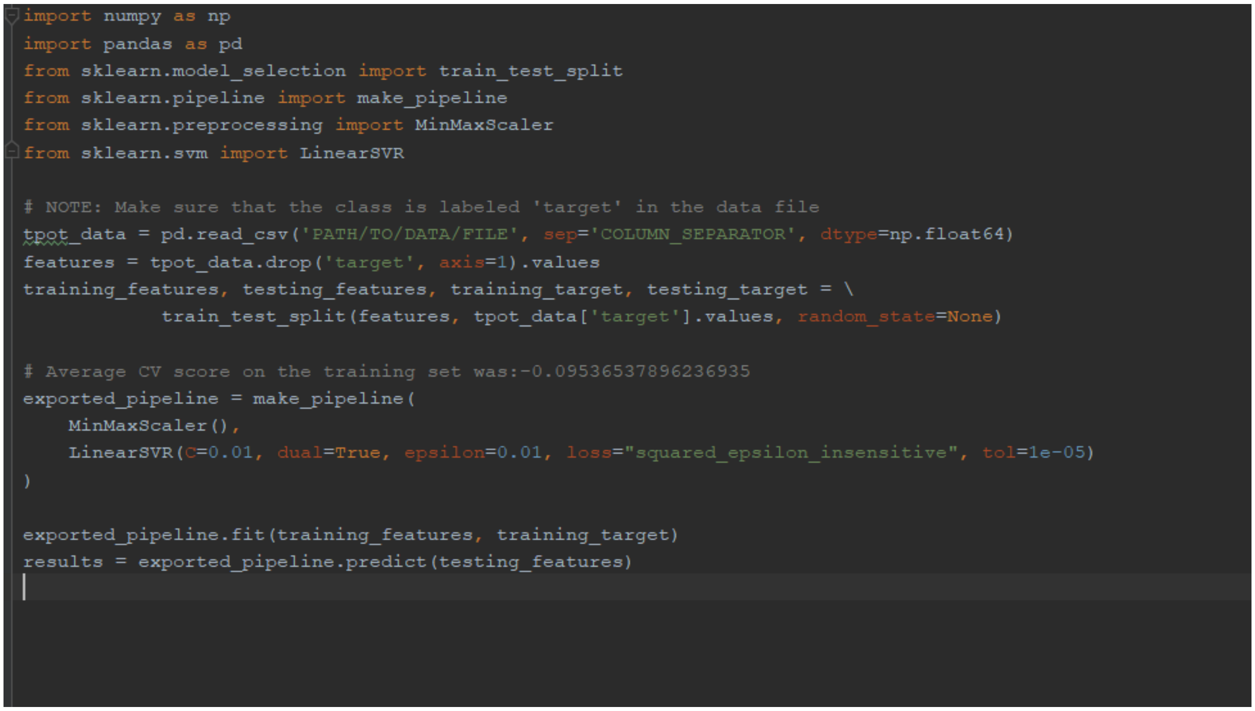
PIPELINE

Classification



```
import numpy as np
import pandas as pd
from sklearn.feature_selection import SelectPercentile, f_classif
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import GaussianNB
from sklearn.pipeline import make_pipeline

# NOTE: Make sure that the class is labeled 'target' in the data file
tpot_data = pd.read_csv('PATH/TO/DATA/FILE', sep='COLUMN_SEPARATOR', dtype=np.float64)
features = tpot_data.drop('target', axis=1).values
training_features, testing_features, training_target, testing_target = \
            train_test_split(features, tpot_data['target'].values, random_state=None)

# Average CV score on the training set was:0.9639000985560039
exported_pipeline = make_pipeline(
    SelectPercentile(score_func=f_classif, percentile=15),
    GaussianNB()
)

exported_pipeline.fit(training_features, training_target)
results = exported_pipeline.predict(testing_features)
```

Prediction



```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.pipeline import make_pipeline
from sklearn.preprocessing import MinMaxScaler
from sklearn.svm import LinearSVR

# NOTE: Make sure that the class is labeled 'target' in the data file
tpot_data = pd.read_csv('PATH/TO/DATA/FILE', sep='COLUMN_SEPARATOR', dtype=np.float64)
features = tpot_data.drop('target', axis=1).values
training_features, testing_features, training_target, testing_target = \
            train_test_split(features, tpot_data['target'].values, random_state=None)

# Average CV score on the training set was:-0.09536537896236935
exported_pipeline = make_pipeline(
    MinMaxScaler(),
    LinearSVR(C=0.01, dual=True, epsilon=0.01, loss="squared_epsilon_insensitive", tol=1e-05)
)

exported_pipeline.fit(training_features, training_target)
results = exported_pipeline.predict(testing_features)
```