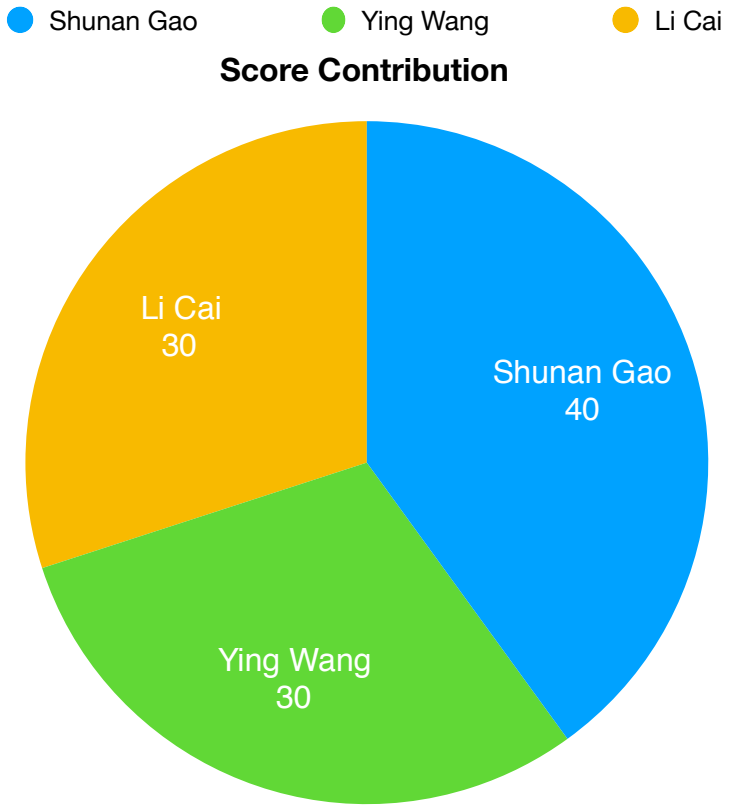
Team-10
Info-7390
Advanced Data Science and Architecture

Assignment Report



Part I Data Wrangling:

docker hub link:
https://hub.docker.com/r/shunangao/hw3-p1/
Then
docker pull shunangao/hw3-p1
Then
docker run shunangao/hw3-p1

1. Download data from website:

To programmatically downloading the file, first the user should create username and password. Once logged in, the user can download all the file required for analysis. We have used the python requests 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.

```
USERNAME='gao.shu@husky.neu.edu'
PASSWORD='gj{vx<Io'
print('username='+uname)
print('password='+pwd)

payload = {
    "username": USERNAME,
    "password": PASSWORD
}


session_requests = requests.session()

login_url = "https://freddiemac.embs.com/FLoan/secure/auth.php"
```

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.

```
result = session_requests.post(
    login_url,
    data = payload,
    headers = dict(referer=login_url)
)
tree = html.fromstring(result.content)
all_links = tree.findall(".//a")
```

Once we log in. We use html.fromstring() to get the url content and use findall() function to get all the link.

```
############### Download zips ###############
for link in all_links:
    href=link.get("href")
    if "sample" in href:
      if int(link.text[-8:-4]) >= 2005:
        url= 'https://freddiemac.embs.com/FLoan/Data/'+href
        r = session_requests.get(url,stream=True)
        with open(os.path.join('downloaded_zips',link.text), 'wb') as f:
            for chunk in r.iter_content(chunk_size=1024):
                if chunk: # filter out keep-alive new chunks
                    f.write(chunk)
```

The data need to be downloaded is in the 'herf' part of the link and for this assignment we need to download all the sample file after 2005.

```
############### Unzip and extract text files ###############
try:
    zip_files = os.listdir('downloaded_zips')
    for f in zip_files:
        z = zipfile.ZipFile(os.path.join('downloaded_zips', f), 'r')
        for file in z.namelist():
            if file.endswith('.txt'):
                z.extract(file, r'downloaded_zips_unzipped')
    logging.info('Zip files successfully extracted to folder: downloaded_zips_unzipped.')
except Exception as e:
        logging.error(str(e))
        exit()
```

Unzip all the sample file and save it to a folder

2. Data preprocessing:

After download the data we need to clean the data for each file and save it to a new file.

```
for file in orig_filelists:

    data =pd.read_csv(file, sep='|', names= col_names_orig, low_memory=False)

    data = data.drop(columns = ['super_conforming_flag','pre_harp_loan_sequence_no'])
    # remove rows contain more than 4 NaN
    data.dropna(thresh=4)
    #remove rows doesn't have loan_sequence_no
    data = data[pd.notnull(data['loan_sequence_no'])]
```

For orig file, we first remove some columns which has too many NaN value. Then we will remove the row which has more than 4 missing value and the row don't have loan sequence number.

```
# replace credit_score with mode
data.credit_score=data.credit_score.replace(r'\s+', np.nan, regex=True).astype('float64')
cs = pd.DataFrame(data['credit_score'])
mode=cs.mode()
data['credit_score'] = data['credit_score'].fillna(mode.iloc[0]['credit_score'])

# replace unknown value of msa with 0000
data.msa=data.msa.replace(r'\s+', np.nan, regex=True)
data['msa'] = data['msa'].fillna('00000').astype('float64')

# replace unknown value of mi_percentage with 0
data.mi_percentage=data.mi_percentage.replace(r'\s+', np.nan, regex=True).astype('float64')
data['mi_percentage'] = data['mi_percentage'].fillna(0)

# replace unknown value of no_of_units with 1
data.no_of_units=data.no_of_units.replace(r'\s+', np.nan, regex=True).astype('float64')
data['no_of_units'] = data['no_of_units'].fillna(1)
```

After that we will fill some missing value with the appropriate value.

```
for file in svcg_filelists:
    data=pd.read_csv(file, sep='|', names= col_names_svcg, low_memory=False)
    #remove rows contain NaN loan_sequence_no
    data = data[pd.notnull(data['loan_sequence_no'])]

    #remove rows contain NaN monthly_report_period
    data = data[pd.notnull(data['monthly_reporting_period'])]

    #remove rows contain NaN current_actual_upb
    data = data[pd.notnull(data['current_actual_upb'])]

    #remove rows contain NaN current_loan_delinquency_status
    data = data[pd.notnull(data['current_loan_delinquency_status'])]

    #remove rows contain NaN loan_age
    data = data[pd.notnull(data['loan_age'])]

    #remove rows contain NaN remaning_months_on_legal_maturity
    data = data[pd.notnull(data['remaning_months_on_legal_maturity'])]
```

For svcg file. First remove rows which has the NaN value in some columns which can not be none.

And the we will remove some columns which has too many NaN data .

```
#remove cols which has too many NaN
data = data.drop(columns = ['repurchase_flag','modification_flag','zero_bal_eff_date','ddlpi',
'mi_recoveries', 'net_sales_proceeds','non_mi_recoveries','expenses','legal_costs',
'maintenance_preservation_cost','taxes_insurance','misc_expenses','actual_loss_calc',
'modification_cost','step_modification_flag','deferred_payment_modification,'])

#replace unknown with 0
data.estimated_loan_to_value=data.estimated_loan_to_value.replace(r'\s+', np.nan, regex=True)
data['estimated_loan_to_value'] = data['estimated_loan_to_value'].fillna('0').astype('float64')
```

All the cleaned orig and svcg files will be save to the new folder for further processing.

Summary File:
To summarize the orig file, we create a new data frame and add some columns. The new columns is to store the summary information of the orig file and for each year is one row. We combine all the year and the save the df to a new csv file.

```
summ_df['year'] = file[-8:-4]
summ_df["aveCreditScore"] = data["credit_score"].mean()
summ_df["loanCount"] = np.count_nonzero(data["loan_sequence_no"])
summ_df["aveMI"] = data["mi_percentage"].mean()
summ_df["aveUnitNumber"] = data["no_of_units"].mean()
summ_df["aveCLTV"] = data["original_cltv"].mean()
summ_df["aveDTI"] = data["original_dti_ratio"].mean()
summ_df["aveLTV"] = data["original_ltv"].mean()
summ_df["totalUPB"] = data["original_upb"].sum()
summ_df["aveUPB"] = data["original_upb"].mean()
summ_df["aveInterestRate"] = data["original_interest_rate"].mean()
summ_df["mostFrequentState"] = pd.DataFrame(data.groupby('property_state').size().rename('cnt')).idxmax()[0]
summ_df["mostFrequentType"] =  pd.DataFrame(data.groupby('property_type').size().rename('cnt')).idxmax()[0]
summ_df["mostFrequentPostalCode"] =  pd.DataFrame(data.groupby('postal_code').size().rename('cnt')).idxmax()[0]
summ_df["aveNumberOfBorrowers"] = data["no_of_borrowers"].mean()
summ_df["mostFrequentSeller"] =  pd.DataFrame(data.groupby('seller_name').size().rename('cnt')).idxmax()[0]
summ_df["mostFrequentServicer"] =  pd.DataFrame(data.groupby('servicer_name').size().rename('cnt')).idxmax()[0]
```

And also the svcg file:

```
summ_df = OrderedDict()
summ_df['year'] = file[-8:-4]
total_loans = data['loan_sequence_no'].nunique()
summ_df['no_distinct_loans_per_year'] = data['loan_sequence_no'].nunique()
summ_df['most_lenth_loan_months'] = data.groupby('loan_sequence_no').size().max()
total_reos = data[data['current_loan_delinquency_status'].astype(str) == 'R']['loan_sequence_no'].nunique()
summ_df['total_reos_per_year_percent'] = 100 * total_reos/total_loans
summ_df['mean_upb_year'] = data['current_actual_upb'].mean()
summ_df['avg_loan_age'] = data['loan_age'].mean()
summ_df['avg_current_interest_rate'] = data['current_interest_rate'].mean()
```

These two summary csv file will also be stored in a new folder.
Then we combine all the cleaned csv file together:

```
concatDf = pd.DataFrame(columns=colNames)
concatDf = pd.concat(dataList ,axis = 0)
concatDf.to_csv(os.path.join('cleanFilesWithSummaries',file_name),index=False)
```

Exploratory Data Analysis:

After finishing the data processing we will use a jupyter notebook to visualize the data.

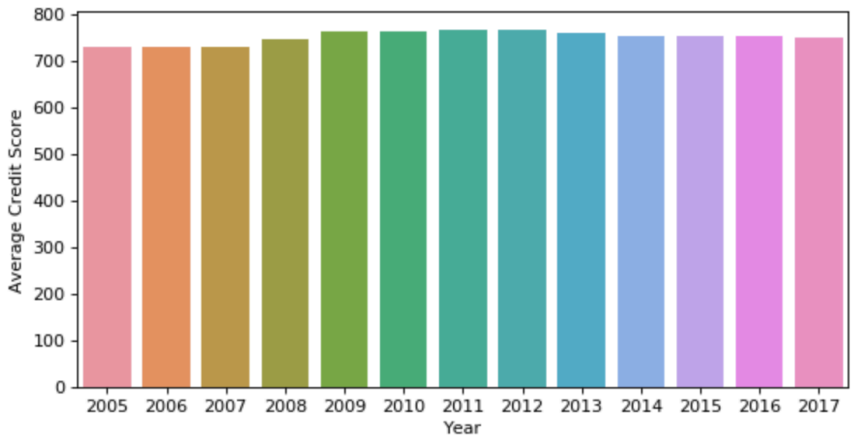
Origination Summary:

The origination file looks like this:

```
orig = orig.sort_values(by=['year'])
orig.head()
```

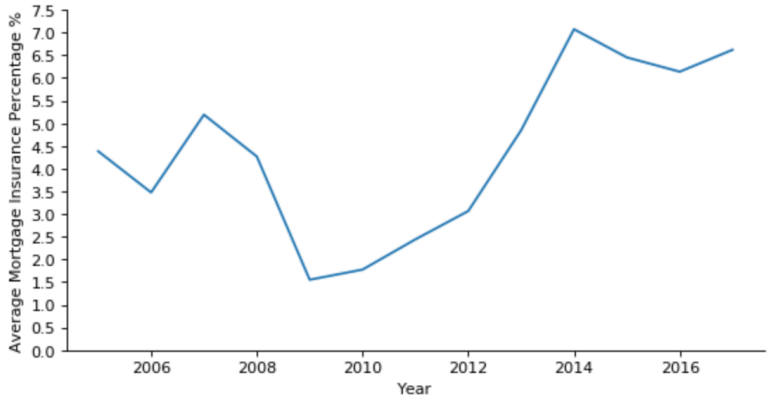
	year	aveCreditScore	loanCount	aveMI	aveUnitNumber	aveCLTV	aveDTI	aveLTV	t
7	2005	728.49904	50000	4.38436	1.02644	71.21610	65.38164	69.56804	8
12	2006	729.95500	50000	3.47626	1.02774	73.13102	57.58518	70.71394	8
9	2007	728.46154	50000	5.19054	1.03666	74.53890	61.62562	72.10360	9
0	2008	746.36954	50000	4.26810	1.03666	71.50462	57.18436	70.28216	1
1	2009	762.33614	50000	1.55242	1.01600	66.85308	34.21672	65.44966	1

The average credit score vs year plot:



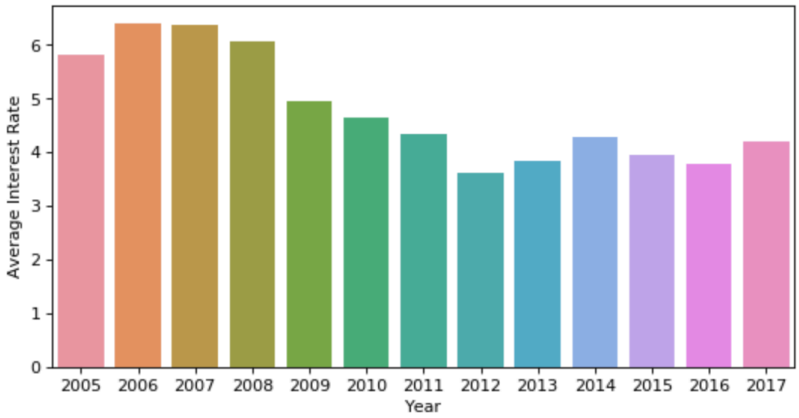
we can see that average credit score doesn't change that much.

Average mortgage insurance percentage vs year plot:



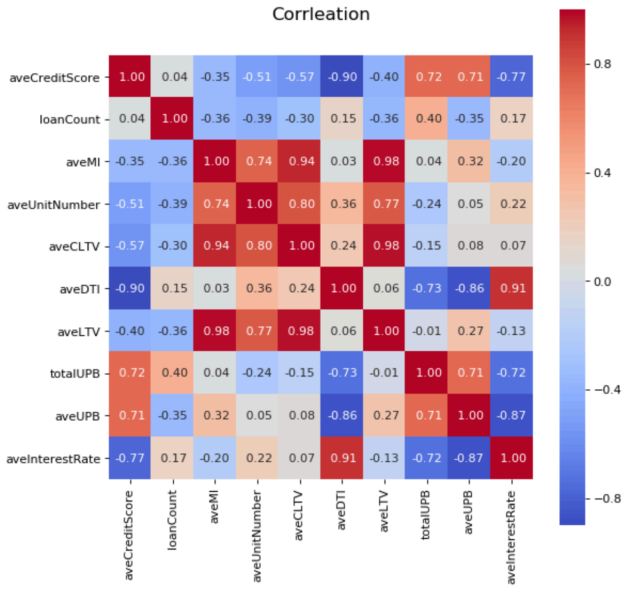
we can see the average MI lowest is 2009 and the peak value is 2014

Average interest rate:



we can see the average interest rate peak value is 2006 and the bottom value is 2012

Correlation:



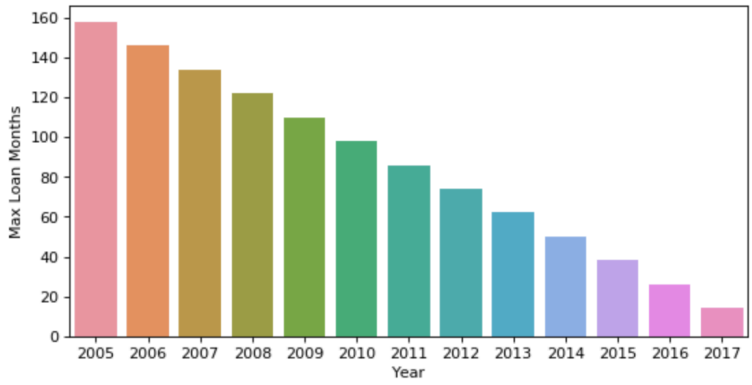
We can see that some of the attribute has the correlation such as credit score and DTI

Performance Summary:

```
svcg = svcg.sort_values(by=['year'])
svcg.head()
```

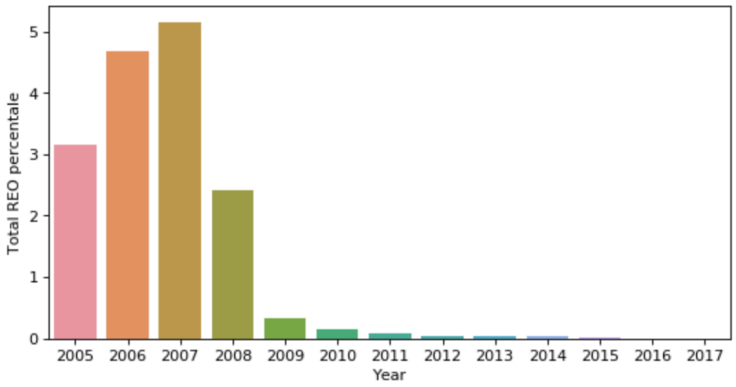
	year	no_distinct_loans_per_year	most_lenth_loan_months	total_reos_per_year_percent	mean_upb_year
8	2005	49999	158	3.144063	147470.359979
5	2006	49997	146	4.670280	157320.881169
6	2007	49996	134	5.154412	160294.361364
12	2008	49995	122	2.416242	169830.639024
11	2009	49998	110	0.322013	175750.124093

The longest loan month each year:



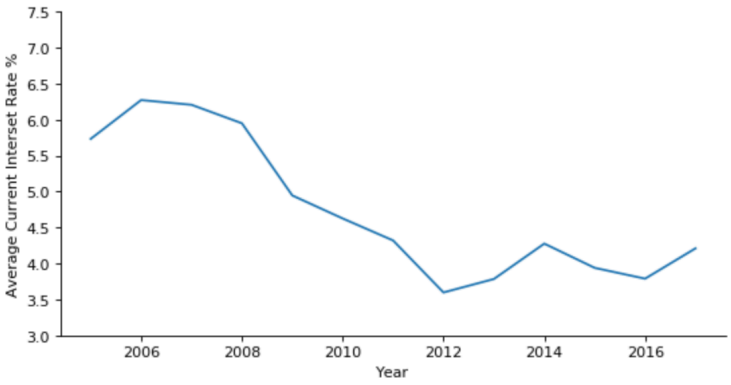
The longest loan month is decreasing because the more recent the more less

Total REO percentage each year:



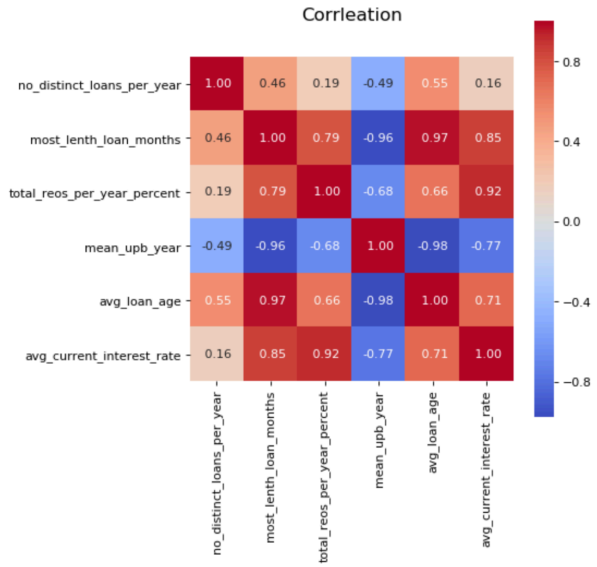
The total REO reach the peak value at 2007 and after 2008 it decrease significantly

Average Interest rate each year:



The average interest rate is totally decreasing by year

Correlation:



We can see that some attribute have strong relation such as mean upb and loan age

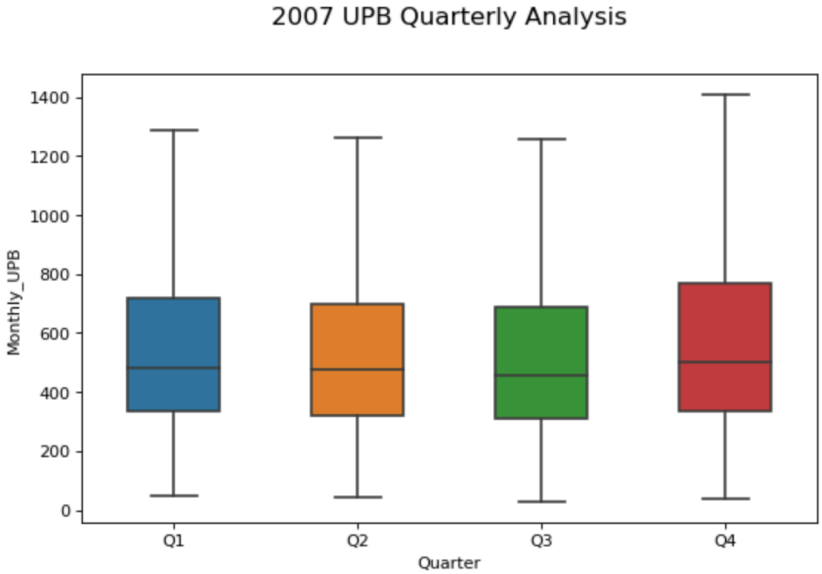
Quarterly Analysis 2007:

Origination file: orig_2007.csv

```
orig_2007.head()
```

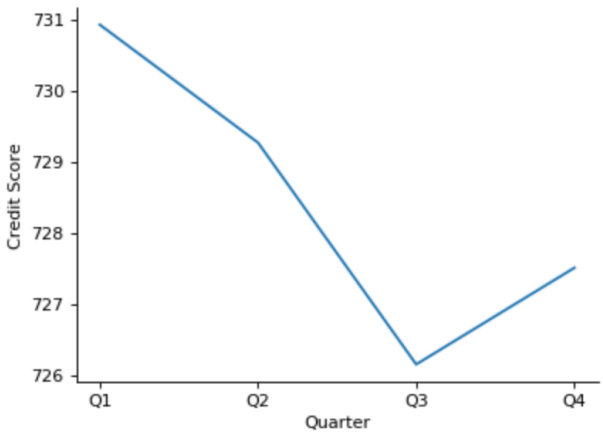
	credit_score	first_pay_date	first_time_homebuyer	maturity_date	msa	mi_percentage	no_of_units
0	606.0	200703	N	203702	0.0	0.0	1.0
1	712.0	200703	N	203702	0.0	0.0	1.0
2	698.0	200704	N	203703	24020.0	0.0	1.0
3	757.0	200703	N	203702	0.0	0.0	1.0
4	761.0	200703	N	203702	24660.0	0.0	1.0

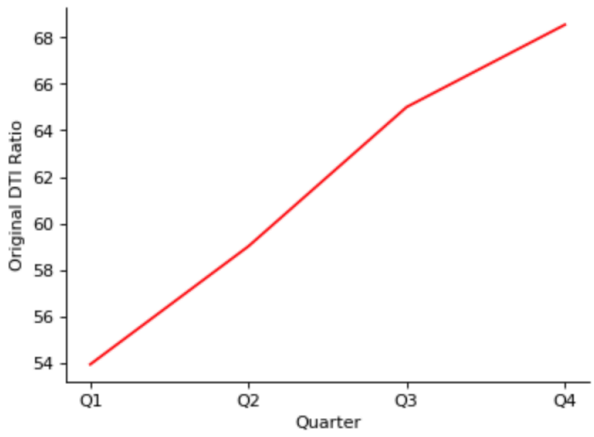
UPB Quarterly distribution: pretty the same



Assignment 3 Dec

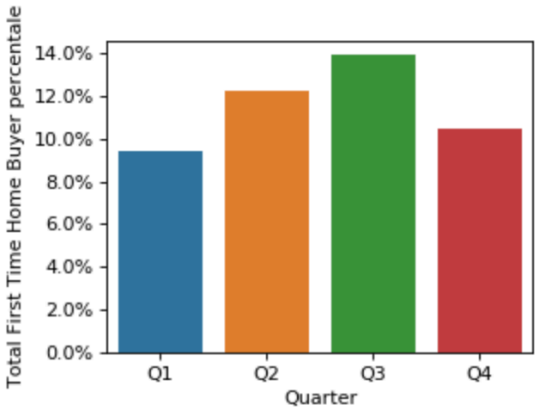
Credit score and Original DTI Ratio:



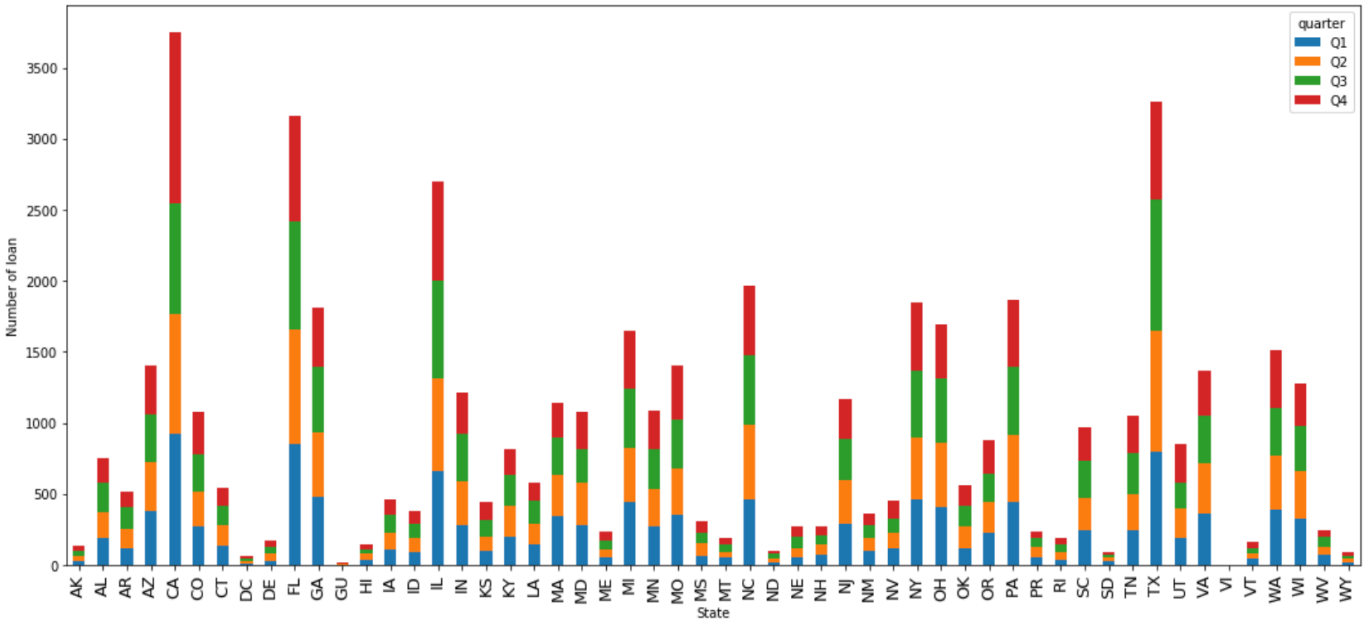


The credit score decreased while DTI ratio is increased

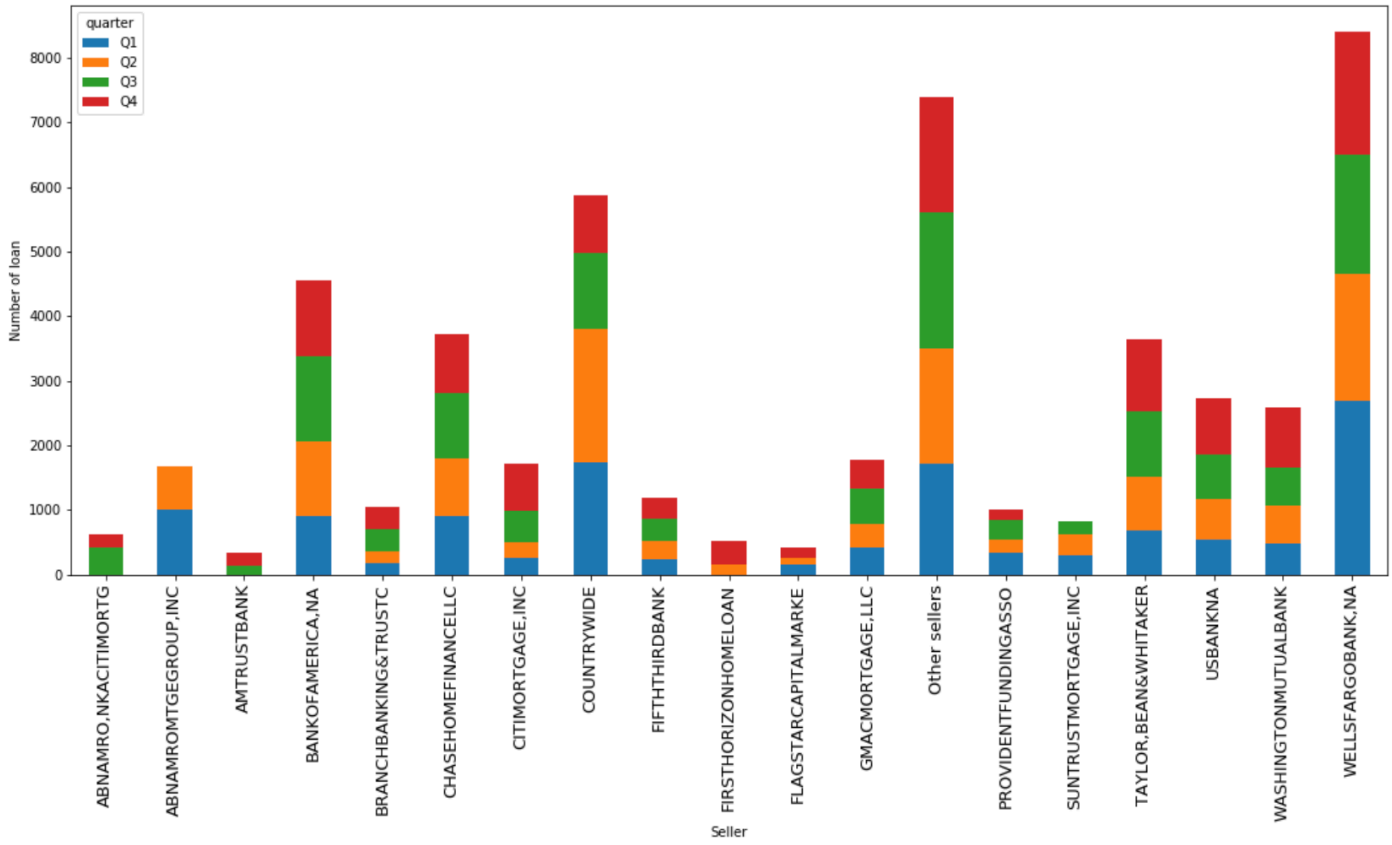
First time home buyer percentage each quarter: Q3 have the most first time buyers



Number of loans in each state quarterly: totally CA no.1 but Q3 TX beaten CA



Number of loans from different seller quarterly: Some seller may sell 0 loan in a quarter

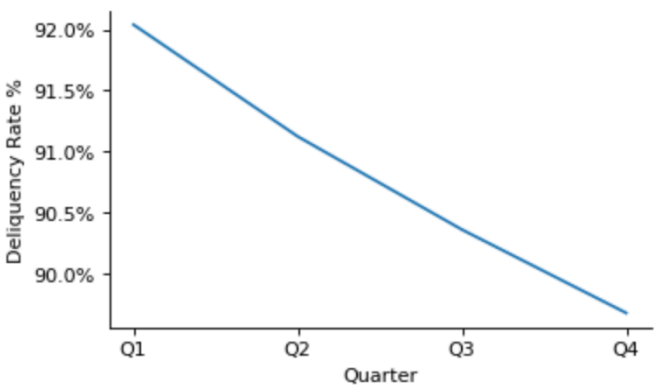


Performance file: svcg_2007.csv

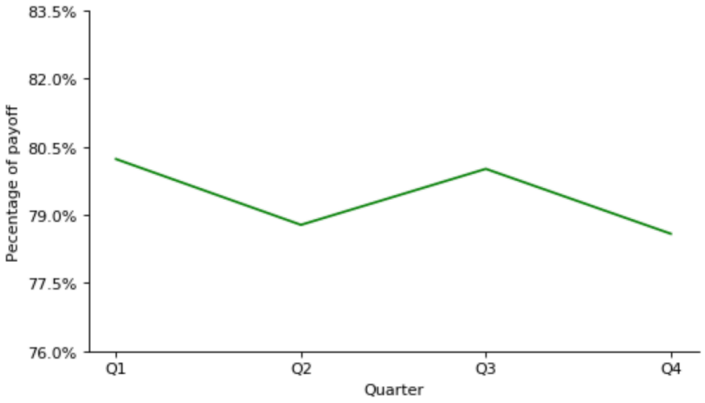
```
svcg_2007.head()
```

	loan_sequence_no	monthly_reporting_period	current_actual_upb	current_loan_delinquency_status
0	F107Q1000009	200702	110000.0	0
1	F107Q1000009	200703	110000.0	0
2	F107Q1000009	200704	110000.0	0
3	F107Q1000009	200705	110000.0	0
4	F107Q1000009	200706	110000.0	0

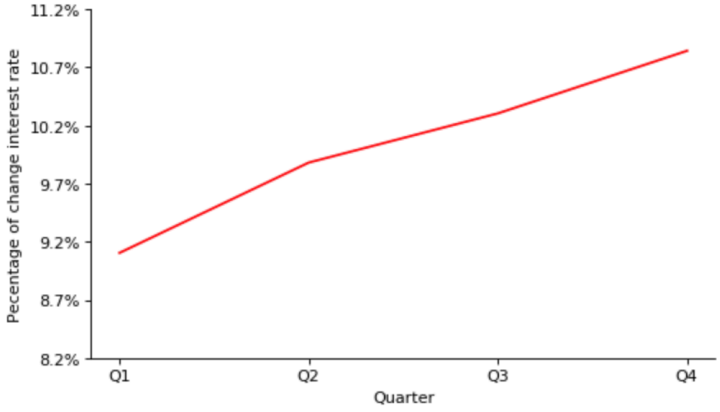
Percentage of delinquency status: decreasing but not significantly



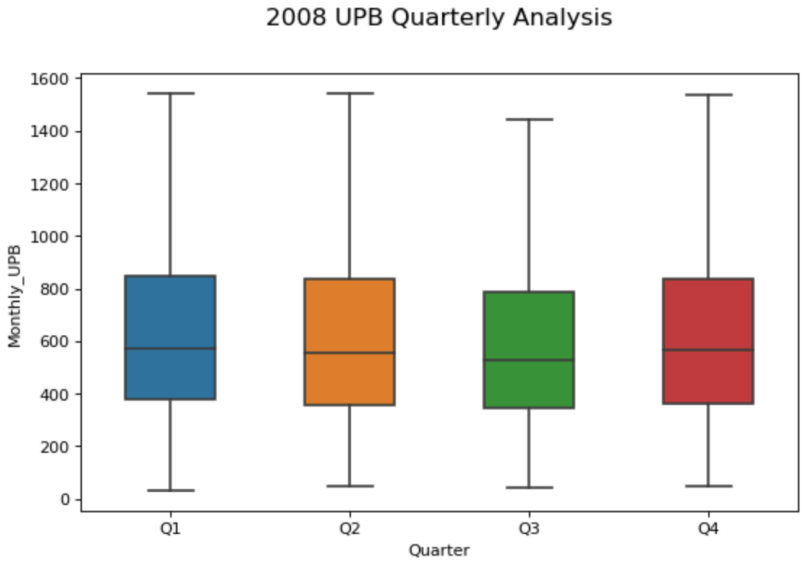
Percentage of loans which is paid off : stable between 79-81 percent



Percentage of loan which interest rate has changed: increasing

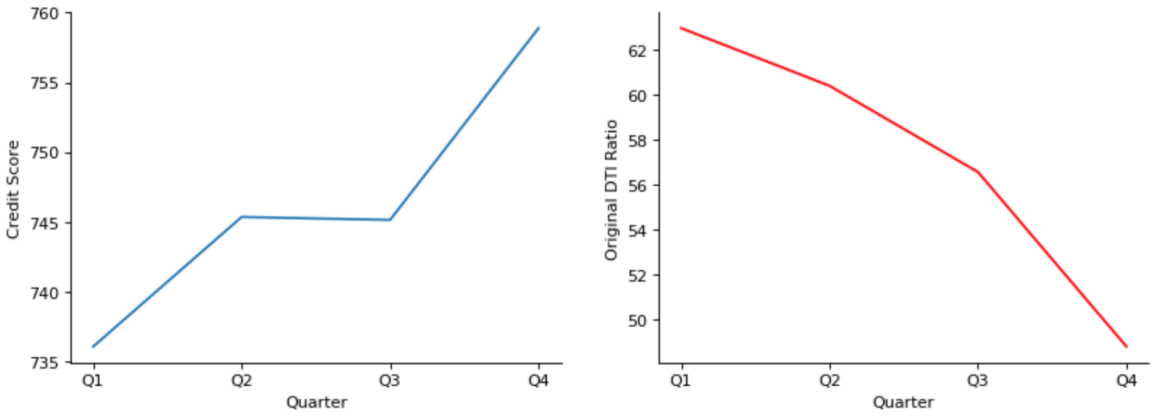


Quarterly Analysis 2008:



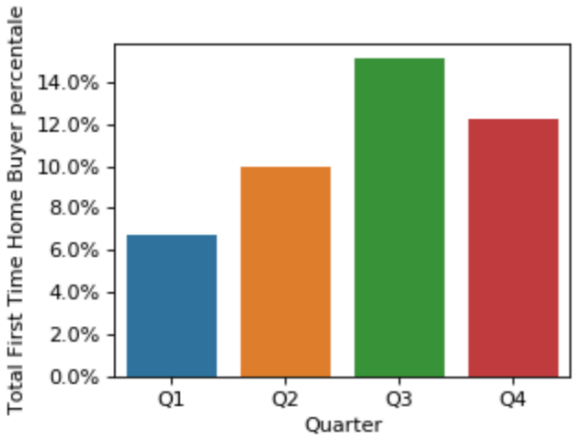
The un paid balance is stable for each quarter

Credit score and Original DTI Ratio:

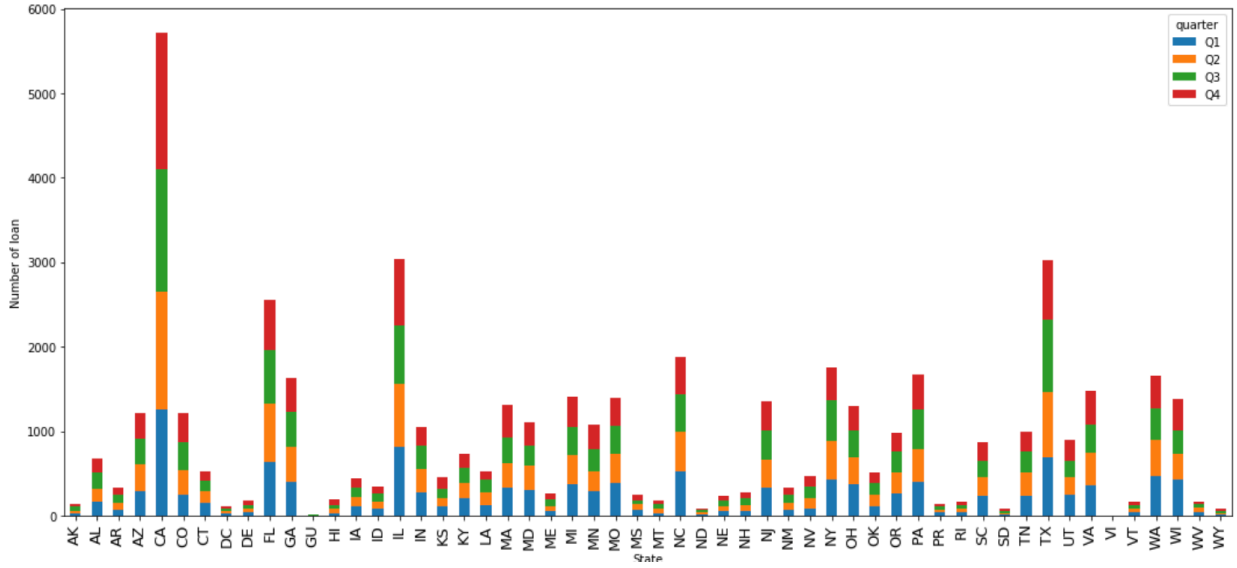


Different from 2007 the credit score in 2008 is increasing quarterly and the DTI ratio is decreasing so these two attribute may have negative correlation.

First time home buyer percentage each quarter:

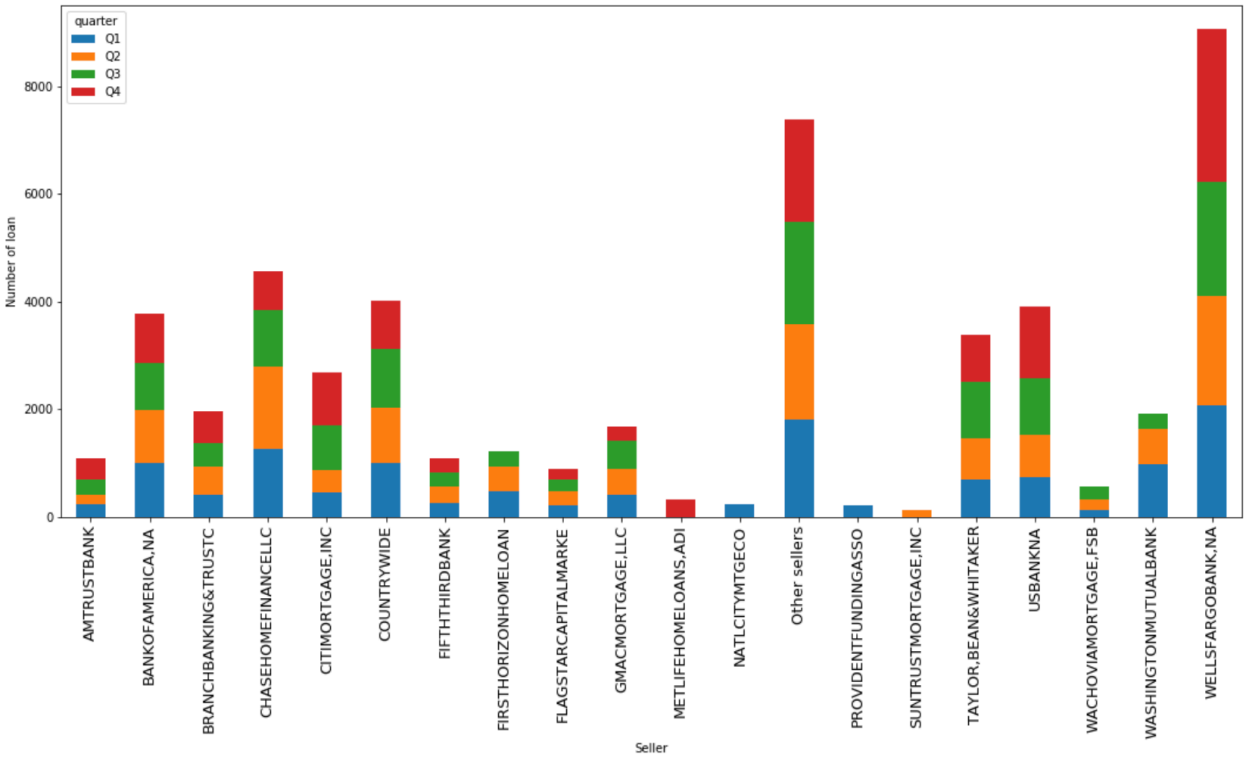


Same with 2007 the Q3 also has the most first time home buyers
Number of loans in each state quarterly:



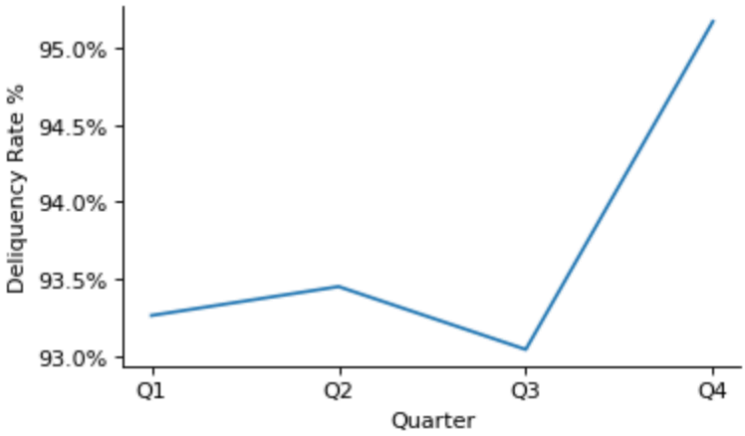
CA also the NO.1 state but in 2008, every quarter the CA have the largest loan number.

Number of loans from different seller quarterly:



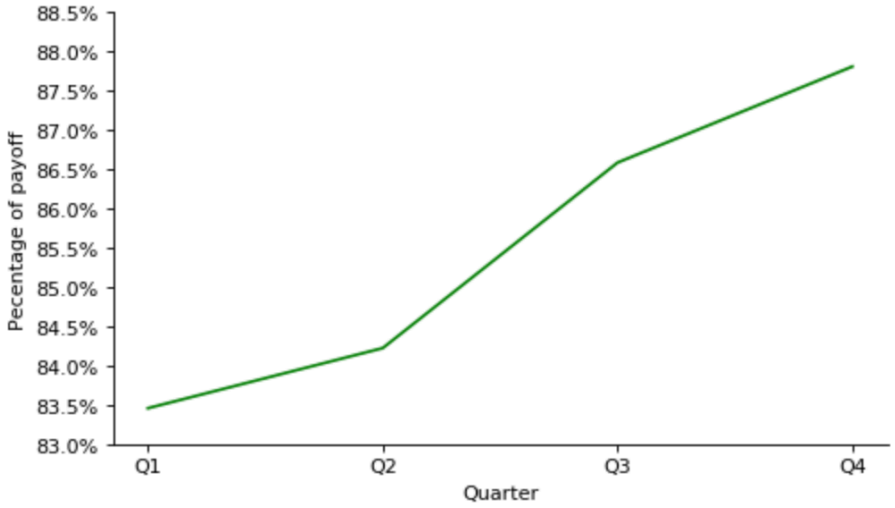
We can see that some seller have 0 sales in a quarter and Wells Fargo also like 2007 is the largest seller

Percentage of delinquency status:



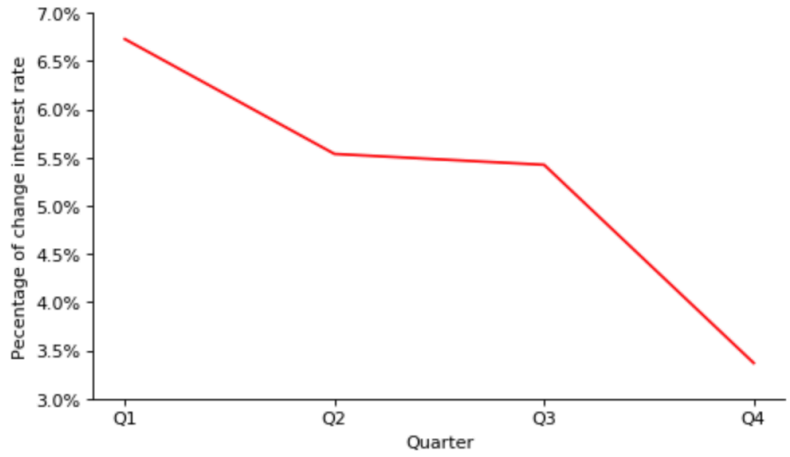
While Q3 has the lowest delinquency rate but it's still higher than the highest delinquency rate Q1 in 2007

Percentage of loans which is paid off :



As we can see the percentage of payoff is increasing and higher than 2007

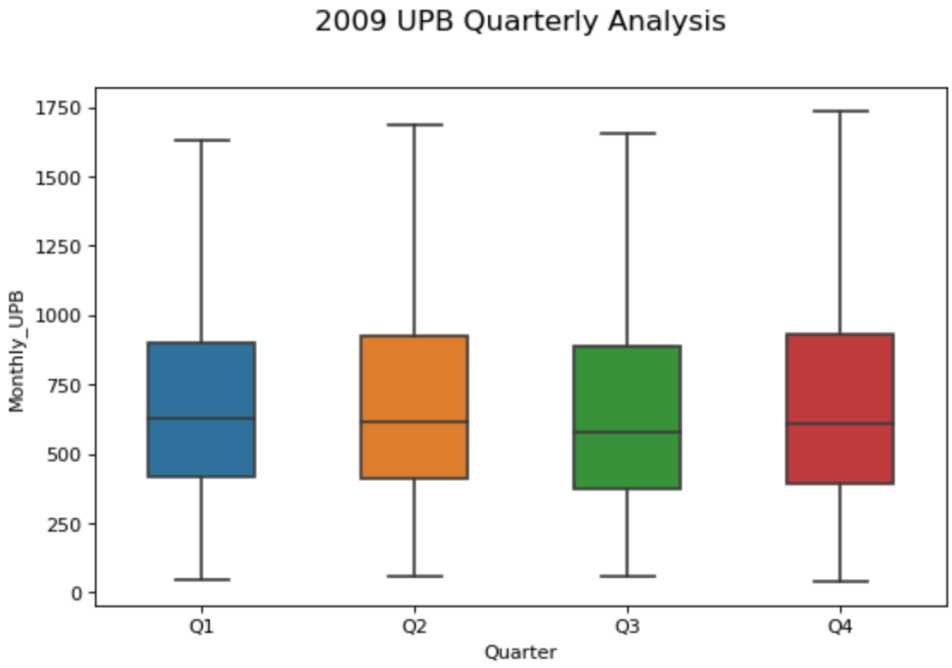
Percentage of loan which interest rate has changed:



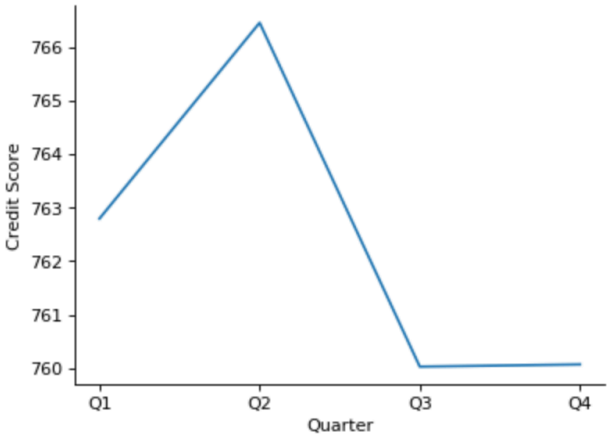
As we can see the changing interest rate number is decreasing from Q1-Q4 which is very different from 2007 and the peak value Q1 is lower than the bottom value Q1 in 2007.

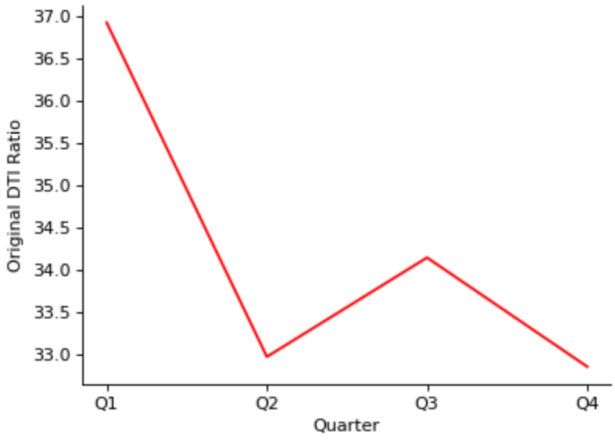
Quarterly Analysis 2009:

UPB Quarterly distribution: pretty the same



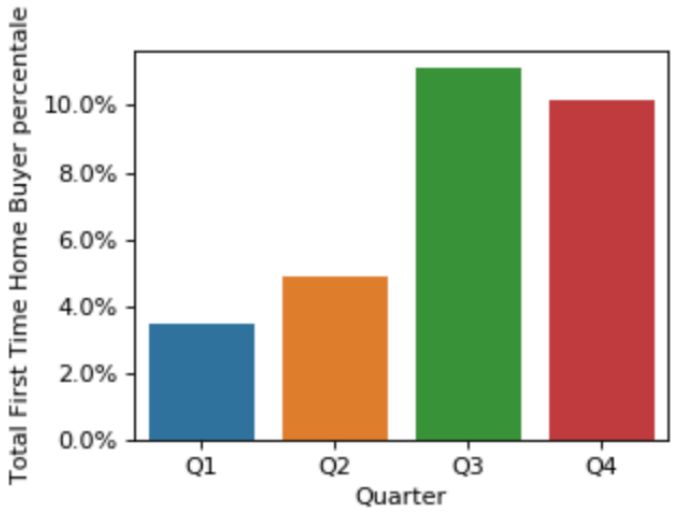
Credit score and Original DTI Ratio:





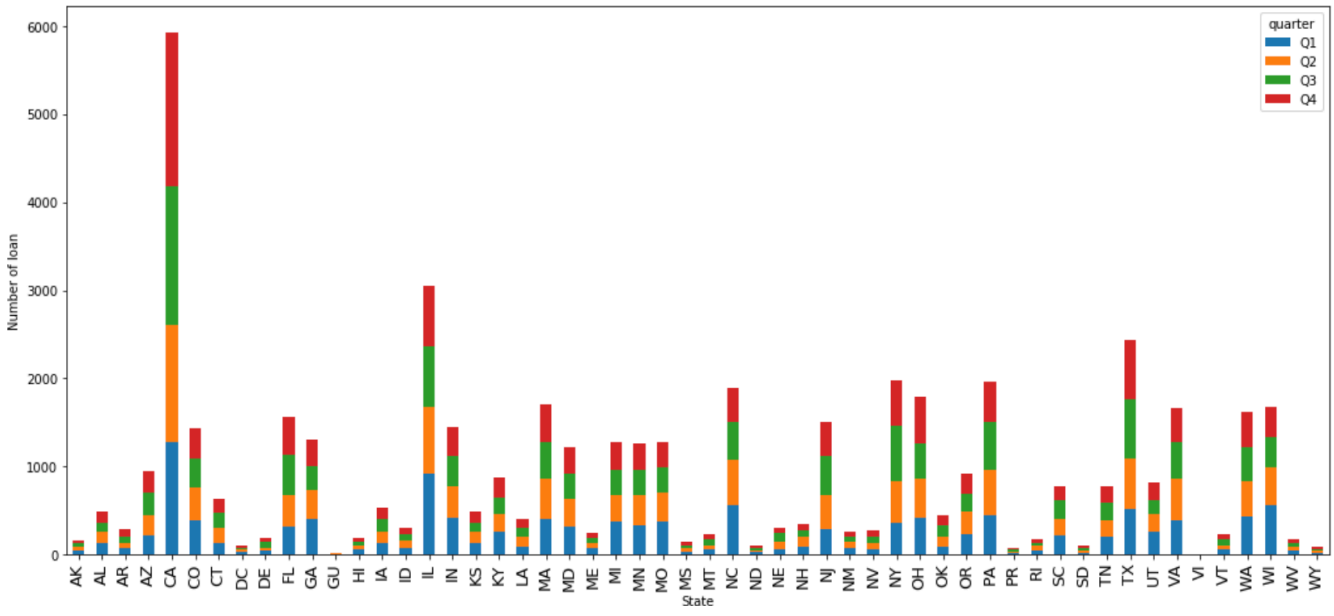
The credit score reached at the peak value at Q2 while the DTI reached the bottom value at Q2, and when the credit score increate the DTI ratio decrease. While Q1-Q2 the DTI ratio decreased significantly and the credit score not increased that significantly.

First time home buyer percentage each quarter:



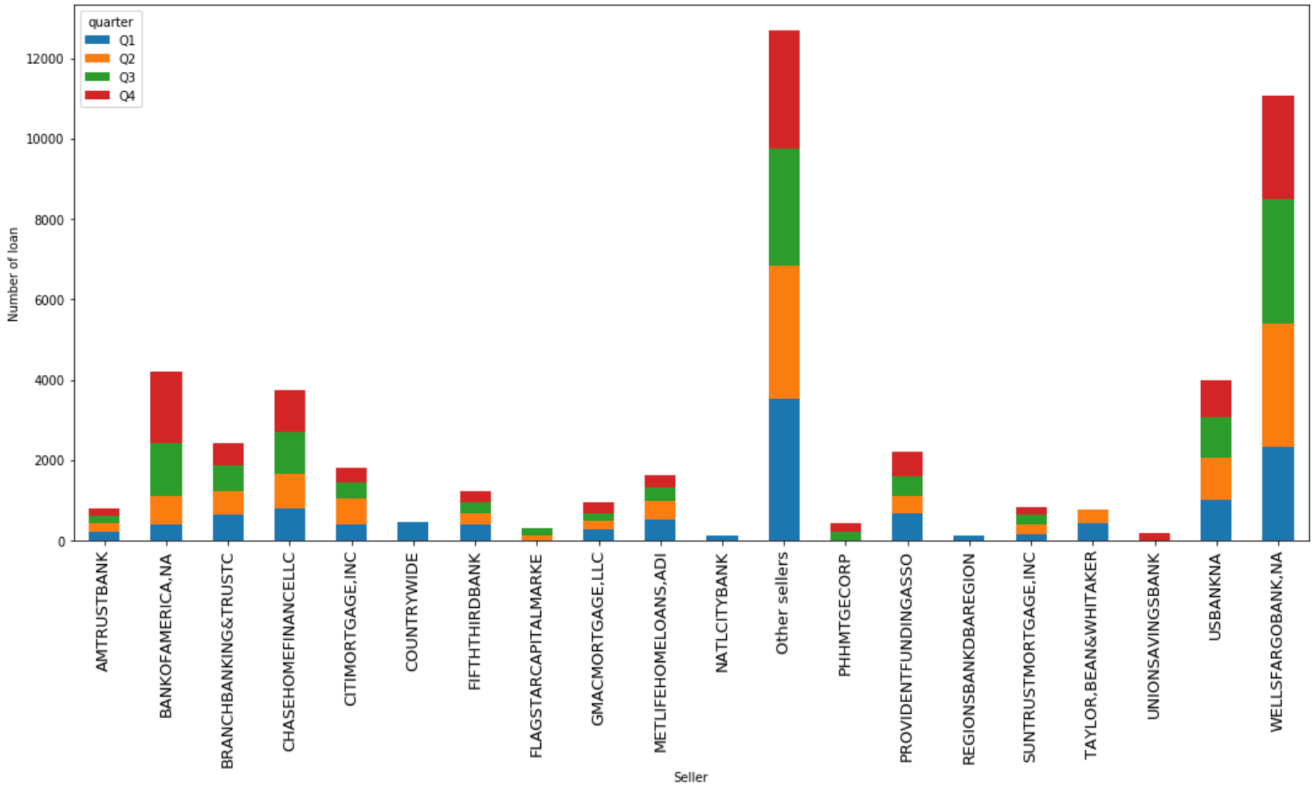
Same with 2007 and 2008, Q3 also has the most first time home buyers.

Number of loans in each state quarterly:



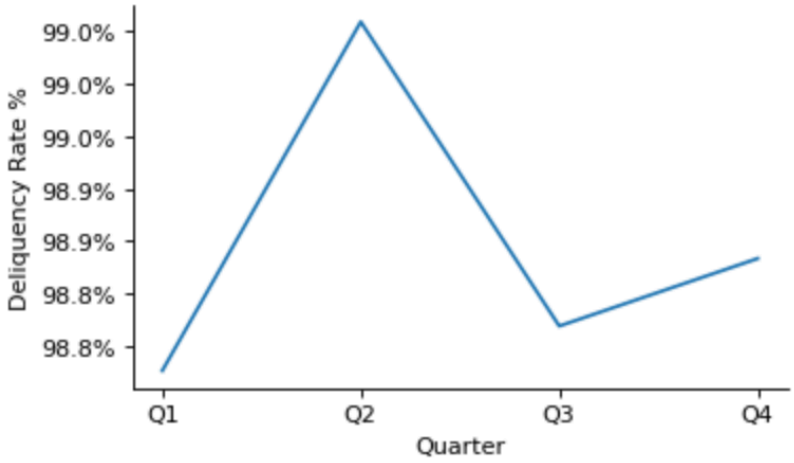
CA also the NO.1 state and every quarter the CA have the largest loan number.

Number of loans from different seller quarterly:



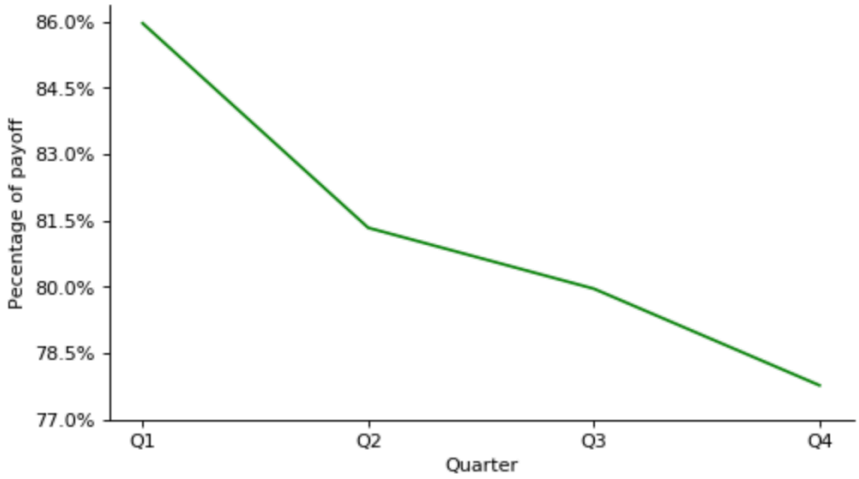
Different from 2007 and 2008 the largest seller is “other sellers” and for each quarter the other seller have the largest sales.

Percentage of delinquency status:



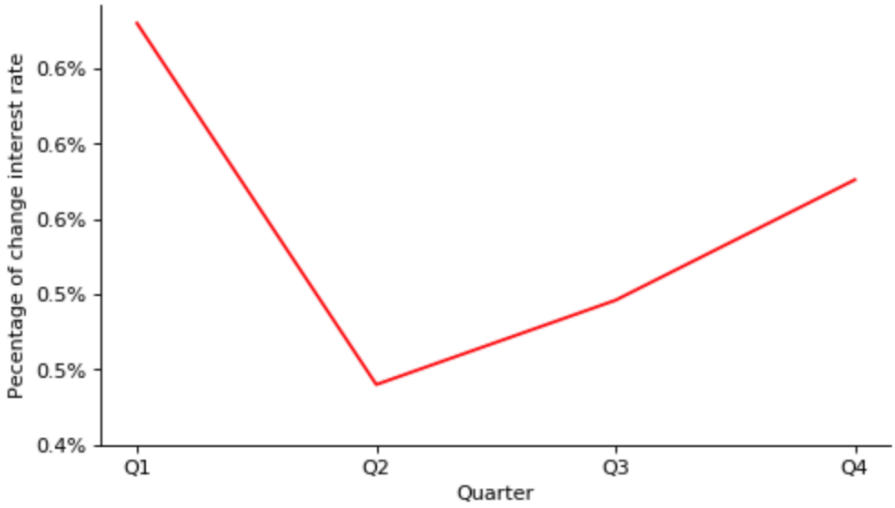
The delinquency rate is above all the 2008 and it change not very significantly only between 98-99 percent.

Percentage of loans which is paid off :



The payoff percentage is different form 2008. it's decreasing about 10 percent which is huge compared to 2007,2008.

Percentage of loan which interest rate has changed:



The percentage of loan changes is pretty low compared to 2007 and 2008. only 0.4-0.6 percent.

```
from sklearn.metrics import mean_absolute_error, mean_squared_error
from math import sqrt
def mean_absolute_percentage_error(y_true, y_pred):
    y_true, y_pred = np.array(y_true), np.array(y_pred)
    return np.mean(np.abs((y_true - y_pred) / y_true)) * 100

def printPerformance(pred):
    print(pred)
    print("RMSE: %.2f"
          % sqrt(mean_squared_error(y_test, pred)))
```

```
#What if there is a financial crisis...

df_train = pd.concat([df7,df72,df73,df74],axis = 0).sample(frac = 0.1)
df_test = pd.concat([df8,df72,df73,df74],axis = 0).sample(frac = 0.1)

#Pipline random forest
from sklearn.ensemble import RandomForestRegressor
randomForest = RandomForestRegressor(max_depth= 18, n_estimators = 16, random_state=2)

X_train = df_train[k_best_features]
y_train = df_train['0']
X_test = df_test[k_best_features]
y_test = df_test['0']

randomForest.fit(X_train,y_train)
randomForest.predict(X_test)

#Two Years Later..
X_test = df9[k_best_features]
y_test = df9['0']
randomForest.fit(X_train,y_train)
printPerformance(randomForest.predict(X_test))
```

Part II: Prediction

1. Download & pre-process the data :

Run a python script, in which includes the following major functions:

a.Data download. Parameters may passed to the function to download whatever file requested.

```
def get_data_from_url(quarter):
    print('downloading...')
    urllib.request.urlretrieve('https://freddiemac.embs.com/FLoan/Data/historical_data1_' + str(quarter) + '.zip',
                               DIR_NAME + str(quarter) + '.zip')
    # unzip_files(year)
    try:
        zip_ref = zipfile.ZipFile(DIR_NAME + str(quarter) + '.zip', 'r')
        zip_ref.extractall(DIR_NAME)
        zip_ref.close()

        write_into_csv(quarter)
    except zipfile.BadZipfile:
        print (zipfile.BadZipfile)
```

b. Data cleaning. The cleansed data is written to a csv file with renamed columns.

```
def data_cleaning():
    fileList = fileList = glob.glob('data/*.csv')
    for file in fileList:
        df = pd.read_csv(file,error_bad_lines=False)
        print('categorical cleaning...')
        for col in ['C','G','H','M','N','O','P','Q','T','U','V','W','X']:
            mode = pd.DataFrame(df.groupby(col).size().rename('cnt')).idxmax()[0]
            df[col] = df[col].fillna(mode)
        print('numerical cleaning...')
        for col in ['E','F','I','J','K','L','R']:
            dfmean = df[(df[col] != 999)|(df[col] != None)]
            mean = int(dfmean[col].mean(axis=0))
            df[col] = df[col].fillna(mean)

        filename = "%sclean.csv"%file[:-4]
        df.to_csv(filename)

        print('cleaning finished.')
```

c. Data processing. The columns are processed as dummy values to be used against prediction.
d. Main function as the entrance point. In this function the program logs in and and uses the defined function.

```
def start_execution():
    with requests.Session() as sess:
        sess.get(login_page_url);
        php_session_cookie = sess.cookies['PHPSESSID']
        login_payload = {'username' : USERNAME, 'password' : PASSWORD,'cookie':php_session_cookie}
        sess.post(login_page_url, data = login_payload)
        download_page_payload = {'accept': 'Yes', 'action': 'acceptTandC', 'acceptSubmit': 'Continue', 'cookie': php_session_cookie}
        sess.post(download_page_url, data=download_page_payload)
        create_directory(DIR_NAME)
#        create_csv()

        get_data_from_url(START)
        get_data_from_url("Q12003")
        get_data_from_url("Q12005")
        get_data_from_url("Q12007")
        get_data_from_url("Q12009")
        get_data_from_url(END)

        get_data_from_url("Q22005")
        get_data_from_url("Q32005")
        get_data_from_url("Q42005")

        get_data_from_url("Q22007")
        get_data_from_url("Q32007")
        get_data_from_url("Q42007")

        data_cleaning()
        preprocessing()
```

Please note that all output files are put into a data/ directory.

2. Local Analysis

According to:

```
selector = SelectKBest(f_regression, k=20).fit(X_train,y_train)
k_best_features = X_train.columns.values[selector.get_support()]
```

The program can select the best features for further analysis:

```
k_best_features = ['D', 'F', 'I', 'K', 'L', 'U', 'C_Y', 'H_I', 'H_P', 'Q_MH', 'T_P',
 'W_BANKOFAMERICA_NA', 'W_COUNTRYWIDE', 'W_GMACMTGECORP', 'W_NATLCITYMTGECO',
 'W_WASHINGTONMUTUALBANK', 'X_BANKOFAMERICA_NA', 'X_COUNTRYWIDE',
 'X_GMACMORTGAGE_LLC', 'X_NATLCITYMTGECO']
```

These features are used against the following required algorithms and functions.

```
#Linear, random forestk, and neural network models
from sklearn.linear_model import LinearRegression

linear = LinearRegression()
linear.fit(X_train,y_train)
linearPredict = linear.predict(X_test)
printPerformance(linearPredict)

from sklearn.ensemble import RandomForestRegressor
randomForest = RandomForestRegressor(max_depth= 18, n_estimators = 16, random_state=2)
randomForest.fit(X_train,y_train)
randomForestPredict = randomForest.predict(X_test)
printPerformance(randomForestPredict)

from sklearn.neural_network import MLPRegressor
neuralNetwork = MLPRegressor()
neuralNetwork.fit(X_train,y_train)
neuralNetworkPredict = neuralNetwork.predict(X_test)
printPerformance(neuralNetworkPredict)

# Build RF classifier to use in feature selection
from sklearn.ensemble import RandomForestClassifier
from mlxtend.feature_selection import SequentialFeatureSelector as sfs
clf = RandomForestClassifier(n_estimators=100, n_jobs=-1)
```

For each algorithm, the program can also provide stepwise and exhaustive feature selection approaches.

```
#Forward Selection
fs = sfs(clf,
         k_features=9,
         forward=False,
         floating=False,
         n_jobs=-1,
         verbose=2,
         scoring='neg_mean_absolute_error',
         cv=10)
fs.fit(X_train,y_train)
print('Best MAE score: %.2f' % fs.best_score_ * (-1))
print('Best subset:', fs.best_feature_names_)
#Backward Selection
bs = sfs(clf,
         k_features=9,
         forward=False
#Exhaustive Selection

from mlxtend.feature_selection import ExhaustiveFeatureSelector as efs
es = efs(clf,
         min_features=8,
         max_features=11,
         scoring='neg_mean_absolute_error',
         n_jobs=-1,
         print_progress=True,
         cv=8)
es.fit(X_train,y_train_encoded)
print('Best MAE score: %.2f' % efs.best_score_ * (-1))
print('Best subset:', efs.best_feature_names_)
```

…and then validates against each algorithm.

```
#Validation
from sklearn.model_selection import cross_val_score
scores = cross_val_score(linear, X_test, y_test, cv=3)
print(scores)
scores = cross_val_score(randomForest, X_test, y_test, cv=3)
print(scores)
scores = cross_val_score(neuralNetwork, X_test, y_test, cv=3)
print(scores)
```

3. AutoML Usages

a.Run a TPOT python script like:

```
#tpot
from tpot import TPOTRegressor

pipeline_optimizer = TPOTRegressor(generations=5, population_size=20, cv=5,
                                   random_state=42, verbosity=2)
pipeline_optimizer.fit(X_train, y_train)
print(pipeline_optimizer.score(X_test, y_test))
pipeline_optimizer.export('tpot_exported_pipeline.py')
```

…and it gives the following pipeline:

```
#Pipeline Exported
from sklearn.linear_model import LassoLarsCV
from sklearn.model_selection import train_test_split

# NOTE: Make sure that the class is labeled 'target' in the data file
tpot_data = df
features = tpot_data.drop('0', axis=1).values
training_features, testing_features, training_target, testing_target = \
            train_test_split(features, tpot_data['0'].values, random_state=42)

# Average CV score on the training set was:-6.254833533625527e-26
exported_pipeline = LassoLarsCV(normalize=True)

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

The program checks its performance by running the following function.

b. Run a H2o jupyter notebook. Please note that this step requires your credentials after mounting driverless AI in docker.

```
import h2o
from h2o.automl import H2OAutoML
h2o.init()

h2oq12005 = h2o.import_file("data/Q12005.csv")
h2oq22005 = h2o.import_file("data/Q22005.csv")

X = h2oq22005.columns.remove("0")
y = "0"
h2oML = H2OAutoML(max_models = 30, max_runtime_secs=300, seed = 1)
h2oML.train(x= x, y = y, training_frame= h2oq12005, leaderboard_frame= h2oq22005)

aml.leaderboard.as_data_frame()
```

4. What-If analysis:

For this part, read different data frames as training sets and see the performance of each.

```
#What if there is a economy boom...

df_train = pd.concat([df0,df2,df4,df6,df8,df10,df12],axis = 0).sample(frac = 0.1)

X_train = df_train[k_best_features].head(shape)
y_train = df_train['0'].tail(df_test.shape)
X_test = df12
randomForest.fit(X_train,y_train)
printPerformance(randomForest.predict(X_test))
```

```
#What if there is a regime change from election

X_train = df6[k_best_features].head(shape)
y_train = df6['0'].tail(df_test.shape)
X_test = df12[k_best_features]
y_test = df12['0']

randomForest.fit(X_train,y_train)
printPerformance(randomForest.predict(X_test))
```