diamond

March 18, 2022

[ ]:
```
from google.colab import drive
drive.mount('/content/drive')
import sys
import os
path_to_module = '/content/drive/MyDrive/'
sys.path.append(path_to_module)
os.chdir(path_to_module)
```

```
Mounted at /content/drive
```

[ ]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
diamonds = pd.read_csv('./diamonds.csv')
```

[ ]:
```
cut_num = {'Fair':1, 'Good':2, 'Very Good':3, 'Premium':4, 'Ideal':5}
for row in range(len(diamonds)):
  diamonds['cut'][row] = cut_num[diamonds['cut'][row]]
for row in range(len(diamonds)):
  diamonds['color'][row] = 75 - ord(diamonds['color'][row]) #J:1,..D:7
clarity_num = {'I1':1, 'SI2':2, 'SI1':3, 'VS2':4, 'VS1':5, 'VVS2':6, 'VVS1':7,␣
 ↪'IF':8}
for row in range(len(diamonds)):
  diamonds['clarity'][row] = clarity_num[diamonds['clarity'][row]]
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:3:
SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  This is separate from the ipykernel package so we can avoid doing imports
until
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:5:
SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
```

```
  """
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:8:
SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
```

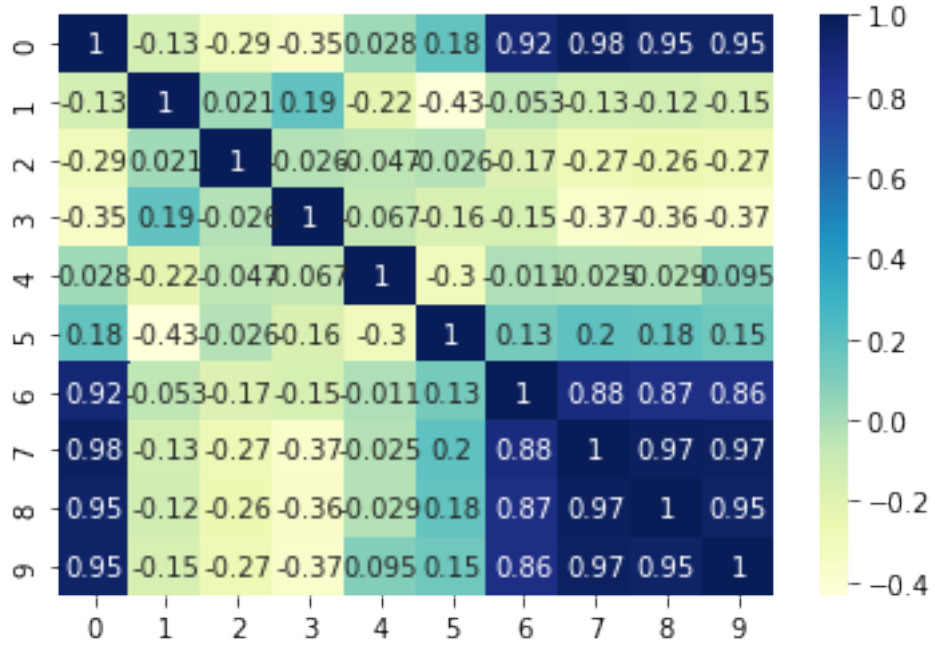
[ ]:
```
diamond_data = diamonds.to_numpy()[:, 1:].astype('float')
```

1 Q1

[ ]:
```
#normalize data
idx = [0, 4, 5, 6, 7, 8, 9]
diamond_data[:, idx] = (diamond_data[:, idx] - np.mean(diamond_data[:, idx],␣
 ↪axis=0)) / np.std(diamond_data[:, idx], axis=0)
```

2 Q2

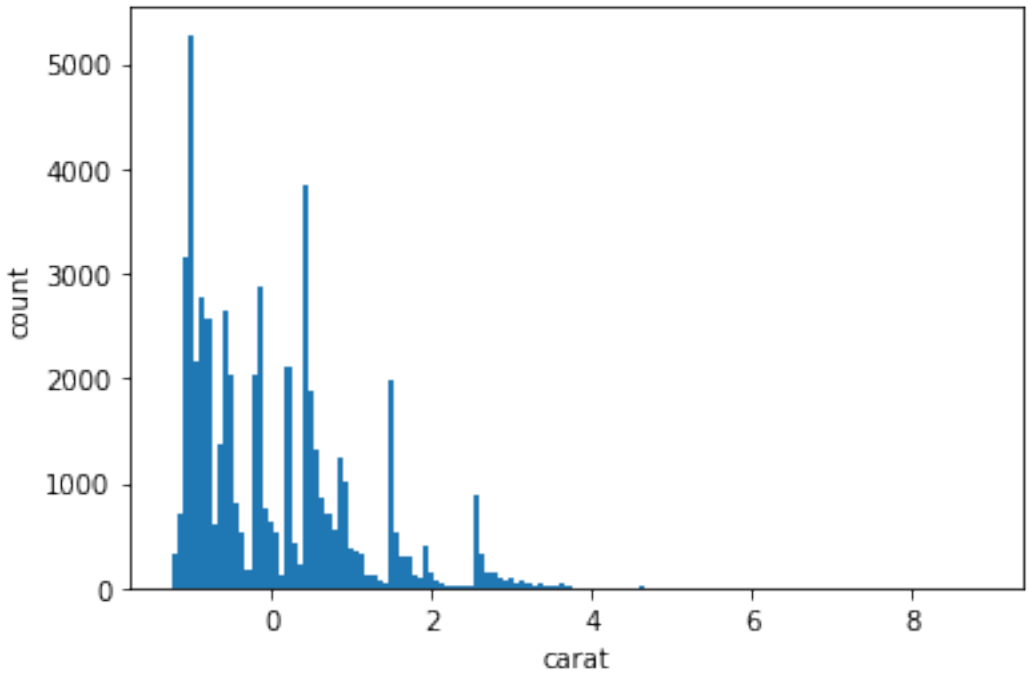
[ ]:
```
import seaborn as sb
corr = np.corrcoef(diamond_data.T)
dataplot = sb.heatmap(corr, cmap="YlGnBu", annot=True)
plt.show()
```

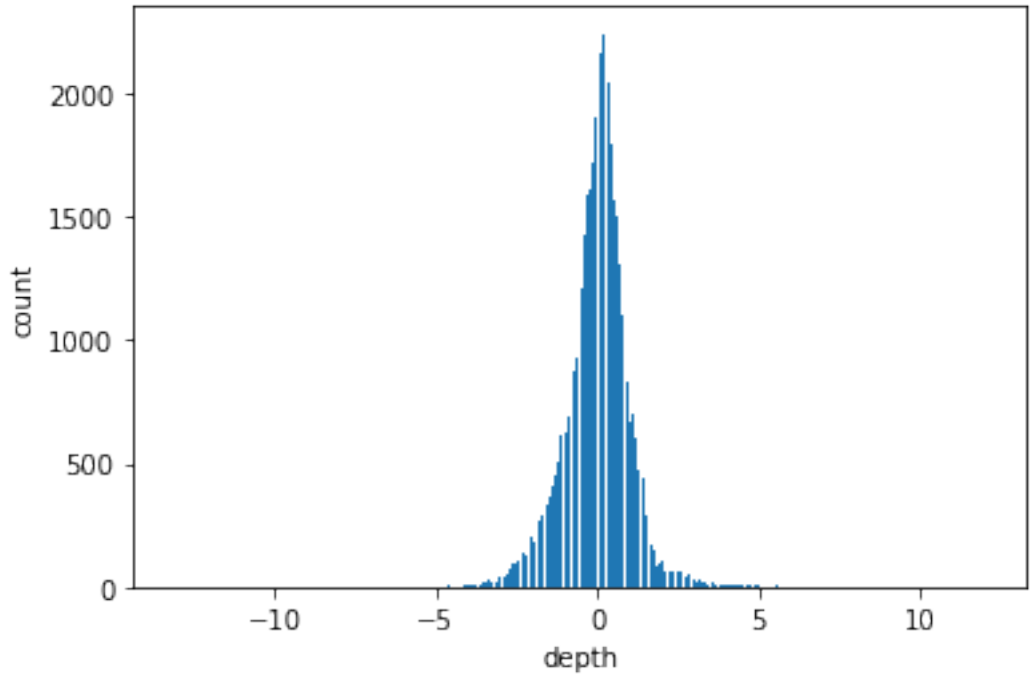


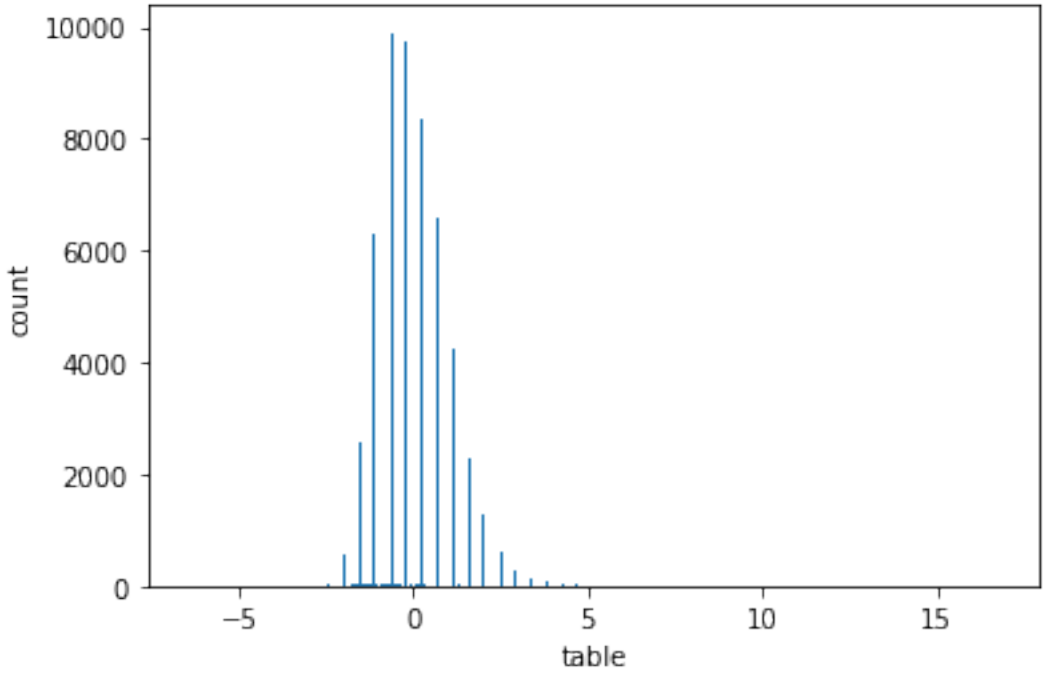
From the figure above, we can see that the price is relative to carat, x, y, and z. At the same time, we can see x, y, z, and carat are also highly relativeness, which makes sense since the size and the weight are proportional.

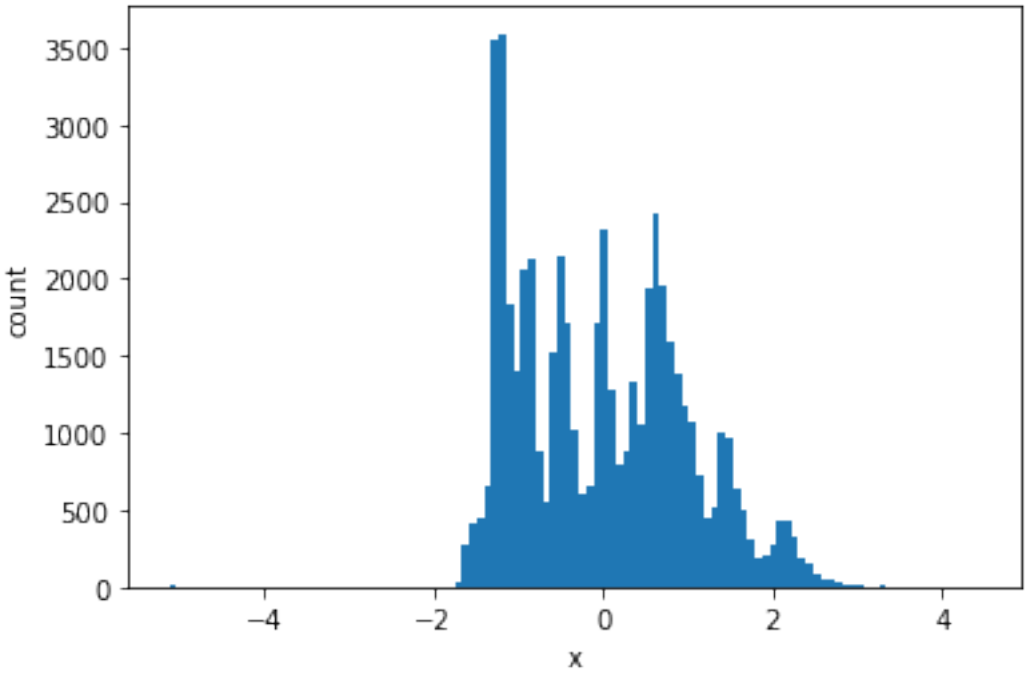
3 Q3

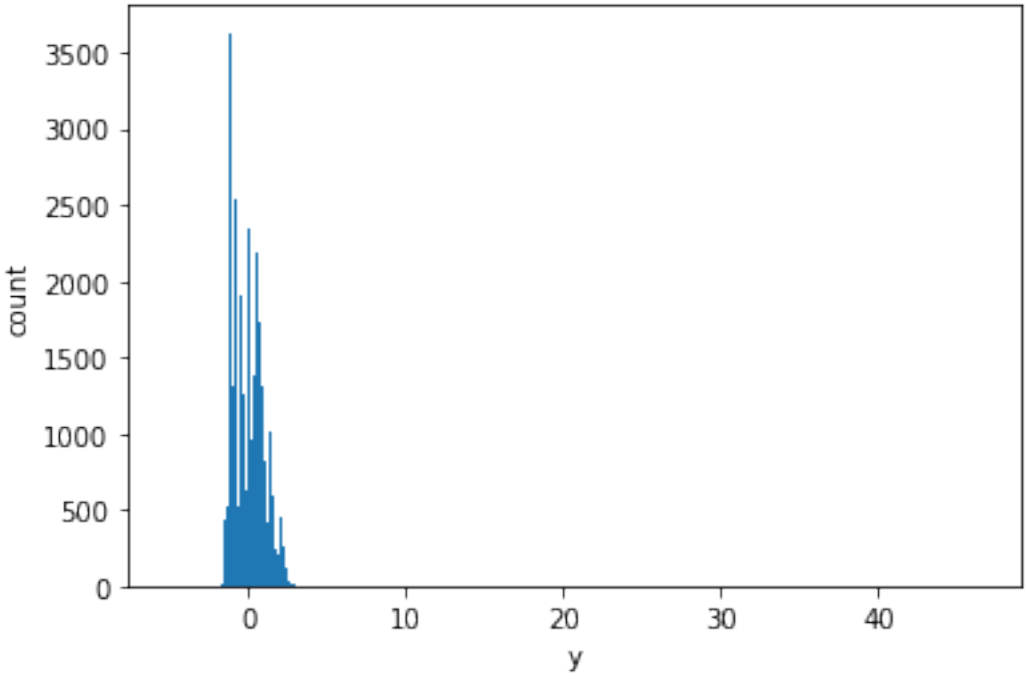
[ ]:
```
features = list(diamonds.columns)[1:]
for i in [0, 4, 5, 7, 8, 9]:
  plt.hist(diamond_data[:, i], bins='auto')
  plt.xlabel(features[i])
  plt.ylabel('count')
  plt.show()
```

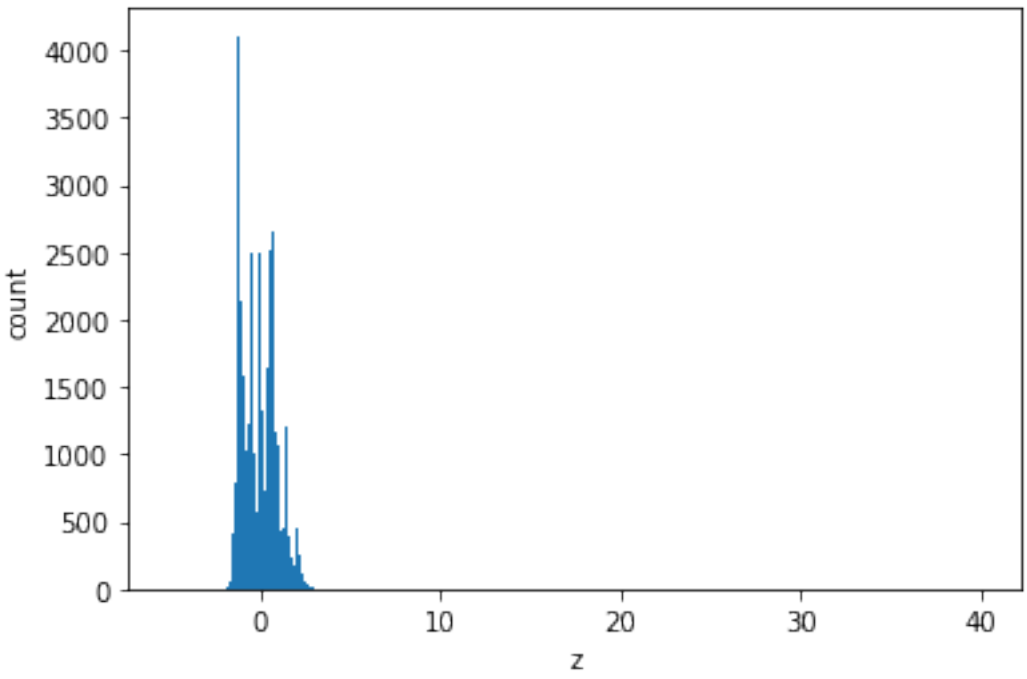



2000
1500
1000
500
0
count
−10
−5
0
5
10
depth


10000
8000
6000
4000
2000
0
count
−5
0
5
10
15
table


3500
3000
2500
2000
1500
1000
500
0
count
−4
−2
0
2
4
x

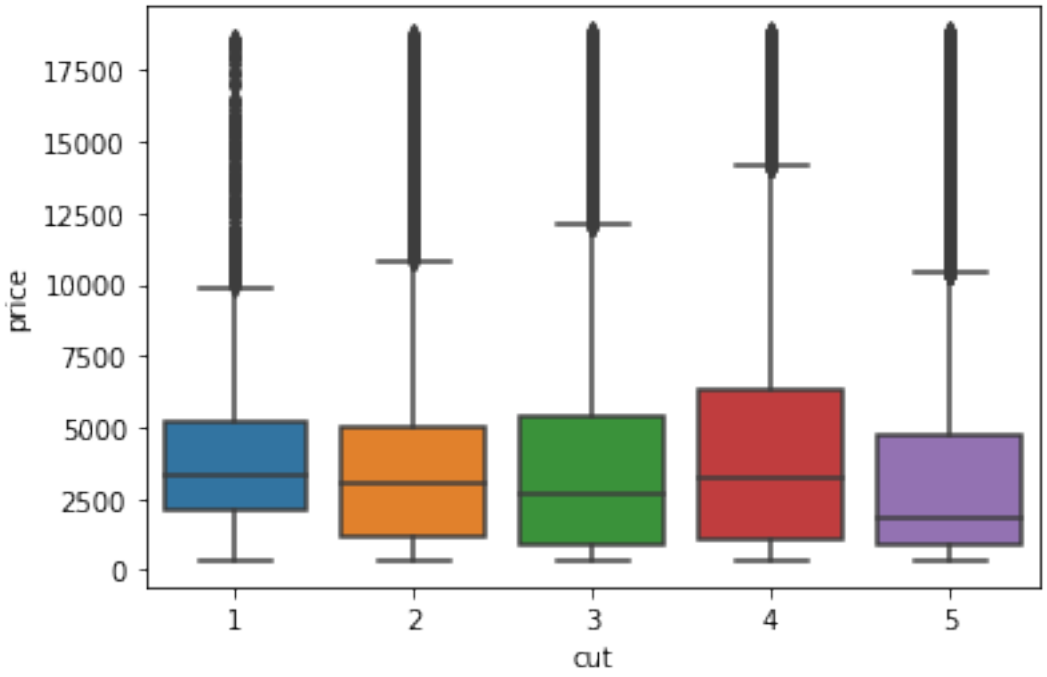

3500
3000
2500
2000
1500
1000
500
0
count
0
10
20
30
40
y

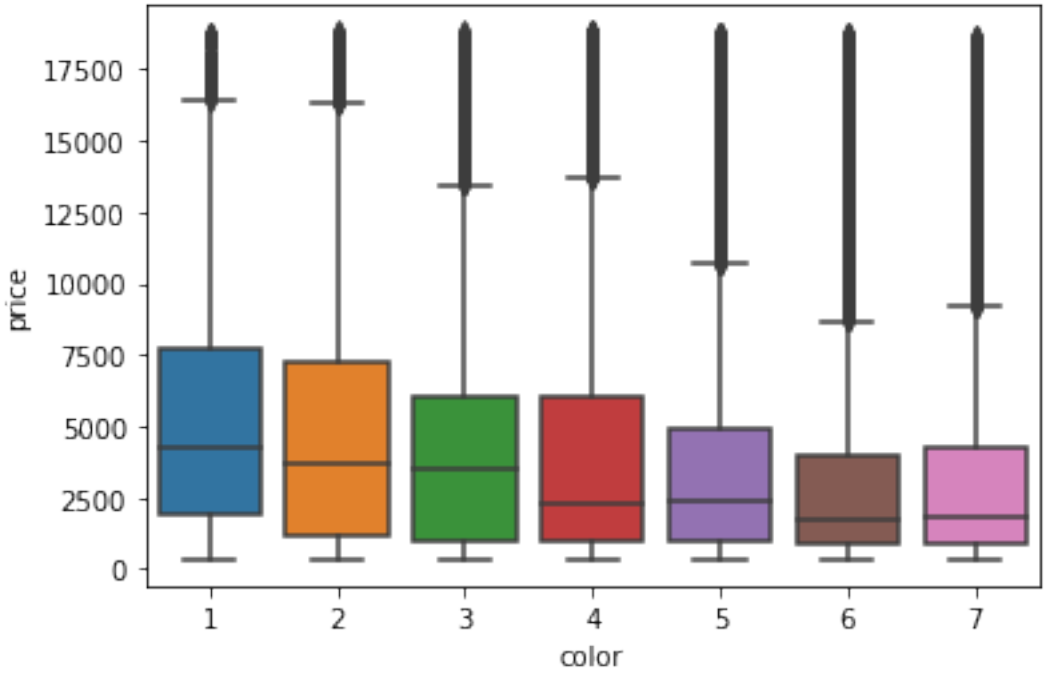


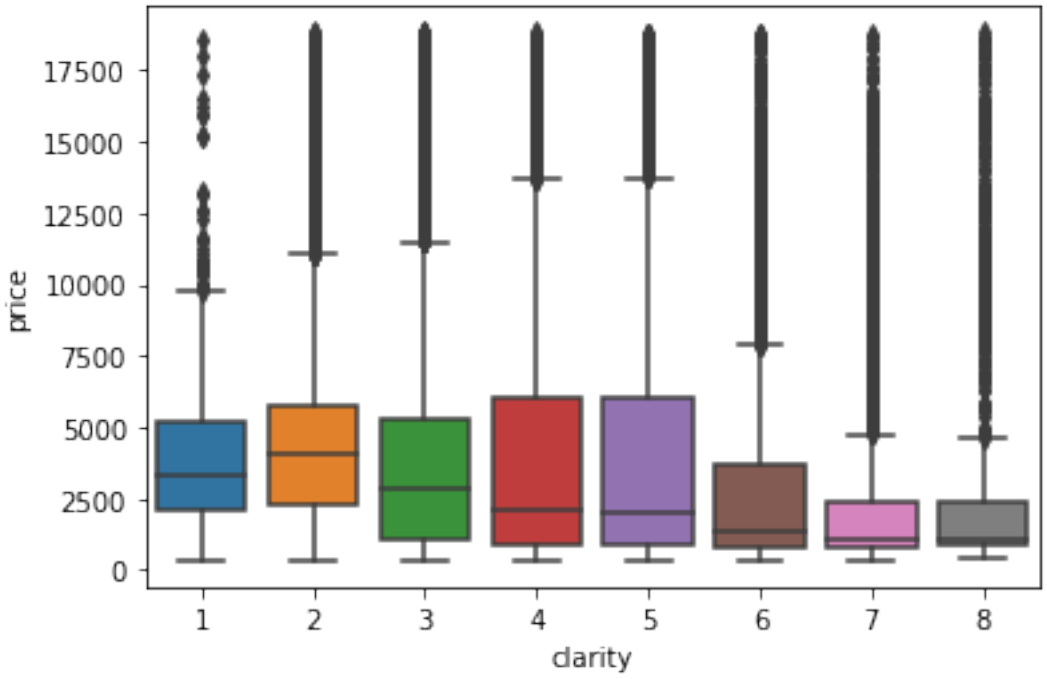
There are some methods for dealing with skew data. For instance, we can do log transformation, remove outliers, square root or Box Cox transformation.

4 Q4

[ ]:
```
for f in ['cut', 'color', 'clarity']:
  ax = sb.boxplot(x=f, y="price", data=diamonds)
  plt.show()
```


price
17500
15000
12500
10000
7500
5000
2500
0
1
2
3
4
5
cut

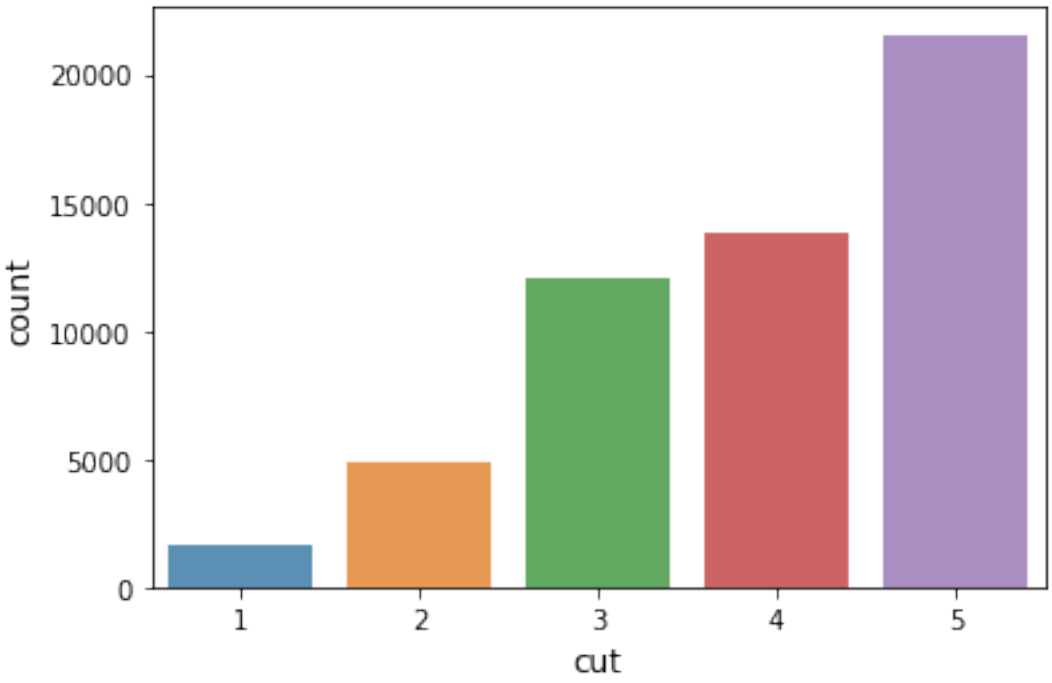

price
17500
15000
12500
10000
7500
5000
2500
0
1
2
3
4
5
6
7
color

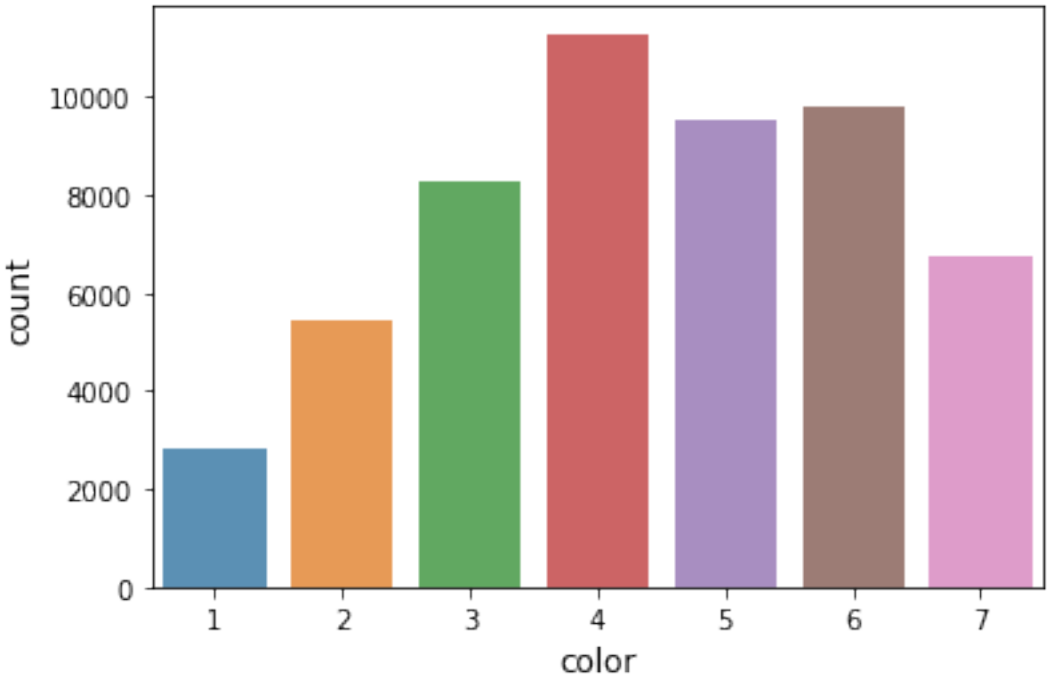


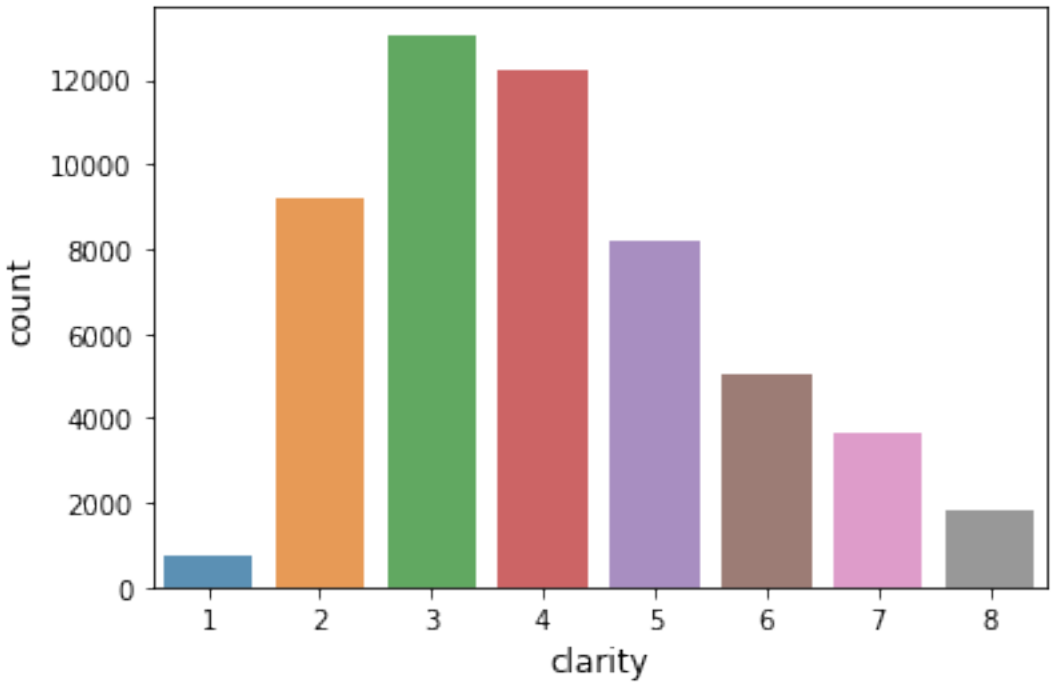
For the figure above, we can see that these features are not much relative to prices. We can also observe that from the heapmap about correlation by Q2.

5 Q5

```
[ ]: for f in ['cut', 'color', 'clarity']:
       histogram=sb.barplot(x=diamonds[f].value_counts().index, y=diamonds[f].
      ↪value_counts().values, alpha=0.8)
       plt.ylabel('count', fontsize=12)
       plt.xlabel(f, fontsize=12)
       plt.show()
```


20000
15000
10000
5000
0
count
1
2
3
4
5
cut


10000
8000
6000
4000
2000
0
count
1
2
3
4
5
6
7
color



6 Q7

[ ]:
```
from sklearn.feature_selection import mutual_info_regression
from sklearn.feature_selection import f_regression
ind = [i for i in range(10)]
ind.remove(6)
print('mutual_info_regression:{}'.format(mutual_info_regression(diamond_data[:,␣
↪ind], diamond_data[:, 6])))
print('F-scores:{}'.format(f_regression(diamond_data[:, ind], diamond_data[:,␣
↪6])[0]))
```

```
mutual_info_regression:[1.65217134 0.05764221 0.138603   0.21460453 0.03173889
0.03447966
 1.41298153 1.42125432 1.36028551]
F-scores:[3.04051487e+05 1.54784468e+02 1.65440124e+03 1.18800706e+03
 6.11586346e+00 8.86119363e+02 1.93741523e+05 1.60915662e+05
 1.54923267e+05]
```

[ ]:
```
#feature extraction
from sklearn.model_selection import train_test_split
diamond_x = diamond_data[:, [0,7,8,9]]
diamond_y = diamond_data[:, 6]
diamond_x_train, diamond_x_test, diamond_y_train, diamond_y_test =␣
↪train_test_split(diamond_x, diamond_y, test_size=0.2, random_state=42)
```

The rmse may decrease if we remove unimportant features since we can reduce overfitting risk.

7 Q8

linear : $\min_w ||xw - y||_2^2$

ridge : $\min_w ||xw - y||_2^2 + \alpha||w||_2^2$

lasso : $\min_w ||xw - y||_2^2 + \alpha||w||_1$

For ridge regression, the solution is more stable and the output is non-sparse. And ridge regression has analytical solution. As for lasso regression, the output could be sparse, that is to say, some features don't have influence on the model.

8 Q9

[ ]:
```
#linear regression without regularization
from sklearn.linear_model import LinearRegression, Ridge, Lasso
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import mean_squared_error

lrg = LinearRegression().fit(diamond_x_train, diamond_y_train)
err = mean_squared_error(lrg.predict(diamond_x_test), diamond_y_test)
print('rmse for linear regression without regularization={}'.format(err**0.5))
```

```
rmse for linear regression without regularization=0.38146412821849784
```

[ ]:
```
#Ridge regression
reg = {'alpha':[10**i for i in range(-5, 6)]}
rid = Ridge()
clf = GridSearchCV(estimator=rid, param_grid=reg, cv=10,␣
 ↪scoring='neg_root_mean_squared_error', n_jobs=-1)
clf.fit(diamond_x_train, diamond_y_train)

print("Best Parameters: "+str(clf.best_params_))
rid = Ridge(alpha=clf.best_params_['alpha']).fit(diamond_x_train,␣
 ↪diamond_y_train)
err = mean_squared_error(rid.predict(diamond_x_test), diamond_y_test)
print('rmse for linear regression with ridge regularization={}'.format(err**0.
 ↪5))
```

```
Best Parameters: {'alpha': 100}
rmse for linear regression with ridge regularization=0.38184414037141706
```

[ ]:
```
las = Lasso()
clf = GridSearchCV(estimator=las, param_grid=reg, cv=10,␣
 ↪scoring='neg_root_mean_squared_error', n_jobs=-1)
clf.fit(diamond_x_train, diamond_y_train)
```

```
print("Best Parameters: "+str(clf.best_params_))
las = Lasso(alpha=clf.best_params_['alpha']).fit(diamond_x_train,␣
 ↪diamond_y_train)
err = mean_squared_error(las.predict(diamond_x_test), diamond_y_test)
print('rmse for linear regression with Lasso regularization={}'.format(err**0.
 ↪5))
```

```
Best Parameters: {'alpha': 0.001}
rmse for linear regression with Lasso regularization=0.38193470320410033
```

9 Q10

If we don't apply regularization, then feature scaling would not influence the outcome. It would just influence on the weight of the model, but the optimization solution is the same. However, feature scaling makes influence on models with regularization. Because the weight term is in the objective function, we may sacrifice the bias to make weight smaller.

10 Q11

[ ]:
```
print('p-value:{}'.format(f_regression(diamond_data[:, ind], diamond_data[:,␣
 ↪6])[1]))
```

```
p-value:[0.00000000e+000 1.74601933e-035 0.00000000e+000 1.57172076e-257
 1.34004530e-002 3.76996315e-193 0.00000000e+000 0.00000000e+000
 0.00000000e+000]
```

While p-value is smaller, which means this feature is more important. P-value is the probability of obtaining test results at least as extreme as the results actually observed, under the assumption that the null hypothesis is correct.

11 Q12

By heatmap of correlation, the most salient feature is the carat. That makes sense since the price usually depends on the weight of the diamond.

12 Q13

[ ]:
```
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import PolynomialFeatures

pipeline = Pipeline([
    ('poly', PolynomialFeatures()),
    ('clf', None),
]
)
```

```
param_grid = [{
        "poly__degree":[i for i in range(1, 7)],
        "clf__alpha":[0] + [10**i for i in range(-5, 6)],
        "clf":(
            Ridge(fit_intercept=False),
            Lasso(fit_intercept=False),
        ),
    }]
```

[ ]:
```
cv = GridSearchCV(pipeline,cv=10,param_grid=param_grid,scoring='neg_root_mean_squared_error',n_jobs=-1)
cv.fit(diamond_x_train, diamond_y_train)
```

[ ]:
```
GridSearchCV(cv=10,
             estimator=Pipeline(steps=[('poly', PolynomialFeatures()),
                                       ('clf', None)]),
             n_jobs=-1,
             param_grid=[{'clf': (Ridge(fit_intercept=False),
                                  Lasso(alpha=0.001, fit_intercept=False)),
                          'clf__alpha': [0, 1e-05, 0.0001, 0.001, 0.01, 0.1, 1,
                                         10, 100, 1000, 10000, 100000],
                          'poly__degree': [1, 2, 3, 4, 5, 6]}],
             scoring='neg_root_mean_squared_error')
```

[ ]:
```
print(cv.best_params_)
```

```
{'clf': Lasso(alpha=0.001, fit_intercept=False), 'clf__alpha': 0.001,
'poly__degree': 1}
```

The best model is when degree is 1 and with Lasso regularization and parameter is 0.001.

Increasing the degree means the hypothesis set becomes larger, so we can fit training set better. However, this also increases the risk of overfitting and make performance on test set is bad.

Therefore, we can view the degree as a hyperparameter, and choose by cross validation.

13 Q14

[ ]:
```
lrg = LinearRegression().fit(diamond_x_train, diamond_y_train)
err = mean_squared_error(lrg.predict(diamond_x_test), diamond_y_test)
print('rmse for linear regression with original feature={}'.format(err**0.5))

poly = PolynomialFeatures(2, interaction_only=True)
diamond_x_train_poly = poly.fit_transform(diamond_x_train)
diamond_x_test_poly = poly.fit_transform(diamond_x_test)
```

```
lrg = LinearRegression(fit_intercept=False).fit(diamond_x_train_poly,␣
 ↪diamond_y_train)
err = mean_squared_error(lrg.predict(diamond_x_test_poly), diamond_y_test)
print('rmse for linear regression with features x_1*x_2={}'.format(err**0.5))
```

```
rmse for linear regression with original feature=0.38146412821849784
rmse for linear regression with features x_1*x_2=0.36508788894317895
```

The error decreases after we apply features x_1x_2. The reason is both features are proportional to the target and add these features could help us have more useful dimensions. At the same time, the number of features won't be too much and leads to overfitting.

14 Q15

[ ]:
```
from sklearn.neural_network import MLPRegressor
mlp = MLPRegressor().fit(diamond_x_train, diamond_y_train)
err = mean_squared_error(mlp.predict(diamond_x_test), diamond_y_test)
print('rmse for multi-layer perceptron={}'.format(err**0.5))
```

```
rmse for multi-layer perceptron=0.3396707136884523
```

Why does it do much better than linear regression? For linear regression, rmse values were observed to be approximately 0.38 for all regularization schemes. However, rmse from MLP is observed to be a smaller value of 0.34, and this is due to the non-linear relationship between the predictors and the outcome. Neural networks such as MLP will outperform linear regression in a complex dataset since it has multiple layers and conducts back propagation to deal with non-linearities.

15 Q16

[ ]:
```
param_mlp = {'hidden_layer_sizes':␣
 ↪[(100,),(200,200),(300,300,300),(300,300,200,100),(400,400,400,400)],
          'alpha': [10**i for i in range(-5, -2)], #weight decay
          }

mlp = MLPRegressor(early_stopping=True)
clf = GridSearchCV(estimator=mlp, param_grid=param_mlp, cv=3,␣
 ↪scoring='neg_root_mean_squared_error', n_jobs=-1, verbose=3)
clf.fit(diamond_x_train, diamond_y_train)

print("Best Parameters: "+str(clf.best_params_))
mlp = MLPRegressor(hidden_layer_sizes=clf.best_params_['hidden_layer_sizes'],␣
 ↪alpha=clf.best_params_['alpha']).fit(diamond_x_train, diamond_y_train)
err = mean_squared_error(mlp.predict(diamond_x_test), diamond_y_test)
print('rmse for multi-layer perceptron={}'.format(err**0.5))
```

```
Fitting 3 folds for each of 15 candidates, totalling 45 fits
Best Parameters: {'alpha': 0.0001, 'hidden_layer_sizes': (400, 400, 400, 400)}
```

```
rmse for multi-layer perceptron=0.33722512793402576
```

Best Parameters: {'alpha': 0.0001, 'hidden_layer_sizes': (400, 400, 400, 400)}

rmse for multi-layer perceptron=0.33722512793402576

16 Q17

[ ]:
```
param_mlp = {'hidden_layer_sizes': [(400,400,400,400)],
          'activation': ['identity','relu','tanh','logistic'],
          'alpha': [0.0001], #weight decay
          }

mlp = MLPRegressor(early_stopping=True)
clf = GridSearchCV(estimator=mlp, param_grid=param_mlp, cv=3,␣
 ↪scoring='neg_root_mean_squared_error', n_jobs=-1, verbose=10)
clf.fit(diamond_x_train, diamond_y_train)

print("Best Parameters: "+str(clf.best_params_))
mlp = MLPRegressor(hidden_layer_sizes=clf.best_params_['hidden_layer_sizes'],␣
 ↪activation=clf.best_params_['activation'], alpha=clf.best_params_['alpha']).
 ↪fit(diamond_x_train, diamond_y_train)
err = mean_squared_error(mlp.predict(diamond_x_test), diamond_y_test)
print('rmse for multi-layer perceptron={}'.format(err**0.5))
```

```
Fitting 3 folds for each of 4 candidates, totalling 12 fits
Best Parameters: {'activation': 'relu', 'alpha': 0.0001, 'hidden_layer_sizes':
(400, 400, 400, 400)}
rmse for multi-layer perceptron=0.33971334661540203
```

What activation function should be used for the output? From above, "ReLU" is the best activation function.

Best Parameters: {'activation': 'relu', 'alpha': 0.0001, 'hidden_layer_sizes': (400, 400, 400, 400)}

rmse for multi-layer perceptron=0.33971334661540203

17 Q18

What is the risk of increasing the depth of the network too far? Increasing depth of the network means increasing the number of parameters and therefore, may cause overfitting on training data and accuracy on test data may decrease.

18 Q19

[ ]:
```
from sklearn.ensemble import RandomForestRegressor

param_rfr = {"max_depth": [2,4,8,16, None],
```

```
            "n_estimators":[50,100,200,300,400,500],
            "max_features": [0.1, 0.25, 0.5, 0.75, 1.0],}

rfr = RandomForestRegressor()
clf = GridSearchCV(estimator=rfr, param_grid=param_rfr, cv=3,␣
 ↪scoring='neg_root_mean_squared_error', n_jobs=-1, verbose=10)
clf.fit(diamond_x_train, diamond_y_train)

print("Best Parameters: "+str(clf.best_params_))
rfr = RandomForestRegressor(max_depth=clf.best_params_['max_depth'],␣
 ↪n_estimators=clf.best_params_['n_estimators'], max_features=clf.
 ↪best_params_['max_features']).fit(diamond_x_train, diamond_y_train)
err = mean_squared_error(rfr.predict(diamond_x_test), diamond_y_test)
print('rmse for Random Forest Regressor={}'.format(err**0.5))
```

```
Fitting 3 folds for each of 150 candidates, totalling 450 fits
```

```
/usr/local/lib/python3.7/dist-
packages/joblib/externals/loky/process_executor.py:705: UserWarning: A worker
stopped while some jobs were given to the executor. This can be caused by a too
short worker timeout or by a memory leak.
  "timeout or by a memory leak.", UserWarning
```

```
Best Parameters: {'max_depth': 8, 'max_features': 0.75, 'n_estimators': 200}
rmse for multi-layer perceptron=0.33867940390322027
```

Best Parameters: {'max_depth': 8, 'max_features': 0.75, 'n_estimators': 200}

rmse for multi-layer perceptron=0.33867940390322027

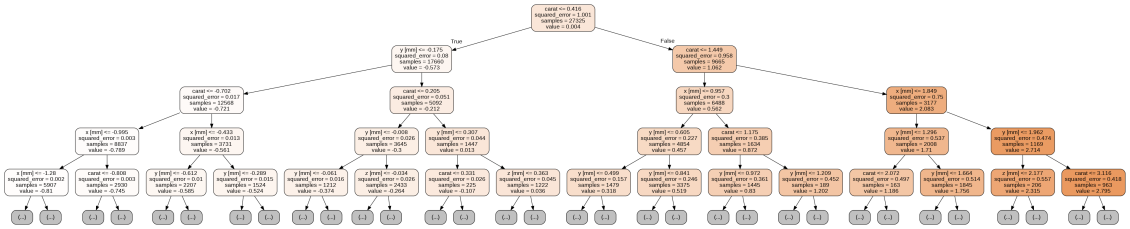
Explain how these hyper-parameters affect the overall performance? ***max_depth*** - The overall performance as measured by the RMSE seems to plateau when maximum depth=8-12 with a RMSE score of 0.337-0.338. Both increasing or decreasing the depth has implications of increasing the RMSE score as well. ***max_features*** - The impact of altering the maximum features on the RMSE score is less compared to the other hyperparameters. The best is at 75% of n_features however, changing this value has small impact. ***n_estimators*** - The impact of altering the number of estimators is also less compared to that of the maximum depth. When the classification is conducted with the best parameters as discovered above but with n_estimators altered to "5", the obtained RMSE score is 0.343. However, when n_estimators=100, the RMSE is 0.336 and when n_estimators=100, RMSE is also 0.336 and therefore it is observed that changing the n_estimators will not have much significance unless it takes a very small value. **Do some of them have regularization effect?** In overall, max_depth has a regularization effect since it is able to tune the complexity of the tree and adjust between under-fitting and over-fitting.

19 Q20

Why does random forest perform well? Random forest performs well with this dataset because high dimensional structured datasets with variables that individually contribute to prediction work better with random forest and they can also reduce the risks of overfitting.

20 Q21

[ ]:
```
from sklearn.tree import export_graphviz

estimator = rfr.estimators_[0]
export_graphviz(estimator,
                out_file='tree.dot',
                feature_names = ["carat", "x [mm]", "y [mm]" ,"z [mm]"],
                rounded = True, proportion = False,
                max_depth = 4,
                filled = True)

from subprocess import call
call(['dot', '-Tpng', 'tree.dot', '-o', 'tree.png'])

from IPython.display import Image
Image(filename = 'tree.png')
```

[ ]:


Which feature is selected for branching at the root node? At the root node, "carat" is selected as the branching feature.

What can you infer about the importance of features? While the feature at the root node is "carat," at depth 1 the selected features are 'y [mm]' and 'carat.' Furthermore, at depth 2 'x [mm]' and 'carat' is observed, and at depth 3, 'x[mm]', 'y[mm]', and 'carrat' is observed. The feature 'z [mm]' does not appear until depth 4 and therefore one could infer that 'carat' is the most salient feature which distinguishes the price of diamonds, with its proportions in the x and y dimensions being next, and z being the most undistinguishing feature of all.

21 Q22

[ ]:
```
#LightGBM
!pip install lightgbm
import lightgbm as lgb

lgbm = lgb.LGBMRegressor()
param_lgbm = {
    'max_depth': [2,4,8,16],
    'num_leaves': [2,4,8,16,32,64,128],
```

```
    "n_estimators":[10,50,100,200,300,400,500],

}

clf = GridSearchCV(estimator=lgbm, param_grid=param_lgbm, cv=3,␣
 ↪scoring='neg_root_mean_squared_error', n_jobs=-1, verbose=3, refit=True)
clf.fit(diamond_x_train, diamond_y_train)

print("Best Parameters: "+str(clf.best_params_))
lgbm = lgb.LGBMRegressor(max_depth=clf.best_params_['max_depth'],␣
 ↪num_leaves=clf.best_params_['num_leaves'], n_estimators=clf.
 ↪best_params_['n_estimators']).fit(diamond_x_train, diamond_y_train)
err = mean_squared_error(lgbm.predict(diamond_x_test), diamond_y_test)
print('rmse for LightGBM Regressor={}'.format(err**0.5))
```

```
---------------------------------------------------------------------------
KeyboardInterrupt                         Traceback (most recent call last)
<ipython-input-13-8e523a9e0618> in <module>()
     12
     13 clf = GridSearchCV(estimator=lgbm, param_grid=param_lgbm, cv=3,␣
 ↪scoring='neg_root_mean_squared_error', n_jobs=-1, verbose=3, refit=True)
---> 14 clf.fit(diamond_x_train, diamond_y_train)
     15
     16 print("Best Parameters: "+str(clf.best_params_))

/usr/local/lib/python3.7/dist-packages/sklearn/model_selection/_search.py in␣
 ↪fit(self, X, y, groups, **fit_params)
    889                 return results
    890
--> 891             self._run_search(evaluate_candidates)
    892
    893             # multimetric is determined here because in the case of a␣
 ↪callable

/usr/local/lib/python3.7/dist-packages/sklearn/model_selection/_search.py in␣
 ↪_run_search(self, evaluate_candidates)
   1390     def _run_search(self, evaluate_candidates):
   1391         """Search all candidates in param_grid"""
-> 1392         evaluate_candidates(ParameterGrid(self.param_grid))
   1393
   1394

/usr/local/lib/python3.7/dist-packages/sklearn/model_selection/_search.py in␣
 ↪evaluate_candidates(candidate_params, cv, more_results)
    849                     )
    850                     for (cand_idx, parameters), (split_idx, (train,␣
 ↪test)) in product(
```

```
--> 851                         enumerate(candidate_params), enumerate(cv.
 ↪split(X, y, groups))
    852                     )
    853                 )

/usr/local/lib/python3.7/dist-packages/joblib/parallel.py in __call__(self,␣
 ↪iterable)
   1054
   1055             with self._backend.retrieval_context():
-> 1056                 self.retrieve()
   1057             # Make sure that we get a last message telling us we are do e
   1058             elapsed_time = time.time() - self._start_time

/usr/local/lib/python3.7/dist-packages/joblib/parallel.py in retrieve(self)
    933             try:
    934                 if getattr(self._backend, 'supports_timeout', False):
--> 935                     self._output.extend(job.get(timeout=self.timeout))
    936                 else:
    937                     self._output.extend(job.get())

/usr/local/lib/python3.7/dist-packages/joblib/_parallel_backends.py in␣
 ↪wrap_future_result(future, timeout)
    540         AsyncResults.get from multiprocessing."""
    541         try:
--> 542             return future.result(timeout=timeout)
    543         except CfTimeoutError as e:
    544             raise TimeoutError from e

/usr/lib/python3.7/concurrent/futures/_base.py in result(self, timeout)
    428                 return self.__get_result()
    429
--> 430             self._condition.wait(timeout)
    431
    432             if self._state in [CANCELLED, CANCELLED_AND_NOTIFIED]:

/usr/lib/python3.7/threading.py in wait(self, timeout)
    294         try:    # restore state no matter what (e.g., KeyboardInterrupt
    295             if timeout is None:
--> 296                 waiter.acquire()
    297                 gotit = True
    298             else:

KeyboardInterrupt:
```

Best Parameters: {‘max_depth’: 8, ‘n_estimators’: 50, ‘num_leaves’: 32}

rmse for multi-layer perceptron=0.3384697445278097

[ ]:
```
#CatBoost
!pip install catboost
from catboost import CatBoostRegressor

ctb = CatBoostRegressor(thread_count=-1)
param_ctb = {
    'depth': [6,8,10],
    'l2_leaf_reg': [1,2,4],
    'random_strength':[0,1,2],
    'learning_rate': [0.01,0.03,0.1]

}

clf = GridSearchCV(estimator=ctb, param_grid=param_ctb, cv=3,␣
 ↪scoring='neg_root_mean_squared_error')
clf.fit(diamond_x_train, diamond_y_train)

print("Best Parameters: "+str(clf.best_params_))
ctb = CatBoostRegressor(depth=clf.best_params_['depth'], l2_leaf_reg=clf.
 ↪best_params_['l2_leaf_reg'], random_strength=clf.
 ↪best_params_['random_strength'],learning_rate=clf.
 ↪best_params_['learning_rate']).fit(diamond_x_train, diamond_y_train)
err = mean_squared_error(ctb.predict(diamond_x_test), diamond_y_test)
print('rmse for CatBoost Regressor={}'.format(err**0.5))
```

Best Parameters: {'depth': 10, 'l2_leaf_reg': 4, 'learning_rate': 0.03, 'random_strength': 2}

rmse for CatBoost Regressor=0.33763716231667684

22 Q23

[ ]:
```
#LightGBM
!pip install scikit-optimize
from skopt import BayesSearchCV
light = lgb.LGBMRegressor()
param_light = {
    'max_depth': [2,4,8,16],
    'num_leaves': [2,4,8,16,32,64,128],
    "n_estimators":[25,50,100,200,300,400,500],

}

clf = BayesSearchCV(estimator=light, search_spaces=param_light, cv=2,␣
 ↪scoring='neg_root_mean_squared_error', n_jobs=-1, verbose=3, refit=True)
clf.fit(diamond_x_train, diamond_y_train)

print("Best Parameters: "+str(clf.best_params_))
```

```
light = lgb.LGBMRegressor(max_depth=clf.best_params_['max_depth'],␣
 ↪num_leaves=clf.best_params_['num_leaves'], n_estimators=clf.
 ↪best_params_['n_estimators']).fit(diamond_x_train, diamond_y_train)
y_pred_test=light.predict(diamond_x_test)
RMSE = np.sqrt(mean_squared_error(y_pred_test, diamond_y_test))
print('rmse for LightGBM Regressor={}'.format(RMSE))
```

Best Parameters: [('max_depth', 8), ('n_estimators', 100), ('num_leaves', 32)]

rmse for LightGBM Regressor=0.3382200723070511

[ ]:
```
#CatBoost
ctb = CatBoostRegressor(thread_count=-1)
param_ctb = {
    'depth': [6,8,10],
    'l2_leaf_reg': [1,2,4],
    'random_strength':[0,1,2],
    'learning_rate': [0.01,0.03,0.1]

}

clf = BayesSearchCV(estimator=ctb, search_spaces=param_ctb, cv=3,␣
 ↪scoring='neg_root_mean_squared_error')
clf.fit(diamond_x_train, diamond_y_train)

print("Best Parameters: "+str(clf.best_params_))
ctb = CatBoostRegressor(depth=clf.best_params_['depth'], l2_leaf_reg=clf.
 ↪best_params_['l2_leaf_reg'], random_strength=clf.
 ↪best_params_['random_strength'],learning_rate=clf.
 ↪best_params_['learning_rate']).fit(diamond_x_train, diamond_y_train)
err = mean_squared_error(ctb.predict(diamond_x_test), diamond_y_test)
print('rmse for CatBoost Regressor={}'.format(err**0.5))
```

Best Parameters: [('depth', 8), ('l2_leaf_reg', 4), ('learning_rate', 0.03), ('random_strength', 1)]

rmse for CatBoost Regressor=0.3383022980011272

23 Q24

Comparing different parameter effects

[ ]:
```
light = lgb.LGBMRegressor(max_depth=2, n_estimators=100, num_leaves=32).
 ↪fit(diamond_x_train, diamond_y_train)
y_pred_test=light.predict(diamond_x_test)
RMSE = np.sqrt(mean_squared_error(y_pred_test, diamond_y_test))
print('rmse for LightGBM Regressor={}'.format(RMSE))
```

```
rmse for LightGBM Regressor=0.3443026547285279
```

[ ]:
```
light = lgb.LGBMRegressor(max_depth=8, n_estimators=100, num_leaves=32).
  ↪fit(diamond_x_train, diamond_y_train)
y_pred_test=light.predict(diamond_x_test)
RMSE = np.sqrt(mean_squared_error(y_pred_test, diamond_y_test))
print('rmse for LightGBM Regressor={}'.format(RMSE))
```

```
rmse for LightGBM Regressor=0.3382200723070511
```

[ ]:
```
light = lgb.LGBMRegressor(max_depth=8, n_estimators=50, num_leaves=32).
  ↪fit(diamond_x_train, diamond_y_train)
y_pred_test=light.predict(diamond_x_test)
RMSE = np.sqrt(mean_squared_error(y_pred_test, diamond_y_test))
print('rmse for LightGBM Regressor={}'.format(RMSE))
```

```
rmse for LightGBM Regressor=0.3384697445278097
```

[ ]:
```
light = lgb.LGBMRegressor(max_depth=8, n_estimators=10, num_leaves=32).
  ↪fit(diamond_x_train, diamond_y_train)
y_pred_test=light.predict(diamond_x_test)
RMSE = np.sqrt(mean_squared_error(y_pred_test, diamond_y_test))
print('rmse for LightGBM Regressor={}'.format(RMSE))
```

```
rmse for LightGBM Regressor=0.47040280093326436
```

[ ]:
```
light = lgb.LGBMRegressor(max_depth=8, n_estimators=100, num_leaves=4).
  ↪fit(diamond_x_train, diamond_y_train)
y_pred_test=light.predict(diamond_x_test)
RMSE = np.sqrt(mean_squared_error(y_pred_test, diamond_y_test))
print('rmse for LightGBM Regressor={}'.format(RMSE))
```

```
rmse for LightGBM Regressor=0.3440970935755289
```

Which of them helps with performance? There is subtle difference in the RMSE scores with LightGBM being 0.33822 and CatBoost being 0.33830. However, LightGBM is slightly better. In terms of hyperparameters, optimizing n_estimators has the most significant effect.

Which helps with regularization (shrinks the generalization gap)? LightGBM seems to be doing better with the generalization gap (as seen in Q25 below) the RMSE for training set for LightGBM is 0.3294786348038795 and for validation set, 0.34149909705262727. Thus, the difference is merely 0.012. On the contrast, CatBoost training set is 0.31499329017072764 and validation set is 0.3408280489012143, which makes the difference 0.026 and almost the twice of LightGBM.

Which affects the fitting efficiency?

Additionally, the runtime of CatBoost is orders of magnitudes larger than LightGBM however, its performance on categorical data is better since it works on smaller subsets to reduce overfitting. On the other hand, LightGBM is much faster and can handle a large dataset but requires data to be pre-processed and is prone to overfitting.

24 Q25

[ ]:
```
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_validate
from numpy import absolute
from numpy import mean
from numpy import sqrt

kf = KFold(n_splits=10, random_state=1, shuffle=True)
```

[ ]:
```
# Linear Regression
lrg = LinearRegression()
scores = cross_validate(lrg, diamond_x_train, diamond_y_train,␣
 ↪scoring='neg_root_mean_squared_error', cv=kf, n_jobs=-1,␣
 ↪return_train_score=True)
train_rmse = mean(absolute(scores['train_score']))
test_rmse = mean(absolute(scores['test_score']))
print('rmse for Linear Regression train set={}'.format(train_rmse))
print('rmse for Linear Regression validation set={}'.format(test_rmse))
```

```
rmse for Linear Regression train set=0.3820919523598028
rmse for Linear Regression validation set=0.38482808422746334
```

[ ]:
```
# Polynomial Regression
poly = PolynomialFeatures(degree=1)
diamond_x_train_poly = poly.fit_transform(diamond_x_train)
diamond_x_test_poly = poly.fit_transform(diamond_x_test)
lrg = Lasso(alpha=0.001, fit_intercept=False).fit(diamond_x_train_poly,␣
 ↪diamond_y_train)
scores = cross_validate(lrg, diamond_x_train_poly, diamond_y_train,␣
 ↪scoring='neg_root_mean_squared_error', cv=kf, n_jobs=-1,␣
 ↪return_train_score=True)
train_rmse = mean(absolute(scores['train_score']))
test_rmse = mean(absolute(scores['test_score']))
print('rmse for Polynomial Regression train set={}'.format(train_rmse))
print('rmse for Polynomial Regression validation set={}'.format(test_rmse))
```

```
rmse for Polynomial Regression train set=0.3822987133542557
rmse for Polynomial Regression validation set=0.3839580364498527
```

[ ]:
```
# Neural Network
mlp = MLPRegressor(early_stopping=True, activation='relu', alpha=0.0001,␣
 ↪hidden_layer_sizes=(400,400,400,400))
scores = cross_validate(mlp, diamond_x_train, diamond_y_train,␣
 ↪scoring='neg_root_mean_squared_error', cv=kf, n_jobs=-1,␣
 ↪return_train_score=True)
train_rmse = mean(absolute(scores['train_score']))
test_rmse = mean(absolute(scores['test_score']))
```

```
print('rmse for Neural Network train set={}'.format(train_rmse))
print('rmse for Neural Network validation set={}'.format(test_rmse))
```

```
rmse for Random Forest Regressor train set=0.3458847193914507
rmse for Random Forest Regressor validation set=0.3477865034834301
```

[ ]:
```
# Random Forest
rfr = RandomForestRegressor(max_depth=8, max_features=0.75, n_estimators=200,␣
 ↪oob_score=True, warm_start=True)
scores = cross_validate(rfr, diamond_x_train, diamond_y_train,␣
 ↪scoring='neg_root_mean_squared_error', cv=kf, n_jobs=-1,␣
 ↪return_train_score=True, return_estimator=True)
train_rmse = mean(absolute(scores['train_score']))
test_rmse = mean(absolute(scores['test_score']))
print('rmse for Random Forest Regressor train set={}'.format(train_rmse))
print('rmse for Random Forest Regressor validation set={}'.format(test_rmse))
```

```
rmse for Random Forest Regressor train set=0.3293121712291798
rmse for Random Forest Regressor validation set=0.3417309236664362
```

[ ]:
```
oob_error = 1-np.mean([i.oob_score_ for i in scores['estimator']])
print('Out of Bag Error={}'.format(oob_error))
```

```
Out of Bag Error=0.11679711928542724
```

[ ]:
```
# LightGBM
lgbm = lgb.LGBMRegressor(max_depth=8, num_leaves=32, n_estimators=50)
scores = cross_validate(lgbm, diamond_x_train, diamond_y_train,␣
 ↪scoring='neg_root_mean_squared_error', cv=kf, n_jobs=-1,␣
 ↪return_train_score=True)
train_rmse = mean(absolute(scores['train_score']))
test_rmse = mean(absolute(scores['test_score']))
print('rmse for LightGBM train set={}'.format(train_rmse))
print('rmse for LightGBM validation set={}'.format(test_rmse))
```

```
rmse for LightGBM train set=0.3294786348038795
rmse for LightGBM validation set=0.34149909705262727
```

[ ]:
```
# CatBoost
ctb = CatBoostRegressor(thread_count=-1, depth=10, l2_leaf_reg=4,␣
 ↪learning_rate=0.03, random_strength=2)
scores = cross_validate(ctb, diamond_x_train, diamond_y_train,␣
 ↪scoring='neg_root_mean_squared_error', cv=kf, n_jobs=-1,␣
 ↪return_train_score=True)
train_rmse = mean(absolute(scores['train_score']))
test_rmse = mean(absolute(scores['test_score']))
print('rmse for CatBoost train set={}'.format(train_rmse))
print('rmse for CatBoost validation set={}'.format(test_rmse))
```

```
rmse for CatBoost train set=0.31499329017072764
rmse for CatBoost validation set=0.3408280489012143
```

Why is the training RMSE different from that of validation set? The regression model is designed to fit the dataset and therefore the model is created in order to minimize the training RMSE as where the test RMSE is the result of fitting the model using the training set. Therefore, the training RMSE can be much smaller than the test RMSE, which would indicate overfitting.

25 Q26

From Q25, Out of Bag Error=0.11679711928542724.

Explain what OOB error and R2 score means. Out of Bag Error measures the averaged prediction error for random forests calculated using tree predictions that don't contain a training sample in their bootstrap sample. The R2 score is the regression score function between the predicted and target sets.

[ ]:

gas

March 18, 2022

[ ]:
```
from google.colab import drive
drive.mount('/content/drive')
import sys
import os
path_to_module = '/content/drive/MyDrive/gas'
sys.path.append(path_to_module)
os.chdir(path_to_module)
```

```
Mounted at /content/drive
```

[ ]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
gt_2011 = pd.read_csv('./gt_2011.csv')
gt_2012 = pd.read_csv('./gt_2012.csv')
gt_2013 = pd.read_csv('./gt_2013.csv')
gt_2014 = pd.read_csv('./gt_2014.csv')
gt_2015 = pd.read_csv('./gt_2015.csv')
features = gt_2011.columns
```

[ ]:
```
#choose CO2
gt_2011 = gt_2011.to_numpy()[:,:-1].astype('float')
gt_2011 = np.concatenate((np.ones((len(gt_2011), 1)), gt_2011), axis=1)
gt_2012 = gt_2012.to_numpy()[:,:-1].astype('float')
gt_2012 = np.concatenate((2*np.ones((len(gt_2012), 1)), gt_2012), axis=1)
gt_2013 = gt_2013.to_numpy()[:,:-1].astype('float')
gt_2013 = np.concatenate((3*np.ones((len(gt_2013), 1)), gt_2013), axis=1)
gt_2014 = gt_2014.to_numpy()[:,:-1].astype('float')
gt_2014 = np.concatenate((4*np.ones((len(gt_2014), 1)), gt_2014), axis=1)
gt_2015 = gt_2015.to_numpy()[:,:-1].astype('float')
gt_2015 = np.concatenate((5*np.ones((len(gt_2015), 1)), gt_2015), axis=1)
```

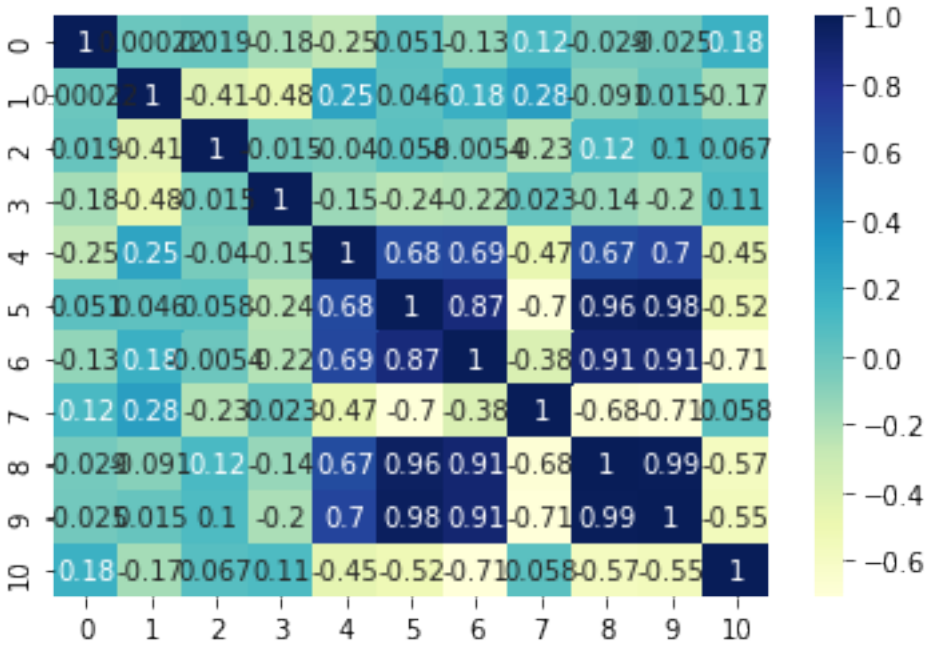
[ ]:
```
gt = np.concatenate((gt_2011,gt_2012,gt_2013,gt_2014,gt_2015))
```

1 Q1

```
[ ]: #normalize data
idx = [i for i in range(1, 11)]
gt[:, idx] = (gt[:, idx] - np.mean(gt[:, idx], axis=0)) / np.std(gt[:, idx],␣
  ↪axis=0)
```

2 Q2

```
[ ]: import seaborn as sb
corr = np.corrcoef(gt.T)
dataplot = sb.heatmap(corr, cmap="YlGnBu", annot=True)
plt.show()
```

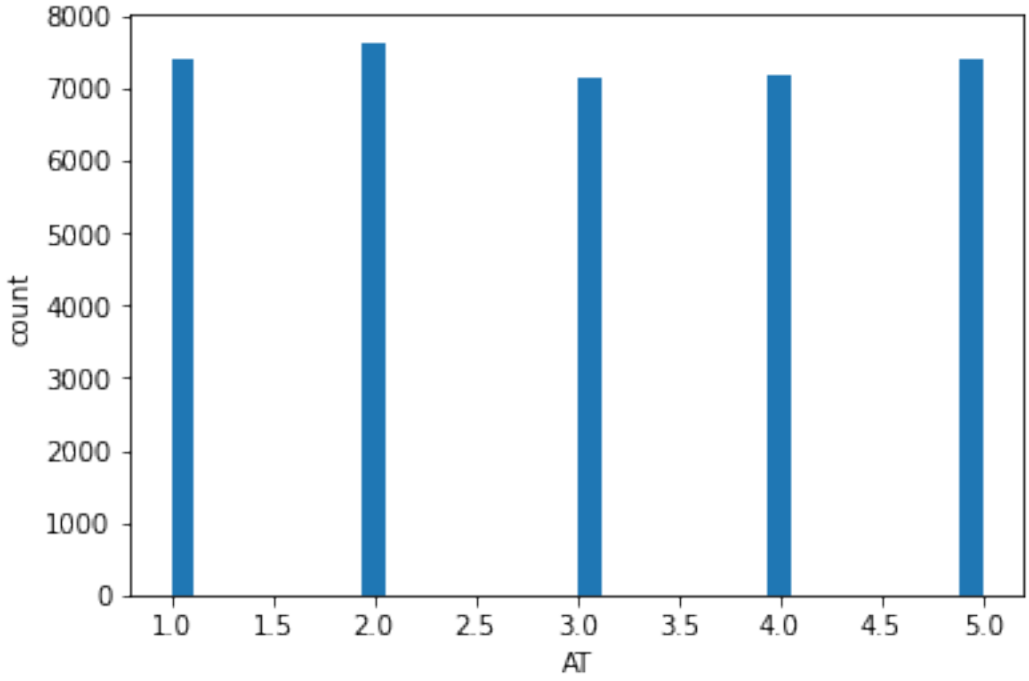


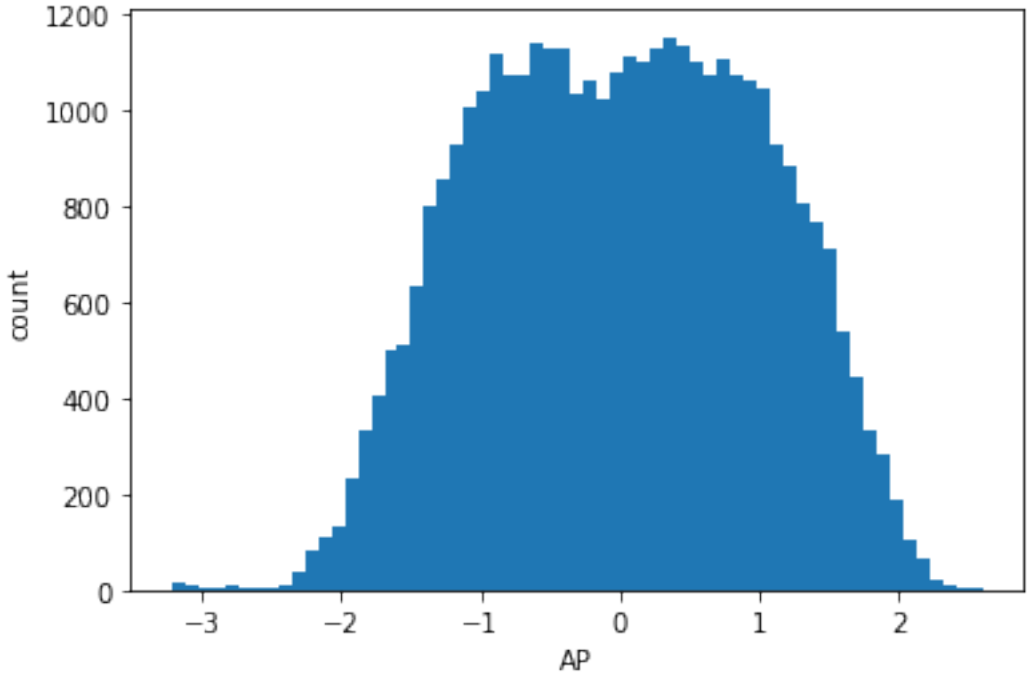
From the figure above, we can see that the target is more relative to AFDP, GTEP, TIT, TEY, and CDP. At the same time, we can see AFDP, GTEP, TIT, TEY, and CDP are also highly relativeness.

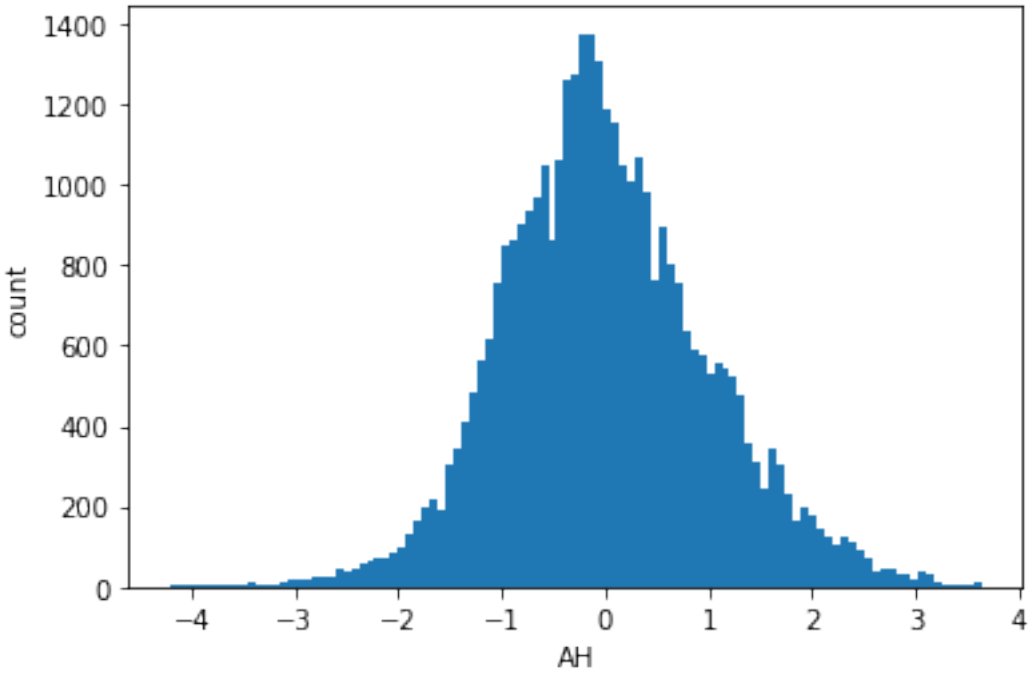
3 Q3

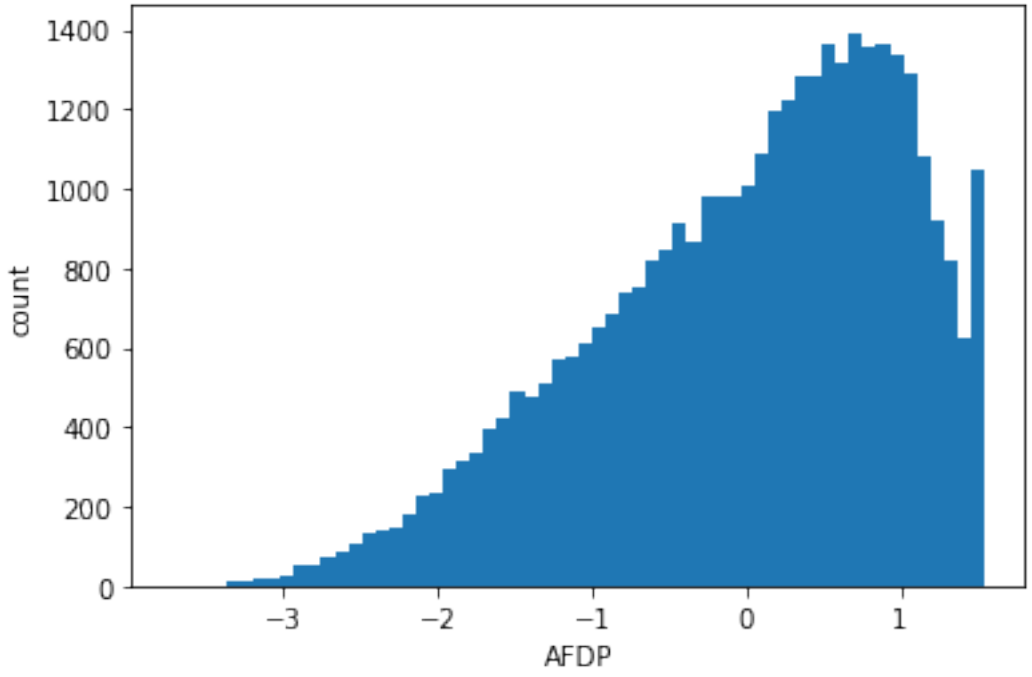
```
[ ]: features = features[:-1]
for i in range(9):
  plt.hist(gt[:, i], bins='auto')
  plt.xlabel(features[i])
  plt.ylabel('count')
```

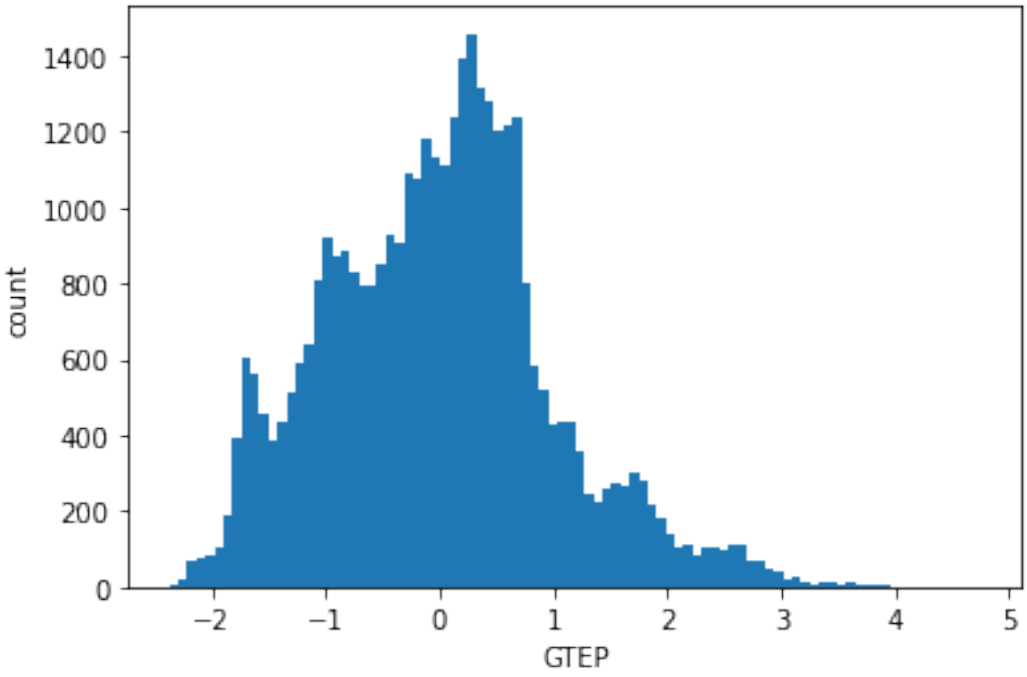
```
plt.show()
```

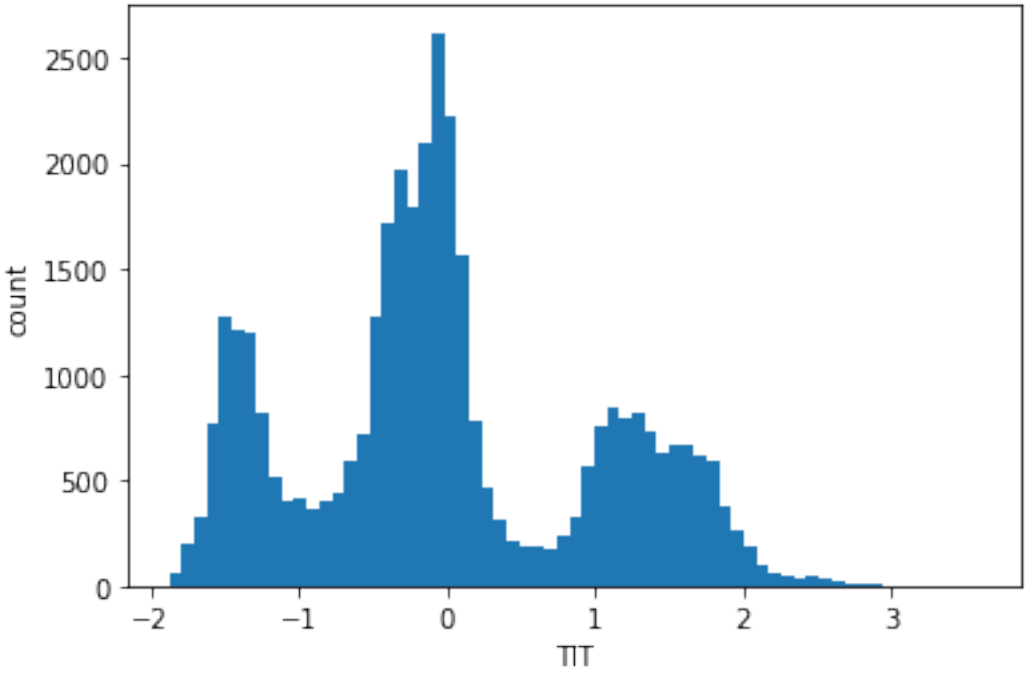

8000
7000
6000
5000
4000
3000
2000
1000
0
count
1.0
1.5
2.0
2.5
3.0
3.5
4.0
4.5
5.0
AT

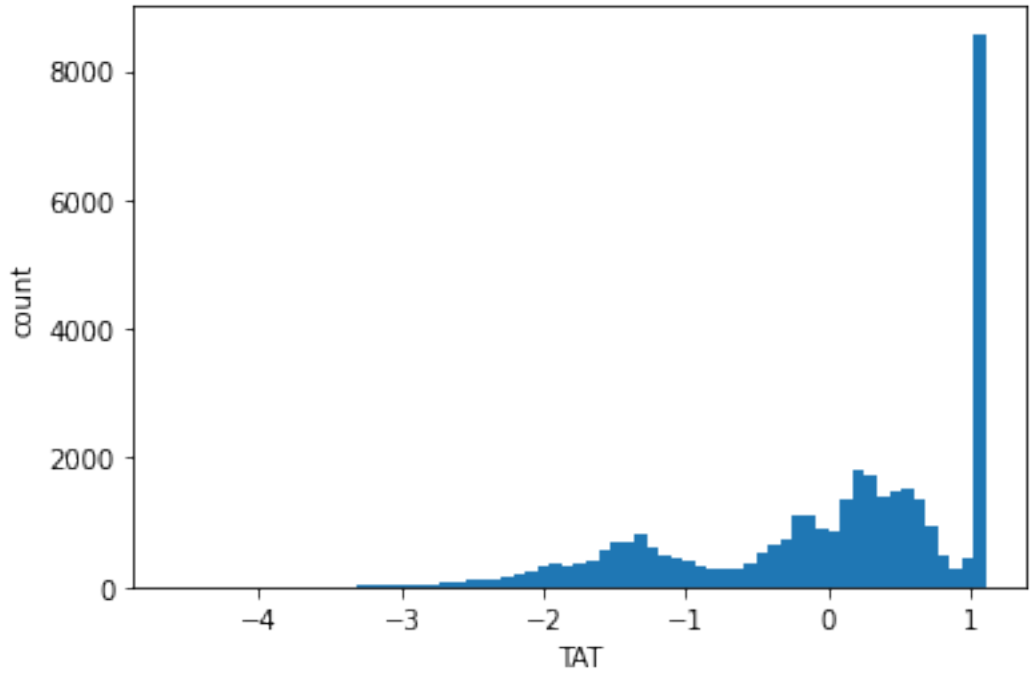

1200
1000
800
600
400
200
0
count
−3
−2
−1
0
1
2
AP

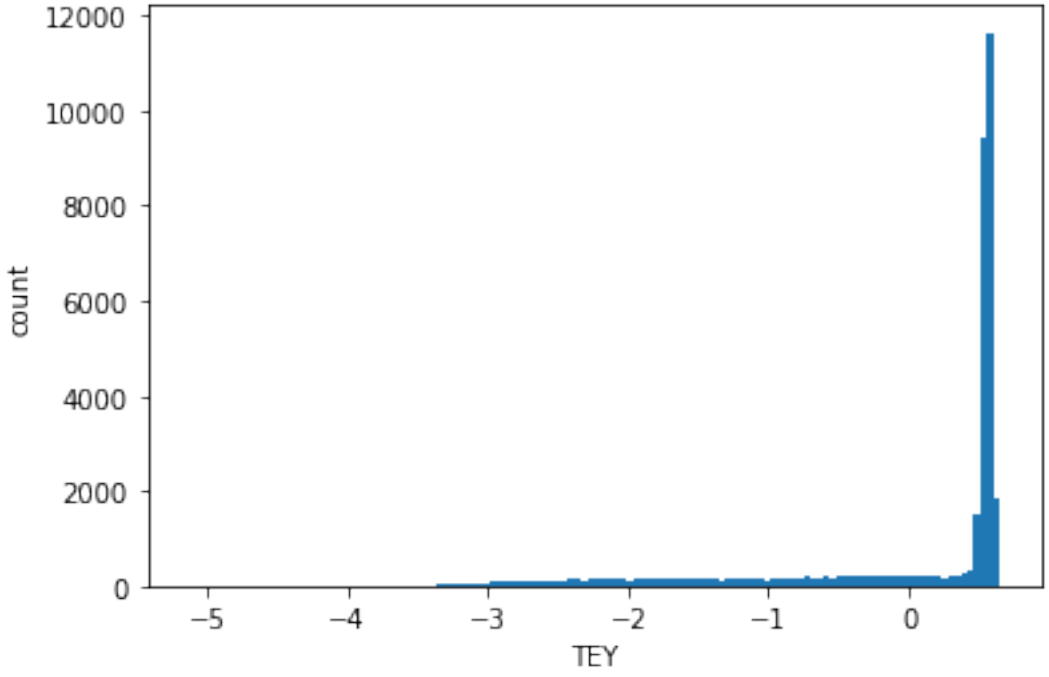

1400
1200
1000
800
600
400
200
0
count
−4
−3
−2
−1
0
1
2
3
4
AH

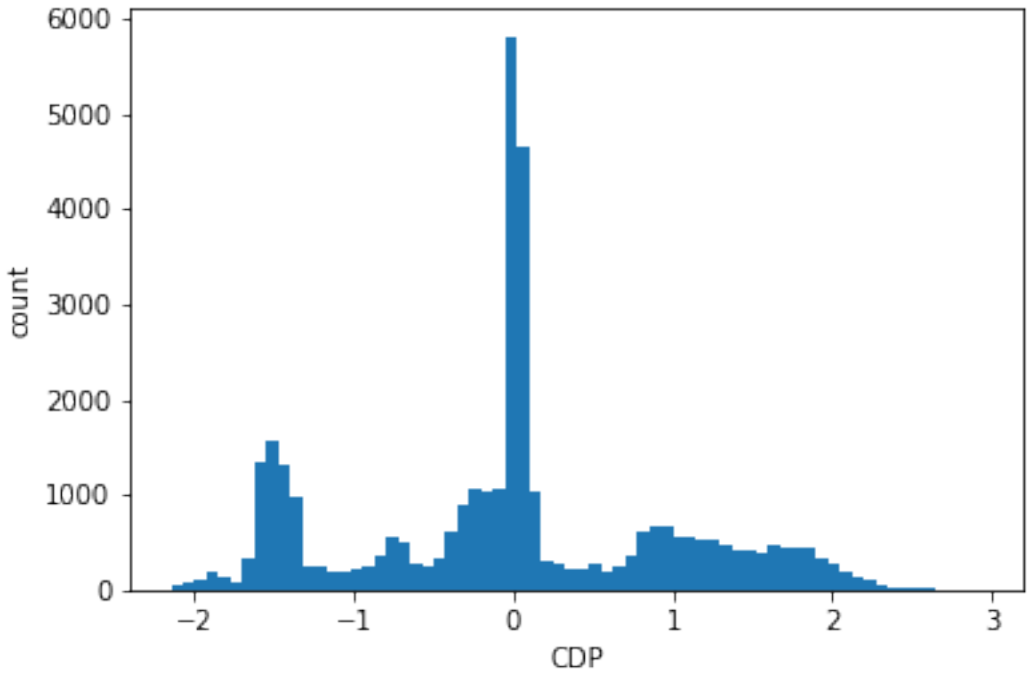

1400
1200
1000
800
600
400
200
0
count
−3
−2
−1
0
1
AFDP


1400
1200
1000
800
600
400
200
0
count
−2
−1
0
1
2
3
4
5
GTEP


2500
2000
1500
1000
500
0
count
−2
−1
0
1
2
3
TIT


8000
6000
4000
2000
0
count
−4
−3
−2
−1
0
1
TAT

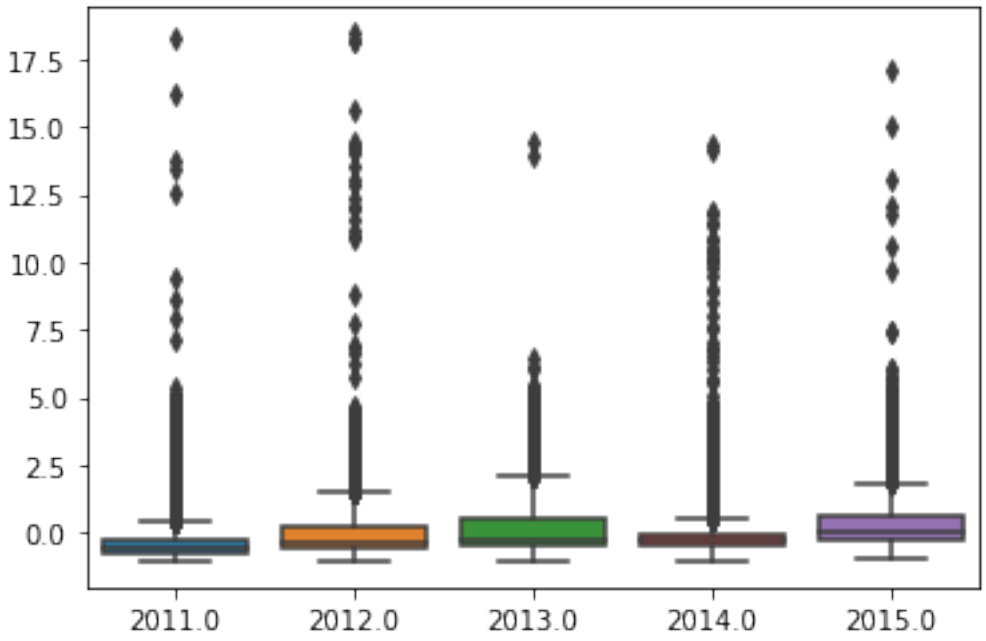

12000
10000
8000
6000
4000
2000
0
count
−5
−4
−3
−2
−1
0
TEY



There are some methods for dealing with skew data. For instance, we can do log transformation, remove outliers, square root or Box Cox transformation.

4 Q4

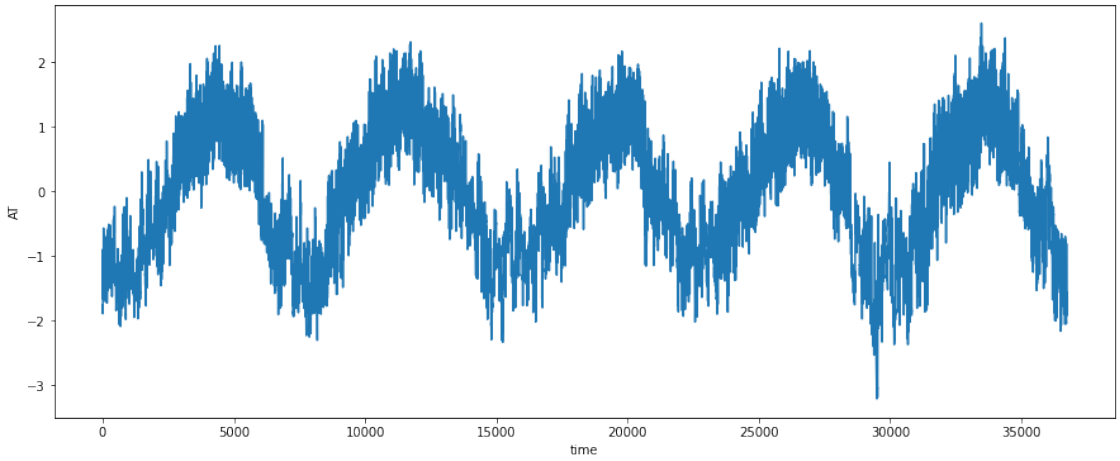
[ ]:
```
data = {'year':2010+gt[:, 0], 'CO2':gt[:, -1]}
ax = sb.boxplot(x='year', y="CO2", data=data)
plt.show()
```

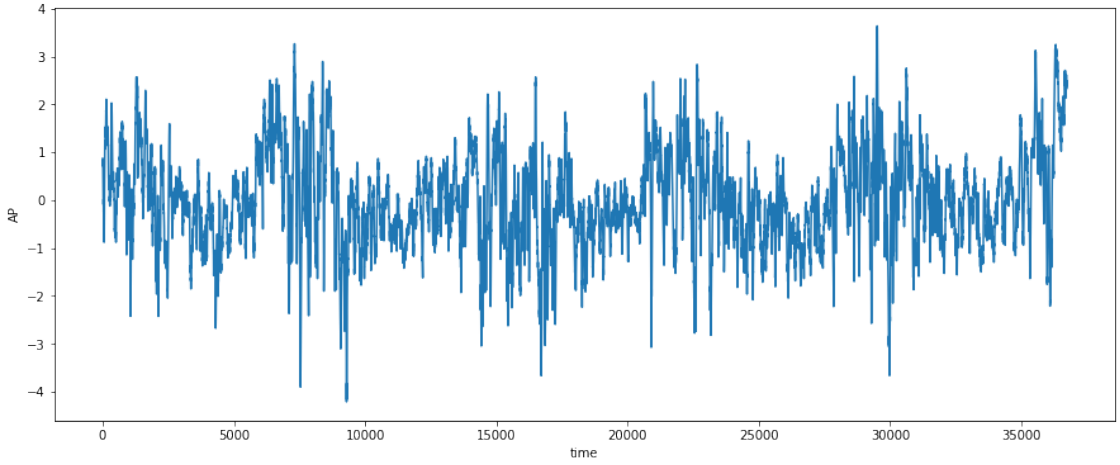


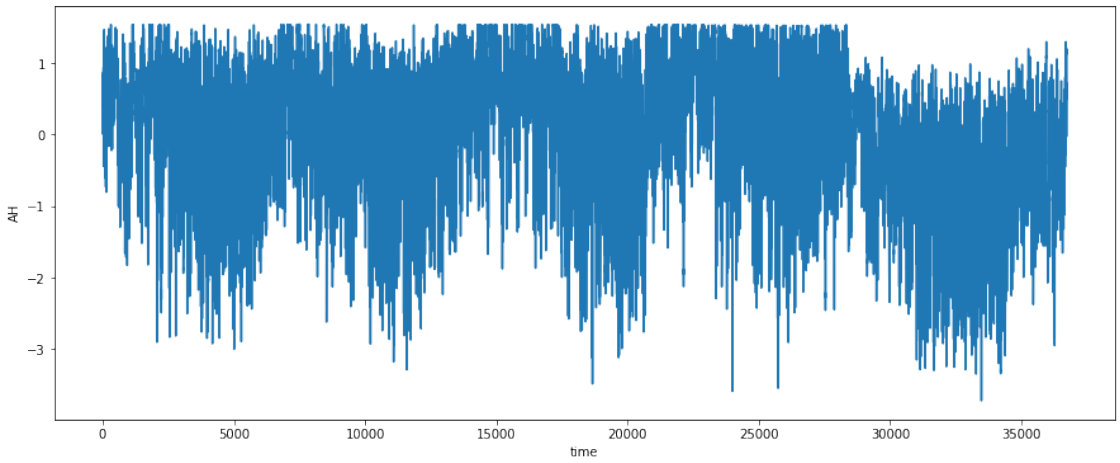
For the figure above, we can see that these features are not much relative to CO2. We can also observe that from the heapmap about correlation by Q2.

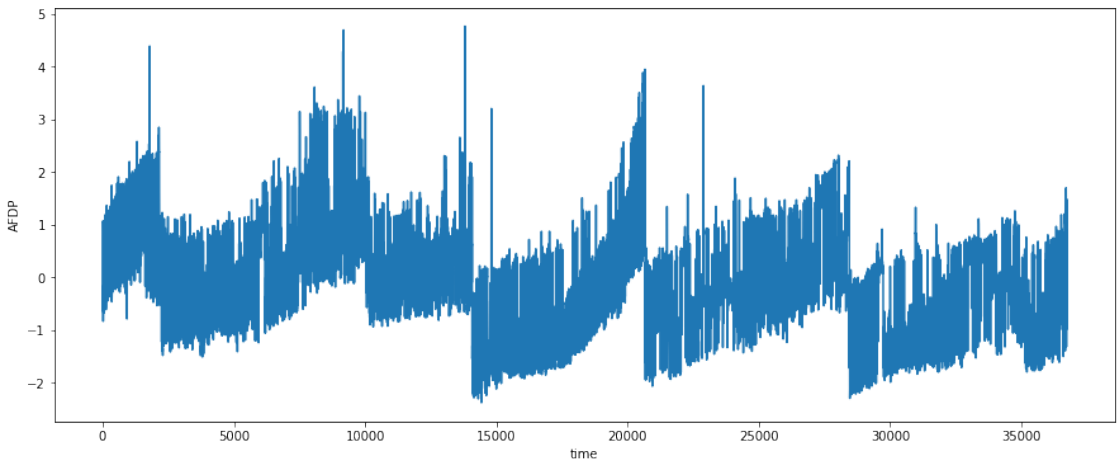
5 Q6

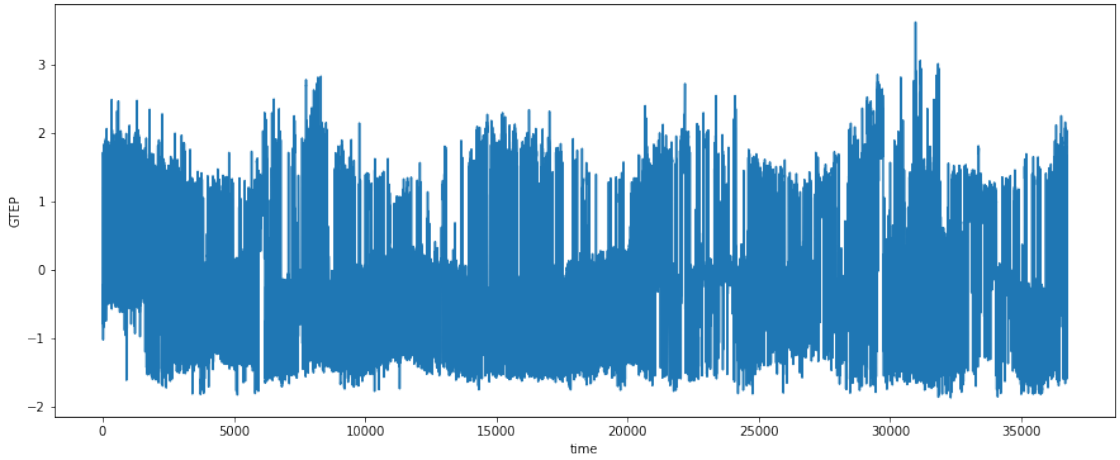
```
[ ]: for i, f in enumerate(features[:-1]):
       plt.plot(gt[:, i+1])
       plt.xlabel('time')
       plt.ylabel(f)
       plt.gcf().set_size_inches(15, 6)
       plt.show()
```

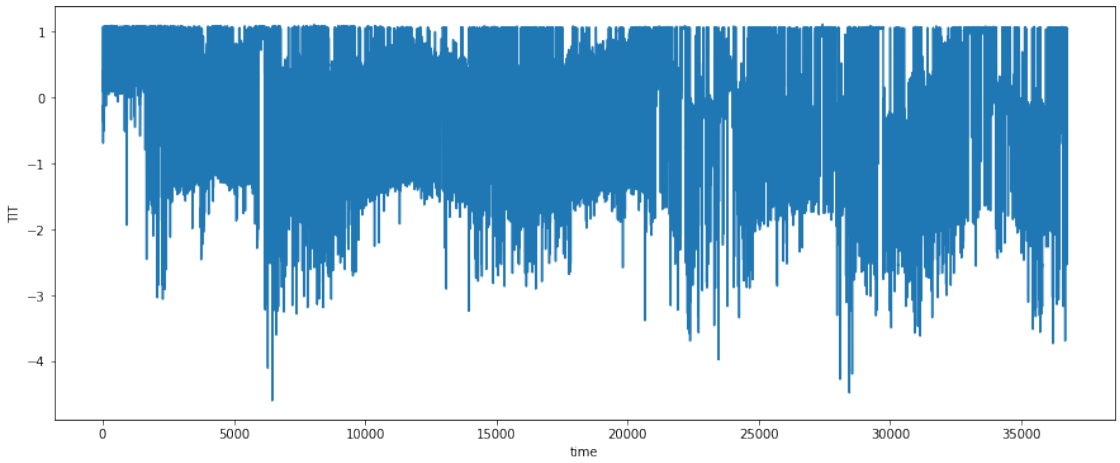


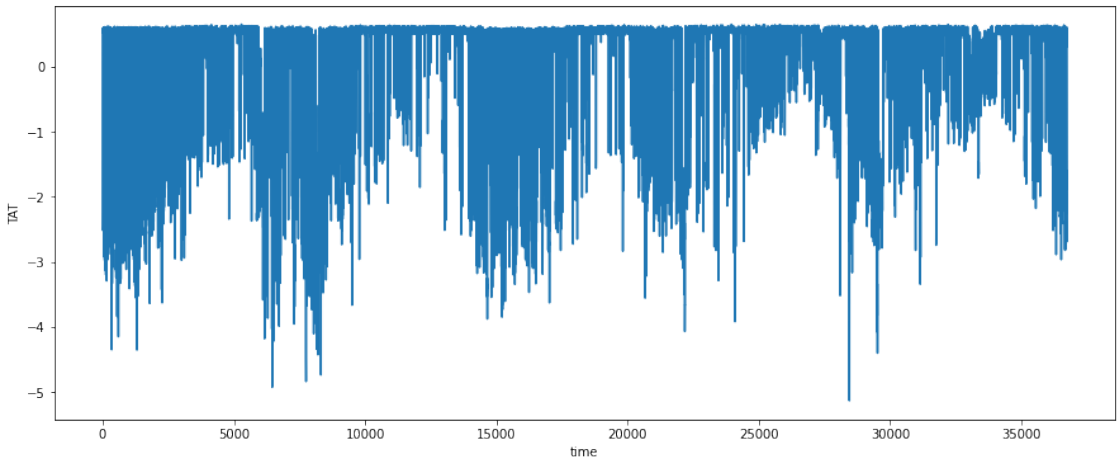

AP
time

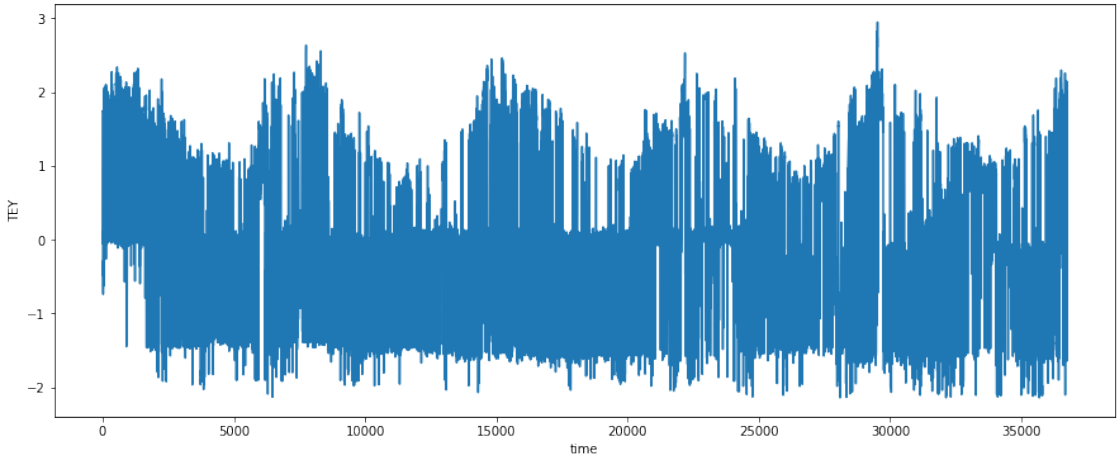

AH
time

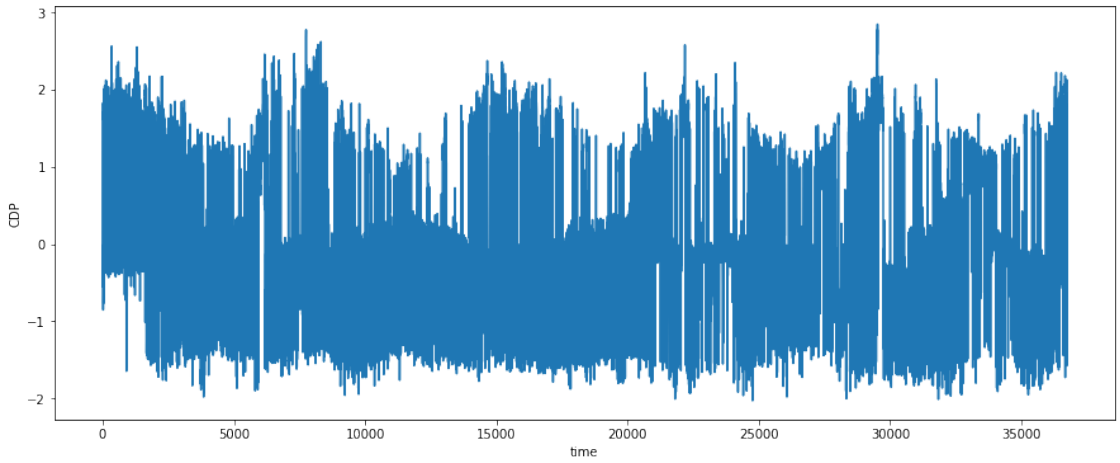

AFDP
time


GTEP
3
2
1
0
−1
−2
0
5000
10000
15000
20000
25000
30000
35000
time


TIT
1
0
−1
−2
−3
−4
0
5000
10000
15000
20000
25000
30000
35000
time


TAT
0
−1
−2
−3
−4
−5
0
5000
10000
15000
20000
25000
30000
35000
time





From the figure above, we can see that there's the same pattern every year on AT, AP and AH, which may indicate that the feature of which year is not so important.

6 Q7

[ ]:
```
from sklearn.feature_selection import mutual_info_regression
from sklearn.feature_selection import f_regression
ind = [i for i in range(10)]
print('mutual_info_regression:{}'.format(mutual_info_regression(gt[:, ind], gt[:
  ↪, -1])))
print('F-scores:{}'.format(f_regression(gt[:, ind], gt[:, -1])[0]))
```

```
mutual_info_regression:[0.12359336 0.10490318 0.0418297  0.02552565 0.2786647
0.44540263
 0.53698837 0.16074676 0.49587231 0.47391084]
F-scores:[ 1208.14433729  1151.22090472   165.87752857   422.08013102
  9245.08377364 13534.97054407 36558.68834551   125.50084238
 17660.02276429 16015.41677409]
```

[ ]:
```
#feature extraction
from sklearn.model_selection import train_test_split
gt_x = gt[:, [4,5,6,8,9]]
gt_y = gt[:, -1]
gt_x_train, gt_x_test, gt_y_train, gt_y_test = train_test_split(gt_x, gt_y,␣
 ↪test_size=0.2, random_state=42)
```

7 Q8

linear : $\min_w ||xw - y||_2^2$

ridge : $\min_w ||xw - y||_2^2 + \alpha||w||_2^2$

lasso : $\min_w ||xw - y||_2^2 + \alpha||w||_1$

For ridge regression, the solution is more stable and the output is non-sparse. And ridge regression has analytical solution. As for lasso regression, the output could be sparse, that is to say, some features don't have influence on the model.

8 Q9

[ ]:
```
#linear regression without regularization
from sklearn.linear_model import LinearRegression, Ridge, Lasso
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import mean_squared_error

lrg = LinearRegression().fit(gt_x_train, gt_y_train)
err = mean_squared_error(lrg.predict(gt_x_test), gt_y_test)
print('rmse for linear regression without regularization={}'.format(err**0.5))
```

```
rmse for linear regression without regularization=0.6960362297005847
```

[ ]:
```
#Ridge regression
reg = {'alpha':[10**i for i in range(-5, 6)]}
rid = Ridge()
clf = GridSearchCV(estimator=rid, param_grid=reg, cv=10,␣
 ↪scoring='neg_root_mean_squared_error', n_jobs=-1)
clf.fit(gt_x_train, gt_y_train)

print("Best Parameters: "+str(clf.best_params_))
```

```
rid = Ridge(alpha=clf.best_params_['alpha']).fit(gt_x_train, gt_y_train)
err = mean_squared_error(rid.predict(gt_x_test), gt_y_test)
print('rmse for linear regression with ridge regularization={}'.format(err**0.
 ↪5))
```

```
Best Parameters: {'alpha': 1}
rmse for linear regression with ridge regularization=0.6960464360748981
```

[ ]:
```
las = Lasso()
clf = GridSearchCV(estimator=las, param_grid=reg, cv=10,␣
 ↪scoring='neg_root_mean_squared_error', n_jobs=-1)
clf.fit(gt_x_train, gt_y_train)

print("Best Parameters: "+str(clf.best_params_))
las = Lasso(alpha=clf.best_params_['alpha']).fit(gt_x_train, gt_y_train)
err = mean_squared_error(las.predict(gt_x_test), gt_y_test)
print('rmse for linear regression with Lasso regularization={}'.format(err**0.
 ↪5))
```

```
Best Parameters: {'alpha': 1e-05}
rmse for linear regression with Lasso regularization=0.6960388548512793
```

9 Q10

If we don't apply regularization, then feature scaling would not influence the outcome. It would just influence on the weight of the model, but the optimization solution is the same. However, feature scaling makes influence on models with regularization. Because the weight term is in the objective function, we may sacrifice the bias to make weight smaller.

10 Q11

[ ]:
```
print('p-value:{}'.format(f_regression(gt[:, ind], gt[:, -1])[1]))
```

```
p-value:[1.75586548e-260 1.70122203e-248 7.10695129e-038 2.88245714e-093
 0.00000000e+000 0.00000000e+000 0.00000000e+000 4.40808602e-029
 0.00000000e+000 0.00000000e+000]
```

11 Q12

By heatmap of correlation, the most salient feature is the TIT.

12 Q13

[ ]:
```
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import PolynomialFeatures
```

```
pipeline = Pipeline([
    ('poly', PolynomialFeatures()),
    ('clf', None),
]
)

param_grid = [{
        "poly__degree":[i for i in range(1, 7)],
        "clf__alpha":[0] + [10**i for i in range(-5, 6)],
        "clf":(
            Ridge(fit_intercept=False),
            Lasso(fit_intercept=False),
        ),
    }]
```

[ ]:
```
cv = GridSearchCV(pipeline,cv=10,param_grid=param_grid,scoring='neg_root_mean_squared_error',n_jobs=-1)
cv.fit(gt_x_train, gt_y_train)
```

[ ]:
```
print(cv.best_params_)
```

```
{'clf': Ridge(alpha=1, fit_intercept=False), 'clf__alpha': 1, 'poly__degree': 5}
```

The best model is when degree is 5 and with ridge regularization with parameter 1.

Increasing the degree means the hypothesis set becomes larger, so we can fit training set better. However, this also increases the risk of overfitting and make performance on test set is bad.

Therefore, we can view the degree as a hyperparameter, and choose by cross validation.

13 Q14

14 Q15

[ ]:
```
from sklearn.neural_network import MLPRegressor
mlp = MLPRegressor().fit(gt_x_train, gt_y_train)
err = mean_squared_error(mlp.predict(gt_x_test), gt_y_test)
print('rmse for multi-layer perceptron={}'.format(err**0.5))
```

```
rmse for multi-layer perceptron=0.5595110806261948

/usr/local/lib/python3.7/dist-
packages/sklearn/neural_network/_multilayer_perceptron.py:696:
ConvergenceWarning: Stochastic Optimizer: Maximum iterations (200) reached and
the optimization hasn't converged yet.
  ConvergenceWarning,
```

rmse for multi-layer perceptron=0.5595110806261948

Why does it do much better than linear regression? Again, MLP outperforms linear regression for gas datasets as well, since it is a complex dataset with multiple layers and neural networks such as MLP conducts back propagation to deal with non-linearities.

15 Q16

[ ]:
```
param_mlp = {'hidden_layer_sizes':␣
 ↪[(100,),(200,200),(300,300,300),(300,300,200,100),(400,400,400,400)],
          'alpha': [10**i for i in range(-5, -2)], #weight decay
          }

mlp = MLPRegressor(early_stopping=True)
clf = GridSearchCV(estimator=mlp, param_grid=param_mlp, cv=3,␣
 ↪scoring='neg_root_mean_squared_error', n_jobs=-1, verbose=3)
clf.fit(gt_x_train, gt_y_train)

print("Best Parameters: "+str(clf.best_params_))
mlp = MLPRegressor(hidden_layer_sizes=clf.best_params_['hidden_layer_sizes'],␣
 ↪alpha=clf.best_params_['alpha']).fit(gt_x_train, gt_y_train)
err = mean_squared_error(mlp.predict(gt_x_test), gt_y_test)
print('rmse for multi-layer perceptron={}'.format(err**0.5))
```

```
Fitting 3 folds for each of 15 candidates, totalling 45 fits
Best Parameters: {'alpha': 0.0001, 'hidden_layer_sizes': (400, 400, 400, 400)}
rmse for multi-layer perceptron=0.5560591124316742
```

Best Parameters: {'alpha': 0.0001, 'hidden_layer_sizes': (400, 400, 400, 400)}

rmse for multi-layer perceptron=0.5560591124316742

16 Q17

[ ]:
```
param_mlp = {'hidden_layer_sizes': [(400,400,400,400)],
          'activation': ['identity','relu','tanh','logistic'],
          'alpha': [0.0001], #weight decay
          }

mlp = MLPRegressor(early_stopping=True)
clf = GridSearchCV(estimator=mlp, param_grid=param_mlp, cv=3,␣
 ↪scoring='neg_root_mean_squared_error', n_jobs=-1, verbose=10)
clf.fit(gt_x_train, gt_y_train)

print("Best Parameters: "+str(clf.best_params_))
mlp = MLPRegressor(hidden_layer_sizes=clf.best_params_['hidden_layer_sizes'],␣
 ↪activation=clf.best_params_['activation'], alpha=clf.best_params_['alpha']).
 ↪fit(gt_x_train, gt_y_train)
err = mean_squared_error(mlp.predict(gt_x_test), gt_y_test)
```

```
print('rmse for multi-layer perceptron={}'.format(err**0.5))
```

```
Fitting 3 folds for each of 4 candidates, totalling 12 fits
Best Parameters: {'activation': 'relu', 'alpha': 0.0001, 'hidden_layer_sizes':
(400, 400, 400, 400)}
rmse for multi-layer perceptron=0.5184101375089821
```

What activation function should be used for the output? From above, "ReLU" is the best activation function.

Best Parameters: {'activation': 'relu', 'alpha': 0.0001, 'hidden_layer_sizes': (400, 400, 400, 400)}

rmse for multi-layer perceptron=0.5184101375089821

17 Q18

Refer to diamonds answer.

18 Q19

[ ]:
```
from sklearn.ensemble import RandomForestRegressor

param_rfr = {"max_depth": [2,4,8,16, None],
             "n_estimators":[50,100,200,300,400,500],
             "max_features": [0.1, 0.25, 0.5, 0.75, 1.0],}

rfr = RandomForestRegressor()
clf = GridSearchCV(estimator=rfr, param_grid=param_rfr, cv=3,␣
 ↪scoring='neg_root_mean_squared_error', n_jobs=-1, verbose=10)
clf.fit(gt_x_train, gt_y_train)

print("Best Parameters: "+str(clf.best_params_))
rfr = RandomForestRegressor(max_depth=clf.best_params_['max_depth'],␣
 ↪n_estimators=clf.best_params_['n_estimators'], max_features=clf.
 ↪best_params_['max_features']).fit(gt_x_train, gt_y_train)
err = mean_squared_error(rfr.predict(gt_x_test), gt_y_test)
print('rmse for Random Forest Regressor={}'.format(err**0.5))
```

```
Fitting 3 folds for each of 150 candidates, totalling 450 fits

/usr/local/lib/python3.7/dist-
packages/joblib/externals/loky/process_executor.py:705: UserWarning: A worker
stopped while some jobs were given to the executor. This can be caused by a too
short worker timeout or by a memory leak.
  "timeout or by a memory leak.", UserWarning

Best Parameters: {'max_depth': None, 'max_features': 0.5, 'n_estimators': 200}
rmse for Random Forest Regressor=0.5186939368408344
```

Best Parameters: {'max_depth': None, 'max_features': 0.5, 'n_estimators': 200}

rmse for Random Forest Regressor=0.5186939368408344

Explain how these hyper-parameters affect the overall performance? ***max_depth*** - For max_depth=[4, 8, 12, 16, 32, None] with remaining parameters kept at best paramater values, the RMSE scores were RMSE=[0.630, 0.551, 0.533, 0.522, 0.522, 0.519]. From this, one can observe that whilst increasing max_depth value has less impact after max_depth=16, changing the values of max_depth still has a significant impact on the performance scores at lower values. ***max_features*** - The impact of altering the maximum features on the RMSE score is less compared to the other hyperparameters. The best is at 50% of n_features however, changing this value has small impact. When tested with max_features=[0.25, 0.50, 0.75, 1.0] while the remaining parameters were set at best parameter values, the RMSE scores were [0.542, 0.519, 0.520, 0.521]. This indicates that while minimum RMSE is achieved when max_features are approximately 50-75%, altering them to values outside this range does not have much impact as compared to changing max_depth. ***n_estimators*** - As seen with the diamonds dataset, the impact of altering the number of estimators is also less compared to that of the maximum depth. When experimenting with n_estimators=[10,50,100,200] and the rest parameters taking the best parameter values, the RMSE scores were (in corresponding order) [0.541, 0.530, 0.527, 0.522], which indicates that after n_estimators=100, the RMSE score plateaus. **Do some of them have regularization effect?** In overall, max_depth has a regularization effect since it is able to tune the complexity of the tree and adjust between under-fitting and over-fitting.

[ ]:
```
from sklearn.ensemble import RandomForestRegressor
rfr = RandomForestRegressor(max_depth=16, n_estimators=200, max_features=0.5).
 ↪fit(gt_x_train, gt_y_train)
err = mean_squared_error(rfr.predict(gt_x_test), gt_y_test)
print('rmse for Random Forest Regressor={}'.format(err**0.5))
rfr = RandomForestRegressor(max_depth=32, n_estimators=200, max_features=0.5).
 ↪fit(gt_x_train, gt_y_train)
err = mean_squared_error(rfr.predict(gt_x_test), gt_y_test)
print('rmse for Random Forest Regressor={}'.format(err**0.5))
rfr = RandomForestRegressor(max_depth=8, n_estimators=200, max_features=0.5).
 ↪fit(gt_x_train, gt_y_train)
err = mean_squared_error(rfr.predict(gt_x_test), gt_y_test)
print('rmse for Random Forest Regressor={}'.format(err**0.5))
```

```
rmse for Random Forest Regressor=0.522265121111036
rmse for Random Forest Regressor=0.521875120849809
rmse for Random Forest Regressor=0.5513274786710335
```

[ ]:
```
rfr = RandomForestRegressor(max_depth=12, n_estimators=200, max_features=0.5).
 ↪fit(gt_x_train, gt_y_train)
err = mean_squared_error(rfr.predict(gt_x_test), gt_y_test)
print('rmse for Random Forest Regressor={}'.format(err**0.5))
rfr = RandomForestRegressor(max_depth=4, n_estimators=200, max_features=0.5).
 ↪fit(gt_x_train, gt_y_train)
err = mean_squared_error(rfr.predict(gt_x_test), gt_y_test)
print('rmse for Random Forest Regressor={}'.format(err**0.5))
```

```
rmse for Random Forest Regressor=0.5333056552360366
```

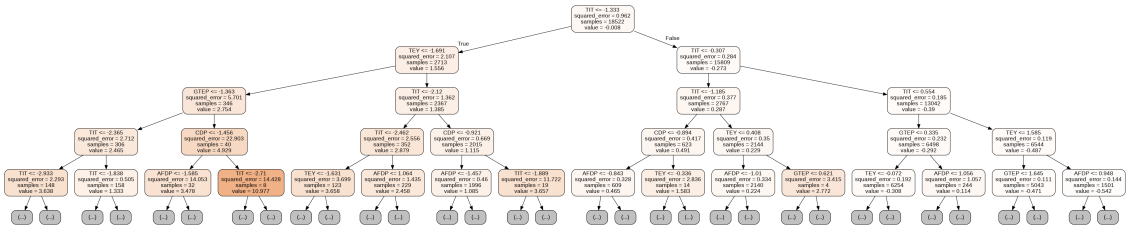
```
rmse for Random Forest Regressor=0.629693120252785
```

19 Q20

Refer to answer on diamonds dataset.

20 Q21

[ ]:
```
from sklearn.tree import export_graphviz

estimator = rfr.estimators_[0]
export_graphviz(estimator,
                out_file='tree.dot',
                feature_names = ["AFDP","GTEP", "TIT", "TEY" ,"CDP"],
                rounded = True, proportion = False,
                max_depth = 4,
                filled = True)

from subprocess import call
call(['dot', '-Tpng', 'tree.dot', '-o', 'tree.png'])

from IPython.display import Image
Image(filename = 'tree.png')
```

[ ]:


Which feature is selected for branching at the root node? At the root node, "TIT" is selected as the branching feature.

What can you infer about the importance of features? As "TIT" is used in the root node and also mainly used in depth 1,2,3 one can observe that "TIT" is the most distinguishing feature out of all. The next important features are "GTEP", "TEY", "CDP" as they are also used in the first 3 depths. In the 4th layer, one observes the emergence of the remaining branching feature, "AFDP" as well.

21 Q22-24: Skipped (Refer to Diamonds)

22 Q25

[ ]:
```
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_validate
from numpy import absolute
from numpy import mean
from numpy import sqrt

kf = KFold(n_splits=10, random_state=1, shuffle=True)
```

[ ]:
```
# Linear Regression
lrg = LinearRegression()
scores = cross_validate(lrg, gt_x_train, gt_y_train,␣
 ↪scoring='neg_root_mean_squared_error', cv=kf, n_jobs=-1,␣
 ↪return_train_score=True)
train_rmse = mean(absolute(scores['train_score']))
test_rmse = mean(absolute(scores['test_score']))
print('rmse for Linear Regression train set={}'.format(train_rmse))
print('rmse for Linear Regression validation set={}'.format(test_rmse))
```

```
rmse for Linear Regression train set=0.6626296054379304
rmse for Linear Regression validation set=0.6604983770366001
```

[ ]:
```
# Polynomial Regression
poly = PolynomialFeatures(degree=5)
gt_x_train_poly = poly.fit_transform(gt_x_train)
gt_x_test_poly = poly.fit_transform(gt_x_test)
lrg = Ridge(alpha=1, fit_intercept=False).fit(gt_x_train_poly, gt_y_train)
scores = cross_validate(lrg, gt_x_train_poly, gt_y_train,␣
 ↪scoring='neg_root_mean_squared_error', cv=kf, n_jobs=-1,␣
 ↪return_train_score=True)
train_rmse = mean(absolute(scores['train_score']))
test_rmse = mean(absolute(scores['test_score']))
print('rmse for Polynomial Regression train set={}'.format(train_rmse))
print('rmse for Polynomial Regression validation set={}'.format(test_rmse))
```

```
rmse for Polynomial Regression train set=0.5329308808186568
rmse for Polynomial Regression validation set=0.5564554351254646
```

[ ]:
```
# Neural Network
mlp = MLPRegressor(early_stopping=True, activation='relu', alpha=0.0001,␣
 ↪hidden_layer_sizes=(400,400,400,400))
scores = cross_validate(mlp, gt_x_train, gt_y_train,␣
 ↪scoring='neg_root_mean_squared_error', cv=kf, n_jobs=-1,␣
 ↪return_train_score=True)
train_rmse = mean(absolute(scores['train_score']))
```

```
test_rmse = mean(absolute(scores['test_score']))
print('rmse for Neural Network train set={}'.format(train_rmse))
print('rmse for Neural Network validation set={}'.format(test_rmse))
```

```
rmse for Neural Network train set=0.4960347738960905
rmse for Neural Network validation set=0.5306114733674373
```

[ ]:
```
# Random Forest
rfr = RandomForestRegressor(max_depth=8, max_features=0.75, n_estimators=200,␣
 ↪oob_score=True, warm_start=True)
scores = cross_validate(rfr, gt_x_train, gt_y_train,␣
 ↪scoring='neg_root_mean_squared_error', cv=kf, n_jobs=-1,␣
 ↪return_train_score=True, return_estimator=True)
train_rmse = mean(absolute(scores['train_score']))
test_rmse = mean(absolute(scores['test_score']))
print('rmse for Random Forest Regressor train set={}'.format(train_rmse))
print('rmse for Random Forest Regressor validation set={}'.format(test_rmse))
```

```
rmse for Random Forest Regressor train set=0.4383951239831803
rmse for Random Forest Regressor validation set=0.5403582124024631
```

[ ]:
```
oob_error = 1-np.mean([i.oob_score_ for i in scores['estimator']])
print('Out of Bag Error={}'.format(oob_error))
```

```
Out of Bag Error=0.3029522093477166
```

23 Q26

From above, Out of Bag Error=0.3029522093477166 and Out of Bag Score=0.6970477906522834. For explanation on OOB score and R2, refer to the answer on diamonds dataset.

twitter

March 18, 2022

1 Part 2: Twitter Data

Install cuDF & cuML Reference: https://colab.research.google.com/drive/1rY7Ln6rEE1pOlfSHCYOVaqt8OvD

[ ]:
```
# This get the RAPIDS-Colab install files and test check your GPU.  Run this␣
 ↪and the next cell only.
# Please read the output of this cell.  If your Colab Instance is not RAPIDS␣
 ↪compatible, it will warn you and give you remediation steps.
!git clone https://github.com/rapidsai/rapidsai-csp-utils.git
!python rapidsai-csp-utils/colab/env-check.py
```

[ ]:
```
# This will update the Colab environment and restart the kernel.  Don't run the␣
 ↪next cell until you see the session crash.
!bash rapidsai-csp-utils/colab/update_gcc.sh
import os
os._exit(00)
```

[ ]:
```
# This will install CondaColab.  This will restart your kernel one last time. ␣
 ↪Run this cell by itself and only run the next cell once you see the session␣
 ↪crash.
import condacolab
condacolab.install()
```

[ ]:
```
# you can now run the rest of the cells as normal
import condacolab
condacolab.check()
```

[ ]:
```
# Installing RAPIDS is now 'python rapidsai-csp-utils/colab/install_rapids.py␣
 ↪<release> <packages>'
# The <release> options are 'stable' and 'nightly'.  Leaving it blank or adding␣
 ↪any other words will default to stable.
!python rapidsai-csp-utils/colab/install_rapids.py stable
import os
os.environ['NUMBAPRO_NVVM'] = '/usr/local/cuda/nvvm/lib64/libnvvm.so'
os.environ['NUMBAPRO_LIBDEVICE'] = '/usr/local/cuda/nvvm/libdevice/'
os.environ['CONDA_PREFIX'] = '/usr/local'
```

==========

[ ]:
```
# from matplotlib import path
from google.colab import drive
drive.mount('/content/drive')
import sys
import os
path_to_module = '/content/drive/MyDrive/'
sys.path.append(path_to_module)
os.chdir(path_to_module)
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call
drive.mount("/content/drive", force_remount=True).
```

[ ]:
```
gpu_info = !nvidia-smi
gpu_info = '\n'.join(gpu_info)
if gpu_info.find('failed') >= 0:
    print('Not connected to a GPU')
else:
    print(gpu_info)

from psutil import virtual_memory
ram_gb = virtual_memory().total / 1e9
print('Your runtime has {:.1f} gigabytes of available RAM\n'.format(ram_gb))

if ram_gb < 20:
    print('Not using a high-RAM runtime')
else:
    print('You are using a high-RAM runtime!')
```

```
Fri Mar 18 21:16:22 2022
+-----------------------------------------------------------------------------+
| NVIDIA-SMI 460.32.03    Driver Version: 460.32.03    CUDA Version: 11.2     |
|-------------------------------+----------------------+----------------------+
| GPU  Name        Persistence-M| Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp  Perf  Pwr:Usage/Cap|         Memory-Usage | GPU-Util  Compute M. |
|                               |                      |               MIG M. |
|===============================+======================+======================|
|   0  Tesla P100-PCIE…  Off  | 00000000:00:04.0 Off |                    0 |
| N/A   36C    P0    26W / 250W |      0MiB / 16280MiB |      0%      Default |
|                               |                      |                  N/A |
+-------------------------------+----------------------+----------------------+

+-----------------------------------------------------------------------------+
| Processes:                                                                  |
|  GPU   GI   CI        PID   Type   Process name                  GPU Memory |
|        ID   ID                                                   Usage      |
|=============================================================================|
|  No running processes found                                                 |
+-----------------------------------------------------------------------------+
```

```
Your runtime has 27.3 gigabytes of available RAM

You are using a high-RAM runtime!
```

[ ]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import string
import nltk
nltk.download("all")
import json
import datetime, time
import pytz
import csv
import math

import warnings
warnings.filterwarnings('ignore')

from nltk import pos_tag, word_tokenize
from nltk.stem.wordnet import WordNetLemmatizer
from nltk.corpus import wordnet as wn
from nltk.corpus import stopwords
from nltk.corpus import wordnet
from nltk.corpus import words
from nltk.stem.lancaster import LancasterStemmer as LS

from string import punctuation
from collections import Counter
import re
import seaborn as sns

from sklearn.model_selection import train_test_split
from sklearn.feature_extraction import text
from sklearn.feature_extraction.text import ENGLISH_STOP_WORDS

from sklearn.model_selection import cross_validate
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import accuracy_score, roc_curve, confusion_matrix,␣
 ↪recall_score, precision_score, f1_score

# from sklearn.decomposition import TruncatedSVD
# from sklearn.feature_extraction.text import CountVectorizer
# from sklearn.feature_extraction.text import TfidfTransformer
# from sklearn.feature_extraction.text import TfidfVectorizer
# from sklearn.svm import SVC
# from sklearn.linear_model import LogisticRegression
```

```
# from sklearn.naive_bayes import GaussianNB

import cudf
import cuml

from cuml.preprocessing.text.stem import PorterStemmer
from cuml.feature_extraction.text import CountVectorizer
from cuml.feature_extraction.text import TfidfTransformer
from cuml.feature_extraction.text import TfidfVectorizer

from cuml.decomposition import TruncatedSVD

from cuml.ensemble import RandomForestClassifier as cuRFC
from cuml.ensemble import RandomForestRegressor as cuRFR
from cuml.svm import SVC

from cuml.naive_bayes import GaussianNB
from cuml.linear_model import LinearRegression
from cuml.linear_model import LogisticRegression

from cuml.metrics.regression import mean_squared_error
from cuml.metrics.regression import mean_absolute_error
from cuml.metrics.regression import r2_score
```

```
[nltk_data] Downloading collection 'all'
[nltk_data]    |
[nltk_data]    | Downloading package abc to /root/nltk_data…
[nltk_data]    |   Package abc is already up-to-date!
[nltk_data]    | Downloading package alpino to /root/nltk_data…
[nltk_data]    |   Package alpino is already up-to-date!
[nltk_data]    | Downloading package averaged_perceptron_tagger to
[nltk_data]    |     /root/nltk_data…
[nltk_data]    |   Package averaged_perceptron_tagger is already up-
[nltk_data]    |       to-date!
[nltk_data]    | Downloading package averaged_perceptron_tagger_ru to
[nltk_data]    |     /root/nltk_data…
[nltk_data]    |   Package averaged_perceptron_tagger_ru is already
[nltk_data]    |       up-to-date!
[nltk_data]    | Downloading package basque_grammars to
[nltk_data]    |     /root/nltk_data…
[nltk_data]    |   Package basque_grammars is already up-to-date!
[nltk_data]    | Downloading package biocreative_ppi to
[nltk_data]    |     /root/nltk_data…
[nltk_data]    |   Package biocreative_ppi is already up-to-date!
[nltk_data]    | Downloading package bllip_wsj_no_aux to
[nltk_data]    |     /root/nltk_data…
[nltk_data]    |   Package bllip_wsj_no_aux is already up-to-date!
[nltk_data]    | Downloading package book_grammars to
```

```
[nltk_data]    |     /root/nltk_data…
[nltk_data]    |   Package book_grammars is already up-to-date!
[nltk_data]    | Downloading package brown to /root/nltk_data…
[nltk_data]    |   Package brown is already up-to-date!
[nltk_data]    | Downloading package brown_tei to /root/nltk_data…
[nltk_data]    |   Package brown_tei is already up-to-date!
[nltk_data]    | Downloading package cess_cat to /root/nltk_data…
[nltk_data]    |   Package cess_cat is already up-to-date!
[nltk_data]    | Downloading package cess_esp to /root/nltk_data…
[nltk_data]    |   Package cess_esp is already up-to-date!
[nltk_data]    | Downloading package chat80 to /root/nltk_data…
[nltk_data]    |   Package chat80 is already up-to-date!
[nltk_data]    | Downloading package city_database to
[nltk_data]    |     /root/nltk_data…
[nltk_data]    |   Package city_database is already up-to-date!
[nltk_data]    | Downloading package cmudict to /root/nltk_data…
[nltk_data]    |   Package cmudict is already up-to-date!
[nltk_data]    | Downloading package comparative_sentences to
[nltk_data]    |     /root/nltk_data…
[nltk_data]    |   Package comparative_sentences is already up-to-
[nltk_data]    |       date!
[nltk_data]    | Downloading package comtrans to /root/nltk_data…
[nltk_data]    |   Package comtrans is already up-to-date!
[nltk_data]    | Downloading package conll2000 to /root/nltk_data…
[nltk_data]    |   Package conll2000 is already up-to-date!
[nltk_data]    | Downloading package conll2002 to /root/nltk_data…
[nltk_data]    |   Package conll2002 is already up-to-date!
[nltk_data]    | Downloading package conll2007 to /root/nltk_data…
[nltk_data]    |   Package conll2007 is already up-to-date!
[nltk_data]    | Downloading package crubadan to /root/nltk_data…
[nltk_data]    |   Package crubadan is already up-to-date!
[nltk_data]    | Downloading package dependency_treebank to
[nltk_data]    |     /root/nltk_data…
[nltk_data]    |   Package dependency_treebank is already up-to-date!
[nltk_data]    | Downloading package dolch to /root/nltk_data…
[nltk_data]    |   Package dolch is already up-to-date!
[nltk_data]    | Downloading package europarl_raw to
[nltk_data]    |     /root/nltk_data…
[nltk_data]    |   Package europarl_raw is already up-to-date!
[nltk_data]    | Downloading package extended_omw to
[nltk_data]    |     /root/nltk_data…
[nltk_data]    |   Package extended_omw is already up-to-date!
[nltk_data]    | Downloading package floresta to /root/nltk_data…
[nltk_data]    |   Package floresta is already up-to-date!
[nltk_data]    | Downloading package framenet_v15 to
[nltk_data]    |     /root/nltk_data…
[nltk_data]    |   Package framenet_v15 is already up-to-date!
[nltk_data]    | Downloading package framenet_v17 to
```

```
[nltk_data]    |     /root/nltk_data…
[nltk_data]    |   Package framenet_v17 is already up-to-date!
[nltk_data]    | Downloading package gazetteers to /root/nltk_data…
[nltk_data]    |   Package gazetteers is already up-to-date!
[nltk_data]    | Downloading package genesis to /root/nltk_data…
[nltk_data]    |   Package genesis is already up-to-date!
[nltk_data]    | Downloading package gutenberg to /root/nltk_data…
[nltk_data]    |   Package gutenberg is already up-to-date!
[nltk_data]    | Downloading package ieer to /root/nltk_data…
[nltk_data]    |   Package ieer is already up-to-date!
[nltk_data]    | Downloading package inaugural to /root/nltk_data…
[nltk_data]    |   Package inaugural is already up-to-date!
[nltk_data]    | Downloading package indian to /root/nltk_data…
[nltk_data]    |   Package indian is already up-to-date!
[nltk_data]    | Downloading package jeita to /root/nltk_data…
[nltk_data]    |   Package jeita is already up-to-date!
[nltk_data]    | Downloading package kimmo to /root/nltk_data…
[nltk_data]    |   Package kimmo is already up-to-date!
[nltk_data]    | Downloading package knbc to /root/nltk_data…
[nltk_data]    |   Package knbc is already up-to-date!
[nltk_data]    | Downloading package large_grammars to
[nltk_data]    |     /root/nltk_data…
[nltk_data]    |   Package large_grammars is already up-to-date!
[nltk_data]    | Downloading package lin_thesaurus to
[nltk_data]    |     /root/nltk_data…
[nltk_data]    |   Package lin_thesaurus is already up-to-date!
[nltk_data]    | Downloading package mac_morpho to /root/nltk_data…
[nltk_data]    |   Package mac_morpho is already up-to-date!
[nltk_data]    | Downloading package machado to /root/nltk_data…
[nltk_data]    |   Package machado is already up-to-date!
[nltk_data]    | Downloading package masc_tagged to /root/nltk_data…
[nltk_data]    |   Package masc_tagged is already up-to-date!
[nltk_data]    | Downloading package maxent_ne_chunker to
[nltk_data]    |     /root/nltk_data…
[nltk_data]    |   Package maxent_ne_chunker is already up-to-date!
[nltk_data]    | Downloading package maxent_treebank_pos_tagger to
[nltk_data]    |     /root/nltk_data…
[nltk_data]    |   Package maxent_treebank_pos_tagger is already up-
[nltk_data]    |       to-date!
[nltk_data]    | Downloading package moses_sample to
[nltk_data]    |     /root/nltk_data…
[nltk_data]    |   Package moses_sample is already up-to-date!
[nltk_data]    | Downloading package movie_reviews to
[nltk_data]    |     /root/nltk_data…
[nltk_data]    |   Package movie_reviews is already up-to-date!
[nltk_data]    | Downloading package mte_teip5 to /root/nltk_data…
[nltk_data]    |   Package mte_teip5 is already up-to-date!
[nltk_data]    | Downloading package mwa_ppdb to /root/nltk_data…
```

```
[nltk_data]    |   Package mwa_ppdb is already up-to-date!
[nltk_data]    | Downloading package names to /root/nltk_data…
[nltk_data]    |   Package names is already up-to-date!
[nltk_data]    | Downloading package nombank.1.0 to /root/nltk_data…
[nltk_data]    |   Package nombank.1.0 is already up-to-date!
[nltk_data]    | Downloading package nonbreaking_prefixes to
[nltk_data]    |     /root/nltk_data…
[nltk_data]    |   Package nonbreaking_prefixes is already up-to-date!
[nltk_data]    | Downloading package nps_chat to /root/nltk_data…
[nltk_data]    |   Package nps_chat is already up-to-date!
[nltk_data]    | Downloading package omw to /root/nltk_data…
[nltk_data]    |   Package omw is already up-to-date!
[nltk_data]    | Downloading package omw-1.4 to /root/nltk_data…
[nltk_data]    |   Package omw-1.4 is already up-to-date!
[nltk_data]    | Downloading package opinion_lexicon to
[nltk_data]    |     /root/nltk_data…
[nltk_data]    |   Package opinion_lexicon is already up-to-date!
[nltk_data]    | Downloading package panlex_swadesh to
[nltk_data]    |     /root/nltk_data…
[nltk_data]    |   Package panlex_swadesh is already up-to-date!
[nltk_data]    | Downloading package paradigms to /root/nltk_data…
[nltk_data]    |   Package paradigms is already up-to-date!
[nltk_data]    | Downloading package pe08 to /root/nltk_data…
[nltk_data]    |   Package pe08 is already up-to-date!
[nltk_data]    | Downloading package perluniprops to
[nltk_data]    |     /root/nltk_data…
[nltk_data]    |   Package perluniprops is already up-to-date!
[nltk_data]    | Downloading package pil to /root/nltk_data…
[nltk_data]    |   Package pil is already up-to-date!
[nltk_data]    | Downloading package pl196x to /root/nltk_data…
[nltk_data]    |   Package pl196x is already up-to-date!
[nltk_data]    | Downloading package porter_test to /root/nltk_data…
[nltk_data]    |   Package porter_test is already up-to-date!
[nltk_data]    | Downloading package ppattach to /root/nltk_data…
[nltk_data]    |   Package ppattach is already up-to-date!
[nltk_data]    | Downloading package problem_reports to
[nltk_data]    |     /root/nltk_data…
[nltk_data]    |   Package problem_reports is already up-to-date!
[nltk_data]    | Downloading package product_reviews_1 to
[nltk_data]    |     /root/nltk_data…
[nltk_data]    |   Package product_reviews_1 is already up-to-date!
[nltk_data]    | Downloading package product_reviews_2 to
[nltk_data]    |     /root/nltk_data…
[nltk_data]    |   Package product_reviews_2 is already up-to-date!
[nltk_data]    | Downloading package propbank to /root/nltk_data…
[nltk_data]    |   Package propbank is already up-to-date!
[nltk_data]    | Downloading package pros_cons to /root/nltk_data…
[nltk_data]    |   Package pros_cons is already up-to-date!
```

```
[nltk_data]    | Downloading package ptb to /root/nltk_data…
[nltk_data]    |   Package ptb is already up-to-date!
[nltk_data]    | Downloading package punkt to /root/nltk_data…
[nltk_data]    |   Package punkt is already up-to-date!
[nltk_data]    | Downloading package qc to /root/nltk_data…
[nltk_data]    |   Package qc is already up-to-date!
[nltk_data]    | Downloading package reuters to /root/nltk_data…
[nltk_data]    |   Package reuters is already up-to-date!
[nltk_data]    | Downloading package rslp to /root/nltk_data…
[nltk_data]    |   Package rslp is already up-to-date!
[nltk_data]    | Downloading package rte to /root/nltk_data…
[nltk_data]    |   Package rte is already up-to-date!
[nltk_data]    | Downloading package sample_grammars to
[nltk_data]    |     /root/nltk_data…
[nltk_data]    |   Package sample_grammars is already up-to-date!
[nltk_data]    | Downloading package semcor to /root/nltk_data…
[nltk_data]    |   Package semcor is already up-to-date!
[nltk_data]    | Downloading package senseval to /root/nltk_data…
[nltk_data]    |   Package senseval is already up-to-date!
[nltk_data]    | Downloading package sentence_polarity to
[nltk_data]    |     /root/nltk_data…
[nltk_data]    |   Package sentence_polarity is already up-to-date!
[nltk_data]    | Downloading package sentiwordnet to
[nltk_data]    |     /root/nltk_data…
[nltk_data]    |   Package sentiwordnet is already up-to-date!
[nltk_data]    | Downloading package shakespeare to /root/nltk_data…
[nltk_data]    |   Package shakespeare is already up-to-date!
[nltk_data]    | Downloading package sinica_treebank to
[nltk_data]    |     /root/nltk_data…
[nltk_data]    |   Package sinica_treebank is already up-to-date!
[nltk_data]    | Downloading package smultron to /root/nltk_data…
[nltk_data]    |   Package smultron is already up-to-date!
[nltk_data]    | Downloading package snowball_data to
[nltk_data]    |     /root/nltk_data…
[nltk_data]    |   Package snowball_data is already up-to-date!
[nltk_data]    | Downloading package spanish_grammars to
[nltk_data]    |     /root/nltk_data…
[nltk_data]    |   Package spanish_grammars is already up-to-date!
[nltk_data]    | Downloading package state_union to /root/nltk_data…
[nltk_data]    |   Package state_union is already up-to-date!
[nltk_data]    | Downloading package stopwords to /root/nltk_data…
[nltk_data]    |   Package stopwords is already up-to-date!
[nltk_data]    | Downloading package subjectivity to
[nltk_data]    |     /root/nltk_data…
[nltk_data]    |   Package subjectivity is already up-to-date!
[nltk_data]    | Downloading package swadesh to /root/nltk_data…
[nltk_data]    |   Package swadesh is already up-to-date!
[nltk_data]    | Downloading package switchboard to /root/nltk_data…
```

```
[nltk_data]    |   Package switchboard is already up-to-date!
[nltk_data]    | Downloading package tagsets to /root/nltk_data…
[nltk_data]    |   Package tagsets is already up-to-date!
[nltk_data]    | Downloading package timit to /root/nltk_data…
[nltk_data]    |   Package timit is already up-to-date!
[nltk_data]    | Downloading package toolbox to /root/nltk_data…
[nltk_data]    |   Package toolbox is already up-to-date!
[nltk_data]    | Downloading package treebank to /root/nltk_data…
[nltk_data]    |   Package treebank is already up-to-date!
[nltk_data]    | Downloading package twitter_samples to
[nltk_data]    |     /root/nltk_data…
[nltk_data]    |   Package twitter_samples is already up-to-date!
[nltk_data]    | Downloading package udhr to /root/nltk_data…
[nltk_data]    |   Package udhr is already up-to-date!
[nltk_data]    | Downloading package udhr2 to /root/nltk_data…
[nltk_data]    |   Package udhr2 is already up-to-date!
[nltk_data]    | Downloading package unicode_samples to
[nltk_data]    |     /root/nltk_data…
[nltk_data]    |   Package unicode_samples is already up-to-date!
[nltk_data]    | Downloading package universal_tagset to
[nltk_data]    |     /root/nltk_data…
[nltk_data]    |   Package universal_tagset is already up-to-date!
[nltk_data]    | Downloading package universal_treebanks_v20 to
[nltk_data]    |     /root/nltk_data…
[nltk_data]    |   Package universal_treebanks_v20 is already up-to-
[nltk_data]    |       date!
[nltk_data]    | Downloading package vader_lexicon to
[nltk_data]    |     /root/nltk_data…
[nltk_data]    |   Package vader_lexicon is already up-to-date!
[nltk_data]    | Downloading package verbnet to /root/nltk_data…
[nltk_data]    |   Package verbnet is already up-to-date!
[nltk_data]    | Downloading package verbnet3 to /root/nltk_data…
[nltk_data]    |   Package verbnet3 is already up-to-date!
[nltk_data]    | Downloading package webtext to /root/nltk_data…
[nltk_data]    |   Package webtext is already up-to-date!
[nltk_data]    | Downloading package wmt15_eval to /root/nltk_data…
[nltk_data]    |   Package wmt15_eval is already up-to-date!
[nltk_data]    | Downloading package word2vec_sample to
[nltk_data]    |     /root/nltk_data…
[nltk_data]    |   Package word2vec_sample is already up-to-date!
[nltk_data]    | Downloading package wordnet to /root/nltk_data…
[nltk_data]    |   Package wordnet is already up-to-date!
[nltk_data]    | Downloading package wordnet2021 to /root/nltk_data…
[nltk_data]    |   Package wordnet2021 is already up-to-date!
[nltk_data]    | Downloading package wordnet31 to /root/nltk_data…
[nltk_data]    |   Package wordnet31 is already up-to-date!
[nltk_data]    | Downloading package wordnet_ic to /root/nltk_data…
[nltk_data]    |   Package wordnet_ic is already up-to-date!
```

```
[nltk_data]    | Downloading package words to /root/nltk_data…
[nltk_data]    |   Package words is already up-to-date!
[nltk_data]    | Downloading package ycoe to /root/nltk_data…
[nltk_data]    |   Package ycoe is already up-to-date!
[nltk_data]    |
[nltk_data]  Done downloading collection all
```

[ ]:
```
tweetFiles = ["./ECE219_tweet_data/tweets_#superbowl.txt",
              "./ECE219_tweet_data/tweets_#nfl.txt",
              "./ECE219_tweet_data/tweets_#gohawks.txt",
              "./ECE219_tweet_data/tweets_#gopatriots.txt",
              "./ECE219_tweet_data/tweets_#patriots.txt",
              "./ECE219_tweet_data/tweets_#sb49.txt"]
```

[ ]:
```
# Explore and Test Key of Data
with open(tweetFiles[3], 'r', encoding="utf8") as file:
    first_line = file.readline()
    json_obj = json.loads(first_line)

    print('=' * 50)
    for key in json_obj:
        print(key)

    for key in json_obj:
        print('\n' + '-' * 50)
        print(key + ':')
        print(json_obj[key])
```

```
==================================================
firstpost_date
title
url
tweet
author
original_author
citation_date
metrics
highlight
type
citation_url

--------------------------------------------------
firstpost_date:
1420835445

--------------------------------------------------
title:
LeGarrette Blount does the Ray Lewis Dance #ThrowbackThursday
```

https://t.co/F5FX5KVmdX
Hope to see it at least 3 times tomorrow.
#GoPatriots

----
url:
http://twitter.com/NESportsFan1106/status/553650101105606656

----
tweet:
{'contributors': None, 'truncated': False, 'text': 'LeGarrette Blount does the Ray Lewis Dance #ThrowbackThursday https://t.co/F5FX5KVmdX\nHope to see it at least 3 times tomorrow. \n#GoPatriots', 'in_reply_to_status_id': None, 'id': 553650101105606656, 'favorite_count': 0, 'source': 'Twitter for Android', 'retweeted': False, 'coordinates': None, 'timestamp_ms': '1420835445082', 'entities': {'symbols': [], 'user_mentions': [], 'trends': [], 'hashtags': [{'indices': [43, 61], 'text': 'ThrowbackThursday'}, {'indices': [129, 140], 'text': 'GoPatriots'}], 'urls': [{'indices': [62, 85], 'url': 'https://t.co/F5FX5KVmdX', 'expanded_url': 'https://vine.co/v/OphHATaAggX', 'display_url': 'vine.co/v/OphHATaAggX'}]}, 'in_reply_to_screen_name': None, 'in_reply_to_user_id': None, 'retweet_count': 0, 'id_str': '553650101105606656', 'favorited': False, 'user': {'follow_request_sent': None, 'profile_use_background_image': True, 'geo_enabled': True, 'description': 'Follow me for News on Boston Sports Teams! #GoPatriots #GoRedSox #GoBruins #GoCeltics\r\nHuge Boston Sports Fan', 'verified': False, 'profile_image_url_https': 'https://pbs.twimg.com/profile_images/551201267783983105/wWelNsyR_normal.jpeg', 'profile_sidebar_fill_color': 'DDEEF6', 'is_translator': False, 'id': 753321800, 'profile_text_color': '333333', 'followers_count': 3317, 'profile_sidebar_border_color': 'C0DEED', 'id_str': '753321800', 'default_profile_image': False, 'location': 'Boston, Massachusetts', 'utc_offset': -18000, 'statuses_count': 31129, 'profile_background_color': 'C0DEED', 'friends_count': 2948, 'profile_link_color': '0084B4', 'profile_image_url': 'http://pbs.twimg.com/profile_images/551201267783983105/wWelNsyR_normal.jpeg', 'notifications': None, 'profile_background_image_url_https': 'https://abs.twimg.com/images/themes/theme1/bg.png', 'profile_banner_url': 'https://pbs.twimg.com/profile_banners/753321800/1395953837', 'profile_background_image_url': 'http://abs.twimg.com/images/themes/theme1/bg.png', 'name': 'Boston Sports Fan', 'lang': 'en', 'profile_background_tile': False, 'favourites_count': 8660, 'screen_name': 'NESportsFan1106', 'url': None, 'created_at': 'Sun Aug 12 15:43:27 +0000 2012', 'contributors_enabled': False, 'time_zone': 'Eastern Time (US & Canada)', 'protected': False, 'default_profile': True, 'following': None, 'listed_count': 42}, 'geo': None, 'in_reply_to_user_id_str': None, 'possibly_sensitive': False, 'lang': 'en', 'created_at': 'Fri Jan 09 20:30:45 +0000 2015', 'filter_level': 'medium', 'in_reply_to_status_id_str': None,

'place': None}

---
author:
{'author_img':
'http://pbs.twimg.com/profile_images/562109096519012352/49vxe_9q_normal.jpeg',
'name': 'Alex Kroll', 'url': 'http://twitter.com/patsnation87', 'nick':
'patsnation87', 'followers': 2895.0, 'image_url':
'http://pbs.twimg.com/profile_images/562109096519012352/49vxe_9q_normal.jpeg',
'type': 'twitter', 'description': "I'm a die-hard Patriots, Suns, Badgers and
Brewers fan #TeamPatriots | #PatsNation | #FuelTheFire | #GoSuns | #Brewcrew |
#Bucks | #FeerTheDeer"}

---
original_author:
{'author_img':
'http://pbs.twimg.com/profile_images/551201267783983105/wWelNsyR_normal.jpeg',
'description': 'Follow me for News on Boston Sports Teams! #GoPatriots
#GoRedSox #GoBruins #GoCeltics\r\nHuge Boston Sports Fan', 'url':
'http://twitter.com/nesportsfan1106', 'nick': 'nesportsfan1106', 'followers':
3811.0, 'image_url':
'http://pbs.twimg.com/profile_images/551201267783983105/wWelNsyR_normal.jpeg',
'type': 'twitter', 'name': 'Boston Sports Fan'}

---
citation_date:
1421257541

---
metrics:
{'acceleration': 0, 'ranking_score': 4.6402764, 'citations': {'influential': 0,
'total': 6, 'data': [{'timestamp': 1421257499, 'citations': 1}], 'matching': 1,
'replies': 0}, 'peak': 1421257559, 'impressions': 2695, 'momentum': 1}

---
highlight:
LeGarrette Blount does the Ray Lewis Dance #ThrowbackThursday
https://t.co/F5FX5KVmdX
Hope to see it at least 3 times tomorrow.
#GoPatriots

---
type:
retweet:native

---
citation_url:
http://twitter.com/Patsnation87/status/555420502970609664

2 Question 27

[ ]:
```
max_time_seconds = {tweetFiles[0]: -1,
                    tweetFiles[1]: -1,
                    tweetFiles[2]: -1,
                    tweetFiles[3]: -1,
                    tweetFiles[4]: -1,
                    tweetFiles[5]: -1}

min_time_seconds = {tweetFiles[0]: -1,
                    tweetFiles[1]: -1,
                    tweetFiles[2]: -1,
                    tweetFiles[3]: -1,
                    tweetFiles[4]: -1,
                    tweetFiles[5]: -1}
```

[ ]:
```
def report(tweetFile):
    with open(tweetFile, 'r', encoding="utf8") as file:
        lines = file.readlines()

    num_tweets = 0
    time_seconds, num_followers, num_retweets = [], [], []
    for line in lines:
        json_object = json.loads(line)
        num_tweets += 1
        time_seconds.append(json_object['citation_date'])
        num_followers.append(json_object['author']['followers'])
        num_retweets.append(json_object['metrics']['citations']['total'])

    max_time_seconds[tweetFile] = max(time_seconds)
    min_time_seconds[tweetFile] = min(time_seconds)

    print("Statistics of {}".format(tweetFile))
    print("Average number of tweets per hour: {}".format(num_tweets /␣
 ↪((max_time_seconds[tweetFile]-min_time_seconds[tweetFile]) / (60*60))))
    print("Average number of followers of users posting the tweets per tweet:␣
 ↪{}".format(sum(num_followers) / num_tweets))
    print("Average number of retweets per tweet: {}".format(sum(num_retweets) /␣
 ↪num_tweets))
    print('')
```

[ ]:
```
for tweetFile in tweetFiles:
    report(tweetFile)
```

```
Statistics of ./ECE219_tweet_data/tweets_#superbowl.txt
Average number of tweets per hour: 2072.11840170408
Average number of followers of users posting the tweets per tweet:
8814.96799424623
```

```
Average number of retweets per tweet: 2.3911895819207736

Statistics of ./ECE219_tweet_data/tweets_#nfl.txt
Average number of tweets per hour: 397.0213901819841
Average number of followers of users posting the tweets per tweet:
4662.37544523693
Average number of retweets per tweet: 1.5344602655543254

Statistics of ./ECE219_tweet_data/tweets_#gohawks.txt
Average number of tweets per hour: 292.48785062173687
Average number of followers of users posting the tweets per tweet:
2217.9237355281984
Average number of retweets per tweet: 2.0132093991319877

Statistics of ./ECE219_tweet_data/tweets_#gopatriots.txt
Average number of tweets per hour: 40.95469800606194
Average number of followers of users posting the tweets per tweet:
1427.2526051635405
Average number of retweets per tweet: 1.4081919101697078

Statistics of ./ECE219_tweet_data/tweets_#patriots.txt
Average number of tweets per hour: 750.89426460689
Average number of followers of users posting the tweets per tweet:
3280.4635616550277
Average number of retweets per tweet: 1.7852871288476946

Statistics of ./ECE219_tweet_data/tweets_#sb49.txt
Average number of tweets per hour: 1276.8570598680474
Average number of followers of users posting the tweets per tweet:
10374.160292019487
Average number of retweets per tweet: 2.52713444111402
```

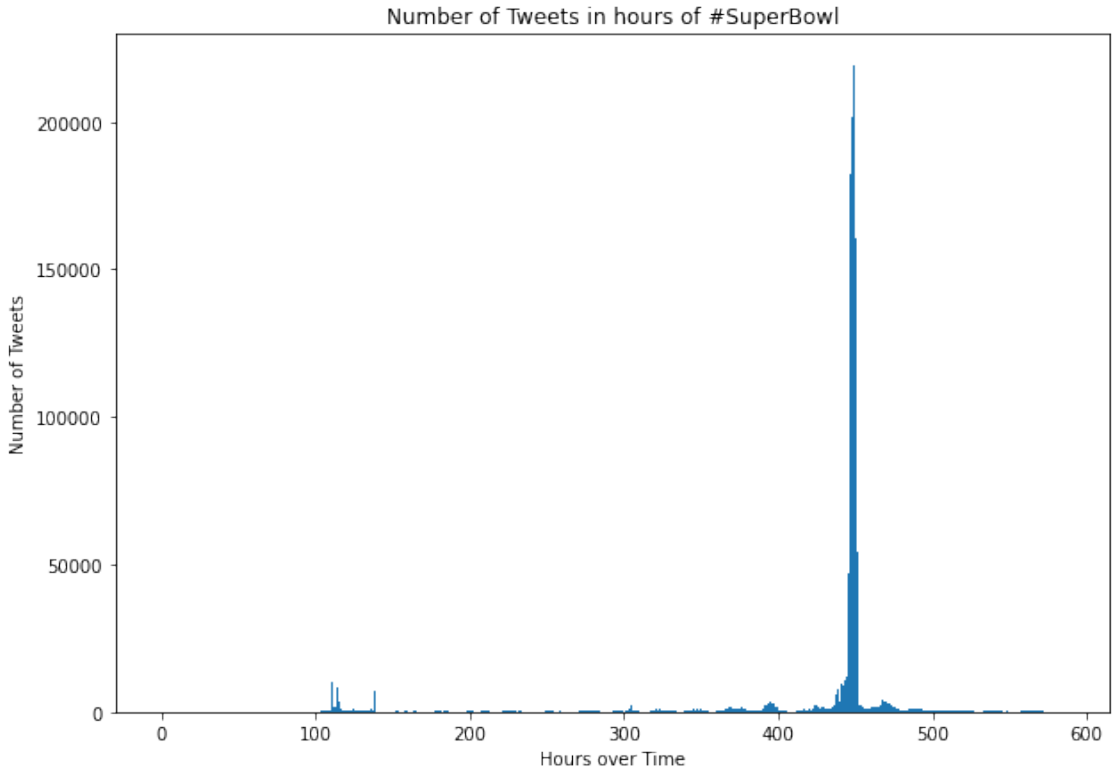
3 Question 28

[ ]:
```python
def plot_tweets_hour(tweetFile, title):
    with open(tweetFile, 'r', encoding="utf8") as file:
        lines = file.readlines()

    max_seconds = max_time_seconds[tweetFile]
    min_seconds = min_time_seconds[tweetFile]

    total_hours = int((max_seconds-min_seconds) / (60*60) + 1)
    x = range(total_hours)
    y = [0] * total_hours

    for line in lines:
```

```
        json_object = json.loads(line)
        y[int((json_object['citation_date']-min_seconds) / (60*60))] += 1

    plt.title("Number of Tweets in hours of " + title)
    plt.bar(x, y, 1)
    plt.xlabel("Hours over Time")
    plt.ylabel("Number of Tweets")
    plt.gcf().set_size_inches(10, 7)
    plt.show()
```

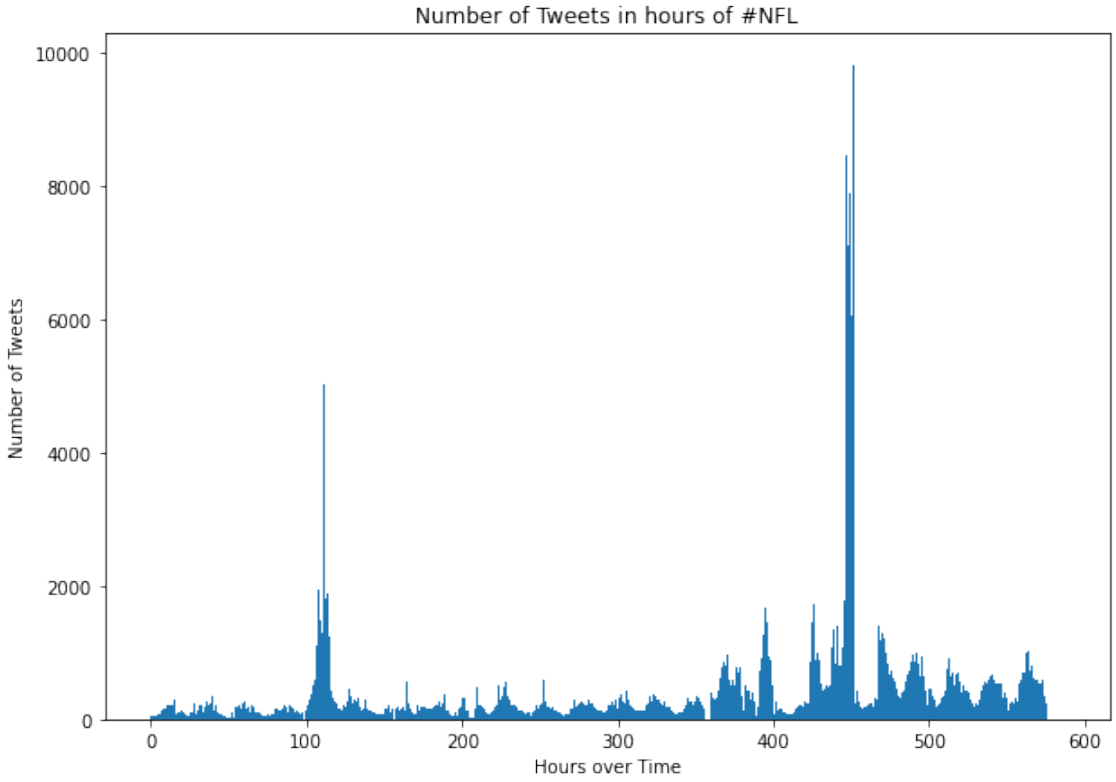
- #SuperBowl

[ ]:
```
plot_tweets_hour(tweetFiles[0], "#SuperBowl")
```



- #NFL

[ ]:
```
plot_tweets_hour(tweetFiles[1], "#NFL")
```



4 Question 29

- Describe out task: Library of Prediction Tasks given a tweet

1. Predict the hashtags.
2. Predict the number of retweets.
3. Predict the number of likes.
4. Predict the number of quotes.
5. Predict the relative(UNIX) time when a tweet was posted.
6. Creativity - Predict the number of followers of the person tweeting.

- Explore the data

1. Because all the six datasets are too big and we don't have great resource of devices, we decide to choose the tweets posted during the Superbowl competition, 1367089 samples.
2. Superbowl Start Time we chose: 1422831600, 02/01/2015 3:00 pm PST
3. Superbowl End Time we chose: 1422849600, 02/01/2015 8:00 pm PST
4. Split into the training data 80%, 1093671 samples, and the testing data 20%, 273418 samples.

- Extract features

1. Extract the text data in the tweet. After cleaning, we use lemmatization or stemming (we will compare the results of these two). - Project1

2. Notice that the hashtags in the text will be removed by both lemmatization or stemming, so it will not be included in the features.
3. Use the Vectorizer and TfidfTransformer to convert the text to value. - Project1
4. Use TruncatedSVD to do dimensionality recudtion, dim: 4700 -> 150. Because of the resource of our devices, the limitation is 150. If we can do more than 150, the performance may increase. - Project1
5. Extract other features which will be useful, e.g. retweets_count, likes_count, quotes_count, followers_count, relative_time. It is because we think that they all may have positive correlation. At some specific time, for example: scoring, the retweet or likes counts will also increase.

[ ]:
```
# Superbowl Start Time: 1422833400 02/01/2015 3:30 pm PST
# Superbowl End Time:   1422846420 02/01/2015 7:07 pm PST

# Superbowl Start Time: 1422831600 02/01/2015 3:00 pm PST
# Superbowl End Time:   1422849600 02/01/2015 8:00 pm PST
```

[ ]:
```
hashtags_labels = {0: '#superbowl', 1: '#nfl', 2: '#gohawks', 3: '#gopatriots',␣
↪4: '#patriots', 5: '#sb49'}
superbowl_time = {'start': 1422831600, 'end': 1422849600}

# Extract features
def extractData(tweetFiles):
    data = []
    y = []
    y_retweet = []
    y_like = []
    y_quote = []
    y_time = []
    follower_1 = []
    follower_2 = []

    for index, tweetFile in enumerate(tweetFiles):
        with open(tweetFile, 'r', encoding="utf8") as file:
            lines = file.readlines()
            for line in lines:
                json_object = json.loads(line)
                time = json_object['citation_date']
                if time < superbowl_time['start'] or time >␣
↪superbowl_time['end']:
                    continue
                text = json_object['tweet']['text'].lower()
                retweet = json_object['tweet']['retweet_count']
                like = json_object['tweet']['favorite_count']
                quote = json_object['metrics']['citations']['total']
                follower1 = json_object['author']['followers']
                follower2 = json_object['tweet']['user']['followers_count']
```

```
                data.append(text)
                y.append(index)
                y_retweet.append(retweet)
                y_like.append(like)
                y_quote.append(quote)
                y_time.append(time - superbowl_time['start'])
                follower_1.append(follower1)
                follower_2.append(follower2)

    return np.array(data), np.array(y), np.array(y_retweet).astype('float64'),␣
 ↪np.array(y_like).astype('float64'), np.array(y_quote).astype('float64'), np.
 ↪array(y_time).astype('float64'), np.array(follower_1).astype('float64'), np.
 ↪array(follower_2).astype('float64')
```

[ ]:
```
data, y, y_retweet, y_like, y_quote, y_time, follower_1, follower_2 =␣
 ↪extractData(tweetFiles)
```

[ ]:
```
print(data.shape)
print(y.shape)
```

```
(1367089,)
(1367089,)
```

[ ]:
```
# Save
pd.DataFrame(data).to_csv("./ECE219_tweet_data/data.csv", header=None,␣
 ↪index=None)

pd.DataFrame(y).to_csv("./ECE219_tweet_data/y.csv", header=None, index=None)

pd.DataFrame(y_retweet).to_csv("./ECE219_tweet_data/y_retweet.csv",␣
 ↪header=None, index=None)

pd.DataFrame(y_like).to_csv("./ECE219_tweet_data/y_like.csv", header=None,␣
 ↪index=None)

pd.DataFrame(y_quote).to_csv("./ECE219_tweet_data/y_quote.csv", header=None,␣
 ↪index=None)

pd.DataFrame(y_time).to_csv("./ECE219_tweet_data/y_time.csv", header=None,␣
 ↪index=None)

pd.DataFrame(follower_1).to_csv("./ECE219_tweet_data/follower_1.csv",␣
 ↪header=None, index=None)

pd.DataFrame(follower_2).to_csv("./ECE219_tweet_data/follower_2.csv",␣
 ↪header=None, index=None)
```

[ ]:
```
X_train, X_test, y_train, y_test, y_retweet_train, y_retweet_test,␣
↪y_like_train, y_like_test, y_quote_train, y_quote_test, y_time_train,␣
↪y_time_test, follower_1_train, follower_1_test, follower_2_train,␣
↪follower_2_test = train_test_split(data, y, y_retweet, y_like, y_quote,␣
↪y_time, follower_1, follower_2, test_size=0.2, random_state=42, shuffle=True)
```

[ ]:
```
print("X: training samples = "+str(len(X_train))+"\n"+"X: test samples =␣
↪"+str(len(X_test)))
print("y: training samples = "+str(len(y_train))+"\n"+"y: test samples =␣
↪"+str(len(y_test)))
```

```
X: training samples = 1093671
X: test samples = 273418
y: training samples = 1093671
y: test samples = 273418
```

[ ]:
```
# Save
pd.DataFrame(X_train).to_csv("./ECE219_tweet_data/X_train.csv", header=None,␣
↪index=None)
pd.DataFrame(X_test).to_csv("./ECE219_tweet_data/X_test.csv", header=None,␣
↪index=None)

pd.DataFrame(y_train).to_csv("./ECE219_tweet_data/y_train.csv", header=None,␣
↪index=None)
pd.DataFrame(y_test).to_csv("./ECE219_tweet_data/y_test.csv", header=None,␣
↪index=None)

pd.DataFrame(y_retweet_train).to_csv("./ECE219_tweet_data/y_retweet_train.csv",␣
↪header=None, index=None)
pd.DataFrame(y_retweet_test).to_csv("./ECE219_tweet_data/y_retweet_test.csv",␣
↪header=None, index=None)

pd.DataFrame(y_like_train).to_csv("./ECE219_tweet_data/y_like_train.csv",␣
↪header=None, index=None)
pd.DataFrame(y_like_test).to_csv("./ECE219_tweet_data/y_like_test.csv",␣
↪header=None, index=None)

pd.DataFrame(y_quote_train).to_csv("./ECE219_tweet_data/y_quote_train.csv",␣
↪header=None, index=None)
pd.DataFrame(y_quote_test).to_csv("./ECE219_tweet_data/y_quote_test.csv",␣
↪header=None, index=None)

pd.DataFrame(y_time_train).to_csv("./ECE219_tweet_data/y_time_train.csv",␣
↪header=None, index=None)
pd.DataFrame(y_time_test).to_csv("./ECE219_tweet_data/y_time_test.csv",␣
↪header=None, index=None)
```

```
pd.DataFrame(follower_1_train).to_csv("./ECE219_tweet_data/follower_1_train.
 ↪csv", header=None, index=None)
pd.DataFrame(follower_1_test).to_csv("./ECE219_tweet_data/follower_1_test.csv",␣
 ↪header=None, index=None)

pd.DataFrame(follower_2_train).to_csv("./ECE219_tweet_data/follower_2_train.
 ↪csv", header=None, index=None)
pd.DataFrame(follower_2_test).to_csv("./ECE219_tweet_data/follower_2_test.csv",␣
 ↪header=None, index=None)
```

[ ]:
```
###Cleaning Preprocessor
#Cleans the data, removes raw input other than text
def clean(text):
    text = re.sub(r'^https?:\/\/.*[\r\n]*', '', text, flags=re.MULTILINE)
    text = re.sub(r'https?:\/\/.*[\r\n]*', '', text, flags=re.MULTILINE)
    text = re.sub(r'www.*[\r\n]*', '', text, flags=re.MULTILINE)
    text = re.sub(r'\S*@\S*\s?', '', text, flags=re.MULTILINE) #remove email␣
 ↪address
    text=re.sub(r'\w*\d+\w*', '', text).strip() #remove any word with numbers
    texter = re.sub(r"<br />", " ", text)
    texter = re.sub(r"&quot;", "\"",texter)
    texter = re.sub('&#39;', "\"", texter)
    texter = re.sub('\n', " ", texter)
    texter = re.sub('\__', "", texter)
    texter = re.sub('\|', "", texter)
    texter = re.sub('\/', " ", texter)
    texter = re.sub('--', "", texter)
    texter = re.sub(r'\d+', '', texter) #remove word with just numbers
    texter = re.sub(' u '," you ", texter)
    texter = re.sub(' +', ' ', texter)
    texter = re.sub(r"(!)\1+", r"!", texter)
    texter = re.sub(r"(\?)\1+", r"?", texter)
    texter = re.sub(' &amp; ', ' and ', texter)
    texter = re.sub('\r', ' ',texter)
    clean = re.compile('<.*?>')
    texter = texter.encode('ascii', 'ignore').decode('ascii')
    texter = re.sub(clean, '', texter)
    if texter == "":
        texter = ""
    return texter

#get word pos
def get_wordnet_pos(word):
    tag = nltk.pos_tag([word])[0][1][0].upper()
    tag_dict = {"J": wordnet.ADJ,
                "N": wordnet.NOUN,
                "V": wordnet.VERB,
```

```
                "R": wordnet.ADV}

    return tag_dict.get(tag, wordnet.NOUN)

#construct vocabulary
stop_words_scikit = text.ENGLISH_STOP_WORDS
stop_words_nltk = stopwords.words('english')
stop_words = set.
 ↪union(set(stop_words_scikit),set(stop_words_nltk),set(punctuation))

english_vocab = set(w.lower() for w in nltk.corpus.words.words())
```

[ ]:
```
###Lemmatization & Stemming (Reference: https://www.machinelearningplus.com/nlp/
 ↪lemmatization-examples-python/)
#Check each word inside each of the full_text corpus, makes words lower-case,␣
 ↪removes non-English words
#converts words to its simplest form i.e plural to singluar, etc using␣
 ↪lemmatizer
#appends that word, joins them into a sentence and adds it to␣
 ↪lemmatized_full_text

cleaned_X_train = []
cleaned_X_test = []

for row in range(X_train.shape[0]):
    cleaned_X_train.append(clean(X_train[row]))

for row in range(X_test.shape[0]):
    cleaned_X_test.append(clean(X_test[row]))

lemmatizer = WordNetLemmatizer().lemmatize
ls = LS()

lemmatized_X_train = []
lemmatized_X_test = []
stemmed_X_train = []
stemmed_X_test = []

for full_text in cleaned_X_train:
    wordlist_lemmatized = []
    wordlist_stemmed = []
    for word in nltk.word_tokenize(full_text):
        word=word.lower()
        if (word in english_vocab) and word.isalpha() and ((word in stop_words)␣
 ↪== False):
            wordlist_lemmatized.append(lemmatizer(word, get_wordnet_pos(word)))
            wordlist_stemmed.append(ls.stem(word))
```

```
    lemmatized_X_train.append(' '.join(wordlist_lemmatized))
    stemmed_X_train.append(' '.join(wordlist_stemmed))

for full_text in cleaned_X_test:
    wordlist_lemmatized = []
    wordlist_stemmed = []
    for word in nltk.word_tokenize(full_text):
        word=word.lower()
        if (word in english_vocab) and word.isalpha() and ((word in stop_words)␣
 ↪== False):
            wordlist_lemmatized.append(lemmatizer(word, get_wordnet_pos(word)))
            wordlist_stemmed.append(ls.stem(word))
    lemmatized_X_test.append(' '.join(wordlist_lemmatized))
    stemmed_X_test.append(' '.join(wordlist_stemmed))
```

[ ]:
```
# Save
with open("./ECE219_tweet_data/lemmatized_X_train.csv", 'w') as f:
    write = csv.writer(f)
    write.writerows(lemmatized_X_train)

with open("./ECE219_tweet_data/lemmatized_X_test.csv", 'w') as f:
    write = csv.writer(f)
    write.writerows(lemmatized_X_test)

with open("./ECE219_tweet_data/stemmed_X_train.csv", 'w') as f:
    write = csv.writer(f)
    write.writerows(stemmed_X_train)

with open("./ECE219_tweet_data/stemmed_X_test.csv", 'w') as f:
    write = csv.writer(f)
    write.writerows(stemmed_X_test)

with open("./ECE219_tweet_data/cleaned_X_train.csv", 'w') as f:
    write = csv.writer(f)
    write.writerows(cleaned_X_train)

with open("./ECE219_tweet_data/cleaned_X_test.csv", 'w') as f:
    write = csv.writer(f)
    write.writerows(cleaned_X_test)
```

[ ]:
```
# lemmatized
tfidf_vectorizer = TfidfVectorizer(min_df=30, stop_words='english')

X_train_tfidf = tfidf_vectorizer.fit_transform(cudf.Series(lemmatized_X_train)).
 ↪toarray()
X_test_tfidf = tfidf_vectorizer.transform(cudf.Series(lemmatized_X_test)).
 ↪toarray()
```

[ ]:
```
# # stemmed
# tfidf_vectorizer = TfidfVectorizer(min_df=30, stop_words='english')

# X_train_tfidf = tfidf_vectorizer.fit_transform(cudf.Series(stemmed_X_train)).
 ↪toarray()
# X_test_tfidf = tfidf_vectorizer.transform(cudf.Series(stemmed_X_test)).
 ↪toarray()
```

[ ]:
```
# LSI
svd = TruncatedSVD(n_components = 150, random_state = 42)
X_train_lsi = svd.fit_transform(X_train_tfidf)
X_test_lsi = svd.transform(X_test_tfidf)
```

[ ]:
```
# Save - lemmatized
pd.DataFrame(X_train_lsi).to_csv("./ECE219_tweet_data/X_train_lsi.csv",␣
 ↪header=None, index=None)

pd.DataFrame(X_test_lsi).to_csv("./ECE219_tweet_data/X_test_lsi.csv",␣
 ↪header=None, index=None)
```

[ ]:
```
# # Save - stemmed
# pd.DataFrame(X_train_lsi).to_csv("./ECE219_tweet_data/X_train_lsi_stemmed.
 ↪csv", header=None, index=None)

# pd.DataFrame(X_test_lsi).to_csv("./ECE219_tweet_data/X_test_lsi_stemmed.csv",␣
 ↪header=None, index=None)
```

[ ]:
```
# Load - lemmatized
X_train_lsi = pd.read_csv("./ECE219_tweet_data/X_train_lsi.csv", header=None)
X_train_lsi = X_train_lsi.to_numpy()

X_test_lsi = pd.read_csv("./ECE219_tweet_data/X_test_lsi.csv", header=None)
X_test_lsi = X_test_lsi.to_numpy()
```

[ ]:
```
# # Load - stemmed
# X_train_lsi = pd.read_csv("./ECE219_tweet_data/X_train_lsi_stemmed.csv",␣
 ↪header=None)
# X_train_lsi = X_train_lsi.to_numpy()

# X_test_lsi = pd.read_csv("./ECE219_tweet_data/X_test_lsi_stemmed.csv",␣
 ↪header=None)
# X_test_lsi = X_test_lsi.to_numpy()
```

[ ]:
```
print(X_train_lsi.shape)
print(X_test_lsi.shape)
```

```
(1093671, 150)
(273418, 150)
```

[ ]:
```
# Load
y = pd.read_csv("./ECE219_tweet_data/y.csv", header=None).to_numpy().ravel()
y_train = pd.read_csv("./ECE219_tweet_data/y_train.csv", header=None).
 ↪to_numpy().ravel()
y_test = pd.read_csv("./ECE219_tweet_data/y_test.csv", header=None).to_numpy().
 ↪ravel()

y_retweet = pd.read_csv("./ECE219_tweet_data/y_retweet.csv", header=None).
 ↪to_numpy().ravel().astype('float64')
y_retweet_train = pd.read_csv("./ECE219_tweet_data/y_retweet_train.csv",␣
 ↪header=None).to_numpy().ravel().astype('float64')
y_retweet_test = pd.read_csv("./ECE219_tweet_data/y_retweet_test.csv",␣
 ↪header=None).to_numpy().ravel().astype('float64')

y_like = pd.read_csv("./ECE219_tweet_data/y_like.csv", header=None).to_numpy().
 ↪ravel().astype('float64')
y_like_train = pd.read_csv("./ECE219_tweet_data/y_like_train.csv", header=None).
 ↪to_numpy().ravel().astype('float64')
y_like_test = pd.read_csv("./ECE219_tweet_data/y_like_test.csv", header=None).
 ↪to_numpy().ravel().astype('float64')

y_quote = pd.read_csv("./ECE219_tweet_data/y_quote.csv", header=None).
 ↪to_numpy().ravel().astype('float64')
y_quote_train = pd.read_csv("./ECE219_tweet_data/y_quote_train.csv",␣
 ↪header=None).to_numpy().ravel().astype('float64')
y_quote_test = pd.read_csv("./ECE219_tweet_data/y_quote_test.csv", header=None).
 ↪to_numpy().ravel().astype('float64')

y_time = pd.read_csv("./ECE219_tweet_data/y_time.csv", header=None).to_numpy().
 ↪ravel().astype('float64')
y_time_train = pd.read_csv("./ECE219_tweet_data/y_time_train.csv", header=None).
 ↪to_numpy().ravel().astype('float64')
y_time_test = pd.read_csv("./ECE219_tweet_data/y_time_test.csv", header=None).
 ↪to_numpy().ravel().astype('float64')

follower_1 = pd.read_csv("./ECE219_tweet_data/follower_1.csv", header=None).
 ↪to_numpy().ravel().astype('float64')
follower_1_train = pd.read_csv("./ECE219_tweet_data/follower_1_train.csv",␣
 ↪header=None).to_numpy().ravel().astype('float64')
follower_1_test = pd.read_csv("./ECE219_tweet_data/follower_1_test.csv",␣
 ↪header=None).to_numpy().ravel().astype('float64')

follower_2 = pd.read_csv("./ECE219_tweet_data/follower_2.csv", header=None).
 ↪to_numpy().ravel().astype('float64')
follower_2_train = pd.read_csv("./ECE219_tweet_data/follower_2_train.csv",␣
 ↪header=None).to_numpy().ravel().astype('float64')
```

```
follower_2_test = pd.read_csv("./ECE219_tweet_data/follower_2_test.csv",␣
↪header=None).to_numpy().ravel().astype('float64')
```

5 Predict the Hashtags

[ ]:
```
def plt_confusion_matrix(y_test, pred_test, txt):
    cmx_data = confusion_matrix(y_test, pred_test)
    df_cmx = pd.DataFrame(cmx_data)
    sns.heatmap(df_cmx, fmt='d', annot=True, square=True)
    plt.title('Confusion Matrix for ' + txt)
    plt.xlabel('Predicted Category')
    plt.ylabel('True Category')
    plt.show()

def arpf(y_test, pred_test):
    print("Accuracy of the testing data: " + str(accuracy_score(y_test,␣
↪pred_test)))
    print("Recall of the testing data: " + str(recall_score(y_test, pred_test,␣
↪average='micro')))
    print("Precision of the testing data: " + str(precision_score(y_test,␣
↪pred_test, average='micro')))
    print("F-1 Score of the testing data: " + str(f1_score(y_test, pred_test,␣
↪average='micro')))
```

[ ]:
```
X_train_lsi_hashtag = X_train_lsi
X_test_lsi_hashtag = X_test_lsi
```

[ ]:
```
print(Counter(y_test))
```

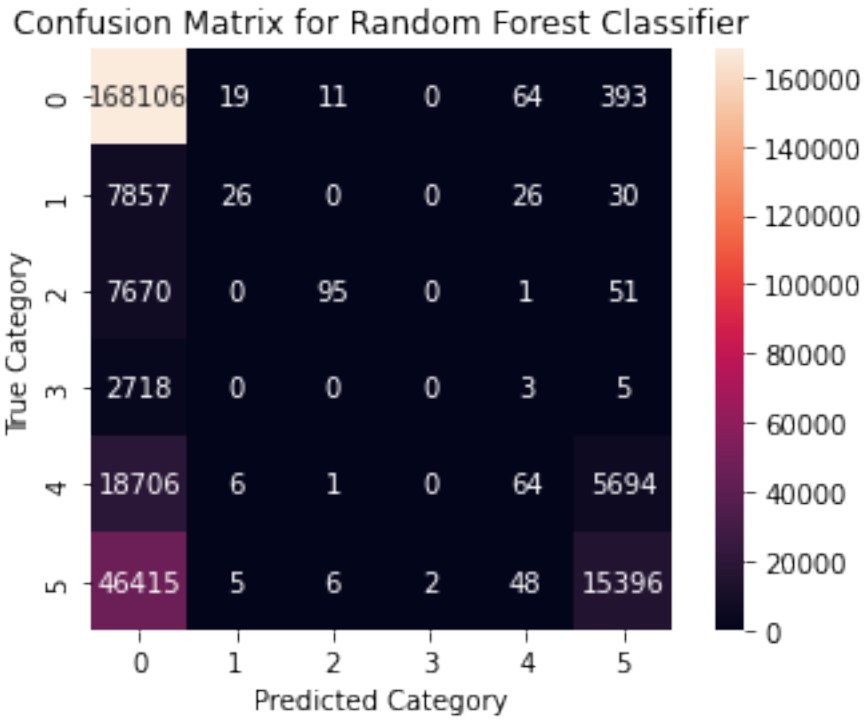
```
Counter({0: 168593, 5: 61872, 4: 24471, 1: 7939, 2: 7817, 3: 2726})
```

- Baseline of this model: the most number of the label / the total number of labels in the testing data
- In our case, the number of label 0 / the total number of labels = 168593 / 273418 = 61.66%
- Accuracy should be at least 61.66%

- Random Forest Classifier using Lemmatization

[ ]:
```
hashtag_model = cuRFC(n_estimators=200,
                      split_criterion = 'gini',
                      n_bins=1024,
                      max_depth=16,
                      max_features=1.0)

hashtag_model.fit(X_train_lsi_hashtag, y_train)

pred_test = hashtag_model.predict(X_test_lsi_hashtag)
```

```
#confusion matrix
plt_confusion_matrix(y_test, pred_test, 'Random Forest Classifier')

#accuracy, recall, precision, F-1 Score
arpf(y_test, pred_test)
```

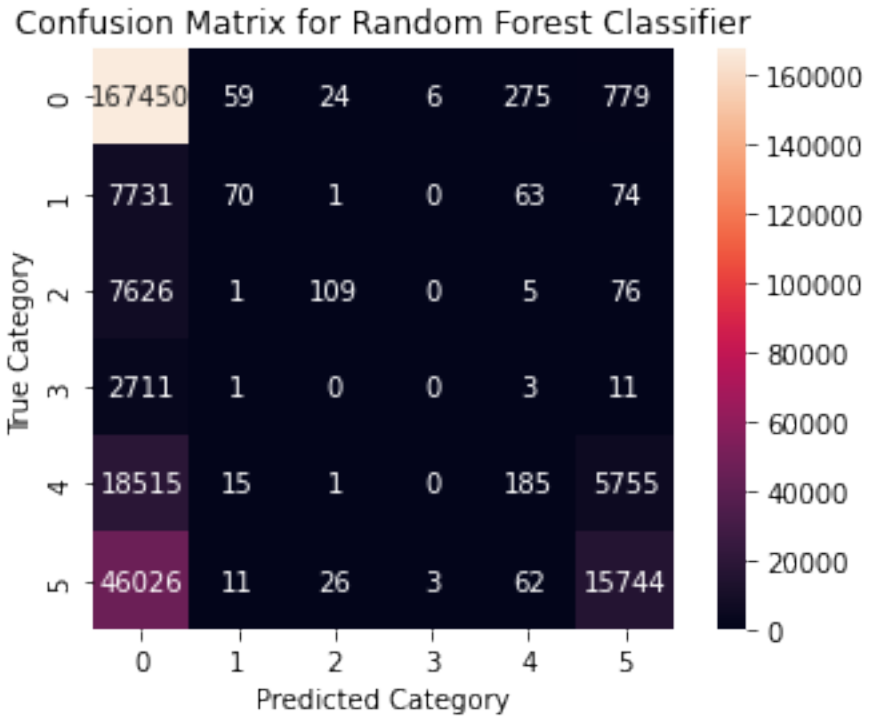



```
Accuracy of the testing data: 0.6718175101858692
Recall of the testing data: 0.6718175101858692
Precision of the testing data: 0.6718175101858692
F-1 Score of the testing data: 0.6718175101858692
```

- Random Forest Classifier using Stemming

```
[ ]: hashtag_model = cuRFC(n_estimators=200,
                       split_criterion = 'gini',
                       n_bins=1024,
                       max_depth=16,
                       max_features=1.0)

hashtag_model.fit(X_train_lsi_hashtag, y_train)

pred_test = hashtag_model.predict(X_test_lsi_hashtag)

#confusion matrix
```

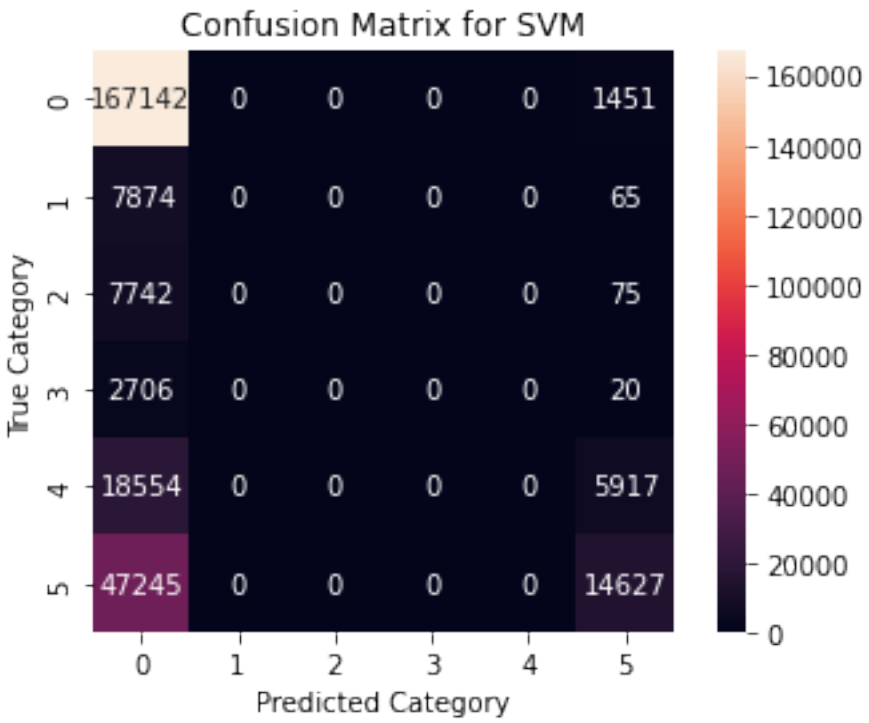
```
plt_confusion_matrix(y_test, pred_test, 'Random Forest Classifier')

#accuracy, recall, precision, F-1 Score
arpf(y_test, pred_test)
```




```
Accuracy of the testing data: 0.6713457051108559
Recall of the testing data: 0.6713457051108559
Precision of the testing data: 0.6713457051108559
F-1 Score of the testing data: 0.6713457051108559
```

- SVM Classifier using Lemmatization

[ ]:
```
clf = SVC(kernel='linear', C=0.1, max_iter=10000)
clf.fit(X_train_lsi_hashtag, y_train)

pred_test = clf.predict(X_test_lsi_hashtag)

#confusion matrix
plt_confusion_matrix(y_test, pred_test, 'SVM')

#accuracy, recall, precision, F-1 Score
arpf(y_test, pred_test)
```




```
Accuracy of the testing data: 0.6648026099232677
Recall of the testing data: 0.6648026099232677
Precision of the testing data: 0.6648026099232677
F-1 Score of the testing data: 0.6648026099232677
```

Conclusion: - Lemmatization vs. Stemming: We can see that there is no much difference on the performance. In the next parts, we will all use lemmatization. - Random Forest Classifier vs. SVM Classifier: Both are higher than our baseline. And, RFC is doing a little bit better.

6 Predict the number of Retweets

- In the next regression prediction parts, since the units of RMSE and MAE in each prediction are different. We will focus on r-squared score to be our baseline. The data we want to predict varies and some features may not have much correlation. So, the general baseline we decide: r-squared score >= 0.25
- Adding normalized features of likes_count, quotes_count, time_count, and two kinds of followers_count

[ ]:
```
def rmse_mae(y_test, pred_test):
    print("RMSE of the testing data: ", math.sqrt(mean_squared_error(y_test,␣
 ↪pred_test)))
    print("MAE of the testing data: ", mean_absolute_error(y_test, pred_test))
    print("r-squared score of the testing data: ", r2_score(y_test, pred_test))
```

```
[ ]: # Normalize the data which will be used as features in the latter parts of␣
      ↪prediction
y_retweet_train_norm = y_retweet_train / np.linalg.norm(y_retweet_train)
y_retweet_test_norm = y_retweet_test / np.linalg.norm(y_retweet_test)

y_like_train_norm = y_like_train / np.linalg.norm(y_like_train)
y_like_test_norm = y_like_test / np.linalg.norm(y_like_test)

y_quote_train_norm = y_quote_train / np.linalg.norm(y_quote_train)
y_quote_test_norm = y_quote_test / np.linalg.norm(y_quote_test)

y_time_train_norm = y_time_train / np.linalg.norm(y_time_train)
y_time_test_norm = y_time_test / np.linalg.norm(y_time_test)

follower_1_train_norm = follower_1_train / np.linalg.norm(follower_1_train)
follower_1_test_norm = follower_1_test / np.linalg.norm(follower_1_test)

follower_2_train_norm = follower_2_train / np.linalg.norm(follower_2_train)
follower_2_test_norm = follower_2_test / np.linalg.norm(follower_2_test)
```

```
[ ]: # Adding the useful features
X_train_lsi_retweet = X_train_lsi
X_test_lsi_retweet = X_test_lsi

X_train_lsi_retweet = np.concatenate((X_train_lsi_retweet, y_like_train_norm.
  ↪reshape(y_like_train_norm.shape[0], 1)), axis=1)
X_test_lsi_retweet = np.concatenate((X_test_lsi_retweet, y_like_test_norm.
  ↪reshape(y_like_test_norm.shape[0], 1)), axis=1)

X_train_lsi_retweet = np.concatenate((X_train_lsi_retweet, y_quote_train_norm.
  ↪reshape(y_quote_train_norm.shape[0], 1)), axis=1)
X_test_lsi_retweet = np.concatenate((X_test_lsi_retweet, y_quote_test_norm.
  ↪reshape(y_quote_test_norm.shape[0], 1)), axis=1)

X_train_lsi_retweet = np.concatenate((X_train_lsi_retweet, y_time_train_norm.
  ↪reshape(y_time_train_norm.shape[0], 1)), axis=1)
X_test_lsi_retweet = np.concatenate((X_test_lsi_retweet, y_time_test_norm.
  ↪reshape(y_time_test_norm.shape[0], 1)), axis=1)

X_train_lsi_retweet = np.concatenate((X_train_lsi_retweet,␣
  ↪follower_1_train_norm.reshape(follower_1_train_norm.shape[0], 1)), axis=1)
X_test_lsi_retweet = np.concatenate((X_test_lsi_retweet, follower_1_test_norm.
  ↪reshape(follower_1_test_norm.shape[0], 1)), axis=1)

X_train_lsi_retweet = np.concatenate((X_train_lsi_retweet,␣
  ↪follower_2_train_norm.reshape(follower_2_train_norm.shape[0], 1)), axis=1)
```

```
X_test_lsi_retweet = np.concatenate((X_test_lsi_retweet, follower_2_test_norm.
  ↪reshape(follower_2_test_norm.shape[0], 1)), axis=1)
```

[ ]:
```
RMSE = []
MAE = []
r2 = []
```

- Linear Regression

[ ]:
```
lr = LinearRegression(fit_intercept = True, normalize = False,
                      algorithm = 'svd')

reg = lr.fit(X_train_lsi_retweet, y_retweet_train)

reg_pred_test = reg.predict(X_test_lsi_retweet)
rmse_mae(y_retweet_test, reg_pred_test)
```

```
RMSE of the testing data:  5.644783864767152
MAE of the testing data:  0.11503527150080936
r-squared score of the testing data:  0.6546839532194542
```

[ ]:
```
RMSE.append(math.sqrt(mean_squared_error(y_retweet_test, reg_pred_test)))
MAE.append(float(mean_absolute_error(y_retweet_test, reg_pred_test)))
r2.append(r2_score(y_retweet_test, reg_pred_test))
```

- Random Forest Regressor

[ ]:
```
retweet_model = cuRFR(n_estimators=200,
                      split_criterion='mse',
                      n_bins=1024,
                      max_features=1.0,
                      accuracy_metric='r2')

retweet_model.fit(X_train_lsi_retweet, y_retweet_train)

retweet_pred_test = retweet_model.predict(X_test_lsi_retweet)
```

[ ]:
```
rmse_mae(y_retweet_test, retweet_pred_test)
#print("r-squared score of the training data: ", retweet_model.
  ↪score(X_train_lsi_retweet, y_retweet_train))
```

```
RMSE of the testing data:  7.602861512660419
MAE of the testing data:  0.10957864989728443
r-squared score of the testing data:  0.3735646103965934
```

[ ]:
```
RMSE.append(math.sqrt(mean_squared_error(y_retweet_test, retweet_pred_test)))
MAE.append(float(mean_absolute_error(y_retweet_test, retweet_pred_test)))
r2.append(r2_score(y_retweet_test, retweet_pred_test))
```

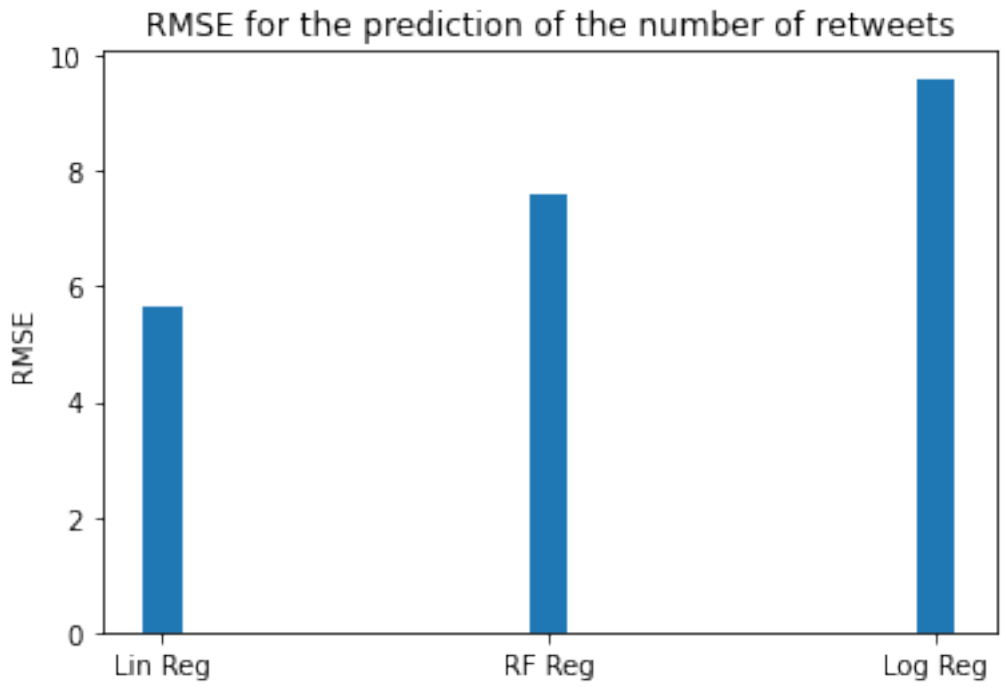
- Logistic Regression

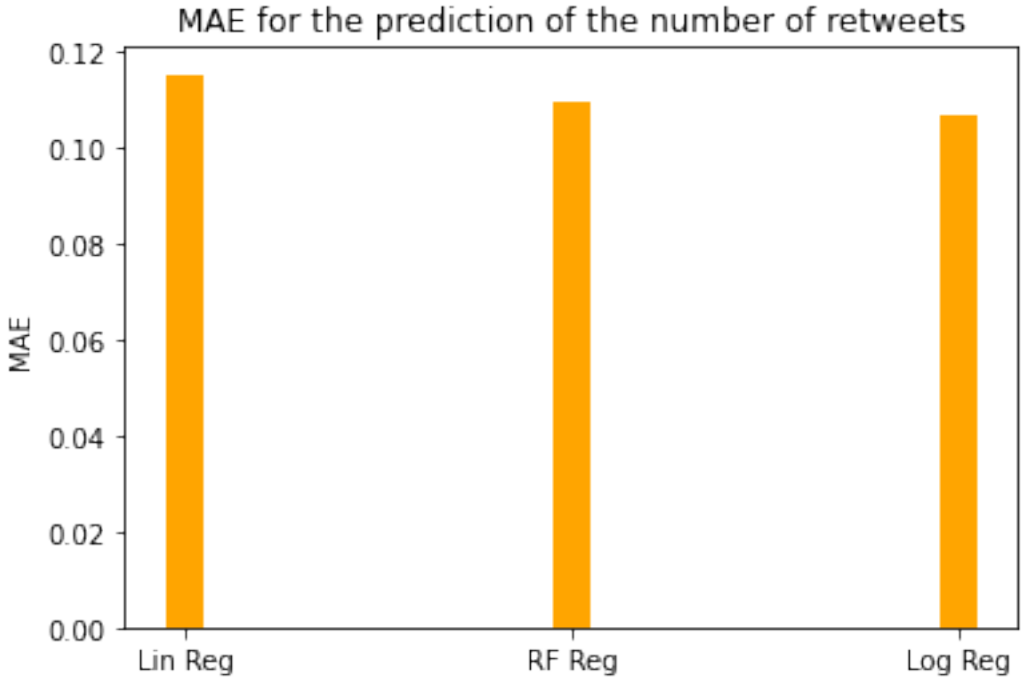
[ ]:
```
lr = LogisticRegression(penalty='l2', C=10, max_iter=10000, tol=0.0001,␣
 ↪solver='qn')

reg2 = lr.fit(X_train_lsi_retweet, y_retweet_train)

reg2_pred_test = reg2.predict(X_test_lsi_retweet)
rmse_mae(y_retweet_test, reg2_pred_test)
```

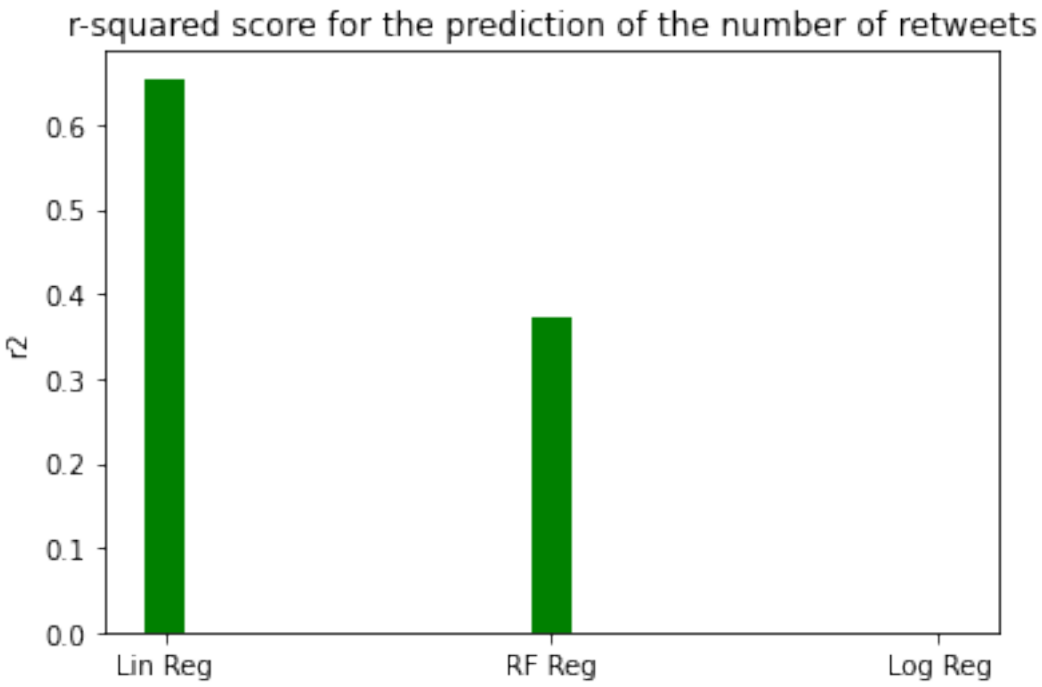
```
RMSE of the testing data:  9.604919875023233
MAE of the testing data:  0.10640484532839828
r-squared score of the testing data:  0.000208091581444525
```

[ ]:
```
RMSE.append(math.sqrt(mean_squared_error(y_retweet_test, reg2_pred_test)))
MAE.append(float(mean_absolute_error(y_retweet_test, reg2_pred_test)))
r2.append(r2_score(y_retweet_test, reg2_pred_test))
```

[ ]:
```
width = 0.1
xticks = ['Lin Reg', 'RF Reg', 'Log Reg']

plt.bar(xticks, RMSE, width)
plt.ylabel('RMSE')
plt.title('RMSE for the prediction of the number of retweets')
plt.show()

plt.bar(xticks, MAE, width, color='orange')
plt.ylabel('MAE')
plt.title('MAE for the prediction of the number of retweets')
plt.show()

plt.bar(xticks, r2, width, color='green')
plt.ylabel('r2')
plt.title('r-squared score for the prediction of the number of retweets')
plt.show()
```


RMSE for the prediction of the number of retweets
RMSE
10
8
6
4
2
0
Lin Reg
RF Reg
Log Reg


MAE for the prediction of the number of retweets
MAE
0.12
0.10
0.08
0.06
0.04
0.02
0.00
Lin Reg
RF Reg
Log Reg



Conclusion: - Linear vs. Random Forest vs. Logistic: We can see that Linear Regression does the best in this task. The r-squared score of Linear Regression is good. In the next parts, we will only consider use Linear Regression and Random Forest Regression. - The unit of RMSE is the number of retweets. We think these numbers we get are reasonable since for RMSE, the number has been squared and it will make the error increase significantly.

7 Predict the number of Likes

- Adding normalized features of retweetd_count, quotes_count, time_count, and two kinds of followers_count

[ ]:
```
# Adding the useful features
X_train_lsi_like = X_train_lsi
X_test_lsi_like = X_test_lsi

X_train_lsi_like = np.concatenate((X_train_lsi_like, y_retweet_train_norm.
 ↪reshape(y_retweet_train_norm.shape[0], 1)), axis=1)
X_test_lsi_like = np.concatenate((X_test_lsi_like, y_retweet_test_norm.
 ↪reshape(y_retweet_test_norm.shape[0], 1)), axis=1)

X_train_lsi_like = np.concatenate((X_train_lsi_like, y_quote_train_norm.
 ↪reshape(y_quote_train_norm.shape[0], 1)), axis=1)
X_test_lsi_like = np.concatenate((X_test_lsi_like, y_quote_test_norm.
 ↪reshape(y_quote_test_norm.shape[0], 1)), axis=1)
```

```
X_train_lsi_like = np.concatenate((X_train_lsi_like, y_time_train_norm.
 ↪reshape(y_time_train_norm.shape[0], 1)), axis=1)
X_test_lsi_like = np.concatenate((X_test_lsi_like, y_time_test_norm.
 ↪reshape(y_time_test_norm.shape[0], 1)), axis=1)

X_train_lsi_like = np.concatenate((X_train_lsi_like, follower_1_train_norm.
 ↪reshape(follower_1_train_norm.shape[0], 1)), axis=1)
X_test_lsi_like = np.concatenate((X_test_lsi_like, follower_1_test_norm.
 ↪reshape(follower_1_test_norm.shape[0], 1)), axis=1)

X_train_lsi_like = np.concatenate((X_train_lsi_like, follower_2_train_norm.
 ↪reshape(follower_2_train_norm.shape[0], 1)), axis=1)
X_test_lsi_like = np.concatenate((X_test_lsi_like, follower_2_test_norm.
 ↪reshape(follower_2_test_norm.shape[0], 1)), axis=1)
```

[ ]:
```
RMSE = []
MAE = []
r2 = []
```

- Linear Regression

[ ]:
```
lr = LinearRegression(fit_intercept = True, normalize = False,
                      algorithm = 'svd')

reg = lr.fit(X_train_lsi_like, y_like_train)

reg_pred_test = reg.predict(X_test_lsi_like)
rmse_mae(y_like_test, reg_pred_test)
```

```
RMSE of the testing data:  8.26305575002017
MAE of the testing data:  0.17662692014483053
r-squared score of the testing data:  0.4077727536915532
```

[ ]:
```
RMSE.append(math.sqrt(mean_squared_error(y_like_test, reg_pred_test)))
MAE.append(float(mean_absolute_error(y_like_test, reg_pred_test)))
r2.append(r2_score(y_like_test, reg_pred_test))
```

- Random Forest Regressor

[ ]:
```
like_model = cuRFR(n_estimators=200,
                   split_criterion='mse',
                   n_bins=1024,
                   max_features=1.0,
                   accuracy_metric='r2')

like_model.fit(X_train_lsi_like, y_like_train)
```

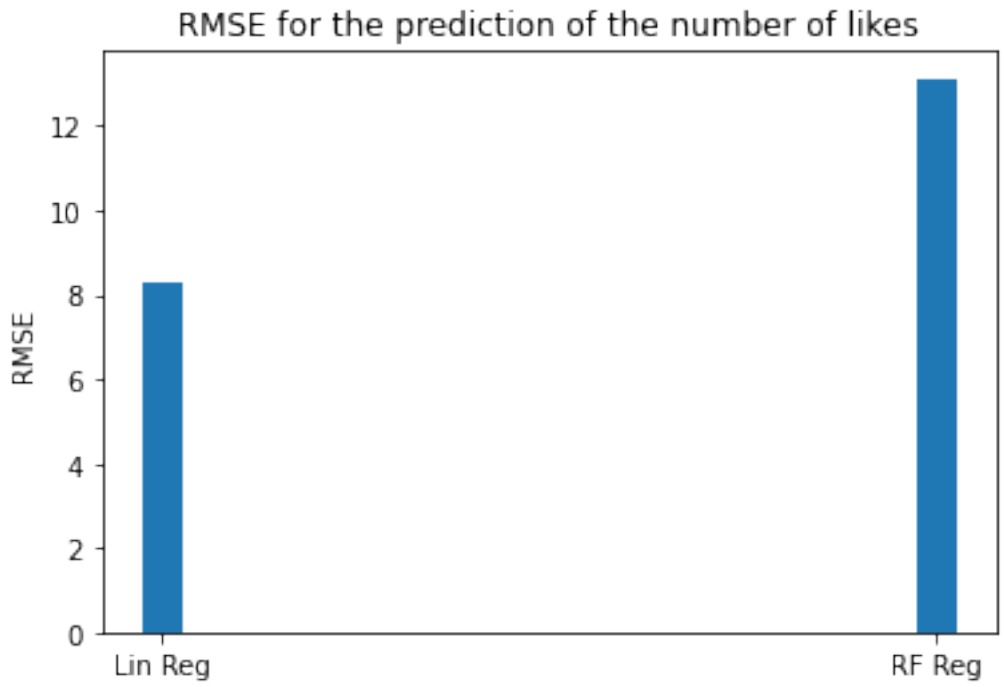
```
like_pred_test = like_model.predict(X_test_lsi_like)
```

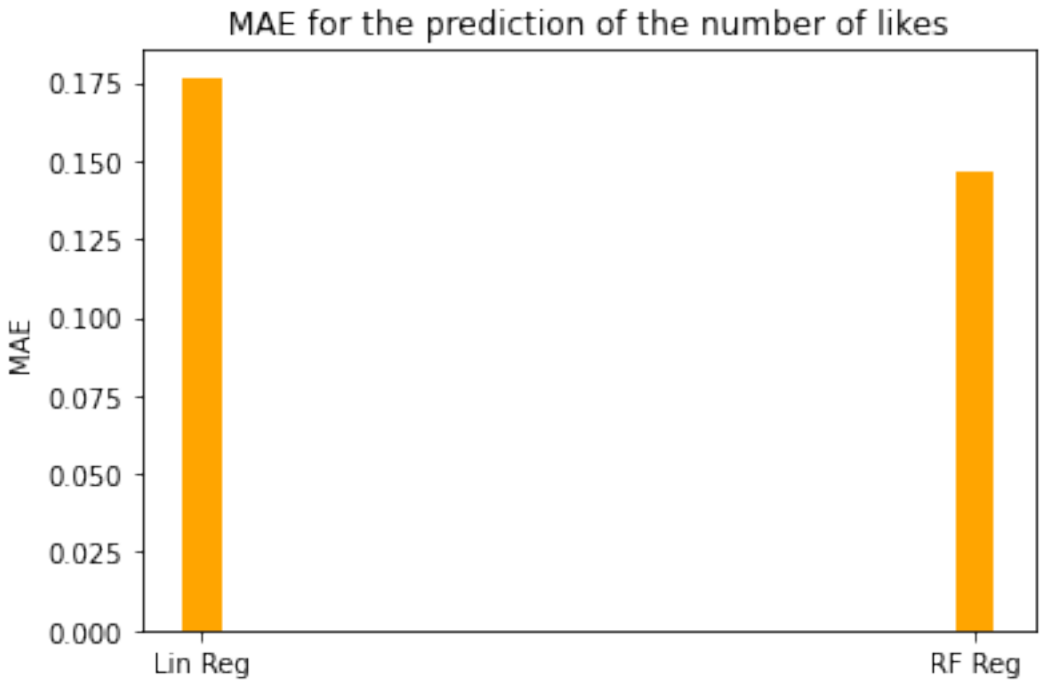
[ ]:
```
rmse_mae(y_like_test, like_pred_test)
# print("r-squared score of the training data: ", like_model.
  ↪score(X_train_lsi_like, y_like_train))
```

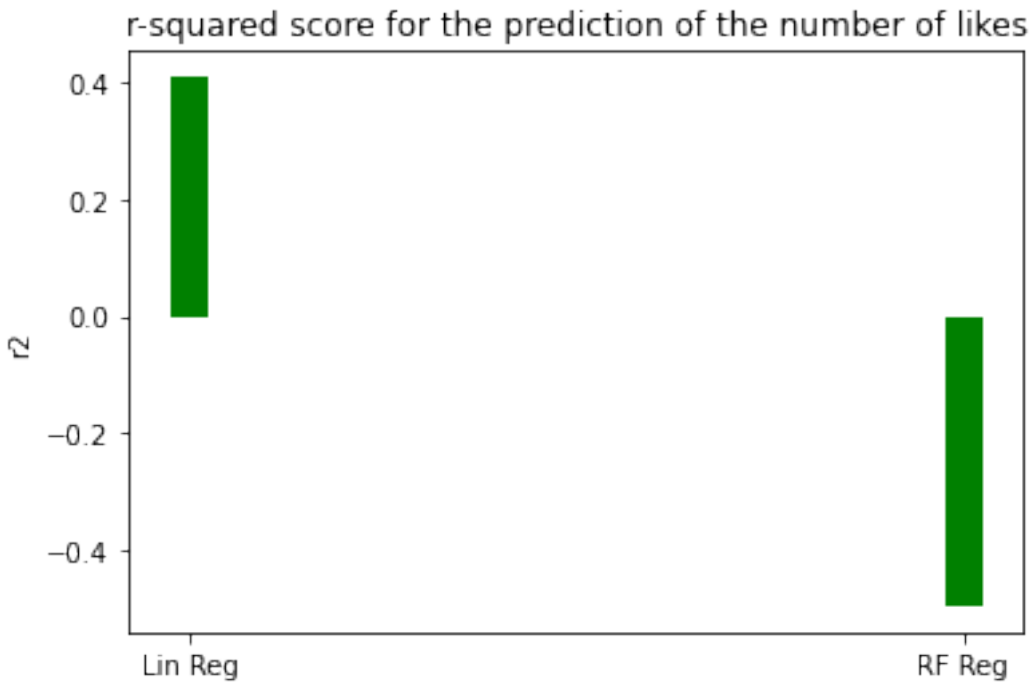
```
RMSE of the testing data:  13.123886703124283
MAE of the testing data:  0.14682242981601903
r-squared score of the testing data:  -0.493935897986453
```

[ ]:
```
RMSE.append(math.sqrt(mean_squared_error(y_like_test, like_pred_test)))
MAE.append(float(mean_absolute_error(y_like_test, like_pred_test)))
r2.append(r2_score(y_like_test, like_pred_test))
```

[ ]:
```
width = 0.05
xticks = ['Lin Reg', 'RF Reg']

plt.bar(xticks, RMSE, width)
plt.ylabel('RMSE')
plt.title('RMSE for the prediction of the number of likes')
plt.show()

plt.bar(xticks, MAE, width, color='orange')
plt.ylabel('MAE')
plt.title('MAE for the prediction of the number of likes')
plt.show()

plt.bar(xticks, r2, width, color='green')
plt.ylabel('r2')
plt.title('r-squared score for the prediction of the number of likes')
plt.show()
```


RMSE for the prediction of the number of likes
RMSE
0
2
4
6
8
10
12
Lin Reg
RF Reg


MAE for the prediction of the number of likes
MAE
0.000
0.025
0.050
0.075
0.100
0.125
0.150
0.175
Lin Reg
RF Reg



Conclusion: - Linear vs. Random Forest: We can see that Linear Regression does better in this task. The r-squared score of Linear Regression is relatively good. - The unit of RMSE is the number of likes. We think these numbers we get are reasonable since for RMSE, the number has been squared and it will make the error increase significantly.

8 Predict the number of Quotes

- Adding normalized features of retweets_count, likes_count, time_count, and two kinds of followers_count

[ ]:
```
# Adding the useful features
X_train_lsi_quote = X_train_lsi
X_test_lsi_quote = X_test_lsi

X_train_lsi_quote = np.concatenate((X_train_lsi_quote, y_retweet_train_norm.
 ↪reshape(y_retweet_train_norm.shape[0], 1)), axis=1)
X_test_lsi_quote = np.concatenate((X_test_lsi_quote, y_retweet_test_norm.
 ↪reshape(y_retweet_test_norm.shape[0], 1)), axis=1)

X_train_lsi_quote = np.concatenate((X_train_lsi_quote, y_like_train_norm.
 ↪reshape(y_like_train_norm.shape[0], 1)), axis=1)
X_test_lsi_quote = np.concatenate((X_test_lsi_quote, y_like_test_norm.
 ↪reshape(y_like_test_norm.shape[0], 1)), axis=1)
```

```
X_train_lsi_quote = np.concatenate((X_train_lsi_quote, y_time_train_norm.
 ↪reshape(y_time_train_norm.shape[0], 1)), axis=1)
X_test_lsi_quote = np.concatenate((X_test_lsi_quote, y_time_test_norm.
 ↪reshape(y_time_test_norm.shape[0], 1)), axis=1)

X_train_lsi_quote = np.concatenate((X_train_lsi_quote, follower_1_train_norm.
 ↪reshape(follower_1_train_norm.shape[0], 1)), axis=1)
X_test_lsi_quote = np.concatenate((X_test_lsi_quote, follower_1_test_norm.
 ↪reshape(follower_1_test_norm.shape[0], 1)), axis=1)

X_train_lsi_quote = np.concatenate((X_train_lsi_quote, follower_2_train_norm.
 ↪reshape(follower_2_train_norm.shape[0], 1)), axis=1)
X_test_lsi_quote = np.concatenate((X_test_lsi_quote, follower_2_test_norm.
 ↪reshape(follower_2_test_norm.shape[0], 1)), axis=1)
```

[ ]:
```
RMSE = []
MAE = []
r2 = []
```

- Linear Regression

[ ]:
```
lr = LinearRegression(fit_intercept = True, normalize = False,
                      algorithm = 'svd')

reg = lr.fit(X_train_lsi_quote, y_quote_train)

reg_pred_test = reg.predict(X_test_lsi_quote)
rmse_mae(y_quote_test, reg_pred_test)
```

```
RMSE of the testing data:  27.32441702985442
MAE of the testing data:  1.5213741040916045
r-squared score of the testing data:  0.589646935819709
```

[ ]:
```
RMSE.append(math.sqrt(mean_squared_error(y_quote_test, reg_pred_test)))
MAE.append(float(mean_absolute_error(y_quote_test, reg_pred_test)))
r2.append(r2_score(y_quote_test, reg_pred_test))
```

- Random Forest Regressor

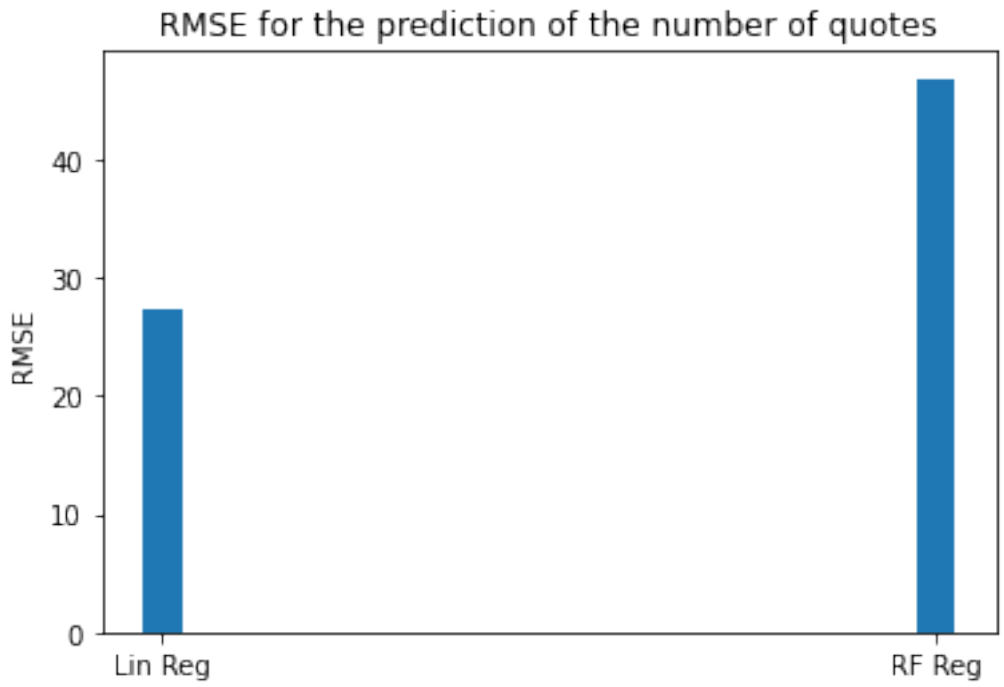
[ ]:
```
quote_model = cuRFR(n_estimators=200,
                    split_criterion='mse',
                    n_bins=1024,
                    max_features=1.0,
                    accuracy_metric='r2')

quote_model.fit(X_train_lsi_quote, y_quote_train)

quote_pred_test = quote_model.predict(X_test_lsi_quote)
```

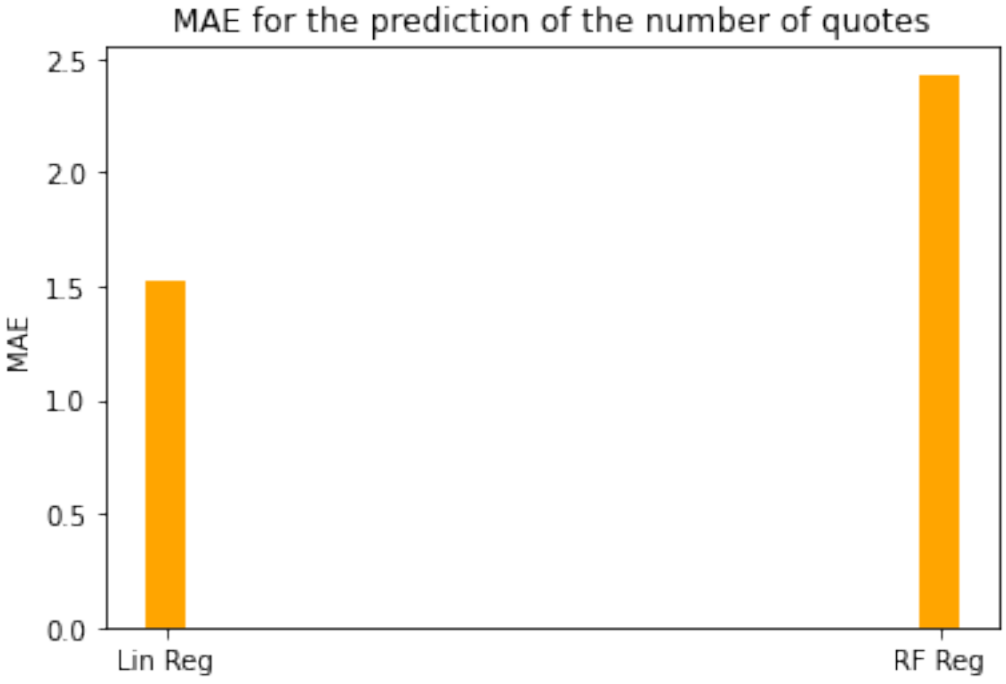
[ ]:
```
rmse_mae(y_quote_test, quote_pred_test)
# print("r-squared score of the training data: ", quote_model.
  ↪score(X_train_lsi_quote, y_quote_train))
```

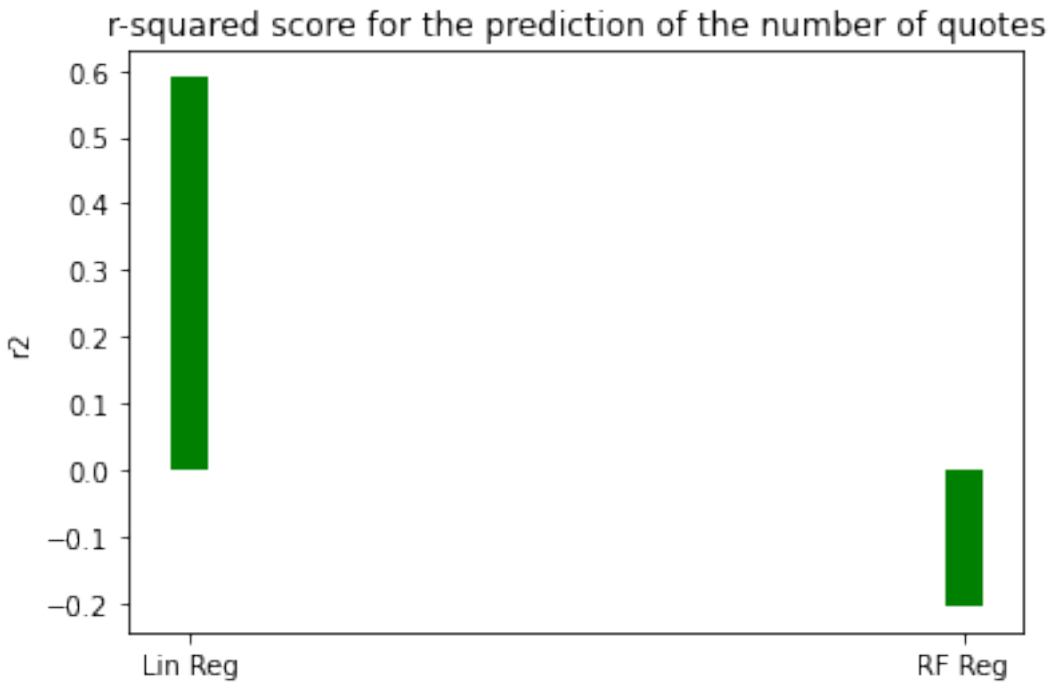
```
RMSE of the testing data:  46.80687848253037
MAE of the testing data:  2.4308691499758477
r-squared score of the testing data:  -0.20413513683569362
```

[ ]:
```
RMSE.append(math.sqrt(mean_squared_error(y_quote_test, quote_pred_test)))
MAE.append(float(mean_absolute_error(y_quote_test, quote_pred_test)))
r2.append(r2_score(y_quote_test, quote_pred_test))
```

[ ]:
```
width = 0.05
xticks = ['Lin Reg', 'RF Reg']

plt.bar(xticks, RMSE, width)
plt.ylabel('RMSE')
plt.title('RMSE for the prediction of the number of quotes')
plt.show()

plt.bar(xticks, MAE, width, color='orange')
plt.ylabel('MAE')
plt.title('MAE for the prediction of the number of quotes')
plt.show()

plt.bar(xticks, r2, width, color='green')
plt.ylabel('r2')
plt.title('r-squared score for the prediction of the number of quotes')
plt.show()
```


RMSE for the prediction of the number of quotes
RMSE
0
10
20
30
40
Lin Reg
RF Reg


MAE for the prediction of the number of quotes
MAE
0.0
0.5
1.0
1.5
2.0
2.5
Lin Reg
RF Reg



Conclusion: - Linear vs. Random Forest: We can see that Linear Regression does better in this task. The r-squared score of Linear Regression is good. - The unit of RMSE is the number of quotes. We think these numbers we get are reasonable since for RMSE, the number has been squared and it will make the error increase significantly.

9 Predict the relative(UNIX) time when a tweet was posted

- Adding normalized features of retweets_count, likes_count, quotes_count, and two kinds of followers_count

[ ]:
```
# Adding the useful features
X_train_lsi_time = X_train_lsi
X_test_lsi_time = X_test_lsi

X_train_lsi_time = np.concatenate((X_train_lsi_time, y_retweet_train_norm.
 ↪reshape(y_retweet_train_norm.shape[0], 1)), axis=1)
X_test_lsi_time = np.concatenate((X_test_lsi_time, y_retweet_test_norm.
 ↪reshape(y_retweet_test_norm.shape[0], 1)), axis=1)

X_train_lsi_time = np.concatenate((X_train_lsi_time, y_like_train_norm.
 ↪reshape(y_like_train_norm.shape[0], 1)), axis=1)
X_test_lsi_time = np.concatenate((X_test_lsi_time, y_like_test_norm.
 ↪reshape(y_like_test_norm.shape[0], 1)), axis=1)
```

```
X_train_lsi_time = np.concatenate((X_train_lsi_time, y_quote_train_norm.
 ↪reshape(y_quote_train_norm.shape[0], 1)), axis=1)
X_test_lsi_time = np.concatenate((X_test_lsi_time, y_quote_test_norm.
 ↪reshape(y_quote_test_norm.shape[0], 1)), axis=1)

X_train_lsi_time = np.concatenate((X_train_lsi_time, follower_1_train_norm.
 ↪reshape(follower_1_train_norm.shape[0], 1)), axis=1)
X_test_lsi_time = np.concatenate((X_test_lsi_time, follower_1_test_norm.
 ↪reshape(follower_1_test_norm.shape[0], 1)), axis=1)

X_train_lsi_time = np.concatenate((X_train_lsi_time, follower_2_train_norm.
 ↪reshape(follower_2_train_norm.shape[0], 1)), axis=1)
X_test_lsi_time = np.concatenate((X_test_lsi_time, follower_2_test_norm.
 ↪reshape(follower_2_test_norm.shape[0], 1)), axis=1)
```

[ ]:
```
RMSE = []
MAE = []
r2 = []
```

- Linear Regression

[ ]:
```
lr = LinearRegression(fit_intercept = True, normalize = False,
                      algorithm = 'svd')

reg = lr.fit(X_train_lsi_time, y_time_train)

reg_pred_test = reg.predict(X_test_lsi_time)
rmse_mae(y_time_test, reg_pred_test)
```

```
RMSE of the testing data:  4355.861641044072
MAE of the testing data:  3585.0382982385895
r-squared score of the testing data:  0.13610066565181378
```

[ ]:
```
RMSE.append(math.sqrt(mean_squared_error(y_time_test, reg_pred_test)))
MAE.append(float(mean_absolute_error(y_time_test, reg_pred_test)))
r2.append(r2_score(y_time_test, reg_pred_test))
```

- Random Forest Regressor

[ ]:
```
time_model = cuRFR(n_estimators=200,
                   split_criterion='mse',
                   n_bins=1024,
                   max_features=1.0,
                   max_depth=256,
                   accuracy_metric='r2')

time_model.fit(X_train_lsi_time, y_time_train)
```

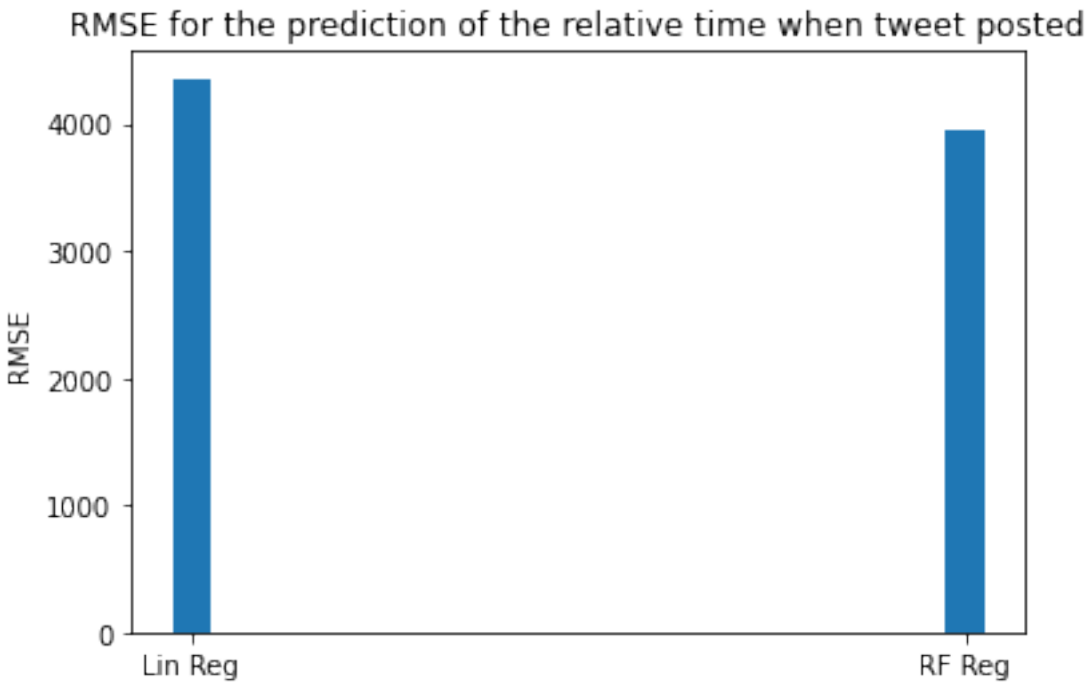
```
time_pred_test = time_model.predict(X_test_lsi_time)
```

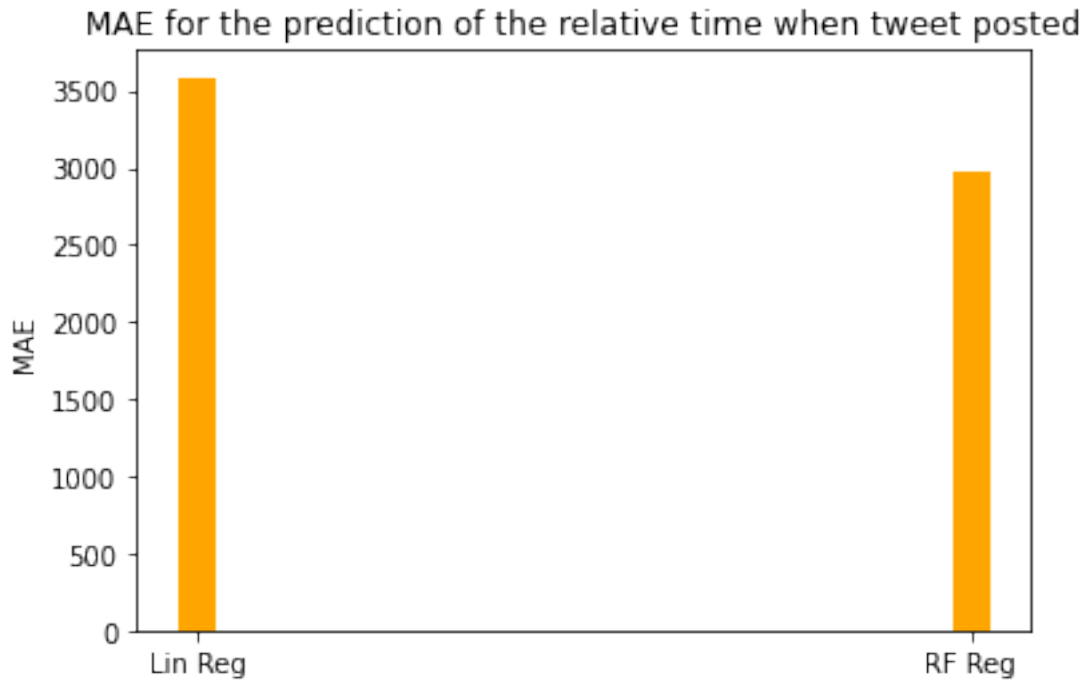
[ ]:
```
rmse_mae(y_time_test, time_pred_test)
print("r-squared score of the training data: ", time_model.
  ↪score(X_train_lsi_time, y_time_train))
```

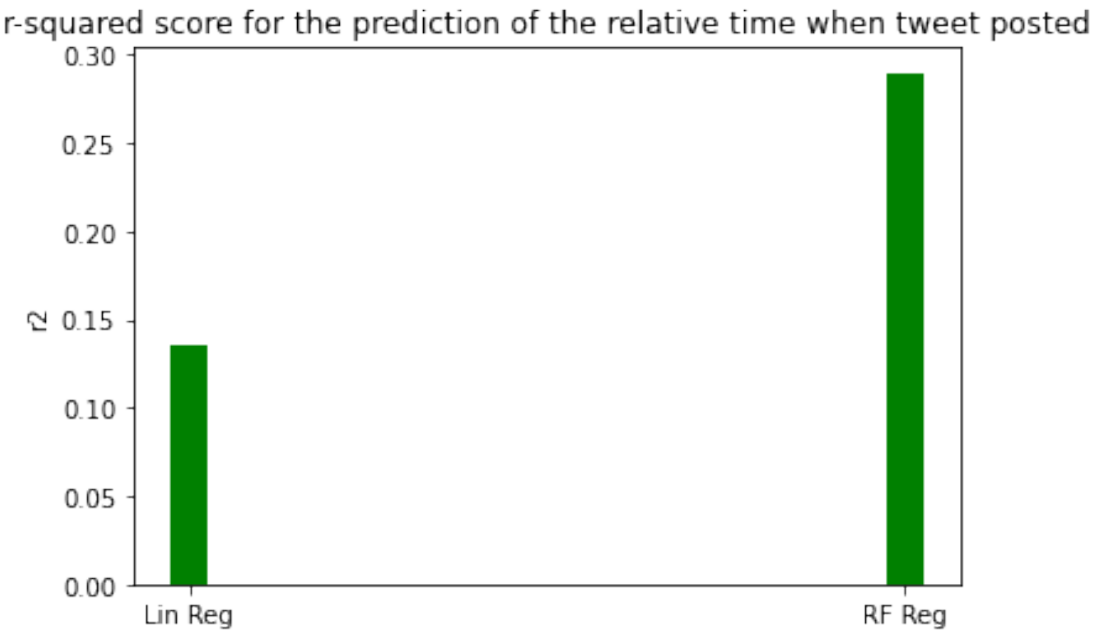
```
RMSE of the testing data:  3949.7112764462186
MAE of the testing data:  2972.6146395761853
r-squared score of the testing data:  0.2896936678305433
r-squared score of the training data:  0.7881966093588663
```

[ ]:
```
RMSE.append(math.sqrt(mean_squared_error(y_time_test, time_pred_test)))
MAE.append(float(mean_absolute_error(y_time_test, time_pred_test)))
r2.append(r2_score(y_time_test, time_pred_test))
```

[ ]:
```
width = 0.05
xticks = ['Lin Reg', 'RF Reg']

plt.bar(xticks, RMSE, width)
plt.ylabel('RMSE')
plt.title('RMSE for the prediction of the relative time when tweet posted')
plt.show()

plt.bar(xticks, MAE, width, color='orange')
plt.ylabel('MAE')
plt.title('MAE for the prediction of the relative time when tweet posted')
plt.show()

plt.bar(xticks, r2, width, color='green')
plt.ylabel('r2')
plt.title('r-squared score for the prediction of the relative time when tweet␣
  ↪posted')
plt.show()
```


RMSE for the prediction of the relative time when tweet posted
RMSE
4000
3000
2000
1000
0
Lin Reg
RF Reg


MAE for the prediction of the relative time when tweet posted
MAE
3500
3000
2500
2000
1500
1000
500
0
Lin Reg
RF Reg



Conclusion: - Linear vs. Random Forest: We can see that Random Forest Regression does better in this task. - The units of RMSE and MAE are seconds. We think these numbers we get are reasonable since the timing distributes broadly and we use five-hour data. - Compared to MAE, the number of RMSE has been squared and it will make the error increase significantly, especially the unit is second in this task.

10 Creativity - Predict the number of followers of the person tweeting

- Adding normalized features of retweets_count, likes_count, quotes_count, and time_count

[ ]:
```
# Adding the useful features
X_train_lsi_follower = X_train_lsi
X_test_lsi_follower = X_test_lsi

X_train_lsi_follower = np.concatenate((X_train_lsi_follower,␣
 ↪y_retweet_train_norm.reshape(y_retweet_train_norm.shape[0], 1)), axis=1)
X_test_lsi_follower = np.concatenate((X_test_lsi_follower, y_retweet_test_norm.
 ↪reshape(y_retweet_test_norm.shape[0], 1)), axis=1)

X_train_lsi_follower = np.concatenate((X_train_lsi_follower, y_like_train_norm.
 ↪reshape(y_like_train_norm.shape[0], 1)), axis=1)
X_test_lsi_follower = np.concatenate((X_test_lsi_follower, y_like_test_norm.
 ↪reshape(y_like_test_norm.shape[0], 1)), axis=1)
```

```
X_train_lsi_follower = np.concatenate((X_train_lsi_follower, y_quote_train_norm.
 ↪reshape(y_quote_train_norm.shape[0], 1)), axis=1)
X_test_lsi_follower = np.concatenate((X_test_lsi_follower, y_quote_test_norm.
 ↪reshape(y_quote_test_norm.shape[0], 1)), axis=1)

X_train_lsi_follower = np.concatenate((X_train_lsi_follower, y_time_train_norm.
 ↪reshape(y_time_train_norm.shape[0], 1)), axis=1)
X_test_lsi_follower = np.concatenate((X_test_lsi_follower, y_time_test_norm.
 ↪reshape(y_time_test_norm.shape[0], 1)), axis=1)
```

[ ]:
```
RMSE = []
MAE = []
r2 = []
```

- Linear Regression

[ ]:
```
lr = LinearRegression(fit_intercept = True, normalize = False,
                      algorithm = 'svd')

reg = lr.fit(X_train_lsi_follower, follower_1_train)

reg_pred_test = reg.predict(X_test_lsi_follower)
rmse_mae(follower_1_test, reg_pred_test)
```

```
RMSE of the testing data:  68744.36962092745
MAE of the testing data:  3395.827596530905
r-squared score of the testing data:  0.73726706888122
```

[ ]:
```
RMSE.append(math.sqrt(mean_squared_error(follower_1_test, reg_pred_test)))
MAE.append(float(mean_absolute_error(follower_1_test, reg_pred_test)))
r2.append(r2_score(follower_1_test, reg_pred_test))
```

- Random Forest Regressor

[ ]:
```
follower_model = cuRFR(n_estimators=200,
                   split_criterion='mse',
                   n_bins=1024,
                   max_features=1.0,
                   accuracy_metric='r2')

follower_model.fit(X_train_lsi_follower, follower_1_train)

follower_pred_test = follower_model.predict(X_test_lsi_follower)
```

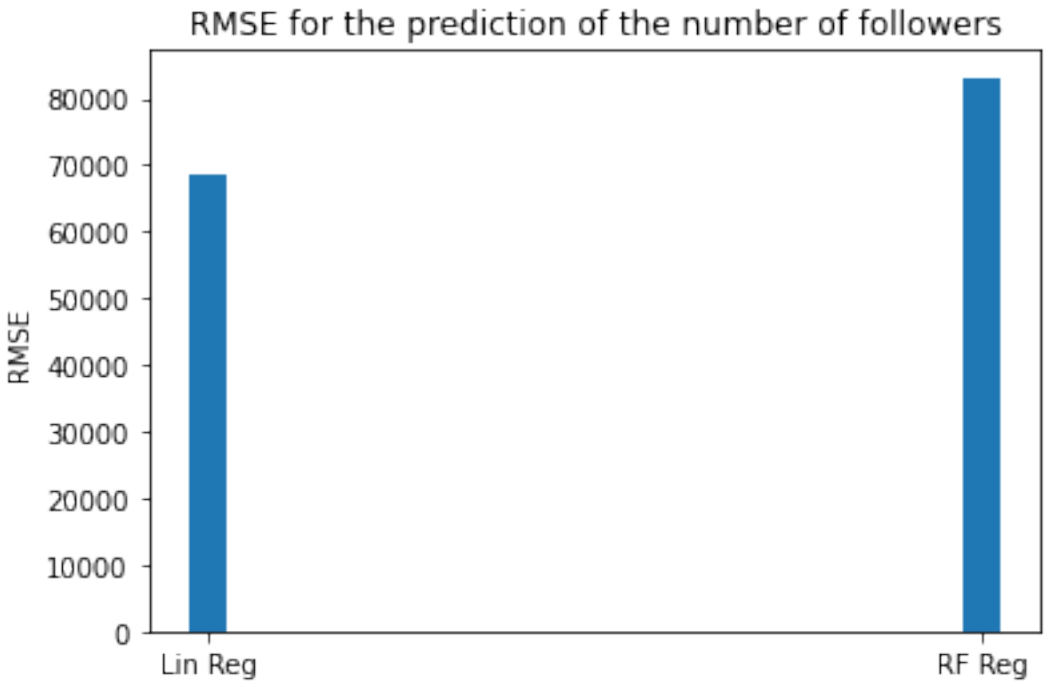
[ ]:
```
rmse_mae(follower_1_test, follower_pred_test)
# print("r-squared score of the training data: ", follower_model.
 ↪score(X_train_lsi_follower, follower_1_train))
```

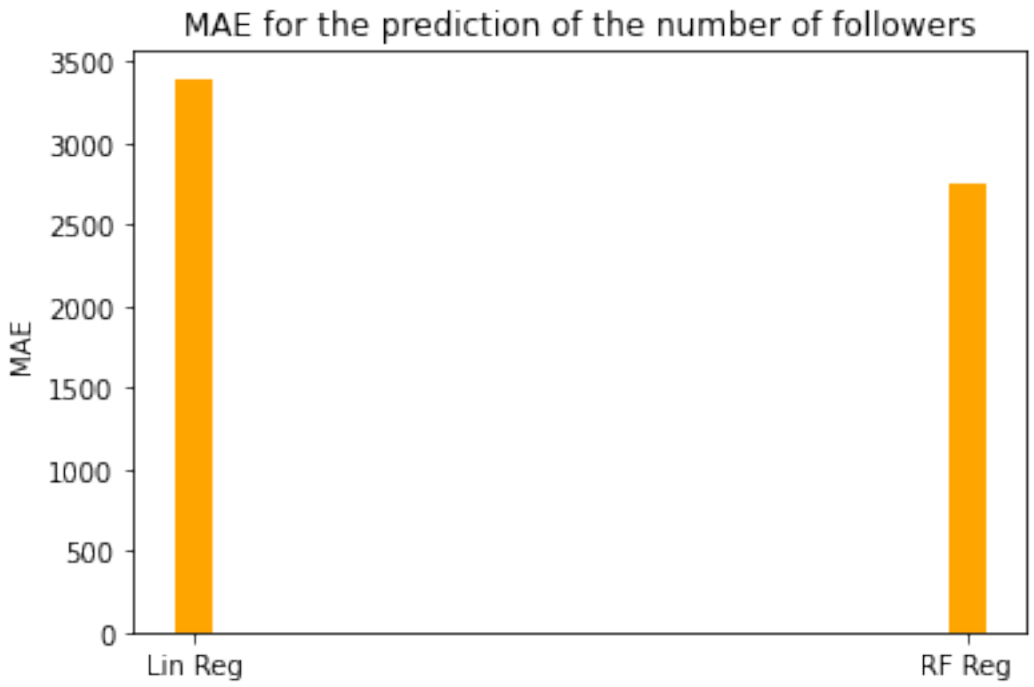
```
RMSE of the testing data:  83164.44330403922
```

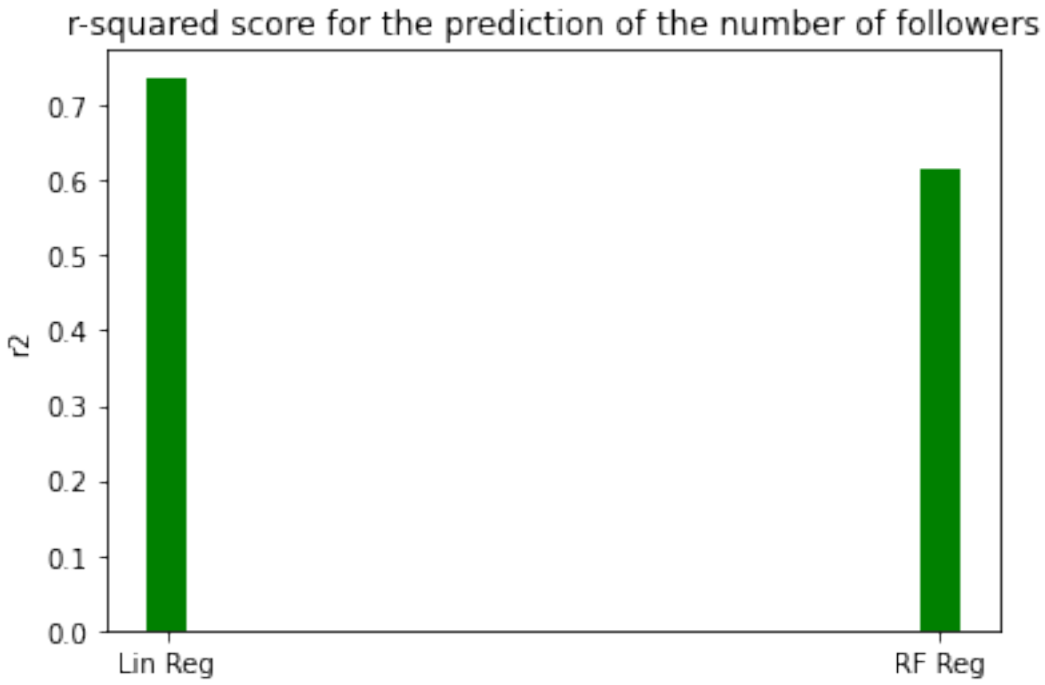
```
MAE of the testing data:  2748.714073029891
r-squared score of the testing data:  0.6154829403507002
```

[ ]:
```
RMSE.append(math.sqrt(mean_squared_error(follower_1_test, follower_pred_test)))
MAE.append(float(mean_absolute_error(follower_1_test, follower_pred_test)))
r2.append(r2_score(follower_1_test, follower_pred_test))
```

[ ]:
```
width = 0.05
xticks = ['Lin Reg', 'RF Reg']

plt.bar(xticks, RMSE, width)
plt.ylabel('RMSE')
plt.title('RMSE for the prediction of the number of followers')
plt.show()

plt.bar(xticks, MAE, width, color='orange')
plt.ylabel('MAE')
plt.title('MAE for the prediction of the number of followers')
plt.show()

plt.bar(xticks, r2, width, color='green')
plt.ylabel('r2')
plt.title('r-squared score for the prediction of the number of followers')
plt.show()
```







Conclusion: - Linear vs. Random Forest: We can see that both Linear Regression and Random Forest Regression do great in this task. Linear Regression is a little bit better since the r-squared

score of Linear Regression is higher. - The units of RMSE and MAE are the number of followers. We think these numbers we get are reasonable since the numbers of followers are usually very high and varies, e.g. 0.1K ~ 100K. - For RMSE, the number has been squared and it will make the error increase more significantly than MAE.