Project 4 : Regression Analysis and Define Your Own Task!

Kuei-Tzu Hu 206300553

Sreya Muppalla 505675909

Christina Lee 406299676

1. Introduction

Regression analysis is a statistical procedure for estimating the relationship between a target variable and a set of features that jointly inform about the target. In this project, we explore specific-to-regression feature engineering methods and model selection that jointly improve the performance of regression. You will conduct different experiments and identify the relative significance of the different options.

```
from google.colab import drive
drive.mount('/content/drive')
%cd /content/drive/My Drive/ECE ENGR 219/
```

```
Mounted at /content/drive
/content/drive/My Drive/ECE ENGR 219
```

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
```

2. Datasets

2.1 Dataset 1: Diamond Characteristics

This dataset contains information about 150000 round-cut diamonds.

- features:
  . carat: weight of the diamond
  . cut: quality of the cut
  . clarity: measured diamond clarity
  . length: measured length in mm
  . width: measured width in mm
  . depth: measured depth in mm
  . depth percent: diamond’s total height divided by it’s total width
  . table percent: width of top of diamond relative to widest point
  . gridle min: refers to the thinnest part of the girdle
  . gridle max: refers to the thickest part of the girdle

- target variable: i.e what we would like to predict:
  . price: price in US dollars

3.1.2 Data Inspection

Question 1.1

Which features have the highest absolute correlation with the target variable. In the context of either dataset, describe what the correlation patterns suggest.

- carat has the highest absolute correlation.
- Correlation patterns suggest the relevance(strength of the linear association) between the given feature and the target variable. Higher absolute correlation indicates higher relevance, which usually coresponds to feature importance. Shown below, the correlation pattern of the diamond dataset suggest the most to least relevant features(except for price, which is 1 because it's the same variable) with price.

price	1.000000
carat	0.913479
length	0.869521
width	0.841887
depth	0.299696
polish_Excellent	0.054928
color_encoded	0.047189
symmetry_Excellent	0.047149
table_percent	0.042453
depth_percent	0.025469
cut_Excellent	0.024356
clarity_encoded	0.018669
maxgirdle_encoded	0.000822
mingirdle_encoded	0.000188

```
dataset1 = pd.read_csv("/content/drive/MyDrive/ECE ENGR 
219/diamonds_ece219.csv")
dataset1 = dataset1.drop(columns=['Unnamed: 0'])

dataset1.head()
```

{"type":"dataframe","variable_name":"dataset1"}

```
# Inspect non-numerical features

print('color : ', np.unique(dataset1["color"]))
print('clarity : ', np.unique(dataset1["clarity"]))
print('cut : ', np.unique(dataset1["cut"]))
print('symmetry : ', np.unique(dataset1["symmetry"]))
print('polish : ', np.unique(dataset1["polish"]))
print('girdle_min : ', np.unique(dataset1["girdle_min"]))
print('girdle_max : ', np.unique(dataset1["girdle_max"]))
```

```
color :  ['D' 'E' 'F' 'G' 'H' 'I' 'J' 'K' 'L' 'M']
clarity :  ['I1' 'I2' 'I3' 'IF' 'SI1' 'SI2' 'VS1' 'VS2' 'VVS1' 'VVS2']
cut :  ['Excellent' 'Very Good']
```

```
symmetry :  ['Excellent' 'Very Good']
polish :  ['Excellent' 'Very Good']
girdle_min :  ['M' 'STK' 'STN' 'TK' 'TN' 'VTK' 'VTN' 'XTK' 'XTN'
'unknown']
girdle_max :  ['M' 'STK' 'STN' 'TK' 'TN' 'VTK' 'VTN' 'XTK' 'XTN'
'unknown']
```

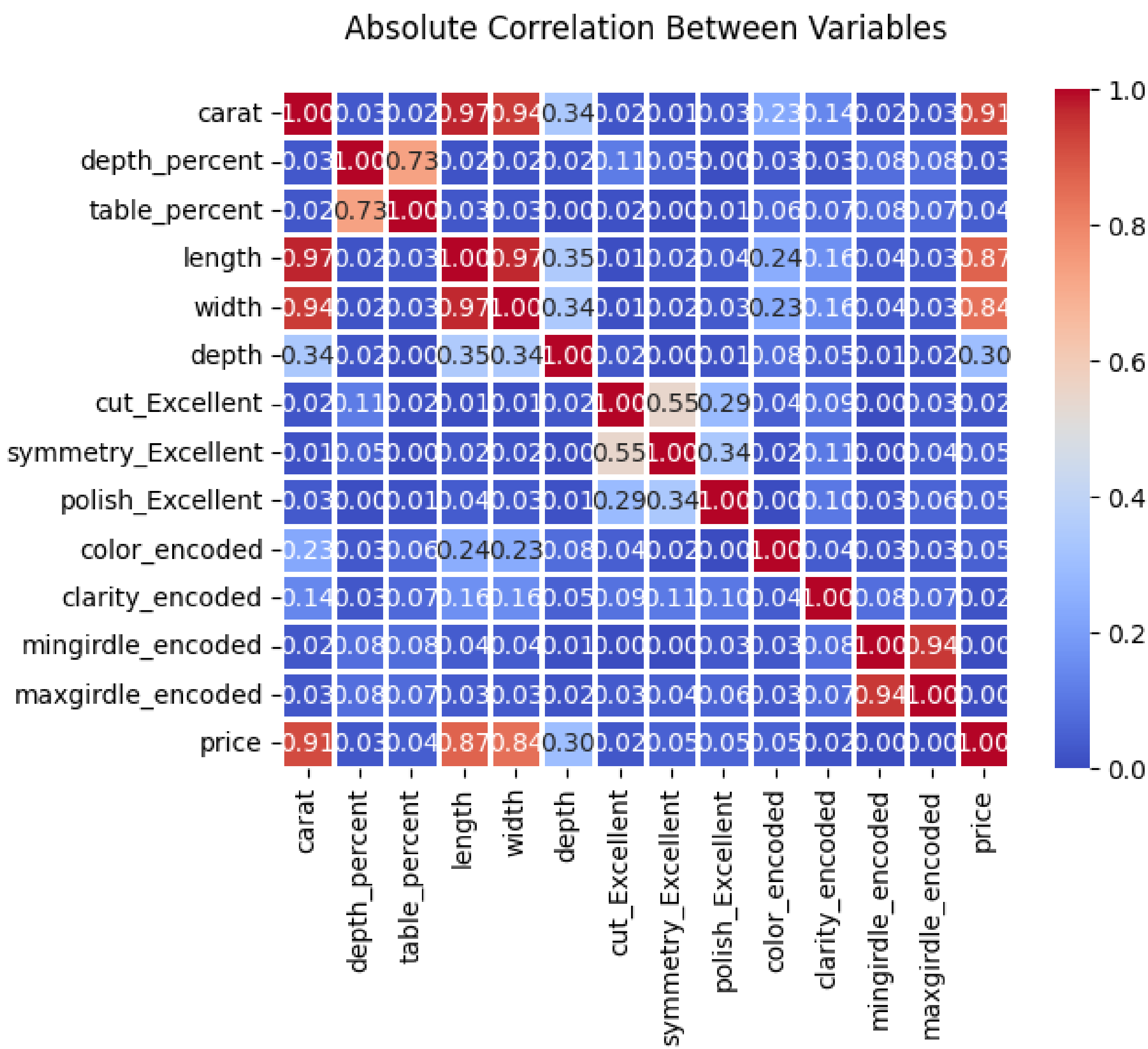
```
dataset2 = dataset1.copy(deep=True)

#Numerical Encodings
# label based on context
https://www.brilliantearth.com/diamond/buying-guide/clarity/?nbt=nb
%3Aadwords%3Ag
%3A13196368544%3A134246532224%3A548694297825&nb_adtype=&nb_kwd=diamond
%20size%20chart&nb_ti=kwd-
1701737760&nb_mi=&nb_pc=&nb_pi=&nb_ppi=&nb_placement=&nb_li_ms=&nb_lp_
ms=&nb_fii=&nb_ap=&nb_mt=b&utm_source=google&utm_medium=cpc&utm_campai
gn=SEM_Search_US_ER_Education&nbt=nb%3Aadwords%3Ag
%3A13196368544%3A134246532224%3A548694297825&nb_adtype=&nb_kwd=diamond
%20size%20chart&nb_ti=kwd-
1701737760&nb_mi=&nb_pc=&nb_pi=&nb_ppi=&nb_placement=&nb_li_ms=&nb_lp_
ms=&nb_fii=&nb_ap=&nb_mt=b&gad_source=1&gclid=CjwKCAiA0PuuBhBsEiwAS7fs
NSH3F3yQJukq_W6s7Py1nw-63ZxzmUhDPVoIYxBu_GLXcBSnnWBaDhoCxGIQAvD_BwE

color_dict = {'M' : 1, # Faint Color Diamond Grades
              'L' : 2,
              'K' : 3,
              'J' : 4, # Near Colorless Diamond Grades
              'I' : 5,
              'H': 6,
              'G': 7,
              'F': 8, # Colorless Diamond Grades
              'E': 9,
              'D': 10
              }


clarity_dict = {'I3' : 1, # included 3
                'l2' : 2, # included 2
                'l1' : 3, # included 1
                'SI2' : 4, # slightly included 2
                'SI1': 5, # slightly included 1
                'VS2': 6, # very slightly included 2
                'VS1': 7, # very slightly included 1
                'VVS2': 8, # very, very slightly included 2
                'VVS1': 9, # very, very slightly included 1
                'IF': 10 # Internally Flawless
                }

girdle_dict = {'unknown' : 0,
```

```
               'XTN' : 1, # extremely thin
               'VTN' : 2, # very thin
               'TN' : 3, # thin
               'STN' : 4, # slightly thin
               'M' : 5, # medium
               'STK' : 6, # slightly thick
               'TK' : 7, # thick
               'VTK' : 8, # very thick
               'XTK' : 9, # extremely thick
               }

dataset2 = pd.get_dummies(dataset2, columns=['cut', 'symmetry',
'polish'])
dataset2['color_encoded'] = dataset2.color.map(color_dict)
dataset2['clarity_encoded'] = dataset2.clarity.map(clarity_dict)
dataset2['mingirdle_encoded'] = dataset2.girdle_min.map(girdle_dict)
dataset2['maxgirdle_encoded'] = dataset2.girdle_max.map(girdle_dict)

diamonds = dataset2.drop(columns=['color','clarity', 'girdle_min',
'girdle_max', 'cut_Very Good', 'symmetry_Very Good', 'polish_Very
Good'])
diamonds.dropna()
# move price to last column
diamonds = diamonds[[col for col in diamonds.columns if col !=
'price'] + ['price']]

diamonds.head()
```

{"type":"dataframe","variable_name":"diamonds"}

```
import seaborn as sns

corr = diamonds.corr().abs()
sns.heatmap(corr, vmin=0, vmax=1, cmap = 'coolwarm', annot=True,
fmt='.2f', linewidths=2)
plt.title("Absolute Correlation Between Variables", pad=20)
```

```
Text(0.5, 1.0, 'Absolute Correlation Between Variables')
```




```
corr_target = diamonds.corrwith(diamonds["price"]).abs()
print(corr_target.sort_values(ascending=False))

price                   1.000000
carat                   0.913479
length                  0.869521
width                   0.841887
depth                   0.299696
polish_Excellent        0.054928
color_encoded           0.047189
symmetry_Excellent      0.047149
table_percent           0.042453
depth_percent           0.025469
cut_Excellent           0.024356
clarity_encoded         0.018669
maxgirdle_encoded       0.000822
```

```
mingirdle_encoded     0.000188
dtype: float64
```

Question 1.2

Plot the histogram of numerical features.

- shown below

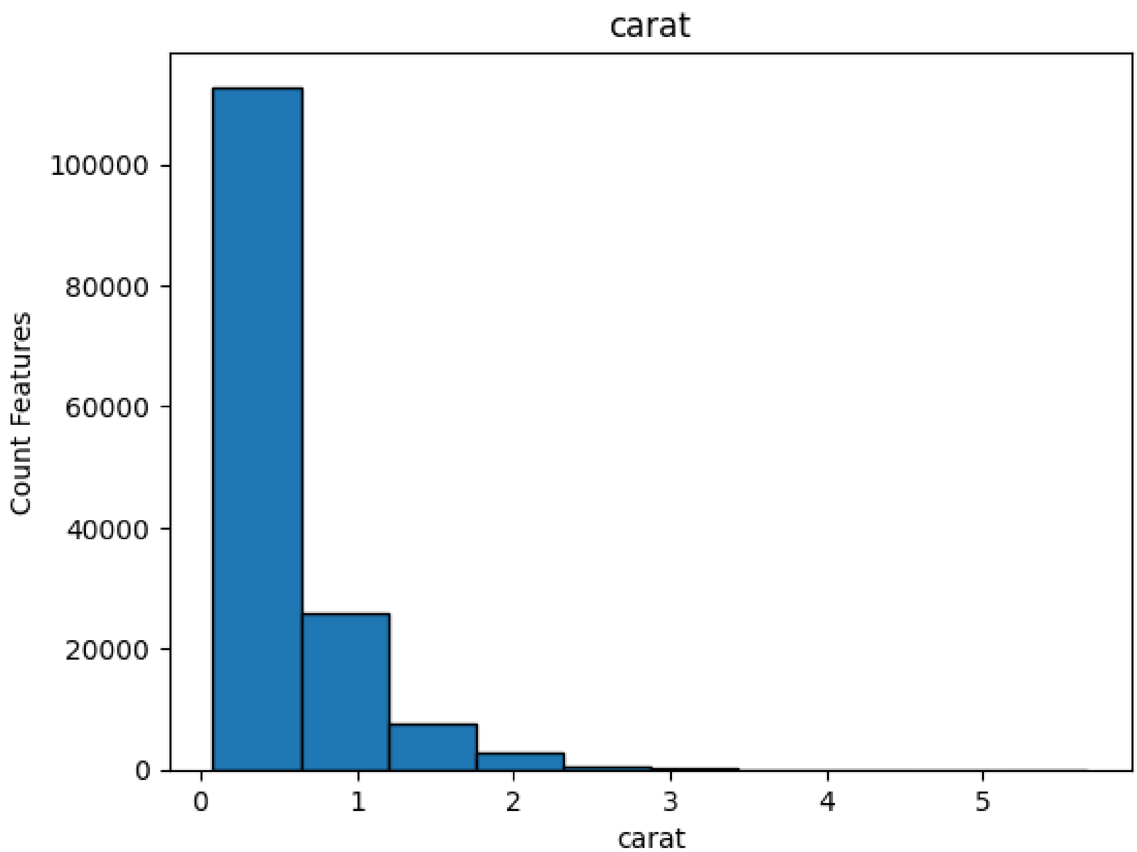
What preprocessing can be done if the distribution of a feature has high skewness?

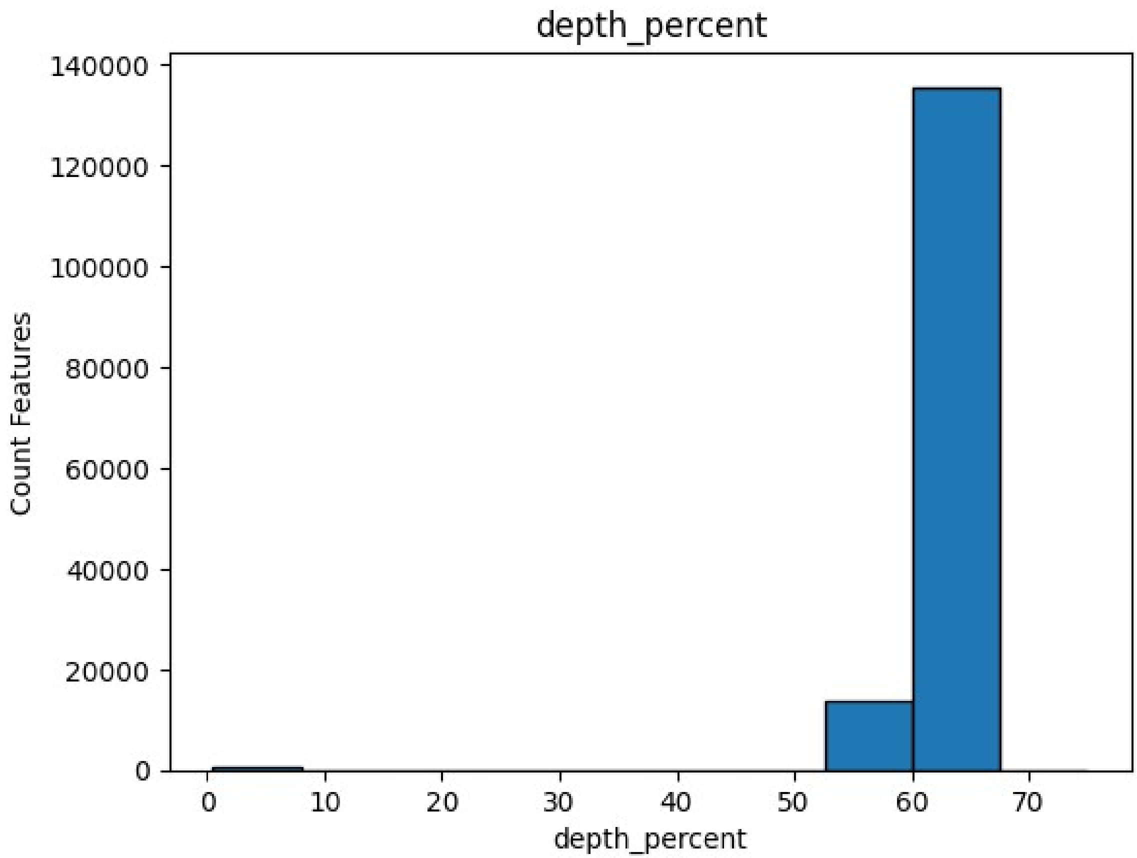
- High skewness means a distribution curve has a shorter tail on one end a distribution curve and a long tail on the other. In other words, it means that the data set the data is not evenly distributed and the data points favor one side of the distribution due to the nature of the underlying data.

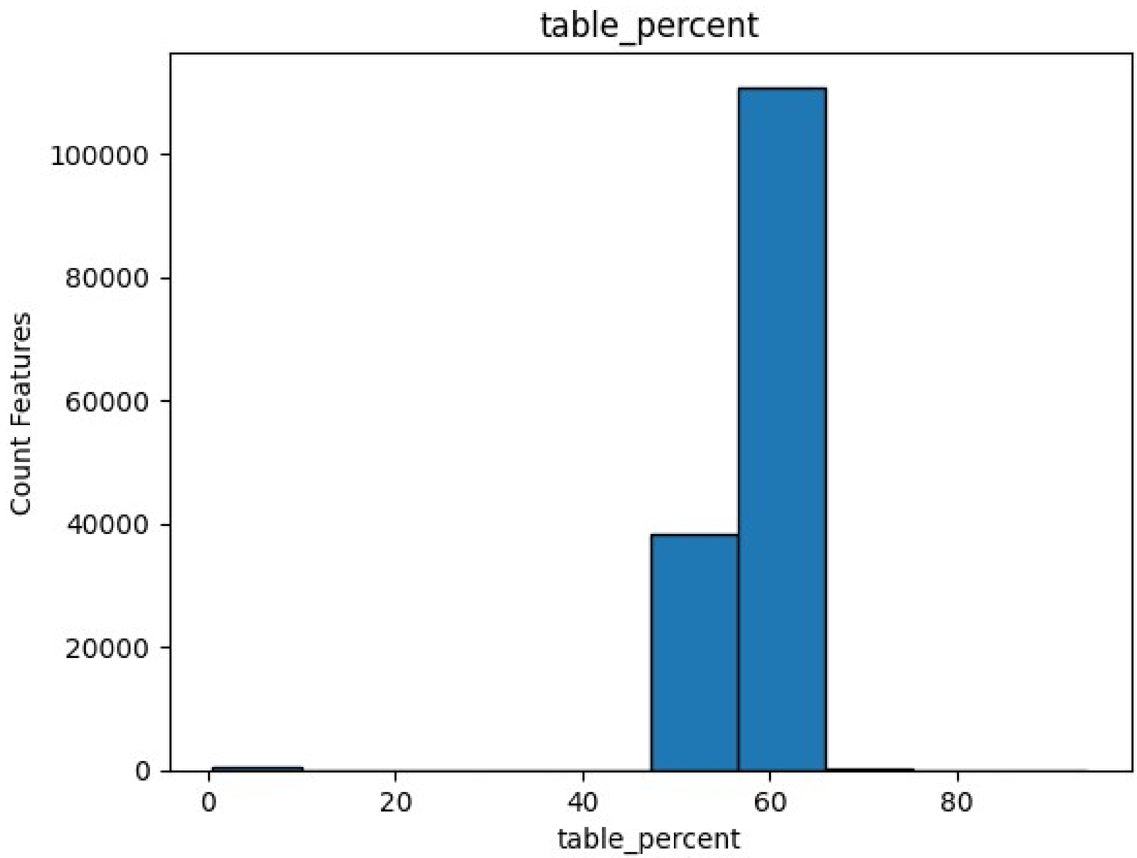
- There are various ways to transform data skewness, depending on the type and degree of skewness, and the goal of the transformation. For reducing positive skewness only, logarithmic transformation(apply natural logarithm function to data) and square root transformation(apply square root function to data) could be used. Logarithmic transformation cannot handle negative or zero values, while square root transformation can handle zero.

- For reducing both positive and negative skewness, cube root transformation is useful for data that follows a skewed normal distribution, also able to handle zero, negatie, and positive variables. Box-Cox transformation and Yeo-Johnson transformation applies a power function to minimize skewness, but parameter estimation is required for both of these methods, which can be done through maximum likelihood or cross-validation.

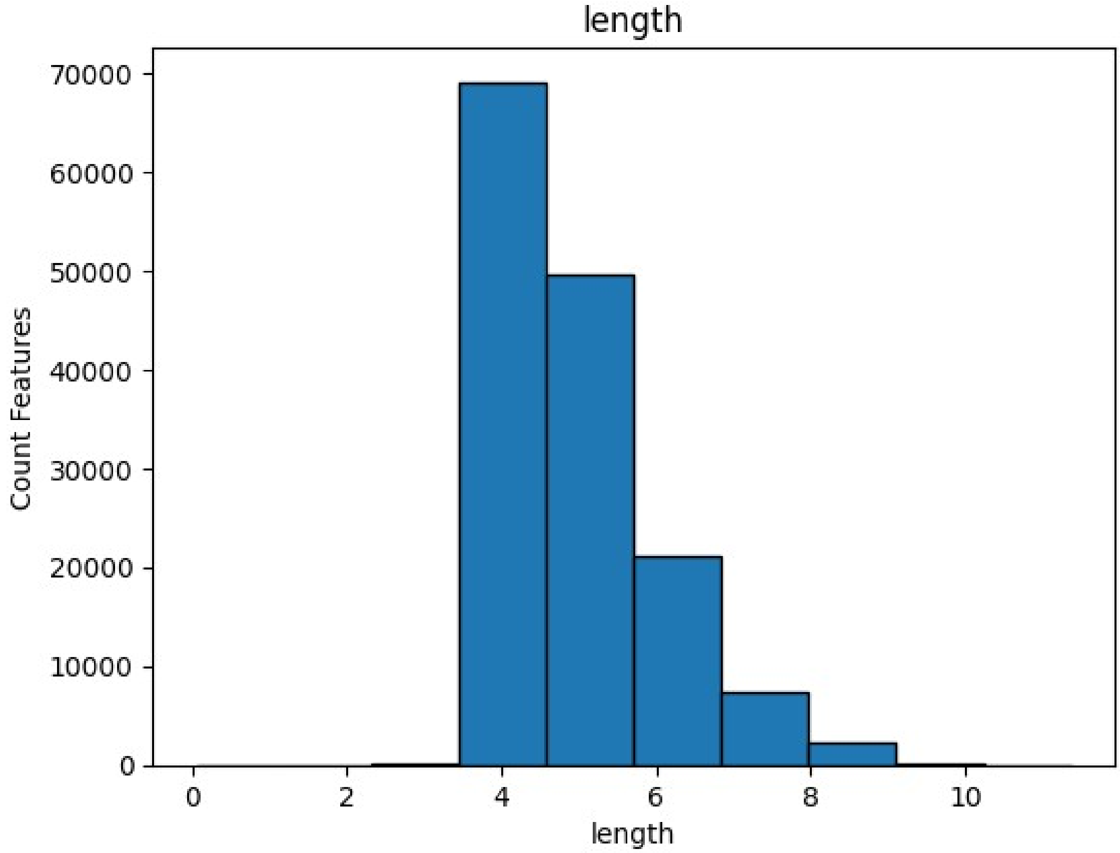
```python
num_features = ['carat', 'depth_percent', 'table_percent', 'length',
'width', 'depth', 'price']

for i in np.arange(len(num_features)) :
  plt.figure()
  plt.hist(diamonds[num_features[i]], edgecolor = "black")
  plt.xlabel(f"{num_features[i]}");
  plt.ylabel("Count Features");
  plt.title(f"{num_features[i]}")
```

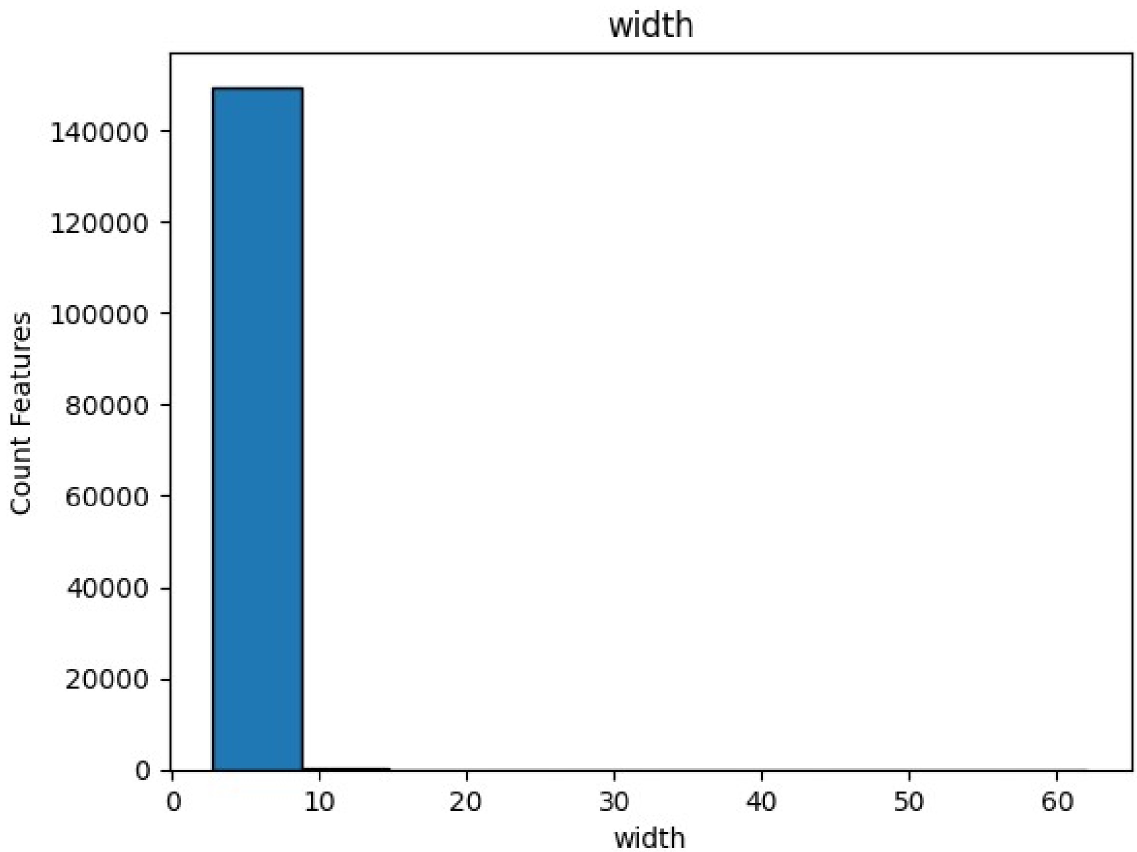

carat
Count Features
100000
80000
60000
40000
20000
0
0
1
2
3
4
5
carat

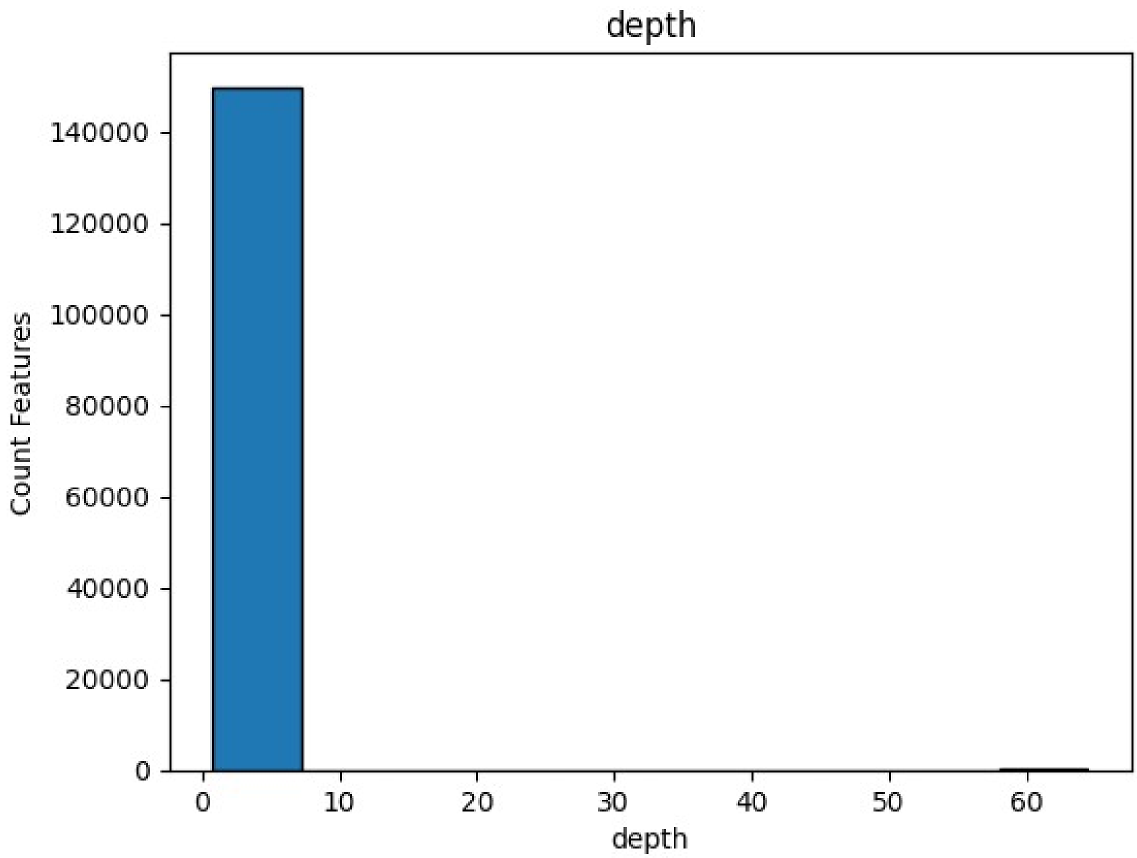

depth_percent
140000
120000
100000
80000
60000
40000
20000
0
Count Features
0
10
20
30
40
50
60
70
depth_percent

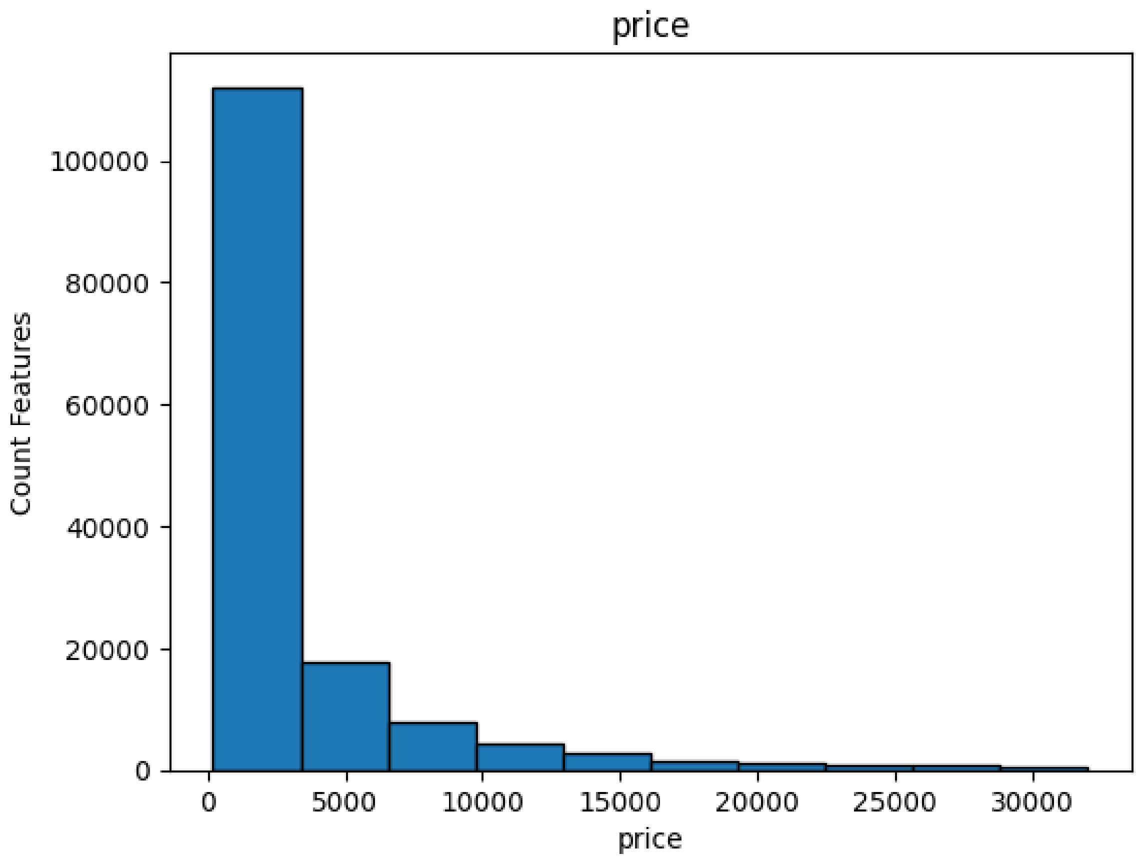


table_percent
Count Features
0
20000
40000
60000
80000
100000
0
20
40
60
80
table_percent


length
Count Features
70000
60000
50000
40000
30000
20000
10000
0
0
2
4
6
8
10
length


width
Count Features
140000
120000
100000
80000
60000
40000
20000
0
0
10
20
30
40
50
60
width


depth
Count Features
140000
120000
100000
80000
60000
40000
20000
0
0
10
20
30
40
50
60
depth

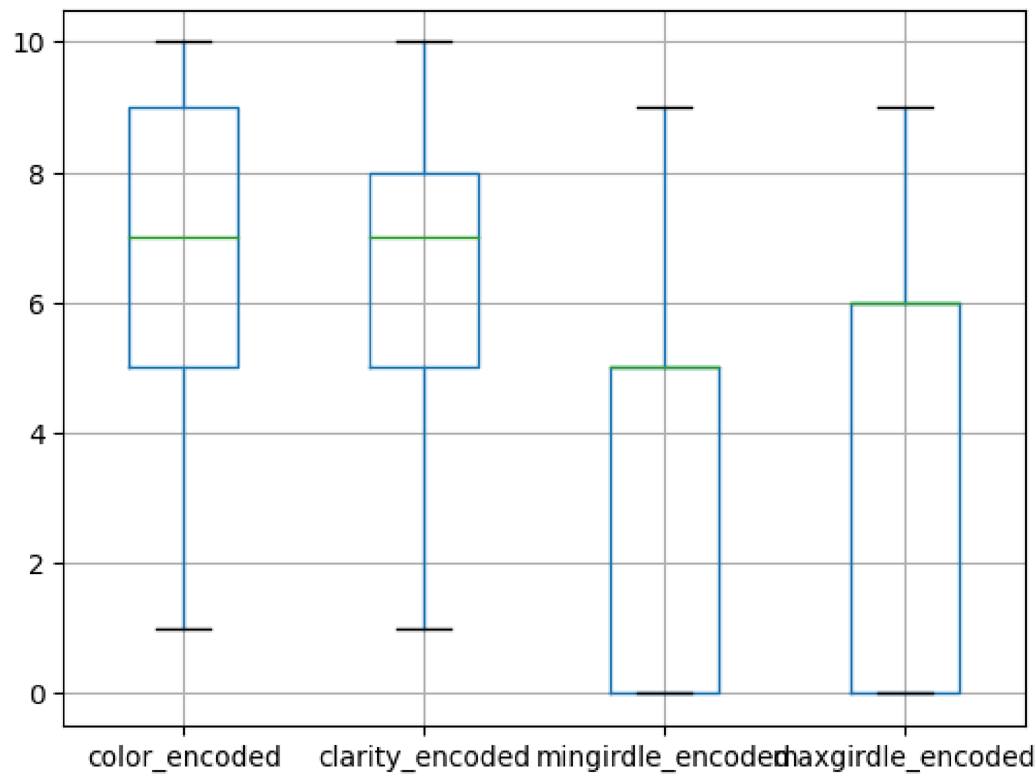


Question 1.3

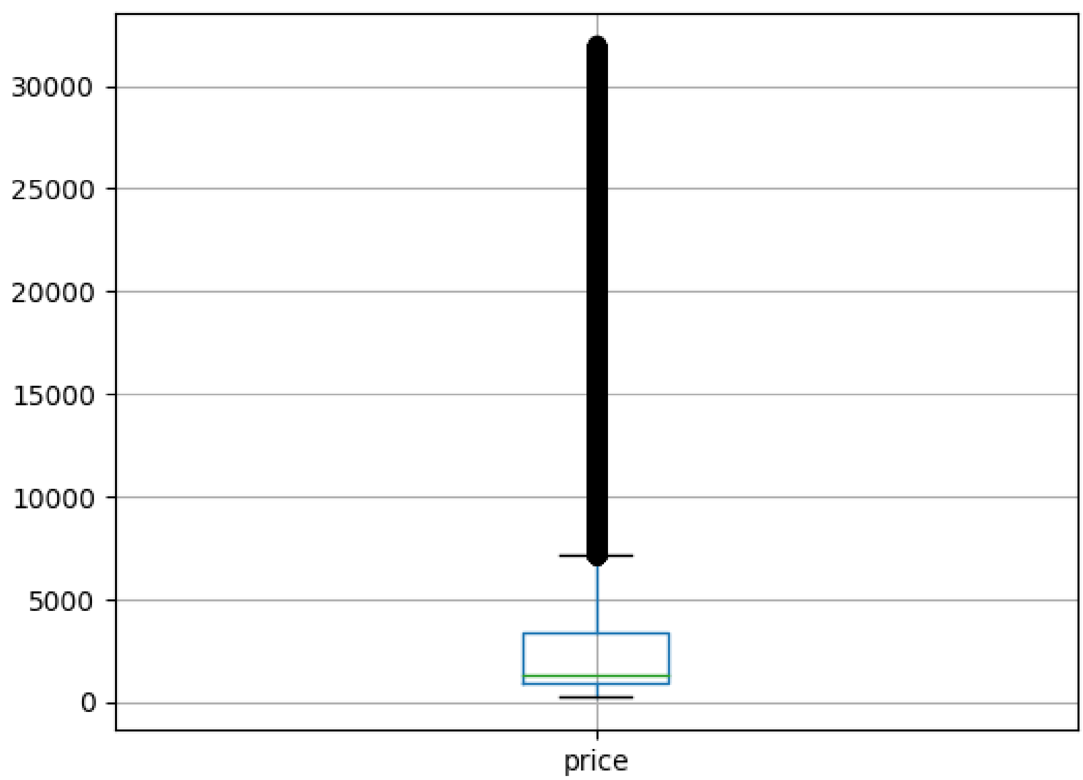
Construct and inspect the box plot of categorical features vs target variable. What do you find?

- Categorical and Price seperately
  - price box has a huge outlier
  - color and clarity has smae median
  - girdle is highly skewed
- Categorical vs Price
  - there are many outliers for each categorical feature, meaning that there are a significant number of samples that fall out of the box plots based on the inter-quartile range.

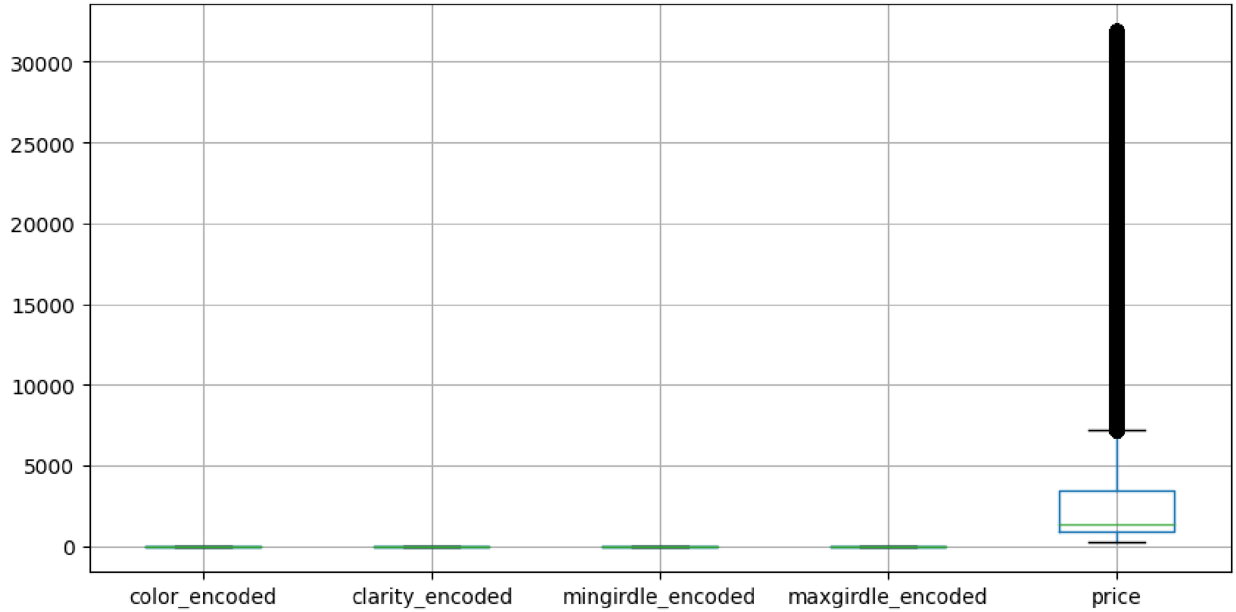
```
boxplot = diamonds.boxplot(column = ['color_encoded',
'clarity_encoded', 'mingirdle_encoded', 'maxgirdle_encoded'])
```



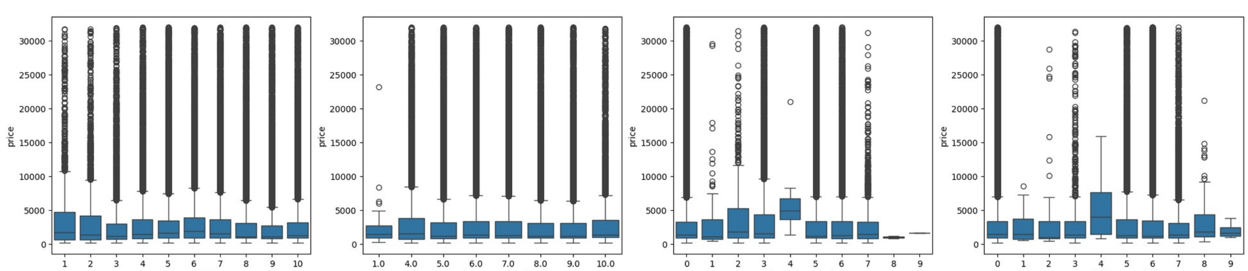

```
boxplot_p = diamonds.boxplot(column = ['price'])
```



```
box_c = diamonds.boxplot(column = ['color_encoded', 'clarity_encoded',
'mingirdle_encoded', 'maxgirdle_encoded', 'price'], figsize=(10,5))
```




```
fig, axs = plt.subplots(1,4,figsize=(25,5))
sns.boxplot(data=diamonds.sort_values("color_encoded"),x="color_encode
d", y="price", ax=axs[0])
sns.boxplot(data=diamonds.sort_values("clarity_encoded"),x="clarity_en
coded", y="price", ax=axs[1])
sns.boxplot(data=diamonds.sort_values("mingirdle_encoded"),x="mingirdl
e_encoded", y="price", ax=axs[2])
sns.boxplot(data=diamonds.sort_values("maxgirdle_encoded"),x="maxgirdl
e_encoded", y="price", ax=axs[3])

<Axes: xlabel='maxgirdle_encoded', ylabel='price'>
```

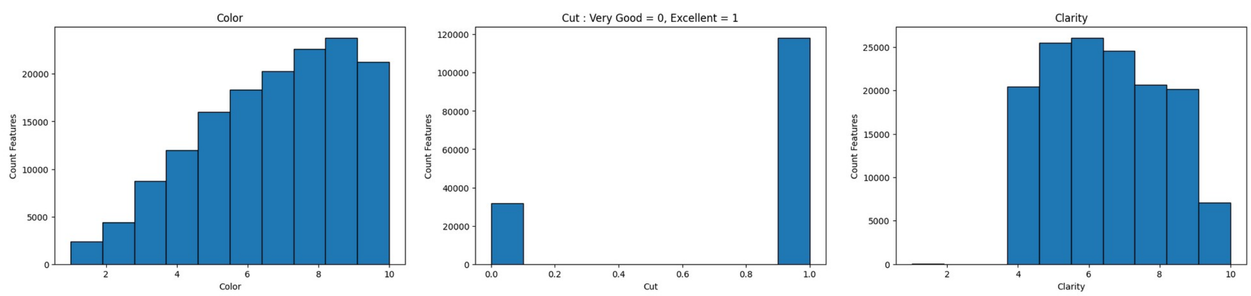


Question 1.4

For the Diamonds dataset, plot the counts by color, cut and clarity.
For the wine quality dataset, plot histogram for quality scores.

```
fig, axs = plt.subplots(1,3,figsize=(25,5))

plt.subplot(1, 3, 1)
plt.hist(diamonds['color_encoded'], edgecolor = "black")
plt.xlabel("Color")
plt.ylabel("Count Features")
plt.title("Color")

plt.subplot(1, 3, 2)
plt.hist(diamonds['cut_Excellent'], edgecolor = "black")
plt.xlabel("Cut")
plt.ylabel("Count Features");
plt.title("Cut : Very Good = 0, Excellent = 1")

plt.subplot(1, 3, 3)
plt.hist(diamonds['clarity_encoded'], edgecolor = "black")
plt.xlabel("Clarity")
plt.ylabel("Count Features")
plt.title("Clarity")
```

```
Text(0.5, 1.0, 'Clarity')
```



3.1.3 Standardization

Question 2.1

Standardize feature columns and prepare them for training.

```
from sklearn.preprocessing import StandardScaler

d_scale = StandardScaler()

from sklearn.model_selection import train_test_split

diamonds = diamonds.dropna(axis=0)
Xd = diamonds.loc[:, diamonds.columns != 'price']
Yd = diamonds.price

X_train, X_test, y_train, y_test = train_test_split(Xd, Yd,
test_size=0.2, random_state=42)
```

```
Xtrain_s = d_scale.fit_transform(X_train, y_train)
Xtrain_s = Xtrain_s[:, ~np.isnan(Xtrain_s).any(axis=0)]
Xtest_s = d_scale.transform(X_test)
```

3.1.4 Feature Selection

Question 2.2

. sklearn.feature selection.mutual_info_regression
. sklearn.feature selection.f_regression

You **may** use these functions to select features that yield better regression results (especially in the classical models).

Describe how this step qualitatively affects the performance of your models in terms of test RMSE. Is it true for all model types? Also list two features for either dataset that has the lowest MI w.r.t to the target.

- mutual information regression and f regrassion shows the relevance between the given features and the target feature. By looking at these scores, we can select the features that has significant relevence with the target value, making the model performance better.
- As F-test captures only linear dependency, mutual information can capture any kind of dependency between variables, enabling to show more accurant relevance between features for most of the cases.
- The two features with lowest MI are table percent and polish.

From this point on, you are free to use any combination of features, as long as the performance on the regression model is on par (or slightly worse) than the Neural Network model.

- We tried various choices of feature selection, and found out that the regression gives the best result(lowest rmse) when we use all the features. Therefore we chose to select every features.

```
from sklearn.feature_selection import mutual_info_regression,
f_regression

mi = mutual_info_regression(Xtrain_s, y_train)

mi

array([1.37932702, 0.04128527, 0.02304781, 1.20800748, 1.21706893,
       1.16895949, 0.03058611, 0.02753621, 0.01147866, 0.18303087,
       0.14816494, 0.02659134, 0.0402315 ])

mi_info = (
    pd.DataFrame({
        'feature_name': diamonds.columns[0:13],
        'mutual information score': mi
    })
    .sort_values('mutual information score', ascending=False)
```

```python
    .reset_index(drop=True)
)
print(mi_info)
```

```
          feature_name  mutual information score
0                carat                  1.379327
1                width                  1.217069
2               length                  1.208007
3                depth                  1.168959
4        color_encoded                  0.183031
5      clarity_encoded                  0.148165
6        depth_percent                  0.041285
7    maxgirdle_encoded                  0.040232
8        cut_Excellent                  0.030586
9   symmetry_Excellent                  0.027536
10   mingirdle_encoded                  0.026591
11       table_percent                  0.023048
12    polish_Excellent                  0.011479
```

```python
f_statistic, p_values = f_regression(Xtrain_s, y_train)

print(f_statistic)
print(p_values)
```

```
[6.20037878e+05 6.88053174e+01 2.03046214e+02 3.75155313e+05
 2.81714962e+05 1.13292868e+04 8.05690760e+01 2.95598960e+02
 4.04957886e+02 2.68679769e+02 4.90295856e+01 1.58103740e+00
 2.53383889e+00]
[0.00000000e+00 1.09830063e-16 4.94508605e-46 0.00000000e+00
 0.00000000e+00 0.00000000e+00 2.84785112e-19 3.62361296e-66
 6.54869905e-90 2.57556344e-60 2.53498221e-12 2.08613868e-01
 1.11431563e-01]
```

```python
f_info = (
    pd.DataFrame({
        'feature_name': diamonds.columns[0:13],
        'f statistic score': f_statistic
    })
    .sort_values('f statistic score', ascending=False)
    .reset_index(drop=True)
)
print(f_info)
```

```
          feature_name  f statistic score
0                carat      620037.877719
1               length      375155.312920
2                width      281714.962264
3                depth       11329.286814
4     polish_Excellent         404.957886
5   symmetry_Excellent         295.598960
6        color_encoded         268.679769
```

```
7        table_percent        203.046214
8        cut_Excellent         80.569076
9        depth_percent         68.805317
10     clarity_encoded         49.029586
11   maxgirdle_encoded          2.533839
12   mingirdle_encoded          1.581037
```

```
p_info = (
    pd.DataFrame({
        'feature_name': diamonds.columns[0:13],
        'p value score': p_values
    })
    .sort_values('p value score', ascending=False)
    .reset_index(drop=True)
)
print(p_info)
```

```
          feature_name  p value score
0    mingirdle_encoded   2.086139e-01
1    maxgirdle_encoded   1.114316e-01
2      clarity_encoded   2.534982e-12
3        depth_percent   1.098301e-16
4        cut_Excellent   2.847851e-19
5        table_percent   4.945086e-46
6        color_encoded   2.575563e-60
7   symmetry_Excellent   3.623613e-66
8      polish_Excellent   6.548699e-90
9                carat   0.000000e+00
10              length   0.000000e+00
11               width   0.000000e+00
12               depth   0.000000e+00
```

3.2 Training

Once the data is prepared, we would like to train multiple algorithms and compare their performance using average RMSE from 10-fold cross-validation (please refer to part 3.3).

3.3 Evaluation

Perform 10-fold cross-validation and measure average RMSE errors for training and validation sets. For random forest model, measure “Out-of-Bag Error” (OOB) as well.

```
from sklearn.model_selection import KFold, cross_val_score,
GridSearchCV
from sklearn.metrics import mean_squared_error
from sklearn.ensemble import RandomForestRegressor
```

```
from sklearn.linear_model import LinearRegression, Lasso, Ridge
from sklearn.metrics import r2_score

'''
def RMSE_10fold(model, X, y, k=10) :
  kf = KFold(n_splits=k)
  score = cross_val_score(model, X, y, cv= kf,
scoring="neg_mean_squared_error")
  rms_avg = np.mean(-score)
  rms_avg = np.sqrt(rms_avg)
  return rms_avg
'''

def RMSE_10fold(model, X, y) :
  score = cross_val_score(model, X, y, cv=10,
scoring='neg_root_mean_squared_error')
  rmse_avg = np.mean(-score)
  return rmse_avg
```

3.3.1 Linear Regression

What is the objective function? Train three models:

(a) ordinary least squares (linear regression without regularization),

(b) Lasso

(c) Ridge regression

Question 4.1

Explain how each regularization scheme affects the learned parameter set.

- Lasso uses L1-norm regularization and Ridge uses L2-norm regularization with the base of linear regression.

The L1 norm is calculated as the sum of the absolute vector values, where the absolute value of a scalar uses the notation |a1|. In effect, the norm is a calculation of the Manhattan distance from the origin of the vector space.

The L2 norm calculates the distance of the vector coordinate from the origin of the vector space, calculated as the square root of the sum of the squared vector values. As such, it is calculated as the Euclidean distance from the origin. The result is a positive distance value.

As L2 is Euclidean distance, there is always one right answer as to how to get between two points fastest. On the other side, as L1 is the Manhattan distance, there are many solutions to getting between two points.

```
lr = LinearRegression()
lr.fit(Xtrain_s, y_train)
y_pred_lr = lr.predict(Xtest_s)
```

```
score_lr = lr.score(Xtest_s, y_test)
print("Linear Regression Accuracy:", score_lr)
print("Avg RMSE LR = ", RMSE_10fold(lr, Xtest_s, y_test))
```

```
Linear Regression Accuracy: 0.8907552739552268
Avg RMSE LR =  1571.4483649983126
```

```
print("Linear Regression Weights : ", lr.coef_)
print("Linear Regression Bias : ", lr.intercept_)
```

```
Linear Regression Weights :  [ 5.64845530e+03 -1.46482586e+02
1.10358163e+02 -1.06174252e+03
 -3.73435693e+01 -2.20405064e+01  9.39880808e+01  3.82422549e+01
  4.61987256e+00  7.99187081e+02  4.63345029e+02 -1.40629273e+02
  2.01948312e+02]
Linear Regression Bias :  3279.35935931605
```

Lasso model is a regression model where loss function is the linear least squares function(linear regression) with L1-norm regularization.

```
lasso = Lasso(alpha=0.1)
lasso.fit(Xtrain_s, y_train)

y_pred_lasso = lasso.predict(Xtest_s)
score_lasso = lasso.score(Xtest_s, y_test)

print("Lasso Accuracy = ", score_lasso)
print("Avg RMSE Lasso = ", RMSE_10fold(lasso, Xtest_s, y_test))
```

```
Lasso Accuracy =  0.8907502349265347

/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/
_coordinate_descent.py:631: ConvergenceWarning: Objective did not
converge. You might want to increase the number of iterations, check
the scale of the features or consider increasing regularisation.
Duality gap: 1.473e+09, tolerance: 5.785e+07
  model = cd_fast.enet_coordinate_descent(

Avg RMSE Lasso =  1567.9597373937333
```

```
print("Lasso Weights : ", lasso.coef_)
print("Lasso Bias : ", lasso.intercept_)
```

```
Lasso Weights :  [ 5.64456686e+03 -1.46009221e+02  1.09893821e+02 -
1.05795284e+03
 -3.72360250e+01 -2.20477118e+01  9.39063278e+01  3.81349687e+01
  4.50849853e+00  7.99166392e+02  4.63371151e+02 -1.38648305e+02
  1.99997850e+02]
Lasso Bias :  3279.35935931605
```

Ridge model is a regression model where loss function is the linear least squares function(linear regression) with L2-norm regularization.

```
ridge = Ridge(alpha=0.1)
ridge.fit(Xtrain_s, y_train)

y_pred_r = ridge.predict(Xtest_s)
score_ridge = ridge.score(Xtest_s, y_test)
print("Ridge Accuracy = ", score_ridge)
print("Avg RMSE Ridge = ", RMSE_10fold(ridge, Xtest_s, y_test))
```

```
Ridge Accuracy =  0.8907551595222356
Avg RMSE Ridge =  1571.4444475369896
```

```
print("Ridge Weights : ", ridge.coef_)
print("Ridge Bias : ", ridge.intercept_)
```

```
Ridge Weights :  [ 5.64834834e+03 -1.46479196e+02  1.10355138e+02 -
1.06162739e+03
 -3.73534165e+01 -2.20413546e+01  9.39868392e+01  3.82416242e+01
  4.61979830e+00  7.99187339e+02  4.63347070e+02 -1.40623735e+02
  2.01943619e+02]
Ridge Bias :  3279.35935931605
```

Question 4.2

Report your choice of the best regularization scheme along with the optimal penalty parameter and explain how you computed it.

- optimal penalty parameter is the 'alpha' in function
- Lasso : {'alpha': 0.01, 'max_iter': 2000, 'selection': 'random', 'tol': 0.0001}
- Ridge : {'alpha': 0.01, 'max_iter': 2000, 'solver': 'sag', 'tol': 0.01}
- we used GridSearchCV function to search the best parameters for each functions

```
grids_l = {'alpha' : [0.01, 0.1, 1, 10],
         'max_iter' : np.arange(500, 2500, 500),
         'tol' : [1e-4, 1e-3, 1e-2],
         'selection' : ['cyclic', 'random']
        }

# define search
search = GridSearchCV(lasso, grids_l,
scoring='neg_mean_absolute_error', cv=KFold(n_splits=10), n_jobs=-1,
error_score=0)
# perform the search
results = search.fit(Xtrain_s, y_train)
# summarize
```

```
print('RMSE: %.6f' % -results.best_score_)
print('Config: %s' % results.best_params_)

RMSE: 874.596263
Config: {'alpha': 0.01, 'max_iter': 2000, 'selection': 'random',
'tol': 0.0001}
```

```
from sklearn.model_selection import GridSearchCV
# define grid
grids_r = {'alpha' : (0.01, 10, 0.1),
         'max_iter' : (100, 2000, 100),
         'tol' : (1e-4, 1e-2, 1e-3),
         'solver' : ['cholesky', 'lsqr', 'svd', 'sag', 'saga', 'auto',
'sparse_cg']
        }

# define search
search = GridSearchCV(ridge, grids_r,
scoring='neg_mean_absolute_error', cv=10, n_jobs=-1, error_score=0)
# perform the search
results = search.fit(Xtrain_s, y_train)
# summarize
print('RMSE: %.6f' % -results.best_score_)
print('Config: %s' % results.best_params_)

RMSE: 874.008188
Config: {'alpha': 0.01, 'max_iter': 2000, 'solver': 'sag', 'tol':
0.01}
```

Question 4.3

Does feature standardization play a role in improving the model performance (in the cases with ridge regularization)? Justify your answer.

- Feature standardization does play a role in improving the model performance, but in a very small scale for the given dataset.
- Feature standardization makes the values of each feature in the data have zero-mean (subtract the mean in the numerator) and unit-variance. However, StandardScaler is sensitive to outliers, and the features may scale differently from each other in the presence of outliers. By question 1.3, we can see that the categorical features have a large set of outliers with the price. This could be the reason of the standardization having less effect of improving the performance.

```
ridge_ns = Ridge(alpha=0.01, max_iter=2000, solver='sag', tol=0.01)
ridge_ns.fit(X_train, y_train)

y_pred_r_ns = ridge_ns.predict(X_test)
score_ridge_ns = ridge_ns.score(X_test, y_test)
```

```
rmse_ridge_ns = RMSE_10fold(ridge_ns, X_test, y_test)

print("Non Standardization Ridge Accuracy = ", score_ridge_ns)

print("Non Standardization Ridge Avg RMSE = ", rmse_ridge_ns)

Non Standardization Ridge Accuracy =  0.8901293343307807
Non Standardization Ridge Avg RMSE =  1608.3354617425325

ridge_s = Ridge(alpha=0.01, max_iter=2000, solver='sag', tol=0.01)
ridge_s.fit(Xtrain_s, y_train)

y_pred_r_s = ridge_s.predict(Xtest_s)
score_ridge_s = ridge_s.score(Xtest_s, y_test)

rmse_ridge_s = RMSE_10fold(ridge_s, Xtest_s, y_test)

print("Standardization Ridge Accuracy = ", score_ridge_s)
print("Standardization Ridge Avg RMSE = ", rmse_ridge_s)

Standardization Ridge Accuracy =  0.8907121259395575
Standardization Ridge Avg RMSE =  1571.9259501298427
```

Question 4.4

Some linear regression packages return p-values for different features. (E.g: scipy.stats.linregress and statsmodels.regression.linear model.OLS)

What is the meaning of these p-values and how can you infer the most significant features? A qualitative reasoning is sufficient.

- The p-values in regression models are a statistical number to conclude if there is a relationship between the given feature and the target feature. It helps to see if the relationships that we observe in a sample also exist in the larger population. When a p value is large, it indicates there is insufficient evidence in your sample to conclude that a non-zero correlation exists. So this means that those features are not helpful in determining the target value, in this case the price.

```
from scipy.stats import linregress

slope, intercept, r, p, se = linregress(Xtrain_s[:,0].T, y_train)
print('p value for carat feature with price : ',p)

p value for carat feature with price is:  0.0

slope, intercept, r, p, se = linregress(Xtrain_s[:,11].T, y_train)
print('p value for girdle_min feature with price : ',p)

p value for girdle_min feature with price is:  0.20861386806642787
```

3.3.2 Polynomial Regression

Perform polynomial regression by crafting products of features you selected in part 3.1.4 up to a certain degree (max degree 6) and applying ridge regression on the compound features. You can use scikit-learn library to build such features. Avoid overfitting by proper regularization. Answer the following:

Question 5.1

What are the most salient features? Why?

- the most salient features are carat, depth, and width
- the salient(important) features has larger weight(ceofficient). In polynomial features, the features are multiplied with another feature, so we can find the feature importance by inspecting the combination of features. As carat, depth, and width composes the hightest polynomial coefficients, we can conclude that they have the largest weight individually also.

```
from sklearn.preprocessing import PolynomialFeatures

pr = PolynomialFeatures(degree = 3)
X_poly = pr.fit_transform(Xtrain_s)
r_poly = Ridge(alpha=0.01)
r_poly.fit(X_poly, y_train)
y_pred_poly = r_poly.predict(X_poly)

rmse = np.sqrt(mean_squared_error(y_train, y_pred_poly))
r2 = r2_score(y_train, y_pred_poly)

print("RMSE : ", rmse)
print("R2 score : ", r2)
```

```
RMSE :  703.4698060366562
R2 score :  0.9776251173563598
```

```
pol_feat = (
    pd.DataFrame({
        'feature_name':
pr.get_feature_names_out(input_features=Xd.columns),
        'polynomial feaures coefficient': np.abs(r_poly.coef_)
    })
    .sort_values('polynomial feaures coefficient', ascending=False)
    .reset_index(drop=True)
)
print(pol_feat.head())
```

```
                feature_name  polynomial feaures coefficient
0                carat depth                     6690.213975
1                width depth                     4079.548139
2             carat^2 length                     3448.230409
```

```
3  carat width color_encoded                     3381.887668
4              carat^2 width                     3125.428435

X_t_poly = pr.fit_transform(Xtest_s)
r_poly = Ridge(alpha=0.1)
r_poly.fit(X_poly, y_train)
yt_pred_poly = r_poly.predict(X_t_poly)

rmse = np.sqrt(mean_squared_error(y_test, yt_pred_poly))
r2 = r2_score(y_test, yt_pred_poly)

print("RMSE : ", rmse)
print("R2 score : ", r2)

RMSE :  729.0126713067305
R2 score :  0.9763685876177075
```

Question 5.2

What degree of polynomial is best? How did you find the optimal degree? What does a very high-order polynomial imply about the fit on the training data? What about its performance on testing data?

- Degree of 3 of polynomial works best as it has the smallest RMSE with high score for the test dataset.
- This optimal degree was found by trying the individual dregrees from 2 to 4, as degree of 1 is linear regresison.
- With degree over 4, the model training showed overfitting to the training dataset, giving hight score for the training dataset but functioning very badly for the testing dataset.

```
def create_polynomial_regression_model(degree):
  "Creates a polynomial regression model for the given degree"

  poly_features = PolynomialFeatures(degree=degree)

  # transforms the existing features to higher degree features.
  X_train_poly = poly_features.fit_transform(Xtrain_s)

  # fit the transformed features to Linear Regression
  poly_model = Ridge(alpha=0.1)
  poly_model.fit(X_train_poly, y_train)

  # predicting on training data-set
  y_train_predicted = poly_model.predict(X_train_poly)

  # predicting on test data-set
  y_test_predict =
poly_model.predict(poly_features.fit_transform(Xtest_s))

  # evaluating the model on training dataset
  rmse_train = np.sqrt(mean_squared_error(y_train, y_train_predicted))
```

```
  r2_train = r2_score(y_train, y_train_predicted)

  # evaluating the model on test dataset
  rmse_test = np.sqrt(mean_squared_error(y_test, y_test_predict))
  r2_test = r2_score(y_test, y_test_predict)

  print("Model performance for the training set")
  print("-------------------------------------------")
  print("Train set RMSE = {}".format(rmse_train))
  print("Train set R2 score = {}".format(r2_train))

  print("\n")

  print("Model performance for the test set")
  print("-------------------------------------------")
  print("Test set RMSE = {}".format(rmse_test))
  print("Test set R2 score = {}".format(r2_test))
  print("----------------------------------------------------------")

# this requires large size RAM(colab pro) from defree 5
for i in np.arange(2, 5) :
  print("\n")
  print("Degree = ", i)
  create_polynomial_regression_model(i)
```

```


Degree =  2
Model performance for the training set
-----------------------------------------
Train set RMSE = 832.6080377475163
Train set R2 score = 0.968656243761814


Model performance for the test set
-----------------------------------------
Test set RMSE = 817.9738745442479
Test set R2 score = 0.9702492176638191
--------------------------------------------------------


Degree =  3
Model performance for the training set
-----------------------------------------
Train set RMSE = 703.5671366306475
Train set R2 score = 0.9776189254451941


Model performance for the test set
-----------------------------------------
Test set RMSE = 729.0126713067305
```

```
Test set R2 score = 0.9763685876177075
--------------------------------------------------------


Degree =  4

/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/
_ridge.py:216: LinAlgWarning: Ill-conditioned matrix (rcond=1.06147e-
16): result may not be accurate.
  return linalg.solve(A, Xy, assume_a="pos", overwrite_a=True).T

Model performance for the training set
-----------------------------------------
Train set RMSE = 647.5027228471321
Train set R2 score = 0.9810437232302042


Model performance for the test set
-----------------------------------------
Test set RMSE = 14988.169267570729
Test set R2 score = -8.988877866656368
--------------------------------------------------------
```

3.3.3 Neural Network

You will train a multi-layer perceptron (fully connected neural network). You can simply use the sklearn implementation:

Question 6.1

Adjust your network size (number of hidden neurons and depth), and weight decay as regularization. Find a good hyper-parameter set systematically (no more than 20 experiments in total).

```
from sklearn.neural_network import MLPRegressor
import itertools
from sklearn.model_selection import GridSearchCV

lst = np.arange(10, 101, 40)
print(lst)
layers = []
for n in [1, 2]:
  combs = list(itertools.combinations_with_replacement(lst, n))
  print(combs)
  layers.extend(combs)

print(len(layers))
```

```
[10 50 90]
[(10,), (50,), (90,)]
```

```
[(10, 10), (10, 50), (10, 90), (50, 50), (50, 90), (90, 90)]
9
```

```
params = {'hidden_layer_sizes': layers}
nn = MLPRegressor(random_state=42, max_iter=1000)
gs = GridSearchCV(nn, params, cv=2, n_jobs=1,
scoring='neg_root_mean_squared_error', verbose=2,
return_train_score=True)

gs.fit(X_poly, y_train)
print("Best Config:", gs.best_params_)
print("Best RMSE:", -gs.best_score_)
```

```
Fitting 2 folds for each of 9 candidates, totalling 18 fits
[CV] END ...........................hidden_layer_sizes=(10,); total
time=  29.3s
[CV] END ...........................hidden_layer_sizes=(10,); total
time=  59.5s
[CV] END ...........................hidden_layer_sizes=(50,); total
time=  40.8s
[CV] END ...........................hidden_layer_sizes=(50,); total
time=  49.9s
[CV] END ...........................hidden_layer_sizes=(90,); total
time=  55.9s
[CV] END ...........................hidden_layer_sizes=(90,); total
time=  47.0s
[CV] END ........................hidden_layer_sizes=(10, 10); total
time=  52.3s
[CV] END ........................hidden_layer_sizes=(10, 10); total
time= 3.1min
[CV] END ........................hidden_layer_sizes=(10, 50); total
time= 4.6min
[CV] END ........................hidden_layer_sizes=(10, 50); total
time= 5.1min
[CV] END ........................hidden_layer_sizes=(10, 90); total
time= 2.9min
[CV] END ........................hidden_layer_sizes=(10, 90); total
time= 3.9min
[CV] END ........................hidden_layer_sizes=(50, 50); total
time= 5.0min
[CV] END ........................hidden_layer_sizes=(50, 50); total
time=  47.8s
[CV] END ........................hidden_layer_sizes=(50, 90); total
time= 8.1min
[CV] END ........................hidden_layer_sizes=(50, 90); total
time= 1.6min
[CV] END ........................hidden_layer_sizes=(90, 90); total
time= 6.2min
[CV] END ........................hidden_layer_sizes=(90, 90); total
time=  32.2s
```

```
Best Config: {'hidden_layer_sizes': (10, 50)}
Best RMSE: -680.0506720277828
```

```
params = {'activation': ['identity','relu','tanh','logistic']}
nn = MLPRegressor(hidden_layer_sizes=(10,50), random_state=42,
max_iter=1000)
gs = GridSearchCV(nn, params, cv=2, n_jobs=1,
scoring='neg_root_mean_squared_error', verbose=3,
return_train_score=True)

gs.fit(X_poly, y_train)
print("Best Config:", gs.best_params_)
print("Best RMSE:", -gs.best_score_)
```

```
Fitting 2 folds for each of 4 candidates, totalling 8 fits
[CV 1/2] END activation=identity;, score=(train=-910.243, test=-
996.976) total time=  38.0s
[CV 2/2] END activation=identity;, score=(train=-861.017, test=-
804.003) total time=  37.5s
[CV 1/2] END activation=relu;, score=(train=-599.045, test=-658.524)
total time= 4.8min
[CV 2/2] END activation=relu;, score=(train=-607.008, test=-701.577)
total time= 5.4min

/usr/local/lib/python3.10/dist-packages/sklearn/neural_network/
_multilayer_perceptron.py:686: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (1000) reached and the optimization
hasn't converged yet.
  warnings.warn(

[CV 1/2] END activation=tanh;, score=(train=-1803.865, test=-1832.126)
total time=12.4min

/usr/local/lib/python3.10/dist-packages/sklearn/neural_network/
_multilayer_perceptron.py:686: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (1000) reached and the optimization
hasn't converged yet.
  warnings.warn(

[CV 2/2] END activation=tanh;, score=(train=-1816.560, test=-1814.817)
total time=12.1min

/usr/local/lib/python3.10/dist-packages/sklearn/neural_network/
_multilayer_perceptron.py:686: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (1000) reached and the optimization
hasn't converged yet.
  warnings.warn(

[CV 1/2] END activation=logistic;, score=(train=-1800.807, test=-
1829.220) total time= 8.8min
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/neural_network/
_multilayer_perceptron.py:686: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (1000) reached and the optimization
hasn't converged yet.
  warnings.warn(

[CV 2/2] END activation=logistic;, score=(train=-1812.426, test=-
1807.335) total time= 9.1min
Best Config: {'activation': 'relu'}
Best RMSE: -680.0506720277828

params = {'alpha': [10.0**x for x in np.arange(-2,3)]}
nn = MLPRegressor(hidden_layer_sizes=(10,50), activation='relu',
random_state=42, max_iter=1000)
gs = GridSearchCV(nn, params, cv=2, n_jobs=1,
scoring='neg_root_mean_squared_error', verbose=2,
return_train_score=True)

gs.fit(X_poly, y_train)
print("Best Config:", gs.best_params_)
print("Best RMSE:", -gs.best_score_)

Fitting 2 folds for each of 5 candidates, totalling 10 fits
[CV] END .........................................alpha=0.01; total
time= 2.4min
[CV] END .........................................alpha=0.01; total
time= 4.6min
[CV] END ..........................................alpha=0.1; total
time= 3.3min
[CV] END ..........................................alpha=0.1; total
time= 4.2min
[CV] END ..........................................alpha=1.0; total
time= 4.3min
[CV] END ..........................................alpha=1.0; total
time= 3.0min
[CV] END .........................................alpha=10.0; total
time=  36.6s
[CV] END .........................................alpha=10.0; total
time= 4.6min
[CV] END ........................................alpha=100.0; total
time=  40.3s
[CV] END ........................................alpha=100.0; total
time= 2.2min
Best Config: {'alpha': 0.01}
Best RMSE: -704.2584275880779
```

Question 6.2

How does the performance generally compare with linear regression? Why?

- The performance is generally better than linear regression.

- Linear regression assumes a linear relationship while a neural network can model non-linear relationships in data.
- Neural networks can also automatically learn relevant features from the data.
- Hyperparameters also allow neural networks to offer more flexibility in model architecture.

Question 6.3

What activation function did you use for the output and why? You may use none.

- Relu: the rectified linear unit function, returns f(x) = max(0, x)
- This gave us the best RMSE out of the 4 options from gridsearch.

Question 6.4

What is the risk of increasing the depth of the network too far?

- A network that is too deep can lead to overfitting because of a higher capacity to memorize the training data causing it to fail to generalize new data.
- Too many layers can also make optimization harder due to the increased complexity and computation time.

3.3.4 Random Forest

We will train a random forest regression model on datasets, and answer the following:

Question 7.1

Random forests have the following hyper-parameters:

- Maximum number of features
- Number of trees
- Depth of each tree

Explain how these hyper-parameters affect the overall performance. Describe if and how each hyper-parameter results in a regularization effect during training.

- Number of estimators choose the number of trees. More trees usually increases accuracy but slowers learning.
  - For the given model, the number of estimators showed a very slight increase of performance as number grew.
- Maximum number of feature is the number of features to consider each time when making the split decision. It can be used to interpret regularization and control overfitting. If the independent variables are highly correlated, we should decrease the maximum number of features.
  - For the given model, the maximum number of features increased the performance as number grew, but for the test set, the degree of the performance improvement got very slow.
- Maximum depth of each tree decides the degree of detail to capture. Adding more depth makes the model more complex and captures more information about the data. Increasing maximum dapth can increase training accuracy, but can cause overfitting.

Therfore limiting the depth of can be a regularization method since it helps prevent overfitting.

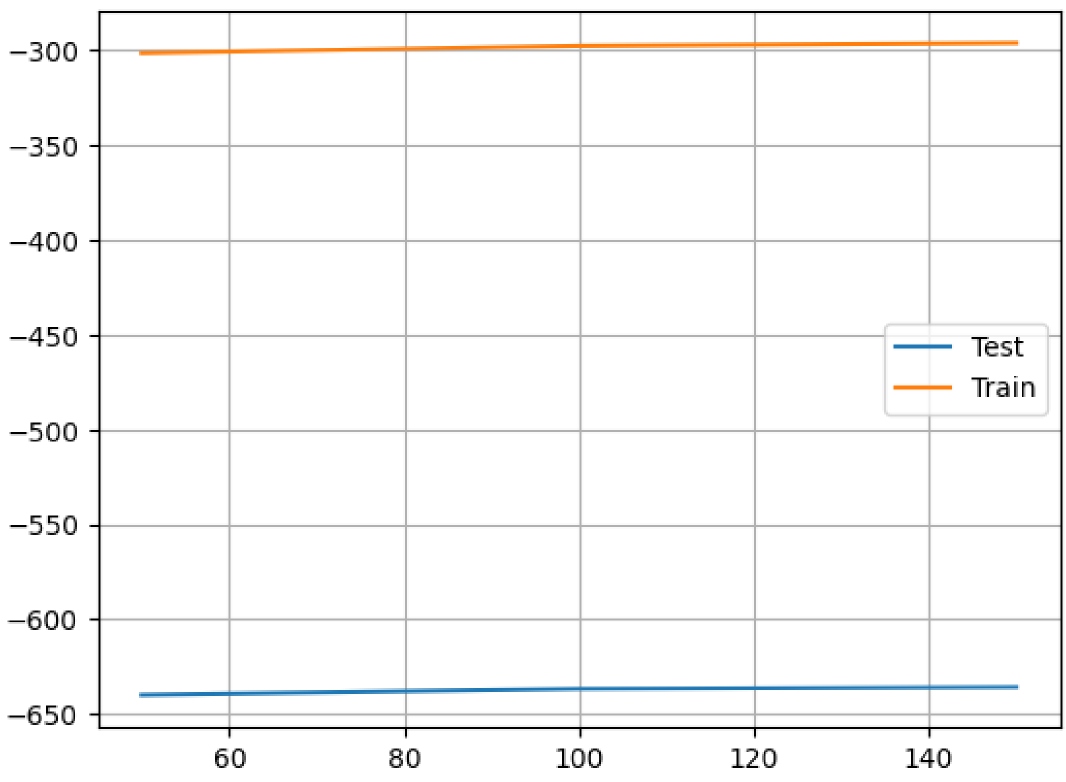
- For the given model, the maximum depth increased signiticantly for the train set as the number grew, but didn't show much improvement for the test set after 10.

```
from sklearn.ensemble import RandomForestRegressor
from sklearn.tree import export_graphviz
import pydot
from IPython.display import Image
from sklearn.model_selection import GridSearchCV

params = {'n_estimators': np.arange(50, 151, 50),
          'max_features': np.arange(2, 14, 2),
          'max_depth':np.arange(5, 21, 5)
          }
rf = RandomForestRegressor(random_state=42,  oob_score=True)

gs_rf = GridSearchCV(rf, params, cv=2, n_jobs=1, verbose=1,
                     scoring='neg_root_mean_squared_error',
return_train_score=True)

gs_rf.fit(Xtrain_s, y_train)

Fitting 2 folds for each of 72 candidates, totalling 144 fits

GridSearchCV(cv=2,
             estimator=RandomForestRegressor(oob_score=True,
random_state=42),
             n_jobs=1,
             param_grid={'max_depth': array([ 5, 10, 15, 20]),
                         'max_features': array([ 2,  4,  6,  8, 10,
12]),
                         'n_estimators': array([ 50, 100, 150])},
             return_train_score=True,
scoring='neg_root_mean_squared_error',
             verbose=1)

rf_result = pd.DataFrame(gs_rf.cv_results_)
[['mean_test_score','mean_train_score','param_max_features','param_n_e
stimators','param_max_depth']]
print('Best Config:', gs_rf.best_params_)
print('Test RMSE:', -gs_rf.best_score_)
print('Train RMSE:', -max(rf_result.mean_train_score))

Best Config: {'max_depth': 15, 'max_features': 10, 'n_estimators':
150}
Test RMSE: 635.8782355788082
Train RMSE: 241.6692680857762

rf_best = RandomForestRegressor(n_estimators=150, max_features=10,
max_depth=15, random_state=42, oob_score=True)
```

```
rf_best.fit(Xtrain_s, y_train)
print('OOB Score:', rf_best.oob_score_)
```

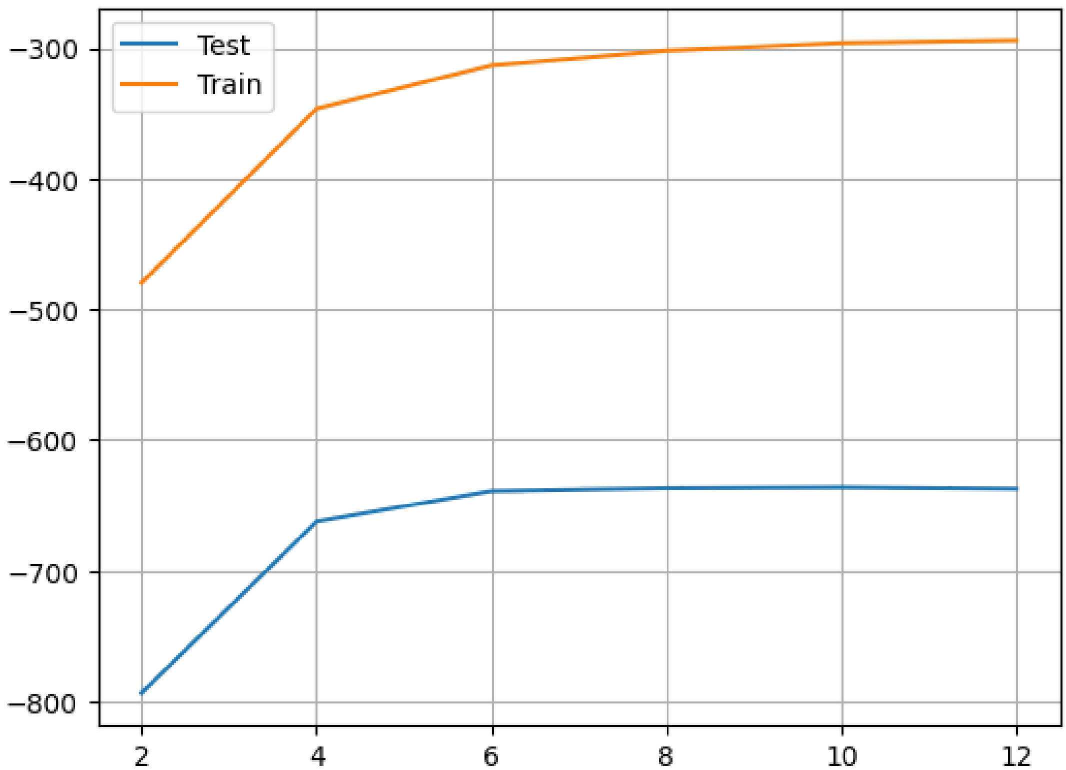
```
OOB Score: 0.9830593711555686
```

```
n_estimators = np.arange(50, 151, 50).reshape(3)
max_features = np.arange(2, 14, 2).reshape(6)
max_depth = np.arange(5, 21, 5).reshape(4)

test_score = list((rf_result[(rf_result['param_max_depth'] == 15) &
(rf_result['param_max_features'] == 10)]).mean_test_score)
train_score = list((rf_result[(rf_result['param_max_depth'] == 15) &
(rf_result['param_max_features'] == 10)]).mean_train_score)
plt.plot(n_estimators, test_score, label = 'Test')
plt.plot(n_estimators, train_score, label = 'Train')

plt.legend()
plt.grid()
```

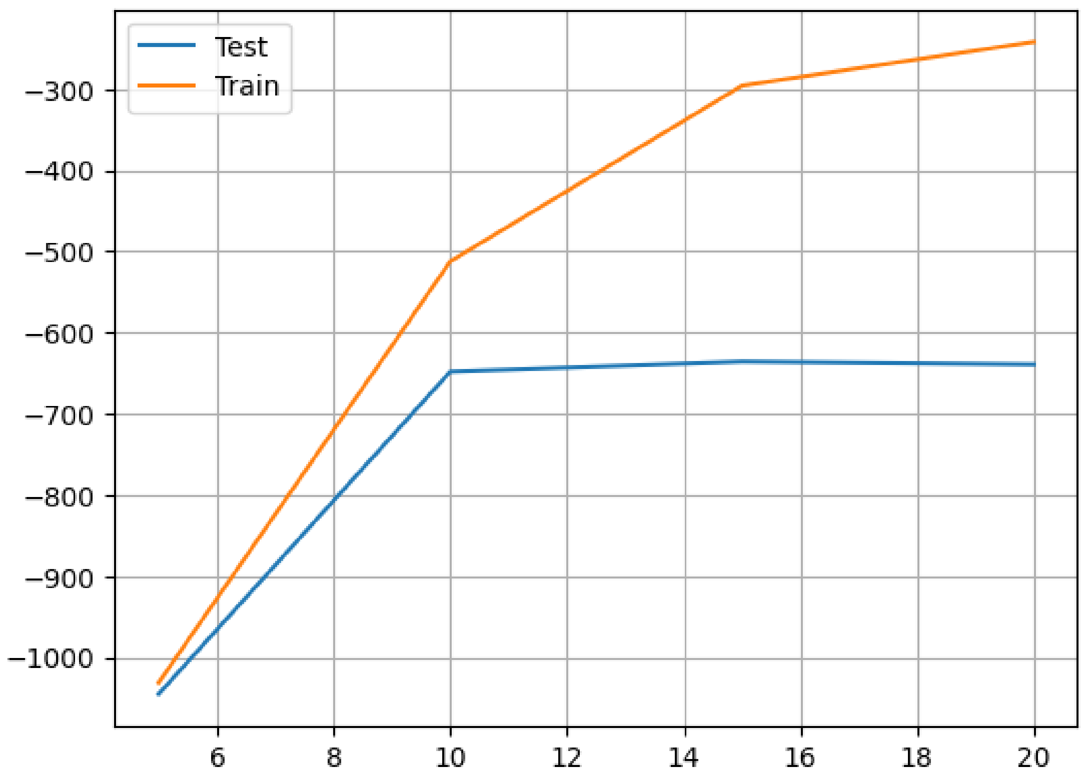



```
test_score = list((rf_result[(rf_result['param_max_depth'] == 15) &
(rf_result['param_n_estimators'] == 150)]).mean_test_score)
train_score = list((rf_result[(rf_result['param_max_depth'] == 15) &
(rf_result['param_n_estimators'] == 150)]).mean_train_score)
plt.plot(max_features, test_score, label = 'Test')
plt.plot(max_features, train_score, label = 'Train')
```

```
plt.legend()
plt.grid()
```




```
test_score = list((rf_result[(rf_result['param_max_features'] == 10) &
(rf_result['param_n_estimators'] == 150)]).mean_test_score)
train_score = list((rf_result[(rf_result['param_max_features'] == 10)
& (rf_result['param_n_estimators'] == 150)]).mean_train_score)
plt.plot(max_depth, test_score, label = 'Test')
plt.plot(max_depth, train_score, label = 'Train')
plt.legend()
plt.grid()
```



Question 7.2

How do random forests create a highly non-linear decision boundary despite the fact that all we do at each layer is apply a threshold on a feature?

- By combining multiple decision trees with different thresholds on different features and aggregating their predictions.

Question 7.3

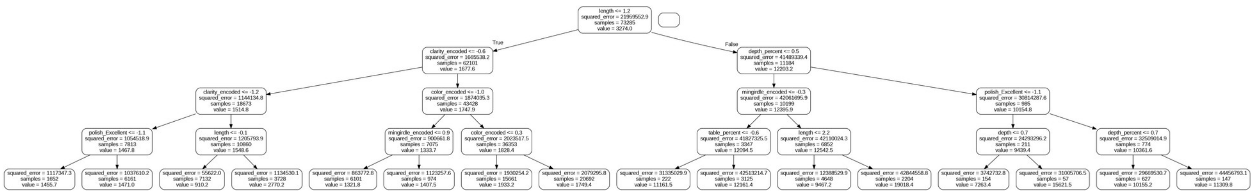
Randomly pick a tree in your random forest model (with maximum depth of 4) and plot its structure. Which feature is selected for branching at the root node? What can you infer about the importance of this feature as opposed to others? Do the important features correspond to what you got in part 3.3.1?

- Length is the feature selected for branching at the root node.
- This means that length may be more important for predicting price than the other features.
- This aligns with the conclusions from 3.3.1

```
rf_rand = RandomForestRegressor(n_estimators=80, max_features=1,
max_depth=4, random_state=42, oob_score=True)
rf_rand.fit(Xtrain_s, y_train)
```

```
RandomForestRegressor(max_depth=4, max_features=1, n_estimators=80,
                      oob_score=True, random_state=42)
```

```
tree = rf_rand.estimators_[1]
export_graphviz(tree, out_file = 'tree.dot', feature_names =
diamonds.columns[0:13], rounded = True, precision = 1)
(graph, ) = pydot.graph_from_dot_file('tree.dot')
Image(graph.create_png())
```



3.3.5 LightGBM, CatBoost and Bayesian Optimization

Question 8.1

Read the documentation of LightGBM OR CatBoost and determine the important hyperparameters along with a search space for the tuning of these parameters (keep the search space small).

- determined learning rate, depth, and l2 regularization coefficient(l2_leaf_reg) as important hyperparameters
- search space indicated in params
- polynomial features discarded due to long running time

```
!pip install catboost

Collecting catboost
  Downloading catboost-1.2.3-cp310-cp310-manylinux2014_x86_64.whl
(98.5 MB)
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 98.5/98.5 MB 3.4 MB/s eta
0:00:00
ent already satisfied: graphviz in /usr/local/lib/python3.10/dist-
packages (from catboost) (0.20.1)
Requirement already satisfied: matplotlib in
/usr/local/lib/python3.10/dist-packages (from catboost) (3.7.1)
Requirement already satisfied: numpy>=1.16.0 in
/usr/local/lib/python3.10/dist-packages (from catboost) (1.25.2)
Requirement already satisfied: pandas>=0.24 in
/usr/local/lib/python3.10/dist-packages (from catboost) (1.5.3)
Requirement already satisfied: scipy in
/usr/local/lib/python3.10/dist-packages (from catboost) (1.11.4)
Requirement already satisfied: plotly in
/usr/local/lib/python3.10/dist-packages (from catboost) (5.15.0)
Requirement already satisfied: six in /usr/local/lib/python3.10/dist-
packages (from catboost) (1.16.0)
Requirement already satisfied: python-dateutil>=2.8.1 in
/usr/local/lib/python3.10/dist-packages (from pandas>=0.24->catboost)
(2.8.2)
```

```
Requirement already satisfied: pytz>=2020.1 in
/usr/local/lib/python3.10/dist-packages (from pandas>=0.24->catboost)
(2023.4)
Requirement already satisfied: contourpy>=1.0.1 in
/usr/local/lib/python3.10/dist-packages (from matplotlib->catboost)
(1.2.0)
Requirement already satisfied: cycler>=0.10 in
/usr/local/lib/python3.10/dist-packages (from matplotlib->catboost)
(0.12.1)
Requirement already satisfied: fonttools>=4.22.0 in
/usr/local/lib/python3.10/dist-packages (from matplotlib->catboost)
(4.49.0)
Requirement already satisfied: kiwisolver>=1.0.1 in
/usr/local/lib/python3.10/dist-packages (from matplotlib->catboost)
(1.4.5)
Requirement already satisfied: packaging>=20.0 in
/usr/local/lib/python3.10/dist-packages (from matplotlib->catboost)
(24.0)
Requirement already satisfied: pillow>=6.2.0 in
/usr/local/lib/python3.10/dist-packages (from matplotlib->catboost)
(9.4.0)
Requirement already satisfied: pyparsing>=2.3.1 in
/usr/local/lib/python3.10/dist-packages (from matplotlib->catboost)
(3.1.2)
Requirement already satisfied: tenacity>=6.2.0 in
/usr/local/lib/python3.10/dist-packages (from plotly->catboost)
(8.2.3)
Installing collected packages: catboost
Successfully installed catboost-1.2.3
```

```
!pip install ipywidgets
```

```
Requirement already satisfied: ipywidgets in
/usr/local/lib/python3.10/dist-packages (7.7.1)
Requirement already satisfied: ipykernel>=4.5.1 in
/usr/local/lib/python3.10/dist-packages (from ipywidgets) (5.5.6)
Requirement already satisfied: ipython-genutils~=0.2.0 in
/usr/local/lib/python3.10/dist-packages (from ipywidgets) (0.2.0)
Requirement already satisfied: traitlets>=4.3.1 in
/usr/local/lib/python3.10/dist-packages (from ipywidgets) (5.7.1)
Requirement already satisfied: widgetsnbextension~=3.6.0 in
/usr/local/lib/python3.10/dist-packages (from ipywidgets) (3.6.6)
Requirement already satisfied: ipython>=4.0.0 in
/usr/local/lib/python3.10/dist-packages (from ipywidgets) (7.34.0)
Requirement already satisfied: jupyterlab-widgets>=1.0.0 in
/usr/local/lib/python3.10/dist-packages (from ipywidgets) (3.0.10)
Requirement already satisfied: jupyter-client in
/usr/local/lib/python3.10/dist-packages (from ipykernel>=4.5.1-
>ipywidgets) (6.1.12)
Requirement already satisfied: tornado>=4.2 in
```

```
/usr/local/lib/python3.10/dist-packages (from ipykernel>=4.5.1-
>ipywidgets) (6.3.3)
Requirement already satisfied: setuptools>=18.5 in
/usr/local/lib/python3.10/dist-packages (from ipython>=4.0.0-
>ipywidgets) (67.7.2)
Collecting jedi>=0.16 (from ipython>=4.0.0->ipywidgets)
  Downloading jedi-0.19.1-py2.py3-none-any.whl (1.6 MB)
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 1.6/1.6 MB 15.7 MB/s eta
0:00:00
ent already satisfied: decorator in /usr/local/lib/python3.10/dist-
packages (from ipython>=4.0.0->ipywidgets) (4.4.2)
Requirement already satisfied: pickleshare in
/usr/local/lib/python3.10/dist-packages (from ipython>=4.0.0-
>ipywidgets) (0.7.5)
Requirement already satisfied: prompt-toolkit!=3.0.0,!
=3.0.1,<3.1.0,>=2.0.0 in /usr/local/lib/python3.10/dist-packages (from
ipython>=4.0.0->ipywidgets) (3.0.43)
Requirement already satisfied: pygments in
/usr/local/lib/python3.10/dist-packages (from ipython>=4.0.0-
>ipywidgets) (2.16.1)
Requirement already satisfied: backcall in
/usr/local/lib/python3.10/dist-packages (from ipython>=4.0.0-
>ipywidgets) (0.2.0)
Requirement already satisfied: matplotlib-inline in
/usr/local/lib/python3.10/dist-packages (from ipython>=4.0.0-
>ipywidgets) (0.1.6)
Requirement already satisfied: pexpect>4.3 in
/usr/local/lib/python3.10/dist-packages (from ipython>=4.0.0-
>ipywidgets) (4.9.0)
Requirement already satisfied: notebook>=4.4.1 in
/usr/local/lib/python3.10/dist-packages (from
widgetsnbextension~=3.6.0->ipywidgets) (6.5.5)
Requirement already satisfied: parso<0.9.0,>=0.8.3 in
/usr/local/lib/python3.10/dist-packages (from jedi>=0.16-
>ipython>=4.0.0->ipywidgets) (0.8.3)
Requirement already satisfied: jinja2 in
/usr/local/lib/python3.10/dist-packages (from notebook>=4.4.1-
>widgetsnbextension~=3.6.0->ipywidgets) (3.1.3)
Requirement already satisfied: pyzmq<25,>=17 in
/usr/local/lib/python3.10/dist-packages (from notebook>=4.4.1-
>widgetsnbextension~=3.6.0->ipywidgets) (23.2.1)
Requirement already satisfied: argon2-cffi in
/usr/local/lib/python3.10/dist-packages (from notebook>=4.4.1-
>widgetsnbextension~=3.6.0->ipywidgets) (23.1.0)
Requirement already satisfied: jupyter-core>=4.6.1 in
/usr/local/lib/python3.10/dist-packages (from notebook>=4.4.1-
>widgetsnbextension~=3.6.0->ipywidgets) (5.7.2)
Requirement already satisfied: nbformat in
/usr/local/lib/python3.10/dist-packages (from notebook>=4.4.1-
```

```
>widgetsnbextension~=3.6.0->ipywidgets) (5.10.2)
Requirement already satisfied: nbconvert>=5 in
/usr/local/lib/python3.10/dist-packages (from notebook>=4.4.1-
>widgetsnbextension~=3.6.0->ipywidgets) (6.5.4)
Requirement already satisfied: nest-asyncio>=1.5 in
/usr/local/lib/python3.10/dist-packages (from notebook>=4.4.1-
>widgetsnbextension~=3.6.0->ipywidgets) (1.6.0)
Requirement already satisfied: Send2Trash>=1.8.0 in
/usr/local/lib/python3.10/dist-packages (from notebook>=4.4.1-
>widgetsnbextension~=3.6.0->ipywidgets) (1.8.2)
Requirement already satisfied: terminado>=0.8.3 in
/usr/local/lib/python3.10/dist-packages (from notebook>=4.4.1-
>widgetsnbextension~=3.6.0->ipywidgets) (0.18.1)
Requirement already satisfied: prometheus-client in
/usr/local/lib/python3.10/dist-packages (from notebook>=4.4.1-
>widgetsnbextension~=3.6.0->ipywidgets) (0.20.0)
Requirement already satisfied: nbclassic>=0.4.7 in
/usr/local/lib/python3.10/dist-packages (from notebook>=4.4.1-
>widgetsnbextension~=3.6.0->ipywidgets) (1.0.0)
Requirement already satisfied: python-dateutil>=2.1 in
/usr/local/lib/python3.10/dist-packages (from jupyter-client-
>ipykernel>=4.5.1->ipywidgets) (2.8.2)
Requirement already satisfied: ptyprocess>=0.5 in
/usr/local/lib/python3.10/dist-packages (from pexpect>4.3-
>ipython>=4.0.0->ipywidgets) (0.7.0)
Requirement already satisfied: wcwidth in
/usr/local/lib/python3.10/dist-packages (from prompt-toolkit!=3.0.0,!
=3.0.1,<3.1.0,>=2.0.0->ipython>=4.0.0->ipywidgets) (0.2.13)
Requirement already satisfied: platformdirs>=2.5 in
/usr/local/lib/python3.10/dist-packages (from jupyter-core>=4.6.1-
>notebook>=4.4.1->widgetsnbextension~=3.6.0->ipywidgets) (4.2.0)
Requirement already satisfied: jupyter-server>=1.8 in
/usr/local/lib/python3.10/dist-packages (from nbclassic>=0.4.7-
>notebook>=4.4.1->widgetsnbextension~=3.6.0->ipywidgets) (1.24.0)
Requirement already satisfied: notebook-shim>=0.2.3 in
/usr/local/lib/python3.10/dist-packages (from nbclassic>=0.4.7-
>notebook>=4.4.1->widgetsnbextension~=3.6.0->ipywidgets) (0.2.4)
Requirement already satisfied: lxml in /usr/local/lib/python3.10/dist-
packages (from nbconvert>=5->notebook>=4.4.1-
>widgetsnbextension~=3.6.0->ipywidgets) (4.9.4)
Requirement already satisfied: beautifulsoup4 in
/usr/local/lib/python3.10/dist-packages (from nbconvert>=5-
>notebook>=4.4.1->widgetsnbextension~=3.6.0->ipywidgets) (4.12.3)
Requirement already satisfied: bleach in
/usr/local/lib/python3.10/dist-packages (from nbconvert>=5-
>notebook>=4.4.1->widgetsnbextension~=3.6.0->ipywidgets) (6.1.0)
Requirement already satisfied: defusedxml in
/usr/local/lib/python3.10/dist-packages (from nbconvert>=5-
>notebook>=4.4.1->widgetsnbextension~=3.6.0->ipywidgets) (0.7.1)
```

```
Requirement already satisfied: entrypoints>=0.2.2 in
/usr/local/lib/python3.10/dist-packages (from nbconvert>=5-
>notebook>=4.4.1->widgetsnbextension~=3.6.0->ipywidgets) (0.4)
Requirement already satisfied: jupyterlab-pygments in
/usr/local/lib/python3.10/dist-packages (from nbconvert>=5-
>notebook>=4.4.1->widgetsnbextension~=3.6.0->ipywidgets) (0.3.0)
Requirement already satisfied: MarkupSafe>=2.0 in
/usr/local/lib/python3.10/dist-packages (from nbconvert>=5-
>notebook>=4.4.1->widgetsnbextension~=3.6.0->ipywidgets) (2.1.5)
Requirement already satisfied: mistune<2,>=0.8.1 in
/usr/local/lib/python3.10/dist-packages (from nbconvert>=5-
>notebook>=4.4.1->widgetsnbextension~=3.6.0->ipywidgets) (0.8.4)
Requirement already satisfied: nbclient>=0.5.0 in
/usr/local/lib/python3.10/dist-packages (from nbconvert>=5-
>notebook>=4.4.1->widgetsnbextension~=3.6.0->ipywidgets) (0.10.0)
Requirement already satisfied: packaging in
/usr/local/lib/python3.10/dist-packages (from nbconvert>=5-
>notebook>=4.4.1->widgetsnbextension~=3.6.0->ipywidgets) (24.0)
Requirement already satisfied: pandocfilters>=1.4.1 in
/usr/local/lib/python3.10/dist-packages (from nbconvert>=5-
>notebook>=4.4.1->widgetsnbextension~=3.6.0->ipywidgets) (1.5.1)
Requirement already satisfied: tinycss2 in
/usr/local/lib/python3.10/dist-packages (from nbconvert>=5-
>notebook>=4.4.1->widgetsnbextension~=3.6.0->ipywidgets) (1.2.1)
Requirement already satisfied: fastjsonschema in
/usr/local/lib/python3.10/dist-packages (from nbformat-
>notebook>=4.4.1->widgetsnbextension~=3.6.0->ipywidgets) (2.19.1)
Requirement already satisfied: jsonschema>=2.6 in
/usr/local/lib/python3.10/dist-packages (from nbformat-
>notebook>=4.4.1->widgetsnbextension~=3.6.0->ipywidgets) (4.19.2)
Requirement already satisfied: six>=1.5 in
/usr/local/lib/python3.10/dist-packages (from python-dateutil>=2.1-
>jupyter-client->ipykernel>=4.5.1->ipywidgets) (1.16.0)
Requirement already satisfied: argon2-cffi-bindings in
/usr/local/lib/python3.10/dist-packages (from argon2-cffi-
>notebook>=4.4.1->widgetsnbextension~=3.6.0->ipywidgets) (21.2.0)
Requirement already satisfied: attrs>=22.2.0 in
/usr/local/lib/python3.10/dist-packages (from jsonschema>=2.6-
>nbformat->notebook>=4.4.1->widgetsnbextension~=3.6.0->ipywidgets)
(23.2.0)
Requirement already satisfied: jsonschema-specifications>=2023.03.6 in
/usr/local/lib/python3.10/dist-packages (from jsonschema>=2.6-
>nbformat->notebook>=4.4.1->widgetsnbextension~=3.6.0->ipywidgets)
(2023.12.1)
Requirement already satisfied: referencing>=0.28.4 in
/usr/local/lib/python3.10/dist-packages (from jsonschema>=2.6-
>nbformat->notebook>=4.4.1->widgetsnbextension~=3.6.0->ipywidgets)
(0.33.0)
Requirement already satisfied: rpds-py>=0.7.1 in
```

```
/usr/local/lib/python3.10/dist-packages (from jsonschema>=2.6->nbformat->notebook>=4.4.1->widgetsnbextension~=3.6.0->ipywidgets) (0.18.0)
Requirement already satisfied: anyio<4,>=3.1.0 in /usr/local/lib/python3.10/dist-packages (from jupyter-server>=1.8->nbclassic>=0.4.7->notebook>=4.4.1->widgetsnbextension~=3.6.0->ipywidgets) (3.7.1)
Requirement already satisfied: websocket-client in /usr/local/lib/python3.10/dist-packages (from jupyter-server>=1.8->nbclassic>=0.4.7->notebook>=4.4.1->widgetsnbextension~=3.6.0->ipywidgets) (1.7.0)
Requirement already satisfied: cffi>=1.0.1 in /usr/local/lib/python3.10/dist-packages (from argon2-cffi-bindings->argon2-cffi->notebook>=4.4.1->widgetsnbextension~=3.6.0->ipywidgets) (1.16.0)
Requirement already satisfied: soupsieve>1.2 in /usr/local/lib/python3.10/dist-packages (from beautifulsoup4->nbconvert>=5->notebook>=4.4.1->widgetsnbextension~=3.6.0->ipywidgets) (2.5)
Requirement already satisfied: webencodings in /usr/local/lib/python3.10/dist-packages (from bleach->nbconvert>=5->notebook>=4.4.1->widgetsnbextension~=3.6.0->ipywidgets) (0.5.1)
Requirement already satisfied: idna>=2.8 in /usr/local/lib/python3.10/dist-packages (from anyio<4,>=3.1.0->jupyter-server>=1.8->nbclassic>=0.4.7->notebook>=4.4.1->widgetsnbextension~=3.6.0->ipywidgets) (3.6)
Requirement already satisfied: sniffio>=1.1 in /usr/local/lib/python3.10/dist-packages (from anyio<4,>=3.1.0->jupyter-server>=1.8->nbclassic>=0.4.7->notebook>=4.4.1->widgetsnbextension~=3.6.0->ipywidgets) (1.3.1)
Requirement already satisfied: exceptiongroup in /usr/local/lib/python3.10/dist-packages (from anyio<4,>=3.1.0->jupyter-server>=1.8->nbclassic>=0.4.7->notebook>=4.4.1->widgetsnbextension~=3.6.0->ipywidgets) (1.2.0)
Requirement already satisfied: pycparser in /usr/local/lib/python3.10/dist-packages (from cffi>=1.0.1->argon2-cffi-bindings->argon2-cffi->notebook>=4.4.1->widgetsnbextension~=3.6.0->ipywidgets) (2.21)
Installing collected packages: jedi
Successfully installed jedi-0.19.1
```

```python
from catboost import CatBoostRegressor, cv, Pool

cb_model = CatBoostRegressor(random_seed=42, learning_rate=0.1, depth=2, l2_leaf_reg=10, verbose=False)
cb_model.fit(Xtrain_s, y_train)
cb_preds = cb_model.predict(Xtest_s)

#cb_model.fit(X_poly, y_train)
```

```
#X_t_poly = pr.fit_transform(Xtest_s)
#cb_preds = cb_model.predict(X_t_poly)

np.sqrt(mean_squared_error(cb_preds, y_test))
```

```
730.274732184759
```

```
feature_importances = cb_model.get_feature_importance
feature_names = diamonds.columns

cb_model.set_feature_names(Xd.columns)

print(cb_model.get_feature_importance(prettified=True))
```

```
              Feature Id  Importances
0                  carat    32.727191
1                 length    24.815221
2                  width    22.984052
3          color_encoded     9.618398
4                  depth     6.171057
5        clarity_encoded     3.398237
6     maxgirdle_encoded     0.109600
7          table_percent     0.087339
8    symmetry_Excellent     0.025369
9          depth_percent     0.024409
10     polish_Excellent     0.015469
11         cut_Excellent     0.013866
12    mingirdle_encoded     0.009792
```

```
cb_model = CatBoostRegressor(random_seed=42, learning_rate=0.1,
depth=2)
cb_model.fit(X_poly, y_train)
X_t_poly = pr.fit_transform(Xtest_s)
cb_preds = cb_model.predict(X_t_poly)
np.sqrt(mean_squared_error(cb_preds, y_test))

A = pr.get_feature_names_out(input_features=Xd.columns)

cb_model.set_feature_names(A)

print(cb_model.get_feature_importance(prettified=True))
```

```
                                                 Feature Id  Importances
0                                             carat width^2     6.917199
1                                            length^2 width     6.714558
2                                                     width     6.578526
3                                                  length^3     5.609705
4                                         carat width depth     5.601405
..                                                      ...          ...
555    color_encoded mingirdle_encoded maxgirdle_encoded     0.000000
556                clarity_encoded maxgirdle_encoded^2     0.000000
```

```
557                                 mingirdle_encoded^3     0.000000
558                mingirdle_encoded maxgirdle_encoded^2     0.000000
559                                 maxgirdle_encoded^3     0.000000

[560 rows x 2 columns]
```

```
params = {

        #'n_estimators':   Integer(1, 500, 50), # No of boosted trees
or iterations to fit (default: 100).

        'depth': Integer(1, 16),

        'learning_rate': Real(0.01, 1.0, 'log-uniform'), # Prob of
interval 1 to 10 is same as 10 to 100
                                                         # Equal prob
of selection from 0.01 to 0.1, 0.1
                                                         # to 1
                                                         # In a
loguniform distributon, log-transformed
                                                         # random
variable is uniformly distributed

        'l2_leaf_reg': (1, 100, 10) # L2 regularization

           }
```

Question 8.2

Apply Bayesian optimization using skopt.BayesSearchCV from scikit-optmize to find the ideal hyperparameter combination in your search space. Keep your search space small enough to finish running on a single Google Colab instance within 60 minutes. Report the best hyperparameter set found and the corresponding RMSE.

```
!pip install scikit-optimize
```

```
Collecting scikit-optimize
  Downloading scikit_optimize-0.10.1-py2.py3-none-any.whl (107 kB)
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 0.0/107.7 kB ? eta -:--:--
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 107.7/107.7 kB 3.0 MB/s eta
0:00:00
ent already satisfied: joblib>=0.11 in /usr/local/lib/python3.10/dist-
packages (from scikit-optimize) (1.3.2)
Collecting pyaml>=16.9 (from scikit-optimize)
  Downloading pyaml-23.12.0-py3-none-any.whl (23 kB)
Requirement already satisfied: numpy>=1.20.3 in
/usr/local/lib/python3.10/dist-packages (from scikit-optimize)
(1.25.2)
Requirement already satisfied: scipy>=1.1.0 in
/usr/local/lib/python3.10/dist-packages (from scikit-optimize)
```

```
(1.11.4)
Requirement already satisfied: scikit-learn>=1.0.0 in
/usr/local/lib/python3.10/dist-packages (from scikit-optimize) (1.2.2)
Requirement already satisfied: packaging>=21.3 in
/usr/local/lib/python3.10/dist-packages (from scikit-optimize) (24.0)
Requirement already satisfied: PyYAML in
/usr/local/lib/python3.10/dist-packages (from pyaml>=16.9->scikit-
optimize) (6.0.1)
Requirement already satisfied: threadpoolctl>=2.0.0 in
/usr/local/lib/python3.10/dist-packages (from scikit-learn>=1.0.0-
>scikit-optimize) (3.3.0)
Installing collected packages: pyaml, scikit-optimize
Successfully installed pyaml-23.12.0 scikit-optimize-0.10.1
```

```
from skopt import BayesSearchCV
from skopt.space import Real, Categorical, Integer

reg = CatBoostRegressor(verbose = 2)

opt = BayesSearchCV(estimator=reg,
                    search_spaces=params,
                    n_iter = 10,
                    scoring='neg_root_mean_squared_error',
                    cv=KFold(n_splits=10),
                    n_points=3,
# number of hyperparameter sets evaluated at the same time
                    n_jobs=-1,
# number of jobs
                    return_train_score=True,
                    refit=False,
                    random_state=42)
# random state for replicability

res = opt.fit(Xtrain_s, y_train)
```

```
/usr/local/lib/python3.10/dist-packages/joblib/externals/loky/
process_executor.py:752: 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.
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/joblib/externals/loky/process_
executor.py:752: 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.
  warnings.warn(
```

```
print("Best set of params")
print(opt.best_params_)
print('Best RMSE:', -1*opt.best_score_)
```

```
Best set of params
OrderedDict([('depth', 7), ('l2_leaf_reg', 1), ('learning_rate',
0.08078499224286771)])
Best RMSE: 573.7923417596137

best_params = opt.best_params_
cbr = CatBoostRegressor(verbose = False, **best_params)

cbr.fit(Xtrain_s, y_train)
cbr_preds = cbr.predict(Xtest_s)

np.sqrt(mean_squared_error(cbr_preds, y_test))

575.0998460386147
```

Question 8.3

Qualitatively interpret the effect of the hyperparameters using the Bayesian optimization results: Which of them helps with performance? Which helps with regularization (shrinks the generalization gap)? Which affects the fitting efficiency?

- The 'depth' parameter chooses the depth of each trees. Increasing the depth can improve the training accuracy but when too deep, it might also cause overfitting, which increases the generalization gap. It also increases training time, which reduces fitting efficiency. Therefore appropriate depth helps with performance, regularization, and affects the fitting efficiency.

- For the given model, we can see that depth higher than 6 generally increases the performance, but increasing the depth even more doesn't assure the best performance. Generalization gap also tends to increase as depth gets higher(we can see overfitting by the high peak of training score with low test score), and fitting time increases dramatically.

- The 'learning rate' parameter chooses the learning rate when calculating the gradient descent. Smaller learning rate can improve the training accuracy but it can increase training time, and also has a risk of being captured at the local minima. Larger learning rate shortens the training time and lowers the risk of getting stuck in local minima, but it might not converge well when the rate is too large. Therefore appropriate learning rate helps with performance and increases fitting efficiency.

- For the given model, we can see lower learning rate generally increases the performance, but the lowest learning rate doesn't make the best performance. As the test score tends to follow the graph of train score, we cannot observe obvious overfitting. However, fitting time does tend to increase as learning rate gets lower.

- The 'l2_leaf_reg' parameter chooses the coefficient at the L2 regularization term of the cost function. Appropriate regularization scheme helps with performance, shrinks generalization gap, and affects the fitting efficiency.

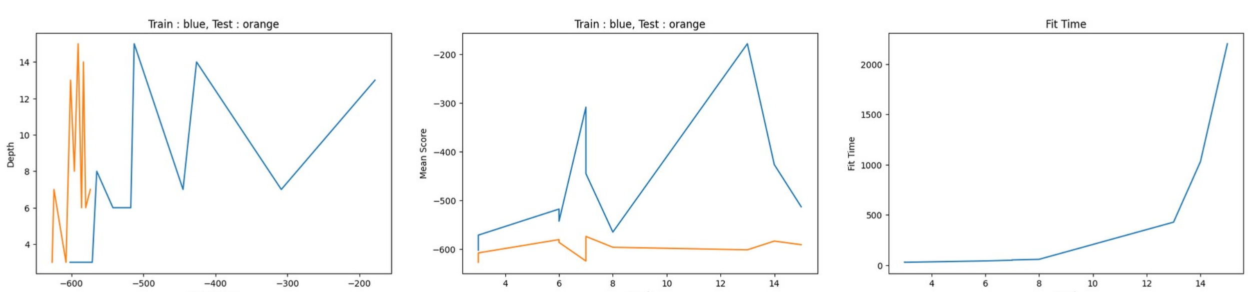
- For the given model, we can see lower coefficient gives better performance, but increases the possibility of experiencing overfitting. Also, fitting time tends to be longer for smaller parameter, but as most of the L2 leaf samples have the same value(which is 10), it is more convincing to interpret it as the affect of depth and learning rate.

```
cb_result = pd.DataFrame(opt.cv_results_)
[['mean_test_score','mean_train_score','param_depth','param_learning_r
ate','param_l2_leaf_reg', 'mean_fit_time']]

cb_result
```

{"summary":"{\n \"name\": \"cb_result\",\n \"rows\": 10,\n \"fields\": [\n   {\n   \"column\": \"mean_test_score\",\n   \"properties\": {\n   \"dtype\": \"number\",\n   \"std\": 18.07504812821892,\n   \"min\": -626.93397927236,\n   \"max\": -573.7923417596137,\n   \"num_unique_values\": 10,\n   \"samples\": [\n   -607.6209742002893,\n   -580.2971298476546,\n   -626.93397927236\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   },\n   {\n   \"column\": \"mean_train_score\",\n   \"properties\": {\n   \"dtype\": \"number\",\n   \"std\": 133.2294857753414,\n   \"min\": -601.9879533451331,\n   \"max\": -178.14106667566904,\n   \"num_unique_values\": 10,\n   \"samples\": [\n   -571.1175513152145,\n   -517.5895397524998,\n   -601.9879533451331\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   },\n   {\n   \"column\": \"param_depth\",\n   \"properties\": {\n   \"dtype\": \"date\",\n   \"min\": 3,\n   \"max\": 15,\n   \"num_unique_values\": 7,\n   \"samples\": [\n   7,\n   6,\n   8\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   },\n   {\n   \"column\": \"param_learning_rate\",\n   \"properties\": {\n   \"dtype\": \"date\",\n   \"min\": 0.013323731791098763,\n   \"max\": 0.7340675018434776,\n   \"num_unique_values\": 10,\n   \"samples\": [\n   0.13022094602394507,\n   0.0673344419215237,\n   0.18683498597281528\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   },\n   {\n   \"column\": \"param_l2_leaf_reg\",\n   \"properties\": {\n   \"dtype\": \"date\",\n   \"min\": 1,\n   \"max\": 100,\n   \"num_unique_values\": 3,\n   \"samples\": [\n   10,\n   100,\n   1\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   },\n   {\n   \"column\": \"mean_fit_time\",\n   \"properties\": {\n   \"dtype\": \"number\",\n   \"std\": 709.70564101947,\n   \"min\": 28.974887132644653,\n   \"max\": 2201.422492527962,\n   \"num_unique_values\": 10,\n   \"samples\": [\n   28.974887132644653,\n   42.71263122558594,\n

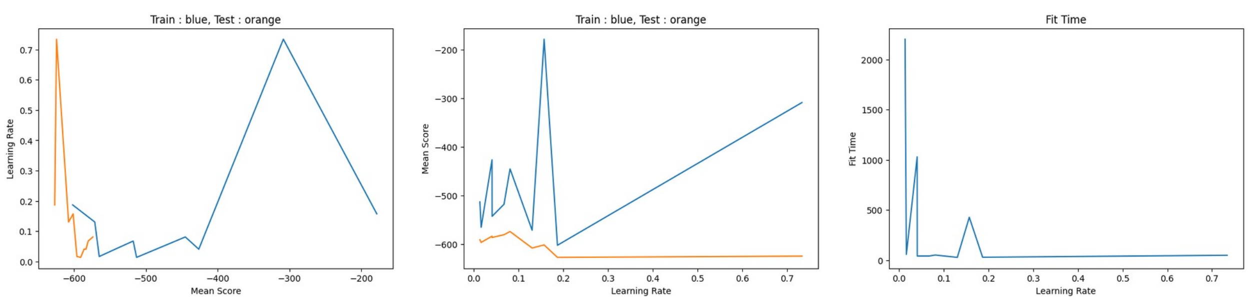
```
30.30214431285858\n        ],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n      }\n    }\n  ]\n}","type":"dataframe","variable_name":"cb_result"}
```


```
cb_train = cb_result.sort_values('mean_train_score',
ascending=True).reset_index(drop=True)
cb_test = cb_result.sort_values('mean_test_score',
ascending=True).reset_index(drop=True)

cb_d = cb_result.sort_values('param_depth',
ascending=True).reset_index(drop=True)
fig, axs = plt.subplots(1,3,figsize=(25,5))

plt.subplot(1, 3, 1)
plt.plot(cb_train['mean_train_score'], cb_train['param_depth'], label
= 'Train')
plt.plot(cb_test['mean_test_score'], cb_test['param_depth'], label =
'Test')
plt.xlabel("Mean Score")
plt.ylabel("Depth")
plt.title("Train : blue, Test : orange")

plt.subplot(1, 3, 2)
plt.plot(cb_d['param_depth'], cb_d['mean_train_score'], label =
'Train')
plt.plot(cb_d['param_depth'], cb_d['mean_test_score'], label = 'Test')
plt.xlabel("Depth")
plt.ylabel("Mean Score")
plt.title("Train : blue, Test : orange")

plt.subplot(1, 3, 3)
plt.plot(cb_d['param_depth'], cb_d['mean_fit_time'], label = 'Train')
plt.xlabel("Depth")
plt.ylabel("Fit Time")
plt.title("Fit Time")

Text(0.5, 1.0, 'Fit Time')
```




```
cb_lr = cb_result.sort_values('param_learning_rate',
ascending=True).reset_index(drop=True)
```

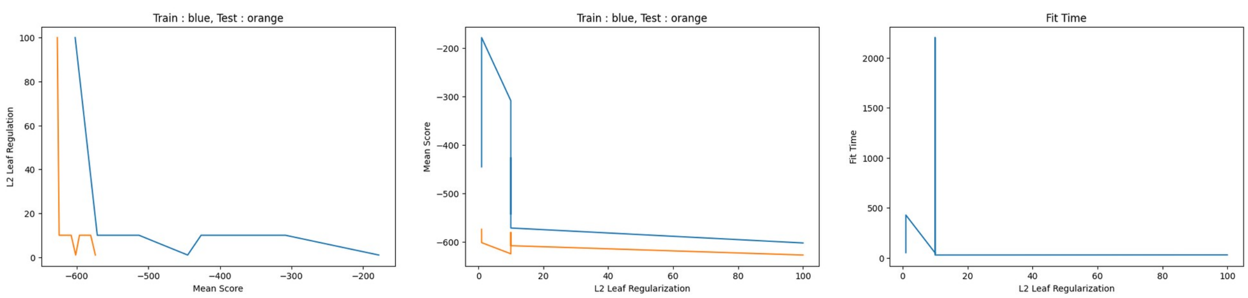
```
fig, axs = plt.subplots(1,3,figsize=(25,5))

plt.subplot(1, 3, 1)
plt.plot(cb_train['mean_train_score'],
cb_train['param_learning_rate'], label = 'Train')
plt.plot(cb_test['mean_test_score'], cb_test['param_learning_rate'],
label = 'Test')
plt.xlabel("Mean Score")
plt.ylabel("Learning Rate")
plt.title("Train : blue, Test : orange")

plt.subplot(1, 3, 2)
plt.plot(cb_lr['param_learning_rate'], cb_lr['mean_train_score'],
label = 'Train')
plt.plot(cb_lr['param_learning_rate'], cb_lr['mean_test_score'], label
= 'Test')
plt.xlabel("Learning Rate")
plt.ylabel("Mean Score")
plt.title("Train : blue, Test : orange")

plt.subplot(1, 3, 3)
plt.plot(cb_lr['param_learning_rate'], cb_lr['mean_fit_time'], label =
'Train')
plt.xlabel("Learning Rate")
plt.ylabel("Fit Time")
plt.title("Fit Time")

Text(0.5, 1.0, 'Fit Time')
```




```
cb_l2 = cb_result.sort_values('param_l2_leaf_reg',
ascending=True).reset_index(drop=True)
fig, axs = plt.subplots(1,3,figsize=(25,5))

plt.subplot(1, 3, 1)
plt.plot(cb_train['mean_train_score'], cb_train['param_l2_leaf_reg'],
label = 'Train')
plt.plot(cb_test['mean_test_score'], cb_test['param_l2_leaf_reg'],
label = 'Test')
plt.xlabel("Mean Score")
plt.ylabel("L2 Leaf Regulation")
```

```
plt.title("Train : blue, Test : orange")

plt.subplot(1, 3, 2)
plt.plot(cb_l2['param_l2_leaf_reg'], cb_l2['mean_train_score'], label
= 'Train')
plt.plot(cb_l2['param_l2_leaf_reg'], cb_l2['mean_test_score'], label =
'Test')
plt.xlabel("L2 Leaf Regularization")
plt.ylabel("Mean Score")
plt.title("Train : blue, Test : orange")

plt.subplot(1, 3, 3)
plt.plot(cb_l2['param_l2_leaf_reg'], cb_l2['mean_fit_time'], label =
'Train')
plt.xlabel("L2 Leaf Regularization")
plt.ylabel("Fit Time")
plt.title("Fit Time")
```

```
Text(0.5, 1.0, 'Fit Time')
```



```
from google.colab import drive
drive.mount('/content/drive')
%cd /content/drive/My Drive/ECE ENGR 219/
```

```
Drive already mounted at /content/drive; to attempt to forcibly
remount, call drive.mount("/content/drive", force_remount=True).
/content/drive/My Drive/ECE ENGR 219
```

```
import json
import matplotlib.pyplot as plt
from collections import defaultdict
import numpy as np
import pandas as pd
import seaborn as sns
import random
from sklearn.model_selection import train_test_split
from sklearn.metrics import precision_recall_fscore_support
from sklearn.metrics import accuracy_score
import tensorflow as tf
import re
import math
import matplotlib.pyplot as plt
import datetime
import pytz
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Embedding,
GlobalAveragePooling1D
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences

#GET STATS DATA
def get_tweets(hashtag):
  tweets_stats = []
  with open('/content/drive/MyDrive/ECE ENGR
219/ECE219_tweet_data/tweets_' + hashtag +'.txt') as file:
    for line in file:
      obj = json.loads(line)
      tweets_stats.append([obj['citation_date'], obj['author']
['followers'], obj['metrics']['citations']['total']])
  return tweets_stats
tweets_stats = {}
hashtag_list = ['#gopatriots', '#gohawks', '#patriots', '#nfl',
'#sb49', '#superbowl']
for hashtag in hashtag_list:
  tweets_stats[hashtag] = get_tweets(hashtag)
```

#Question 9.1

```
for hashtag in hashtag_list:
  #Average number of tweets per hour
```

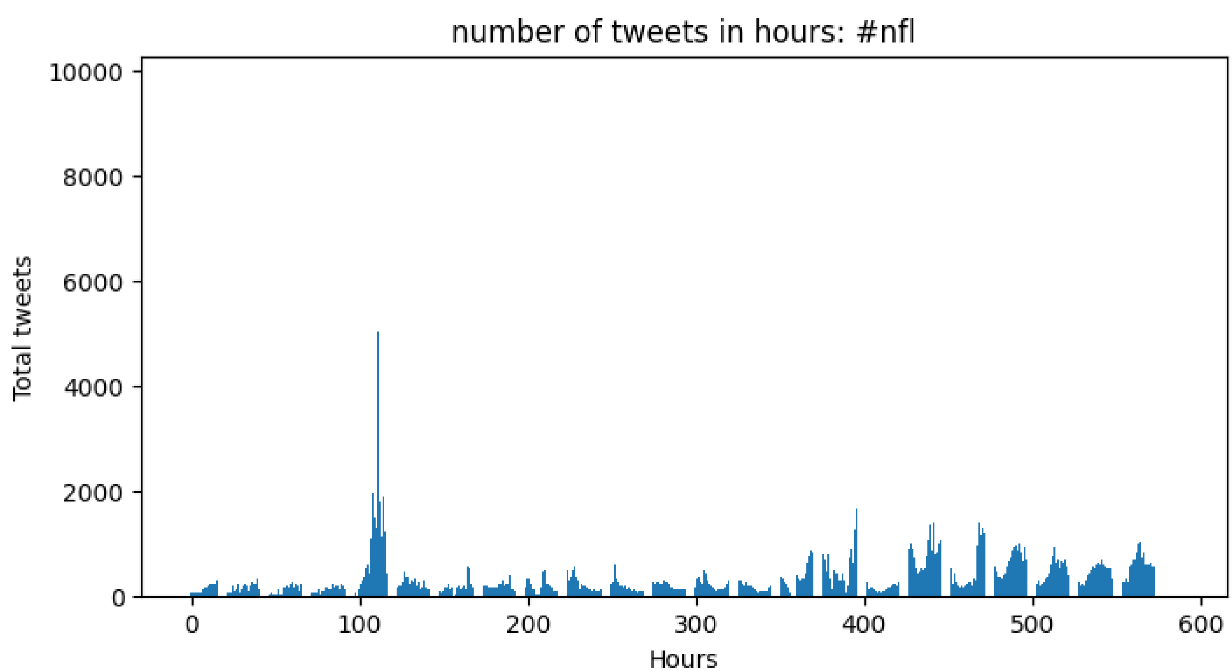
```
  hours = [tweet[1] for tweet in tweets_stats[hashtag]]
  print(hashtag, 'average number of tweets per hour:',
len(tweets_stats[hashtag])/((max(hours)-min(hours))/3600.))
  #Average number of followers of users posting the tweets per tweet
  total_fol = sum([tweet[1] for tweet in tweets_stats[hashtag]])
  print(hashtag, 'average number of followers of users posting the
tweets per tweet:', total_fol/len(tweets_stats[hashtag]))
  #Average number of retweets per tweet
  total_fol = sum([tweet[2] for tweet in tweets_stats[hashtag]])
  print(hashtag, 'average number of retweets per tweet:',
total_fol/len(tweets_stats[hashtag]))
  print()
```

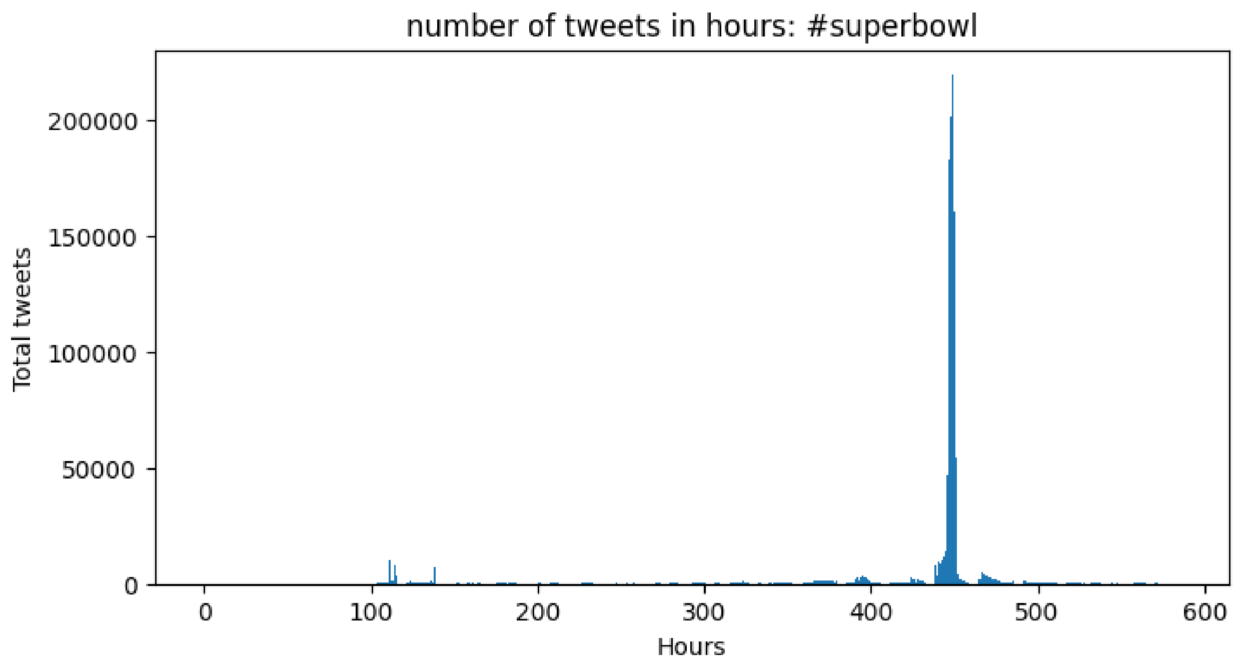
```
#gopatriots average number of tweets per hour: 27.507083985075123
#gopatriots average number of followers of users posting the tweets
per tweet: 1427.2526051635405
#gopatriots average number of retweets per tweet: 1.4081919101697078

#gohawks average number of tweets per hour: 128.9393704190874
#gohawks average number of followers of users posting the tweets per
tweet: 2217.9237355281984
#gohawks average number of retweets per tweet: 2.0132093991319877

#patriots average number of tweets per hour: 163.91593002428715
#patriots average number of followers of users posting the tweets per
tweet: 3280.4635616550277
#patriots average number of retweets per tweet: 1.7852871288476946

#nfl average number of tweets per hour: 87.16640540688259
#nfl average number of followers of users posting the tweets per
tweet: 4662.37544523693
#nfl average number of retweets per tweet: 1.5344602655543254

#sb49 average number of tweets per hour: 40.93186763024069
#sb49 average number of followers of users posting the tweets per
tweet: 10374.160292019487
#sb49 average number of retweets per tweet: 2.52713444111402

#superbowl average number of tweets per hour: 106.833220862131
#superbowl average number of followers of users posting the tweets per
tweet: 8814.96799424623
#superbowl average number of retweets per tweet: 2.3911895819207736
```

#Question 9.2

```
def report_tweets(filename):
    with open(filename, 'r') as file:
        t_max = 0
        t_min = np.inf
```

```
        lines = file.readlines()
        for line in lines:
            json_obj = json.loads(line)
            t_max = max(t_max, json_obj['citation_date'])
            t_min = min(t_min, json_obj['citation_date'])
        all_hours = math.ceil((t_max - t_min) / 3600)
        n_tweets = [0] * all_hours
        for line in lines:
            json_obj = json.loads(line)
            index = math.floor((json_obj['citation_date'] - t_min) /
3600)
            n_tweets[index] += 1
        return n_tweets

hashtags = ['#nfl','#superbowl']

for hashtag in hashtags:
    all_tweets = report_tweets('/content/drive/MyDrive/ECE ENGR
219/ECE219_tweet_data/tweets_'+hashtag+'.txt')
    plt.figure(figsize=(8,4))
    plt.bar(range(len(all_tweets)),all_tweets)
    plt.xlabel('Hours')
    plt.ylabel('Total tweets')
    plt.title('number of tweets in hours: '+hashtag)
```





#Question 10

Describe Task: I chose to create 3 different models for 3 different tasks. The first involved predicting the one out of the 10 most used hashtags for a given tweet. At first I wanted to predict multiple of the top 10 hashtags, but after doing some data exploration I realized that a majority of the tweets only contained one of the top 10 tweets. So then I decided to do a multi class classification task using fastText as the model to make predictions. The second task I worked on was a binary classification task that used the data solely from the patriots and hawks tweets dataset to predict which team the user tweeting it supported. Again, here I used fastText to do the prediction. In the last task, I used a tokenizer to get the embeddings of the tweets and passed it through a neural network to predict the number of retweets a user could have gotten, given a tweet.

Describe Feature Engineering Process:

- Subsampled with goptatriots tweet text because that dataset had the least amount of data out of all txt files
  - Graphed frequency of all the tweet data
- Created a heat map with correlations between all the numerical features
  - When training the neural network, chose to use the feature most correlated with number of retweets(momentum with a correlation of 0.55) which ended up reducing the rise by a lot
  - Didnt put other numerical features which had less than 0.5 for correlation
- When using fast text, I would have added more textual features, but fastText was pretty good at predicting without any additional information other than tweet
- I also removed hashtags for the tasks necessary because didn’t want model to ‘cheat’

```
%cd ECE219_tweet_data/fastText-0.9.2
!make
!pip install .
```

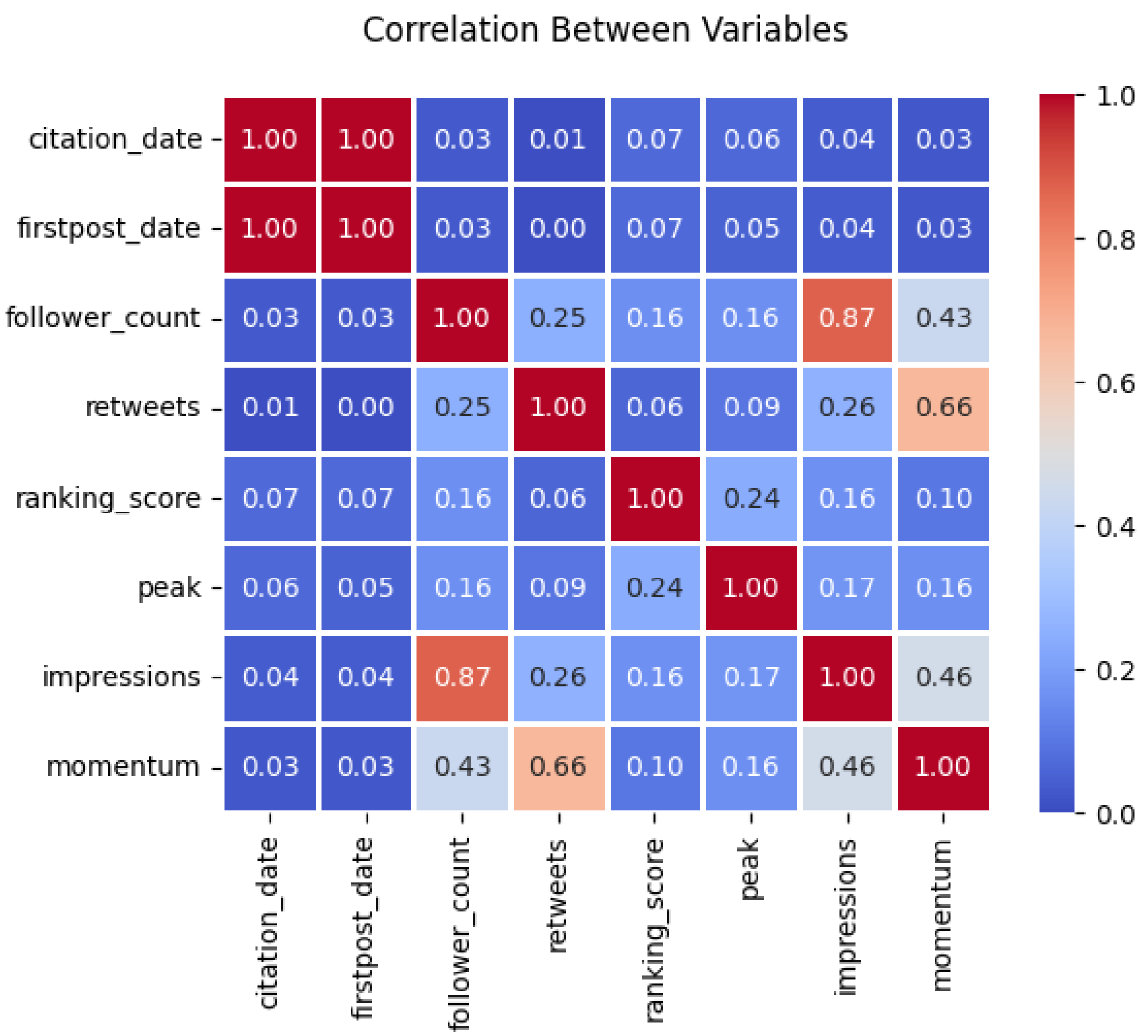
```
/content/drive/MyDrive/ECE ENGR 219/ECE219_tweet_data/fastText-0.9.2
make: Nothing to be done for 'opt'.
Processing /content/drive/MyDrive/ECE ENGR
219/ECE219_tweet_data/fastText-0.9.2
  Preparing metadata (setup.py) ... ent already satisfied:
pybind11>=2.2 in /usr/local/lib/python3.10/dist-packages (from
fasttext==0.9.2) (2.11.1)
Requirement already satisfied: setuptools>=0.7.0 in
/usr/local/lib/python3.10/dist-packages (from fasttext==0.9.2)
(67.7.2)
Requirement already satisfied: numpy in
/usr/local/lib/python3.10/dist-packages (from fasttext==0.9.2)
(1.25.2)
Building wheels for collected packages: fasttext
  Building wheel for fasttext (setup.py) ... e=fasttext-0.9.2-cp310-
cp310-linux_x86_64.whl size=4199782
sha256=d5adc5ea6ad79489c8dae992fd6baca8f55802485b36b1c1ac36977acec0521
9
  Stored in directory:
/root/.cache/pip/wheels/1f/e3/48/c142b860724c501aa9d814ec942e58aea80cd
6b352839f1d05
Successfully built fasttext
Installing collected packages: fasttext
  Attempting uninstall: fasttext
    Found existing installation: fasttext 0.9.2
    Uninstalling fasttext-0.9.2:
      Successfully uninstalled fasttext-0.9.2
Successfully installed fasttext-0.9.2
```

```
#GET ALL DATA
hashtag_tracker = defaultdict(int)
tweets_data = []
hashtag_list = ['#gopatriots', '#gohawks', '#patriots', '#nfl',
'#sb49', '#superbowl']
for hashtag in hashtag_list:
  with open('/content/drive/MyDrive/ECE ENGR
219/ECE219_tweet_data/tweets_' + hashtag +'.txt') as file:
    for line in file:
      obj = json.loads(line)
      stats = {}
      stats['hashtag'] = hashtag
      stats['tweet'] = obj['tweet']['text']
      stats['display_name'] = obj['original_author']['name']
      stats['handle'] = obj['original_author']['nick']
      stats['citation_date'] = obj['citation_date']
      stats['firstpost_date'] = obj['firstpost_date']
```

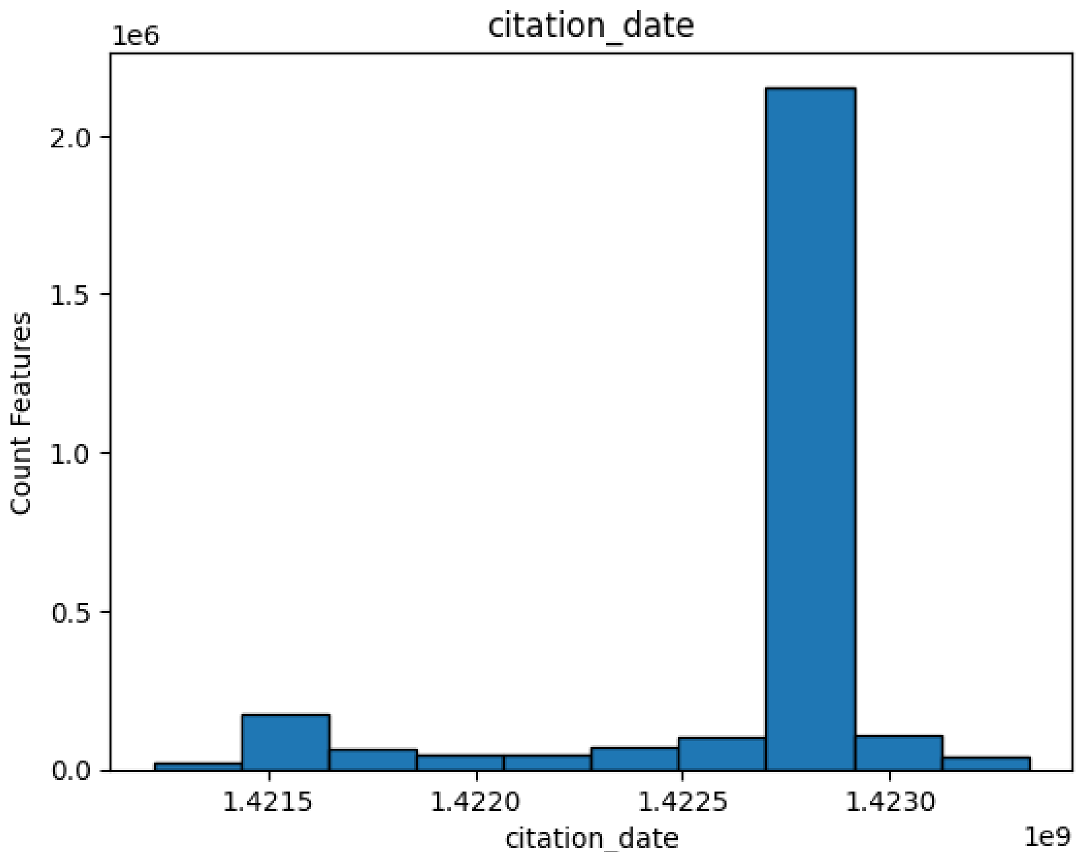
```
      stats['follower_count'] = obj['author']['followers']
      stats['retweets'] = obj['metrics']['citations']['total']
      stats['ranking_score'] = obj['metrics']['ranking_score']
      stats['peak'] = obj['metrics']['peak']
      stats['impressions'] = obj['metrics']['impressions']
      stats['momentum'] = obj['metrics']['momentum']
      list_hashtags = []
      for h in obj['tweet']['entities']['hashtags']:
        list_hashtags.append(h['text'])
        hashtag_tracker[h['text']] += 1
      stats['list_hashtag'] = list_hashtags
      tweets_data.append(stats)

df_tweets = pd.DataFrame(tweets_data)
len_pat = len(df_tweets[df_tweets['hashtag'] == '#gopatriots'])
df_hawks = df_tweets[df_tweets['hashtag'] == '#gohawks'][:(len_pat +
1)]
df_nfl = df_tweets[df_tweets['hashtag'] == '#gonfl'][:(len_pat + 1)]
df_pat = df_tweets[df_tweets['hashtag'] == '#patriots'][:(len_pat +
1)]
df_sb = df_tweets[df_tweets['hashtag'] == '#sb49'][:(len_pat + 1)]
df_superbowl = df_tweets[df_tweets['hashtag'] == '#superbowl'][:
(len_pat + 1)]
df_short = df_tweets[(df_tweets['hashtag'] ==
'#gopatriots')].append([df_hawks, df_nfl, df_pat, df_sb,
df_superbowl])

<ipython-input-5-fbcb0fddebef>:8: FutureWarning: The frame.append
method is deprecated and will be removed from pandas in a future
version. Use pandas.concat instead.
  df_short = df_tweets[(df_tweets['hashtag'] ==
'#gopatriots')].append([df_hawks, df_nfl, df_pat, df_sb,
df_superbowl])
```

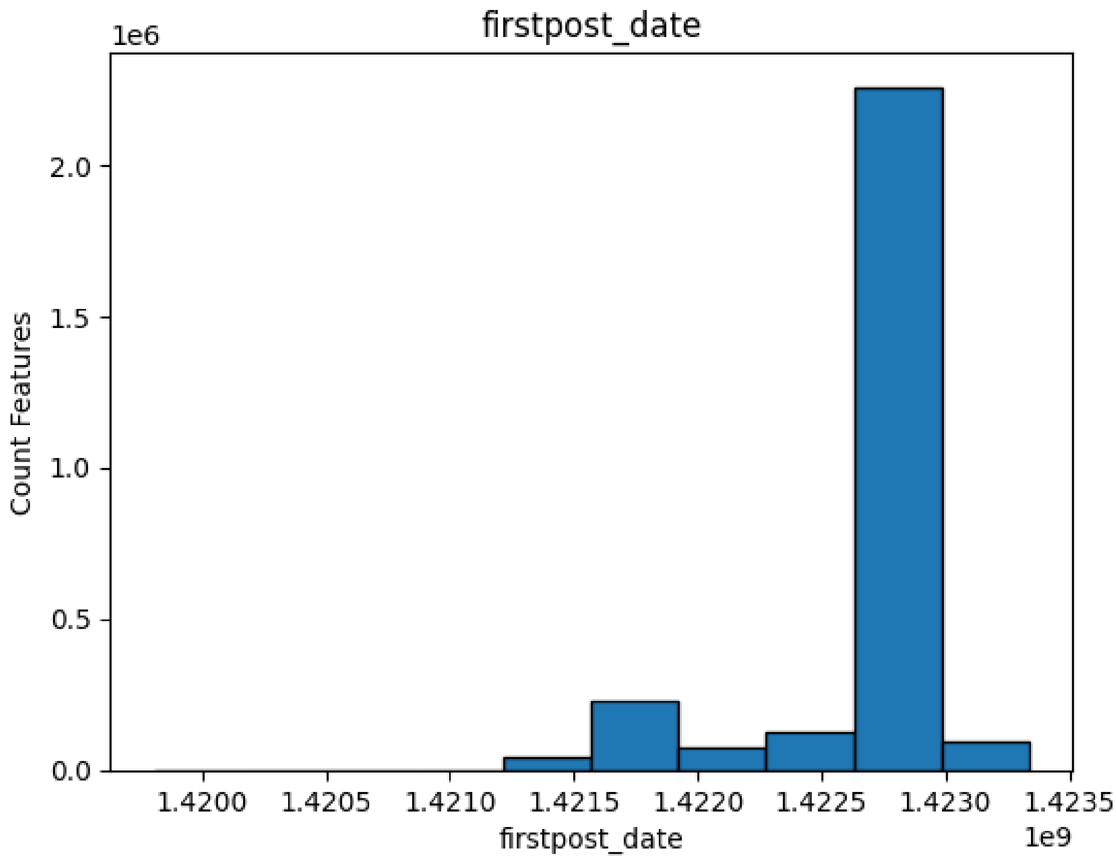
Explore Data

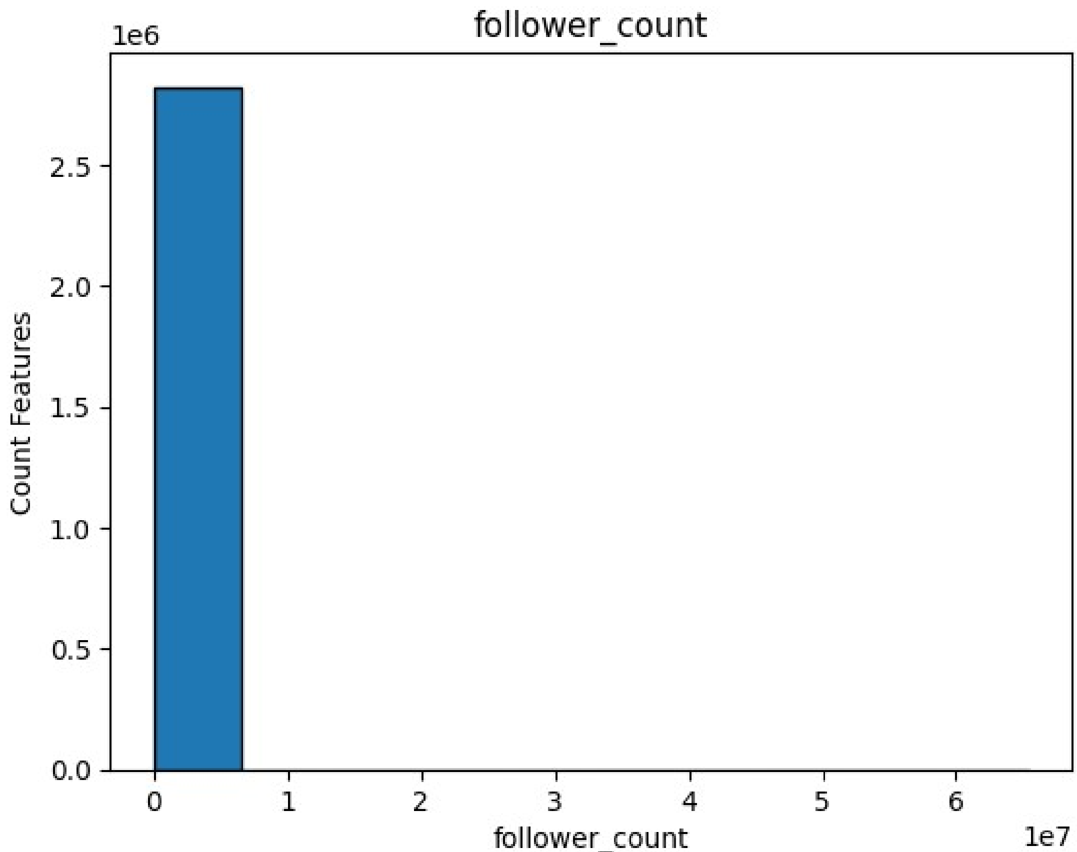
```
corrM = df_short.corr()
corr = corrM.abs()
sns.heatmap(corr, vmin=0, vmax=1, cmap = 'coolwarm', annot=True,
fmt='.2f', linewidths=2)
plt.title("Correlation Between Variables", pad=20)

<ipython-input-19-b15afff02c4b>:1: FutureWarning: The default value of
numeric_only in DataFrame.corr is deprecated. In a future version, it
will default to False. Select only valid columns or specify the value
of numeric_only to silence this warning.
  corrM = df_short.corr()

Text(0.5, 1.0, 'Correlation Between Variables')
```

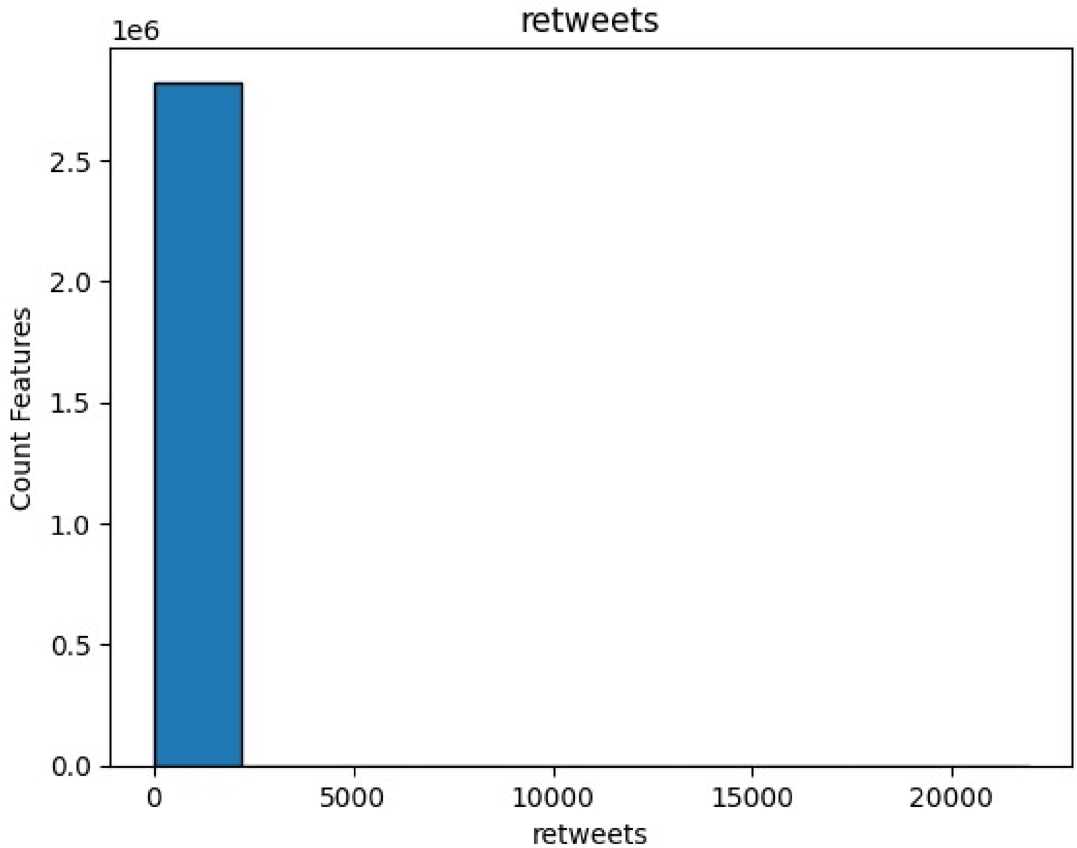


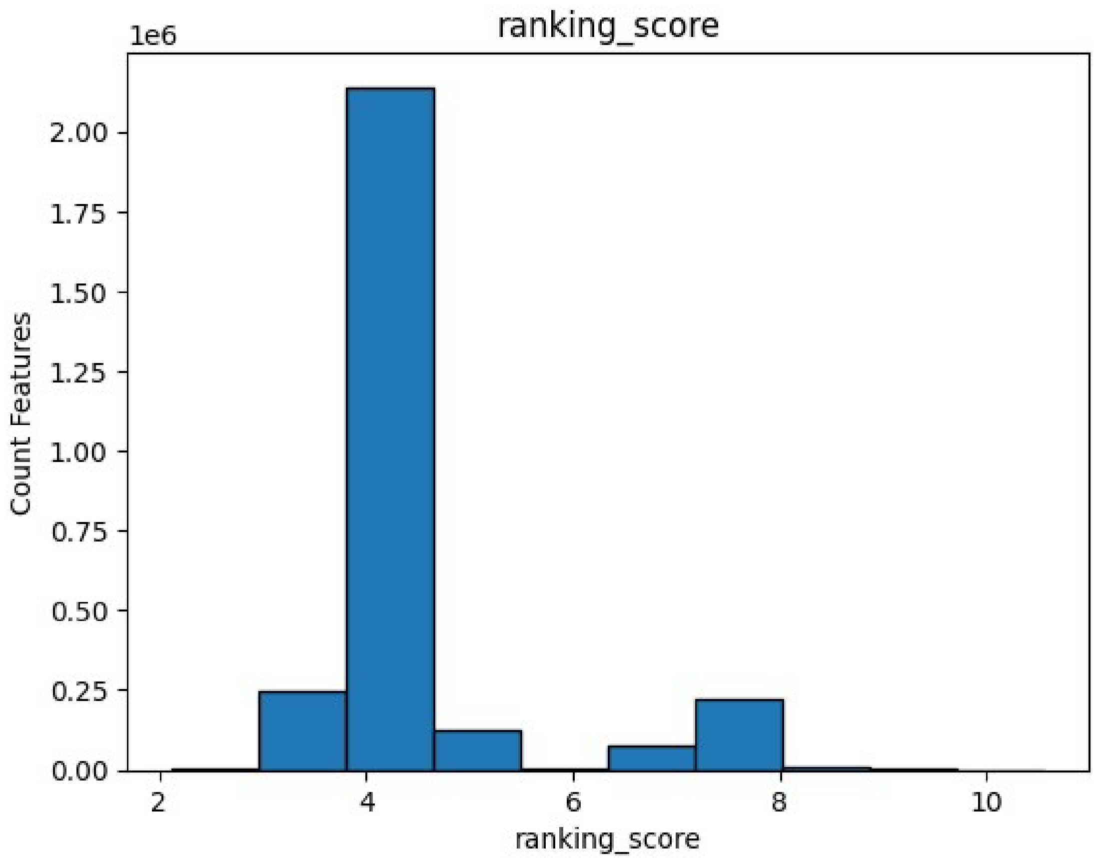

```
num_features = ['citation_date', 'firstpost_date', 'follower_count',
'retweets', 'ranking_score', 'peak', 'impressions', 'momentum']

for i in np.arange(len(num_features)) :
  plt.figure()
  plt.hist(df_tweets[num_features[i]], edgecolor = "black")
  plt.xlabel(f"{num_features[i]}");
  plt.ylabel("Count Features");
  plt.title(f"{num_features[i]}")
```

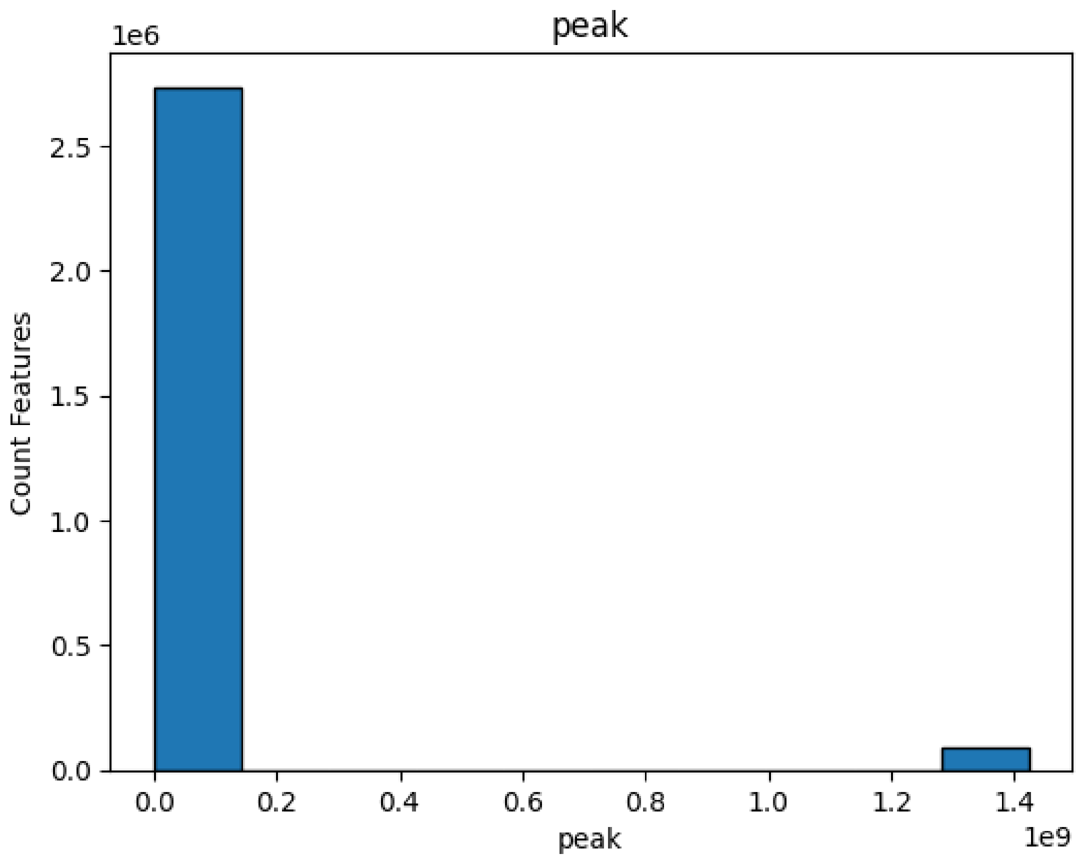

citation_date
1e6
2.0
1.5
1.0
0.5
0.0
Count Features
1.4215
1.4220
1.4225
1.4230
citation_date
1e9

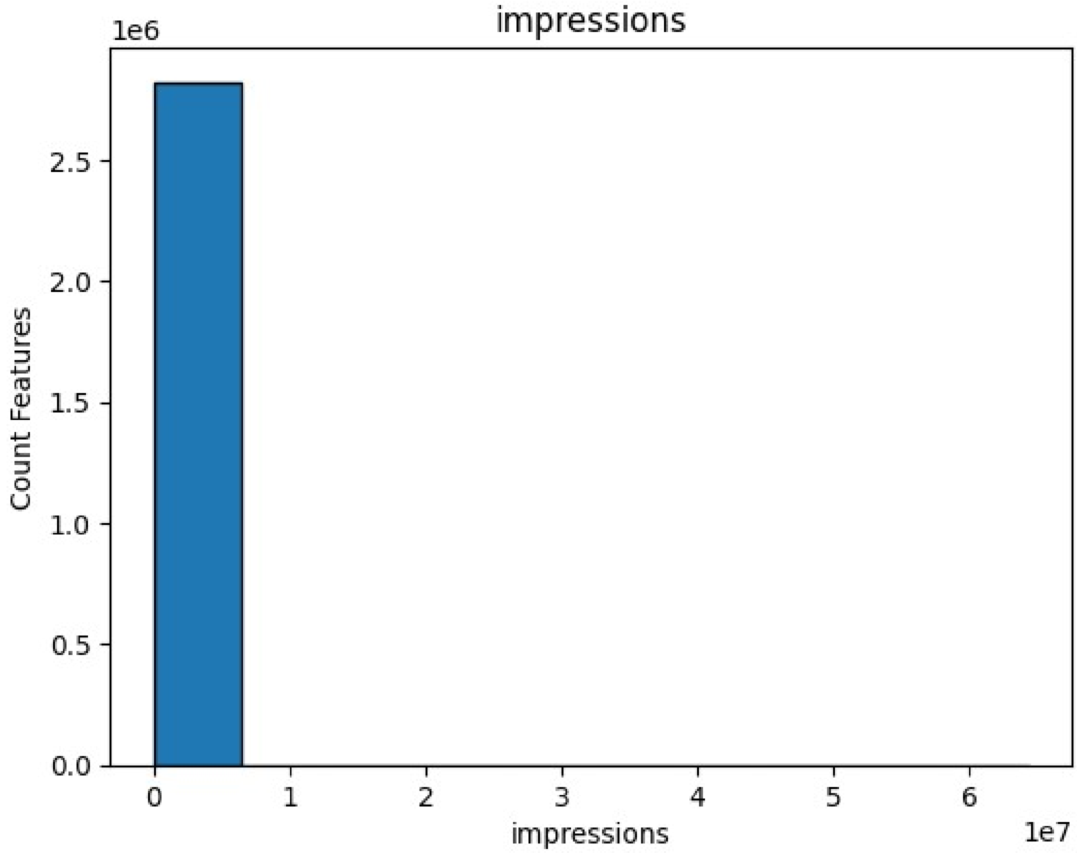

firstpost_date
1e6
2.0
1.5
1.0
0.5
0.0
Count Features
1.4200
1.4205
1.4210
1.4215
1.4220
1.4225
1.4230
1.4235
firstpost_date
1e9

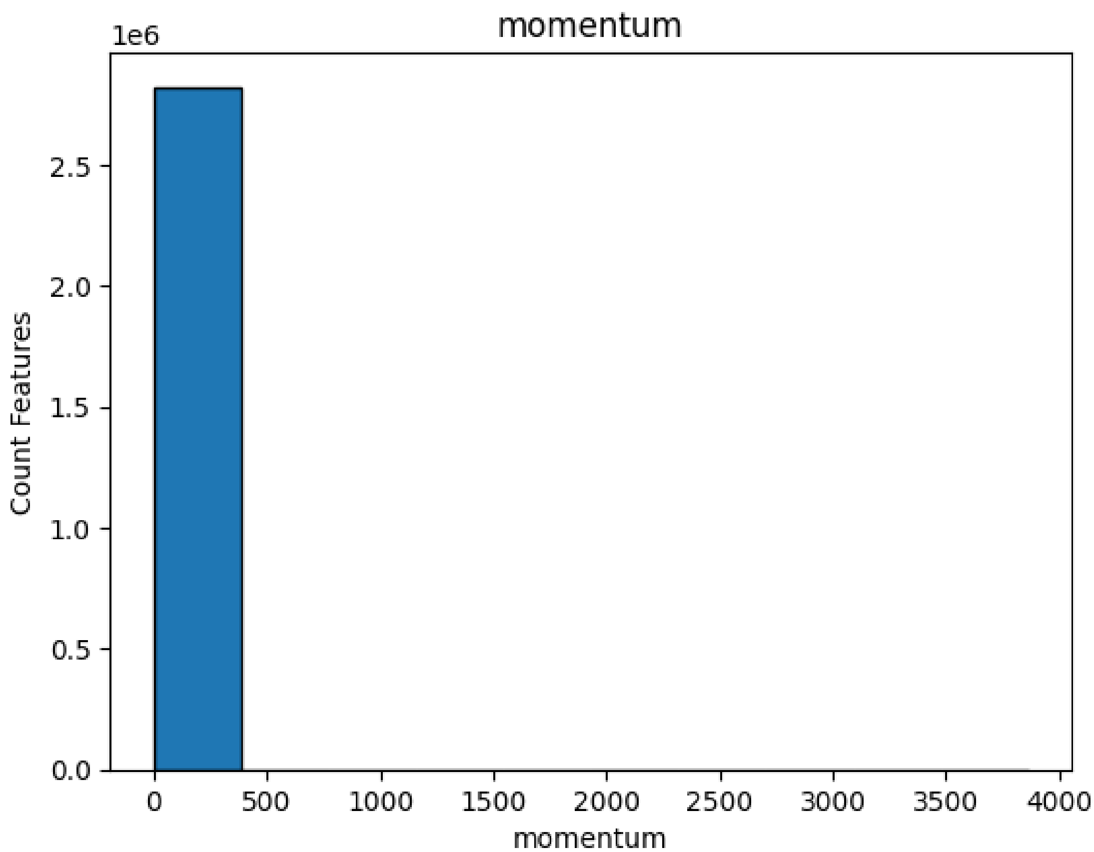

follower_count
1e6
2.5
2.0
1.5
1.0
0.5
0.0
Count Features
0
1
2
3
4
5
6
follower_count
1e7


retweets
1e6
2.5
2.0
1.5
1.0
0.5
0.0
Count Features
0
5000
10000
15000
20000
retweets


ranking_score
1e6
2.00
1.75
1.50
1.25
1.00
0.75
0.50
0.25
0.00
Count Features
2
4
6
8
10
ranking_score

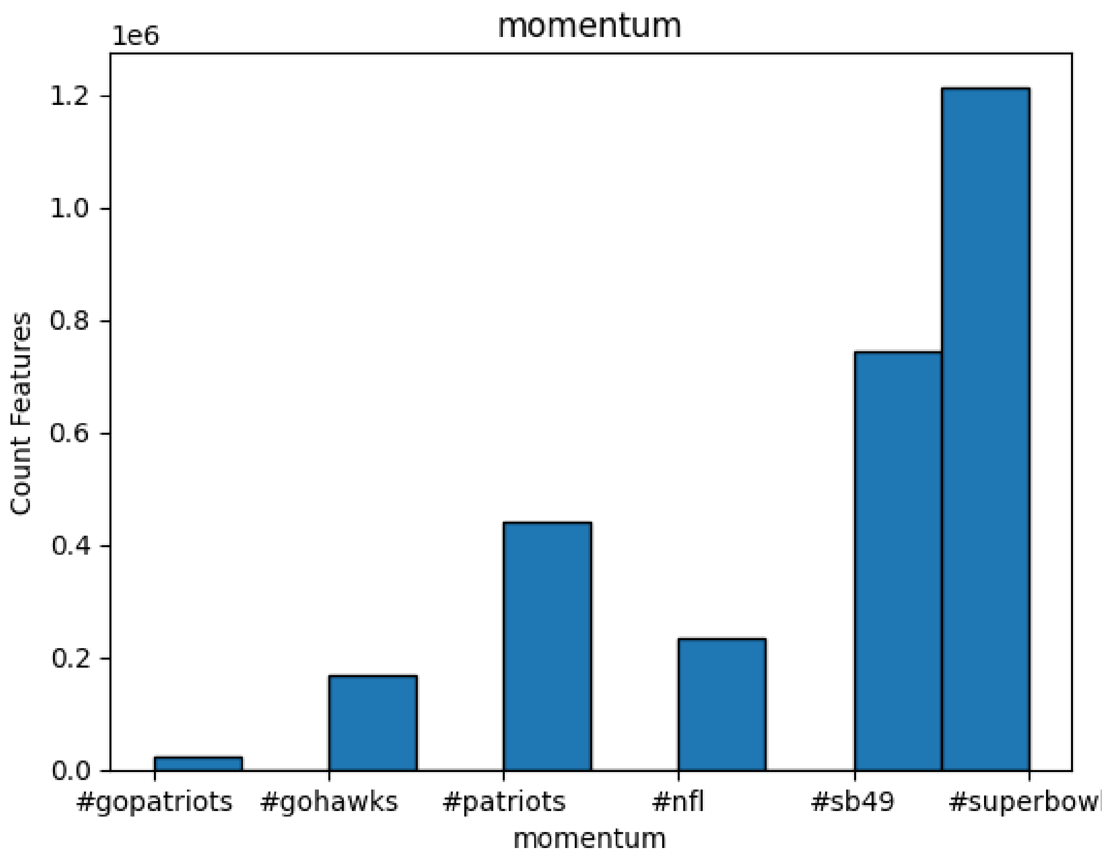

peak
1e6
2.5
2.0
1.5
1.0
0.5
0.0
Count Features
0.0
0.2
0.4
0.6
0.8
1.0
1.2
1.4
peak
1e9


impressions
1e6
2.5
2.0
1.5
1.0
0.5
0.0
Count Features
0
1
2
3
4
5
6
impressions
1e7




```
plt.figure()
plt.hist(df_tweets['hashtag'], edgecolor = "black")
plt.xlabel(f"{num_features[i]}");
plt.ylabel("Count Features");
plt.title(f"{num_features[i]}")
```

```
Text(0.5, 1.0, 'momentum')
```




```
sorted(hashtag_tracker, key=hashtag_tracker.get, reverse=True)[:10]
```

```
['SB49',
 'SuperBowl',
 'SuperBowlXLIX',
 'PatriotsWIN',
 'Patriots',
 'NFL',
 'SeahawksWIN',
 'GoHawks',
 'Seahawks',
 'superbowl']
```

Predict Hashtag

```
hashtag_classes = sorted(hashtag_tracker, key=hashtag_tracker.get,
reverse=True)[:10]

df_hashtag = pd.DataFrame(columns=['tweet', 'top_hashtag'])

count_hashtags = {}
for h in hashtag_classes:
  count_hashtags[h] = 10000
```

```
i = 0
for index, row in df_short.iterrows():
  check = False
  check2 = True
  for h in row['list_hashtag']:
    tmp = []
    if h in count_hashtags and count_hashtags[h] > 0:
      count_hashtags[h] -= 1
      tmp.append(h)
      check = True
  if len(tmp) > 0:
    df_hashtag.loc[i] = [row['tweet'], tmp[0]]
    i += 1
  for key in count_hashtags:
    if count_hashtags[key] > 0:
      check2 = False
  if check2:
    break
```

Step to removes the hashtags

```
df_hashtag['tweet'] = df_hashtag['tweet'].apply(lambda x: re.sub("@[A-
Za-z0-9_]+","", x))

X_train,X_test,y_train,y_test =
train_test_split(df_hashtag['tweet'],df_hashtag['top_hashtag'],test_si
ze=0.2, shuffle=True)

with open('/content/drive/MyDrive/ECE ENGR
219/ECE219_tweet_data/hashtag_train_data.txt', 'w') as writefile:
  for i in range(len(X_train)):
    writefile.write("__label__"+y_train.iloc[i]+"
"+X_train.iloc[i].replace("\n"," ")+"\n")

import fasttext
model_hashtag =
fasttext.train_supervised(input="/content/drive/MyDrive/ECE ENGR
219/ECE219_tweet_data/hashtag_train_data.txt", lr=0.5, epoch=25,
wordNgrams=2, bucket=200000, dim=50, loss='ova')

preds = []
for i in range(len(X_test)):
  preds.append(model_hashtag.predict(X_test.iloc[i].replace("\n"," "))
[0][0][9:])
```

Precision, Recall, Fscores, Support(in theat order) for each hashtag in the lists below

```
precision_recall_fscore_support(y_test, preds)
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/metrics/
_classification.py:1344: UndefinedMetricWarning: Recall and F-score
are ill-defined and being set to 0.0 in labels with no true samples.
Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
```

```
(array([0.9742268 , 0.94295302, 0.95721925, 1.        , 0.95813953,
        0.92631579, 0.        , 0.97364341, 0.98029557, 0.98165138]),
 array([0.98878924, 0.93979933, 0.97707424, 0.625     , 0.96827262,
        0.81230769, 0.        , 0.96615385, 0.97358121, 0.97272727]),
 array([0.98145401, 0.94137353, 0.96704484, 0.76923077, 0.96317943,
        0.86557377, 0.        , 0.96988417, 0.97692685, 0.97716895]),
 array([1338,  299,  916,   16,  851,  325,    0,  650, 1022,  110]))
```

Accuracy Score: 0.9638140039804596

```
accuracy_score(y_test, preds)
```

```
0.9638140039804596
```

Predict Team

```
len_pat = len(df_tweets[df_tweets['hashtag'] == '#gopatriots'])
df_hawks = df_tweets[df_tweets['hashtag'] == '#gohawks'][:(len_pat +
1)]
df_teams = df_tweets[(df_tweets['hashtag'] ==
'#gopatriots')].append(df_hawks)
```

```
<ipython-input-104-29ceecf5815a>:3: FutureWarning: The frame.append
method is deprecated and will be removed from pandas in a future
version. Use pandas.concat instead.
  df_teams = df_tweets[(df_tweets['hashtag'] ==
'#gopatriots')].append(df_hawks)
```

```
X_train,X_test,y_train,y_test =
train_test_split(df_teams['tweet'],df_teams['hashtag'],test_size=0.2,
shuffle=True)
```

```
with open('/content/drive/MyDrive/ECE ENGR
219/ECE219_tweet_data/team_train_data.txt', 'w') as writefile:
  for i in range(len(X_train)):
    writefile.write("__label__"+y_train.iloc[i]+"
"+X_train.iloc[i].replace("\n"," ")+"\n")
```

```
import fasttext
model = fasttext.train_supervised(input="/content/drive/MyDrive/ECE
ENGR 219/ECE219_tweet_data/team_train_data.txt", lr=1.0, epoch=25,
wordNgrams=2, bucket=200000, dim=50, loss='hs')
```

```
preds = []
for i in range(len(X_test)):
  preds.append(model.predict(X_test.iloc[i].replace("\n"," "))[0][0]
[9:])
```

Precision, Recall, Fscores, Support(in theat order) for each team in the lists below

```
precision_recall_fscore_support(y_test, preds)
```

```
(array([0.99611231, 0.99518325]),
 array([0.99503776, 0.99622642]),
 array([0.99557474, 0.99570456]),
 array([4635, 4770]))
```

Accuracy Score: 0.9956406166932483

```
accuracy_score(y_test, preds)
```

```
0.9956406166932483
```

Predict Retweets

```
X_train,X_test,y_train,y_test = train_test_split(df_short[['tweet',
'momentum']], df_short['retweets'],test_size=0.2, shuffle=True)
tweet_data = X_train['tweet']
targets = y_train

max_words = 1000
tokenizer = Tokenizer(num_words=max_words)
tokenizer.fit_on_texts(tweet_data)
sequences = tokenizer.texts_to_sequences(tweet_data)

max_sequence_length = max(len(seq) for seq in sequences)
padded_sequences = pad_sequences(sequences,
maxlen=max_sequence_length)

tmp_padded_sequences = np.insert(padded_sequences, 33,
X_train['momentum'], axis=1)

from tensorflow.keras.layers import Dense, Dropout
model = Sequential([
    Dense(64, activation='relu', input_shape=(34,)),
    Dropout(0.2),
    Dense(32, activation='relu'),
    Dropout(0.2),
    Dense(1)
])
model.compile(optimizer='adam', loss='mean_squared_error')
history = model.fit(tmp_padded_sequences, y_train, epochs=10,
batch_size=64, validation_split=0.1)
```

```
Epoch 1/10
1323/1323 [==============================] - 5s 3ms/step - loss:
2870.9443 - val_loss: 259.7613
Epoch 2/10
1323/1323 [==============================] - 4s 3ms/step - loss:
2593.2625 - val_loss: 259.1746
Epoch 3/10
1323/1323 [==============================] - 4s 3ms/step - loss:
2589.3987 - val_loss: 259.1754
Epoch 4/10
1323/1323 [==============================] - 3s 2ms/step - loss:
2580.9280 - val_loss: 259.4706
Epoch 5/10
1323/1323 [==============================] - 3s 2ms/step - loss:
2557.8718 - val_loss: 259.1740
Epoch 6/10
1323/1323 [==============================] - 4s 3ms/step - loss:
2555.9016 - val_loss: 258.1605
Epoch 7/10
1323/1323 [==============================] - 4s 3ms/step - loss:
2416.3438 - val_loss: 256.3393
Epoch 8/10
1323/1323 [==============================] - 3s 2ms/step - loss:
2202.2822 - val_loss: 256.9002
Epoch 9/10
1323/1323 [==============================] - 3s 2ms/step - loss:
2024.6555 - val_loss: 255.0045
Epoch 10/10
1323/1323 [==============================] - 4s 3ms/step - loss:
1341.2083 - val_loss: 251.6341
```

MSE training with momentum feature: 172.7850799560547

```
sequences_test = tokenizer.texts_to_sequences(X_test['tweet'])
padded_sequences_test = pad_sequences(sequences_test,
maxlen=max_sequence_length)
tmp_padded_sequences_test = np.insert(padded_sequences_test,
max_sequence_length, X_test['momentum'], axis=1)
# Evaluate the model
loss = model.evaluate(tmp_padded_sequences_test, y_test)
print("Test Loss:", loss)

735/735 [==============================] - 3s 3ms/step - loss:
172.7851
Test Loss: 172.7850799560547
```

MSE prior to training with momentum feature: 271.0411682128906

```
tokenizer.fit_on_texts(X_test['tweet'])
sequences = tokenizer.texts_to_sequences(X_test['tweet'])
max_sequence_length = max(len(seq) for seq in sequences)
padded_sequences = pad_sequences(sequences,
maxlen=max_sequence_length)
# Evaluate the model
loss = model.evaluate(padded_sequences, y_test)
print("Test Loss:", loss)
```

```
735/735 [==============================] - 4s 5ms/step - loss:
271.0412
Test Loss: 271.0411682128906
```