Name : Vignesh Nagarajan (UID: 606185377)

In [4]:

```
from google.colab import drive
drive.mount('/content/drive')
```

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

In [5]:

```
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
from sklearn.metrics import roc_curve, auc, mean_squared_error
from sklearn.preprocessing import LabelEncoder
import seaborn as sns
from sklearn.preprocessing import StandardScaler, PolynomialFeatures
from sklearn.feature_selection import SelectKBest, mutual_info_regression, f_regression
from sklearn.pipeline import Pipeline, make_pipeline
from sklearn.model_selection import cross_validate, GridSearchCV
from sklearn.linear_model import LinearRegression, Ridge, Lasso
from sklearn.neural_network import MLPRegressor
from statsmodels.regression.linear_model import OLS
from sklearn.ensemble import RandomForestRegressor
from sklearn.tree import export_graphviz
from sklearn.model_selection import train_test_split
```

In [6]:

```
%cd '/content/drive/MyDrive/219/Project5'
```

```
/content/drive/MyDrive/219/Project5
```

In [7]:

```
data = pd.read_csv("Diamonds_ece219.csv")
```

In [8]:

```
data['girdle_min_enc'] = data['girdle_min'].replace({'XTN': 1, 'VTN':2, 'TN':3, 'STN':4, 'M':5, 'STK':6, 'TK':7,
'VTK':8,'XTK' :9,'unknown':-10})
data['girdle_max_enc'] = data['girdle_max'].replace({'XTN': 1, 'VTN':2, 'TN':3, 'STN':4, 'M':5, 'STK':6, 'TK':7,
'VTK':8,'XTK' :9,'unknown':-10})
```

Data inspection

In [9]:

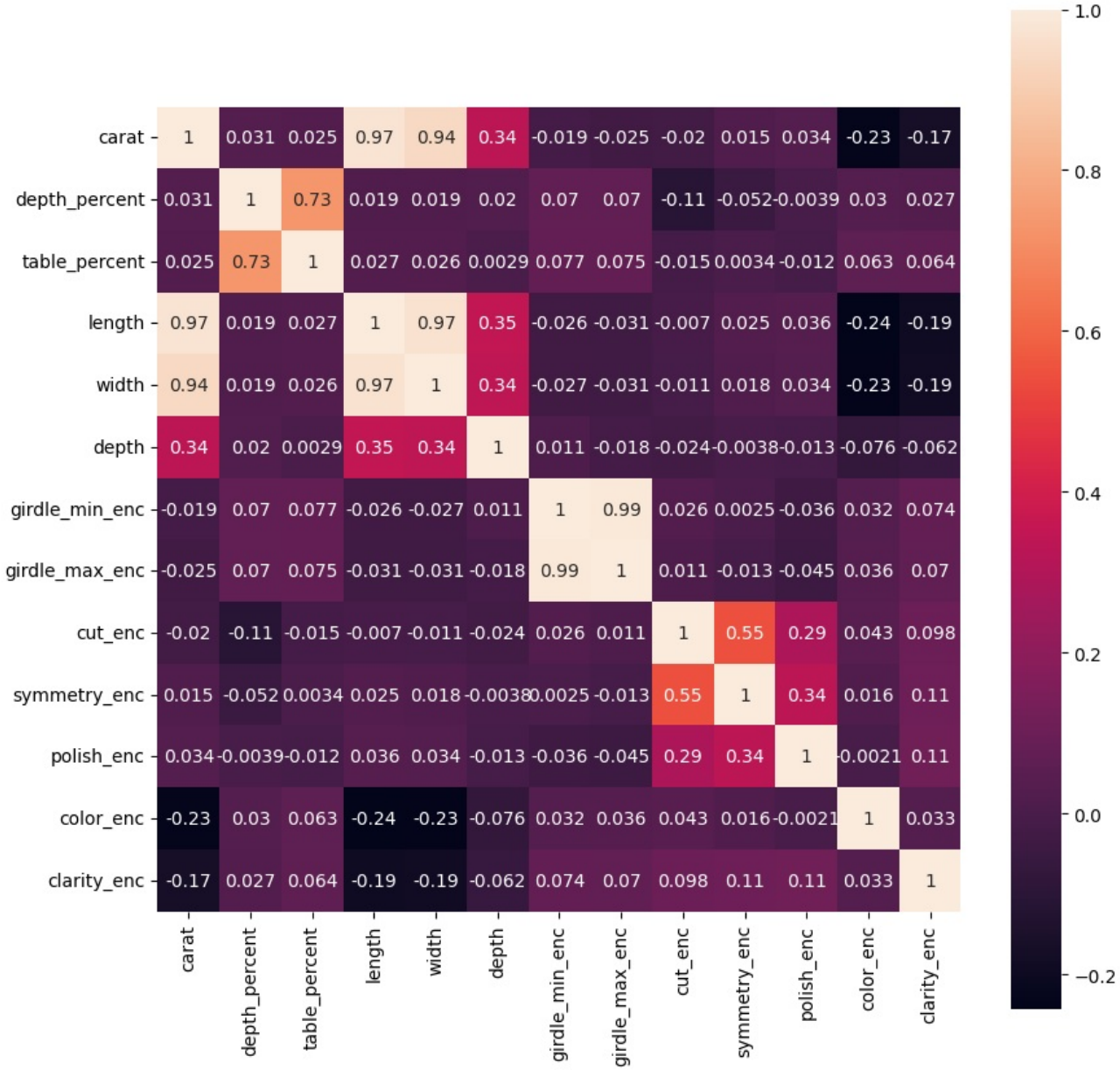
```
# Categorical to numerical

cat2num = {'Very Good': 1, 'Excellent': 2}
color_dict = { 'M':1,
               'L':2,
              'K':3,
               'J' : 4,
           'I' : 5,
           'H': 6,
           'G': 7,
           'F': 8,
           'E': 9,
           'D': 10}

data['cut_enc'] = data['cut'].map(cat2num)
data['symmetry_enc'] = data['symmetry'].map(cat2num)
data['polish_enc'] = data['polish'].map(cat2num)
data['color_enc'] = data['color'].map(color_dict)
```

In [10]:

```
clarity_ranking = {
    'I3': 1,
    'I2': 2,
    'I1': 3,
    'SI2': 4,
    'SI1': 5,
    'VS2': 6,
    'VS1': 7,
    'VVS2': 8,
    'VVS1': 9,
    'IF': 10
}

data['clarity_enc'] = data['clarity'].map(clarity_ranking)
```

In [11]:

```
data = data.drop(columns=['Unnamed: 0'])
dataenc = data.copy(deep = True)
dataenc = dataenc.drop(columns= ['clarity','cut','symmetry','color','polish','girdle_min','girdle_max'])
data =data.drop(columns=['clarity_enc','cut_enc','symmetry_enc','color_enc','polish_enc','girdle_min_enc','girdle
_max_enc'])
corr = dataenc.loc[:, dataenc.columns != 'price'].corr()
plt.figure(figsize=(10, 10))
sns.heatmap(data=corr, square=True , annot=True, cbar=True)
plt.show()
```



In [43]:

```
plt.figure(figsize = (5,4))
h_map = sns.heatmap(dataenc.corr()[["price"]].sort_values(by = "price",ascending=False),vmin=-1, vmax=1, annot=Tr
ue,cmap ='magma')
h_map.set_title("Features Correlating with Price")
plt.show()
```

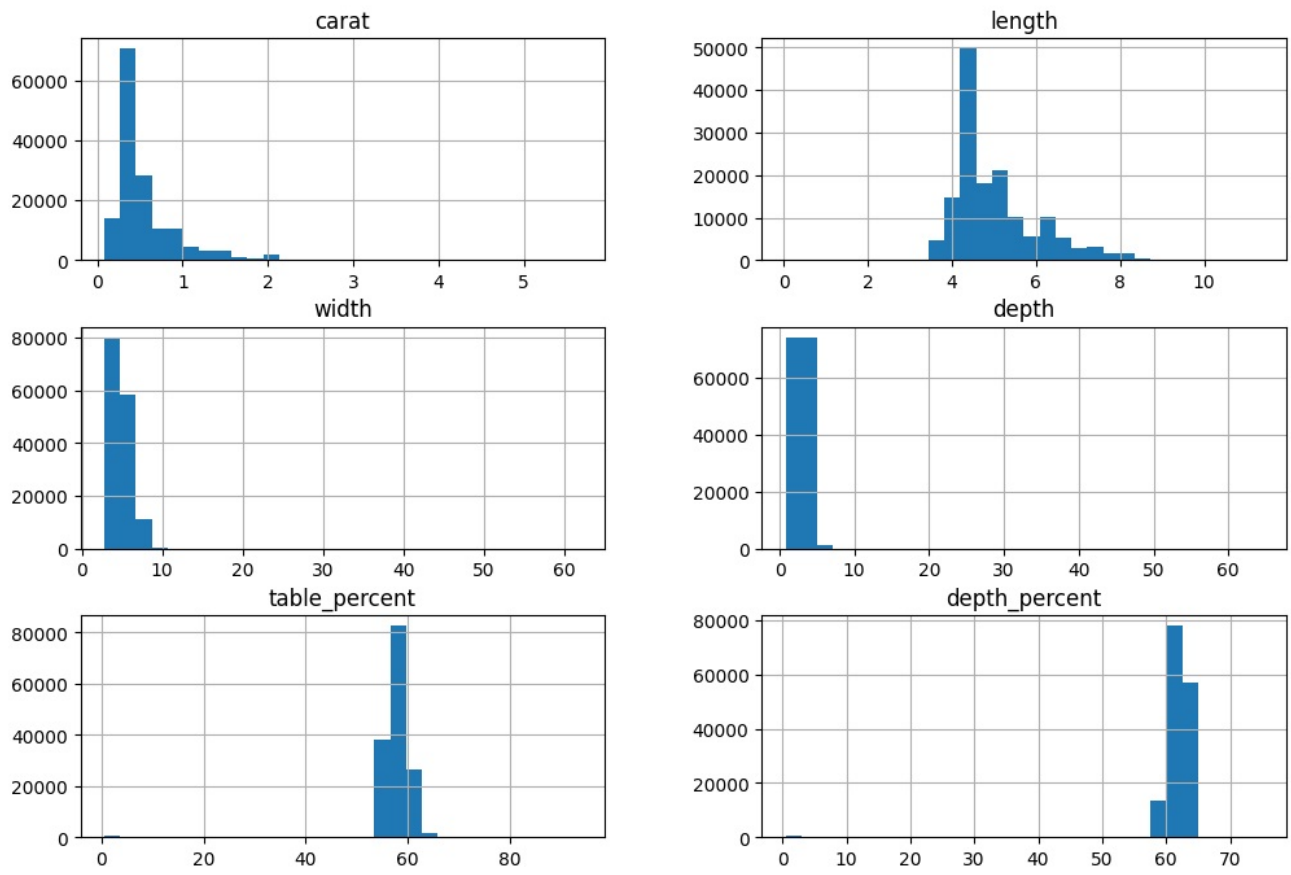
Answer 1.1

As noted above in the first correlation map, length and width have a high correlation and have the same underlying domain knowledge, and hence are highly correlated. Same can be said about depth_percent and table_percent. We also note that length, width and carat have high correlations amongst themselves which makes sense because we know that carat weight is related to the length, width, and depth of a diamond.

From the second correlation map of features with target variable, we observe that carat, length and width have high correlations to the target variable "Price". This makes sense qualitatively since higher these mentioned feature values are, the price is also expected to be higher

In [46]:

```
from scipy.stats import boxcox

numerical_features = ['carat', 'length', 'width', 'depth', 'table_percent', 'depth_percent' ]

dataenc[numerical_features].hist(bins=30, figsize=(12, 8))
plt.suptitle('Histograms of Numerical Features', y=1.02)
plt.show()

skewness = dataenc[numerical_features].apply(lambda x: x.skew())
print("Skewness of Features:")
print(skewness)
```




```
Skewness of Features:
carat             2.331773
length            1.283604
width             4.115348
depth            27.493299
table_percent   -11.046563
depth_percent   -13.559608
dtype: float64
```

Answer 1.2

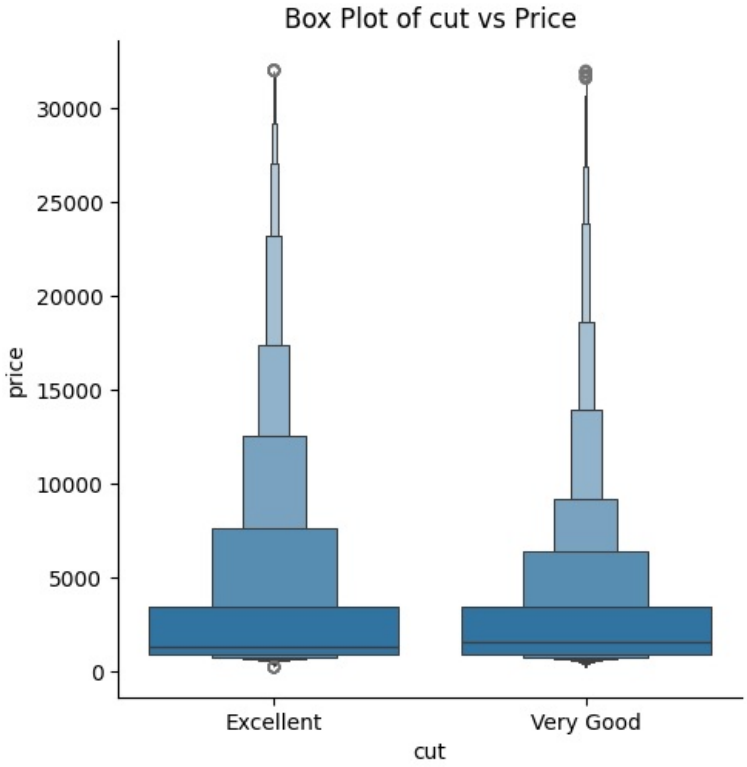
Histogram for numerical features is shown above.

For features with high skewness, power transformation can be utilized to convert skewed distributions into Gaussian distributions with zero mean and unit variance. Two types of power transformations available are box-cox and yeo-johnson. Additionally, simpler transformations such as square root, reciprocal, or log transformations of the feature can be applied. Alternatively, standardization alone can be employed to achieve optimal model performance. The decision has been made to solely utilize standardization.

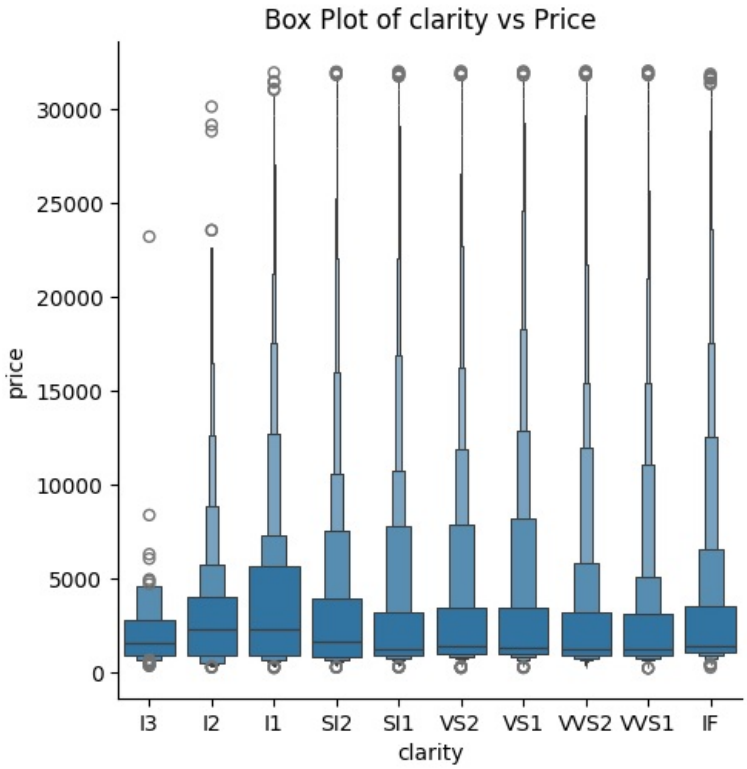
In [ ]:

```
categorical_features = ['cut', 'clarity', 'symmetry', 'polish', 'girdle_min', 'girdle_max', 'color']

figure_size = (8, 6)  # Adjust the overall figure size

for i, feature in enumerate(categorical_features, 1):
    plt.figure(figsize=figure_size)
    if feature == 'clarity':
      sns.catplot(x=feature, y='price', data=data,kind='boxen',order =clarity_ranking.keys())
    else :
      sns.catplot(x=feature, y='price', data=data,kind='boxen')
    plt.title(f'Box Plot of {feature} vs Price')
    plt.show()
```

```
<Figure size 800x600 with 0 Axes>
```

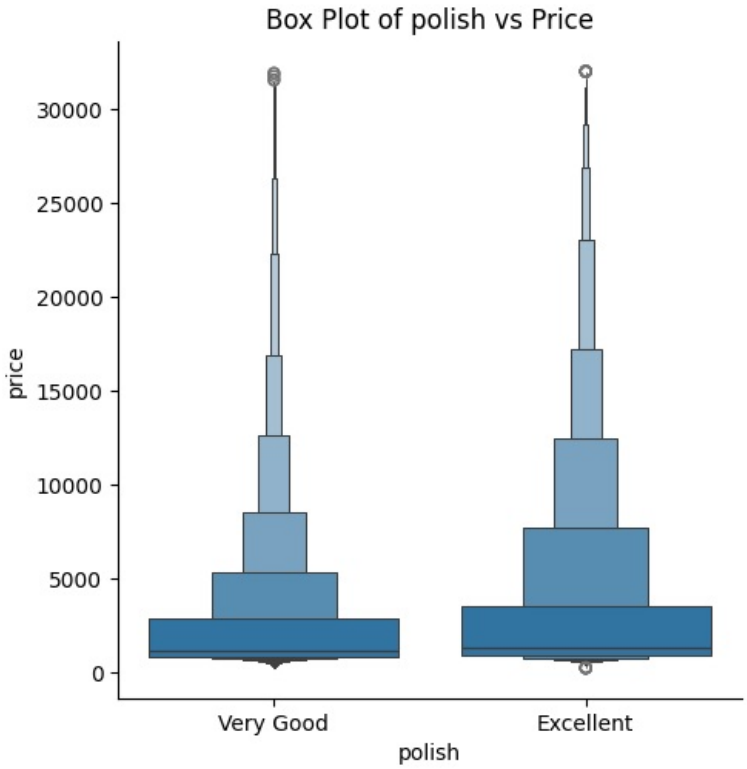


<Figure size 800x600 with 0 Axes>

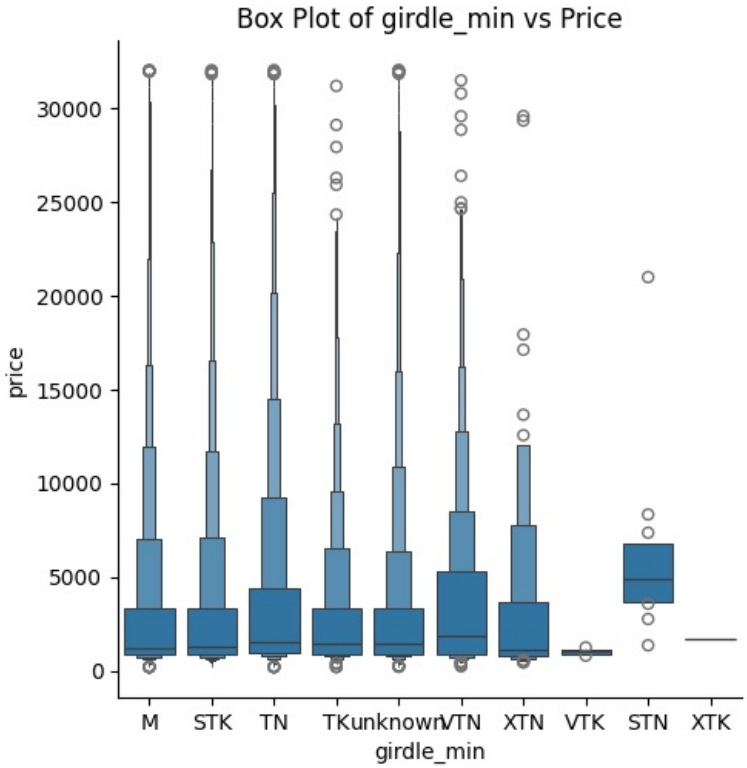


<Figure size 800x600 with 0 Axes>

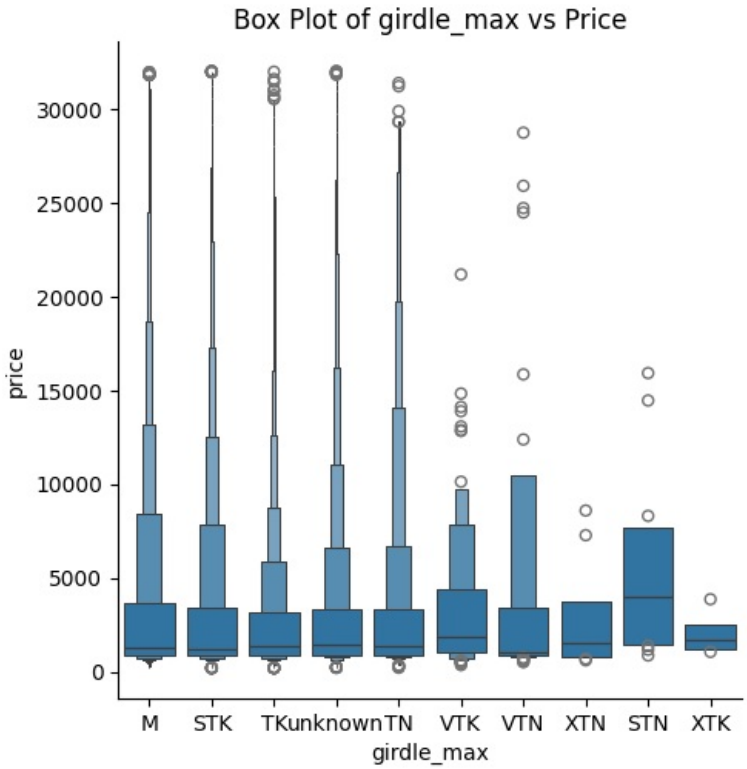
<Figure size 800x600 with 0 Axes>



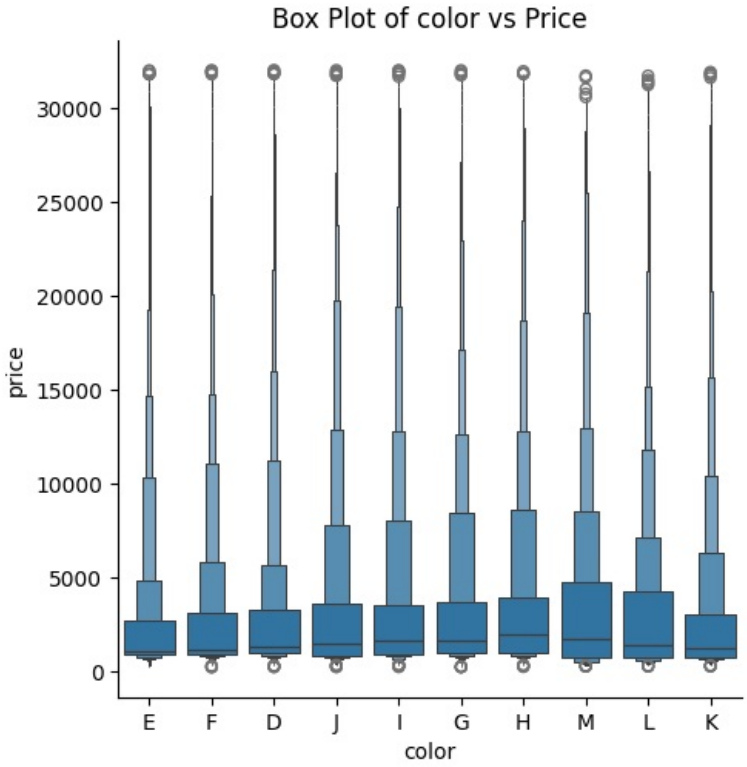
<Figure size 800x600 with 0 Axes>



<Figure size 800x600 with 0 Axes>



<Figure size 800x600 with 0 Axes>



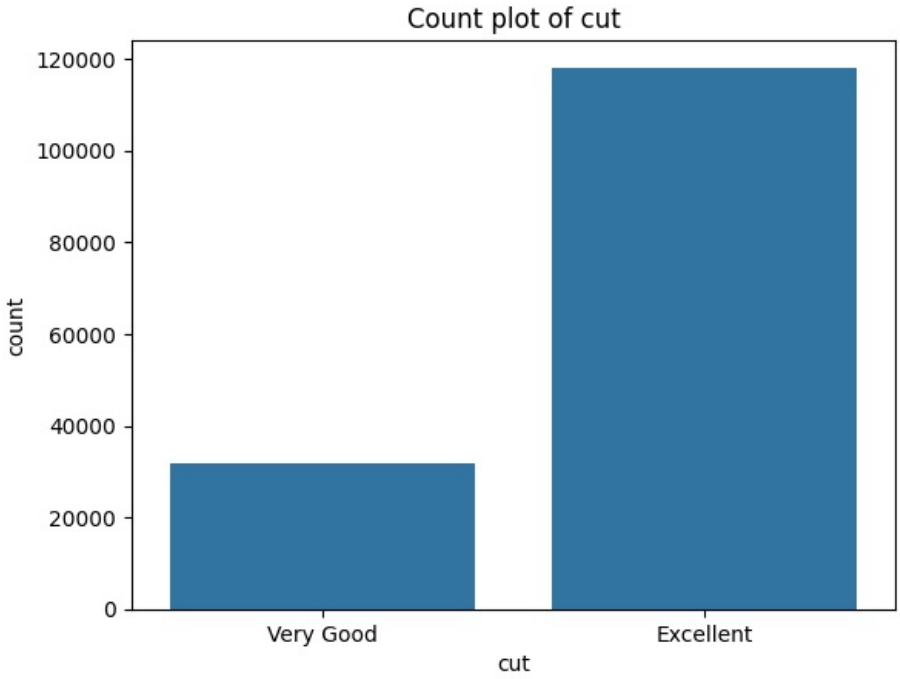
Answer 1.3

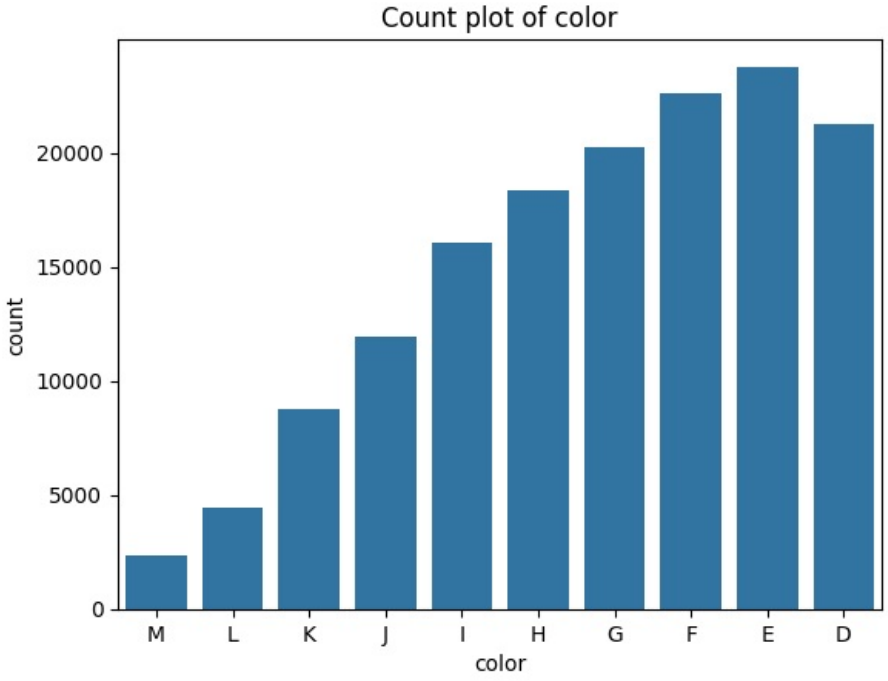
The plots are as shown above. It is apparent that higher cut quality is associated with higher prices. Conversely, there is a negative correlation between price and color, indicating that prices decrease as color improves. Regarding clarity, prices increase from I1 to VS1 but decrease between VS1 and IF. Subsequent analysis has revealed that categorical features are less effective for regression performance compared to numerical features.

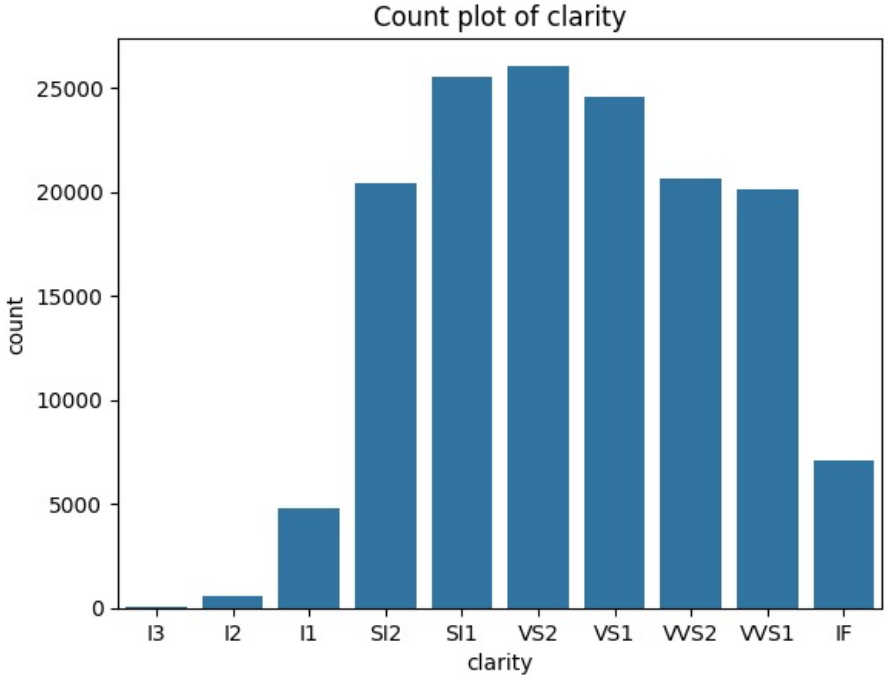
Plotting counts by color, cut and clarity

In [47]:

```
plt.figure()
sns.countplot(x=data["cut"],order = cat2num.keys())
plt.title("Count plot of cut")

plt.figure()
sns.countplot(x=data["color"],order = color_dict.keys())
plt.title("Count plot of color")

plt.figure()
sns.countplot(x=data["clarity"],order = clarity_ranking.keys())
plt.title("Count plot of clarity")
plt.show()
```


Count plot of cut
120000
100000
80000
60000
40000
20000
0
count
Very Good
Excellent
cut


Count plot of color
20000
15000
10000
5000
0
count
M
L
K
J
I
H
G
F
E
D
color


Count plot of clarity
25000
20000
15000
10000
5000
0
count
I3
I2
I1
SI2
SI1
VS2
VS1
VVS2
VVS1
IF
clarity

Answer 1.4

The plots are as shown above.

Standardize data (Answer 2.1)

We standardize columns as shown below. Note that categorical features have been ordinally encoded in previous cells and stored in dataframe 'dataenc'

In [12]:

```
feats = dataenc.loc[:, dataenc.columns != 'price'].to_numpy()
target = dataenc.price
scaler = StandardScaler()
feats_scaled = scaler.fit_transform(feats)
```

Linear + Ridge + Lasso Regression

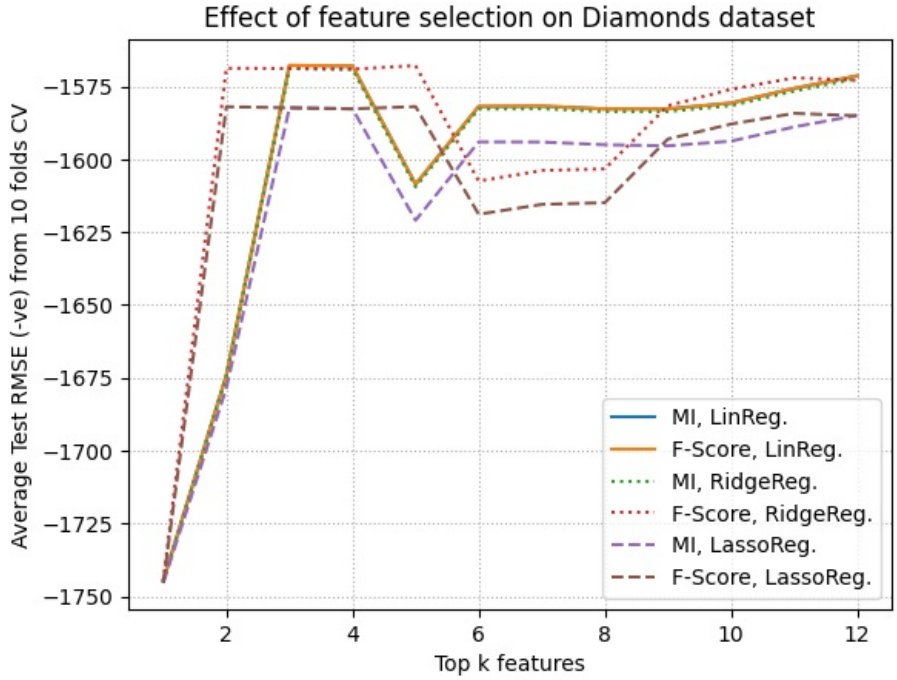
In [ ]:

```
diamonds_RMSE_MIR = []
diamonds_RMSE_FR = []


diamonds_RMSE_MIR_LR= []
diamonds_RMSE_FR_LR = []

diamonds_RMSE_MIR_RR= []
diamonds_RMSE_FR_RR = []


for i in range(1,feats.shape[1]):
    print('Testing LinReg, diamonds dataset for k = ', i)
    feats_M = SelectKBest(score_func=mutual_info_regression, k=i).fit_transform(feats, target)
    feats_F = SelectKBest(score_func=f_regression, k=i).fit_transform(feats, target)

    diamOut = cross_validate(LinearRegression(), feats_M, target, scoring=['neg_root_mean_squared_error'], cv=10,
n_jobs=-1)
    diamonds_RMSE_MIR.append(diamOut['test_neg_root_mean_squared_error'].mean())

    diamout = cross_validate(LinearRegression(), feats_F, target, scoring=['neg_root_mean_squared_error'], cv=10,
n_jobs=-1)
    diamonds_RMSE_FR.append(diamOut['test_neg_root_mean_squared_error'].mean())

    print('Testing RidgeReg, diamonds dataset for k = ', i)
    diamOut = cross_validate(Ridge(), feats_M, target, scoring=['neg_root_mean_squared_error'], cv=10,n_jobs=-1)
    diamonds_RMSE_MIR_RR.append(diamOut['test_neg_root_mean_squared_error'].mean())
    diamOut = cross_validate(Ridge(), feats_F, target, scoring=['neg_root_mean_squared_error'], cv=10,n_jobs=-1)
    diamonds_RMSE_FR_RR.append(diamOut['test_neg_root_mean_squared_error'].mean())

    print('Testing LassoReg, diamonds dataset for k = ', i)
    diamOut = cross_validate(Lasso(), feats_M, target, scoring=['neg_root_mean_squared_error'], cv=10,n_jobs=-1)
    diamonds_RMSE_MIR_LR.append(diamOut['test_neg_root_mean_squared_error'].mean())
    diamOut = cross_validate(Lasso(), feats_F, target, scoring=['neg_root_mean_squared_error'], cv=10,n_jobs=-1)
    diamonds_RMSE_FR_LR.append(diamOut['test_neg_root_mean_squared_error'].mean())
```

```
Testing LinReg, diamonds dataset for k =  1
Testing RidgeReg, diamonds dataset for k =  1
Testing LassoReg, diamonds dataset for k =  1
Testing LinReg, diamonds dataset for k =  2
Testing RidgeReg, diamonds dataset for k =  2
Testing LassoReg, diamonds dataset for k =  2
Testing LinReg, diamonds dataset for k =  3
Testing RidgeReg, diamonds dataset for k =  3
Testing LassoReg, diamonds dataset for k =  3
Testing LinReg, diamonds dataset for k =  4
Testing RidgeReg, diamonds dataset for k =  4
Testing LassoReg, diamonds dataset for k =  4
Testing LinReg, diamonds dataset for k =  5
Testing RidgeReg, diamonds dataset for k =  5
Testing LassoReg, diamonds dataset for k =  5
Testing LinReg, diamonds dataset for k =  6
Testing RidgeReg, diamonds dataset for k =  6
Testing LassoReg, diamonds dataset for k =  6
Testing LinReg, diamonds dataset for k =  7
Testing RidgeReg, diamonds dataset for k =  7
Testing LassoReg, diamonds dataset for k =  7
Testing LinReg, diamonds dataset for k =  8
Testing RidgeReg, diamonds dataset for k =  8
Testing LassoReg, diamonds dataset for k =  8
Testing LinReg, diamonds dataset for k =  9
Testing RidgeReg, diamonds dataset for k =  9
Testing LassoReg, diamonds dataset for k =  9
Testing LinReg, diamonds dataset for k =  10
Testing RidgeReg, diamonds dataset for k =  10
Testing LassoReg, diamonds dataset for k =  10
Testing LinReg, diamonds dataset for k =  11
Testing RidgeReg, diamonds dataset for k =  11
Testing LassoReg, diamonds dataset for k =  11
Testing LinReg, diamonds dataset for k =  12
Testing RidgeReg, diamonds dataset for k =  12
Testing LassoReg, diamonds dataset for k =  12
```

In [ ]:

```
plt.plot(np.arange(1,len(diamonds_RMSE_MIR)+1,1),diamonds_RMSE_MIR)
plt.plot(np.arange(1,len(diamonds_RMSE_FR)+1,1),diamonds_RMSE_FR)
plt.plot(np.arange(1,len(diamonds_RMSE_MIR_RR)+1,1),diamonds_RMSE_MIR_RR,':')
plt.plot(np.arange(1,len(diamonds_RMSE_FR_RR)+1,1),diamonds_RMSE_FR_RR,':')
plt.plot(np.arange(1,len(diamonds_RMSE_MIR_LR)+1,1),diamonds_RMSE_MIR_LR,'--')
plt.plot(np.arange(1,len(diamonds_RMSE_FR_LR)+1,1),diamonds_RMSE_FR_LR,'--')
plt.legend(['MI, LinReg.','F-Score, LinReg.','MI, RidgeReg.',
            'F-Score, RidgeReg.','MI, LassoReg.','F-Score, LassoReg.'],loc='best')
plt.grid(linestyle=':')
plt.xlabel('Top k features')
plt.ylabel('Average Test RMSE (-ve) from 10 folds CV')
plt.title('Effect of feature selection on Diamonds dataset')
plt.show()
```



In [53]:

```
mut_info_selector = SelectKBest(mutual_info_regression, k = "all")
mut_info_selector.fit_transform(dataenc.loc[:, dataenc.columns != 'price'],target) # accepts row vector
print(mut_info_selector.scores_)
print('\n\n\n')
print("All names: ",mut_info_selector.feature_names_in_)
sort_idx_mi = np.argsort(mut_info_selector.scores_)[::-1]
print("Sorted names(in decreasing order): ", mut_info_selector.feature_names_in_[sort_idx_mi])
```

```
[1.37545797 0.04404253 0.02785272 1.192858   1.20586607 1.15865957
 0.02535653 0.0384929  0.02459617 0.0231724  0.01211    0.182262
 0.16630678]



All names:  ['carat' 'depth_percent' 'table_percent' 'length' 'width' 'depth'
 'girdle_min_enc' 'girdle_max_enc' 'cut_enc' 'symmetry_enc' 'polish_enc'
 'color_enc' 'clarity_enc']
Sorted names(in decreasing order):  ['carat' 'width' 'length' 'depth' 'color_enc' 'clarity_enc'
 'depth_percent' 'girdle_max_enc' 'table_percent' 'girdle_min_enc'
 'cut_enc' 'symmetry_enc' 'polish_enc']
```

Answer 2.2

Mutual information (MI) and F scores functions are used to select the most important features. This step is extremely useful as selecting just a part of input features helps in boosting the model performance for testing, especially for linear models.

• In simpler words, this helps us avoid the problem of overfitting by not considering the less relevant or redundant features.

• Mutual information is calculated between two variables and measures the reduction in uncertainty for one variable given a known value of the other variable. The mutual information between two random variables X and Y can be stated formally as follows:

$$I(X, Y) = H(X) - H(X|Y)$$

• The F1 score can be interpreted as a harmonic mean of the precision and recall, where an F1 score reaches its best value at 1 and worst score at 0. The relative contribution of precision and recall to the F1 score are equal. The formula for the F1 score is:

$$F1 = \frac{2 * Precision * Recall}{Precision + Recall}$$

The effect of feature selection on Diamonds dataset can be seen in the plot above where the average test RMSE is plotted against top 'k' features. From the above plots, we can infer that:

• Linear, ridge, and lasso regressions exhibit increasing test scores that eventually converge at a certain threshold. This convergence is attributed to the informative nature of the top features, which sufficiently convey the necessary information. Feature selection diminishes model complexity while enhancing interpretability.

• Mutual information (MI) effectively captures non-linear relationships between features and target variables but operates at a slower pace compared to F1-score and necessitates a larger sample size.

We see that features with **Least MI wrt Price are Symmetry and Polish**.

In [13]:

```
k_val = 7 # Inferred from the graph above

#Unstandardized
feats_f = SelectKBest(score_func=f_regression, k=k_val).fit_transform(feats, target)
feats_mir = SelectKBest(score_func=mutual_info_regression, k=k_val).fit_transform(feats, target)

#Standardized
feats_fs = SelectKBest(score_func=f_regression, k=k_val).fit_transform(feats_scaled, target)
feats_mirs = SelectKBest(score_func=mutual_info_regression, k=k_val).fit_transform(feats_scaled, target)
```

Answer 2.2 (continued)

From plot previously, we note that k=7 is ideal and we select k=7 best using F-score and MI. For most of the parts, I'll use best k=7 based on F1-score.

Regression analyses

Answer 4.1

- Ordinary Least Squares (OLS): $\min_\beta ||Y - X\beta||_2^2$
- Lasso: $\min_\beta ||Y - X\beta||_2^2 + \lambda||\beta||_1$
- Ridge: $\min_\beta ||Y - X\beta||_2^2 + \lambda||\beta||_2^2$

Lasso regression is essentially linear regression with L1 regularization. It is particularly suitable for feature selection and constructing simple and sparse regression models, where features with zero weights are discarded. This is because L1 regularization aims to shrink all weights to zero, regardless of their magnitude. L1 regularization tends to shrink weights to zero more readily than L2 regularization for similar test accuracies, as it assumes priors on weights sampled from an isotropic Laplace distribution, which has a lower Q factor compared to the Gaussian distribution. Consequently, only a subset of features remains active in the learned hypothesis.

On the other hand, **Ridge regression** is linear regression with L2 regularization. It assumes priors on weights sampled from a unit Gaussian distribution and is suitable for reducing the effects of collinear features, which can increase the model's variance. The subgradient of the norm of weights in Ridge regression depends not only on the sign but also on the magnitude of the weights. This scaling of the weight variance reduces the model's reliance on new features and encourages distributed contribution from all features. As a result, Ridge regression tends to produce regression models with diffuse weights compared to the sparse weights obtained from L1 regularization. Therefore, Ridge regression promotes the participation of all features in the learned hypothesis to prevent overfitting.

In summary, the L2 regularization term serves the purpose of shrinkage, while L1 regularization can be employed for feature selection or screening purposes.

Linear Regression

In [ ]:

```
XdiamOut = cross_validate(LinearRegression(), feats_f, target, scoring=['neg_root_mean_squared_error'], cv=10,n_j
obs=-1,return_train_score=True)
print('No standardization, F1, linear regression: Test=',XdiamOut['test_neg_root_mean_squared_error'].mean(),',Tr
ain=',XdiamOut['train_neg_root_mean_squared_error'].mean())
XdiamOut = cross_validate(LinearRegression(), feats_mir, target, scoring=['neg_root_mean_squared_error'], cv=10,n
_jobs=-1,return_train_score=True)
print('No standardization, MI, linear regression: Test=',XdiamOut['test_neg_root_mean_squared_error'].mean(),',Tr
ain=',XdiamOut['train_neg_root_mean_squared_error'].mean())
XdiamOut = cross_validate(LinearRegression(), feats_fs, target, scoring=['neg_root_mean_squared_error'], cv=10,n_
jobs=-1,return_train_score=True)
print('Standardization, F1, linear regression:  Test=',XdiamOut['test_neg_root_mean_squared_error'].mean(),',Trai
n=',XdiamOut['train_neg_root_mean_squared_error'].mean())
XdiamOut = cross_validate(LinearRegression(), feats_mirs, target, scoring=['neg_root_mean_squared_error'], cv=10,
n_jobs=-1,return_train_score=True)
print('Standardization, MI, linear regression: Test=',XdiamOut['test_neg_root_mean_squared_error'].mean(),',Train
=',XdiamOut['train_neg_root_mean_squared_error'].mean())
```

```
No standardization, F1, linear regression: Test= -1602.8617683848083 ,Train= -1653.4029604972463
No standardization, MI, linear regression: Test= -1581.7193267609136 ,Train= -1556.1352753058177
Standardization, F1, linear regression:  Test= -1602.8617683848702 ,Train= -1653.4029604972463
Standardization, MI, linear regression: Test= -1581.7193267609773 ,Train= -1556.1352753058177
```

Ridge Regression

In [ ]:

```
pipe_RR = Pipeline([('model', Ridge(random_state=42))])
param_grid = {
    'model__alpha': [10.0**x for x in np.arange(-5,5)]
}


print("Testing Diamonds ..\n")
griddiamRR_F = GridSearchCV(pipe_RR, param_grid=param_grid, cv=10, n_jobs=-1, verbose=1,
                      scoring='neg_root_mean_squared_error', return_train_score=True).fit(feats_f, target)
griddiamRR_FS = GridSearchCV(pipe_RR, param_grid=param_grid, cv=10, n_jobs=-1, verbose=1,
                      scoring='neg_root_mean_squared_error', return_train_score=True).fit(feats_fs, target)
griddiamRR_M = GridSearchCV(pipe_RR, param_grid=param_grid, cv=10, n_jobs=-1, verbose=1,
                      scoring='neg_root_mean_squared_error', return_train_score=True).fit(feats_mir, target)
gridBikeRR_MS = GridSearchCV(pipe_RR, param_grid=param_grid, cv=10, n_jobs=-1, verbose=1,
                      scoring='neg_root_mean_squared_error', return_train_score=True).fit(feats_mirs, target)
```

```
Testing Diamonds ..

Fitting 10 folds for each of 10 candidates, totalling 100 fits
Fitting 10 folds for each of 10 candidates, totalling 100 fits
Fitting 10 folds for each of 10 candidates, totalling 100 fits
Fitting 10 folds for each of 10 candidates, totalling 100 fits
```

In [ ]:

```
print(' No standardization, F1, RidgeRegression, Test RMSE:',griddiamRR_F.best_score_,
      ',alpha:',griddiamRR_F.best_params_,'train RMSE',max(griddiamRR_F.cv_results_['mean_train_score']))
print(' Standardization, F1, RidgeRegression, Test RMSE:',griddiamRR_FS.best_score_,
      ',alpha:',griddiamRR_FS.best_params_,'train RMSE',max(griddiamRR_FS.cv_results_['mean_train_score']))
print(' No standardization, MI, RidgeRegression, Test RMSE:',griddiamRR_M.best_score_,
      ',alpha:',griddiamRR_M.best_params_,'train RMSE',max(griddiamRR_M.cv_results_['mean_train_score']))
print(' Standardization, MI, RidgeRegression, Test RMSE:',gridBikeRR_MS.best_score_,
      ',alpha:',gridBikeRR_MS.best_params_,'train RMSE',max(gridBikeRR_MS.cv_results_['mean_train_score']))
```

```
 No standardization, F1, RidgeRegression, Test RMSE: -1602.8617779660865 ,alpha: {'model__alpha': 1e
-05} train RMSE -1653.4029604972468
 Standardization, F1, RidgeRegression, Test RMSE: -1602.861769981662 ,alpha: {'model__alpha': 1e-05}
train RMSE -1653.4029604972463
 No standardization, MI, RidgeRegression, Test RMSE: -1581.7193357158208 ,alpha: {'model__alpha': 1e
-05} train RMSE -1556.1352753058177
 Standardization, MI, RidgeRegression, Test RMSE: -1581.7193281640834 ,alpha: {'model__alpha': 1e-05
} train RMSE -1556.1352753058177
```

Lasso Regression

In [ ]:

```
pipe_LAR = Pipeline([('model', Lasso(random_state=42))])
param_grid = {
    'model__alpha': [10.0**x for x in np.arange(-5,5)]
}


print("Testing Diamond ..\n")
griddiamLAR_F = GridSearchCV(pipe_LAR, param_grid=param_grid, cv=10, n_jobs=-1, verbose=1,
                      scoring='neg_root_mean_squared_error', return_train_score=True).fit(feats_f, target)
griddiamLAR_FS = GridSearchCV(pipe_LAR, param_grid=param_grid, cv=10, n_jobs=-1, verbose=1,
                      scoring='neg_root_mean_squared_error', return_train_score=True).fit(feats_fs, target)
griddiamLAR_M = GridSearchCV(pipe_LAR, param_grid=param_grid, cv=10, n_jobs=-1, verbose=1,
                      scoring='neg_root_mean_squared_error', return_train_score=True).fit(feats_mir, target)
griddiamLAR_MS = GridSearchCV(pipe_LAR, param_grid=param_grid, cv=10, n_jobs=-1, verbose=1,
                      scoring='neg_root_mean_squared_error', return_train_score=True).fit(feats_mirs, target)

print('No standardization, F1, LassoRegression, Test RMSE:',griddiamLAR_F.best_score_,
      ',alpha:',griddiamLAR_F.best_params_,'train RMSE',max(griddiamLAR_F.cv_results_['mean_train_score']))
print('Standardization, F1, LassoRegression, Test RMSE:',griddiamLAR_FS.best_score_,
      ',alpha:',griddiamLAR_FS.best_params_,'train RMSE',max(griddiamLAR_FS.cv_results_['mean_train_score']))
print('No standardization, MI, LassoRegression, Test RMSE:',griddiamLAR_M.best_score_,
      ',alpha:',griddiamLAR_M.best_params_,'train RMSE',max(griddiamLAR_M.cv_results_['mean_train_score']))
print('Standardization, MI, LassoRegression, Test RMSE:',griddiamLAR_MS.best_score_,
      ',alpha:',griddiamLAR_MS.best_params_,'train RMSE',max(griddiamLAR_MS.cv_results_['mean_train_score']))
```

```
Testing Diamond ..

Fitting 10 folds for each of 10 candidates, totalling 100 fits
Fitting 10 folds for each of 10 candidates, totalling 100 fits
Fitting 10 folds for each of 10 candidates, totalling 100 fits
Fitting 10 folds for each of 10 candidates, totalling 100 fits
No standardization, F1, LassoRegression, Test RMSE: -1602.861957239226 ,alpha: {'model__alpha': 1e-0
5} train RMSE -1653.4029604972607
Standardization, F1, LassoRegression, Test RMSE: -1602.861866591929 ,alpha: {'model__alpha': 1e-05}
train RMSE -1653.4029604972511
No standardization, MI, LassoRegression, Test RMSE: -1581.5851091157976 ,alpha: {'model__alpha': 0.0
1} train RMSE -1556.1352753058316
Standardization, MI, LassoRegression, Test RMSE: -1581.5569809806207 ,alpha: {'model__alpha': 0.1} t
rain RMSE -1556.1352753058222
```

Answer 4.2

For this question, we analyze the performance of train and test RMSE of linear regression, Lasso regression, and Ridge regression on the top 6 selected features from the diamonds dataset and top 10 features from the gas emission dataset.

To compute the optimal penalty parameter for each regularization scheme as well as test the performance of the model, we construct a pipeline scheme by performing a grid search with 10-fold cross-validation with λ in the range of $[10^{-5}, 10^{5}]$. We also used LassoCV and GridCV() to calculate the optimal penalty terms. We further explored the effects of feature scaling/standardization that is explained in the next question.

The results are explained in below table:

Standardization	Feature Selection	Regression Model	Test RMSE	Train RMSE	Alpha Value
No	F1	Linear	-1602.8618	-1653.4030	N/A
No	MI	Linear	-1581.7193	-1556.1353	N/A
Standardization	F1	Linear	-1602.8618	-1653.4030	N/A
Standardization	MI	Linear	-1581.7193	-1556.1353	N/A
No	F1	Ridge	-1602.8618	-1653.4030	1e-05
Standardization	F1	Ridge	-1602.8620	-1653.4030	1e-05
No	MI	Ridge	-1581.7193	-1556.1353	1e-05
Standardization	MI	Ridge	-1581.7193	-1556.1353	1e-05
No	F1	Lasso	-1602.8620	-1653.4030	1e-05
Standardization	F1	Lasso	-1602.8619	-1653.4030	1e-05
No	MI	Lasso	-1581.5851	-1556.1353	0.01
Standardization	MI	Lasso	-1581.5570	-1556.1353	0.1

Answer 4.3

From the analysis presented in above table, it's evident that feature scaling influences the RMSE when regularization is applied, as evidenced by the optimal RMSE achieved.

Feature scaling preserves the original distribution of the dataset. In the absence of regularization, any alterations in values due to feature scaling would be reflected in the model's weights to minimize RMSE. However, since the data distribution remains unchanged, the impact of feature scaling on the coefficients would be linear, resulting in no discernible performance improvements or deteriorations in test RMSE without regularization.

As discussed in Question 1.2, standardizing data helps address distribution skewness. Consider two prominent features, one with a significantly smaller range and magnitude compared to the other. Without normalization, the model may assign disproportionately larger coefficients to the smaller feature to maintain balance with the larger one. However, when regularization is applied, it may penalize or remove the coefficients of the smaller feature, while minimally affecting those of the larger feature. Feature scaling ensures that regularization does not unduly penalize smaller features, leading to more robust and well-conditioned models.

p-Value

In [ ]:

```
p_ex = OLS(target, dataenc.loc[:, dataenc.columns != 'price']).fit()
print(p_ex.pvalues.sort_values(ascending=True))
```

```
carat             0.000000e+00
depth_percent     0.000000e+00
length            0.000000e+00
color_enc         0.000000e+00
clarity_enc       0.000000e+00
cut_enc           4.225408e-60
girdle_max_enc    4.927433e-44
table_percent     8.403301e-40
girdle_min_enc    3.542865e-30
symmetry_enc      9.821138e-07
depth             5.303614e-03
width             6.645046e-03
polish_enc        3.640656e-02
dtype: float64
```

Answer 4.4

The p-value for each feature tests the null hypothesis that the feature has no correlation with the target variable i.e, P-values in the linear regression model measures the probability of feature coefficients being equal to zero. Hence, if the p-value for some feature is very close to 0, we will have the confidence to say that particular feature is significant in the linear model. For the diamond dataset, the p- values were as follows:

Feature	p-value
carat	0.000000e+00
depth_percent	0.000000e+00
length	0.000000e+00
color_enc	0.000000e+00
clarity_enc	0.000000e+00
cut_enc	4.225408e-60
girdle_max_enc	4.927433e-44
table_percent	8.403301e-40
girdle_min_enc	3.542865e-30
symmetry_enc	9.821138e-07
depth	5.303614e-03
width	6.645046e-03
polish_enc	3.640656e-02

Polynomial Regression

In [ ]:

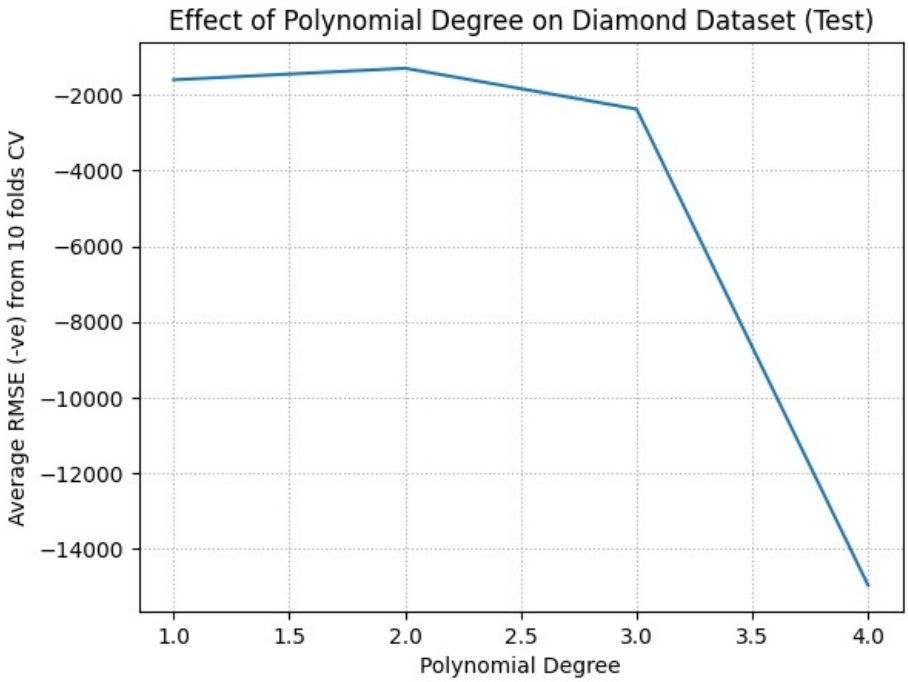
```
degree_list = np.arange(1,5,1)
pipe_PR_diam = Pipeline([
    ('PR', PolynomialFeatures()),
    ('model', Ridge(random_state=42))
])
param_grid_PR = {
    'PR__degree': degree_list,
    'model__alpha': [10.0**x for x in np.arange(-4,5)]

}

polyreg_gs = GridSearchCV(pipe_PR_diam, param_grid=param_grid_PR, cv=10, n_jobs=-1, verbose=1,
                     scoring='neg_root_mean_squared_error', return_train_score=True).fit(feats_fs,target)
```

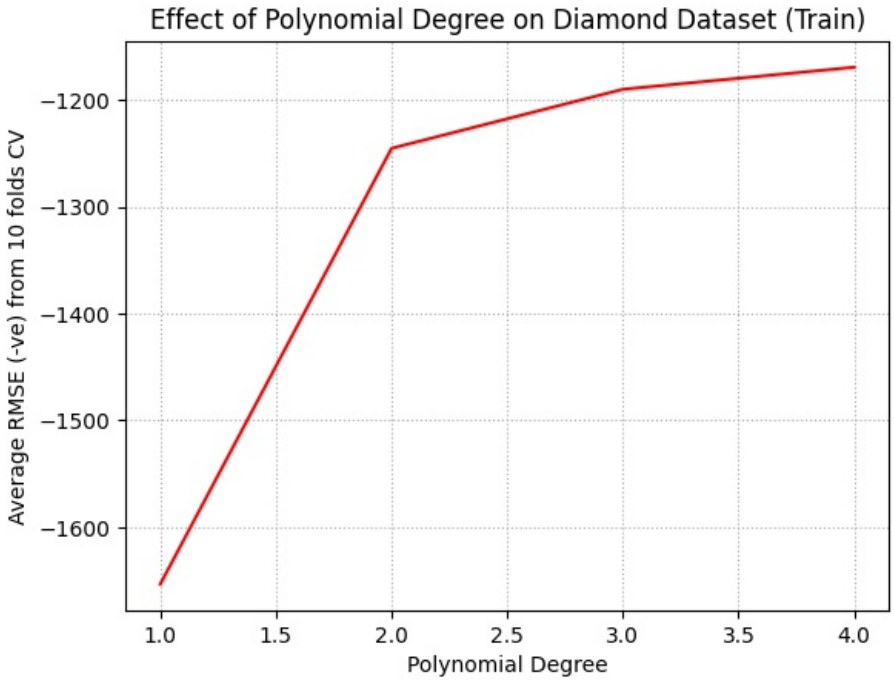
```
Fitting 10 folds for each of 36 candidates, totalling 360 fits
```

In [ ]:

```
poly_result = pd.DataFrame(polyreg_gs.cv_results_)[['mean_test_score','mean_train_score','param_PR__degree','para
m_model__alpha']]
diam_score = []
diam_train = []
diam_alpha = []
for i in degree_list:
    diam_score.append((poly_result.loc[poly_result['param_PR__degree'] == i]).max().mean_test_score)
    diam_train.append((poly_result.loc[poly_result['param_PR__degree'] == i]).max().mean_train_score)
    diam_alpha.append(float(poly_result['param_model__alpha'][
        (poly_result.loc[poly_result['param_PR__degree'] == i])
        [['mean_test_score']].idxmax()].to_numpy()))
plt.plot(degree_list,diam_score)
plt.grid(linestyle=':')
plt.xlabel('Polynomial Degree')
plt.ylabel('Average RMSE (-ve) from 10 folds CV')
plt.title('Effect of Polynomial Degree on Test Dataset')
plt.show()
plt.plot(degree_list,diam_train,'r')
plt.grid(linestyle=':')
plt.xlabel('Polynomial Degree')
plt.ylabel('Average RMSE (-ve) from 10 folds CV')
plt.title('Effect of Polynomial Degree on Train Dataset')
plt.show()
```

```
<ipython-input-12-6ac521358339>:8: DeprecationWarning: Conversion of an array with ndim > 0 to a sca
lar is deprecated, and will error in future. Ensure you extract a single element from your array bef
ore performing this operation. (Deprecated NumPy 1.25.)
  diam_alpha.append(float(poly_result['param_model__alpha'][
<ipython-input-12-6ac521358339>:8: DeprecationWarning: Conversion of an array with ndim > 0 to a sca
lar is deprecated, and will error in future. Ensure you extract a single element from your array bef
ore performing this operation. (Deprecated NumPy 1.25.)
  diam_alpha.append(float(poly_result['param_model__alpha'][
<ipython-input-12-6ac521358339>:8: DeprecationWarning: Conversion of an array with ndim > 0 to a sca
lar is deprecated, and will error in future. Ensure you extract a single element from your array bef
ore performing this operation. (Deprecated NumPy 1.25.)
  diam_alpha.append(float(poly_result['param_model__alpha'][
<ipython-input-12-6ac521358339>:8: DeprecationWarning: Conversion of an array with ndim > 0 to a sca
lar is deprecated, and will error in future. Ensure you extract a single element from your array bef
ore performing this operation. (Deprecated NumPy 1.25.)
  diam_alpha.append(float(poly_result['param_model__alpha'][
```





Salient features

In [ ]:

```
chY = SelectKBest(score_func=f_regression, k=7)
XTranscode_Test = chY.fit_transform(dataenc.loc[:, dataenc.columns != 'price'], dataenc.price)
column_names = dataenc.loc[:, dataenc.columns != 'price'].columns[chY.get_support()]

b_params = polyreg_gs.best_estimator_.get_params()
b_coefs = b_params['model'].coef_
b_feature_name = list(column_names)
b_names = b_params['PR'].get_feature_names_out(b_feature_name)
b_sorted_indice = np.argsort(-abs(b_coefs))
salient_features =[b_names[i] for i in b_sorted_indice[:5]]
print ('Top 5 Salient features :',salient_features)
```

```
Top 5 Salient features : ['width', 'length', 'carat', 'carat color_enc', 'length^2']
```

Answer 5.1

The most salient features are those with greatest absolute coefficients. We performed a gridsearch with 10-fold cross validation on each dataset to find these values For the diamonds dataset, the top 5 salient features are: ['width', 'length', 'carat', 'carat color_enc', 'length^2']

which makes sense with the correlation map of features with target variable price with carat features occupying the most salient features.

Answer 5.2

- Performance in terms of train and test RMSE of polynomial regression models with various degrees was assessed on the top 7 selected features from the diamonds dataset using F1-Score.
- Ridge regression with L2 regularization was employed to mitigate overfitting, with the penalty parameter optimized through grid search with 10-fold cross-validation.
- For the diamonds dataset, the optimal polynomial degree was determined to be 2, characterized by the lowest test RMSE, indicating minimal overfitting risk.
- Unconstrained increases in polynomial degree can lead to exponential growth in parameters, causing computational bottlenecks. Moreover, higher degrees may introduce excessive complexity and overfitting due to increased feature combinations.
- This pattern was consistent across both datasets, with test RMSE rising as polynomial degree increased.

Neural network

In [ ]:

```
pipe_NN = Pipeline([
    ('model', MLPRegressor(random_state=42,max_iter=1000, verbose = True,alpha = 0.007,activation = 'relu',early_
stopping = True))
])

param_grid_NN = {
    'model__hidden_layer_sizes': [(50,), (30,20)],
    # 'model__alpha': [5*10.0**x for x in np.arange(-2,0)],
    # 'model__activation': ['logistic', 'relu']
}
```

In [ ]:

```
griddiam_NN = GridSearchCV(pipe_NN, param_grid=param_grid_NN, cv=10, n_jobs=-1, verbose=10,
                                scoring='neg_root_mean_squared_error', return_train_score=True).fit(feats_fs, t
arget)
```

```
Fitting 10 folds for each of 2 candidates, totalling 20 fits
Iteration 1, loss = 16413756.83762271
Validation score: -0.441311
Iteration 2, loss = 15206423.85166683
Validation score: -0.284877
Iteration 3, loss = 13142687.32267415
Validation score: -0.078803
Iteration 4, loss = 10766284.90304198
Validation score: 0.137970
Iteration 5, loss = 8449895.94200918
Validation score: 0.337322
Iteration 6, loss = 6421941.14511140
Validation score: 0.505152
Iteration 7, loss = 4790081.90786373
Validation score: 0.636082
Iteration 8, loss = 3575024.23562442
Validation score: 0.729588
Iteration 9, loss = 2749403.75111931
Validation score: 0.790165
Iteration 10, loss = 2244458.62964041
Validation score: 0.825365
Iteration 11, loss = 1953165.00409470
Validation score: 0.845368
Iteration 12, loss = 1777408.73379939
Validation score: 0.857622
```

```
Iteration 13, loss = 1657844.78726404
Validation score: 0.866212
Iteration 14, loss = 1565357.23365779
Validation score: 0.872825
Iteration 15, loss = 1490110.29784903
Validation score: 0.878190
Iteration 16, loss = 1427467.13883458
Validation score: 0.882615
Iteration 17, loss = 1375169.23128363
Validation score: 0.886245
Iteration 18, loss = 1331971.86304574
Validation score: 0.889233
Iteration 19, loss = 1295670.56653785
Validation score: 0.891719
Iteration 20, loss = 1265564.15227435
Validation score: 0.893807
Iteration 21, loss = 1240221.99364294
Validation score: 0.895615
Iteration 22, loss = 1218560.15046733
Validation score: 0.897159
Iteration 23, loss = 1199671.74470484
Validation score: 0.898582
Iteration 24, loss = 1182295.56219723
Validation score: 0.899877
Iteration 25, loss = 1166071.12181677
Validation score: 0.901086
Iteration 26, loss = 1150803.63956254
Validation score: 0.902238
Iteration 27, loss = 1136118.92093915
Validation score: 0.903333
Iteration 28, loss = 1122352.73388073
Validation score: 0.904350
Iteration 29, loss = 1109010.82725483
Validation score: 0.905327
Iteration 30, loss = 1096128.86858385
Validation score: 0.906246
Iteration 31, loss = 1084293.21006930
Validation score: 0.907100
Iteration 32, loss = 1073200.21373007
Validation score: 0.907885
Iteration 33, loss = 1062725.38350632
Validation score: 0.908593
Iteration 34, loss = 1052982.88515630
Validation score: 0.909251
Iteration 35, loss = 1043915.21513731
Validation score: 0.909870
Iteration 36, loss = 1035313.09134001
Validation score: 0.910465
Iteration 37, loss = 1027159.13561959
Validation score: 0.911003
Iteration 38, loss = 1019513.58830012
Validation score: 0.911477
Iteration 39, loss = 1012266.43498192
Validation score: 0.911920
Iteration 40, loss = 1005805.59687912
Validation score: 0.912315
Iteration 41, loss = 999738.03497932
Validation score: 0.912710
Iteration 42, loss = 993666.37208511
Validation score: 0.913056
Iteration 43, loss = 988247.74516535
Validation score: 0.913372
Iteration 44, loss = 983166.24860095
Validation score: 0.913657
Iteration 45, loss = 978221.69973620
Validation score: 0.913942
Iteration 46, loss = 973548.53454200
Validation score: 0.914190
Iteration 47, loss = 969071.87456608
Validation score: 0.914388
Iteration 48, loss = 965158.83961691
Validation score: 0.914624
Iteration 49, loss = 961175.78862148
Validation score: 0.914810
Iteration 50, loss = 957540.62854954
Validation score: 0.915006
Iteration 51, loss = 953993.15071051
Validation score: 0.915176
Iteration 52, loss = 950551.00335517
Validation score: 0.915342
Iteration 53, loss = 947332.10135158
Validation score: 0.915487
Iteration 54, loss = 944202.44867643
```

```
Validation score: 0.915627
Iteration 55, loss = 941174.81555782
Validation score: 0.915771
Iteration 56, loss = 938210.72518845
Validation score: 0.915886
Iteration 57, loss = 935232.51786413
Validation score: 0.916040
Iteration 58, loss = 932544.46676322
Validation score: 0.916160
Iteration 59, loss = 929702.70443196
Validation score: 0.916288
Iteration 60, loss = 927160.12800166
Validation score: 0.916394
Iteration 61, loss = 924492.24803285
Validation score: 0.916521
Iteration 62, loss = 922198.77790036
Validation score: 0.916630
Iteration 63, loss = 919666.75045868
Validation score: 0.916742
Iteration 64, loss = 917149.56756938
Validation score: 0.916861
Iteration 65, loss = 914698.83201978
Validation score: 0.916954
Iteration 66, loss = 912320.23427783
Validation score: 0.917037
Iteration 67, loss = 909981.73190983
Validation score: 0.917181
Iteration 68, loss = 907801.01489050
Validation score: 0.917280
Iteration 69, loss = 905529.36039105
Validation score: 0.917372
Iteration 70, loss = 903302.90156380
Validation score: 0.917464
Iteration 71, loss = 901263.81681199
Validation score: 0.917566
Iteration 72, loss = 899185.36344034
Validation score: 0.917659
Iteration 73, loss = 897138.22258301
Validation score: 0.917761
Iteration 74, loss = 895105.39294822
Validation score: 0.917810
Iteration 75, loss = 893276.85598047
Validation score: 0.917924
Iteration 76, loss = 891409.05099041
Validation score: 0.918018
Iteration 77, loss = 889574.41186825
Validation score: 0.918073
Iteration 78, loss = 887635.09303302
Validation score: 0.918193
Iteration 79, loss = 885900.93274245
Validation score: 0.918265
Iteration 80, loss = 884125.81032907
Validation score: 0.918357
Iteration 81, loss = 882421.43887451
Validation score: 0.918434
Iteration 82, loss = 880705.27773646
Validation score: 0.918509
Iteration 83, loss = 879025.20065808
Validation score: 0.918599
Iteration 84, loss = 877275.66966933
Validation score: 0.918661
Iteration 85, loss = 875602.40501110
Validation score: 0.918762
Iteration 86, loss = 874075.14538139
Validation score: 0.918836
Iteration 87, loss = 872410.75287650
Validation score: 0.918893
Iteration 88, loss = 870915.02722829
Validation score: 0.919006
Iteration 89, loss = 869386.72279720
Validation score: 0.919083
Iteration 90, loss = 867888.93755970
Validation score: 0.919149
Iteration 91, loss = 866419.74909449
Validation score: 0.919233
Iteration 92, loss = 865009.95307824
Validation score: 0.919292
Iteration 93, loss = 863587.24501342
Validation score: 0.919324
Iteration 94, loss = 862120.57853472
Validation score: 0.919456
Iteration 95, loss = 860980.47973544
Validation score: 0.919521
```

```
Iteration 96, loss = 859626.31459778
Validation score: 0.919596
Iteration 97, loss = 858343.88837127
Validation score: 0.919651
Iteration 98, loss = 856919.70512532
Validation score: 0.919745
Iteration 99, loss = 855802.31298739
Validation score: 0.919800
Iteration 100, loss = 854555.11312987
Validation score: 0.919877
Iteration 101, loss = 853356.71338804
Validation score: 0.919932
Iteration 102, loss = 852130.40244207
Validation score: 0.919999
Iteration 103, loss = 851013.48969127
Validation score: 0.920069
Iteration 104, loss = 849870.43650940
Validation score: 0.920152
Iteration 105, loss = 848859.75874938
Validation score: 0.920209
Validation score did not improve more than tol=0.000100 for 10 consecutive epochs. Stopping.
```

In [ ]:

```
print('Test RMSE:', griddiam_NN.best_score_,
      ', \nBest Parameters:', griddiam_NN.best_params_,
      ', \nTrain RMSE:', max(griddiam_NN.cv_results_['mean_train_score']))
```

```
Test RMSE: -1316.730587329082 ,
Best Parameters: {'model__hidden_layer_sizes': (50,)} ,
Train RMSE: -1201.6367771719893
```

Answer 6.1

Grid-search for Neural network is implemented as above. Since the network was taking too long (it took > 8 hours with no result), I could only experiment with two different model settings with 10 folds each, equalling to 20 fits overall. Hidden layer sizes is the parameter that is experimented with, the first model being a single hidden layer network with hidden layer size = (50,) and the second being a two-hidden layer network with hidden layer size = (30,20). Activation function is set to "ReLU" and alpha (regularization parameter) is set to 0.007.

Metric	Value
Test RMSE	-1316.730587329082
Train RMSE	-1201.6367771719893

Answer 6.2

As seen above, the *negative test RMSE* is higher for neural networks in comparison to linear regression, which is obvious since neural networks are also able to capture non-linear complex relationships in the data in addition to linear relationships that could be captured by linear regression model variants.

Answer 6.3

The ReLU (Rectified Linear Unit) activation function was selected for its array of advantages over the identity activation function and other alternatives:

1. Non-linearity: ReLU introduces non-linearity into the network, facilitating the learning of intricate and non-linear relationships between input and output. This enhances the network's ability to capture underlying data patterns, thereby improving predictive accuracy.
2. Sparsity: ReLU can generate sparse representations where some output values are zero while others are non-zero. This sparsity aids in reducing the network's parameter count, enhancing computational efficiency.
3. Gradient propagation: ReLU helps mitigate the vanishing gradient problem by facilitating smoother gradient propagation throughout the network. This feature contributes to easier training of deep networks and enhances overall performance.
4. Interpretability: Compared to activation functions like sigmoid or tanh, ReLU produces output values closer to the input range, making the network's behavior more interpretable. This simplifies understanding and interpretation of the network's predictions.

In contrast, the identity activation function is a linear function that merely returns the input value. When employed, additional layers in the neural network become redundant, as all layers can be condensed into a single weight vector derived from the multiplication of weight vectors. Sigmoid and tanh functions are prone to saturation at extreme values, leading to limited learning capabilities.

Answer 6.4

Elevating the depth of a neural network past a specific threshold poses several risks, including:

1. Overfitting: Excessive complexity in a neural network may cause it to memorize training data rather than discerning general patterns applicable to new data. This phenomenon, known as overfitting, results in superior performance on training data but inferior performance on test data, particularly problematic with noisy or unrepresentative training data.
2. Vanishing gradients: Deep networks can suffer from diminutive or zero gradients during training, hindering weight updates and impeding learning. This vanishing gradient problem complicates the training process for deep networks.
3. Exploding gradients: Conversely, gradients can become excessively large, causing overly drastic weight updates during training, leading to numerical instability and hampering network training.
4. Computational complexity: Increasing network depth augments the number of parameters to be learned, heightening computational demands and slowing down the training process. This complicates scalability to larger datasets and deployment on resource-constrained devices.
5. Optimization challenges: Deeper networks pose a more intricate optimization problem, making it arduous to identify the optimal weight set that minimizes the loss function.

In essence, escalating the depth of a neural network beyond a certain threshold may yield diminishing returns or even deteriorate performance. It's crucial to strike a balance between network complexity, available data volume, and task complexity. Techniques like regularization, dropout, and batch normalization can help mitigate these risks and facilitate the training of deep neural networks.

Random Forest Regression

In [16]:

```
pipe_RF = Pipeline([
    ('model', RandomForestRegressor(random_state=42, oob_score=True,n_jobs=-1,verbose=True))
])

param_grid_RF = {
    'model__max_features': np.arange(2,5,1),
    'model__n_estimators': np.arange(1, 61, 10),
    'model__max_depth': np.arange(1, 11, 1)

}
randomforest_gs = GridSearchCV(pipe_RF, param_grid=param_grid_RF, cv=10, n_jobs=-1, verbose=10,
                               scoring='neg_root_mean_squared_error', return_train_score=True).fit(feats_fs, t
arget)
```

```
Fitting 10 folds for each of 180 candidates, totalling 1800 fits

[Parallel(n_jobs=-1)]: Using backend ThreadingBackend with 2 concurrent workers.
[Parallel(n_jobs=-1)]: Done  31 out of  31 | elapsed:    3.2s finished
```

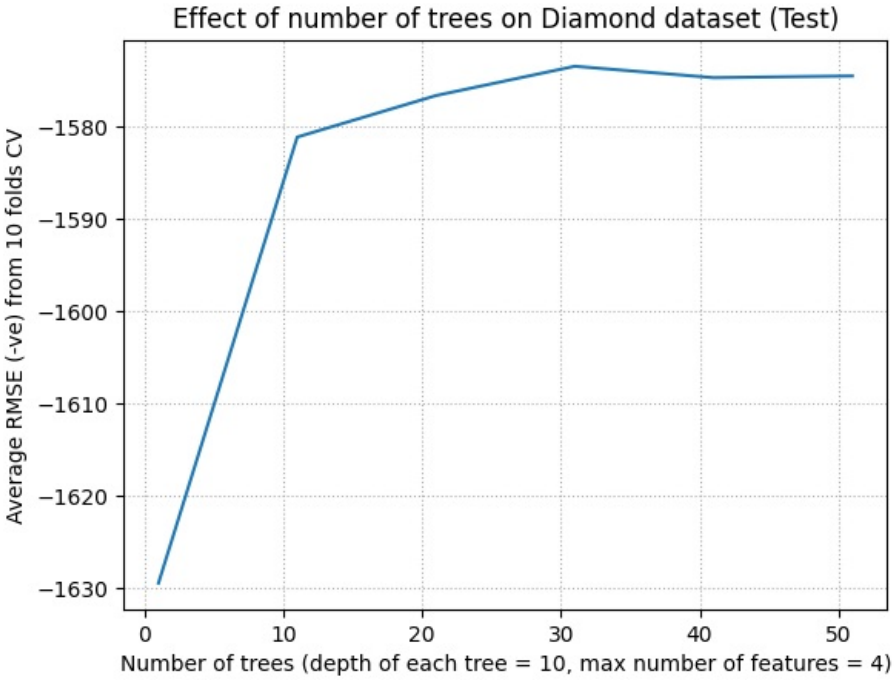
In [17]:

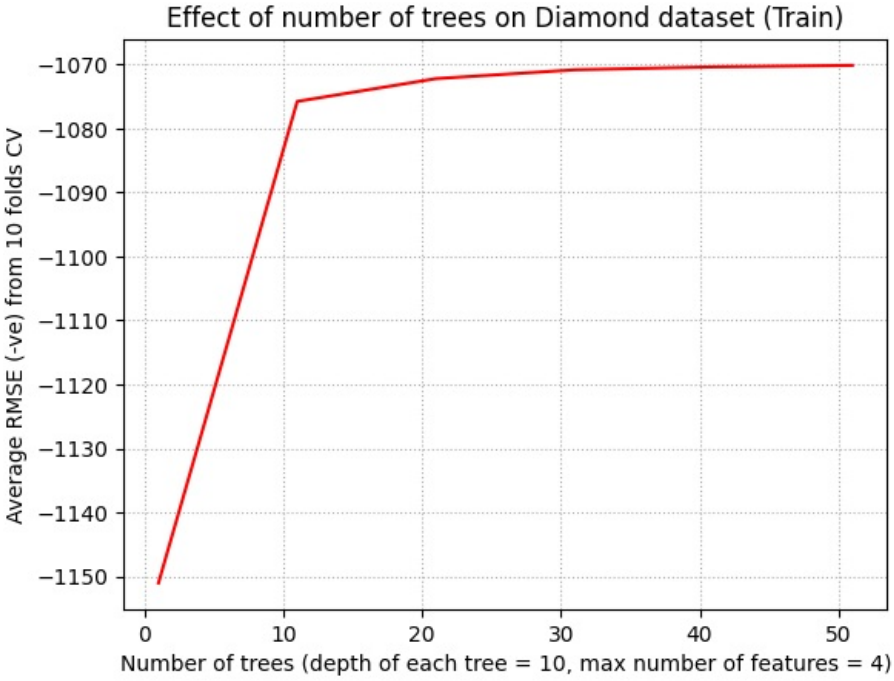
```
rf_result = pd.DataFrame(randomforest_gs.cv_results_)[['mean_test_score','mean_train_score','param_model__max_fea
tures','param_model__n_estimators','param_model__max_depth']]
print('Best parameters :',randomforest_gs.best_params_,',Test RMSE:',randomforest_gs.best_score_)
print('Train RMSE:',max(rf_result.mean_train_score))
```

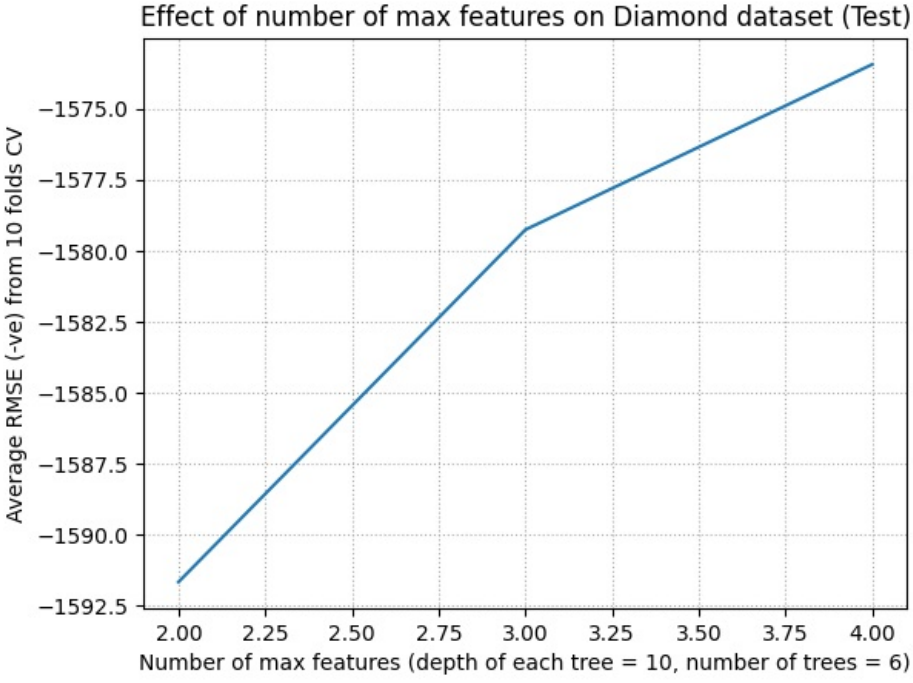
```
Best parameters : {'model__max_depth': 10, 'model__max_features': 4, 'model__n_estimators': 31} ,Tes
t RMSE: -1573.4477357508983
Train RMSE: -1070.1653334937887
```

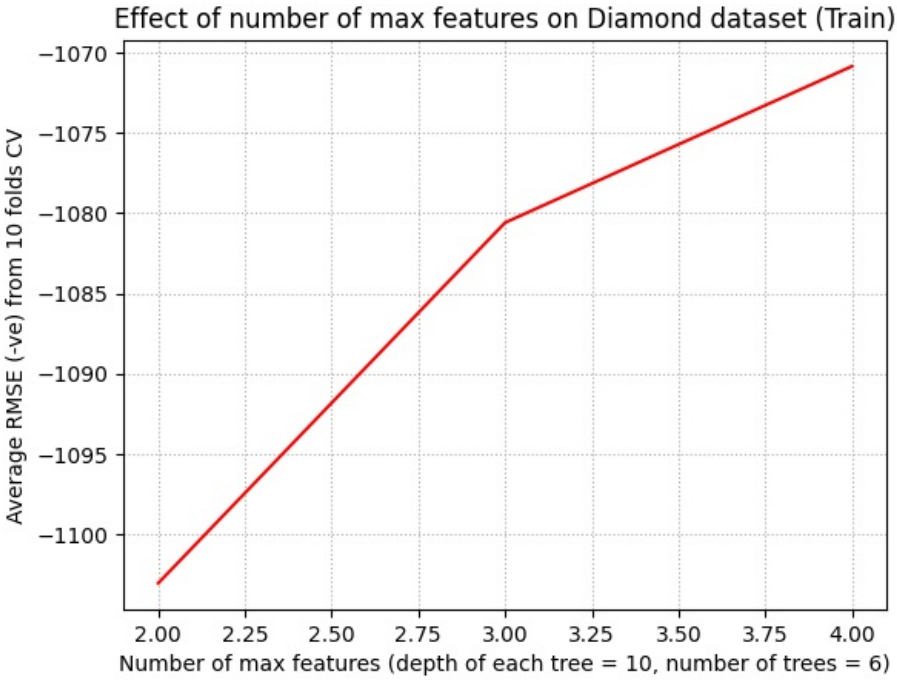
In [27]:

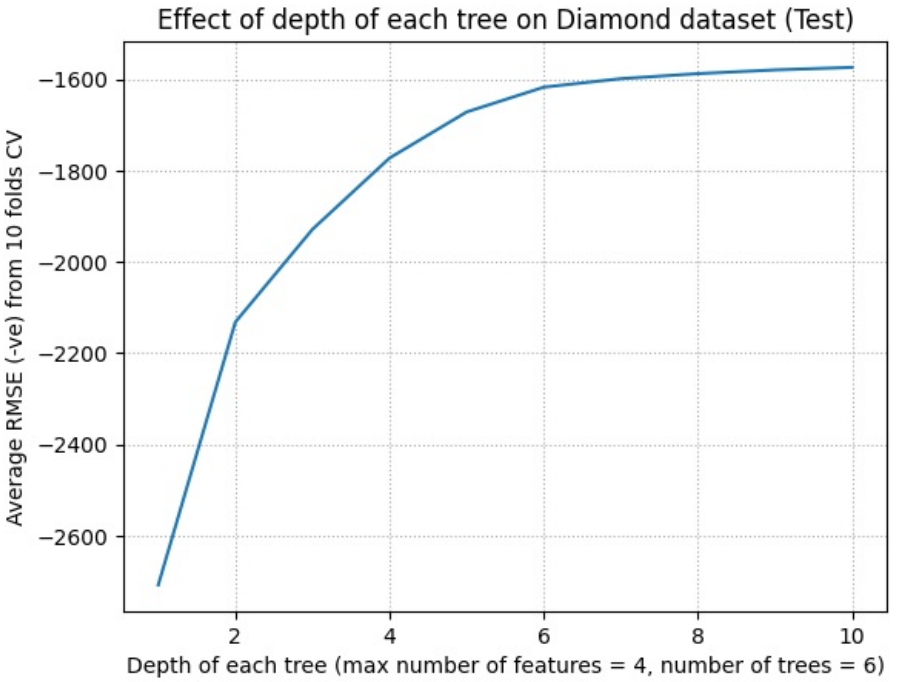
```
max_features = np.arange(2,5,1).reshape(3)
n_estimators = np.arange(1, 61, 10).reshape(6)
max_depth = np.arange(1, 11, 1).reshape(10)

diam_score = list((rf_result[(rf_result['param_model__max_depth'] == 10) & (rf_result['param_model__max_features'
] == 4)]).mean_test_score)
diam_train = list((rf_result[(rf_result['param_model__max_depth'] == 10) & (rf_result['param_model__max_features'
] == 4)]).mean_train_score)
plt.plot(n_estimators,diam_score)
plt.grid(linestyle=':')
plt.xlabel('Number of trees (depth of each tree = 10, max number of features = 4)')
plt.ylabel('Average RMSE (-ve) from 10 folds CV')
plt.title('Effect of number of trees on Diamond dataset (Test)')
plt.show()
plt.plot(n_estimators,diam_train,'r')
plt.grid(linestyle=':')
plt.xlabel('Number of trees (depth of each tree = 10, max number of features = 4)')
plt.ylabel('Average RMSE (-ve) from 10 folds CV')
plt.title('Effect of number of trees on Diamond dataset (Train)')
plt.show()

diam_score = list((rf_result[(rf_result['param_model__max_depth'] == 10) & (rf_result['param_model__n_estimators'
] == 31)]).mean_test_score)
diam_train = list((rf_result[(rf_result['param_model__max_depth'] == 10) & (rf_result['param_model__n_estimators'
] == 31)]).mean_train_score)
plt.plot(max_features,diam_score)
plt.grid(linestyle=':')
plt.xlabel('Number of max features (depth of each tree = 10, number of trees = 6)')
plt.ylabel('Average RMSE (-ve) from 10 folds CV')
plt.title('Effect of number of max features on Diamond dataset (Test)')
plt.show()
plt.plot(max_features,diam_train,'r')
plt.grid(linestyle=':')
plt.xlabel('Number of max features (depth of each tree = 10, number of trees = 6)')
plt.ylabel('Average RMSE (-ve) from 10 folds CV')
plt.title('Effect of number of max features on Diamond dataset (Train)')
plt.show()

bike_score = list((rf_result[(rf_result['param_model__max_features'] == 4) & (rf_result['param_model__n_estimator
s'] == 31)]).mean_test_score)
bike_train = list((rf_result[(rf_result['param_model__max_features'] == 4) & (rf_result['param_model__n_estimator
s'] == 31)]).mean_train_score)
plt.plot(max_depth,bike_score)
plt.grid(linestyle=':')
plt.xlabel('Depth of each tree (max number of features = 4, number of trees = 6)')
plt.ylabel('Average RMSE (-ve) from 10 folds CV')
plt.title('Effect of depth of each tree on Diamond dataset (Test)')
plt.show()
plt.plot(max_depth,bike_train,'r')
plt.grid(linestyle=':')
plt.xlabel('Depth of each tree  (max number of features = 4, number of trees = 6)')
plt.ylabel('Average RMSE (-ve) from 10 folds CV')
plt.title('Effect of depth of each tree on Diamond dataset (Train)')
plt.show()
```

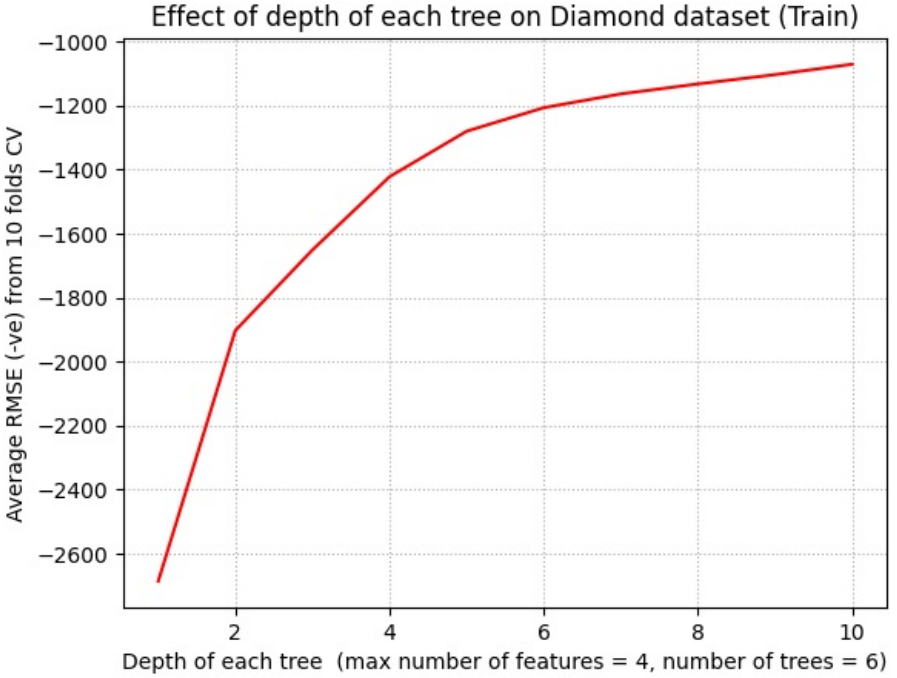

Effect of number of trees on Diamond dataset (Test)
Average RMSE (-ve) from 10 folds CV
−1580
−1590
−1600
−1610
−1620
−1630
0
10
20
30
40
50
Number of trees (depth of each tree = 10, max number of features = 4)


Effect of number of trees on Diamond dataset (Train)
Average RMSE (-ve) from 10 folds CV
−1070
−1080
−1090
−1100
−1110
−1120
−1130
−1140
−1150
0
10
20
30
40
50
Number of trees (depth of each tree = 10, max number of features = 4)


Effect of number of max features on Diamond dataset (Test)
Average RMSE (-ve) from 10 folds CV
−1575.0
−1577.5
−1580.0
−1582.5
−1585.0
−1587.5
−1590.0
−1592.5
2.00
2.25
2.50
2.75
3.00
3.25
3.50
3.75
4.00
Number of max features (depth of each tree = 10, number of trees = 6)


Effect of number of max features on Diamond dataset (Train)
Average RMSE (-ve) from 10 folds CV
−1070
−1075
−1080
−1085
−1090
−1095
−1100
2.00
2.25
2.50
2.75
3.00
3.25
3.50
3.75
4.00
Number of max features (depth of each tree = 10, number of trees = 6)


Effect of depth of each tree on Diamond dataset (Test)
Average RMSE (-ve) from 10 folds CV
−1600
−1800
−2000
−2200
−2400
−2600
2
4
6
8
10
Depth of each tree (max number of features = 4, number of trees = 6)


Effect of depth of each tree on Diamond dataset (Train)
Average RMSE (-ve) from 10 folds CV
−1000
−1200
−1400
−1600
−1800
−2000
−2200
−2400
−2600
2
4
6
8
10
Depth of each tree (max number of features = 4, number of trees = 6)

Answer 7.1

--> **Effect of the number of trees:** The quantity of trees in a random forest regressor can significantly influence its overall performance. Generally, increasing the number of trees can enhance the accuracy and stability of the model's predictions, up to a certain threshold. Insufficient trees may lead to high variance and overfitting, indicating that the model has memorized the training data excessively and might not generalize well to new data. Typically, boosting the number of trees improves the model's performance to a point of diminishing returns. Beyond this threshold, additional trees might not notably enhance performance but could increase computational expenses and training duration.

--> **Effect of the maximum number of features:** The maximum number of features utilized in each split of a Random Forest Regressor also profoundly impacts the model's overall performance. On one hand, incorporating more features can enhance the model's capability to capture intricate patterns and interactions within the data. A larger feature set offers more information to the model, facilitating better discrimination between classes or improved prediction of the target variable. Moreover, a broader range of features enhances diversity among the forest's trees, potentially bolstering performance and mitigating overfitting.

Nevertheless, excessively incorporating features can lead to overfitting, where the model learns noise instead of genuine patterns in the training data. Some features might be irrelevant or redundant, augmenting the model's complexity without commensurate performance gains. Therefore, utilizing a conservative number of features can impose a regularization effect, guarding against overfitting.

--> **Effect of tree depth:** The depth of each tree within a Random Forest Regressor significantly influences overall model performance. Deeper trees can capture intricate data relationships, resulting in enhanced accuracy and better alignment with the training data. This is because deeper trees partition the data into smaller, more homogeneous subsets, facilitating the learning of detailed patterns and feature interactions.

However, deeper trees also heighten the risk of overfitting, wherein the model learns noise instead of genuine patterns, leading to poor generalization on new data. Overfitting occurs when the model becomes excessively complex, fitting noise in the training data and resulting in high variance and suboptimal performance on unseen data.

• The performance of polynomial regression with various degrees on the top 6 and 10 selected features from the diamonds dataset was evaluated in terms of train and test RMSE.

• Prioritizing F1-Score over Mutual Information (MI) for feature selection, the top 6 and 10 features from F1 were chosen. Subsequently, ridge regression was applied to introduce L2 regularization and mitigate overfitting. The penalty parameter was optimized using grid search with 10-fold cross-validation.

• In pursuit of optimal hyperparameters, a hyperparameter space was defined, and a 10-fold cross-validation grid search pipeline was constructed. The hyperparameter space encompassed:

Hyperparameter	Range
Maximum features	2-4
Number of estimators	10-60
Depth of each tree	1-10

The optimal parameters for the diamond dataset were determined as follows:

Best Hyperparameter	Value
Maximum features	4
Number of estimators	31
Depth of each tree	10

Results for above best setting:

Metric	Value
Test RMSE	-1573.4477357508983
Train RMSE	-1070.1653334937887

Regularization effect

Based on the plots above presented, it's evident that the overall model performance doesn't exhibit a consistent trend concerning the number of trees. While increasing the number of trees tends to enhance or stabilize performance, the degree of improvement varies unpredictably. When all other hyperparameters are held constant, the sole impact of the number of trees on the model's loss is a stochastic decrease. It's advisable to select a sufficiently large value for the number of trees within computational constraints, as increasing the number of trees doesn't lead to overfitting.

• Within a random forest, each tree, along with its output target variable, behaves as independent and identically distributed random variables, as per the weak law of large numbers (WLLN). This is because the trees are grown using randomization techniques on individual bootstrap subsamples, which are uncorrelated with the growth of other trees. Consequently, WLLN suggests that both the target variable and tree-decision possess finite variance, leading to the overall decision (and any other statistic) of the random forest converging towards a mean value as the number of trees approaches infinity, as per Jensen’s inequality.

• The expected error rate for a random forest ensemble exhibits a non-monotonous relationship with the number of trees. Specifically, error metrics like RMSE become noisy once a sufficient number of estimators have been employed. The convergence rate of the error rate curve is independent of the number of trees and solely relies on the distribution of the expected value of the decisions made by the trees.

• While increasing the number of trees does not lead to overfitting in accordance with WLLN, other hyperparameters may introduce unnecessary variance in the model and foster over-correlation within the ensemble, thereby compromising the independence of trees within the random forest.

Therefore, we can observe a consistent improvement in training RMSE with the increasing maximum number of features, leading to a convergence towards a mean value. However, the test RMSE initially improves but then starts to deteriorate after reaching a certain threshold. This suggests that the number of features also acts as a form of regularization, akin to the depth of the tree. Excessively large values lead to overfitting on the training set and poor generalization on the test set. This occurs because augmenting the number of features for each tree enhances individual tree capacity but also amplifies correlation between trees, disrupting the independence of trees within the random forest. Selecting a subset of features aims to mitigate this issue by reducing correlation among trees and improving generalization error rates, thereby strengthening the random forest.

Answer 7.2

Random forests construct a complex non-linear decision boundary by amalgamating multiple decision trees, each of which employs a distinct subset of features for decision-making. Despite each decision tree applying a threshold on a single feature at each node, the collective effect of multiple trees with varied feature subsets enables the capture of intricate non-linear relationships among features.

In random forests, individual decision trees are trained on randomized subsets of both the training data and available features. This randomness serves to mitigate overfitting and enhance the diversity among the trees in the forest.

During training, each decision tree recursively partitions the data into smaller subsets based on the values of selected features at each node. By leveraging diverse feature subsets, the trees in the forest can capture various facets of the underlying relationship between input features and the target variable. The final prediction of the random forest is then derived by aggregating the predictions of all individual trees.

The ensemble of multiple decision trees with differing feature subsets facilitates the creation of a highly non-linear decision boundary, notwithstanding each tree's reliance on a single feature threshold at each node. This versatility arises from the combined utilization of diverse thresholds on different features, enabling the model to encapsulate complex interactions among the features and accurately represent non-linear relationships between input features and the target variable.

Tree visualization

In [12]:

```
vis_tree = RandomForestRegressor(random_state=42,max_depth=4, max_features=3, n_estimators=10).fit(feats_fs,targe
t)
```

In [15]:

```
chY = SelectKBest(score_func=f_regression, k=7)
XTranscode_Test = chY.fit_transform(dataenc.loc[:, dataenc.columns != 'price'], dataenc["price"])
column_names = dataenc.loc[:, dataenc.columns != 'price'].columns[chY.get_support()]
```

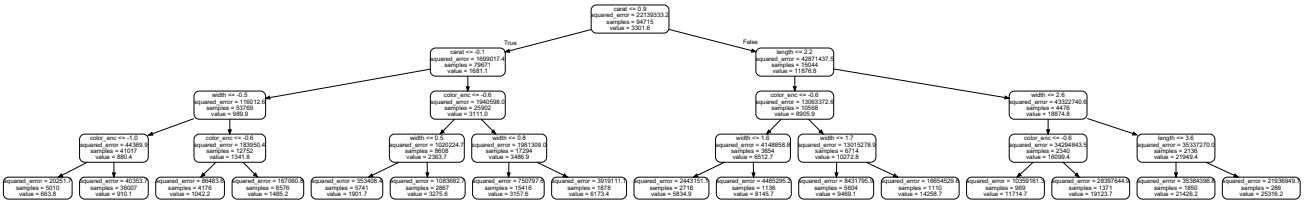
In [ ]:

```
import pydot
from IPython.display import Image
import graphviz
tree = vis_tree.estimators_[1]
export_graphviz(tree, out_file = 'tree.dot', feature_names = column_names, rounded = True, precision = 1)
(graph, ) = pydot.graph_from_dot_file('tree.dot')
Image(graph.create_png())
```

In [20]:

```
graphviz.Source.from_file('tree.dot')
```

Out[20]:



Answer 7.3

For the diamond dataset, from branching at the root node in the above figure, "price" is selected. This means that feature is the most important feature used to start the splitting process. In a decision tree, the features closer to the root node are more salient and significant than the features near the leaf nodes. The root node is carat as expected. Carat feature has the most descriptive statistics for the price as it was seen from MI and F score. From the tree we can see that the mostly used features for division are carat, length, width and color.

--> It can be concluded that the most important features in both datasets align closely with those whose p-values were deemed significant in linear regression analyses conducted previously. This observation suggests a substantial overlap between the most influential features identified based on p-values in linear regression and those determined through random decision trees within the random forest, particularly for the diamonds dataset.

Answer 7.4 Out of Bag error

In [14]:

```
print('OOB = :',RandomForestRegressor(random_state=42,max_depth=10,
                                      max_features=4, n_estimators=31, oob_score=True).fit(feats_fs,target).oo
b_score_)
```

```
OOB = : 0.9363356357319846
```

OOB Score = 0.9363 as shown above.

OOB Score: In Random Forest Regression, the Out-of-Bag (OOB) score serves as an estimate of the model's predictive accuracy, computed using samples excluded from the training process. This method involves constructing multiple decision trees by randomly selecting subsets of features and observations, aggregating their predictions to generate a final output. During tree construction, a random subset of observations is utilized for training each tree, while the remaining samples constitute the OOB set for evaluating model performance.

The OOB samples are exclusively used for assessment and not for training any decision tree. Therefore, each OOB sample can be predicted using the corresponding decision trees, enabling a comparison with the actual values. Averaging the prediction errors across all OOB samples provides an estimate of the model's generalization performance, known as the OOB score.

This metric is valuable for evaluating model performance and fine-tuning parameters, as it leverages samples not involved in training, offering a more dependable estimate of predictive accuracy on unseen data compared to conventional cross-validation methods.

R2 Score: R2, or the coefficient of determination, is a widely-used statistical metric in regression analysis, indicating the goodness of fit of a regression model to the data. R2 ranges from 0 to 1, representing the proportion of variance in the dependent variable explained by the independent variables. A value of 0 indicates the model explains none of the variability, while 1 indicates it explains all variability.

R2 provides a standardized scale for model performance and is inversely correlated with RMSE. When RMSE is 0, R2 is 1, indicating perfect fit. Conversely, negative values indicate poorer than average performance. In the case of random forests, the OOB scoring leverages R2 to assess model performance without additional data.

LightGBM

In [30]:

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

```
Collecting scikit-optimize
  Downloading scikit_optimize-0.10.1-py2.py3-none-any.whl (107 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 107.7/107.7 kB 1.2 MB/s eta 0:00:00
Requirement 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 scik
it-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
```

In [31]:

```
from skopt import BayesSearchCV
from skopt.space import Real, Categorical, Integer
from skopt.plots import plot_objective, plot_histogram

from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error,make_scorer,r2_score
from sklearn.model_selection import cross_validate
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import KFold
from sklearn.datasets import load_digits
from sklearn.pipeline import Pipeline
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, OrdinalEncoder, OneHotEncoder
from sklearn.compose import ColumnTransformer, make_column_transformer

import lightgbm as lgb

import random
np.random.seed(42)
random.seed(42)
```

In [32]:

```
feats_fs_train, feats_fs_test, target_train, target_test = train_test_split(feats_fs, target, test_size=0.1, rand
om_state=42)
```

In [33]:

```
pipe = Pipeline([
    ('model', lgb.LGBMRegressor(random_state=42))
])

lgb_search = {
    'model': Categorical([lgb.LGBMRegressor(random_state=42)]),
    'model__learning_rate': Real(0.00001, 0.9, prior='log-uniform'),
    'model__max_depth': Integer(-1,500),
    'model__num_iterations': Integer(2, 500),
    'model__num_leaves': Integer(5,500),
    'model__colsample_bytree': Real(0.05, 1.0, 'uniform'),
    'model__subsample': Real(0.05, 1.0, 'uniform'),
    'model__reg_lambda': Real(1e-8, 1e+3, 'log-uniform'),
    'model__reg_alpha': Real(1e-8, 1e+3, 'log-uniform'),
    'model__n_estimators': Integer(5,500),
    'model__min_data_in_leaf': Integer(2,200)
}

opt = BayesSearchCV(
    pipe,
    [(lgb_search)],
    cv=KFold(n_splits=10,shuffle = True, random_state = 42),
    scoring=('neg_root_mean_squared_error'),
    verbose=10,
    return_train_score=True,
    iid=False,
    n_jobs=-1
)
```

```
/usr/local/lib/python3.10/dist-packages/skopt/searchcv.py:334: UserWarning: The `iid` parameter has
been deprecated and will be ignored.
  warnings.warn(
```

Answer 8.1

Learning Rate (learning_rate): This parameter determines the rate at which the LightGBM model learns during training.

Max. Depth (max_depth): It specifies the maximum depth that each decision tree within the model can reach. Values of 0 or -1 indicate no depth limit.

No. of Iterations (num_iterations): This indicates the number of boosting iterations or training cycles.

No. of Leaves (num_leaves): It defines the maximum number of leaves allowed for each decision tree.

Subsample Ratio of Features (colsample_bytree): This parameter controls the ratio of features (columns) sampled for each tree during training.

Subsample Ratio of Training Instance (subsample): It determines the ratio of training instances sampled for each iteration.

L1 Regularization Term (reg_lambda): This term introduces L1 regularization to penalize weights in the objective function.

L2 Regularization Term (reg_alpha): This term introduces L2 regularization to penalize weights in the objective function.

No. of Estimators (n_estimators): It specifies the number of boosting trees to be fitted in the model.

Min. Data in Leaf (min_data_in_leaf): This sets the minimum number of observations required for a node to be added to the decision tree.

--> Note that the hyperparameter search space is as shown:

Hyperparameter	Range
model	Categorical
model__learning_rate	Real(0.00001, 0.9)
model__max_depth	Integer(-1, 500)
model__num_iterations	Integer(2, 500)
model__num_leaves	Integer(5, 500)
model__colsample_bytree	Real(0.05, 1.0)
model__subsample	Real(0.05, 1.0)
model__reg_lambda	Real(1e-8, 1e+3)
model__reg_alpha	Real(1e-8, 1e+3)
model__n_estimators	Integer(5, 500)
model__min_data_in_leaf	Integer(2, 200)

In [ ]:

```
opt.fit(feats_fs_train, target_train)
```

In [35]:

```
import joblib
joblib.dump(opt, 'rf_bayes_search.pkl')
bayes_search = joblib.load('rf_bayes_search.pkl')
df_b = pd.DataFrame(bayes_search.cv_results_)
sorted_b = df_b.sort_values(by='rank_test_score', ascending=True)
sorted_b.iloc[:10]
```

Out[35]:

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_model	param_model__colsar
24	5.958891	0.661478	0.903919	0.167350	LGBMRegressor(learning_rate=0.1024916400047666...	
11	3.123369	0.588430	0.275703	0.072837	LGBMRegressor(learning_rate=0.1024916400047666...	
9	3.300038	0.376784	0.277451	0.063314	LGBMRegressor(learning_rate=0.1024916400047666...	
44	3.163803	0.555316	0.258982	0.046034	LGBMRegressor(learning_rate=0.1024916400047666...	
29	4.672231	1.338226	0.425339	0.104324	LGBMRegressor(learning_rate=0.1024916400047666...	
43	3.371541	0.088237	0.349363	0.060690	LGBMRegressor(learning_rate=0.1024916400047666...	
34	2.942527	0.504418	0.262924	0.056317	LGBMRegressor(learning_rate=0.1024916400047666...	
20	3.643166	1.056137	0.344657	0.084514	LGBMRegressor(learning_rate=0.1024916400047666...	
49	8.165066	0.710763	1.221800	0.237543	LGBMRegressor(learning_rate=0.1024916400047666...	
27	15.568985	3.628823	2.606781	0.047594	LGBMRegressor(learning_rate=0.1024916400047666...	

10 rows × 42 columns

In [36]:

```
preds = opt.predict(feats_fs_test)
r2 = r2_score(target_test, preds)
rmse = mean_squared_error(target_test, preds, squared=False)
print("Train RMSE for best model: " , -sorted_b.iloc[0, -3])
print("Best Avg. Val. RMSE: ", -opt.best_score_)
print("Test RMSE: " , rmse)
print("Test R2: " ,r2)
```

```
[LightGBM] [Warning] min_data_in_leaf is set=10, min_child_samples=20 will be ignored. Current value
: min_data_in_leaf=10
Train RMSE for best model:  1117.6553011733251
Best Avg. Val. RMSE:  1178.2321468577388
Test RMSE:  1182.5691497796877
Test R2:  0.9366571654669723
```

In [37]:

```
opt.best_params_
```

Out[37]:

```
OrderedDict([('model',
              LGBMRegressor(learning_rate=0.10249164000476667, max_depth=10,
                            min_data_in_leaf=10, n_estimators=5, num_iterations=400,
                            num_leaves=353, random_state=42, reg_alpha=1000.0,
                            reg_lambda=677.5730894730251, subsample=0.752794294518232)),
             ('model__colsample_bytree', 1.0),
             ('model__learning_rate', 0.10249164000476667),
             ('model__max_depth', 10),
             ('model__min_data_in_leaf', 10),
             ('model__n_estimators', 5),
             ('model__num_iterations', 400),
             ('model__num_leaves', 353),
             ('model__reg_alpha', 1000.0),
             ('model__reg_lambda', 677.5730894730251),
             ('model__subsample', 0.752794294518232)])
```

Answer 8.2

Search results:

Metric	Value
Train RMSE	1117.6553011733251
Best Avg. Val. RMSE	1178.2321468577388
Test RMSE	1182.5691497796877
Test R2	0.9366571654669723

The below table shows the optimal values of hyperparameters from the best model.

Best Hyperparameter found	Value
learning_rate	0.10249164000476667
max_depth	10
min_data_in_leaf	10
n_estimators	5
num_iterations	400
num_leaves	353
reg_alpha	1000.0
reg_lambda	677.5730894730251
subsample	0.752794294518232
colsample_bytree (model)	1.0
learning_rate (model)	0.10249164000476667
max_depth (model)	10
min_data_in_leaf (model)	10
n_estimators (model)	5
num_iterations (model)	400
num_leaves (model)	353
reg_alpha (model)	1000.0
reg_lambda (model)	677.5730894730251
subsample (model)	0.752794294518232

Answer 8.3

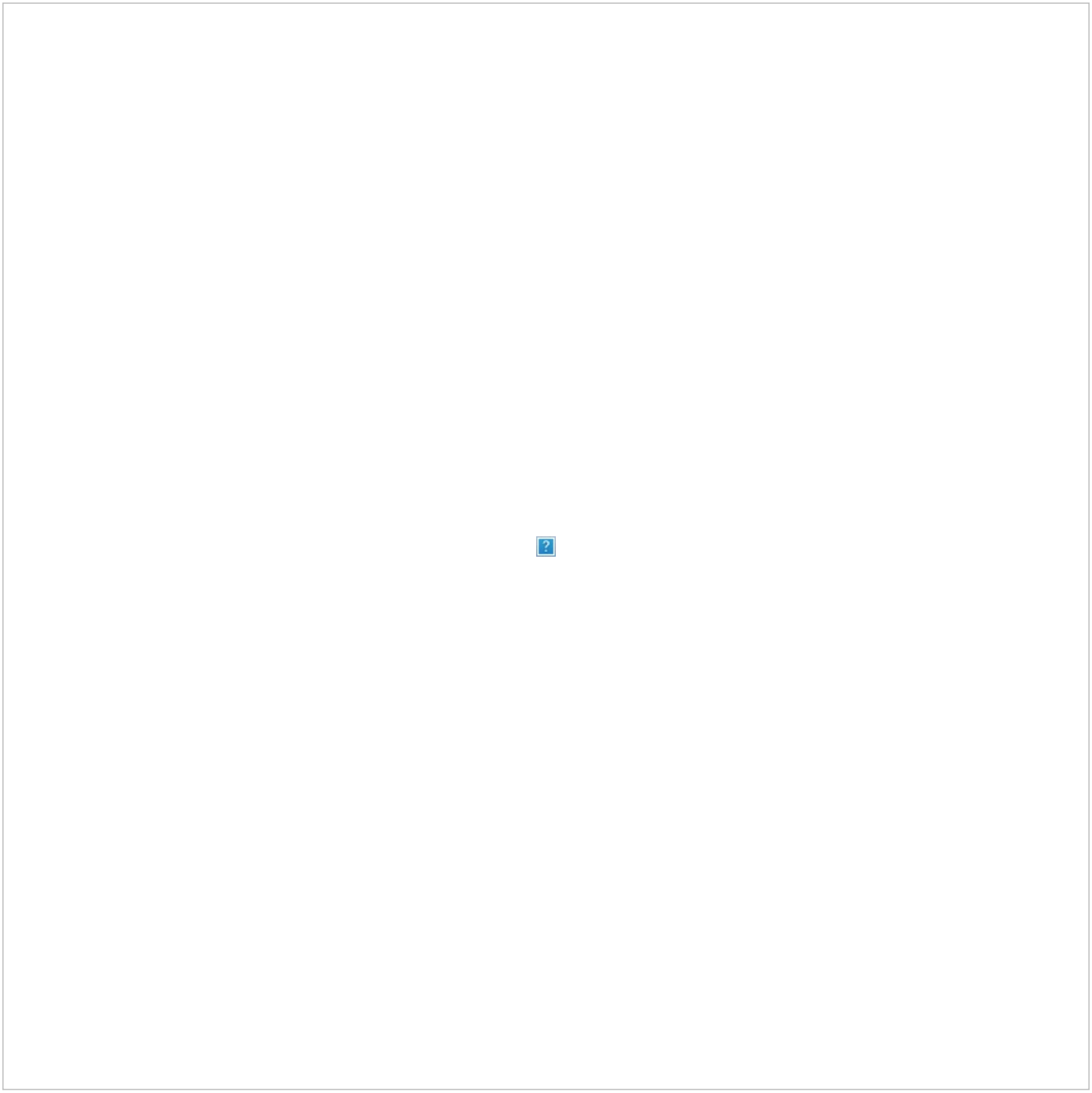
Explanations are given in cell outputs as below

In [38]:

```
import seaborn as sns
hparam_list = [
       'param_model__colsample_bytree',
       'param_model__learning_rate',
       'param_model__max_depth',
       'param_model__min_data_in_leaf',
       'param_model__n_estimators',
       'param_model__num_iterations',
       'param_model__num_leaves',
       'param_model__reg_alpha',
       'param_model__reg_lambda',
       'param_model__subsample',
       'mean_test_score',
       'mean_fit_time',
       'mean_train_score',
       'mean_score_time']
df_b_perf = df_b[hparam_list].apply(lambda x: pd.to_numeric(x), axis=1)
df_b_perf = df_b_perf.rename(columns={'mean_test_score': 'mean_val_neg_rmse'})
df_b_perf = df_b_perf.rename(columns={'mean_train_score': 'mean_train_neg_rmse'})
plt.figure(figsize=(15,15))
#sns.heatmap(df_b_perf.corr(method='pearson'), cbar=True, annot=True)
sns.heatmap( df_b_perf.corr(method='pearson').loc[:, ['mean_val_neg_rmse']].sort_values(by='mean_val_neg_rmse', a
scending=False),
             cbar=True,
             annot=True )
```

Out[38]:

<Axes: >



To measure hyperparameters effect on the performance, I calculated the Pearson correlation coefficient for negative average validation RMSE with respect to each hyperparameter.

In above figure, I display correlation coefficients sorted by performance using RMSE. Notably, the subsample ratio of features (columns), number of iterations, learning rate, L2 regularization term, L1 regularization term, and max. depth hyperparameters have significant impacts on validation performance, in that order. Among these, the subsample ratio of features, number of iterations, and learning rate exert particularly strong influences.

Moreover, I observe a positive correlation between the subsample ratio of features and validation performance. Similarly, the number of training iterations, learning rate, and L2 regularization also demonstrate a strong positive relationship with validation performance.

An interesting finding is the occurrence of a very large L2 regularization term value. This behavior can be explained by examining other hyperparameters such as a high number of iterations (500) and leaves (500). The complexity introduced by these high values prompts the model to utilize high L2 regularization.

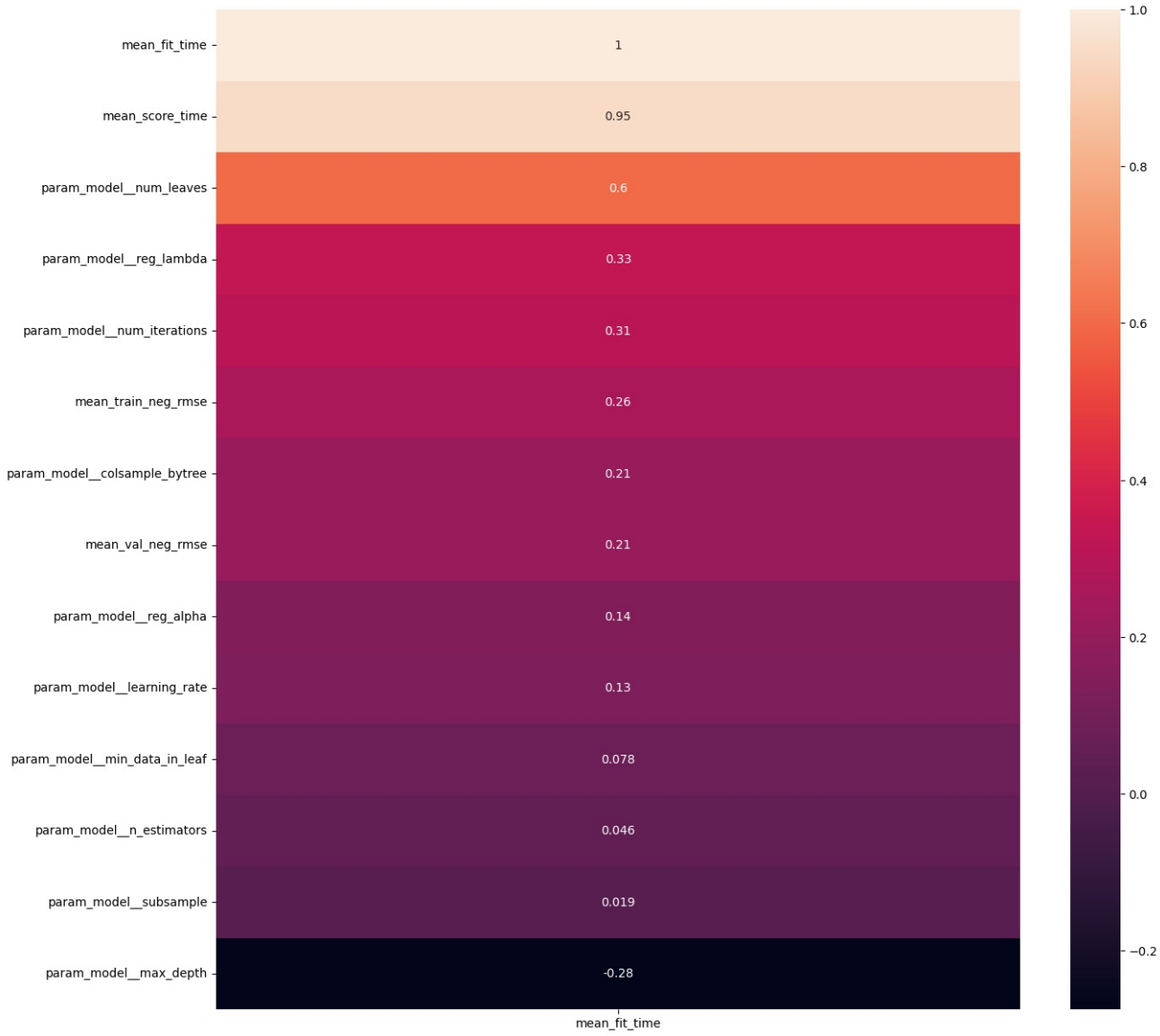
Conversely, the L1 regularization term and max. depth exhibit a negative relationship with validation performance.

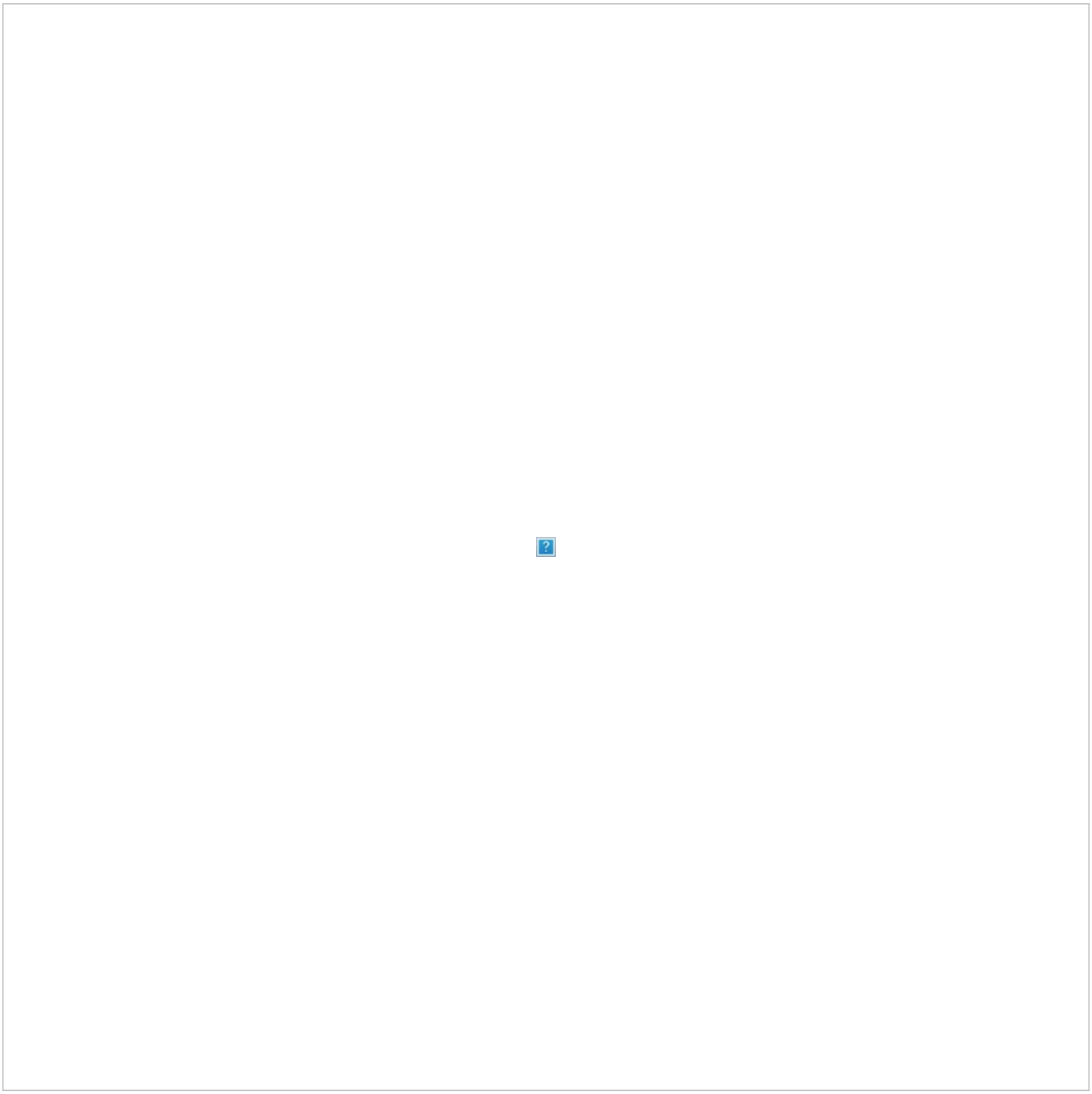
In [39]:

```
plt.figure(figsize=(15,15))
sns.heatmap( df_b_perf.corr(method='pearson').loc[:, ['mean_fit_time']].sort_values(by='mean_fit_time', ascending
=False),
            cbar=True,
            annot=True )
```

Out[39]:

<Axes: >





Above figure illustrates the correlation between the absolute difference in average training and validation RMSEs, serving as a measure of regularization or generalization gap effects. The negative correlation coefficients indicate that increasing L1 and L2 regularization terms enhances generalization. Likewise, raising the minimum number of observations required for a leaf node improves generalization.

While reducing the number of estimators and max. depth typically correlates with better generalization, the associated correlation coefficients lack significance. This could be due to a relatively limited subspace search, where these hyperparameters may be offset by others with opposing effects.

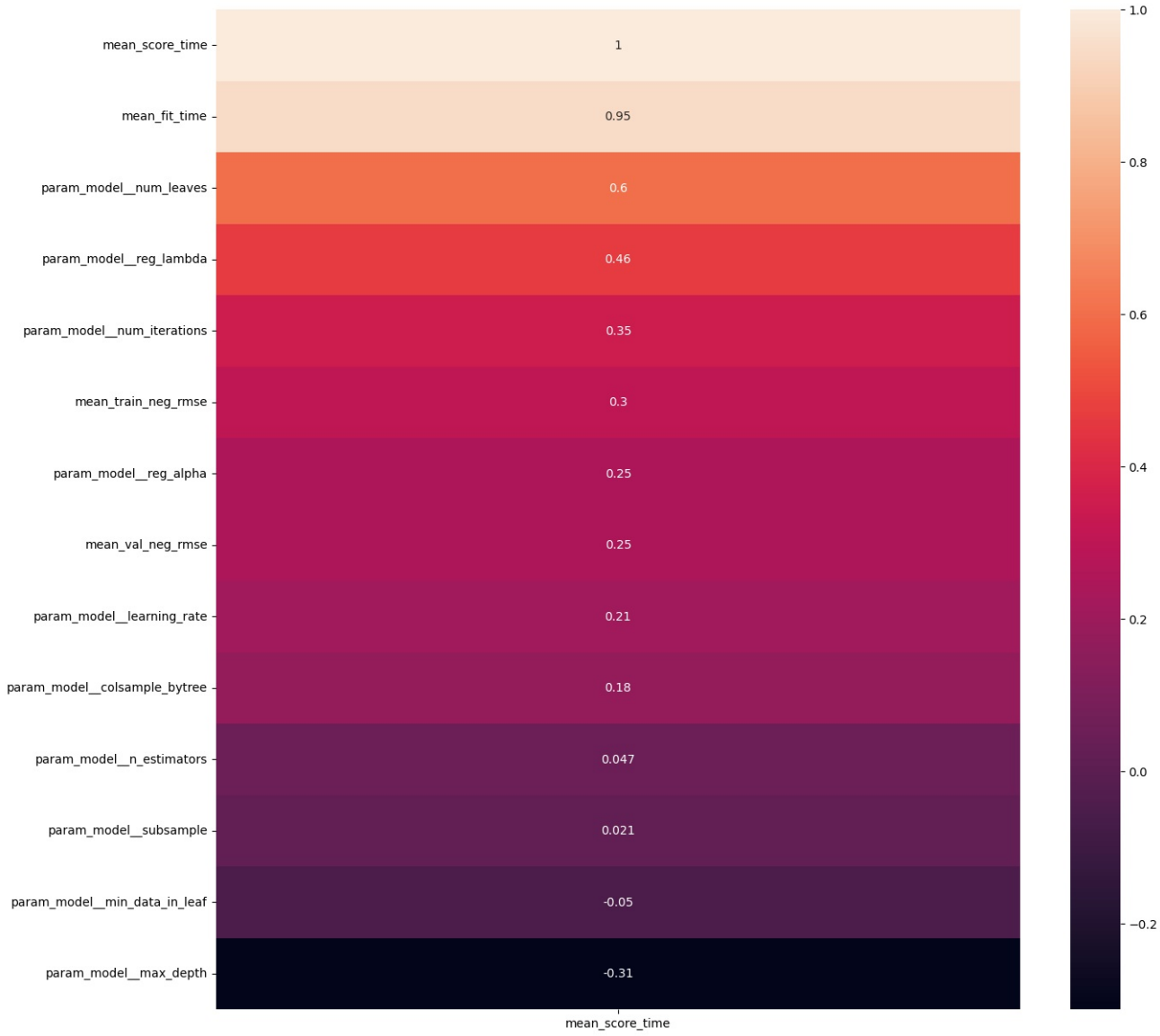
Moreover, increasing the number of training iterations, subsample ratio of features (columns), and number of leaves substantially decreases generalization, as expected, given their propensity to increase complexity. Similarly, raising the learning rate diminishes generalization, as excessively high rates may lead to overfitting.

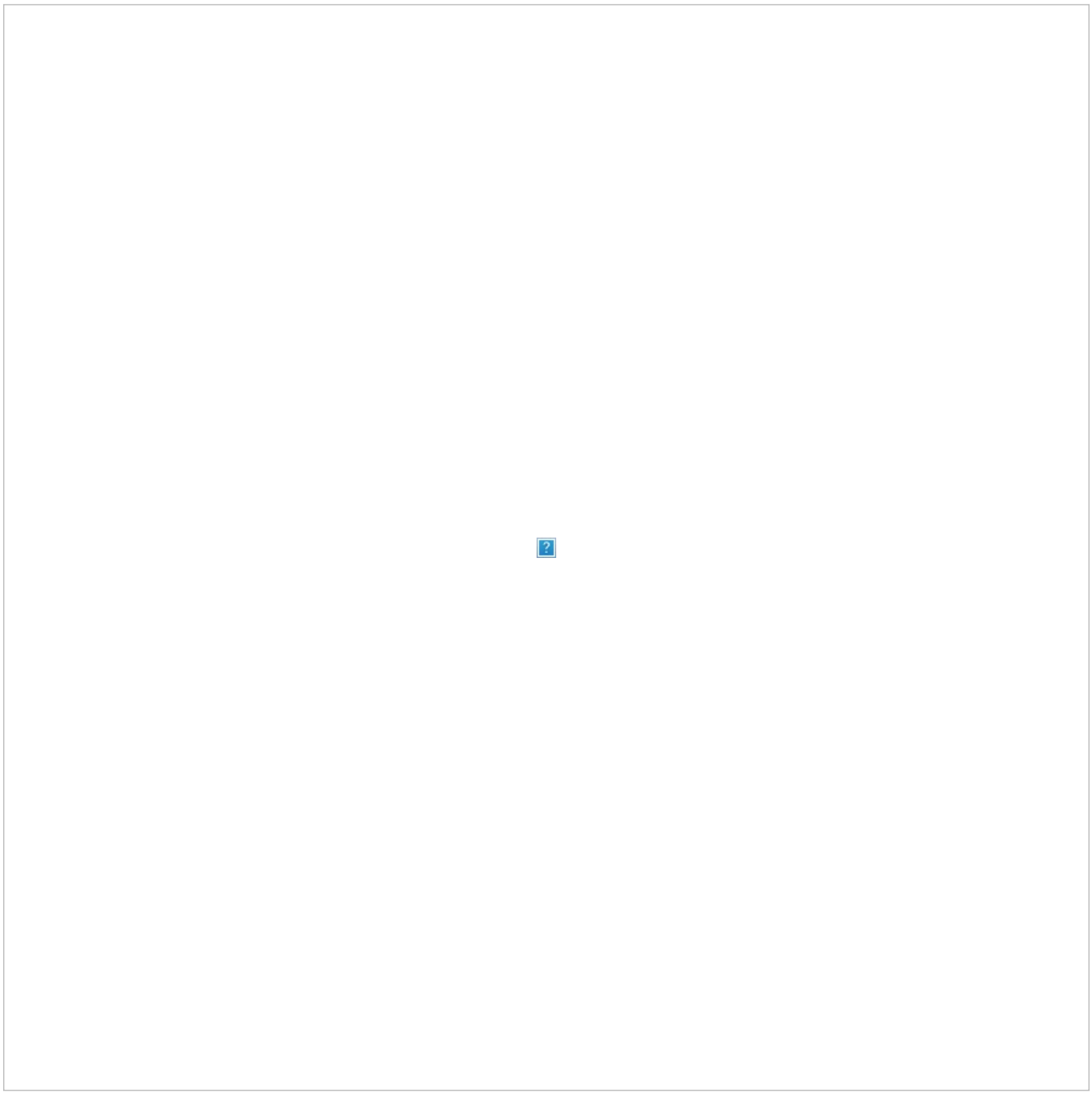
In [40]:

```
plt.figure(figsize=(15,15))
sns.heatmap( df_b_perf.corr(method='pearson').loc[:, ['mean_score_time']].sort_values(by='mean_score_time', ascen
ding=False),
            cbar=True,
            annot=True )
```

Out[40]:

<Axes: >





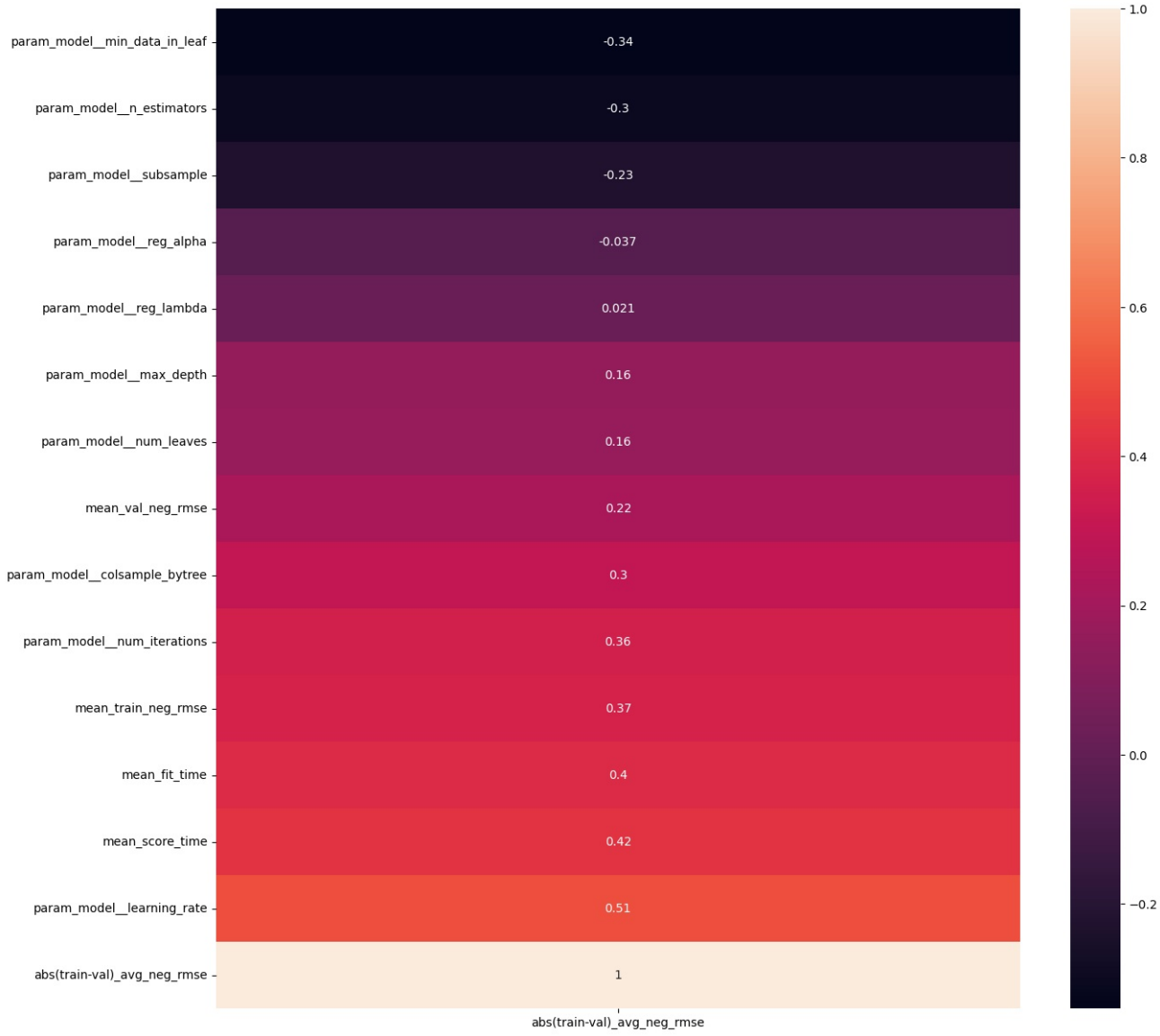
As depicted in above figure, I assessed the impact of hyperparameters on model training efficiency by computing sorted Pearson correlation coefficients for the average fit time relative to the hyperparameters.

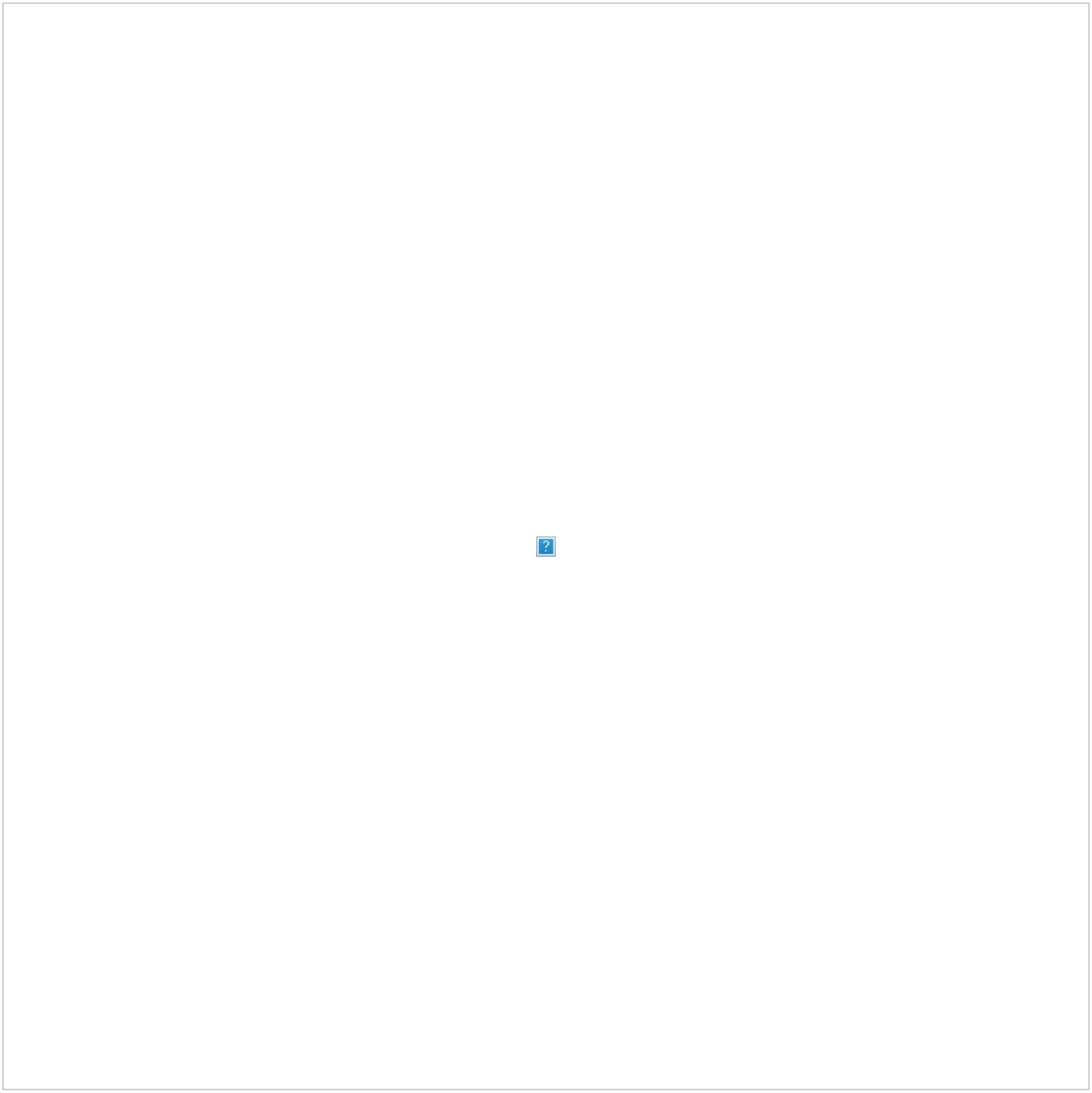
In [41]:

```
df_b_perf['abs(train-val)_avg_neg_rmse'] = np.abs(df_b_perf['mean_train_neg_rmse'] - df_b_perf['mean_val_neg_rmse
'])
plt.figure(figsize=(15,15))
sns.heatmap( df_b_perf.corr(method='pearson').loc[:, ['abs(train-val)_avg_neg_rmse']].sort_values(by='abs(train-v
al)_avg_neg_rmse', ascending=True),
            cbar=True,
            annot=True )
```

Out[41]:

<Axes: >





Similarly, as illustrated in above figure, I gauged the impact of hyperparameters on model evaluation efficiency by computing sorted Pearson correlation coefficients for the average score time relative to the hyperparameters.

It is noted that increasing the number of leaves, subsample ratio of features (columns), and the number of training iterations significantly prolongs the average fit time, consequently diminishing efficiency.

Although elevating the values of the number of estimators and max. depth typically correlates with reduced efficiency, the associated correlation coefficients lack significance. Once again, this observation might be attributed to a relatively limited subspace search, wherein these hyperparameters may be counterbalanced by others with opposing effects.

Furthermore, augmenting the L1 regularization term and the minimum number of observations required for a leaf node decreases the average fit time.

Name : Vignesh Nagarajan (UID: 606185377)

In [ ]:

```
%cd '/content/drive/MyDrive/Twitter_project'
```

```
/content/drive/MyDrive/Twitter_project
```

In [1]:

```
import json
import numpy as np
import statsmodels.api as sm
from sklearn import metrics

def report_statistics(filename):
    with open(filename, 'r') as file:
        lines = file.readlines()
        max_time = 0
        min_time = np.inf
        total_followers = 0
        total_retweets = 0
        total_tweets = len(lines)
        for line in lines:
            json_obj = json.loads(line)
            if json_obj['citation_date'] > max_time:
                max_time = json_obj['citation_date']
            if json_obj['citation_date'] < min_time:
                min_time = json_obj['citation_date']
            total_followers += json_obj['author']['followers']
            total_retweets += json_obj['metrics']['citations']['total']
        avg_tweets_per_h = total_tweets * 3600 / (max_time - min_time)
        avg_followers_per_tweet = total_followers / total_tweets
        avg_retweets_per_tweet = total_retweets / total_tweets
        print(filename)
        print('Average number of tweets per hour: ', avg_tweets_per_h)
        print('Average number of followers of users posting the tweets per tweet: ', avg_followers_per_tweet)
        print('Average number of retweets per tweet: ', avg_retweets_per_tweet)
        print('-' * 50)
```

In [4]:

```
files = ['/content/drive/MyDrive/Twitter_project/tweets/tweets_#gohawks.txt', '/content/drive/MyDrive/Twitter_project/tweets/tweets_#gopatriots.txt',
         '/content/drive/MyDrive/Twitter_project/tweets/tweets_#nfl.txt', '/content/drive/MyDrive/Twitter_project/tweets/tweets_#patriots.txt',
         '/content/drive/MyDrive/Twitter_project/tweets/tweets_#sb49.txt', '/content/drive/MyDrive/Twitter_project/tweets/tweets_#superbowl.txt']

for file in files:
    report_statistics(file)
```

```
/content/drive/MyDrive/Twitter_project/tweets/tweets_#gohawks.txt
Average number of tweets per hour:  292.48785062173687
Average number of followers of users posting the tweets per tweet:  2217.9237355281984
Average number of retweets per tweet:  2.0132093991319877
--------------------------------------------------
/content/drive/MyDrive/Twitter_project/tweets/tweets_#gopatriots.txt
Average number of tweets per hour:  40.954698006061946
Average number of followers of users posting the tweets per tweet:  1427.2526051635405
Average number of retweets per tweet:  1.4081919101697078
--------------------------------------------------
/content/drive/MyDrive/Twitter_project/tweets/tweets_#nfl.txt
Average number of tweets per hour:  397.0213901819841
Average number of followers of users posting the tweets per tweet:  4662.37544523693
Average number of retweets per tweet:  1.5344602655543254
--------------------------------------------------
/content/drive/MyDrive/Twitter_project/tweets/tweets_#patriots.txt
Average number of tweets per hour:  750.8942646068899
Average number of followers of users posting the tweets per tweet:  3280.4635616550277
Average number of retweets per tweet:  1.7852871288476946
--------------------------------------------------
/content/drive/MyDrive/Twitter_project/tweets/tweets_#sb49.txt
Average number of tweets per hour:  1276.8570598680474
Average number of followers of users posting the tweets per tweet:  10374.160292019487
Average number of retweets per tweet:  2.52713444111402
--------------------------------------------------
/content/drive/MyDrive/Twitter_project/tweets/tweets_#superbowl.txt
Average number of tweets per hour:  2072.11840170408
Average number of followers of users posting the tweets per tweet:  8814.96799424623
Average number of retweets per tweet:  2.3911895819207736
--------------------------------------------------
```

Answer 9.1

Hashtag	Tweets per Hour	Followers per Tweet	Retweets per Tweet
#gohawks	292.49	2217.92	2.01
#gopatriots	40.95	1427.25	1.41
#nfl	397.02	4662.38	1.53
#patriots	750.89	3280.46	1.79
#sb49	1276.86	10374.16	2.53
#superbowl	2072.12	8814.97	2.39

The data indicates that #sb49 and #superbowl consistently exhibit higher values across all three metrics compared to the other hashtags. This could be attributed to their broader appeal, as they are associated with the overall event rather than specific teams. Conversely, hashtags like #gohawks, #gopatriots, and #patriots may attract a smaller audience consisting mainly of supporters of those particular teams.

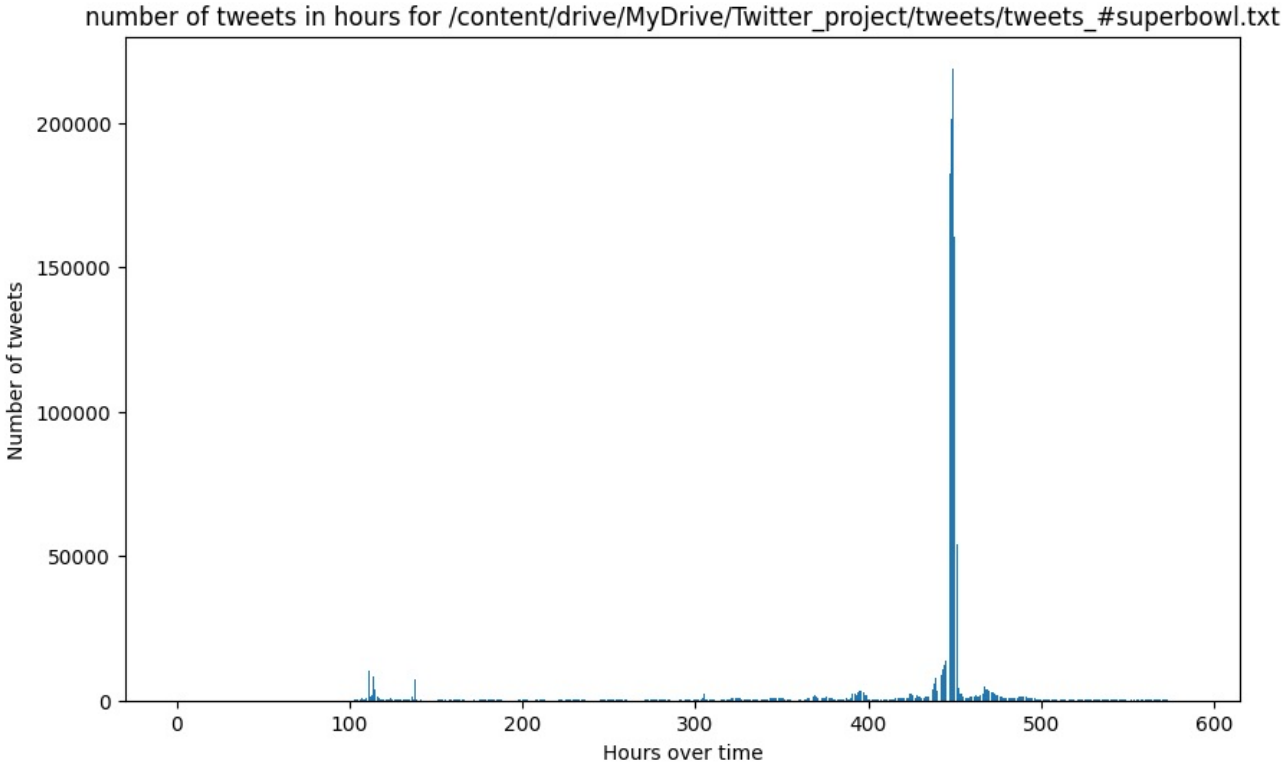
Furthermore, the popularity of #gohawks likely exceeded that of #gopatriots due to the Seattle Seahawks' status as the defending champions. However, the eventual victory of the New England Patriots may have caused a surge in the usage of the #patriots hashtag while diminishing the popularity of #gohawks.

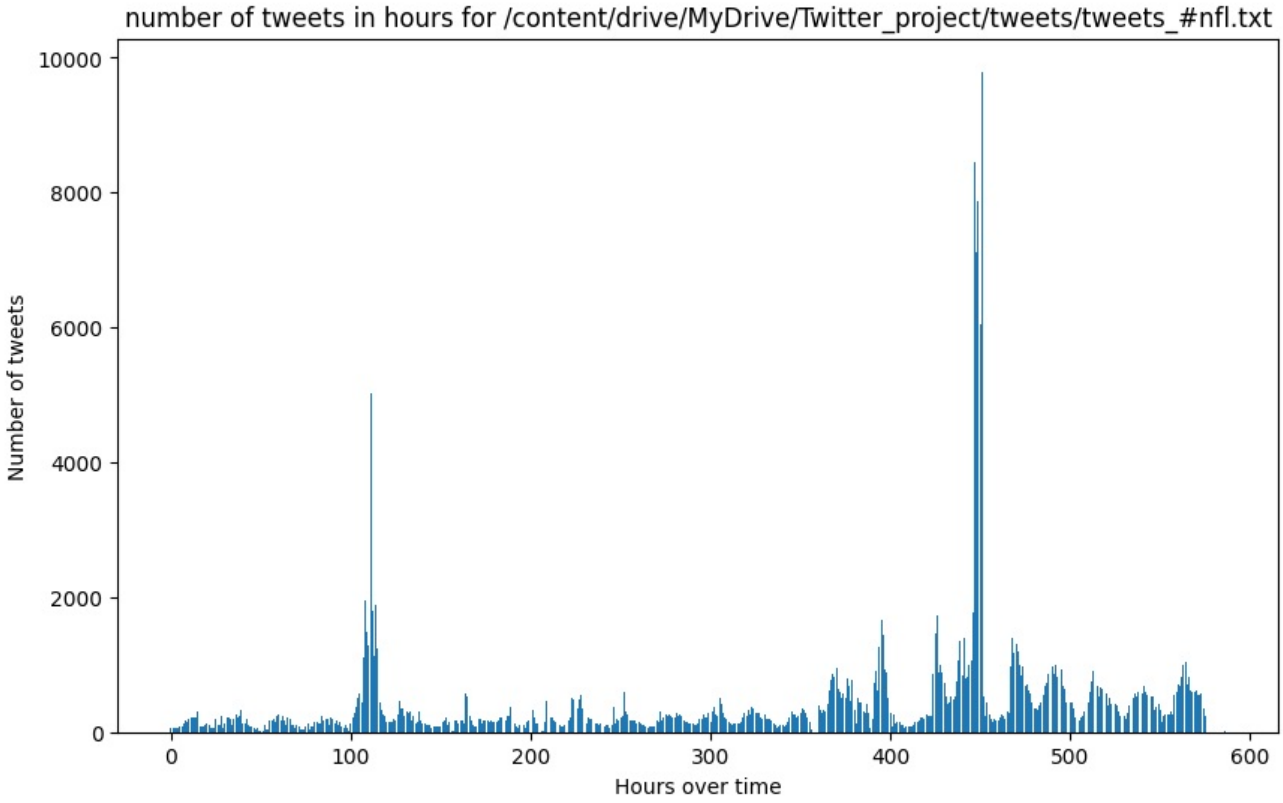
In [8]:

```
import math
import matplotlib.pyplot as plt
import datetime
import pytz

pst_tz = pytz.timezone('America/Los_Angeles')

def report_tweets(filename):
    with open(filename, 'r') as file:
        lines = file.readlines()
        max_time = 0
        min_time = np.inf
        total_tweets = len(lines)
        for line in lines:
            json_obj = json.loads(line)
            if json_obj['citation_date'] > max_time:
                max_time = json_obj['citation_date']
            if json_obj['citation_date'] < min_time:
                min_time = json_obj['citation_date']

        total_hours = math.ceil((max_time - min_time) / 3600)
        n_tweets = [0] * total_hours
        for line in lines:
            json_obj = json.loads(line)
            index = math.floor((json_obj['citation_date'] - min_time) / 3600)
            n_tweets[index] += 1
        return n_tweets
```

In [ ]:

```
from matplotlib import pyplot as plt
plotdata = ['/content/drive/MyDrive/Twitter_project/tweets/tweets_#superbowl.txt','/content/drive/MyDrive/Twitter
_project/tweets/tweets_#nfl.txt']

for file in plotdata:
    n_tweets = report_tweets(file)
    plt.figure(figsize=(10,6))
    plt.bar(range(len(n_tweets)),n_tweets)
    plt.xlabel('Hours over time')
    plt.ylabel('Number of tweets')
    plt.title('number of tweets in hours for '+file)
```





Answer 9.2

Plots are as shown above

Answer 10

- Data preprocessing steps were conducted prior to feature selection and prediction tasks:
  - Tweets were extracted from the dataset using the JSON object reference json_obj['tweet']['text'].
  - Labels for each tweet were created as they were not originally present in the dataset. These labels indicate which team's fan the tweet is associated with.
  - Regular expressions were utilized to find all hashtags in the tweet, which were then scanned to determine if they matched the patterns for "hawks" or "patriots."
  - One-hot label encoding was applied, where [0,1] indicated a tweet from a Patriots fan and [1,0] indicated a tweet from a Hawks fan.
  - The "#" symbol was removed from the tweet text before feature selection to ensure that words associated with hashtags were considered by the feature selection models.
  - Tweets with labels belonging to both Patriots and Hawks, or neither, were removed from the dataset, resulting in a dataset size of 158153 tweets.
  - The dataset was split into training and testing sets using a test size of 0.2.
- Feature selection was performed using methods learned in previous projects:
  - Each tweet in the dataset was cleaned by removing HTML artifacts and then passed through a lemmatization function utilizing POS tagging.
  - A TfidfVectorizer(stop words='english', min_df=3) was fitted to the lemmatized data features, assigning weights to each word in the dataset and building a model.
  - To reduce the number of features, the top 50 features were selected using SVD decomposition with the TruncatedSVD() function.
  - The selected features were projected onto both the training and testing sets, resulting in final shapes of (126522, 50) for x train and (31631, 50) for x test.
- Team classification was performed using two models:
  - The first model utilized LogisticRegression() with parameter grid search conducted using GridSearchCV() for a 4-fold cross-validation. Parameters included 'L2' penalty and no penalty, with C values ranging from 0.01 to 100. The best model found had parameters C=0.01 and penalty='none'.
  - The second model used RandomForestClassifier() with a parameter grid search performed, varying max depth from 10 to 200. The best model from the grid search had a max depth of 50.

In [9]:

```
import math, datetime
def report_features(filename):
    with open(filename, 'r') as file:
        lines = file.readlines()
        max_time = 0
        min_time = np.inf
        total_tweets = len(lines)
        total_followers = 0
        total_retweets = 0
        for line in lines:
            json_obj = json.loads(line)
            if json_obj['citation_date'] > max_time:
                max_time = json_obj['citation_date']
            if json_obj['citation_date'] < min_time:
                min_time = json_obj['citation_date']
            total_followers += json_obj['author']['followers']
            total_retweets += json_obj['metrics']['citations']['total']

        total_hours = math.ceil((max_time - min_time) / 3600)
        #initialize features
        features = np.zeros((total_hours,5))
        for hour in range(total_hours):
            features[hour][4] = datetime.datetime.fromtimestamp((min_time + hour * 3600), pst_tz).hour
        for line in lines:
            json_obj = json.loads(line)
            index = math.floor((json_obj['citation_date'] - min_time) / 3600)
            features[index][0] += 1
            features[index][1] += json_obj['metrics']['citations']['total']
            features[index][2] += json_obj['author']['followers']
            features[index][3] = max(features[index][3], json_obj['author']['followers'])

    return features




for file in files:
    features = report_features(file)
    x = features[:-1,:]
    y_true = features[1:,0]

    lr_fit = sm.OLS(y_true,x).fit()
    y_pred = lr_fit.predict()
    print('Hashtag: ' + file)
    print('MSE: ', metrics.mean_squared_error(y_true, y_pred))
    print(lr_fit.summary())
    print('\n')
```

```
Hashtag: /content/drive/MyDrive/Twitter_project/tweets/tweets_#gohawks.txt
MSE:  717636.4421300612
                                 OLS Regression Results
=======================================================================================
Dep. Variable:                      y   R-squared (uncentered):                   0.528
Model:                            OLS   Adj. R-squared (uncentered):              0.524
Method:                 Least Squares   F-statistic:                              128.2
Date:                Mon, 18 Mar 2024   Prob (F-statistic):                    5.72e-91
Time:                        03:28:31   Log-Likelihood:                         -4716.9
No. Observations:                 578   AIC:                                      9444.
Df Residuals:                     573   BIC:                                      9466.
Df Model:                           5
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
x1             1.5364      0.161      9.566      0.000       1.221       1.852
x2            -0.1744      0.040     -4.414      0.000      -0.252      -0.097
x3            -0.0003   7.78e-05     -3.905      0.000      -0.000      -0.000
x4             0.0004      0.000      2.276      0.023    4.87e-05       0.001
x5             5.5039      2.897      1.900      0.058      -0.187      11.195
==============================================================================
Omnibus:                      896.948   Durbin-Watson:                   2.225
Prob(Omnibus):                  0.000   Jarque-Bera (JB):           817586.157
Skew:                           8.264   Prob(JB):                         0.00
Kurtosis:                     186.508   Cond. No.                     2.20e+05
==============================================================================

Notes:
[1] R² is computed without centering (uncentered) since the model does not contain a constant.
[2] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[3] The condition number is large, 2.2e+05. This might indicate that there are
strong multicollinearity or other numerical problems.
```

Hashtag: /content/drive/MyDrive/Twitter_project/tweets/tweets_#gopatriots.txt
MSE: 30607.578574064657

OLS Regression Results

Dep. Variable:	y	R-squared (uncentered):	0.606
Model:	OLS	Adj. R-squared (uncentered):	0.602
Method:	Least Squares	F-statistic:	174.7
Date:	Mon, 18 Mar 2024	Prob (F-statistic):	1.90e-112
Time:	03:28:34	Log-Likelihood:	-3778.9
No. Observations:	574	AIC:	7568.
Df Residuals:	569	BIC:	7590.
Df Model:	5		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
x1	-0.1425	0.288	-0.495	0.621	-0.708	0.423
x2	0.6196	0.201	3.087	0.002	0.225	1.014
x3	0.0002	0.000	0.767	0.443	-0.000	0.001
x4	-0.0003	0.000	-1.385	0.167	-0.001	0.000
x5	0.4429	0.566	0.782	0.434	-0.669	1.555

Omnibus:	428.376	Durbin-Watson:	2.013
Prob(Omnibus):	0.000	Jarque-Bera (JB):	332741.978
Skew:	1.903	Prob(JB):	0.00
Kurtosis:	120.890	Cond. No.	3.00e+04

Notes:
[1] R² is computed without centering (uncentered) since the model does not contain a constant.
[2] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[3] The condition number is large, 3e+04. This might indicate that there are
strong multicollinearity or other numerical problems.

Hashtag: /content/drive/MyDrive/Twitter_project/tweets/tweets_#nfl.txt
MSE: 274661.9677320641

OLS Regression Results

Dep. Variable:	y	R-squared (uncentered):	0.651
Model:	OLS	Adj. R-squared (uncentered):	0.648
Method:	Least Squares	F-statistic:	216.7
Date:	Mon, 18 Mar 2024	Prob (F-statistic):	3.16e-130
Time:	03:29:07	Log-Likelihood:	-4500.8
No. Observations:	586	AIC:	9012.
Df Residuals:	581	BIC:	9034.
Df Model:	5		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
x1	0.6449	0.136	4.736	0.000	0.377	0.912
x2	-0.1748	0.066	-2.654	0.008	-0.304	-0.045
x3	0.0001	2.49e-05	4.049	0.000	5.19e-05	0.000
x4	-9.726e-05	3.27e-05	-2.971	0.003	-0.000	-3.3e-05
x5	7.5391	1.967	3.833	0.000	3.676	11.402

Omnibus:	614.482	Durbin-Watson:	2.366
Prob(Omnibus):	0.000	Jarque-Bera (JB):	342827.399
Skew:	3.861	Prob(JB):	0.00
Kurtosis:	121.242	Cond. No.	3.90e+05

Notes:
[1] R² is computed without centering (uncentered) since the model does not contain a constant.
[2] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[3] The condition number is large, 3.9e+05. This might indicate that there are
strong multicollinearity or other numerical problems.

Hashtag: /content/drive/MyDrive/Twitter_project/tweets/tweets_#patriots.txt
MSE: 4581686.32283954

OLS Regression Results

Dep. Variable:	y	R-squared (uncentered):	0.715
Model:	OLS	Adj. R-squared (uncentered):	0.712
Method:	Least Squares	F-statistic:	290.9
Date:	Mon, 18 Mar 2024	Prob (F-statistic):	1.55e-155
Time:	03:30:09	Log-Likelihood:	-5325.4
No. Observations:	586	AIC:	1.066e+04
Df Residuals:	581	BIC:	1.068e+04

```
Df Model:                           5
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
x1             1.2195      0.079     15.391      0.000       1.064       1.375
x2            -0.3437      0.069     -4.994      0.000      -0.479      -0.209
x3          4.251e-05   2.69e-05      1.581      0.115   -1.03e-05    9.53e-05
x4          6.516e-05   8.62e-05      0.756      0.450      -0.000       0.000
x5             9.3970      7.372      1.275      0.203      -5.081      23.875
==============================================================================
Omnibus:                     1011.154   Durbin-Watson:                   1.958
Prob(Omnibus):                  0.000   Jarque-Bera (JB):           944705.008
Skew:                          10.427   Prob(JB):                         0.00
Kurtosis:                     198.592   Cond. No.                     6.81e+05
==============================================================================

Notes:
[1] R² is computed without centering (uncentered) since the model does not contain a constant.
[2] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[3] The condition number is large, 6.81e+05. This might indicate that there are
strong multicollinearity or other numerical problems.


Hashtag: /content/drive/MyDrive/Twitter_project/tweets/tweets_#sb49.txt
MSE:  13170997.953818945
                                 OLS Regression Results
==============================================================================
Dep. Variable:                      y   R-squared (uncentered):                   0.842
Model:                            OLS   Adj. R-squared (uncentered):              0.840
Method:                 Least Squares   F-statistic:                              613.3
Date:                Mon, 18 Mar 2024   Prob (F-statistic):                   3.58e-228
Time:                        03:31:43   Log-Likelihood:                         -5596.3
No. Observations:                 582   AIC:                                  1.120e+04
Df Residuals:                     577   BIC:                                  1.122e+04
Df Model:                           5
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
x1             1.1630      0.089     13.020      0.000       0.988       1.338
x2            -0.1791      0.081     -2.224      0.027      -0.337      -0.021
x3          1.162e-05   1.27e-05      0.912      0.362   -1.34e-05    3.66e-05
x4             0.0002   3.96e-05      4.439      0.000    9.79e-05       0.000
x5           -14.8322     12.369     -1.199      0.231     -39.125       9.461
==============================================================================
Omnibus:                      945.647   Durbin-Watson:                   1.398
Prob(Omnibus):                  0.000   Jarque-Bera (JB):           680692.869
Skew:                           9.257   Prob(JB):                         0.00
Kurtosis:                     169.514   Cond. No.                     6.37e+06
==============================================================================

Notes:
[1] R² is computed without centering (uncentered) since the model does not contain a constant.
[2] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[3] The condition number is large, 6.37e+06. This might indicate that there are
strong multicollinearity or other numerical problems.


Hashtag: /content/drive/MyDrive/Twitter_project/tweets/tweets_#superbowl.txt
MSE:  34137549.2586825
                                 OLS Regression Results
==============================================================================
Dep. Variable:                      y   R-squared (uncentered):                   0.870
Model:                            OLS   Adj. R-squared (uncentered):              0.869
Method:                 Least Squares   F-statistic:                              777.5
Date:                Mon, 18 Mar 2024   Prob (F-statistic):                   2.25e-254
Time:                        03:34:32   Log-Likelihood:                         -5903.8
No. Observations:                 585   AIC:                                  1.182e+04
Df Residuals:                     580   BIC:                                  1.184e+04
Df Model:                           5
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
x1             2.6639      0.111     24.048      0.000       2.446       2.881
x2            -0.1808      0.037     -4.847      0.000      -0.254      -0.108
x3            -0.0002   1.13e-05    -20.110      0.000      -0.000      -0.000
x4             0.0011   9.56e-05     11.127      0.000       0.001       0.001
x5           -54.0652     21.452     -2.520      0.012     -96.198     -11.932
==============================================================================
Omnibus:                     1106.718   Durbin-Watson:                   1.837
Prob(Omnibus):                  0.000   Jarque-Bera (JB):          1715405.711
```

```
Skew:                          12.531   Prob(JB):                         0.00
Kurtosis:                     267.098   Cond. No.                     9.57e+06
==============================================================================

Notes:
[1] R² is computed without centering (uncentered) since the model does not contain a constant.
[2] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[3] The condition number is large, 9.57e+06. This might indicate that there are
strong multicollinearity or other numerical problems.
```

In [10]:

```python
def get_time_interval(filename):
    with open(filename, 'r') as file:
        lines = file.readlines()
        max_time = 0
        min_time = np.inf
        for line in lines:
            json_obj = json.loads(line)
            if json_obj['citation_date'] > max_time:
                max_time = json_obj['citation_date']
            if json_obj['citation_date'] < min_time:
                min_time = json_obj['citation_date']
        return max_time, min_time

max_t = []
min_t = []
for file in files:
    max_time, min_time = get_time_interval(file)
    max_t.append(max_time)
    min_t.append(min_time)

max_time_agg = min(max_t)
min_time_agg = max(min_t)
```

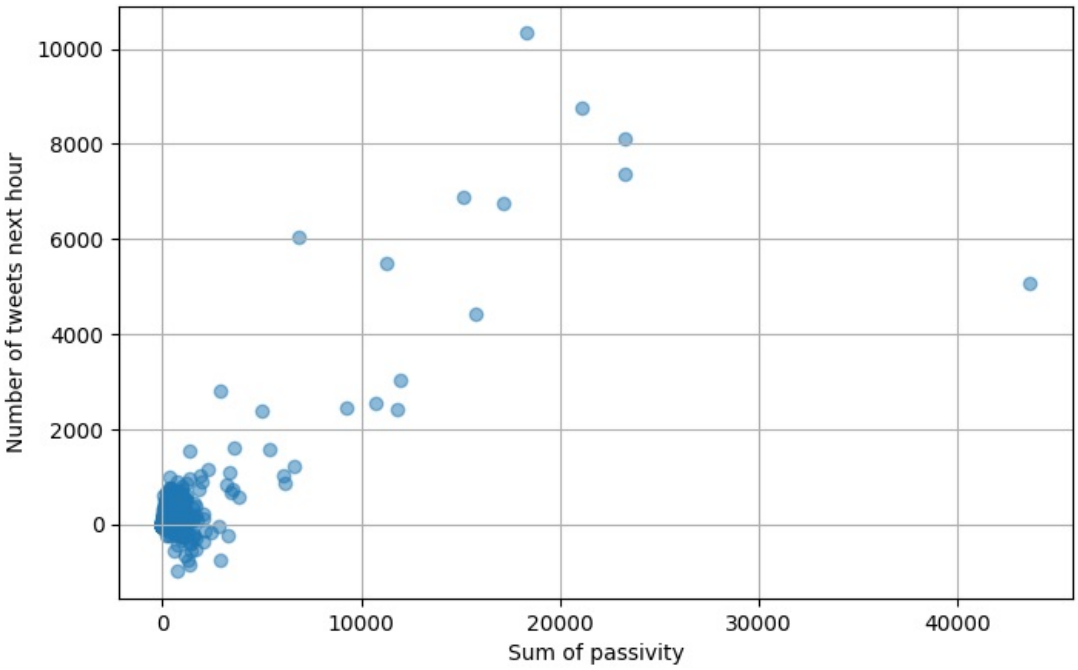
In [11]:

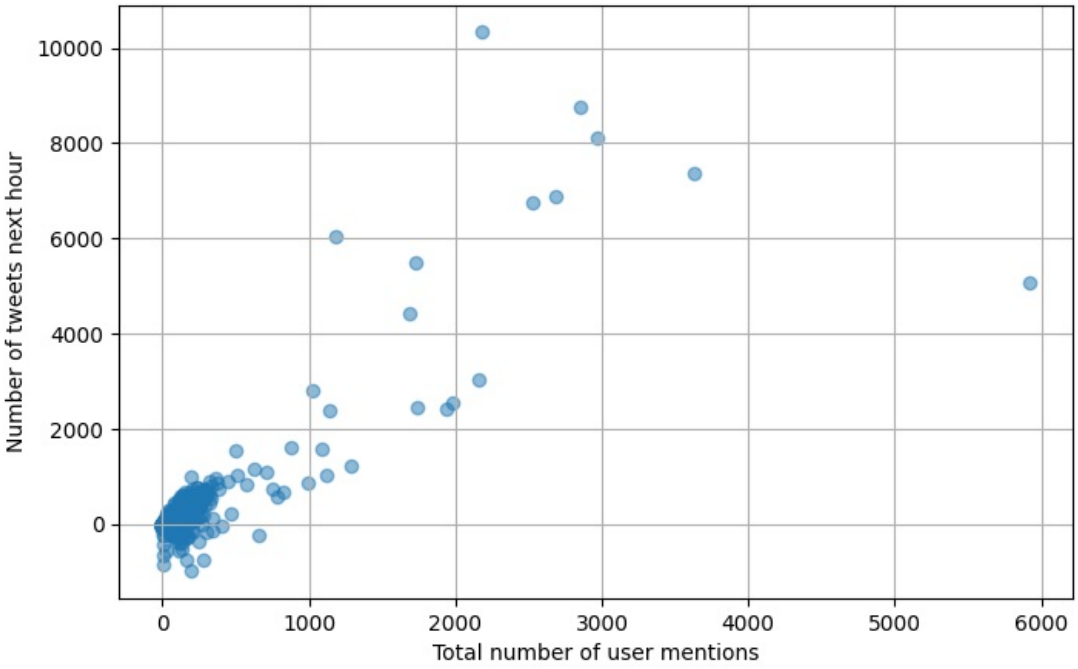
```
feature_names = ['Number of tweets', 'Total number of retweets', 'Sum of the number of followers',
                 'Maximum number of followers', 'Time of the day', 'Sum of ranking score',
                 'Sum of passivity', 'Total number of unique users','Total number of unique authors',
                 'Total number of user mentions']

mnth_to_int = {'Jan': 1, 'Feb': 2, 'Mar': 3, 'Apr': 4, 'May': 5, 'Jun': 6,
               'Jul': 7, 'Aug': 8, 'Sep': 9, 'Oct': 10, 'Nov': 11, 'Dec': 12}

def get_days(user_create_time, tweet_create_time):
    user_create_date = user_create_time.split(' ')
    tweet_create_date = tweet_create_time.split(' ')
    user_create_date = datetime.datetime(int(user_create_date[-1]), mnth_to_int[user_create_date[1]], int(user_cr
eate_date[2]))
    tweet_create_date = datetime.datetime(int(tweet_create_date[-1]), mnth_to_int[tweet_create_date[1]], int(twee
t_create_date[2]))
    created_days = tweet_create_date - user_create_date
    created_days = created_days.days
    return created_days

def report_features2(filename, min_time, max_time):
    with open(filename, 'r') as file:
        lines = file.readlines()

        total_hours = math.ceil((max_time - min_time) / 3600)
        user_ids = {hour:set() for hour in range(total_hours)}
        author_nicks = {hour:set() for hour in range(total_hours)}

        features = np.zeros((total_hours,len(feature_names)))

        for hour in range(total_hours):
            features[hour][4] = datetime.datetime.fromtimestamp((min_time + hour * 3600), pst_tz).hour

        for line in lines:
            json_obj = json.loads(line)

            if json_obj['citation_date'] >= min_time and json_obj['citation_date'] <= max_time:
                index = math.floor((json_obj['citation_date'] - min_time) / 3600)
                features[index][0] += 1 #number of tweets
                features[index][1] += json_obj['metrics']['citations']['total']
                features[index][2] += json_obj['author']['followers']
                features[index][3] = max(features[index][3], json_obj['author']['followers'])

                features[index][5] += json_obj['metrics']['ranking_score']
                n_days = get_days(json_obj['tweet']['user']['created_at'], json_obj['tweet']['created_at'])
                features[index][6] += n_days/(1.0 + json_obj['tweet']['user']['statuses_count'])
                if json_obj['tweet']['user']['id'] not in user_ids[index]:
                    user_ids[index].add(json_obj['tweet']['user']['id'])
                features[index][7] = len(user_ids[index])
                if json_obj['author']['nick'] not in author_nicks[index]:
                    author_nicks[index].add(json_obj['author']['nick'])
                features[index][8] = len(author_nicks[index])
                features[index][9] += len(json_obj['tweet']['entities']['user_mentions'])

        return features
def scatter_plot(features, hashtag, y_pred, pvalues, feature_names):
    ranked_index = np.argsort(pvalues)
    print('Hashtag: ' + hashtag)
    for i in range(3):
        plt.figure(figsize = (8,5))
        plt.scatter(features[:,ranked_index[i]], y_pred, alpha=0.5)
        plt.xlabel(feature_names[ranked_index[i]])
        plt.ylabel("Number of tweets next hour")
        plt.grid(True)
        plt.show()
    print('-' * 80)
```

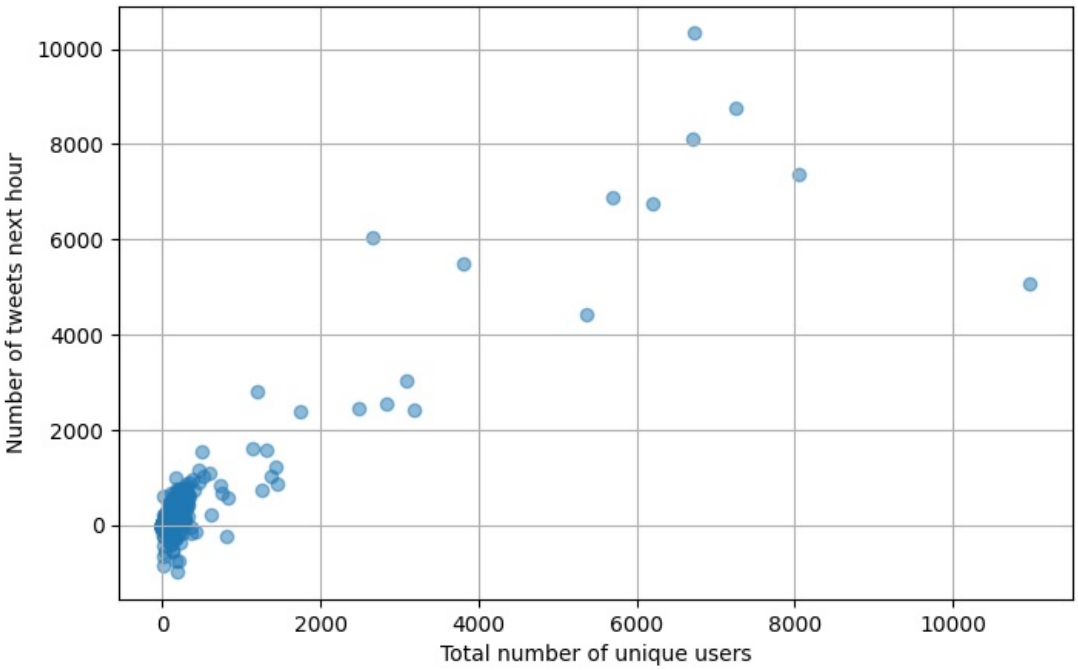
In [12]:

```
import statsmodels.api as sm
from sklearn import metrics


for file in files:
    features = report_features2(file, min_time_agg, max_time_agg)
    x = features[:-1,:] #training features
    y_true = features[1:,0] #true labels

    lr_fit = sm.OLS(y_true,x).fit()
    y_pred = lr_fit.predict()
    pvalues = lr_fit.pvalues
    print('Hashtag: ' + file)
    print('MSE: ', metrics.mean_squared_error(y_true, y_pred))
    print(lr_fit.summary())
    scatter_plot(x, file, y_pred, pvalues, feature_names)
    print('\n')
```

```
Hashtag: /content/drive/MyDrive/Twitter_project/tweets/tweets_#gohawks.txt
MSE:  296042.31731156004
                                 OLS Regression Results
=======================================================================================
Dep. Variable:                          y   R-squared (uncentered):                   0.783
Model:                                OLS   Adj. R-squared (uncentered):              0.779
Method:                     Least Squares   F-statistic:                              201.8
Date:                    Mon, 18 Mar 2024   Prob (F-statistic):                   1.43e-178
Time:                            03:38:11   Log-Likelihood:                         -4407.0
No. Observations:                     571   AIC:                                      8834.
Df Residuals:                         561   BIC:                                      8878.
Df Model:                              10
Covariance Type:                nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
x1            -0.0227      4.277     -0.005      0.996      -8.424       8.379
x2            -0.0854      0.030     -2.851      0.005      -0.144      -0.027
x3         -4.578e-05   7.74e-05     -0.591      0.555      -0.000       0.000
x4             0.0002      0.000      1.469      0.143   -6.04e-05       0.000
x5            -2.5729      2.078     -1.238      0.216      -6.655       1.509
x6             0.7652      0.832      0.920      0.358      -0.869       2.399
x7            -0.6288      0.055    -11.412      0.000      -0.737      -0.521
x8           -99.6988     17.115     -5.825      0.000    -133.315     -66.082
x9            97.3100     17.047      5.708      0.000      63.826     130.794
x10            3.0031      0.357      8.421      0.000       2.303       3.704
==============================================================================
Omnibus:                      631.309   Durbin-Watson:                   2.047
Prob(Omnibus):                  0.000   Jarque-Bera (JB):            98754.441
Skew:                           4.748   Prob(JB):                         0.00
Kurtosis:                      66.723   Cond. No.                     2.59e+06
==============================================================================

Notes:
[1] R² is computed without centering (uncentered) since the model does not contain a constant.
[2] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[3] The condition number is large, 2.59e+06. This might indicate that there are
strong multicollinearity or other numerical problems.
Hashtag: /content/drive/MyDrive/Twitter_project/tweets/tweets_#gohawks.txt
```


Number of tweets next hour
10000
8000
6000
4000
2000
0
0
10000
20000
30000
40000
Sum of passivity


Number of tweets next hour
10000
8000
6000
4000
2000
0
0
1000
2000
3000
4000
5000
6000
Total number of user mentions


Number of tweets next hour
10000
8000
6000
4000
2000
0
0
2000
4000
6000
8000
10000
Total number of unique users

Hashtag: /content/drive/MyDrive/Twitter_project/tweets/tweets_#gopatriots.txt
MSE: 21727.945699825843

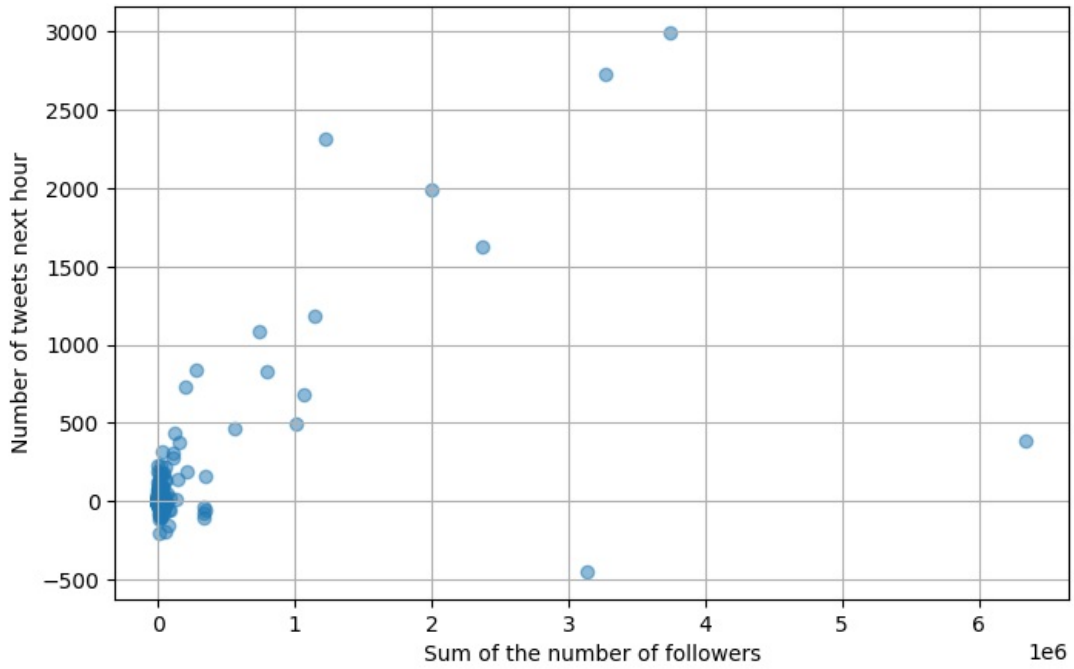
OLS Regression Results

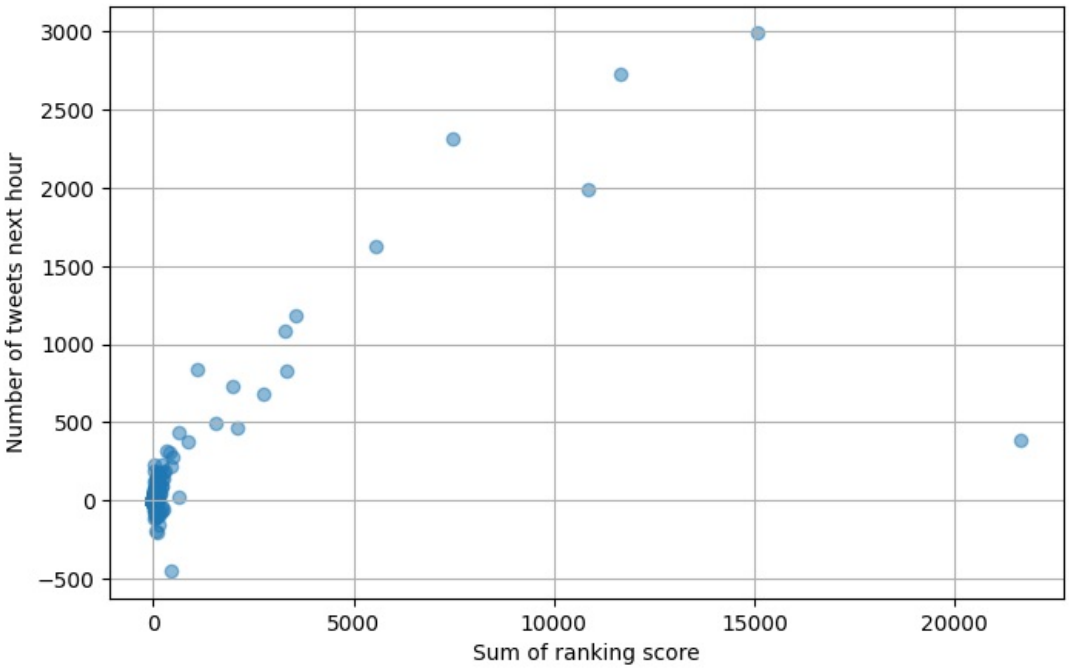
Dep. Variable:	y	R-squared (uncentered):	0.742
Model:	OLS	Adj. R-squared (uncentered):	0.738
Method:	Least Squares	F-statistic:	161.6
Date:	Mon, 18 Mar 2024	Prob (F-statistic):	5.56e-158
Time:	03:38:15	Log-Likelihood:	-3661.3
No. Observations:	571	AIC:	7343.
Df Residuals:	561	BIC:	7386.
Df Model:	10		
Covariance Type:	nonrobust		

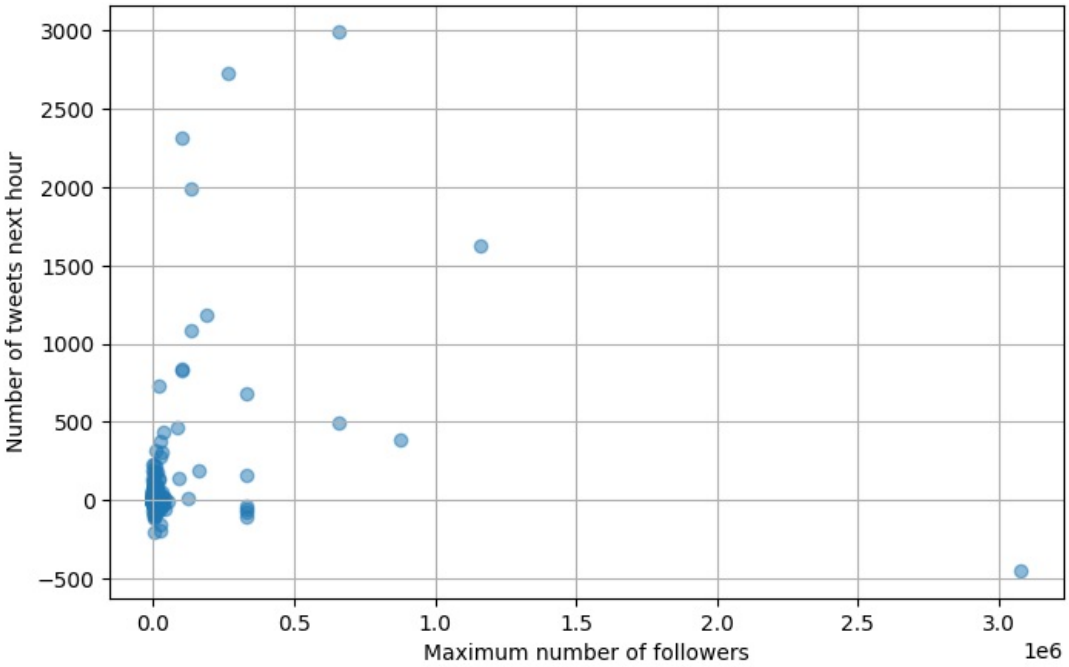
	coef	std err	t	P>|t|	[0.025	0.975]
x1	-40.5461	3.691	-10.986	0.000	-47.795	-33.297
x2	-0.5275	0.229	-2.304	0.022	-0.977	-0.078
x3	-0.0019	0.000	-13.422	0.000	-0.002	-0.002
x4	0.0017	0.000	11.098	0.000	0.001	0.002
x5	-0.4769	0.492	-0.970	0.333	-1.443	0.489
x6	7.5205	0.619	12.146	0.000	6.304	8.737
x7	0.3157	0.069	4.550	0.000	0.179	0.452
x8	61.6785	23.340	2.643	0.008	15.835	107.522
x9	-54.3140	23.768	-2.285	0.023	-101.000	-7.628
x10	4.4767	0.598	7.492	0.000	3.303	5.650

Omnibus:	678.303	Durbin-Watson:	2.128
Prob(Omnibus):	0.000	Jarque-Bera (JB):	317973.744
Skew:	4.976	Prob(JB):	0.00
Kurtosis:	118.178	Cond. No.	2.22e+06

Notes:
[1] R^2 is computed without centering (uncentered) since the model does not contain a constant.
[2] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[3] The condition number is large, 2.22e+06. This might indicate that there are
strong multicollinearity or other numerical problems.
Hashtag: /content/drive/MyDrive/Twitter_project/tweets/tweets_#gopatriots.txt




Number of tweets next hour
3000
2500
2000
1500
1000
500
0
−500
0
5000
10000
15000
20000
Sum of ranking score


Number of tweets next hour
3000
2500
2000
1500
1000
500
0
−500
0.0
0.5
1.0
1.5
2.0
2.5
3.0
1e6
Maximum number of followers

Hashtag: /content/drive/MyDrive/Twitter_project/tweets/tweets_#nfl.txt
MSE: 184288.3990593135

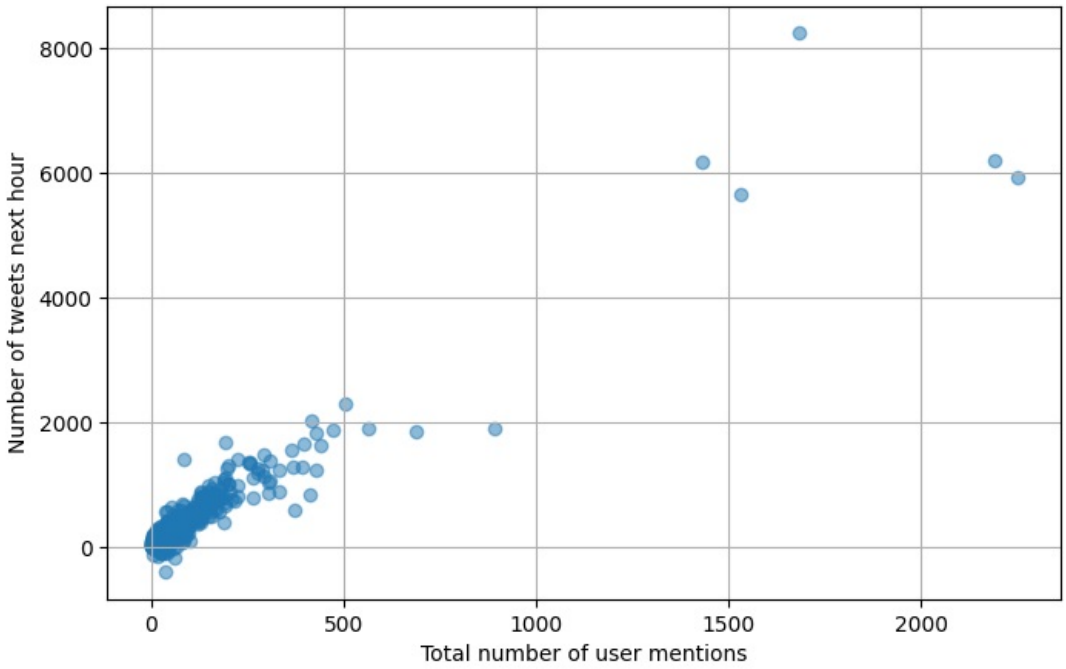
OLS Regression Results

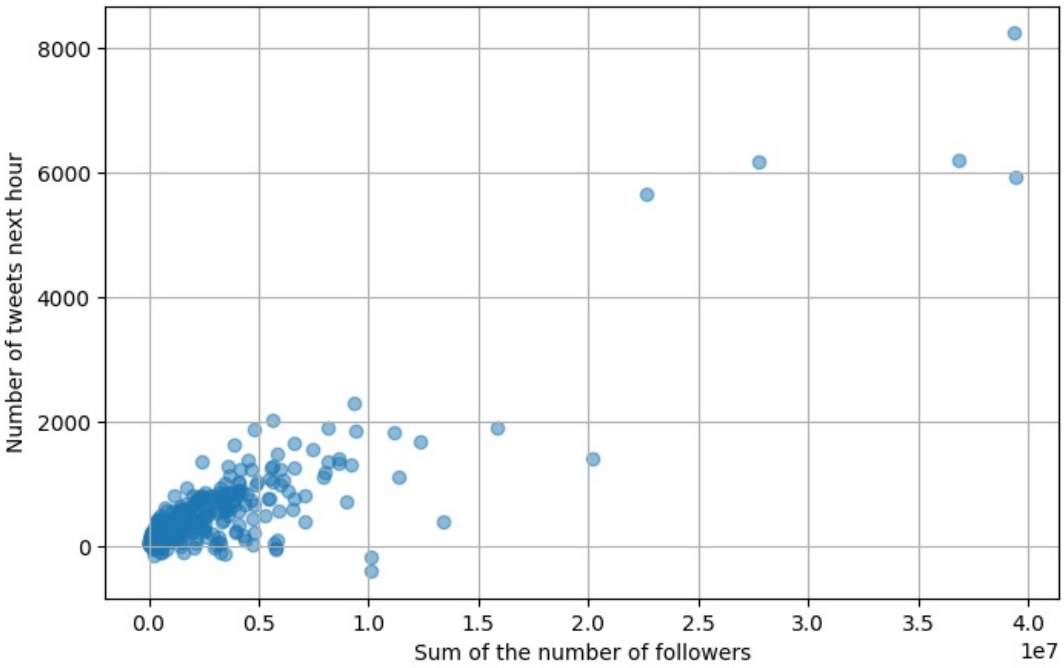
Dep. Variable:	y	R-squared (uncentered):	0.768
Model:	OLS	Adj. R-squared (uncentered):	0.764
Method:	Least Squares	F-statistic:	186.0
Date:	Mon, 18 Mar 2024	Prob (F-statistic):	7.02e-171
Time:	03:38:36	Log-Likelihood:	-4271.7
No. Observations:	571	AIC:	8563.
Df Residuals:	561	BIC:	8607.
Df Model:	10		
Covariance Type:	nonrobust		

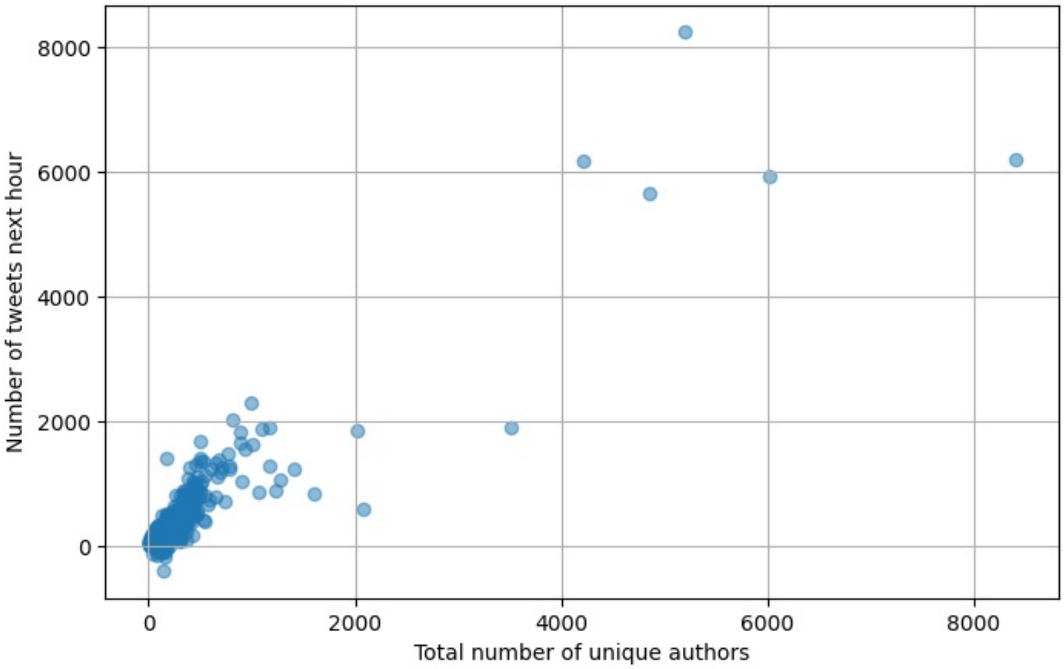
	coef	std err	t	P>|t|	[0.025	0.975]
x1	5.8995	1.413	4.176	0.000	3.124	8.675
x2	-0.0639	0.069	-0.923	0.356	-0.200	0.072
x3	0.0002	2.34e-05	6.511	0.000	0.000	0.000
x4	-0.0002	3.38e-05	-5.920	0.000	-0.000	-0.000
x5	3.5264	1.878	1.878	0.061	-0.162	7.215
x6	-1.2316	0.309	-3.988	0.000	-1.838	-0.625
x7	-0.0892	0.063	-1.421	0.156	-0.212	0.034
x8	79.5622	12.460	6.386	0.000	55.089	104.035
x9	-80.4155	12.584	-6.390	0.000	-105.133	-55.698
x10	3.9251	0.573	6.851	0.000	2.800	5.051

Omnibus:	542.024	Durbin-Watson:	1.969
Prob(Omnibus):	0.000	Jarque-Bera (JB):	86087.002
Skew:	3.588	Prob(JB):	0.00
Kurtosis:	62.723	Cond. No.	4.24e+06

Notes:
[1] R^2 is computed without centering (uncentered) since the model does not contain a constant.
[2] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[3] The condition number is large, 4.24e+06. This might indicate that there are
strong multicollinearity or other numerical problems.
Hashtag: /content/drive/MyDrive/Twitter_project/tweets/tweets_#nfl.txt




Number of tweets next hour
8000
6000
4000
2000
0
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
4.0
1e7
Sum of the number of followers


Number of tweets next hour
8000
6000
4000
2000
0
0
2000
4000
6000
8000
Total number of unique authors

Hashtag: /content/drive/MyDrive/Twitter_project/tweets/tweets_#patriots.txt
MSE: 3619767.990944837

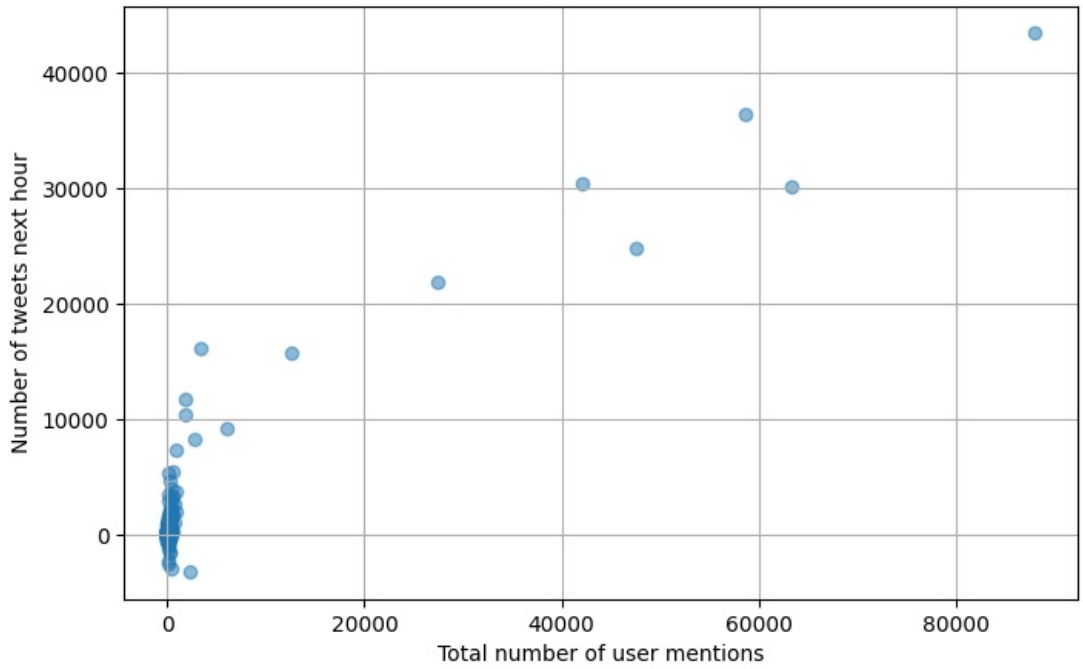
OLS Regression Results

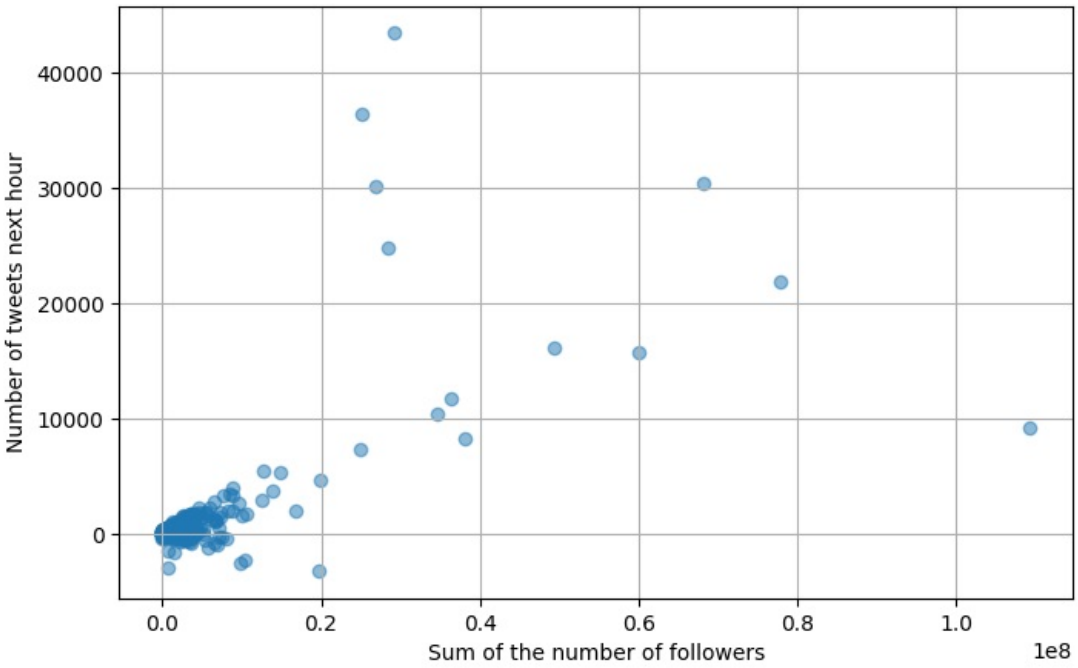
Dep. Variable:	y	R-squared (uncentered):	0.783
Model:	OLS	Adj. R-squared (uncentered):	0.779
Method:	Least Squares	F-statistic:	202.0
Date:	Mon, 18 Mar 2024	Prob (F-statistic):	1.18e-178
Time:	03:39:13	Log-Likelihood:	-5121.8
No. Observations:	571	AIC:	1.026e+04
Df Residuals:	561	BIC:	1.031e+04
Df Model:	10		
Covariance Type:	nonrobust		

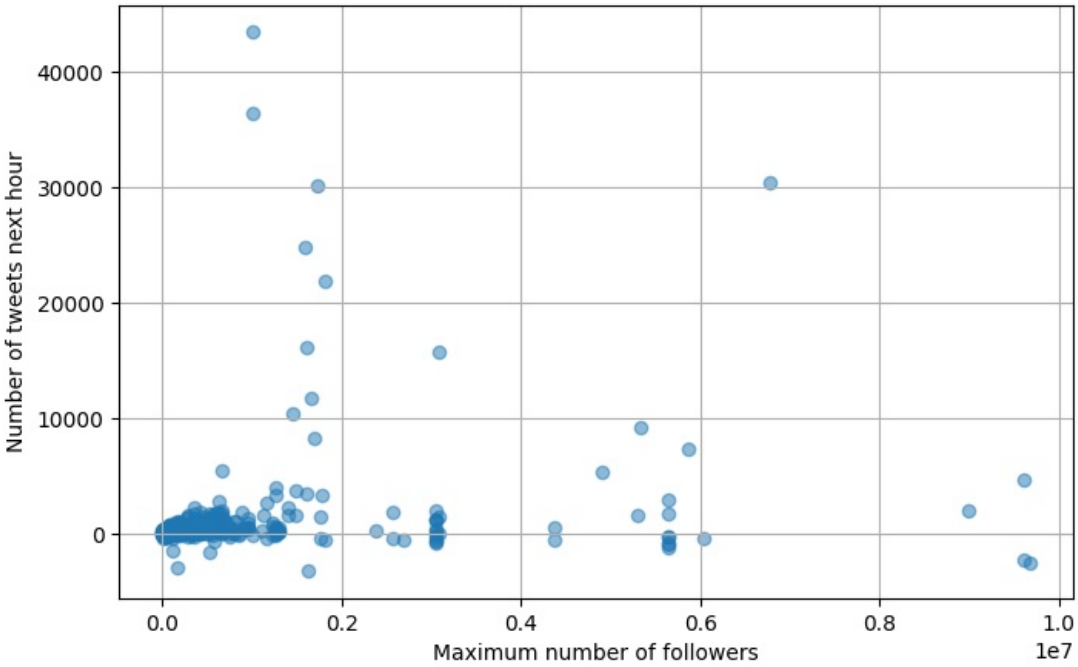
	coef	std err	t	P>|t|	[0.025	0.975]
x1	-26.0999	3.860	-6.762	0.000	-33.681	-18.519
x2	-0.0272	0.084	-0.325	0.745	-0.192	0.137
x3	0.0006	6.41e-05	9.538	0.000	0.000	0.001
x4	-0.0009	0.000	-7.476	0.000	-0.001	-0.001
x5	9.5584	6.696	1.427	0.154	-3.595	22.711
x6	4.7119	0.725	6.500	0.000	3.288	6.136
x7	-0.0893	0.045	-1.987	0.047	-0.178	-0.001
x8	41.7522	37.414	1.116	0.265	-31.736	115.240
x9	-38.1127	37.304	-1.022	0.307	-111.385	35.159
x10	2.1742	0.179	12.129	0.000	1.822	2.526

Omnibus:	938.060	Durbin-Watson:	1.898
Prob(Omnibus):	0.000	Jarque-Bera (JB):	500473.514
Skew:	9.614	Prob(JB):	0.00
Kurtosis:	146.757	Cond. No.	5.59e+06

Notes:
[1] R^2 is computed without centering (uncentered) since the model does not contain a constant.
[2] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[3] The condition number is large, 5.59e+06. This might indicate that there are
strong multicollinearity or other numerical problems.
Hashtag: /content/drive/MyDrive/Twitter_project/tweets/tweets_#patriots.txt




Number of tweets next hour
40000
30000
20000
10000
0
0.0
0.2
0.4
0.6
0.8
1.0
Sum of the number of followers
1e8


Number of tweets next hour
40000
30000
20000
10000
0
0.0
0.2
0.4
0.6
0.8
1.0
Maximum number of followers
1e7

Hashtag: /content/drive/MyDrive/Twitter_project/tweets/tweets_#sb49.txt
MSE: 8860126.964503309

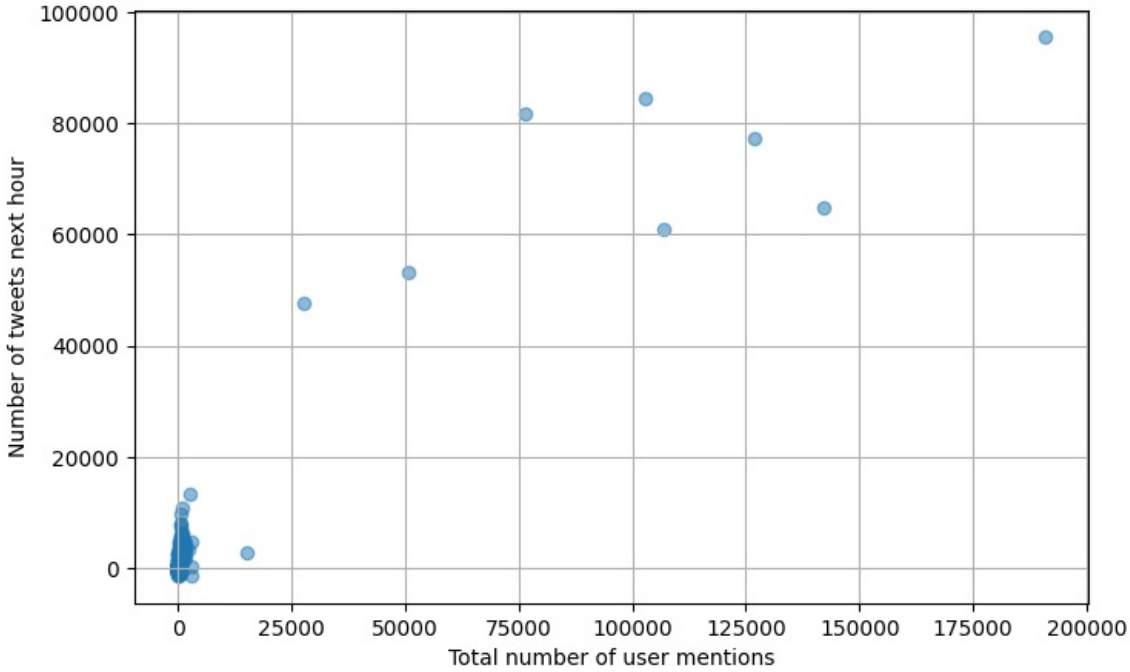
OLS Regression Results

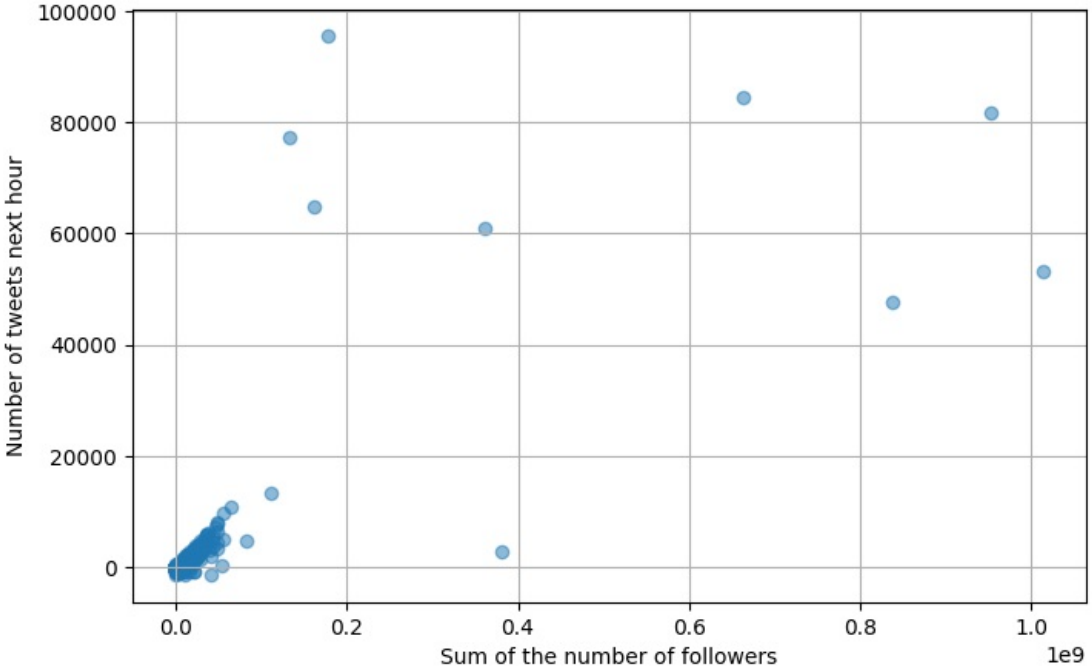
Dep. Variable:	y	R-squared (uncentered):	0.895
Model:	OLS	Adj. R-squared (uncentered):	0.894
Method:	Least Squares	F-statistic:	480.7
Date:	Mon, 18 Mar 2024	Prob (F-statistic):	1.30e-267
Time:	03:40:15	Log-Likelihood:	-5377.4
No. Observations:	571	AIC:	1.077e+04
Df Residuals:	561	BIC:	1.082e+04
Df Model:	10		
Covariance Type:	nonrobust		

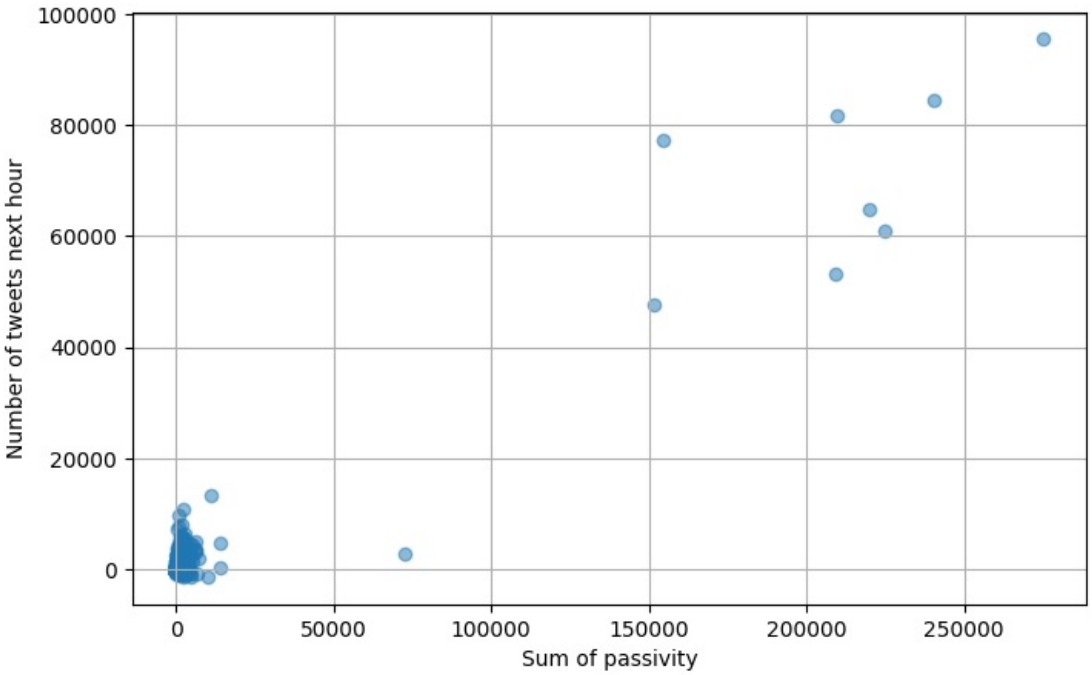
	coef	std err	t	P>|t|	[0.025	0.975]
x1	-16.0877	8.446	-1.905	0.057	-32.677	0.502
x2	-0.0598	0.072	-0.830	0.407	-0.201	0.082
x3	0.0002	3e-05	6.827	0.000	0.000	0.000
x4	-0.0002	5.2e-05	-3.073	0.002	-0.000	-5.76e-05
x5	-15.5423	10.336	-1.504	0.133	-35.844	4.759
x6	2.7952	1.793	1.559	0.120	-0.727	6.318
x7	-0.4180	0.065	-6.397	0.000	-0.546	-0.290
x8	-111.1858	70.212	-1.584	0.114	-249.097	26.726
x9	113.0474	69.914	1.617	0.106	-24.278	250.373
x10	2.4027	0.249	9.637	0.000	1.913	2.892

Omnibus:	979.614	Durbin-Watson:	1.676
Prob(Omnibus):	0.000	Jarque-Bera (JB):	886991.851
Skew:	10.269	Prob(JB):	0.00
Kurtosis:	194.989	Cond. No.	6.21e+07

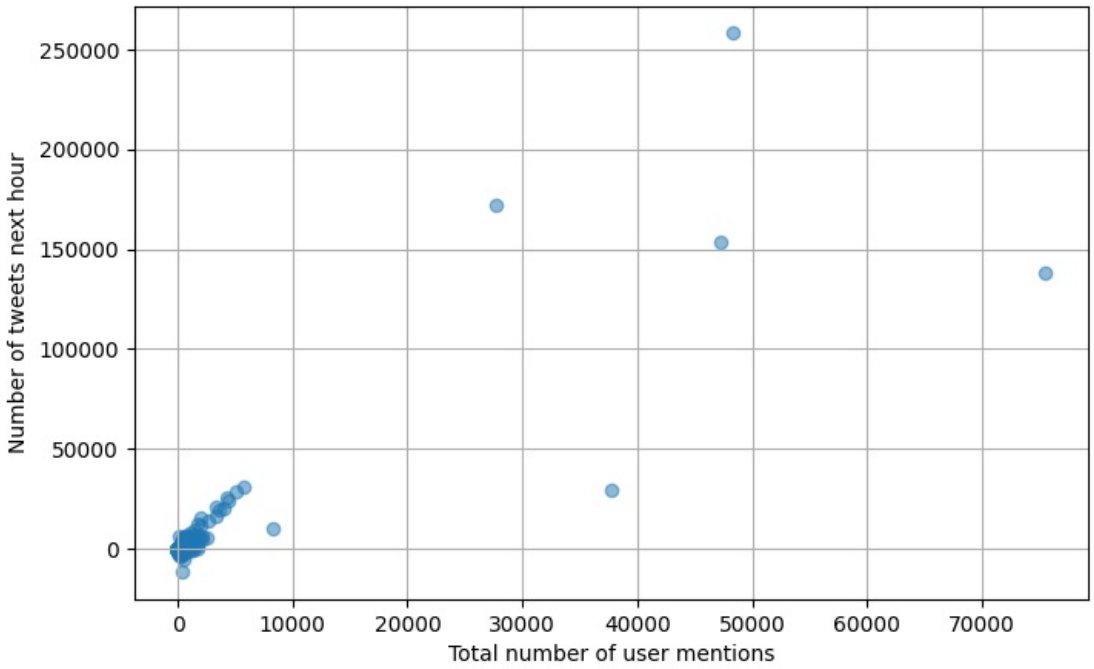
Notes:
[1] R^2 is computed without centering (uncentered) since the model does not contain a constant.
[2] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[3] The condition number is large, 6.21e+07. This might indicate that there are
strong multicollinearity or other numerical problems.
Hashtag: /content/drive/MyDrive/Twitter_project/tweets/tweets_#sb49.txt

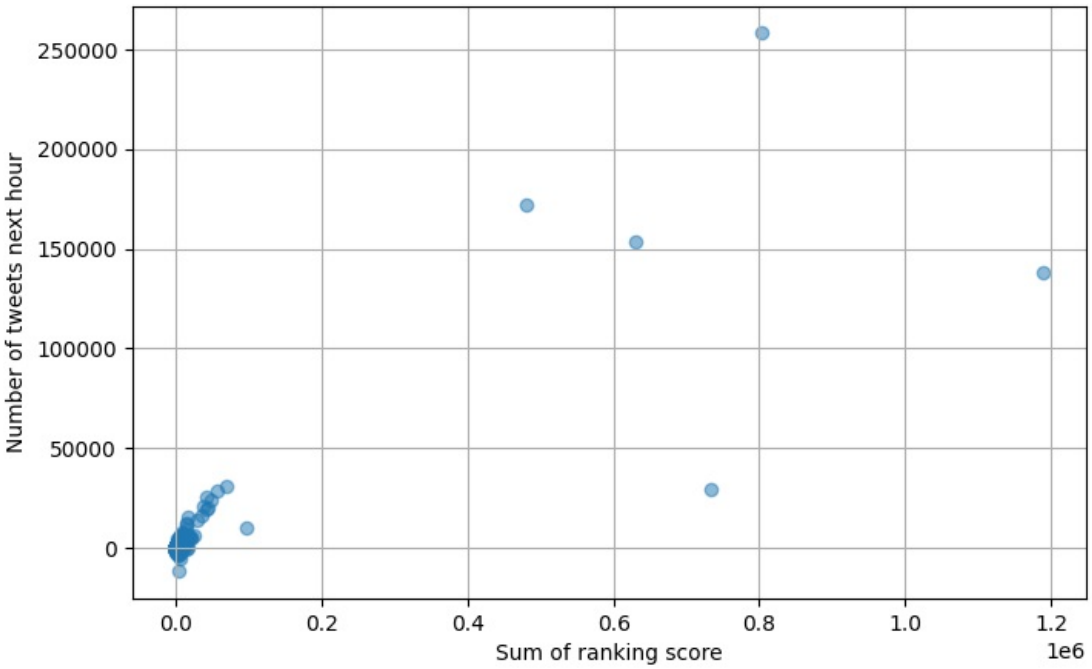


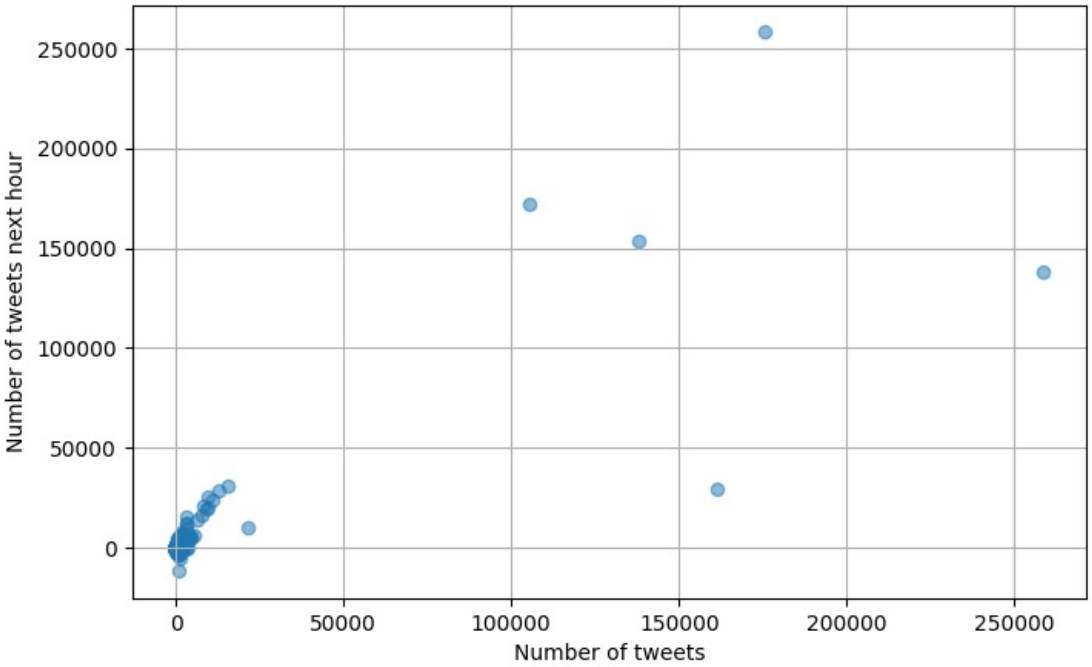

100000
80000
60000
40000
20000
0
Number of tweets next hour
0.0
0.2
0.4
0.6
0.8
1.0
1e9
Sum of the number of followers


100000
80000
60000
40000
20000
0
Number of tweets next hour
0
50000
100000
150000
200000
250000
Sum of passivity

```
--------------------------------------------------------------------------------


Hashtag: /content/drive/MyDrive/Twitter_project/tweets/tweets_#superbowl.txt
MSE:  15458967.54483511
                                 OLS Regression Results
================================================================================================
Dep. Variable:                        y   R-squared (uncentered):                   0.943
Model:                              OLS   Adj. R-squared (uncentered):              0.942
Method:                   Least Squares   F-statistic:                              932.3
Date:                  Mon, 18 Mar 2024   Prob (F-statistic):                        0.00
Time:                          03:41:59   Log-Likelihood:                         -5536.3
No. Observations:                   571   AIC:                                  1.109e+04
Df Residuals:                       561   BIC:                                  1.114e+04
Df Model:                            10
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
x1           -41.2998      5.588     -7.391      0.000     -52.275     -30.325
x2            -0.7473      0.119     -6.294      0.000      -0.981      -0.514
x3            -0.0001   2.31e-05     -6.431      0.000      -0.000      -0.000
x4          9.802e-05   8.58e-05      1.143      0.254   -7.04e-05       0.000
x5           -33.3818     14.823     -2.252      0.025     -62.498      -4.266
x6             8.8292      1.163      7.589      0.000       6.544      11.114
x7             0.1727      0.047      3.640      0.000       0.079       0.266
x8            36.5074     25.912      1.409      0.159     -14.389      87.404
x9           -34.0838     25.813     -1.320      0.187     -84.785      16.618
x10            8.8973      0.408     21.820      0.000       8.096       9.698
==============================================================================
Omnibus:                     1012.058   Durbin-Watson:                   1.939
Prob(Omnibus):                  0.000   Jarque-Bera (JB):          1207472.208
Skew:                          10.891   Prob(JB):                         0.00
Kurtosis:                     227.226   Cond. No.                     2.39e+07
==============================================================================

Notes:
[1] R² is computed without centering (uncentered) since the model does not contain a constant.
[2] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[3] The condition number is large, 2.39e+07. This might indicate that there are
strong multicollinearity or other numerical problems.
Hashtag: /content/drive/MyDrive/Twitter_project/tweets/tweets_#superbowl.txt
```




Number of tweets next hour
250000
200000
150000
100000
50000
0
0.0
0.2
0.4
0.6
0.8
1.0
1.2
1e6
Sum of ranking score


Number of tweets next hour
250000
200000
150000
100000
50000
0
0
50000
100000
150000
200000
250000
Number of tweets

In [13]:

```
# find the intersected time intervals for all twitter data
def get_time_interval(filename):
    with open(filename, 'r') as file:
        lines = file.readlines()
        max_time = 0
        min_time = np.inf
        for line in lines:
            json_obj = json.loads(line)
            if json_obj['citation_date'] > max_time:
                max_time = json_obj['citation_date']
            if json_obj['citation_date'] < min_time:
                min_time = json_obj['citation_date']
        return max_time, min_time

max_t = []
min_t = []
for file in files:
    max_time, min_time = get_time_interval(file)
    max_t.append(max_time)
    min_t.append(min_time)

max_time_agg = min(max_t)
min_time_agg = max(min_t)
```

In [15]:

```
print(max_time_agg)
print(min_time_agg)
```

```
1423295675
1421238675
```

Fanbase prediction

In [3]:

```
import json
# exclude any tweets whose author is not from Washington or Massachusetts
WA_substrings = ['Washington', 'Seattle', 'WA']
MA_substrings = ['Massachusetts', 'Boston', 'MA']

superbowl_dataset_trimmed = []

with open('/content/drive/MyDrive/Twitter_project/tweets/tweets_#superbowl.txt', 'r') as file:
    lines = file.readlines()

    for line in lines:
        json_obj = json.loads(line)
        location = json_obj['tweet']['user']['location']

        for w in WA_substrings:
            if w in location:
                superbowl_dataset_trimmed.append((json_obj['tweet']['text'], 'Washington'))
                break

        for m in MA_substrings:
            if m in location:
                superbowl_dataset_trimmed.append((json_obj['tweet']['text'], 'Massachusetts'))
                break
```

Training a binary classifier (Logistic regression)

In [4]:

```
import numpy as np
from sklearn.model_selection import train_test_split

x_superbowl = np.array(superbowl_dataset_trimmed)[:, 0]
y_superbowl = np.array(superbowl_dataset_trimmed)[:, 1]

y_superbowl_binary = np.zeros(y_superbowl.shape)
y_superbowl_binary[y_superbowl == 'Washington'] = 1

x_train, x_test, y_train, y_test = train_test_split(x_superbowl, y_superbowl_binary, test_size=0.1, random_state=
42)
```

In [5]:

```
import nltk
# nltk.download('wordnet')
# nltk.download('punkt')
# nltk.download('averaged_perceptron_tagger')

from nltk import pos_tag
from nltk.tokenize import word_tokenize
from nltk.stem.wordnet import WordNetLemmatizer
from nltk.corpus import wordnet as wn
from collections import defaultdict

# pos_tags: treebank to wordnet
tag_map = defaultdict(lambda: wn.NOUN)
tag_map['J'] = wn.ADJ
tag_map['V'] = wn.VERB
tag_map['R'] = wn.ADV

wnl = WordNetLemmatizer()

def lemmatize(data):
    lemmatized = []
    for doc in data:
        tokens = word_tokenize(doc)
        words = [wnl.lemmatize(word, tag_map[tag[0]]) for word,tag in pos_tag(tokens)
                 if wnl.lemmatize(word, tag_map[tag[0]]).isalpha()]
        sentence = ' '.join(words)
        lemmatized.append(sentence)
    return lemmatized
```

In [6]:

```
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.decomposition import TruncatedSVD

x_train_lemmatized = lemmatize(x_train)
x_test_lemmatized = lemmatize(x_test)

tfidf_vectorizer = TfidfVectorizer(stop_words='english', min_df=3)
x_train_tfidf = tfidf_vectorizer.fit_transform(x_train_lemmatized)
x_test_tfidf = tfidf_vectorizer.transform(x_test_lemmatized)

svd = TruncatedSVD(n_components=50, random_state=42)
x_train_svd = svd.fit_transform(x_train_tfidf)
x_test_svd = svd.transform(x_test_tfidf)
```

In [25]:

```
# Logistic Regression: GridSearch
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import GridSearchCV
import pandas as pd


grid_logistic = GridSearchCV(estimator=LogisticRegression(random_state=42),
                        param_grid={'C':[10**-3,10**-2,10**-1,10**0,10**1,10**2,10**3],
                                    'penalty': ['l1', 'l2', 'elasticnet']},
                        cv=5, n_jobs=-1, verbose=1).fit(x_train_svd, y_train)

result_logistic = pd.DataFrame(grid_logistic.cv_results_)[['mean_test_score', 'param_C', 'param_penalty']]
result_logistic = result_logistic.sort_values(by=['mean_test_score'], ascending=False).reset_index(drop=True)
result_logistic.head()
```

```
Fitting 5 folds for each of 21 candidates, totalling 105 fits

/usr/local/lib/python3.10/dist-packages/sklearn/model_selection/_validation.py:378: FitFailedWarning
:
70 fits failed out of a total of 105.
The score on these train-test partitions for these parameters will be set to nan.
If these failures are not expected, you can try to debug them by setting error_score='raise'.

Below are more details about the failures:
--------------------------------------------------------------------------------
35 fits failed with the following error:
Traceback (most recent call last):
  File "/usr/local/lib/python3.10/dist-packages/sklearn/model_selection/_validation.py", line 686, i
n _fit_and_score
    estimator.fit(X_train, y_train, **fit_params)
  File "/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_logistic.py", line 1162, in fi
t
    solver = _check_solver(self.solver, self.penalty, self.dual)
  File "/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_logistic.py", line 54, in _che
ck_solver
    raise ValueError(
ValueError: Solver lbfgs supports only 'l2' or 'none' penalties, got l1 penalty.

--------------------------------------------------------------------------------
35 fits failed with the following error:
Traceback (most recent call last):
  File "/usr/local/lib/python3.10/dist-packages/sklearn/model_selection/_validation.py", line 686, i
n _fit_and_score
    estimator.fit(X_train, y_train, **fit_params)
  File "/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_logistic.py", line 1162, in fi
t
    solver = _check_solver(self.solver, self.penalty, self.dual)
  File "/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_logistic.py", line 54, in _che
ck_solver
    raise ValueError(
ValueError: Solver lbfgs supports only 'l2' or 'none' penalties, got elasticnet penalty.

  warnings.warn(some_fits_failed_message, FitFailedWarning)
/usr/local/lib/python3.10/dist-packages/sklearn/model_selection/_search.py:952: UserWarning: One or
more of the test scores are non-finite: [       nan 0.56505312        nan        nan 0.69919142
nan
        nan 0.71877536        nan        nan 0.72356763        nan
        nan 0.72423577        nan        nan 0.72432794        nan
        nan 0.7243049         nan]
  warnings.warn(
```

Out[25]:

	mean_test_score	param_C	param_penalty
0	0.724328	100	l2
1	0.724305	1000	l2
2	0.724236	10	l2
3	0.723568	1	l2
4	0.718775	0.1	l2

In [26]:

```
logistic_optim = LogisticRegression(penalty=grid_logistic.best_params_['penalty'],
                                    C=grid_logistic.best_params_['C'], random_state=42)

logistic_optim.fit(x_train_svd, y_train)
```

Out[26]:

```
LogisticRegression
LogisticRegression(C=100, random_state=42)
```

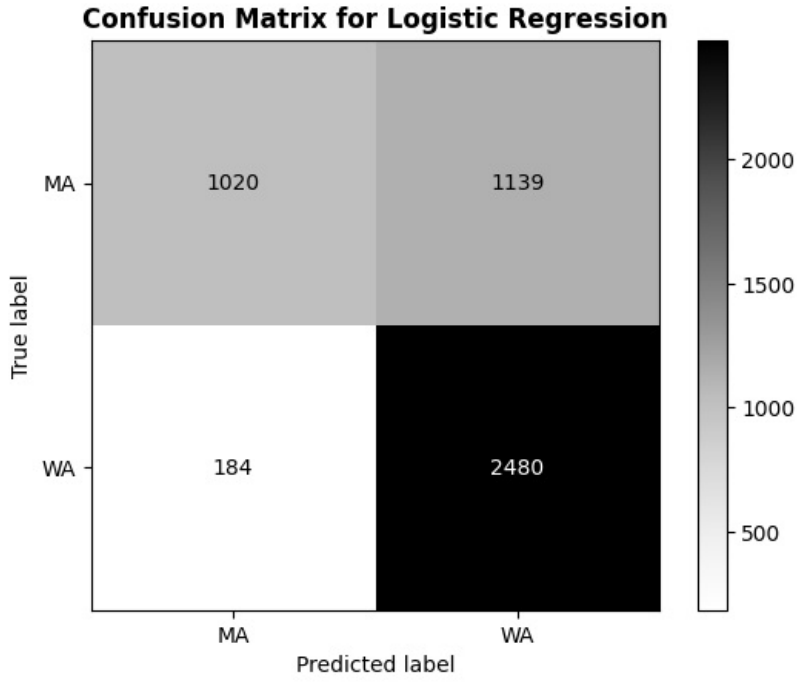
In [27]:

```
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score
from sklearn.metrics import recall_score
from sklearn.metrics import precision_score
from sklearn.metrics import f1_score

y_pred_logistic = logistic_optim.predict(x_test_svd)
y_pred_prob_logistic = logistic_optim.predict_proba(x_test_svd)[:,1]

print('Logistic Regression:')
print('confusion_matrix:\n', confusion_matrix(y_test, y_pred_logistic))
print('accuracy:', accuracy_score(y_test, y_pred_logistic))
print('recall:', recall_score(y_test, y_pred_logistic))
print('precision:', precision_score(y_test, y_pred_logistic))
print('f1_score:', f1_score(y_test, y_pred_logistic))
```

```
Logistic Regression:
confusion_matrix:
 [[1020 1139]
 [ 184 2480]]
accuracy: 0.7256894049346879
recall: 0.9309309309309309
precision: 0.6852721746338768
f1_score: 0.7894318000954959
```

In [31]:

```
import matplotlib.pyplot as plt
from sklearn.metrics import ConfusionMatrixDisplay

class_names = ['MA', 'WA']

disp = ConfusionMatrixDisplay(confusion_matrix=confusion_matrix(y_test, y_pred_logistic), display_labels=class_na
mes)
disp.plot(values_format='d', cmap=plt.cm.Greys)
plt.title('Confusion Matrix for Logistic Regression', fontweight='bold')
plt.show()
plt.tight_layout()
```



<Figure size 640x480 with 0 Axes>

Random Forest

In [25]:

```
from sklearn.pipeline import Pipeline
from sklearn.model_selection import KFold, GridSearchCV
from sklearn.preprocessing import StandardScaler
from sklearn.ensemble import RandomForestClassifier

pipe_rfc = Pipeline([
    ('standardize', StandardScaler()),
    ('model', RandomForestClassifier(random_state=42))
])

param_grid = {
    'model__max_depth': [10, 30, 50, 70, 100, 200, None]
}


grid_rfc = GridSearchCV(pipe_rfc, param_grid=param_grid, cv=KFold(5, shuffle=True, random_state=42), n_jobs=-1, v
erbose=1)
grid_rfc.fit(x_train_svd, y_train)

result_rfc = pd.DataFrame(grid_rfc.cv_results_)[['mean_test_score', 'param_model__max_depth']]
result_rfc = result_rfc.sort_values(by=['mean_test_score'], ascending=False).reset_index(drop=True)
result_rfc.head()
```

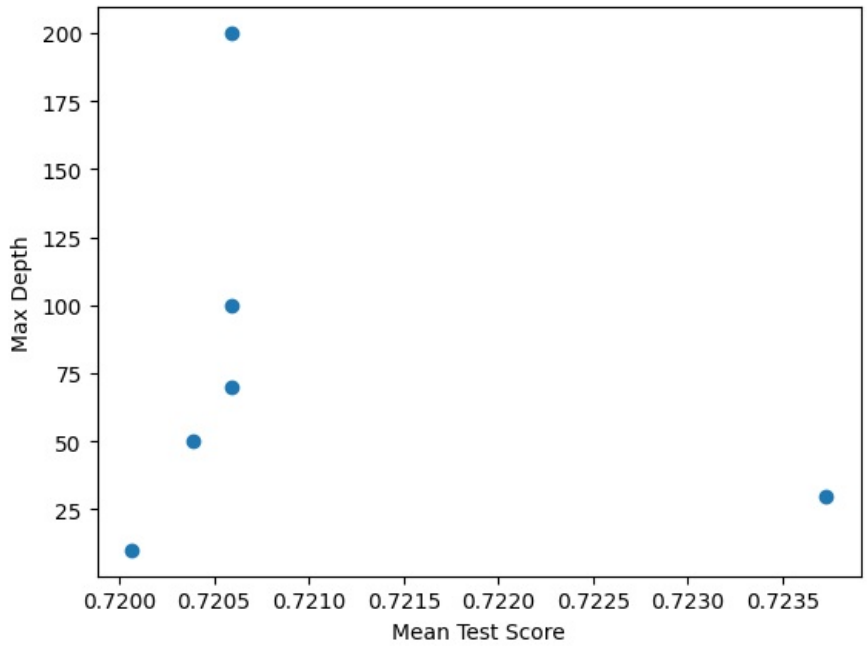
```
Fitting 5 folds for each of 7 candidates, totalling 35 fits
```

Out[25]:

	mean_test_score	param_model__max_depth
0	0.723729	30
1	0.720595	70
2	0.720595	100
3	0.720595	200
4	0.720595	None

In [37]:

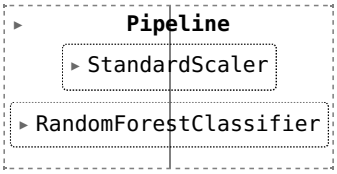
```
# @title Mean Test Score vs Max Depth

import matplotlib.pyplot as plt
plt.scatter(result_rfc['mean_test_score'], result_rfc['param_model__max_depth'])
plt.xlabel('Mean Test Score')
_ = plt.ylabel('Max Depth')
```



In [28]:

```
pipe_rfc_optim = Pipeline([
        ('standardize', StandardScaler()),
        ('model', RandomForestClassifier(max_depth=30, random_state=42))
])

pipe_rfc_optim.fit(x_train_svd, y_train)
```

Out[28]:



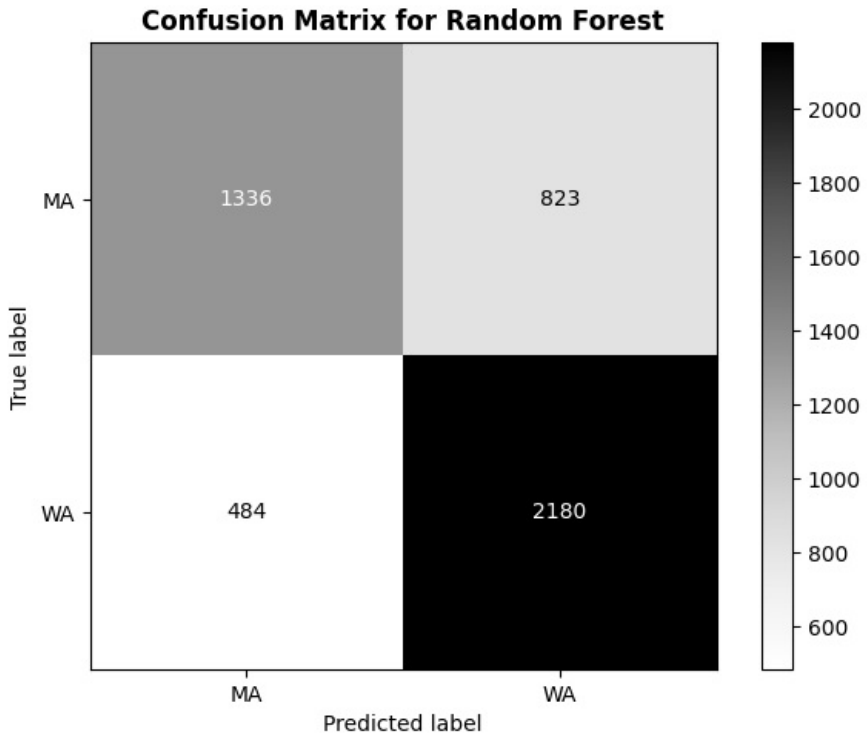
In [29]:

```
y_pred_rfc = pipe_rfc_optim.predict(x_test_svd)
y_pred_prob_rfc = pipe_rfc_optim.predict_proba(x_test_svd)[:,1]

print('Random Forest Classifer:')
print('confusion_matrix:\n', confusion_matrix(y_test, y_pred_rfc))
print('accuracy:', accuracy_score(y_test, y_pred_rfc))
print('recall:', recall_score(y_test, y_pred_rfc))
print('precision:', precision_score(y_test, y_pred_rfc))
print('f1_score:', f1_score(y_test, y_pred_rfc))
```

```
Random Forest Classifer:
confusion_matrix:
 [[1336  823]
 [ 484 2180]]
accuracy: 0.7290068422143894
recall: 0.8183183183183184
precision: 0.7259407259407259
f1_score: 0.7693665078524792
```

In [30]:

```
disp = ConfusionMatrixDisplay(confusion_matrix=confusion_matrix(y_test, y_pred_rfc), display_labels=class_names)
disp.plot(values_format='d', cmap=plt.cm.Greys)
plt.tight_layout()
plt.title('Confusion Matrix for Random Forest', fontweight='bold')
plt.show()
```



SVC

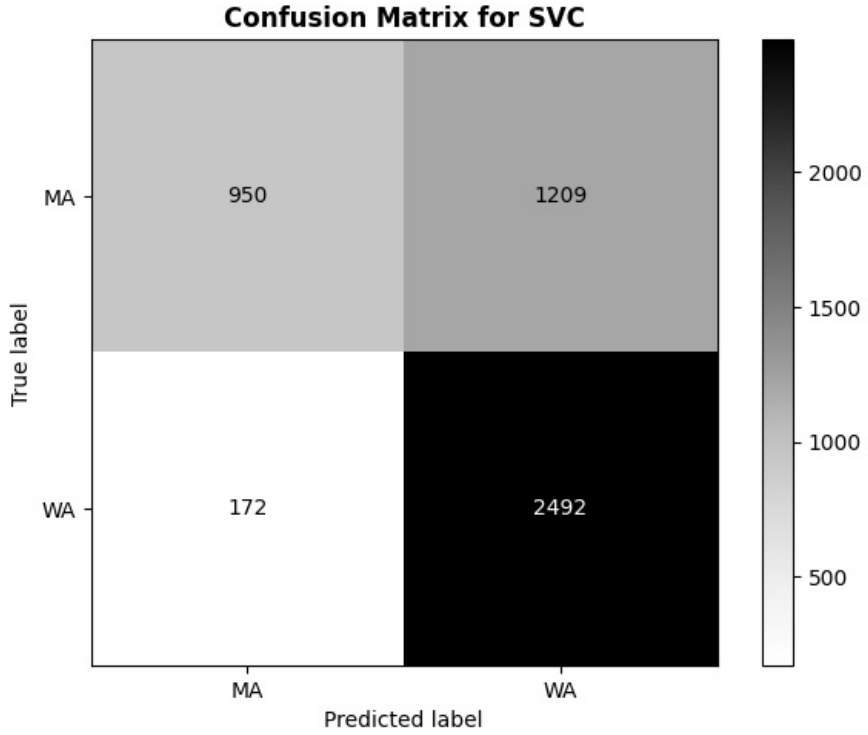
In [23]:

```
from sklearn.svm import SVC

pipe_svc_optim = Pipeline([
        ('standardize', StandardScaler()),
        ('model', SVC(random_state=42, C = 1, kernel = 'linear' , gamma = 'auto'))
])

pipe_svc_optim.fit(x_train_svd, y_train)

y_pred_svc = pipe_svc_optim.predict(x_test_svd)
y_pred_prob_svc = pipe_svc_optim.predict(x_test_svd)

print('SVC:')
print('confusion_matrix:\n', confusion_matrix(y_test, y_pred_svc))
print('accuracy:', accuracy_score(y_test, y_pred_svc))
print('recall:', recall_score(y_test, y_pred_svc))
print('precision:', precision_score(y_test, y_pred_svc))
print('f1_score:', f1_score(y_test, y_pred_svc))
```

```
SVC:
confusion_matrix:
 [[ 950 1209]
 [ 172 2492]]
accuracy: 0.7136636947957703
recall: 0.9354354354354354
precision: 0.6733315320183734
f1_score: 0.7830322073841319
```

In [24]:

```
disp = ConfusionMatrixDisplay(confusion_matrix=confusion_matrix(y_test, y_pred_svc), display_labels=class_names)
disp.plot(values_format='d', cmap=plt.cm.Greys)
plt.tight_layout()
plt.title('Confusion Matrix for SVC', fontweight='bold')
plt.show()
```

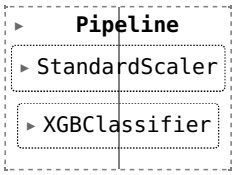


XGBoost

In [8]:

```
from xgboost import XGBClassifier
from sklearn.pipeline import Pipeline
from sklearn.model_selection import KFold, GridSearchCV
from sklearn.preprocessing import StandardScaler


# Create a pipeline for XGBoost Classifier
pipe_xgb = Pipeline([
    ('standardize', StandardScaler()),
    ('model', XGBClassifier(random_state=42))  # Adjust parameters as needed
])

# Fit the pipeline on the training data
pipe_xgb.fit(x_train_svd, y_train)
```

Out[8]:



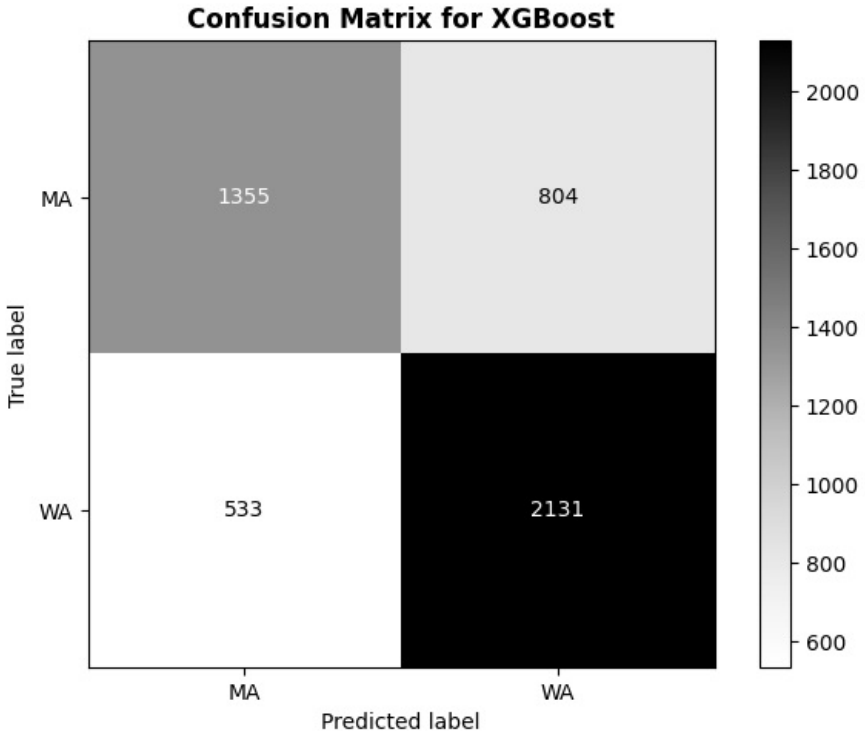
In [10]:

```
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score
from sklearn.metrics import recall_score
from sklearn.metrics import precision_score
from sklearn.metrics import f1_score
from sklearn.metrics import ConfusionMatrixDisplay

y_pred_xgb = pipe_xgb.predict(x_test_svd)
y_pred_prob_xgb = pipe_xgb.predict_proba(x_test_svd)[:,1]

print('XGBoost:')
print('confusion_matrix:\n', confusion_matrix(y_test, y_pred_xgb))
print('accuracy:', accuracy_score(y_test, y_pred_xgb))
print('recall:', recall_score(y_test, y_pred_xgb))
print('precision:', precision_score(y_test, y_pred_xgb))
print('f1_score:', f1_score(y_test, y_pred_xgb))
```

```
XGBoost:
confusion_matrix:
 [[1355  804]
 [ 533 2131]]
accuracy: 0.7227866473149492
recall: 0.799924924924925
precision: 0.7260647359454855
f1_score: 0.7612073584568673
```

In [13]:

```
class_names = ['MA', 'WA']
from matplotlib import pyplot as plt
disp = ConfusionMatrixDisplay(confusion_matrix=confusion_matrix(y_test, y_pred_xgb), display_labels=class_names)
disp.plot(values_format='d', cmap=plt.cm.Greys)
plt.tight_layout()
plt.title('Confusion Matrix for XGBoost', fontweight='bold')
plt.show()
```

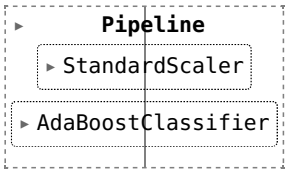


AdaBoost

In [14]:

```
from sklearn.ensemble import AdaBoostClassifier

# Define the pipeline with feature scaling and AdaBoostClassifier
pipe_adaboost_optim = Pipeline([
    ('standardize', StandardScaler()),
    ('model', AdaBoostClassifier(random_state=42))
])

# Fit the pipeline on the training data
pipe_adaboost_optim.fit(x_train_svd, y_train)
```

Out[14]:



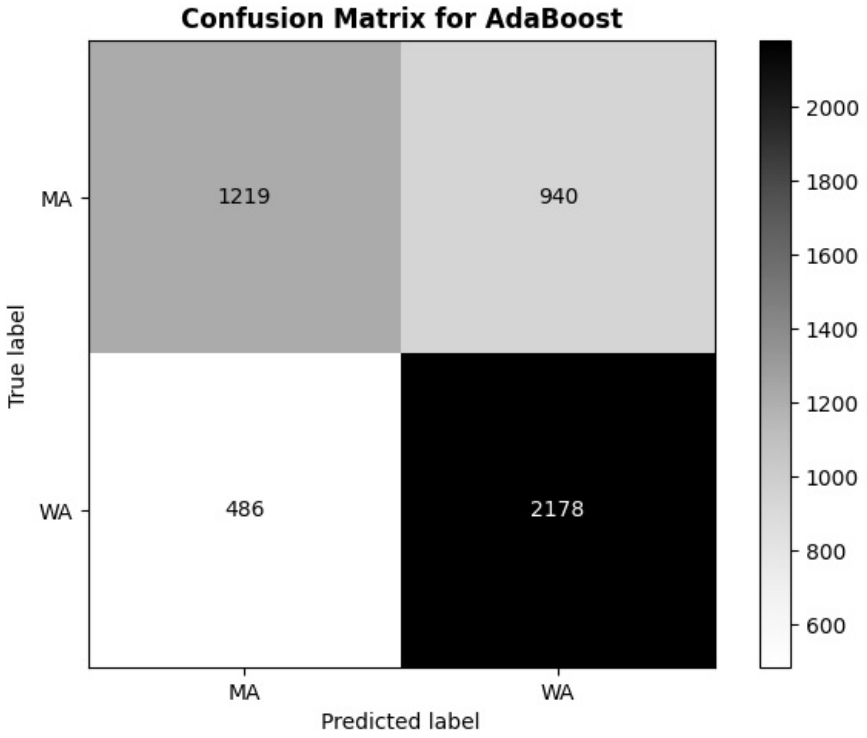
In [15]:

```
y_pred_adaboost = pipe_adaboost_optim.predict(x_test_svd)
y_pred_prob_adaboost = pipe_adaboost_optim.predict_proba(x_test_svd)[:, 1]

print('AdaBoost:')
print('confusion_matrix:\n', confusion_matrix(y_test, y_pred_adaboost))
print('accuracy:', accuracy_score(y_test, y_pred_adaboost))
print('recall:', recall_score(y_test, y_pred_adaboost))
print('precision:', precision_score(y_test, y_pred_adaboost))
print('f1_score:', f1_score(y_test, y_pred_adaboost))
```

```
AdaBoost:
confusion_matrix:
 [[1219  940]
 [ 486 2178]]
accuracy: 0.70433340244661
recall: 0.8175675675675675
precision: 0.6985246953175113
f1_score: 0.7533725354548598
```

In [16]:

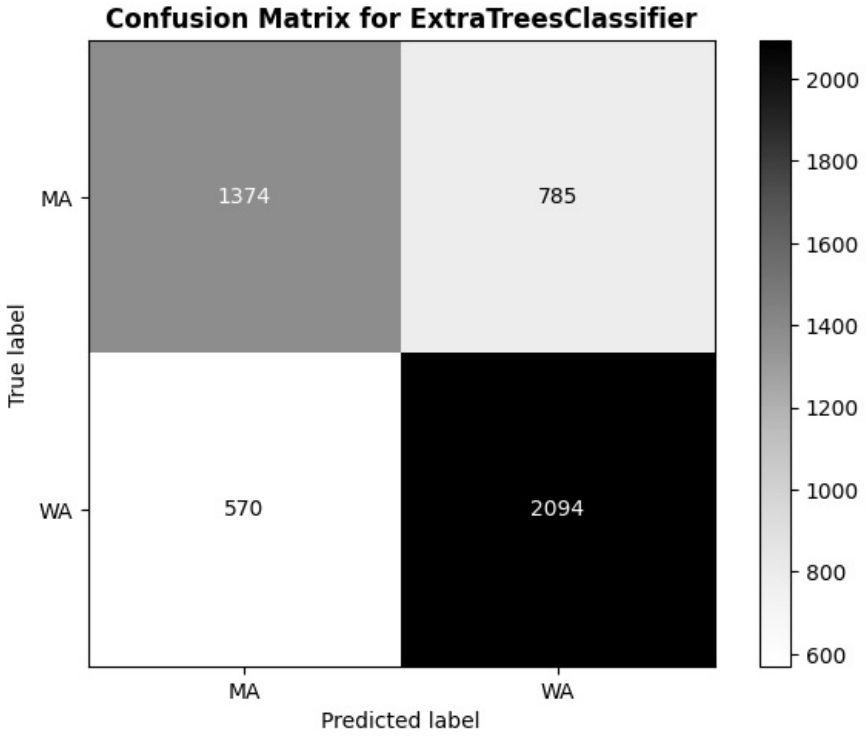
```
conf_matrix_adaboost = confusion_matrix(y_test, y_pred_adaboost)

# Plot confusion matrix for AdaBoostClassifier
disp_adaboost = ConfusionMatrixDisplay(confusion_matrix=conf_matrix_adaboost, display_labels=class_names)
disp_adaboost.plot(values_format='d', cmap=plt.cm.Greys)
plt.tight_layout()
plt.title('Confusion Matrix for AdaBoost', fontweight='bold')
plt.show()
```



ExtraTreesClassifier

In [17]:

```
from sklearn.ensemble import ExtraTreesClassifier

pipe_extra_trees = Pipeline([
    ('standardize', StandardScaler()),
    ('model', ExtraTreesClassifier(random_state=42))
])

pipe_extra_trees.fit(x_train_svd, y_train)

y_pred_extra_trees = pipe_extra_trees.predict(x_test_svd)

conf_matrix_extra_trees = confusion_matrix(y_test, y_pred_extra_trees)

disp_extra_trees = ConfusionMatrixDisplay(confusion_matrix=conf_matrix_extra_trees, display_labels=class_names)
disp_extra_trees.plot(values_format='d', cmap=plt.cm.Greys)
plt.tight_layout()
plt.title('Confusion Matrix for ExtraTreesClassifier', fontweight='bold')
plt.show()
```



In [18]:

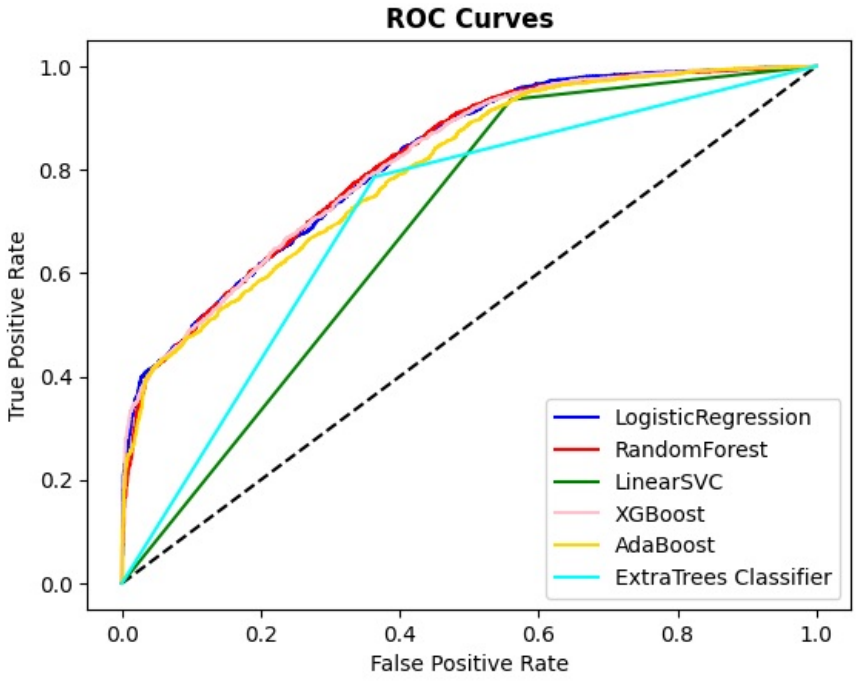
```
from sklearn.metrics import accuracy_score, recall_score, precision_score, f1_score

# Calculate scores for ExtraTreesClassifier
accuracy_extra_trees = accuracy_score(y_test, y_pred_extra_trees)
recall_extra_trees = recall_score(y_test, y_pred_extra_trees)
precision_extra_trees = precision_score(y_test, y_pred_extra_trees)
f1_extra_trees = f1_score(y_test, y_pred_extra_trees)

# Print scores for ExtraTreesClassifier
print('ExtraTreesClassifier:')
print('confusion_matrix:\n', conf_matrix_extra_trees)
print('accuracy:', accuracy_extra_trees)
print('recall:', recall_extra_trees)
print('precision:', precision_extra_trees)
print('f1_score:', f1_extra_trees)
```

```
ExtraTreesClassifier:
confusion_matrix:
 [[1374  785]
 [ 570 2094]]
accuracy: 0.7190545303752851
recall: 0.786036036036036
precision: 0.7273358805140674
f1_score: 0.7555475374346021
```

ROC

In [33]:

```
# aggregated ROC curves
from sklearn.metrics import roc_curve

fpr_logistic, tpr_logistic, _ = roc_curve(y_test, y_pred_prob_logistic)
fpr_rfc, tpr_rfc, _  = roc_curve(y_test, y_pred_prob_rfc)
fpr_svc, tpr_svc, _  = roc_curve(y_test, y_pred_prob_svc)
fpr_xgb, tpr_xgb, _  = roc_curve(y_test, y_pred_prob_xgb)
fpr_adaboost, tpr_adaboost, _  = roc_curve(y_test, y_pred_prob_adaboost)
fpr_extra_trees, tpr_extra_trees, _ = roc_curve(y_test, y_pred_extra_trees)

plt.plot([0, 1], [0, 1], 'k--')

plt.plot(fpr_logistic, tpr_logistic, label = 'LogisticRegression', color='b', linewidth=1.5)
plt.plot(fpr_rfc, tpr_rfc, label = 'RandomForest', color='r', linewidth=1.5)
plt.plot(fpr_svc, tpr_svc, label = 'LinearSVC', color='g', linewidth=1.5)
plt.plot(fpr_xgb, tpr_xgb, label = 'XGBoost', color='pink', linewidth=1.5)
plt.plot(fpr_adaboost, tpr_adaboost, label = 'AdaBoost', color='gold', linewidth=1.5)
plt.plot(fpr_extra_trees, tpr_extra_trees, label = 'ExtraTrees Classifier', color='cyan', linewidth=1.5)


plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.legend(loc='lower right')
plt.title('ROC Curves', weight='bold')
plt.show()
```



Results

ROC plots and confusion matrices for each of the models are plotted.

The results are summarised in the table below :

Model	Accuracy	Recall	Precision	F1 Score
Logistic Regression (Baseline)	0.725	0.931	0.685	**0.789**
Random Forest Classifier	**0.729**	0.818	0.726	0.769
SVC	0.714	**0.935**	0.673	0.783
XGBoost	0.723	0.800	0.726	0.761
AdaBoost	0.704	0.818	0.699	0.753
ExtraTreesClassifier	0.719	0.786	**0.727**	0.756

Based on Accuracy, Random Forest Classifier gives the best results. Logistic Regression gives best results in terms of F1 Score. The inference from these results is that the model does pretty well for the task of classifying between the fan teams of Patriots and Hawks from the superbowl dataset. This inference is backed by the accuracy metrics and the confusion matrix where a clear binary classification between 2 classes is observed.

Model-wise analysis:

1. **Logistic Regression**: Achieved a relatively high accuracy score of 72.6%, indicating its ability to make correct predictions overall. However, it had a lower recall score of 93.1%, suggesting that it might struggle to identify all positive instances.
2. **Random Forest Classifier**: Achieved the highest accuracy score of 72.9%, showcasing its effectiveness in making correct predictions. It also had a relatively high recall score of 81.8%, indicating its capability to identify positive instances.
3. **Support Vector Classifier (SVC)**: Although it achieved a high recall score of 93.5%, suggesting its effectiveness in identifying positive instances, its overall accuracy was slightly lower at 71.4%.
4. **XGBoost**: Demonstrated balanced performance with an accuracy of 72.3% and a recall score of 80.0%, indicating its ability to correctly classify instances and identify positive cases.
5. **AdaBoost**: While it had a moderate accuracy of 70.4%, it had the lowest precision and F1 scores among the models, suggesting a higher rate of false positives and a balance between precision and recall.
6. **ExtraTreesClassifier**: Achieved a decent accuracy score of 71.9% and a balanced F1 score of 75.6%, indicating its robustness in handling imbalanced datasets and making accurate predictions.

In summary, each model has its strengths and weaknesses. Random Forest Classifier performed well overall, achieving the highest accuracy, while Support Vector Classifier had the highest recall score. XGBoost showed balanced performance across different metrics. However, AdaBoost had the lowest precision and F1 scores, indicating potential challenges in minimizing false positives. ExtraTreesClassifier demonstrated robustness in handling imbalanced datasets, as reflected in its balanced F1 score.