EC ENGR 219

2023 Winter

Project 4:

Regression Analysis and

Define Your Own Task!

Wenxin Cheng	706070535	wenxin0319@g.ucla.edu
Yuxin Yin	606073780	yyxyy999@g.ucla.edu
Yingqian Zhao	306071513	zhaoyq99@g.ucla.edu

QUESTION 1: Data Inspection

Question 1.1: Plot a heatmap of the Pearson correlation matrix of the dataset columns. Report which features have the highest absolute correlation with the target variable. In the context of either dataset, describe what the correlation patterns suggest.

Before we plot the heat map of the Pearson correlation matrix of the dataset columns, we firstly used pandas to read the diamonds.csv dataset1. To have an overview of the original datasets, values with corresponding metrics per diamond is shown in the following table.

	Carat	Cut	Color	Clarity	Depth	Table	Price	x	y	z
1	0.23	Ideal	E	SI2	61.5	55	330	3.95	3.98	2.43
2	0.21	Premium	E	SI1	59.8	61	327	3.89	3.84	2.31
3	0.23	Good	E	VS1	56.9	65	328	4.05	4.07	2.31
4	0.29	Premium	I	VS2	62.4	58	337	4.2	4.23	2.63
5	0.31	Good	J	SI2	63.3	58.1	338	4.34	4.35	2.75

It can be found that there are eight metrics associated with this diamond dataset:

- Carat: Weights of diamond (0.2-5.01);

- Cut: The quality of diamond arranged in an decreasing order: Ideal, Premium, Very Good, Good, Fair;

- Color: The color of each diamond ranging from J level(worst) to D level(best);

- Clarity: A measurement of how clear the diamond is (I1(worst), SI2, SI1, VS2, VS1, VVS2, VVS1, IF(best));

- x, y, z: length, width, depth of the diamond in mm;

- Depth: Total depth percentage = z / mean(x, y) = 2 * z / (x+y) (43-79);

- Table: Width of top of diamond relative to widest point (43 – 95)

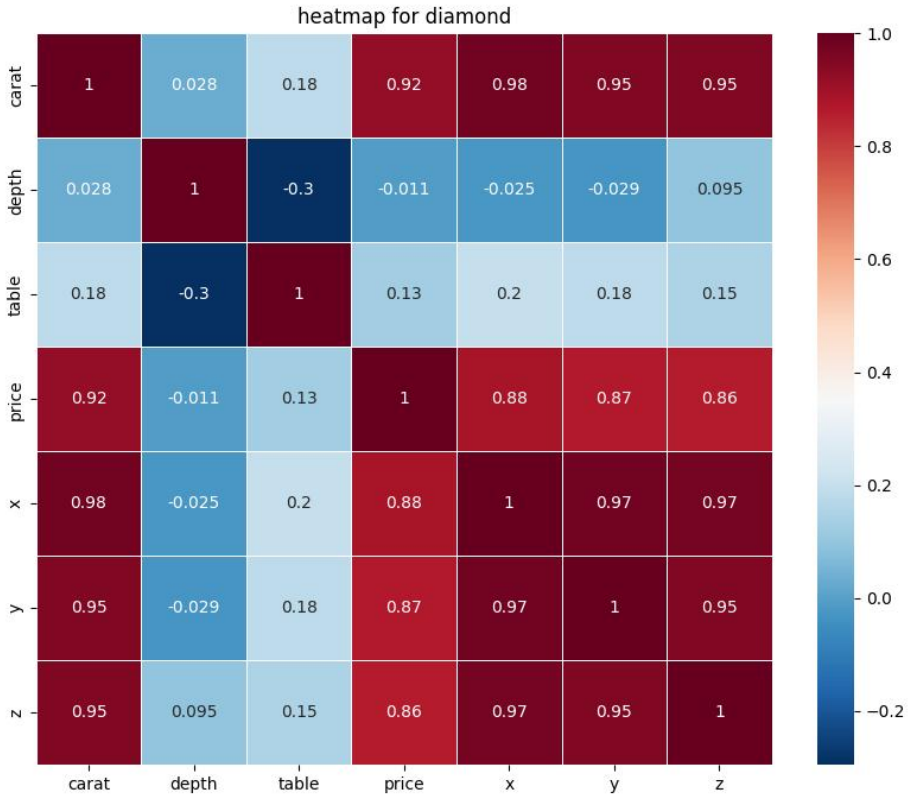
In addition to these features, there is the target variable what we would like to predict:

- price: price in US dollars ($326 – $18,823).

As for Gas Turbine CO and NOx Emission dataset, using the drop function we picked NOx to model and dropped CO. From the table below, it can be seen that after dropping, gas now does not include CO but only with 10 sensor measurements. There are 5 CSV files for each year. Concatenate all data points and add a column for the corresponding year and treat it as a categorical feature.

	AT	AP	AH	AFDP	GTEP	TIT	TAT	TEY	CDP	NOX	year
0	4.5878	1018.7	83.675	3.5758	23.979	1086.2	549.83	134.67	11.898	81.952	2011
1	4.2932	1018.3	84.235	3.5709	23.951	1086.1	550.05	134.67	11.892	82.377	2011
2	3.9045	1018.4	84.858	3.5828	23.99	1086.5	550.19	135.1	12.042	83.776	2011
3	3.7436	1018.3	85.434	3.5808	23.911	1086.5	550.17	135.03	11.99	82.505	2011
4	3.7516	1017.8	85.182	3.5781	23.917	1085.9	550	134.67	11.91	82.028	2011

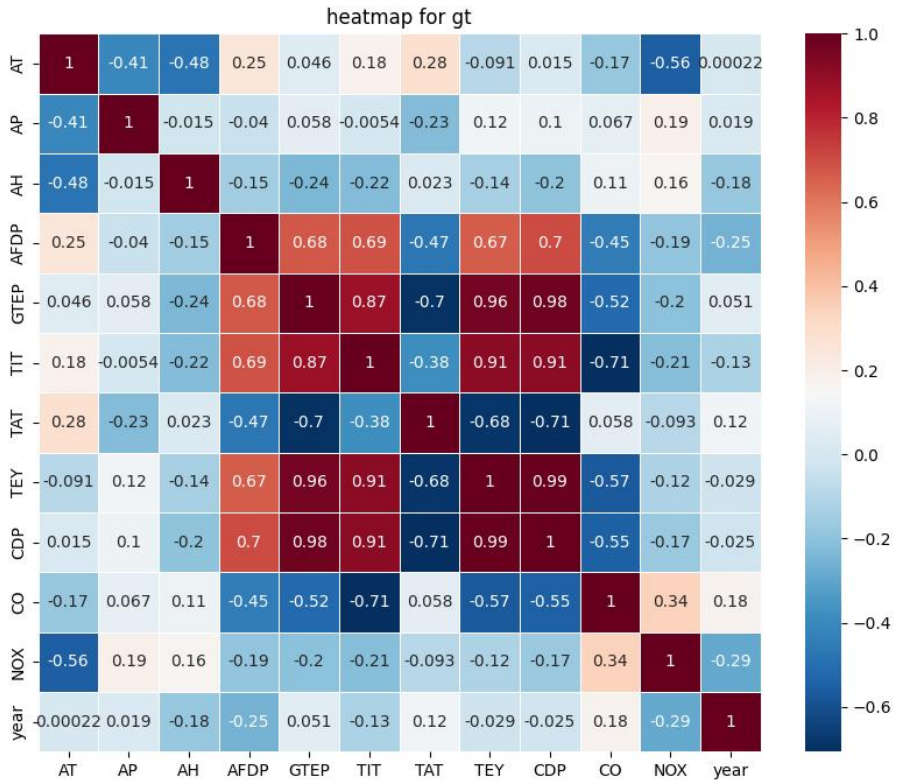
Heatmap of the Pearson Correlation Matrix of Diamond dataset is displayed below:



From the plot above, it can be seen that the features having the highest absolute correlation with the target variable price are **carat, x, y and z**.

The relationship between price and carats is highly correlated, which is logical since a pure diamond will always be more expensive than one that is less pure. Furthermore, as the size of the diamond increases, so does its weight, indicating a larger quantity of diamond, resulting in a higher price. On the other hand, the correlation between depth and table and the price is low, which makes sense as these factors do not directly influence the price.

Heatmap of the Pearson Correlation Matrix of Gas Emission dataset is displayed below:



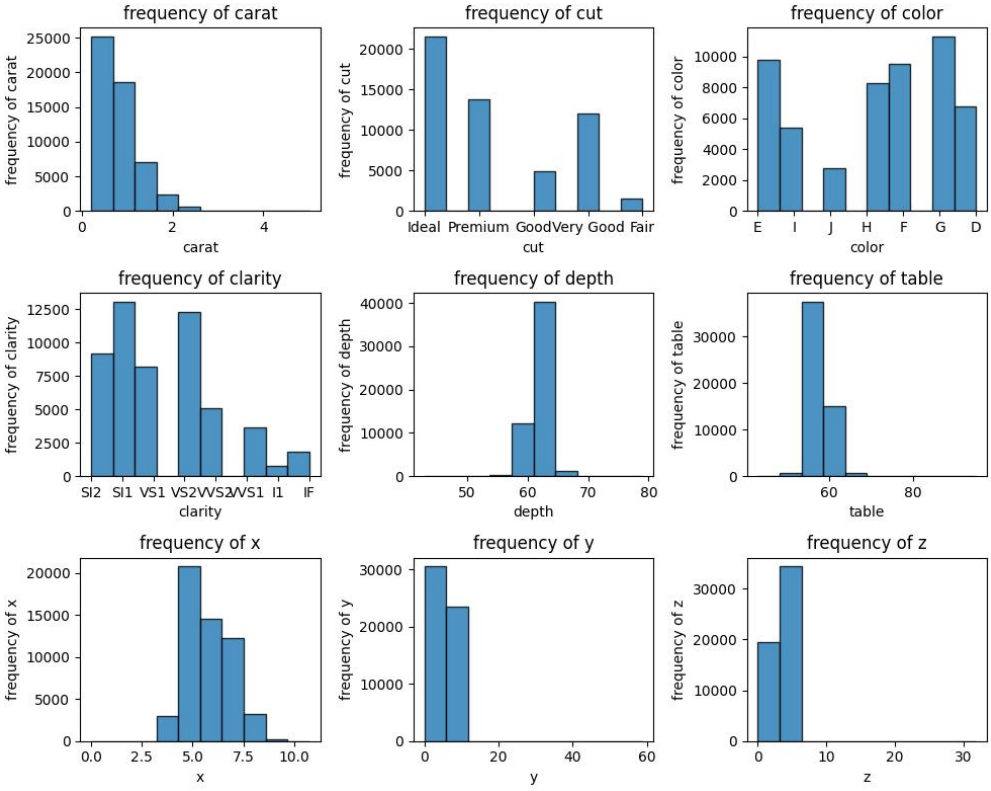
From the plot above, it can be seen that the features having the highest absolute correlation with the target variable NOX are **AT, CO and TIT**.

The Gas Emission dataset exhibits weaker absolute magnitudes of correlation compared to the diamond dataset, which is possibly due to the fact that the gas emission correlation involves many hidden and complex dependencies spread across multiple factors. This is in contrast to the concise features that accurately characterize diamonds. Notably, all Gas Emission features that are highly correlated with the CO target variable are related to Turbine measurements. Furthermore, all features exhibit a negative correlation, indicating that as the target variable value increases, the other variables' values are expected to decrease.

Question 1.2: Plot the histogram of numerical features. What preprocessing can be done if the distribution of a feature has high skewness?

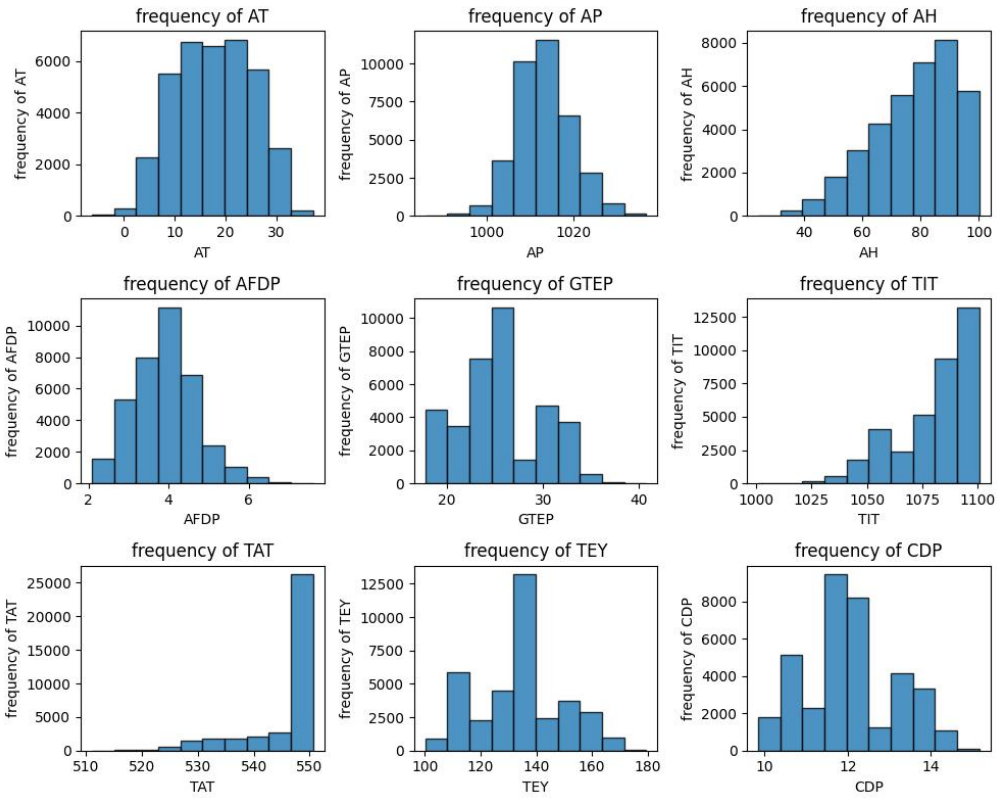
Histograms of Numerical Features of Diamond Dataset:

The histograms of the X dimension, Table and Depth Features in the Diamond dataset appear to be centered around a mean and resemble a non-zero mean Gaussian Distribution. However, the Y and Z dimensions have a narrower range and exhibit an impulsive or dirac-like distribution. These variations may be due to constraints in the manufacturing process or the instruments used. The Carat Feature is left-skewed, indicating that there are more lower purity diamonds than highly pure ones, possibly due to market demand for lower-carat diamonds.



Histograms of Numerical Features of Gas Turbine Emission Dataset:

The first two features, Ambient Temperature and Ambient Pressure, have almost symmetrical, non-zero mean Gaussian distributions. The other four features, Ambient Humidity, AFDP, GTEP, TIT, and TEY, have skewed distributions with two being positively skewed and two being negatively skewed. The TAT feature has values that are closely spaced within a narrow range. Skewed features may not be interpreted correctly by machine learning models, as these models assume normal statistical distributions for input features. Algorithms such as Least Squares regression and Mean square require the mean to represent central tendency in the case of skewness, but the mean may not be a good representation for skewed features.



If a feature has a highly skewed distribution, there are several preprocessing steps that can be taken to address it.

- One possible approach is to apply a log transformation, which can be particularly useful for positively skewed features.

- Another technique is the Box-Cox transformation, which aims to transform the feature into a normal distribution.

- Winsorization involves replacing extreme values with the closest values within a specified range.

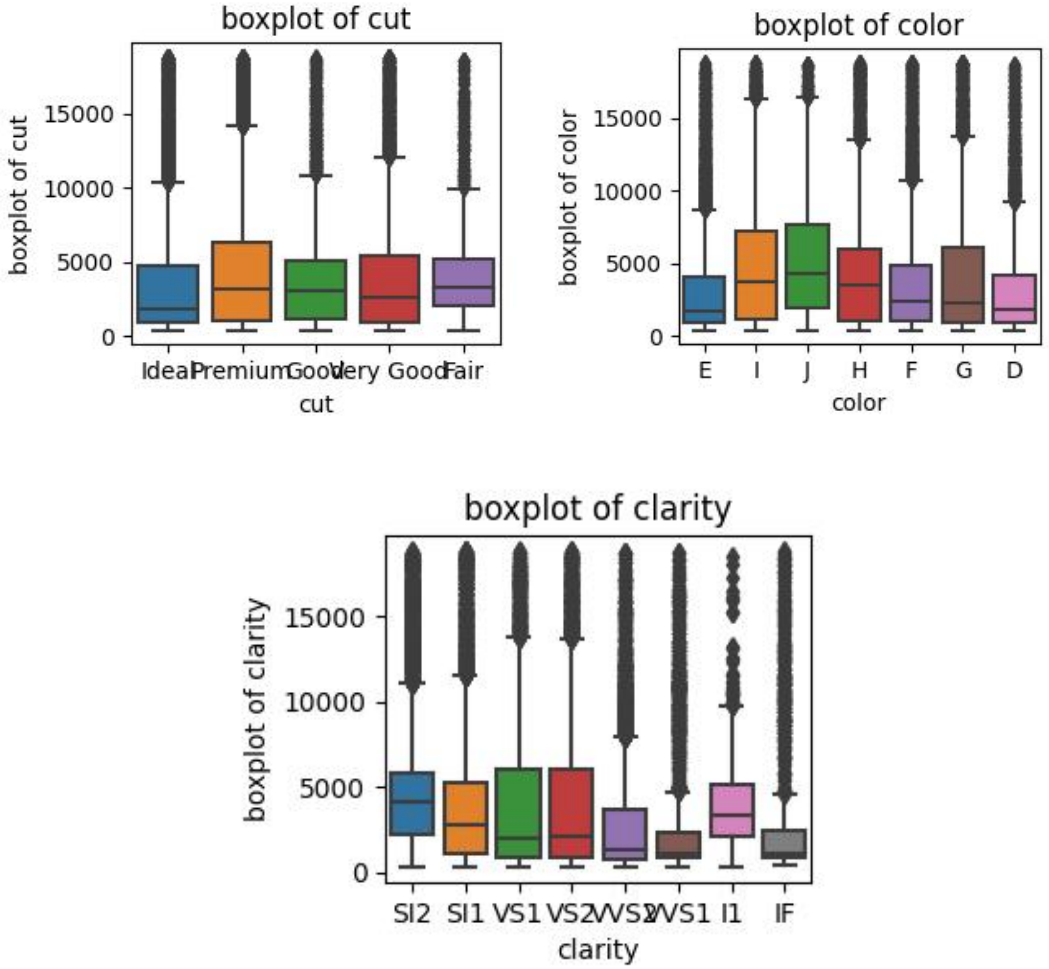
- Standardization scales the feature to have a mean of zero and a standard deviation of one.

- Finally, binning can be used to categorize the feature's values into bins and reduce the effect of extreme values. It's important to choose the right technique depending on the data distribution and the analysis requirements.

Question 1.3: Construct and inspect the box plot of categorical features vs target variable. What do you find?

Box Plot: A box plot is a way to visually represent a dataset that shows the distribution of data on a number line. It consists of a box that represents the middle 50% of the data, with a line in the middle that shows the median. The whiskers extend from the box to show the range of the data, except for outliers, which are represented by individual points beyond the whiskers. This type of plot is useful for quickly identifying the main features of a dataset, such as central tendency, spread, and outliers. It is commonly used in data analysis and statistics.

It is shown as below the boxplot of Categorical features vs target variable in the Diamond dataset:



It is worth to mention that the box plots of other features vs target variable are impractical and difficult to distinguish from them. And price vs cut, color and clarity are the noticeable plots to be claimed and demonstrated further in this report.

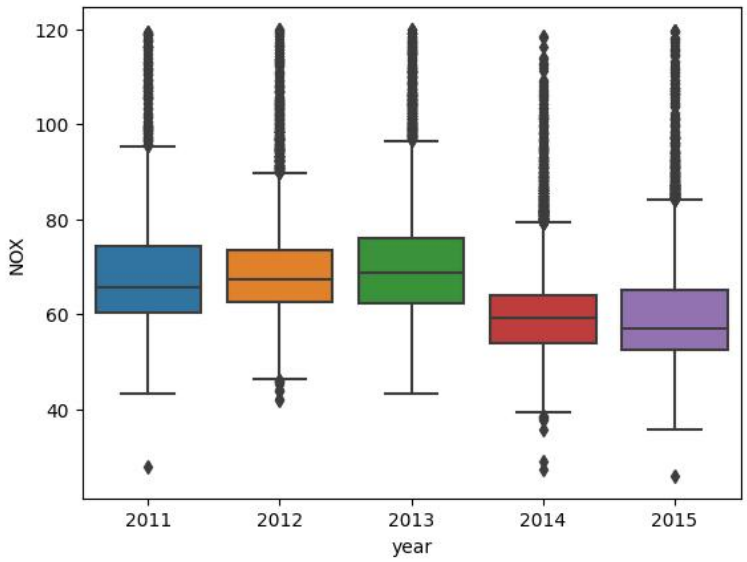
Firstly, from **the box plot of price vs cut,** the median of premium cut diamond reached the largest with the amount of around 2500 or even higher. It is literally against our intuition that ideal cut should approach an overall higher median. Conversely, it remains

the lowest median among all the ranges of cut. And moreover, the median of very good cut is smaller than good, which is lower than the fair cut. They are not a trending high or monotonously low with accordance to the intuitive sense of good or bad cut for a diamond.

Secondly, regarding **the color feature,** it is noticed that the colors J and I have a small number of very low outliers, whereas the other colors do not exhibit such outliers. Additionally, G, E, and D colors are highly skewed, while I, J, and H colors are more evenly distributed around the median.

Lastly, **the box plot price vs clarity** suggests that diamonds with clarity grades of 'VS1' and 'VS2' are more predictable by the model. Moreover, the clarity grades of 'I1', 'SI2', and 'IF/WS1' seem to be related to a clear set of prices. The clarity categories of 'IF' and 'WS1' have the largest number of outliers and their prices are skewed mostly to the right of the median price. Diamonds with 'SI1' clarity have a price range almost identical to that of 'VS1' but with a comparatively higher number of outliers. These diamonds have a symmetric distribution around their median price. On the other hand, the clarity grades of 'VS1' and 'VS2' are very similar to each other in terms of median, minimum, and maximum prices.

It is shown as below the boxplot of Categorical features vs target variable in the GT Emission dataset:



The box plot of NOX emission vs year indicates that in 2013, it reaches a peak of emission amount with a majority of NOX compared with other four years. Moreover, 2013 contributes to the fewest outliers among all. From a trending point of view, we can conclude that after 2013, both NOX emission in 2014 and 2015 dropped apparently. But the outliers also mostly come from 2014 and 2015. As for 2011 and 2012 become least predictable ones in this dataset.

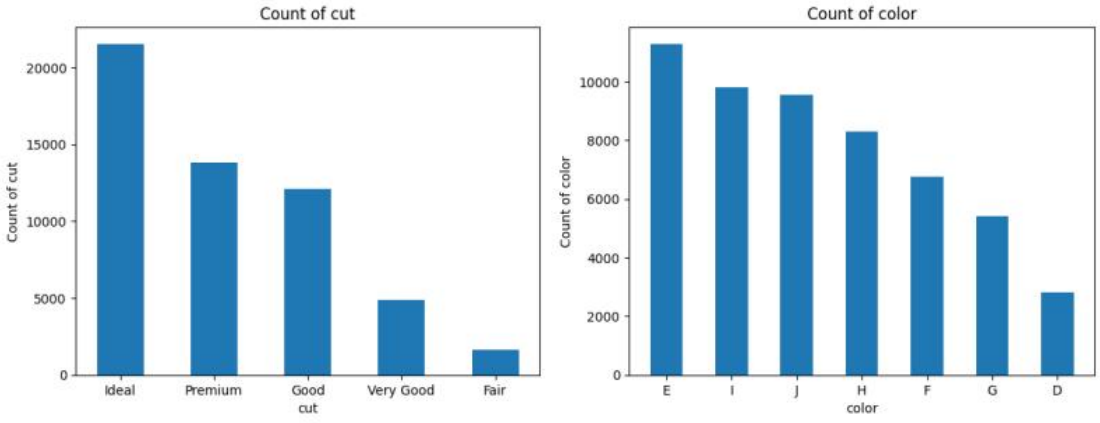
For the Diamonds dataset, plot the counts by color, cut and clarity. Here we defined a function named *plot_count()* as shown below to plot the counts of diamonds by color,

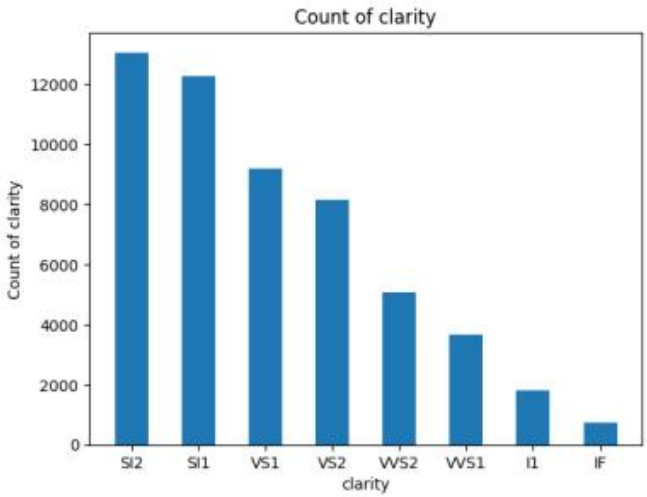
cut and clarity.

```
def plot_count(dataset,features):
    for feature in features:
        x_axis = list(dataset[feature].unique())
        y_axis = dataset[feature].value_counts().to_list()
        plt.bar(x_axis, y_axis, width=0.5)
        plt.ylabel(f"Count of {str(feature)}")
        plt.xlabel(f"{str(feature)}")
        plt.title(f"Count of {str(feature)}")
        plt.show()
plot_count(diamond_data,diamonds_labels)
```

The cut vs counts plot indicates that the majority of diamonds in the dataset have an ideal cut, followed by premium, good, very good, and fair cuts in decreasing order. Interestingly, the ideal cut accounts for a larger portion of the dataset than the premium cut, and the good cut category has a larger share than the very good cut category. This suggests that there is a better market demand for good to moderate quality cut diamonds at a moderate price rather than more expensive premium diamonds.

The color and clarity vs count plots show a consistent pattern of higher diamond production based on certain colors and clarity, with no increasing or decreasing trends.

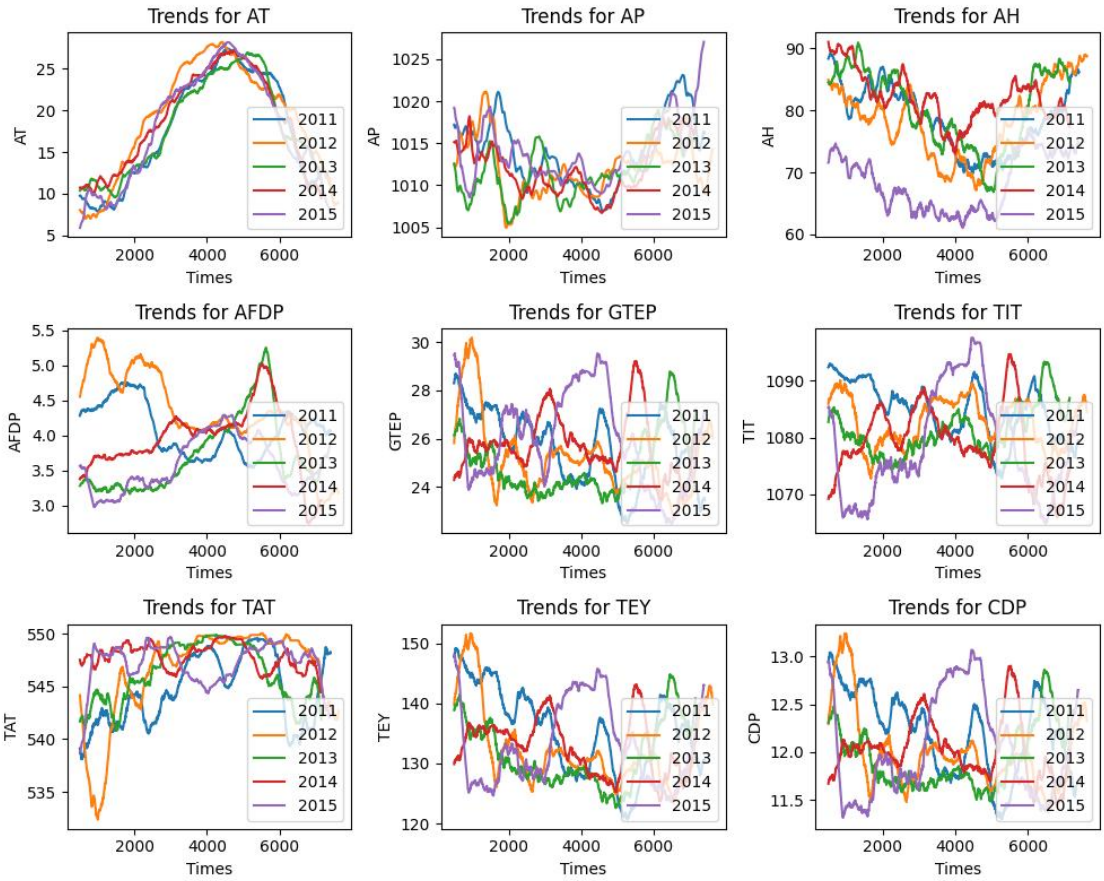




Question 1.4: For the Gas Emission dataset, plot the yearly trends for each feature and compare them. The data points don't have timestamps but you may assume the indices are times.

To plot the yearly trends for each feature in the Gas Emission dataset, we defined a function named *plot_trend()* and had the dataset smooth processed as the tremendous number of the original data.

It is worth to mention here that before **smooth processing**, the trending plot are highly frequency distributed causing trouble with distinguishing each performance and trends of each year. After smooth processing with 500, we obtained a considerable smooth plot for yearly trends of each feature as shown below. Here we plotted each year appended with a colour legend to let it put more clearly.



When we examine the gas emission trends over the years individually, we can identify some common patterns.

Gas sensor measurements like GTEP, TEY, and CDP exhibit similar variations in trends within each year as well as across all the years, but with different ranges of values for each year.

Conversely, the gas measurements for AH, TIT, and TAT show an opposite trend. Specifically, while the previous set of emission measurements decreased towards the middle of each year and then increased towards the end of the year, this set of gas

measurements had higher values at the start and end of each year and lower values at the midpoint.

This suggests that while the overall emission of GT gases needs to be minimized, the constituent gas measurements that characterize the environmental impacts of gas emission do not necessarily need to be minimized. In other words, some gases may be emitted at higher levels as long as other gases decrease proportionately.

- ## QUESTION 2: Standardization

Question 2.1: Standardize feature columns and prepare them for training.

Standardization: Many machine learning models require standardized datasets for optimal performance. When features have significantly different scales or variances, some features may dominate the model's objective function, resulting in poor performance.

Standardization involves transforming the data to have zero mean and unit variance, resulting in features that resemble a standard normal distribution. This transformation is necessary for many models to work properly because they assume that the features are normally distributed with a mean of zero and a variance of one.

Features that are not standardized may cause the model to learn incorrectly from the data and lead to poor predictions.

Therefore, before training the dataset, we firstly standardized the feature columns in the diamond dataset in the following code. As some features are nonnumerical like cut, colour and clarity, we used numbers to encode them and defined a *encode_feature()* function as below to prepare them for the latter training process.

To put it more clearly, we encoded **cut** with accordance to the following metrics:

Fair -1, Good - 2, Ideal - 3, Premium - 4, Very Good - 5.

Furthermore, we encoded **colour** with accordance to the following metrics:

J - 1, I - 2, H - 3, G - 4, F - 5, E - 6, D - 7.

And finally, we encoded **clarity** with accordance to the following metrics:

I1 - 1, SI2 - 2, SI1 - 3, VS2 - 4, VS1 - 5, WS2 - 6, WS1 - 7, IF - 8.

```
# Standardization
# for diamonds encoding
cut_encoding = {'Fair' : 1, 'Good' : 2, 'Ideal' : 3, 'Premium' : 4, 'Very Good' : 5}
color_encoding = {'J' : 1, 'I' : 2, 'H' : 3, 'G' : 4, 'F' : 5, 'E' : 6, 'D' : 7}
clarity_encoding = {'I1' : 1, 'SI2' : 2, 'SI1' : 3, 'VS2' : 4, 'VS1' : 5, 'VVS2' : 6, 'VVS1' : 7, 'IF' : 8}

def encode_feature(df, feature, encoding_dict):
  features = df[feature]
  encoded_feats = [encoding_dict[f] for f in features]
  df[feature] = encoded_feats

encode_feature(diamond_data, 'cut', cut_encoding)
encode_feature(diamond_data, 'color', color_encoding)
encode_feature(diamond_data, 'clarity', clarity_encoding)
```

After processing the dataset and standardizing the feature columns, we implemented head() function to reach an overview of the standardized diamond dataset in the following figure. After standardization, the metrics of cut, color and clarity changed from characters to the numerical data which is easier to handle in the following training process.

	carat	cut	color	clarity	depth	table	price	x	y	z
0	-1.198168	-0.538099	0.937163	-1.245215	-0.174092	-1.099672	-0.903594	-1.587837	-1.536196	-1.571129
1	-1.240361	0.434949	0.937163	-0.638095	-1.360738	1.585529	-0.904346	-1.641325	-1.658774	-1.741175
2	-1.198168	-1.511147	0.937163	0.576145	-3.385019	3.375663	-0.904095	-1.498691	-1.457395	-1.741175
3	-1.071587	0.434949	-1.414272	-0.030975	0.454133	0.242928	-0.901839	-1.364971	-1.317305	-1.287720
4	-1.029394	-1.511147	-2.002131	-1.245215	1.082358	0.242928	-0.901588	-1.240167	-1.212238	-1.117674

Compared with the table 1 in the question 1, we found that not only the cut, color and clarity features transformed severely, other features were also standardized into a distribution close to 0 and 1. This transformation is necessary for many models to work properly because they assume that the features are normally distributed with a mean of zero and a variance of one.

Similarly, we put the GT Emission dataset's standardized form in the following table. Since the original data in the GT dataset does not include any characters, we simply utilized *preprocessing()* to complete standardization here.

	AT	AP	AH	AFDP	GTEP	TIT	TAT	TEY	CDP	NOX	year
0	-1.762362	0.871052	0.401627	-0.451875	-0.377702	0.272119	0.536589	0.074502	-0.149273	1.426499	2011
1	-1.801920	0.809164	0.440351	-0.458207	-0.384376	0.266417	0.568742	0.074502	-0.154783	1.462891	2011
2	-1.854113	0.824636	0.483432	-0.442831	-0.375081	0.289227	0.589203	0.102033	-0.017015	1.582687	2011
3	-1.875718	0.809164	0.523263	-0.445415	-0.393909	0.289227	0.586280	0.097551	-0.064774	1.473852	2011
4	-1.874644	0.731804	0.505837	-0.448904	-0.392479	0.255012	0.561434	0.074502	-0.138251	1.433006	2011

Question 2.2: You **may use these functions to select features that yield better regression results (especially in the classical models). Describe how this step qualitatively affects the performance of your models in terms of test RMSE. Is it true for all model types? Also list two features for either dataset that has the lowest MI w.r.t to the target.**

sklearn.feature_selection.mutual_info_regression estimates the mutual information between the label and each feature.

Mutual information is a non-negative value that measures the dependence between two variables, and it is zero if the variables are independent. Higher mutual information values indicate a higher degree of dependence.

sklearn.feature_selection.f_regression provides F scores, which allow for a comparison of the significance of the model's improvement as new variables are added.

Top five features in the datasets using two feature selection

Diamond dataset	**MI**	carat	y	x	z	clarity
	F	carat	x	y	z	color
Gas Emission dataset	**MI**	TIT	TEY	AT	GTEP	CDP
	F	AT	TIT	GTEP	AP	AFDP

All the results of mutual_info_regression:

Diamonds: ['carat', 'y', 'x', 'z', 'clarity', 'color', 'cut', 'table', 'depth']

GT: ['TIT', 'TEY', 'AT', 'GTEP', 'CDP', 'AFDP', 'TAT', 'AP', 'AH']

So the two features have the lowest MI is **table** and **depth** for Diamonds, **AP** and **AH** for GT.

From the table above, we can conclude that selecting top features can have both positive and negative effects on the performance of a model in terms of test RMSE.

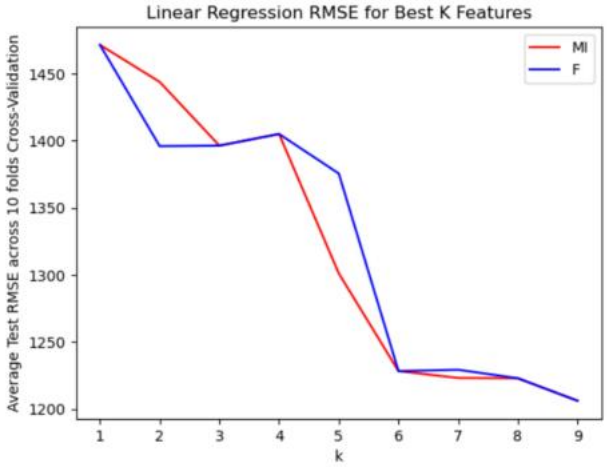
On one hand, selecting top features can improve the performance of the model by reducing the dimension of the input data and removing noisy or irrelevant features that may hinder the model's ability to learn meaningful patterns from the data. By selecting only the most informative features, the model can be better equipped to capture the underlying relationships between the input and output variables and make more accurate predictions.

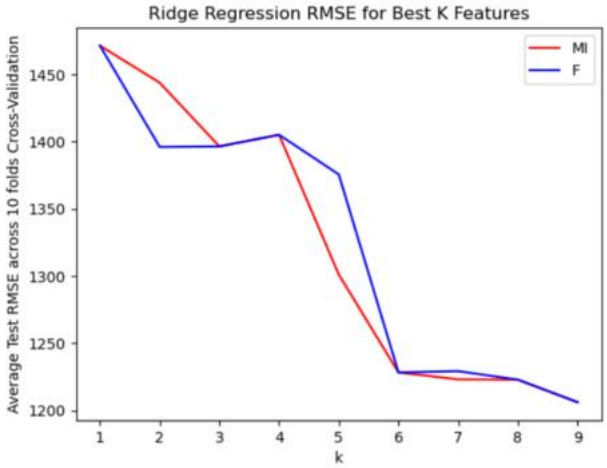
On the other hand, selecting top features can also have negative effects on the performance of the model if important information is lost by excluding relevant features from the input data. If the model is unable to capture the complexity of the data due to the loss of important features, it may result in higher test RMSE and lower predictive accuracy.

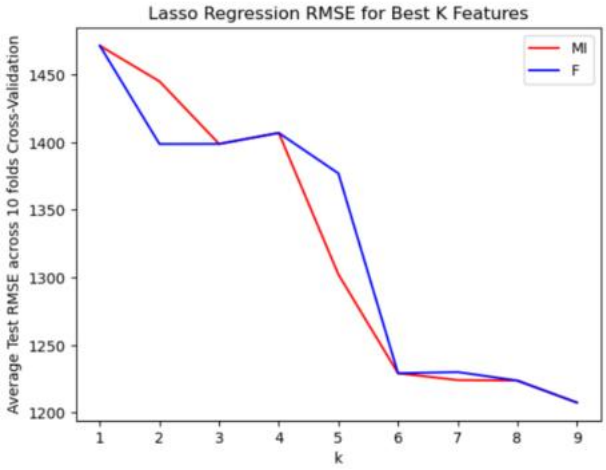
In general, the effect of selecting top features on model performance in terms of test

RMSE will depend on the specific data set and the characteristics of the features being considered. It is important to carefully evaluate the trade-off between the reduction in dimension and the potential loss of relevant information when selecting top features for a given model. Additionally, it is important to use cross-validation techniques to assess the model's performance on multiple test sets and to ensure that the model's performance is not being overestimated due to chance.

	carat	y	x	z	clarity
0	-1.198168	-1.536196	-1.587837	-1.571129	-1.245215
1	-1.240361	-1.658774	-1.641325	-1.741175	-0.638095
2	-1.198168	-1.457395	-1.498691	-1.741175	0.576145
3	-1.071587	-1.317305	-1.364971	-1.287720	-0.030975
4	-1.029394	-1.212238	-1.240167	-1.117674	-1.245215






Lasso Regression RMSE for Best K Features
MI
F
Average Test RMSE across 10 folds Cross-Validation
1450
1400
1350
1300
1250
1200
1
2
3
4
5
6
7
8
9
k

- **QUESTION 3: Evaluation**

For random forest model, measure "Out-of-Bag Error" (OOB) as well. Explain what OOB error and R^2 score means given this link.

Out-of-Bag (OOB) Error is a method for estimating the prediction error of a supervised learning algorithm, such as a decision tree or random forest.

When building a random forest model, each tree in the forest is trained on a bootstrap sample of the original data, which means that some of the data is left out of the training process. This left-out data is called the out-of-bag data, and it can be used to estimate the prediction error of the model.

The OOB error is calculated by predicting the out-of-bag data using the trees that were not trained on that particular data point. The predictions are then compared to the true values of the out-of-bag data, and the average prediction error is calculated.

The OOB error is a useful way to estimate the prediction error of a random forest model without the need for a separate validation set. It can also be used to tune the parameters of the model, such as the number of trees, to find the optimal number of trees that minimize the prediction error.

R^2 score(the coefficient of determination) is a commonly used metric to evaluate the performance of regression models, including random forest regression models.

R^2 score is a statistical measure that represents the proportion of variance in the target variable that is explained by the independent variables in the model. It ranges from 0 to 1, where 0 indicates that the model does not explain any of the variance in the target variable, and 1 indicates that the model explains all of the variance in the target variable.

In the case of a random forest regression model, the R^2 score measures the goodness of fit of the model by comparing the total sum of squares (TSS) to the residual sum of squares (RSS). TSS is the total variance in the target variable, and RSS is the variance that is not explained by the model. The formula for calculating the R^2 score is:

$$R^2 = 1 - (RSS/TSS)$$

A higherR^2score indicates that the model is better at explaining the variance in the target variable. However, it is important to note that a high R^2score does not necessarily mean that the model is accurate or predictive. Other metrics, such as mean squared error or mean absolute error, should also be considered in conjunction with R^2score to evaluate the performance of the model.

Our outcomes of OOB and R^2score for best random forest model of diamond dataset and GT dataset are shown in the following table separately.

	Diamond	GT
OOB	0.903	0.5951
R^2 score	0.9202	0.6063

- ## QUESTION 4: Linear Regression

What is the objective function? Train three models: (a) ordinary least squares (linear regression without regularization), (b) Lasso and (c) Ridge regression, and answer the following questions.

Question 4.1: Explain how each regularization scheme affects the learned parameter set.

In linear regression, **the objective function** is typically the sum of the squared errors between the predicted values of the dependent variable (i.e., the variable we want to predict) and the actual values of the dependent variable in the training data. The goal of linear regression is to find the values of the model parameters that minimize the objective function.

The objective function can be defined as a mathematical expression that measures the performance of a model or system and guides the optimization process.

In the context of linear regression, the objective function is used to measure how well the model fits the training data, by calculating the sum of the squared differences between the predicted values and the actual values of the dependent variable. The goal is to minimize the objective function by adjusting the values of the model parameters, which leads to a better fit to the data and improved predictive performance.

The choice of objective function is a critical aspect of optimization and machine learning, as it determines the quality of the learned model or system. It is important to select an objective function that reflects the specific goals and constraints of the problem at hand, and to balance the trade-off between model complexity and performance.

Then we began to explain how each regularization scheme affects the learned parameter set. Since we have outlined top features in diamond datasets and Gas Emission dataset, for each feature, we passed its data into RMSE score calculation with different regulation schemes including the following nine items. And we showed each result RMSE score per feature in the following table.

linear regression,

linear regression and with mutual_info_regression,

linear regression and with f_regression,

Ridge regression,

Ridge regression and with mutual_info_regression,

Ridge regression and with f_regression,

Lassso regression,

Lasso regression and with mutual_info_regression,

Lasso regression and with f_regression.

Different regulation schemes' RMSE scores with diamond dataset:

RMSE score	Top 1	Top 2	Top 3	Top 4	Top 5	Top 6	Top 7	Top 8
Linear	-0.3023	-0.3023	-0.3023	-0.3023	-0.3023	-0.3023	-0.3023	-0.3023
Linear +mutual	-0.3688	-0.3619	-0.35	-0.3522	-0.3261	-0.3079	-0.3066	-0.3065
Linear +f	-0.3688	-0.3499	-0.35	-0.3522	-0.3448	-0.3079	-0.3081	-0.3065
Ridge	-0.3023	-0.3023	-0.3023	-0.3023	-0.3023	-0.3023	-0.3023	-0.3023
Ridge +mutual	-0.3688	-0.3619	-0.3501	-0.3522	-0.3262	-0.3079	-0.3066	-0.3065
Ridge +f	-0.3688	-0.3499	-0.3501	-0.3522	-0.3448	-0.3079	-0.3081	-0.3065
Lasso	-0.8875	-0.8875	-0.8875	-0.8875	-0.8875	-0.8875	-0.8875	-0.8875
Lasso +mutual	-0.8875	-0.8875	-0.8875	-0.8875	-0.8875	-0.8875	-0.8875	-0.8875
Lasso +f	-0.8875	-0.8875	-0.8875	-0.8875	-0.8875	-0.8875	-0.8875	-0.8875

Different regulation schemes' RMSE scores with Gas Emission dataset:

RMSE score	Top 1	Top 2	Top 3	Top 4	Top 5	Top 6	Top 7	Top 8
Linear	-0.7382	-0.7382	-0.7382	-0.7382	-0.7382	-0.7382	-0.7382	-0.7382
Linear +mutual	-0.9985	-0.9841	-0.8208	-0.8344	-0.8097	-0.8218	-0.7656	-0.7661
Linear +f	-0.8475	-0.849	-0.8274	-0.8325	-0.8394	-0.8511	-0.8178	-0.8061
Ridge	-0.7379	-0.7379	-0.7379	-0.7379	-0.7379	-0.7379	-0.7379	-0.7379
Ridge +mutual	-0.9985	-0.9841	-0.8208	-0.8342	-0.8096	-0.8217	-0.7656	-0.7661
Ridge +f	-0.8475	-0.849	-0.8274	-0.8325	-0.8394	-0.8510	-0.8178	-0.8059
Lasso	-1.0059	-1.0059	-1.0059	-1.0059	-1.0059	-1.0059	-1.0059	-1.0059
Lasso +mutual	-1.0059	-1.0059	-1.0059	-1.0059	-1.0059	-1.0059	-1.0059	-1.0059
Lasso +f	-1.0059	-1.0059	-1.0059	-1.0059	-1.0059	-1.0059	-1.0059	-1.0059

Each regularization scheme affects the learned parameter set. Linear regression aims to minimize the sum of the squared residuals between the predicted and actual values. However, when the number of features is large, this can lead to overfitting, where the model learns noise instead of signal, leading to poor generalization on unseen data. Regularization is a technique that introduces additional constraints to the model to prevent overfitting.

(a) Ordinary Least Squares (OLS) :

OLS finds the parameter values that minimize the sum of the squared residuals between the predicted and actual values. It does not introduce any additional constraints on the model. Therefore, it can be prone to overfitting, especially when the number of features is large. The learned parameter set tends to have large values, which can result in overfitting.

(b) Lasso regression:

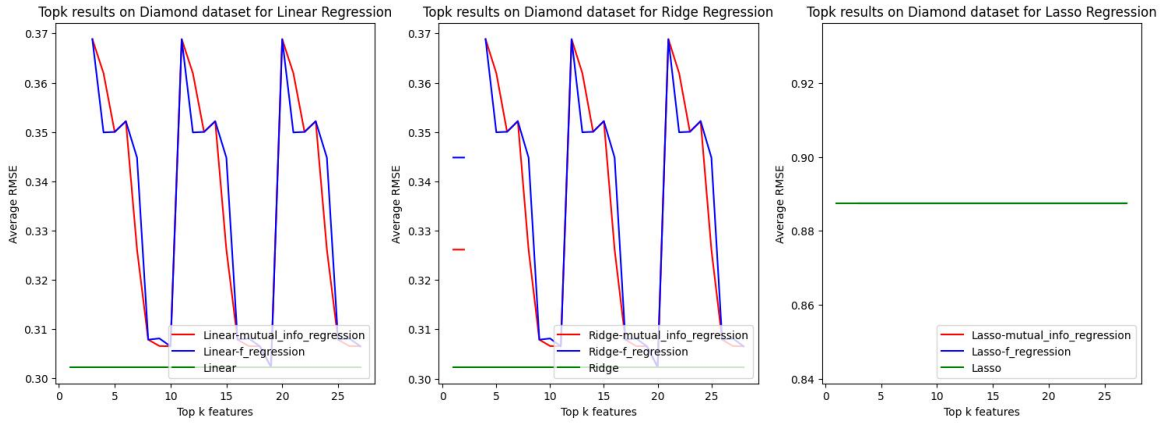
Lasso regression adds a penalty term to the OLS cost function, which is the sum of the absolute values of the parameters, multiplied by a regularization parameter. This penalty term encourages the model to shrink some of the coefficients towards zero, resulting in sparsity, where some coefficients become exactly zero. This effect is useful for feature selection, as it automatically selects the most important features while discarding the less important ones. Therefore, the learned parameter set tends to have smaller values, and some coefficients can become exactly zero.

(c) Ridge regression:

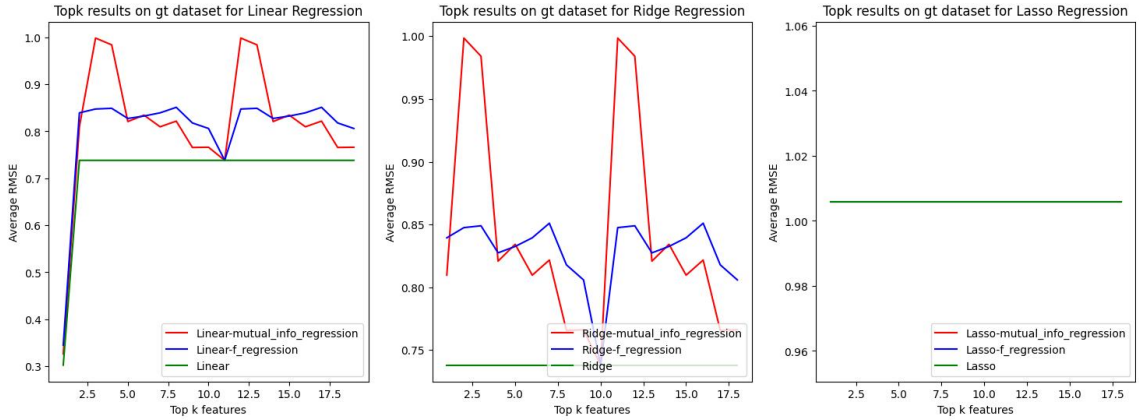
Ridge regression adds a penalty term to the OLS cost function, which is the sum of the squares of the parameters, multiplied by a regularization parameter. This penalty term encourages the model to reduce the magnitude of all parameter values, but none of them are exactly zero. This effect is useful when all features are important and need to be considered. Therefore, the learned parameter set tends to have smaller values overall, but no coefficients are exactly zero.

In summary, regularization methods such as Lasso and Ridge regression introduce additional constraints to the model to prevent overfitting. Lasso regression tends to produce sparse solutions, while Ridge regression does not. The learned parameter set tends to have smaller values in both cases, but the specific effect depends on the regularization parameter and the nature of the data.

Top8 results on Diamond dataset for Linear, Ridge and Lasso Regression:



Top8 results on Gas Emission dataset for Linear, Ridge and Lasso Regression:



To obtain a clearer comparison of three regression with and without regulation, we plotted Top8 results on both Diamond and Gas Emission dataset for Linear, Ridge and Lasso Regression.

Question 4.2: Report your choice of the best regularization scheme along with the optimal penalty parameter and explain how you computed it.

The best regularization scheme for Diamond dataset is **Linear Regression,** and its optimal penalty parameter is shown as below.

- best k: 6

- best score function: function f_regression at 0x000001C491599790

- model: Linear

- Best score with select k best and Linear Regression: -0.3009287838936753

The best regularization scheme for Gas Emission dataset is **Lasso(alpha=0.001),** and its optimal penalty parameter is shown as below.

- best k: 9

- best score function: function mutual_info_regression at 0x000001C491BDFCA0

- model: Lasso(alpha=0.001)

- alpha: 0.001

- Best score with select k best and Ridge Regression: -0.3009287838936753

Regularization is a technique used to prevent overfitting in machine learning models by adding a penalty term to the loss function. This penalty term discourages the model from assigning too much importance to any particular feature or set of features. There are several regularization schemes available, including Lasso regularization, Ridge regularization.

Lasso regularization, adds a penalty term that is proportional to the absolute value of the model's weights. Ridge regularization, adds a penalty term that is proportional to the square of the model's weights.

To compute the optimal penalty parameter for regularization, we used **grid search.**

Grid search can be used to find the optimal penalty parameter. In grid search, a range of penalty parameter values is specified, and the model is trained and evaluated for each value in the range. The penalty parameter value that results in the lowest validation error is selected as the optimal value.

Firstly, we defined the range of values for the penalty parameter to search over. And then we created a grid of hyperparameter values. This can be done using the GridSearchCV function in scikit-learn. Specify the hyperparameters to tune, the range of values to search over, and the number of cross-validation folds to use. Here we used Ridge and Lasso as penalty parameter and set alpha as 0.001 and 0.0001. And finally we defined a grid_search_result() function to print out the results of calculation.

```
# Finding the optimal penalty parameter
location = "cachedir"
```

```
memory = Memory(location=location, verbose=10)

pipe_ = Pipeline([
    ('kbest', SelectKBest()),
    ('model', "passthrough")
], memory = memory)
param_grid = [{
        'kbest__score_func': (mutual_info_regression, f_regression),
        'kbest__k': (1, 2, 3, 4, 5, 6, 7, 8, 9),
        'model': [Ridge(), Lasso()],
        'model__alpha': [10.0**x for x in np.arange(-3,4)]
    }
]
# Finding the optimal penalty parameter
location = "cachedir"
memory = Memory(location=location, verbose=10)
pipe_ = Pipeline([
    ('kbest', SelectKBest()),
    ('model', "passthrough")
], memory = memory)

param_grid = [{
        'kbest__score_func': (mutual_info_regression, f_regression),
        'kbest__k': (1, 2, 3, 4, 5, 6, 7, 8, 9),
        'model': [Ridge(), Lasso()],
        'model__alpha': [10.0**x for x in np.arange(-3,4)]
    }
]
def _print_gridsearch_result(model, title):
    # print(model.cv_results_.keys())
    # print(f"Best estimator for {title}: ", model.best_estimator_)
    print(f"Best parameters for {title}: ", model.best_params_)
    print(f"Best score for {title}: ", model.best_score_)
grid_diamond = GridSearchCV(pipe_, param_grid = param_grid, cv = 10, n_jobs = -1, verbos
e = 1,
                    scoring = 'neg_root_mean_squared_error', return_train_score = True).
fit(diamond_standard_x, diamond_standard_y)
grid_gt = GridSearchCV(pipe_, param_grid = param_grid, cv = 10, n_jobs = -1, verbose = 1,

                    scoring = 'neg_root_mean_squared_error', return_train_score = True).
fit(gt_standard_x, gt_standard_y)
```

Question 4.3: Does feature standardization play a role in improving the model performance (in the cases with ridge regularization)? Justify your answer.

Feature standardization can play a role in improving the model performance when using Ridge regularization. Ridge regression adds a penalty term to the cost function based on the sum of the squares of the parameters. This penalty term is multiplied by a regularization parameter that determines the amount of shrinkage applied to the model coefficients.

Feature standardization involves scaling the features such that they have zero mean and unit variance. This scaling can help to prevent some features from dominating the optimization process and make the regularization parameter more effective. Without standardization, features with large variance will have a larger effect on the optimization than features with small variance. This can result in a model that is biased towards those features with larger variance.

By standardizing the features, each feature will have a similar scale, and the regularization parameter can have a more balanced effect on all the features. This can lead to better model performance, as the Ridge regression will be able to effectively balance the contributions of all the features to the model.

In summary, feature standardization can play a crucial role in improving the model performance when using Ridge regularization, as it can prevent some features from dominating the optimization process and make the regularization parameter more effective.

Question 4.4: Some linear regression packages return p-values for different features. What is the meaning of these p-values and how can you infer the most significant features?

P-value: In statistics, p-value is the probability of obtaining a test statistic as extreme as, or more extreme than, the observed test statistic under the null hypothesis.

In simpler terms, it is the probability of observing a result as extreme as or more extreme than the one we have, assuming that the null hypothesis is true. The null hypothesis is a statement we assume to be true until there is sufficient evidence to reject it.

P-values are used to determine the statistical significance of a result. If the p-value is less than a pre-specified significance level, it is considered statistically significant, and we reject the null hypothesis. If the p-value is greater than the significance level, we fail to reject the null hypothesis.

P-values for Diamond dataset:

y	2.11E-01
z	1.61E-01
cut	4.62E-56
depth	2.05E-113
x	1.33E-125
table	2.04E-133
carat	**0.00E+00**
color	**0.00E+00**
clarity	**0.00E+00**

P-values for Gas Emission dataset:

GTEP	4.23E-02
CDP	4.36E-03
AFDP	5.74E-16
AP	1.25E-186
AT	**0.00E+00**
AH	**0.00E+00**
TIT	**0.00E+00**
TAT	**0.00E+00**
TEY	**0.00E+00**

We implemented built-in function pvalue() to calculate each feature's p-value and sort it in an descending trend to approach the above two tables including Diamond and Gas Emission dataset. From the tables above, it indicates that y has the highest p-value in Diamond dataset and GTEP has the highest p-value in Gas Emission dataset. Therefore, based on the above description about p-value, we can infer that **the most significant features for Diamond dataset are carat, color and clarity. As for Gas Emission dataset, the most significant features are TEY, TAT, AT, TIT and AH.**

QUESTION 5: Polynomial Regression

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

- **Question 5.1: What are the most salient features? Why?**

We use the GridSearch, Fitting 10 folds for each of 6 candidates, totaling 60 fits. Scoring='neg_root_mean_squared_error'

```
poly_pipe_diamond_ = Pipeline([
    ('poly_transform', PolynomialFeatures()),
    ('model', Ridge(alpha=0.001))
],memory=memory)

poly_pipe_gt_ = Pipeline([
    ('poly_transform', PolynomialFeatures()),
    ('model', Lasso(alpha=0.001))
],memory=memory)

poly_param_= {
     'poly_transform__degree': np.arange(1,6,1)
}
```

Top 6 Salient features (Diamond) in order:

['carat', 'x', 'clarity','color', 'depth', 'z']

Most salient feature: carat

Top 9 Salient features (Gas Turbine) in order:

['TAT', 'TEY', 'TIT', 'AT', 'CDP', 'AT TEY^2', 'CDP^2', 'AFDP TAT', 'TIT TAT']

Most salient feature: TAT

- **Question 5.2: What degree of polynomial is best? How did you find the optimal degree? What does a very high-order polynomial imply about the fit on the training data? What about its performance on testing data?**

	mean_test_score	mean_train_score	param_poly_transform__degree
0	-3.078705e-01	-0.306212	1
1	-3.681719e-01	-0.195480	2
2	-2.228821e+00	-0.154887	3
3	-1.821926e+02	-0.143899	4
4	-1.242804e+04	-0.136352	5
5	-4.282573e+06	-0.131512	6

	mean_test_score	mean_train_score	param_poly_transform__degree
0	-0.734310	-0.691730	1
1	-0.595685	-0.504313	2
2	-0.595111	-0.459569	3
3	-0.660176	-0.433690	4
4	-0.705431	-0.414580	5

Polynomial Features of Diamonds

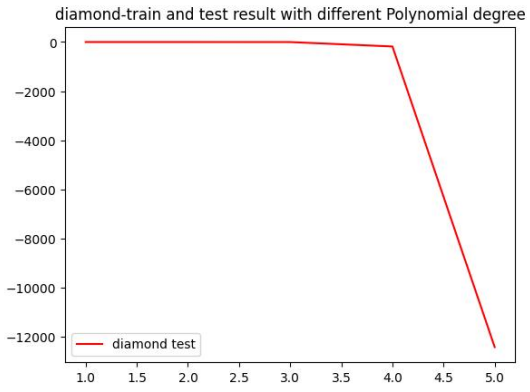
Best parameters: degree = 4 (less than 6)

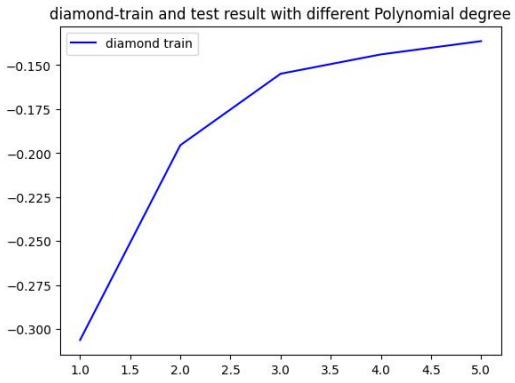
Best score = -0.306212

Polynomial Features of Gas Turbine：

Best parameters: degree = 3 (less than 6)

Best score = -0.691730





The optimal degree of a polynomial regression model can be determined where the model is trained on a subset of the data and tested on another subset to assess its performance. The degree that produces **the best performance on the testing data** is chosen as the optimal degree.

A very high-order polynomial can lead to **overfitting** of the training data. Overfitting occurs when the model becomes too complex and captures noise and random variations in the training data instead of the underlying trend.

As the order of the polynomial increases, the model becomes more complex, which can lead to more extreme fluctuations in the predicted values. Thus, a very high-order polynomial can imply that the fit is **highly sensitive** to small changes in the input variables. This can result in poor performance on the testing data, as the model is unable to capture the true underlying pattern in the data and is instead fitting to noise and random variations.

- **QUESTION 6: Neural Network**

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

- **Question 6.1: Adjust your network size (number of hidden neurons and depth), and weight decay as regularization. Find a good hyper-parameter set systematically (no more than 20 experiments in total).**

```
mlpr = Pipeline([
    ('model', MLPRegressor()),
], memory=memory)

param_list = {
    "model__hidden_layer_sizes": [(30, 40), (30, 60), (40, 40), (40, 60), (60, 60)],
    "model__activation": ["identity", "logistic", "tanh", "relu"],
    "model__solver": ["lbfgs", "sgd", "adam"],
}
```

We use the GridSearch, Fitting 5 folds for each of 60 candidates, totaling 300 fits.

Neural Network of Diamonds

Best parameters:

Model Activation': 'relu'

Model hidden layer sizes: (30, 60)

Model solver: 'sgd'

Best score = -0.2482196345112592

Original RMSE = 0.4982164534730454

Neural Network of Gas Turbine

Best parameters

Model activation: 'logistic'

Model hidden layer sizes: (30, 60)

Model solver: 'sgd'

Best score = -0.628648815583329

New RMSE = 0.7928737702707341

- **Question 6.2: How does the performance generally compare with linear regression? Why?**

  The performance with neural network is generally expected to **outperform** linear regression in many real-world problems.

  Linear regression is simple and computationally efficient, but may not be suitable for modeling more complex relationships between the variables. On the other hand, **neural network** can capture non-linear relationships between the input and output variables by using multiple layers of non-linear activation functions. This allows the network to learn more complex patterns and relationships in the data, making it more powerful and flexible than linear regression models.

  Additionally, MLPs can also perform feature learning by **automatically discovering relevant and informative features** from the input data, whereas linear regression models typically require pre-selected features.

- **Question 6.3: What activation function did you use for the output and why? You may use none.**

  Model activation: ["identity", "logistic", "tanh", "relu"]
  For Diamond, we choose **relu**.
  For Gas Turbine, we choose **logistic**.

  ReLU activation function is a non-linear function that sets all negative values to zero and keeps all positive values, while **logistic** is a non-linear function that maps input values to a range between 0 and 1.

- **Question 6.4: What is the risk of increasing the depth of the network too far?**

  As the depth of a neural network increases, the gradients used to update the weights during training can become very small or even vanish altogether. This can make it difficult to train the network effectively. On the other hand, too much depth of a neural network can cause the opposite problem of **vanishing gradients** - **exploding gradients**. This occurs when the gradients become too large and cause the weights to update too much, leading to instability during training.

  A very deep neural network can have an excessive number of parameters, which can lead to **overfitting** if the data set used for training is not large enough. Overfitting occurs when the model becomes too complex and starts to fit the noise in the data, rather than the underlying patterns.

  Training a very deep neural network can require a significant amount of **computational resources,** including memory and processing power. This can be a practical limitation for many applications.

  Increasing the depth of a neural network can lead to **diminishing returns**. At some point, the additional layers may not add much value to the model's performance, and the training time may not be worth the marginal improvements.

- **QUESTION 7: Random Forest**

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

  - **Question 7.1**

    Random forests have the following hyper-parameters:
    - **Maximum number of features;**
    - **Number of trees;**
    - **Depth of each tree;**

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

We use the GridSearch,Fitting 10 folds for each of 48 candidates, totaling 480 fits.

```
pipeline_forest = Pipeline([
    ('model', RandomForestRegressor())
],memory=memory)

param_grid_forest = {
    'model__max_features': np.arange(1, 5, 1),
    'model__n_estimators': np.arange(10, 40, 10),
    'model__max_depth': np.arange(1, 5, 1)
}
```

Random Forest Regressor for diamond

Best parameters:

Max **depth** of each tree: 4

Max number of **features**: 4

Number of **estimators**: 10

Best score = -0.32233482446629436

Random Forest Regressor for Gas Turbine

Best parameters:

Max **depth** of each tree: 4

Max number of **features**: 4

Number of **estimators**: 30

Best score = -0.7032853751208423

Maximum number of features: This hyperparameter controls the maximum number of features to consider when splitting each tree node. By limiting the number of features, the model can reduce the risk of overfitting and improve its generalization performance. A smaller value for the maximum number of features can lead to a simpler and more interpretable model, while a larger value can lead to a more complex model that may overfit the training data.

Number of trees: This hyperparameter controls the number of decision trees in the

ensemble. By increasing the number of trees, the model can improve its performance and reduce the risk of overfitting. However, increasing the number of trees also increases the computational cost and can lead to longer training times.

Depth of each tree: This hyperparameter controls the maximum depth of each decision tree in the ensemble. By limiting the depth of the trees, the model can prevent overfitting and improve its generalization performance. However, a shallow tree may not be able to capture complex patterns in the data, while a deeper tree can overfit the training data.

- **Question 7.2: How do random forests create a highly non-linear decision boundary despite the fact that all we do at each layer is apply a threshold on a feature?**

  Each decision tree in a random forest can **select different features** to split the data. By randomly selecting subsets of features for each tree, the forest can capture a wide range of nonlinearities and interactions between features.

  By **combining the output** of multiple decision trees, the random forest can create complex decision boundaries that are not achievable by a single tree. This is because each tree may make a different set of splits based on the selected features, and the ensemble of trees combines these splits to create a more robust decision boundary.

  The **hierarchical structure of decision trees** allows the random forest to create a nested set of decision boundaries, with each tree refining the splits made by the previous trees. This enables the model to capture complex decision boundaries in a step-wise manner.

- **Question 7.3: Randomly pick a tree in your random forest model (with maximum depth of 4) and plot its structure. Which feature is selected for branching at the root node? What can you infer about the importance of this feature as opposed to others? Do the important features correspond to what you got in part 3.3.1?**

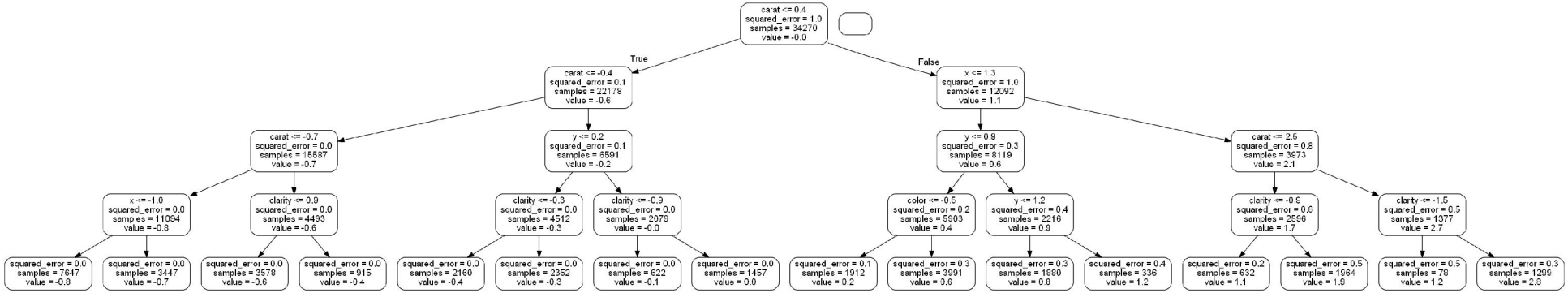
  Column_s: ['carat', 'color', 'clarity', 'x', 'y', 'z']
  Column_g: ['TIT, 'AP', 'AH', 'AFDP', 'GTEP', 'AT', 'TAT', 'TEY', 'CDP']

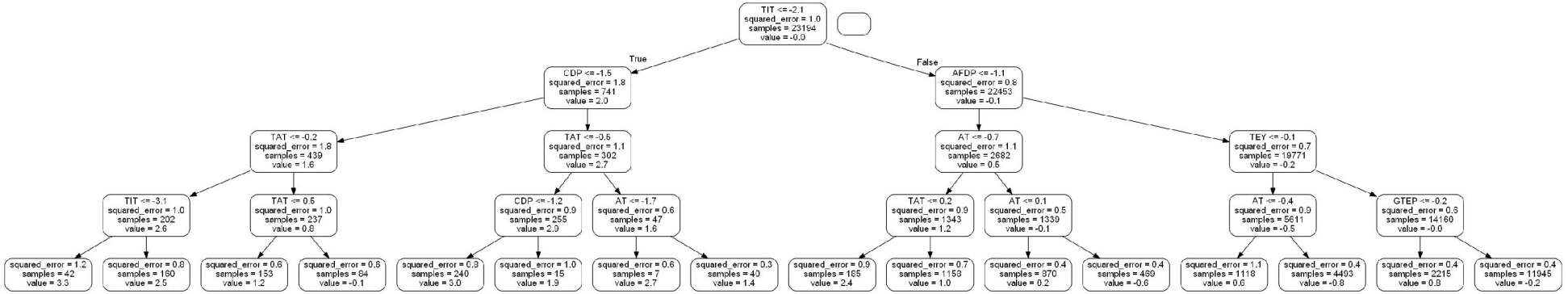
  For **Diamond** dataset:

  '**carat**' is selected for branching at the root node.
  We can infer that 'carat' is the most important feature in this tree.
  It **corresponds to** what we got in part 3.3.1.

  For **Gas Turbine** dataset:

  'TIT' is selected for branching at the root node.
  We can infer that 'TIT' is the most important feature in this tree.
  It **corresponds to** what we got in part 3.3.1.


carat <= 0.4
squared_error = 1.0
samples = 34270
value = -0.0
True
False
carat <= -0.4
squared_error = 0.1
samples = 22178
value = -0.6
x <= 1.3
squared_error = 1.0
samples = 12092
value = 1.1
carat <= -0.7
squared_error = 0.0
samples = 15597
value = -0.7
y <= 0.2
squared_error = 0.1
samples = 6591
value = -0.2
y <= 0.9
squared_error = 0.3
samples = 8119
value = 0.6
carat <= 2.5
squared_error = 0.8
samples = 3973
value = 2.1
x <= -1.0
squared_error = 0.0
samples = 11094
value = -0.8
clarity <= 0.9
squared_error = 0.0
samples = 4493
value = -0.6
clarity <= -0.3
squared_error = 0.0
samples = 4512
value = -0.3
clarity <= -0.9
squared_error = 0.0
samples = 2079
value = -0.0
color <= -0.6
squared_error = 0.2
samples = 5903
value = 0.4
y <= 1.2
squared_error = 0.4
samples = 2216
value = 0.9
clarity <= -0.9
squared_error = 0.6
samples = 2596
value = 1.7
clarity <= -1.5
squared_error = 0.5
samples = 1377
value = 2.7
squared_error = 0.0
samples = 7647
value = -0.8
squared_error = 0.0
samples = 3447
value = -0.7
squared_error = 0.0
samples = 3578
value = -0.6
squared_error = 0.0
samples = 915
value = -0.4
squared_error = 0.0
samples = 2160
value = -0.4
squared_error = 0.0
samples = 2352
value = -0.3
squared_error = 0.0
samples = 622
value = -0.1
squared_error = 0.0
samples = 1457
value = 0.0
squared_error = 0.1
samples = 1912
value = 0.2
squared_error = 0.3
samples = 3991
value = 0.6
squared_error = 0.3
samples = 1880
value = 0.8
squared_error = 0.4
samples = 336
value = 1.2
squared_error = 0.2
samples = 632
value = 1.1
squared_error = 0.5
samples = 1964
value = 1.9
squared_error = 0.5
samples = 78
value = 1.2
squared_error = 0.3
samples = 1299
value = 2.3


TIT <= -2.1
squared_error = 1.0
samples = 23194
value = -0.0
True
False
CDP <= -1.5
squared_error = 1.8
samples = 741
value = 2.0
AFDP <= -1.1
squared_error = 0.8
samples = 22453
value = -0.1
TAT <= -0.2
squared_error = 1.8
samples = 439
value = 1.6
TAT <= -0.5
squared_error = 1.1
samples = 302
value = 2.7
AT <= -0.7
squared_error = 1.1
samples = 2682
value = 0.5
TEY <= -0.1
squared_error = 0.7
samples = 19771
value = -0.2
TIT <= -3.1
squared_error = 1.0
samples = 202
value = 2.6
TAT <= 0.5
squared_error = 1.0
samples = 237
value = 0.8
CDP <= -1.2
squared_error = 0.9
samples = 255
value = 2.9
AT <= -1.7
squared_error = 0.6
samples = 47
value = 1.6
TAT <= 0.2
squared_error = 0.9
samples = 1343
value = 1.2
AT <= 0.1
squared_error = 0.5
samples = 1339
value = -0.1
AT <= -0.4
squared_error = 0.9
samples = 5611
value = -0.5
GTEP <= -0.2
squared_error = 0.6
samples = 14160
value = -0.0
squared_error = 1.2
samples = 42
value = 3.3
squared_error = 0.8
samples = 160
value = 2.5
squared_error = 0.6
samples = 153
value = 1.2
squared_error = 0.8
samples = 84
value = -0.1
squared_error = 0.3
samples = 240
value = 3.0
squared_error = 1.0
samples = 15
value = 1.9
squared_error = 0.6
samples = 7
value = 2.7
squared_error = 0.3
samples = 40
value = 1.4
squared_error = 0.9
samples = 165
value = 2.4
squared_error = 0.7
samples = 1158
value = 1.0
squared_error = 0.4
samples = 870
value = 0.2
squared_error = 0.4
samples = 469
value = -0.6
squared_error = 1.1
samples = 1118
value = 0.6
squared_error = 0.4
samples = 4493
value = -0.8
squared_error = 0.4
samples = 2215
value = 0.8
squared_error = 0.4
samples = 11945
value = -0.2

- **Question 7.4: Measure "Out-of-Bag Error" (OOB). Explain what OOB error and R2 score means.**

  Best Random Forest Model for **Diamond** Dataset:

  OOB score: 0.9113

  R^2 score: 0.9283

  Best Random Forest Model for **Gas Turbine** Dataset:

  OOB score: 0.5951

  R^2 score: 0.6063

  OOB error: The OOB error is the error rate of the model on these OOB samples. The OOB error is a useful metric because it provides an estimate of how well the model will perform on new, unseen data. It is calculated by aggregating the predictions from all the trees in the forest, with each tree predicting on the OOB samples it did not use during training. The OOB error is then computed as the average difference between the predicted and actual values for these samples.

  R2 score: The R2 score, also known as the coefficient of determination, is a measure of how well the model fits the data. It represents the proportion of the variance in the target variable that is explained by the model. The R2 score ranges from 0 to 1, with a higher score indicating a better fit to the data. An R2 score of 1 indicates that the model perfectly fits the data, while a score of 0 indicates that the model does not explain any of the variance in the target variable.

- **QUESTION 8: LightGBM, CatBoost and Bayesian Optimization**

  Boosted tree methods have shown advantages when dealing with tabular data, and recent advances make these algorithms scalable to large scale data and enable natural treatment of (high-cardinality) categorical features. Two of the most successful examples are Light-GBM and CatBoost.

  Both algorithms have many hyperparameters that influence their performance. This results in large search space of hyperparameters, making the tuning of the hyperparameters hard with naive random search and grid search. Therefore, one may want to utilize "smarter" hyperparameter search schemes. We specifically explore one of them: Bayesian optimization.

  In this part, pick either one of the datasets and apply LightGBM OR CatBoost. If you do both, we will only look at the first one.

  Question 8.1: Read the documentation of LightGBM OR CatBoost and determine the important hyperparameters along with a search space for the tuning of these parameters (keep the search space small).

  For LightGBM:
  learning_rate: This hyperparameter controls the step size taken during gradient descent. Suggested search space:
  num_leaves: This hyperparameter controls the maximum number of leaves in one tree.
  max_depth: This hyperparameter controls the maximum depth of a tree.
  n_estimators : Number of boosted trees to fit.

```
lbg_pip_ = Pipeline([
    ('model', lgb.LGBMRegressor())
],memory = memory)

param_lgb_ = {
    'model__num_leaves': [7, 14, 21, 28, 31, 50],
    'model__learning_rate': [0.1, 0.03, 0.003],
    'model__max_depth': [-1, 3, 5],
    'model__n_estimators': [50, 100, 200, 500],
}
```

Question 8.2: Apply Bayesian optimization using skopt.BayesSearchCV from scikit-optimize to find the ideal hyperparameter combination in your search space. Report the best hyperparameter set found and the corresponding RMSE.

Fitting 10 folds for each of 1 candidate, totaling 10 fits.

LightGBM of Diamonds

Best parameters:

Learning rate: 0.1

Max depth: 8

Number of estimators: 500

Number of leaves: 23

LightGBM of Gas Turbine

Best parameters:

Learning rate: 0.1

Max depth: 5

Number of estimators: 200

Number of leaves: 31

LightGBM of Diamonds

Best score = -0.15290674289254738

RMSE = 0.3910329179142689

LightGBM of Gas Turbine

Best score = -0.5926654920848259

RMSE = 0.769847707072526

- **Question 8.3: Qualitatively interpret the effect of the hyperparameters using the Bayesian optimization results: Which of them helps with performance? Which helps with regularization (shrinks the generalization gap)? Which affects the fitting efficiency?**

  Bayesian optimization results indicate that **increasing the number of trees** or **the maximum number of features** can improve the model's **performance**. This is because more trees or more features can capture more complex patterns in the data and improve the model's ability to generalize to new data.

  Reducing the maximum depth of each tree or the **number of trees** can help with **regularization**. This is because a smaller tree or fewer trees can reduce the model's complexity and prevent overfitting, leading to better generalization performance.

  The learning rate or **the batch size** can affect **the fitting efficiency** of the model. This is because these hyperparameters can control how quickly the model learns from the data and how many samples are used in each iteration, affecting the speed and stability of the training process.

Project 4: Regression Analysis

Wenxin Cheng 706070535 wenxin0319@g.ucla.edu

Yuxin Yin 606073780 yyxyy999@g.ucla.edu

Yingqian Zhao 306071513 zhaoyq99@g.ucla.edu

0 pre-install packages

In [7]:
```
# !pip install statsmodels
# !pip install scipy
# conda install -c anaconda graphviz
```

In [8]:
```
import pandas as pd
from sklearn import preprocessing
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.feature_selection import SelectKBest, mutual_info_regression, f_regression
from sklearn.linear_model import LinearRegression, Ridge, Lasso
from sklearn.pipeline import Pipeline
from sklearn.model_selection import cross_validate,GridSearchCV
from sklearn.preprocessing import StandardScaler, PolynomialFeatures
import statsmodels.api as sm
import scipy.stats as ss
from tempfile import mkdtemp
from joblib import Memory
import warnings
warnings.filterwarnings('ignore')
from sklearn.neural_network import MLPRegressor
from sklearn.ensemble import RandomForestRegressor
from sklearn.tree import export_graphviz
import pydot
from IPython.display import Image
```

In [9]:
```
## reading Dataset1 Diamonds.csv
diamond_data = pd.read_csv('diamonds.csv', index_col=0)
print(diamond_data.head())
```

```
   carat      cut color clarity  depth  table  price     x     y     z
1   0.23    Ideal     E     SI2   61.5   55.0    330  3.95  3.98  2.43
2   0.21  Premium     E     SI1   59.8   61.0    327  3.89  3.84  2.31
3   0.23     Good     E     VS1   56.9   65.0    328  4.05  4.07  2.31
4   0.29  Premium     I     VS2   62.4   58.0    337  4.20  4.23  2.63
5   0.31     Good     J     SI2   63.3   58.0    338  4.34  4.35  2.75
```

In [17]:
```
## reading Dataset2 pp_gas_emission
gt2011 = pd.read_csv("pp_gas_emission/gt_2011.csv")
gt2012 = pd.read_csv("pp_gas_emission/gt_2012.csv")
gt2013 = pd.read_csv("pp_gas_emission/gt_2013.csv")
gt2014 = pd.read_csv("pp_gas_emission/gt_2014.csv")
gt2015 = pd.read_csv("pp_gas_emission/gt_2015.csv")
gt2011["year"] = [2011] * len(gt2011)
gt2012["year"] = [2012] * len(gt2012)
gt2013["year"] = [2013] * len(gt2013)
gt2014["year"] = [2014] * len(gt2014)
gt2015["year"] = [2015] * len(gt2015)
```

```
gt_data_ = pd.concat([gt2011, gt2012, gt2013, gt2014, gt2015])
#picking NOX droping CO
# gt_data_.drop('CO',axis = 1)
gt_data = gt_data_.copy().drop('CO',axis = 1)
```

Question 1

In [11]:
```
diamond_data.head()
```

Out[11]:

	carat	cut	color	clarity	depth	table	price	x	y	z
1	0.23	Ideal	E	SI2	61.5	55.0	330	3.95	3.98	2.43
2	0.21	Premium	E	SI1	59.8	61.0	327	3.89	3.84	2.31
3	0.23	Good	E	VS1	56.9	65.0	328	4.05	4.07	2.31
4	0.29	Premium	I	VS2	62.4	58.0	337	4.20	4.23	2.63
5	0.31	Good	J	SI2	63.3	58.0	338	4.34	4.35	2.75

In [18]:
```
gt_data.head()
```

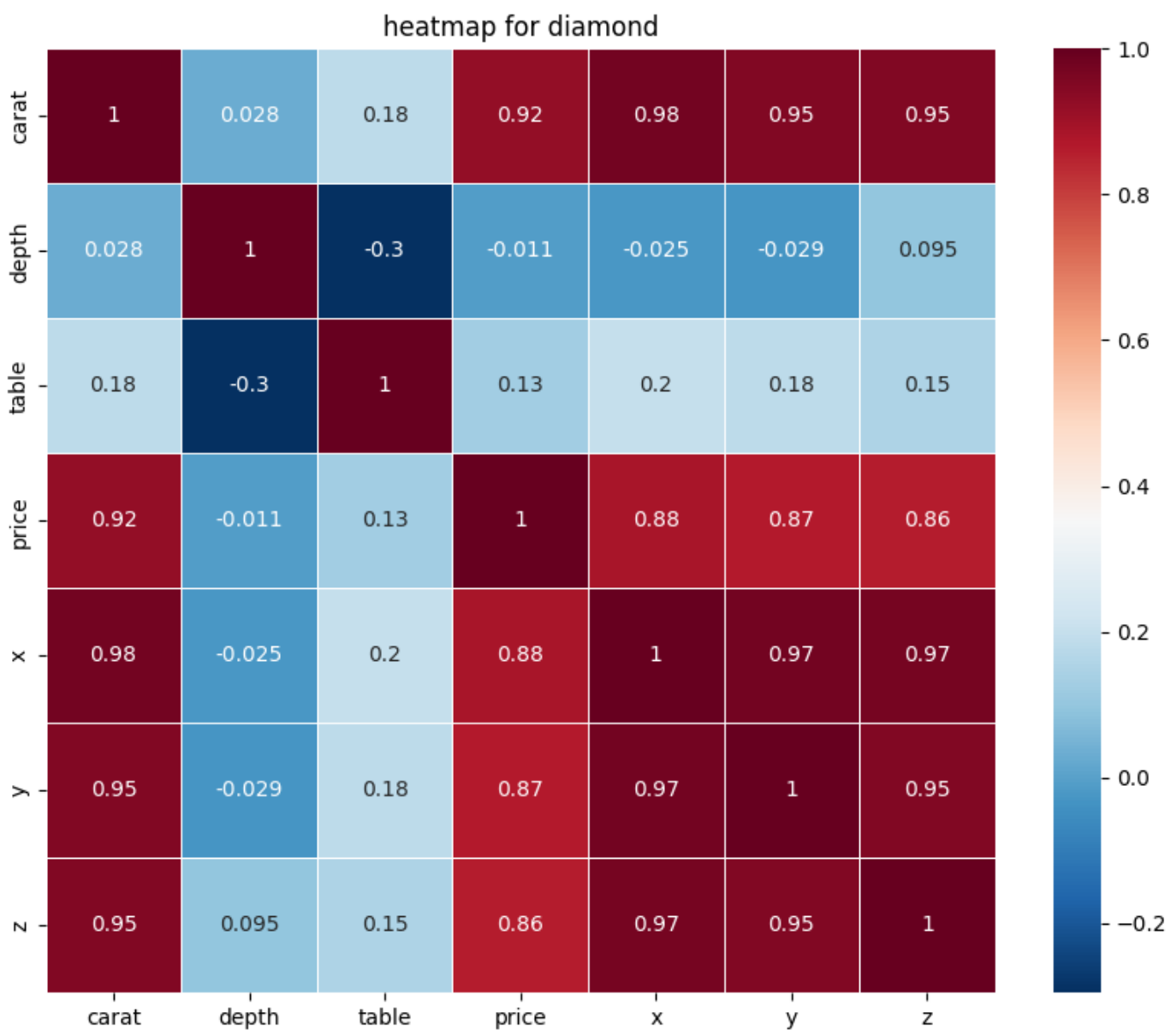
Out[18]:

	AT	AP	AH	AFDP	GTEP	TIT	TAT	TEY	CDP	NOX	year
0	4.5878	1018.7	83.675	3.5758	23.979	1086.2	549.83	134.67	11.898	81.952	2011
1	4.2932	1018.3	84.235	3.5709	23.951	1086.1	550.05	134.67	11.892	82.377	2011
2	3.9045	1018.4	84.858	3.5828	23.990	1086.5	550.19	135.10	12.042	83.776	2011
3	3.7436	1018.3	85.434	3.5808	23.911	1086.5	550.17	135.03	11.990	82.505	2011
4	3.7516	1017.8	85.182	3.5781	23.917	1085.9	550.00	134.67	11.910	82.028	2011

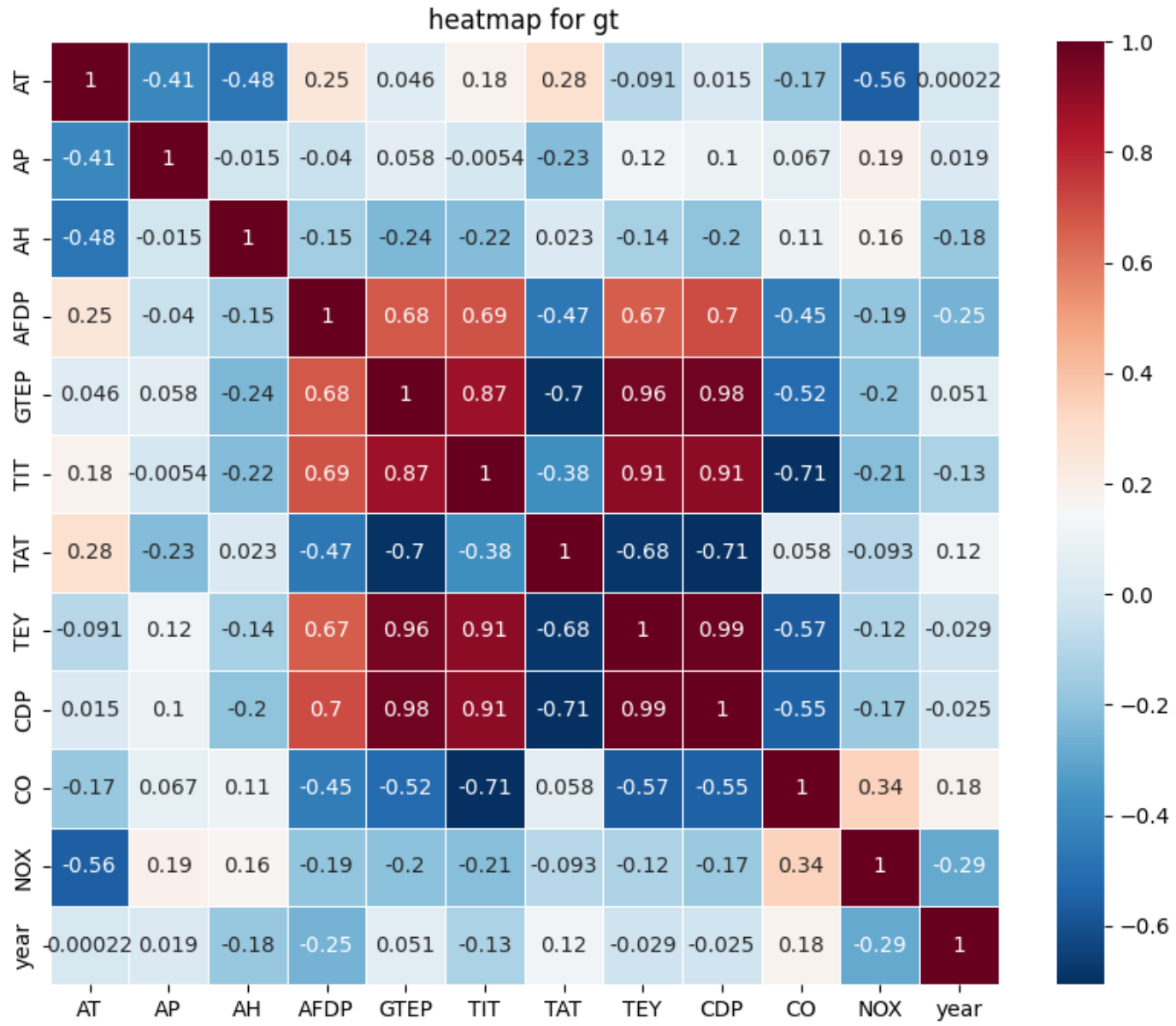
Question1.1

In [12]:
```
diamond_corr = diamond_data.corr()
gt_corr = gt_data.corr()

def plot_heatmap(dataset, title):
    plt.figure(figsize=(10,8))
    sns.heatmap(dataset,
                xticklabels=dataset.columns,
                yticklabels=dataset.columns,
                cmap='RdBu_r',
                annot=True,
                linewidth=0.5)
    plt.title(f"heatmap for {title}")
```

In [13]:
```
plot_heatmap(diamond_corr, "diamond")
```



In [14]:
```
plot_heatmap(gt_corr, "gt")
```

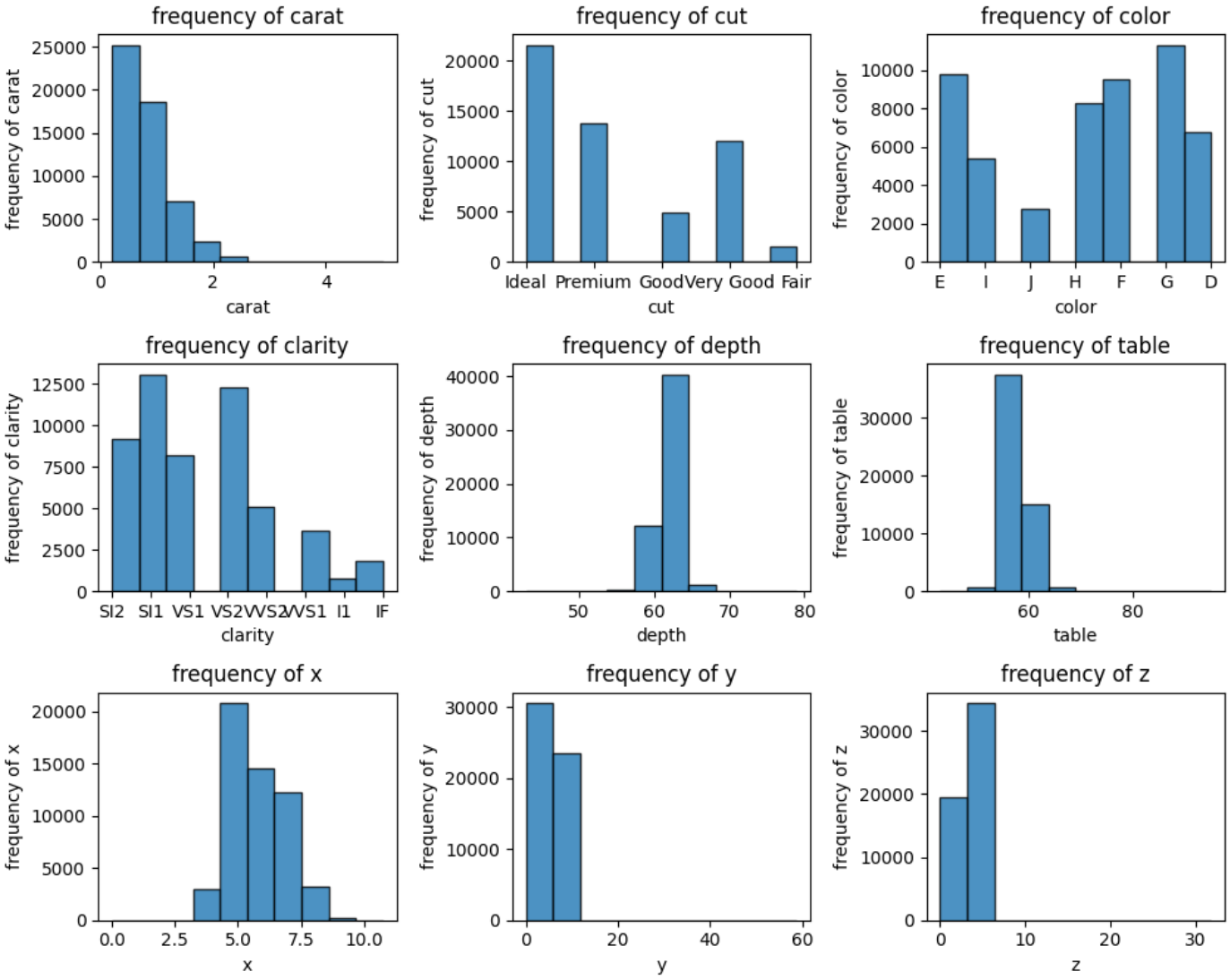


Question1.2

In [19]:
```
def plot_histogram(dataset, features):
    fig, axs = plt.subplots(3, 3, figsize=(10, 8))
    for i in range(3):
        for j in range(3):
            index = i * 3 + j
            feature = features[index]
            axs[i, j].hist(dataset[feature], edgecolor='k', linewidth=1, alpha=0.8)
            axs[i, j].set_title(f"frequency of {str(feature)}")
            axs[i, j].set(xlabel=str(feature), ylabel=f"frequency of {str(feature)}")
    fig.tight_layout()
```

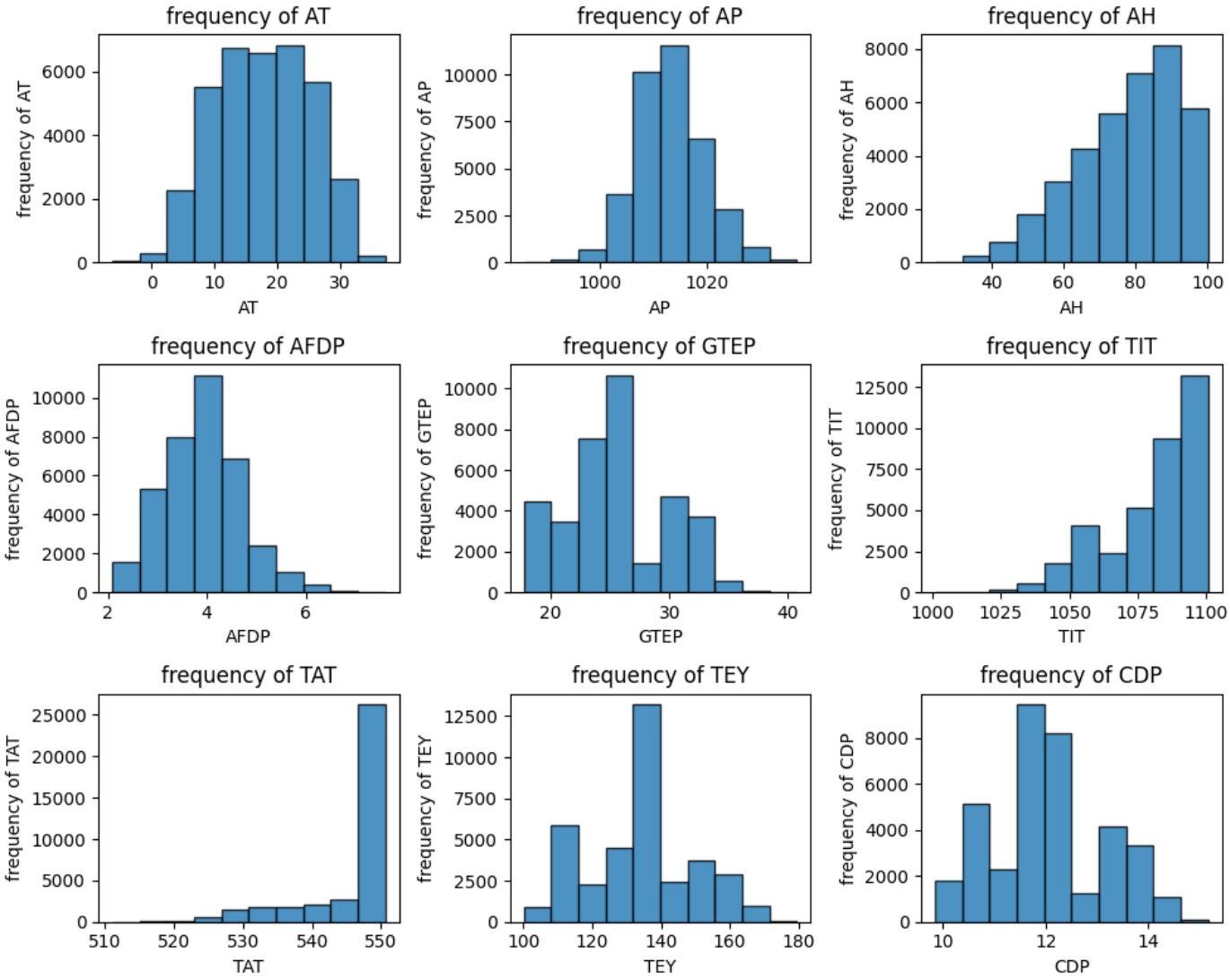
In [20]:
```
diamonds_features = ["carat","cut","color","clarity","depth","table","x","y","z"]
print(len(diamonds_features))
plot_histogram(diamond_data,diamonds_features)
```

```
9
```



In [21]:
```
gt_features = ["AT","AP","AH","AFDP","GTEP","TIT","TAT","TEY","CDP"]
print(len(gt_features))
plot_histogram(gt_data,gt_features)
```

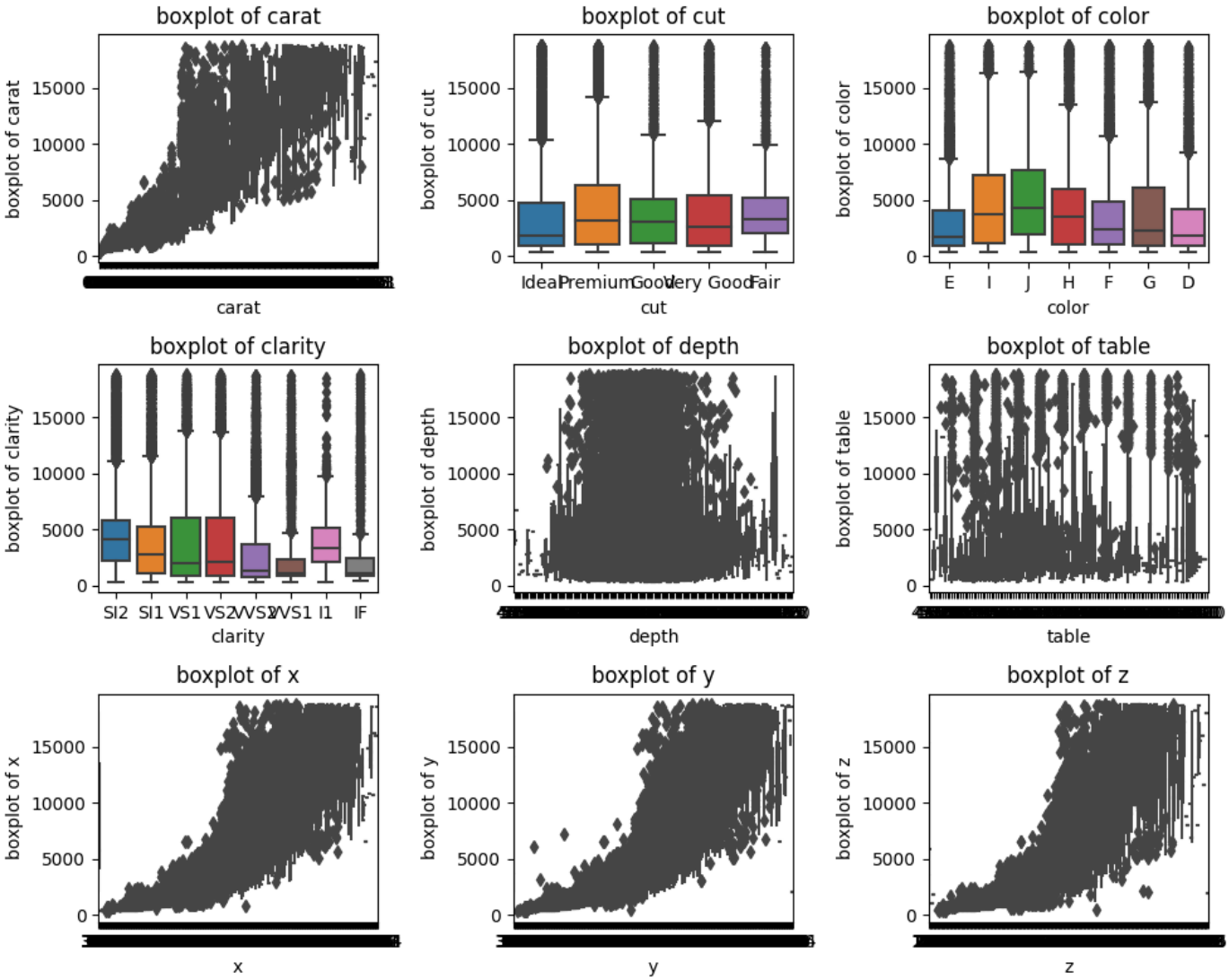
```
9
```



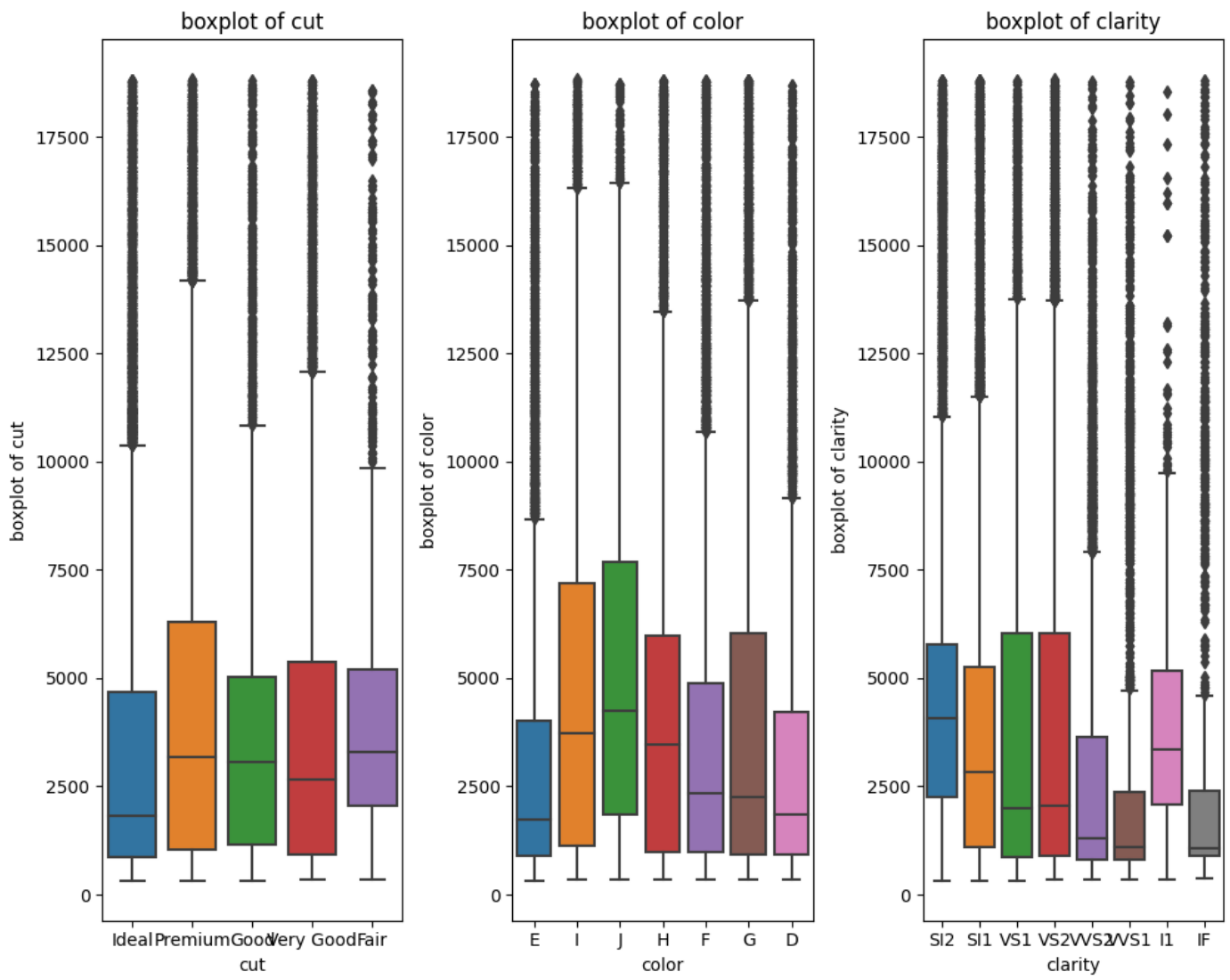
Question1.3

In [22]:
```
def plot_boxplot(dataset, features, target, row, col):
    fig, axes = plt.subplots(row, col, figsize=(10, 8))

    for name, ax in zip(features, axes.flatten()):
        sns.boxplot(y=dataset[target], x= dataset[name], orient='v', ax=ax)
        ax.set_title(f"boxplot of {str(name)}")
        ax.set(xlabel=str(name), ylabel=f"boxplot of {str(name)}")
    fig.tight_layout()
```

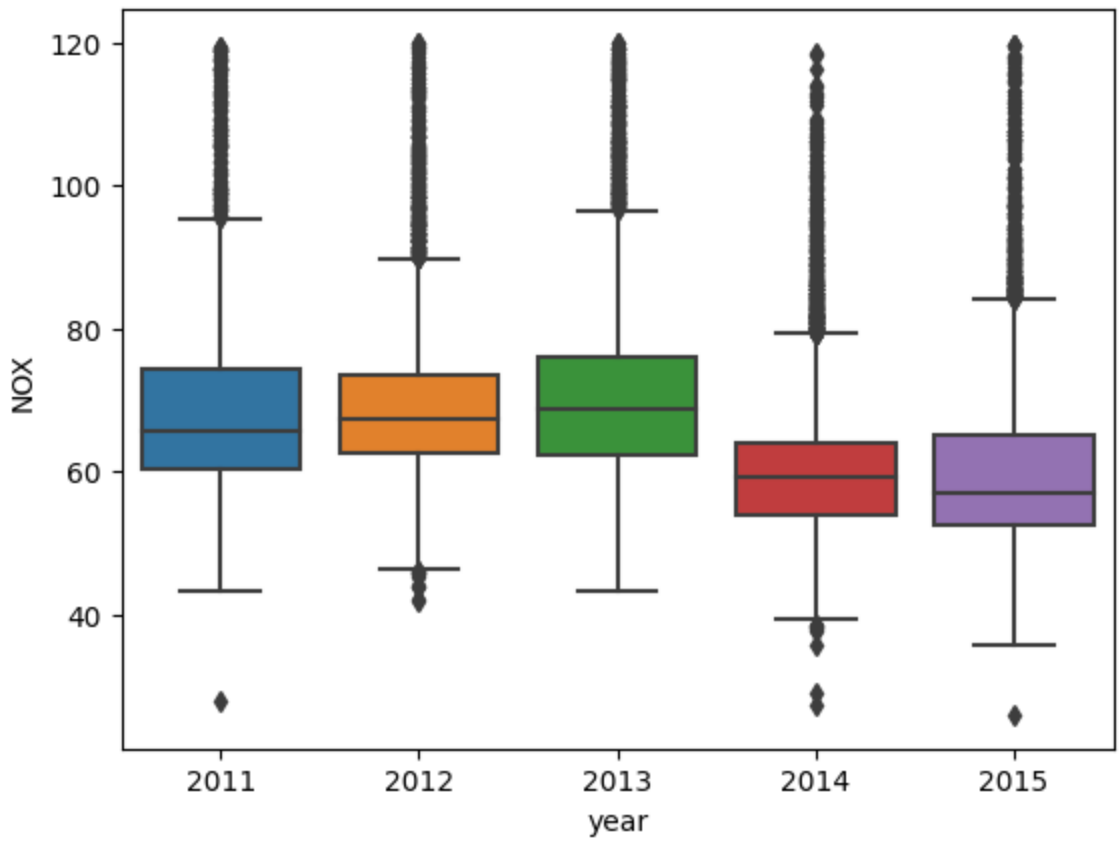
In [23]:
```
plot_boxplot(diamond_data,diamonds_features,"price",3, 3)
```



In [24]:
```
diamonds_labels = ["cut","color","clarity"]
plot_boxplot(diamond_data,diamonds_labels,"price",1, 3)
```

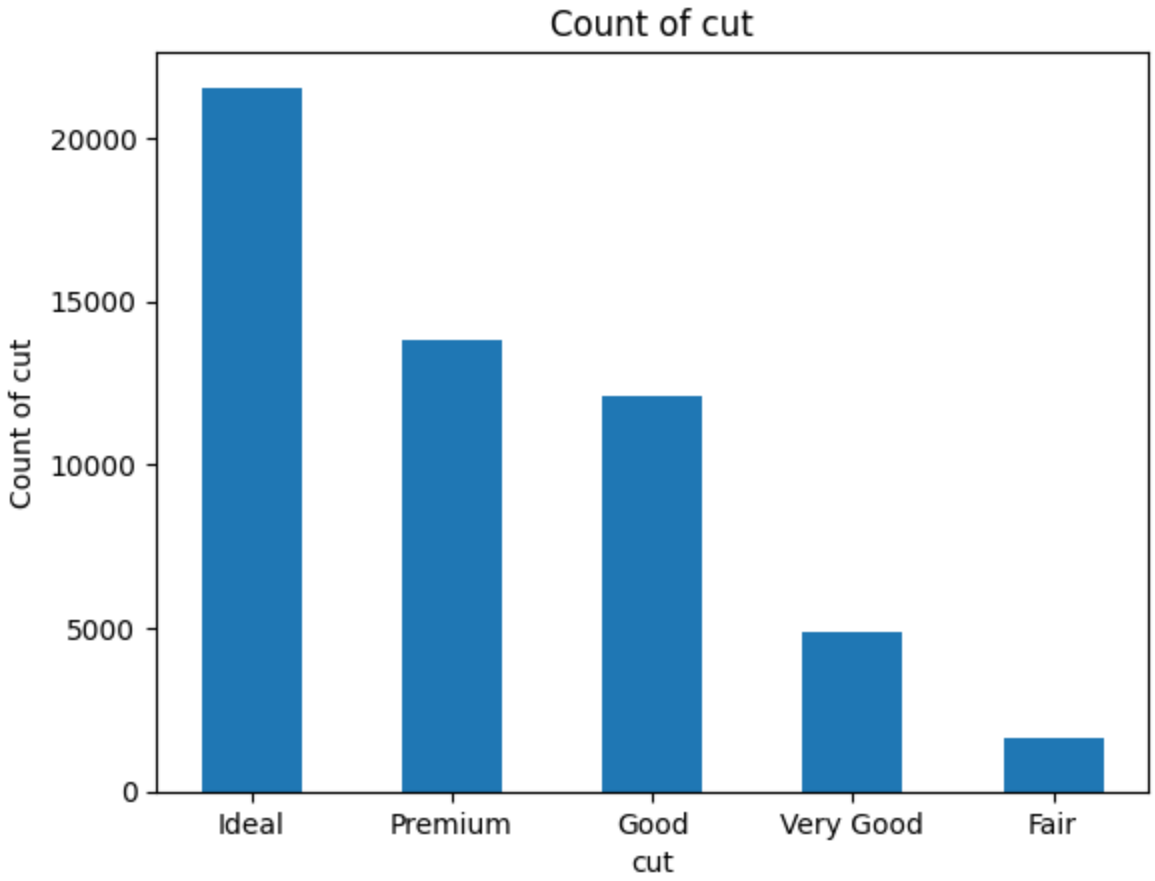


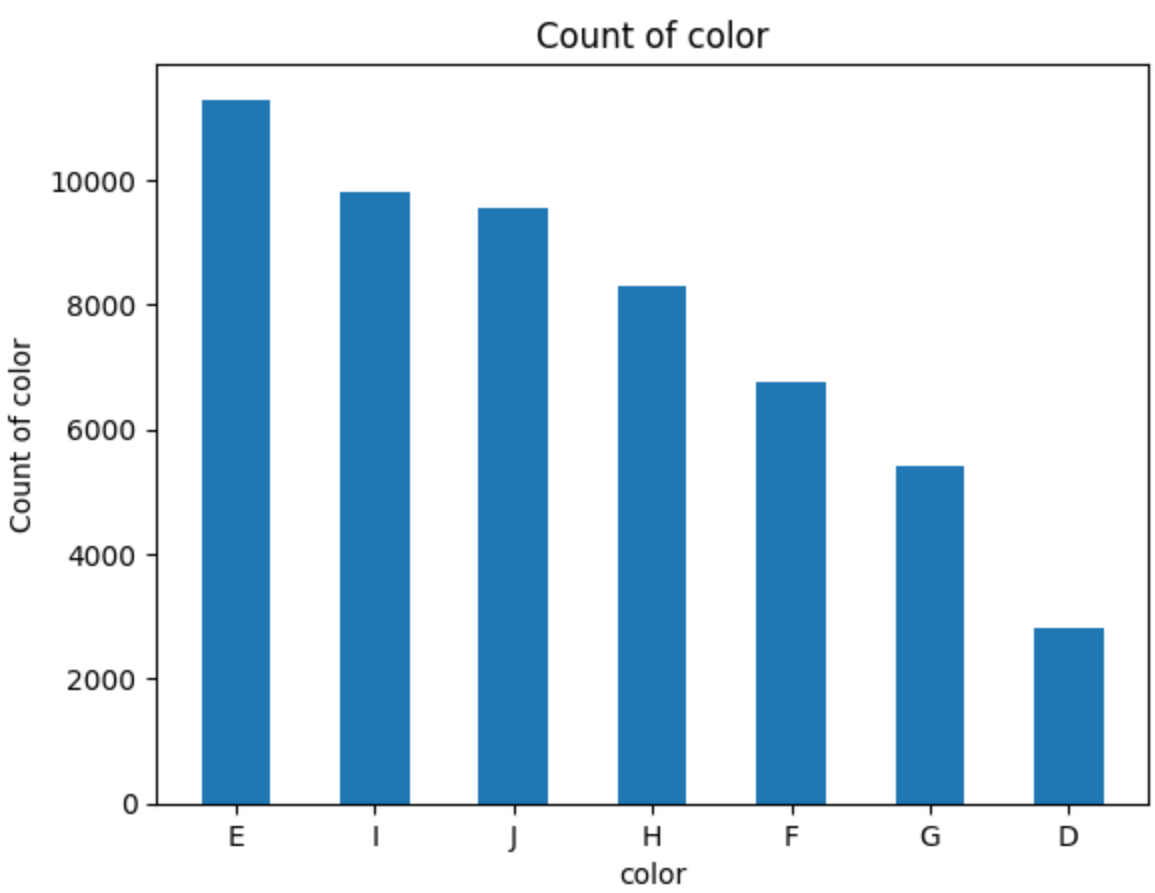
In [25]:
```
sns.boxplot(x = gt_data["year"],y = gt_data["NOX"], order=list(set(gt_data["year"])))
plt.show()
```

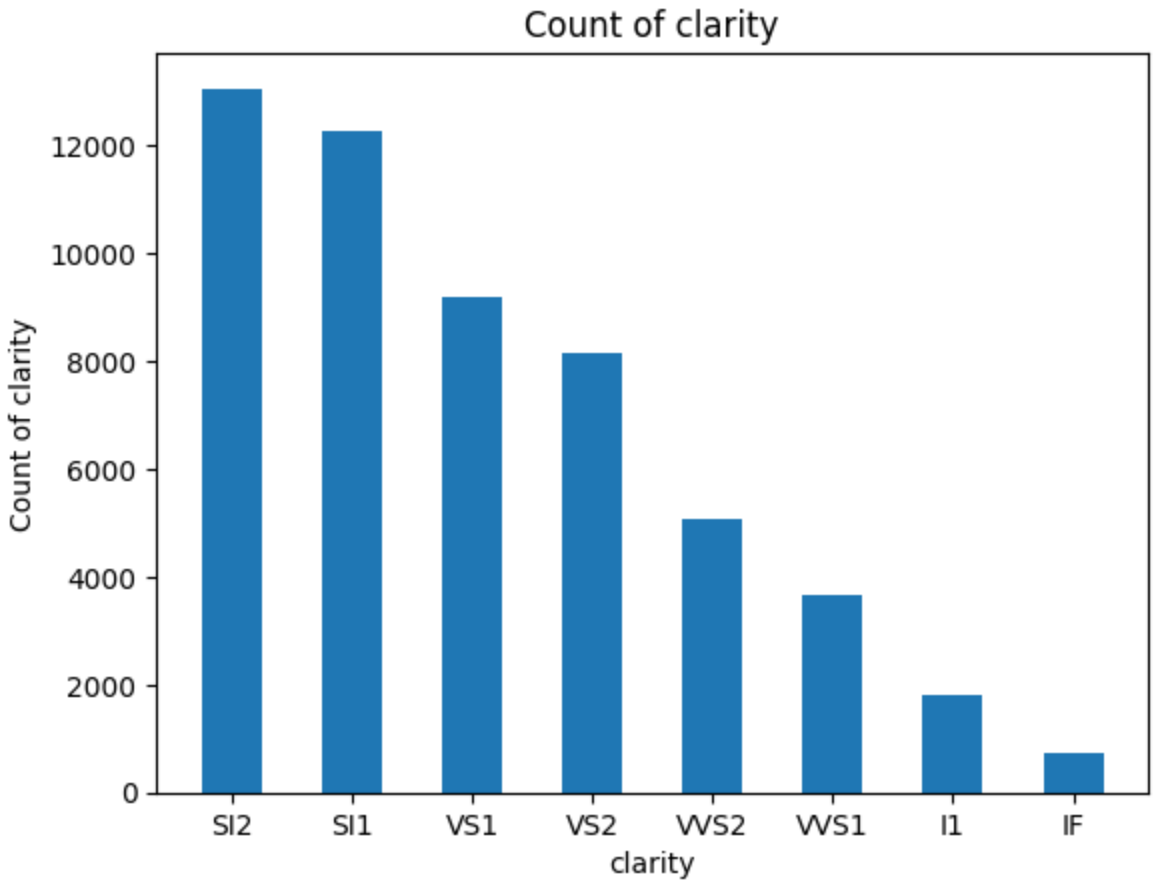


In [26]:
```
def plot_count(dataset,features):
    for feature in features:
        x_axis = list(dataset[feature].unique())
        y_axis = dataset[feature].value_counts().to_list()
        plt.bar(x_axis, y_axis, width=0.5)
        plt.ylabel(f"Count of {str(feature)}")
        plt.xlabel(f"{str(feature)}")
        plt.title(f"Count of {str(feature)}")
        plt.show()
```

In [27]:
```
plot_count(diamond_data,diamonds_labels)
```





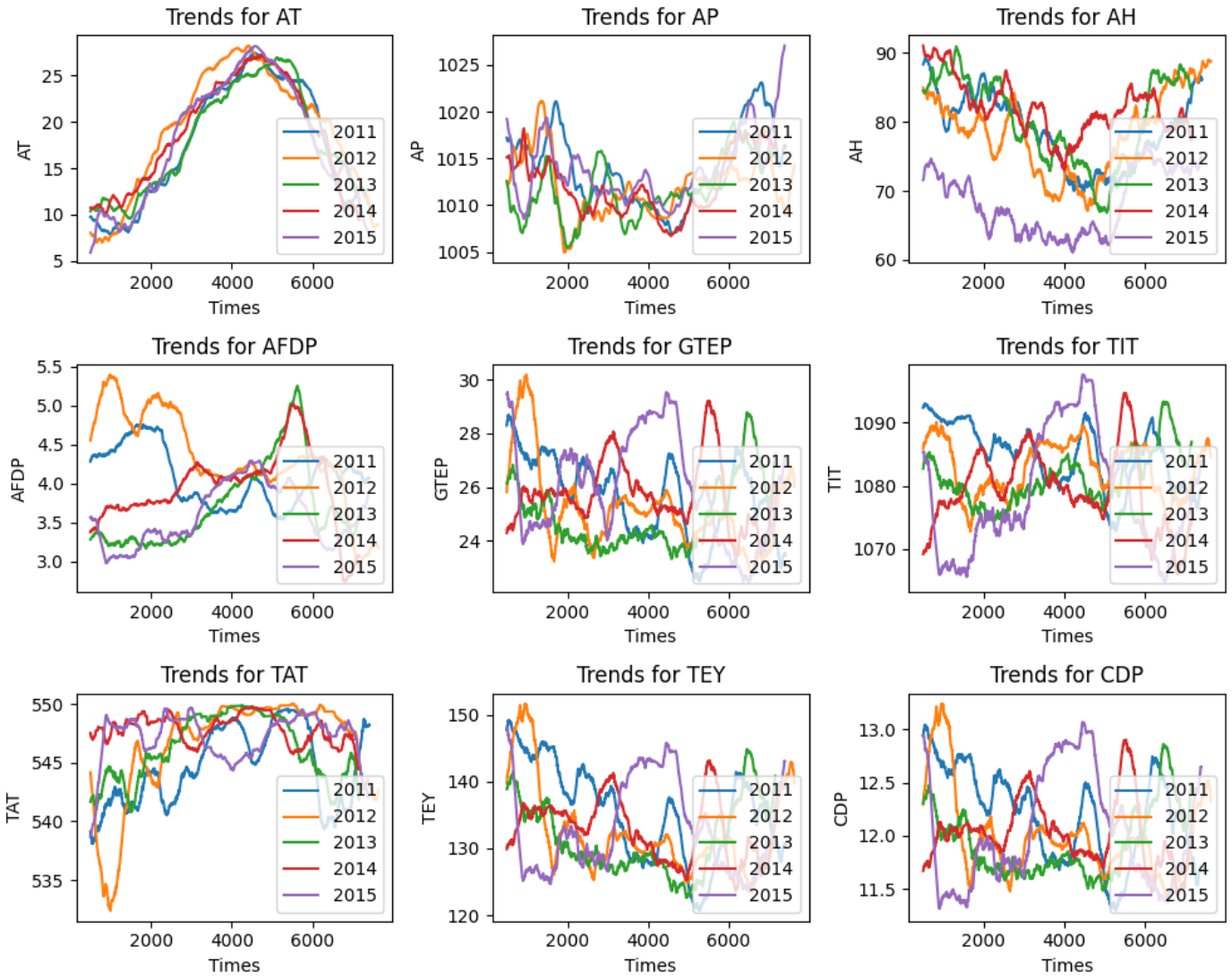


Question1.4

In [28]:
```
def plot_trend(dataset, features):
    year_list = [2011,2012,2013,2014,2015]
    fig, axs = plt.subplots(3, 3, figsize=(10, 8))
    for i in range(3):
```

```
        for j in range(3):
            index = i * 3 + j
            feature = features[index]
            for year in year_list:
                data_ = gt_data.loc[gt_data['year'] == year,feature].rolling(500).mean()
                axs[i, j].plot(data_,label = str(year))
                axs[i, j].set_title("Trends for "+ str(feature))
                axs[i, j].set(xlabel="Times", ylabel=str(feature))
                axs[i, j].legend(loc='lower right')
    fig.tight_layout()
```

In [29]:
```
plot_trend(gt_data, gt_features)
```



Question 2

Question2.1

In [2]:
```
# Standardization
# for diamonds encoding
cut_encoding = {'Fair' : 1, 'Good' : 2, 'Ideal' : 3, 'Premium' : 4, 'Very Good' : 5}
color_encoding = {'J' : 1, 'I' : 2, 'H' : 3, 'G' : 4, 'F' : 5, 'E' : 6, 'D' : 7}
clarity_encoding = {'I1' : 1, 'SI2' : 2, 'SI1' : 3, 'VS2' : 4, 'VS1' : 5, 'VVS2' : 6, 'V

def encode_feature(df, feature, encoding_dict):
  features = df[feature]
  encoded_feats = [encoding_dict[f] for f in features]
  df[feature] = encoded_feats
```

```
encode_feature(diamond_data, 'cut', cut_encoding)
encode_feature(diamond_data, 'color', color_encoding)
encode_feature(diamond_data, 'clarity', clarity_encoding)
```

In [4]:
```
# standard
diamond_standard = pd.DataFrame(preprocessing.scale(diamond_data),columns = diamond_data

gt_scale = gt_data.copy().drop('year',axis = 1)
#picking NOX droping CO
gt_standard = pd.DataFrame(preprocessing.scale(gt_scale),columns = gt_scale.columns)
gt_standard['year'] = np.asanyarray(gt_data['year'])
```

In [5]:
```
diamond_standard.head()
```

Out[5]:

	carat	cut	color	clarity	depth	table	price	x	y	
0	-1.198168	-0.538099	0.937163	-1.245215	-0.174092	-1.099672	-0.903594	-1.587837	-1.536196	-1.
1	-1.240361	0.434949	0.937163	-0.638095	-1.360738	1.585529	-0.904346	-1.641325	-1.658774	-1.
2	-1.198168	-1.511147	0.937163	0.576145	-3.385019	3.375663	-0.904095	-1.498691	-1.457395	-1.
3	-1.071587	0.434949	-1.414272	-0.030975	0.454133	0.242928	-0.901839	-1.364971	-1.317305	-1.2
4	-1.029394	-1.511147	-2.002131	-1.245215	1.082358	0.242928	-0.901588	-1.240167	-1.212238	-1.

In [6]:
```
gt_standard.head()
```

Out[6]:

	AT	AP	AH	AFDP	GTEP	TIT	TAT	TEY	CDP	N
0	-1.762362	0.871052	0.401627	-0.451875	-0.377702	0.272119	0.536589	0.074502	-0.149273	1.4264
1	-1.801920	0.809164	0.440351	-0.458207	-0.384376	0.266417	0.568742	0.074502	-0.154783	1.4628
2	-1.854113	0.824636	0.483432	-0.442831	-0.375081	0.289227	0.589203	0.102033	-0.017015	1.5826
3	-1.875718	0.809164	0.523263	-0.445415	-0.393909	0.289227	0.586280	0.097551	-0.064774	1.4738
4	-1.874644	0.731804	0.505837	-0.448904	-0.392479	0.255012	0.561434	0.074502	-0.138251	1.4330

Question2.2

In [34]:
```
def select_topn_important_features(X, Y, n):
  Mutual_ = mutual_info_regression(X, Y)
  F_ = f_regression(X, Y)

  topn_M = np.argsort(Mutual_)[::-1][:n]
  topn_F = np.argsort(F_[0])[::-1][:n]

  all_m = np.argsort(Mutual_)[::-1]
  all_f = np.argsort(F_[0])[::-1]

  X_topn_M = X.iloc[:, topn_M]
  X_topn_F = X.iloc[:, topn_F]

  all_m_ = X.iloc[:, all_m]
  all_f_ = X.iloc[:, all_f]

  return X_topn_M, X_topn_F, all_m_, all_f_
```

In [8]:
```
diamond_standard_x = diamond_standard.copy().drop('price',axis = 1)
diamond_standard_y = diamond_standard["price"]
```

```
gt_standard_x = gt_standard.copy().drop(['NOX','year'],axis = 1)
gt_standard_y = gt_standard["NOX"]
```

In [35]:
```
diamond_top5_M, diamond_top5_F, dall_m, dall_f = select_topn_important_features(diamond_
gt_top5_M, gt_top5_F,gall_m, gall_f = select_topn_important_features(gt_standard_x, gt_s

print("diamond Top5 by mutual_info_regression:")
print(diamond_top5_M.columns)

print("diamond Top5 by f_regression:")
print(diamond_top5_F.columns)

print("gt Top5 by mutual_info_regression:")
print(gt_top5_M.columns)

print("gt Top5 by f_regression:")
print(gt_top5_F.columns)
```

```
diamond Top5 by mutual_info_regression:
Index(['carat', 'y', 'x', 'z', 'clarity'], dtype='object')
diamond Top5 by f_regression:
Index(['carat', 'x', 'y', 'z', 'color'], dtype='object')
gt Top5 by mutual_info_regression:
Index(['TIT', 'TEY', 'AT', 'GTEP', 'CDP'], dtype='object')
gt Top5 by f_regression:
Index(['AT', 'TIT', 'GTEP', 'AP', 'AFDP'], dtype='object')
```

In [1]:
```
print(dall_m.columns)
```

```
Index(['carat', 'y', 'x', 'z', 'clarity', 'color', 'cut', 'table', 'depth'], dtype='obje
ct')
```

In [2]:
```
print(gall_m.columns)
```

```
Index(['TIT', 'TEY', 'AT', 'GTEP', 'CDP', 'AFDP', 'TAT', 'AP', 'AH'], dtype='object')
```

In [21]:
```
diamond_top5_M.head()
```

Out[21]:

	carat	y	x	z	clarity
0	-1.198168	-1.536196	-1.587837	-1.571129	-1.245215
1	-1.240361	-1.658774	-1.641325	-1.741175	-0.638095
2	-1.198168	-1.457395	-1.498691	-1.741175	0.576145
3	-1.071587	-1.317305	-1.364971	-1.287720	-0.030975
4	-1.029394	-1.212238	-1.240167	-1.117674	-1.245215

Question3

In [9]:
```
rf_diamond_ = RandomForestRegressor(n_estimators=10, max_features=4, max_depth=4, oob_sc
rf_diamond_.fit(diamond_standard_x, diamond_standard_y)

print('Best Random Forest Model for Diamond Dataset:')
print('OOB score: %.4f' %(rf_diamond_.oob_score_))
print('R^2 score: %.4f' %(rf_diamond_.score(diamond_standard_x, diamond_standard_y)))
```

```
Best Random Forest Model for Diamond Dataset:
OOB score: 0.9030
R^2 score: 0.9202
```

In [10]:
```
rf_gt_ = RandomForestRegressor(n_estimators=30, max_features=4, max_depth=4, oob_score=T
rf_gt_.fit(gt_standard_x, gt_standard_y)
```

```
print('Best Random Forest Model for Diamond Dataset:')
print('OOB score: %.4f' %(rf_gt_.oob_score_))
print('R^2 score: %.4f' %(rf_gt_.score(gt_standard_x, gt_standard_y)))
```

```
Best Random Forest Model for Diamond Dataset:
OOB score: 0.5951
R^2 score: 0.6063
```

Question4

Question4.1

In [ ]:
```
# now we are do experiments of exactly how many features we need to select for two datas

#with selection
diamonds_rmse_lr_m = []  #diamonds rmse score for linear regression and with mutual_info
diamonds_rmse_lr_f = []
diamonds_rmse_r_m = []
diamonds_rmse_r_f = []
diamonds_rmse_la_m = []
diamonds_rmse_la_f = []

#without selection
diamonds_rmse_lr = [] #diamonds rmse score for linear regression
diamonds_rmse_r = []
diamonds_rmse_la = []

#with selection
gt_rmse_lr_m = []  #gt rmse score for linear regression and with mutual_info_regression
gt_rmse_lr_f = []
gt_rmse_r_m = []
gt_rmse_r_f = []
gt_rmse_la_m = []
gt_rmse_la_f = []

#without selection
gt_rmse_lr = [] #gt rmse score for linear regression
gt_rmse_r = []
gt_rmse_la = []
```

In [40]:
```
# for diamonds part
for k in range(1, len(diamonds_features)):
    diamond_topk_M, diamond_topk_F = select_topn_important_features(diamond_standard_x,
    #diamonds rmse score for linear regression
    score_ = cross_validate(LinearRegression(), diamond_standard_x, diamond_standard_y,
    score__ = score_['test_neg_root_mean_squared_error'].mean()
    print(f"diamonds rmse score for linear regression, {score__:.4f}")
    diamonds_rmse_lr.append(score__)

    #diamonds rmse score for linear regression and with mutual_info_regression
    score_ = cross_validate(LinearRegression(), diamond_topk_M, diamond_standard_y, scor
    score__ = score_['test_neg_root_mean_squared_error'].mean()
    print(f"diamonds rmse score for linear regression and with mutual_info_regression, {
    diamonds_rmse_lr_m.append(score__)

    #diamonds rmse score for linear regression and with f_regression
    score_ = cross_validate(LinearRegression(), diamond_topk_F, diamond_standard_y, scor
    score__ = score_['test_neg_root_mean_squared_error'].mean()
    print(f"diamonds rmse score for linear regression and with f_regression, {score__:.4
    diamonds_rmse_lr_f.append(score__)

    #diamonds rmse score for Ridge regression
    score_ = cross_validate(Ridge(), diamond_standard_x, diamond_standard_y, scoring = [
```

```
    score__ = score_['test_neg_root_mean_squared_error'].mean()
    print(f"diamonds rmse score for Ridge regression, {score__:.4f}")
    diamonds_rmse_r.append(score__)

    #diamonds rmse score for Ridge regression and with mutual_info_regression
    score_ = cross_validate(Ridge(), diamond_topk_M, diamond_standard_y, scoring = ['neg
    score__ = score_['test_neg_root_mean_squared_error'].mean()
    print(f"diamonds rmse score for Ridge regression and with mutual_info_regression, {s
    diamonds_rmse_r_m.append(score__)

    #diamonds rmse score for Ridge regression and with f_regression
    score_ = cross_validate(Ridge(), diamond_topk_F, diamond_standard_y, scoring = ['neg
    score__ = score_['test_neg_root_mean_squared_error'].mean()
    print(f"diamonds rmse score for Ridge regression and with f_regression, {score__:.4f
    diamonds_rmse_r_f.append(score__)

    #diamonds rmse score for Lasso regression
    score_ = cross_validate(Lasso(), diamond_standard_x, diamond_standard_y, scoring = [
    score__ = score_['test_neg_root_mean_squared_error'].mean()
    print(f"diamonds rmse score for Lasso regression, {score__:.4f}")
    diamonds_rmse_la.append(score__)

    #diamonds rmse score for Lasso regression and with mutual_info_regression
    score_ = cross_validate(Lasso(), diamond_topk_M, diamond_standard_y, scoring = ['neg
    score__ = score_['test_neg_root_mean_squared_error'].mean()
    print(f"diamonds rmse score for Lasso regression and with mutual_info_regression, {s
    diamonds_rmse_la_m.append(score__)

    #diamonds rmse score for Lasso regression and with f_regression
    score_ = cross_validate(Lasso(), diamond_topk_F, diamond_standard_y, scoring = ['neg
    score__ = score_['test_neg_root_mean_squared_error'].mean()
    print(f"diamonds rmse score for Lasso regression and with f_regression, {score__:.4f
    diamonds_rmse_la_f.append(score__)
```

```
diamonds rmse score for linear regression, -0.3023
diamonds rmse score for linear regression and with mutual_info_regression, -0.3688 top1
diamonds rmse score for linear regression and with f_regression, -0.3688 top1
diamonds rmse score for Ridge regression, -0.3023
diamonds rmse score for Ridge regression and with mutual_info_regression, -0.3688 top1
diamonds rmse score for Ridge regression and with f_regression, -0.3688 top1
diamonds rmse score for Lasso regression, -0.8875
diamonds rmse score for Lasso regression and with mutual_info_regression, -0.8875 top1
diamonds rmse score for Lasso regression and with f_regression, -0.8875 top1
diamonds rmse score for linear regression, -0.3023
diamonds rmse score for linear regression and with mutual_info_regression, -0.3619 top2
diamonds rmse score for linear regression and with f_regression, -0.3499 top2
diamonds rmse score for Ridge regression, -0.3023
diamonds rmse score for Ridge regression and with mutual_info_regression, -0.3619 top2
diamonds rmse score for Ridge regression and with f_regression, -0.3499 top2
diamonds rmse score for Lasso regression, -0.8875
diamonds rmse score for Lasso regression and with mutual_info_regression, -0.8875 top2
diamonds rmse score for Lasso regression and with f_regression, -0.8875 top2
diamonds rmse score for linear regression, -0.3023
diamonds rmse score for linear regression and with mutual_info_regression, -0.3500 top3
diamonds rmse score for linear regression and with f_regression, -0.3500 top3
diamonds rmse score for Ridge regression, -0.3023
diamonds rmse score for Ridge regression and with mutual_info_regression, -0.3501 top3
diamonds rmse score for Ridge regression and with f_regression, -0.3501 top3
diamonds rmse score for Lasso regression, -0.8875
diamonds rmse score for Lasso regression and with mutual_info_regression, -0.8875 top3
diamonds rmse score for Lasso regression and with f_regression, -0.8875 top3
diamonds rmse score for linear regression, -0.3023
diamonds rmse score for linear regression and with mutual_info_regression, -0.3522 top4
diamonds rmse score for linear regression and with f_regression, -0.3522 top4
diamonds rmse score for Ridge regression, -0.3023
diamonds rmse score for Ridge regression and with mutual_info_regression, -0.3522 top4
```

```
diamonds rmse score for Ridge regression and with f_regression, -0.3522 top4
diamonds rmse score for Lasso regression, -0.8875
diamonds rmse score for Lasso regression and with mutual_info_regression, -0.8875 top4
diamonds rmse score for Lasso regression and with f_regression, -0.8875 top4
diamonds rmse score for linear regression, -0.3023
diamonds rmse score for linear regression and with mutual_info_regression, -0.3261 top5
diamonds rmse score for linear regression and with f_regression, -0.3448 top5
diamonds rmse score for Ridge regression, -0.3023
diamonds rmse score for Ridge regression and with mutual_info_regression, -0.3262 top5
diamonds rmse score for Ridge regression and with f_regression, -0.3448 top5
diamonds rmse score for Lasso regression, -0.8875
diamonds rmse score for Lasso regression and with mutual_info_regression, -0.8875 top5
diamonds rmse score for Lasso regression and with f_regression, -0.8875 top5
diamonds rmse score for linear regression, -0.3023
diamonds rmse score for linear regression and with mutual_info_regression, -0.3079 top6
diamonds rmse score for linear regression and with f_regression, -0.3079 top6
diamonds rmse score for Ridge regression, -0.3023
diamonds rmse score for Ridge regression and with mutual_info_regression, -0.3079 top6
diamonds rmse score for Ridge regression and with f_regression, -0.3079 top6
diamonds rmse score for Lasso regression, -0.8875
diamonds rmse score for Lasso regression and with mutual_info_regression, -0.8875 top6
diamonds rmse score for Lasso regression and with f_regression, -0.8875 top6
diamonds rmse score for linear regression, -0.3023
diamonds rmse score for linear regression and with mutual_info_regression, -0.3066 top7
diamonds rmse score for linear regression and with f_regression, -0.3081 top7
diamonds rmse score for Ridge regression, -0.3023
diamonds rmse score for Ridge regression and with mutual_info_regression, -0.3066 top7
diamonds rmse score for Ridge regression and with f_regression, -0.3081 top7
diamonds rmse score for Lasso regression, -0.8875
diamonds rmse score for Lasso regression and with mutual_info_regression, -0.8875 top7
diamonds rmse score for Lasso regression and with f_regression, -0.8875 top7
diamonds rmse score for linear regression, -0.3023
diamonds rmse score for linear regression and with mutual_info_regression, -0.3065 top8
diamonds rmse score for linear regression and with f_regression, -0.3065 top8
diamonds rmse score for Ridge regression, -0.3023
diamonds rmse score for Ridge regression and with mutual_info_regression, -0.3065 top8
diamonds rmse score for Ridge regression and with f_regression, -0.3065 top8
diamonds rmse score for Lasso regression, -0.8875
diamonds rmse score for Lasso regression and with mutual_info_regression, -0.8875 top8
diamonds rmse score for Lasso regression and with f_regression, -0.8875 top8
```

In [41]:

```
# for gt part
for k in range(1, len(gt_features)):
    gt_topk_M, gt_topk_F = select_topn_important_features(gt_standard_x, gt_standard_y,
    #gt rmse score for linear regression
    score_ = cross_validate(LinearRegression(), gt_standard_x, gt_standard_y, scoring =
    score__ = score_['test_neg_root_mean_squared_error'].mean()
    print(f"gt rmse score for linear regression, {score__:.4f}")
    gt_rmse_lr.append(score__)

    #gt rmse score for linear regression and with mutual_info_regression
    score_ = cross_validate(LinearRegression(), gt_topk_M, gt_standard_y, scoring = ['ne
    score__ = score_['test_neg_root_mean_squared_error'].mean()
    print(f"gt rmse score for linear regression and with mutual_info_regression, {score_
    gt_rmse_lr_m.append(score__)

    #gt rmse score for linear regression and with f_regression
    score_ = cross_validate(LinearRegression(), gt_topk_F, gt_standard_y, scoring = ['ne
    score__ = score_['test_neg_root_mean_squared_error'].mean()
    print(f"gt rmse score for linear regression and with f_regression, {score__:.4f} top
    gt_rmse_lr_f.append(score__)

    #gt rmse score for Ridge regression
    score_ = cross_validate(Ridge(), gt_standard_x, gt_standard_y, scoring = ['neg_root_
    score__ = score_['test_neg_root_mean_squared_error'].mean()
```

```
    print(f"gt rmse score for Ridge regression, {score__:.4f}")
    gt_rmse_r.append(score__)

    #gt rmse score for Ridge regression and with mutual_info_regression
    score_ = cross_validate(Ridge(), gt_topk_M, gt_standard_y, scoring = ['neg_root_mean
    score__ = score_['test_neg_root_mean_squared_error'].mean()
    print(f"gt rmse score for Ridge regression and with mutual_info_regression, {score__
    gt_rmse_r_m.append(score__)

    #gt rmse score for Ridge regression and with f_regression
    score_ = cross_validate(Ridge(), gt_topk_F, gt_standard_y, scoring = ['neg_root_mean
    score__ = score_['test_neg_root_mean_squared_error'].mean()
    print(f"gt rmse score for Ridge regression and with f_regression, {score__:.4f} top{
    gt_rmse_r_f.append(score__)

    #gt rmse score for Lasso regression
    score_ = cross_validate(Lasso(), gt_standard_x, gt_standard_y, scoring = ['neg_root_
    score__ = score_['test_neg_root_mean_squared_error'].mean()
    print(f"gt rmse score for Lasso regression, {score__:.4f}")
    gt_rmse_la.append(score__)

    #gt rmse score for Lasso regression and with mutual_info_regression
    score_ = cross_validate(Lasso(), gt_topk_M, gt_standard_y, scoring = ['neg_root_mean
    score__ = score_['test_neg_root_mean_squared_error'].mean()
    print(f"gt rmse score for Lasso regression and with mutual_info_regression, {score__
    gt_rmse_la_m.append(score__)

    #gt rmse score for Lasso regression and with f_regression
    score_ = cross_validate(Lasso(), gt_topk_F, gt_standard_y, scoring = ['neg_root_mean
    score__ = score_['test_neg_root_mean_squared_error'].mean()
    print(f"gt rmse score for Lasso regression and with f_regression, {score__:.4f} top{
    gt_rmse_la_f.append(score__)
```

```
gt rmse score for linear regression, -0.7382
gt rmse score for linear regression and with mutual_info_regression, -0.9985 top1
gt rmse score for linear regression and with f_regression, -0.8475 top1
gt rmse score for Ridge regression, -0.7379
gt rmse score for Ridge regression and with mutual_info_regression, -0.9985 top1
gt rmse score for Ridge regression and with f_regression, -0.8475 top1
gt rmse score for Lasso regression, -1.0059
gt rmse score for Lasso regression and with mutual_info_regression, -1.0059 top1
gt rmse score for Lasso regression and with f_regression, -1.0059 top1
gt rmse score for linear regression, -0.7382
gt rmse score for linear regression and with mutual_info_regression, -0.9841 top2
gt rmse score for linear regression and with f_regression, -0.8490 top2
gt rmse score for Ridge regression, -0.7379
gt rmse score for Ridge regression and with mutual_info_regression, -0.9841 top2
gt rmse score for Ridge regression and with f_regression, -0.8490 top2
gt rmse score for Lasso regression, -1.0059
gt rmse score for Lasso regression and with mutual_info_regression, -1.0059 top2
gt rmse score for Lasso regression and with f_regression, -1.0059 top2
gt rmse score for linear regression, -0.7382
gt rmse score for linear regression and with mutual_info_regression, -0.8208 top3
gt rmse score for linear regression and with f_regression, -0.8274 top3
gt rmse score for Ridge regression, -0.7379
gt rmse score for Ridge regression and with mutual_info_regression, -0.8208 top3
gt rmse score for Ridge regression and with f_regression, -0.8274 top3
gt rmse score for Lasso regression, -1.0059
gt rmse score for Lasso regression and with mutual_info_regression, -1.0059 top3
gt rmse score for Lasso regression and with f_regression, -1.0059 top3
gt rmse score for linear regression, -0.7382
gt rmse score for linear regression and with mutual_info_regression, -0.8344 top4
gt rmse score for linear regression and with f_regression, -0.8325 top4
gt rmse score for Ridge regression, -0.7379
gt rmse score for Ridge regression and with mutual_info_regression, -0.8342 top4
gt rmse score for Ridge regression and with f_regression, -0.8325 top4
```

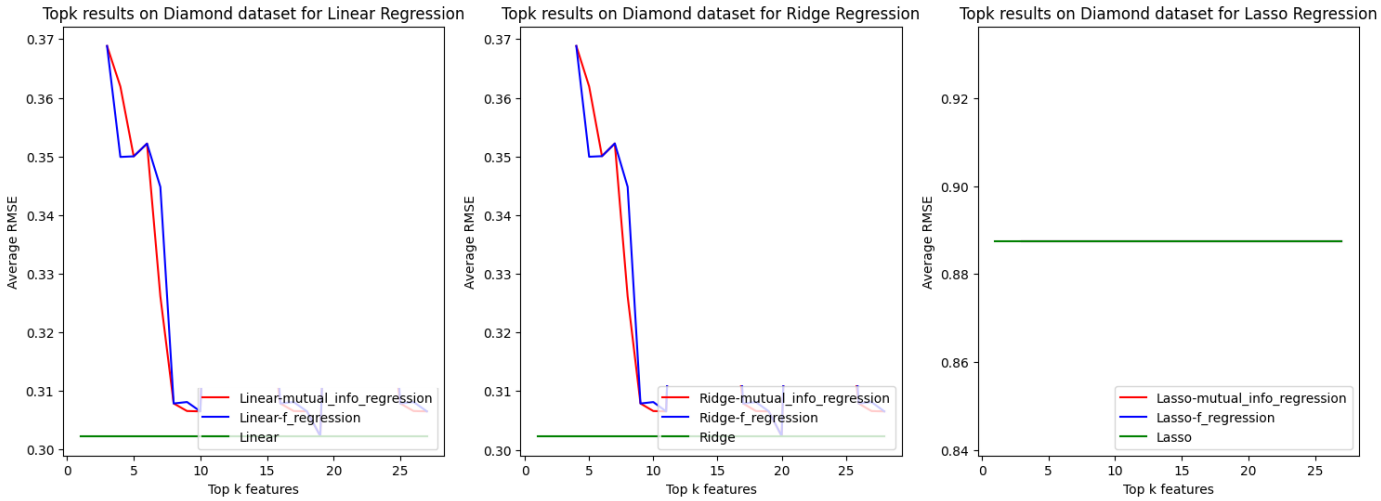
```
gt rmse score for Lasso regression, -1.0059
gt rmse score for Lasso regression and with mutual_info_regression, -1.0059 top4
gt rmse score for Lasso regression and with f_regression, -1.0059 top4
gt rmse score for linear regression, -0.7382
gt rmse score for linear regression and with mutual_info_regression, -0.8097 top5
gt rmse score for linear regression and with f_regression, -0.8394 top5
gt rmse score for Ridge regression, -0.7379
gt rmse score for Ridge regression and with mutual_info_regression, -0.8096 top5
gt rmse score for Ridge regression and with f_regression, -0.8394 top5
gt rmse score for Lasso regression, -1.0059
gt rmse score for Lasso regression and with mutual_info_regression, -1.0059 top5
gt rmse score for Lasso regression and with f_regression, -1.0059 top5
gt rmse score for linear regression, -0.7382
gt rmse score for linear regression and with mutual_info_regression, -0.8218 top6
gt rmse score for linear regression and with f_regression, -0.8511 top6
gt rmse score for Ridge regression, -0.7379
gt rmse score for Ridge regression and with mutual_info_regression, -0.8217 top6
gt rmse score for Ridge regression and with f_regression, -0.8510 top6
gt rmse score for Lasso regression, -1.0059
gt rmse score for Lasso regression and with mutual_info_regression, -1.0059 top6
gt rmse score for Lasso regression and with f_regression, -1.0059 top6
gt rmse score for linear regression, -0.7382
gt rmse score for linear regression and with mutual_info_regression, -0.7656 top7
gt rmse score for linear regression and with f_regression, -0.8178 top7
gt rmse score for Ridge regression, -0.7379
gt rmse score for Ridge regression and with mutual_info_regression, -0.7656 top7
gt rmse score for Ridge regression and with f_regression, -0.8178 top7
gt rmse score for Lasso regression, -1.0059
gt rmse score for Lasso regression and with mutual_info_regression, -1.0059 top7
gt rmse score for Lasso regression and with f_regression, -1.0059 top7
gt rmse score for linear regression, -0.7382
gt rmse score for linear regression and with mutual_info_regression, -0.7661 top8
gt rmse score for linear regression and with f_regression, -0.8061 top8
gt rmse score for Ridge regression, -0.7379
gt rmse score for Ridge regression and with mutual_info_regression, -0.7661 top8
gt rmse score for Ridge regression and with f_regression, -0.8059 top8
gt rmse score for Lasso regression, -1.0059
gt rmse score for Lasso regression and with mutual_info_regression, -1.0059 top8
gt rmse score for Lasso regression and with f_regression, -1.0059 top8
```

In [52]:

```
fig, axes = plt.subplots(1, 3, figsize=(18, 6))

#plot diamonds linear
axes[0].plot(np.arange(1, len(diamonds_rmse_lr_m) + 1, 1), np.negative(diamonds_rmse_lr_
axes[0].plot(np.arange(1, len(diamonds_rmse_lr_f) + 1, 1), np.negative(diamonds_rmse_lr_
axes[0].plot(np.arange(1, len(diamonds_rmse_lr) + 1, 1), np.negative(diamonds_rmse_lr),

axes[0].legend(loc='lower right')
axes[0].set(xlabel='Top k features',ylabel='Average RMSE')
axes[0].set_title('Topk results on Diamond dataset for Linear Regression')

#plot diamonds Ridge
axes[1].plot(np.arange(1, len(diamonds_rmse_r_m) + 1, 1), np.negative(diamonds_rmse_r_m)
axes[1].plot(np.arange(1, len(diamonds_rmse_r_f) + 1, 1), np.negative(diamonds_rmse_r_f)
axes[1].plot(np.arange(1, len(diamonds_rmse_r) + 1, 1), np.negative(diamonds_rmse_r), co

axes[1].legend(loc='lower right')
axes[1].set(xlabel='Top k features',ylabel='Average RMSE')
axes[1].set_title('Topk results on Diamond dataset for Ridge Regression')

#plot diamonds Lasso
axes[2].plot(np.arange(1, len(diamonds_rmse_la_m) + 1, 1), np.negative(diamonds_rmse_la_
axes[2].plot(np.arange(1, len(diamonds_rmse_la_f) + 1, 1), np.negative(diamonds_rmse_la_
axes[2].plot(np.arange(1, len(diamonds_rmse_la) + 1, 1), np.negative(diamonds_rmse_la),
```

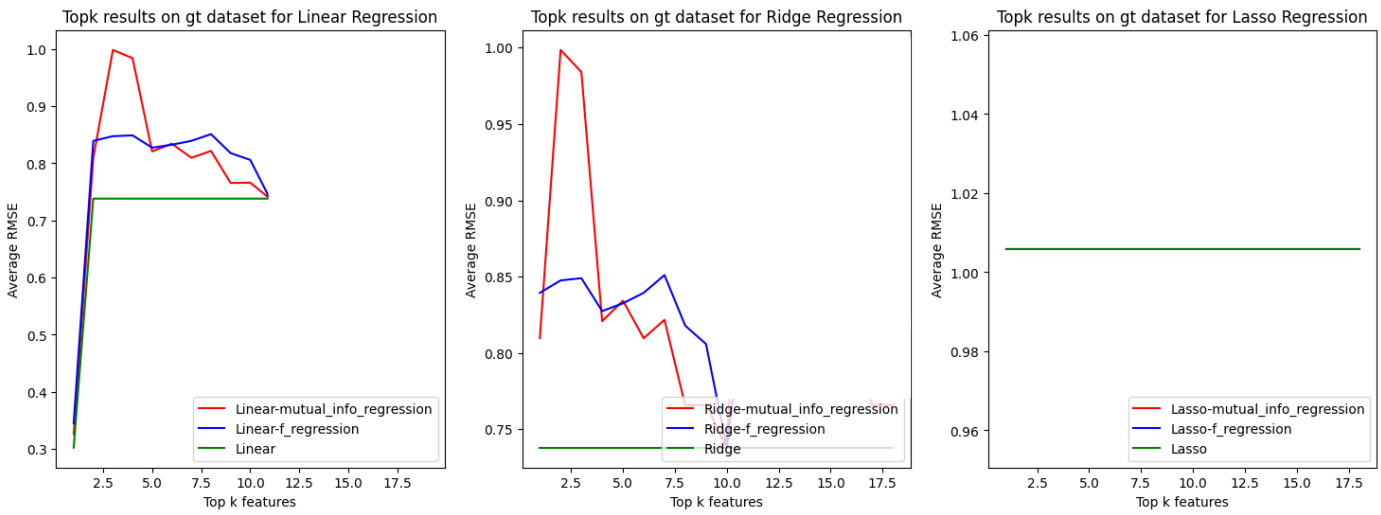
```
axes[2].legend(loc='lower right')
axes[2].set(xlabel='Top k features',ylabel='Average RMSE')
axes[2].set_title('Topk results on Diamond dataset for Lasso Regression')
```

Out[52]:
```
Text(0.5, 1.0, 'Topk results on Diamond dataset for Lasso Regression')
```



In [53]:
```
fig, axes = plt.subplots(1, 3, figsize=(18, 6))

#plot gt linear
axes[0].plot(np.arange(1, len(gt_rmse_lr_m) + 1, 1), np.negative(gt_rmse_lr_m), color =
axes[0].plot(np.arange(1, len(gt_rmse_lr_f) + 1, 1), np.negative(gt_rmse_lr_f), color =
axes[0].plot(np.arange(1, len(gt_rmse_lr) + 1, 1), np.negative(gt_rmse_lr), color = 'g',

axes[0].legend(loc='lower right')
axes[0].set(xlabel='Top k features',ylabel='Average RMSE')
axes[0].set_title('Topk results on gt dataset for Linear Regression')

#plot gt Ridge
axes[1].plot(np.arange(1, len(gt_rmse_r_m) + 1, 1), np.negative(gt_rmse_r_m), color = 'r
axes[1].plot(np.arange(1, len(gt_rmse_r_f) + 1, 1), np.negative(gt_rmse_r_f), color = 'b
axes[1].plot(np.arange(1, len(gt_rmse_r) + 1, 1), np.negative(gt_rmse_r), color = 'g', l

axes[1].legend(loc='lower right')
axes[1].set(xlabel='Top k features',ylabel='Average RMSE')
axes[1].set_title('Topk results on gt dataset for Ridge Regression')

#plot gt Lasso
axes[2].plot(np.arange(1, len(gt_rmse_la_m) + 1, 1), np.negative(gt_rmse_la_m), color =
axes[2].plot(np.arange(1, len(gt_rmse_la_f) + 1, 1), np.negative(gt_rmse_la_f), color =
axes[2].plot(np.arange(1, len(gt_rmse_la) + 1, 1), np.negative(gt_rmse_la), color = 'g',

axes[2].legend(loc='lower right')
axes[2].set(xlabel='Top k features',ylabel='Average RMSE')
axes[2].set_title('Topk results on gt dataset for Lasso Regression')
```

Out[53]:
```
Text(0.5, 1.0, 'Topk results on gt dataset for Lasso Regression')
```



Question4.2

In [34]:
```
# Finding the optimal penalty parameter
location = "cachedir"
memory = Memory(location=location, verbose=10)

pipe_ = Pipeline([
    ('kbest', SelectKBest()),
    ('model', "passthrough")
], memory = memory)

param_grid = [{
        'kbest__score_func': (mutual_info_regression, f_regression),
        'kbest__k': (1, 2, 3, 4, 5, 6, 7, 8, 9),
        'model': [Ridge(), Lasso()],
        'model__alpha': [10.0**x for x in np.arange(-3,4)]
    }
]
```

In [43]:
```
def _print_gridsearch_result(model, title):
    # print(model.cv_results_.keys())
    # print(f"Best estimator for {title}: ", model.best_estimator_)
    print(f"Best parameters for {title}: ", model.best_params_)
    print(f"Best score for {title}: ", model.best_score_)
```

In [37]:
```
grid_diamond = GridSearchCV(pipe_, param_grid = param_grid, cv = 10, n_jobs = -1, verbos
                    scoring = 'neg_root_mean_squared_error', return_train_score = True)
```

```
Fitting 10 folds for each of 252 candidates, totalling 2520 fits
[Memory]0.0s, 0.0min    : Loading _fit_transform_one from cachedir\joblib\sklearn\pipeli
ne\_fit_transform_one\d116ca552df2ab81931753098ad2ebb1
_____________________________________fit_transform_one cache loaded - 0.0s, 0.0min
```

In [42]:
```
grid_gt = GridSearchCV(pipe_, param_grid = param_grid, cv = 10, n_jobs = -1, verbose = 1
                    scoring = 'neg_root_mean_squared_error', return_train_score = True)
```

```
Fitting 10 folds for each of 252 candidates, totalling 2520 fits
_____________________________________________________________________________________
[Memory] Calling sklearn.pipeline._fit_transform_one...
_fit_transform_one(SelectKBest(k=9,
            score_func=<function mutual_info_regression at 0x000001C491BDFCA0>),
              AT        AP        AH      AFDP      GTEP       TIT       TAT  \
0     -1.762362  0.871052  0.401627 -0.451875 -0.377702  0.272119  0.536589
1     -1.801920  0.809164  0.440351 -0.458207 -0.384376  0.266417  0.568742
2     -1.854113  0.824636  0.483432 -0.442831 -0.375081  0.289227  0.589203
3     -1.875718  0.809164  0.523263 -0.445415 -0.393909  0.289227  0.586280
4     -1.874644  0.731804  0.505837 -0.448904 -0.392479  0.255012  0.561434
```

```
...          ...       ...       ...       ...       ...       ...       ...
36728 -1.891401  2.387315  1.060287 -0.981255 -1.543602 -2.533516 -0.667691
36729 -1.818812  2.402787  1.118097 -0.947401 -1.560524 -2.499301 -0.566847
36730 -1.642292  2.387315  1.199903 -0.791702 -1.598418 -2.476491 -0.391466
36731 -1.588354  2.418260  1.129438  0.074403 -0.476847 -0.258214  0.577511
36732 -1.567474  2.433732  1.153434 -0.065016 -0.724469 -0.771440  0.302748

            TEY       CDP
0      0.074502 -0.149273
1      0.074502 -0.154783
2      0.102033 -0.017015
3      0.097551 -0.064774
4      0.074502 -0.138251
...         ...       ...
36728 -1.563948 -1.515021
36729 -1.582516 -1.576558
36730 -1.645262 -1.468179
36731 -0.134226 -0.265917
36732 -0.518388 -0.549721

[36733 rows x 9 columns],
0        1.426499
1        1.462891
2        1.582687
3        1.473852
4        1.433006
           ...
36728    2.044745
36729    2.017086
36730    2.642011
36731   -0.047530
36732    3.763160
Name: NOX, Length: 36733, dtype: float64,
None, message_clsname='Pipeline', message=None)
_____________________________________________________fit_transform_one - 1.4s, 0.0min
```

In [45]:
```
_print_gridsearch_result(grid_diamond, "diamond with select k best and Ridge/Lasso Regre
```

```
Best parameters for diamond with select k best and Ridge/Lasso Regression:  {'kbest__k':
6, 'kbest__score_func': <function f_regression at 0x000001C491599790>, 'model': Ridge(al
pha=0.001), 'model__alpha': 0.001}
Best score for diamond with select k best and Ridge/Lasso Regression:  -0.30092878389367
53
```

In [46]:
```
_print_gridsearch_result(grid_gt, "gt with select k best and Ridge/Lasso Regression")
```

```
Best parameters for gt with select k best and Ridge/Lasso Regression:  {'kbest__k': 9,
'kbest__score_func': <function mutual_info_regression at 0x000001C491BDFCA0>, 'model': L
asso(alpha=0.001), 'model__alpha': 0.001}
Best score for gt with select k best and Ridge/Lasso Regression:  -0.7343103801066775
```

Question4.3 is stated on our report

Question4.4

In [62]:
```
print("diamond")
for feature in diamonds_features:
    slope, intercept, r, p, stderr = ss.linregress(diamond_standard_x[feature], diamond_
    print(f"{str(feature)} p = {p:.4f}")
```

```
diamond
carat p = 0.0000
cut p = 0.0000
color p = 0.0000
clarity p = 0.0000
```

```
depth p = 0.0134
table p = 0.0000
x p = 0.0000
y p = 0.0000
z p = 0.0000
```

In [63]:
```
print("gt")
for feature in gt_features:
    slope, intercept, r, p, stderr = ss.linregress(gt_standard_x[feature], gt_standard_y
    print(f"{str(feature)} p = {p:.4f}")
```

```
gt
AT p = 0.0000
AP p = 0.0000
AH p = 0.0000
AFDP p = 0.0000
GTEP p = 0.0000
TIT p = 0.0000
TAT p = 0.0000
TEY p = 0.0000
CDP p = 0.0000
```

In [54]:
```
diamond_sm_fit = sm.OLS(diamond_standard_y, sm.add_constant(diamond_standard_x)).fit()
diamond_sm_fit.pvalues.sort_values(ascending=False)
```

Out[54]:
```
const       1.000000e+00
y           2.107936e-01
z           1.613988e-01
cut         4.621671e-56
depth      2.045036e-113
x          1.327592e-125
table      2.044967e-133
carat       0.000000e+00
color       0.000000e+00
clarity     0.000000e+00
dtype: float64
```

In [55]:
```
gt_sm_fit = sm.OLS(gt_standard_y, sm.add_constant(gt_standard_x)).fit()
gt_sm_fit.pvalues.sort_values(ascending=False)
```

Out[55]:
```
const     1.000000e+00
GTEP      4.225138e-02
CDP       4.355106e-03
AFDP      5.737233e-16
AP       1.254925e-186
AT        0.000000e+00
AH        0.000000e+00
TIT       0.000000e+00
TAT       0.000000e+00
TEY       0.000000e+00
dtype: float64
```

Question5

Question5.1

In [20]:
```
diamond_standard_x_top6 = SelectKBest(score_func = f_regression, k = 6).fit_transform(di
gt_standard_x_top9 = SelectKBest(score_func = mutual_info_regression, k = 9).fit_transfo
```

In [27]:
```
location = "cachedir"
memory = Memory(location=location, verbose=10)

poly_pipe_diamond_ = Pipeline([
    ('poly_transform', PolynomialFeatures()),
```

```
    ('model', Ridge(alpha=0.001))
],memory=memory)

poly_pipe_gt_ = Pipeline([
    ('poly_transform', PolynomialFeatures()),
    ('model', Lasso(alpha=0.001))
],memory=memory)

poly_param_= {
    'poly_transform__degree': np.arange(1,6,1)
}
```

In [85]:
```
poly_grid_diamond  = GridSearchCV(poly_pipe_diamond_ , param_grid=poly_param_, cv=10, n_
```

```
Fitting 10 folds for each of 4 candidates, totalling 40 fits
[Memory]0.0s, 0.0min    : Loading _fit_transform_one from cachedir\joblib\sklearn\pipeli
ne\_fit_transform_one\012759a68728318cec0c39231233ebab
_____________________________________fit_transform_one cache loaded - 0.0s, 0.0min
```

In [29]:
```
poly_grid_gt  = GridSearchCV(poly_pipe_gt_ , param_grid=poly_param_, cv=10, n_jobs=-1, v
```

```
Fitting 10 folds for each of 4 candidates, totalling 40 fits
[Memory]0.0s, 0.0min    : Loading _fit_transform_one from cachedir\joblib\sklearn\pipeli
ne\_fit_transform_one\8315585baec85d14a2daa7692ec6c276
_____________________________________fit_transform_one cache loaded - 0.0s, 0.0min
```

In [86]:
```
_print_gridsearch_result(poly_grid_diamond, "PolynomialFeatures of Diamonds" )
```

```
Best parameters for PolynomialFeatures of Diamonds:  {'poly_transform__degree': 1}
Best score for PolynomialFeatures of Diamonds:  -0.30787053657268926
```

In [34]:
```
print(poly_grid_diamond.cv_results_.keys())
```

```
dict_keys(['mean_fit_time', 'std_fit_time', 'mean_score_time', 'std_score_time', 'param_
poly_transform__degree', 'params', 'split0_test_score', 'split1_test_score', 'split2_tes
t_score', 'split3_test_score', 'split4_test_score', 'split5_test_score', 'split6_test_sc
ore', 'split7_test_score', 'split8_test_score', 'split9_test_score', 'mean_test_score',
'std_test_score', 'rank_test_score', 'split0_train_score', 'split1_train_score', 'split2
_train_score', 'split3_train_score', 'split4_train_score', 'split5_train_score', 'split6
_train_score', 'split7_train_score', 'split8_train_score', 'split9_train_score', 'mean_t
rain_score', 'std_train_score'])
```

In [32]:
```
_print_gridsearch_result(poly_grid_gt, "PolynomialFeatures of gt" )
```

```
Best parameters for PolynomialFeatures of gt:  {'poly_transform__degree': 3}
Best score for PolynomialFeatures of gt:  -0.5951110578174913
```

In [71]:
```
s_d = SelectKBest(score_func = f_regression, k = 6)
s_d.fit_transform(diamond_standard_x, diamond_standard_y)
column_s = diamond_standard_x.columns[s_d.get_support()].tolist()

d_params = poly_grid_diamond.best_estimator_.get_params()
d_coefs = d_params['model'].coef_
d_names = d_params['poly_transform'].get_feature_names(column_s)
d_sorted_indice = np.argsort(-abs(d_coefs))
salient_features =[d_names[i] for i in d_sorted_indice[:6]]
print ('Top 6 Salient features (Diamond) in order:',salient_features)
```

```
Top 6 Salient features (Diamond) in order: ['carat', 'clarity', 'x', 'color', 'z', 'y']
```

In [1]:
```
s_g = SelectKBest(score_func = mutual_info_regression, k = 9)
s_g.fit_transform(gt_standard_x, gt_standard_y)
column_g = gt_standard_x.columns[s_g.get_support()].tolist()

d_params = poly_grid_gt.best_estimator_.get_params()
d_coefs = d_params['model'].coef_
d_names = d_params['poly_transform'].get_feature_names(column_g)
```

```
d_sorted_indice = np.argsort(-abs(d_coefs))
salient_features =[d_names[i] for i in d_sorted_indice[:9]]
print ('Top 9 Salient features (Diamond) in order:',salient_features)
```

```
Top 9 Salient features (gt) in order: ['TIT', 'TEY', 'TAT', 'AT', 'CDP', 'AT TEY^2', 'CD
P^2', 'AFDP TAT', 'TIT TAT']
```

Question5.2 is stated on our report

Question6

Question6.1

In [11]:
```
mlpr = Pipeline([
    ('model', MLPRegressor()),

], memory=memory)

param_list = {
    "model__hidden_layer_sizes": [(30, 40), (30, 60), (40, 40), (40, 60), (60, 60)],
    "model__activation": ["identity", "logistic", "tanh", "relu"],
    "model__solver": ["lbfgs", "sgd", "adam"],
}

grid_diamond_mlp = GridSearchCV(mlpr, param_grid = param_list, cv = 5, n_jobs = -1, verb
                      scoring = 'neg_root_mean_squared_error', return_train_score = True)
```

```
Fitting 5 folds for each of 60 candidates, totalling 300 fits
```

In [12]:
```
grid_gt_mlp = GridSearchCV(mlpr, param_grid = param_list, cv = 5, n_jobs = -1, verbose =
                      scoring = 'neg_root_mean_squared_error', return_train_score = True)
```

```
Fitting 5 folds for each of 60 candidates, totalling 300 fits
```

In [13]:
```
_print_gridsearch_result(grid_diamond_mlp, "Neural Network of Diamonds" )
rmse = np.sqrt(-grid_diamond_mlp.best_score_)
print(f"original rmse = {rmse}")

_print_gridsearch_result(grid_gt_mlp, "Neural Network of gt" )
rmse = np.sqrt(-grid_gt_mlp.best_score_)
print(f"new rmse = {rmse}")
```

```
Best parameters for Neural Network of Diamonds:  {'model__activation': 'relu', 'model__h
idden_layer_sizes': (30, 60), 'model__solver': 'sgd'}
Best score for Neural Network of Diamonds:  -0.2482196345112592
original rmse = 0.4982164534730454
Best parameters for Neural Network of gt:  {'model__activation': 'logistic', 'model__hid
den_layer_sizes': (30, 60), 'model__solver': 'sgd'}
Best score for Neural Network of gt:  -0.628648815583329
new rmse = 0.7928737702707341
```

Question6.2 6.3 6.4 are stated on our report

Question7

Question7.1

In [11]:
```
from sklearn.ensemble import RandomForestRegressor
pipeline_forest = Pipeline([
    ('model', RandomForestRegressor())
],memory=memory)

param_grid_forest = {
```

```
    'model__max_features': np.arange(1, 5, 1),
    'model__n_estimators': np.arange(10, 40, 10),
    'model__max_depth': np.arange(1, 5, 1)
}
```

In [14]:
```
grid_diamond_forest = GridSearchCV(pipeline_forest, param_grid = param_grid_forest, cv =
                      scoring = 'neg_root_mean_squared_error', return_train_score = True)
```

```
Fitting 10 folds for each of 48 candidates, totalling 480 fits
```

In [16]:
```
_print_gridsearch_result(grid_diamond_forest, "RandomForestRegressor for diamond")
```

```
Best parameters for RandomForestRegressor for diamond:  {'model__max_depth': 4, 'model__
max_features': 4, 'model__n_estimators': 10}
Best score for RandomForestRegressor for diamond:  -0.32233482446629436
```

In [15]:
```
grid_gt_forest = GridSearchCV(pipeline_forest, param_grid = param_grid_forest, cv = 10,
                      scoring = 'neg_root_mean_squared_error', return_train_score = True)
```

```
Fitting 10 folds for each of 48 candidates, totalling 480 fits
```

In [17]:
```
_print_gridsearch_result(grid_gt_forest, "RandomForestRegressor for gt")
```

```
Best parameters for RandomForestRegressor for gt:  {'model__max_depth': 4, 'model__max_f
eatures': 4, 'model__n_estimators': 30}
Best score for RandomForestRegressor for gt:  -0.7032853751208423
```

Question7.2 is stated on our report

Question7.3

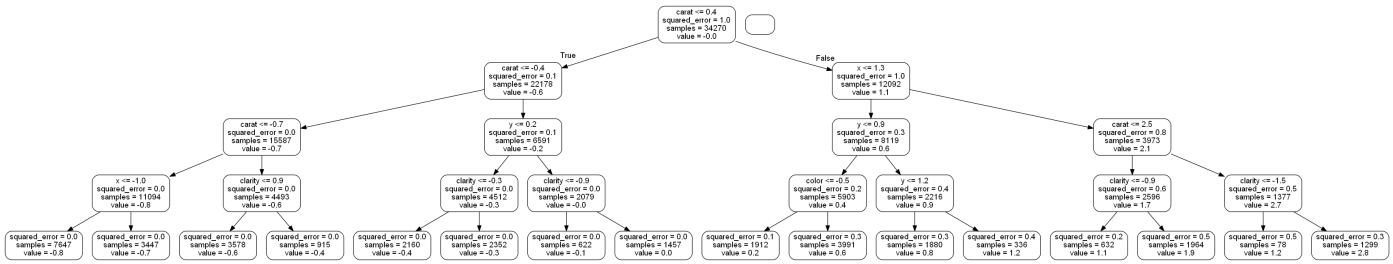
In [11]:
```
s_d = SelectKBest(score_func = f_regression, k = 6)
s_d.fit_transform(diamond_standard_x, diamond_standard_y)
column_s = diamond_standard_x.columns[s_d.get_support()].tolist()

s_g = SelectKBest(score_func = mutual_info_regression, k = 9)
s_g.fit_transform(gt_standard_x, gt_standard_y)
column_g = gt_standard_x.columns[s_g.get_support()].tolist()
```

In [2]:
```
print(column_s)
print(column_g)
```

```
 ['carat', 'color', 'clarity', 'x', 'y', 'z']
 ['TIT', 'AP', 'AH', 'AFDP', 'GTEP', 'AT', 'TAT', 'TEY', 'CDP']
```

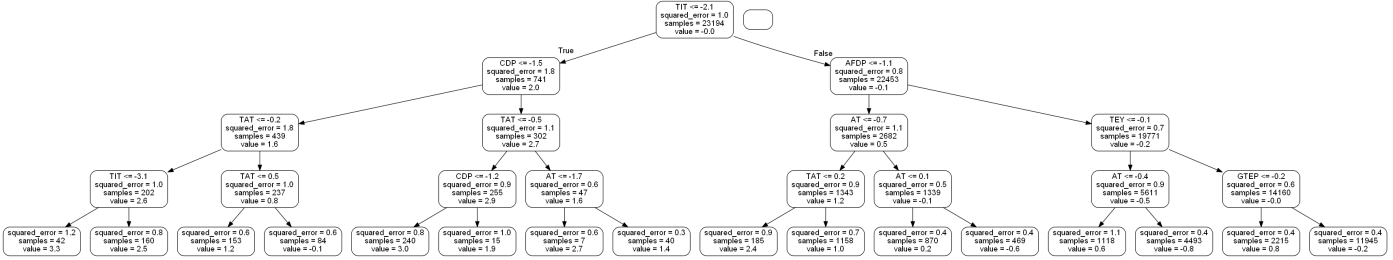
In [15]:
```
rf_best_diamond = RandomForestRegressor(max_depth=4, max_features=4, n_estimators=10)
rf_best_diamond.fit(diamond_standard_x_top6, diamond_standard_y)
export_graphviz(rf_best_diamond.estimators_[1], out_file = 'rf_best_diamond.dot', featur
(graph, ) = pydot.graph_from_dot_file('rf_best_diamond.dot')
Image(graph.create_png())
```

Out[15]:



In [16]:
```
rf_best_gt = RandomForestRegressor(max_depth=4, max_features=4, n_estimators=30)
rf_best_gt.fit(gt_standard_x_top9, gt_standard_y)
export_graphviz(rf_best_gt.estimators_[1], out_file = 'rf_best_gt.dot', feature_names =
(graph, ) = pydot.graph_from_dot_file('rf_best_gt.dot')
Image(graph.create_png())
```

Out[16]:



Question7.4

In [2]:
```
rf_diamond_ = RandomForestRegressor(n_estimators=10, max_features=4, max_depth=4, oob_sc
rf_diamond_.fit(diamond_standard_x_top6, diamond_standard_y)

print('Best Random Forest Model for Diamond Dataset:')
print('OOB score: %.4f' %(rf_diamond_.oob_score_))
print('R^2 score: %.4f' %(rf_diamond_.score(diamond_standard_x_top6, diamond_standard_y)
```

```
Best Random Forest Model for Diamond Dataset:
 OOB score: 0.9113
 R^2 score: 0.9283
```

In [3]:
```
rf_gt_ = RandomForestRegressor(n_estimators=30, max_features=4, max_depth=4, oob_score=T
rf_gt_.fit(gt_standard_x, gt_standard_y)

print('Best Random Forest Model for Diamond Dataset:')
print('OOB score: %.4f' %(rf_gt_.oob_score_))
print('R^2 score: %.4f' %(rf_gt_.score(gt_standard_x, gt_standard_y)))
```

```
Best Random Forest Model for gt Dataset:
 OOB score: 0.5951
 R^2 score: 0.6063
```

Question8

Question8.1 is stated on our report

Question8.2 is stated on our report

In [43]:
```
### LightGBM
lbg_pip_ = Pipeline([
    ('model', lgb.LGBMRegressor())
],memory = memory)

param_lgb_ = {
    'model__num_leaves': [7, 14, 21, 28, 31, 50],
    'model__learning_rate': [0.1, 0.03, 0.003],
    'model__max_depth': [-1, 3, 5],
    'model__n_estimators': [50, 100, 200, 500],
}
```

In [44]:
```
lg_diamond = BayesSearchCV(lbg_pip_, search_spaces=param_lgb_, cv=10, n_jobs=-1, verbose
    scoring='neg_root_mean_squared_error', return_train_score=True).fit(diamond_standard
```

```
Fitting 10 folds for each of 1 candidates, totalling 10 fits
Fitting 10 folds for each of 1 candidates, totalling 10 fits
Fitting 10 folds for each of 1 candidates, totalling 10 fits
Fitting 10 folds for each of 1 candidates, totalling 10 fits
Fitting 10 folds for each of 1 candidates, totalling 10 fits
Fitting 10 folds for each of 1 candidates, totalling 10 fits
Fitting 10 folds for each of 1 candidates, totalling 10 fits
Fitting 10 folds for each of 1 candidates, totalling 10 fits
Fitting 10 folds for each of 1 candidates, totalling 10 fits
```

```
Fitting 10 folds for each of 1 candidates, totalling 10 fits
Fitting 10 folds for each of 1 candidates, totalling 10 fits
Fitting 10 folds for each of 1 candidates, totalling 10 fits
Fitting 10 folds for each of 1 candidates, totalling 10 fits
Fitting 10 folds for each of 1 candidates, totalling 10 fits
Fitting 10 folds for each of 1 candidates, totalling 10 fits
Fitting 10 folds for each of 1 candidates, totalling 10 fits
Fitting 10 folds for each of 1 candidates, totalling 10 fits
Fitting 10 folds for each of 1 candidates, totalling 10 fits
Fitting 10 folds for each of 1 candidates, totalling 10 fits
Fitting 10 folds for each of 1 candidates, totalling 10 fits
Fitting 10 folds for each of 1 candidates, totalling 10 fits
Fitting 10 folds for each of 1 candidates, totalling 10 fits
Fitting 10 folds for each of 1 candidates, totalling 10 fits
Fitting 10 folds for each of 1 candidates, totalling 10 fits
Fitting 10 folds for each of 1 candidates, totalling 10 fits
Fitting 10 folds for each of 1 candidates, totalling 10 fits
Fitting 10 folds for each of 1 candidates, totalling 10 fits
Fitting 10 folds for each of 1 candidates, totalling 10 fits
Fitting 10 folds for each of 1 candidates, totalling 10 fits
Fitting 10 folds for each of 1 candidates, totalling 10 fits
Fitting 10 folds for each of 1 candidates, totalling 10 fits
Fitting 10 folds for each of 1 candidates, totalling 10 fits
Fitting 10 folds for each of 1 candidates, totalling 10 fits
Fitting 10 folds for each of 1 candidates, totalling 10 fits
Fitting 10 folds for each of 1 candidates, totalling 10 fits
Fitting 10 folds for each of 1 candidates, totalling 10 fits
Fitting 10 folds for each of 1 candidates, totalling 10 fits
Fitting 10 folds for each of 1 candidates, totalling 10 fits
Fitting 10 folds for each of 1 candidates, totalling 10 fits
Fitting 10 folds for each of 1 candidates, totalling 10 fits
Fitting 10 folds for each of 1 candidates, totalling 10 fits
Fitting 10 folds for each of 1 candidates, totalling 10 fits
Fitting 10 folds for each of 1 candidates, totalling 10 fits
Fitting 10 folds for each of 1 candidates, totalling 10 fits
Fitting 10 folds for each of 1 candidates, totalling 10 fits
Fitting 10 folds for each of 1 candidates, totalling 10 fits
Fitting 10 folds for each of 1 candidates, totalling 10 fits
Fitting 10 folds for each of 1 candidates, totalling 10 fits
Fitting 10 folds for each of 1 candidates, totalling 10 fits
Fitting 10 folds for each of 1 candidates, totalling 10 fits
```

In [45]:
```python
lg_gt = BayesSearchCV(lbg_pip_, search_spaces=param_lgb_, cv=10, n_jobs=-1, verbose=1,
    scoring='neg_root_mean_squared_error', return_train_score=True).fit(gt_standard_x_to
```

```
Fitting 10 folds for each of 1 candidates, totalling 10 fits
Fitting 10 folds for each of 1 candidates, totalling 10 fits
Fitting 10 folds for each of 1 candidates, totalling 10 fits
Fitting 10 folds for each of 1 candidates, totalling 10 fits
Fitting 10 folds for each of 1 candidates, totalling 10 fits
Fitting 10 folds for each of 1 candidates, totalling 10 fits
Fitting 10 folds for each of 1 candidates, totalling 10 fits
Fitting 10 folds for each of 1 candidates, totalling 10 fits
Fitting 10 folds for each of 1 candidates, totalling 10 fits
Fitting 10 folds for each of 1 candidates, totalling 10 fits
Fitting 10 folds for each of 1 candidates, totalling 10 fits
Fitting 10 folds for each of 1 candidates, totalling 10 fits
Fitting 10 folds for each of 1 candidates, totalling 10 fits
Fitting 10 folds for each of 1 candidates, totalling 10 fits
Fitting 10 folds for each of 1 candidates, totalling 10 fits
Fitting 10 folds for each of 1 candidates, totalling 10 fits
Fitting 10 folds for each of 1 candidates, totalling 10 fits
Fitting 10 folds for each of 1 candidates, totalling 10 fits
Fitting 10 folds for each of 1 candidates, totalling 10 fits
Fitting 10 folds for each of 1 candidates, totalling 10 fits
Fitting 10 folds for each of 1 candidates, totalling 10 fits
```

```
Fitting 10 folds for each of 1 candidates, totalling 10 fits
Fitting 10 folds for each of 1 candidates, totalling 10 fits
Fitting 10 folds for each of 1 candidates, totalling 10 fits
Fitting 10 folds for each of 1 candidates, totalling 10 fits
Fitting 10 folds for each of 1 candidates, totalling 10 fits
Fitting 10 folds for each of 1 candidates, totalling 10 fits
Fitting 10 folds for each of 1 candidates, totalling 10 fits
Fitting 10 folds for each of 1 candidates, totalling 10 fits
Fitting 10 folds for each of 1 candidates, totalling 10 fits
Fitting 10 folds for each of 1 candidates, totalling 10 fits
Fitting 10 folds for each of 1 candidates, totalling 10 fits
Fitting 10 folds for each of 1 candidates, totalling 10 fits
Fitting 10 folds for each of 1 candidates, totalling 10 fits
Fitting 10 folds for each of 1 candidates, totalling 10 fits
Fitting 10 folds for each of 1 candidates, totalling 10 fits
Fitting 10 folds for each of 1 candidates, totalling 10 fits
Fitting 10 folds for each of 1 candidates, totalling 10 fits
Fitting 10 folds for each of 1 candidates, totalling 10 fits
Fitting 10 folds for each of 1 candidates, totalling 10 fits
Fitting 10 folds for each of 1 candidates, totalling 10 fits
Fitting 10 folds for each of 1 candidates, totalling 10 fits
Fitting 10 folds for each of 1 candidates, totalling 10 fits
Fitting 10 folds for each of 1 candidates, totalling 10 fits
Fitting 10 folds for each of 1 candidates, totalling 10 fits
Fitting 10 folds for each of 1 candidates, totalling 10 fits
Fitting 10 folds for each of 1 candidates, totalling 10 fits
Fitting 10 folds for each of 1 candidates, totalling 10 fits
Fitting 10 folds for each of 1 candidates, totalling 10 fits
Fitting 10 folds for each of 1 candidates, totalling 10 fits
```

In [46]:
```
_print_gridsearch_result(lg_diamond, "LightGBM of Diamonds" )
```

```
Best parameters for LightGBM of Diamonds:  OrderedDict([('model__learning_rate', 0.1),
('model__max_depth', 5), ('model__n_estimators', 500), ('model__num_leaves', 7)])
Best score for LightGBM of Diamonds:  -0.15290674289254738
RMSE is 0.3910329179142689
```

In [47]:
```
_print_gridsearch_result(lg_gt, "LightGBM of gt" )
```

```
Best parameters for LightGBM of gt:  OrderedDict([('model__learning_rate', 0.1), ('model
__max_depth', 5), ('model__n_estimators', 200), ('model__num_leaves', 31)])
Best score for LightGBM of gt:  -0.5926654920848259
RMSE is 0.769847707072526
```

In [53]:
```
### CatBoostClassifier

# model_cat = CatBoostClassifier()

# grid_cat = {'learning_rate': [0.03, 0.1],
#          'depth': [4, 6, 10],
#          'l2_leaf_reg': [1, 3, 5, 7, 9]}
```

Question8.3 is stated on our report

EC ENGR 219

2023 Winter

Twitter Design

Wenxin Cheng	706070535	wenxin0319@g.ucla.edu
Yuxin Yin	606073780	yyxyy999@g.ucla.edu
Yingqian Zhao	306071513	zhaoyq99@g.ucla.edu

- **QUESTION 9**

Question 9.1: Report the following statistics for each hashtag, i.e. each file has

- **Average number of tweets per hour**
- **Average number of followers of users posting the tweets per tweet (to make it simple, we average over the number of tweets; if a users posted twice, we count the user and the user’s followers twice as well)**
- **Average number of retweets per tweet**

Reading txt file is slow, so we use read chunks to speed up the reading

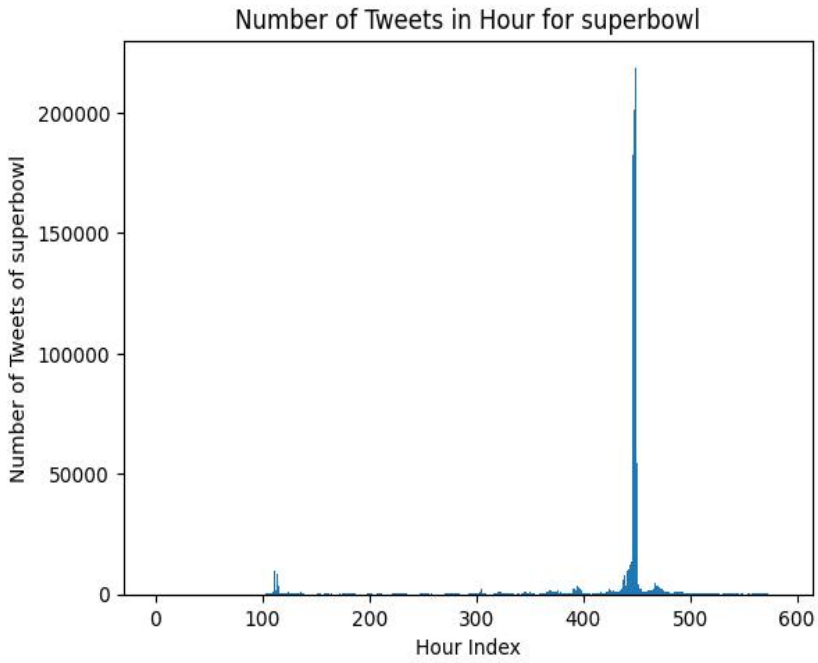
```
1.    def read_in_chunks(file_object, chunk_size=102400):
2.        """Lazy function (generator) to read a file piece by piece.
3.        Default chunk size: 1k."""
4.        while True:
5.            data = file_object.readlines(chunk_size)
6.            if not data:
7.                break
8.            yield data
```

File	#gohawks	#gopatriots	#nfl	#patriots	#sb49	#superbowl
Total number of tweets	169122	23511	233022	440621	743649	1213813
Average number of tweets per hour	292.488	40.955	397.021	750.894	1276.857	2072.118
Average number of followers	2217.924	1427.253	4662.375	3280.464	10374.160	8814.968
Average number of retweets	2.013	1.408	1.534	1.785	2.527	2.391

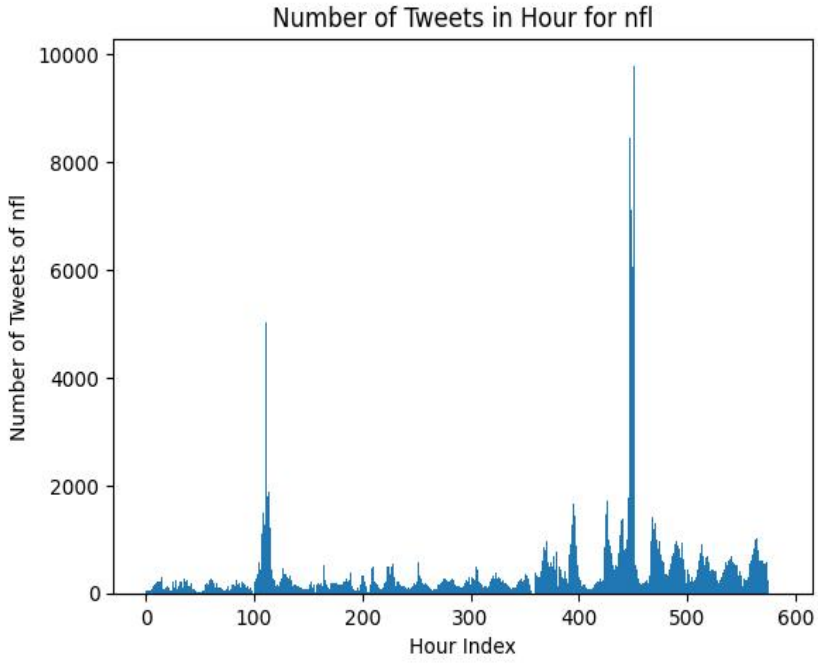
Question 9.2: Plot “number of tweets in hour” over time for #SuperBowl and #NFL (a bar

plot with 1-hour bins). The tweets are stored in separate files for different hashtags and files are named as tweet [#hashtag].txt.

Number of Tweets in Hour for SuperBowl



Number of Tweets in Hour for NFL



● QUESTION 10

Task Description

In this project, our objective was to analyze Twitter data for predictive purposes. Our approach involved performing data cleaning and feature extraction on the original Twitter dataset. We then integrated time period data of different window sizes through various time windows to provide meaningful insights for **regression analysis**.

The regression analysis was based on the features extracted from the Twitter data, and we constructed a prediction model for the **next period based on the number of Twitter posts within a particular time window**.

We also used a classification model to screen out legitimate Twitter posting locations. We performed clustering and classification to construct **a location classification model based on Twitter text**, which enabled us to categorize tweets according to their location.

In addition to the above analysis, we also filtered out mentions of player's names in tweets to construct a **word cloud**. This allowed us to visualize the most commonly used words in tweets related to the Topic of interest, while removing any potentially biased mentions of individual players.

Furthermore, we leveraged the power of GPT-3 to **generate tweets** based on the scores obtained from the regression analy allowed us to simulate how Twitter users might react to different game outcomes or events, and gain a better understanding of the potential impact of these outcomes on social media.

Overall, this project involved a comprehensive analysis of Twitter data using various techniques, including **data cleaning, feature extraction, regression analysis, and classification.** The insights gained from this analysis could have significant implications for predicting future trends based on Twitter data.

- **Ⅰ Data Preparation**
- **Raw Data Analysis**

In this project, our goal was to analyze a large dataset of Twitter posts from a specific time period. To accomplish this, we began by selecting a specific time window to analyze, from **2015/2/1 18:30 to 2015/2/1 22:30**, as this was the period of interest for our analysis.

We then filtered the dataset to include only the tweets that were posted during this specific time window, resulting in a dataset of **204993** tweets.

For each tweet in the filtered dataset, we extracted a set of **14** features. The first feature we extracted was the number of days between the tweet's creation date and the date when the user's account was created. We also extracted a range of other features, including the number of tweets posted by the user, the timestamp of the tweet's creation, the number of times the tweet has been retweeted, and the number of followers of the user who created the tweet.

Features	Function
"num_tweet"	The number of tweets (always 1 since we are analyzing individual tweets)
"created_at"	The timestamp of the tweet's creation
"num_retweet"	The number of times the tweet has been retweeted
"num_followers"	The number of followers of the user who created the tweet
"ranking_score"	A score measuring the influence of the tweet
"user_activity"	The number of tweets posted by the user per day
"user_id"	The user ID of the user who created the tweet
"user_location"	The location of the user who created the tweet (if available)
"user_mentions"	The number of other users mentioned in the tweet
"text"	Tweet text
"polarity"	The sentiment of the tweets
"positive"	Whether the sentiment is positive(True/False)
"neutral"	Whether the sentiment is neutral(True/False)
"negative"	Whether the sentiment is negative(True/False)

Overall, our goal was to extract meaningful insights from the Twitter data, and the features we extracted were selected based on their potential to provide useful information for our analysis.

	num_tweet	created_at	num_retweet	num_followers	ranking_score	user_activity	user_id	user_location	user_mentions	text	polarity	positive	neutral	negative
0	1	2015-02-01 18:23:00	1	2941.0	7.706944	37.000000	34852676	816/770/803	0	Down goes Brady SuperBowlXLIX	-1	False	False	True
1	1	2015-02-01 18:23:00	7	5009.0	7.486162	2.701126	14386730	The Hub of the Universe	0	RT if you still believe GoPats SuperBowlXLIX	0	False	True	False
2	1	2015-02-01 18:23:00	2	1486.0	7.302760	5.843216	133804848	Chicago, IL	0	HOW did he live SuperBowlXLIX	1	True	False	False
3	1	2015-02-01 18:23:00	1	347.0	4.491231	7.604738	190105015	Akron, OH	0	Loctite glue ftw I belong in that commercial S...	0	False	True	False
4	1	2015-02-01 18:23:00	1	54.0	4.365862	2.158416	468055147		0	Seahawks SuperBowl KatyPerry o	0	False	True	False

- **Merge data by window size**

We would like to build a time-series connection between the number of tweets and data features. To achieve this, we chose to use different time windows to analyze the data. Specifically, we selected time windows of **1 minute, 5 minutes, and 10 minutes**.

By using these different time windows, we were able to gain insights into how the number of tweets changed over time and how this was related to various data features. For example, we could analyze how the number of tweets varied over time based on factors such as the number of retweets, the user activity score, or the user location. By selecting different time windows, we could identify patterns and trends that might not be apparent when analyzing the data as a whole.

By merging the data, we got the train feature shape of (25,9) for 10 minutes windows, (50,9) for 5 minutes windows, and (250, 9) for 1 minutes windows. (The bold ones in the following table are not included in the number of features)

range_start	range_end	num_tweet	num_retweet	num_followers	ranking_score	user_activity	user_id	user_location	user_mentions	num_positive	num_neutral	num_negative	text	polarity	unique_user_id
2015-02-01 18:23:00	2015-02-01 18:33:00	20036	42248	204685593.0	92221.22883	300091.205496	[360742914, 26935300, 38567940, 112623622, 156...	816/770/803The Hub of the UniverseChicago, ILA...	6043	5499	12025	2512	NaN	0	17538
2015-02-01 18:33:00	2015-02-01 18:43:00	15873	57399	141105759.0	72237.96470	219360.108740	[2868412421, 38174728, 156827660, 2323841040, ...	ATL/MEMSAP EngineerTallahassee, FloridaUSAATLf...	5369	4704	9567	1602	NaN	0	14111

Features	Function
"range_start"	**The start of tweet's timestamp**
"range_end"	**The end of tweet's timestamp**
"num_tweet"	The number of times the tweet has been tweeted

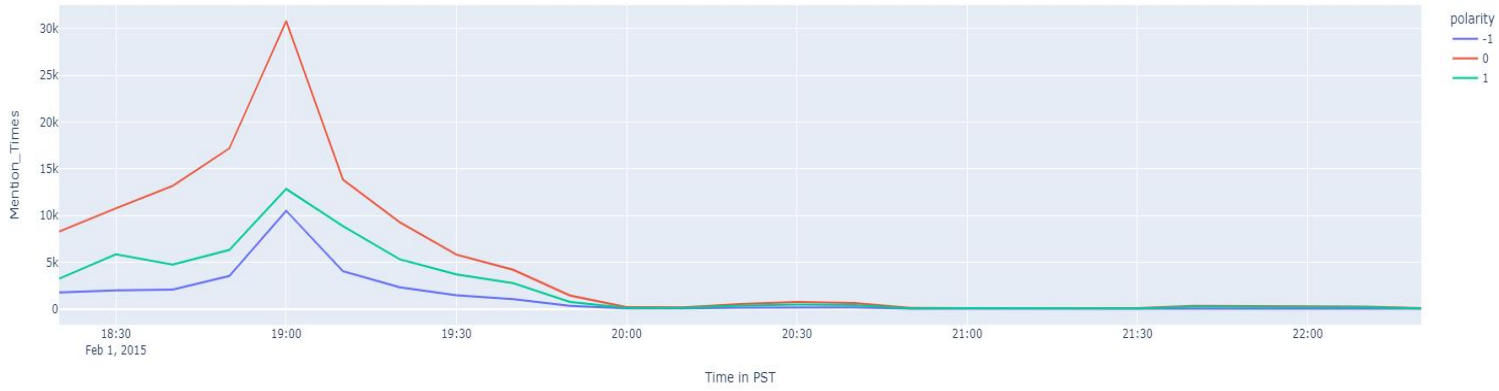
"num_retweet"	The number of times the tweet has been retweeted
"num_followers"	The number of followers of the user who created the tweet
"ranking_score"	A score measuring the influence of the tweet
"user_activity"	The number of tweets posted by the user per day
"user_id"	**The user ID of the user who created the tweet**
"user_location"	**The location of the user who created the tweet (if available)**
"user_mentions"	The number of other users mentioned in the tweet
"positive"	Number of positive sentiment
"neutral"	Number of neural sentiment
"negative"	Number of negative sentiment

- **Find the polarity with the mention of "superbowl"**

	Time in PST	polarity	Mention_Times
0	2015-02-01 18:20:00	-1	1767
1	2015-02-01 18:20:00	0	8281
2	2015-02-01 18:20:00	1	3236
3	2015-02-01 18:30:00	-1	1962
4	2015-02-01 18:30:00	0	10765
...	...	...	...
70	2015-02-01 22:10:00	0	227
71	2015-02-01 22:10:00	1	175
72	2015-02-01 22:20:00	-1	15
73	2015-02-01 22:20:00	0	83
74	2015-02-01 22:20:00	1	38

75 rows × 3 columns

To conduct this analysis, we used a natural language processing (NLP) tool to extract the polarity of tweets related to the Superbowl. The polarity score ranges from -1 to 1, with negative scores indicating negative sentiment, positive scores indicating positive sentiment, and 0 indicating neutral sentiment. We collected tweets that contained the term "Superbowl" and analyzed their polarity trends. The polarity trend analysis of Superbowl tweets revealed interesting insights into the mood of the authors. The results showed that the polarity scores were mostly positive, indicating a generally positive sentiment towards the event. However, there were some negative tweets, suggesting that not everyone had a positive experience or opinion of the Superbowl.

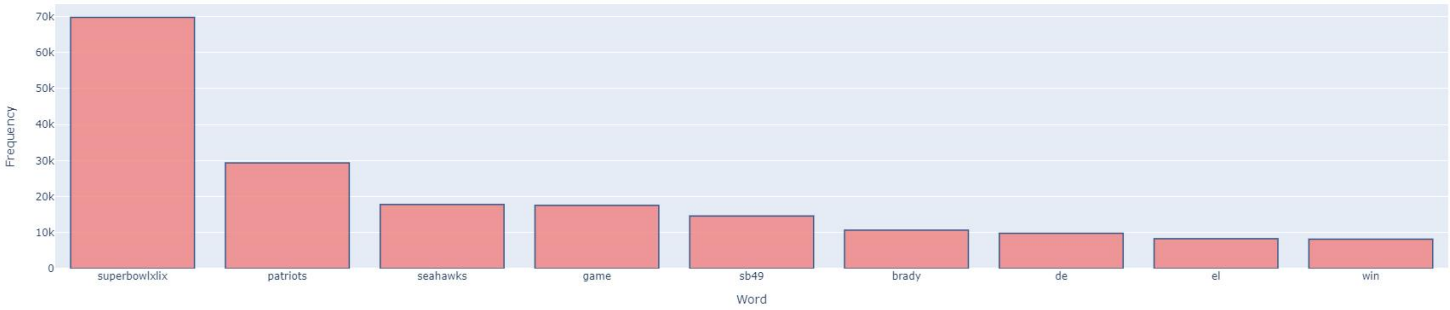


- **Find the Top frequency words used in the Tweets**

we first filtered the dataset to remove any mentions of player names, as this could potentially bias our analysis.

Next, we used the FreqDist function from the Natural Language Toolkit (NLTK) to create a frequency distribution of the filtered words. This allowed us to identify the words that were used most frequently in the dataset.

By analyzing the frequency distribution, we were able to identify the Top frequency words used in the tweets. These words could provide insights into the Topics that were most commonly discussed on Twitter during the period of interest.



Ⅱ Regression Model on Tweet's features

In this part, our goal was to analyze a large dataset of Twitter posts and use regression analysis to gain insights into the relationships between different data features and the number of tweets.

To achieve this, we first generated x and y variables from our dataset. However, because we had 25 lines in our dataset, we needed to drop the last item of x and the first item of y. This ensured that our dataset was consistent and allowed us to perform accurate regression analysis.

Next, we analyzed the significance of each feature in our dataset using the t-test and p-value by OLS Regression. This allowed us to identify which features were most strongly correlated with the number of tweets and to gain insights into how these features impacted Twitter activity during the period of interest.

```
    def scatter_plot(features, y_pred, pvalues, feature_names):
        # Obtain the indices that would sort the p-values in ascending order
        ranked_index = np.argsort(pvalues)
        print(ranked_index)
        for i in range(9):
            plt.figure(figsize = (8,5))
            # Create a scatter plot of the ith feature against the predicted values
            plt.scatter(features.iloc[:,ranked_index[i]], y_pred, alpha=0.5)
            plt.xlabel(feature_names[ranked_index[i]])
            plt.ylabel("Number of tweets next 10 minutes")
            plt.grid(True)
            plt.show()
        print('-' * 80)

    # Fit a linear regression model to the data and obtain the predicted values and p-values
    lr_fit = sm.OLS(superbowl_y,superbowl_x).fit()
    y_pred = lr_fit.predict()
    pvalues = lr_fit.pvalues
    print('MSE: ', metrics.mean_squared_error(superbowl_y, y_pred))
    print(lr_fit.summary())
    scatter_plot(superbowl_x, y_pred, pvalues, feature_names)
    print('\n')
```

we sorted the features from **most strongly** correlated to the least strongly correlated. Our results showed that the most strongly correlated features were **'num_retweet', 'num_followers',**

'ranking_score', 'user_activity', 'user_mentions', 'num_positive', 'num_neutral', 'num_negative', and 'unique_user_id'.

The 'num_retweet' feature had the strongest correlation with the number of tweets, followed closely by 'num_followers' and 'ranking_score'. These features suggest that tweets that are retweeted frequently, by users with a large number of followers or a high ranking score, are more likely to generate higher levels of Twitter activity.

And we got our MSE for 755168.0833725476.

This is the OLS Regression Results

OLS Regression Results

Dep. Variable:	y	R-squared (uncentered):	0.997
Model:	OLS	Adj. R-squared (uncentered):	0.994
Method:	Least Squares	F-statistic:	481.5
Date:	Mon, 13 Mar 2023	Prob (F-statistic):	8.26e-17
Time:	21:23:42	Log-Likelihood:	-196.47
No. Observations:	24	AIC:	410.9
Df Residuals:	15	BIC:	421.5
Df Model:	9		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
num_retweet	0.2927	0.077	3.811	0.002	0.129	0.456
num_followers	-9.607e-05	2.27e-05	-4.241	0.001	-0.000	-4.78e-05
ranking_score	-1.4769	1.116	-1.323	0.206	-3.857	0.903
user_activity	-0.0340	0.023	-1.456	0.166	-0.084	0.016
user_mentions	5.8656	1.054	5.565	0.000	3.619	8.112
num_positive	-2.9337	8.243	-0.356	0.727	-20.504	14.637
num_neutral	14.7212	7.175	2.052	0.058	-0.573	30.015
num_negative	8.4435	6.140	1.375	0.189	-4.643	21.530
unique_user_id	-2.6738	2.862	-0.934	0.365	-8.775	3.427

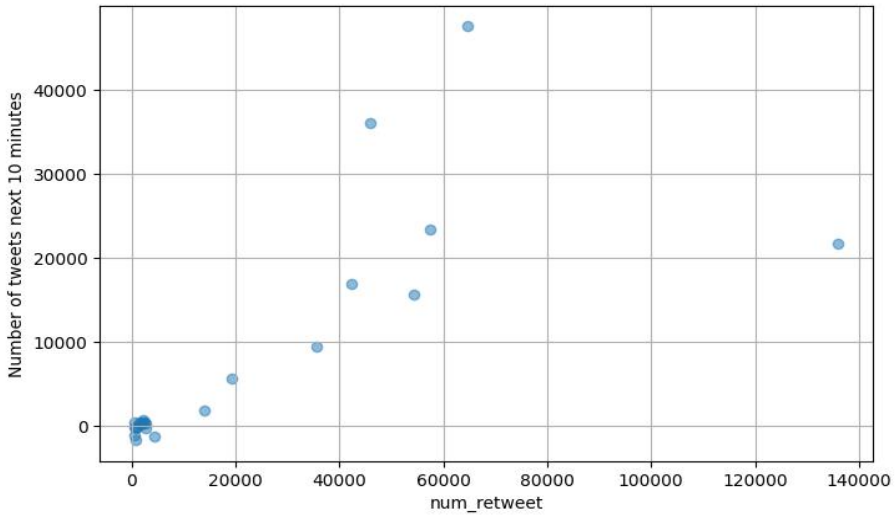
Omnibus:	0.116	Durbin-Watson:	1.540
Prob(Omnibus):	0.943	Jarque-Bera (JB):	0.330
Skew:	0.080	Prob(JB):	0.848
Kurtosis:	2.448	Cond. No.	6.53e+06

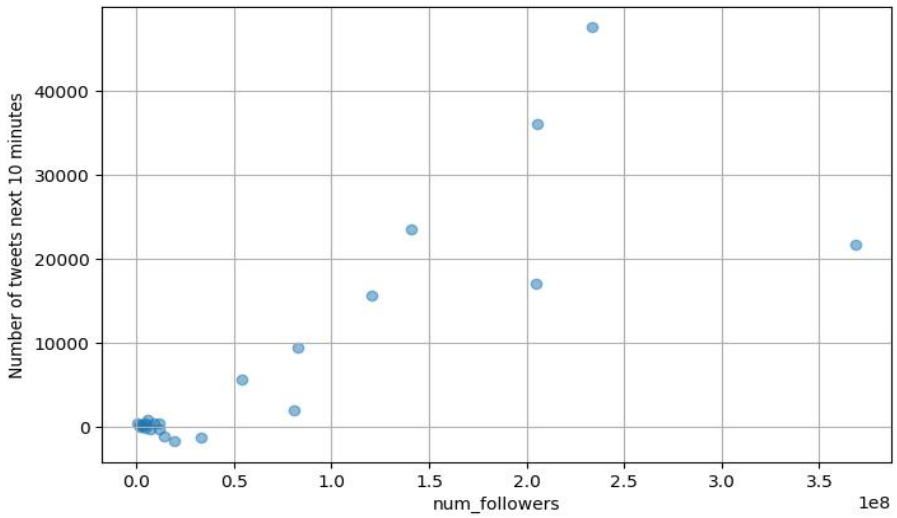
Looking at the results, we can see that the model has a high **R-squared value of 0.997**, indicating that the model explains a significant proportion of the variance in the number of tweets based on the selected features.

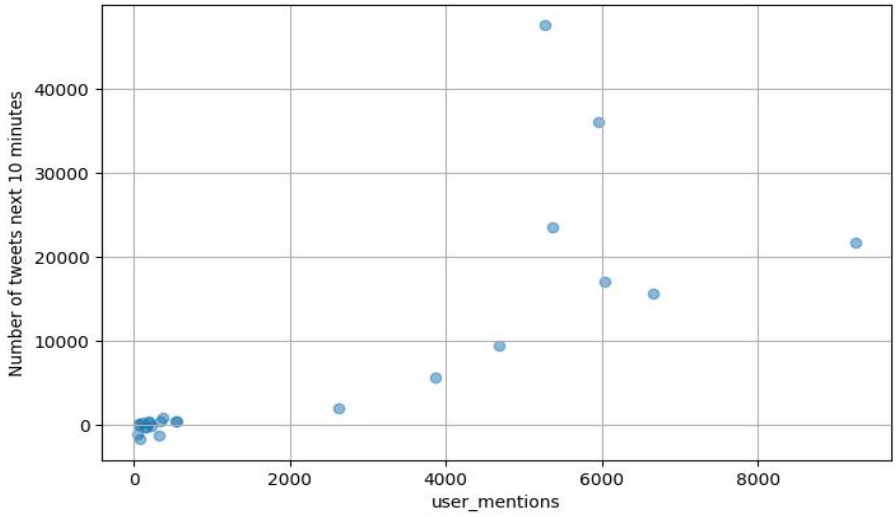
The predictor variables with **p-values less than 0.05** are considered to be statistically significant. In this case, we can see that **'num_retweet', 'num_followers', and 'user_mentions'** have p-values less than 0.05, indicating that they have a statistically significant relationship with

the number of tweets.

Scatter plots are a useful tool for visualizing the relationships between two variables and can help us identify patterns or trends in our data. We used scatter plots to visualize the relationships between different features of Twitter data and the number of tweets in the next 10 minutes. By plotting the **number of retweets, the number of followers, and the number of user mentions** against the number of tweets in the next 10 minutes, we were able to visually assess the relationships between these variables. We found that there was a positive correlation between each of these variables and the number of tweets, suggesting that they are all important predictors of Twitter activity during the period of interest.







- **Data Standardization**

After Data Standardization, we used the **Mutual information regression** and **F-regression** methods to identify the Top n most important features in our datasets. Mutual information regression is a method that measures the dependence between two variables, It calculates the mutual information between each feature and the target variable and selects the features with the highest mutual information values. F-regression, measures the linear relationship between two variables. It calculates the F-score between each feature and the target variable and selects the features with the highest F-scores.

```
def select_Topn_important_features(X, Y, n):
    Mutual_ = mutual_info_regression(X, Y)
    F_ = f_regression(X, Y)

    # Select the Top n features based on their mutual information and F-test scores
    Topn_M = np.argsort(Mutual_)[::-1][:n]
    Topn_F = np.argsort(F_[0])[::-1][:n]

    # Sort all the features based on their mutual information and F-test scores
    all_m = np.argsort(Mutual_)[::-1]
    all_f = np.argsort(F_[0])[::-1]

    # Extract the Top n features and all features based on their mutual information scores
    X_Topn_M = X.iloc[:, Topn_M]
    X_Topn_F = X.iloc[:, Topn_F]

```

```
    # Extract the Top n features and all features based on their F-test scores
    all_m_ = X.iloc[:, all_m]
    all_f_ = X.iloc[:, all_f]

    return X_Topn_M, X_Topn_F, all_m_, all_f_
```

The results from Mutual Info Regression is **'num_followers', 'num_retweet', 'num_negative', 'user_mentions','user_activity', 'num_positive', 'num_neutral', 'ranking_score', 'unique_user_id'.**

The results from F-Regression is **'num_retweet', 'num_positive', 'user_mentions', 'num_negative',' 'num_followers', 'unique_user_id', 'ranking_score', 'user_activity',' 'num_neutral'.**

Based on the results from the Mutual Info Regression and F-Regression methods, we can see that there is some overlap in the most important features, but there are also some differences. The Top features identified by both methods are **'num_followers', 'num_retweet', 'user_mentions', 'num_negative', 'unique_user_id', and 'ranking_score',** indicating that these features have a strong relationship with the number of tweets.

And the results also **align with** the p-values result.

- **Linear Regression, Ridge regularization, and Lasso regularization**

we used cross-validation and three different regression models - LinearRegression, Ridge, and Lasso - to determine how many features we need in our predictive model.

We performed cross-validation with different numbers of features, ranging from 1 to 9, and evaluated the performance of the LinearRegression, Ridge, and Lasso models. By comparing the performance metrics (mean squared error, mean absolute error, and R-squared) for each model and number of features, we were able to determine the optimal number of features to include in our predictive model.

linear regression,

linear regression and with mutual_info_regression,

linear regression and with f_regression,

Ridge regression,

Ridge regression and with mutual_info_regression,

Ridge regression and with f_regression,

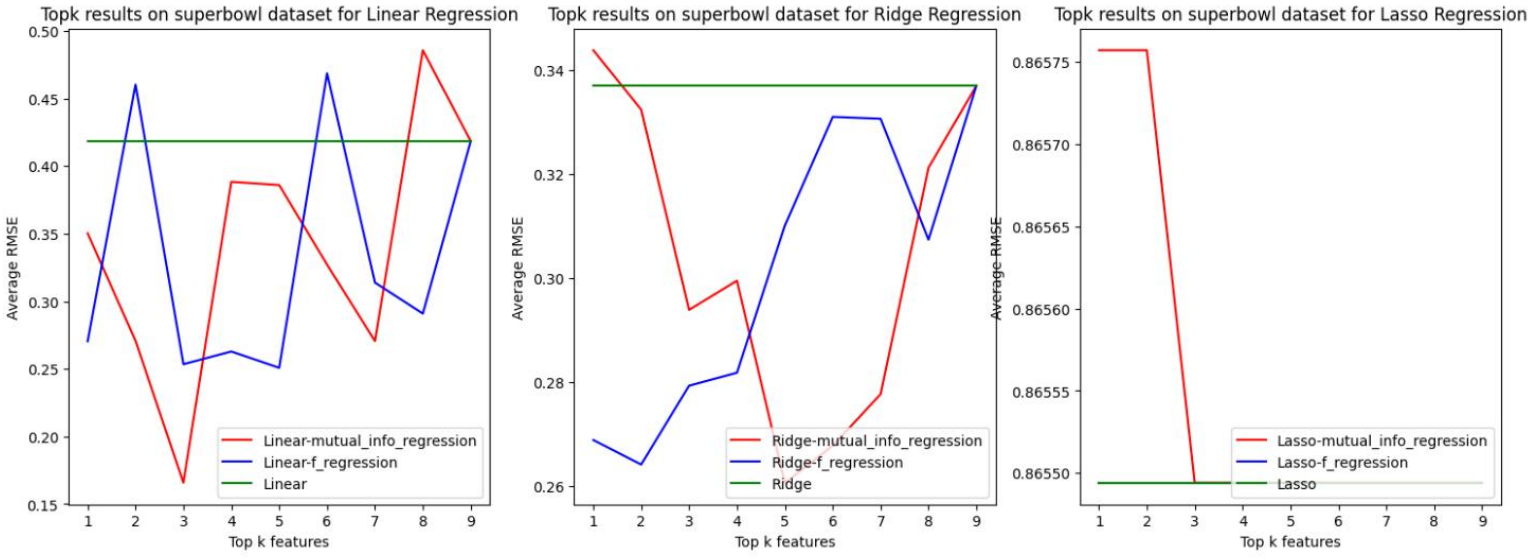
Lassso regression,

Lasso regression and with mutual_info_regression,

Lasso regression and with f_regression.

Different regulation schemes' RMSE scores with diamond dataset:

RMSE score	Top 1	Top 2	Top 3	Top 4	Top 5	Top 6	Top 7	Top 8	Top9
Linear					-0.4186				
Linear +mutual	-0.3504	-0.2709	-0.1658	-0.3885	-0.3861	-0.3272	-0.2706	-0.4858	-0.4186
Linear +f	-0.2704	-0.4604	-0.2535	-0.2609	-0.2508	-0.4688	-0.3139	-0.2909	-0.4186
Ridge					-0.3371				
Ridge +mutual	-0.3439	-0.3325	-0.2939	-0.2995	-0.2606	-0.2677	-0.2777	-0.3213	-0.3371
Ridge +f	-0.2688	-0.2641	-0.2793	-0.2818	-0.3101	-0.3310	-0.3307	-0.3074	-0.3371
Lasso					-0.8655				
Lasso +mutual	-0.8658	-0.8658	-0.8655	-0.8655	-0.8655	-0.8655	-0.8655	-0.8655	-0.8655
Lasso +f	-0.8655	-0.8655	-0.8655	-0.8655	-0.8655	-0.8655	-0.8655	-0.8655	-0.8655



The results show the linear regression model without any feature selection has an RMSE of

-0.4186. When using the Mutual Info Regression method to select the Top 1 feature, the RMSE improved to -0.3504, and with the F-Regression method, the RMSE improved even further to -0.2704.

For the Ridge regression model, the RMSE without feature selection is -0.3371. When using Mutual Info Regression to select the Top 2 features, the RMSE slightly increased to -0.3439, but with F-Regression, it improved to -0.2688.

For the Lasso regression model, the RMSE without feature selection is the highest among all models with an RMSE of -0.8655. Using either Mutual Info Regression or F-Regression to select the Top features did not improve the RMSE.

Overall, the F-Regression method appears to be more effective in selecting the Top features for regression models, with the lowest RMSE scores achieved when using this method.

- **Cross Validate to find the optimal answer**

```
pipe_ = Pipeline([
    ('kbest', SelectKBest()),
    ('model', "passthrough")
], memory = memory)

param_grid = [{
        'kbest__score_func': (mutual_info_regression, f_regression),
        'kbest__k': (1, 2, 3, 4, 5, 6, 7, 8, 9),
        'model': [Ridge(), Lasso()],
        'model__alpha': [10.0**x for x in np.arange(-3,4)]
    }
]
```

We got the best parameters for superbowl is score_function = f_regression, and select Top 7 features, then use the Ridge regression with alpha = 0.001, which Best score is -0.18191415917366846.

Then we construct the superbowl Top7 features for further experiments.

```
superbowl_Top7 = SelectKBest(score_func = f_regression, k = 7).fit_transform(superbowl_merge_data_standard_x, superbowl_merge_data_standard_y)
```

- **Polynomial regression with Ridge regularization**

Polynomial regression with Ridge regularization is a type of regression analysis used to model the relationship between a dependent variable and one or more independent variables, where the relationship is modeled as an nth-degree polynomial. Ridge regularization is used to prevent overfitting by adding a penalty term to the loss function, which shrinks the regression coefficients towards zero.

```
    poly_pipe_superbowl_ = Pipeline([
        ('poly_transform', PolynomialFeatures()),
        ('model', Ridge(alpha=0.001))
    ],memory=memory)

    poly_param_= {
        'poly_transform__degree': np.arange(1,10,1)
    }
```

We got the best parameters for superbowl is poly_transform__degree = 1 , which Best score is -0.3822844598287635.

And the Top 7 Salient features we got from the polynomial regression is 'num_negative', 'user_mentions','num_followers','user_activity','ranking_score','unique_user_id,'num_neutral'. In results, **the 'user_mentions','num_followers' is align** with the previous experiments.

- **Multi-layer perceptron (MLP) regression**

In MLP regression, the input variables are fed into the network, and the network learns to predict the output variable based on the patterns in the data. The weights and biases of the network are adjusted iteratively during the training process using an optimization algorithm to minimize the error between the predicted output and the actual output.

```
    mlpr = Pipeline([
        ('model', MLPRegressor()),

    ], memory=memory)

    param_list = {
        "model__hidden_layer_sizes": [(30, 40), (30, 60), (40, 40), (40, 60), (6
0, 60)],
```

```
        "model__activation": ["identity", "logistic", "tanh", "relu"],
        "model__solver": ["lbfgs", "sgd", "adam"],
    }
```

We got the Best parameters for Neural Network of Superbowl is 'model__activation': 'identity', 'model__hidden_layer_sizes': (60, 60), 'model__solver': 'sgd'

Best score for Neural Network of Superbowl: -0.3281072454469807, which RMSE is 0.5728064642154282

- **RandomForest Regression**

RandomForest Regression ensemble learning method that combines multiple decision trees to create a more robust and accurate model. In this algorithm, a large number of decision trees are created, each of which provides a prediction for the target variable. The final prediction is then made by aggregating the predictions of all the individual trees.

```
pipeline_forest = Pipeline([
    ('model', RandomForestRegressor())
],memory=memory)

param_grid_forest = {
    'model__max_features': np.arange(1, 5, 1),
    'model__n_estimators': np.arange(10, 40, 10),
    'model__max_depth': np.arange(1, 5, 1)
}
```

We got Best parameters for RandomForestRegressor for superbowl is 'model__max_depth': 2, 'model__max_features': 2, 'model__n_estimators': 20

Best score for RandomForestRegressor for superbowl: -0.23065258627850366

With OOB score: 0.5292, R^2 score: 0.9046. The R^2 score of 0.9046 is quite high, indicating that the model is a good fit for the data. The negative MSE value (-0.2306) also indicates that the model is performing better than the baseline model. The OOB score of 0.5292 is also a good indicator of model performance, as it shows how well the model can generalize to new, unseen data. Overall, it seems like the RandomForestRegressor is a good choice for this dataset.

- **LightGBM gradient boosting Regression**

```
lbg_pip_ = Pipeline([
    ('model', lgb.LGBMRegressor())
],memory = memory)

param_lgb_ = {
    'model__num_leaves': [7, 14, 21, 28, 31, 50],
    'model__learning_rate': [0.1, 0.03, 0.003],
    'model__max_depth': [-1, 3, 5],
    'model__n_estimators': [50, 100, 200, 500],
}
```

We got Best parameters for LightGBM of superbowls with ('model__learning_rate', 0.03), ('model__max_depth', -1), ('model__n_estimators', 500), ('model__num_leaves', 31)

Best score for LightGBM of superbowls: -0.8657568961600333

- **Conclusion**

For all, we deem the score_function = f_regression, and select Top 7 features, then use the Ridge regression with alpha = 0.001 is the best model. And the RandomForestRegressor for superbowl is 'model__max_depth': 2, 'model__max_features': 2, 'model__n_estimators': 20 is the second best model

- **Try on window size =1 min**

  We have (250,14) train_data.shape

  - **OLS Regression**

    we got our MSE for 156224.0969515157

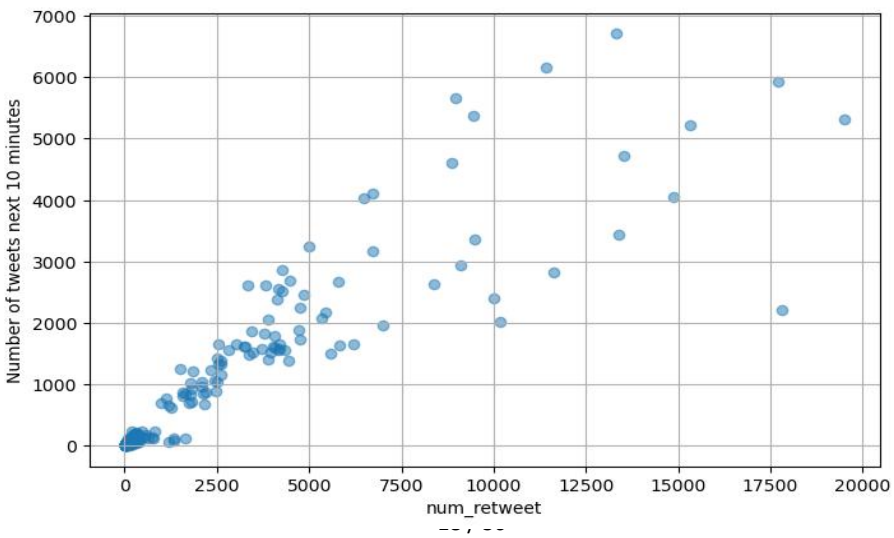
    P-value is ranked by **'num_retweet', 'num_followers', 'ranking_score', 'user_activity', 'user_mentions', 'num_positive', 'num_neutral', 'num_negative', 'unique_user_id'.**

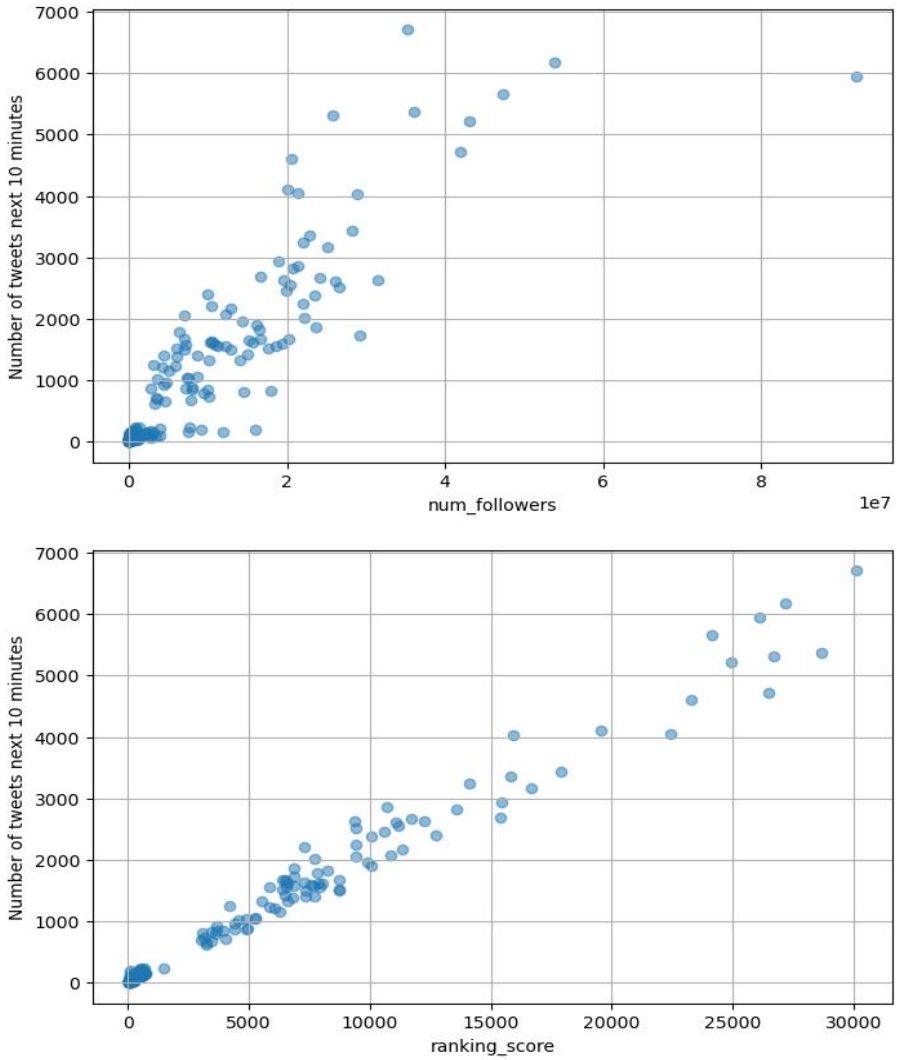
OLS Regression Results

Dep. Variable:	y	R-squared (uncentered):	0.940
Model:	OLS	Adj. R-squared (uncentered):	0.938
Method:	Least Squares	F-statistic:	402.4
Date:	Tue, 14 Mar 2023	Prob (F-statistic):	8.57e-136
Time:	15:00:01	Log-Likelihood:	-1775.6
No. Observations:	240	AIC:	3569.
Df Residuals:	231	BIC:	3601.
Df Model:	9		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
num_retweet	0.0377	0.020	1.842	0.067	-0.003	0.078
num_followers	9.285e-06	6.16e-06	1.507	0.133	-2.85e-06	2.14e-05
ranking_score	1.4483	0.428	3.386	0.001	0.605	2.291
user_activity	0.0018	0.005	0.346	0.730	-0.008	0.012
user_mentions	1.0776	0.402	2.684	0.008	0.286	1.869
num_positive	-1.5432	3.339	-0.462	0.644	-8.121	5.035
num_neutral	-0.9289	3.383	-0.275	0.784	-7.595	5.738
num_negative	0.3506	3.318	0.106	0.916	-6.188	6.889
unique_user_id	-5.2054	2.542	-2.048	0.042	-10.214	-0.197

Omnibus:	170.365	Durbin-Watson:	2.233
Prob(Omnibus):	0.000	Jarque-Bera (JB):	4411.142
Skew:	2.349	Prob(JB):	0.00
Kurtosis:	23.471	Cond. No.	3.10e+06





■ Mutual-info-regression and F-regression

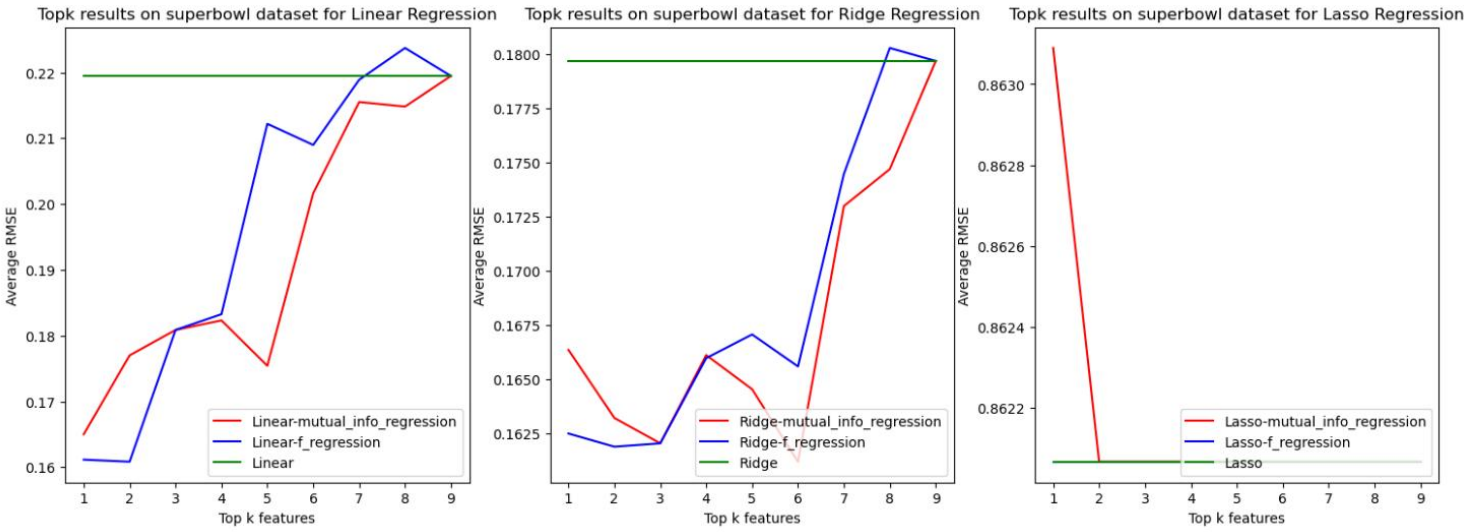
Results by Mutual-info-regression is 'num_followers', 'user_activity', 'num_retweet', 'num_negative', 'user_mentions', 'num_positive', 'num_neutral', 'ranking_score', 'unique_user_id'

Results by F-regression is 'user_mentions', 'num_followers', 'num_retweet', 'user_activity', 'num_positive', 'num_negative', 'unique_user_id', 'num_neutral', 'ranking_score'.

Model	Feature Selection Method	RMSE Score	Features
Linear Regression	mutual_info_regression	-0.1650	Top1
Linear Regression	f_regression	-0.1612	Top1
Ridge Regression	mutual_info_regression	-0.1663	Top1
Ridge Regression	f_regression	-0.1625	Top1

Model	Feature Selection Method	RMSE Score	Features
Lasso Regression	mutual_info_regression	-0.8631	Top1
Lasso Regression	f_regression	-0.8621	Top1
Linear Regression	mutual_info_regression	-0.1770	Top2
Linear Regression	f_regression	-0.1608	Top2
Ridge Regression	mutual_info_regression	-0.1632	Top2
Ridge Regression	f_regression	-0.1619	Top2
Lasso Regression	mutual_info_regression	-0.8621	Top2
Lasso Regression	f_regression	-0.8621	Top2
Linear Regression	mutual_info_regression	-0.1809	Top3
Linear Regression	f_regression	-0.1809	Top3
Ridge Regression	mutual_info_regression	-0.1620	Top3
Ridge Regression	f_regression	-0.1620	Top3
Lasso Regression	mutual_info_regression	-0.8621	Top3
Lasso Regression	f_regression	-0.8621	Top3
Linear Regression	mutual_info_regression	-0.1823	Top4
Linear Regression	f_regression	-0.1833	Top4
Ridge Regression	mutual_info_regression	-0.1661	Top4
Ridge Regression	f_regression	-0.1660	Top4
Lasso Regression	mutual_info_regression	-0.8621	Top4
Lasso Regression	f_regression	-0.8621	Top4
Linear Regression	mutual_info_regression	-0.1754	Top5
Linear Regression	f_regression	-0.2122	Top5
Ridge Regression	mutual_info_regression	-0.1645	Top5
Ridge Regression	f_regression	-0.1671	Top5
Lasso Regression	mutual_info_regression	-0.8621	Top5
Lasso Regression	f_regression	-0.8621	Top5
Linear Regression	mutual_info_regression	-0.2016	Top6
Linear Regression	f_regression	-0.2090	Top6
Ridge Regression	**mutual_info_regression**	**-0.1612**	**Top6**
Ridge Regression	f_regression	-0.1656	Top6
Lasso Regression	mutual_info_regression	-0.8621	Top6
Lasso Regression	f_regression	-0.8621	Top6
Linear Regression	Mutual Info Regression	-0.2155	Top 7
Linear Regression	F Regression	-0.2189	Top 7
Ridge Regression	Mutual Info Regression	-0.1730	Top 7

Model	Feature Selection Method	RMSE Score	Features
Ridge Regression	F Regression	-0.1745	Top 7
Lasso Regression	Mutual Info Regression	-0.8621	Top 7
Lasso Regression	F Regression	-0.8621	Top 7
Linear Regression	Mutual Info Regression	-0.2148	Top 8
Linear Regression	F Regression	-0.2237	Top 8
Ridge Regression	Mutual Info Regression	-0.1747	Top 8
Ridge Regression	F Regression	-0.1803	Top 8
Lasso Regression	Mutual Info Regression	-0.8621	Top 8
Lasso Regression	F Regression	-0.8621	Top 8
Linear Regression	Mutual Info Regression	-0.2195	Top 9
Linear Regression	F Regression	-0.2195	Top 9
Ridge Regression	Mutual Info Regression	-0.1797	Top 9
Ridge Regression	F Regression	-0.1797	Top 9



The Best results is score_function = mutual_info_regression, and select Top 6 features, then use the Ridge regression with alpha = 0.001 is the best model.

- **RandomForest GridSearch**

```
pipe_rf = Pipeline([
    ('standardize', StandardScaler()),
    ('model', RandomForestRegressor(random_state=42))
])

param_grid = {
    'model__max_depth': [10, 30, 50, 70, 100, 200],
```

```
        'model__max_features': ['auto', 'sqrt'],
        'model__min_samples_leaf': [1, 2, 3],
        'model__min_samples_split': [2, 5, 10],
        'model__n_estimators': [200, 400]
    }
```

The Top5 results are stated:

	mean_test_score	param_model__max_depth	param_model__max_features	param_model__min_samples_leaf	param_model__min_samples_split	param_model__n_estimators
0	-161354.634897	10	auto	1	2	400
1	-162792.741378	10	auto	1	5	400
2	-163112.769889	30	auto	1	2	400
3	-163112.769889	200	auto	1	2	400
4	-163112.769889	70	auto	1	2	400

■ GradientBoosting GridSearch

```
pipe_gb = Pipeline([
    ('standardize', StandardScaler()),
    ('model', GradientBoostingRegressor(random_state=42))
])

param_grid = {
    'model__max_depth': [10, 30, 50, 70, 100, 200],
    'model__max_features': ['auto', 'sqrt'],
    'model__min_samples_leaf': [1, 2, 3],
    'model__min_samples_split': [2, 5, 10],
    'model__n_estimators': [200, 400]
}
```

The Top5 results are stated:

	mean_test_score	param_model__max_depth	param_model__max_features	param_model__min_samples_leaf	param_model__min_samples_split	param_model__n_estimators
0	-137558.444653	50	sqrt	1	5	400
1	-137558.444653	100	sqrt	1	5	400
2	-137558.444653	70	sqrt	1	5	400
3	-137558.444653	200	sqrt	1	5	400
4	-137558.459325	30	sqrt	1	5	400

■ NeuralNetwork GridSearch

```
pipe_nn_noscale = Pipeline([
    ('model', MLPRegressor(random_state=42, max_iter=2000))
])

param_grid = {
    'model__hidden_layer_sizes': [(x,y) for x in np.arange(1, 51) for y in n
p.arange(1, 51)]
```

```
7.      }
```

The Top5 results are stated:

	mean_test_score	param_model__hidden_layer_sizes
0	-238146.783734	(5, 8)
1	-245119.714812	(25, 1)
2	-249601.147172	(9, 3)
3	-290023.313343	(5, 6)
4	-304567.791884	(8, 8)

- **Try on window size = 5 min**

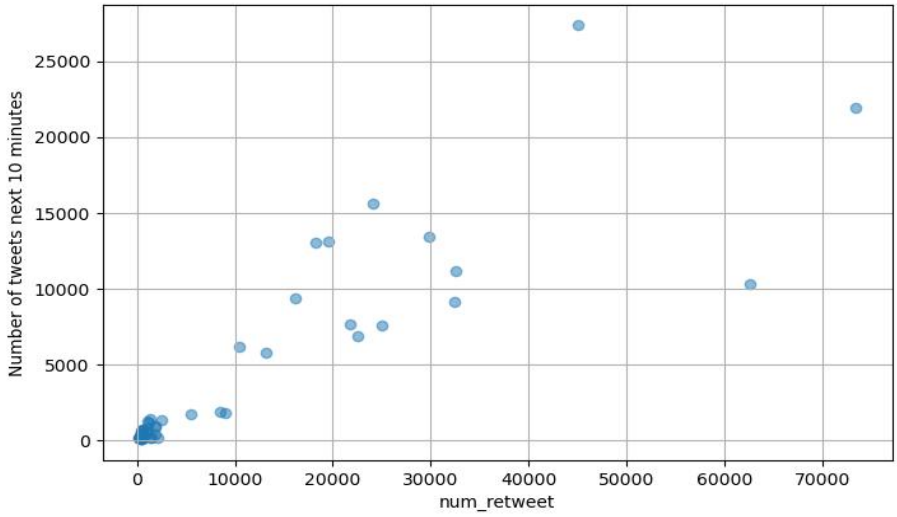
  We have (50,14) train_data.shape

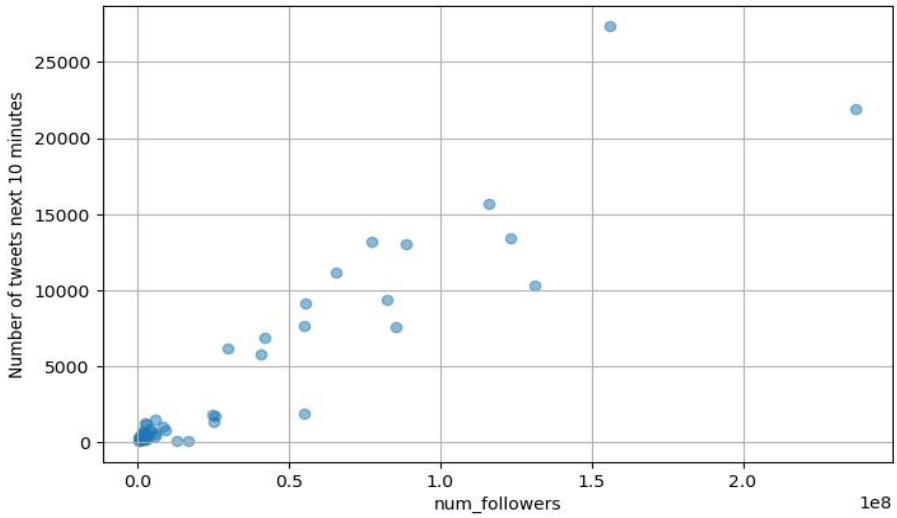
  - **OLS Regression**

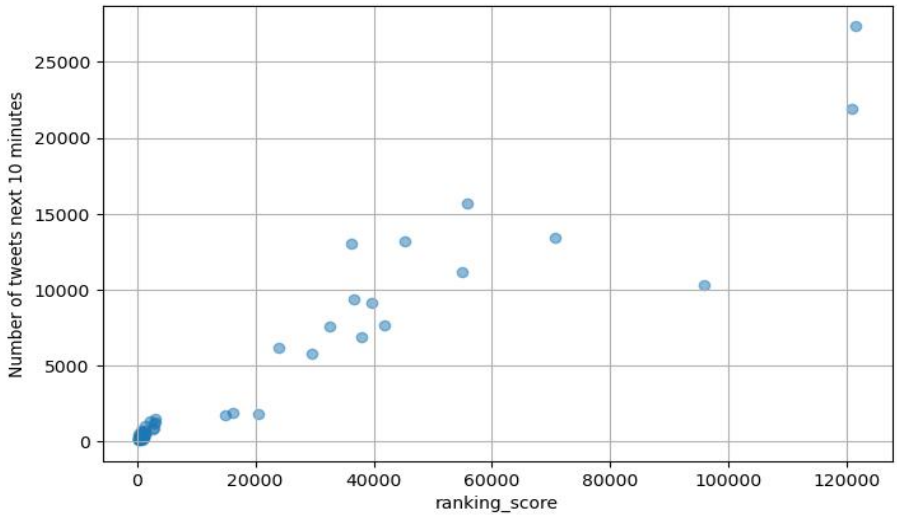
    we got our MSE for 6352334.382528166

    P-value is ranked by 'num_retweet', 'num_followers', 'ranking_score', 'user_activity', 'user_mentions', 'num_positive', 'num_neutral', 'num_negative', 'unique_user_id', **which is the same with the window_size = 1min**

```
                            OLS Regression Results
==============================================================================================
Dep. Variable:                          y   R-squared (uncentered):                   0.897
Model:                                OLS   Adj. R-squared (uncentered):              0.873
Method:                     Least Squares   F-statistic:                              37.75
Date:                    Tue, 14 Mar 2023   Prob (F-statistic):                    1.61e-16
Time:                            15:17:58   Log-Likelihood:                         -444.05
No. Observations:                      48   AIC:                                      906.1
Df Residuals:                          39   BIC:                                      922.9
Df Model:                               9
Covariance Type:                nonrobust
==============================================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
----------------------------------------------------------------------------------------------
num_retweet       -0.0505      0.160     -0.316      0.754      -0.374       0.273
num_followers  -1.174e-05   4.38e-05     -0.268      0.790      -0.000    7.69e-05
ranking_score      5.6989      2.266      2.515      0.016       1.116      10.282
user_activity     -0.0155      0.037     -0.423      0.675      -0.090       0.059
user_mentions      8.3991      2.790      3.011      0.005       2.757      14.042
num_positive     -35.5276     15.380     -2.310      0.026     -66.638      -4.418
num_neutral      -29.0873     13.798     -2.108      0.042     -56.997      -1.177
num_negative     -19.7659     12.493     -1.582      0.122     -45.034       5.503
unique_user_id     3.0421      7.477      0.407      0.686     -12.082      18.166
==============================================================================
Omnibus:                       59.300   Durbin-Watson:                   2.410
Prob(Omnibus):                  0.000   Jarque-Bera (JB):              555.454
Skew:                           2.935   Prob(JB):                    2.42e-121
Kurtosis:                      18.597   Cond. No.                     3.62e+06
==============================================================================
```


Number of tweets next 10 minutes
25000
20000
15000
10000
5000
0
0
10000
20000
30000
40000
50000
60000
70000
num_retweet


Number of tweets next 10 minutes
25000
20000
15000
10000
5000
0
0.0
0.5
1.0
1.5
2.0
num_followers
1e8


Number of tweets next 10 minutes
25000
20000
15000
10000
5000
0
0
20000
40000
60000
80000
100000
120000
ranking_score

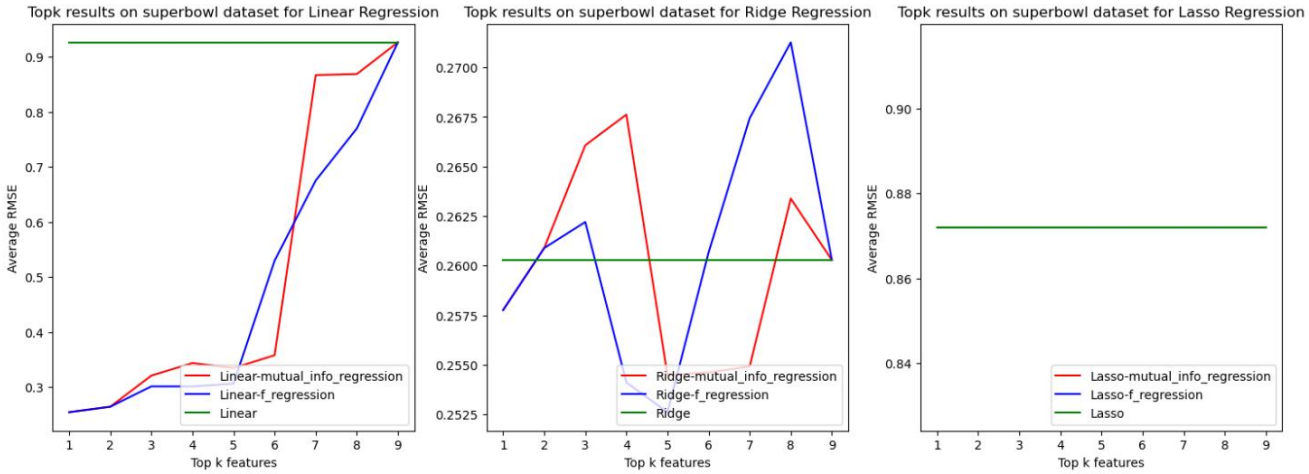
■ Mutual-info-regression and F-regression

Results by Mutual-info-regression is 'num_followers', 'num_retweet', 'num_negative', 'user_activity', 'user_mentions', 'unique_user_id', 'num_positive', 'ranking_score', 'num_neutral'

Results by F-regression is 'user_mentions', 'user_mentions', 'num_retweet', 'num_positive', 'num_negative', 'num_followers', 'user_activity', 'unique_user_id', 'ranking_score', 'num_neutral'

Model	Feature Selection Method	RMSE Score	Rank
Linear Regression	mutual_info_regression	-0.2540	top1
Linear Regression	f_regression	-0.2540	top1
Ridge Regression	mutual_info_regression	-0.2578	top1
Ridge Regression	f_regression	-0.2578	top1
Lasso Regression	mutual_info_regression	-0.8720	top1
Lasso Regression	f_regression	-0.8720	top1
Linear Regression	mutual_info_regression	-0.2640	top2
Linear Regression	f_regression	-0.2640	top2
Ridge Regression	mutual_info_regression	-0.2609	top2
Ridge Regression	f_regression	-0.2609	top2
Lasso Regression	mutual_info_regression	-0.8720	top2
Lasso Regression	f_regression	-0.8720	top2
Linear Regression	mutual_info_regression	-0.3205	top3
Linear Regression	f_regression	-0.3010	top3
Ridge Regression	mutual_info_regression	-0.2661	top3
Ridge Regression	f_regression	-0.2622	top3
Lasso Regression	mutual_info_regression	-0.8720	top3
Lasso Regression	f_regression	-0.8720	top3
Linear Regression	mutual_info_regression	-0.3434	top4
Linear Regression	f_regression	-0.3009	top4
Ridge Regression	mutual_info_regression	-0.2676	top4
Ridge Regression	f_regression	-0.2541	top4
Lasso Regression	mutual_info_regression	-0.8720	top4
Lasso Regression	f_regression	-0.8720	top4
Linear Regression	mutual_info_regression	-0.3343	top5
Linear Regression	f_regression	-0.3060	top5
Ridge Regression	mutual_info_regression	-0.2545	top5
Ridge Regression	f_regression	-0.2526	top5

Model	Feature Selection Method	RMSE Score	Rank
Lasso Regression	mutual_info_regression	-0.8720	top5
Lasso Regression	f_regression	-0.8720	top5
Linear Regression	mutual_info_regression	-0.3576	top6
Linear Regression	f_regression	-0.5294	top6
Ridge Regression	**mutual_info_regression**	**-0.2546**	**top6**
Ridge Regression	f_regression	-0.2607	top6
Lasso Regression	mutual_info_regression	-0.8720	top6
Lasso Regression	f_regression	-0.8720	top6
Linear Regression	mutual_info_regression	-0.8667	top7
Linear Regression	f_regression	-0.6752	top7
Ridge Regression	mutual_info_regression	-0.2549	top7
Ridge Regression	f_regression	-0.2674	top7
Lasso Regression	mutual_info_regression	-0.8720	top7
Lasso Regression	f_regression	-0.8720	top7
Linear Regression	mutual_info_regression	-0.8686	top8
Linear Regression	f_regression	-0.7697	top8
Ridge Regression	mutual_info_regression	-0.2634	top8
Ridge Regression	f_regression	-0.2713	top8
Lasso Regression	mutual_info_regression	-0.8720	top8
Lasso Regression	f_regression	-0.8720	top8
Linear Regression	mutual_info_regression	-0.9262	top9
Linear Regression	f_regression	-0.9262	top9
Ridge Regression	mutual_info_regression	-0.2603	top9
Ridge Regression	f_regression	-0.2603	top9
Lasso Regression	mutual_info_regression	-0.8720	top9
Lasso Regression	f_regression	-0.8720	top9



The Best results is score_function = mutual_info_regression, and select Top 6 features, then use the Ridge regression with alpha = 0.001 is the best model. **Which is align with the window = 1's result**

- **RandomForest GridSearch**

	mean_test_score	param_model__max_depth	param_model__max_features	param_model__min_samples_leaf	param_model__min_samples_split	param_model__n_estimators
0	-1.167336e+07	10	sqrt	1	2	200
1	-1.169774e+07	70	sqrt	1	2	200
2	-1.169774e+07	30	sqrt	1	2	200
3	-1.169774e+07	200	sqrt	1	2	200
4	-1.169774e+07	50	sqrt	1	2	200

- **GradientBoosting GridSearch**

- **NeuralNetwork GridSearch**

	mean_test_score	param_model__max_depth	param_model__max_features	param_model__min_samples_leaf	param_model__min_samples_split	param_model__n_estimators
0	-6.611812e+06	10	auto	3	2	400
1	-6.611812e+06	10	auto	3	5	400
2	-6.655455e+06	50	auto	3	2	400
3	-6.655455e+06	50	auto	3	5	400
4	-6.655455e+06	70	auto	3	5	400

	mean_test_score	param_model__hidden_layer_sizes
0	-9.935143e+06	(6, 12)
1	-1.004485e+07	(39, 11)
2	-1.022622e+07	(5, 8)
3	-1.108928e+07	(33, 1)
4	-1.123810e+07	(33, 2)

Ⅲ Classification model on tweet's location prediction

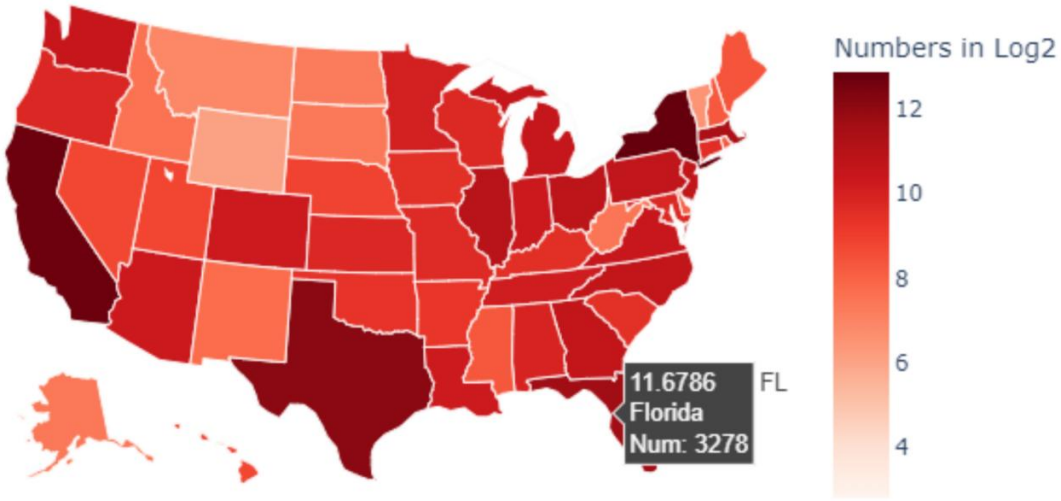
The purpose of this report is to present the findings of a project aimed at predicting the location of tweets based on a classification model. The dataset used for this project contained information on tweets and their associated user locations. The goal was to identify the majority areas of tweets and create a list of US states and their abbreviations. Furthermore, we filtered the dataset by iterating over each user location to reduce the dataset size and improve model accuracy.

W filtered the dataset by iterating over each user location. We filtered out tweets that did not contain a user location or had a location outside of the US. This process reduced the dataset size to 8,988 tweets.

State	Number	Full State Name
NY	7483	New York
CA	6862	California
TX	4560	Texas
FL	3278	Florida
MA	2600	Massachusetts
IL	2019	Illinois
OH	1778	Ohio
PA	1653	Pennsylvania
GA	1627	Georgia
NJ	1612	New Jersey
NC	1584	North Carolina
MI	1460	Michigan
WA	1436	Washington
VA	1356	Virginia
DC	1335	District of Columbia
AZ	1285	Arizona
IN	1271	Indiana
LA	1251	Louisiana
CO	1234	Colorado
TN	1161	Tennessee
MN	1067	Minnesota
AL	972	Alabama
KS	883	Kansas
OR	873	Oregon
WI	863	Wisconsin
MD	852	Maryland
MO	807	Missouri
CT	755	Connecticut

KY	728	Kentucky
SC	710	South Carolina
IA	709	Iowa
OK	663	Oklahoma
AR	623	Arkansas
NE	472	Nebraska
UT	450	Utah
HI	442	Hawaii
NV	441	Nevada
AS	363	American Samoa
ME	340	Maine
RI	326	Rhode Island
MS	321	Mississippi
PR	295	Puerto Rico
DE	287	Delaware
NM	213	New Mexico
ID	188	Idaho
WV	176	West Virginia
AK	168	Alaska
SD	166	South Dakota
ND	153	North Dakota
MT	120	Montana
VT	91	Vermont
WY	68	Wyoming
GU	46	Guam
VI	40	Virgin Islands
FM	34	Federated States of Micronesia
MP	15	Northern Mariana Islands
MH	12	Marshall Islands
PW	7	Palau

To better visualize the geographical distribution of the states and territories listed in the table above, we can create a heatmap based on their location data. By creating a heatmap, we can quickly identify the states and territories with the highest and lowest scores, and visually analyze any geographic patterns that may emerge.



Now we are having a dict:

```
{'AL': 'Alabama', 'AK': 'Alaska', 'AS': 'American Samoa', 'AZ': 'Arizona', 'AR': 'A
rkansas', 'CA': 'California', 'CO': 'Colorado', 'CT': 'Connecticut', 'DE': 'Delawar
e', 'DC': 'District of Columbia', 'FM': 'Federated States of Micronesia', 'FL': 'Fl
orida', 'GA': 'Georgia', 'GU': 'Guam', 'HI': 'Hawaii', 'ID': 'Idaho', 'IL': 'Illino
is', 'IN': 'Indiana', 'IA': 'Iowa', 'KS': 'Kansas', 'KY': 'Kentucky', 'LA': 'Louisi
ana', 'ME': 'Maine', 'MH': 'Marshall Islands', 'MD': 'Maryland', 'MA': 'Massachuset
ts', 'MI': 'Michigan', 'MN': 'Minnesota', 'MS': 'Mississippi', 'MO': 'Missouri', 'M
T': 'Montana', 'NE': 'Nebraska', 'NV': 'Nevada', 'NH': 'New Hampshire', 'NJ': 'New
Jersey', 'NM': 'New Mexico', 'NY': 'New York', 'NC': 'North Carolina', 'ND': 'North
 Dakota', 'MP': 'Northern Mariana Islands', 'OH': 'Ohio', 'OK': 'Oklahoma', 'OR': '
Oregon', 'PW': 'Palau', 'PA': 'Pennsylvania', 'PR': 'Puerto Rico', 'RI': 'Rhode Isl
and', 'SC': 'South Carolina', 'SD': 'South Dakota', 'TN': 'Tennessee', 'TX': 'Texas
', 'UT': 'Utah', 'VT': 'Vermont', 'VI': 'Virgin Islands', 'VA': 'Virginia', 'WA': '
Washington', 'WV': 'West Virginia', 'WI': 'Wisconsin', 'WY': 'Wyoming'}
```

And maps the locations into numbers:

```
{'Alabama': 1, 'Alaska': 2, 'American Samoa': 3, 'Arizona': 4, 'Arkansas': 5, 'Cali
fornia': 6, 'Colorado': 7, 'Connecticut': 8, 'Delaware': 9, 'District of Columbia':
 10, 'Federated States of Micronesia': 11, 'Florida': 12, 'Georgia': 13, 'Guam': 14,
 'Hawaii': 15, 'Idaho': 16, 'Illinois': 17, 'Indiana': 18, 'Iowa': 19, 'Kansas': 20,
 'Kentucky': 21, 'Louisiana': 22, 'Maine': 23, 'Marshall Islands': 24, 'Maryland':
25, 'Massachusetts': 26, 'Michigan': 27, 'Minnesota': 28, 'Mississippi': 29, 'Misso
uri': 30, 'Montana': 31, 'Nebraska': 32, 'Nevada': 33, 'New Hampshire': 34, 'New Je
rsey': 35, 'New Mexico': 36, 'New York': 37, 'North Carolina': 38, 'North Dakota':
39, 'Northern Mariana Islands': 40, 'Ohio': 41, 'Oklahoma': 42, 'Oregon': 43, 'Pala
u': 44, 'Pennsylvania': 45, 'Puerto Rico': 46, 'Rhode Island': 47, 'South Carolina':
 48, 'South Dakota': 49, 'Tennessee': 50, 'Texas': 51, 'Utah': 52, 'Vermont': 53, '
```

```
Virgin Islands': 54, 'Virginia': 55, 'Washington': 56, 'West Virginia': 57, 'Wiscon
sin': 58, 'Wyoming': 59}
```

- **Feature Extraction and MultinomialNB Classifier**

```
    # Feature extraction
    vectorizer = CountVectorizer()
    X = vectorizer.fit_transform(map_filter_superbowl['text'])
    y = map_filter_superbowl["user_location"]

    # Splitting data into training and testing sets
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, ran
dom_state=42)

    # Training the model
    classifier = MultinomialNB()
    classifier.fit(X_train, y_train)

    # Testing the model
    predictions = classifier.predict(X_test)
    print_result(y_test, predictions, "MultinomialNB")
```

And we get the result of accuracy= 49.44, recall= 23.46, precision= 50.57, f1= 26.48.

- **HardMarginSVM and SoftMarginSVM**

```
    # Computing Hard and Soft Margin SVMs
    HardMargin_SVM = LinearSVC(C=1000, random_state=42, max_iter = 3000)
    HardMargin_SVM2 = LinearSVC(C=100000, random_state=42, max_iter = 3000)
    SoftMargin_SVM = LinearSVC(C=0.0001, random_state=42, max_iter = 3000)

    X_hardSVM_pred = HardMargin_SVM.fit(X_train,y_train).predict(X_test) # predi
cting labels for hard margin SVM
    X_hardSVM_pred2 = HardMargin_SVM2.fit(X_train,y_train).predict(X_test) # pre
dicting labels for hard margin SVM
    X_softSVM_pred = SoftMargin_SVM.fit(X_train,y_train).predict(X_test) # predi
cting labels for soft margin SVM
```

And we got the result of

hard margin svm C=1000: accuracy= 81.42, recall= 67.75, precision= 76.21, f1= 70.03

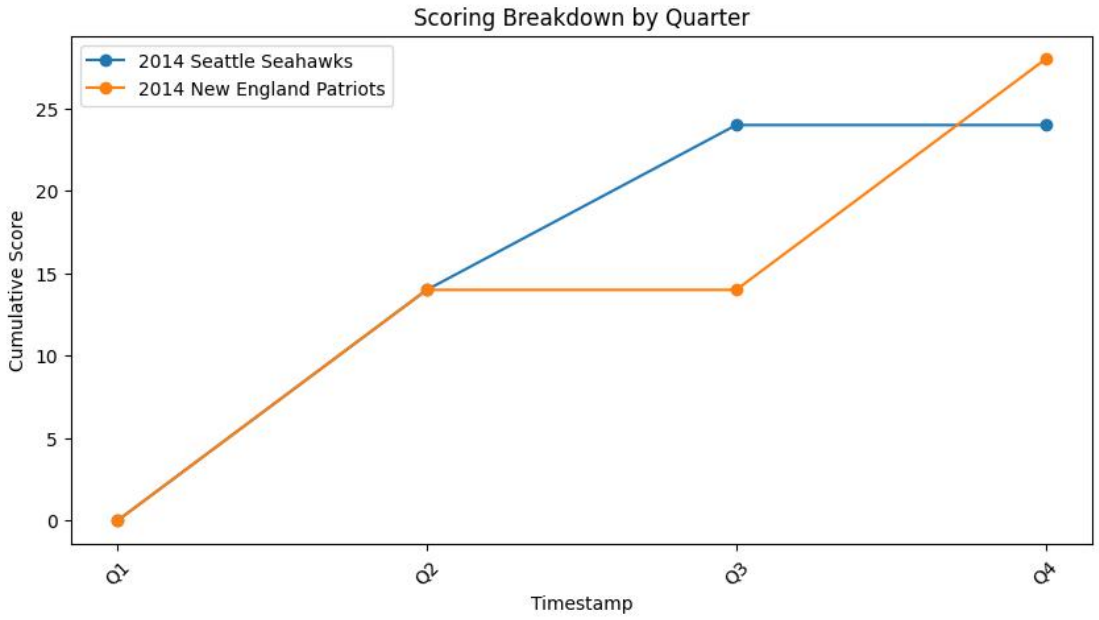
hard margin svm C=100000: accuracy= 81.42, recall= 67.75, precision= 76.21, f1= 70.03

soft margin svm C=0.0001: accuracy= 35.82, recall= 15.79, precision= 15.27, f1= 14.82.

Based on the results provided, we can see that the hard margin SVM models with C=1000 and C=100000 have the same performance metrics, with an accuracy of 81.42%, recall of 67.75%, precision of 76.21%, and F1 score of 70.03%. This indicates that increasing the value of C from 1000 to 100000 did not have a significant impact on the performance of the model.

On the other hand, the soft margin SVM model with C=0.0001 has a much lower performance than the hard margin SVM models, with an accuracy of 35.82%, recall of 15.79%, precision of 15.27%, and F1 score of 14.82%. This indicates that the soft margin SVM model is not able to correctly classify the data points as well as the hard margin SVM models.

- **IV The result score of both team and generation of tweets**
  - **the relationship between timestamp and scores**



  - **Use openai to generate tweets**

```
    game_info = {
        "team1": "2014 Seattle Seahawks",
        "team2": "2014 New England Patriots",
        "team1_score": 0,
        "team2_score": 0,
        "Quarter": 1
    }
    prompts = [
        f"Write a tweet summarizing the game between the {game_info['team1']} and the {game_info['team2']}, "
        f"where {game_info['team1']} scored {game_info['team1_score']} and {game_info['team2']} scored {game_info['team2_score']}. "
    ]

    for prompt in prompts:
        converted_prompt = f"Write a story using '{prompt}' as the prompt."
        completions = openai.Completion.create(
            engine="text-davinci-002",
            prompt=converted_prompt,
            max_tokens=1024,
            n=1,
            stop=None,
            temperature=0.5
        )
```

```
        message = completions.choices[0].text
        print(message)
```

And we got the generated tweet to be:

“The 2014 Seattle Seahawks and the 2014 New England Patriots faced off in a game where both teams scored 0. The Seahawks defense held strong, but the Patriots offense was unable to break through. In the end, it was a defensive battle between two of the best teams in the NFL.”

- **Find max mentioned player**

```
    # checks if a given tweet contains a given player name
    def contains_player(tweet, player):
        return player in tweet.lower()

    player_counts = {}
    tweets_with_player = []
    for player in players:
        ## Filter the dataframe to get only the tweets that mention the current player
        player_tweets = superbowl_data[superbowl_data["text"].apply(lambda tweet: contains_player(tweet, player))]["text"].tolist()
        ## Count the number of times the current player is mentioned in the filtered tweets
        player_count =  sum([tweet.count(player) for tweet in player_tweets])
        if player_count > 0:
            player_counts[player] = player_count

        # Add the filtered tweets to the list of tweets that mention any of the players
        tweets_with_player.extend(superbowl_data[superbowl_data["text"].apply(lambda tweet: contains_player(tweet, player))]["text"].tolist())
```

With this function, we get the occurrence of each player in tweets

tom brady: 85

marshawn lynch: 59

richard sherman: 15

russell wilson: 13

malcolm butler: 5

chris matthews: 4

julian edelman: 4

jermaine kearse: 3

rob gronkowski: 2

rob ninkovich: 2

ricardo lockette: 2

vince wilfork: 1

legarrette blount: 1

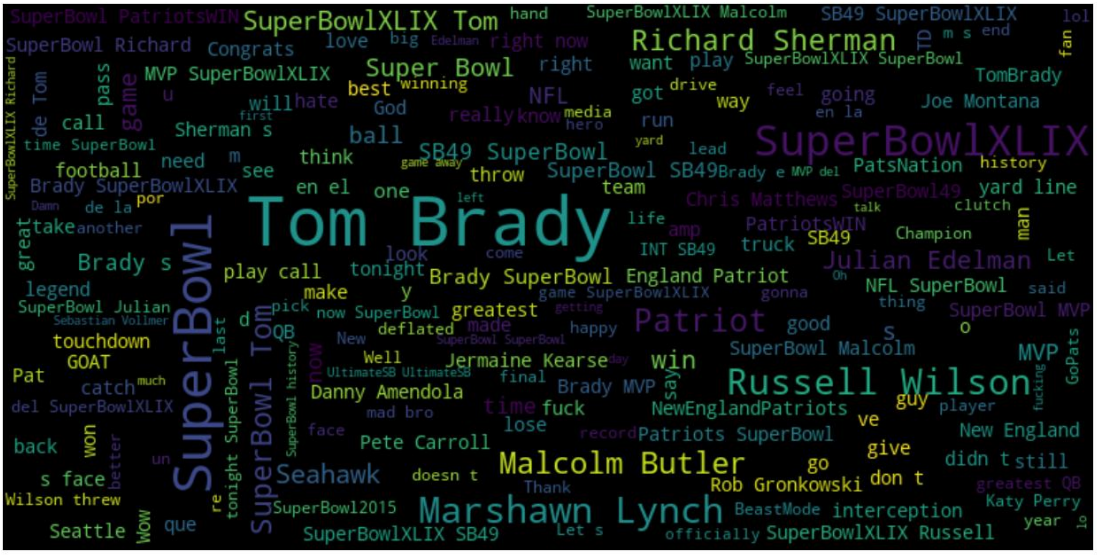
danny amendola: 1

jeremy lane: 1

cliff avril: 1

tharold simon: 1

And we draw the word cloud of players mentioned in the tweets



Conclusion

Overall, this project involved a comprehensive analysis of Twitter data using various techniques, including **data cleaning, feature extraction, regression analysis, and classification.** The insights gained from this analysis could have significant implications for predicting future trends based on Twitter data.

Final Project: Show Us Your Skills: Twitter Data

Wenxin Cheng 706070535 wenxin0319@g.ucla.edu

Yuxin Yin 606073780 yyxyy999@g.ucla.edu

Yingqian Zhao 306071513 zhaoyq99@g.ucla.edu

Question9

In [1]:
```
import os, re, csv, json, datetime
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import json
import datetime
import pytz
pst_tz = pytz.timezone('America/Los_Angeles')
```

In [2]:
```
tags2filename = {
    "t_gohawks" : "./ECE219_tweet_data/tweets_#gohawks.txt",
    "t_gopatriots" : "./ECE219_tweet_data/tweets_#gopatriots.txt",
    "t_nfl" : "./ECE219_tweet_data/tweets_#nfl.txt",
    "t_patriots" : "./ECE219_tweet_data/tweets_#patriots.txt",
    "t_sb49" : "./ECE219_tweet_data/tweets_#sb49.txt",
    "t_superbowl" : "./ECE219_tweet_data/tweets_#superbowl.txt"
}
```

In [5]:
```
# filename = t_gopatriots
def read_in_chunks(file_object, chunk_size=102400):
    """Lazy function (generator) to read a file piece by piece.
    Default chunk size: 1k."""
    while True:
        data = file_object.readlines(chunk_size)
        if not data:
            break
        yield data

def process_line_list(line_list):
    json_file = []
    citation_dates = []
    followers = []
    retweets = []
    num_tweets = 0
    for line in line_list:
        tweet = json.loads(line)
        citation_dates.append(tweet["citation_date"])
        followers.append(tweet["author"]["followers"])
        retweets.append(tweet['metrics']['citations']['total'])
        num_tweets += 1
    return {
        "num_tweets": num_tweets,
        "num_followers": sum(followers),
        "num_retweets": sum(retweets),
        "max_citation_date": max(citation_dates),
        "min_citation_date": min(citation_dates)
    }

def accumulate_results(results):
```

```
    num_tweets = sum([d["num_tweets"] for d in results])
    num_followers = sum([d["num_followers"] for d in results])
    num_retweets = sum([d["num_retweets"] for d in results])
    max_citation_date = max([d["max_citation_date"] for d in results])
    min_citation_date = min([d["min_citation_date"] for d in results])
    tws_per_hour = num_tweets / ((max_citation_date - min_citation_date) / 3600.0)
    avg_followers = num_followers / num_tweets
    avg_retweets = num_retweets / num_tweets
    print(f" Total number of tweets: {num_tweets}")
    print(f" Average number of tweets per hour: {tws_per_hour:.3f}")
    print(f" Average number of followers: {avg_followers:.3f}")
    print(f" Average number of retweets: {avg_retweets:.3f}")

def read_and_report(filename):
    with open(filename) as f:
        results = []
        for line_list in read_in_chunks(f):
            results.append(process_line_list(line_list))
        accumulate_results(results)
```

Question9.1

In [6]:
```
for tag, filename in tags2filename.items():
    print(f"start to read and report {filename}")
    read_and_report(filename)
```

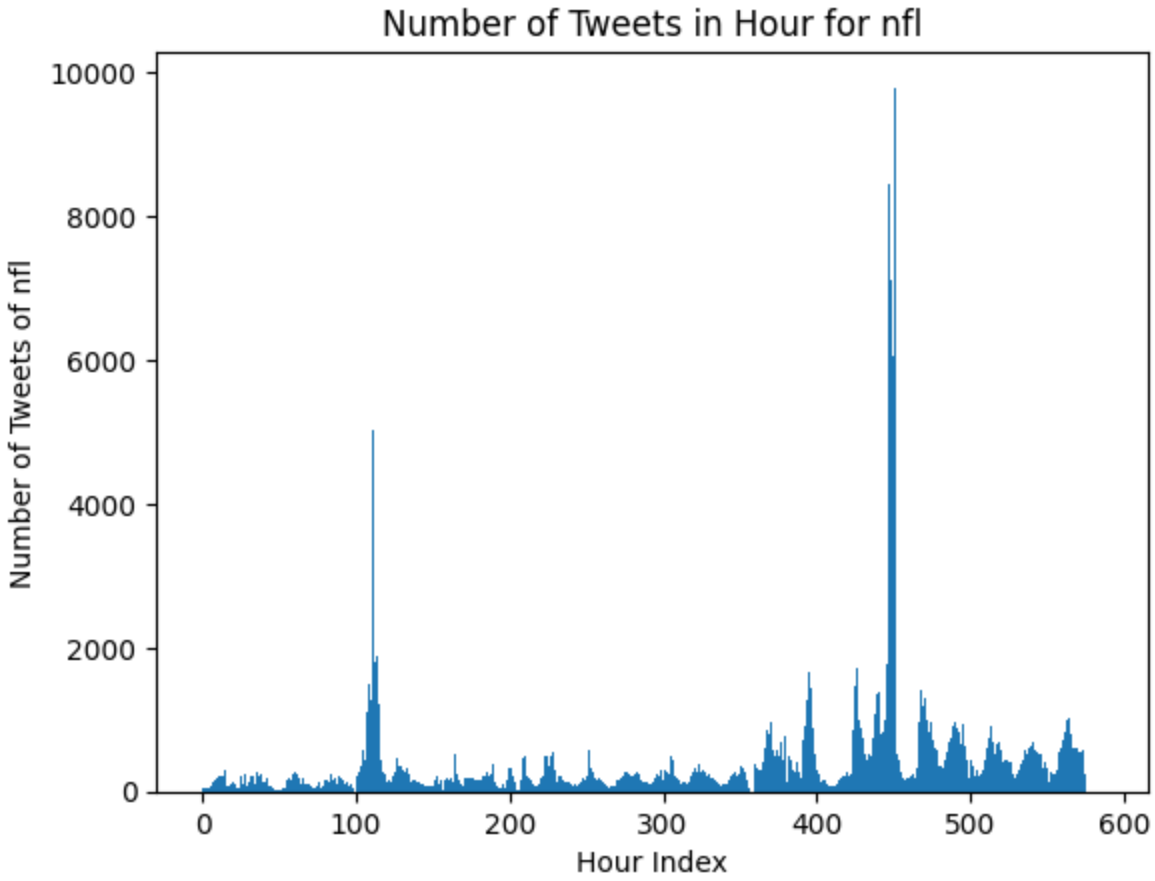
```
start to read and report ./ECE219_tweet_data/tweets_#gohawks.txt
 Total number of tweets: 169122
 Average number of tweets per hour: 292.488
 Average number of followers: 2217.924
 Average number of retweets: 2.013
start to read and report ./ECE219_tweet_data/tweets_#gopatriots.txt
 Total number of tweets: 23511
 Average number of tweets per hour: 40.955
 Average number of followers: 1427.253
 Average number of retweets: 1.408
start to read and report ./ECE219_tweet_data/tweets_#nfl.txt
 Total number of tweets: 233022
 Average number of tweets per hour: 397.021
 Average number of followers: 4662.375
 Average number of retweets: 1.534
start to read and report ./ECE219_tweet_data/tweets_#patriots.txt
 Total number of tweets: 440621
 Average number of tweets per hour: 750.894
 Average number of followers: 3280.464
 Average number of retweets: 1.785
start to read and report ./ECE219_tweet_data/tweets_#sb49.txt
 Total number of tweets: 743649
 Average number of tweets per hour: 1276.857
 Average number of followers: 10374.160
 Average number of retweets: 2.527
start to read and report ./ECE219_tweet_data/tweets_#superbowl.txt
 Total number of tweets: 1213813
 Average number of tweets per hour: 2072.118
 Average number of followers: 8814.968
 Average number of retweets: 2.391
```

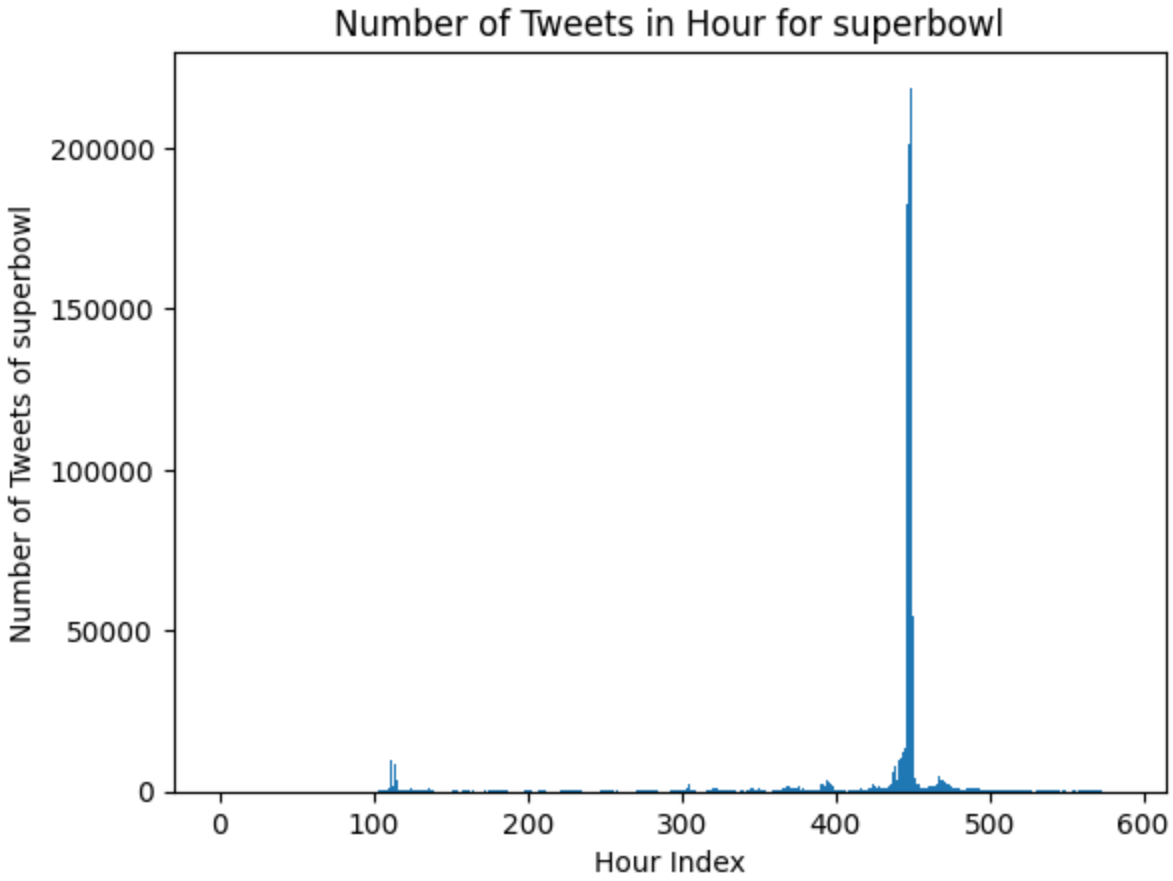
Question9.2

In [7]:
```
def plot_tweet_in_hour(filename): # for SuperBowl and NFL
    with open(filename) as f:
        citation_dates = []
        for line_list in read_in_chunks(f):
            for line in line_list:
```

```
            tweet = json.loads(line)
            citation_dates.append(tweet["citation_date"])

    earliest = min(citation_dates)

    counts_per_hour = [0] * (int)((max(citation_dates) - min(citation_dates))/3600.0
    for c_ in citation_dates:
        counts_per_hour[(int)((c_- earliest) / 3600.0)] += 1

    plt.bar(np.arange(len(counts_per_hour)), counts_per_hour, 1)
    plt.xlabel("Hour Index")
    filename_ = filename.split('#')[1].split('.')[0]
    plt.ylabel("Number of Tweets of " + filename_)
    plt.title("Number of Tweets in Hour for " + filename_)
    plt.show()
```

In [8]:
```
needs_plot = ["t_nfl","t_superbowl"]
for i in needs_plot:
    plot_tweet_in_hour(tags2filename[i])
```





Final Project: Show Us Your Skills: Twitter Data

Wenxin Cheng 706070535 wenxin0319@g.ucla.edu

Yuxin Yin 606073780 yyxyy999@g.ucla.edu

Yingqian Zhao 306071513 zhaoyq99@g.ucla.edu

Question10

Since time was start at 2015/2/1 18:36 and last for 3h 36min, so we select the posted tweet time from 2015/2/1 18:30 - 2015/2/1 22:30

In [10]:
```
from textblob import TextBlob
from datetime import datetime, timedelta

# Define a function to read a file in chunks
def read_in_chunks(file_object, chunk_size=102400):
    while True:
        data = file_object.readlines(chunk_size)
        if not data:
            break
        yield data

# Define variables for the start and end times of the Super Bowl in 2015
# and the time delta between each data point
superbowl_start_time = int(datetime(2015, 2, 1, 18, 30, tzinfo=pst_tz).timestamp())
superbowl_end_time = int(datetime(2015, 2, 1, 22, 30, tzinfo=pst_tz).timestamp())
time_delta = int(timedelta(minutes = 10).seconds)
```

extract each features we may use from original dataset

In [15]:
```
# Define a dictionary to map month names to integer values
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}

# Define a function to calculate the number of days between a user's creation time and a
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(
        int(user_create_date[-1]), mnth_to_int[user_create_date[1]], int(user_create_dat
    tweet_create_date = datetime(
        int(tweet_create_date[-1]), mnth_to_int[tweet_create_date[1]], int(tweet_create_
    created_days = tweet_create_date - user_create_date
    # Convert the number of days to an integer value
    created_days = created_days.days
    return created_days

# Define a function to clean the text of a tweet by removing URLs and special characters
def clean_tweet_text(tweet_text):
    tweet_text_ = re.sub(r"http\S+", "", tweet_text)
    return ' '.join(re.sub("(@[A-Za-z0-9]+)|([^0-9A-Za-z \t])|(\w+:\/\/\S+)", " ", tweet

def extract_feature_from_tweet_text(tweet_text):
    tweet_text = clean_tweet_text(tweet_text)
    # Create a TextBlob object to analyze the sentiment of the tweet
    analysis = TextBlob(tweet_text)
    pol = analysis.sentiment.polarity
    return {
        "text": tweet_text,
        "polarity": -1 if pol < -0.1 else (0 if -0.1 <= pol <= 0.1 else 1),
        "positive": pol > 0.1,
        "neutral": -0.1 <= pol <= 0.1,
        "negative": pol < -0.1
    }

def extract_feature_from_tweet_obj(tweet_obj):
    n_days = get_days(tweet_obj['tweet']['user']['created_at'], tweet_obj['tweet']['crea
    # Create a dictionary of basic features for the tweet
    feature = {
        "num_tweet": 1,
        "created_at": tweet_obj['firstpost_date'],
        "num_retweet": tweet_obj['metrics']['citations']['total'],
        "num_followers": tweet_obj['author']['followers'],
        "ranking_score": tweet_obj['metrics']['ranking_score'],
        "user_activity": tweet_obj['tweet']['user']['statuses_count'] / (n_days + 1),
        "user_id": tweet_obj['tweet']['user']['id'],
        "user_location": tweet_obj['tweet']['user']['location'],
        "user_mentions": len(tweet_obj['tweet']['entities']['user_mentions'])
    }
    feature.update(extract_feature_from_tweet_text(tweet_obj['tweet']['text']))
    return feature
```

save the data from time series(10 mins per period)

In [19]:
```
# This function takes in a train object and a feature object, and merges the feature obj
def merge_feature_into_train_data(train_obj, feature):
    train_obj["num_tweet"] += feature["num_tweet"]
    train_obj["num_retweet"] += feature["num_retweet"]
    train_obj["num_followers"] += feature["num_followers"]
    train_obj["ranking_score"] += feature["ranking_score"]
    train_obj["user_activity"] += feature["user_activity"]
    train_obj["user_id"].add(feature["user_id"])
    train_obj["user_location"] += feature["user_location"]
    train_obj["user_mentions"] += feature["user_mentions"]
```

```
    train_obj["num_positive"] += feature["positive"]
    train_obj["num_neutral"] += feature["neutral"]
    train_obj["num_negative"] += feature["negative"]
    train_obj["unique_user_id"] = len(train_obj["user_id"])

# This function takes in a filename and returns a pandas dataframe of the training data.
def get_train_data(filename): # for SuperBowl
    num_train_data = (superbowl_end_time - superbowl_start_time) // time_delta + 1
    train_data = [{
        "range_start": datetime.fromtimestamp(superbowl_start_time + i*time_delta),
        "range_end": datetime.fromtimestamp(superbowl_start_time + (i+1)*time_delta),
        "num_tweet": 0,
        "num_retweet": 0,
        "num_followers": 0,
        "ranking_score": 0,
        "user_activity": 0,
        "user_id": set(),
        "user_location": "",
        "user_mentions": 0,
        "num_positive": 0,
        "num_neutral": 0,
        "num_negative": 0,
        "text": "",
        "polarity": 0
    } for i in range(num_train_data)]
    with open(filename) as f:
        # Open the output file for writing all the data
        with open('data.json', 'w') as outfile:
            for line_list in read_in_chunks(f):
                for line in line_list:
                    tweet_obj = json.loads(line)
                    firstpost_date = int(tweet_obj["firstpost_date"])
                    # If the tweet is outside the range we care about, skip it
                    if not superbowl_start_time <= firstpost_date <= superbowl_end_time:
                        continue
                    # Calculate which time slot this tweet falls into
                    index = (firstpost_date - superbowl_start_time) // time_delta
                    feature = extract_feature_from_tweet_obj(tweet_obj)
                    json.dump(feature, outfile)
                    outfile.write('\n')
                    # Merge the feature into the appropriate slot in the training data
                    merge_feature_into_train_data(train_data[index], feature)
    return pd.DataFrame(train_data)
```

In [20]:
```
train_data = get_train_data(tags2filename["t_superbowl"])

# Save the training data as a CSV file for better read latter
train_data.to_csv('data.csv', index=False)
```

read the train times series data and all train features from csv/json file to save time

In [59]:
```
# Load the Super Bowl time slots training data from the CSV file
superbowl_merge_data = pd.read_csv('data.csv')

# Load all Super Bowl training data from the JSON file
superbowl_data = []
with open('data.json', 'r') as infile:
    for line in infile:
        superbowl_data.append(json.loads(line))

# Create a DataFrame from the list of JSON objects
superbowl_data = pd.DataFrame(superbowl_data)
superbowl_data["created_at"] = superbowl_data["created_at"].apply(datetime.datetime.from
```

```
print(superbowl_data.shape)
superbowl_data.head(5)
```

```
(204993, 14)
```

Out[59]:

	num_tweet	created_at	num_retweet	num_followers	ranking_score	user_activity	user_id	user_lo
0	1	2015-02-01 18:23:00	1	2941.0	7.706944	37.000000	34852676	816/77
1	1	2015-02-01 18:23:00	7	5009.0	7.486162	2.701126	14386730	The the Ur
2	1	2015-02-01 18:23:00	2	1486.0	7.302760	5.843216	133804848	Chic
3	1	2015-02-01 18:23:00	1	347.0	4.491231	7.604738	190105015	Akr
4	1	2015-02-01 18:23:00	1	54.0	4.365862	2.158416	468055147	

In [61]:

```
print(superbowl_merge_data.shape)
superbowl_merge_data.head(2)
```

Out[61]:

	range_start	range_end	num_tweet	num_retweet	num_followers	ranking_score	user_activity	
0	2015-02-01 18:23:00	2015-02-01 18:33:00	20036	42248	204685593.0	92221.22883	300091.205496	{360 26 38 112
1	2015-02-01 18:33:00	2015-02-01 18:43:00	15873	57399	141105759.0	72237.96470	219360.108740	{2868 38 156 2323

Find the polarity with the mention of "superbowl"

In [62]:

```
TRACK_WORDS = 'superbowl'
# Check if each tweet contains the tracked word
superbowl_data['contains_superbowl'] = superbowl_data['text'].str.contains(TRACK_WORDS,
```

In [63]:

```
# Group the data by polarity and time (in 10-minute intervals) and count the number of t
result = superbowl_data.groupby([pd.Grouper(key='created_at', freq='600s'), 'polarity'])

# Rename the columns of the resulting DataFrame
result = result.rename(columns={
    "id_str": "Num of '{}' mentions".format(TRACK_WORDS),
    "created_at": "Time in PST",
    0: "Mention_Times"
})
```

In [64]:

```
result
```

Out[64]:

	Time in PST	polarity	Mention_Times
0	2015-02-01 18:20:00	-1	1767

1	2015-02-01 18:20:00	0	8281
2	2015-02-01 18:20:00	1	3236
3	2015-02-01 18:30:00	-1	1962
4	2015-02-01 18:30:00	0	10765
...	...	...	...
70	2015-02-01 22:10:00	0	227
71	2015-02-01 22:10:00	1	175
72	2015-02-01 22:20:00	-1	15
73	2015-02-01 22:20:00	0	83
74	2015-02-01 22:20:00	1	38

75 rows × 3 columns

In [55]:
```
# Extract the time series (i.e., timestamps) for tweets with neutral polarity
time_series = result["Time in PST"][result['polarity']==0].reset_index(drop=True)
```

In [56]:
```
import plotly.express as px
fig = px.line(result, x='Time in PST',                    \
    y='Mention_Times',            \
    color='polarity')
fig.show()
```

In [66]:
```
import re
content = ' '.join(superbowl_data["text"])
content = re.sub(r"http\S+", "", content)
# Remove "RT" (retweet) and "&amp;" characters from the text
content = content.replace('RT ', ' ').replace('&amp;', 'and')
content = re.sub('[^A-Za-z0-9]+', ' ', content)
content = content.lower()

import nltk
nltk.download('punkt')
nltk.download('stopwords')
```

```
[nltk_data] Downloading package punkt to C:\Users\wenxin
[nltk_data]     cheng\AppData\Roaming\nltk_data...
[nltk_data]   Package punkt is already up-to-date!
[nltk_data] Downloading package stopwords to C:\Users\wenxin
[nltk_data]     cheng\AppData\Roaming\nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
```

Out[66]:
```
True
```

Find the top freqency words used in the Tweets

In [68]:
```
from nltk.probability import FreqDist
from nltk.tokenize import word_tokenize
from nltk.corpus import stopwords

# Tokenize the preprocessed text into individual words
tokenized_word = word_tokenize(content)
stop_words=set(stopwords.words("english"))


# Filter out stopwords from the tokenized text to create a list of meaningful words
filtered_sent=[]
```

```
for w in tokenized_word:
    if w not in stop_words:
        filtered_sent.append(w)

# Use the FreqDist function from NLTK to create a frequency distribution of the filtered
fdist = FreqDist(filtered_sent)
fd = pd.DataFrame(fdist.most_common(10),                    \
    columns = ["Word","Frequency"]).drop([0]).reindex()
```

In [69]:
```
import plotly.express as px
fig = px.bar(fd, x="Word", y="Frequency")
fig.update_traces(marker_color='rgb(240,128,128)',          \
    marker_line_color='rgb(8,48,107)',                      \
    marker_line_width=1.5, opacity=0.8)
fig.show()
```

Task1 find the relation between time serires train features and num_tweets next 10 minutes

Get the time series train data

In [3]:
```
superbowl_merge_data_drop = superbowl_merge_data.copy().drop(['range_start','range_end',
```

In [4]:
```
print(superbowl_merge_data_drop.shape)
superbowl_merge_data_drop.head(3)
```

```
(25, 9)
```

Out[4]:

	num_retweet	num_followers	ranking_score	user_activity	user_mentions	num_positive	num_neutra
0	42248	204685593.0	92221.228830	300091.205496	6043	5499	1202
1	57399	141105759.0	72237.964700	219360.108740	5369	4704	956
2	45992	205330757.0	107368.651934	386185.215763	5963	5462	1551

In [5]:
```
# because we need to generate x and y, we have 25 lines, so the last item of x and first
superbowl_x = superbowl_merge_data_drop.drop(superbowl_merge_data_drop.index[-1])
superbowl_y = superbowl_merge_data["num_tweet"]
superbowl_y = superbowl_y.drop(superbowl_y.index[0])
superbowl_y = pd.DataFrame(superbowl_y,columns = ["num_tweet"]).values.ravel()
```

In [6]:
```
print(superbowl_x.shape)
superbowl_x.head(3)
```

```
(24, 9)
```

Out[6]:

	num_retweet	num_followers	ranking_score	user_activity	user_mentions	num_positive	num_neutra
0	42248	204685593.0	92221.228830	300091.205496	6043	5499	1202
1	57399	141105759.0	72237.964700	219360.108740	5369	4704	956
2	45992	205330757.0	107368.651934	386185.215763	5963	5462	1551

In [7]:
```
print(superbowl_y.shape)
```

```
(24,)
```

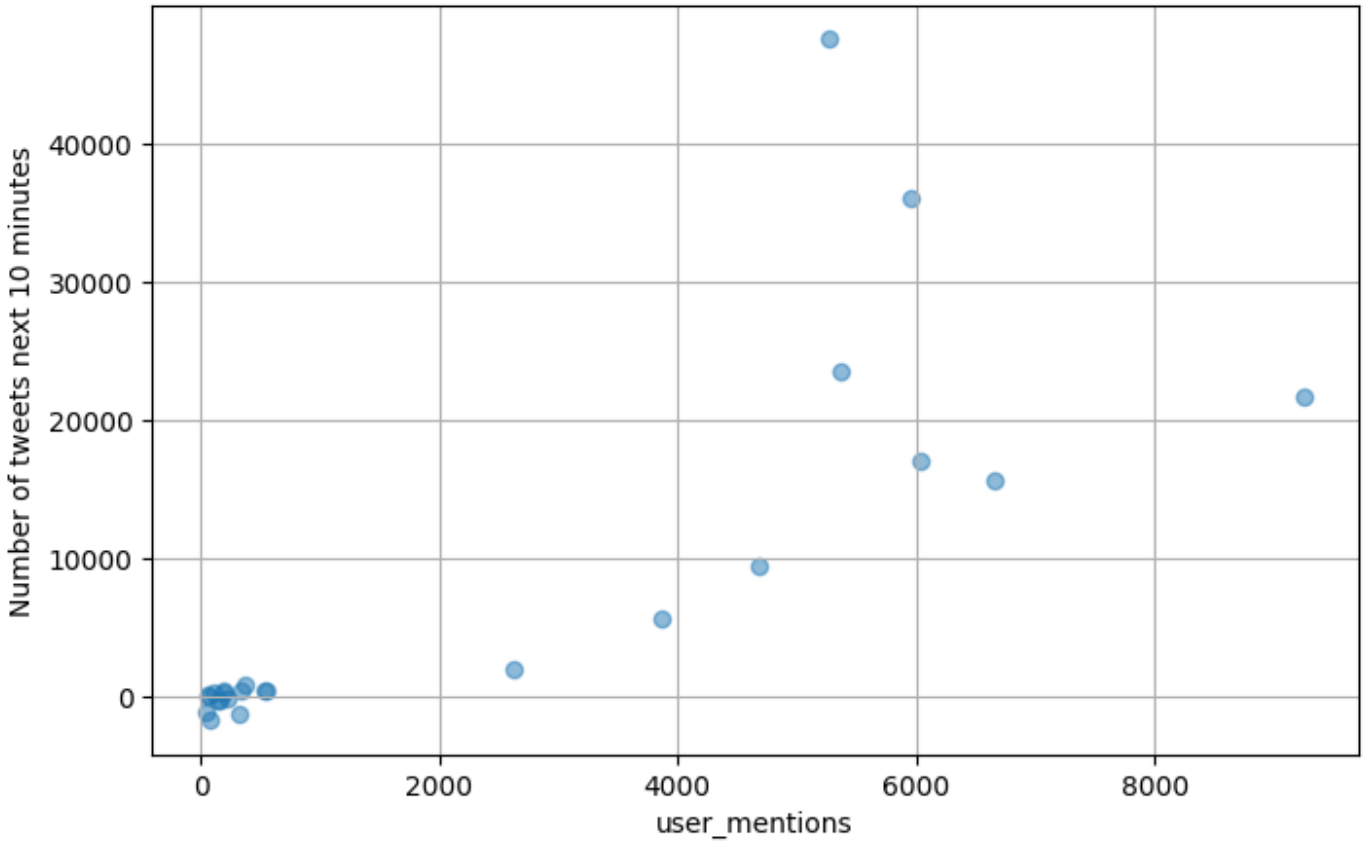
analyse the significance of each feature using the t-test and p-value

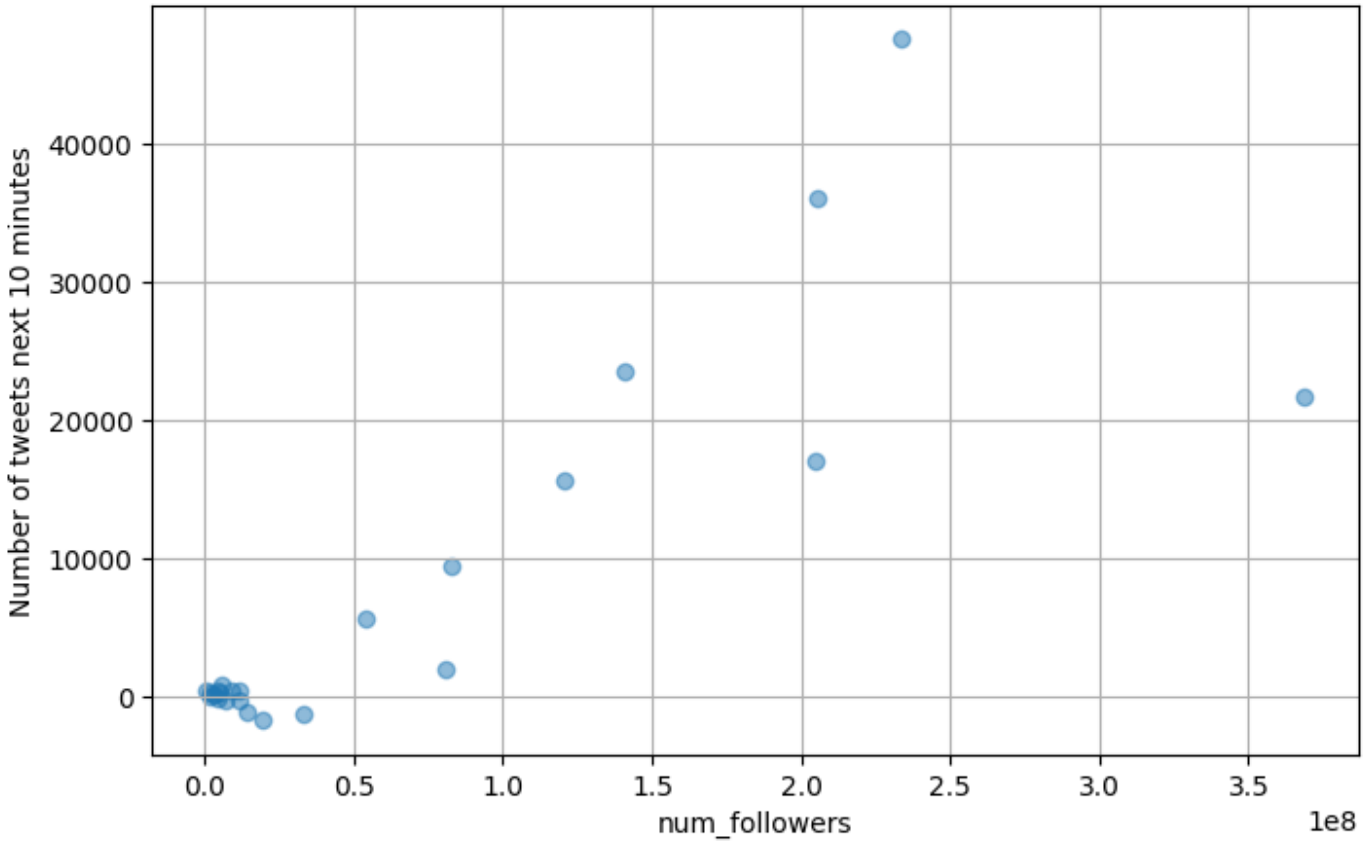
OLS Regression

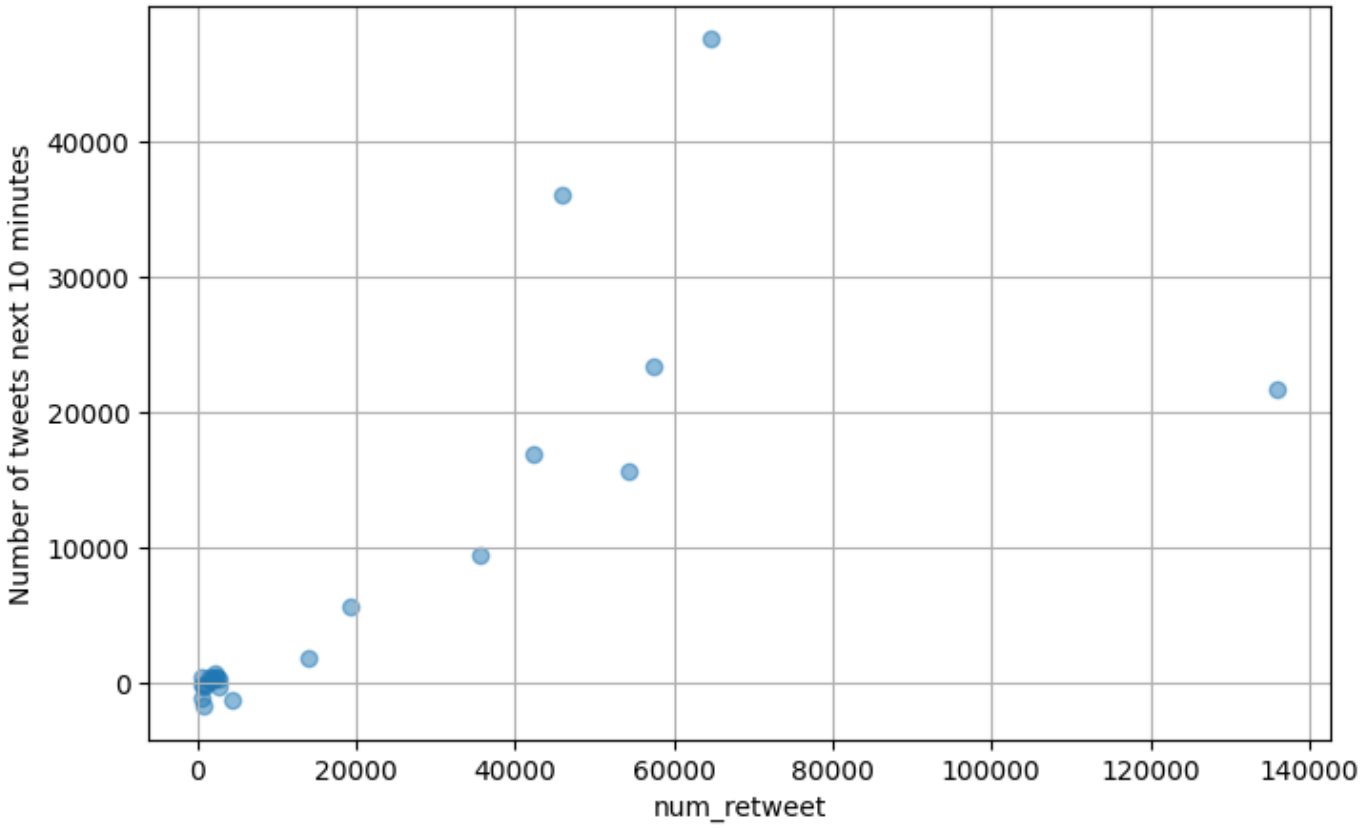
In [9]:
```
import statsmodels.api as sm
from sklearn import metrics
import numpy as np
import matplotlib.pyplot as plt

feature_names = list(superbowl_x.columns)
print(feature_names)

def scatter_plot(features, y_pred, pvalues, feature_names):
    # Obtain the indices that would sort the p-values in ascending order
    ranked_index = np.argsort(pvalues)
    print(ranked_index)
    for i in range(9):
        plt.figure(figsize = (8,5))
        # Create a scatter plot of the ith feature against the predicted values
        plt.scatter(features.iloc[:,ranked_index[i]], y_pred, alpha=0.5)
        plt.xlabel(feature_names[ranked_index[i]])
        plt.ylabel("Number of tweets next 10 minutes")
        plt.grid(True)
        plt.show()
    print('-' * 80)

# Fit a linear regression model to the data and obtain the predicted values and p-values
lr_fit = sm.OLS(superbowl_y,superbowl_x).fit()
y_pred = lr_fit.predict()
pvalues = lr_fit.pvalues
print('MSE: ', metrics.mean_squared_error(superbowl_y, y_pred))
print(lr_fit.summary())
scatter_plot(superbowl_x, y_pred, pvalues, feature_names)
print('\n')
```

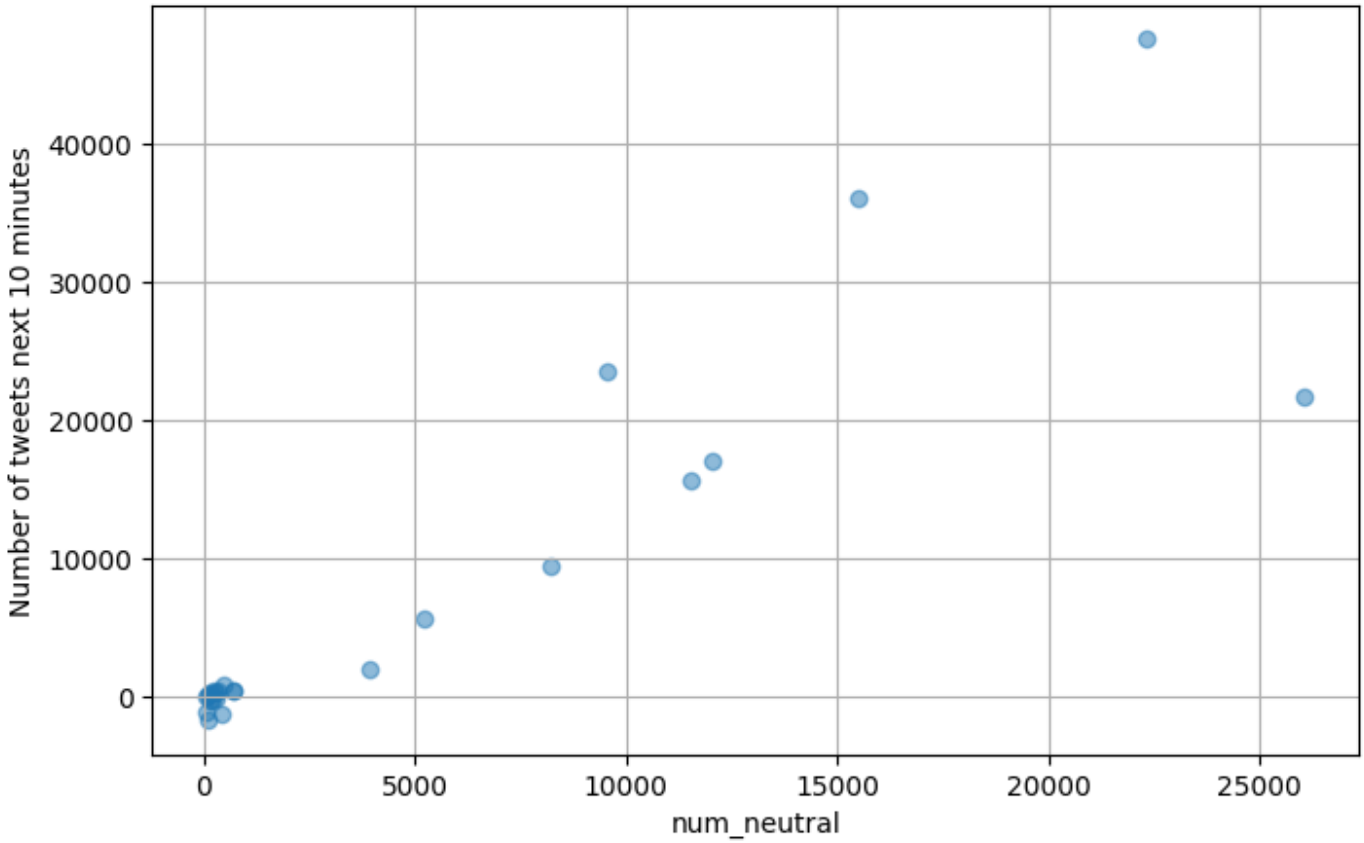
```
['num_retweet', 'num_followers', 'ranking_score', 'user_activity', 'user_mentions', 'num
_positive', 'num_neutral', 'num_negative', 'unique_user_id']
MSE:  755168.0833725476
                                 OLS Regression Results
================================================================================================
Dep. Variable:                          y   R-squared (uncentered):                   0.997
Model:                                OLS   Adj. R-squared (uncentered):              0.994
Method:                     Least Squares   F-statistic:                              481.5
Date:                    Mon, 13 Mar 2023   Prob (F-statistic):                    8.26e-17
Time:                            21:23:42   Log-Likelihood:                         -196.47
No. Observations:                      24   AIC:                                      410.9
Df Residuals:                          15   BIC:                                      421.5
Df Model:                               9
Covariance Type:                nonrobust
=================================================================================
                    coef    std err          t      P>|t|      [0.025      0.975]
---------------------------------------------------------------------------------
num_retweet        0.2927      0.077      3.811      0.002       0.129       0.456
num_followers  -9.607e-05   2.27e-05     -4.241      0.001      -0.000   -4.78e-05
ranking_score     -1.4769      1.116     -1.323      0.206      -3.857       0.903
user_activity     -0.0340      0.023     -1.456      0.166      -0.084       0.016
user_mentions      5.8656      1.054      5.565      0.000       3.619       8.112
num_positive      -2.9337      8.243     -0.356      0.727     -20.504      14.637
num_neutral       14.7212      7.175      2.052      0.058      -0.573      30.015
num_negative       8.4435      6.140      1.375      0.189      -4.643      21.530
unique_user_id    -2.6738      2.862     -0.934      0.365      -8.775       3.427
==============================================================================
Omnibus:                        0.116   Durbin-Watson:                   1.540
Prob(Omnibus):                  0.943   Jarque-Bera (JB):                0.330
Skew:                           0.080   Prob(JB):                        0.848
Kurtosis:                       2.448   Cond. No.                     6.53e+06
==============================================================================

Notes:
[1] R² is computed without centering (uncentered) since the model does not contain a con
```

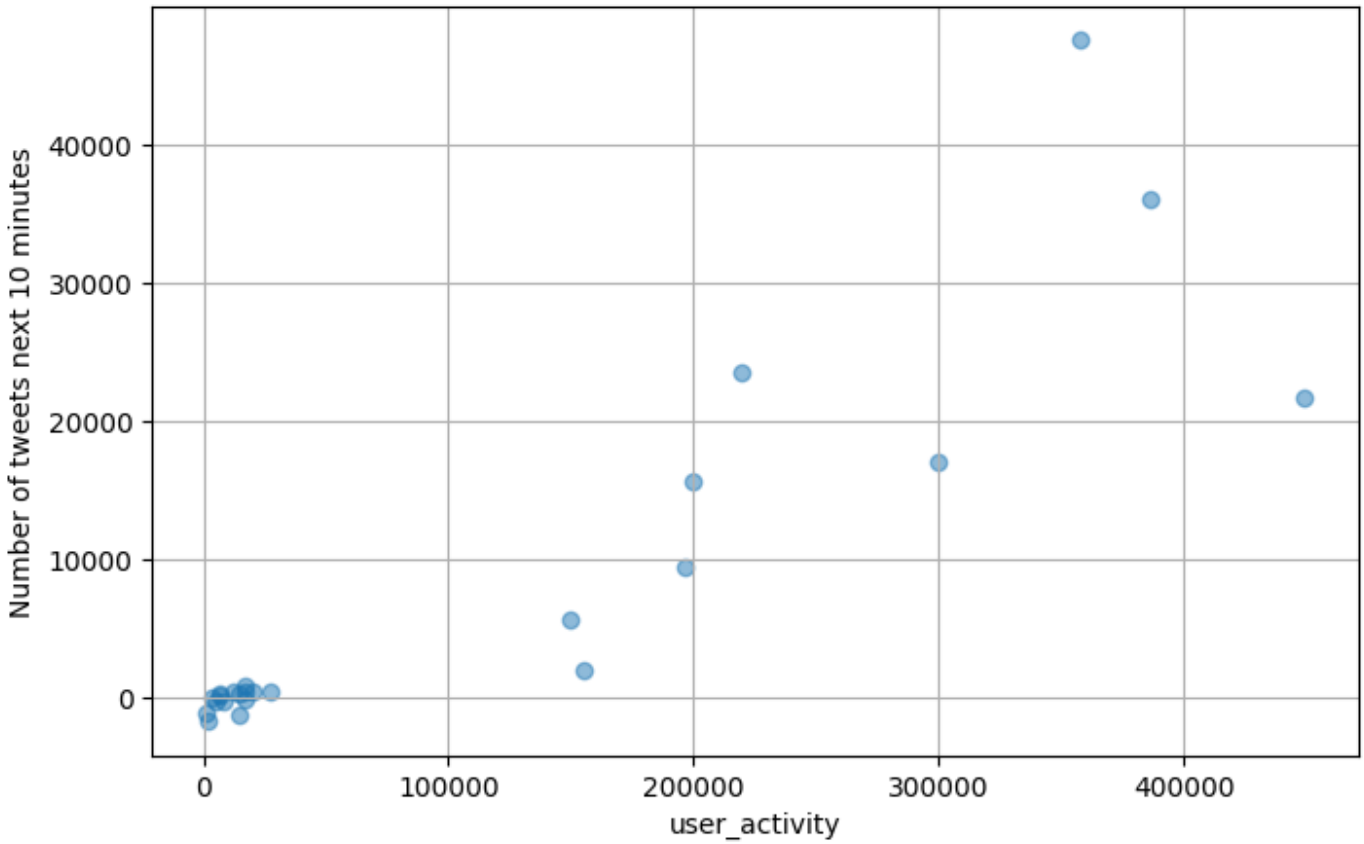
```
stant.
[2] Standard Errors assume that the covariance matrix of the errors is correctly specifi
ed.
[3] The condition number is large, 6.53e+06. This might indicate that there are
strong multicollinearity or other numerical problems.
num_retweet       4
num_followers     1
ranking_score     0
user_activity     6
user_mentions     3
num_positive      7
num_neutral       2
num_negative      8
unique_user_id    5
dtype: int64
```

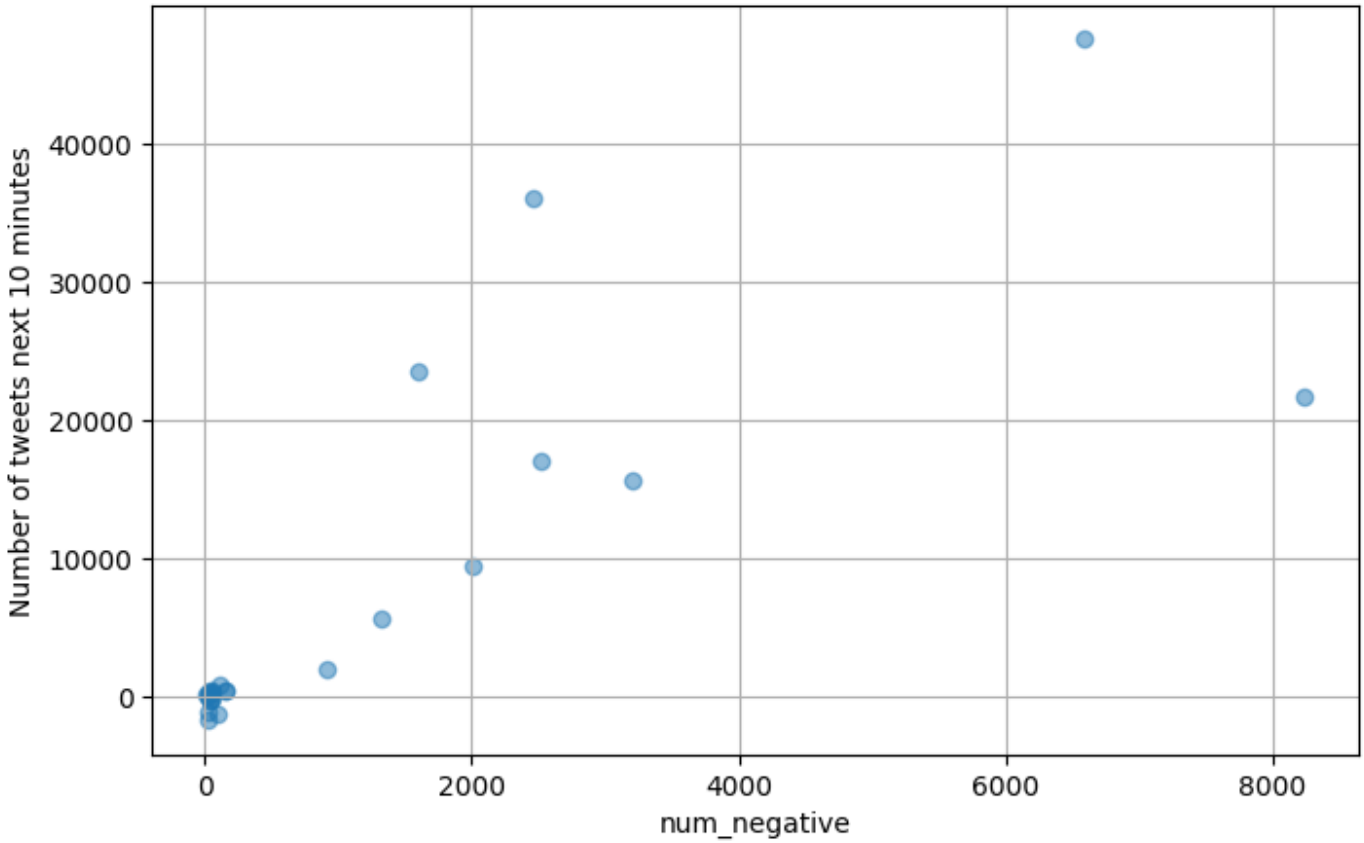


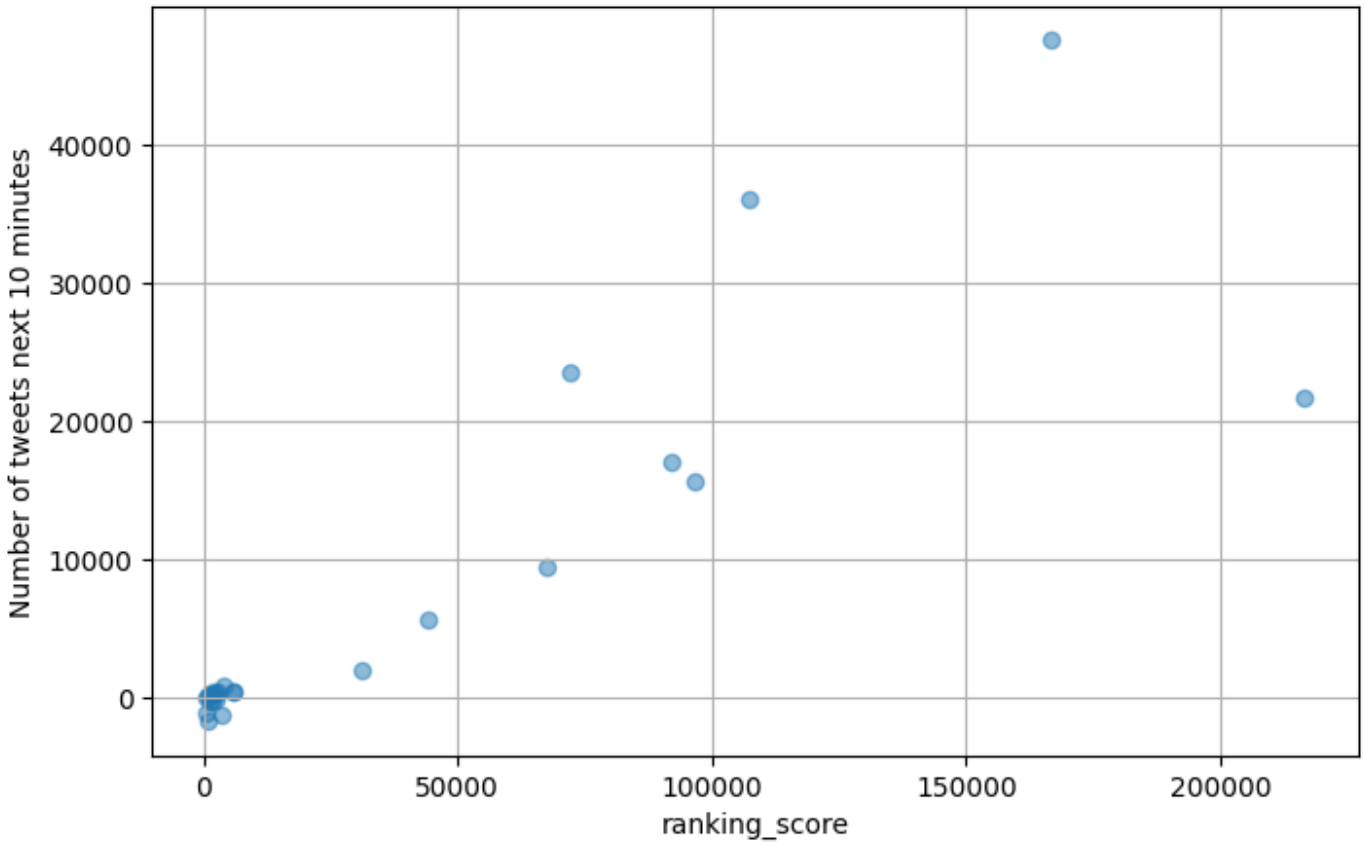

Number of tweets next 10 minutes
40000
30000
20000
10000
0
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
num_followers
1e8

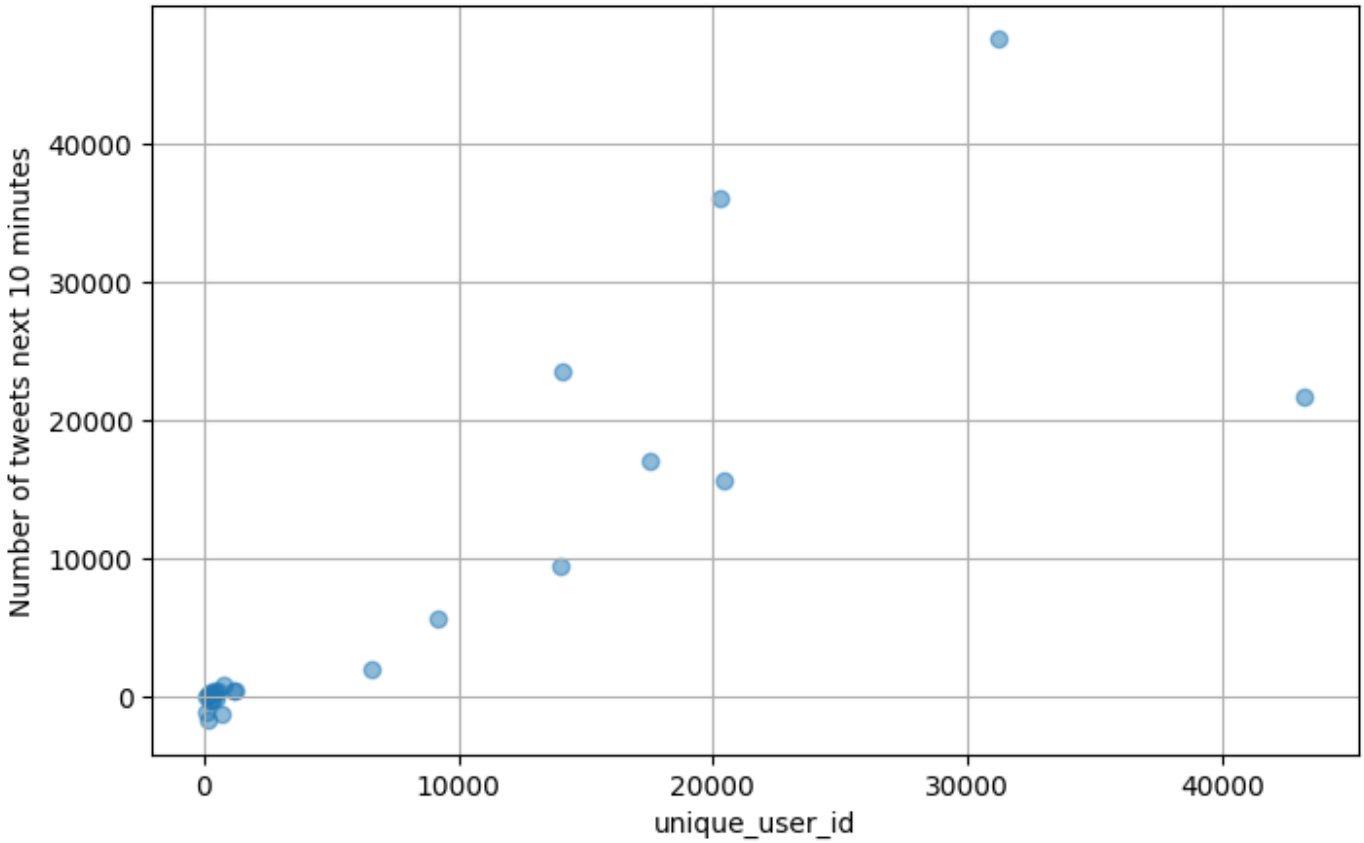

Number of tweets next 10 minutes
40000
30000
20000
10000
0
0
20000
40000
60000
80000
100000
120000
140000
num_retweet

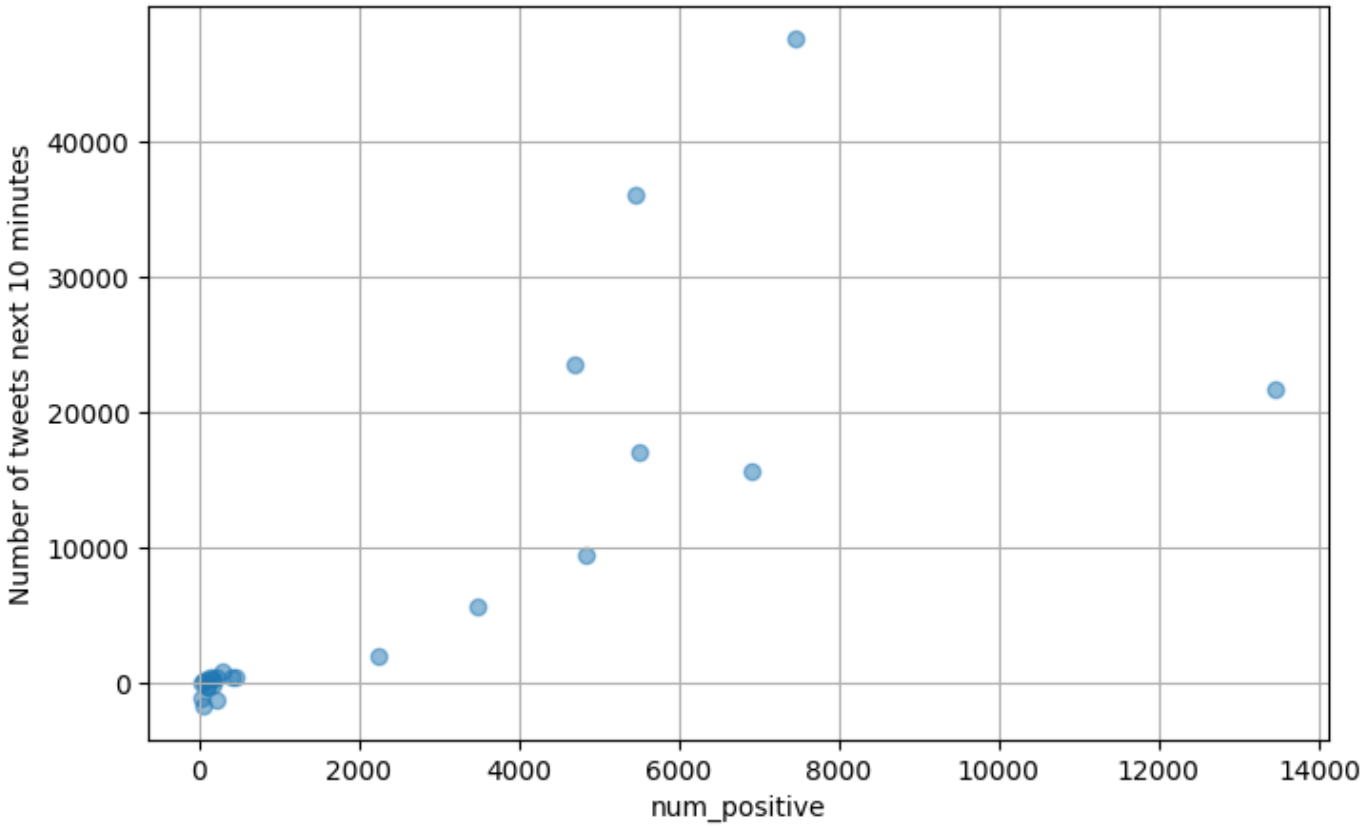

Number of tweets next 10 minutes
40000
30000
20000
10000
0
0
5000
10000
15000
20000
25000
num_neutral


Number of tweets next 10 minutes
40000
30000
20000
10000
0
0
100000
200000
300000
400000
user_activity


Number of tweets next 10 minutes
40000
30000
20000
10000
0
0
2000
4000
6000
8000
num_negative


Number of tweets next 10 minutes
40000
30000
20000
10000
0
0
50000
100000
150000
200000
ranking_score





The regression analysis of time series train features and num_tweets next 10 minutes

Data Standarizaion

In [65]:
```
from sklearn.preprocessing import StandardScaler
from sklearn import preprocessing
```

```
superbowl_merge_data_standard_x = pd.DataFrame(preprocessing.scale(superbowl_merge_data_
# superbowl_merge_data_standard_x = superbowl_merge_data_drop
```

In [66]:
```
superbowl_merge_data_standard_x = superbowl_merge_data_standard_x.drop(superbowl_merge_d
```

In [67]:
```
print(superbowl_merge_data_standard_x.shape)
```

```
(24, 9)
```

In [68]:
```
superbowl_merge_data_standard_y =  pd.DataFrame(preprocessing.scale(superbowl_merge_data

superbowl_merge_data_standard_y = superbowl_merge_data_standard_y.drop(superbowl_merge_d
print(superbowl_merge_data_standard_y.shape)
```

```
(24, 1)
```

In [69]:
```
superbowl_merge_data_standard_y.head()
```

Out[69]:

	num_tweet
1	0.608591
2	1.208436
3	2.231415
4	3.136456
5	1.067655

mutual_info_regression and f_regression

In [ ]:
```
from sklearn.feature_selection import SelectKBest, mutual_info_regression, f_regression
import numpy as np

# Define a function that selects the top n most important features using mutual informat
def select_topn_important_features(X, Y, n):
  Mutual_ = mutual_info_regression(X, Y)
  F_ = f_regression(X, Y)

  # Select the top n features based on their mutual information and F-test scores
  topn_M = np.argsort(Mutual_)[::-1][:n]
  topn_F = np.argsort(F_[0])[::-1][:n]

  # Sort all the features based on their mutual information and F-test scores
  all_m = np.argsort(Mutual_)[::-1]
  all_f = np.argsort(F_[0])[::-1]

  # Extract the top n features and all features based on their mutual information scores
  X_topn_M = X.iloc[:, topn_M]
  X_topn_F = X.iloc[:, topn_F]

  # Extract the top n features and all features based on their F-test scores
  all_m_ = X.iloc[:, all_m]
  all_f_ = X.iloc[:, all_f]

  return X_topn_M, X_topn_F, all_m_, all_f_
```

In [71]:
```
superbowl_top3_M, superbowl_top3_F, sall_m, sall_f = select_topn_important_features(supe
```

```
C:\Users\wenxin cheng\AppData\Roaming\Python\Python38\site-packages\sklearn\utils\valida
tion.py:993: DataConversionWarning: A column-vector y was passed when a 1d array was exp
ected. Please change the shape of y to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
C:\Users\wenxin cheng\AppData\Roaming\Python\Python38\site-packages\sklearn\utils\valida
```

```
tion.py:993: DataConversionWarning: A column-vector y was passed when a 1d array was exp
ected. Please change the shape of y to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
```

In [72]:
```
print("superbowl Top3 by mutual_info_regression:")
print(superbowl_top3_M.columns)

print("superbowl Top3 by f_regression:")
print(superbowl_top3_F.columns)

print("superbowl all by mutual_info_regression:")
print(sall_m.columns)

print("superbowl all by f_regression:")
print(sall_f.columns)
```

```
superbowl Top3 by mutual_info_regression:
Index(['unique_user_id', 'ranking_score', 'num_neutral'], dtype='object')
superbowl Top3 by f_regression:
Index(['num_neutral', 'user_activity', 'ranking_score'], dtype='object')
superbowl all by mutual_info_regression:
Index(['unique_user_id', 'ranking_score', 'num_neutral', 'num_positive',
       'user_activity', 'user_mentions', 'num_negative', 'num_retweet',
       'num_followers'],
      dtype='object')
superbowl all by f_regression:
Index(['num_neutral', 'user_activity', 'ranking_score', 'unique_user_id',
       'num_followers', 'num_negative', 'user_mentions', 'num_positive',
       'num_retweet'],
      dtype='object')
```

Test Linear Regression, Ridge regularization, and Lasso regularization

Judge how many train features need to select

In [74]:
```
# now we are do experiments of exactly how many features we need to select

#with selection
superbowl_rmse_lr_m = []  #superbowl rmse score for linear regression and with mutual_in
superbowl_rmse_lr_f = []
superbowl_rmse_r_m = []
superbowl_rmse_r_f = []
superbowl_rmse_la_m = []
superbowl_rmse_la_f = []

#without selection
superbowl_rmse_lr = [] #superbowl rmse score for linear regression
superbowl_rmse_r = []
superbowl_rmse_la = []
```

In [76]:
```
from sklearn.model_selection import cross_validate,GridSearchCV
from sklearn.linear_model import LinearRegression, Ridge, Lasso

for k in range(1, 10):
    superbowl_topk_M, superbowl_topk_F = select_topn_important_features(superbowl_merge_
    #superbowl rmse score for linear regression
    score_ = cross_validate(LinearRegression(), superbowl_merge_data_standard_x, superbo
    score__ = score_['test_neg_root_mean_squared_error'].mean()
    print(f"superbowl rmse score for linear regression, {score__:.4f}")
    superbowl_rmse_lr.append(score__)

    #superbowl rmse score for linear regression and with mutual_info_regression
    score_ = cross_validate(LinearRegression(), superbowl_topk_M, superbowl_merge_data_s
    score__ = score_['test_neg_root_mean_squared_error'].mean()
    print(f"superbowl rmse score for linear regression and with mutual_info_regression,
```

```
    superbowl_rmse_lr_m.append(score__)

    #superbowl rmse score for linear regression and with f_regression
    score_ = cross_validate(LinearRegression(), superbowl_topk_F, superbowl_merge_data_s
    score__ = score_['test_neg_root_mean_squared_error'].mean()
    print(f"superbowl rmse score for linear regression and with f_regression, {score__:.
    superbowl_rmse_lr_f.append(score__)

    #superbowl rmse score for Ridge regression
    score_ = cross_validate(Ridge(), superbowl_merge_data_standard_x, superbowl_merge_da
    score__ = score_['test_neg_root_mean_squared_error'].mean()
    print(f"superbowl rmse score for Ridge regression, {score__:.4f}")
    superbowl_rmse_r.append(score__)

    #superbowl rmse score for Ridge regression and with mutual_info_regression
    score_ = cross_validate(Ridge(), superbowl_topk_M, superbowl_merge_data_standard_y,
    score__ = score_['test_neg_root_mean_squared_error'].mean()
    print(f"superbowl rmse score for Ridge regression and with mutual_info_regression, {
    superbowl_rmse_r_m.append(score__)

    #superbowl rmse score for Ridge regression and with f_regression
    score_ = cross_validate(Ridge(), superbowl_topk_F, superbowl_merge_data_standard_y,
    score__ = score_['test_neg_root_mean_squared_error'].mean()
    print(f"superbowl rmse score for Ridge regression and with f_regression, {score__:.4
    superbowl_rmse_r_f.append(score__)

    #superbowl rmse score for Lasso regression
    score_ = cross_validate(Lasso(), superbowl_merge_data_standard_x, superbowl_merge_da
    score__ = score_['test_neg_root_mean_squared_error'].mean()
    print(f"superbowl rmse score for Lasso regression, {score__:.4f}")
    superbowl_rmse_la.append(score__)

    #superbowl rmse score for Lasso regression and with mutual_info_regression
    score_ = cross_validate(Lasso(), superbowl_topk_M, superbowl_merge_data_standard_y,
    score__ = score_['test_neg_root_mean_squared_error'].mean()
    print(f"superbowl rmse score for Lasso regression and with mutual_info_regression, {
    superbowl_rmse_la_m.append(score__)

    #superbowl rmse score for Lasso regression and with f_regression
    score_ = cross_validate(Lasso(), superbowl_topk_F, superbowl_merge_data_standard_y,
    score__ = score_['test_neg_root_mean_squared_error'].mean()
    print(f"superbowl rmse score for Lasso regression and with f_regression, {score__:.4
    superbowl_rmse_la_f.append(score__)
```

```
superbowl rmse score for linear regression, -0.4186
C:\Users\wenxin cheng\AppData\Roaming\Python\Python38\site-packages\sklearn\utils\valida
tion.py:993: DataConversionWarning: A column-vector y was passed when a 1d array was exp
ected. Please change the shape of y to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
C:\Users\wenxin cheng\AppData\Roaming\Python\Python38\site-packages\sklearn\utils\valida
tion.py:993: DataConversionWarning: A column-vector y was passed when a 1d array was exp
ected. Please change the shape of y to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
C:\Users\wenxin cheng\AppData\Roaming\Python\Python38\site-packages\sklearn\utils\valida
tion.py:993: DataConversionWarning: A column-vector y was passed when a 1d array was exp
ected. Please change the shape of y to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
C:\Users\wenxin cheng\AppData\Roaming\Python\Python38\site-packages\sklearn\utils\valida
tion.py:993: DataConversionWarning: A column-vector y was passed when a 1d array was exp
ected. Please change the shape of y to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
superbowl rmse score for linear regression and with mutual_info_regression, -0.3504 top1
superbowl rmse score for linear regression and with f_regression, -0.2704 top1
superbowl rmse score for Ridge regression, -0.3371
superbowl rmse score for Ridge regression and with mutual_info_regression, -0.3439 top1
```

```
superbowl rmse score for Ridge regression and with f_regression, -0.2688 top1
superbowl rmse score for Lasso regression, -0.8655
superbowl rmse score for Lasso regression and with mutual_info_regression, -0.8658 top1
superbowl rmse score for Lasso regression and with f_regression, -0.8655 top1
superbowl rmse score for linear regression, -0.4186
superbowl rmse score for linear regression and with mutual_info_regression, -0.2709 top2
superbowl rmse score for linear regression and with f_regression, -0.4604 top2
superbowl rmse score for Ridge regression, -0.3371
superbowl rmse score for Ridge regression and with mutual_info_regression, -0.3325 top2
```

```
C:\Users\wenxin cheng\AppData\Roaming\Python\Python38\site-packages\sklearn\utils\valida
tion.py:993: DataConversionWarning: A column-vector y was passed when a 1d array was exp
ected. Please change the shape of y to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
C:\Users\wenxin cheng\AppData\Roaming\Python\Python38\site-packages\sklearn\utils\valida
tion.py:993: DataConversionWarning: A column-vector y was passed when a 1d array was exp
ected. Please change the shape of y to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
```

```
superbowl rmse score for Ridge regression and with f_regression, -0.2641 top2
superbowl rmse score for Lasso regression, -0.8655
superbowl rmse score for Lasso regression and with mutual_info_regression, -0.8658 top2
superbowl rmse score for Lasso regression and with f_regression, -0.8655 top2
superbowl rmse score for linear regression, -0.4186
superbowl rmse score for linear regression and with mutual_info_regression, -0.1658 top3
superbowl rmse score for linear regression and with f_regression, -0.2535 top3
superbowl rmse score for Ridge regression, -0.3371
superbowl rmse score for Ridge regression and with mutual_info_regression, -0.2939 top3
superbowl rmse score for Ridge regression and with f_regression, -0.2793 top3
superbowl rmse score for Lasso regression, -0.8655
superbowl rmse score for Lasso regression and with mutual_info_regression, -0.8655 top3
superbowl rmse score for Lasso regression and with f_regression, -0.8655 top3
superbowl rmse score for linear regression, -0.4186
superbowl rmse score for linear regression and with mutual_info_regression, -0.3885 top4
```

```
C:\Users\wenxin cheng\AppData\Roaming\Python\Python38\site-packages\sklearn\utils\valida
tion.py:993: DataConversionWarning: A column-vector y was passed when a 1d array was exp
ected. Please change the shape of y to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
C:\Users\wenxin cheng\AppData\Roaming\Python\Python38\site-packages\sklearn\utils\valida
tion.py:993: DataConversionWarning: A column-vector y was passed when a 1d array was exp
ected. Please change the shape of y to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
C:\Users\wenxin cheng\AppData\Roaming\Python\Python38\site-packages\sklearn\utils\valida
tion.py:993: DataConversionWarning: A column-vector y was passed when a 1d array was exp
ected. Please change the shape of y to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
C:\Users\wenxin cheng\AppData\Roaming\Python\Python38\site-packages\sklearn\utils\valida
tion.py:993: DataConversionWarning: A column-vector y was passed when a 1d array was exp
ected. Please change the shape of y to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
```

```
superbowl rmse score for linear regression and with f_regression, -0.2629 top4
superbowl rmse score for Ridge regression, -0.3371
superbowl rmse score for Ridge regression and with mutual_info_regression, -0.2995 top4
superbowl rmse score for Ridge regression and with f_regression, -0.2818 top4
superbowl rmse score for Lasso regression, -0.8655
superbowl rmse score for Lasso regression and with mutual_info_regression, -0.8655 top4
superbowl rmse score for Lasso regression and with f_regression, -0.8655 top4
superbowl rmse score for linear regression, -0.4186
superbowl rmse score for linear regression and with mutual_info_regression, -0.3861 top5
superbowl rmse score for linear regression and with f_regression, -0.2508 top5
superbowl rmse score for Ridge regression, -0.3371
superbowl rmse score for Ridge regression and with mutual_info_regression, -0.2606 top5
superbowl rmse score for Ridge regression and with f_regression, -0.3101 top5
```

```
C:\Users\wenxin cheng\AppData\Roaming\Python\Python38\site-packages\sklearn\utils\valida
tion.py:993: DataConversionWarning: A column-vector y was passed when a 1d array was exp
ected. Please change the shape of y to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
```

```
C:\Users\wenxin cheng\AppData\Roaming\Python\Python38\site-packages\sklearn\utils\validation.py:993: DataConversionWarning: A column-vector y was passed when a 1d array was expected. Please change the shape of y to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
superbowl rmse score for Lasso regression, -0.8655
superbowl rmse score for Lasso regression and with mutual_info_regression, -0.8655 top5
superbowl rmse score for Lasso regression and with f_regression, -0.8655 top5
superbowl rmse score for linear regression, -0.4186
superbowl rmse score for linear regression and with mutual_info_regression, -0.3272 top6
superbowl rmse score for linear regression and with f_regression, -0.4688 top6
superbowl rmse score for Ridge regression, -0.3371
superbowl rmse score for Ridge regression and with mutual_info_regression, -0.2677 top6
superbowl rmse score for Ridge regression and with f_regression, -0.3310 top6
superbowl rmse score for Lasso regression, -0.8655
superbowl rmse score for Lasso regression and with mutual_info_regression, -0.8655 top6
superbowl rmse score for Lasso regression and with f_regression, -0.8655 top6
C:\Users\wenxin cheng\AppData\Roaming\Python\Python38\site-packages\sklearn\utils\validation.py:993: DataConversionWarning: A column-vector y was passed when a 1d array was expected. Please change the shape of y to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
C:\Users\wenxin cheng\AppData\Roaming\Python\Python38\site-packages\sklearn\utils\validation.py:993: DataConversionWarning: A column-vector y was passed when a 1d array was expected. Please change the shape of y to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
C:\Users\wenxin cheng\AppData\Roaming\Python\Python38\site-packages\sklearn\utils\validation.py:993: DataConversionWarning: A column-vector y was passed when a 1d array was expected. Please change the shape of y to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
C:\Users\wenxin cheng\AppData\Roaming\Python\Python38\site-packages\sklearn\utils\validation.py:993: DataConversionWarning: A column-vector y was passed when a 1d array was expected. Please change the shape of y to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
superbowl rmse score for linear regression, -0.4186
superbowl rmse score for linear regression and with mutual_info_regression, -0.2706 top7
superbowl rmse score for linear regression and with f_regression, -0.3139 top7
superbowl rmse score for Ridge regression, -0.3371
superbowl rmse score for Ridge regression and with mutual_info_regression, -0.2777 top7
superbowl rmse score for Ridge regression and with f_regression, -0.3307 top7
superbowl rmse score for Lasso regression, -0.8655
superbowl rmse score for Lasso regression and with mutual_info_regression, -0.8655 top7
superbowl rmse score for Lasso regression and with f_regression, -0.8655 top7
superbowl rmse score for linear regression, -0.4186
superbowl rmse score for linear regression and with mutual_info_regression, -0.4858 top8
superbowl rmse score for linear regression and with f_regression, -0.2909 top8
superbowl rmse score for Ridge regression, -0.3371
C:\Users\wenxin cheng\AppData\Roaming\Python\Python38\site-packages\sklearn\utils\validation.py:993: DataConversionWarning: A column-vector y was passed when a 1d array was expected. Please change the shape of y to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
C:\Users\wenxin cheng\AppData\Roaming\Python\Python38\site-packages\sklearn\utils\validation.py:993: DataConversionWarning: A column-vector y was passed when a 1d array was expected. Please change the shape of y to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
superbowl rmse score for Ridge regression and with mutual_info_regression, -0.3213 top8
superbowl rmse score for Ridge regression and with f_regression, -0.3074 top8
superbowl rmse score for Lasso regression, -0.8655
superbowl rmse score for Lasso regression and with mutual_info_regression, -0.8655 top8
superbowl rmse score for Lasso regression and with f_regression, -0.8655 top8
superbowl rmse score for linear regression, -0.4186
superbowl rmse score for linear regression and with mutual_info_regression, -0.4186 top9
superbowl rmse score for linear regression and with f_regression, -0.4186 top9
superbowl rmse score for Ridge regression, -0.3371
superbowl rmse score for Ridge regression and with mutual_info_regression, -0.3371 top9
superbowl rmse score for Ridge regression and with f_regression, -0.3371 top9
superbowl rmse score for Lasso regression, -0.8655
```

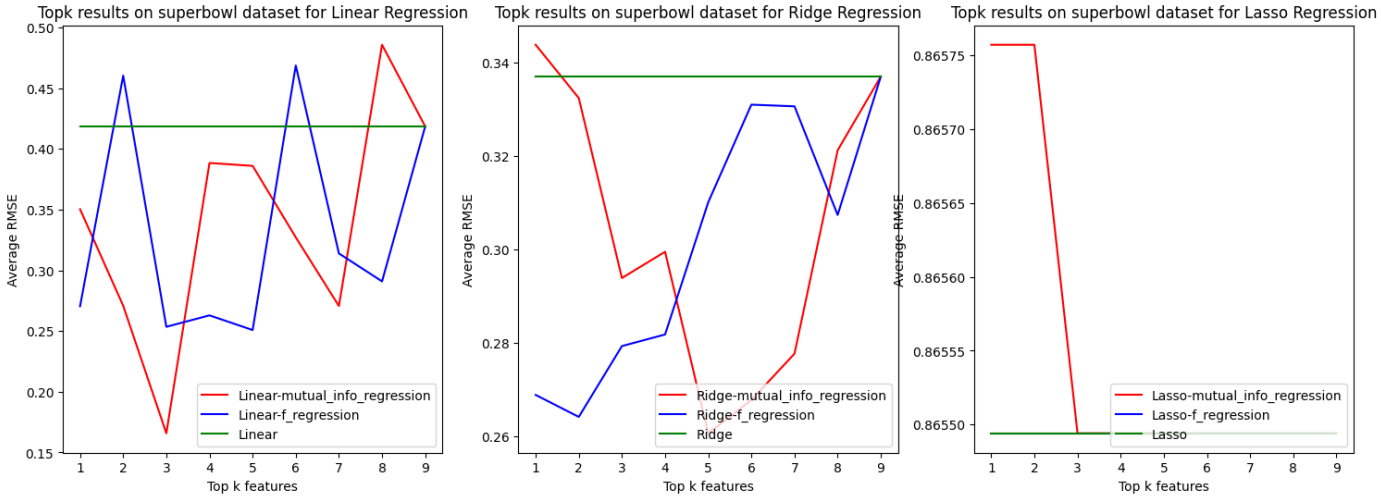
```
superbowl rmse score for Lasso regression and with mutual_info_regression, -0.8655 top9
superbowl rmse score for Lasso regression and with f_regression, -0.8655 top9
```

In [78]:

```
import matplotlib.pyplot as plt
fig, axes = plt.subplots(1, 3, figsize=(18, 6))

#plot superbowl linear
axes[0].plot(np.arange(1, len(superbowl_rmse_lr_m) + 1, 1), np.negative(superbowl_rmse_l
axes[0].plot(np.arange(1, len(superbowl_rmse_lr_f) + 1, 1), np.negative(superbowl_rmse_l
axes[0].plot(np.arange(1, len(superbowl_rmse_lr) + 1, 1), np.negative(superbowl_rmse_lr)

axes[0].legend(loc='lower right')
axes[0].set(xlabel='Top k features',ylabel='Average RMSE')
axes[0].set_title('Topk results on superbowl dataset for Linear Regression')

#plot superbowl Ridge
axes[1].plot(np.arange(1, len(superbowl_rmse_r_m) + 1, 1), np.negative(superbowl_rmse_r_
axes[1].plot(np.arange(1, len(superbowl_rmse_r_f) + 1, 1), np.negative(superbowl_rmse_r_
axes[1].plot(np.arange(1, len(superbowl_rmse_r) + 1, 1), np.negative(superbowl_rmse_r),

axes[1].legend(loc='lower right')
axes[1].set(xlabel='Top k features',ylabel='Average RMSE')
axes[1].set_title('Topk results on superbowl dataset for Ridge Regression')

#plot superbowl Lasso
axes[2].plot(np.arange(1, len(superbowl_rmse_la_m) + 1, 1), np.negative(superbowl_rmse_l
axes[2].plot(np.arange(1, len(superbowl_rmse_la_f) + 1, 1), np.negative(superbowl_rmse_l
axes[2].plot(np.arange(1, len(superbowl_rmse_la) + 1, 1), np.negative(superbowl_rmse_la)

axes[2].legend(loc='lower right')
axes[2].set(xlabel='Top k features',ylabel='Average RMSE')
axes[2].set_title('Topk results on superbowl dataset for Lasso Regression')
```

Out[78]:

```
Text(0.5, 1.0, 'Topk results on superbowl dataset for Lasso Regression')
```



Use Cross Validate to find the optimal estimator and parameters

In [79]:

```
# Finding the optimal penalty parameter
from joblib import Memory
from sklearn.pipeline import Pipeline

location = "cachedir"
memory = Memory(location=location, verbose=10)

pipe_ = Pipeline([
    ('kbest', SelectKBest()),
    ('model', "passthrough")
], memory = memory)
```

```
param_grid = [{
        'kbest__score_func': (mutual_info_regression, f_regression),
        'kbest__k': (1, 2, 3, 4, 5, 6, 7, 8, 9),
        'model': [Ridge(), Lasso()],
        'model__alpha': [10.0**x for x in np.arange(-3,4)]
    }
]
```

In [80]:
```
grid_superbowl = GridSearchCV(pipe_, param_grid = param_grid, cv = 10, n_jobs = -1, verb
                     scoring = 'neg_root_mean_squared_error', return_train_score = True)
```

```
Fitting 10 folds for each of 252 candidates, totalling 2520 fits

________________________________________________________________________________
[Memory] Calling sklearn.pipeline._fit_transform_one...
_fit_transform_one(SelectKBest(k=7, score_func=<function f_regression at 0x000002174FAD6
C10>),     num_retweet  num_followers  ranking_score  user_activity  user_mentions  \
0      0.709762       1.483264       0.958372       1.423014       1.418241
1      1.190321       0.808309       0.610127       0.838521       1.174154
2      0.828514       1.490113       1.222343       2.046335       1.389269
3      1.420657       1.787576       2.258981       1.841310       1.140836
4      3.681958       3.224027       3.124603       2.501960       2.580734
5      1.091963       0.590189       1.036742       0.695530       1.641324
6      0.503246       0.189436       0.529063       0.673372       0.924634
7     -0.015152      -0.112757       0.123819       0.335083       0...,
    num_tweet
1    0.608591
2    1.208436
3    2.231415
4    3.136456
5    1.067655
6    0.542523
7    0.145165
8   -0.086032
9   -0.590146
10  -0.626947
11  -0.625202
12  -0.545969
13  -0.549538
14  -0.581263
15  -0.638527
16  -0.637417
17  -0.639875
18  -0.639796
19  -0.629406
20  -0.603471
21  -0.607912
22  -0.611164
23  -0.616240
24  -0.650107,
None, message_clsname='Pipeline', message=None)
_____________________________________________fit_transform_one - 0.0s, 0.0min
C:\Users\wenxin cheng\AppData\Roaming\Python\Python38\site-packages\sklearn\utils\valida
tion.py:993: DataConversionWarning: A column-vector y was passed when a 1d array was exp
ected. Please change the shape of y to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
```

In [85]:
```
def _print_gridsearch_result(model, title):
    # print(model.cv_results_.keys())
    # print(f"Best estimator for {title}: ", model.best_estimator_)
    print(f"Best parameters for {title}: ", model.best_params_)
    print(f"Best score for {title}: ", model.best_score_)
```

In [86]:
```
_print_gridsearch_result(grid_superbowl, "superbowl with select k best and Ridge/Lasso R
```

```
Best parameters for superbowl with select k best and Ridge/Lasso Regression:  {'kbest__
```

```
k': 7, 'kbest__score_func': <function f_regression at 0x000002174FAD6C10>, 'model': Ridg
e(alpha=0.001), 'model__alpha': 0.001}
Best score for superbowl with select k best and Ridge/Lasso Regression:  -0.181914159173
66846
```

In [90]:
```
superbowl_top7 = SelectKBest(score_func = f_regression, k = 7).fit_transform(superbowl_m
```

```
C:\Users\wenxin cheng\AppData\Roaming\Python\Python38\site-packages\sklearn\utils\valida
tion.py:993: DataConversionWarning: A column-vector y was passed when a 1d array was exp
ected. Please change the shape of y to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
```

polynomial regression with Ridge regularization

In [92]:
```
# a pipeline for polynomial regression with Ridge regularization
location = "cachedir"
memory = Memory(location=location, verbose=10)
from sklearn.preprocessing import StandardScaler, PolynomialFeatures

poly_pipe_superbowl_ = Pipeline([
    ('poly_transform', PolynomialFeatures()),
    ('model', Ridge(alpha=0.001))
],memory=memory)

poly_param_= {
    'poly_transform__degree': np.arange(1,10,1)
}
```

In [93]:
```
poly_superbowl = GridSearchCV(poly_pipe_superbowl_ , param_grid=poly_param_, cv=10, n_jo
```

```
Fitting 10 folds for each of 9 candidates, totalling 90 fits
_____________________________________________________________________________________
[Memory] Calling sklearn.pipeline._fit_transform_one...
_fit_transform_one(PolynomialFeatures(degree=1), array([[ 1.483264, ...,  0.906954],
       ...,
       [-0.681962, ..., -0.62034 ]]),
    num_tweet
1    0.608591
2    1.208436
3    2.231415
4    3.136456
5    1.067655
6    0.542523
7    0.145165
8   -0.086032
9   -0.590146
10  -0.626947
11  -0.625202
12  -0.545969
13  -0.549538
14  -0.581263
15  -0.638527
16  -0.637417
17  -0.639875
18  -0.639796
19  -0.629406
20  -0.603471
21  -0.607912
22  -0.611164
23  -0.616240
24  -0.650107,
None, message_clsname='Pipeline', message=None)
_____________________________________________________fit_transform_one - 0.0s, 0.0min
```

In [94]:
```
_print_gridsearch_result(poly_superbowl, "PolynomialFeatures of Superbowl" )
```

```
Best parameters for PolynomialFeatures of Superbowl:  {'poly_transform__degree': 1}
Best score for PolynomialFeatures of Superbowl:  -0.3822844598287635
```

In [103…
```
s_d = SelectKBest(score_func = f_regression, k = 7)
s_d.fit_transform(superbowl_merge_data_standard_x, superbowl_merge_data_standard_y)
column_s = superbowl_merge_data_standard_x.columns[s_d.get_support()].tolist()

d_params = poly_superbowl.best_estimator_.get_params()
d_coefs = d_params['model'].coef_
d_names = d_params['poly_transform'].get_feature_names(column_s)

d_sorted_indice = np.argsort(-abs(d_coefs))[0]
# print(d_names)
# print(d_sorted_indice)
salient_features =[d_names[i] for i in d_sorted_indice[:7]]
print ('Top 7 Salient features (superbowl) in order:',salient_features)
```

```
Top 7 Salient features (superbowl) in order: ['num_neutral', 'unique_user_id', 'ranking_
score', 'user_activity', 'num_followers', 'user_mentions', 'num_negative']
```

```
C:\Users\wenxin cheng\AppData\Roaming\Python\Python38\site-packages\sklearn\utils\valida
tion.py:993: DataConversionWarning: A column-vector y was passed when a 1d array was exp
ected. Please change the shape of y to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
C:\Users\wenxin cheng\AppData\Roaming\Python\Python38\site-packages\sklearn\utils\deprec
ation.py:87: FutureWarning: Function get_feature_names is deprecated; get_feature_names
is deprecated in 1.0 and will be removed in 1.2. Please use get_feature_names_out instea
d.
  warnings.warn(msg, category=FutureWarning)
```

Multi-layer perceptron (MLP) regression

In [105…
```
from sklearn.neural_network import MLPRegressor

mlpr = Pipeline([
    ('model', MLPRegressor()),

], memory=memory)

param_list = {
    "model__hidden_layer_sizes": [(30, 40), (30, 60), (40, 40), (40, 60), (60, 60)],
    "model__activation": ["identity", "logistic", "tanh", "relu"],
    "model__solver": ["lbfgs", "sgd", "adam"],
}

grid_superbowl_mlp = GridSearchCV(mlpr, param_grid = param_list, cv = 5, n_jobs = -1, ve
                      scoring = 'neg_root_mean_squared_error', return_train_score = True)
```

```
Fitting 5 folds for each of 60 candidates, totalling 300 fits
```

```
C:\Users\wenxin cheng\AppData\Roaming\Python\Python38\site-packages\sklearn\neural_netwo
rk\_multilayer_perceptron.py:1599: DataConversionWarning: A column-vector y was passed w
hen a 1d array was expected. Please change the shape of y to (n_samples, ), for example
using ravel().
  y = column_or_1d(y, warn=True)
C:\Users\wenxin cheng\AppData\Roaming\Python\Python38\site-packages\sklearn\neural_netwo
rk\_multilayer_perceptron.py:692: ConvergenceWarning: Stochastic Optimizer: Maximum iter
ations (200) reached and the optimization hasn't converged yet.
  warnings.warn(
```

In [106…
```
_print_gridsearch_result(grid_superbowl_mlp, "Neural Network of Superbowl" )
rmse = np.sqrt(-grid_superbowl_mlp.best_score_)
print(f"rmse = {rmse}")
```

```
Best parameters for Neural Network of Superbowl:  {'model__activation': 'identity', 'mod
el__hidden_layer_sizes': (60, 60), 'model__solver': 'sgd'}
```

```
Best score for Neural Network of Superbowl:  -0.3281072454469807
rmse = 0.5728064642154282
```

RandomForest Regression

In [107…
```
from sklearn.ensemble import RandomForestRegressor
pipeline_forest = Pipeline([
    ('model', RandomForestRegressor())
],memory=memory)

param_grid_forest = {
    'model__max_features': np.arange(1, 5, 1),
    'model__n_estimators': np.arange(10, 40, 10),
    'model__max_depth': np.arange(1, 5, 1)
}
```

In [108…
```
grid_superbowl_forest = GridSearchCV(pipeline_forest, param_grid = param_grid_forest, cv
                      scoring = 'neg_root_mean_squared_error', return_train_score = True)
```

```
Fitting 10 folds for each of 48 candidates, totalling 480 fits
C:\Users\wenxin cheng\AppData\Roaming\Python\Python38\site-packages\sklearn\pipeline.py:
394: DataConversionWarning: A column-vector y was passed when a 1d array was expected. P
lease change the shape of y to (n_samples,), for example using ravel().
  self._final_estimator.fit(Xt, y, **fit_params_last_step)
```

In [109…
```
_print_gridsearch_result(grid_superbowl_forest, "RandomForestRegressor for superbowl")
```

```
Best parameters for RandomForestRegressor for superbowl:  {'model__max_depth': 2, 'model
__max_features': 2, 'model__n_estimators': 20}
Best score for RandomForestRegressor for superbowl:  -0.23065258627850366
```

In [111…
```
rf_superbowl_ = RandomForestRegressor(n_estimators=20, max_features=2, max_depth=2, oob_
rf_superbowl_.fit(superbowl_top7, superbowl_merge_data_standard_y)

print('Best Random Forest Model for superbowl Dataset:')
print('OOB score: %.4f' %(rf_superbowl_.oob_score_))
print('R^2 score: %.4f' %(rf_superbowl_.score(superbowl_top7, superbowl_merge_data_stand
```

```
Best Random Forest Model for superbowl Dataset:
OOB score: 0.5292
R^2 score: 0.9046
C:\Users\wenxin cheng\AppData\Local\Temp\ipykernel_17032\1150329333.py:2: DataConversion
Warning: A column-vector y was passed when a 1d array was expected. Please change the sh
ape of y to (n_samples,), for example using ravel().
  rf_superbowl_.fit(superbowl_top7, superbowl_merge_data_standard_y)
```

LightGBM gradient boosting Regression

In [112…
```
### LightGBM
import lightgbm as lgb

lbg_pip_ = Pipeline([
    ('model', lgb.LGBMRegressor())
],memory = memory)

param_lgb_ = {
    'model__num_leaves': [7, 14, 21, 28, 31, 50],
    'model__learning_rate': [0.1, 0.03, 0.003],
    'model__max_depth': [-1, 3, 5],
    'model__n_estimators': [50, 100, 200, 500],
}
```

Bayesian optimization

In [113…

```
from skopt import BayesSearchCV


lg_superbowl = BayesSearchCV(lbg_pip_, search_spaces=param_lgb_, cv=10, n_jobs=-1, verbo
    scoring='neg_root_mean_squared_error', return_train_score=True).fit(superbowl_top7,
```

```
Fitting 10 folds for each of 1 candidates, totalling 10 fits
Fitting 10 folds for each of 1 candidates, totalling 10 fits
Fitting 10 folds for each of 1 candidates, totalling 10 fits
Fitting 10 folds for each of 1 candidates, totalling 10 fits
Fitting 10 folds for each of 1 candidates, totalling 10 fits
Fitting 10 folds for each of 1 candidates, totalling 10 fits
Fitting 10 folds for each of 1 candidates, totalling 10 fits
Fitting 10 folds for each of 1 candidates, totalling 10 fits
Fitting 10 folds for each of 1 candidates, totalling 10 fits
Fitting 10 folds for each of 1 candidates, totalling 10 fits
Fitting 10 folds for each of 1 candidates, totalling 10 fits
Fitting 10 folds for each of 1 candidates, totalling 10 fits
Fitting 10 folds for each of 1 candidates, totalling 10 fits
Fitting 10 folds for each of 1 candidates, totalling 10 fits
Fitting 10 folds for each of 1 candidates, totalling 10 fits
Fitting 10 folds for each of 1 candidates, totalling 10 fits
Fitting 10 folds for each of 1 candidates, totalling 10 fits
Fitting 10 folds for each of 1 candidates, totalling 10 fits
Fitting 10 folds for each of 1 candidates, totalling 10 fits
Fitting 10 folds for each of 1 candidates, totalling 10 fits
Fitting 10 folds for each of 1 candidates, totalling 10 fits
Fitting 10 folds for each of 1 candidates, totalling 10 fits
Fitting 10 folds for each of 1 candidates, totalling 10 fits
d:\Anaconda3\envs\py38\lib\site-packages\skopt\optimizer\optimizer.py:449: UserWarning:
The objective has been evaluated at this point before.
  warnings.warn("The objective has been evaluated "
Fitting 10 folds for each of 1 candidates, totalling 10 fits
d:\Anaconda3\envs\py38\lib\site-packages\skopt\optimizer\optimizer.py:449: UserWarning:
The objective has been evaluated at this point before.
  warnings.warn("The objective has been evaluated "
Fitting 10 folds for each of 1 candidates, totalling 10 fits
Fitting 10 folds for each of 1 candidates, totalling 10 fits
Fitting 10 folds for each of 1 candidates, totalling 10 fits
Fitting 10 folds for each of 1 candidates, totalling 10 fits
Fitting 10 folds for each of 1 candidates, totalling 10 fits
Fitting 10 folds for each of 1 candidates, totalling 10 fits
Fitting 10 folds for each of 1 candidates, totalling 10 fits
Fitting 10 folds for each of 1 candidates, totalling 10 fits
Fitting 10 folds for each of 1 candidates, totalling 10 fits
Fitting 10 folds for each of 1 candidates, totalling 10 fits
Fitting 10 folds for each of 1 candidates, totalling 10 fits
Fitting 10 folds for each of 1 candidates, totalling 10 fits
Fitting 10 folds for each of 1 candidates, totalling 10 fits
Fitting 10 folds for each of 1 candidates, totalling 10 fits
d:\Anaconda3\envs\py38\lib\site-packages\skopt\optimizer\optimizer.py:449: UserWarning:
The objective has been evaluated at this point before.
  warnings.warn("The objective has been evaluated "
Fitting 10 folds for each of 1 candidates, totalling 10 fits
d:\Anaconda3\envs\py38\lib\site-packages\skopt\optimizer\optimizer.py:449: UserWarning:
The objective has been evaluated at this point before.
  warnings.warn("The objective has been evaluated "
Fitting 10 folds for each of 1 candidates, totalling 10 fits
Fitting 10 folds for each of 1 candidates, totalling 10 fits
Fitting 10 folds for each of 1 candidates, totalling 10 fits
Fitting 10 folds for each of 1 candidates, totalling 10 fits
Fitting 10 folds for each of 1 candidates, totalling 10 fits
Fitting 10 folds for each of 1 candidates, totalling 10 fits
d:\Anaconda3\envs\py38\lib\site-packages\skopt\optimizer\optimizer.py:449: UserWarning:
```

```
The objective has been evaluated at this point before.
  warnings.warn("The objective has been evaluated "
Fitting 10 folds for each of 1 candidates, totalling 10 fits
Fitting 10 folds for each of 1 candidates, totalling 10 fits
d:\Anaconda3\envs\py38\lib\site-packages\skopt\optimizer\optimizer.py:449: UserWarning:
The objective has been evaluated at this point before.
  warnings.warn("The objective has been evaluated "
Fitting 10 folds for each of 1 candidates, totalling 10 fits
Fitting 10 folds for each of 1 candidates, totalling 10 fits
d:\Anaconda3\envs\py38\lib\site-packages\skopt\optimizer\optimizer.py:449: UserWarning:
The objective has been evaluated at this point before.
  warnings.warn("The objective has been evaluated "
Fitting 10 folds for each of 1 candidates, totalling 10 fits
C:\Users\wenxin cheng\AppData\Roaming\Python\Python38\site-packages\sklearn\utils\valida
tion.py:993: DataConversionWarning: A column-vector y was passed when a 1d array was exp
ected. Please change the shape of y to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
```

In [114…

```
_print_gridsearch_result(lg_superbowl, "LightGBM of superbowls" )
```

```
Best parameters for LightGBM of superbowls:  OrderedDict([('model__learning_rate', 0.0
3), ('model__max_depth', -1), ('model__n_estimators', 500), ('model__num_leaves', 31)])
Best score for LightGBM of superbowls:  -0.8657568961600333
```

Try the 1 minutes interval

In [58]:

```
# This function takes in a train object and a feature object, and merges the feature obj
def merge_feature_into_train_data(train_obj, feature):
    train_obj["num_tweet"] += feature["num_tweet"]
    train_obj["num_retweet"] += feature["num_retweet"]
    train_obj["num_followers"] += feature["num_followers"]
    train_obj["ranking_score"] += feature["ranking_score"]
    train_obj["user_activity"] += feature["user_activity"]
    train_obj["user_id"].add(feature["user_id"])
    train_obj["user_location"] += feature["user_location"]
    train_obj["user_mentions"] += feature["user_mentions"]
    train_obj["num_positive"] += feature["positive"]
    train_obj["num_neutral"] += feature["neutral"]
    train_obj["num_negative"] += feature["negative"]
    train_obj["unique_user_id"] = len(train_obj["user_id"])


def merge_different_intervals(): # for SuperBowl
    time_delta = int(timedelta(minutes = 1).seconds)

    num_train_data = (superbowl_end_time - superbowl_start_time) // time_delta + 1

    train_data = [{
        "range_start": superbowl_start_time + i*time_delta,
        "range_end": superbowl_start_time + (i+1)*time_delta,
        "num_tweet": 0,
        "num_retweet": 0,
        "num_followers": 0,
        "ranking_score": 0,
        "user_activity": 0,
        "user_id": set(),
        "user_location": "",
        "user_mentions": 0,
        "num_positive": 0,
        "num_neutral": 0,
        "num_negative": 0,
        "text": "",
        "polarity": 0
    }
```

```
    for i in range(num_train_data)]

    for idx, row in superbowl_data.iterrows():
        # Calculate which time slot this tweet falls into
        if superbowl_start_time <= row["created_at"] <= superbowl_end_time:
            index = (row["created_at"]- superbowl_start_time) // time_delta
            # Merge the feature into the appropriate slot in the training data
            merge_feature_into_train_data(train_data[index], row)
    return pd.DataFrame(train_data)
```

In [59]:
```
superbowl_merge_1 = merge_different_intervals()
superbowl_merge_1.to_csv('superbowl_merge_1.csv', index=False)
```

In [61]:
```
superbowl_merge_1.shape
```

Out[61]:
```
(241, 16)
```

In [62]:
```
superbowl_merge_data_drop = superbowl_merge_1.copy().drop(['range_start','range_end','us
```

In [63]:
```
superbowl_x = superbowl_merge_data_drop.drop(superbowl_merge_data_drop.index[-1])
superbowl_y = superbowl_merge_1["num_tweet"]
superbowl_y = superbowl_y.drop(superbowl_y.index[0])
superbowl_y = pd.DataFrame(superbowl_y,columns = ["num_tweet"]).values.ravel()
```

In [64]:
```
superbowl_x
```

Out[64]:

	num_retweet	num_followers	ranking_score	user_activity	user_mentions	num_positive	num_neutr
0	4071	6320414.0	7796.709831	22453.355038	579	480	10
1	3034	20295517.0	6476.715966	25663.899065	497	366	8
2	4334	18597858.0	5822.767599	19172.983770	432	383	7
3	3440	23712225.0	6849.486202	24109.103574	481	432	8
4	3320	19579440.0	9386.665890	26187.226104	533	407	12
...	...	...	...	...	...	...	
235	63	101773.0	187.321615	721.504420	19	20	
236	100	131757.0	253.560633	1591.937729	37	23	
237	65	135456.0	226.558879	2228.397522	31	15	
238	57	36182.0	149.703310	422.127708	13	9	
239	102	104743.0	212.989823	1444.595889	16	14	

240 rows × 9 columns

In [66]:
```
# RandomForest GridSearch
import pandas as pd
from sklearn.pipeline import Pipeline
from sklearn.model_selection import KFold, GridSearchCV
from sklearn.preprocessing import StandardScaler
from sklearn.ensemble import RandomForestRegressor


pipe_rf = Pipeline([
    ('standardize', StandardScaler()),
    ('model', RandomForestRegressor(random_state=42))
])
```

```
param_grid = {
    'model__max_depth': [10, 30, 50, 70, 100, 200],
    'model__max_features': ['auto', 'sqrt'],
    'model__min_samples_leaf': [1, 2, 3],
    'model__min_samples_split': [2, 5, 10],
    'model__n_estimators': [200, 400]
}

grid_rf = GridSearchCV(pipe_rf, param_grid=param_grid, cv=KFold(5, shuffle=True, random_
                       scoring='neg_mean_squared_error').fit(superbowl_x, superbowl_y)
result_rf = pd.DataFrame(grid_rf.cv_results_)[['mean_test_score', 'param_model__max_dept
                                               'param_model__min_samples_leaf', 'param_mod
                                               'param_model__n_estimators']]
result_rf = result_rf.sort_values(by=['mean_test_score'], ascending=False).reset_index(d
result_rf.head()
```

```
Fitting 5 folds for each of 216 candidates, totalling 1080 fits
```

Out[66]:

	mean_test_score	param_model__max_depth	param_model__max_features	param_model__min_samples
0	-161354.634897	10	auto	
1	-162792.741378	10	auto	
2	-163112.769889	30	auto	
3	-163112.769889	200	auto	
4	-163112.769889	70	auto	

In [90]:

```
# GradientBoosting GridSearch
import pandas as pd
from sklearn.pipeline import Pipeline
from sklearn.model_selection import KFold, GridSearchCV
from sklearn.preprocessing import StandardScaler
from sklearn.ensemble import GradientBoostingRegressor


pipe_gb = Pipeline([
    ('standardize', StandardScaler()),
    ('model', GradientBoostingRegressor(random_state=42))
])

param_grid = {
    'model__max_depth': [10, 30, 50, 70, 100, 200],
    'model__max_features': ['auto', 'sqrt'],
    'model__min_samples_leaf': [1, 2, 3],
    'model__min_samples_split': [2, 5, 10],
    'model__n_estimators': [200, 400]
}


grid_gb = GridSearchCV(pipe_gb, param_grid=param_grid, cv=KFold(5, shuffle=True, random_
                       scoring='neg_mean_squared_error').fit(superbowl_x, superbowl_y)
result_gb = pd.DataFrame(grid_gb.cv_results_)[['mean_test_score', 'param_model__max_dept
                                               'param_model__min_samples_leaf', 'param_mod
                                               'param_model__n_estimators']]
result_gb = result_gb.sort_values(by=['mean_test_score'], ascending=False).reset_index(d
result_gb.head()
```

```
Fitting 5 folds for each of 216 candidates, totalling 1080 fits
```

Out[90]:

	mean_test_score	param_model__max_depth	param_model__max_features	param_model__min_samples
0	-137558.444653	50	sqrt	
1	-137558.444653	100	sqrt	
2	-137558.444653	70	sqrt	

3	-137558.444653	200	sqrt
4	-137558.459325	30	sqrt

In [92]:
```
# NeuralNetwork GridSearch

import pandas as pd
from sklearn.pipeline import Pipeline
from sklearn.model_selection import KFold, GridSearchCV
from sklearn.preprocessing import StandardScaler
from sklearn.neural_network import MLPRegressor

pipe_nn_noscale = Pipeline([
    ('model', MLPRegressor(random_state=42, max_iter=2000))
])

param_grid = {
    'model__hidden_layer_sizes': [(x,y) for x in np.arange(1, 51) for y in np.arange(1,
}


grid_nn_noscale = GridSearchCV(pipe_nn_noscale, param_grid=param_grid, cv=KFold(5, shuff
                               scoring='neg_mean_squared_error').fit(superbowl_x, superb
result_nn_noscale = pd.DataFrame(grid_nn_noscale.cv_results_)[['mean_test_score', 'param
result_nn_noscale = result_nn_noscale.sort_values(by=['mean_test_score'], ascending=Fals
result_nn_noscale.head()
```

```
Fitting 5 folds for each of 2500 candidates, totalling 12500 fits
```

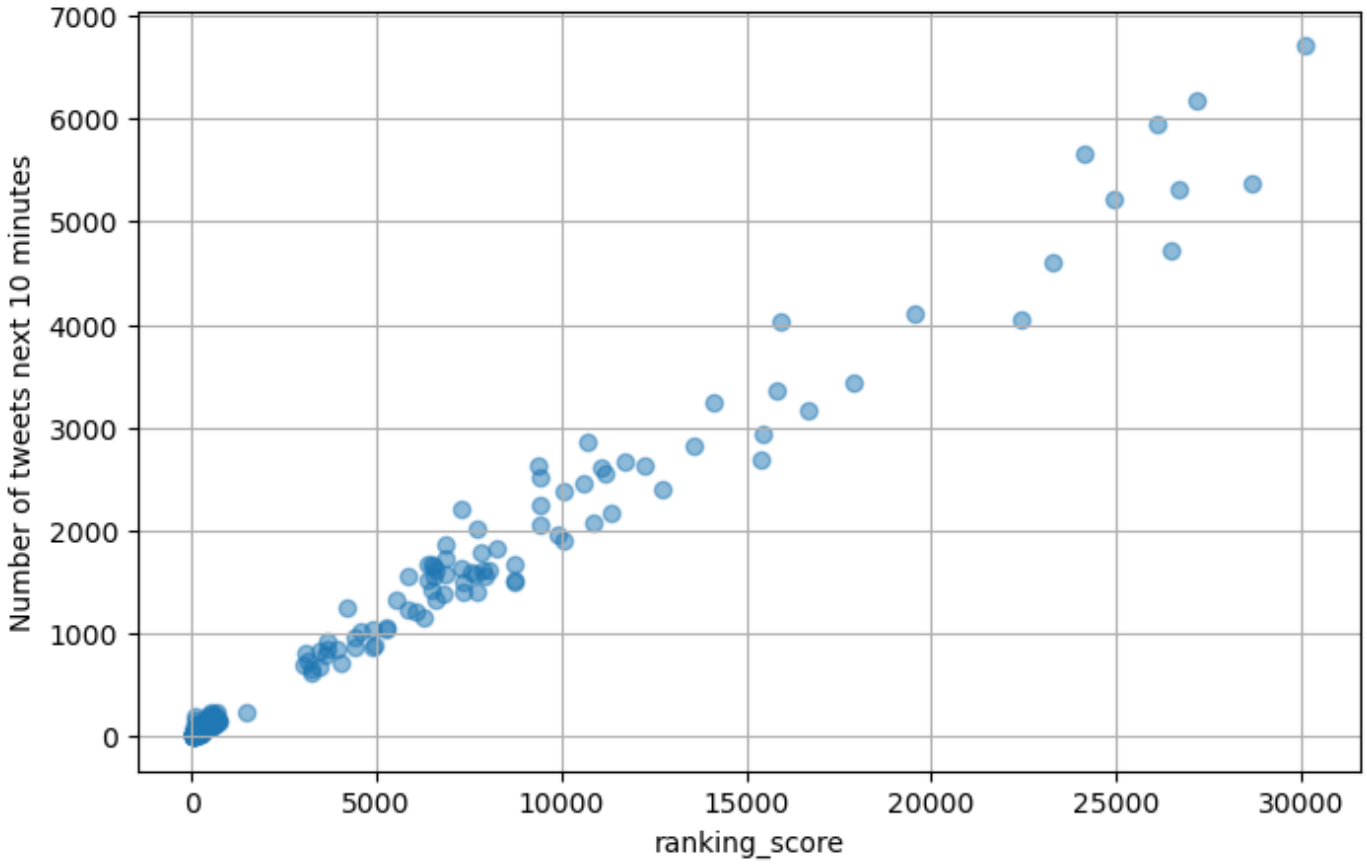
Out[92]:

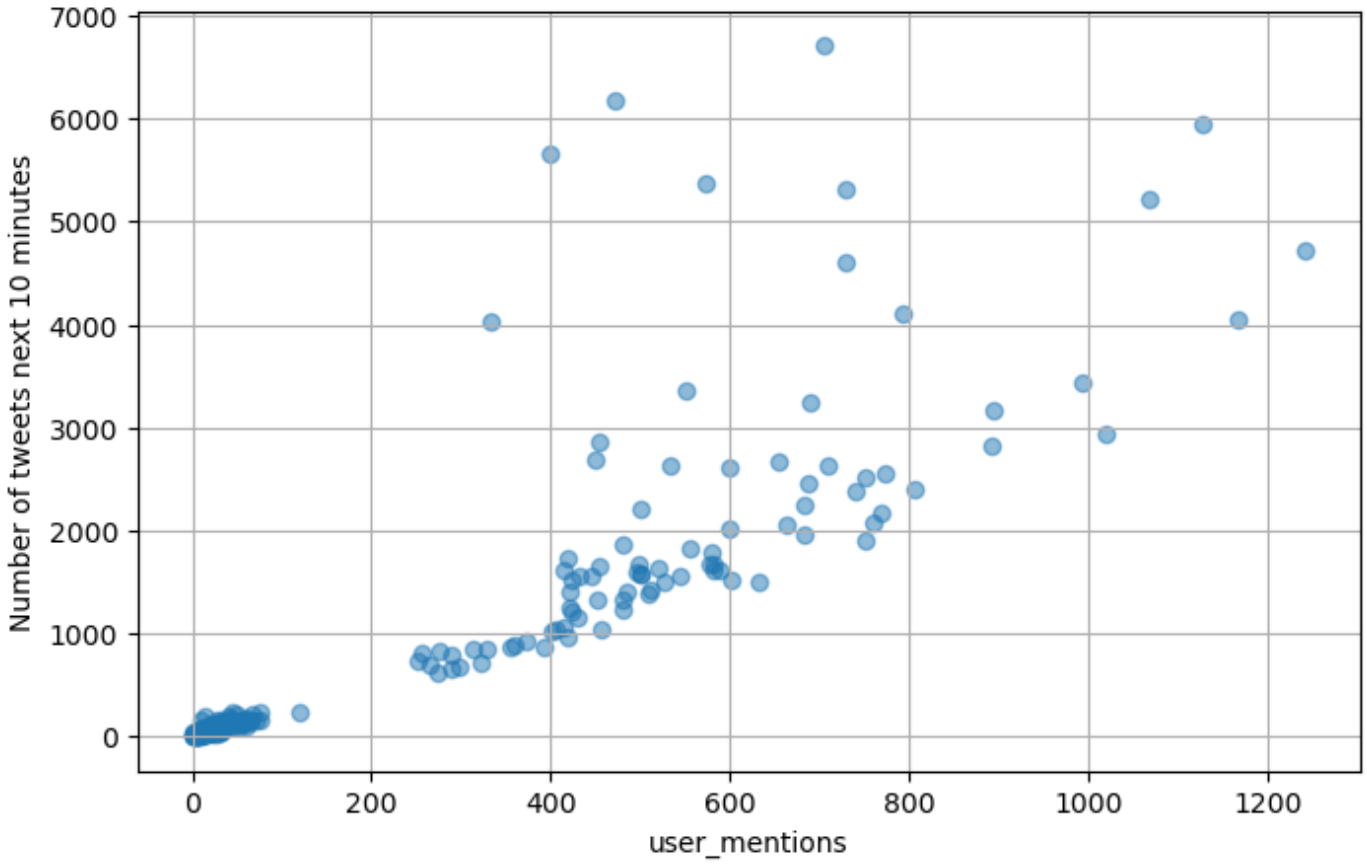
	mean_test_score	param_model__hidden_layer_sizes
0	-238146.783734	(5, 8)
1	-245119.714812	(25, 1)
2	-249601.147172	(9, 3)
3	-290023.313343	(5, 6)
4	-304567.791884	(8, 8)

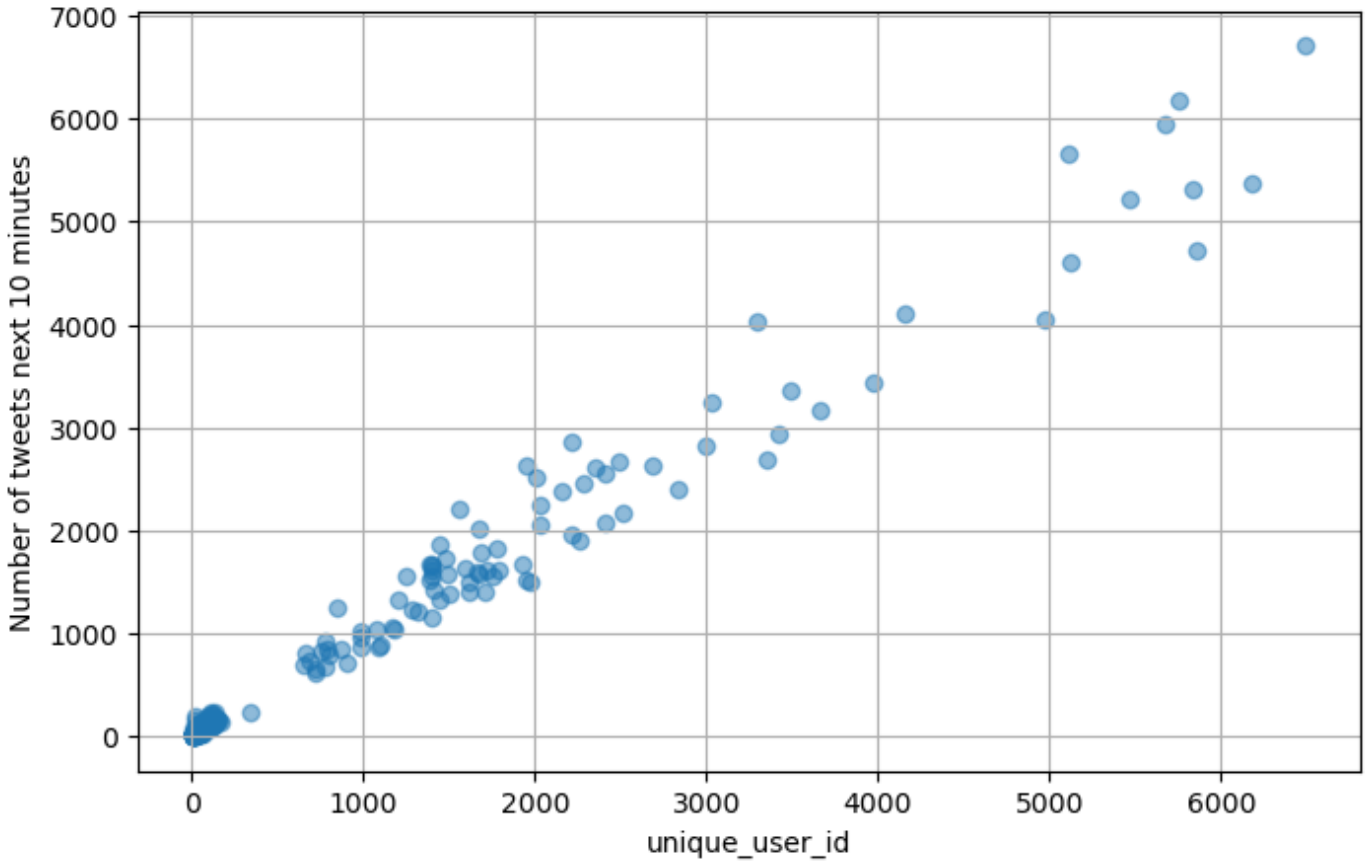
In [67]:
```
import statsmodels.api as sm
from sklearn import metrics
import numpy as np
import matplotlib.pyplot as plt

feature_names = list(superbowl_x.columns)
print(feature_names)

def scatter_plot(features, y_pred, pvalues, feature_names):
    # Obtain the indices that would sort the p-values in ascending order
    ranked_index = np.argsort(pvalues)
    print(ranked_index)
    for i in range(9):
        plt.figure(figsize = (8,5))
        # Create a scatter plot of the ith feature against the predicted values
        plt.scatter(features.iloc[:,ranked_index[i]], y_pred, alpha=0.5)
        plt.xlabel(feature_names[ranked_index[i]])
        plt.ylabel("Number of tweets next 10 minutes")
        plt.grid(True)
        plt.show()
    print('-' * 80)

# Fit a linear regression model to the data and obtain the predicted values and p-values
lr_fit = sm.OLS(superbowl_y,superbowl_x).fit()
y_pred = lr_fit.predict()
```

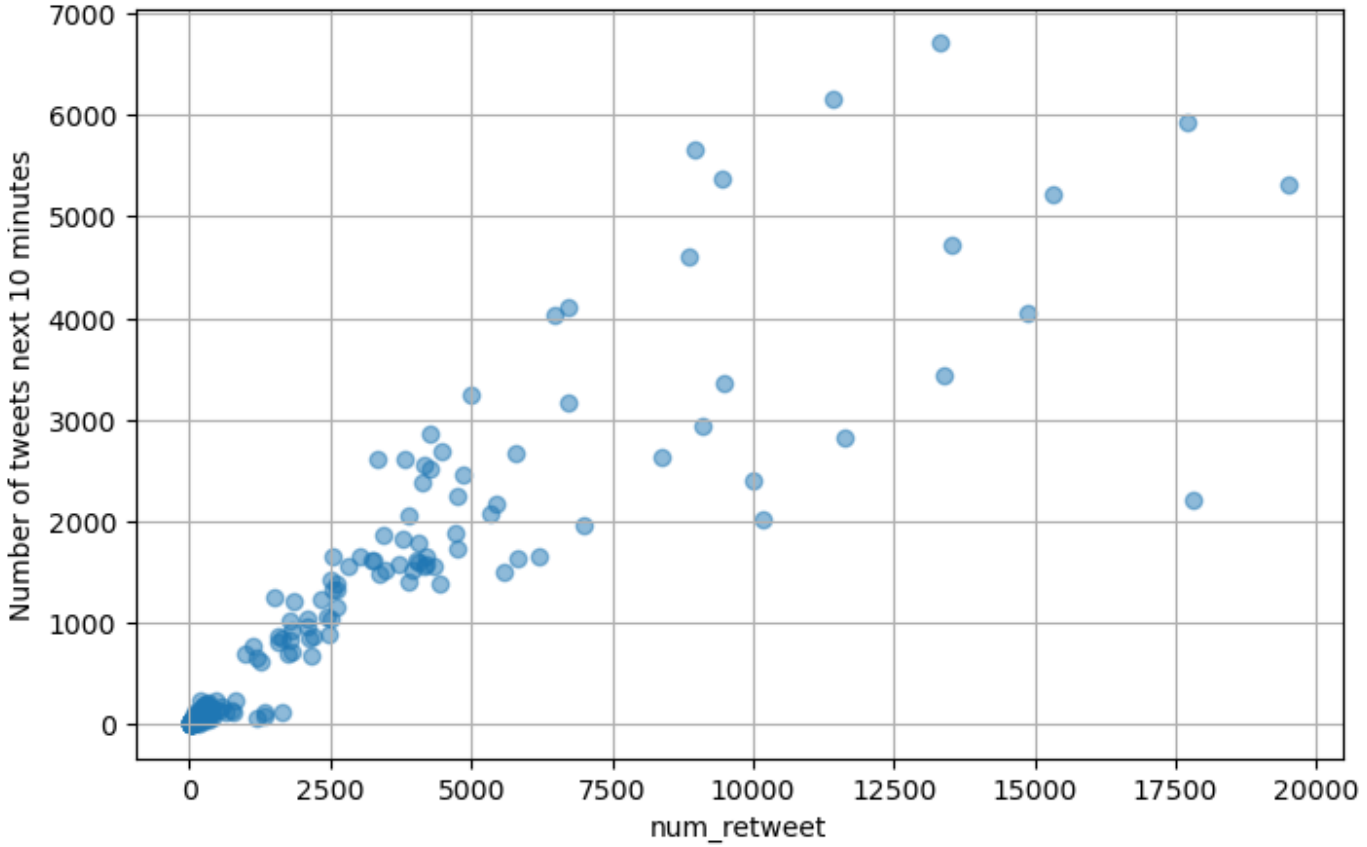
```
pvalues = lr_fit.pvalues
print('MSE: ', metrics.mean_squared_error(superbowl_y, y_pred))
print(lr_fit.summary())
scatter_plot(superbowl_x, y_pred, pvalues, feature_names)
print('\n')
```

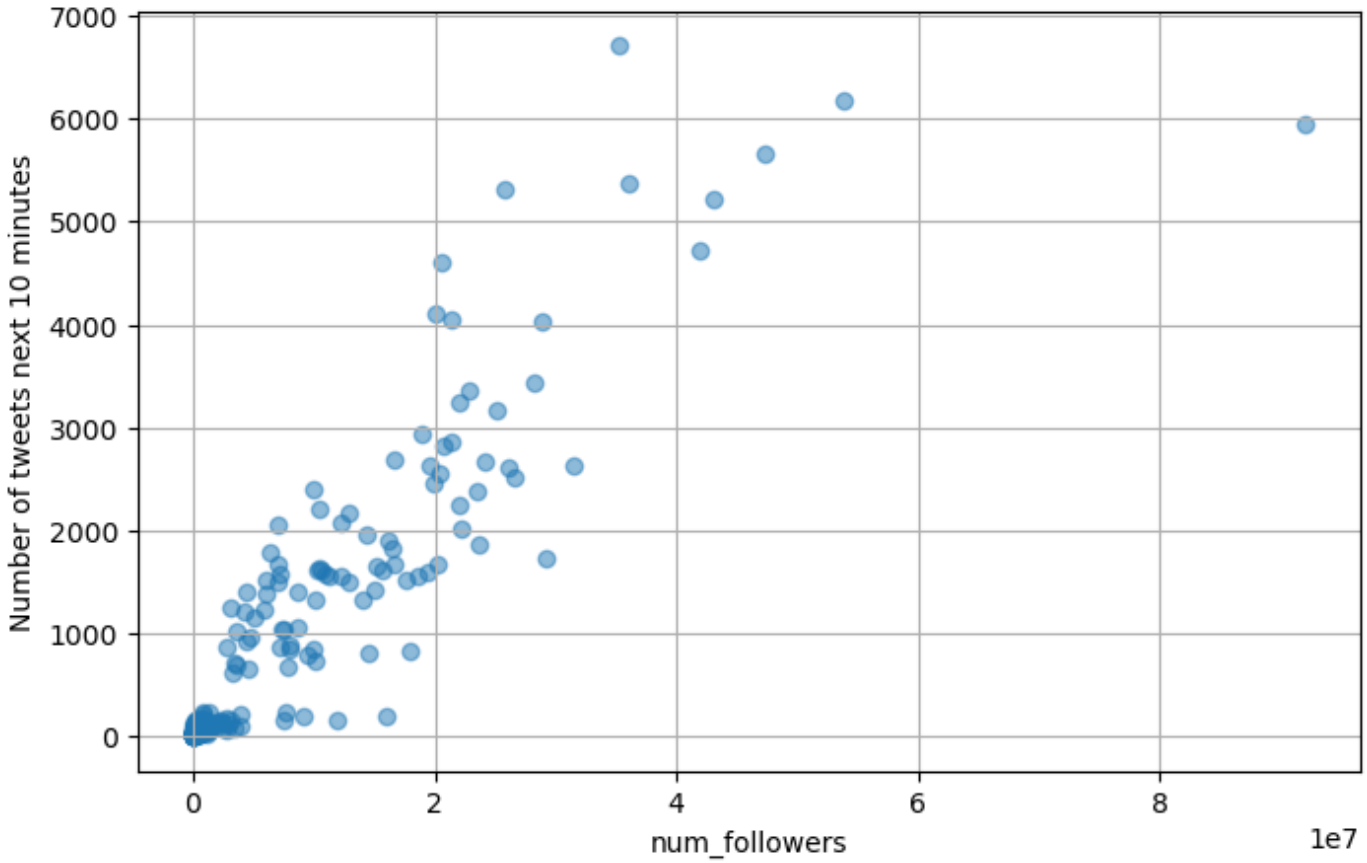
```
['num_retweet', 'num_followers', 'ranking_score', 'user_activity', 'user_mentions', 'num
_positive', 'num_neutral', 'num_negative', 'unique_user_id']
MSE:  156224.0969515157
                                 OLS Regression Results
==============================================================================================
Dep. Variable:                          y   R-squared (uncentered):                   0.940
Model:                                OLS   Adj. R-squared (uncentered):              0.938
Method:                     Least Squares   F-statistic:                              402.4
Date:                    Tue, 14 Mar 2023   Prob (F-statistic):                   8.57e-136
Time:                            15:00:01   Log-Likelihood:                         -1775.6
No. Observations:                     240   AIC:                                      3569.
Df Residuals:                         231   BIC:                                      3601.
Df Model:                               9
Covariance Type:                nonrobust
=====================================================================================
                    coef    std err          t      P>|t|      [0.025      0.975]
-------------------------------------------------------------------------------------
num_retweet        0.0377      0.020      1.842      0.067      -0.003       0.078
num_followers   9.285e-06   6.16e-06      1.507      0.133   -2.85e-06    2.14e-05
ranking_score      1.4483      0.428      3.386      0.001       0.605       2.291
user_activity      0.0018      0.005      0.346      0.730      -0.008       0.012
user_mentions      1.0776      0.402      2.684      0.008       0.286       1.869
num_positive      -1.5432      3.339     -0.462      0.644      -8.121       5.035
num_neutral       -0.9289      3.383     -0.275      0.784      -7.595       5.738
num_negative       0.3506      3.318      0.106      0.916      -6.188       6.889
unique_user_id    -5.2054      2.542     -2.048      0.042     -10.214      -0.197
==============================================================================
Omnibus:                      170.365   Durbin-Watson:                   2.233
Prob(Omnibus):                  0.000   Jarque-Bera (JB):             4411.142
Skew:                           2.349   Prob(JB):                         0.00
Kurtosis:                      23.471   Cond. No.                     3.10e+06
==============================================================================

Notes:
[1] R² is computed without centering (uncentered) since the model does not contain a con
stant.
[2] Standard Errors assume that the covariance matrix of the errors is correctly specifi
ed.
[3] The condition number is large, 3.1e+06. This might indicate that there are
strong multicollinearity or other numerical problems.
num_retweet       2
num_followers     4
ranking_score     8
user_activity     0
user_mentions     1
num_positive      5
num_neutral       3
num_negative      6
unique_user_id    7
dtype: int64
```

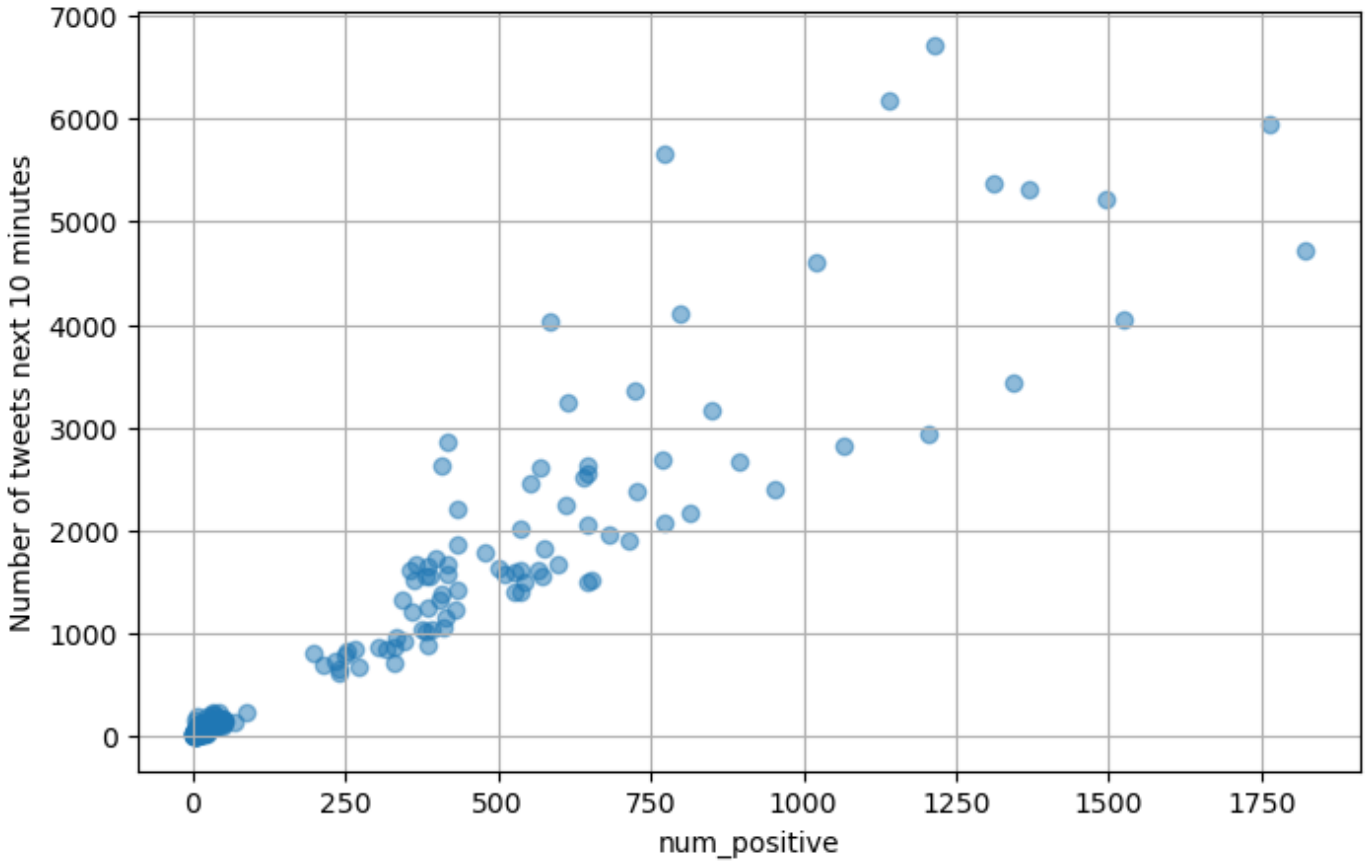

Number of tweets next 10 minutes
7000
6000
5000
4000
3000
2000
1000
0
0
5000
10000
15000
20000
25000
30000
ranking_score

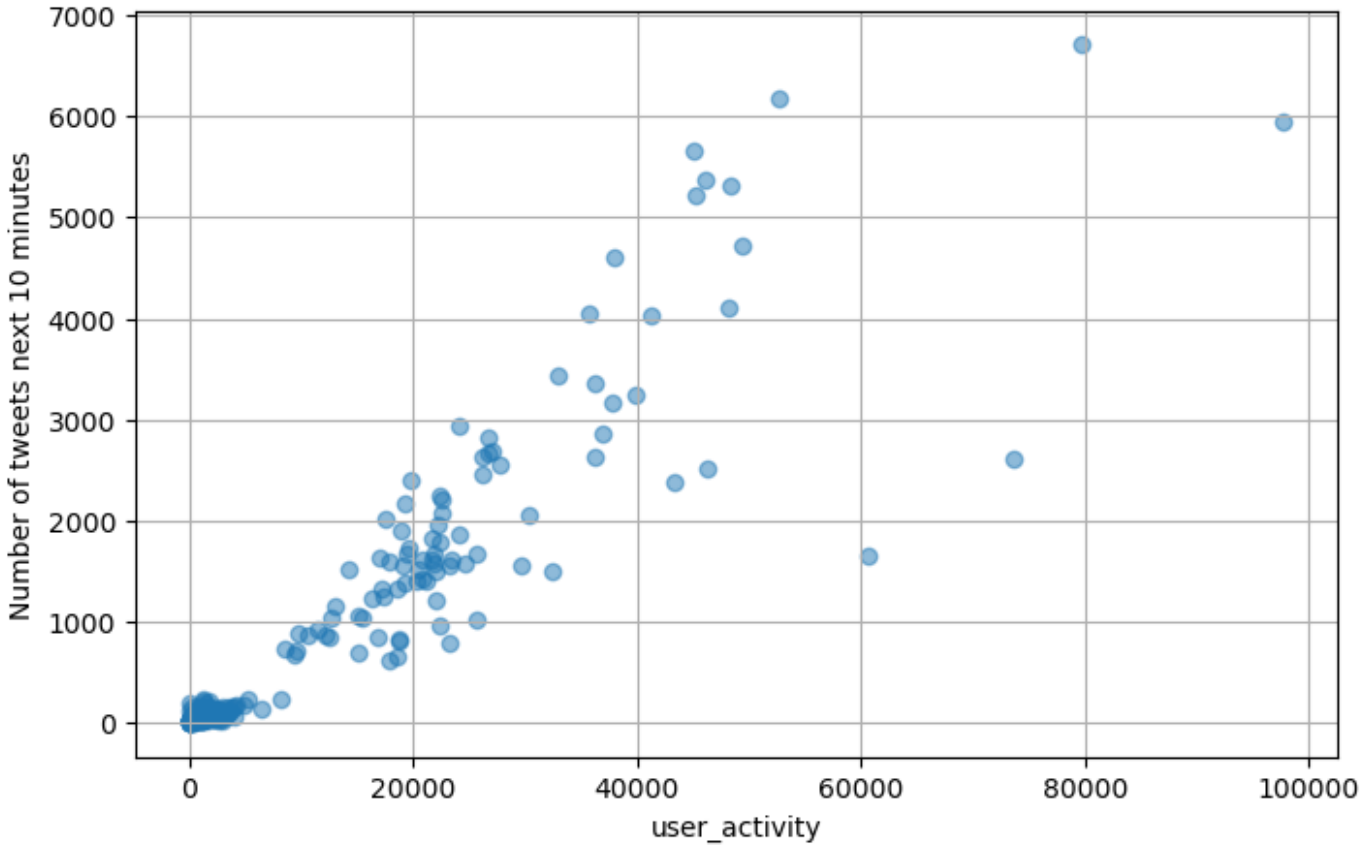

Number of tweets next 10 minutes
7000
6000
5000
4000
3000
2000
1000
0
0
200
400
600
800
1000
1200
user_mentions

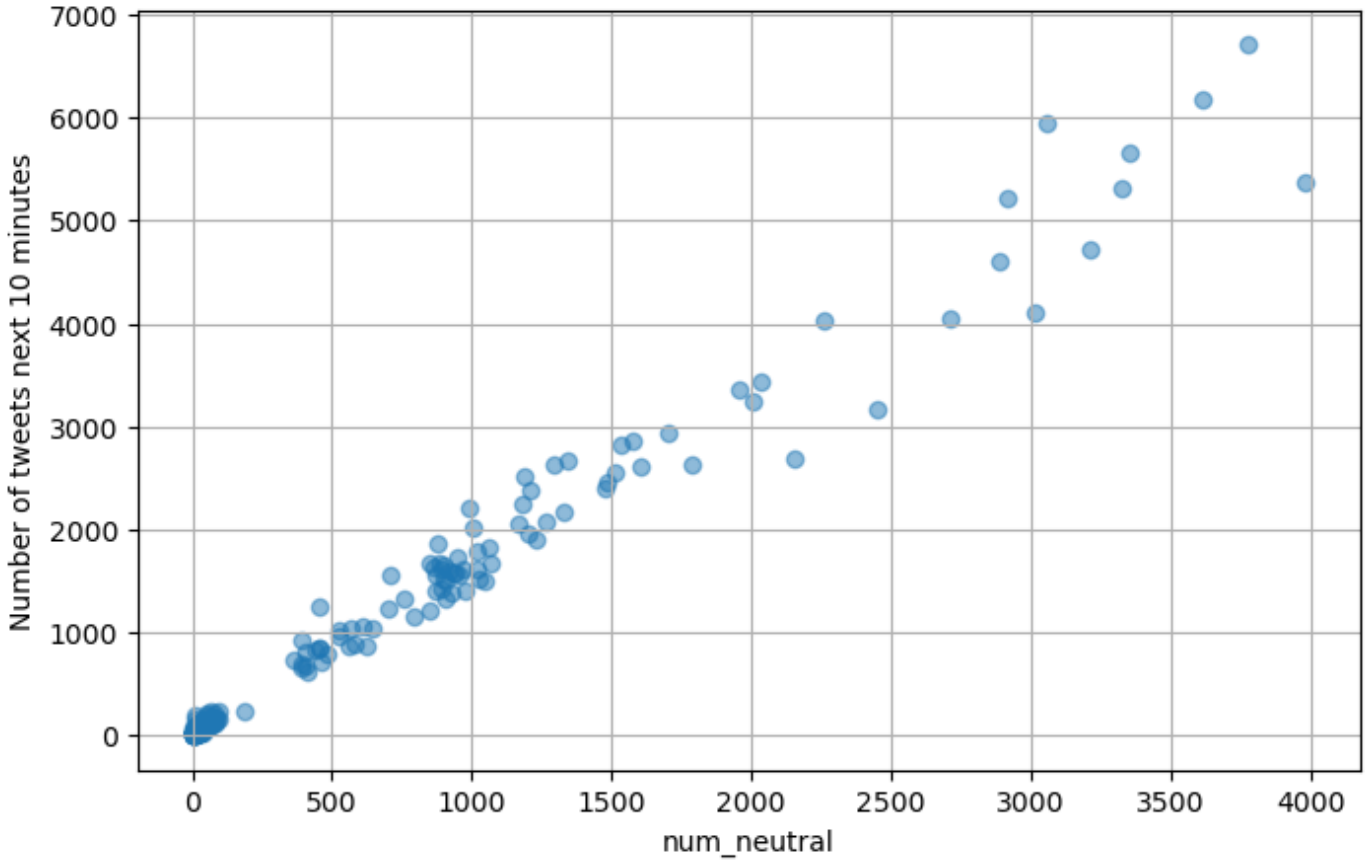

Number of tweets next 10 minutes
unique_user_id
0
1000
2000
3000
4000
5000
6000
7000

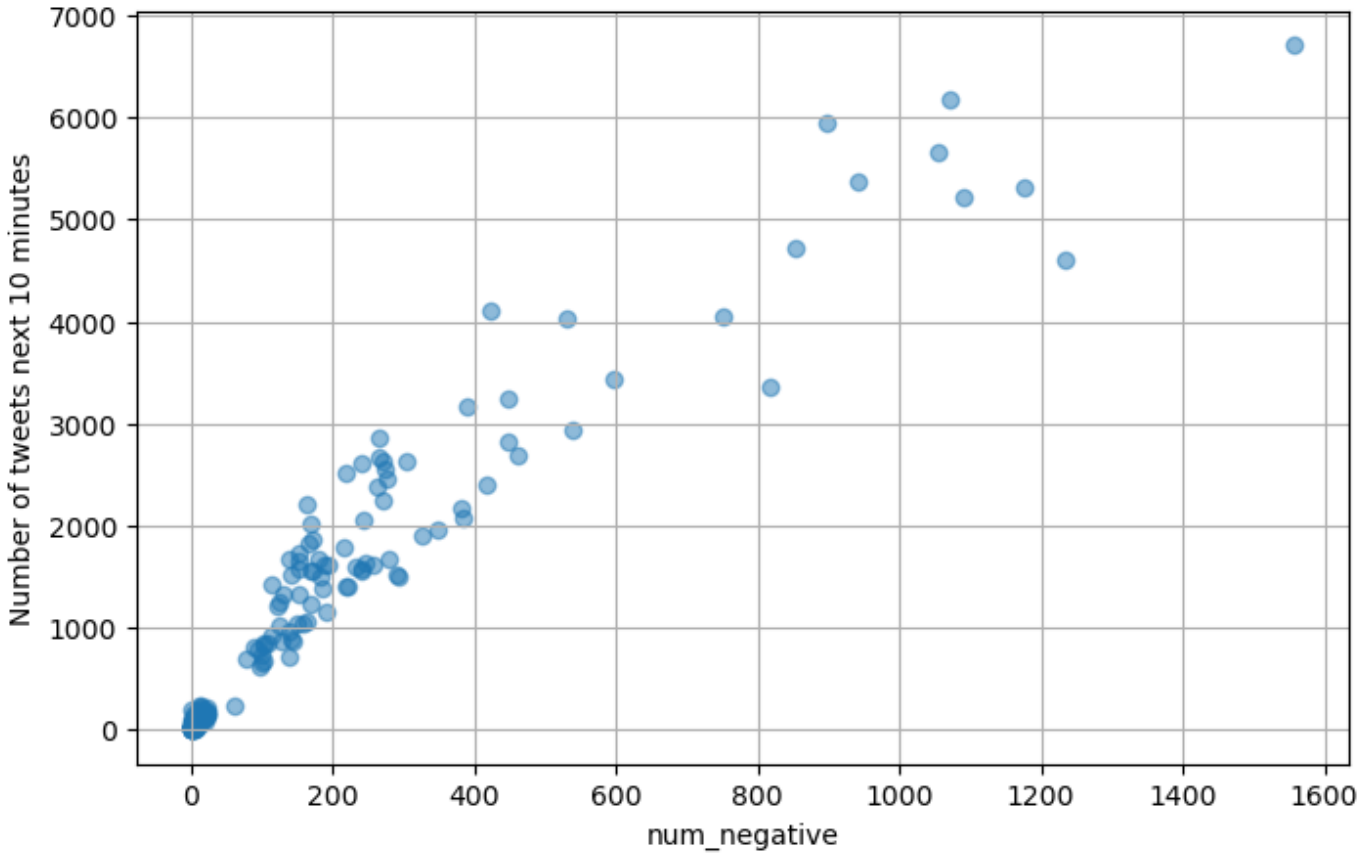

Number of tweets next 10 minutes
num_retweet
0
2500
5000
7500
10000
12500
15000
17500
20000
1000
2000
3000
4000
5000
6000
7000


Number of tweets next 10 minutes
7000
6000
5000
4000
3000
2000
1000
0
0
2
4
6
8
num_followers
1e7


Number of tweets next 10 minutes
7000
6000
5000
4000
3000
2000
1000
0
0
250
500
750
1000
1250
1500
1750
num_positive


Number of tweets next 10 minutes
7000
6000
5000
4000
3000
2000
1000
0
0
20000
40000
60000
80000
100000
user_activity


Number of tweets next 10 minutes
7000
6000
5000
4000
3000
2000
1000
0
0
500
1000
1500
2000
2500
3000
3500
4000
num_neutral



In [68]:
```
from sklearn.preprocessing import StandardScaler
from sklearn import preprocessing

superbowl_merge_data_standard_x = pd.DataFrame(preprocessing.scale(superbowl_merge_data_
# superbowl_merge_data_standard_x = superbowl_merge_data_drop
```

In [69]:
```
superbowl_merge_data_standard_x = superbowl_merge_data_standard_x.drop(superbowl_merge_d
```

In [71]:
```
print(superbowl_merge_data_standard_x.shape)
```

```
(240, 9)
```

In [78]:
```
superbowl_merge_data_standard_y =  pd.DataFrame(preprocessing.scale(superbowl_y),columns
print(superbowl_merge_data_standard_y.shape)
```

```
(240,)
```

In [84]:
```
from sklearn.feature_selection import SelectKBest, mutual_info_regression, f_regression
import numpy as np

# Define a function that selects the top n most important features using mutual informat
def select_topn_important_features(X, Y, n):
  Mutual_ = mutual_info_regression(X, Y)
  F_ = f_regression(X, Y)

  # Select the top n features based on their mutual information and F-test scores
  topn_M = np.argsort(Mutual_)[::-1][:n]
  topn_F = np.argsort(F_[0])[::-1][:n]

  # Sort all the features based on their mutual information and F-test scores
  all_m = np.argsort(Mutual_)[::-1]
  all_f = np.argsort(F_[0])[::-1]
```

```
  # Extract the top n features and all features based on their mutual information scores
  X_topn_M = X.iloc[:, topn_M]
  X_topn_F = X.iloc[:, topn_F]

  # Extract the top n features and all features based on their F-test scores
  all_m_ = X.iloc[:, all_m]
  all_f_ = X.iloc[:, all_f]

  return X_topn_M, X_topn_F
```

In [80]:
```
superbowl_top3_M, superbowl_top3_F, sall_m, sall_f = select_topn_important_features(supe
```

In [81]:
```
print("superbowl Top3 by mutual_info_regression:")
print(superbowl_top3_M.columns)

print("superbowl Top3 by f_regression:")
print(superbowl_top3_F.columns)

print("superbowl all by mutual_info_regression:")
print(sall_m.columns)

print("superbowl all by f_regression:")
print(sall_f.columns)
```

```
superbowl Top3 by mutual_info_regression:
Index(['unique_user_id', 'ranking_score', 'num_neutral'], dtype='object')
superbowl Top3 by f_regression:
Index(['ranking_score', 'num_neutral', 'unique_user_id'], dtype='object')
superbowl all by mutual_info_regression:
Index(['unique_user_id', 'ranking_score', 'num_neutral', 'num_positive',
       'user_mentions', 'num_negative', 'num_retweet', 'user_activity',
       'num_followers'],
      dtype='object')
superbowl all by f_regression:
Index(['ranking_score', 'num_neutral', 'unique_user_id', 'num_negative',
       'num_positive', 'user_activity', 'num_retweet', 'num_followers',
       'user_mentions'],
      dtype='object')
```

In [85]:
```
# now we are do experiments of exactly how many features we need to select

#with selection
superbowl_rmse_lr_m = []  #superbowl rmse score for linear regression and with mutual_in
superbowl_rmse_lr_f = []
superbowl_rmse_r_m = []
superbowl_rmse_r_f = []
superbowl_rmse_la_m = []
superbowl_rmse_la_f = []

#without selection
superbowl_rmse_lr = [] #superbowl rmse score for linear regression
superbowl_rmse_r = []
superbowl_rmse_la = []
```

In [86]:
```
from sklearn.model_selection import cross_validate,GridSearchCV
from sklearn.linear_model import LinearRegression, Ridge, Lasso

for k in range(1, 10):
    superbowl_topk_M, superbowl_topk_F = select_topn_important_features(superbowl_merge_
    #superbowl rmse score for linear regression
    score_ = cross_validate(LinearRegression(), superbowl_merge_data_standard_x, superbo
    score__ = score_['test_neg_root_mean_squared_error'].mean()
    print(f"superbowl rmse score for linear regression, {score__:.4f}")
    superbowl_rmse_lr.append(score__)
```

```
    #superbowl rmse score for linear regression and with mutual_info_regression
    score_ = cross_validate(LinearRegression(), superbowl_topk_M, superbowl_merge_data_s
    score__ = score_['test_neg_root_mean_squared_error'].mean()
    print(f"superbowl rmse score for linear regression and with mutual_info_regression,
    superbowl_rmse_lr_m.append(score__)

    #superbowl rmse score for linear regression and with f_regression
    score_ = cross_validate(LinearRegression(), superbowl_topk_F, superbowl_merge_data_s
    score__ = score_['test_neg_root_mean_squared_error'].mean()
    print(f"superbowl rmse score for linear regression and with f_regression, {score__:.
    superbowl_rmse_lr_f.append(score__)

    #superbowl rmse score for Ridge regression
    score_ = cross_validate(Ridge(), superbowl_merge_data_standard_x, superbowl_merge_da
    score__ = score_['test_neg_root_mean_squared_error'].mean()
    print(f"superbowl rmse score for Ridge regression, {score__:.4f}")
    superbowl_rmse_r.append(score__)

    #superbowl rmse score for Ridge regression and with mutual_info_regression
    score_ = cross_validate(Ridge(), superbowl_topk_M, superbowl_merge_data_standard_y,
    score__ = score_['test_neg_root_mean_squared_error'].mean()
    print(f"superbowl rmse score for Ridge regression and with mutual_info_regression, {
    superbowl_rmse_r_m.append(score__)

    #superbowl rmse score for Ridge regression and with f_regression
    score_ = cross_validate(Ridge(), superbowl_topk_F, superbowl_merge_data_standard_y,
    score__ = score_['test_neg_root_mean_squared_error'].mean()
    print(f"superbowl rmse score for Ridge regression and with f_regression, {score__:.4
    superbowl_rmse_r_f.append(score__)

    #superbowl rmse score for Lasso regression
    score_ = cross_validate(Lasso(), superbowl_merge_data_standard_x, superbowl_merge_da
    score__ = score_['test_neg_root_mean_squared_error'].mean()
    print(f"superbowl rmse score for Lasso regression, {score__:.4f}")
    superbowl_rmse_la.append(score__)

    #superbowl rmse score for Lasso regression and with mutual_info_regression
    score_ = cross_validate(Lasso(), superbowl_topk_M, superbowl_merge_data_standard_y,
    score__ = score_['test_neg_root_mean_squared_error'].mean()
    print(f"superbowl rmse score for Lasso regression and with mutual_info_regression, {
    superbowl_rmse_la_m.append(score__)

    #superbowl rmse score for Lasso regression and with f_regression
    score_ = cross_validate(Lasso(), superbowl_topk_F, superbowl_merge_data_standard_y,
    score__ = score_['test_neg_root_mean_squared_error'].mean()
    print(f"superbowl rmse score for Lasso regression and with f_regression, {score__:.4
    superbowl_rmse_la_f.append(score__)
```

```
superbowl rmse score for linear regression, -0.2195
superbowl rmse score for linear regression and with mutual_info_regression, -0.1650 top1
superbowl rmse score for linear regression and with f_regression, -0.1612 top1
superbowl rmse score for Ridge regression, -0.1797
superbowl rmse score for Ridge regression and with mutual_info_regression, -0.1663 top1
superbowl rmse score for Ridge regression and with f_regression, -0.1625 top1
superbowl rmse score for Lasso regression, -0.8621
superbowl rmse score for Lasso regression and with mutual_info_regression, -0.8631 top1
superbowl rmse score for Lasso regression and with f_regression, -0.8621 top1
superbowl rmse score for linear regression, -0.2195
superbowl rmse score for linear regression and with mutual_info_regression, -0.1770 top2
superbowl rmse score for linear regression and with f_regression, -0.1608 top2
superbowl rmse score for Ridge regression, -0.1797
superbowl rmse score for Ridge regression and with mutual_info_regression, -0.1632 top2
superbowl rmse score for Ridge regression and with f_regression, -0.1619 top2
superbowl rmse score for Lasso regression, -0.8621
superbowl rmse score for Lasso regression and with mutual_info_regression, -0.8621 top2
superbowl rmse score for Lasso regression and with f_regression, -0.8621 top2
```

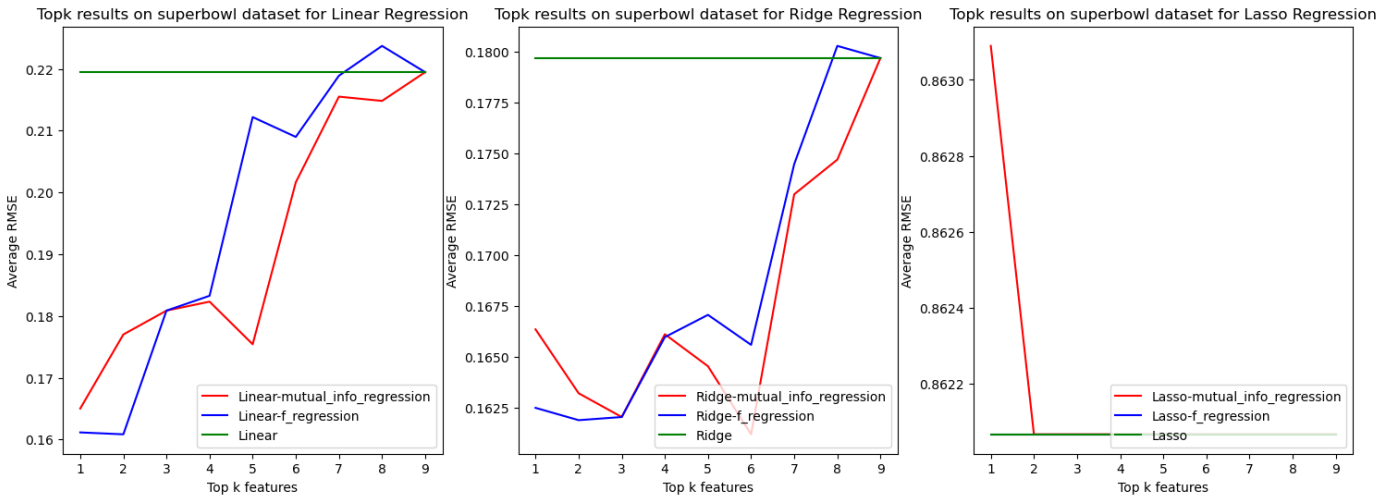
```
superbowl rmse score for linear regression, -0.2195
superbowl rmse score for linear regression and with mutual_info_regression, -0.1809 top3
superbowl rmse score for linear regression and with f_regression, -0.1809 top3
superbowl rmse score for Ridge regression, -0.1797
superbowl rmse score for Ridge regression and with mutual_info_regression, -0.1620 top3
superbowl rmse score for Ridge regression and with f_regression, -0.1620 top3
superbowl rmse score for Lasso regression, -0.8621
superbowl rmse score for Lasso regression and with mutual_info_regression, -0.8621 top3
superbowl rmse score for Lasso regression and with f_regression, -0.8621 top3
superbowl rmse score for linear regression, -0.2195
superbowl rmse score for linear regression and with mutual_info_regression, -0.1823 top4
superbowl rmse score for linear regression and with f_regression, -0.1833 top4
superbowl rmse score for Ridge regression, -0.1797
superbowl rmse score for Ridge regression and with mutual_info_regression, -0.1661 top4
superbowl rmse score for Ridge regression and with f_regression, -0.1660 top4
superbowl rmse score for Lasso regression, -0.8621
superbowl rmse score for Lasso regression and with mutual_info_regression, -0.8621 top4
superbowl rmse score for Lasso regression and with f_regression, -0.8621 top4
superbowl rmse score for linear regression, -0.2195
superbowl rmse score for linear regression and with mutual_info_regression, -0.1754 top5
superbowl rmse score for linear regression and with f_regression, -0.2122 top5
superbowl rmse score for Ridge regression, -0.1797
superbowl rmse score for Ridge regression and with mutual_info_regression, -0.1645 top5
superbowl rmse score for Ridge regression and with f_regression, -0.1671 top5
superbowl rmse score for Lasso regression, -0.8621
superbowl rmse score for Lasso regression and with mutual_info_regression, -0.8621 top5
superbowl rmse score for Lasso regression and with f_regression, -0.8621 top5
superbowl rmse score for linear regression, -0.2195
superbowl rmse score for linear regression and with mutual_info_regression, -0.2016 top6
superbowl rmse score for linear regression and with f_regression, -0.2090 top6
superbowl rmse score for Ridge regression, -0.1797
superbowl rmse score for Ridge regression and with mutual_info_regression, -0.1612 top6
superbowl rmse score for Ridge regression and with f_regression, -0.1656 top6
superbowl rmse score for Lasso regression, -0.8621
superbowl rmse score for Lasso regression and with mutual_info_regression, -0.8621 top6
superbowl rmse score for Lasso regression and with f_regression, -0.8621 top6
superbowl rmse score for linear regression, -0.2195
superbowl rmse score for linear regression and with mutual_info_regression, -0.2155 top7
superbowl rmse score for linear regression and with f_regression, -0.2189 top7
superbowl rmse score for Ridge regression, -0.1797
superbowl rmse score for Ridge regression and with mutual_info_regression, -0.1730 top7
superbowl rmse score for Ridge regression and with f_regression, -0.1745 top7
superbowl rmse score for Lasso regression, -0.8621
superbowl rmse score for Lasso regression and with mutual_info_regression, -0.8621 top7
superbowl rmse score for Lasso regression and with f_regression, -0.8621 top7
superbowl rmse score for linear regression, -0.2195
superbowl rmse score for linear regression and with mutual_info_regression, -0.2148 top8
superbowl rmse score for linear regression and with f_regression, -0.2237 top8
superbowl rmse score for Ridge regression, -0.1797
superbowl rmse score for Ridge regression and with mutual_info_regression, -0.1747 top8
superbowl rmse score for Ridge regression and with f_regression, -0.1803 top8
superbowl rmse score for Lasso regression, -0.8621
superbowl rmse score for Lasso regression and with mutual_info_regression, -0.8621 top8
superbowl rmse score for Lasso regression and with f_regression, -0.8621 top8
superbowl rmse score for linear regression, -0.2195
superbowl rmse score for linear regression and with mutual_info_regression, -0.2195 top9
superbowl rmse score for linear regression and with f_regression, -0.2195 top9
superbowl rmse score for Ridge regression, -0.1797
superbowl rmse score for Ridge regression and with mutual_info_regression, -0.1797 top9
superbowl rmse score for Ridge regression and with f_regression, -0.1797 top9
superbowl rmse score for Lasso regression, -0.8621
superbowl rmse score for Lasso regression and with mutual_info_regression, -0.8621 top9
superbowl rmse score for Lasso regression and with f_regression, -0.8621 top9
```

In [87]:
```
import matplotlib.pyplot as plt
```

```
fig, axes = plt.subplots(1, 3, figsize=(18, 6))

#plot superbowl linear
axes[0].plot(np.arange(1, len(superbowl_rmse_lr_m) + 1, 1), np.negative(superbowl_rmse_l
axes[0].plot(np.arange(1, len(superbowl_rmse_lr_f) + 1, 1), np.negative(superbowl_rmse_l
axes[0].plot(np.arange(1, len(superbowl_rmse_lr) + 1, 1), np.negative(superbowl_rmse_lr)

axes[0].legend(loc='lower right')
axes[0].set(xlabel='Top k features',ylabel='Average RMSE')
axes[0].set_title('Topk results on superbowl dataset for Linear Regression')

#plot superbowl Ridge
axes[1].plot(np.arange(1, len(superbowl_rmse_r_m) + 1, 1), np.negative(superbowl_rmse_r_
axes[1].plot(np.arange(1, len(superbowl_rmse_r_f) + 1, 1), np.negative(superbowl_rmse_r_
axes[1].plot(np.arange(1, len(superbowl_rmse_r) + 1, 1), np.negative(superbowl_rmse_r),

axes[1].legend(loc='lower right')
axes[1].set(xlabel='Top k features',ylabel='Average RMSE')
axes[1].set_title('Topk results on superbowl dataset for Ridge Regression')

#plot superbowl Lasso
axes[2].plot(np.arange(1, len(superbowl_rmse_la_m) + 1, 1), np.negative(superbowl_rmse_l
axes[2].plot(np.arange(1, len(superbowl_rmse_la_f) + 1, 1), np.negative(superbowl_rmse_l
axes[2].plot(np.arange(1, len(superbowl_rmse_la) + 1, 1), np.negative(superbowl_rmse_la)

axes[2].legend(loc='lower right')
axes[2].set(xlabel='Top k features',ylabel='Average RMSE')
axes[2].set_title('Topk results on superbowl dataset for Lasso Regression')
```

Out[87]:

```
Text(0.5, 1.0, 'Topk results on superbowl dataset for Lasso Regression')
```



In [88]:

```
# Finding the optimal penalty parameter
from joblib import Memory
from sklearn.pipeline import Pipeline

location = "cachedir"
memory = Memory(location=location, verbose=10)

pipe_ = Pipeline([
    ('kbest', SelectKBest()),
    ('model', "passthrough")
], memory = memory)

param_grid = [{
        'kbest__score_func': (mutual_info_regression, f_regression),
        'kbest__k': (1, 2, 3, 4, 5, 6, 7, 8, 9),
        'model': [Ridge(), Lasso()],
        'model__alpha': [10.0**x for x in np.arange(-3,4)]
```

```
    }
]
```

In [89]:
```
grid_superbowl = GridSearchCV(pipe_, param_grid = param_grid, cv = 10, n_jobs = -1, verb
                    scoring = 'neg_root_mean_squared_error', return_train_score = True)
```

```
Fitting 10 folds for each of 252 candidates, totalling 2520 fits
_____________________________________________________________________________________
[Memory] Calling sklearn.pipeline._fit_transform_one...
_fit_transform_one(SelectKBest(k=6,
            score_func=<function mutual_info_regression at 0x000001D011E13820>),
     num_retweet  num_followers  ranking_score  user_activity  user_mentions  \
0       0.572113      -0.037429       0.628675       0.750375       1.241756
1       0.276907       1.212140       0.417785       0.956063       0.957631
2       0.646982       1.060345       0.313306       0.540213       0.732410
3       0.392484       1.517641       0.477341       0.856452       0.902192
4       0.358324       1.148113       0.882697       0.989590       1.082369
..           ...            ...            ...            ...            ...
235    -0.568855      -0.593462      -0.587048      -0.641907      -0.698609
236    -0.558322      -0.590781      -0.576466      -0.58614...,
array([ 0.424299, ..., -0.614222]), None, message_clsname='Pipeline', message=None)
_____________________________________________________fit_transform_one - 0.0s, 0.0min
```

Try the 5 minutes interval

In [6]:
```
superbowl_merge_5 = merge_different_intervals(5)
superbowl_merge_5.to_csv('superbowl_merge_5.csv', index=False)
```

In [7]:
```
superbowl_merge_5.shape
```

Out[7]:
```
(49, 16)
```

In [8]:
```
superbowl_merge_data_drop = superbowl_merge_5.copy().drop(['range_start','range_end','us
```

In [9]:
```
superbowl_x = superbowl_merge_data_drop.drop(superbowl_merge_data_drop.index[-1])
superbowl_y = superbowl_merge_5["num_tweet"]
superbowl_y = superbowl_y.drop(superbowl_y.index[0])
superbowl_y = pd.DataFrame(superbowl_y,columns = ["num_tweet"]).values.ravel()
```

In [10]:
```
superbowl_x
```

Out[10]:

	num_retweet	num_followers	ranking_score	user_activity	user_mentions	num_positive	num_neutr
0	18199	88505454.0	36332.345488	117586.567551	2522	2068	478
1	24049	116180139.0	55888.883342	182504.637944	3521	3431	723
2	32435	55625225.0	39661.108591	122537.520333	2807	2609	516
3	24964	85480534.0	32576.856109	96822.588407	2562	2095	440
4	16176	82163158.0	36694.440885	162629.060762	2197	1951	514
5	29816	123167599.0	70674.211049	223556.155001	3766	3511	103
6	19600	77381689.0	45329.705349	122030.875747	2421	2367	615
7	45061	155969643.0	121524.110384	235835.998115	2856	5084	1614
8	73426	237439681.0	120701.882300	280203.822931	3946	6493	148
9	62529	131223451.0	95823.774938	168913.121608	5307	6961	1118
10	32497	65588142.0	54910.008486	102797.003764	3766	3933	650
11	21801	54970967.0	41808.331313	96812.961516	2893	2990	502

12	22501	42044941.0	37926.426929	109953.060352	2470	2646	461
13	13236	40763767.0	29659.858376	86596.384795	2210	2186	358
14	10427	29695483.0	23898.670841	73199.066904	2026	1736	291
15	8966	24647055.0	20433.627481	76625.428694	1848	1760	229
16	8514	54833276.0	16256.919362	74127.666929	1371	1197	201
17	5454	25679109.0	14985.739539	81216.857436	1261	1059	192
18	2571	25036842.0	2285.072718	11230.590655	199	136	21
19	1820	8267164.0	1347.850990	3220.103626	129	90	15
20	510	4575430.0	726.759025	2010.381932	76	42	8
21	427	2545858.0	687.371220	3036.646770	69	54	1
22	923	5556480.0	740.502986	3969.053104	83	50	8
23	1818	5909616.0	781.233467	4299.656204	72	60	7
24	1281	2995492.0	2869.232326	12961.515964	254	218	32
25	1292	5678312.0	3168.827673	7065.800379	296	244	36
26	1129	2437638.0	3123.305922	12619.062771	315	230	39
27	1152	9237390.0	2624.394593	14822.798442	224	192	30
28	1828	4007065.0	2931.177174	13816.453953	265	218	35
29	471	1423842.0	998.905001	2928.521394	104	71	12
30	479	506644.0	350.170646	4691.627396	30	28	3
31	1349	2314602.0	359.148610	2012.474407	32	25	4
32	538	16705519.0	435.728795	1105.182888	37	35	4
33	199	2572285.0	335.064223	862.940930	39	30	3
34	163	871109.0	290.824144	644.165157	22	22	2
35	502	13094623.0	345.926133	811.460229	32	24	3
36	288	241576.0	327.997626	1630.080205	25	22	3
37	403	1876451.0	356.368414	1929.015063	34	21	3
38	1448	1354460.0	266.121794	766.878573	27	24	2
39	364	701757.0	940.328439	5896.519758	89	75	12
40	1073	2704111.0	1362.819125	8947.657745	158	112	15
41	563	1650418.0	1293.284668	8428.808582	177	110	15
42	498	1891770.0	1147.153845	7707.151897	108	82	13
43	502	2779976.0	1339.804213	9252.664536	126	104	16
44	2189	3230987.0	1111.708280	5699.220241	104	79	14
45	654	1846064.0	1128.829311	9348.912346	89	95	11
46	293	214285.0	872.107548	6261.727990	86	66	11
47	387	509911.0	1030.134260	6408.563268	116	81	12

In [11]:

```
# RandomForest GridSearch
import pandas as pd
from sklearn.pipeline import Pipeline
```

```
from sklearn.model_selection import KFold, GridSearchCV
from sklearn.preprocessing import StandardScaler
from sklearn.ensemble import RandomForestRegressor


pipe_rf = Pipeline([
    ('standardize', StandardScaler()),
    ('model', RandomForestRegressor(random_state=42))
])

param_grid = {
    'model__max_depth': [10, 30, 50, 70, 100, 200],
    'model__max_features': ['auto', 'sqrt'],
    'model__min_samples_leaf': [1, 2, 3],
    'model__min_samples_split': [2, 5, 10],
    'model__n_estimators': [200, 400]
}

grid_rf = GridSearchCV(pipe_rf, param_grid=param_grid, cv=KFold(5, shuffle=True, random_
                       scoring='neg_mean_squared_error').fit(superbowl_x, superbowl_y)
result_rf = pd.DataFrame(grid_rf.cv_results_)[['mean_test_score', 'param_model__max_dept
                                               'param_model__min_samples_leaf', 'param_mod
                                               'param_model__n_estimators']]
result_rf = result_rf.sort_values(by=['mean_test_score'], ascending=False).reset_index(d
result_rf.head()
```

```
Fitting 5 folds for each of 216 candidates, totalling 1080 fits
```

Out[11]:

	mean_test_score	param_model__max_depth	param_model__max_features	param_model__min_samples
0	-1.167336e+07	10	sqrt	
1	-1.169774e+07	70	sqrt	
2	-1.169774e+07	30	sqrt	
3	-1.169774e+07	200	sqrt	
4	-1.169774e+07	50	sqrt	

In [12]:

```
# GradientBoosting GridSearch
import pandas as pd
from sklearn.pipeline import Pipeline
from sklearn.model_selection import KFold, GridSearchCV
from sklearn.preprocessing import StandardScaler
from sklearn.ensemble import GradientBoostingRegressor


pipe_gb = Pipeline([
    ('standardize', StandardScaler()),
    ('model', GradientBoostingRegressor(random_state=42))
])

param_grid = {
    'model__max_depth': [10, 30, 50, 70, 100, 200],
    'model__max_features': ['auto', 'sqrt'],
    'model__min_samples_leaf': [1, 2, 3],
    'model__min_samples_split': [2, 5, 10],
    'model__n_estimators': [200, 400]
}


grid_gb = GridSearchCV(pipe_gb, param_grid=param_grid, cv=KFold(5, shuffle=True, random_
                       scoring='neg_mean_squared_error').fit(superbowl_x, superbowl_y)
result_gb = pd.DataFrame(grid_gb.cv_results_)[['mean_test_score', 'param_model__max_dept
                                               'param_model__min_samples_leaf', 'param_mod
                                               'param_model__n_estimators']]
```

```
result_gb = result_gb.sort_values(by=['mean_test_score'], ascending=False).reset_index(d
result_gb.head()
```

```
Fitting 5 folds for each of 216 candidates, totalling 1080 fits
```

Out[12]:

	mean_test_score	param_model__max_depth	param_model__max_features	param_model__min_samples
0	-6.611812e+06	10	auto	
1	-6.611812e+06	10	auto	
2	-6.655455e+06	50	auto	
3	-6.655455e+06	50	auto	
4	-6.655455e+06	70	auto	

In [13]:

```
# NeuralNetwork GridSearch

import pandas as pd
from sklearn.pipeline import Pipeline
from sklearn.model_selection import KFold, GridSearchCV
from sklearn.preprocessing import StandardScaler
from sklearn.neural_network import MLPRegressor

pipe_nn_noscale = Pipeline([
    ('model', MLPRegressor(random_state=42, max_iter=2000))
])

param_grid = {
    'model__hidden_layer_sizes': [(x,y) for x in np.arange(1, 51) for y in np.arange(1,
}


grid_nn_noscale = GridSearchCV(pipe_nn_noscale, param_grid=param_grid, cv=KFold(5, shuff
                               scoring='neg_mean_squared_error').fit(superbowl_x, superb
result_nn_noscale = pd.DataFrame(grid_nn_noscale.cv_results_)[['mean_test_score', 'param
result_nn_noscale = result_nn_noscale.sort_values(by=['mean_test_score'], ascending=Fals
result_nn_noscale.head()
```

```
Fitting 5 folds for each of 2500 candidates, totalling 12500 fits
```

Out[13]:

	mean_test_score	param_model__hidden_layer_sizes
0	-9.935143e+06	(6, 12)
1	-1.004485e+07	(39, 11)
2	-1.022622e+07	(5, 8)
3	-1.108928e+07	(33, 1)
4	-1.123810e+07	(33, 2)

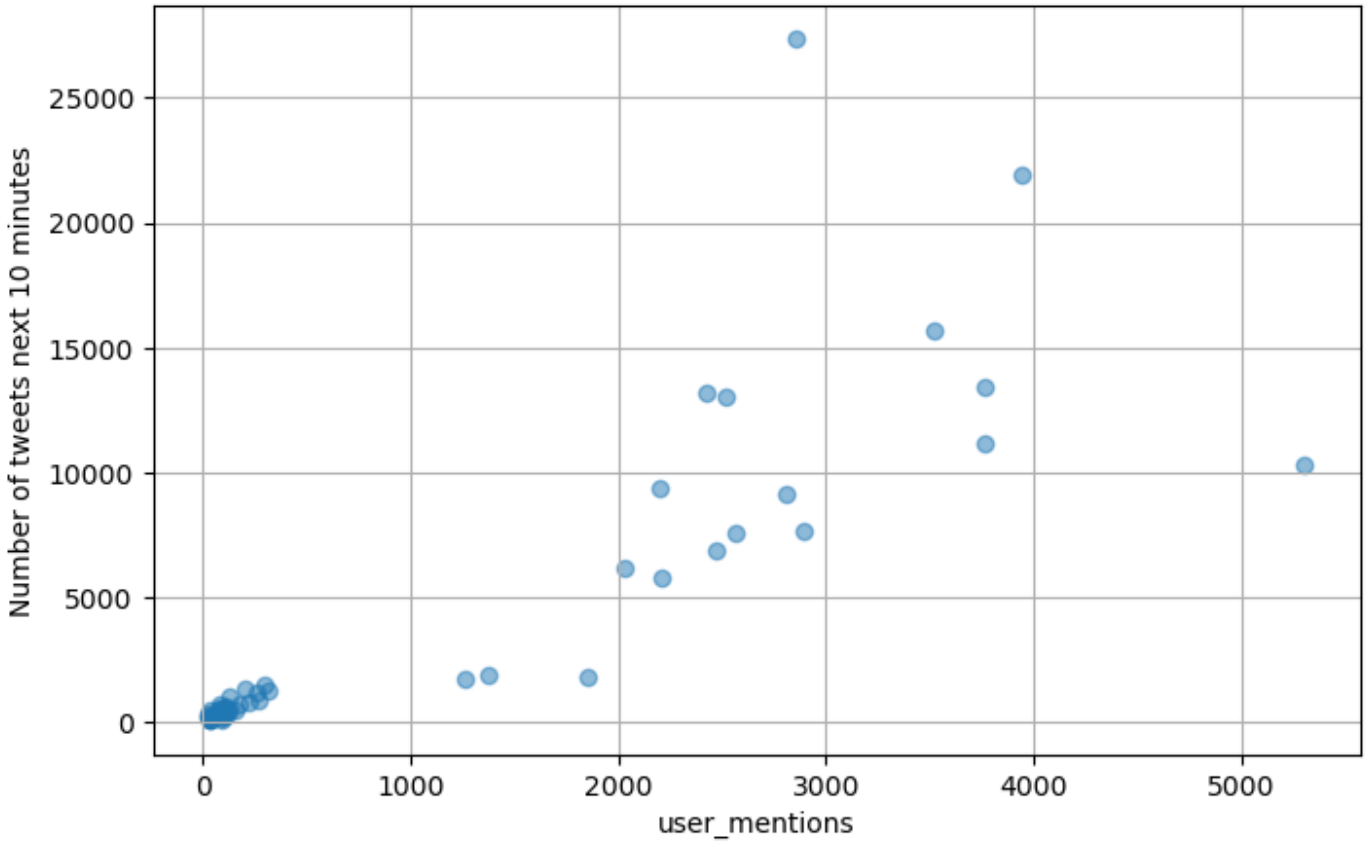
In [14]:

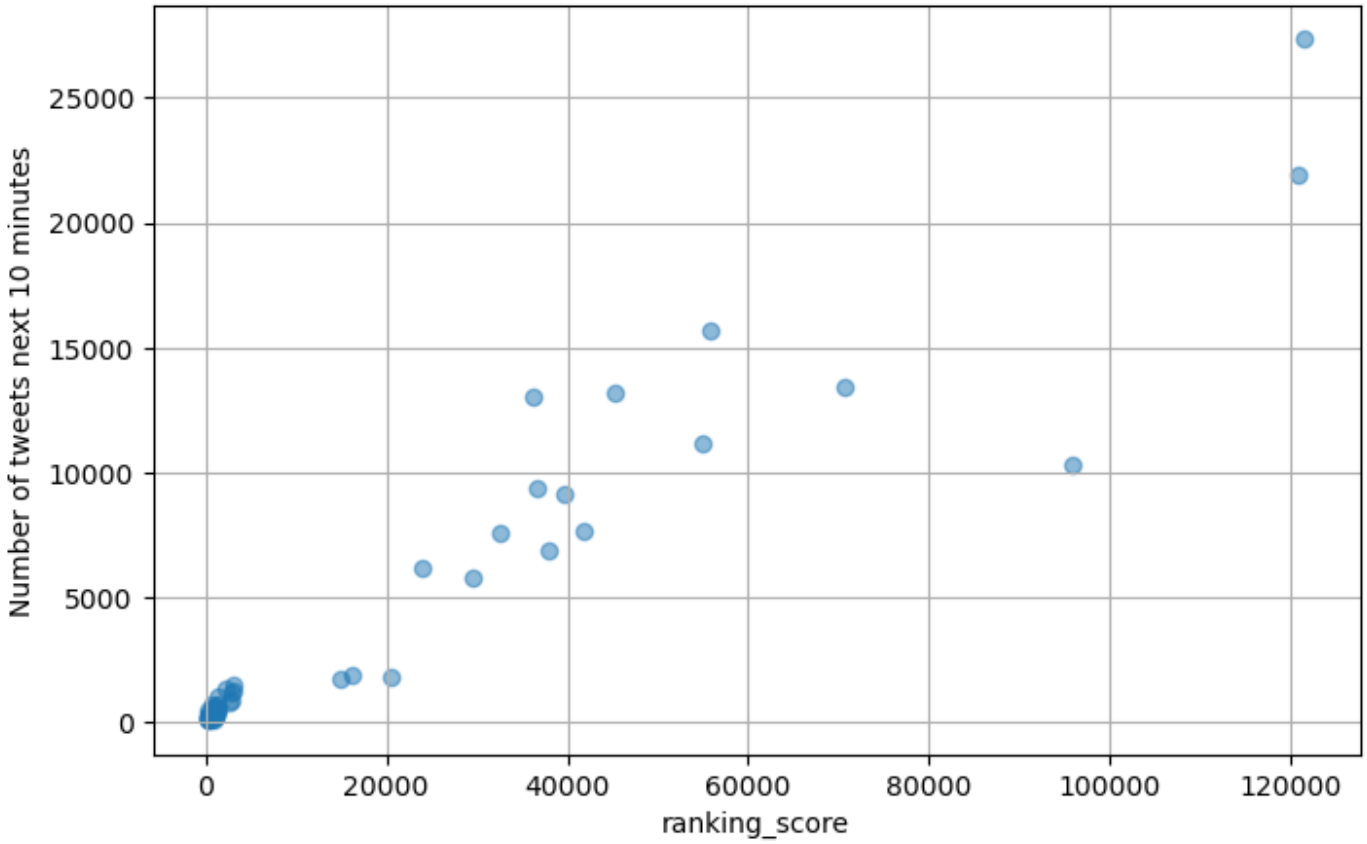
```
import statsmodels.api as sm
from sklearn import metrics
import numpy as np
import matplotlib.pyplot as plt

feature_names = list(superbowl_x.columns)
print(feature_names)

def scatter_plot(features, y_pred, pvalues, feature_names):
    # Obtain the indices that would sort the p-values in ascending order
    ranked_index = np.argsort(pvalues)
    print(ranked_index)
    for i in range(9):
        plt.figure(figsize = (8,5))
```

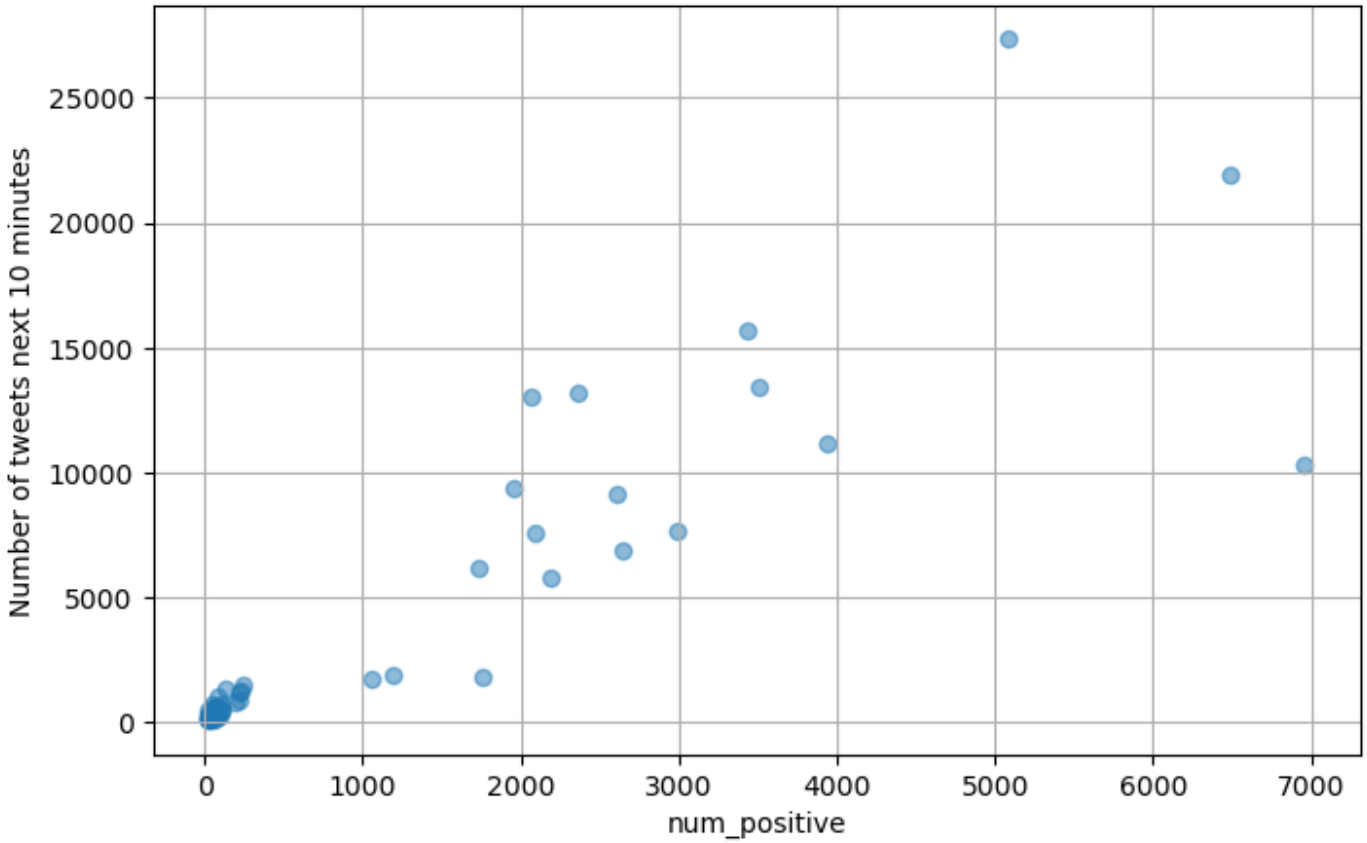
```
        # Create a scatter plot of the ith feature against the predicted values
        plt.scatter(features.iloc[:,ranked_index[i]], y_pred, alpha=0.5)
        plt.xlabel(feature_names[ranked_index[i]])
        plt.ylabel("Number of tweets next 10 minutes")
        plt.grid(True)
        plt.show()
    print('-' * 80)

# Fit a linear regression model to the data and obtain the predicted values and p-values
lr_fit = sm.OLS(superbowl_y,superbowl_x).fit()
y_pred = lr_fit.predict()
pvalues = lr_fit.pvalues
print('MSE: ', metrics.mean_squared_error(superbowl_y, y_pred))
print(lr_fit.summary())
scatter_plot(superbowl_x, y_pred, pvalues, feature_names)
print('\n')
```

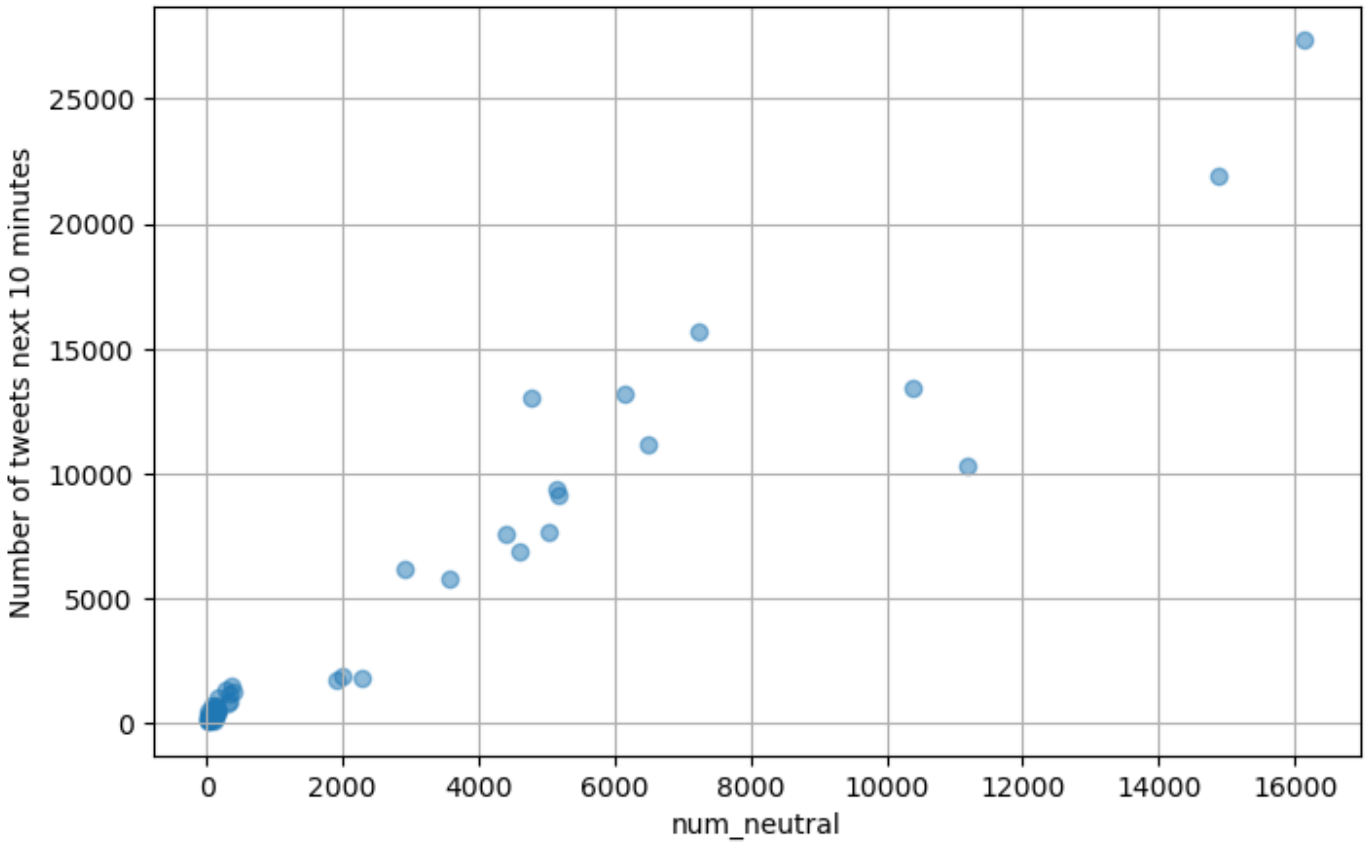
```
['num_retweet', 'num_followers', 'ranking_score', 'user_activity', 'user_mentions', 'num
_positive', 'num_neutral', 'num_negative', 'unique_user_id']
MSE:  6352334.382528166
                                 OLS Regression Results
================================================================================================
Dep. Variable:                          y   R-squared (uncentered):                   0.897
Model:                                OLS   Adj. R-squared (uncentered):              0.873
Method:                     Least Squares   F-statistic:                              37.75
Date:                    Tue, 14 Mar 2023   Prob (F-statistic):                    1.61e-16
Time:                            15:17:58   Log-Likelihood:                         -444.05
No. Observations:                      48   AIC:                                      906.1
Df Residuals:                          39   BIC:                                      922.9
Df Model:                               9
Covariance Type:                nonrobust
================================================================================================
                     coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------------------------
num_retweet       -0.0505      0.160     -0.316      0.754      -0.374       0.273
num_followers  -1.174e-05   4.38e-05     -0.268      0.790      -0.000    7.69e-05
ranking_score      5.6989      2.266      2.515      0.016       1.116      10.282
user_activity     -0.0155      0.037     -0.423      0.675      -0.090       0.059
user_mentions      8.3991      2.790      3.011      0.005       2.757      14.042
num_positive     -35.5276     15.380     -2.310      0.026     -66.638      -4.418
num_neutral      -29.0873     13.798     -2.108      0.042     -56.997      -1.177
num_negative     -19.7659     12.493     -1.582      0.122     -45.034       5.503
unique_user_id     3.0421      7.477      0.407      0.686     -12.082      18.166
==============================================================================
Omnibus:                       59.300   Durbin-Watson:                   2.410
Prob(Omnibus):                  0.000   Jarque-Bera (JB):              555.454
Skew:                           2.935   Prob(JB):                    2.42e-121
Kurtosis:                      18.597   Cond. No.                     3.62e+06
==============================================================================

Notes:
[1] R² is computed without centering (uncentered) since the model does not contain a con
stant.
[2] Standard Errors assume that the covariance matrix of the errors is correctly specifi
ed.
[3] The condition number is large, 3.62e+06. This might indicate that there are
strong multicollinearity or other numerical problems.
num_retweet       4
num_followers     2
ranking_score     5
user_activity     6
user_mentions     7
num_positive      3
num_neutral       8
num_negative      0
```

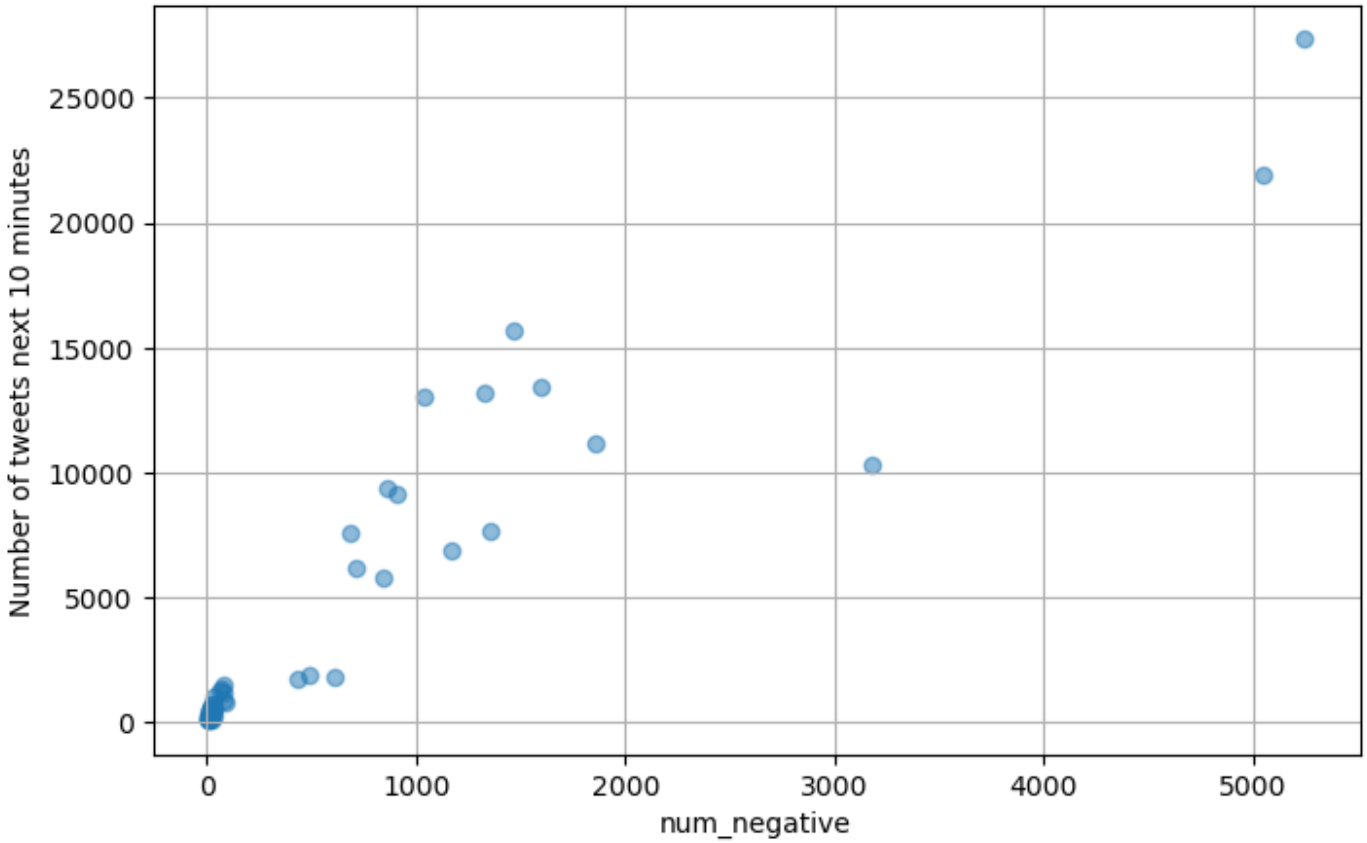
```
unique_user_id    1
dtype: int64
```

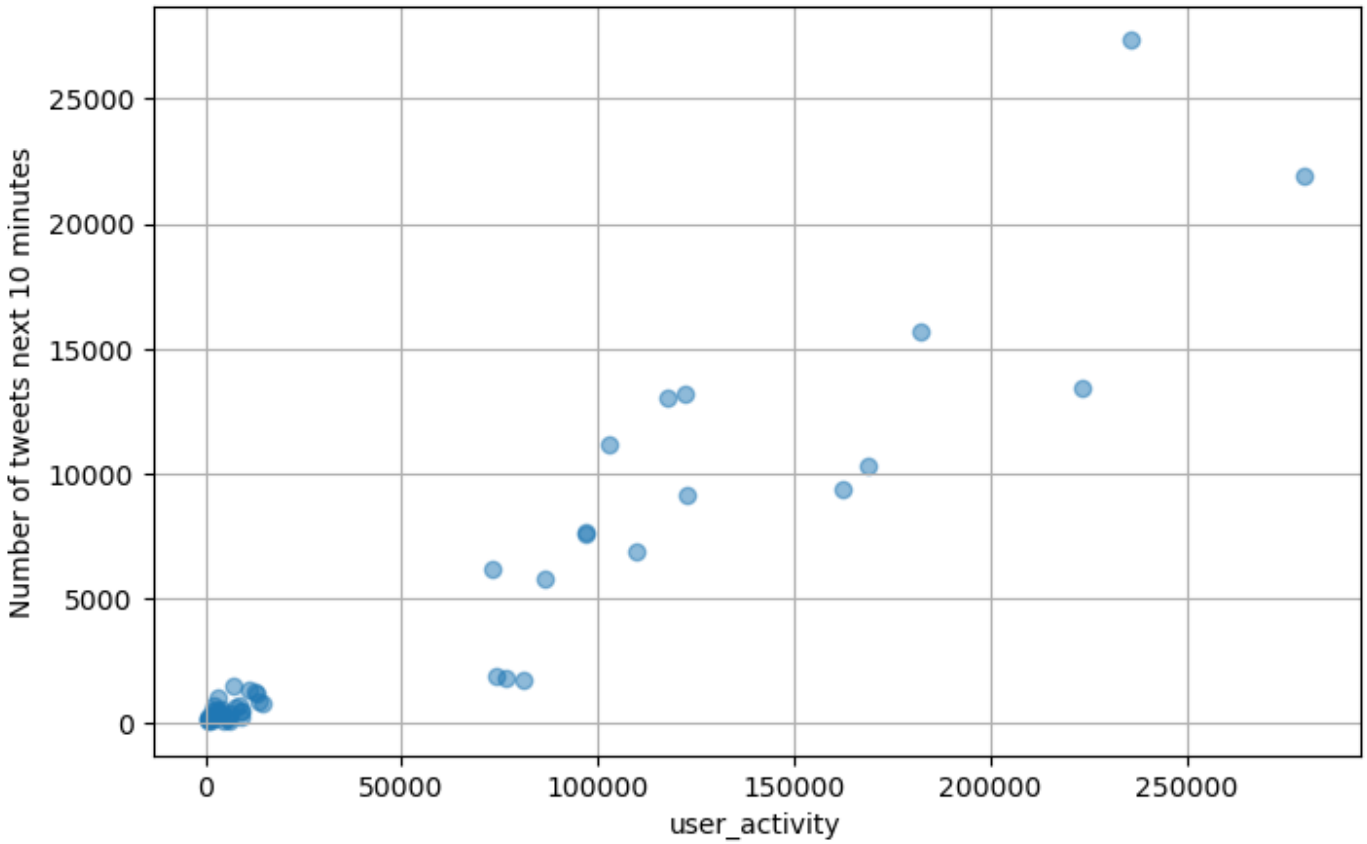


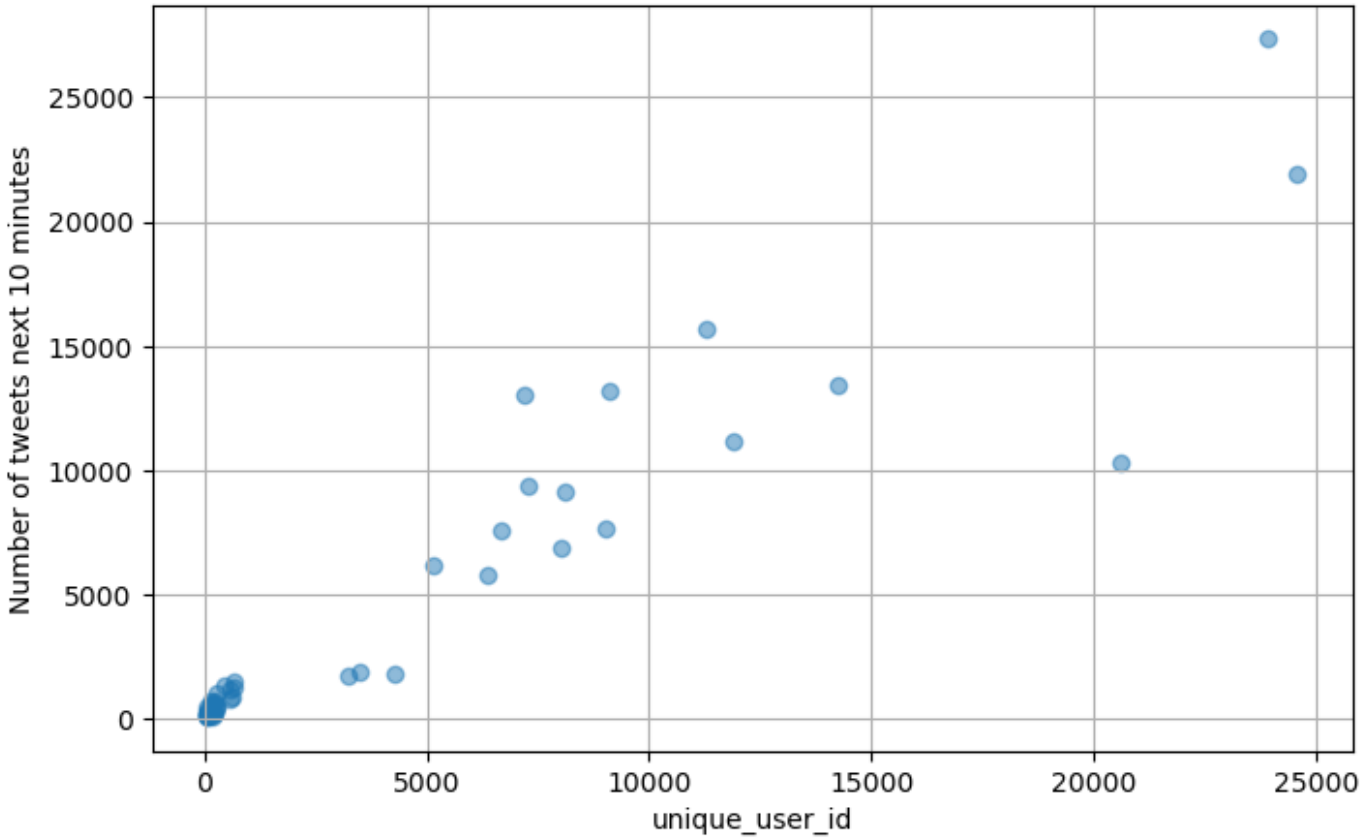


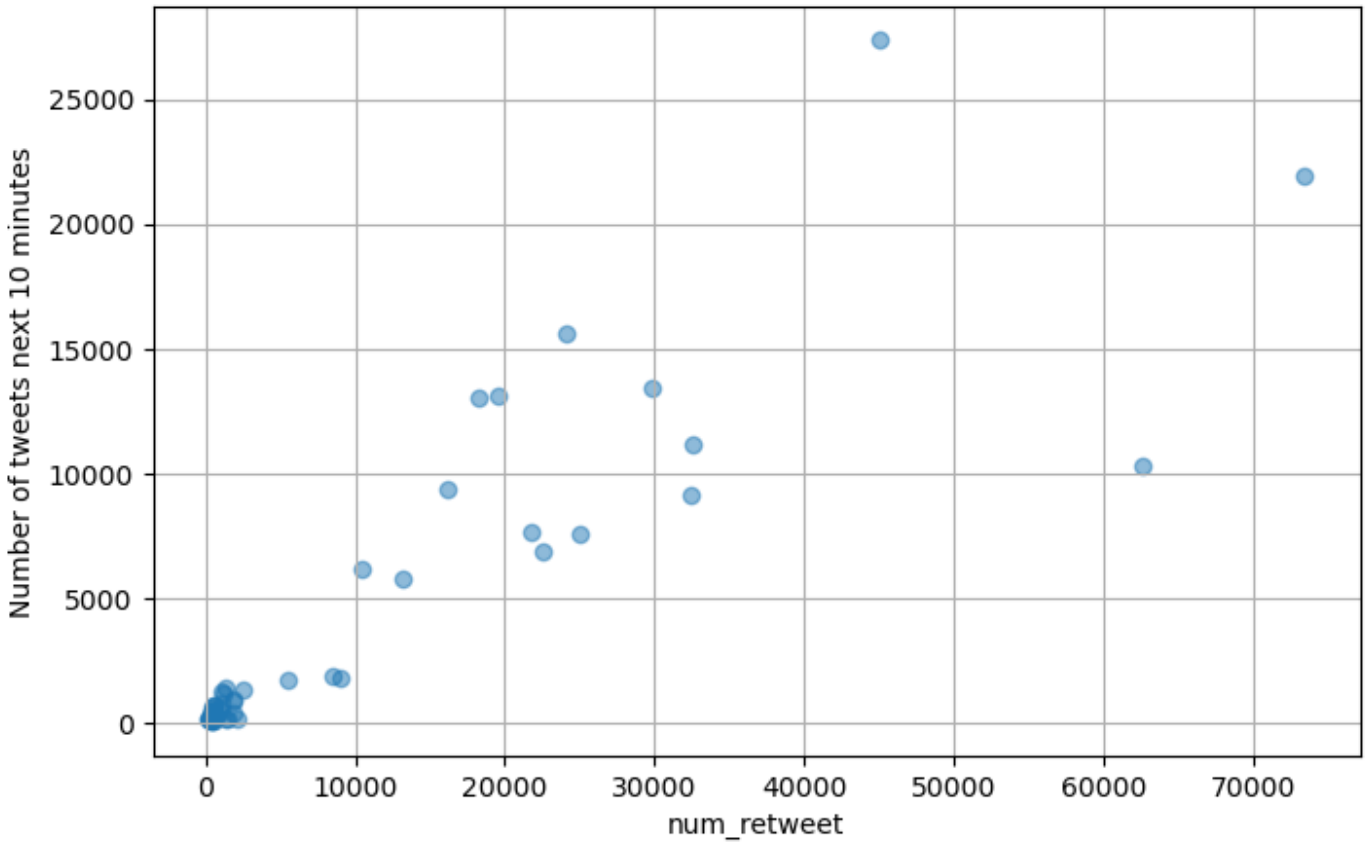

Number of tweets next 10 minutes
25000
20000
15000
10000
5000
0
0
1000
2000
3000
4000
5000
6000
7000
num_positive

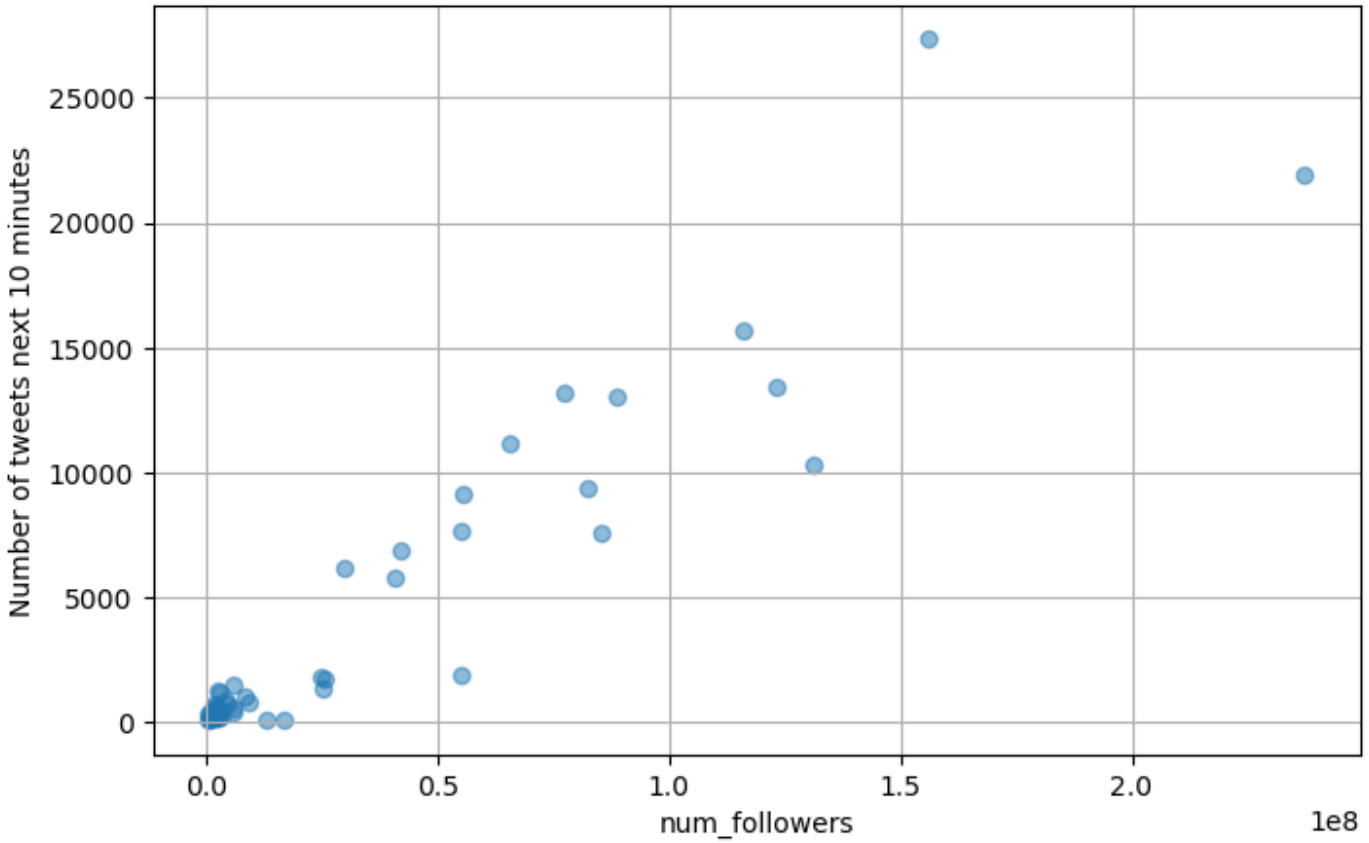

Number of tweets next 10 minutes
25000
20000
15000
10000
5000
0
0
2000
4000
6000
8000
10000
12000
14000
16000
num_neutral


Number of tweets next 10 minutes
25000
20000
15000
10000
5000
0
0
1000
2000
3000
4000
5000
num_negative


Number of tweets next 10 minutes
25000
20000
15000
10000
5000
0
0
50000
100000
150000
200000
250000
user_activity


Number of tweets next 10 minutes
25000
20000
15000
10000
5000
0
0
5000
10000
15000
20000
25000
unique_user_id


Number of tweets next 10 minutes
25000
20000
15000
10000
5000
0
0
10000
20000
30000
40000
50000
60000
70000
num_retweet



In [15]:
```
from sklearn.preprocessing import StandardScaler
from sklearn import preprocessing

superbowl_merge_data_standard_x = pd.DataFrame(preprocessing.scale(superbowl_merge_data_
# superbowl_merge_data_standard_x = superbowl_merge_data_drop
```

In [16]:
```
superbowl_merge_data_standard_x = superbowl_merge_data_standard_x.drop(superbowl_merge_d
```

In [17]:
```
print(superbowl_merge_data_standard_x.shape)
```

```
(48, 9)
```

In [18]:
```
superbowl_merge_data_standard_y =  pd.DataFrame(preprocessing.scale(superbowl_y),columns
print(superbowl_merge_data_standard_y.shape)
```

```
(48,)
```

In [24]:
```
from sklearn.feature_selection import SelectKBest, mutual_info_regression, f_regression
import numpy as np

# Define a function that selects the top n most important features using mutual informat
def select_topn_important_features(X, Y, n):
  Mutual_ = mutual_info_regression(X, Y)
  F_ = f_regression(X, Y)

  # Select the top n features based on their mutual information and F-test scores
  topn_M = np.argsort(Mutual_)[::-1][:n]
  topn_F = np.argsort(F_[0])[::-1][:n]

  # Sort all the features based on their mutual information and F-test scores
  all_m = np.argsort(Mutual_)[::-1]
  all_f = np.argsort(F_[0])[::-1]

  # Extract the top n features and all features based on their mutual information scores
```

```
  X_topn_M = X.iloc[:, topn_M]
  X_topn_F = X.iloc[:, topn_F]

  # Extract the top n features and all features based on their F-test scores
  all_m_ = X.iloc[:, all_m]
  all_f_ = X.iloc[:, all_f]

  return X_topn_M, X_topn_F
```

In [22]:
```
superbowl_top3_M, superbowl_top3_F, sall_m, sall_f = select_topn_important_features(supe
```

In [23]:
```
print("superbowl Top3 by mutual_info_regression:")
print(superbowl_top3_M.columns)

print("superbowl Top3 by f_regression:")
print(superbowl_top3_F.columns)

print("superbowl all by mutual_info_regression:")
print(sall_m.columns)

print("superbowl all by f_regression:")
print(sall_f.columns)
```

```
superbowl Top3 by mutual_info_regression:
Index(['num_neutral', 'ranking_score', 'num_positive'], dtype='object')
superbowl Top3 by f_regression:
Index(['num_neutral', 'ranking_score', 'unique_user_id'], dtype='object')
superbowl all by mutual_info_regression:
Index(['num_neutral', 'ranking_score', 'num_positive', 'unique_user_id',
       'user_mentions', 'user_activity', 'num_negative', 'num_retweet',
       'num_followers'],
      dtype='object')
superbowl all by f_regression:
Index(['num_neutral', 'ranking_score', 'unique_user_id', 'user_activity',
       'num_followers', 'num_negative', 'num_positive', 'num_retweet',
       'user_mentions'],
      dtype='object')
```

In [25]:
```
# now we are do experiments of exactly how many features we need to select

#with selection
superbowl_rmse_lr_m = []  #superbowl rmse score for linear regression and with mutual_in
superbowl_rmse_lr_f = []
superbowl_rmse_r_m = []
superbowl_rmse_r_f = []
superbowl_rmse_la_m = []
superbowl_rmse_la_f = []

#without selection
superbowl_rmse_lr = [] #superbowl rmse score for linear regression
superbowl_rmse_r = []
superbowl_rmse_la = []
```

In [26]:
```
from sklearn.model_selection import cross_validate,GridSearchCV
from sklearn.linear_model import LinearRegression, Ridge, Lasso

for k in range(1, 10):
    superbowl_topk_M, superbowl_topk_F = select_topn_important_features(superbowl_merge_
    #superbowl rmse score for linear regression
    score_ = cross_validate(LinearRegression(), superbowl_merge_data_standard_x, superbo
    score__ = score_['test_neg_root_mean_squared_error'].mean()
    print(f"superbowl rmse score for linear regression, {score__:.4f}")
    superbowl_rmse_lr.append(score__)

    #superbowl rmse score for linear regression and with mutual_info_regression
```

```
    score_ = cross_validate(LinearRegression(), superbowl_topk_M, superbowl_merge_data_s
    score__ = score_['test_neg_root_mean_squared_error'].mean()
    print(f"superbowl rmse score for linear regression and with mutual_info_regression,
    superbowl_rmse_lr_m.append(score__)

    #superbowl rmse score for linear regression and with f_regression
    score_ = cross_validate(LinearRegression(), superbowl_topk_F, superbowl_merge_data_s
    score__ = score_['test_neg_root_mean_squared_error'].mean()
    print(f"superbowl rmse score for linear regression and with f_regression, {score__:.
    superbowl_rmse_lr_f.append(score__)

    #superbowl rmse score for Ridge regression
    score_ = cross_validate(Ridge(), superbowl_merge_data_standard_x, superbowl_merge_da
    score__ = score_['test_neg_root_mean_squared_error'].mean()
    print(f"superbowl rmse score for Ridge regression, {score__:.4f}")
    superbowl_rmse_r.append(score__)

    #superbowl rmse score for Ridge regression and with mutual_info_regression
    score_ = cross_validate(Ridge(), superbowl_topk_M, superbowl_merge_data_standard_y,
    score__ = score_['test_neg_root_mean_squared_error'].mean()
    print(f"superbowl rmse score for Ridge regression and with mutual_info_regression, {
    superbowl_rmse_r_m.append(score__)

    #superbowl rmse score for Ridge regression and with f_regression
    score_ = cross_validate(Ridge(), superbowl_topk_F, superbowl_merge_data_standard_y,
    score__ = score_['test_neg_root_mean_squared_error'].mean()
    print(f"superbowl rmse score for Ridge regression and with f_regression, {score__:.4
    superbowl_rmse_r_f.append(score__)

    #superbowl rmse score for Lasso regression
    score_ = cross_validate(Lasso(), superbowl_merge_data_standard_x, superbowl_merge_da
    score__ = score_['test_neg_root_mean_squared_error'].mean()
    print(f"superbowl rmse score for Lasso regression, {score__:.4f}")
    superbowl_rmse_la.append(score__)

    #superbowl rmse score for Lasso regression and with mutual_info_regression
    score_ = cross_validate(Lasso(), superbowl_topk_M, superbowl_merge_data_standard_y,
    score__ = score_['test_neg_root_mean_squared_error'].mean()
    print(f"superbowl rmse score for Lasso regression and with mutual_info_regression, {
    superbowl_rmse_la_m.append(score__)

    #superbowl rmse score for Lasso regression and with f_regression
    score_ = cross_validate(Lasso(), superbowl_topk_F, superbowl_merge_data_standard_y,
    score__ = score_['test_neg_root_mean_squared_error'].mean()
    print(f"superbowl rmse score for Lasso regression and with f_regression, {score__:.4
    superbowl_rmse_la_f.append(score__)
```

```
superbowl rmse score for linear regression, -0.9262
superbowl rmse score for linear regression and with mutual_info_regression, -0.2540 top1
superbowl rmse score for linear regression and with f_regression, -0.2540 top1
superbowl rmse score for Ridge regression, -0.2603
superbowl rmse score for Ridge regression and with mutual_info_regression, -0.2578 top1
superbowl rmse score for Ridge regression and with f_regression, -0.2578 top1
superbowl rmse score for Lasso regression, -0.8720
superbowl rmse score for Lasso regression and with mutual_info_regression, -0.8720 top1
superbowl rmse score for Lasso regression and with f_regression, -0.8720 top1
superbowl rmse score for linear regression, -0.9262
superbowl rmse score for linear regression and with mutual_info_regression, -0.2640 top2
superbowl rmse score for linear regression and with f_regression, -0.2640 top2
superbowl rmse score for Ridge regression, -0.2603
superbowl rmse score for Ridge regression and with mutual_info_regression, -0.2609 top2
superbowl rmse score for Ridge regression and with f_regression, -0.2609 top2
superbowl rmse score for Lasso regression, -0.8720
superbowl rmse score for Lasso regression and with mutual_info_regression, -0.8720 top2
superbowl rmse score for Lasso regression and with f_regression, -0.8720 top2
superbowl rmse score for linear regression, -0.9262
```

```
superbowl rmse score for linear regression and with mutual_info_regression, -0.3205 top3
superbowl rmse score for linear regression and with f_regression, -0.3010 top3
superbowl rmse score for Ridge regression, -0.2603
superbowl rmse score for Ridge regression and with mutual_info_regression, -0.2661 top3
superbowl rmse score for Ridge regression and with f_regression, -0.2622 top3
superbowl rmse score for Lasso regression, -0.8720
superbowl rmse score for Lasso regression and with mutual_info_regression, -0.8720 top3
superbowl rmse score for Lasso regression and with f_regression, -0.8720 top3
superbowl rmse score for linear regression, -0.9262
superbowl rmse score for linear regression and with mutual_info_regression, -0.3434 top4
superbowl rmse score for linear regression and with f_regression, -0.3009 top4
superbowl rmse score for Ridge regression, -0.2603
superbowl rmse score for Ridge regression and with mutual_info_regression, -0.2676 top4
superbowl rmse score for Ridge regression and with f_regression, -0.2541 top4
superbowl rmse score for Lasso regression, -0.8720
superbowl rmse score for Lasso regression and with mutual_info_regression, -0.8720 top4
superbowl rmse score for Lasso regression and with f_regression, -0.8720 top4
superbowl rmse score for linear regression, -0.9262
superbowl rmse score for linear regression and with mutual_info_regression, -0.3343 top5
superbowl rmse score for linear regression and with f_regression, -0.3060 top5
superbowl rmse score for Ridge regression, -0.2603
superbowl rmse score for Ridge regression and with mutual_info_regression, -0.2545 top5
superbowl rmse score for Ridge regression and with f_regression, -0.2526 top5
superbowl rmse score for Lasso regression, -0.8720
superbowl rmse score for Lasso regression and with mutual_info_regression, -0.8720 top5
superbowl rmse score for Lasso regression and with f_regression, -0.8720 top5
superbowl rmse score for linear regression, -0.9262
superbowl rmse score for linear regression and with mutual_info_regression, -0.3576 top6
superbowl rmse score for linear regression and with f_regression, -0.5294 top6
superbowl rmse score for Ridge regression, -0.2603
superbowl rmse score for Ridge regression and with mutual_info_regression, -0.2546 top6
superbowl rmse score for Ridge regression and with f_regression, -0.2607 top6
superbowl rmse score for Lasso regression, -0.8720
superbowl rmse score for Lasso regression and with mutual_info_regression, -0.8720 top6
superbowl rmse score for Lasso regression and with f_regression, -0.8720 top6
superbowl rmse score for linear regression, -0.9262
superbowl rmse score for linear regression and with mutual_info_regression, -0.8667 top7
superbowl rmse score for linear regression and with f_regression, -0.6752 top7
superbowl rmse score for Ridge regression, -0.2603
superbowl rmse score for Ridge regression and with mutual_info_regression, -0.2549 top7
superbowl rmse score for Ridge regression and with f_regression, -0.2674 top7
superbowl rmse score for Lasso regression, -0.8720
superbowl rmse score for Lasso regression and with mutual_info_regression, -0.8720 top7
superbowl rmse score for Lasso regression and with f_regression, -0.8720 top7
superbowl rmse score for linear regression, -0.9262
superbowl rmse score for linear regression and with mutual_info_regression, -0.8686 top8
superbowl rmse score for linear regression and with f_regression, -0.7697 top8
superbowl rmse score for Ridge regression, -0.2603
superbowl rmse score for Ridge regression and with mutual_info_regression, -0.2634 top8
superbowl rmse score for Ridge regression and with f_regression, -0.2713 top8
superbowl rmse score for Lasso regression, -0.8720
superbowl rmse score for Lasso regression and with mutual_info_regression, -0.8720 top8
superbowl rmse score for Lasso regression and with f_regression, -0.8720 top8
superbowl rmse score for linear regression, -0.9262
superbowl rmse score for linear regression and with mutual_info_regression, -0.9262 top9
superbowl rmse score for linear regression and with f_regression, -0.9262 top9
superbowl rmse score for Ridge regression, -0.2603
superbowl rmse score for Ridge regression and with mutual_info_regression, -0.2603 top9
superbowl rmse score for Ridge regression and with f_regression, -0.2603 top9
superbowl rmse score for Lasso regression, -0.8720
superbowl rmse score for Lasso regression and with mutual_info_regression, -0.8720 top9
superbowl rmse score for Lasso regression and with f_regression, -0.8720 top9
```

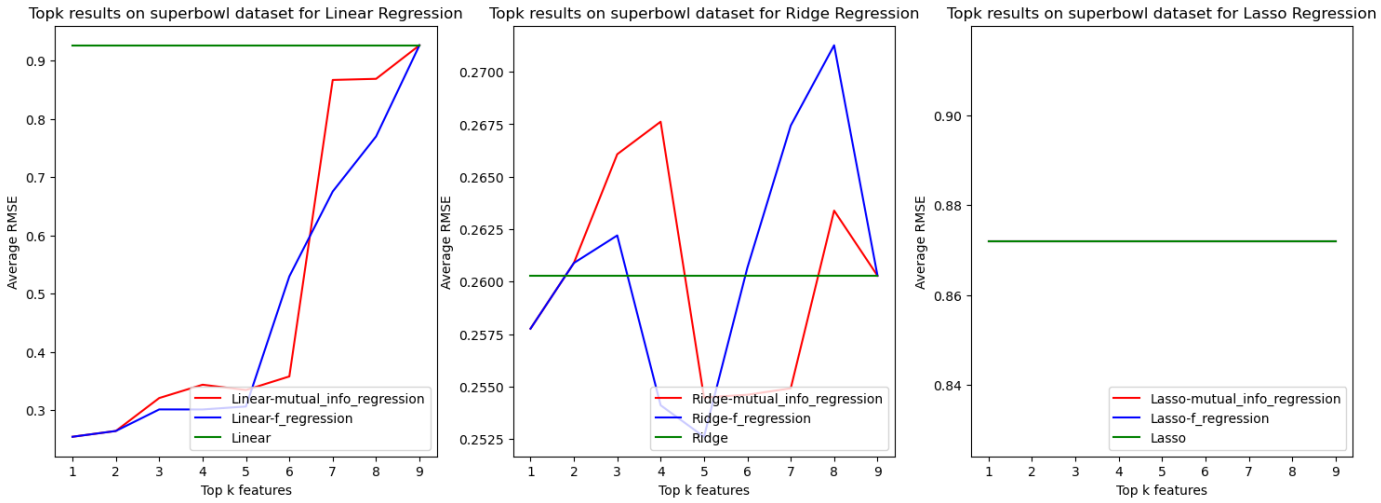
In [27]:

```
import matplotlib.pyplot as plt
fig, axes = plt.subplots(1, 3, figsize=(18, 6))
```

```
#plot superbowl linear
axes[0].plot(np.arange(1, len(superbowl_rmse_lr_m) + 1, 1), np.negative(superbowl_rmse_l
axes[0].plot(np.arange(1, len(superbowl_rmse_lr_f) + 1, 1), np.negative(superbowl_rmse_l
axes[0].plot(np.arange(1, len(superbowl_rmse_lr) + 1, 1), np.negative(superbowl_rmse_lr)

axes[0].legend(loc='lower right')
axes[0].set(xlabel='Top k features',ylabel='Average RMSE')
axes[0].set_title('Topk results on superbowl dataset for Linear Regression')

#plot superbowl Ridge
axes[1].plot(np.arange(1, len(superbowl_rmse_r_m) + 1, 1), np.negative(superbowl_rmse_r_
axes[1].plot(np.arange(1, len(superbowl_rmse_r_f) + 1, 1), np.negative(superbowl_rmse_r_
axes[1].plot(np.arange(1, len(superbowl_rmse_r) + 1, 1), np.negative(superbowl_rmse_r),

axes[1].legend(loc='lower right')
axes[1].set(xlabel='Top k features',ylabel='Average RMSE')
axes[1].set_title('Topk results on superbowl dataset for Ridge Regression')

#plot superbowl Lasso
axes[2].plot(np.arange(1, len(superbowl_rmse_la_m) + 1, 1), np.negative(superbowl_rmse_l
axes[2].plot(np.arange(1, len(superbowl_rmse_la_f) + 1, 1), np.negative(superbowl_rmse_l
axes[2].plot(np.arange(1, len(superbowl_rmse_la) + 1, 1), np.negative(superbowl_rmse_la)

axes[2].legend(loc='lower right')
axes[2].set(xlabel='Top k features',ylabel='Average RMSE')
axes[2].set_title('Topk results on superbowl dataset for Lasso Regression')
```

Out[27]:

```
Text(0.5, 1.0, 'Topk results on superbowl dataset for Lasso Regression')
```



In [28]:

```
# Finding the optimal penalty parameter
from joblib import Memory
from sklearn.pipeline import Pipeline

location = "cachedir"
memory = Memory(location=location, verbose=10)

pipe_ = Pipeline([
    ('kbest', SelectKBest()),
    ('model', "passthrough")
], memory = memory)

param_grid = [{
        'kbest__score_func': (mutual_info_regression, f_regression),
        'kbest__k': (1, 2, 3, 4, 5, 6, 7, 8, 9),
        'model': [Ridge(), Lasso()],
        'model__alpha': [10.0**x for x in np.arange(-3,4)]
    }
]
```

In [29]:
```
grid_superbowl = GridSearchCV(pipe_, param_grid = param_grid, cv = 10, n_jobs = -1, verb
                    scoring = 'neg_root_mean_squared_error', return_train_score = True)
```

```
Fitting 10 folds for each of 252 candidates, totalling 2520 fits
[Memory] Calling sklearn.pipeline._fit_transform_one...
_fit_transform_one(SelectKBest(k=6,
            score_func=<function mutual_info_regression at 0x00000131A4696310>),
    num_retweet  num_followers  ranking_score  user_activity  user_mentions  \
0      0.495734       1.113654       0.572525       0.900956       1.019180
1      0.855501       1.670369       1.218276       1.804127       1.727762
2      1.371230       0.452222       0.682440       0.969836       1.221328
3      0.911773       1.052804       0.448520       0.612078       1.047551
4      0.371322       0.986070       0.584481       1.527609       0.788660
5      1.210165       1.810932       1.706483       2.375256       1.901538
6      0.581894       0.889884       0.869615       0.962788       0.947541
7      2.147713       2.470791       3.385532       2.546099       1...,
array([ 1.199476,  0.684041,  0.460154,  0.575458,  1.698332,  0.858342,
        3.341869,  3.331713,  2.572824,  1.222776,  0.785904,  0.644611,
        0.375169,  0.18877 ,  0.082726, -0.060957, -0.102926, -0.541442,
       -0.571762, -0.590431, -0.592074, -0.590133, -0.589087, -0.519934,
       -0.510077, -0.510674, -0.526058, -0.5162  , -0.580275, -0.601932,
       -0.60238 , -0.599691, -0.602529, -0.604172, -0.602679, -0.603575,
       -0.603127, -0.605218, -0.581918, -0.567579, -0.570716, -0.576392,
       -0.570268, -0.576392, -0.576392, -0.583262, -0.57908 , -0.612835]),
None, message_clsname='Pipeline', message=None)
_____________________________________________fit_transform_one - 0.0s, 0.0min
```

Try the 1 minutes interval

In [58]:
```
# This function takes in a train object and a feature object, and merges the feature obj
def merge_feature_into_train_data(train_obj, feature):
    train_obj["num_tweet"] += feature["num_tweet"]
    train_obj["num_retweet"] += feature["num_retweet"]
    train_obj["num_followers"] += feature["num_followers"]
    train_obj["ranking_score"] += feature["ranking_score"]
    train_obj["user_activity"] += feature["user_activity"]
    train_obj["user_id"].add(feature["user_id"])
    train_obj["user_location"] += feature["user_location"]
    train_obj["user_mentions"] += feature["user_mentions"]
    train_obj["num_positive"] += feature["positive"]
    train_obj["num_neutral"] += feature["neutral"]
    train_obj["num_negative"] += feature["negative"]
    train_obj["unique_user_id"] = len(train_obj["user_id"])


def merge_different_intervals(): # for SuperBowl
    time_delta = int(timedelta(minutes = 1).seconds)

    num_train_data = (superbowl_end_time - superbowl_start_time) // time_delta + 1

    train_data = [{
        "range_start": superbowl_start_time + i*time_delta,
        "range_end": superbowl_start_time + (i+1)*time_delta,
        "num_tweet": 0,
        "num_retweet": 0,
        "num_followers": 0,
        "ranking_score": 0,
        "user_activity": 0,
        "user_id": set(),
        "user_location": "",
        "user_mentions": 0,
        "num_positive": 0,
        "num_neutral": 0,
```

```
        "num_negative": 0,
        "text": "",
        "polarity": 0
    }
    for i in range(num_train_data)]

    for idx, row in superbowl_data.iterrows():
        # Calculate which time slot this tweet falls into
        if superbowl_start_time <= row["created_at"] <= superbowl_end_time:
            index = (row["created_at"]- superbowl_start_time) // time_delta
            # Merge the feature into the appropriate slot in the training data
            merge_feature_into_train_data(train_data[index], row)
    return pd.DataFrame(train_data)
```

In [59]:
```
superbowl_merge_1 = merge_different_intervals()
superbowl_merge_1.to_csv('superbowl_merge_1.csv', index=False)
```

In [61]:
```
superbowl_merge_1.shape
```

Out[61]:
```
(241, 16)
```

In [62]:
```
superbowl_merge_data_drop = superbowl_merge_1.copy().drop(['range_start','range_end','us
```

In [63]:
```
superbowl_x = superbowl_merge_data_drop.drop(superbowl_merge_data_drop.index[-1])
superbowl_y = superbowl_merge_1["num_tweet"]
superbowl_y = superbowl_y.drop(superbowl_y.index[0])
superbowl_y = pd.DataFrame(superbowl_y,columns = ["num_tweet"]).values.ravel()
```

In [64]:
```
superbowl_x
```

Out[64]:

	num_retweet	num_followers	ranking_score	user_activity	user_mentions	num_positive	num_neutr
0	4071	6320414.0	7796.709831	22453.355038	579	480	10
1	3034	20295517.0	6476.715966	25663.899065	497	366	8
2	4334	18597858.0	5822.767599	19172.983770	432	383	7
3	3440	23712225.0	6849.486202	24109.103574	481	432	8
4	3320	19579440.0	9386.665890	26187.226104	533	407	12
...	...	...	...	...	...	...	
235	63	101773.0	187.321615	721.504420	19	20	
236	100	131757.0	253.560633	1591.937729	37	23	
237	65	135456.0	226.558879	2228.397522	31	15	
238	57	36182.0	149.703310	422.127708	13	9	
239	102	104743.0	212.989823	1444.595889	16	14	

240 rows × 9 columns

In [66]:
```
# RandomForest GridSearch
import pandas as pd
from sklearn.pipeline import Pipeline
from sklearn.model_selection import KFold, GridSearchCV
from sklearn.preprocessing import StandardScaler
from sklearn.ensemble import RandomForestRegressor


pipe_rf = Pipeline([
```

```
    ('standardize', StandardScaler()),
    ('model', RandomForestRegressor(random_state=42))
])

param_grid = {
    'model__max_depth': [10, 30, 50, 70, 100, 200],
    'model__max_features': ['auto', 'sqrt'],
    'model__min_samples_leaf': [1, 2, 3],
    'model__min_samples_split': [2, 5, 10],
    'model__n_estimators': [200, 400]
}

grid_rf = GridSearchCV(pipe_rf, param_grid=param_grid, cv=KFold(5, shuffle=True, random_
                       scoring='neg_mean_squared_error').fit(superbowl_x, superbowl_y)
result_rf = pd.DataFrame(grid_rf.cv_results_)[['mean_test_score', 'param_model__max_dept
                                               'param_model__min_samples_leaf', 'param_mod
                                               'param_model__n_estimators']]
result_rf = result_rf.sort_values(by=['mean_test_score'], ascending=False).reset_index(d
result_rf.head()
```

```
Fitting 5 folds for each of 216 candidates, totalling 1080 fits
```

Out[66]:

	mean_test_score	param_model__max_depth	param_model__max_features	param_model__min_samples
0	-161354.634897	10	auto	
1	-162792.741378	10	auto	
2	-163112.769889	30	auto	
3	-163112.769889	200	auto	
4	-163112.769889	70	auto	

In [90]:

```
# GradientBoosting GridSearch
import pandas as pd
from sklearn.pipeline import Pipeline
from sklearn.model_selection import KFold, GridSearchCV
from sklearn.preprocessing import StandardScaler
from sklearn.ensemble import GradientBoostingRegressor


pipe_gb = Pipeline([
    ('standardize', StandardScaler()),
    ('model', GradientBoostingRegressor(random_state=42))
])

param_grid = {
    'model__max_depth': [10, 30, 50, 70, 100, 200],
    'model__max_features': ['auto', 'sqrt'],
    'model__min_samples_leaf': [1, 2, 3],
    'model__min_samples_split': [2, 5, 10],
    'model__n_estimators': [200, 400]
}


grid_gb = GridSearchCV(pipe_gb, param_grid=param_grid, cv=KFold(5, shuffle=True, random_
                       scoring='neg_mean_squared_error').fit(superbowl_x, superbowl_y)
result_gb = pd.DataFrame(grid_gb.cv_results_)[['mean_test_score', 'param_model__max_dept
                                               'param_model__min_samples_leaf', 'param_mod
                                               'param_model__n_estimators']]
result_gb = result_gb.sort_values(by=['mean_test_score'], ascending=False).reset_index(d
result_gb.head()
```

```
Fitting 5 folds for each of 216 candidates, totalling 1080 fits
```

Out[90]:

	mean_test_score	param_model__max_depth	param_model__max_features	param_model__min_samples

0	-137558.444653	50	sqrt
1	-137558.444653	100	sqrt
2	-137558.444653	70	sqrt
3	-137558.444653	200	sqrt
4	-137558.459325	30	sqrt

In [92]:
```
# NeuralNetwork GridSearch

import pandas as pd
from sklearn.pipeline import Pipeline
from sklearn.model_selection import KFold, GridSearchCV
from sklearn.preprocessing import StandardScaler
from sklearn.neural_network import MLPRegressor

pipe_nn_noscale = Pipeline([
    ('model', MLPRegressor(random_state=42, max_iter=2000))
])

param_grid = {
    'model__hidden_layer_sizes': [(x,y) for x in np.arange(1, 51) for y in np.arange(1,
}


grid_nn_noscale = GridSearchCV(pipe_nn_noscale, param_grid=param_grid, cv=KFold(5, shuff
                               scoring='neg_mean_squared_error').fit(superbowl_x, superb
result_nn_noscale = pd.DataFrame(grid_nn_noscale.cv_results_)[['mean_test_score', 'param
result_nn_noscale = result_nn_noscale.sort_values(by=['mean_test_score'], ascending=Fals
result_nn_noscale.head()
```

```
Fitting 5 folds for each of 2500 candidates, totalling 12500 fits
```

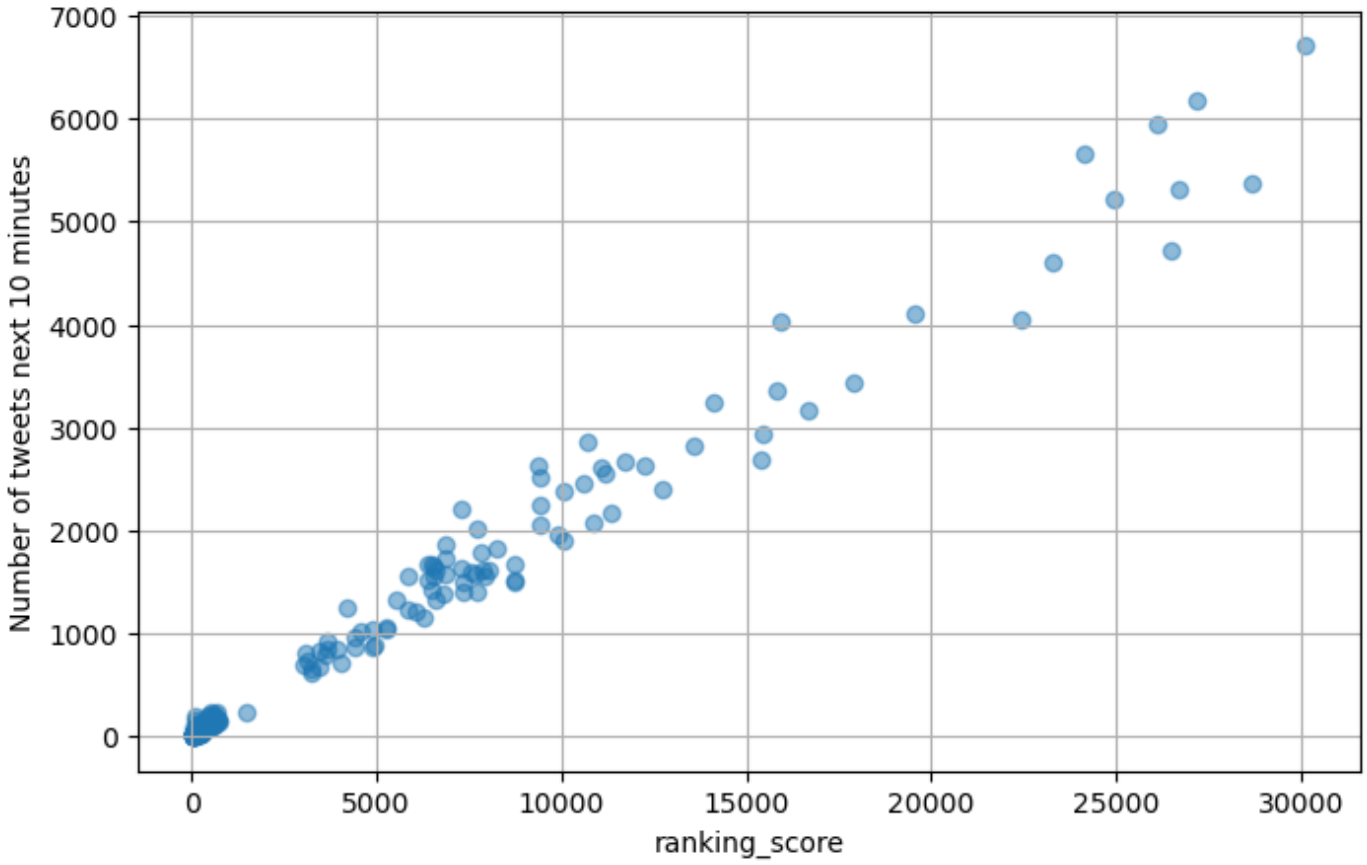
Out[92]:

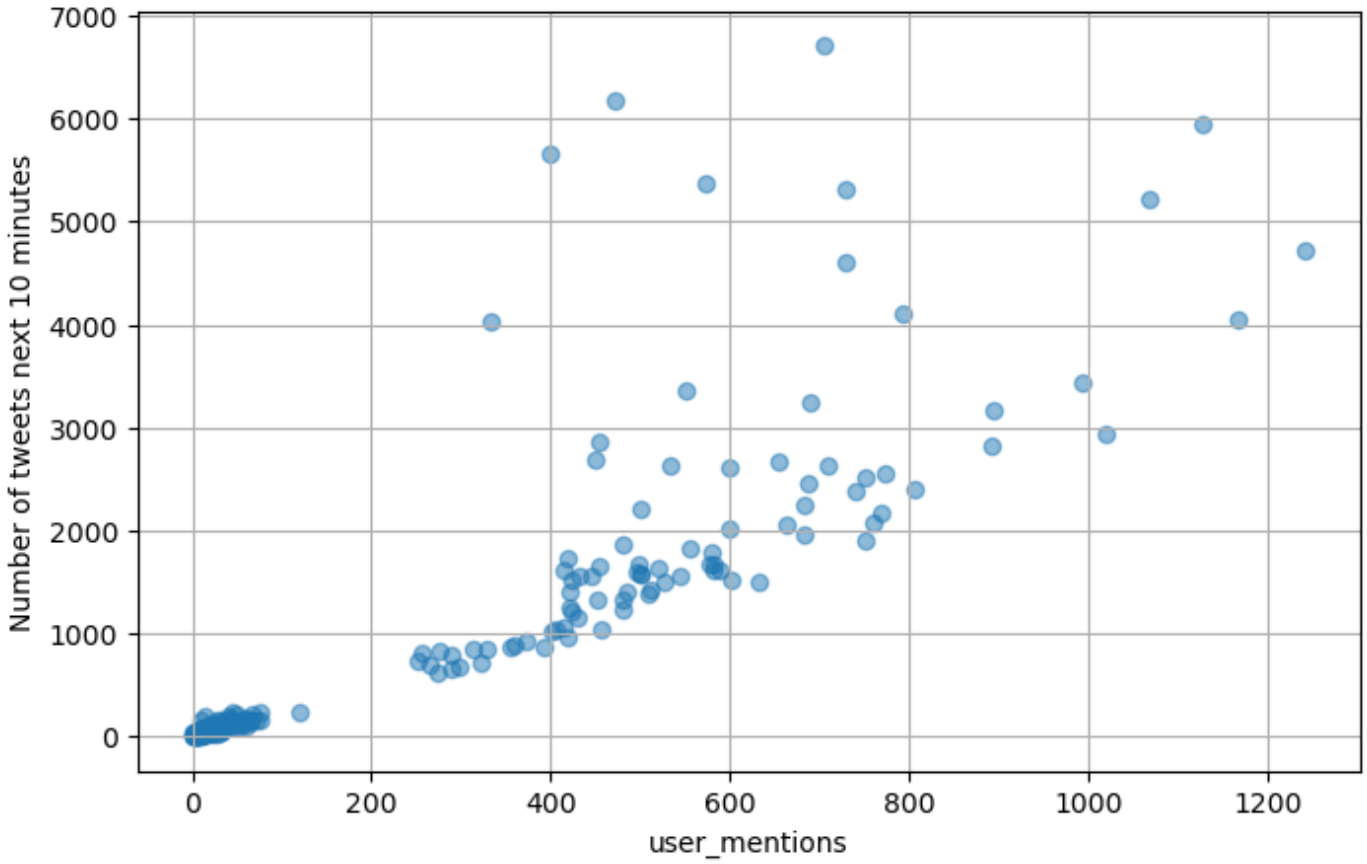
	mean_test_score	param_model__hidden_layer_sizes
0	-238146.783734	(5, 8)
1	-245119.714812	(25, 1)
2	-249601.147172	(9, 3)
3	-290023.313343	(5, 6)
4	-304567.791884	(8, 8)

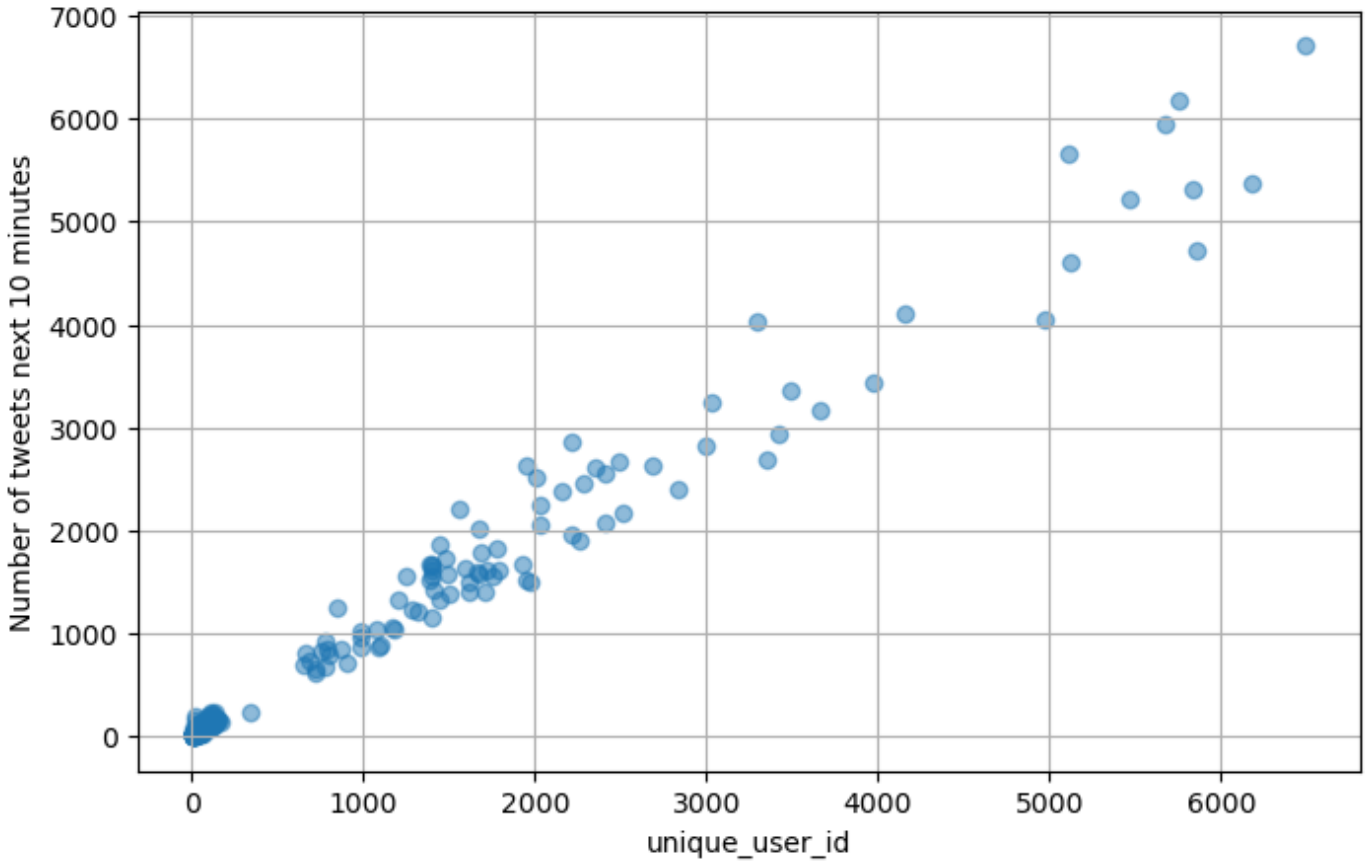
In [67]:
```
import statsmodels.api as sm
from sklearn import metrics
import numpy as np
import matplotlib.pyplot as plt

feature_names = list(superbowl_x.columns)
print(feature_names)

def scatter_plot(features, y_pred, pvalues, feature_names):
    # Obtain the indices that would sort the p-values in ascending order
    ranked_index = np.argsort(pvalues)
    print(ranked_index)
    for i in range(9):
        plt.figure(figsize = (8,5))
        # Create a scatter plot of the ith feature against the predicted values
        plt.scatter(features.iloc[:,ranked_index[i]], y_pred, alpha=0.5)
        plt.xlabel(feature_names[ranked_index[i]])
        plt.ylabel("Number of tweets next 10 minutes")
        plt.grid(True)
        plt.show()
```

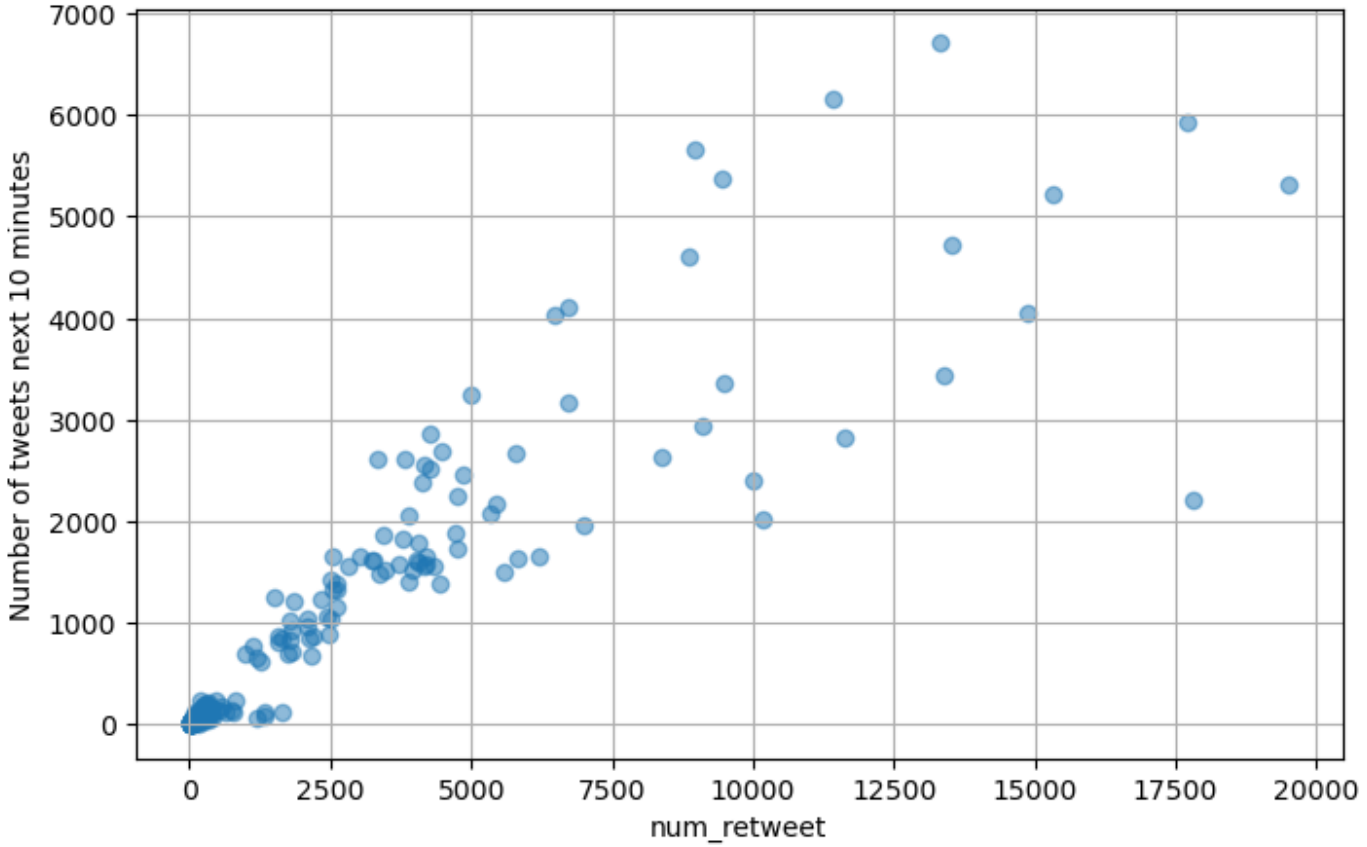
```
    print('-' * 80)

# Fit a linear regression model to the data and obtain the predicted values and p-values
lr_fit = sm.OLS(superbowl_y,superbowl_x).fit()
y_pred = lr_fit.predict()
pvalues = lr_fit.pvalues
print('MSE: ', metrics.mean_squared_error(superbowl_y, y_pred))
print(lr_fit.summary())
scatter_plot(superbowl_x, y_pred, pvalues, feature_names)
print('\n')
```

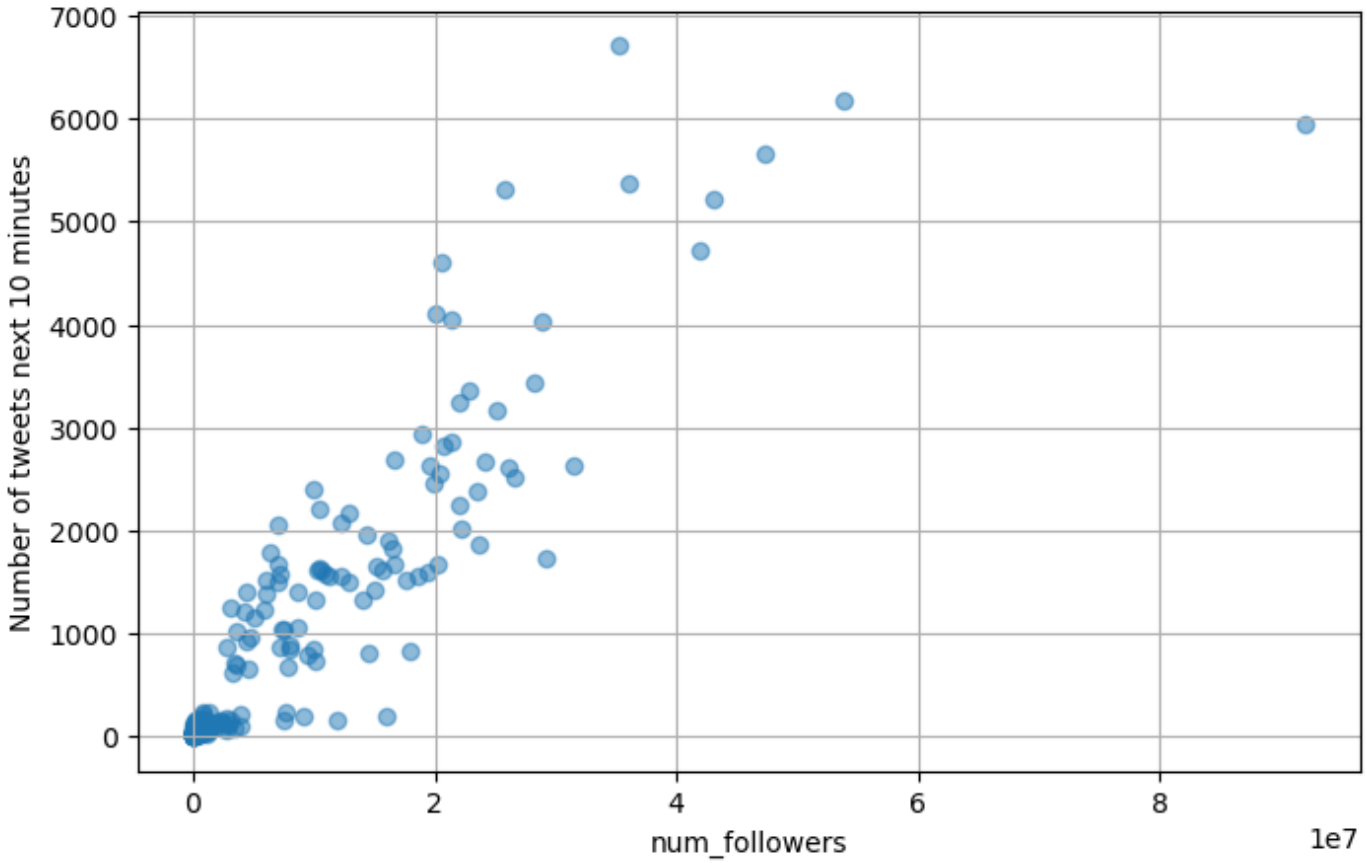
```
['num_retweet', 'num_followers', 'ranking_score', 'user_activity', 'user_mentions', 'num
_positive', 'num_neutral', 'num_negative', 'unique_user_id']
MSE:  156224.0969515157
                                 OLS Regression Results
================================================================================
Dep. Variable:                      y   R-squared (uncentered):                   0.940
Model:                            OLS   Adj. R-squared (uncentered):              0.938
Method:                 Least Squares   F-statistic:                              402.4
Date:                Tue, 14 Mar 2023   Prob (F-statistic):                   8.57e-136
Time:                        15:00:01   Log-Likelihood:                         -1775.6
No. Observations:                 240   AIC:                                      3569.
Df Residuals:                     231   BIC:                                      3601.
Df Model:                           9
Covariance Type:            nonrobust
================================================================================
                   coef    std err          t      P>|t|      [0.025      0.975]
--------------------------------------------------------------------------------
num_retweet        0.0377      0.020      1.842      0.067      -0.003       0.078
num_followers   9.285e-06   6.16e-06      1.507      0.133   -2.85e-06    2.14e-05
ranking_score      1.4483      0.428      3.386      0.001       0.605       2.291
user_activity      0.0018      0.005      0.346      0.730      -0.008       0.012
user_mentions      1.0776      0.402      2.684      0.008       0.286       1.869
num_positive      -1.5432      3.339     -0.462      0.644      -8.121       5.035
num_neutral       -0.9289      3.383     -0.275      0.784      -7.595       5.738
num_negative       0.3506      3.318      0.106      0.916      -6.188       6.889
unique_user_id    -5.2054      2.542     -2.048      0.042     -10.214      -0.197
==============================================================================
Omnibus:                      170.365   Durbin-Watson:                   2.233
Prob(Omnibus):                  0.000   Jarque-Bera (JB):             4411.142
Skew:                           2.349   Prob(JB):                         0.00
Kurtosis:                      23.471   Cond. No.                     3.10e+06
==============================================================================

Notes:
[1] R² is computed without centering (uncentered) since the model does not contain a con
stant.
[2] Standard Errors assume that the covariance matrix of the errors is correctly specifi
ed.
[3] The condition number is large, 3.1e+06. This might indicate that there are
strong multicollinearity or other numerical problems.
num_retweet       2
num_followers     4
ranking_score     8
user_activity     0
user_mentions     1
num_positive      5
num_neutral       3
num_negative      6
unique_user_id    7
dtype: int64
```

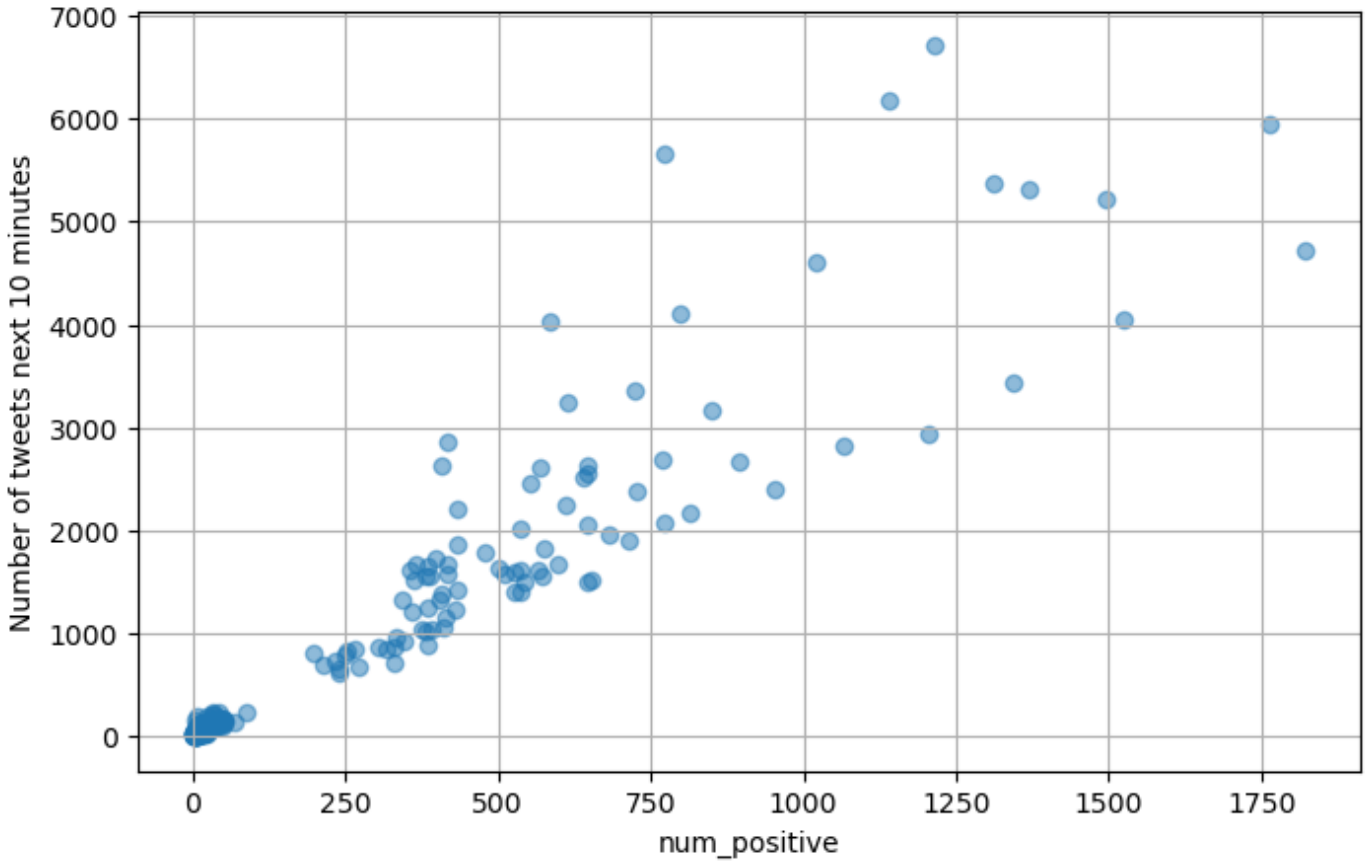

Number of tweets next 10 minutes
7000
6000
5000
4000
3000
2000
1000
0
0
5000
10000
15000
20000
25000
30000
ranking_score

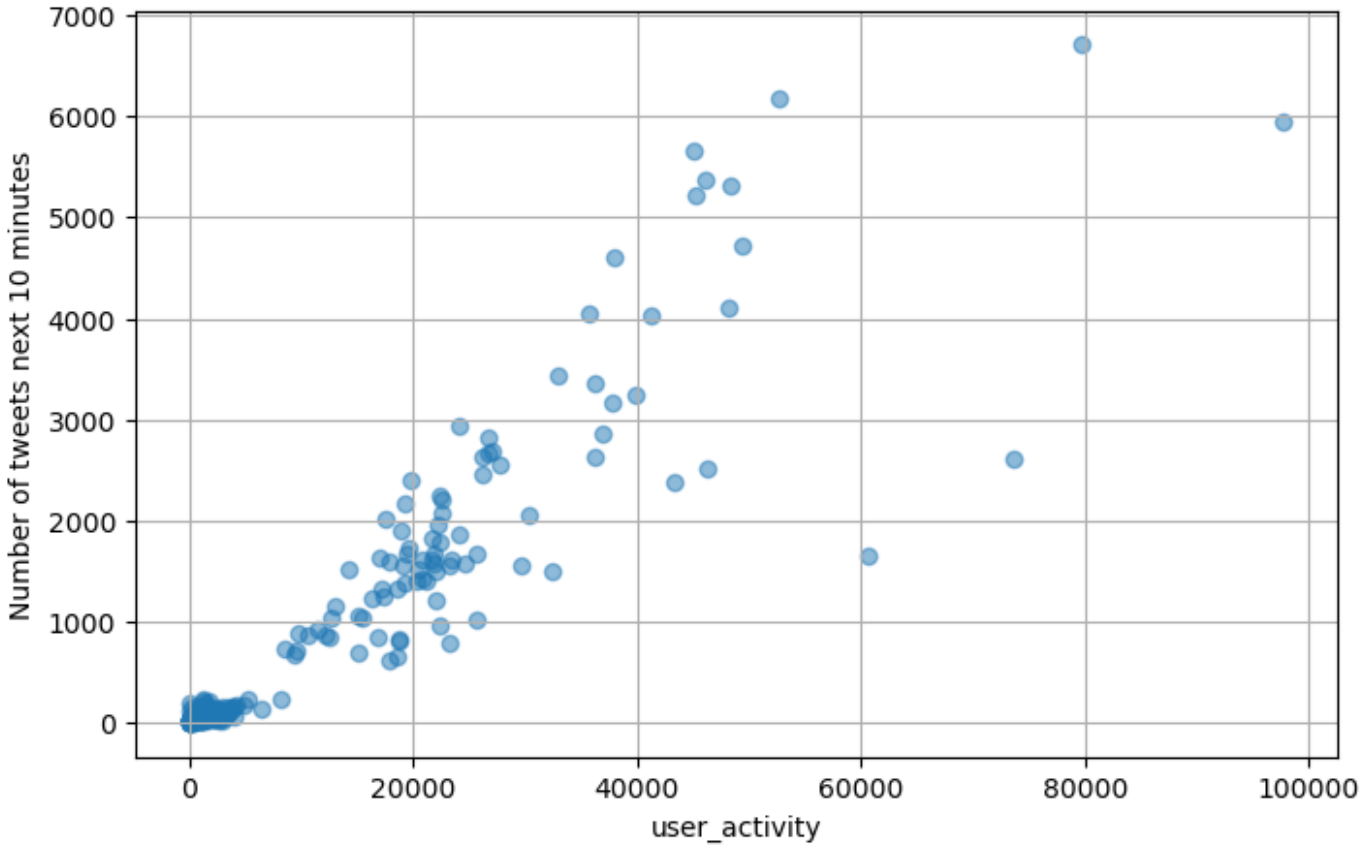

Number of tweets next 10 minutes
7000
6000
5000
4000
3000
2000
1000
0
0
200
400
600
800
1000
1200
user_mentions

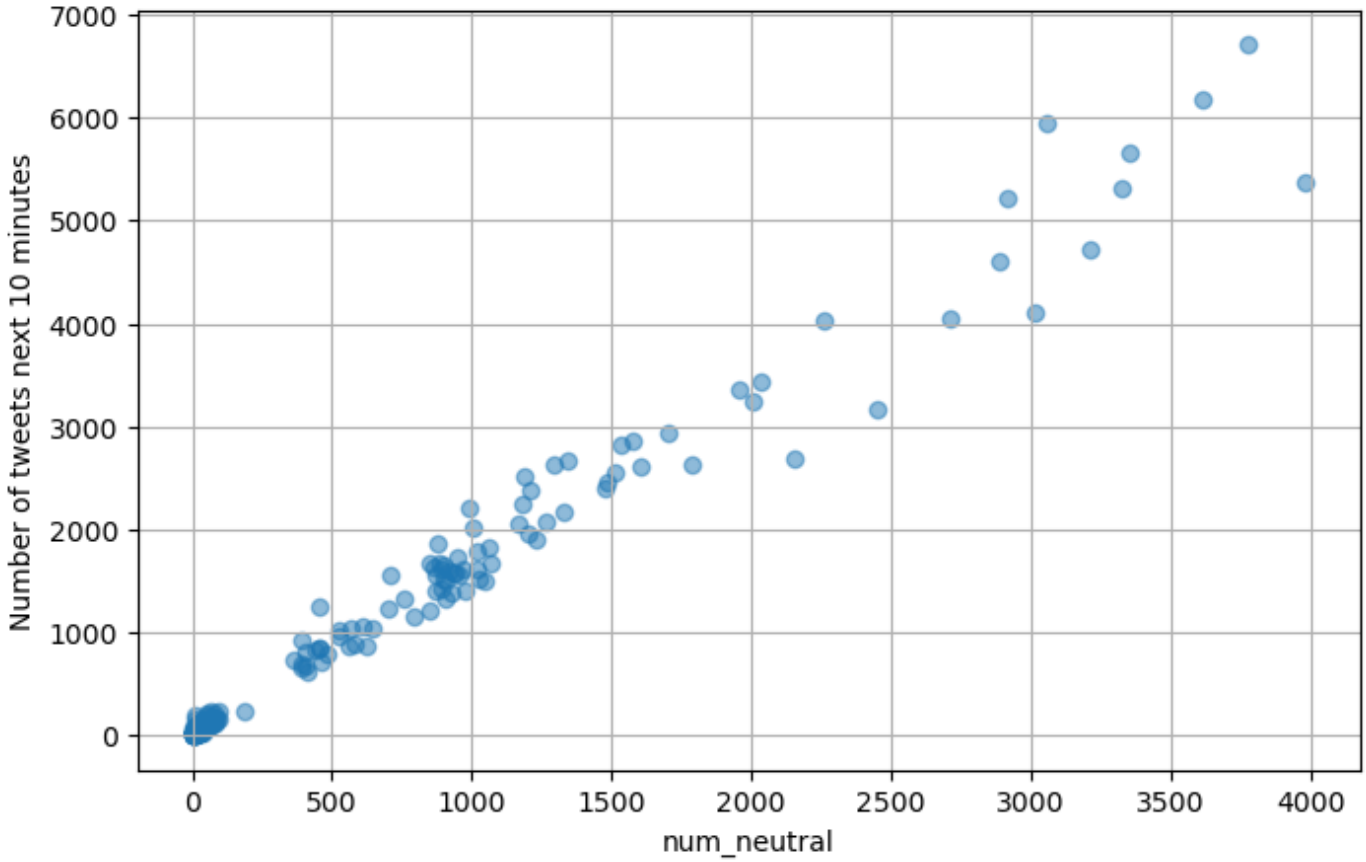

Number of tweets next 10 minutes
unique_user_id
0
1000
2000
3000
4000
5000
6000
7000

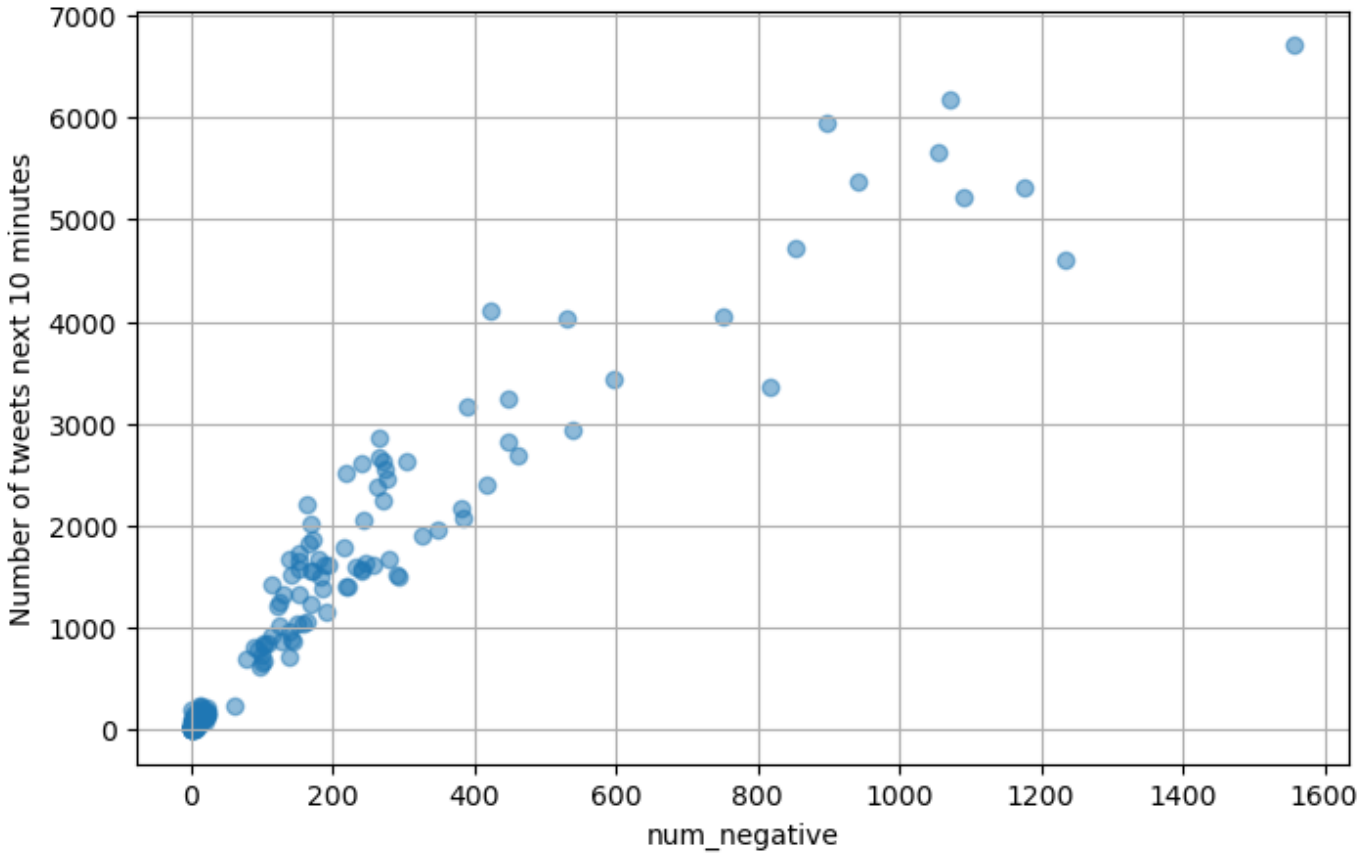

Number of tweets next 10 minutes
num_retweet
0
2500
5000
7500
10000
12500
15000
17500
20000
1000
2000
3000
4000
5000
6000
7000


Number of tweets next 10 minutes
7000
6000
5000
4000
3000
2000
1000
0
0
2
4
6
8
num_followers
1e7


Number of tweets next 10 minutes
7000
6000
5000
4000
3000
2000
1000
0
0
250
500
750
1000
1250
1500
1750
num_positive


Number of tweets next 10 minutes
7000
6000
5000
4000
3000
2000
1000
0
0
20000
40000
60000
80000
100000
user_activity


Number of tweets next 10 minutes
7000
6000
5000
4000
3000
2000
1000
0
0
500
1000
1500
2000
2500
3000
3500
4000
num_neutral



In [68]:
```
from sklearn.preprocessing import StandardScaler
from sklearn import preprocessing

superbowl_merge_data_standard_x = pd.DataFrame(preprocessing.scale(superbowl_merge_data_
# superbowl_merge_data_standard_x = superbowl_merge_data_drop
```

In [69]:
```
superbowl_merge_data_standard_x = superbowl_merge_data_standard_x.drop(superbowl_merge_d
```

In [71]:
```
print(superbowl_merge_data_standard_x.shape)
```

```
(240, 9)
```

In [78]:
```
superbowl_merge_data_standard_y =  pd.DataFrame(preprocessing.scale(superbowl_y),columns
print(superbowl_merge_data_standard_y.shape)
```

```
(240,)
```

In [84]:
```
from sklearn.feature_selection import SelectKBest, mutual_info_regression, f_regression
import numpy as np

# Define a function that selects the top n most important features using mutual informat
def select_topn_important_features(X, Y, n):
  Mutual_ = mutual_info_regression(X, Y)
  F_ = f_regression(X, Y)

  # Select the top n features based on their mutual information and F-test scores
  topn_M = np.argsort(Mutual_)[::-1][:n]
  topn_F = np.argsort(F_[0])[::-1][:n]

  # Sort all the features based on their mutual information and F-test scores
  all_m = np.argsort(Mutual_)[::-1]
  all_f = np.argsort(F_[0])[::-1]
```

```
  # Extract the top n features and all features based on their mutual information scores
  X_topn_M = X.iloc[:, topn_M]
  X_topn_F = X.iloc[:, topn_F]

  # Extract the top n features and all features based on their F-test scores
  all_m_ = X.iloc[:, all_m]
  all_f_ = X.iloc[:, all_f]

  return X_topn_M, X_topn_F
```

In [80]:
```
superbowl_top3_M, superbowl_top3_F, sall_m, sall_f = select_topn_important_features(supe
```

In [81]:
```
print("superbowl Top3 by mutual_info_regression:")
print(superbowl_top3_M.columns)

print("superbowl Top3 by f_regression:")
print(superbowl_top3_F.columns)

print("superbowl all by mutual_info_regression:")
print(sall_m.columns)

print("superbowl all by f_regression:")
print(sall_f.columns)
```

```
superbowl Top3 by mutual_info_regression:
Index(['unique_user_id', 'ranking_score', 'num_neutral'], dtype='object')
superbowl Top3 by f_regression:
Index(['ranking_score', 'num_neutral', 'unique_user_id'], dtype='object')
superbowl all by mutual_info_regression:
Index(['unique_user_id', 'ranking_score', 'num_neutral', 'num_positive',
       'user_mentions', 'num_negative', 'num_retweet', 'user_activity',
       'num_followers'],
      dtype='object')
superbowl all by f_regression:
Index(['ranking_score', 'num_neutral', 'unique_user_id', 'num_negative',
       'num_positive', 'user_activity', 'num_retweet', 'num_followers',
       'user_mentions'],
      dtype='object')
```

In [85]:
```
# now we are do experiments of exactly how many features we need to select

#with selection
superbowl_rmse_lr_m = []  #superbowl rmse score for linear regression and with mutual_in
superbowl_rmse_lr_f = []
superbowl_rmse_r_m = []
superbowl_rmse_r_f = []
superbowl_rmse_la_m = []
superbowl_rmse_la_f = []

#without selection
superbowl_rmse_lr = [] #superbowl rmse score for linear regression
superbowl_rmse_r = []
superbowl_rmse_la = []
```

In [86]:
```
from sklearn.model_selection import cross_validate,GridSearchCV
from sklearn.linear_model import LinearRegression, Ridge, Lasso

for k in range(1, 10):
    superbowl_topk_M, superbowl_topk_F = select_topn_important_features(superbowl_merge_
    #superbowl rmse score for linear regression
    score_ = cross_validate(LinearRegression(), superbowl_merge_data_standard_x, superbo
    score__ = score_['test_neg_root_mean_squared_error'].mean()
    print(f"superbowl rmse score for linear regression, {score__:.4f}")
    superbowl_rmse_lr.append(score__)
```

```
    #superbowl rmse score for linear regression and with mutual_info_regression
    score_ = cross_validate(LinearRegression(), superbowl_topk_M, superbowl_merge_data_s
    score__ = score_['test_neg_root_mean_squared_error'].mean()
    print(f"superbowl rmse score for linear regression and with mutual_info_regression,
    superbowl_rmse_lr_m.append(score__)

    #superbowl rmse score for linear regression and with f_regression
    score_ = cross_validate(LinearRegression(), superbowl_topk_F, superbowl_merge_data_s
    score__ = score_['test_neg_root_mean_squared_error'].mean()
    print(f"superbowl rmse score for linear regression and with f_regression, {score__:.
    superbowl_rmse_lr_f.append(score__)

    #superbowl rmse score for Ridge regression
    score_ = cross_validate(Ridge(), superbowl_merge_data_standard_x, superbowl_merge_da
    score__ = score_['test_neg_root_mean_squared_error'].mean()
    print(f"superbowl rmse score for Ridge regression, {score__:.4f}")
    superbowl_rmse_r.append(score__)

    #superbowl rmse score for Ridge regression and with mutual_info_regression
    score_ = cross_validate(Ridge(), superbowl_topk_M, superbowl_merge_data_standard_y,
    score__ = score_['test_neg_root_mean_squared_error'].mean()
    print(f"superbowl rmse score for Ridge regression and with mutual_info_regression, {
    superbowl_rmse_r_m.append(score__)

    #superbowl rmse score for Ridge regression and with f_regression
    score_ = cross_validate(Ridge(), superbowl_topk_F, superbowl_merge_data_standard_y,
    score__ = score_['test_neg_root_mean_squared_error'].mean()
    print(f"superbowl rmse score for Ridge regression and with f_regression, {score__:.4
    superbowl_rmse_r_f.append(score__)

    #superbowl rmse score for Lasso regression
    score_ = cross_validate(Lasso(), superbowl_merge_data_standard_x, superbowl_merge_da
    score__ = score_['test_neg_root_mean_squared_error'].mean()
    print(f"superbowl rmse score for Lasso regression, {score__:.4f}")
    superbowl_rmse_la.append(score__)

    #superbowl rmse score for Lasso regression and with mutual_info_regression
    score_ = cross_validate(Lasso(), superbowl_topk_M, superbowl_merge_data_standard_y,
    score__ = score_['test_neg_root_mean_squared_error'].mean()
    print(f"superbowl rmse score for Lasso regression and with mutual_info_regression, {
    superbowl_rmse_la_m.append(score__)

    #superbowl rmse score for Lasso regression and with f_regression
    score_ = cross_validate(Lasso(), superbowl_topk_F, superbowl_merge_data_standard_y,
    score__ = score_['test_neg_root_mean_squared_error'].mean()
    print(f"superbowl rmse score for Lasso regression and with f_regression, {score__:.4
    superbowl_rmse_la_f.append(score__)
```

```
superbowl rmse score for linear regression, -0.2195
superbowl rmse score for linear regression and with mutual_info_regression, -0.1650 top1
superbowl rmse score for linear regression and with f_regression, -0.1612 top1
superbowl rmse score for Ridge regression, -0.1797
superbowl rmse score for Ridge regression and with mutual_info_regression, -0.1663 top1
superbowl rmse score for Ridge regression and with f_regression, -0.1625 top1
superbowl rmse score for Lasso regression, -0.8621
superbowl rmse score for Lasso regression and with mutual_info_regression, -0.8631 top1
superbowl rmse score for Lasso regression and with f_regression, -0.8621 top1
superbowl rmse score for linear regression, -0.2195
superbowl rmse score for linear regression and with mutual_info_regression, -0.1770 top2
superbowl rmse score for linear regression and with f_regression, -0.1608 top2
superbowl rmse score for Ridge regression, -0.1797
superbowl rmse score for Ridge regression and with mutual_info_regression, -0.1632 top2
superbowl rmse score for Ridge regression and with f_regression, -0.1619 top2
superbowl rmse score for Lasso regression, -0.8621
superbowl rmse score for Lasso regression and with mutual_info_regression, -0.8621 top2
superbowl rmse score for Lasso regression and with f_regression, -0.8621 top2
```

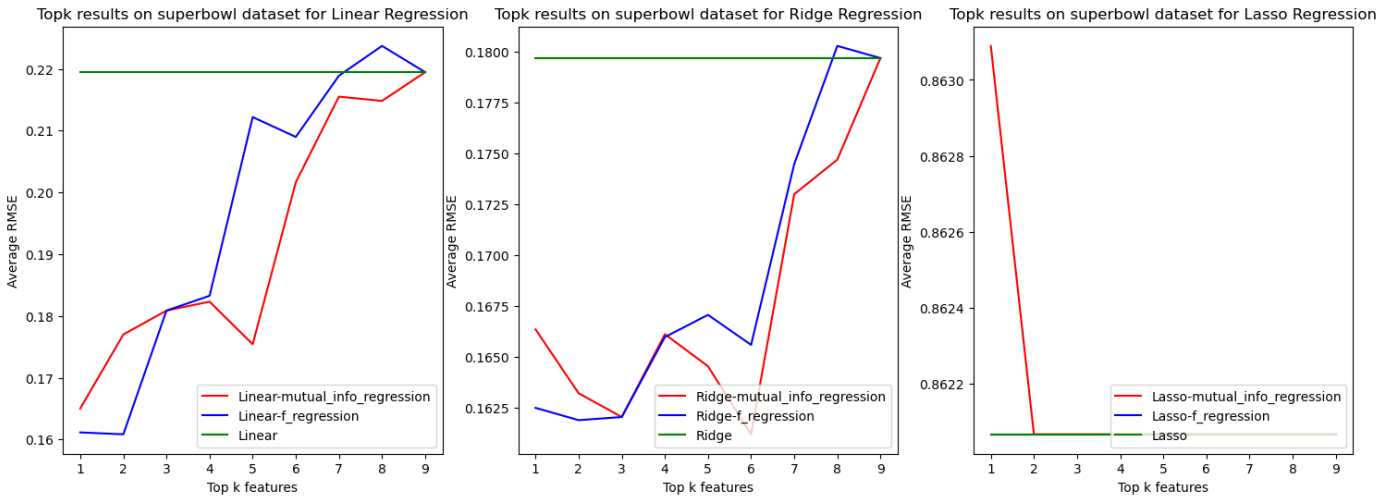
```
superbowl rmse score for linear regression, -0.2195
superbowl rmse score for linear regression and with mutual_info_regression, -0.1809 top3
superbowl rmse score for linear regression and with f_regression, -0.1809 top3
superbowl rmse score for Ridge regression, -0.1797
superbowl rmse score for Ridge regression and with mutual_info_regression, -0.1620 top3
superbowl rmse score for Ridge regression and with f_regression, -0.1620 top3
superbowl rmse score for Lasso regression, -0.8621
superbowl rmse score for Lasso regression and with mutual_info_regression, -0.8621 top3
superbowl rmse score for Lasso regression and with f_regression, -0.8621 top3
superbowl rmse score for linear regression, -0.2195
superbowl rmse score for linear regression and with mutual_info_regression, -0.1823 top4
superbowl rmse score for linear regression and with f_regression, -0.1833 top4
superbowl rmse score for Ridge regression, -0.1797
superbowl rmse score for Ridge regression and with mutual_info_regression, -0.1661 top4
superbowl rmse score for Ridge regression and with f_regression, -0.1660 top4
superbowl rmse score for Lasso regression, -0.8621
superbowl rmse score for Lasso regression and with mutual_info_regression, -0.8621 top4
superbowl rmse score for Lasso regression and with f_regression, -0.8621 top4
superbowl rmse score for linear regression, -0.2195
superbowl rmse score for linear regression and with mutual_info_regression, -0.1754 top5
superbowl rmse score for linear regression and with f_regression, -0.2122 top5
superbowl rmse score for Ridge regression, -0.1797
superbowl rmse score for Ridge regression and with mutual_info_regression, -0.1645 top5
superbowl rmse score for Ridge regression and with f_regression, -0.1671 top5
superbowl rmse score for Lasso regression, -0.8621
superbowl rmse score for Lasso regression and with mutual_info_regression, -0.8621 top5
superbowl rmse score for Lasso regression and with f_regression, -0.8621 top5
superbowl rmse score for linear regression, -0.2195
superbowl rmse score for linear regression and with mutual_info_regression, -0.2016 top6
superbowl rmse score for linear regression and with f_regression, -0.2090 top6
superbowl rmse score for Ridge regression, -0.1797
superbowl rmse score for Ridge regression and with mutual_info_regression, -0.1612 top6
superbowl rmse score for Ridge regression and with f_regression, -0.1656 top6
superbowl rmse score for Lasso regression, -0.8621
superbowl rmse score for Lasso regression and with mutual_info_regression, -0.8621 top6
superbowl rmse score for Lasso regression and with f_regression, -0.8621 top6
superbowl rmse score for linear regression, -0.2195
superbowl rmse score for linear regression and with mutual_info_regression, -0.2155 top7
superbowl rmse score for linear regression and with f_regression, -0.2189 top7
superbowl rmse score for Ridge regression, -0.1797
superbowl rmse score for Ridge regression and with mutual_info_regression, -0.1730 top7
superbowl rmse score for Ridge regression and with f_regression, -0.1745 top7
superbowl rmse score for Lasso regression, -0.8621
superbowl rmse score for Lasso regression and with mutual_info_regression, -0.8621 top7
superbowl rmse score for Lasso regression and with f_regression, -0.8621 top7
superbowl rmse score for linear regression, -0.2195
superbowl rmse score for linear regression and with mutual_info_regression, -0.2148 top8
superbowl rmse score for linear regression and with f_regression, -0.2237 top8
superbowl rmse score for Ridge regression, -0.1797
superbowl rmse score for Ridge regression and with mutual_info_regression, -0.1747 top8
superbowl rmse score for Ridge regression and with f_regression, -0.1803 top8
superbowl rmse score for Lasso regression, -0.8621
superbowl rmse score for Lasso regression and with mutual_info_regression, -0.8621 top8
superbowl rmse score for Lasso regression and with f_regression, -0.8621 top8
superbowl rmse score for linear regression, -0.2195
superbowl rmse score for linear regression and with mutual_info_regression, -0.2195 top9
superbowl rmse score for linear regression and with f_regression, -0.2195 top9
superbowl rmse score for Ridge regression, -0.1797
superbowl rmse score for Ridge regression and with mutual_info_regression, -0.1797 top9
superbowl rmse score for Ridge regression and with f_regression, -0.1797 top9
superbowl rmse score for Lasso regression, -0.8621
superbowl rmse score for Lasso regression and with mutual_info_regression, -0.8621 top9
superbowl rmse score for Lasso regression and with f_regression, -0.8621 top9
```

In [87]:
```
import matplotlib.pyplot as plt
```

```
fig, axes = plt.subplots(1, 3, figsize=(18, 6))

#plot superbowl linear
axes[0].plot(np.arange(1, len(superbowl_rmse_lr_m) + 1, 1), np.negative(superbowl_rmse_l
axes[0].plot(np.arange(1, len(superbowl_rmse_lr_f) + 1, 1), np.negative(superbowl_rmse_l
axes[0].plot(np.arange(1, len(superbowl_rmse_lr) + 1, 1), np.negative(superbowl_rmse_lr)

axes[0].legend(loc='lower right')
axes[0].set(xlabel='Top k features',ylabel='Average RMSE')
axes[0].set_title('Topk results on superbowl dataset for Linear Regression')

#plot superbowl Ridge
axes[1].plot(np.arange(1, len(superbowl_rmse_r_m) + 1, 1), np.negative(superbowl_rmse_r_
axes[1].plot(np.arange(1, len(superbowl_rmse_r_f) + 1, 1), np.negative(superbowl_rmse_r_
axes[1].plot(np.arange(1, len(superbowl_rmse_r) + 1, 1), np.negative(superbowl_rmse_r),

axes[1].legend(loc='lower right')
axes[1].set(xlabel='Top k features',ylabel='Average RMSE')
axes[1].set_title('Topk results on superbowl dataset for Ridge Regression')

#plot superbowl Lasso
axes[2].plot(np.arange(1, len(superbowl_rmse_la_m) + 1, 1), np.negative(superbowl_rmse_l
axes[2].plot(np.arange(1, len(superbowl_rmse_la_f) + 1, 1), np.negative(superbowl_rmse_l
axes[2].plot(np.arange(1, len(superbowl_rmse_la) + 1, 1), np.negative(superbowl_rmse_la)

axes[2].legend(loc='lower right')
axes[2].set(xlabel='Top k features',ylabel='Average RMSE')
axes[2].set_title('Topk results on superbowl dataset for Lasso Regression')
```

Out[87]:

```
Text(0.5, 1.0, 'Topk results on superbowl dataset for Lasso Regression')
```



In [88]:

```
# Finding the optimal penalty parameter
from joblib import Memory
from sklearn.pipeline import Pipeline

location = "cachedir"
memory = Memory(location=location, verbose=10)

pipe_ = Pipeline([
    ('kbest', SelectKBest()),
    ('model', "passthrough")
], memory = memory)

param_grid = [{
        'kbest__score_func': (mutual_info_regression, f_regression),
        'kbest__k': (1, 2, 3, 4, 5, 6, 7, 8, 9),
        'model': [Ridge(), Lasso()],
        'model__alpha': [10.0**x for x in np.arange(-3,4)]
```

```
    }
]
```

In [89]:
```
grid_superbowl = GridSearchCV(pipe_, param_grid = param_grid, cv = 10, n_jobs = -1, verb
                      scoring = 'neg_root_mean_squared_error', return_train_score = True)
```

```
Fitting 10 folds for each of 252 candidates, totalling 2520 fits
[Memory] Calling sklearn.pipeline._fit_transform_one...
_fit_transform_one(SelectKBest(k=6,
            score_func=<function mutual_info_regression at 0x000001D011E13820>),
     num_retweet  num_followers  ranking_score  user_activity  user_mentions  \
0       0.572113      -0.037429       0.628675       0.750375       1.241756
1       0.276907       1.212140       0.417785       0.956063       0.957631
2       0.646982       1.060345       0.313306       0.540213       0.732410
3       0.392484       1.517641       0.477341       0.856452       0.902192
4       0.358324       1.148113       0.882697       0.989590       1.082369
..           ...            ...            ...            ...            ...
235    -0.568855      -0.593462      -0.587048      -0.641907      -0.698609
236    -0.558322      -0.590781      -0.576466      -0.58614...,
array([ 0.424299, ..., -0.614222]), None, message_clsname='Pipeline', message=None)
_____________________________________________fit_transform_one - 0.0s, 0.0min
```

Task2 Predict the tweets' location based on Classification model

Find the tweets' majority areas

In [73]:
```
import itertools
#list of US states and their abbreviations
STATES = ['Alabama', 'AL', 'Alaska', 'AK', 'American Samoa', 'AS', 'Arizona', 'AZ', 'Ark
# create a dictionary where each key is a state abbreviation and the corresponding value
STATE_DICT = dict(itertools.zip_longest(*[iter(STATES)] * 2, fillvalue=""))
INV_STATE_DICT = dict((v,k) for k,v in STATE_DICT.items())
```

In [80]:
```
is_in_US=[]
geo = superbowl_data['user_location']
df = superbowl_data.fillna(" ")
# Iterate over each user location in the dataset
for x in superbowl_data['user_location']:
    check = False
    for s in STATES:
        if s in x:
            is_in_US.append(STATE_DICT[s] if s in STATE_DICT else s)
            check = True
            break
    if not check:
        is_in_US.append(None)

geo_dist = pd.DataFrame(is_in_US, columns=['State'])           \
    .dropna().reset_index()
```

In [81]:
```
import math
#Count the number of occurrences in each group
geo_dist = geo_dist.groupby('State').count().rename(columns={"index": "Number"}).sort_va
geo_dist["Log Num"] = geo_dist["Number"].apply(lambda x: math.log(x, 2))
```

In [82]:
```
geo_dist['Full State Name'] = geo_dist['State'].apply(lambda x: INV_STATE_DICT[x])
geo_dist['text'] = geo_dist['Full State Name'] + '<br>' + 'Num: ' + geo_dist['Number'].a
```

In [83]:
```
geo_dist
```

Out[83]:

	State	Number	Log Num	Full State Name	text

0	NY	7483	12.869401	New York	New York Num: 7483
1	CA	6862	12.744413	California	California Num: 6862
2	TX	4560	12.154818	Texas	Texas Num: 4560
3	FL	3278	11.678600	Florida	Florida Num: 3278
4	MA	2600	11.344296	Massachusetts	Massachusetts Num: 2600
5	IL	2019	10.979425	Illinois	Illinois Num: 2019
6	OH	1778	10.796040	Ohio	Ohio Num: 1778
7	PA	1653	10.690871	Pennsylvania	Pennsylvania Num: 1653
8	GA	1627	10.667999	Georgia	Georgia Num: 1627
9	NJ	1612	10.654636	New Jersey	New Jersey Num: 1612
10	NC	1584	10.629357	North Carolina	North Carolina Num: 1584
11	MI	1460	10.511753	Michigan	Michigan Num: 1460
12	WA	1436	10.487840	Washington	Washington Num: 1436
13	VA	1356	10.405141	Virginia	Virginia Num: 1356
14	DC	1335	10.382624	District of Columbia	District of Columbia Num: 1335
15	AZ	1285	10.327553	Arizona	Arizona Num: 1285
16	IN	1271	10.311748	Indiana	Indiana Num: 1271
17	LA	1251	10.288866	Louisiana	Louisiana Num: 1251
18	CO	1234	10.269127	Colorado	Colorado Num: 1234
19	TN	1161	10.181152	Tennessee	Tennessee Num: 1161
20	MN	1067	10.059344	Minnesota	Minnesota Num: 1067
21	AL	972	9.924813	Alabama	Alabama Num: 972
22	KS	883	9.786270	Kansas	Kansas Num: 883
23	OR	873	9.769838	Oregon	Oregon Num: 873
24	WI	863	9.753217	Wisconsin	Wisconsin Num: 863
25	MD	852	9.734710	Maryland	Maryland Num: 852
26	MO	807	9.656425	Missouri	Missouri Num: 807
27	CT	755	9.560333	Connecticut	Connecticut Num: 755
28	KY	728	9.507795	Kentucky	Kentucky Num: 728
29	SC	710	9.471675	South Carolina	South Carolina Num: 710
30	IA	709	9.469642	Iowa	Iowa Num: 709
31	OK	663	9.372865	Oklahoma	Oklahoma Num: 663
32	AR	623	9.283088	Arkansas	Arkansas Num: 623
33	NE	472	8.882643	Nebraska	Nebraska Num: 472
34	UT	450	8.813781	Utah	Utah Num: 450
35	HI	442	8.787903	Hawaii	Hawaii Num: 442
36	NV	441	8.784635	Nevada	Nevada Num: 441
37	AS	363	8.503826	American Samoa	American Samoa Num: 363
38	ME	340	8.409391	Maine	Maine Num: 340

39	RI	326	8.348728	Rhode Island	Rhode Island Num: 326
40	MS	321	8.326429	Mississippi	Mississippi Num: 321
41	NH	314	8.294621	New Hampshire	New Hampshire Num: 314
42	PR	295	8.204571	Puerto Rico	Puerto Rico Num: 295
43	DE	287	8.164907	Delaware	Delaware Num: 287
44	NM	213	7.734710	New Mexico	New Mexico Num: 213
45	ID	188	7.554589	Idaho	Idaho Num: 188
46	WV	176	7.459432	West Virginia	West Virginia Num: 176
47	AK	168	7.392317	Alaska	Alaska Num: 168
48	SD	166	7.375039	South Dakota	South Dakota Num: 166
49	ND	153	7.257388	North Dakota	North Dakota Num: 153
50	MT	120	6.906891	Montana	Montana Num: 120
51	VT	91	6.507795	Vermont	Vermont Num: 91
52	WY	68	6.087463	Wyoming	Wyoming Num: 68
53	GU	46	5.523562	Guam	Guam Num: 46
54	VI	40	5.321928	Virgin Islands	Virgin Islands Num: 40
55	FM	34	5.087463	Federated States of Micronesia	Federated States of Micronesia Num: 34
56	MP	15	3.906891	Northern Mariana Islands	Northern Mariana Islands Num: 15
57	MH	12	3.584963	Marshall Islands	Marshall Islands Num: 12
58	PW	7	2.807355	Palau	Palau Num: 7

In [84]:
```
import plotly.graph_objects as go
fig = go.Figure(data=go.Choropleth(
    locations=geo_dist['State'], # Spatial coordinates
    z = geo_dist['Log Num'].astype(float), # Data to be color-coded

    locationmode = 'USA-states',
    colorscale = "Reds",
    text=geo_dist['text'],
    marker_line_color='white', # line markers between states
    colorbar_title = "Numbers in Log2"
))

fig.update_layout(
    geo_scope='usa',
)

fig.show()
```

Maps and filter the areas into numbers

In [3]:
```
# Preprocessing
def preprocess_text(text):
    tweet_text_ = re.sub(r"http\S+", "", text)
    words = nltk.word_tokenize(tweet_text_.lower())
    words = [word for word in words if word.isalnum() and word not in stopwords.words('e
    return ' '.join(re.sub("(@[A-Za-z0-9]+)|([^0-9A-Za-z \t])|(\w+:\/\/\S+)", " ", words

import itertools
from surprise import Reader, Dataset, accuracy
```

```
#list of US states and their abbreviations
STATES = ['Alabama', 'AL', 'Alaska', 'AK', 'American Samoa', 'AS', 'Arizona', 'AZ', 'Ark
STATE_DICT = dict(itertools.zip_longest(*[iter(STATES)] * 2, fillvalue=""))
STATE_DICT_REVERSE = {value: key for key, value in STATE_DICT.items()}
print(STATE_DICT_REVERSE)

def in_location(row):
  s = row["user_location"]
  if s in STATES:
   #   return STATE_DICT_REVERSE[s] if s in STATE_DICT_REVERSE else s
     return True
  else:
     return False

def map_location(row):
   s = row["user_location"]
   row["user_location"] = STATE_DICT_REVERSE[s] if s in STATE_DICT_REVERSE else s

filter_superbowl = superbowl_data[superbowl_data.apply(in_location, axis=1)]
map_filter_superbowl = filter_superbowl[filter_superbowl.apply(in_location, axis=1)]
```

```
[nltk_data] Downloading package stopwords to C:\Users\wenxin
[nltk_data]     cheng\AppData\Roaming\nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
{'AL': 'Alabama', 'AK': 'Alaska', 'AS': 'American Samoa', 'AZ': 'Arizona', 'AR': 'Arkans
as', 'CA': 'California', 'CO': 'Colorado', 'CT': 'Connecticut', 'DE': 'Delaware', 'DC':
'District of Columbia', 'FM': 'Federated States of Micronesia', 'FL': 'Florida', 'GA':
'Georgia', 'GU': 'Guam', 'HI': 'Hawaii', 'ID': 'Idaho', 'IL': 'Illinois', 'IN': 'Indian
a', 'IA': 'Iowa', 'KS': 'Kansas', 'KY': 'Kentucky', 'LA': 'Louisiana', 'ME': 'Maine', 'M
H': 'Marshall Islands', 'MD': 'Maryland', 'MA': 'Massachusetts', 'MI': 'Michigan', 'MN':
'Minnesota', 'MS': 'Mississippi', 'MO': 'Missouri', 'MT': 'Montana', 'NE': 'Nebraska',
'NV': 'Nevada', 'NH': 'New Hampshire', 'NJ': 'New Jersey', 'NM': 'New Mexico', 'NY': 'Ne
w York', 'NC': 'North Carolina', 'ND': 'North Dakota', 'MP': 'Northern Mariana Islands',
'OH': 'Ohio', 'OK': 'Oklahoma', 'OR': 'Oregon', 'PW': 'Palau', 'PA': 'Pennsylvania', 'P
R': 'Puerto Rico', 'RI': 'Rhode Island', 'SC': 'South Carolina', 'SD': 'South Dakota',
'TN': 'Tennessee', 'TX': 'Texas', 'UT': 'Utah', 'VT': 'Vermont', 'VI': 'Virgin Islands',
'VA': 'Virginia', 'WA': 'Washington', 'WV': 'West Virginia', 'WI': 'Wisconsin', 'WY': 'W
yoming'}
```

In [4]:

```
locations = list(STATE_DICT.keys())
print(locations)
location_to_number = {location: index + 1 for index, location in enumerate(locations)}
print(location_to_number)
```

```
['Alabama', 'Alaska', 'American Samoa', 'Arizona', 'Arkansas', 'California', 'Colorado',
'Connecticut', 'Delaware', 'District of Columbia', 'Federated States of Micronesia', 'Fl
orida', 'Georgia', 'Guam', 'Hawaii', 'Idaho', 'Illinois', 'Indiana', 'Iowa', 'Kansas',
'Kentucky', 'Louisiana', 'Maine', 'Marshall Islands', 'Maryland', 'Massachusetts', 'Mich
igan', 'Minnesota', 'Mississippi', 'Missouri', 'Montana', 'Nebraska', 'Nevada', 'New Ham
pshire', 'New Jersey', 'New Mexico', 'New York', 'North Carolina', 'North Dakota', 'Nort
hern Mariana Islands', 'Ohio', 'Oklahoma', 'Oregon', 'Palau', 'Pennsylvania', 'Puerto Ri
co', 'Rhode Island', 'South Carolina', 'South Dakota', 'Tennessee', 'Texas', 'Utah', 'Ve
rmont', 'Virgin Islands', 'Virginia', 'Washington', 'West Virginia', 'Wisconsin', 'Wyomi
ng']
{'Alabama': 1, 'Alaska': 2, 'American Samoa': 3, 'Arizona': 4, 'Arkansas': 5, 'Californi
a': 6, 'Colorado': 7, 'Connecticut': 8, 'Delaware': 9, 'District of Columbia': 10, 'Fede
rated States of Micronesia': 11, 'Florida': 12, 'Georgia': 13, 'Guam': 14, 'Hawaii': 15,
'Idaho': 16, 'Illinois': 17, 'Indiana': 18, 'Iowa': 19, 'Kansas': 20, 'Kentucky': 21, 'L
ouisiana': 22, 'Maine': 23, 'Marshall Islands': 24, 'Maryland': 25, 'Massachusetts': 26,
'Michigan': 27, 'Minnesota': 28, 'Mississippi': 29, 'Missouri': 30, 'Montana': 31, 'Nebr
aska': 32, 'Nevada': 33, 'New Hampshire': 34, 'New Jersey': 35, 'New Mexico': 36, 'New Y
ork': 37, 'North Carolina': 38, 'North Dakota': 39, 'Northern Mariana Islands': 40, 'Ohi
o': 41, 'Oklahoma': 42, 'Oregon': 43, 'Palau': 44, 'Pennsylvania': 45, 'Puerto Rico': 4
6, 'Rhode Island': 47, 'South Carolina': 48, 'South Dakota': 49, 'Tennessee': 50, 'Texa
```

```
s': 51, 'Utah': 52, 'Vermont': 53, 'Virgin Islands': 54, 'Virginia': 55, 'Washington': 5
6, 'West Virginia': 57, 'Wisconsin': 58, 'Wyoming': 59}
```

In [5]:
```
map_filter_superbowl["location_map"] = map_filter_superbowl['user_location']
map_filter_superbowl = map_filter_superbowl.replace({"location_map": location_to_number}
```

In [6]:
```
map_filter_superbowl.shape
```

Out[6]:
```
(8988, 15)
```

In [7]:
```
from sklearn.metrics import confusion_matrix, accuracy_score, recall_score, precision_sc
from sklearn import svm
from sklearn.svm import SVC, LinearSVC
```

In [8]:
```
def print_result(y_true,y_pred,name="",average='macro'):
    accuracy, recall, precision, f1 = accuracy_score(y_true,y_pred) * 100, recall_score(
                                      precision_score(y_true,y_pred,average=average)*10
    print("%s: accuracy= %.2f, recall= %.2f, precision= %.2f, f1= %.2f" %(name,accuracy,
    return [accuracy, recall, precision, f1]
```

Feature Extraction and MultinomialNB Classifier

In [10]:
```
# Feature extraction
vectorizer = CountVectorizer()
X = vectorizer.fit_transform(map_filter_superbowl['text'])
y = map_filter_superbowl["user_location"]

# Splitting data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42

# Training the model
classifier = MultinomialNB()
classifier.fit(X_train, y_train)

# Testing the model
predictions = classifier.predict(X_test)
print_result(y_test, predictions, "MultinomialNB")
```

```
MultinomialNB: accuracy= 49.44, recall= 23.46, precision= 50.57, f1= 26.48
```

```
C:\Users\wenxin cheng\AppData\Roaming\Python\Python38\site-packages\sklearn\metrics\_cla
ssification.py:1318: UndefinedMetricWarning: Precision is ill-defined and being set to
0.0 in labels with no predicted samples. Use `zero_division` parameter to control this b
ehavior.
  _warn_prf(average, modifier, msg_start, len(result))
```

Out[10]:
```
[49.44382647385984, 23.461077602318454, 50.57123045078566, 26.480921080385738]
```

HardMarginSVM and SoftMarginSVM

In [13]:
```
# Computing Hard and Soft Margin SVMs
HardMargin_SVM = LinearSVC(C=1000, random_state=42, max_iter = 3000)
HardMargin_SVM2 = LinearSVC(C=100000, random_state=42, max_iter = 3000)
SoftMargin_SVM = LinearSVC(C=0.0001, random_state=42, max_iter = 3000)

X_hardSVM_pred = HardMargin_SVM.fit(X_train,y_train).predict(X_test) # predicting labels
X_hardSVM_pred2 = HardMargin_SVM2.fit(X_train,y_train).predict(X_test) # predicting labe
X_softSVM_pred = SoftMargin_SVM.fit(X_train,y_train).predict(X_test) # predicting labels

print_result(y_test,X_hardSVM_pred, "hard margin svm C=1000" )
print_result(y_test,X_hardSVM_pred2, "hard margin svm C=100000" )
print_result(y_test,X_softSVM_pred, "soft margin svm C=0.0001" )
```

```
C:\Users\wenxin cheng\AppData\Roaming\Python\Python38\site-packages\sklearn\svm\_base.p
```

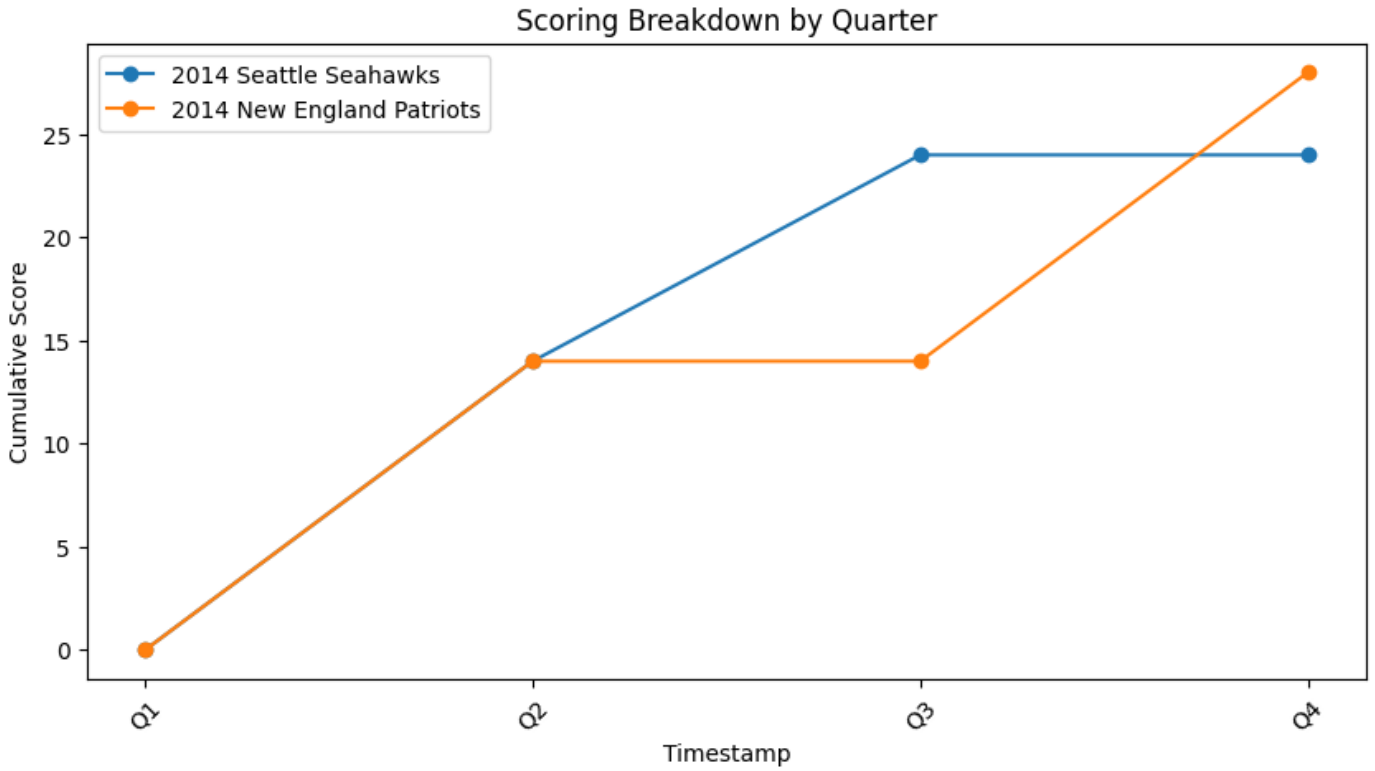
```
y:1206: ConvergenceWarning: Liblinear failed to converge, increase the number of iterati
ons.
  warnings.warn(
C:\Users\wenxin cheng\AppData\Roaming\Python\Python38\site-packages\sklearn\svm\_base.p
y:1206: ConvergenceWarning: Liblinear failed to converge, increase the number of iterati
ons.
  warnings.warn(
hard margin svm C=1000: accuracy= 81.42, recall= 67.75, precision= 76.21, f1= 70.03
hard margin svm C=100000: accuracy= 81.42, recall= 67.75, precision= 76.21, f1= 70.03
soft margin svm C=0.0001: accuracy= 35.82, recall= 15.79, precision= 15.27, f1= 14.82
C:\Users\wenxin cheng\AppData\Roaming\Python\Python38\site-packages\sklearn\svm\_base.p
y:1206: ConvergenceWarning: Liblinear failed to converge, increase the number of iterati
ons.
  warnings.warn(
C:\Users\wenxin cheng\AppData\Roaming\Python\Python38\site-packages\sklearn\metrics\_cla
ssification.py:1318: UndefinedMetricWarning: Recall is ill-defined and being set to 0.0
in labels with no true samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
C:\Users\wenxin cheng\AppData\Roaming\Python\Python38\site-packages\sklearn\metrics\_cla
ssification.py:1318: UndefinedMetricWarning: Precision is ill-defined and being set to
0.0 in labels with no predicted samples. Use `zero_division` parameter to control this b
ehavior.
  _warn_prf(average, modifier, msg_start, len(result))
C:\Users\wenxin cheng\AppData\Roaming\Python\Python38\site-packages\sklearn\metrics\_cla
ssification.py:1318: UndefinedMetricWarning: Recall is ill-defined and being set to 0.0
in labels with no true samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
C:\Users\wenxin cheng\AppData\Roaming\Python\Python38\site-packages\sklearn\metrics\_cla
ssification.py:1318: UndefinedMetricWarning: Precision is ill-defined and being set to
0.0 in labels with no predicted samples. Use `zero_division` parameter to control this b
ehavior.
  _warn_prf(average, modifier, msg_start, len(result))
C:\Users\wenxin cheng\AppData\Roaming\Python\Python38\site-packages\sklearn\metrics\_cla
ssification.py:1318: UndefinedMetricWarning: Precision is ill-defined and being set to
0.0 in labels with no predicted samples. Use `zero_division` parameter to control this b
ehavior.
  _warn_prf(average, modifier, msg_start, len(result))
```

Out[13]:
```
[35.81757508342603, 15.790002295684113, 15.267395232742262, 14.818812702800665]
```

From the results, find the relationship between timestamp and scores

In [16]:
```python
import pandas as pd
from datetime import datetime, timedelta
import matplotlib.pyplot as plt

# Game start time
start_time = datetime(2015, 2, 1, 18, 30)


# Scoring data
data = [
    {"team": "2014 Seattle Seahawks", "quarter": 1, "score": 0, "cumulative_score": 0, "
    {"team": "2014 New England Patriots", "quarter": 1, "score": 0, "cumulative_score":
    {"team": "2014 Seattle Seahawks", "quarter": 2, "score": 14, "cumulative_score": 14,
    {"team": "2014 New England Patriots", "quarter": 2, "score": 14, "cumulative_score":
    {"team": "2014 Seattle Seahawks", "quarter": 3, "score": 10, "cumulative_score": 24,
    {"team": "2014 New England Patriots", "quarter": 3, "score": 0, "cumulative_score":
    {"team": "2014 Seattle Seahawks", "quarter": 4, "score": 0, "cumulative_score": 24,
    {"team": "2014 New England Patriots", "quarter": 4, "score": 14, "cumulative_score":
]

# Create a DataFrame
df = pd.DataFrame(data)
```

```
# Plot the scoring breakdown
plt.figure(figsize=(10, 5))
plt.plot(df[df["team"] == "2014 Seattle Seahawks"]["timestamp"], df[df["team"] == "2014
plt.plot(df[df["team"] == "2014 New England Patriots"]["timestamp"], df[df["team"] == "2

plt.xlabel("Timestamp")
plt.ylabel("Cumulative Score")
plt.title("Scoring Breakdown by Quarter")
plt.legend()
timestamps = [start_time, start_time + timedelta(minutes=15), start_time + timedelta(min
# Format x-axis labels
plt.xticks(timestamps , [f"Q{int(i)}" for i in range(1, 5)], rotation=45)

plt.show()
```



Use generative Model to generate tweets according to the score

In [ ]:
```
import openai
import sys

openai.api_key = "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"
```

In [36]:
```
game_info = {
    "team1": "2014 Seattle Seahawks",
    "team2": "2014 New England Patriots",
    "team1_score": 0,
    "team2_score": 0,
    "Quarter": 1
}
prompts = [
    f"Write a tweet summarizing the game between the {game_info['team1']} and the {game_
    f"where {game_info['team1']} scored {game_info['team1_score']} and {game_info['team2
]
```

In [38]:
```
for prompt in prompts:
    converted_prompt = f"Write a story using '{prompt}' as the prompt."
    completions = openai.Completion.create(
        engine="text-davinci-002",
        prompt=converted_prompt,
```

```
        max_tokens=1024,
        n=1,
        stop=None,
        temperature=0.5
    )
    message = completions.choices[0].text
    print(message)
```

```
The 2014 Seattle Seahawks and the 2014 New England Patriots faced off in a game where bo
th teams scored 0. The Seahawks defense held strong, but the Patriots offense was unable
to break through. In the end, it was a defensive battle between two of the best teams in
the NFL.
```

Find max mentioned player

In [ ]:
```
#Hawks Players: `h_players`
h_players = [
  "Russell Wilson", "Tarvaris Jackson", "B.J. Daniels", "Marshawn Lynch",
  "Robert Turbin", "Christine Michael","Will Tukuafu","Luke Wilson",
  "Tony Moeaki", "Cooper Helfet","Doug Baldwin", "Jermaine Kearse",
  "Ricardo Lockette", "Chris Matthews", "Kevin Norwood", "Bryan Walters",
  "Alvin Bailey", "Justin Britt", "Russell Okung", "Lemuel Jeanpierre",
  "Keavon Milton", "J.R. Sweezy", "James Carpenter", "Max Unger", "Patrick Lewis",
  "Cliff Avril", "Michael Bennett", "Demarcus Dobbs", "David King",
  "O'Brien Schofield","Kevin Williams", "Tony McDaniel", "Landon Cohen",
  "Bruce Irvin", "K.J. Wright", "Bobby Wagner", "Malcolm Smith", "Mike Morgan",
  "Brock Coyle","Richard Sherman", "Byron Maxwell", "Jeremy Lane",
  "DeShawn Shead", "Tharold Simon", "Marcus Burley","Earl Thomas",
  "Kam Chancellor", "Steven Terrell", "Jeron Johnson","Steven Hauschka",
  "Jon Ryan", "Clint Gresham"
]
#Patriots Players: `p_players`
p_players = [
  "Tom Brady", "Jimmy Garoppolo","Shane Vereen", "LeGarrette Blount",
  "Brandon Bolden", "Jonas Gray", "James White","James Develin",
  "Rob Gronkowski", "Michael Hoomanawanui", "Tim Wright","Julian Edelman",
  "Brandon LaFell", "Danny Amendola", "Josh Boyce", "Matthew Slater",
  "Brian Tyms", "Nate Solder", "Sebastian Vollmer", "Jordan Devey",
  "Cameron Fleming", "Dan Connolly", "Marcus Cannon", "Josh Fline",
  "Bryan Stork", "Ryan Wendell","Chandler Jones", "Rob Ninkovich",
  "Alan Branch", "Zach Moore", "Joe Vellano", "Vince Wilfork","Chris Jones",
  "Sealver Siliga","Jonathan Casillas", "Jamie Collins", "Darius Fleming",
  "Dont'a Hightower", "Chris White", "Akeem Ayers","Darrelle Revis",
  "Malcolm Butler", "Brandon Browner", "Kyle Arrington", "Logan Ryan",
  "Patrick Chung", "Devin McCourty", "Nate Ebner", "Duron Harmon",
  "Tavon Wilson","Stephen Gostkowski","Ryan Allen","Danny Aiken"
]
h_players = set([player.lower() for player in h_players])
p_players = set([player.lower() for player in p_players])

#All Players: `players`
players = h_players.union(p_players)
```

In [4]:
```
import pandas as pd
import matplotlib.pyplot as plt
from wordcloud import WordCloud, STOPWORDS
```

In [5]:
```
# checks if a given tweet contains a given player name
def contains_player(tweet, player):
    return player in tweet.lower()

player_counts = {}
tweets_with_player = []
for player in players:
```

```
    ## Filter the dataframe to get only the tweets that mention the current player
    player_tweets = superbowl_data[superbowl_data["text"].apply(lambda tweet: contains_p
    ## Count the number of times the current player is mentioned in the filtered tweets
    player_count =  sum([tweet.count(player) for tweet in player_tweets])
    if player_count > 0:
        player_counts[player] = player_count

    # Add the filtered tweets to the list of tweets that mention any of the players
    tweets_with_player.extend(superbowl_data[superbowl_data["text"].apply(lambda tweet:
```

Occurance of each player in tweets

In [6]:
```
sorted_counts = sorted(player_counts.items(), key=lambda x: x[1], reverse=True)
for player, count in sorted_counts:
    print(f"{player}: {count}")
```

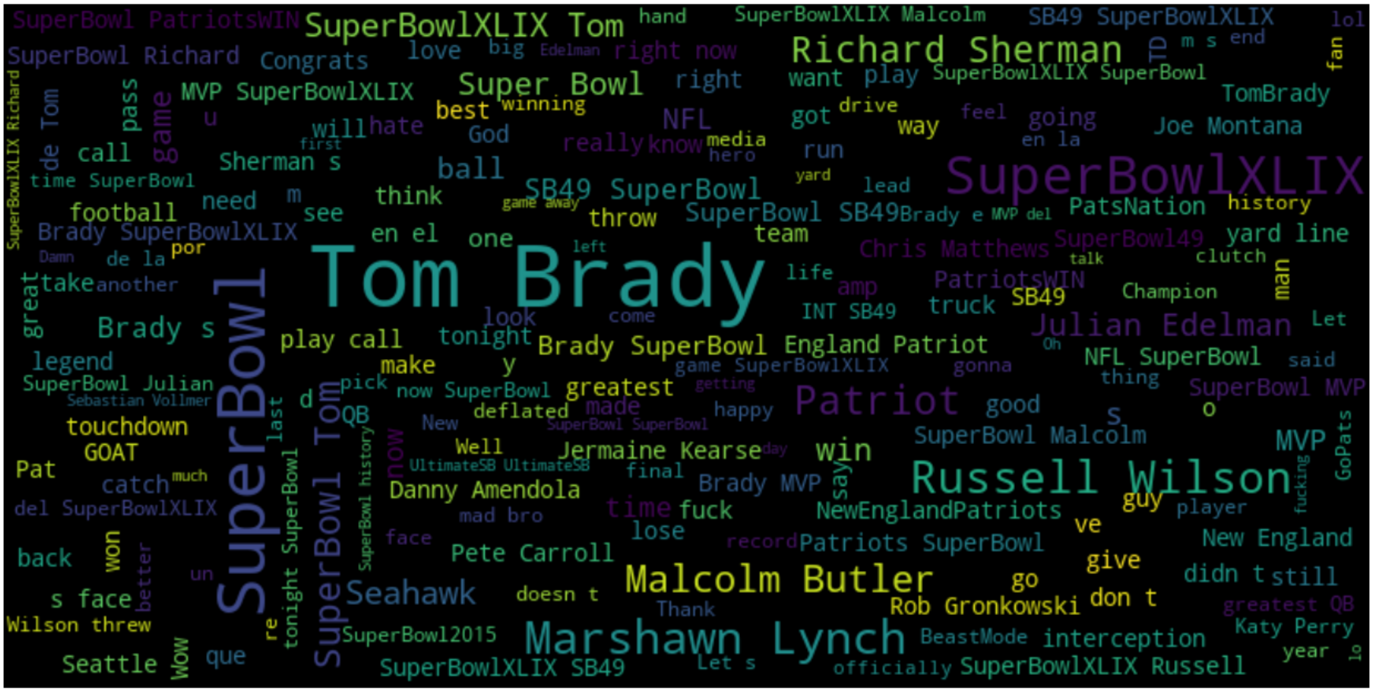
```
tom brady: 85
marshawn lynch: 59
richard sherman: 15
russell wilson: 13
malcolm butler: 5
chris matthews: 4
julian edelman: 4
jermaine kearse: 3
rob gronkowski: 2
rob ninkovich: 2
ricardo lockette: 2
vince wilfork: 1
legarrette blount: 1
danny amendola: 1
jeremy lane: 1
cliff avril: 1
tharold simon: 1
```

In [7]:
```
all_tweets = " ".join(tweets_with_player)
```

In [8]:
```
from PIL import Image
import numpy as np

stopwords = set(STOPWORDS)
stopwords.update(["https", "co", "RT"])
```

WordCloud of each player in tweets

In [10]:
```
wordcloud = WordCloud(stopwords=stopwords, background_color='black',max_font_size=50,wid

plt.figure(figsize=(16, 8))
plt.imshow(wordcloud, interpolation='bilinear')
plt.axis("off")
plt.show()
```


Tom Brady
SuperBowl
SuperBowlXLIX
Russell Wilson
Marshawn Lynch
Malcolm Butler
Richard Sherman
Patriot
Julian Edelman
SuperBowlXLIX Tom
Super Bowl
SuperBowl Tom
SB49 SuperBowl
Seahawk
SuperBowl PatriotsWIN
SuperBowlXLIX Malcolm
SB49 SuperBowlXLIX
SuperBowl Richard
Congrats
MVP SuperBowlXLIX
Brady SuperBowl
England Patriot
Jermaine Kearse
Danny Amendola
Pete Carroll
Rob Gronkowski
Joe Montana
TomBrady
PatsNation
Chris Matthews
SuperBowl49
PatriotsWIN
NFL SuperBowl
SuperBowl MVP
SuperBowl Malcolm
NewEnglandPatriots
Patriots SuperBowl
New England
interception
SuperBowlXLIX Russell
SuperBowlXLIX SB49
SuperBowl2015
BeastMode
Katy Perry
Seattle
Wilson threw
touchdown
GOAT
football
Brady SuperBowlXLIX
del SuperBowlXLIX
SuperBowl Julian
Sebastian Vollmer
Brady MVP
play call
deflated
greatest QB
yard line
Champion
clutch
history
win
MVP
love
hand
big
right now
play
want
best
winning
hate
will
God
really
know
media
hero
NFL
got
drive
way
feel
going
run
ball
think
throw
team
lead
need
see
one
en el
de la
por
take
another
look
come
tonight
make
legend
great
pass
game
call
time
life
amp
truck
SB49
INT SB49
good
happy
made
final
catch
Pat
back
won
s face
que
Wow
doesn t
Thank
don t
go
give
didn t
still
guy
player
year
lol
fan
man
said
thing
talk
fuck
lose
record
officially
Let s
now SuperBowl
UltimateSB UltimateSB
face
mad bro
Well
much
better
tonight SuperBowl
GoPats