Project_4_Part1_Regression_Analysis

Project 4 - Part 1 Regression Analysis¶

Gorkem Camli (105709280)

Library imports¶

Dataset Exploration¶ Diamonds Data

```
(53940, 10)
```

	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

	carat	depth	table	price	x	y	z
count	53940.000000	53940.000000	53940.000000	53940.000000	53940.000000	53940.000000	53940.000000
mean	0.797940	61.749405	57.457184	3934.801557	5.731157	5.734526	3.538734
std	0.474011	1.432621	2.234491	3989.442321	1.121761	1.142135	0.705699
min	0.200000	43.000000	43.000000	327.000000	0.000000	0.000000	0.000000
25%	0.400000	61.000000	56.000000	952.000000	4.710000	4.720000	2.910000
50%	0.700000	61.800000	57.000000	2403.000000	5.700000	5.710000	3.530000
75%	1.040000	62.500000	59.000000	5327.250000	6.540000	6.540000	4.040000
max	5.010000	79.000000	95.000000	18823.000000	10.740000	58.900000	31.800000

```
carat      float64
cut         object
color       object
clarity     object
depth      float64
table      float64
price        int64
x          float64
y          float64
z          float64
dtype: object
```

Gas Turbine CO and NOx Emission Data

```
(36733, 12)
```

	AT	AP	AH	AFDP	GTEP	TIT	TAT	TEY	CDP	CO	NOX	year
0	4.5878	1018.7	83.675	3.5758	23.979	1086.2	549.83	134.67	11.898	0.32663	81.952	2011
1	4.2932	1018.3	84.235	3.5709	23.951	1086.1	550.05	134.67	11.892	0.44784	82.377	2011
2	3.9045	1018.4	84.858	3.5828	23.990	1086.5	550.19	135.10	12.042	0.45144	83.776	2011
3	3.7436	1018.3	85.434	3.5808	23.911	1086.5	550.17	135.03	11.990	0.23107	82.505	2011
4	3.7516	1017.8	85.182	3.5781	23.917	1085.9	550.00	134.67	11.910	0.26747	82.028	2011

	count	mean	std	min	25%	50%	75%	max
AT	36733.0	17.712726	7.447451	-6.234800	11.7810	17.8010	23.6650	37.1030
AP	36733.0	1013.070165	6.463346	985.850000	1008.8000	1012.6000	1017.0000	1036.6000
AH	36733.0	77.867015	14.461355	24.085000	68.1880	80.4700	89.3760	100.2000
AFDP	36733.0	3.925518	0.773936	2.087400	3.3556	3.9377	4.3769	7.6106
GTEP	36733.0	25.563801	4.195957	17.698000	23.1290	25.1040	29.0610	40.7160
TIT	36733.0	1081.428084	17.536373	1000.800000	1071.8000	1085.9000	1097.0000	1100.9000
TAT	36733.0	546.158517	6.842360	511.040000	544.7200	549.8800	550.0400	550.6100
TEY	36733.0	133.506404	15.618634	100.020000	124.4500	133.7300	144.0800	179.5000
CDP	36733.0	12.060525	1.088795	9.851800	11.4350	11.9650	12.8550	15.1590
CO	36733.0	2.372468	2.262672	0.000388	1.1824	1.7135	2.8429	44.1030
NOX	36733.0	65.293067	11.678357	25.905000	57.1620	63.8490	71.5480	119.9100

	year
count	36733
unique	5
top	2012
freq	7628

	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

Before Training¶

Standardization¶

Question 1¶

Diamonds Data¶ For diamonds data, all the categorical features are quality features, therefore I used Ordinal Encoder to encode them. While encoding the given labels, I specified the ordering from low to high quality for the cut, color, clarity features. For example, for cut categories: Fair gets the lowest label number whereas Ideal got the highest. Later, for both numerical and encoded categorical features, standard scaler is applied. Target variable is kept same.

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

```
Raw Diamonds Data
```

	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

`Diamonds Data After Standardization and Category Feature Encoding`

	carat	cut	color	clarity	depth	table	price	x	y	z
1	-1.198168	0.981473	0.937163	-1.245215	-0.174092	-1.099672	330	-1.587837	-1.536196	-1.571129
2	-1.240361	0.085889	0.937163	-0.638095	-1.360738	1.585529	327	-1.641325	-1.658774	-1.741175
3	-1.198168	-1.705279	0.937163	0.576145	-3.385019	3.375663	328	-1.498691	-1.457395	-1.741175
4	-1.071587	0.085889	-1.414272	-0.030975	0.454133	0.242928	337	-1.364971	-1.317305	-1.287720

`Description of Diamonds data after standardization`

	count	mean	std	min	25%	50%	75%	max
carat	53940.0	2.444878e-16	1.000009	-1.261458	-0.839523	-0.206621	0.510668	8.886075
cut	53940.0	1.454281e-16	1.000009	-2.600864	-0.809695	0.085889	0.981473	0.981473
color	53940.0	1.338360e-16	1.000009	-2.002131	-0.826413	-0.238555	0.937163	1.525021
clarity	53940.0	-8.114467e-17	1.000009	-1.852335	-0.638095	-0.030975	0.576145	2.397505
depth	53940.0	-3.996902e-15	1.000009	-13.087603	-0.523105	0.035317	0.523936	12.041392
table	53940.0	9.695207e-17	1.000009	-6.470073	-0.652139	-0.204605	0.690462	16.801666
price	53940.0	3.934802e+03	3989.442321	327.000000	952.000000	2403.000000	5327.250000	18823.000000
x	53940.0	2.782103e-16	1.000009	-5.109120	-0.910325	-0.027776	0.721054	4.465203
y	53940.0	-8.430615e-17	1.000009	-5.020931	-0.888280	-0.021474	0.705242	46.549648
z	53940.0	-2.002271e-16	1.000009	-5.014556	-0.890946	-0.012376	0.710318	40.047576

Gas Data¶ For gas data, I dropped the CO column and will use NOX as the target variable. Only categorical feature is year. It is considered as an ordered feature, we could use ordinal encoder for it. It is also possible to use one-hot-encoding. I used Ordinal Encoder to encode them. While encoding, I specified the ordering from low to high. Later, for both numerical and encoded categorical features, standard scaler is applied. Target variable is kept same (NOX).

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

```
years:  ['2011' '2012' '2013' '2014' '2015']
```

`Raw Gas Data`

	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

`Gas Data After Standardization and Category Feature Encoding`

	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	81.952	-1.3994
1	-1.801920	0.809164	0.440351	-0.458207	-0.384376	0.266417	0.568742	0.074502	-0.154783	82.377	-1.3994

	AT	AP	AH	AFDP	GTEP	TIT	TAT	TEY	CDP	NOX	year
2	-1.854113	0.824636	0.483432	-0.442831	-0.375081	0.289227	0.589203	0.102033	-0.017015	83.776	-1.3994
3	-1.875718	0.809164	0.523263	-0.445415	-0.393909	0.289227	0.586280	0.097551	-0.064774	82.505	-1.3994

`Description of Gas data after standardization`

	count	mean	std	min	25%	50%	75%	max
AT	36733.0	-1.176081e-16	1.000014	-3.215577	-0.796488	0.011853	0.799247	2.603647
AP	36733.0	-1.233647e-14	1.000014	-4.211524	-0.660683	-0.072744	0.608027	3.640553
AH	36733.0	-5.942306e-16	1.000014	-3.719067	-0.669311	0.179998	0.795855	1.544343
AFDP	36733.0	-1.015144e-15	1.000014	-2.375059	-0.736399	0.015741	0.583238	4.761549
GTEP	36733.0	5.230467e-16	1.000014	-1.874640	-0.580281	-0.109583	0.833480	3.611191
TIT	36733.0	9.609823e-15	1.000014	-4.597825	-0.549043	0.255012	0.887990	1.110388
TAT	36733.0	-8.300659e-15	1.000014	-5.132585	-0.210240	0.543896	0.567280	0.650586
TEY	36733.0	9.888369e-16	1.000014	-2.144032	-0.579854	0.014316	0.676995	2.944830
CDP	36733.0	-4.673376e-16	1.000014	-2.028623	-0.574519	-0.087736	0.729692	2.845821
NOX	36733.0	6.529307e+01	11.678357	25.905000	57.162000	63.849000	71.548000	119.910000
year	36733.0	-7.427883e-17	1.000014	-1.399443	-0.694695	0.010053	0.714802	1.419550

I could have also standardized the target variables. The advantage of standardizing the target variables are especially when training models such as NN. Having very large range of target variables can make it difficult to adapt and assign weights to predict the results. This is because larger target results require larger weights.

I also tried one-hot encoding but the result but it makes more sense to use ordinal encoder due to quality nature of the variables and also for the efficiency. I tried also one-hot encoder but results seemed to be better and faster with this one, so I decided to go with ordinal encoder.

Profiling Results¶

Pandas profiling is used to explore the dataset for both diamonds and gas dataset. The results can be seen below:

Data Inspection¶

To further understand the data I did fata inspection and answered the following questions:

Question 2¶ Pearson Correlation Heatmap:

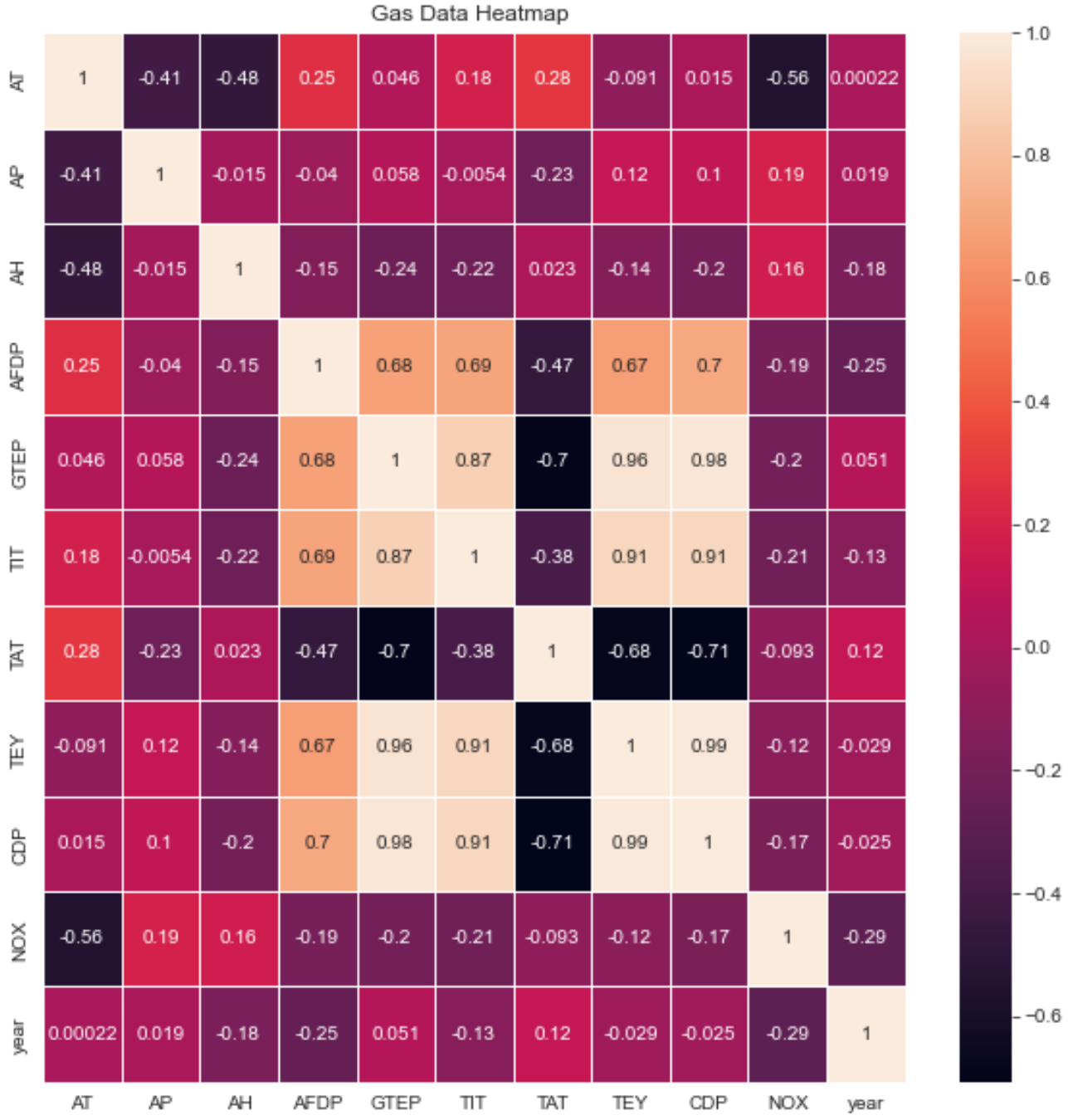
For Pearson correlation the values are between -1 to 1. Values closer -1 means strong negative correlation, values closer 1 means strong positive correlation and values closer to 0 means low correlation. High correlation means there is a strong linear relationship between two variables. If it is closer to 1, means increasing one variable we expect increase in the other. If the value is closer to -1, high negative correlation, increasing one variable will result in the decrease of the other variable. Having a value close to 0 means that there is no strong linear relation betweeen the two variables, changing one doesn't have a clear effect on the other as a direct increase or decrease. Having a pearson correlation 0 or close to 0 doesn't mean there is no relationship between the two, a nonlinear relationship may exist, it is just that we cannot tell nonlinear relationship by just looking Pearson correlation.

For both datasets heatmap with pearson correlation is plotted:

Diamonds Data:

In the diamonds data Pearson Correlation heatmap, we see high correlation on carat,x, y, z and price columns. The pearson correlation values between price and carat is 0.92, x=0.88, y=0.87 and z=0.86. Carat seems to be the most important feature to tell the diamond price. The high correlation suggest carat, x, y, z values are the most predictive features for predicting the price of a diamond. The pearson correlation for these features are positive for target variable price. This means there is a positive relationship between these variables and price, hence the increasing any of these 4 features will lead the diamond price to be higher. These makes sense in the real world as well since the bigger diamonds (dimensions x,y,z) and/or carat (weight) are generally more expensive.

Gas Data:

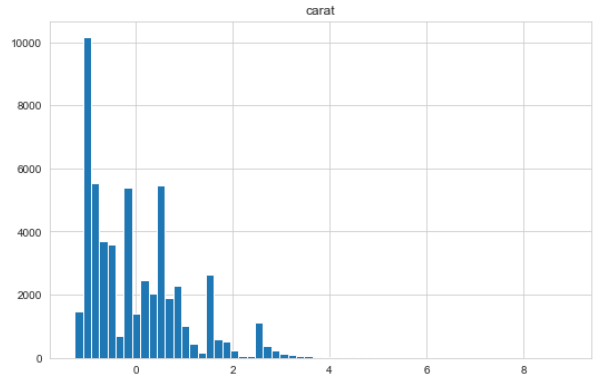


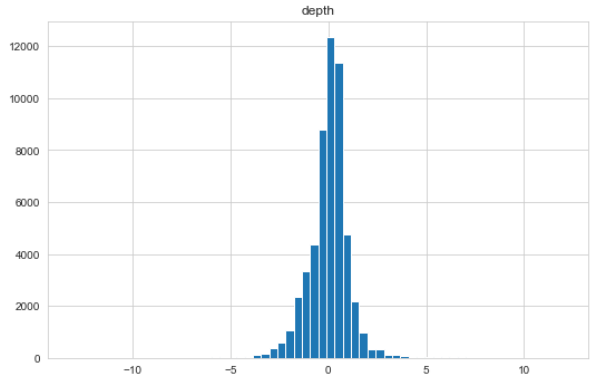
In the gas data Pearson Correlation heatmap, we see negative high correlation on AT column with our target variable NOX. This tells increasing AT tend to lowers NOX values. But rest of the variables don't seem very correlated with the NOX variable. This tells that we might have a hard time to have good results for the predictions (at least for the linear models).

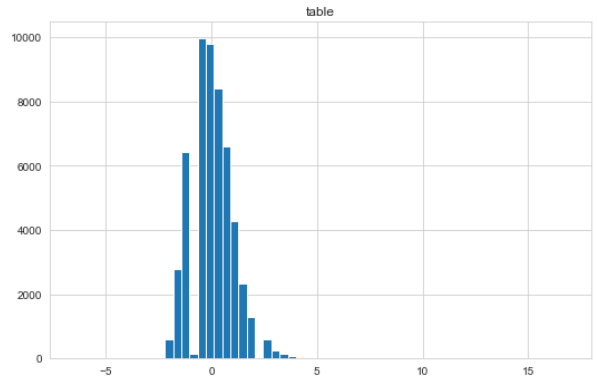
What is interesting though other features seem to be correlated with each other, TAT, TEY, CDP and GTEP looks like very correlated with each other. TAT has high negative correlation with TEY, CDP and GTEP. GTEP correlation with TEY and CDP is very high and positive around 0.96, 0.98. The relations are same betweeen these 3 variables, this means we can drop 2 of them and only keep one, otherwise our models would put more emphasis on these data in order to predict the results.

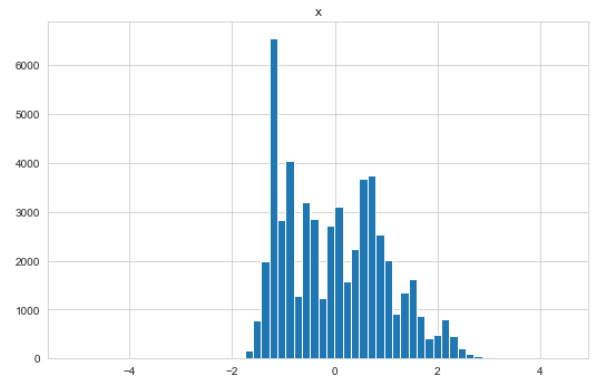
Question 3¶ Histogram of the numerical features for each dataset can be seen below:

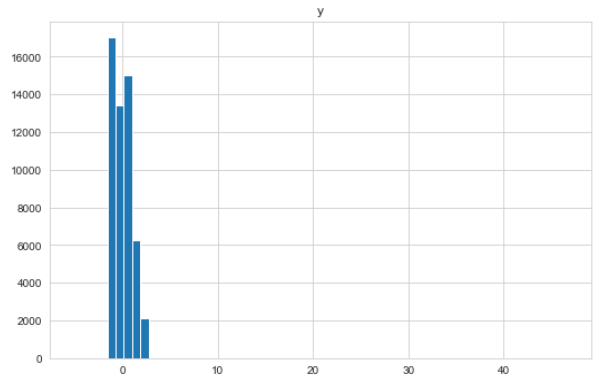
Diamond Dataset:¶ Plot:

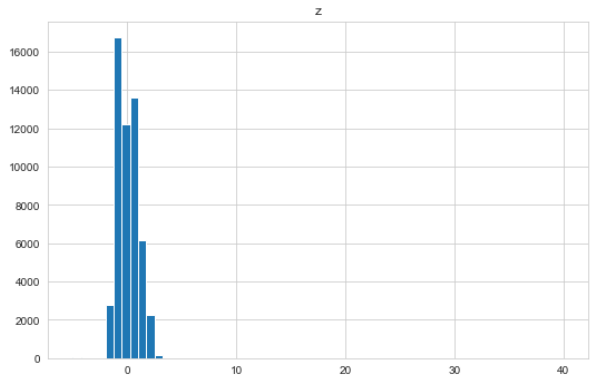










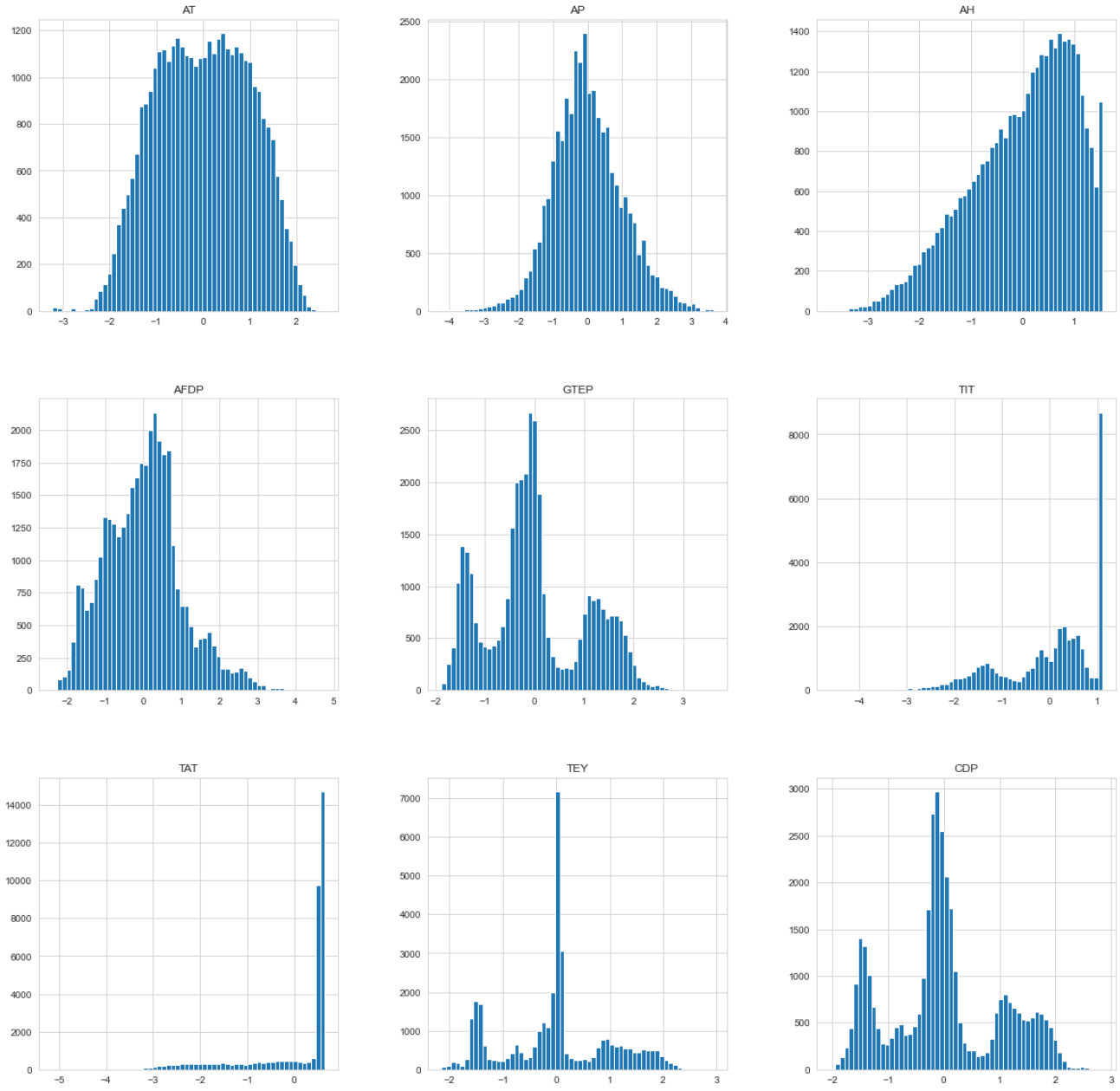


	count	mean	std	min	25%	50%	75%	max
carat	53940.0	2.444878e-16	1.000009	-1.261458	-0.839523	-0.206621	0.510668	8.886075
cut	53940.0	1.454281e-16	1.000009	-2.600864	-0.809695	0.085889	0.981473	0.981473
color	53940.0	1.338360e-16	1.000009	-2.002131	-0.826413	-0.238555	0.937163	1.525021
clarity	53940.0	-8.114467e-17	1.000009	-1.852335	-0.638095	-0.030975	0.576145	2.397505
depth	53940.0	-3.996902e-15	1.000009	-13.087603	-0.523105	0.035317	0.523936	12.041392
table	53940.0	9.695207e-17	1.000009	-6.470073	-0.652139	-0.204605	0.690462	16.801666
price	53940.0	3.934802e+03	3989.442321	327.000000	952.000000	2403.000000	5327.250000	18823.000000
x	53940.0	2.782103e-16	1.000009	-5.109120	-0.910325	-0.027776	0.721054	4.465203
y	53940.0	-8.430615e-17	1.000009	-5.020931	-0.888280	-0.021474	0.705242	46.549648
z	53940.0	-2.002271e-16	1.000009	-5.014556	-0.890946	-0.012376	0.710318	40.047576

Diamond Data:

Carat and x features' histogram seems to be right skewed. Y and z plots have few outliers around 40 and they don't seem to follow gaussian distribution. Depth and table seems to be close to normal distribution.

Gas Dataset:¶ Plot:



	count	mean	std	min	25%	50%	75%	max
AT	36733.0	-1.176081e-16	1.000014	-3.215577	-0.796488	0.011853	0.799247	2.603647
AP	36733.0	-1.233647e-14	1.000014	-4.211524	-0.660683	-0.072744	0.608027	3.640553
AH	36733.0	-5.942306e-16	1.000014	-3.719067	-0.669311	0.179998	0.795855	1.544343
AFDP	36733.0	-1.015144e-15	1.000014	-2.375059	-0.736399	0.015741	0.583238	4.761549
GTEP	36733.0	5.230467e-16	1.000014	-1.874640	-0.580281	-0.109583	0.833480	3.611191
TIT	36733.0	9.609823e-15	1.000014	-4.597825	-0.549043	0.255012	0.887990	1.110388
TAT	36733.0	-8.300659e-15	1.000014	-5.132585	-0.210240	0.543896	0.567280	0.650586

	count	mean	std	min	25%	50%	75%	max
TEY	36733.0	9.888369e-16	1.000014	-2.144032	-0.579854	0.014316	0.676995	2.944830
CDP	36733.0	-4.673376e-16	1.000014	-2.028623	-0.574519	-0.087736	0.729692	2.845821
NOX	36733.0	6.529307e+01	11.678357	25.905000	57.162000	63.849000	71.548000	119.910000
year	36733.0	-7.427883e-17	1.000014	-1.399443	-0.694695	0.010053	0.714802	1.419550

Gas Data:

AT, AP features seem to have close distributions to normal distributions. AH, AFDP are left skewed, CO is right skewed and features such as GTEP, TEY, CDP are tri-modal dsitributions.

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

There are different ways to handle high skewed data and reduce its skewness:

1- Log Transform (natural log)
2- Square root transform
2- Box-Cox transformation

There are different transformation possibilities to address the skewness of the data, and it is difficult to determine which one needs to be applied. This generally depends highly on the current doistribution of the data. Box-Cox transformation is a family of transformations where you select λ value typically from -5 to 5. Optimal λ is chosen based on the data where its transformation is the best approximation of a normally distributed curve. The transformation of data y is simply $y_\text{transformed} = y^\lambda$.

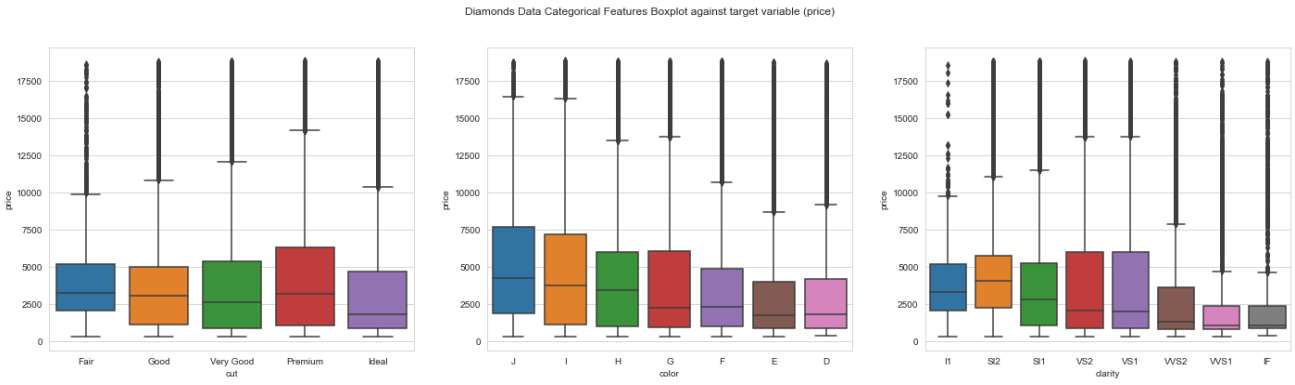
$\lambda=0$ corresponds to the natural log of the data.
$\lambda=0.5$ corresponds to the square root transform of the data.

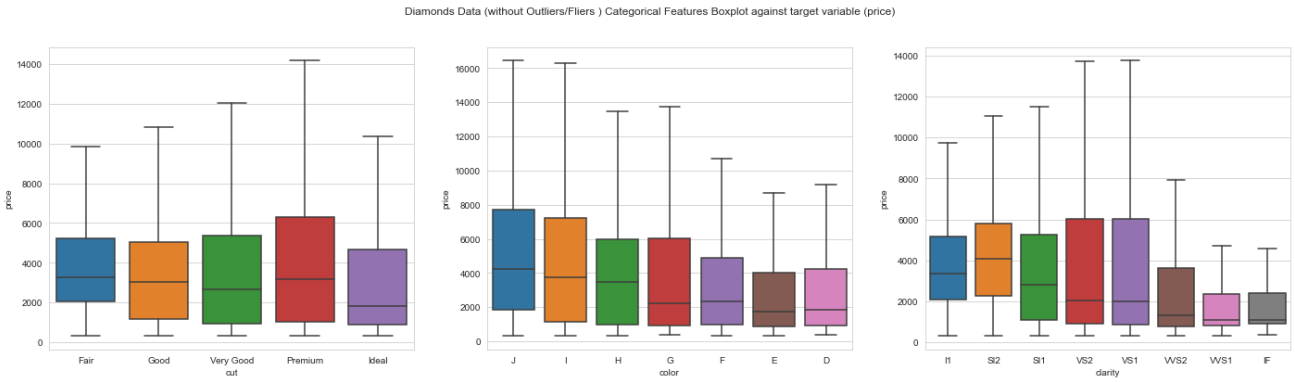
Hence, the first two approach is also contained within the Box-Cox transformation family.

By applying Box-Cox Transformation one can decrese the high skewness and make the data distribution closer to normal distribution.

Question 4¶ Box plot of categorical features in each dataset against target variable. For each dataset I plotted two boxplots one with fliers and the other one without fliers (showfliers=False to not show outliers). I added the plots without fliers as I noticed that there are extreme values that stretch the plots too much and makes the boxplots compact, by not showing outliers, I can zoom in the boxplot itself to visually better see the differences between category values.

Diamonds Data:¶ Plot:





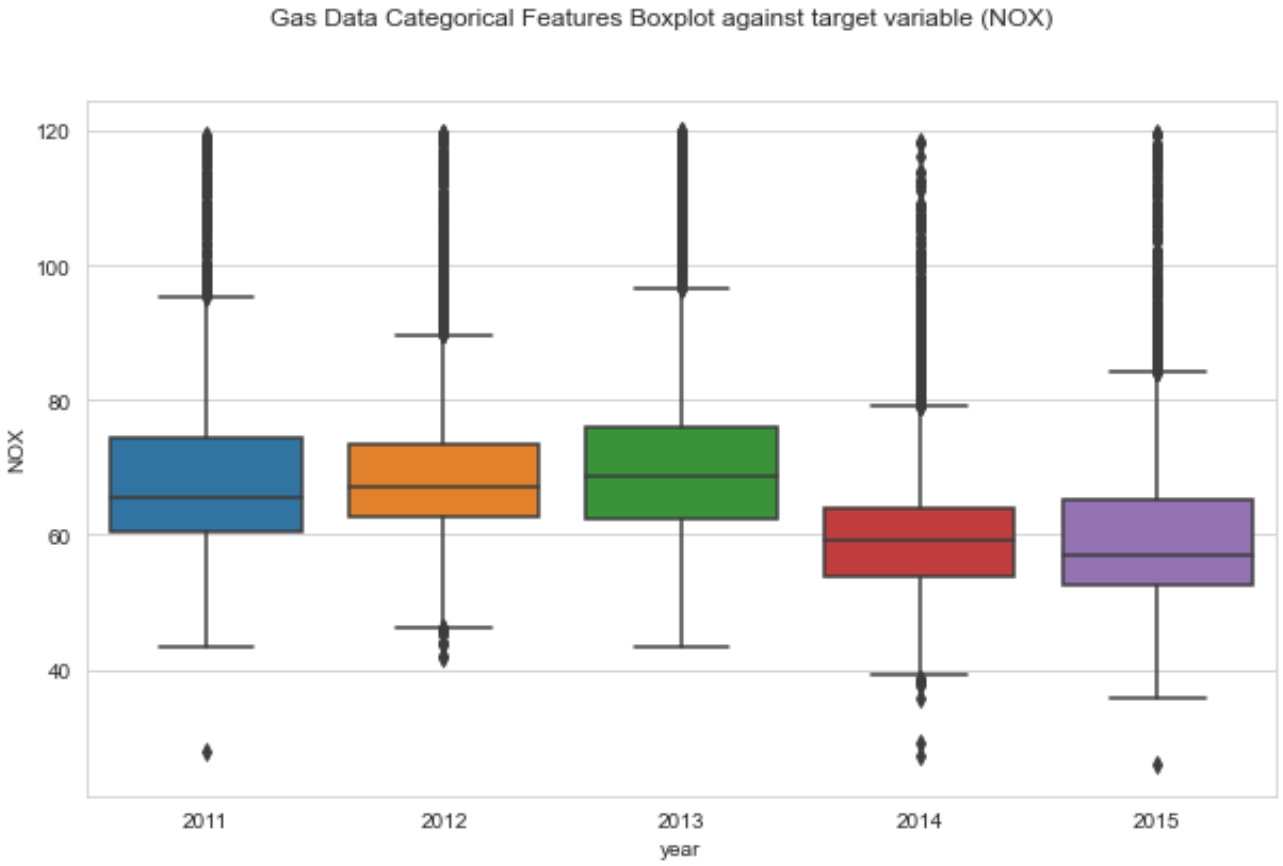
Diamond Dataset:

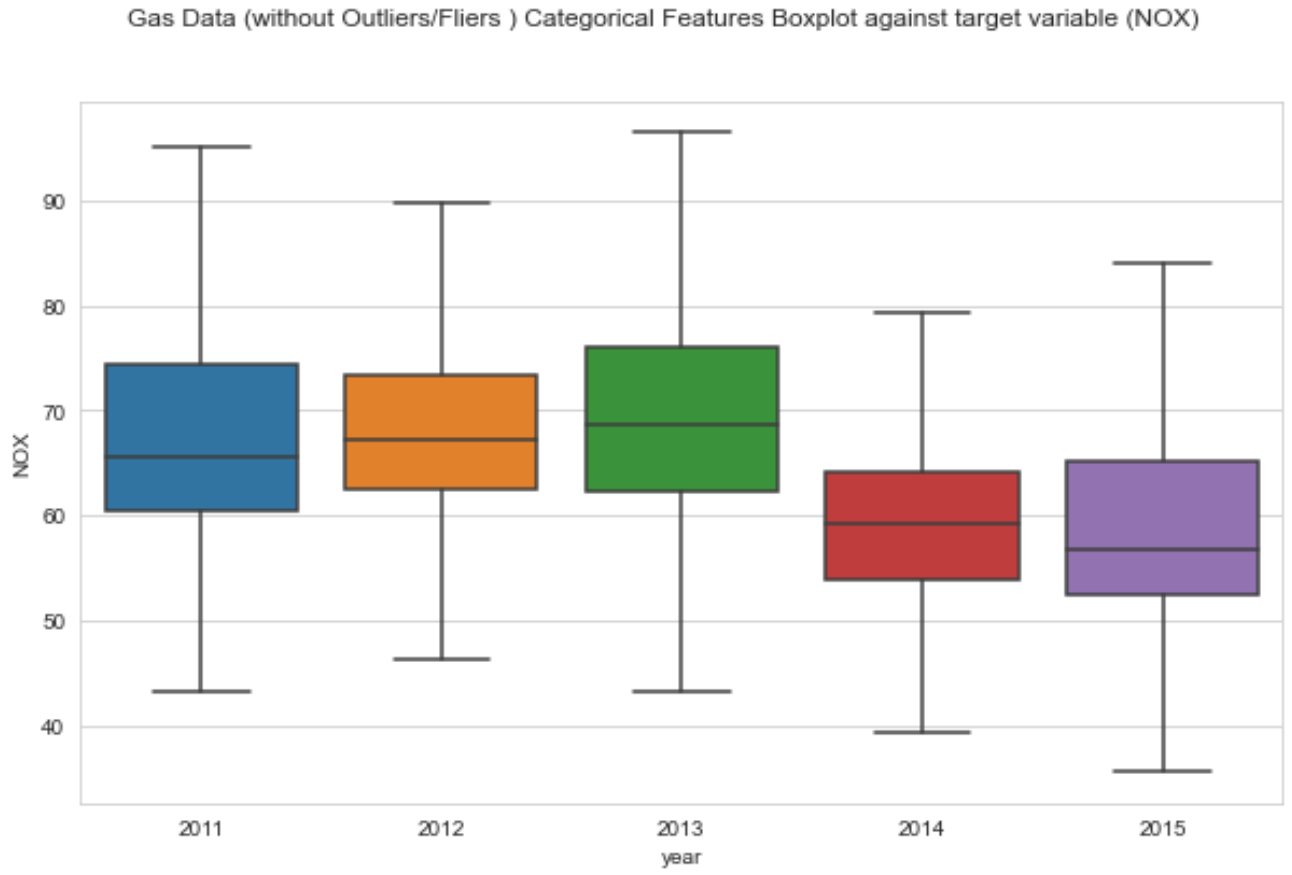
Cut boxplots: The minimum of the cut categories are all very close. The median values are close to each other except for the Ideal cut where the median price is lower compared to its counterparts. One surprising thing to see is that even though Ideal is the best quality, the median, 3rd percentile and maximum (Q3+1.5*IQR) prices are lower compared to Premium and Very Good cut categories. Premium cut has a higher 3rd percentile and maximum (Q3+1.5*IQR) values compared to other categories. All cuts have outlier prices that goes as high as around 17500. From the plot, we can say that the median and minimum price ranges are similar between cut types.

Color boxplots: J color seems to have higher prices since its 1st quartile, median and 3rd quartile are higher than its counter parts. If we group colors by similar price distributions we can say that J,I pricing distribution are close to each other, with the highest price distributions and E,D is the colors where diamond prices are lower overall. Though outliers exists for all color types that has peak prices. For lower color quality the median and 3rd quartile prices are higher compared to the high quality colors.

Clarity boxplots: This is the variable that has most diverse price distributions on its categories. The highest median and minimum value price belongs to SI2. VS2 and VS1 distiributions looks like very similar, these two types of clarity seems to have a lower median but highest 3rd quartile compared to other clarity levels. Best clarity level IF, price distribution is more compact toward lower price ranges. Again all categories have many outliers.

Gas Data:¶ Plot:

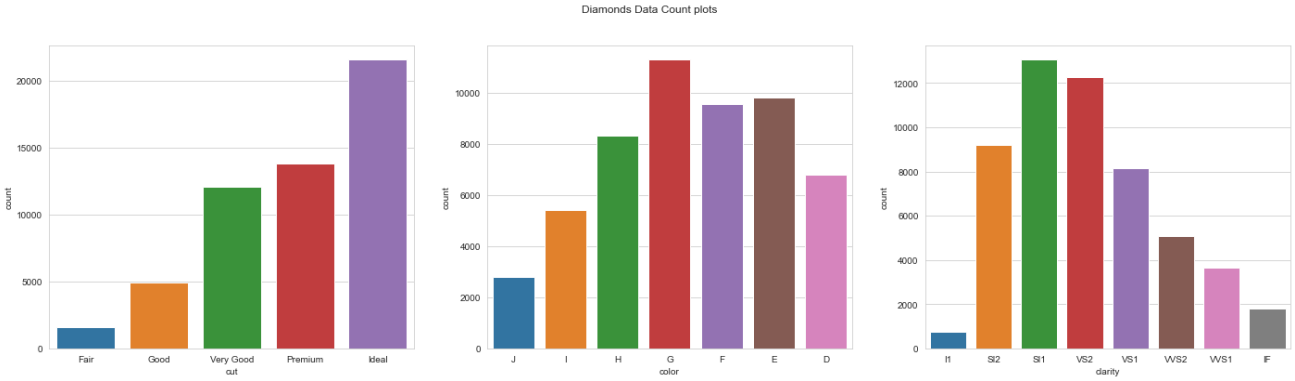

Gas Data Categorical Features Boxplot against target variable (NOX)
NOX
120
100
80
60
40
2011
2012
2013
2014
2015
year



Gas Dataset:

year boxplots: The NOX distributions seem to be very similar for years 2011, 2012 and 2013. Medians are similar for these 3 years. Although 1st, 3rd and minimum, maximum values of these 3 boxplots are different ranges. When we check 2014 and 2015, on the other hand, we see that NOX ranges captures are way lower compared the previous 3 years. The median values are lower, but even the 3rd quartiles and maximum for these 2 years are considerably lower than the previous 3 years. This can help us understand how NOX levels changed in Turkey within the 5 years.

Question 5¶ Plot:



In the diamonds data:

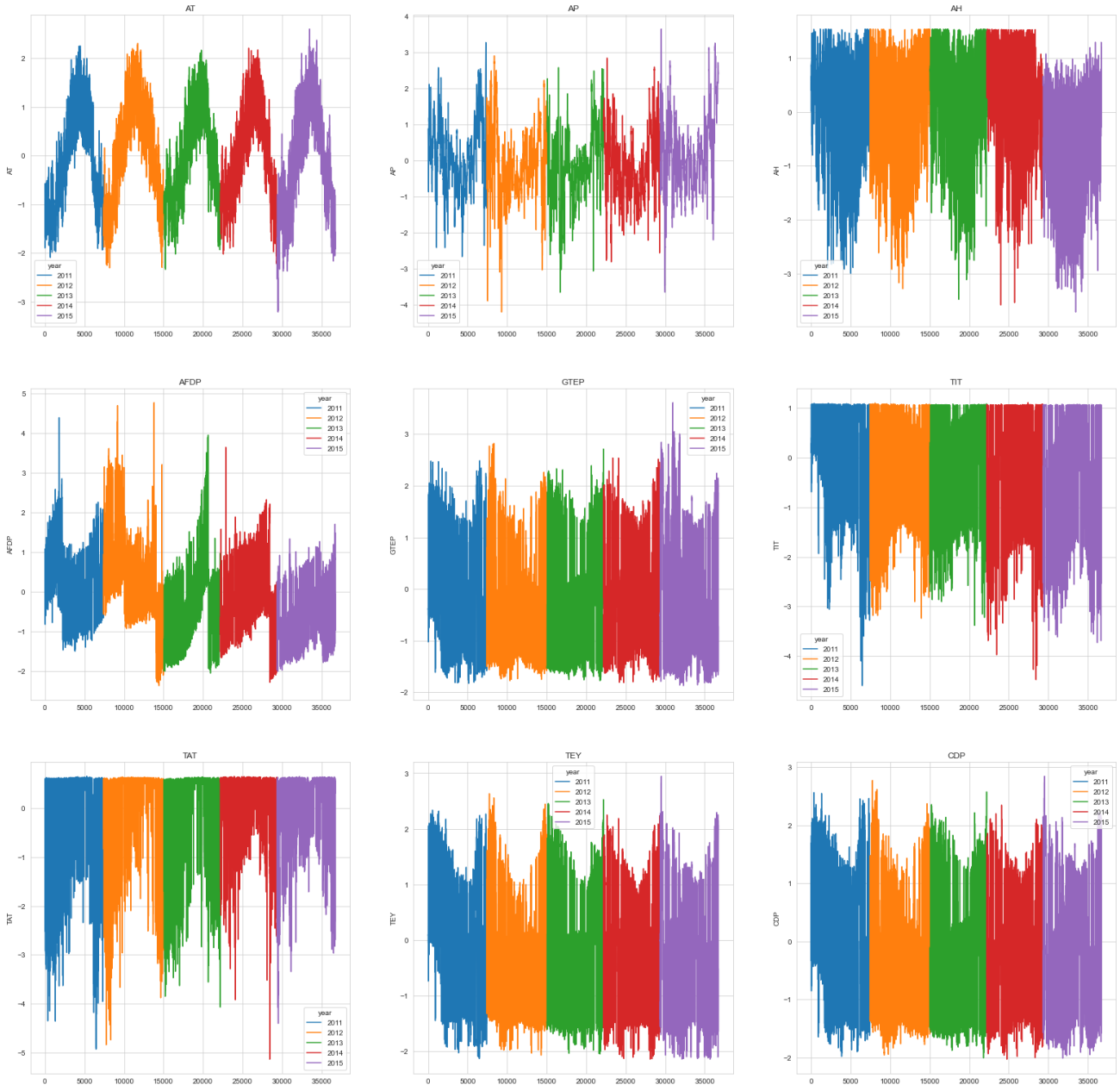
Cut categories is not equally distributed in the data, ideal cut has the largest sample size. Actually cut quality vs sample size for that cut has positive relationship. Better quality cuts have more samples in the dataset.

Color categories are better distributed in the dataset compared to cut, but still not equally distributed per se. J has the lowest sample count in the dataset whereas G has the highest. Looks like the majority of the data has medium level quality colors.

Clarity variable for the data is uneven as well. The majority of the samples in the dataset lies within SI2, SI1 and VS2 categories (low-medium quality). There are fewer samples for the worst and best clarity diamonds.

Question 6¶ 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.

I plotted yearly trends for each feature in the Gas dataset, each year is colored to be able to see patterns more clearly. Indices used as times.



From the abov plots, we can see that there are clear yearly patterns for all of the features. This is most obvious in the AT feature where we can see the same reversed V shape for each year. Some feature results are more varied within year such as TEY and CDP where it makes harder to understand the pattern, however even for those ones we can see some patterns such as towards the middle-end of each year there are some drops for all years compared to the beginning and end og each 5 years.

Feature Selection¶ For this part Mutual Info Regressiong and F regression is used from sklearn library for both datasets. For each preprocessed feature I computed the mutual info and f regression values against target variable.

Question 7¶ Mutual Info (MI) Regression calculates the dependency between two vairables and it should be a nonnegative value. 0 means two variables are independent from each other, higher MI values corresponds to higher dependency between the variables.

F Regreesion performs univariate linear regression tests to capture the relationship between the given variables.

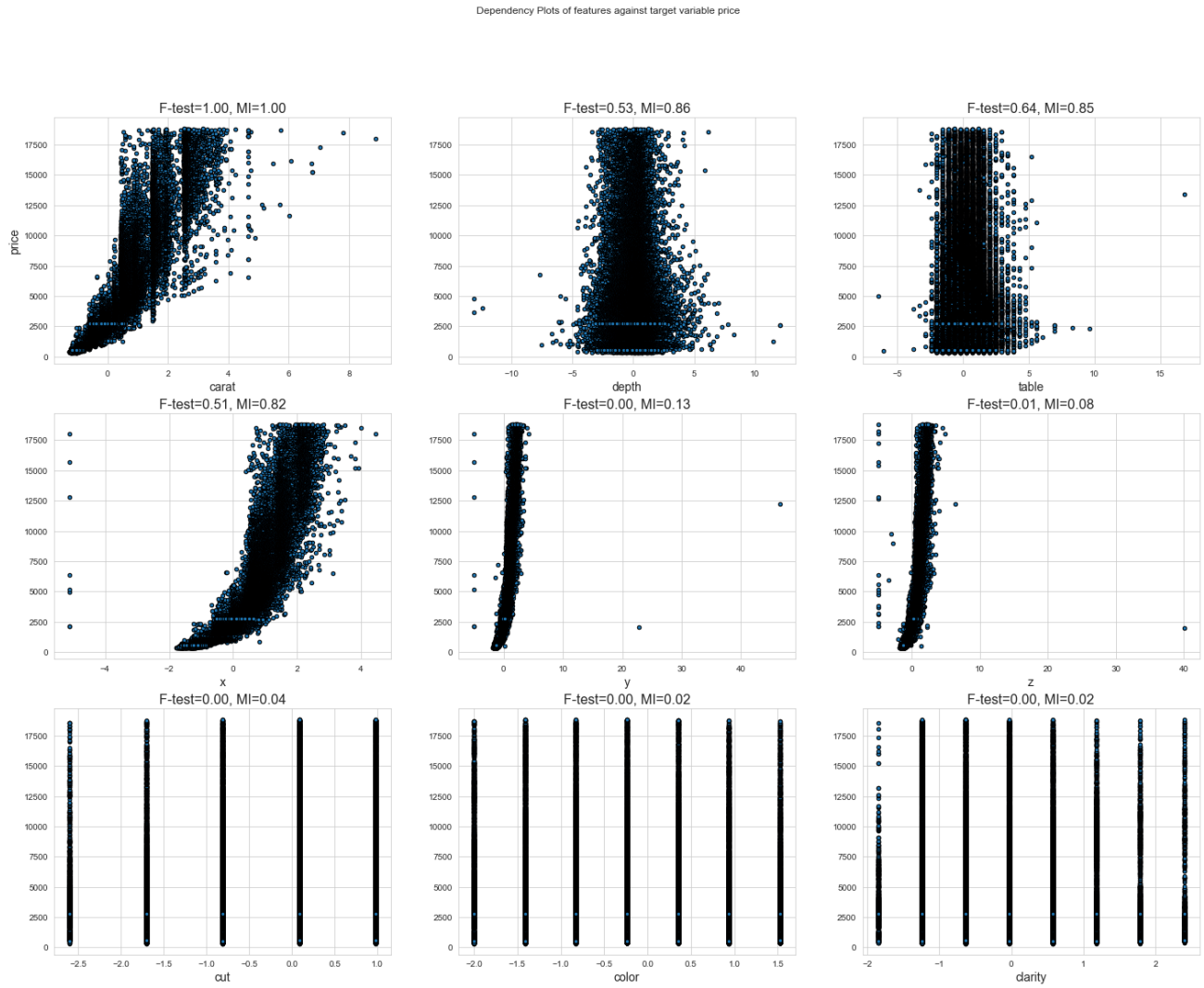
Diamonds Data¶ Diamonds Data Mutual Info Regression & F Regression

	Feature Name	Mutual Info Regression Score	Mutual Info Regression Score - Scaled
0	carat	1.65247	1
4	y	1.42218	0.860636
3	x	1.41241	0.854726
5	z	1.36035	0.823219
8	clarity	0.217059	0.131354
7	color	0.137524	0.0832233
6	cut	0.0600886	0.0363628
2	table	0.0329356	0.0199311
1	depth	0.0301225	0.0182287

	Feature Name	F Regression Score	F Regression Score - Scaled
0	carat	304051	1
4	y	160916	0.529238
3	x	193742	0.6372
5	z	154923	0.50953
8	clarity	1188.01	0.00390726
7	color	1654.4	0.00544119
6	cut	154.784	0.000509073
2	table	886.119	0.00291437
1	depth	6.11586	2.01146e-05

Table above shows the mutual info and f regression reasult for each variable against target variable price. Table is ordered by Mutual Info Regression Score - Scaled column.

Diamonds Data Scatter Plot of features vs target variable along with Scaled Mutual Info and F Regression Scores



Diamonds Data Most important Features:

Mutual Info Regression most important 4 features: carat, y, x, z.

F Regression most important 4 features: carat, x, y, z.

The order between mutual info and f regression for feature importances are slightly different.

I also experimented on different feature sets with Linear Regression model to see how their performance would change. All features in the diamond dataset seems to be helping for prediction since subset of the features perform worse with Linear Regression model. Given that diamonds dataset has very few variables, I decided to keep all of them and skip the feature selection part. In addition to this the experiments done on Q11 supports this decision, with the OLS experiment some results with other linear packages show that all features are important with p value being less than 0.05. You can refer to Q11 explanation for more details on this.

```
10-fold CV Linear Regression Experiment with All Features:
 Features:['carat', 'depth', 'table', 'x', 'y', 'z', 'cut', 'color', 'clarity']
Linear Regression (OLS) Train Avg RMSE accross 10-fold cv:  1206.280596794617
Linear Regression (OLS) Validation Avg RMSE accross 10-fold cv:  1205.045050850985


10-fold CV Linear Regression Experiment with Top 6 Features:
 Features:['carat', 'x', 'y', 'z', 'clarity', 'color']
Linear Regression (OLS) Train Avg RMSE accross 10-fold cv:  1221.6042206336504
```

```
Linear Regression (OLS) Validation Avg RMSE accross 10-fold cv:  1228.2203041711414


10-fold CV Linear Regression Experiment with Top 4 Features:
 Features:['carat', 'x', 'y', 'z']
Linear Regression (OLS) Train Avg RMSE accross 10-fold cv:  1512.5608891324323
Linear Regression (OLS) Validation Avg RMSE accross 10-fold cv:  1405.0166387786403


Top 5 Mutual Info Regr. features:
Linear Regression (OLS) Train Avg RMSE accross 10-fold cv:  1327.377219542444
Linear Regression (OLS) Validation Avg RMSE accross 10-fold cv:  1301.0941071078018


Top 5 F Regr. features:
Linear Regression (OLS) Train Avg RMSE accross 10-fold cv:  1454.506944116645
Linear Regression (OLS) Validation Avg RMSE accross 10-fold cv:  1375.540845810717
```

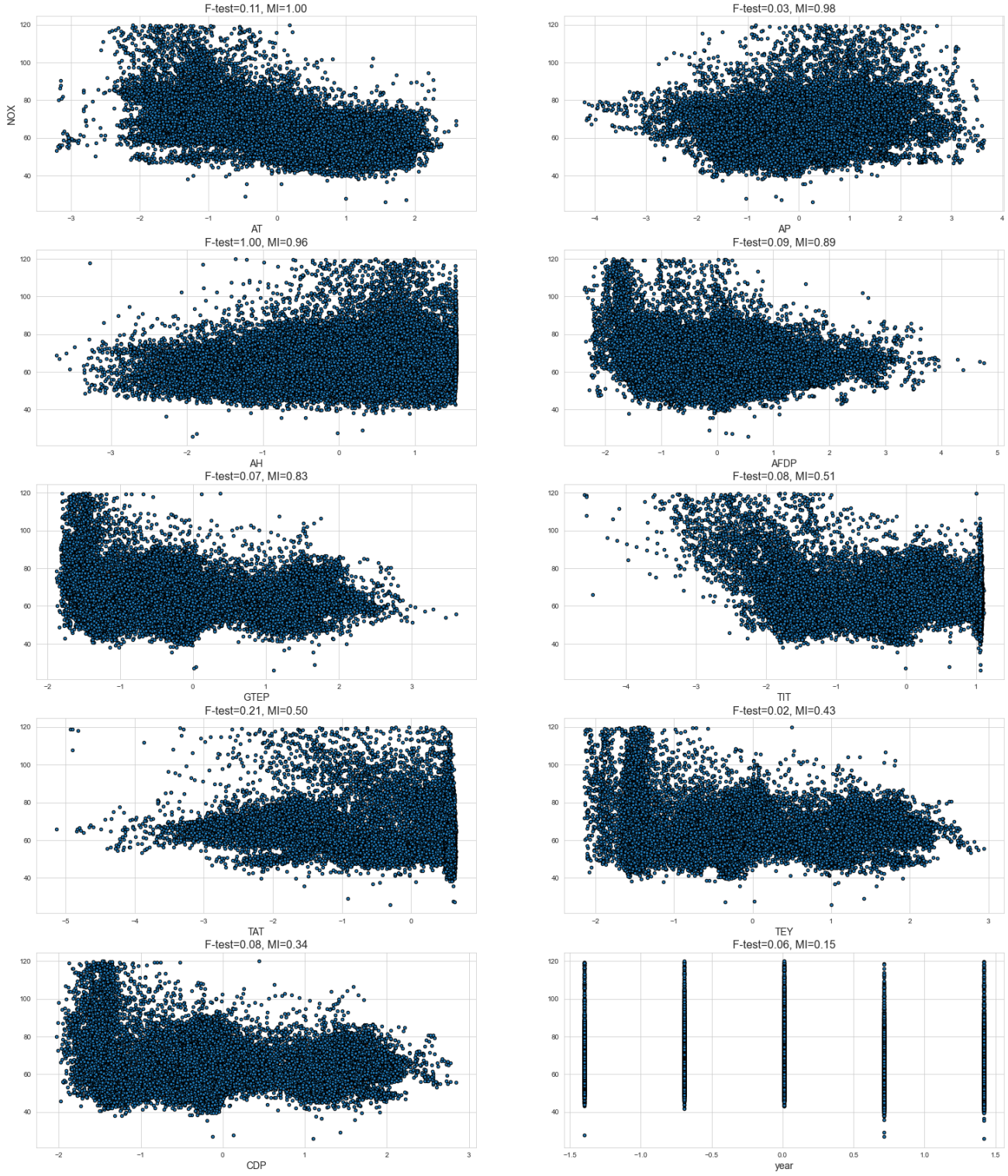
Gas Data¶ Gas Data Mutual Info Regression & F Regression

	Feature Name	Mutual Info Regression Score	Mutual Info Regression Score - Scaled
5	TIT	0.279721	1
7	TEY	0.274431	0.981089
0	AT	0.268504	0.959901
4	GTEP	0.249459	0.891814
8	CDP	0.232809	0.832292
3	AFDP	0.143041	0.51137
9	year	0.141031	0.504185
6	TAT	0.120372	0.430329
1	AP	0.0960705	0.343451
2	AH	0.0430093	0.153758

	Feature Name	F Regression Score	F Regression Score - Scaled
5	TIT	1760.54	0.105911
7	TEY	502.111	0.0302062
0	AT	16622.8	1
4	GTEP	1556.57	0.0936405
8	CDP	1109.82	0.0667649
3	AFDP	1349.46	0.0811813
9	year	3421.69	0.205844
6	TAT	319.01	0.0191911
1	AP	1404.93	0.0845184
2	AH	1023.09	0.0615477

Table above shows the mutual info and f regression reasult for each variable against target variable NOX. Table is ordered by Mutual Info Regression Score - Scaled column. TIT, TEY, AT, GTEP and CDP seems to be the most important features for Mutual Info Regression. For F Regression the most important feature seems to be AT. However, it is also important to note that the unscaled values for both of the results for each feature scores are relatively low.

Gas Data Scatter Plot of features vs target variable along with Scaled Mutual Info and F Regression Scores



Gas Data Most important Features:

Mutual Info Regression most important 5 features: TIT, TEY, AT, GTEP and CDP

F Regression most important 3 features: AT, year, TIT.

Important features found by mutual info and f regression are very different.

I also experimented on different feature sets with Linear Regression model to see how their performance would change. All features in the RMSE dataset seems to be helping for prediction since subset of the features perform worse with Linear Regression model.

```
10-fold CV Linear Regression Experiment with All Features:
 Features:['AT', 'AP', 'AH', 'AFDP', 'GTEP', 'TIT', 'TAT', 'TEY', 'CDP', 'year']
Linear Regression (OLS) Train Avg RMSE accross 10-fold cv:  8.030473106723004
Linear Regression (OLS) Validation Avg RMSE accross 10-fold cv:  8.881846484410762


10-fold CV Linear Regression Experiment with Top MI Reg. Features:
 Features:['TIT', 'TEY', 'AT', 'GTEP', 'CDP']
Linear Regression (OLS) Train Avg RMSE accross 10-fold cv:  9.14421286309772
Linear Regression (OLS) Validation Avg RMSE accross 10-fold cv:  9.455365986158789


10-fold CV Linear Regression Experiment with Top F Reg. Features:
 Features:['AT', 'year', 'TIT']
Linear Regression (OLS) Train Avg RMSE accross 10-fold cv:  8.870111551541857
Linear Regression (OLS) Validation Avg RMSE accross 10-fold cv:  9.278648847152846


Top 5 Mutual Info Regr. features:
Linear Regression (OLS) Train Avg RMSE accross 10-fold cv:  9.14421286309772
Linear Regression (OLS) Validation Avg RMSE accross 10-fold cv:  9.455365986158792


Top 5 F Regr. features:
Linear Regression (OLS) Train Avg RMSE accross 10-fold cv:  8.836818159830631
Linear Regression (OLS) Validation Avg RMSE accross 10-fold cv:  9.356935850552805
```

Though one thing I realized when I was looking at the heatmap 3 of the features are very correlated with each other, and it might makes sense to drop 2 of them. Since this is the feature selection part, I will also explore this in here.

However, it is also important that the decision to drop is only based on what I saw in the heatmap, one could make better decision by knowing the meaning of each gas types and see if they are really correlated with each other. Since I am no expert in this area, I will check below to see if dropping two of them (CDP and GTEP) will help with the model performance.

```
10-fold CV Linear Regression Experiment with Top F Reg. Features:
 Features:['AT', 'AP', 'AH', 'AFDP', 'TIT', 'TAT', 'TEY', 'year']
Linear Regression (OLS) Train Avg RMSE accross 10-fold cv:  8.036895416801087
Linear Regression (OLS) Validation Avg RMSE accross 10-fold cv:  8.713495424542154
```

The model performance is slightly improved, avg. 10-fold cross validation RMSE score decreased from 8.8 to 8.7, we can see that it doesn't hurt deleting these features. Given that 3 of these features (GTEP, TEY and CDP) are very correlated with each other, I considered to drop them. Though as I said I am no expert and a better decision can be made by experts in these area.

In addition to that the names of these features from the table are below: CDP: Compressor discharge pressure TEY: Turbine energy yield GTEP: Gas turbine exhaust pressure

Again this doesn't tell me too much about the data.

I found the paper published for this dataset and its benchmark. I realized that the authors of the paper also saw a correlation between these two variables, point out that it has benefits in predictive modelling and they might be dropped. However, in their experiment results and feature importances I saw that they kept these variables. Given that they are more knowledgable, experts in the field, the ones who prepared and benchmarked the dataset, and I didn't see huge changes in the above results, I finally decided to keep them and use all the features.

Paper link: https://journals.tubitak.gov.tr/elektrik/issues/elk-19-27-6/elk-27-6-54-1807-87.pdf

How does this step affect the performance of your models in terms of test RMSE?

Feature selection has different effects on the performance of the test data. If the removed features contains redundant data, noise or unrelated to the target variable, this step reduces the chances for overfitting, enhance the model generalization and improves the test RMSE score. This means if features selected precisely, test RMSE score is expected to lower and model to perform better. However, if deleted features contain important information that can be used for the prediction, the test RMSE score would increase and model performing worse. That is why, feature selection is a not a trivial step and should be performed carefully on which data we want to keep and remove. Feature selection also help decreasing both training and inference time given that we have to deal with less data.

If we think specifically for feature selection based on mutual information, it is not only limited to linear dependency but it rather quantifies the amount of information obtained from observing one variable about the other one. Whereas f regression captures only linear relationship between variables. Ideally, both feature selection methods expect to decrease redundant variables which leads less opportunity to overfit the model and better generalization. This means we expect to have good performing models with lower RMSE test scores. However, again we need to be careful, for example, with f regression we might possibly discard important nonlinear features during feature selection. This might hurt the model performance and increase the test RMSE score. To conclude, feature selection is a step to help reducing overfitting and to have generalization, improving both performance and time to train and inference of the model and leads to lower test RMSE score if performed successfully.

Linear Regression¶

Question 8¶ What is the objective function?

The Linear Regression model prediction is made:

\begin{equation} \hat{y} = b + \sum_{i=1}^{p} x_iw_i = x^Tw \end{equation}

$\hat{y}$ is predicted value, b bias (intercept term), p is the number for features, w is weights and x is the input data. (In the second part of the equation b is incorparated in the w in this case.).

Using RMSE as our error metric:

Ordinary Least Squares (Linear Regression without Regularization) Objective Function:

\begin{equation} \min_{w} ||(Xw - Y)||^2_2 \end{equation}

Lasso Regression (L1 Regularization) Objective Function:

\begin{equation} \min_{w} ||(Xw - Y)||^2_2 + \lambda ||w||_1 \end{equation}

Ridge Regression (L2 Regularization) Objective Function:

\begin{equation} \min_{w} ||(Xw - Y)||^2_2 + \lambda ||w||^2_2 \end{equation}

X input, Y ground truth label, w parameters and λ is the regularization strength, hyperparameter to be tuned.

Explain how each regularization scheme affects the learned hypotheses.

Regularization techniques are used to avoid model overfitting and improving model generalization by adding a penalty term to the objective function. The intuition behind regularization is that by adding penalty terms related to the model weights (coefficients), the optimizer won't try to increase some weights larger and larger, so it encourages the weights to be kept small.

- **Lasso Regression:**

Lasso is also called L1 Regularization, it adds the absolute value of magnitude of coefficients (L1 norm) as penalty term to the loss function. It's advantage is it can set some coefficients to 0 (create sparse solutions), hence it might eliminate some features and can be considered as a built-in feature selection. It has unstable solution and may have multiple solutions for the optimization problem.

- **Ridge Regression:**

Ridge also called as L2 regularization since it adds squared magnitude of coefficients (squared L2 norm) as penalty term to the loss function. It forces the coefficients to be lower, close to 0 but not 0. The output is therefore non-sparse and it doesn't choose some subset of the features. It is computationally efficient since it has analytical solutions. After L2 regularization the optimal solutions are generally more stable.

Question 9¶ For each model I applied 10-fold cross-validation and measured average RMSE errors for training and validation sets.

How to pick optimal penalty (lambda) parameter? (alpha in sklearn)

There are several ways to select best values for choosing optimal regaularization value for linear models:

1- Via cross validation and grid search
2- Using information criterion (AIC, BIC) for Lasso Regression
3- Using model specific cross validations implemented in sklearn such as LassoCV and RidgeCV.

Lasso and Ridge models also have their own special cross validation implementations in scikit-learn to find a good value of penalty scores. They are called LassoCV and RidgeCV model, but the scores are calculated in terms of MSE rather than RMSE and only test scores are returned. Given that we are also interested on the training scores for the questions, I could't use these functions.

For Lasso Regression you can see the results of different lambda (alpha in sklearn) parameters both with Cross Validation + GridSearch and with AIC, BIC method. For Ridge Regression alpha value is only done with 10-fold cross validation over gridsearch with different lambda (alpha in sklearn) values.

Note: For cross validation results, sklearn scoring function uses negative RMSE values (neg_root_mean_squared_error) to be able to minimize the loss function. That is why the returned RMSE values are negative and we can simply get their absolute results as the RMSE value.

Diamonds Data Results¶ Ordinary Least Squares (Linear Regression without Regularization)¶

```
Linear Regression (OLS) Train Avg RMSE accross 10-fold cv:  1206.280596794617
Linear Regression (OLS) Validation Avg RMSE accross 10-fold cv:  1205.045050850985
```

Check RMSE results for each of the 10-fold:

```
Train RMSEs: [1246.67693953 1248.66216822 1230.78437231 1190.65068211  954.23438537
 1157.68140298 1256.25260037 1252.99820555 1264.05281144 1260.81240006]
Validation RMSEs:  [ 942.74216047  918.48055067 1112.58133725 1480.56810359 2806.51518647
 1699.75830566  815.69881421  877.61209435  670.86071043  725.63324541]
```

Lasso Regression¶

I tried alpha values from 1e-8 to 1e6. Results can be seen from below table:

	mean_train_score	mean_validation_score	model_name	param_model__alpha
14	1206.280597	1205.045051	Lasso	1e-08
15	1206.280597	1205.045051	Lasso	1e-07
16	1206.280597	1205.045052	Lasso	1e-06
17	1206.280597	1205.045064	Lasso	1e-05
18	1206.280597	1205.045161	Lasso	0.0001
19	1206.280597	1205.046257	Lasso	0.001
20	1206.280605	1205.057629	Lasso	0.01

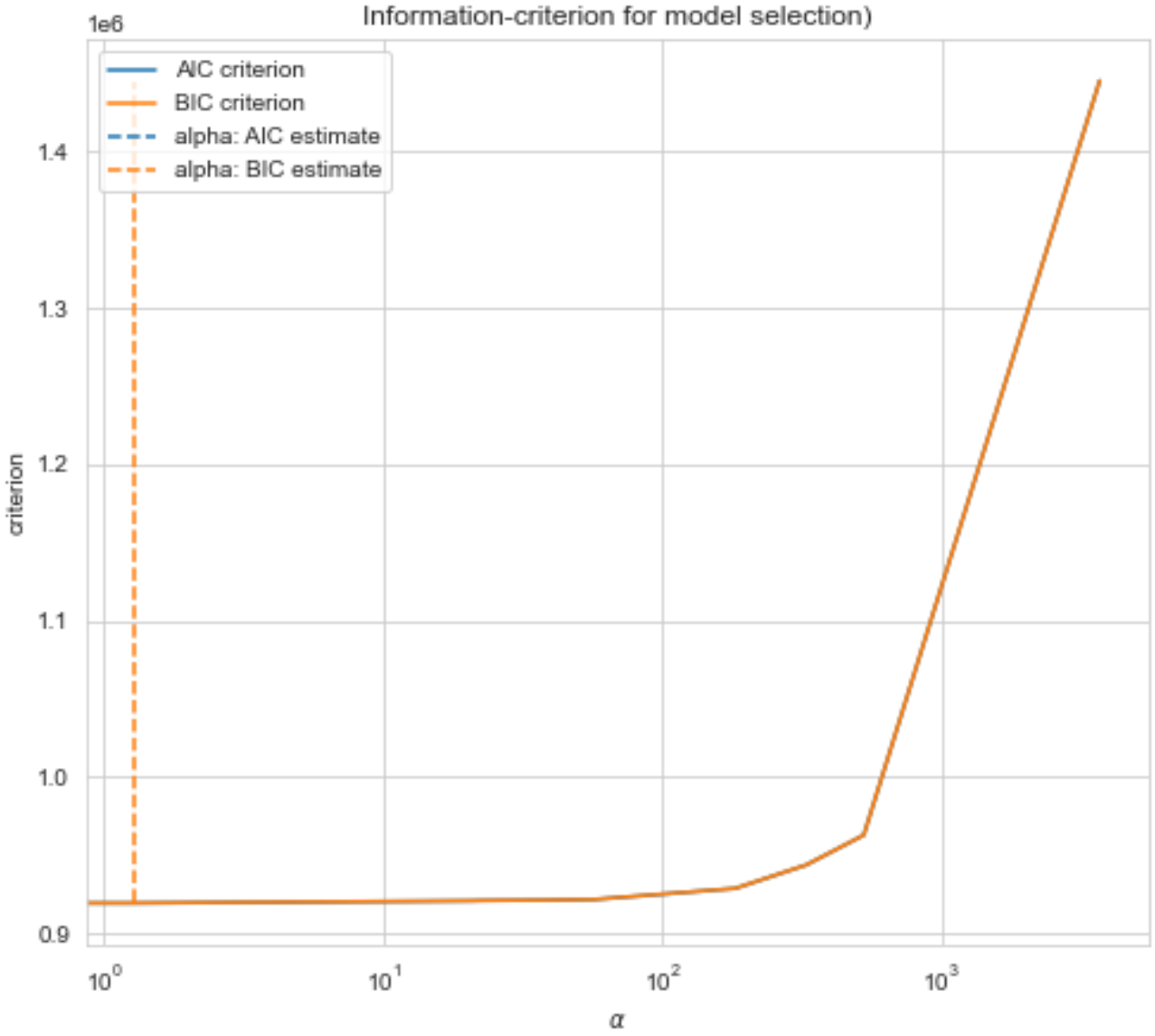
	mean_train_score	mean_validation_score	model_name	param_model___alpha
21	1206.281434	1205.175550	Lasso	0.1
22	1206.357387	1206.498138	Lasso	1
23	1210.657916	1225.290602	Lasso	10
24	1251.275772	1244.896044	Lasso	100
25	1843.911711	1770.022846	Lasso	1000
26	3957.162324	3544.036398	Lasso	10000
27	3957.162324	3544.036398	Lasso	100000

The best average train and validation RMSE scores for Lasso Regression are in the 1205-1206 ranges. A good range for alpha is in this case is around 0.01 and 1. Since the RMSE doesn't change too much, we can pick 1 as the optimal penalty parameter. The reason behind is that we want to have simpler model that could achieve a good performance. Having a higher regularization helps us having simpler model and we can see that with $\alpha=1$ we can achieve average RMSE results very close to the ones we found in the case of no regularization both in training and validation sets.

Penalty Selection based on AIC/BIC for Lasso¶

Lasso model fit with Lars using BIC (Bayes Information criterion) or AIC (Akaike information criterion) for model selection. These two criterion are used to select the regularization parameter. A good model should explain well the data while being simple to achieve that a trade-off between the goodness of fit and the complexity of the model is made.

alphas	AIC criterion	BIC criterion
3676.601788047377	1445444.318225	1445444.318225
526.0062287770186	962964.176573	962973.072200
327.32903256364165	943873.969885	943891.761140
182.5867641590675	928902.677777	928929.364659
57.28201898142053	921854.920728	921890.503239
38.14369461064738	921397.514284	921441.992422
22.73816487193749	921114.657233	921168.030999
20.709994866398944	920959.220855	921021.490248
1.2994435848907804	919456.746629	919527.911649
0.0	919448.355752	919528.416401



According to AIC criterion good alpha value is 0, and BIC criterion is around 1.3. This also supports the result we found above.

Ridge Regression¶

I tried alpha values from 1e-8 to 1e6. Results can be seen from below table:

	mean_train_score	mean_validation_score	model_name	param_model___alpha
0	1206.280597	1205.045051	Ridge	1e-08
1	1206.280597	1205.045051	Ridge	1e-07
2	1206.280597	1205.045051	Ridge	1e-06
3	1206.280597	1205.045052	Ridge	1e-05
4	1206.280597	1205.045059	Ridge	0.0001
5	1206.280597	1205.045137	Ridge	0.001
6	1206.280597	1205.045909	Ridge	0.01
7	1206.280598	1205.053637	Ridge	0.1
8	1206.280754	1205.131037	Ridge	1
9	1206.295931	1205.916546	Ridge	10
10	1207.536689	1214.414691	Ridge	100

	mean_train_score	mean_validation_score	model_name	param_model__alpha
11	1248.699264	1292.818122	Ridge	1000
12	1458.983908	1489.476610	Ridge	10000
13	2112.850610	1950.711595	Ridge	100000

In the case of Rigde regression we can increase the regularization in the range of 1-10. With the similar reasoning explained for Lasso, I will pick 10 as the optimal parameter.

Best Regularization with optimal penalty parameter¶

In the diamonds dataset, both Lasso and Ridge Regularization reach the same optimal results for average RMSE on training and validation sets. These results are also same for Linear Regression without regularization and lies in within 1205-1206 RMSE score. If we only look at the average optimal results, these 3 models don't really have much difference. When I checked the std of the train and validation RMSE scores for 10-fold cv(shown below) and the scores for each fold individually, again there is nothing very different for Lasso and Ridge Regression.

If I choose one, I would pick Ridge regularization since it seems to be better regularization scheme for diamonds data for the overall RMSE scores for higher regularization values. The avg. scores are way lower in Ridge then Lasso. For example Lasso when alpha=1000 avg RMSE validation score=1770 and Ridge when alpha=1000 avg RMSE validation score=1292. Therefore, Ridge regularization is best regularization scheme with optimal penalty 1.

Ridge Regression Results (test score columns are actually validation score here, crossvalidate names them as test.)

	8	9
split0_test_score	-943.047423	-945.772298
split1_test_score	-918.745592	-921.113590
split2_test_score	-1112.593318	-1112.713075
split3_test_score	-1480.207682	-1476.998752
split4_test_score	-2806.756166	-2809.011538
split5_test_score	-1700.057234	-1702.738090
split6_test_score	-815.570266	-814.432034
split7_test_score	-877.496956	-876.470311
split8_test_score	-670.987363	-672.135127
split9_test_score	-725.848366	-727.780644
mean_validation_score	1205.131037	1205.916546
std_test_score	617.997804	618.307116
rank_test_score	16.000000	18.000000

	8	9
split0_train_score	-1246.677053	-1246.688078
split1_train_score	-1248.662286	-1248.673713
split2_train_score	-1230.784516	-1230.798464
split3_train_score	-1190.650864	-1190.668558
split4_train_score	-954.234736	-954.268267
split5_train_score	-1157.681538	-1157.694613
split6_train_score	-1256.252739	-1256.266196
split7_train_score	-1252.998340	-1253.011377
split8_train_score	-1264.052939	-1264.065238
split9_train_score	-1260.812526	-1260.824805
mean_train_score	1206.280754	1206.295931
std_train_score	90.084923	90.078895

Lasso Regression Results (test score columns are actually validation score here, crossvalidate names them as test.)

	22	23
split0_test_score	-948.420031	-1001.156908
split1_test_score	-923.845640	-974.532777
split2_test_score	-1113.481926	-1125.583926
split3_test_score	-1476.159596	-1436.060043
split4_test_score	-2810.451532	-2869.030266
split5_test_score	-1705.121582	-1761.357107
split6_test_score	-812.459728	-788.742713
split7_test_score	-874.713354	-848.425307
split8_test_score	-671.818620	-684.250555
split9_test_score	-728.509373	-763.766418
mean_validation_score	1206.498138	1225.290602
std_test_score	618.777888	632.064742
rank_test_score	19.000000	21.000000

	22	23
split0_train_score	-1246.742106	-1250.603468
split1_train_score	-1248.726980	-1252.591315
split2_train_score	-1230.852788	-1234.881322
split3_train_score	-1190.696045	-1194.615517
split4_train_score	-954.420070	-961.823509
split5_train_score	-1157.758913	-1162.617515
split6_train_score	-1256.316502	-1260.161182
split7_train_score	-1253.061068	-1256.757931
split8_train_score	-1264.115736	-1267.712219
split9_train_score	-1260.883657	-1264.815178
mean_train_score	1206.357387	1210.657916
std_train_score	90.051053	88.987987

Best model feature importance:

From the best found linear regression model (Ridge Regression) we see the coefficients below:

	carat	x	clarity	color	cut	depth	table	y
Coefficients	5092.655074	-984.480857	826.612634	548.94111	134.816544	-114.319163	-59.802654	49.963989
coef_absolute	5092.655074	984.480857	826.612634	548.94111	134.816544	114.319163	59.802654	49.963989

According to coeffs, the most important feature is carat, followed by x, clarity, color and cut. x, depth, table and z has reverse relation with target variable whereas the other features have positive correlation.

Gas Data Results¶ Ordinary Least Squares (Linear Regression without Regularization)¶

```
Linear Regression (OLS) Train Avg RMSE accross 10-fold cv:  8.030473106723004
Linear Regression (OLS) Validation Avg RMSE accross 10-fold cv:  8.881846484410762
```

Check RMSE results for each of the 10-fold:

```
Train RMSEs: [8.20232426 8.0507639  8.1187614  8.17650722 7.74108808 7.83083707
 7.79695147 8.26129408 8.19835912 7.92784446]
Validation RMSEs:  [ 7.43536346  8.57794857  7.89249311  7.21361859 11.91720141 10.28186932
 11.55608868  6.192516    7.87080299  9.88056272]
```

Lasso Regression¶

I tried alpha values from 1e-8 to 1e6. Results can be seen from below table:

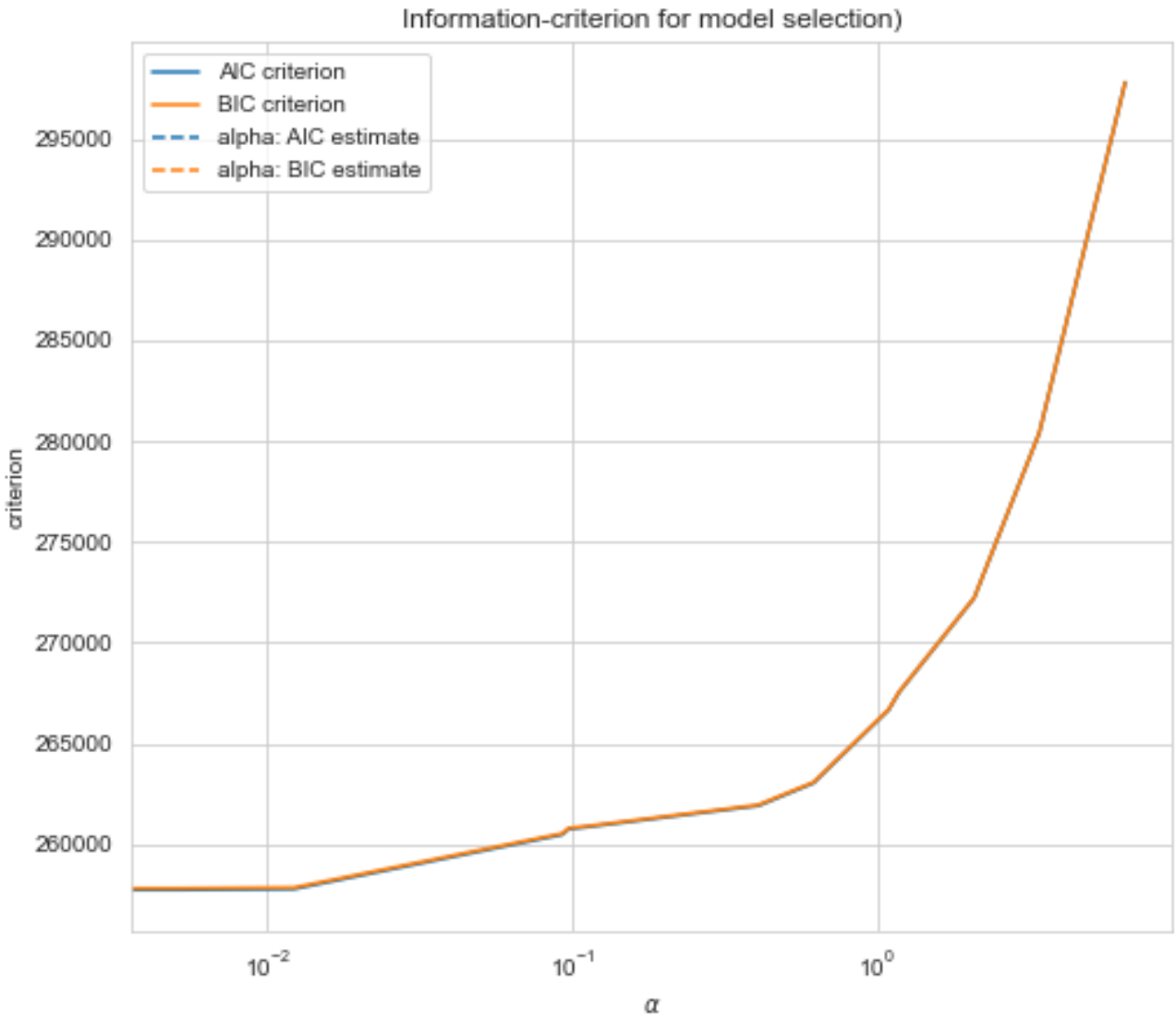
	mean_train_score	mean_validation_score	model_name	param_model___alpha
14	8.030497	8.890395	Lasso	1e-08
15	8.030497	8.890394	Lasso	1e-07
16	8.030497	8.890384	Lasso	1e-06
17	8.030496	8.890279	Lasso	1e-05
18	8.030491	8.889226	Lasso	0.0001
19	8.030507	8.878864	Lasso	0.001
20	8.036236	8.802989	Lasso	0.01
21	8.343862	8.958754	Lasso	0.1
22	8.888678	9.370859	Lasso	1
23	11.656838	11.747458	Lasso	10

	mean_train_score	mean_validation_score	model_name	param_model___alpha
24	11.656838	11.747458	Lasso	100
25	11.656838	11.747458	Lasso	1000
26	11.656838	11.747458	Lasso	10000
27	11.656838	11.747458	Lasso	100000

The best average train and validation RMSE scores for Lasso Regression are in the 8.03-8.9 ranges. A good alpha value in this case is 0.01 where validation score is minimized.

Penalty Selection based on AIC/BIC for Lasso¶

alphas	AIC criterion	BIC criterion
6.518461607428588	297779.636574	297779.636574
3.408417829284328	280395.454558	280403.965989
2.0843371150909515	272207.806361	272224.829222
1.1822885716687122	267530.673487	267556.207779
1.0936028353078546	266662.328503	266696.374226
0.6198812853548217	263032.618171	263075.175325
0.42322756024735103	261998.610442	262049.679027
0.4214186369792278	261986.086025	262037.154610
0.4080053049800678	261905.063659	261947.620813
0.10872696996953297	260829.745824	260872.302978
0.09753867075119184	260748.879327	260799.947911
0.09288997859137962	260471.894052	260531.474068
0.012270190993634241	257779.809840	257847.901287
0.005124243932382602	257735.355737	257811.958615
0.0	257718.905523	257804.019831



According to AIC and BIC criterion good alpha value is 0.

Ridge Regression¶

I tried alpha values from 1e-8 to 1e6. Results can be seen from below table:

	mean_train_score	mean_validation_score	model_name	param_model___alpha
0	8.030473	8.881846	Ridge	1e-08
1	8.030473	8.881846	Ridge	1e-07
2	8.030473	8.881846	Ridge	1e-06
3	8.030473	8.881846	Ridge	1e-05
4	8.030473	8.881846	Ridge	0.0001
5	8.030473	8.881841	Ridge	0.001
6	8.030473	8.881795	Ridge	0.01
7	8.030473	8.881332	Ridge	0.1
8	8.030498	8.876794	Ridge	1
9	8.032520	8.839102	Ridge	10
10	8.094992	8.761643	Ridge	100
11	8.316408	8.949233	Ridge	1000
12	8.778942	9.323730	Ridge	10000

	mean_train_score	mean_validation_score	model_name	param_model___alpha
13	10.333445	10.634421	Ridge	100000

In the case of Rigde best alpha value is 100 where validation score is minimized.

Best Regularization with optimal penalty parameter¶

Top 3 results from experiment:

	mean_train_score	mean_validation_score	model_name	param_model___alpha
10	8.094992	8.761643	Ridge	100
20	8.036236	8.802989	Lasso	0.01
9	8.032520	8.839102	Ridge	10

In the gas dataset, both Lasso and Ridge Regularization reach similar optimal results for average RMSE on training and validation sets. These results are also very close for Linear Regression without regularization and lies in within 8-8.8 RMSE score. Ridge regularization seem to perform slighlty better than Lasso, therefore the best regularization scheme for this dataset is Ridge with alpha 100.

Best model feature importance:

From the best found linear regression model (Ridge Regression) we see the coefficients below:

	Coefficients	coef_absolute
TEY	-15.753769	15.753769
TIT	13.057814	13.057814
AT	-10.803841	10.803841
TAT	-5.660252	5.660252
AH	-3.279827	3.279827
CDP	-2.640984	2.640984
year	-2.102134	2.102134
AP	-1.204434	1.204434
AFDP	0.303540	0.303540
GTEP	-0.263377	0.263377

According to coeffs, the most important 5 features are TEY, followed by TIT, then AT, TAT and AH. Only TIT is the positively correlatead feature with NOX within these 5 important features found.

Question 10¶ The same experiments as Q9 applied on unscaled features.

Diamond Dataset Unscaled Feature results¶

```
Linear Regression (OLS) Train Avg RMSE accross 10-fold cv:  1206.2805967946167
Linear Regression (OLS) Validation Avg RMSE accross 10-fold cv:  1205.045050850984
```

Check RMSE results for each of the 10-fold:

```
Train RMSEs: [1246.67693953 1248.66216822 1230.78437231 1190.65068211  954.23438537
 1157.68140298 1256.25260037 1252.99820555 1264.05281144 1260.81240006]
Validation RMSEs:  [ 942.74216047  918.48055067 1112.58133725 1480.56810359 2806.51518647
 1699.75830566  815.69881421  877.61209435  670.86071043  725.63324541]
```

	mean_train_score	mean_validation_score	model_name	param_model__alpha
14	1206.280597	1205.045051	Lasso	1e-08
15	1206.280597	1205.045051	Lasso	1e-07
16	1206.280597	1205.045053	Lasso	1e-06
17	1206.280597	1205.045073	Lasso	1e-05
18	1206.280597	1205.045263	Lasso	0.0001
19	1206.280597	1205.047165	Lasso	0.001
20	1206.280609	1205.066857	Lasso	0.01
21	1206.281892	1205.272222	Lasso	0.1
22	1206.403372	1207.533286	Lasso	1
23	1215.137979	1238.590955	Lasso	10
24	1362.777782	1459.776195	Lasso	100

	mean_train_score	mean_validation_score	model_name	param_model__alpha
25	2049.825908	1984.620077	Lasso	1000
26	3957.162324	3544.036398	Lasso	10000
27	3957.162324	3544.036398	Lasso	100000

	mean_train_score	mean_validation_score	model_name	param_model__alpha
0	1206.280597	1205.045051	Ridge	1e-08
1	1206.280597	1205.045051	Ridge	1e-07
2	1206.280597	1205.045051	Ridge	1e-06
3	1206.280597	1205.045055	Ridge	1e-05
4	1206.280597	1205.045088	Ridge	0.0001
5	1206.280597	1205.045421	Ridge	0.001
6	1206.280597	1205.048757	Ridge	0.01
7	1206.280619	1205.082125	Ridge	0.1
8	1206.282816	1205.417296	Ridge	1
9	1206.492542	1208.905905	Ridge	10
10	1220.380991	1249.715537	Ridge	100
11	1388.872280	1472.734228	Ridge	1000
12	1597.964835	1620.737422	Ridge	10000
13	2266.684099	2074.013462	Ridge	100000

Gas Dataset Unscaled Feature results¶

```
Linear Regression (OLS) Train Avg RMSE accross 10-fold cv:  8.030473106723004
Linear Regression (OLS) Validation Avg RMSE accross 10-fold cv:  8.881846484410753
```

Check RMSE results for each of the 10-fold:

```
Train RMSEs: [8.20232426 8.0507639  8.1187614  8.17650722 7.74108808 7.83083707
 7.79695147 8.26129408 8.19835912 7.92784446]
Validation RMSEs:  [ 7.43536346  8.57794857  7.89249311  7.21361859 11.91720141 10.28186932
 11.55608868  6.192516    7.87080299  9.88056272]
```

	mean_train_score	mean_validation_score	model_name	param_model__alpha
14	8.030497	8.890395	Lasso	1e-08
15	8.030497	8.890395	Lasso	1e-07
16	8.030497	8.890389	Lasso	1e-06
17	8.030497	8.890334	Lasso	1e-05
18	8.030495	8.889781	Lasso	0.0001
19	8.030495	8.884250	Lasso	0.001
20	8.031513	8.841661	Lasso	0.01
21	8.036915	8.728267	Lasso	0.1
22	8.166387	8.714142	Lasso	1
23	9.562444	9.933988	Lasso	10
24	11.656838	11.747458	Lasso	100
25	11.656838	11.747458	Lasso	1000
26	11.656838	11.747458	Lasso	10000
27	11.656838	11.747458	Lasso	100000

	mean_train_score	mean_validation_score	model_name	param_model__alpha
0	8.030473	8.881846	Ridge	1e-08

	mean_train_score	mean_validation_score	model_name	param_model__alpha
1	8.030473	8.881846	Ridge	1e-07
2	8.030473	8.881846	Ridge	1e-06
3	8.030473	8.881846	Ridge	1e-05
4	8.030473	8.881846	Ridge	0.0001
5	8.030473	8.881846	Ridge	0.001
6	8.030473	8.881843	Ridge	0.01
7	8.030473	8.881816	Ridge	0.1
8	8.030473	8.881545	Ridge	1
9	8.030482	8.878971	Ridge	10
10	8.030814	8.861215	Ridge	100
11	8.032528	8.786004	Ridge	1000
12	8.056832	8.580556	Ridge	10000
13	8.412807	8.791324	Ridge	100000

Does feature scaling play any role (in the cases with and without regularization)?

For diamonds data, Ridge Regression on scaled data has lower average RMSE values for the alpha > 10 compared the same lambda values applied on unscaled data on Ridge. For Lasso, we see that for larger alpha values, we see the same trend, scaled data has either lower or very similar value compared the unscaled data. The trends are same both in train and validation sets. This experiment shows that scaling our data is a good technique, which helps us getting better avg. RMSE scores especially against higher penalty regularization parameter values (seen from above tables). It is also important to note that the best/optimal avg. RMSE values found for scaled and unscaled data are almost same for optimal Ridge and optimal Lasso alpha values lying in the 1205-1206 range. For the unscaled data, the optimal value for regularization parameters are smaller, for Ridge optimal alpha is 1 and for Lasso alpha is 0.1.

From the experiments on diamonds data above, scaling data doesn't seem to play a role on the case without regularization. For the regularization on the other hand, scaling improves the avg. RMSE scores across different lambda values. This means for unscaled data the choice of regularization parameter is more crucial since the RMSE results can explode and grow very quickly when the data is unscaled and an unoptimal lambda parameter is chosen.

`Diamonds Data Scaled Best Results`

	mean_train_score	mean_validation_score	model_name	param_model__alpha
0	1206.280597	1205.045051	Ridge	1e-08
1	1206.280597	1205.045051	Ridge	1e-07
14	1206.280597	1205.045051	Lasso	1e-08

`Diamons Data Unscaled Best Results`

	mean_train_score	mean_validation_score	model_name	param_model__alpha
0	1206.280597	1205.045051	Ridge	1e-08
14	1206.280597	1205.045051	Lasso	1e-08
1	1206.280597	1205.045051	Ridge	1e-07

For the gas data, the validation scores for scaled and unscaled data are again very similar. Unscaled results are slightly better than the scaled ones as it can be seen below. But it doesn't play a big role.

`Gas Data Scaled Best Results`

	mean_train_score	mean_validation_score	model_name	param_model___alpha
10	8.094992	8.761643	Ridge	100
20	8.036236	8.802989	Lasso	0.01
9	8.032520	8.839102	Ridge	10

Gas Data Unscaled Best Results

	mean_train_score	mean_validation_score	model_name	param_model___alpha
12	8.056832	8.580556	Ridge	10000
22	8.166387	8.714142	Lasso	1
21	8.036915	8.728267	Lasso	0.1

Question 11¶ Some linear regression packages return p-values for different features. What is the meaning of them and how can you infer the most significant features?

P-values for each feature tests a null hypothesis that the coefficient has no effect (equal to 0). A typical threshold for p-value is 0.05 (corresponds to 95% confidence interval. A p-value lower than threshold (< 0.05) means we should reject the null hypothesis and the feature is important for the prediction of the target variable. A p-value higher than threshold, means that the feature is statistically not significant to predict the target variable, the change in target variable is not associated with the feature. The p-values can be used to find most meaningful feature and may be used to decide which features to keep in the model and which ones we can remove.

scipy.stats.linregress and statsmodels.regression.linear_model.OLS are two library models that returns p-values for each feature where we can infer most significant features using them. I experimented with statsmodels.regression.linear_model.OLS on diamonds dataset to find important features for scaled and unscaled dataset:

Diamonds Data¶ Diamonds Data Scaled (statsmodels.regression.linear_model.OLS) Results

```
OLS params:
 carat      5092.655074
depth      -114.319163
table       -59.802654
x          -984.480857
y            49.963989
z           -20.679798
cut         134.816544
color       548.941110
clarity     826.612634
dtype: float64
```

These are the coefficients assigned to each feature. The highest coefficient is given to carat meaning carat has the highest positive impact on the diamond price. Whereas x has the largest negative coefficient which means x also has a high impact, this time inverse relation, on the diamond price: lower the x the higher price we can assign.

OLS Summary for Scaled Diamonds Data:

```
                                 OLS Regression Results
================================================================================
Dep. Variable:                  price   R-squared (uncentered):                   0.460
Model:                            OLS   Adj. R-squared (uncentered):              0.460
Method:                 Least Squares   F-statistic:                              5100.
Date:                Fri, 18 Mar 2022   Prob (F-statistic):                        0.00
Time:                        17:04:58   Log-Likelihood:                     -5.2549e+05
No. Observations:               53940   AIC:                                  1.051e+06
Df Residuals:                   53931   BIC:                                  1.051e+06
```

```
Df Model:                           9
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
carat       5092.6551     83.184     61.222      0.000    4929.614    5255.696
depth       -114.3192     23.251     -4.917      0.000    -159.891     -68.748
table        -59.8027     22.299     -2.682      0.007    -103.510     -16.096
x           -984.4809    133.775     -7.359      0.000   -1246.681    -722.281
y             49.9640     80.234      0.623      0.533    -107.295     207.223
z            -20.6798     86.048     -0.240      0.810    -189.334     147.975
cut          134.8165     21.602      6.241      0.000      92.475     177.158
color        548.9411     18.771     29.244      0.000     512.150     585.733
clarity      826.6126     19.645     42.077      0.000     788.108     865.117
==============================================================================
Omnibus:                    12438.546   Durbin-Watson:                   0.100
Prob(Omnibus):                  0.000   Jarque-Bera (JB):           659440.386
Skew:                          -0.141   Prob(JB):                         0.00
Kurtosis:                      20.127   Cond. No.                         18.0
==============================================================================

Notes:
[1] R² is computed without centering (uncentered) since the model does not contain a constant.
[2] Standard Errors assume that the covariance matrix of the errors is correctly specified.
```

By looking the p-values in the above table (P >|t|), we can see that carat, depth, x, cut, color, clarity values have p-values lower than 0.05 and are the most important features. The R^2 and adjusted R^2 values for the model is 0.46 which is pretty low.

Diamonds Data Unscaled (statsmodels.regression.linear_model.OLS) Results

When we used unscaled data to apply OLS the R^2 and adjusted R^2 results are very high, 0.953 with all the diamonds features. Also all the features have p-values lower than 0.05. This means that with unscaled features OLS can fit a good linear model where all features are important for the prediction.

```
OLS params:
 carat      10587.758498
depth         -41.693458
table          -8.901253
x            -770.488938
y              60.599056
z            -121.711175
cut           144.798827
color         324.416568
clarity       506.522714
dtype: float64
                                OLS Regression Results
==============================================================================
Dep. Variable:                  price   R-squared (uncentered):                   0.953
Model:                            OLS   Adj. R-squared (uncentered):              0.953
Method:                 Least Squares   F-statistic:                          1.210e+05
Date:                Fri, 18 Mar 2022   Prob (F-statistic):                        0.00
Time:                        17:04:58   Log-Likelihood:                     -4.5975e+05
No. Observations:               53940   AIC:                                  9.195e+05
Df Residuals:                   53931   BIC:                                  9.196e+05
Df Model:                           9
Covariance Type:            nonrobust
```

	coef	std err	t	P>|t|	[0.025	0.975]
carat	1.059e+04	48.719	217.323	0.000	1.05e+04	1.07e+04
depth	-41.6935	2.025	-20.587	0.000	-45.663	-37.724
table	-8.9013	2.132	-4.175	0.000	-13.080	-4.722
x	-770.4889	33.062	-23.304	0.000	-835.291	-705.687
y	60.5991	20.676	2.931	0.003	20.074	101.124
z	-121.7112	34.465	-3.531	0.000	-189.263	-54.159
cut	144.7988	5.016	28.865	0.000	134.967	154.631
color	324.4166	3.256	99.640	0.000	318.035	330.798
clarity	506.5227	3.485	145.344	0.000	499.692	513.353

Omnibus:	12161.268	Durbin-Watson:	1.132
Prob(Omnibus):	0.000	Jarque-Bera (JB):	614443.485
Skew:	-0.100	Prob(JB):	0.00
Kurtosis:	19.533	Cond. No.	860.

Notes:
[1] R^2 is computed without centering (uncentered) since the model does not contain a constant.
[2] Standard Errors assume that the covariance matrix of the errors is correctly specified.

Gas Data¶ Gas Data Scaled (statsmodels.regression.linear_model.OLS) Results

```
OLS params:
 AT     -12.906630
AP      -1.493527
AH      -3.317940
AFDP     0.289539
GTEP     0.568738
TIT     19.725546
TAT     -7.755266
TEY    -28.881104
CDP      2.063782
year    -1.339015
dtype: float64
```

These are the coefficients assigned to each feature. The highest absolute coefficient is given to TEY which has the highest negative impact on the NOX. Whereas TIT has the largest positive coefficient, and second biggest absolute coefficient, which means TIT also has a high impact, this time it has positive relation, on the NOX prediction.

OLS Summary for Scaled Gas Data:

OLS Regression Results

Dep. Variable:	y	R-squared (uncentered):	0.016
Model:	OLS	Adj. R-squared (uncentered):	0.016
Method:	Least Squares	F-statistic:	60.38
Date:	Fri, 18 Mar 2022	Prob (F-statistic):	2.89e-122
Time:	17:04:58	Log-Likelihood:	-2.0590e+05
No. Observations:	36733	AIC:	4.118e+05
Df Residuals:	36723	BIC:	4.119e+05
Df Model:	10		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
AT	-12.9066	0.992	-13.008	0.000	-14.851	-10.962
AP	-1.4935	0.422	-3.537	0.000	-2.321	-0.666
AH	-3.3179	0.430	-7.713	0.000	-4.161	-2.475
AFDP	0.2895	0.554	0.523	0.601	-0.796	1.375
GTEP	0.5687	1.937	0.294	0.769	-3.228	4.365
TIT	19.7255	3.137	6.289	0.000	13.577	25.874
TAT	-7.7553	1.662	-4.666	0.000	-11.013	-4.498
TEY	-28.8811	5.533	-5.219	0.000	-39.727	-18.036
CDP	2.0638	5.721	0.361	0.718	-9.150	13.278
year	-1.3390	0.619	-2.163	0.031	-2.552	-0.126

Omnibus:	7598.785	Durbin-Watson:	0.006
Prob(Omnibus):	0.000	Jarque-Bera (JB):	26667.084
Skew:	1.024	Prob(JB):	0.00
Kurtosis:	6.637	Cond. No.	47.7

Notes:
[1] R^2 is computed without centering (uncentered) since the model does not contain a constant.
[2] Standard Errors assume that the covariance matrix of the errors is correctly specified.

By looking the p-values in the above table (P >|t|), we can see that all features except AFDP, GTEP and CDP are important features for NOX prediction. They añn have p-values lower than 0.05 and are the most important features. The R^2 and adjusted R^2 values for the model is 0.016 which is pretty low.

Gas Data Unscaled (statsmodels.regression.linear_model.OLS) Results

When we used unscaled data to apply OLS the R^2 and adjusted R^2 results are very high, 0.985 with all the diamonds features. Also all the features have p-values lower than 0.05 except GTEP. This means that with unscaled features OLS can fit a good linear model where all features are important (except GTEP) for the prediction.

```
OLS params:
 AT     -1.717197
AP     -0.240365
AH     -0.230917
AFDP    0.379716
GTEP    0.102995
TIT     1.105622
TAT    -1.133994
TEY    -1.793823
CDP     1.505362
year   -0.959536
dtype: float64
```

OLS Regression Results

Dep. Variable:	y	R-squared (uncentered):	0.985
Model:	OLS	Adj. R-squared (uncentered):	0.985
Method:	Least Squares	F-statistic:	2.441e+05
Date:	Fri, 18 Mar 2022	Prob (F-statistic):	0.00
Time:	17:04:58	Log-Likelihood:	-1.2885e+05
No. Observations:	36733	AIC:	2.577e+05
Df Residuals:	36723	BIC:	2.578e+05
Df Model:	10		
Covariance Type:	nonrobust		

```
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
AT            -1.7172      0.015   -115.065      0.000      -1.746      -1.688
AP            -0.2404      0.007    -34.351      0.000      -0.254      -0.227
AH            -0.2309      0.004    -64.187      0.000      -0.238      -0.224
AFDP           0.3797      0.088      4.324      0.000       0.208       0.552
GTEP           0.1030      0.055      1.874      0.061      -0.005       0.211
TIT            1.1056      0.020     54.190      0.000       1.066       1.146
TAT           -1.1340      0.030    -38.034      0.000      -1.192      -1.076
TEY           -1.7938      0.037    -48.895      0.000      -1.866      -1.722
CDP            1.5054      0.624      2.414      0.016       0.283       2.728
year          -0.9595      0.053    -18.059      0.000      -1.064      -0.855
==============================================================================
Omnibus:                     7518.162   Durbin-Watson:                   0.376
Prob(Omnibus):                  0.000   Jarque-Bera (JB):            26205.615
Skew:                           1.016   Prob(JB):                         0.00
Kurtosis:                       6.605   Cond. No.                     2.35e+04
==============================================================================

Notes:
[1] R² is computed without centering (uncentered) since the model does not contain a constant.
[2] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[3] The condition number is large, 2.35e+04. This might indicate that there are
strong multicollinearity or other numerical problems.
```

Polynomial Regression¶

Question 12¶ For each polynomial regression model I applied 10-fold cross-validation and measured average RMSE errors for training and validation sets.

For this experiment, I created a pipeline to extract polynomial features with PolynomialFeatures() in sklearn and applied Linear Regression on the compound features (I used Lasso and Ridge Regression models for regularization). Grid search over 10-fold cv is applied to tune degrees for polynomial features and alpha for the regularization. Degree range is [2, 3, 4] and alpha range is $[10^{-4}, 10^3]$.

Diamond Data¶

Polynomial Regression Experiment Results¶ Top 10 hyperparameter combination results based on based mean validation score. From the below table, the best combinations 10-fold avg train RMSE: 921.57 avg validation RMSE: 882.22 which are way better than the Linear Regression results which were around 1205-1206 RMSE results for both metrics.

	mean_train_score	mean_validation_score
42	921.572213	882.226435
39	782.419320	1129.772967
36	763.272005	1615.358097
21	795.955216	1667.221668
24	757.769711	1777.435298
27	757.769829	1777.748836
30	757.771652	1780.882997
45	1841.870944	1790.712541
33	757.857294	1809.238839
46	2023.787034	1902.183814

	param_model	param_model__alpha	param_poly_features__degree
42	Lasso(alpha=100, random_state=142)	100.0000	2
39	Lasso(alpha=100, random_state=142)	10.0000	2
36	Lasso(alpha=100, random_state=142)	1.0000	2
21	Ridge(random_state=142)	1000.0000	2
24	Lasso(alpha=100, random_state=142)	0.0001	2
27	Lasso(alpha=100, random_state=142)	0.0010	2
30	Lasso(alpha=100, random_state=142)	0.0100	2
45	Lasso(alpha=100, random_state=142)	1000.0000	2
33	Lasso(alpha=100, random_state=142)	0.1000	2
46	Lasso(alpha=100, random_state=142)	1000.0000	3

Best model found from GridSearch:

The best model is Lasso with alpha 100 and PolynomialFeatures set with degree 2.

```
Best Model: Pipeline(steps=[('poly_features', PolynomialFeatures()),
                ('model', Lasso(alpha=100, random_state=142))])
Best Model Params: {'model': Lasso(alpha=100, random_state=142), 'model__alpha': 100, 'poly_features': Po
```

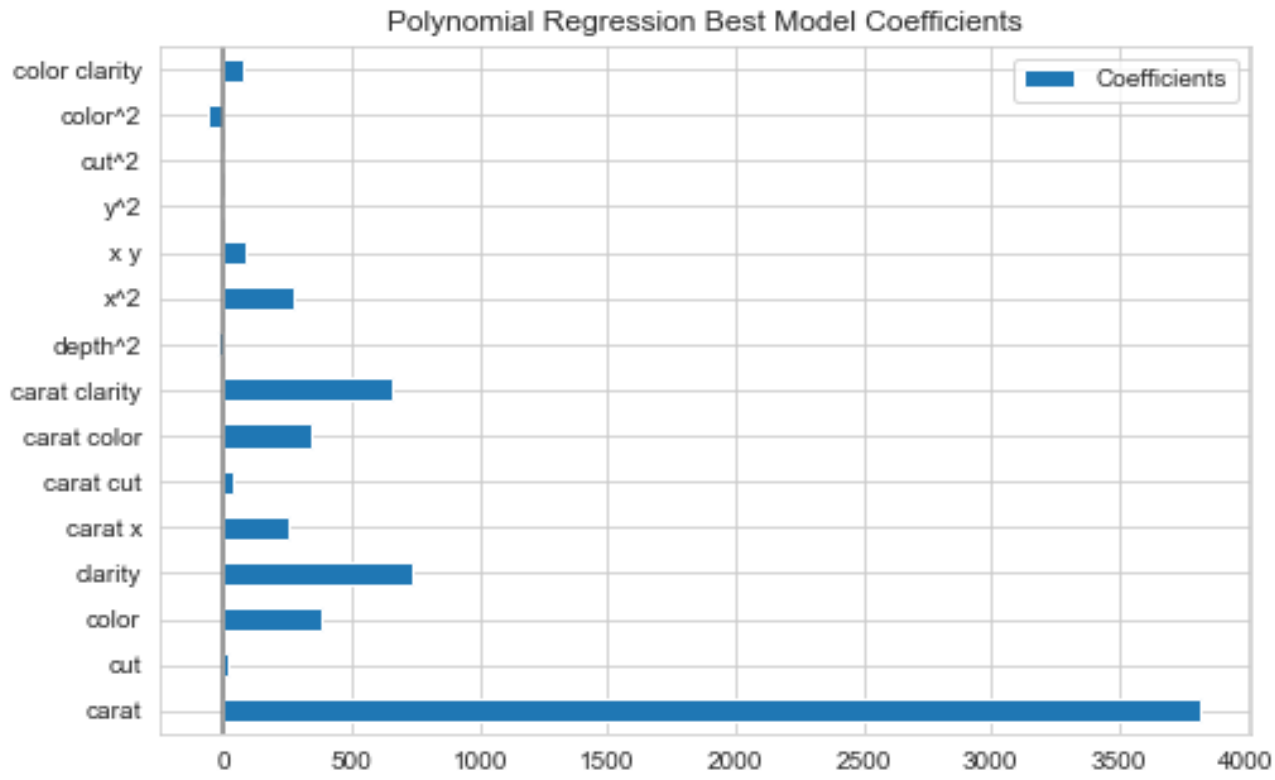
Coefficient of the best model and Most Salient Features:¶

Since the selected best model is with Lasso Regularization, we can see that most of the features have 0 coefficients.

	Coefficients
1	0.000000
carat	3819.165766
depth	-0.000000
table	-0.000000
x	0.000000
y	0.000000
z	0.000000
cut	19.424984
color	379.512189
clarity	736.942242
carat^2	0.000000
carat depth	-0.000000
carat table	-0.000000
carat x	255.502250
carat y	0.000000
carat z	0.000000
carat cut	36.487862
carat color	340.220949
carat clarity	658.568956
depth^2	-25.612413
depth table	0.000000
depth x	-0.000000
depth y	-0.000000
depth z	-0.000000
depth cut	0.000000
depth color	0.000000
depth clarity	0.000000
table^2	-0.000000
table x	-0.000000
table y	-0.000000
table z	-0.000000

	Coefficients
table cut	0.000000
table color	0.000000
table clarity	0.000000
x^2	272.697483
x y	83.223653
x z	0.000000
x cut	0.000000
x color	0.000000
x clarity	0.000000
y^2	-0.961110
y z	0.000000
y cut	0.000000
y color	0.000000
y clarity	0.000000
z^2	0.000000
z cut	0.000000
z color	0.000000
z clarity	0.000000
cut^2	-5.943162
cut color	-0.000000
cut clarity	-0.000000
color^2	-58.635986
color clarity	79.471302
clarity^2	-0.000000

Lasso Regression assigns 0 values to coefficients as seen above, to understand salient features better, I plotted the non-zero coefficients only.



Non-zero Coefficients ordered from highest to lowest absolute coefficient values:

	Coefficients	coef_absolute
carat	3819.165766	3819.165766
clarity	736.942242	736.942242
carat clarity	658.568956	658.568956
color	379.512189	379.512189
carat color	340.220949	340.220949
x^2	272.697483	272.697483
carat x	255.502250	255.502250
x y	83.223653	83.223653
color clarity	79.471302	79.471302
color^2	-58.635986	58.635986
carat cut	36.487862	36.487862
depth^2	-25.612413	25.612413
cut	19.424984	19.424984
cut^2	-5.943162	5.943162
y^2	-0.961110	0.961110

From the coefficients, we can see that Lasso assigns 0 to many of the features and only small number of them have non-zero coefficients. Within the most salient features, the most important one is carat with the biggest coefficient 3819.17. This means the increase in price is positively correlated with the carat coefficient. Clarity is the second most important feature followed by compound carat clarit feature, color, carat color, x^2 and carat x. Color^2, depth^2, cut^2 and y^2 also has negative correlation with diamond price. Decreasing either one of them leads to increase the price. The coefficient values also matter, the bigger the absolute coefficient value means the bigger impact it has on the target variable (pricing the diamond), the more important the feature in terms of predicting the target variable.

Gas Data¶

Polynomial Regression Experiment Results¶ Top 10 hyperparameter combination results based on based mean validation score. From the below table, the best combinations 10-fold avg train RMSE: 5.72 avg validation RMSE: 7.15 which are way better than the Linear Regression results which were around 8.1-8.76 RMSE results for both metrics.

	mean_train_score	mean_validation_score
35	5.719778	7.155798
34	5.827271	7.171435
22	5.508430	7.226445
23	5.273528	7.369680
33	6.246073	7.436532
18	5.682998	7.459753
21	6.055840	7.464865
19	5.119546	7.514624
20	4.895412	7.588859
31	5.156848	7.655462

	param_model	param_model__alpha	param_poly_features__degree
35	Lasso(alpha=0.1, random_state=142)	0.10	4
34	Lasso(alpha=0.1, random_state=142)	0.10	3
22	Ridge(random_state=142)	1000.00	3
23	Ridge(random_state=142)	1000.00	4

	param_model	param_model__alpha	param_poly_features__degree
33	Lasso(alpha=0.1, random_state=142)	0.10	2
18	Ridge(random_state=142)	100.00	2
21	Ridge(random_state=142)	1000.00	2
19	Ridge(random_state=142)	100.00	3
20	Ridge(random_state=142)	100.00	4
31	Lasso(alpha=0.1, random_state=142)	0.01	3

Best model found from GridSearch:

The best model is Lasso with alpha 0.1 and PolynomialFeatures set with degree 4. alpha 0.1 with degree 3 is also the second best result where the validation scores of both options are very close to each other.

```
Best Model: Pipeline(steps=[('poly_features',
                 PolynomialFeatures(degree=4, interaction_only='False')),
                ('model', Lasso(alpha=0.1, random_state=142))])
Best Model Params: {'model': Lasso(alpha=0.1, random_state=142), 'model__alpha': 0.1, 'poly_features': Po
```

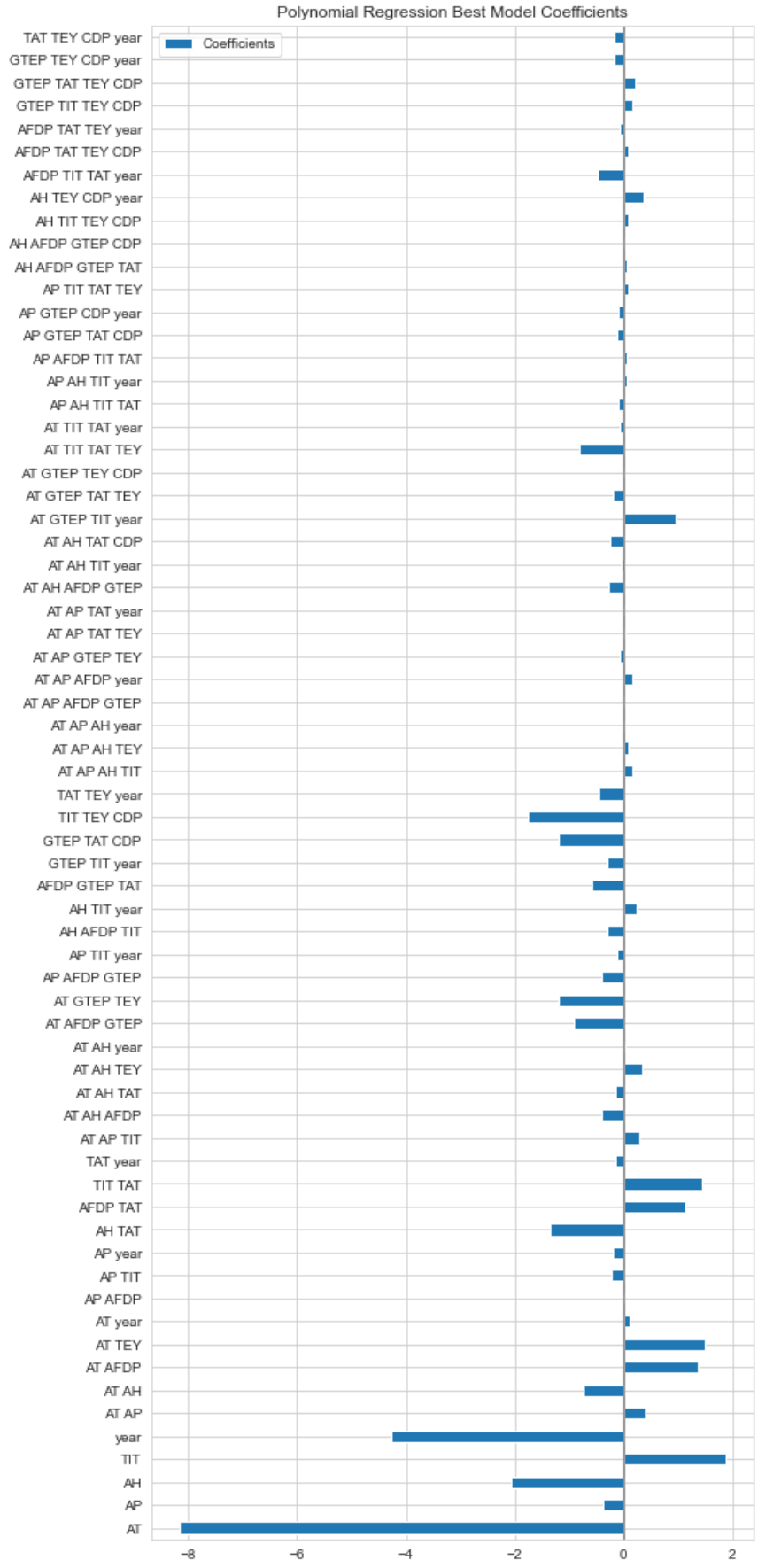
Coefficient of the best model and Most Salient Features:¶

Since the selected best model is with Lasso Regularization, we can see that most of the features have 0 coefficients.

	Coefficients
1	0.000000
AT	-8.175731
AP	-0.380350
AH	-2.069612
AFDP	0.000000
...	...
TIT TAT TEY CDP	0.000000
TIT TAT TEY year	-0.000000
TIT TAT CDP year	-0.000000
TIT TEY CDP year	0.000000
TAT TEY CDP year	-0.168597

386 rows × 1 columns

Lasso Regression assigns 0 values to coefficients as seen above, to understand salient features better, I plotted the non-zero coefficients only.


Polynomial Regression Best Model Coefficients
Coefficients
TAT TEY CDP year
GTEP TEY CDP year
GTEP TAT TEY CDP
GTEP TIT TEY CDP
AFDP TAT TEY year
AFDP TAT TEY CDP
AFDP TIT TAT year
AH TEY CDP year
AH TIT TEY CDP
AH AFDP GTEP CDP
AH AFDP GTEP TAT
AP TIT TAT TEY
AP GTEP CDP year
AP GTEP TAT CDP
AP AFDP TIT TAT
AP AH TIT year
AP AH TIT TAT
AT TIT TAT year
AT TIT TAT TEY
AT GTEP TEY CDP
AT GTEP TAT TEY
AT GTEP TIT year
AT AH TAT CDP
AT AH TIT year
AT AH AFDP GTEP
AT AP TAT year
AT AP TAT TEY
AT AP GTEP TEY
AT AP AFDP year
AT AP AFDP GTEP
AT AP AH year
AT AP AH TEY
AT AP AH TIT
TAT TEY year
TIT TEY CDP
GTEP TAT CDP
GTEP TIT year
AFDP GTEP TAT
AH TIT year
AH AFDP TIT
AP TIT year
AP AFDP GTEP
AT GTEP TEY
AT AFDP GTEP
AT AH year
AT AH TEY
AT AH TAT
AT AH AFDP
AT AP TIT
TAT year
TIT TAT
AFDP TAT
AH TAT
AP year
AP TIT
AP AFDP
AT year
AT TEY
AT AFDP
AT AH
AT AP
year
TIT
AH
AP
AT
−8
−6
−4
−2
0
2

Non-zero Coefficients ordered from highest to lowest absolute coefficient values:

	Coefficients	coef_absolute
AT	-8.175731	8.175731
year	-4.280409	4.280409
AH	-2.069612	2.069612
TIT	1.888307	1.888307
TIT TEY CDP	-1.751613	1.751613
...	...	...
AT AP AFDP GTEP	0.016500	0.016500
AT AP AH year	0.013680	0.013680
AT GTEP TEY CDP	0.008394	0.008394
AP AFDP	0.005628	0.005628
AT AP TAT year	0.002765	0.002765

66 rows × 2 columns

```
Total coefficient counts: 386
Zero coefficient counts: 320
```

From the coefficients, we can see that Lasso assigns 0 to many of the features and only small number of them have non-zero coefficients. 320 out of 386 possible coefficients assigned 0 values.

Within the most salient features, the most important one is AT with -8.17, showcasing negative correlation. This is followed by year and AH with smaller negative correlations. TIT has positive correlation. The first 4 highest correlation values given to non-compounda features and then the most of the remaining non-zero coefficients (the biggest absolute ones) seem to be given to the compound features.

Question 13¶ The experiment is done for degrees 2, 3, and 4. Given that degree=1 would be just linear results, and we want to see the results for polynomial features, I omitted degree=1 in the experiment. I also didn't include degrees higher than 4 because given the dataset, the amount of compound features created would be too much with degrees higher than 5. Another reason is that both dataset are relatively small 30-50k samples, very high degrees would more likely to cause overfit rather then help generalizing the model.

Diamonds Data¶ What degree of polynomial is best? From experiment results we can see that the best degree is 2. The first 9 of the top 10 combinations of our experiment shows that 2 is the best degree values anf Lasso is a better regularization for this dataset.

Best model parameters:

```
{'model': Lasso(alpha=100, random_state=142),
 'model__alpha': 100,
 'poly_features': PolynomialFeatures(),
 'poly_features__degree': 2}
```

Top 10 combinations from gridsearch results sorted with lowest RMSE validation scores:

	mean_train_score	mean_validation_score
42	921.572213	882.226435
39	782.419320	1129.772967
36	763.272005	1615.358097
21	795.955216	1667.221668
24	757.769711	1777.435298
27	757.769829	1777.748836
30	757.771652	1780.882997
45	1841.870944	1790.712541

	mean_train_score	mean_validation_score
33	757.857294	1809.238839
46	2023.787034	1902.183814

	param_model	param_model__alpha	param_poly_features__degree
42	Lasso(alpha=100, random_state=142)	100.0000	2
39	Lasso(alpha=100, random_state=142)	10.0000	2
36	Lasso(alpha=100, random_state=142)	1.0000	2
21	Ridge(random_state=142)	1000.0000	2
24	Lasso(alpha=100, random_state=142)	0.0001	2
27	Lasso(alpha=100, random_state=142)	0.0010	2
30	Lasso(alpha=100, random_state=142)	0.0100	2
45	Lasso(alpha=100, random_state=142)	1000.0000	2
33	Lasso(alpha=100, random_state=142)	0.1000	2
46	Lasso(alpha=100, random_state=142)	1000.0000	3

Gas Data¶ What degree of polynomial is best? From experiment results we can see that best degree chosen by gridsearch is 4. If we look at the top 10 combinations of best results from grid search in the below table, we can see that degree 3 is also a good choice, given that the mean validation score has only very little difference in the second significant digits. So, one can select degree 3 as the best parameter value as well.

Best model parameters:

```
{'model': Lasso(alpha=0.1, random_state=142),
 'model__alpha': 0.1,
 'poly_features': PolynomialFeatures(degree=4, interaction_only='False'),
 'poly_features__degree': 4}
```

Top 10 combinations from gridsearch results sorted with lowest RMSE validation scores:

	mean_train_score	mean_validation_score
35	5.719778	7.155798
34	5.827271	7.171435
22	5.508430	7.226445
23	5.273528	7.369680
33	6.246073	7.436532
18	5.682998	7.459753
21	6.055840	7.464865
19	5.119546	7.514624
20	4.895412	7.588859
31	5.156848	7.655462

	param_model	param_model__alpha	param_poly_features__degree
35	Lasso(alpha=0.1, random_state=142)	0.10	4
34	Lasso(alpha=0.1, random_state=142)	0.10	3
22	Ridge(random_state=142)	1000.00	3
23	Ridge(random_state=142)	1000.00	4
33	Lasso(alpha=0.1, random_state=142)	0.10	2
18	Ridge(random_state=142)	100.00	2
21	Ridge(random_state=142)	1000.00	2
19	Ridge(random_state=142)	100.00	3

	param_model	param_model___alpha	param_poly_features___degree
20	Ridge(random_state=142)	100.00	4
31	Lasso(alpha=0.1, random_state=142)	0.01	3

What does a very high-order polynomial imply about the fit on the training data? How do you choose this parameter?¶ With a higher-order polynomial, we can create more complex models that covers larger hypothesis spaces. With increasing model complexity, it means we can fit the data better, however, if the degree of polynomial is higher than the optimal degree that data lies, then model may overfit the data. In case of overfitting, the fitted curve will start to memorize the training data rather then learn to generalize and find patterns. When model overfits, the training error would be low and test/validation errors would be high since it perform badly on the unseen data. We can see this from the experiment results in the above table as well. For higher degrees we see that training errors (avg RMSEs) are very low but the avg. validation scores are very high compared to the training score.

On the other hand, if the degree order is too small, the model cannot learn enough from the data and model can underfit resulting in high training and validation errors. In order to avoid overfitting or underfitting, and have a model that generalizes well, we need to choose a degree that matches with the true structure of the data. This happens when the the model has a good training score and the gap between the training and validation scores are lowest. One way to choosee optimal degree parameter, is to run cross validation with different degree values and compare the avg. training and validation results. Then choose a degree where model has low trainining RMSE and low test RMSE. The optimal point is generally when the model has low training RMSE and the gap beetween train and validation score is also lowest.

Question 14¶ For the diamond dataset it might make sense to craft new features such as z = x1 × x2, etc. Explain why this might make sense and check if doing so will boost accuracy.

Depending on some dataset a compound feature might make sense on predicting the target variable. For example, for the diamonds dataset the diamond price might be correlated with the diamond volume, and diamond volume can be represented by xyz. Using compounding features (combination of different features) might be more helpful in predicting target variable rather than individual columns by itself. There could be cases where individual feature might not contribute as much as a compound feature to the prediction.

I tried 2 approaches to craft new features such as z= x1 x x2.

Approach 1: Use interaction_only=True in Polynomial Features and run only for the combinations of different features. In this approach I created all possible interacted features and checked whether any of them are important features.

Approach 2: Create craft features by hand for the combinations that makes sense and could be helpful for prediction of target variable:

For diamonds dataset new hand crafted features:

```
- volume of diamond: x * y * z
- table * depth
- density of diamond: carat / (x*y*z)
```

In the results below saw that approach 1 helped us improve the accuracy. Approach 2 by adding only few selected hand-crafted features, instead of the regular features worsen the accuracy.

Approach 1: Polynomial Feature on Interactions only¶

I applied a randomized search with 12 samples (n_iter) with the same parameter range on q12, but this time I only used the compound features to feed in the linear model. I did 10-fold cv and Lasso, Ridge Regularizations.

Polynomial Feature Interaction Only features experiment results on top 10 combination:

	mean_train_score	mean_validation_score
39	805.512807	813.970583
40	679.765887	825.384492
42	927.309710	873.199024
21	811.508988	874.116912
41	647.222578	883.514943
43	874.565986	900.154033
44	859.818400	937.391952
22	683.784705	1067.001516
23	647.120971	1101.171040
18	789.987240	1108.538274

	param_model	param_model__alpha	param_poly_features__degree
39	Lasso(alpha=10, random_state=142)	10.0	2
40	Lasso(alpha=10, random_state=142)	10.0	3
42	Lasso(alpha=10, random_state=142)	100.0	2
21	Ridge(random_state=142)	1000.0	2
41	Lasso(alpha=10, random_state=142)	10.0	4
43	Lasso(alpha=10, random_state=142)	100.0	3
44	Lasso(alpha=10, random_state=142)	100.0	4
22	Ridge(random_state=142)	1000.0	3
23	Ridge(random_state=142)	1000.0	4
18	Ridge(random_state=142)	100.0	2

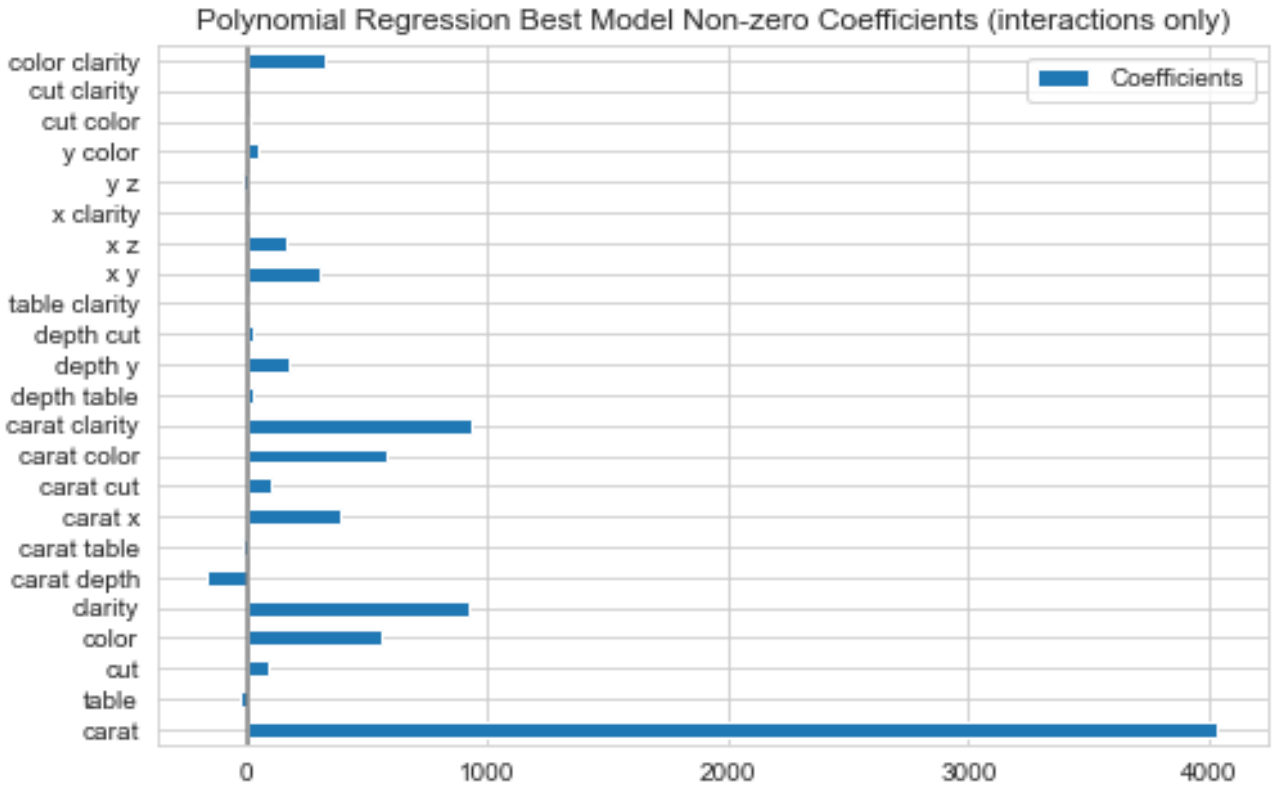
The avg 10-fold CV Train RMSE:805.51 and Validation RMSE=813.97 with interaction only features are much better then the Q12-13 results. This results the performances of the models.

```
Best model:
 Pipeline(steps=[('poly_features', PolynomialFeatures(interaction_only=True)),
                 ('model', Lasso(alpha=10, random_state=142))])
Best model parameters:
 {'model': Lasso(alpha=10, random_state=142), 'model__alpha': 10, 'poly_features': PolynomialFeatures(int
```

The coefficients of the best found model with polynomial interactions only features (non-zero coeffs only):

	Coefficients
carat	4034.341806
table	-32.671368
cut	93.303139
color	557.804933
clarity	918.785316
carat depth	-163.438333
carat table	-19.371410
carat x	384.641881
carat cut	101.794631
carat color	582.467856
carat clarity	937.881468
depth table	23.574794
depth y	173.265004
depth cut	29.052115
table clarity	-6.953471
x y	298.472237

	Coefficients
x z	164.424270
x clarity	-3.652150
y z	-22.220218
y color	44.728856
cut color	12.634970
cut clarity	4.550465
color clarity	321.181135



Non-zero Coefficients ordered from highest to lowest absolute coefficient values:

	Coefficients	coef_absolute
carat	4034.341806	4034.341806
carat clarity	937.881468	937.881468
clarity	918.785316	918.785316
carat color	582.467856	582.467856
color	557.804933	557.804933
carat x	384.641881	384.641881
color clarity	321.181135	321.181135
x y	298.472237	298.472237
depth y	173.265004	173.265004
x z	164.424270	164.424270
carat depth	-163.438333	163.438333
carat cut	101.794631	101.794631
cut	93.303139	93.303139
y color	44.728856	44.728856
table	-32.671368	32.671368
depth cut	29.052115	29.052115

	Coefficients	coef_absolute
depth table	23.574794	23.574794
y z	-22.220218	22.220218
carat table	-19.371410	19.371410
cut color	12.634970	12.634970
table clarity	-6.953471	6.953471
cut clarity	4.550465	4.550465
x clarity	-3.652150	3.652150

We can see that the most important features are very similar as in Q12-Q13, however the model is only using the compounds of different features now.

Approach 2.1: Hand-crafted features - Option 1- create compound features from raw data and then scale¶

In this approach I only hand crafted new feature set. The process is I created the features from raw data and then scaled the. Then I checked the results on best found model and its parameters in Question 12 to see if the model performance is improved. Lasso Regression with 100 alpha is used:

	carat	cut	color	clarity	depth	table	price	x	y
1	-1.198168	0.981473	0.937163	-1.245215	-0.174092	-1.099672	330	-1.587837	-1.536196
2	-1.240361	0.085889	0.937163	-0.638095	-1.360738	1.585529	327	-1.641325	-1.658774
3	-1.198168	-1.705279	0.937163	0.576145	-3.385019	3.375663	328	-1.498691	-1.457395
4	-1.071587	0.085889	-1.414272	-0.030975	0.454133	0.242928	337	-1.364971	-1.317305
5	-1.029394	-1.705279	-2.002131	-1.245215	1.082358	0.242928	338	-1.240167	-1.212238

	z	x*y*z	table*depth	density
1	-1.571129	-1.171294	-1.188533	-0.489328
2	-1.741175	-1.218533	0.728291	-0.183160
3	-1.741175	-1.172894	1.094605	-0.396553
4	-1.287720	-1.062372	0.521653	0.382562
5	-1.117674	-0.996008	0.898804	-0.721797

	count	mean	std	min	25%	50%	75%	max
carat	53940.0	2.444878e-16	1.000009	-1.261458	-0.839523	-0.206621	0.510668	8.886075
cut	53940.0	1.454281e-16	1.000009	-2.600864	-0.809695	0.085889	0.981473	0.981473
color	53940.0	1.338360e-16	1.000009	-2.002131	-0.826413	-0.238555	0.937163	1.525021
clarity	53940.0	-8.114467e-17	1.000009	-1.852335	-0.638095	-0.030975	0.576145	2.397505
depth	53940.0	-3.996902e-15	1.000009	-13.087603	-0.523105	0.035317	0.523936	12.041392
table	53940.0	9.695207e-17	1.000009	-6.470073	-0.652139	-0.204605	0.690462	16.801666
price	53940.0	3.934802e+03	3989.442321	327.000000	952.000000	2403.000000	5327.250000	18823.000000
x	53940.0	2.782103e-16	1.000009	-5.109120	-0.910325	-0.027776	0.721054	4.465203
y	53940.0	-8.430615e-17	1.000009	-5.020931	-0.888280	-0.021474	0.705242	46.549648
z	53940.0	-2.002271e-16	1.000009	-5.014556	-0.890946	-0.012376	0.710318	40.047576
x*y*z	53940.0	3.161480e-16	1.000009	-1.659533	-0.827055	-0.192229	0.523909	47.425019
table*depth	53940.0	-1.559664e-16	1.000009	-8.850773	-0.711675	-0.096817	0.605464	16.039766
density	53940.0	-5.342639e-15	1.000009	-28.717414	-0.360190	-0.037717	0.305142	77.465884

```
Results for features:  ['carat', 'x', 'y', 'z', 'table*depth', 'cut', 'color', 'clarity']
Lasso with new features Train Avg RMSE accross 10-fold cv:  1251.257425103957
```

```
Lasso with new features Validation Avg RMSE accross 10-fold cv:  1244.9244651005379


Results for features:  ['carat', 'x*y*z', 'table', 'depth', 'cut', 'color', 'clarity']
Lasso with new features Train Avg RMSE accross 10-fold cv:  1250.9522429042684
Lasso with new features Validation Avg RMSE accross 10-fold cv:  1244.883579227349


Results for features:  ['carat', 'x', 'y', 'z', 'table', 'depth', 'cut', 'color', 'clarity', 'density']
Lasso with new features Train Avg RMSE accross 10-fold cv:  1251.2749901090597
Lasso with new features Validation Avg RMSE accross 10-fold cv:  1244.8990105660093


Results for features:  ['carat', 'x', 'y', 'z', 'table', 'depth', 'cut', 'color', 'clarity', 'x*y*z', 'ta
Lasso with new features Train Avg RMSE accross 10-fold cv:  1251.0241173671452
Lasso with new features Validation Avg RMSE accross 10-fold cv:  1244.790262782626
```

By only using these 3 crafted features doesn't really help with the model performance, it rather detoriates.

Approach 2.2: Hand-crafted features - Option 2- create compound features from scaled data¶

In this approach I only hand crafted new feature set. The process is I created the features from scaled data directly. Then I checked the results on best found model and its parameters in Question 12 to see if the model performance is improved. Lasso Regression with 100 alpha is used:

	carat	cut	color	clarity	depth	table	price	x	y
1	-1.198168	0.981473	0.937163	-1.245215	-0.174092	-1.099672	330	-1.587837	-1.536196
2	-1.240361	0.085889	0.937163	-0.638095	-1.360738	1.585529	327	-1.641325	-1.658774
3	-1.198168	-1.705279	0.937163	0.576145	-3.385019	3.375663	328	-1.498691	-1.457395
4	-1.071587	0.085889	-1.414272	-0.030975	0.454133	0.242928	337	-1.364971	-1.317305
5	-1.029394	-1.705279	-2.002131	-1.245215	1.082358	0.242928	338	-1.240167	-1.212238

	z	x*y*z	table*depth
1	-1.571129	-1.011985	0.331656
2	-1.741175	-1.231300	-1.267281
3	-1.741175	-1.004910	-7.576892
4	-1.287720	-0.645658	0.276436
5	-1.117674	-0.492275	0.380321

	count	mean	std	min	25%	50%	75%	max
carat	53940.0	2.444878e-16	1.000009	-1.261458	-0.839523	-0.206621	0.510668	8.886075
cut	53940.0	1.454281e-16	1.000009	-2.600864	-0.809695	0.085889	0.981473	0.981473
color	53940.0	1.338360e-16	1.000009	-2.002131	-0.826413	-0.238555	0.937163	1.525021
clarity	53940.0	-8.114467e-17	1.000009	-1.852335	-0.638095	-0.030975	0.576145	2.397505
depth	53940.0	-3.996902e-15	1.000009	-13.087603	-0.523105	0.035317	0.523936	12.041392
table	53940.0	9.695207e-17	1.000009	-6.470073	-0.652139	-0.204605	0.690462	16.801666
price	53940.0	3.934802e+03	3989.442321	327.000000	952.000000	2403.000000	5327.250000	18823.000000
x	53940.0	2.782103e-16	1.000009	-5.109120	-0.910325	-0.027776	0.721054	4.465203
y	53940.0	-8.430615e-17	1.000009	-5.020931	-0.888280	-0.021474	0.705242	46.549648
z	53940.0	-2.002271e-16	1.000009	-5.014556	-0.890946	-0.012376	0.710318	40.047576
x*y*z	53940.0	4.215307e-18	1.000009	-31.151381	-0.253881	-0.086498	-0.007973	151.364031
table*depth	53940.0	-2.357938e-17	1.000009	-24.941493	-0.093218	0.154675	0.309607	57.216800

```
Results for features:  ['carat', 'x*y*z', 'table*depth', 'cut', 'color', 'clarity']
Lasso with new features Train Avg RMSE accross 10-fold cv:  1251.2751638566913
Lasso with new features Validation Avg RMSE accross 10-fold cv:  1244.8986831815869


Results for features:  ['carat', 'x', 'y', 'z', 'table*depth', 'cut', 'color', 'clarity']
Lasso with new features Train Avg RMSE accross 10-fold cv:  1251.2751638566913
Lasso with new features Validation Avg RMSE accross 10-fold cv:  1244.8986831815869


Results for features:  ['carat', 'x*y*z', 'table', 'depth', 'cut', 'color', 'clarity']
Lasso with new features Train Avg RMSE accross 10-fold cv:  1251.2749901090597
Lasso with new features Validation Avg RMSE accross 10-fold cv:  1244.8990105660093
```

The result of this approach is also similar to Approach 2.1. and it doesn't really help model improve.

Neural Network¶

Question 15¶ For question 15, I ran a MLP model with the default parameters in sklearn (MLPRegressor). The default MLP model has 1 hidden layer with 100 neurons and activation function as relu, solver as adam.

Diamond Data¶ From the below results we can see that Neural Network is performed much better.

To compare the best result for linear model avg Train and Validation RMSE were around 1205-1206 and for polynomial regression was avg Train RMSE: 921.572213and Avg. Validation RMSE was: 882.226435.

```
Neural Network Train Avg RMSE accross 10-fold cv:  777.6786147628094
Neural Network Validation Avg RMSE accross 10-fold cv:  747.3036406022716
```

```
Train RMSEs: [820.08474187 809.08852109 788.44200593 726.8316636  614.1550364
 728.2570057  813.74382945 823.98262994 818.55573404 833.64497961]
Validation RMSEs:  [ 643.18346529  618.63124327  823.05710113 1144.74096698 1702.28150625
 1161.93618763  166.91549116  234.49500132  531.01778216  446.77766084]
```

Gas Data¶ From the below results we can see that Neural Network is performed much better.

To compare the best result for linear model avg Train and Validation RMSE were around 8.1-8.76 and for polynomial regression was avg Train RMSE: 5.71 and Avg. Validation RMSE was: 7.15.

```
Neural Network Train Avg RMSE accross 10-fold cv:  4.403767325900161
Neural Network Validation Avg RMSE accross 10-fold cv:  7.151286634532899
```

```
Train RMSEs: [4.47615828 4.5742852  4.48336537 4.5141129  4.21791965 4.17030782
 4.22640164 4.45268037 4.48474852 4.43769352]
Validation RMSEs:  [ 5.38507075  4.59652659  6.32313728  5.57556881 10.56163598  8.31752303
  9.66792933  6.62430684  6.91606435  7.54510338]
```

Why does it do much better than linear regression?

Neural networks can capture non-linear dependencies (with the help of activation functions such as relu) whereas linear models cannot. This means that neural network models have a larger hypothesis space to choose from including non-linear spaces and the complexity of the model is higher. Neural networks working better than linear regression suggest that for both of our dataset, its data has non-linear dependencies that help predicting the target variable.

Question 16¶ Adjust your network size (number of hidden neurons and depth), and weight decay as regularization. Find a good hyper-parameter set systematically.

There are different possibilities to systematically tune hyperparameters. To name a few we can use GridSearch, Randomized Search or HalvingGridSearchCV or HalvingRandomSearchCV from sklearn library. GridSearch is

an exhaustive search of all parameter combinations, whereas Randomized Search sample from the all parameter combinations based on the sample number given by the user and conducts the search from that random sample. With many parameters and many candidates for each parameter, GridSearch takes a very long time to run. RandomizedSearch in that sense can be a good candidate to avoid long run times.

HalvingSearch strategies is similar to GridSearch and RandomizedSearch but it applies succesive halving after every iteration and continues parameter search by halving the candidates after each iteration. The halving strategy is much faster but doesn't support for multiple scoring in sklearn, and for the next chapters, Random Forest part we are required to calculate multiple scores for the search results. In order to stay consistent on the parameter tuning techniques, I decided to go with Randomized Search with a good sample number, I can cover wider range of parameter combinations.

The hyperparameters tuned in this question are number of hidden neurons, number of hidden layers, and weight decay (alpha) parameters in the MLPRegressor object from sklearn.

- Number of hidden layers: 1, 2, 3, 4
- Number of hidden neurons: 30, 50, 100, 120
- The specific hidden_layer_sizes parameters used in the search: [(30,),(50,),(100,), (120,), (30,30), (50, 50), (50, 50, 50), (50,50,50,50)]
- Alpha values used in the range of $[10^{-5},10^3]$; with every 10 power range from -5 to 3 included.

I used Randomized Search with 100 samples and 10-fold cross validation due to the time it takes to train each combination. The same search space is kept for both datasets.

Diamonds Data¶ Neural Network Experiment Results

After the parameter tuning, the model performed better than the default neural network parameters. 10-fold Mean Train RMSE is now 595.01 anf 10-fold validation RMSE is 637.48 for the best model parameter combination.

These RMSE metrics were around 750-780 with the default NN results in Q15.

Top 10 best combination results:

	mean_train_score	mean_validation_score	param_model__alpha	param_model__hidden_layer_sizes
53	595.010078	637.488677	10.00000	(50, 50)
45	601.467299	641.372646	1.00000	(50, 50)
29	616.363558	642.977524	0.01000	(50, 50)
61	593.316732	644.878249	100.00000	(50, 50)
21	620.648627	650.541119	0.00100	(50, 50)
5	626.699392	658.240621	0.00001	(50, 50)
37	603.151681	661.902062	0.10000	(50, 50)
69	598.967497	664.097029	1000.00000	(50, 50)
13	625.323358	667.144400	0.00010	(50, 50)
54	568.869844	684.246047	10.00000	(50, 50, 50)

```
Best Model:
 Pipeline(steps=[('model',
                  MLPRegressor(alpha=10, early_stopping=True,
                               hidden_layer_sizes=(50, 50), random_state=142))])
Best Model Params:
 {'model__hidden_layer_sizes': (50, 50), 'model__alpha': 10, 'model': MLPRegressor(alpha=10, early_stoppi
             random_state=142)}
```

Using 2 hidden layers each with 50 neurons, with alpha 10 further improved the performance of the Neural Network model results.

Gas Data¶ Neural Network Experiment Results

After the parameter tuning, the model performed better than the default neural network parameters. 10-fold Mean Train RMSE is now 5.03 and 10-fold validation RMSE is 6.33 for the best model parameter combination.

Top 10 best combination results:

	mean_train_score	mean_validation_score	param_model__alpha	param_model__hidden_layer_sizes
50	5.034472	6.334995	10.0	(100,)
48	5.075886	6.339038	10.0	(30,)
51	4.979858	6.381388	10.0	(120,)
62	5.112718	6.396729	100.0	(50, 50, 50)
61	5.437319	6.439728	100.0	(50, 50)
49	5.056675	6.479673	10.0	(50,)
63	4.915925	6.642107	100.0	(50, 50, 50, 50)
40	4.828005	6.730642	1.0	(30,)
52	4.541026	6.769982	10.0	(30, 30)
53	4.509395	6.782365	10.0	(50, 50)

The best model parameter is 1 hidden layer with 100 neurons and alpha 10. Notice that the model can also found almost same 10-fold mean validation score with 1 hidden layer with 30 neuron size.

```
Best Model:
 Pipeline(steps=[('model',
                  MLPRegressor(alpha=10, early_stopping=True,
                               random_state=142))])
Best Model Params:
 {'model__hidden_layer_sizes': (100,), 'model__alpha': 10, 'model': MLPRegressor(alpha=10, early_stopping
```

Question 17¶ What activation function should be used for the output?

We want our output to be a numerical value as it is a regression task. The activation function could be identity (linear activation function), in this case the output will be linear output and the predicted result range will be from $-\infty$ to ∞. Another possible activation function could be ReLu, this will also return a numerical value but the range would be from 0 to ∞.

For diamonds data set since we are predicting price, therefore, it may be better to not allow negative range for the results and make it either zero or positive value. That is why we can use ReLU as activation. On another note, the price values on our data is all positive values with a minimum price 327. Therefore, using both linear output (no activation) and Relu would be possible in these two datasets.

For gas dataset as well, the target variable is again seems to be positive valued results, again we could use either of the activation functions.

sklearn MLPRegressor uses identity activation function as default, I kept using this in my experiments.

```
NN model output activation: identity
```

Question 18¶ What is the risk of increasing the depth of the network too far?

Two possible things can happen if we increase the depth of network too far:

1. With the increase of depth, hidden layers, model complexity increases and the **model can overfit** as wider hypothesis space is available to fit the train data. When model overfits, the train performance would be good, ie. low RMSe score however the validation and test scores would be bad since model cannot generalize well and predicts the unseen data badly.
2. Another possibility is the direct opposite, when the model layers increases, during back propagation the gradients of the loss function might approach to zero causing vanishing gradient problem. With vanishing gradient it gets harder for model to learn and train which leads to **model underfit**. When model underfits both train and validation performances would be bad, ie. very high RMSE scores.

We can check both training and testing loss curves during training to spot either of these problems is occuring. We can also check the cross validation results of train and test scores.

Below two tables show the results for 1,2,3,4 hidden layers with each 50 neurons. The first table shows the mean_train_score values on different model alpha and hidden layer size. The second plot shows the mean_validation_scores.

For diamonds dataset, we can see that when we increase the number of hidden layers, depth of the model, the training RMSE decrease but validation RMSE increases, which shows the models start to overfit.

	param_model___hidden_layer_sizes param_model___alpha	(50,)	(50, 50)	(50, 50, 50)	(50, 50, 50, 50)
mean_train_score	0.00001	860.387372	626.699392	566.731348	549.086232
	0.00010	860.387160	625.323358	566.014459	545.684920
	0.00100	860.386317	620.648627	564.883705	548.163076
	0.01000	860.392727	616.363558	563.522269	549.841766
	0.10000	860.609681	603.151681	566.650971	551.695817
	1.00000	862.323489	601.467299	566.704099	547.207621
	10.00000	868.732516	595.010078	568.869844	544.183427
	100.00000	875.627936	593.316732	570.243132	549.337372
	1000.00000	885.655224	598.967497	567.576103	551.231189

	param_model___hidden_layer_sizes param_model___alpha	(50,)	(50, 50)	(50, 50, 50)	(50, 50, 50, 50)
mean_validation_score	0.00001	860.478273	658.240621	752.885226	723.286306
	0.00010	860.475928	667.144400	740.185870	748.312344
	0.00100	860.470553	650.541119	725.238640	722.771248
	0.01000	860.489272	642.977524	723.608246	725.818526
	0.10000	859.995527	661.902062	736.513553	725.074987
	1.00000	863.376201	641.372646	701.309009	718.449520
	10.00000	855.726821	637.488677	684.246047	732.374139
	100.00000	863.994765	644.878249	699.210512	704.885216
	1000.00000	873.184344	664.097029	719.523413	718.318497

For gas dataset, we can see that when we increase the number of hidden layers, depth of the model, the training RMSE decrease but validation RMSE increases for mos tof the cases as well.

	param_model___hidden_layer_sizes param_model___alpha	(50,)	(50, 50)	(50, 50, 50)	(50, 50, 50, 50)
mean_train_score	0.00001	4.637899	4.090331	4.035329	3.884436
	0.00010	4.639643	4.084285	4.009868	3.972271
	0.00100	4.636757	4.100788	4.064147	3.945383
	0.01000	4.639676	4.131832	4.057370	3.820922
	0.10000	4.648805	4.111577	4.077180	3.899991
	1.00000	4.730151	4.100098	4.136366	3.929041
	10.00000	5.056675	4.509395	4.260865	4.118325
	100.00000	6.481601	5.437319	5.112718	4.915925
	1000.00000	8.306890	7.097664	6.680367	6.524656

	param_model___hidden_layer_sizes	(50,)	(50, 50)	(50, 50, 50)	(50, 50, 50, 50)
	param_model___alpha				
mean_validation_score	0.00001	6.901441	7.407305	7.185461	7.324528
	0.00010	6.913977	7.326312	7.107952	7.220236
	0.00100	6.891217	7.372348	7.181652	7.190370
	0.01000	6.978835	7.296752	7.120696	8.076287
	0.10000	6.891648	7.328153	7.045864	7.410987
	1.00000	6.902512	7.298903	7.335931	6.966002
	10.00000	6.479673	6.782365	7.047754	6.986474
	100.00000	7.384260	6.439728	6.396729	6.642107
	1000.00000	8.889102	7.945318	7.607922	7.507126

Random Forest¶

Question 19¶

Diamond Data¶ Random Forest with default Parameters

10-fold CV results with default random forest parameters:

```
RF Train Avg RMSE accross 10-fold cv:  196.18467228232933
RF Validation Avg RMSE accross 10-fold cv:  752.9655008100951
```

Random Forest Parameter Tuning

I applied 10-fold cross validation to tune parameters on randomized search with 200 samples on the following parameters:

- Maximum number of Features: ['auto', 'sqrt', 'log2', None, 2, 3, 4, 6]
- Number of Trees: [20, 40, 60, 80, 100, 120, 140, 160, 180, 200]
- Depth of each tree: [2, 4, 6, 8, 10, None]

RF Parameter tuning Top 10 Results:

The best 10-fold avg train RMSE is 197 and avg validation RMSE is 715 which is much better than the values we found from default Random Forest parameter results.

The avg. validation RMSE is also better than linear and polynomial regression but worse than the Neural Network results.

The R^2 value for 10-fold avg train score is 0.9975 which is very high. The R^2 mean validation value is 0.644 which is on the lower side.

	mean_train_RMSE	mean_validation_RMSE	param_model___max_features	param_model___n_estimators	p
93	197.555441	715.111659	log2	120	N
35	197.555441	715.111659	sqrt	120	N
123	196.395238	715.684125	sqrt	160	N
146	196.395238	715.684125	log2	160	N
57	196.904756	715.867991	3	140	N
143	198.482136	717.630609	sqrt	100	N
103	192.697889	719.262639	4	180	N
21	202.270267	722.666892	log2	60	N
62	195.142830	723.777170	4	100	N
25	208.128003	725.638959	sqrt	40	N

	mean_train_R2	mean_validation_R2	param_model__max_features	param_model__n_estimators	param_m
93	0.997510	0.644635	log2	120	NaN
35	0.997510	0.644635	sqrt	120	NaN
123	0.997539	0.644486	sqrt	160	NaN
146	0.997539	0.644486	log2	160	NaN
57	0.997526	0.644543	3	140	NaN
143	0.997487	0.643213	sqrt	100	NaN
103	0.997632	0.645301	4	180	NaN
21	0.997388	0.637716	log2	60	NaN
62	0.997571	0.642623	4	100	NaN
25	0.997235	0.634496	sqrt	40	NaN

The best parameter combination is max_features: 'sqrt', n_estimators=120, max_depth=None

```
Best Model:
 Pipeline(steps=[('model',
                  RandomForestRegressor(max_features='sqrt', n_estimators=120,
                                        n_jobs=-1, oob_score=True,
                                        random_state=142))])
Best Model Params:
 {'model__n_estimators': 120, 'model__max_features': 'sqrt', 'model__max_depth': None, 'model': RandomFor
                       oob_score=True, random_state=142)}

Best Model OOB score:
 0.981448128437458
```

We can also decrease the number of trees as we can substantially improve computation time without loosing too much from performance as it can be seen from the above table.

It is also important to note that when we have max_depth=None we don't really put a regularization effect on trees, the depth of the trees could grow unlimited. Similarly, having max_features None has unlimited effect as well. I also checked the best parameter results when these values are not equal to None. The best 10 results when we put limited on these parameters can be seen below table:

	148	94	116	34	133	29	47	37	46
mean_train_RMSE	482.615	482.462	482.678	482.917	477.473	488.145	477.528	477.496	477.588
mean_validation_RMSE	776.672	776.816	777.752	778.3	785.171	785.564	785.702	785.895	786.567
param_model__max_features	6	6	6	6	8	6	8	8	8
param_model__n_estimators	160	180	140	120	160	20	120	100	80
param_model__max_depth	10	10	10	10	10	10	10	10	10

By looking these results we can say that the validation RMSE is much bigger and validation R^2 is worse compared to the previous results, however the RMSE gap between train and validation scores are considerably lower. Given the data doesn't have too much samples, it makes more sense to choose a parameter with lower depth and number of trees in order to avoid overfitting, if there is no marginal change in the mean_validation RMSE. However, I beleive that this data doesn't have too much predictive power over price of diamonds given that even the best tuning results can only reach mean_validation_R^2 of 0.644 only.

Gas Data¶ Random Forest with default Parameters

10-fold CV results with default random forest parameters:

```
RF Train Avg RMSE accross 10-fold cv:  1.3334385377542095
RF Validation Avg RMSE accross 10-fold cv:  7.033552307420598
```

Random Forest Parameter Tuning

I applied 10-fold cross validation to tune parameters on randomized search with 200 samples on the following parameters:

- Maximum number of Features: ['auto', 'sqrt', 'log2', None, 2, 3, 4, 6]
- Number of Trees: [20, 40, 60, 80, 100, 120, 140, 160, 180, 200]
- Depth of each tree: [2, 4, 6, 8, 10, None]

RF Parameter tuning Top 10 Results:

The best 10-fold avg train RMSE is 1.34 and avg validation RMSE is 6.76 which is better than the values we found from default Random Forest parameter results.

The R^2 value for 10-fold avg train score is 0.987 which is very high. The R^2 mean validation value is 0.49 which is very low.

	164	7	181	125	49	24	196	69	1
mean_train_RMSE	1.3423	1.34874	1.38603	1.34421	1.34421	1.34421	1.36871	1.32183	1
mean_validation_RMSE	6.76208	6.76296	6.76297	6.76307	6.76307	6.76307	6.76424	6.78154	6
mean_train_R2	0.986713	0.986585	0.985832	0.986675	0.986675	0.986675	0.986183	0.987116	0
mean_validation_R2	0.496042	0.495604	0.496487	0.495475	0.495475	0.495475	0.496209	0.491896	0
param_model__max_features	sqrt	sqrt	sqrt	3	log2	sqrt	3	4	4
param_model__n_estimators	180	140	60	160	160	160	80	180	1
param_model__max_depth	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	N

The best parameter combination is max_features: 'sqrt', n_estimators=180, max_depth=None

```
Best Model:
 Pipeline(steps=[('model',
                  RandomForestRegressor(max_features='sqrt', n_estimators=180,
                                        n_jobs=-1, oob_score=True,
                                        random_state=142))])
Best Model Params:
 {'model__n_estimators': 180, 'model__max_features': 'sqrt', 'model__max_depth': None, 'model': RandomFor
                       oob_score=True, random_state=142)}

Best Model OOB score:
 0.9024957564053923
```

However, if we check the mean validation scores we can considerably lower the n_estimators and limit the max_dept instead of None to have a better regularization for our dataset.

Decreasing the number of trees would substantially improve computation time without loosing too much from performance as it can be seen from the above table.

It is also important to note that when we have max_depth=None we don't really put a regularization effect on trees, the depth of the trees could grow unlimited. Similarly, having max_features None has unlimited effect as well. I also checked the best parameter results when these values are not equal to None. The best 10 results when we put limited on these parameters can be seen below table:

	mean_train_RMSE	mean_validation_RMSE	mean_train_R2	mean_validation_R2	param_model__max_fe
30	3.775723	6.798711	0.894827	0.494492	4
119	3.610713	6.802990	0.903815	0.495103	6
176	3.610805	6.809854	0.903809	0.494213	6
66	3.610107	6.815662	0.903847	0.493449	6
185	3.776789	6.818777	0.894767	0.491180	4
27	3.781066	6.827195	0.894523	0.490624	4
4	3.606747	6.833424	0.904028	0.491379	6
95	3.559517	6.843458	0.906544	0.490892	8

	mean_train_RMSE	mean_validation_RMSE	mean_train_R2	mean_validation_R2	param_model__max_fea
173	3.959350	6.849650	0.884343	0.487073	3
143	3.525061	6.851873	0.908327	0.489752	8
3	3.965165	6.858520	0.883993	0.486477	log2
31	3.628738	6.859863	0.902855	0.487493	6
53	3.530787	6.864812	0.908032	0.488247	8
73	3.964615	6.868910	0.884006	0.484301	sqrt
63	3.533441	6.871307	0.907893	0.487545	8
161	4.389201	6.900573	0.857839	0.483253	6
167	4.318155	6.912368	0.862423	0.484139	8
101	4.318558	6.912621	0.862400	0.483288	8
54	3.976980	6.932200	0.883312	0.475565	log2
129	4.586106	6.941971	0.844839	0.476915	4

By looking these results we can say that the validation RMSE is bigger by only 0.05 point and validation R^2 is 0.006 worse compared to the previous results. So without hurting our results too much we can decrease our selected parameter range to max_depth 10 with even around 40-60 n_estimators.

Given the data doesn't have too much samples, it makes more sense to choose a parameter with lower depth and number of trees in order to avoid overfitting, if there is no marginal change in the mean_validation RMSE. However, I beleive that this data doesn't have too much predictive power over NOX given that even the best tuning results can only reach mean_validation_R^2 of around 0.49 only.

Explain how these hyper-parameters affect the overall performance? Do some of them have regularization effect?

Maximum number of features: (max_features) Initially, increasing maximum number of features will help increasing the model performance. However, having too big max_features would lead the test/validation error to increase, even though training error keeps decreasing. This means model become prone to overfit with a high value of max_features. This value can be used to have regularization effect to avoid overfitting.

Number of trees: (n_estimators) Increasing number of trees at first will help with the model performance, from 1 to optimal number of trees we can see a sharp increase on both training, validation and test scores (increase in cases like accuracy, or decrease in case of RMSE error). After certain number of points however, the scores will start to stabilize and plateaus and won't change much. Having more than optimal number of trees will increase the computation time but wouldn't degrade model performance much.

Depth of each tree: (max_depth), is the longest path from root to leaf. If we increase the max depth too much, the trees can grow bigger and the training data would start to overfit after certain point, this will keep decreasing the training score. However, due to overfit, the model cannot generalize well on unseen test data and test score would increase. Hence, if we have too big maximum depth values, the model performance would start to degrade. Having very small max_depth on the other hand might cause underfitting, with low test and training score, again resulting in low performance. So we need to find an optimal value and fine tune this parameter carefully. We can create a regularization effect by decreasing the max_depth and avoid overfitting.

Question 20¶ Why does random forest perform well?

Random forest is an ensemble model using bagging method, this means we have multiple decision trees working as a group and final prediction is based on a majority vote instead of a decision of an individual tree. Having multiple trees help having a more robust, accurate model and reduce the variance of the model. Giving that the final predictions based on a majority vote, the final prediction result doesn't get affected from a wrong prediction of an (few) individual models because decisions is lead by the most common results. This helps Random Forest to use the power of multiple classifiers, reduce the prone to overfit and performs well.

Question 21¶

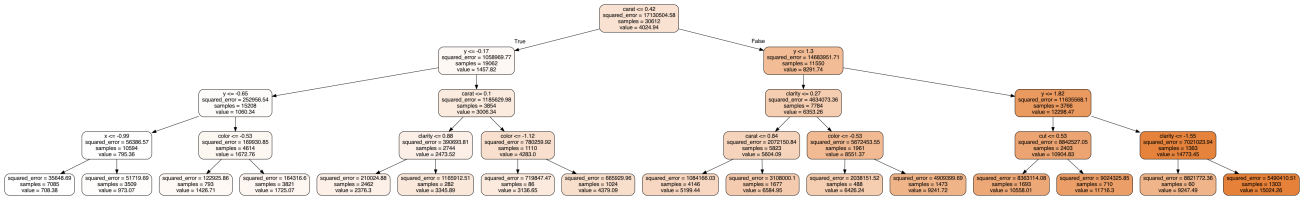
Diamond Data:¶ For this question I created a model with max_depth 4, and the choose the remaining best parameters from the tuning results: when max_depth is 4, n_Estimators 60 and max_features='sqrt' have a good performance. Using higher n_estimators as it can be seen from the below table, doesn't effect the validation RMSE score too much, therefore it is a good tradeoff to have a lower n_estimators since it doesn't really hurt mean_validation RMSE. Although we can see that wiht max depth 4 results the RMSE results are much higher.

	180	55	115	117	56	44	109	106
mean_train_RMSE	1072.82	1078.59	1079.51	1035.39	1053	1112.7	1112.7	1110.16
mean_validation_RMSE	1279.92	1281.97	1283.39	1290.71	1299.49	1301.49	1301.49	1302.32
mean_train_R2	0.925749	0.924969	0.92484	0.930787	0.928445	0.920427	0.920427	0.920779
mean_validation_R2	-0.368551	-0.359162	-0.389649	-0.35644	-0.365391	-0.3462	-0.3462	-0.330555
param_model___max_features	4	4	4	6	6	3	log2	sqrt
param_model___n_estimators	40	160	20	120	20	80	80	60
param_model___max_depth	4	4	4	4	4	4	4	4

```
RF Train Avg RMSE accross 10-fold cv:  1055.1147964059605
RF Validation Avg RMSE accross 10-fold cv:  1283.5570855502583
```

Then I select a tree with max depth 4 and plot it.

```
Selected tree
: DecisionTreeRegressor(max_depth=4, max_features=5, random_state=1726841109)
```



Which feature is selected for branching at the root node?

Carat feature is selected as root node. This makes sense since until now, we kept finding carat is the most important feature to predicting price in the previous questions.

What can you infer about the importance of features?

For the splits of the branches: carat, x, y, clarity, color, cut features are selected, hence the features that are found to be important for this decision tree. The first split is made using carat feature and the standardized carat features lower than 0.42 goes to the left node and the rest goes to the right node, we can see that from almost 31K samples this decision splits the samples almost 2/3 to 1/3 for the next nodes, which means the feature is very imformative in terms of the being best split to separate samples. The most important feature is carat since it is used on the root node, then the the features used closer to the root are more informative, hence important features compared to the ones closer to leaf nodes. The decision rules created by this tree can be clearly seen for each node and its corresponding feature. The split decisions are made based on the feature and its corresponding threshold where mean squared error is the most reduced.

We can also observe that some features are used in several nodes and splits and chosen for new decision splits, such as carat, y, color this shows that these 3 features play more critical role for the final prediction of this specific tree.

Do the important features match what you got in part 3.2.1?

In the best model found after Linear Regression experiments, 3.2.1, the most important features were carat, x, clarity, color cut and depth in order. According to the above decision tree, we can as well see that the important features found are very similar, the only difference y and depth and the rest of the most important features are same.

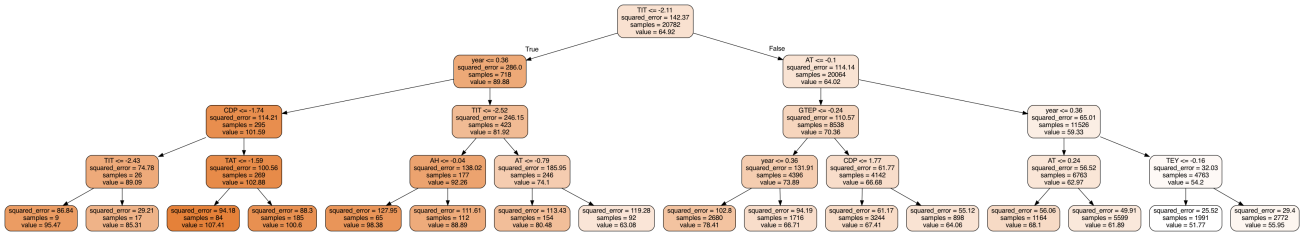
Gas Data:¶ For this question I created a model with max_depth 4, and the choose the remaining best parameters from the tuning results: when max_depth is 4, n_Estimators 40 and max_features='sqrt' have a good performance. Using higher n_estimators as it can be seen from the below table, doesn't effect the validation RMSE score too much, therefore it is a good tradeoff to have a lower n_estimators since it doesn't really hurt mean_validation RMSE. Although we can see that wiht max depth 4 results the RMSE results are higher.

	112	39	25	92	123	152	98	11	1
mean_train_RMSE	6.53428	6.60787	6.54067	6.61155	6.60513	6.8091	6.80822	6.81136	6
mean_validation_RMSE	7.69942	7.71153	7.71403	7.74121	7.74583	7.90723	7.90941	7.91341	7
mean_train_R2	0.685086	0.677974	0.684467	0.677633	0.678264	0.658098	0.6582	0.657878	0
mean_validation_R2	0.373734	0.371504	0.372889	0.368894	0.370643	0.347655	0.347542	0.347506	0
param_model___max_features	6	6	6	4	4	log2	sqrt	sqrt	3
param_model___n_estimators	80	20	120	100	160	100	120	140	6
param_model___max_depth	4	4	4	4	4	4	4	4	4

```
RF Train Avg RMSE accross 10-fold cv:  6.535650579623966
RF Validation Avg RMSE accross 10-fold cv:  7.683507321594336
```

Then I select a tree with max depth 4 and plot it.

```
Selected tree
: DecisionTreeRegressor(max_depth=4, max_features=5, random_state=1726841109)
```



Which feature is selected for branching at the root node?

TIT feature is selected as root node. The split rule is checked TIT <=-2.11.

What can you infer about the importance of features?

For the splits of the branches: TIT, year, CDP, TAT, AH, AT, GTEP, TEY are used. These features are the features found to be the most important ones that is used to split samples. The most important feature is TIT since it is used on the root node, then the the features used closer to the root are more informative, hence important features compared to the ones closer to leaf nodes. We can also observe that some features are used in several nodes and splits and chosen for new decision splits, this shows that these features play more critical role for the final prediction of this specific tree.

Do the important features match what you got in part 3.2.1?

In the best model found after Linear Regression experiments, 3.2.1, the most important features were TEY, TIT, AT, TAT and AH in order. According to the above decision tree, we can as well see that the important features found are very similar, in this case though TIT found to be more important than TEY since TIT is used as the root node.

LightGBM, CatBoost and Bayesian Optimization¶

I picked the Diamonds dataset for this section:

Question 22¶

LightGBM Parameters to tune¶ LightGBM uses leaf-wise tree growth algorithm whereas many other tree algorithms use dept-wiswe tree growth. Leaf-wise algorithms have an advantage of converging faster but also more prone to overfitting if not used appropriatee parameters.

Parameters to tune:

- num_leaves: the parameter is the main one to control complexity of the model according to the documentation. It should be smaller than 2^(max_depth) to avoid overfittting.
- min_data_in_leaf: this parameter helps prevent overfitting as well. The optimal value depends number of training samples and num_leaves. If it is large, it avoids growing too deep tree but also may cause underfitting. According to the documentation setting it in hundreds or thousands is enough for a large dataset. Since our datasets are smaller I will try values starting from 10 to 1000.
- max_depth: This parameter also limit tree depth, hence helps avoid overfitting.

There are also other parameters to use for better performance results suggested in the documentation: use large max_bin (may be slower), use small learning_rate with large num_iterations, use large num_leaves (may cause overfitting) and try dart as boosting_type.

My final set of parameters and the search space for LightGBM is:

- num_leaves = [2**(x) for x in range(4,8)]
- min_data_in_leaf = [10, 20, 50, 75, 100, 250, 500, 750, 1000]
- max_depth = [-1, 3, 4, 5, 6, 7, 8, 10, 20]
- boosting_type = ['gbdt', 'dart']
- n_estimators = [30, 50, 100, 150]
- num_iterations = list(range(100, 301, 50))
- learning_rate = [0.1, 0.05, 0.01]

I decided the above search space range based on the documentation recommendations, further research on the internet and by taking into account my dataset and the samples in there (the number of samples are not big so I tried to keep some parameters smaller, or added smaller numbers as well to try to avoid overfitting).

Reference: https://lightgbm.readthedocs.io/en/latest/Parameters-Tuning.html

CatBoost Parameters to tune¶ For catboost the important parameters to tune for this questions are:

- iterations: maximum number of trees that can be built
- learning_rate
- depth: Depth of the tree
- l2-leaf-reg: L2 regularization coefficient which is used for leaf value calculation.
- od-pval is threshold for overfitting detector, it is set to the recommended range in the documentation.
- random_strength: amount of randomness to use for scoring splits , it helps avoid overfitting,
- bagging_temperature: defines the settings of the Bayesian bootsrap,the higher the value the most aggressive bagging is.
- border_count: number of splits for the numerical features (its alias is max_bin)

I decided on these parameters after reading catboost documentation for parameter tuning part: https://catboo st.ai/en/docs/concepts/parameter-tuning For the search space for each parameter, I decided either based on the recommendation range in documenation or based on the dataset and number of samples I have in the diamonds dataset.

My final set of parameters and the search space for CatBoost is:

- iterations: Integer(50, 300)
- learning_rate: Real(0.01, 1.0, 'log-uniform')
- depth: Integer(4, 10)
- od-pval: Real(10e-10, 10e-2, 'log-uniform'), (is set to the recommended range in the documentation.)
- l2_leaf_reg': Integer(2, 20),

- random_strength': Real(1e-9, 10, 'log-uniform'),
- bagging_temperature': Real(0.0, 1.0),
- border_count': Integer(1, 255),

Question 23¶ Apply Bayesian optimization using skopt.BayesSearchCV from scikit-optmize to search good hyperparameter combinations in your search space. Report the best hyperparameter found and the corresponding RMSE, for both algorithms.

In this question I used the parameters and search spaces I defined for each of them to tune my parameters for LightGBM and CatBoost models. I applied Bayesian optimization with BayesSearchCV for 10-folds for both models.

LightGBM Results¶ To be able to understand how much results are improved with parameter tuning, I also ran a 10-fold cross validation LightGBM with its default parameters. The results are below, we can see that LightGBM is already performing way better than all the other models with its default values.

```
LightGBM Train Avg RMSE accross 10-fold cv:  466.01792200127346
LightGBM Validation Avg RMSE accross 10-fold cv:  657.5308021388921
```

LightGBM Parameter Tuning Results

For LightGBM 200 samples (n_iter=200) are choosen during BayesSearchCV and 10-fold cv is apploed, the parameters tuned can be seen from Q22. The top 10 best mean_validation_score score results are shown in the below table:

	199	122	120	118	117	116	115	114	113
mean_train_score	500.356	500.356	500.356	500.356	500.356	500.356	500.356	500.356	500.3
mean_validation_score	599.346	599.346	599.346	599.346	599.346	599.346	599.346	599.346	599.3
param_model__num_iterations	300	300	300	300	300	300	300	300	300
param_model__learning_rate	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1
param_model__n_estimators	50	50	150	50	50	30	100	50	50
param_model__num_leaves	64	64	32	16	32	32	64	32	32
param_model__min_data_in_leaf	75	75	75	75	75	75	75	75	75
param_model__max_depth	4	4	4	4	4	4	4	4	4
param_model__boosting_type	gbdt	gbdt	gbdt	gbdt	gbdt	gbdt	gbdt	gbdt	gbdt

With parameter tuning we can see that we reached even better results than the default parameter values. The mean train RMSE is around 500.35 and mean validation RMSE is around 599.34. These results are also way better than the models trained on diamonds dataset on the previous questions.

The best parameters found are:

- max depth = 4,
- min_data_in_leaf=75,
- num_iterations=300,
- num_leaves=16,
- learning_rate=0.1,
- n_estimators = 100

The best parameters and estimator chosen by BayesSearchCV is shown below:

```
Best model:
 Pipeline(steps=[('model',
                  LGBMRegressor(max_depth=4, min_data_in_leaf=75, n_jobs=1,
                                num_iterations=300, num_leaves=16,
                                random_state=142))])
Best model parameters:
 OrderedDict([('model', LGBMRegressor(max_depth=4, min_data_in_leaf=75, n_jobs=1, num_iterations=300,
```

```
num_leaves=16, random_state=142)), ('model__boosting_type', 'gbdt'), ('model__learning_rate
```

One thing we can notice is that we actually get same scores for different combinations in the top 10 best results. When I further explored I saw that this is the case for the best 100 experiment combination results. We can explain this as: after some the mean_validation_score starts to saturate and the model cannot learn more from the data. In this case it is better to choose the best parameter combinations by minimizing the computation time and/or by choosing some of the parameters that has better regularization effects.

Below table describes the parameter tuning that has thee best training and validation mean scores. For example, we can choose a best parameter combination wheren n_estimators could be as low as 30 and still can get same results or where param_model___num_leaves is as low as 16.

	count	mean	std	min	25%	50%	75%
mean_train_score	101.0	500.356272	1.142539e-13	500.356272	500.356272	500.356272	500.356
mean_validation_score	101.0	599.345728	2.285077e-13	599.345728	599.345728	599.345728	599.345
param_model___num_iterations	101.0	300.000000	0.000000e+00	300.000000	300.000000	300.000000	300.000
param_model___learning_rate	101.0	0.100000	2.789401e-17	0.100000	0.100000	0.100000	0.10000
param_model___n_estimators	101.0	78.910891	4.337974e+01	30.000000	50.000000	50.000000	100.000
param_model___num_leaves	101.0	36.910891	2.045781e+01	16.000000	16.000000	32.000000	64.0000
param_model___min_data_in_leaf	101.0	75.000000	0.000000e+00	75.000000	75.000000	75.000000	75.0000
param_model___max_depth	101.0	4.000000	0.000000e+00	4.000000	4.000000	4.000000	4.00000

CatBoost Results¶ To be able to understand how much results are improved with parameter tuning, I also ran a 10-fold cross validation CatBoost with its default parameters. The results are below, we can see that CatBoost is now the best model for diamonds dataset compared to the all models we performed in this project even with its default values.

```
CatBoost Train Avg RMSE accross 10-fold cv:  427.7612680829581
CatBoost Validation Avg RMSE accross 10-fold cv:  551.9742619473776
```

CatBoost Parameter Tuning Results

For CatBoost 300 samples (n_iter=300) are choosen during BayesSearchCV and 10-fold cv is applied, the parameters tuned can be seen from Q22. The top 10 best mean_validation_score score results are shown in the below table:

The best results 10-fold mean train score is 475 and mean validation score is 556.

	279	246	288	269	267	272	265
mean_train_score	4.756663e+02	466.669808	472.834587	479.240343	464.279413	462.116113	453.9289
mean_validation_score	5.560678e+02	557.182539	557.381485	557.608886	557.778176	558.255516	558.3399
param_bagging_temperature	4.185244e-01	0.997217	1.000000	0.664465	1.000000	0.316298	1.000000
param_border_count	2.140000e+02	216.000000	214.000000	213.000000	212.000000	212.000000	212.0000
param_depth	6.000000e+00	7.000000	6.000000	6.000000	7.000000	7.000000	7.000000
param_iterations	3.000000e+02	300.000000	300.000000	290.000000	289.000000	286.000000	286.0000
param_l2_leaf_reg	2.000000e+00	2.000000	2.000000	2.000000	2.000000	2.000000	2.000000
param_learning_rate	1.232655e-01	0.102324	0.130456	0.121453	0.111908	0.118755	0.134351
param_od_pval	8.715984e-09	0.098299	0.100000	0.100000	0.100000	0.035189	0.100000
param_random_strength	1.000000e+01	10.000000	10.000000	10.000000	10.000000	10.000000	10.00000

The best parameters found:

	279
mean_train_score	4.756663e+02
mean_validation_score	5.560678e+02

	279
param_bagging_temperature	4.185244e-01
param_border_count	2.140000e+02
param_depth	6.000000e+00
param_iterations	3.000000e+02
param_l2_leaf_reg	2.000000e+00
param_learning_rate	1.232655e-01
param_od_pval	8.715984e-09
param_random_strength	1.000000e+01

```
Best model parameters:
 OrderedDict([('bagging_temperature', 0.4185244058413902), ('border_count', 214), ('depth', 6), ('iterati
```

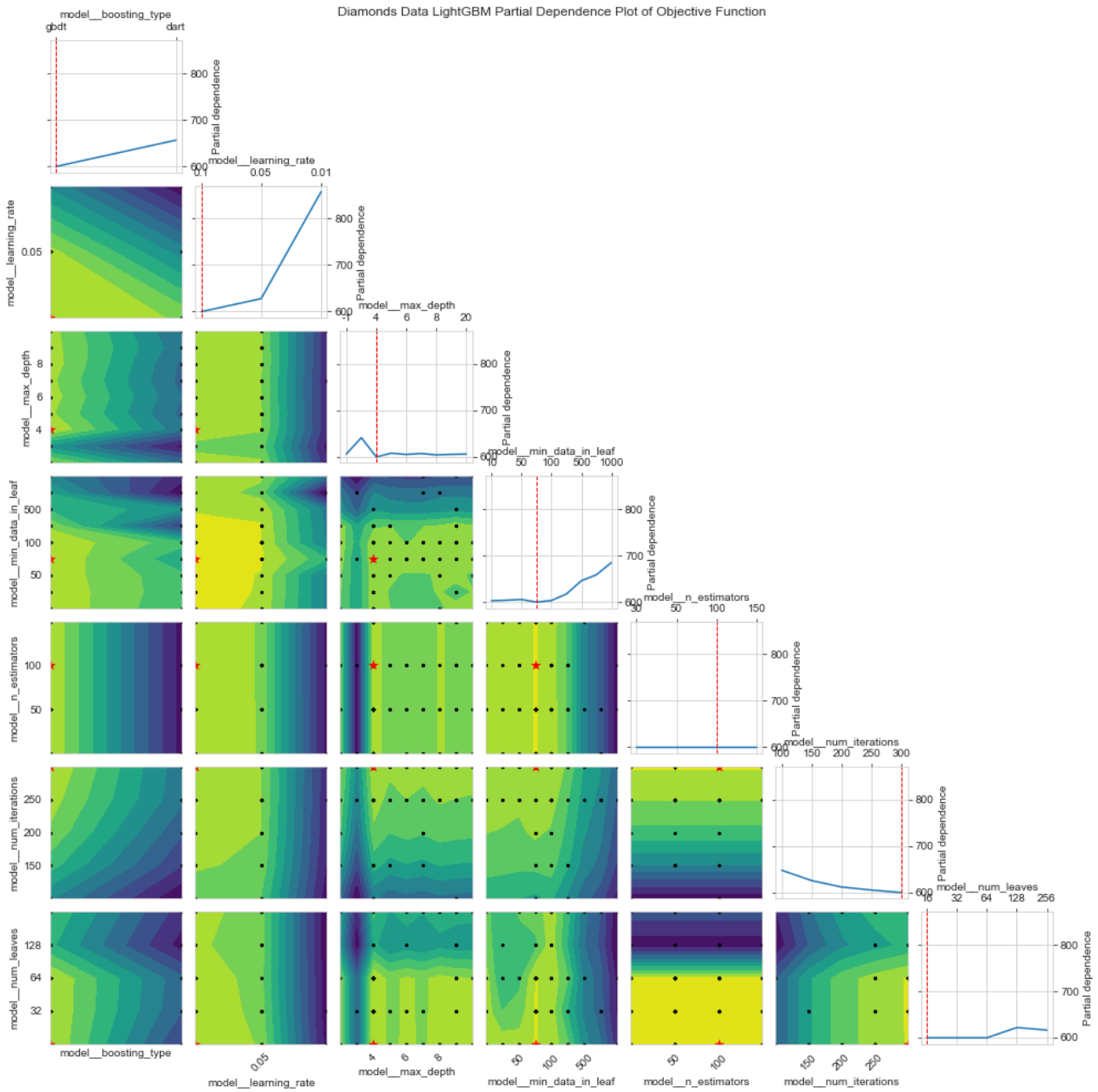
Again the validation results are very close and there isn't marginally very big difference for the first 5-6 combinations so one can choose the best combination based the resources and needs they have.

Question 24¶ Plot_objective plot showd the influence of each search space dimension on the objective function of BayesSearch results. The diagonal shows the effect of a single dimension on objective function and the other subplots show the effect on objective function warying two dimensions. The black dots corresponds to the samples used during optimization, red start indicates the best observed minimum.

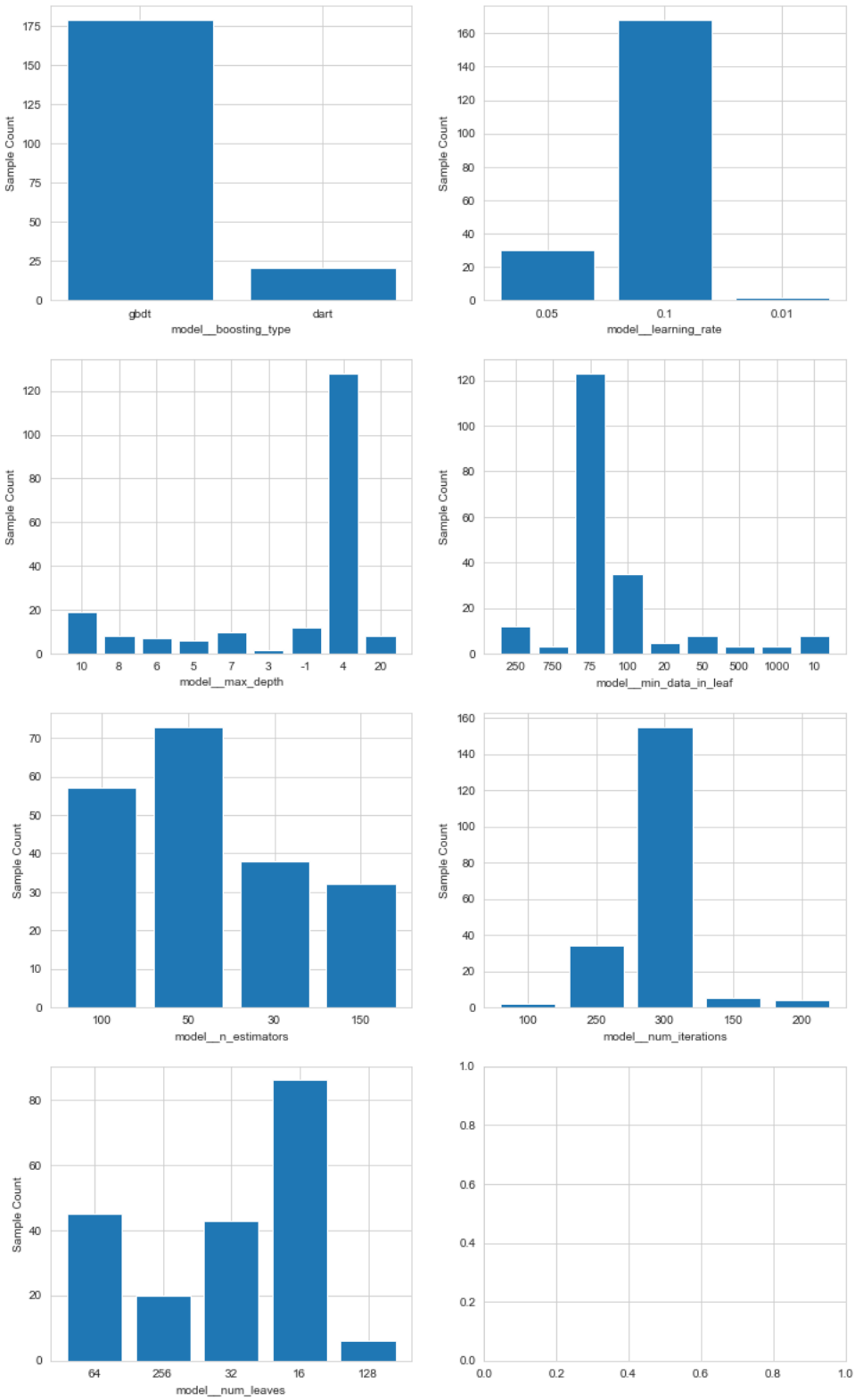
Interpret the effect of the hyperparameters using the Bayesian optimization results:

- **Which of them helps with performance? Which helps with regularization (shrinks the generalization gap)?**
- **Which affects the fitting efficiency?**
- **Endorse your interpretation with numbers and visualizations.**

LightGBM¶ I first plotted the Partial Dependence plot and histogram of the samples used in BayesSearch to better understand the parameters and its values used in the search and bayesian optimization:


Diamonds Data LightGBM Partial Dependence Plot of Objective Function
model__boosting_type
gbdt
dart
model__learning_rate
0.1
0.05
0.01
model__max_depth
model__min_data_in_leaf
model__n_estimators
model__num_iterations
model__num_leaves
Partial dependence
600
700
800

Diamonds Data LightGBM Sample Count Histogram for tuned features



Below tables for each tuned feature shows the mean fit time, mean validation score, mean training score and the RMSE gap between mean train and validation scores:

param__model___boosting__type	mean__validation__score	mean__train__score	RMSE__gap	mean__fit__time
dart	732.384601	660.391542	71.993059	9.015170
gbdt	611.090529	497.698208	113.392321	0.574902

param__model___learning__rate	mean__validation__score	mean__train__score	RMSE__gap	mean__fit__time
0.01	1313.382406	1473.259320	-159.876914	4.552230
0.05	660.143379	524.846458	135.296921	1.253376
0.10	609.132233	501.573150	107.559083	1.461431

param__model___max__depth	mean__validation__score	mean__train__score	RMSE__gap	mean__fit__time
-1	619.903682	446.765218	173.138464	3.806712
3	702.371362	613.193459	89.177903	0.405473
4	604.561980	504.542860	100.019121	0.659794
5	678.761426	585.237979	93.523447	0.826093
6	618.857126	484.653455	134.203671	0.555397
7	782.929403	736.480454	46.448949	2.633360
8	630.309871	509.251233	121.058637	2.092659
10	632.892752	495.497381	137.395371	4.347865
20	654.557147	503.734921	150.822226	3.344031

param__model___min__data__in__leaf	mean__validation__score	mean__train__score	RMSE__gap	mean__fit__time
10	625.887256	457.509027	168.378229	1.207867
20	631.032231	459.093484	171.938746	1.543826
50	626.692511	474.658738	152.033773	3.418163
75	607.232678	502.773876	104.458802	0.552696
100	618.406182	492.621715	125.784467	3.721999
250	662.477765	571.793104	90.684661	0.973568
500	653.891559	564.410250	89.481309	5.694160
750	1070.571722	1191.445502	-120.873780	3.101194
1000	710.841080	663.786936	47.054144	3.726084

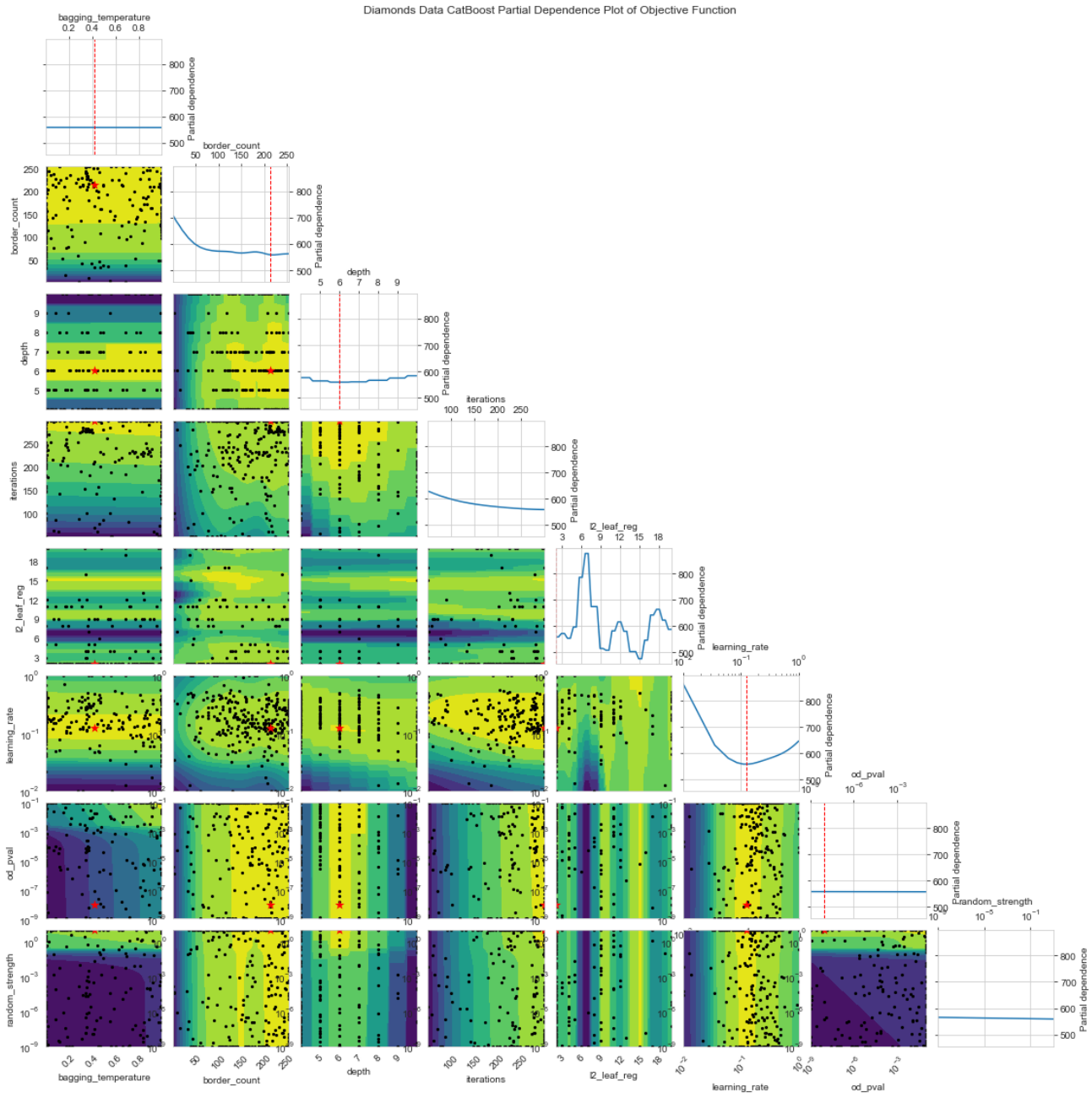
param__model___n__estimators	mean__validation__score	mean__train__score	RMSE__gap	mean__fit__time
30	649.184105	555.880731	93.303374	1.723009
50	619.245019	509.105813	110.139206	1.594337
100	619.466426	497.169168	122.297257	1.052512
150	611.931646	510.292717	101.638929	1.574125

param_model___num_iterations	mean_validation_score	mean_train_score	RMSE_gap	mean_fit_time
100	661.997451	545.298755	116.698696	0.292926
150	715.500251	622.184556	93.315695	1.286618
200	685.856882	577.071667	108.785216	3.212991
250	671.042687	567.586741	103.455946	4.631984
300	608.418750	497.731906	110.686844	0.741082

param_model___num_leaves	mean_validation_score	mean_train_score	RMSE_gap	mean_fit_time
16	620.299137	513.496141	106.802996	0.926784
32	642.961006	553.977861	88.983146	0.851639
64	609.813652	503.870603	105.943049	1.665036
128	626.046208	460.903003	165.143205	2.708304
256	628.717030	476.744514	151.972516	4.236288

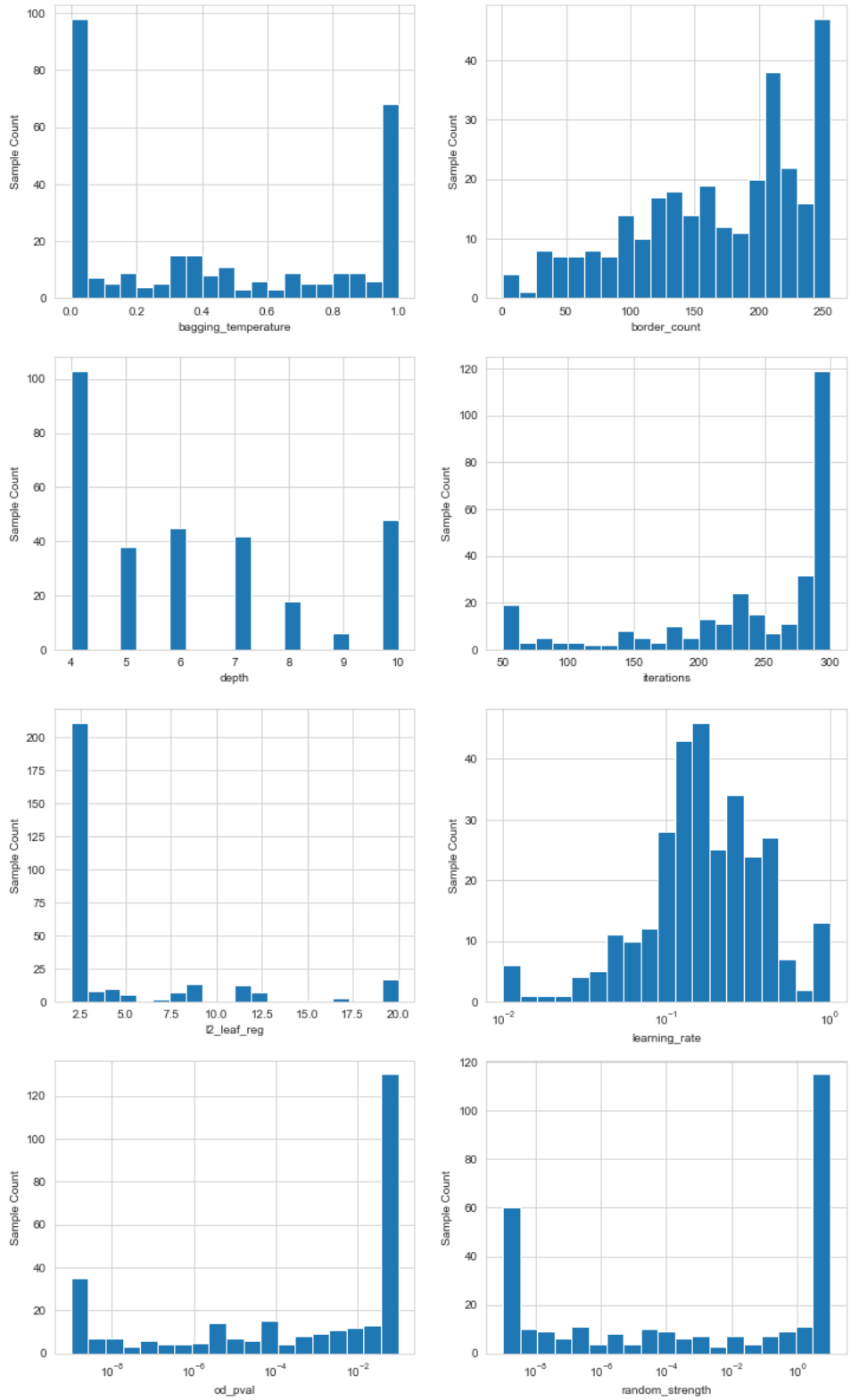
From the above plots and tables, we can interpret find hyperparameters that effect the results most, helps/hurts the fitting efficiency and the ones helps with the regularization.

For the LightGBM, according to the Partial Dependence Plot and above tables, Learning rate, boosting_type, min_data_in_leaf, num_iterations helps with the performance of the model. Num_leaves, min_data_in_leaf, max_depth seems to help with regularization. Learning rate, num_iterations and max_depth again does effect the fitting efficiency, small learning rate combined with high number of iterations and big max_depth value increase the fitting time of the model (mean fit time in the above tables). The boosting_type dart is also have way bigger fitting time compared to gbdt.


Diamonds Data CatBoost Partial Dependence Plot of Objective Function
bagging_temperature
border_count
depth
iterations
l2_leaf_reg
learning_rate
od_pval
random_strength
Partial dependence

CatBoost¶

Diamonds Data CatBoost Sample Count Histogram for tuned features



Below tables for each tuned feature shows the mean fit time, mean validation score, mean training score and the RMSE gap between mean train and validation scores:

param_bagging_temperature	mean_validation_score	mean_train_score	RMSE_gap	mean_fit_time
0.000000	596.420554	490.671380	105.749175	1.467983
0.002968	576.911417	499.741331	77.170087	1.183442
0.004518	574.596597	443.984078	130.612520	1.809631
0.006950	567.890303	496.749494	71.140808	1.119211
0.016229	575.371422	449.212034	126.159388	1.092802
0.016264	578.230057	496.844748	81.385309	0.680561
0.026538	568.175920	473.947187	94.228733	4.381694
0.039710	575.617139	507.465929	68.151210	0.915352
0.042568	561.070924	470.670895	90.400029	1.169266
0.053071	559.642755	476.460018	83.182736	1.181843

param_border_count	mean_validation_score	mean_train_score	RMSE_gap	mean_fit_time
1	2437.885042	2705.875540	-267.990498	0.302329
2	2155.604170	2030.159951	125.444219	1.518940
3	2050.695108	1882.821540	167.873567	0.804732
9	1113.351649	774.600469	338.751180	1.016317
16	868.945692	684.835780	184.109911	0.843478
31	983.486290	919.517794	63.968496	0.858223
32	683.633660	515.289167	168.344492	1.159559
34	674.054715	514.836671	159.218044	2.658881
35	654.210713	569.609654	84.601059	0.928517
37	874.935538	827.448298	47.487241	1.262819

param_depth	mean_validation_score	mean_train_score	RMSE_gap	mean_fit_time
4	619.482407	526.896674	92.585733	0.881004
5	581.061688	485.809802	95.251886	1.270390
6	677.234097	590.801220	86.432877	1.123163
7	577.630664	459.122405	118.508259	1.384647
8	621.180552	466.948582	154.231970	1.486894
9	891.826006	760.145866	131.680140	2.351749
10	621.733797	475.404196	146.329601	3.048691

param_iterations	mean_validation_score	mean_train_score	RMSE_gap	mean_fit_time
50	784.127465	662.170635	121.956830	0.415664
52	628.354904	552.337812	76.017092	0.160174
57	636.942447	565.288087	71.654360	0.201277
60	739.782186	662.767540	77.014646	0.247884
61	611.386375	522.973640	88.412735	0.311200
62	632.731844	551.300124	81.431720	0.649670
64	615.897542	541.200627	74.696915	0.250633
69	682.088667	551.531170	130.557497	0.798380
73	602.712813	491.348203	111.364610	0.503462

param_iterations	mean_validation_score	mean_train_score	RMSE_gap	mean_fit_time
77	635.231990	496.475672	138.756318	0.659570

param_l2_leaf_reg	mean_validation_score	mean_train_score	RMSE_gap	mean_fit_time
2	592.521556	482.540399	109.981157	1.552190
3	593.212444	483.580004	109.632440	0.922385
4	587.557544	474.519385	113.038159	0.829635
5	698.806326	609.794996	89.011330	0.620922
6	635.231990	496.475672	138.756318	0.659570
7	870.853276	663.469141	207.384136	0.588245
8	850.717300	747.746240	102.971060	1.234575
9	738.727187	610.681058	128.046130	1.877911
11	620.824965	486.455947	134.369017	1.480590
12	851.370087	805.707700	45.662387	1.482583

param_learning_rate	mean_validation_score	mean_train_score	RMSE_gap	mean_fit_time
0.010000	1172.742308	1174.963365	-2.221056	1.634913
0.012573	874.935538	827.448298	47.487241	1.262819
0.013914	929.489008	903.911986	25.577022	1.031312
0.017566	632.663857	570.690434	61.973424	5.734875
0.022048	731.142892	633.231155	97.911737	1.501995
0.027387	629.020778	521.534205	107.486573	4.283874
0.027934	735.845770	643.380068	92.465702	0.795100
0.030159	609.701538	500.774356	108.927182	6.437489
0.032528	654.259492	595.037900	59.221592	1.446941
0.035221	594.356834	534.476773	59.880061	2.928060

param_od_pval	mean_validation_score	mean_train_score	RMSE_gap	mean_fit_time
1.000000e-09	584.903536	481.655064	103.248472	1.926178
1.116487e-09	567.806218	484.138349	83.667870	1.005269
1.170697e-09	576.528644	465.930763	110.597881	2.044189
1.185287e-09	570.639234	473.850629	96.788605	0.829658
1.281338e-09	569.995140	463.015873	106.979267	1.397618
1.847439e-09	622.150930	471.792595	150.358334	0.701155
2.654482e-09	573.167168	499.362146	73.805022	5.784815
2.655925e-09	602.712813	491.348203	111.364610	0.503462
2.805438e-09	578.230057	496.844748	81.385309	0.680561
4.801575e-09	671.803006	581.522814	90.280192	0.181508

param_random_strength	mean_validation_score	mean_train_score	RMSE_gap	mean_fit_time
1.000000e-09	607.071588	474.953623	132.117965	1.547080
1.179524e-09	568.576941	454.479939	114.097002	1.115034
1.337562e-09	575.371422	449.212034	126.159388	1.092802

param_random_strength	mean_validation_score	mean_train_score	RMSE_gap	mean_fit_time
1.948190e-09	620.678094	445.365418	175.312676	1.742592
1.970483e-09	576.543032	476.682028	99.861004	0.963859
2.317262e-09	571.148219	490.567075	80.581144	0.908414
2.657184e-09	605.680737	513.124624	92.556113	0.997489
2.911235e-09	731.142892	633.231155	97.911737	1.501995
3.064276e-09	582.312354	470.518416	111.793937	0.786395
3.137981e-09	611.386375	522.973640	88.412735	0.311200

From the above plots and tables, we can interpret find hyperparameters that effect the results most, helps/hurts the fitting efficiency and the ones helps with the regularization.

For CatBoost model, according to the Partial Dependence Plot and above tables, border count, depth, iterations, l2_leaf_reg, learning_rate are thehyperparameters that has the highest effects on model performance. The ones that has most effect is l2_leaf_reg and learning rate. You can observe the changes of RMSE score with the change of these features by looking the diagonal subplots of Partial Dependence plot. Depth, l2_leaf_reg, random_strength all seems to help with regularization. Depth parameter effects the fitting time inversely. Learning rate, iterations again effect the fitting efficiency, small learning rate combined with high number of iterations increase the fitting time of the model.

Evaluation¶

Question 25¶ The two tables below shows the 10-fold avg training and validation RMSE scores for best found model results for each algorithms.

Diamonds Best Results for Each Model

	model	mean_train_score	mean_validation_score	abs_RMSE_gap
0	Lasso	1206.280597	1205.045051	1.235546
1	Ridge	1206.280597	1205.045051	1.235546
2	Polynomial Regression	921.572213	882.226435	39.345778
3	Neural Network	595.010078	637.488677	42.478598
4	Random Forest	197.555441	715.111659	517.556217
5	LightGBM	500.356272	599.345728	98.989456
6	CatBoost	475.666327	556.067794	80.401467

Gas Best Results for Each Model

	model	mean_train_score	mean_validation_score	abs_RMSE_gap
0	Lasso	8.036236	8.802989	0.766753
1	Ridge	8.094992	8.761643	0.666650
2	Polynomial Regression	5.719778	7.155798	1.436020
3	Neural Network	5.034472	6.334995	1.300523
4	Random Forest	1.342304	6.762084	5.419779

Why is the training RMSE different from that of validation set?

Having a higher train validation gap, when train error is low and validation error is high, generally refers to overfitting (model starts to memorize/fitt training data very well that it cannot perform well/generalize on unseen test data).
When both train and validation errors are high and gap is small, this would mean the model underfits the data, which is model cannot learn enough from data to generalize.

Diamonds Data: For the Linear Reg. without Regularization, Lasso and Ridge Regression the train and validation scores are very similar but very high, this might suggest that they are underfitting the dataset we have. The training RMSE and validation RMSE gap is the biggest in Random Forest part, training error is very low and validation is high, suggesting overfitting. As stated in the previous sections, for Random Forest we may pick a better parameter results manually that doesn't overfit, with low gap between the train and test RMSE and low RMSE result (details on RF section). For LightGBM and CatBoost are the model types that have higher RMSE gaps, after Random Forest. Again, we can see that train score are lower and the validation scores are bit higher. This shows that the model fit better to the training data and not as much to the test data. For Polynomial Regression we see that the training score is higher than validation, this is generally not an expected behavior but might be because the regularization value chosen is big on the best model. Neural Network gap is also low, which shows model fit well to both train and validation sets. The best validation results found in all model belongs to CatBoost with 556 RMSE score and the RMSe gap is 80.4, which is acceptable.

Gas Data:

We see similar results on gas data as well, Lasso and Ridge have high train and validation scores, with little gap, referring to underfit. The models cannot fit the data very well and hypothesis space used is not enough to capture the variability in the data. Random Forest looks like overfitting again with very high gap and low train, high validation result, better parameters may bee picked manually. Polynomiald Regression and Neural Network train and validation scores are lower and the gap between them is not much, showing these models are well fit and generalizing well in the unseen test data. The best model found for gas data is neural network.

Question 26¶ Explain what OOB error and R^2 score means.

OOB Error: Out of bag error, measures the prediction error of Random Forest. In Random Forest we use bootsrap samples for training the trees. Then the out of bag samples are the samples that are not selected, left out ones from bootstrap samples. The out of bag samples won't be used in training and therefore can be used as unseen samples and measure model performance for the trees where they are not used as training samples. OOB score is than computed as the number of correctly predicted samples from the out of bag sample, in regression case the oob score is computed using the R^2 score between true values and predicted values for OOB samples. The OOB error is calculated as the average error for predictions from the trees that do not contain in their respective bootstrap sample (oob samples are used).

R^2 score: indicates whether the model is a good fit for the data by measuring the proportion of the variance for target variable explained by the feature variables. $R^2 = 1- (RSS / TSS)$ RSS: sum of squares of residuals, and TSS is total sum of squares. The R^2 value is between 0 and 1, 1 means the model fit data perfectly and 100% of the variance in the target variable explained by the features used in the model.

Diamonds Data¶

```
Best RF Model OOB Score: 0.981448128437458
[ 473.22222222  388.26666667  430.97368421 ... 2643.73809524 2940.55263158
 2779.66666667]

Best RF Model Feature Importances:
carat feature importance:0.30199465230227157
depth feature importance:0.005615268543312274
table feature importance:0.004346108749685826
x feature importance:0.19090287772735992
y feature importance:0.2554629994561144
z feature importance:0.14922971129722173
cut feature importance:0.003289634910874443
color feature importance:0.031206885821142057
clarity feature importance:0.057951861192017756
```

Best RF Model Mean Test and Train RMSE and R2 scores after 10-fold CV:

	mean_train_RMSE	mean_validation_RMSE	mean_train_R2	mean_validation_R2
93	197.555441	715.111659	0.99751	0.644635

For best RF model diamond data, OOB score for 0.98. In Regression case R^2 scores are used to calculate OOB score. 0.98 OOB score is a good result. Avg. validation R^2 is 0.644 which is not very good.

Gas Data¶

```
Best RF Model OOB Score: 0.9024957564053923
[82.41471831 82.39423288 81.85991228 ... 89.10917722 65.38696364
 70.00585   ]

Best RF Model Feature Importances:
AT feature importance:0.3010705060339823
AP feature importance:0.04480777238527264
AH feature importance:0.0405105496218159
AFDP feature importance:0.0819215110852701
GTEP feature importance:0.0800383875929842
TIT feature importance:0.12693861634272424
TAT feature importance:0.07691051650405482
TEY feature importance:0.06028978245172857
CDP feature importance:0.048490805350535224
year feature importance:0.13902155263163204
```

Best RF Model Mean Test and Train RMSE and R2 scores after 10-fold CV:

	mean_train_RMSE	mean_validation_RMSE	mean_train_R2	mean_validation_R2
164	1.342304	6.762084	0.986713	0.496042

For best RF model gas data, OOB score for 0.90 is a good result. Avg. validation R^2 is 0.49 is not very good.

Project_4_Part2_Twitter_Data

Project 4 - Part 2 Twitter Data¶

Gorkem Camli (105709280)

Library imports¶

```
[nltk_data] Downloading package vader_lexicon to
[nltk_data]     /Users/gorkemcamli/nltk_data...
[nltk_data]   Package vader_lexicon is already up-to-date!
```

Read Data¶

Question 27¶

Report the following statistics for each hashtag, i.e. each file.

- 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

Since the time to read all files were taking,I solved this issue by reading files line by line and then only saving the columns I need for the questions 27 and 28. Below we can see the shape of each dataframe created for all 6 files:

```
gohawks (169122, 4)
gopatriots (23511, 4)
nfl (233022, 4)
patriots (440621, 4)
sb49 (743649, 4)
superbowl (1213813, 4)
```

Average number of tweets per hour: For the stats questions, the average number of tweets per hour can be defined different ways:

1 - Find number of tweets tweeted all the hours in data set, take their average.

2 - Total number of tweets / (Number of hours within the textfile's min and max date)

For these definitions the results can differ if there are hour timeframes where no tweet has been posted. (2) definition takes into account the hours that have no tweet, (1) does not. For example if the within the data timeframes there are 100 hours but 2 hours have no twitter post, (2) divides the total tweets by 100, however (1) divides by 98. The second approach makes more sense and accurate, and I choose this as definition. But since the question is not clear on which definition we are expected to go I include both of them

Average number of followers of users posting the tweets per tweet: Mean of followers_count column (['author']['followers'])

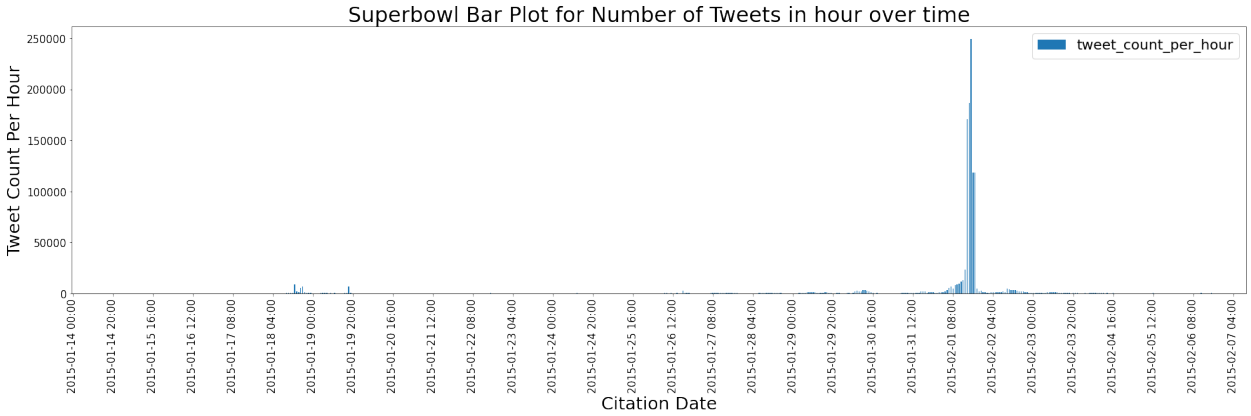
Average number of retweets per tweet: Mean of retweet_count column (['metrics']['citations']['total']) as every row represent a tweet.

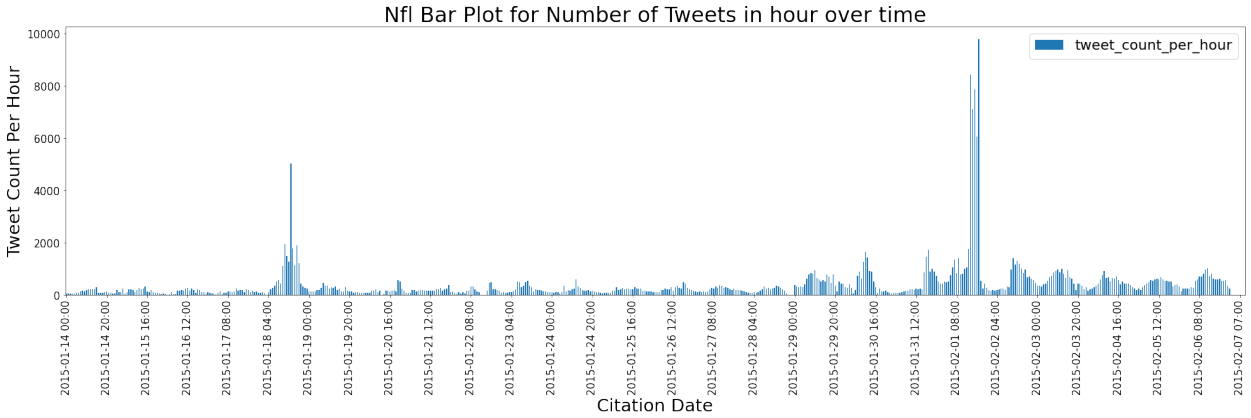
```
--- Stats for #gohawks ---
Average # tweet per hour (definition 1:) 296.705
Average # tweet per hour (definition 2:) 292.488
Average # followers 2217.924
Average # retweet per tweet 2.013


--- Stats for #gopatriots ---
Average # tweet per hour (definition 1:) 53.313
Average # tweet per hour (definition 2:) 40.955
Average # followers 1427.253
Average # retweet per tweet 1.408


--- Stats for #nfl ---
Average # tweet per hour (definition 1:) 399.695
Average # tweet per hour (definition 2:) 397.021
Average # followers 4662.375
Average # retweet per tweet 1.534


--- Stats for #patriots ---
Average # tweet per hour (definition 1:) 750.632
Average # tweet per hour (definition 2:) 750.894
Average # followers 3280.464
Average # retweet per tweet 1.785


--- Stats for #sb49 ---
Average # tweet per hour (definition 1:) 1384.821
Average # tweet per hour (definition 2:) 1276.857
Average # followers 10374.16
Average # retweet per tweet 2.527


--- Stats for #superbowl ---
Average # tweet per hour (definition 1:) 2067.825
Average # tweet per hour (definition 2:) 2072.118
Average # followers 8814.968
Average # retweet per tweet 2.391
```

Question 28¶

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.

Plots below show the number of tweets in hour for #superbowl and #nfl data in 1-hour bins. X axis ticks shows the corresponding day and hour for that bin. Since there are many bins and corresponding ticks for them to make the plot less crowded, I showed xticks in every 20 hours.





For superbowl hashtag we can see that the peak is in Feb 1st 2015 on the superbowl 49 game day. For nfl the biggest peak is again on superbowl day, Feb 1st but there are also smaller peaks happening on other days, which probably also corresponds game days. For example the first small peak, on Jan 18 2015 corresponds the day where Packers and Seahawks have game.

Question 29¶

Task Explanation¶

Follow the steps outlined below:

- Describe your task.
- Explore the data and any metadata (you can even incorporate additional datasets if you choose).
- Describe the feature engineering process. Implement it with reason: Why are you extracting features this way - why not in any other way?
- Generate baselines for your final ML model.
- A thorough evaluation is necessary.
- Be creative in your task design - use things you have learned in other classes too if you are excited about them!

Intro & Task Description:

In this question, I did character-centric time series tracking and a winner prediction task:

- Character-centric Time-Series Tracking part, intends to understand how tweet emotions changing during the Superbowl 49 game for hawks and patriots, as well as for American Football players playing in the game. So thee question I want to answer "Can we track the average perceived emotion across tweets about each player in the game across time in each fan base? Can we correlate these emotions with the game score in the given time interval?".

- Winner Prediction Task: Given tweet emotions expressed to players within a time window, can we predict who is winning the superbowl 49 in that time window? The emotions about player or the team is aggregated based on 1-minute time intervals. So, given the perceived emotion in any minute in the game, who is leading the game, the answer could be Tie, Patriots or Seahawks. I will do 3 class classification task with different emotion features on 3 models (Logistic Regression, Random Forest, Neural Networks -simple MLP-).

These two tasks are not trivial to solve. In the raw data:

- There is no information on which tweet belongs what player.
- There is no information about players or the game.
- We only have timestamp for tweets but we need to map these to the game time where significant events such as score occurs.
- There is no label for the prediction task. We don't know the game status in the tweets data.

In the next sections Data Exploration and Feature Engineering, I explained in details how I tackled these problems.

Data Used:

- '#gopatriots.csv', '#patriots.csv' and '#gohawks.csv' files.
- Additional data:
  - Players information: I collected team player names, positions, position category information for Patriots and Seahawks team members who played at Superbowl 49.
  - Game information: I used the score plays information to extract relative score times from tweets and label which team is the leading the game (patriots, seahawks, tie) when a tweet posted.
  - Emotion data: I used 4 different emotion feature set extracted from different libraries/models: **Nltk.sentiment.vader, TextBlob, CardiffNLP Roberta and AllenNLP Roberta.** I did both tasks with 4 set of features and compared the feature results as well. I chose nltk.sentiment.vader and Textblob as the baseline features, the features extracted with these methods are rather simple. Nltk vader[1] and Textblob are both rule-based sentiment analysis approaches. Whereas sentiment features extracted from Roberta (Robustly Optimized BERT Pre-training Approach) Models use advanced NLP deep learning models which promises better emotion features. CardiffNLP is especially interesting because it is pretrained on Twitter dataset.

Data Preparation¶

Set GAME_START and GAME_END times¶

```
GAME_START 2015-02-01 15:30:00-08:00
GAME_END 2015-02-01 19:06:00-08:00
```

Helper functions¶ (to prepare data extract features, feature engineering and prepare the dataset to be used in the tasks)

Prepare Twitter Data and Additional data¶ To be able to create the tasks mentioned above, I need to first prepare the data, but some of the informations I need is not available within the Twitter dataset provided.

- Find Extra data:
  - Finding players data for each team
  - Finding game related scores, and significant events

Players Data¶ I used players and american football positions data found from below resources. I also crosschecked the team players info from espn website and added the missing players. Also, I realized that some players such as Dan Connolly didn't play in 2015 superbowl 49 (according to ESPN), so I created a new column espn_team_list to keep track of the players who played/not played in SB49.

References:

- players data: https://www.pro-football-reference.com/boxscores/201502010sea.htm#all_team_stats
- positions data: https://www.rookieroad.com/football/positions/position-abbreviations/

The final combined players table preview:

```
(39, 9)
```

	Player	Pos	Team	espn_team_list	Abbreviation	Position	Category
0	Tom Brady	QB	Patriots	True	QB	Quarterback	Offense
1	Shane Vereen	RB	Patriots	True	RB	Running Back	Offense
2	Brandon LaFell	WR	Patriots	True	WR	Wide Receiver	Offense
3	Julian Edelman	WR	Patriots	True	WR	Wide Receiver	Offense
4	Rob Gronkowski	TE	Patriots	True	TE	Tight End	Offense

Raw Tweet Data¶ Read gohawks, go patriots and patriots data.

```
Creating tweet_dfs2...
gohawks (169122, 13)
gopatriots (23511, 13)
patriots (440621, 13)
```

Language distribution of the tweets for top 8 language: (und:undecided)

`gohawks`

	en	und	es	pt	de	fr	ja	tl
tweet.lang	0.841801	0.119198	0.011104	0.008976	0.003737	0.001963	0.001904	0.001898

`gopatriots`

	en	und	es	pt	de	fr	ht	it
tweet.lang	0.547999	0.18234	0.129131	0.089107	0.00906	0.008166	0.005614	0.004211

`patriots`

	en	und	es	pt	fr	de	ht	in
tweet.lang	0.898008	0.048788	0.029399	0.005792	0.004142	0.002921	0.001441	0.001203

Given that we will do sentiment analysis task, I will only use english language tweets.

Create Project Data¶ We need to incorporate the newly collected data with the tweets data. This is not a trivial step as we need to know (1) if a tweet is about a specific player, (2) mapping game time to real time. (1) is handled in this section and (2) has its own specific part since it is more complex.

In this step I prepared the final raw data to be used in the above described tasks:

Steps followed to prepare this data:

- Select features from raw tweet data
- Filter to have english language only
- Concat gohawks and gopatriots data in a single dataframe, put hashtag column to later use to separate them.
- Handle date time information for citation_date column.
- Game Status: create a column to specify when the tweet is posted: 'pre-game, during game' and 'after game'
- Map tweets and players
- Sentiment Analysis
  - Nltk Vader
  - Textblob
  - CardiffNLP Roberta
  - AllenNLP Roberta

Further details for some of the above steps:

Selected features from raw tweets data:

['title', 'tweet.text', 'citation_date','hashtag', 'tweet.lang']

Date time:

Twitter raw data has citation_date column as the tweet post time. This column values are in the form of UNIX time as a scalar number. I first convert this column to a human-readable format using datetime and converted the timezone to PST time zone. I also used PST time zone for significant times in the game such as game start and end time to be able to easily filter out tweets during game time. I also created additional year, month, day, hour columns for convenience.

Find if a tweet about a specific player:

In order to make a character-centric tracking we need to know whether a tweet is related to a player. There are several ways we can see a tweet could be related to a player: if tweet text mentions the player, uses hashtag about player and use the name of the player.

I used a simple logic with full name and partial name to match to find tweets about players. The results are case insensitive. By doing this I can identify players for 15% of the tweet data with 1 or more than 1 players out of 551K tweets. So, around 83K of tweets are matched with at least 1 player. This could be further improved by finding twitter usernames of the players (for mentions) and specific hashtags used for players. However, most of these already contains either full, first or last name of the players meaning most of these case would be handled already with the current logic. Several columns created: player (list of the players found), player_count (number of players identified in a tweet), player2 (same as player but with team names for tweets no player found).

What will happen to the remaining 85% of the tweets? Well, since I didn't want to discard them directly I assigned team names to the tweets that I couldn't find specific players.

Sentiment Analysis:

For sentiment analysis, we need sentiment information. 4 different approach used to extract sentiment information: nltk, textblob, CardiffNLP Roberta and AllenNLP Roberta. With nltk.vader, I found positive, negative, neutral and compound scores for each tweet. Assigned the labels using compound score. With TextBlob, I extracted polarity and sensitivity information and assigned the label using polarity information. CardiffNLP I extracted positive, neutral, negative scores and label. With AllenNLP, I extracted positive and negative scores as well as label. Since roberta model results don't have a compound score, I used the positive scores as the feature set (lower positive score would correspond negative emotion).

For the character centric analysis, we need to aggregate these emotion values, for each character and time interval. I described those steps in the later sections. I experimented extracting sentiment information with

both cleaned and uncleaned tweets, the change in the results were insignificant. I shared them later in the notebook. Also, applying these steps take considerable time, in order to not rerun the same things over and over, I saved the final df and reread when I restart working.

Preview of the project data:

```
Starting posting time datetime features...
(550932, 72)
```

	hashtag	title
0	#gohawks	“@TheDA53: “@nathanSD8: @TheDA53 broncos?! #...
1	#gohawks	Dr. Jim Kurtz & I before Seahawks vs Panth...
2	#gopatriots	NFL PLAYOFFS: Brady throws 3 TDs in win over R...

Data Exploration & Feature Engineering¶

Answer some questions¶

- **Check if data is balanced for each team?**

```
#gopatriots    408565
#gohawks       142367
Name: hashtag, dtype: int64
```

```
#gopatriots    0.741589
#gohawks       0.258411
Name: hashtag, dtype: float64
```

We have highly inbalanced data, gopatriots tweets corresponds to the 74% of the data.

- **Find in how many tweets each player is mentioned?**

player2	title
#gopatriots	357785
#gohawks	115377
Tom Brady	32383
Russell Wilson	8710
Rob Gronkowski	5395
Marshawn Lynch	4845
Richard Sherman	4314
Chris Matthews	2753
Julian Edelman	2439
Earl Thomas	2130
LeGarrette Blount	1948
Doug Baldwin	1541
Darrelle Revis	1342
Kam Chancellor	1010
Vince Wilfork	949
Brandon LaFell	868
Michael Hoomanawanui	737
Jermaine Kearse	646
Bryan Walters	585
Chandler Jones	507
Danny Amendola	491
Jamie Collins	471

player2	title
Tony McDaniel	430
Kyle Arrington	405
Shane Vereen	394
Kevin Williams	367
Bruce Irvin	316
Brandon Browner	255
Bobby Wagner	237
Cliff Avril	191
Devin McCourty	163
Patrick Chung	163
K.J. Wright	141
Dont'a Hightower	132
Ricardo Lockette	122
Michael Bennett	118
Byron Maxwell	88
Rob Ninkovich	71
DeShawn Shead	57
Sealver Siliga	45
Robert Turbin	11

- **What is the ratio of number of players found on tweet dataset?**

Show player_count column's normalized value counts.

```
0     0.858839
1     0.116221
2     0.018329
3     0.005231
4     0.001049
5     0.000231
6     0.000069
7     0.000015
8     0.000011
9     0.000004
10    0.000002
Name: player_count, dtype: float64
```

Cleaning Tweets:¶ I did a minimum cleaning on the tweets to experiment on whether text cleaning will further change the emotions distributions for tweets, and help with the model performances.

Cleaned title column: cleans the links and mentions from the tweet. Make all characters lower case.

Cleaned hashtag title column: in addition to links and mentions, also cleans the hashtags from the tweet. Make all characters lower case.

Creating Sentiment Scoring and Label Features¶

nltk.sentiment.vader¶ As my first baseline sentiment analysis score approach, I used nltk.sentiment.vader.

VADER stands for Valence Aware Dictionary for Sentiment Reasoning. Vader is a model used for text sentiment analysis that is sensitive to both polarity (positive/negative) and intensity (strength) of emotion and one can apply Vader model directly to unlabeled text to get scores for negative, neutral and positive sentiments. Nltk.sentiment.vader also provides a compound score: "is computed by summing the valence

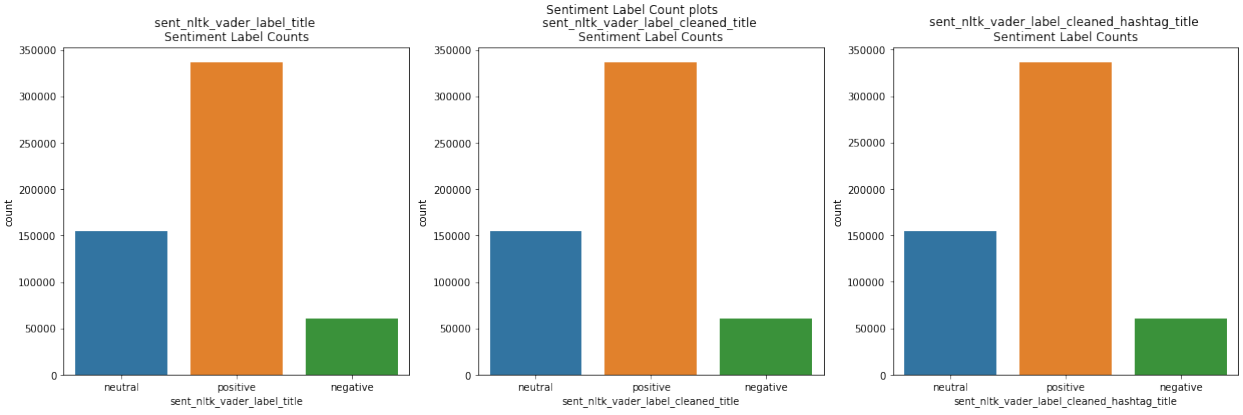
scores of each word in the lexicon, adjusted according to the rules, and then normalized to be between -1 (most extreme negative) and +1 (most extreme positive). This is the most useful metric if you want a single unidimensional measure of sentiment for a given sentence. Calling it a 'normalized, weighted composite score' is accurate."

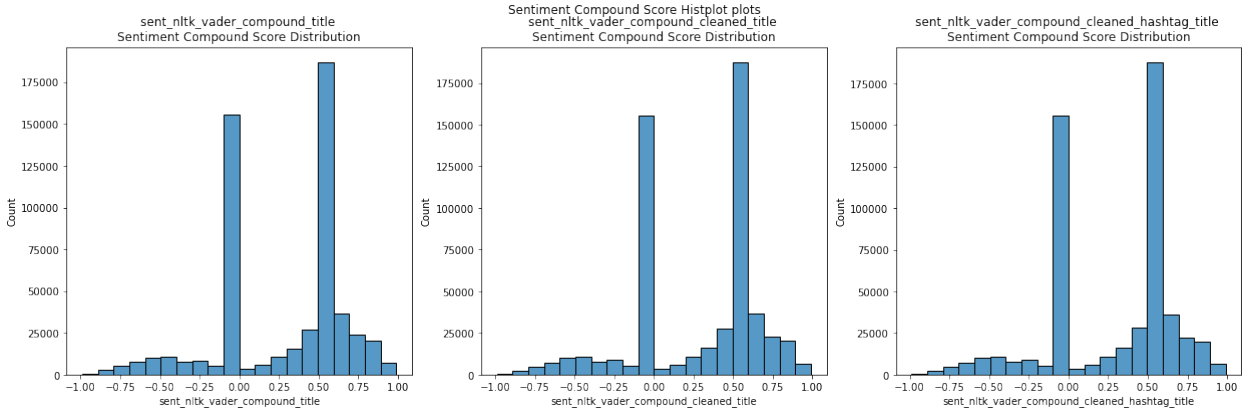
To assign positive, neutral and negative labels, I used the compound score and recommended thresholds in the nltk documentaion: https://github.com/cjhutto/vaderSentiment#about-the-scoring

- positive sentiment: compound score >= 0.05
- neutral sentiment: (compound score > -0.05) and (compound score < 0.05)
- negative sentiment: compound score <= -0.05

I applied Vader model to raw tweets data, cleaned_title and cleaned_hashtag_title columns. When we grouped and check what the ratios for each label, I don't see any huge shift in the overall distribution in emotion for the raw vs cleaned tweets.

```
title

positive    0.610240
neutral     0.280111
negative    0.109649
Name: sent_nltk_vader_label_title, dtype: float64

cleaned_title

positive    0.610313
neutral     0.280203
negative    0.109484
Name: sent_nltk_vader_label_cleaned_title, dtype: float64

cleaned_hashtag_title

positive    0.609850
neutral     0.280681
negative    0.109469
Name: sent_nltk_vader_label_cleaned_hashtag_title, dtype: float64
```

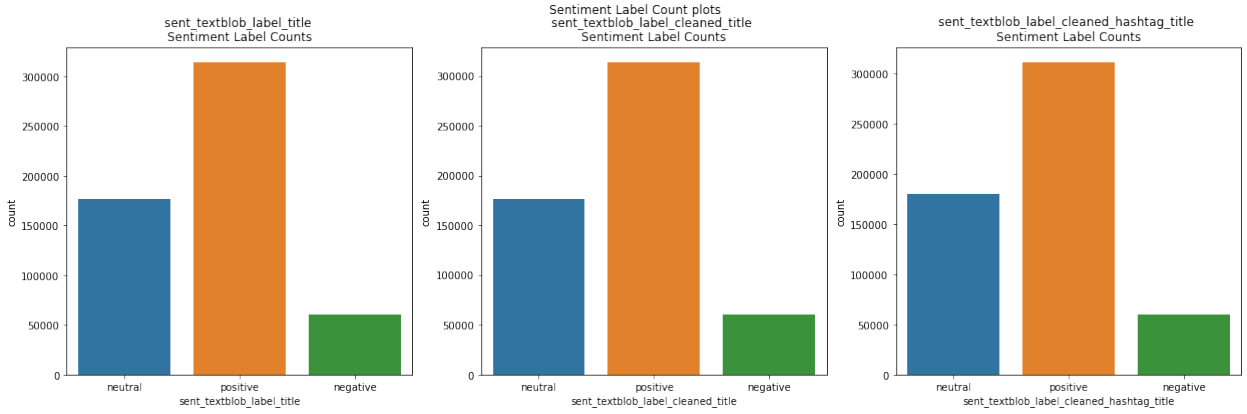


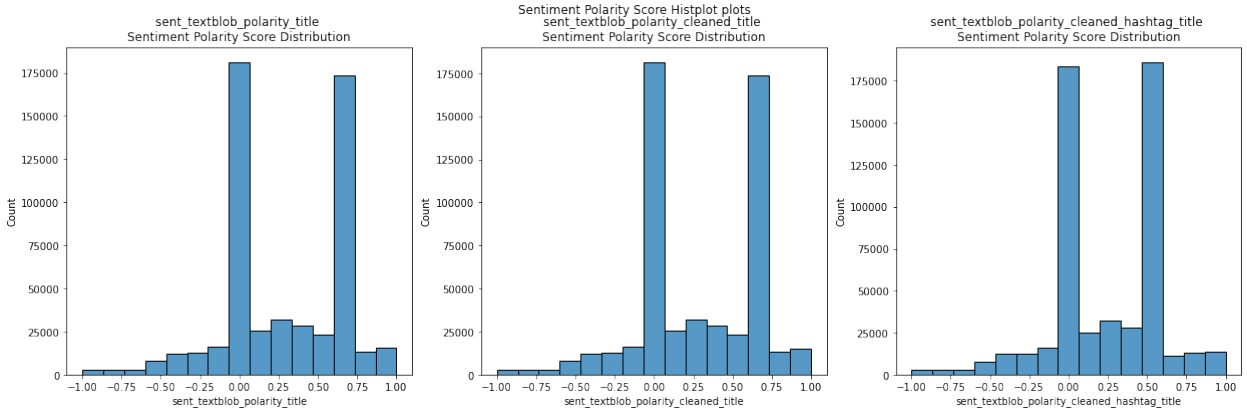


We can see that majority of the labels assigned to positive tag, 60-61%. The compound score has two peaks one at 0 and the other at around 0.5.

TextBlob sentiment¶

```
title

positive    0.569557
neutral     0.320936
negative    0.109507
Name: sent_textblob_label_title, dtype: float64

cleaned_title

positive    0.569103
neutral     0.321288
negative    0.109609
Name: sent_textblob_label_cleaned_title, dtype: float64

cleaned_hashtag_title

positive    0.564556
neutral     0.326089
negative    0.109355
Name: sent_textblob_label_cleaned_hashtag_title, dtype: float64
```





We can see that majority of the labels assigned positive label again, but percentage of positive labels are lower compared to nltk. The polarity distribution is similar to Nltk as it has 2 main peaks, though nltk seems to assign higher scores on positive tweets whereas in this one 0.0-0.75 range has higher bars.

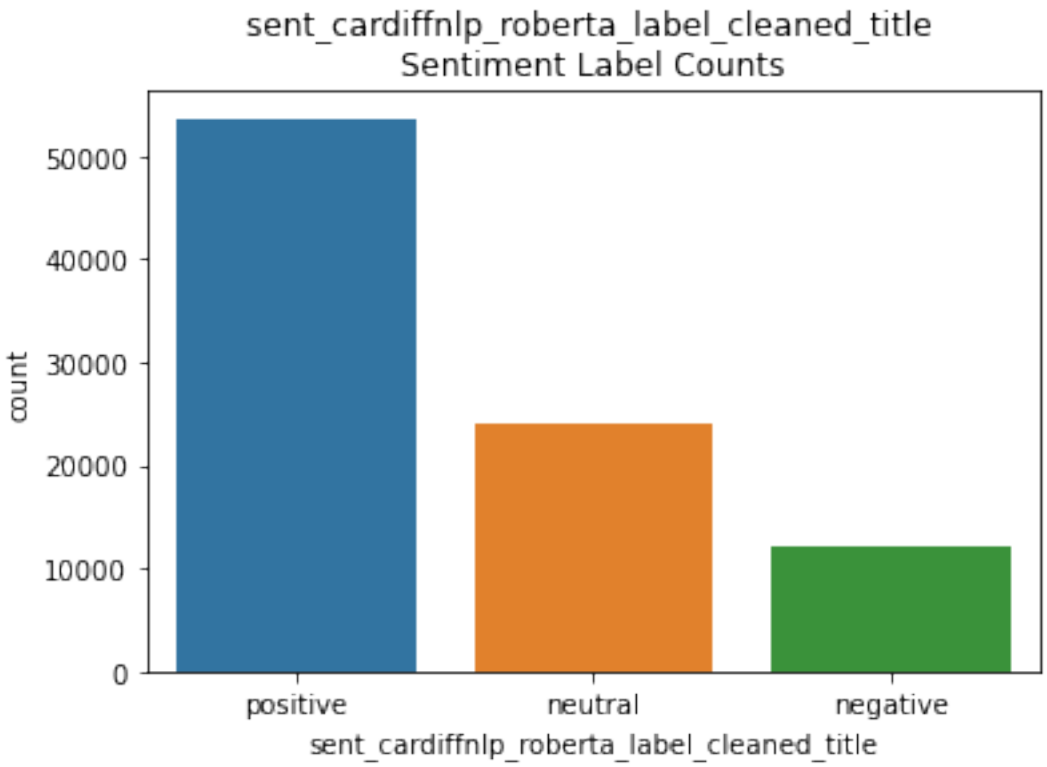
Cardiff NLP Roberta sentiment¶ The code I used to extract features with CardiffNLP Roberta model is in another notebook called "P4_Roberta.ipynb".

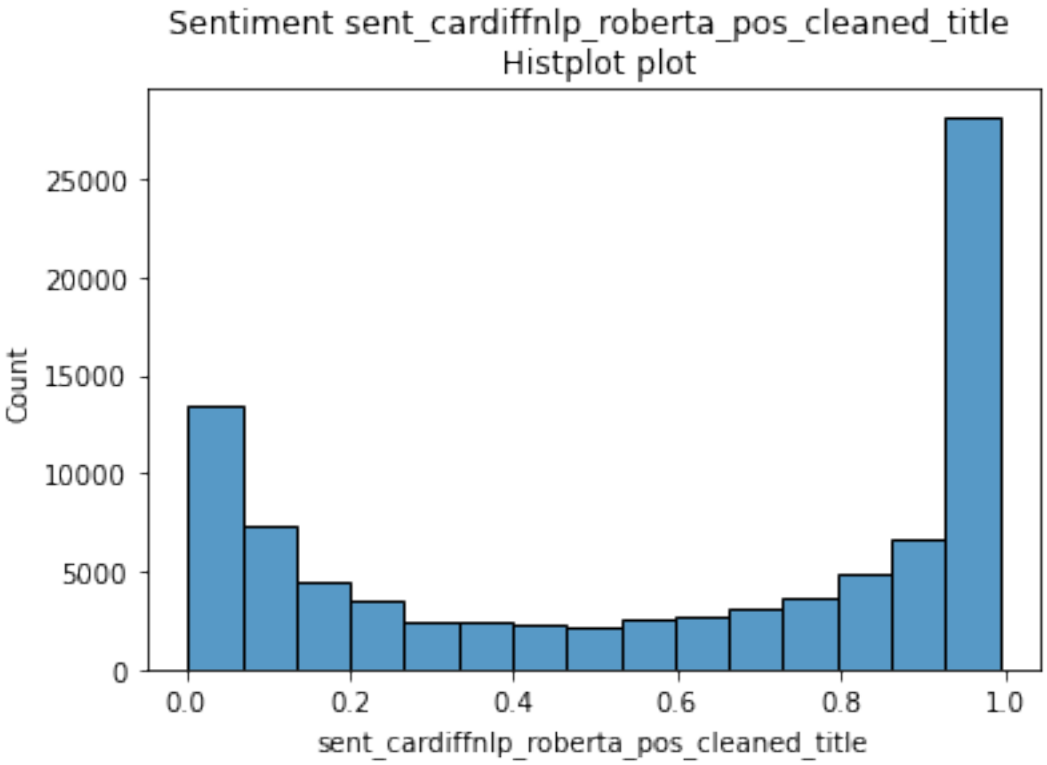
I already extracted the sentiment scores and label from CardiffNLP Roberta using mentioned notebook and saved the results in a csv file. This model is particularly interesting because it is pretrained on Twitter dataset, which I am expecting to see more accurate sentiment score results. 4 columns extracted: positive, neutral, negative scores and label. Since model didn't provide a compounds score I used the positive score in the tracking and prediction task.

I only extracted features for the tweets posted during SB49 game, approx. 90K tweets. Otherwise I have around half million tweets and it was taking too much time to extract them. Also, all my exploration will be mainly focused during game, so this is the most essential information I needed for character-centric time-series tracking and my prediction task. As I didn't see any radical sentiment score changes with the cleaning tweet, I extracted the sentiment results for cleaned_title column only.

```
positive    0.597334
neutral     0.267315
negative    0.135350
Name: sent_cardiffnlp_roberta_label_cleaned_title, dtype: float64
```

Positiive labels again takes the majority of the tweets. We have the lowest neutral label percentage and the highest negative percentage within the 3 first methods. This is a good indication that the extracted features have more polarity.





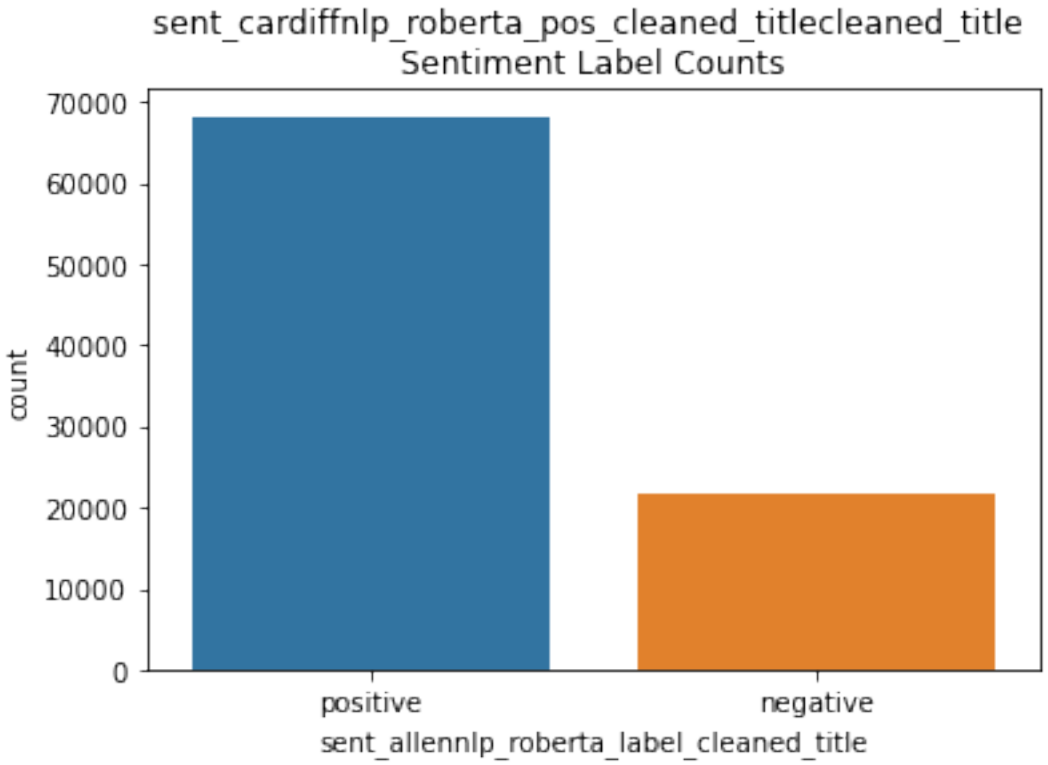
We see there is a lot of tweets that assigned either 1 or very close to 1, and the second peak is towards 0. The polarity between both sides are the most significant so far with the CardiffNLP features.

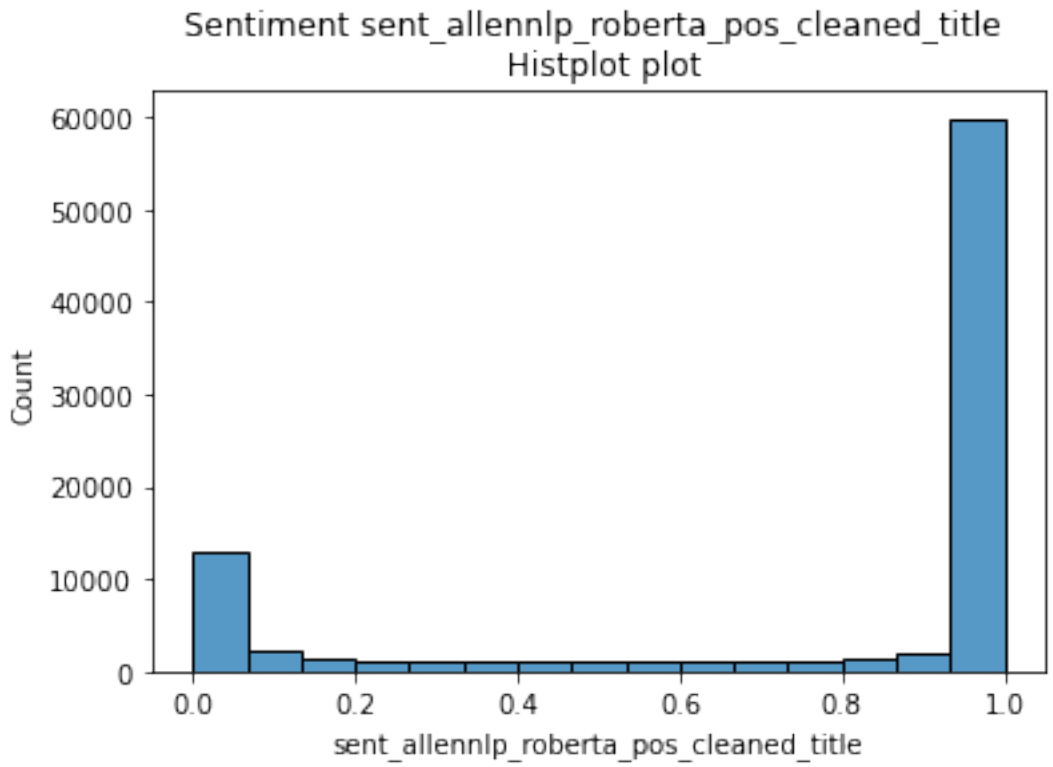
Allen NLP Roberta sentiment¶ The code I used to extract features with AllenNLP Roberta model is in another notebook called "P4_Roberta.ipynb". I already extracted the sentiment scores and label from AllenNLP Roberta using that notebook and saved the results in a csv file. I expect to have an accurate results and good features from this model too. I extracted positive and negative scores and labels.

Similar to CardiffNLP, I only extracted features for the tweets posted during SB49 game and only on cleaned_title. Otherwise I have around half million tweets and it was taking too much time to extract them.

```
positive    0.758964
negative    0.241036
Name: sent_allennlp_roberta_label_cleaned_title, dtype: float64
```

This is a bit harder to compare with other 3 methods as it doesn't have neutral label. Neutral labels in previous features seem to be shared with positive and negative labels. We see the highest negative tweet percentage in all 4 methods.





The polarity is even higher in here. This is a bit concerning, because 75% of the tweets are assigned positive and there the distribution for lower scores seem almost flat and very low. The emotion information for lower scored tweets, might have almost no effect when we aggregate them and get lost within positive scores. Also, it is very interesting that CardiffNLP Roberta and AllenNLP Roberta results are very different even in terms of disitribution. Both provides more polarity and less neutral emotion, but the distribution in CardiffNLP one seems to be more varied whereas AllenNLP one is mostly seem flat except 0 and 1.

Explore Sentiments Examples:¶

```
sent_nltk_vader_compound Top 3 Positive Titles
Ex 1: Score: 0.9906 - Tweet: #patriots win. wow wow wow wow wow wow wow omg wow
wow. #superbowl
Ex 2: Score: 0.987 - Tweet: yesssss, #patriots #win #superbowi !!! great game,
great #halftimeshowkatyperry great sunday!! :-)
happy, happy, happy!! :-) :-) :-)
Ex 3: Score: 0.9843 - Tweet: congratulations  you deserve/earn it! you won!
brady the best! god bless! peace &amp;amp; love! xoxo!
#patriots #brady #superbowl2015!!!


sent_textblob_polarity Top 3 Positive Titles
Ex 1: Score: 1.0 - Tweet: happy friday!! #seahawks #superbowlxlix #gohawks
#nfl #seagals #deflategate
Ex 2: Score: 1.0 - Tweet: best dressed #seahawksfan on #superbowlsunday?
done and done.  #gohawks #12fan #beastmode
Ex 3: Score: 1.0 - Tweet:   a perfect topper to my day. glorious. #gohawks


sent_cardiffnlp_roberta_pos Top 3 Positive Titles
```

```
Ex 1: Score: 0.99376816 - Tweet: i love you so much! thank you for this amazing
experience!  #blessed #gohawks #seahawks
#12thcouple…
Ex 2: Score: 0.9936446 - Tweet: that's my boo  you are beautiful! ahhhh love you!
#gohawks #sb49
Ex 3: Score: 0.9935559000000002 - Tweet:  it was awesome! beautiful! &lt;3 you!! #sb49
#gohawks #12s


sent_allennlp_roberta_pos Top 3 Positive Titles
Ex 1: Score: 0.9999048709869384 - Tweet:  give the strength to the #patriots
Ex 2: Score: 0.9999042749404908 - Tweet: we won ! #patriots
Ex 3: Score: 0.9999040365219116 - Tweet: beautiful #patriots


sent_nltk_vader_compound Top 3 Negative Titles
Ex 1: Score: -0.9897 - Tweet: hell of a game. hell of a finish. hell of an
adjustment. hell of a fight. hell of a comeback.
hell of a situation. hell of a team. #gohawks
Ex 2: Score: -0.9855 - Tweet: no no no no no no no no no no no no no no no no no no
you can do this  give them something to cry about
#superbowlxlix #gohawks
Ex 3: Score: -0.9812 - Tweet:  ooohhhh, kill 'em, doug! haters gonna hate, hate,
hate, hate, hate #gohawks


sent_textblob_polarity Top 3 Negative Titles
Ex 1: Score: -1.0 - Tweet: they hate us cuz they ain't us! #patriots
Ex 2: Score: -1.0 - Tweet: game time!! let's go #patriots!!!!
Ex 3: Score: -1.0 - Tweet: the  are boring.  bring out the   #sbmediaday
#gohawks


sent_cardiffnlp_roberta_neg Top 3 Negative Titles
Ex 1: Score: 0.9836120999999999 - Tweet: i hate everything about the #patriots*, it
actually makes me angry just seeing them in this
game
Ex 2: Score: 0.9834545 - Tweet: im gonna puke this is fuckng stupid #patriots
getting jobbed again
Ex 3: Score: 0.98309845 - Tweet: so irritated that i don't get to enjoy the game.
instead i get to go to this stupid class for cps.
#ijustwannawatchfootball #gohawks


sent_allennlp_roberta_neg Top 3 Negative Titles
Ex 1: Score: 0.9997636675834656 - Tweet: second three-and-out for #patriots, who are
looking a bit uninspired.
Ex 2: Score: 0.9997625946998596 - Tweet: argh, stupid stupid foul. bad choice, guy.
#patriots
Ex 3: Score: 0.9997610449790956 - Tweet: this is a very bad place for the #patriots to be.
```

Examples mostly seem to make sense. Also some of the examples are extremely small or complicated to predict. For example: Nltk.vader's first negative example: 'hell of a game. hell of a finish. hell of an

adjustment. hell of a fight. hell of a comeback. hell of a situation. hell of a team. #gohawks ' might be wrongly predicted. Depending on what time of the game it is posted this could be interpreted as either positive or negative.

Mapping Game Time to Real Time:¶ There are 3 main time information we need to find:

1. Categorize tweets as pre, during and after game.
2. Find quarter and halftime show beginning and end times.
3. Find score times.
4. Find significant event times (interceptions, fumbles).

One of the biggest challenge I had is to map the real time with the game time. Game scores and significant events data available on the internet is relative to the game time such as first touch down is happend in second quarter 9:47 (example: https://www.espn.com/nfl/playbyplay/_/gameId/400749027 check all plays and scoring plays table). However, tweets are based on real time.

Unfortunately, game clock isn't kept counting down continuously during game. Even though game is considered 1 hour with each quarter 15 minutes, I realized that the actual game took generally 2-3 hours. In the superbowl 49 case, it took 3h 36 minutes as written on the internet. Hence, I need to find a way to map the game time and real time where tweets are written.

I tried several things on how to assign time frames. My first approach was to find timestamps of the game (either significant event or scores), but after hours of search it lead me nowhere:

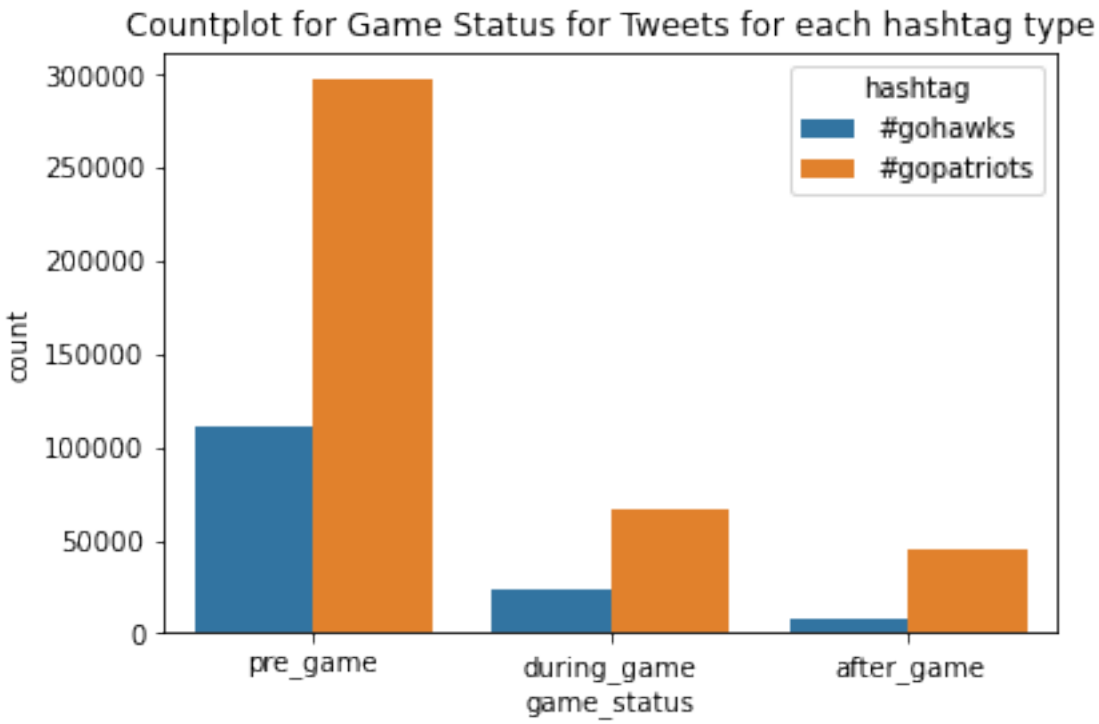
- I couldn't find any data online that is relative to real timestamps like PST, UTC on neither on quarter start and end, nor for significant or score times. Every play by play and score play data I found were on relative to the game time shown minutes within quarters.
- Checked youtube videos of the full game, to extract times manually, either by checking time on the screen or find it relative from video time. In full game video, there is no real time shown on the screen and I realized that the video was shortened where halftime shows removed, so that didn't helped either on mapping the game time to real time.
- I tried to find timestamps from the tweets of official twitter accounts of Seattle Seahawks and Patriots. I check both dataset and then twitter but the tweets from there limited and you can't infer any of the information completely.
- Final resort: refer score and quarter times from tweet dataset. Assumption: if a tweet with score is tweeted than either one of the team is already scored, use the earliest one within the game timeframe.
- As a last resort, I decided to extract the score times and quarter times from the tweets.

PS: I skipped the (4) Find significant event times (interceptions, fumbles), since there is no fumbles in the game and finding interceptions from tweets both very difficult and not very reliable.

1. Categorize tweets as pre, during and after game:¶ We already know the game start time 3:30 PST and how long game lasted 3 hours 36 min. By using start and game end time we can assign pre-game, during game and after game tags to each tweet. The distribution of game status can be seen below:

```
pre_game       408154
during_game     89804
after_game      52974
Name: game_status, dtype: int64
```

```
pre_game       0.740843
during_game    0.163004
after_game     0.096153
Name: game_status, dtype: float64
```



Most of the tweets were posted before game, so American Football fans, especially Patriots and SeaHawks are really fans and they watch the game during game, instead of posting tweet about it.

Distribution of tweets by each fan and game status:

```
hashtag      game_status
#gohawks     after_game       7761
             during_game     23706
             pre_game       110900
#gopatriots  after_game      45213
             during_game     66098
             pre_game       297254
Name: title, dtype: int64
```

2.1. Finding Quarter Times:¶ To go one step further, I used the tweet information to find approximate quarter times.

For each quarter, I checked the tweets that has 'end of' and quarter name in the same tweet. For example for first quarter, I filtered out only tweets that has 'end of' and either '1st quarter' or 'first quarter' substrings.

The results are not too many, so I manually checked the meanings and assigned the earliest tweet that refers to the end of quarter. In this case, the results are driven from users and approximate, but the best mapping I could find.

```
Looking matches for first quarter|1st quarter ...
```

	title	citation_date_fixed
425143	0-0 at the end of the first quarter. #Seahawks...	2015-02-01 15:58:08-08:00
425148	#SB49 end of first quarter. #Seattle 0 #Patrio...	2015-02-01 15:58:09-08:00
425264	At the end of the first quarter the game is ti...	2015-02-01 15:58:26-08:00
425289	It's scoreless between the #Patriots & #Seahaw...	2015-02-01 15:58:29-08:00
425291	At the end of the 1st quarter, score is 0-0! #...	2015-02-01 15:58:29-08:00

	title	citation_date_fixed
425560	At the end of a speedy first quarter, #SuperBo...	2015-02-01 15:59:02-08:00
425561	At the end of a speedy first quarter, #SuperBo...	2015-02-01 15:59:02-08:00
425589	And thatâ€™s the end of the 1st quarter. 0-0. ...	2015-02-01 15:59:05-08:00
425611	Alright, whoever had 0-0 at the end of the 1st...	2015-02-01 15:59:08-08:00
425722	At the end off the first quarter yhe score is ...	2015-02-01 15:59:25-08:00
425867	At the end of the 1st quarter Seattle #Seahawk...	2015-02-01 15:59:52-08:00
426071	And that's the end of the 1st quarter. 0-0 #Se...	2015-02-01 16:00:28-08:00
426073	And that's the end of the 1st quarter. 0-0 #Se...	2015-02-01 16:00:28-08:00
426609	0 0 end of 1st quarter #SuperBowlXLIX who gonn...	2015-02-01 16:01:52-08:00
433033	I lasted until the end of the 1st quarter unti...	2015-02-01 16:13:03-08:00

`Looking matches for second quarter|2nd quarter ...`

	title	citation_date_fixed
444742	agreement btwn #Seahawks & #Patriots: make...	2015-02-01 16:35:29-08:00
461206	@DJAelita @TheFaithBreaker 14 - 14 at the end ...	2015-02-01 17:09:09-08:00
468326	I swear the #SB49 is paid to win.. We were pla...	2015-02-01 17:46:58-08:00
476059	#patriots have to show the same guts the #Seat...	2015-02-01 18:11:09-08:00
538569	Last play last nite was a bad call cuz it didn...	2015-02-02 07:33:55-08:00

`Looking matches for third quarter|3rd quarter ...`

	title	citation_date_fixed
476087	The #Seahawks lead the #Patriots, 24-14, at th...	2015-02-01 18:11:15-08:00
477642	At the end of the 3rd quarter, the #Seahawks h...	2015-02-01 18:18:36-08:00
477644	24-14 at the end of 3rd quarter. Seahawks lead...	2015-02-01 18:18:37-08:00

`Looking matches for fourth quarter|4th quarter ...`

	title	citation_date_fixed
475939	At the end of the 4th quarter of #SuperBowlXLI...	2015-02-01 18:10:43-08:00
483169	You HAVE to have a big lead on the #Patriots a...	2015-02-01 18:39:49-08:00
486112	Made it just for the end of the fourth quarter...	2015-02-01 18:48:30-08:00
489895	Coming from behind in end of 4th quarter is wh...	2015-02-01 18:53:28-08:00
532798	Seahawks last great play near the end of the 4...	2015-02-02 00:15:37-08:00
536105	So... Yes, stupid call at the end of the game....	2015-02-02 05:23:54-08:00

2.2. Finding Halftime Show Times:

I used a similar approach above and used halftime and various strings for Katty Perry who performed at the half time: 'Katty Perry|@kattyperry|katty'

`Check Head 10 of the found tweeets:`

	title	citation_date_fixed
459695	Game tied at the half! Who will win? #SuperBow...	2015-02-01 17:05:02-08:00
459764	Well it's halftime and the game is tied at 14 ...	2015-02-01 17:05:09-08:00
459771	so i catch the game shortly before halftime. ...	2015-02-01 17:05:10-08:00
459802	Shit it's halftime, what channel is the blowjo...	2015-02-01 17:05:16-08:00

	title	citation_date_fixed
459810	Super Bowl 49 #patriots #brady #halftime #icec...	2015-02-01 17:05:17-08:00
459811	#katyperry #halftime \n#patriots — watching Su...	2015-02-01 17:05:17-08:00
459869	Wow exciting superbowl 2 great teams 2 big QB,...	2015-02-01 17:05:26-08:00
459887	That's how you go to halftime. #GoHawks	2015-02-01 17:05:28-08:00
459893	What a game halftime great #SuperBowlXLIX #Pat...	2015-02-01 17:05:30-08:00
459898	Super Bowl halftime shows are a turd oasis wit...	2015-02-01 17:05:31-08:00

```
Check Tail 10 of the found tweeets:
```

	title	citation_date_fixed
462907	Bad ass halftime show! @katyperry @pepsi #SBXL...	2015-02-01 17:18:39-08:00
462912	@katyperry is just incredible, I'm in love #ha...	2015-02-01 17:18:40-08:00
462913	So far my favorite halftime show ever. #katype...	2015-02-01 17:18:40-08:00
462939	Loving the halftime #GoHawks	2015-02-01 17:18:50-08:00
462952	I'm quite enjoying KP halftime show\n\n#GoHawks	2015-02-01 17:18:55-08:00
462959	This halftime show is so damn good! Katy Perry...	2015-02-01 17:18:59-08:00
462995	You're gonna hear me ROAR katyperry #katyperry...	2015-02-01 17:19:20-08:00
463020	I'm so glad Katy Perry is doing halftime... Be...	2015-02-01 17:19:35-08:00
463034	@katyperry @LennyKravitz Awesome halftime show...	2015-02-01 17:19:46-08:00
463058	Missy made a comeback! #superbowl #halftime #p...	2015-02-01 17:19:55-08:00

I also explored other combinations for 'halftime' and 'started|start' and 'halftime and end|ended' tweet matches to further investigate. Also, from Google and youtube videos, I found that the halftime show lasted 12 minutes 35 seconds. I incorporated this information with some buffer time as well.

Again the times are rather approximate, but this will at least help us in approximately when in the game, these tweets are being posted and put the information conceptually in place. Final time intervals I decided are below:

```
first_qt_start : 2015-02-01 15:30:00-08:00
first_qt_end : 2015-02-01 15:58:00-08:00
second_qt_start : 2015-02-01 15:58:00-08:00
second_qt_end : 2015-02-01 17:09:00-08:00
half_time_start : 2015-02-01 17:09:00-08:00
half_time_end : 2015-02-01 17:27:00-08:00
third_qt_start : 2015-02-01 17:27:00-08:00
third_qt_end : 2015-02-01 18:18:00-08:00
fourth_qt_start : 2015-02-01 18:18:00-08:00
fourth_qt_end : 2015-02-01 19:06:00-08:00
```

Distribution of tweets by each fan and game detailed status:

```
hashtag      game_status_detailed
#gohawks     after_game               7755
             first-quarter            3726
             fourth-quarter           4642
             half-time show            900
             pre_game                110900
             second-quarter           8208
             third-quarter            6236
#gopatriots  after_game              45203
             first-quarter           13210
             fourth-quarter          15808
```

```
half-time show            1921
pre_game                297254
second-quarter           27867
third-quarter             7302
Name: title, dtype: int64
```

The final time mapping missing is to identifying the times where scores changed. I address this part on feature engineering section.

3. Finding Score Times:¶ I know the scores of the game, so with regex I searched the tweets where scores shared, and from those tweets I filtered and kept only the correct scores realized within the duration of the game (see scores_list).

scores_list = ['7-0','7-7','14-7','14-14','14-17','14-24','21-24','28-24']

This is especially important because there are some tweets with prediction scores of the game. Also, according to american footbal rules, once touchdown happens a team gets 6 points and then with extras that score can increase up to 1 or 2 more points. Some tweets were posted without the extra points (ie 14-23) so scores*list helped me filter out those as well since I want to be consistent with the Scoring Plays results I dounf in espn website (https://www.espn.com/nfl/playbyplay//*gameId/400749027). I also limited the post date of the tweets after game start time to make sure to get real scores.

Then, my assumption is that if a user tweets the score, then it means there is a change in score and one of the team has scored. And I could use the relative real time of the score by taking the time of the earliest tweet posted with that score. In this way for each score, I can have a relative real tweet times in PST time where each score happened. Of course, the earliest tweet might be posted couple minutes later and might not be exact with the score time. But this is the best approximation and map I could get with the available data I have.

As a sanity check I also looked whether the found time frames fall in the relative place in the quarter the score happened. You can see the final dataframe with score and time mapping below:

	score	citation_date_fixed	leading	quarter
0	0-0	2015-02-01 15:30:00-08:00	Tie	1st Quarter
0	7-0	2015-02-01 16:12:00-08:00	Patriots	2nd Quarter
1	7-7	2015-02-01 16:33:59-08:00	Tie	2nd Quarter
2	14-7	2015-02-01 16:48:10-08:00	Patriots	2nd Quarter
3	14-14	2015-02-01 16:58:56-08:00	Tie	2nd Quarter
4	14-17	2015-02-01 17:39:21-08:00	Seahawks	3rd Quarter
5	14-24	2015-02-01 17:54:34-08:00	Seahawks	3rd Quarter
6	21-24	2015-02-01 18:28:17-08:00	Seahawks	4th Quarter
7	28-24	2015-02-01 18:30:34-08:00	Patriots	4th Quarter

This table shows approximately at what time the score happened according to the tweets, in which quarter and who is leading the game after the score. I will use this data frame to table my data for the prediction task. Real scoring plays relative to game time can be found here: https://www.espn.com/nfl/playbyplay/_/gameId/400749027

Character-centric time-series tracking¶

Can we track the average perceived emotion across tweets about each player in the game across time in each fan base? Can we correlate these emotions with the game score in the given time interval?

In this part, I used the cleaned title column, with 4 different emotion features. The analysis is only done the tweeets that are tweeted during the game.

To find time based perceived emotion for each player on each fan base, I aggregated data over hashtag, player and time interval and take the mean of the selected emotion feature score. I used hashtag for each fan base, by making the assumption that if user post a tweet using #gopatriots, #patriots and #gohawks hashtag they are supporting corresponding team. I also included #patriots mainly because the tweets for #gopatriots were very low. The ratio of #gohawks (170k) to #gopatriots (24K) was 92-8%. Adding #patriots tag increased the total tweets to 550K and changed the balance to 75% patriots to 25% hawks. Using #patriots hashtag and making the assumption that only supporters/fan uses #patriots hashtag is less likely to be true, however, this change helped me find more players in more timeframes and reduced the data sparsity substancially.

For time intervals, I used different ranges 1,2,5,10 minute intervals. This is more related on the accuracy and granularity we are interested in. According the play by play and scoring play tables, put/interception and touchdowns happin 2-5 minute intervals. Aggregating the data too much might dillute the peaks in the important moments. For other plots, we are more interested in general pattern to compare it for different features/playeers etc, to make plots more consumable, I increased the time interval used on those plots. Time interval used either mentioned on x axis or title of the plots/subplots.

¶

Sample Table showing how rounded 1-minute time interval looks like for 5 samples:

	citation_date_fixed	post_time_1_intervals
0	2015-01-14 00:04:41-08:00	2015-01-14 00:04:00-08:00
1	2015-01-14 00:05:50-08:00	2015-01-14 00:05:00-08:00
2	2015-01-14 00:07:18-08:00	2015-01-14 00:07:00-08:00
3	2015-01-14 00:09:58-08:00	2015-01-14 00:09:00-08:00
4	2015-01-14 00:12:20-08:00	2015-01-14 00:12:00-08:00

Image shows the scoring summary of the game shown in ESPN website:

Scoring Summary

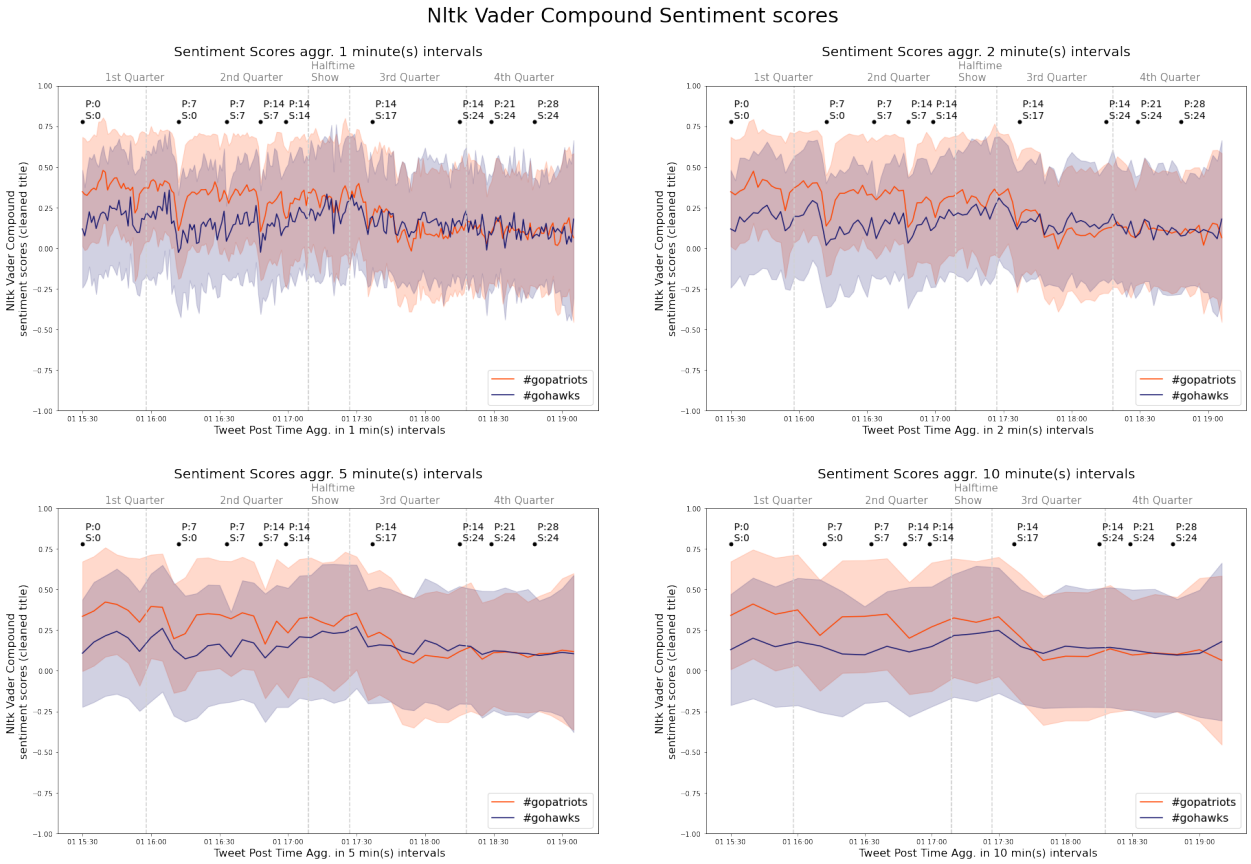
SECOND QUARTER			NE	SEA
TD	9:47	Brandon LaFell 11 Yd pass from Tom Brady (Stephen Gostkowski Kick) *9 plays, 65 yards, 4:10*	7	0
TD	2:16	Marshawn Lynch 3 Yd Run (Steven Hauschka Kick) *8 plays, 70 yards, 4:51*	7	7
TD	0:31	Rob Gronkowski 22 Yd pass from Tom Brady (Stephen Gostkowski Kick) *8 plays, 80 yards, 1:45*	14	7
TD	0:02	Chris Matthews 11 Yd pass from Russell Wilson (Steven Hauschka Kick) *5 plays, 80 yards, 0:29*	14	14
THIRD QUARTER			**NE**	**SEA**
FG	11:09	Steven Hauschka 27 Yd Field Goal *7 plays, 72 yards, 3:51*	14	17
TD	4:54	Doug Baldwin 3 Yd pass from Russell Wilson (Steven Hauschka Kick) *6 plays, 50 yards, 3:13*	14	24
FOURTH QUARTER			**NE**	**SEA**
TD	7:55	Danny Amendola 4 Yd pass from Tom Brady (Stephen Gostkowski Kick) *9 plays, 68 yards, 4:15*	21	24
TD	2:02	Julian Edelman 3 Yd pass from Tom Brady (Stephen Gostkowski Kick) *10 plays, 64 yards, 4:50*	28	24

ref: https://www.espn.com/nfl/game/_/gameId/400749027

Fans' emotion overtime aggregated on different time intervals¶

I start with comparing different time intervals. Below plots show emotion of each fan group over 1,2,5 and 10 minute(s) intervals. X-axis shows the time during game (Feb 1st 15:30 to 19:00), y-axis shows the nltk.vader compound score applied on cleaned title. The aggregation is made taking mean of the compound score for each hashtag within given time interval. Dashed gray lines shows the approximated quarter and halftime show end/start times. The dots with score text shows the game score during that point in time. First score belongs to Patriots and second belongs to Seahawks. 7-0 means Patriots 7, Seahawks 0. Transparent band colors shows the standard deviation of the aggregated data. I used nltk vader sentiment features:

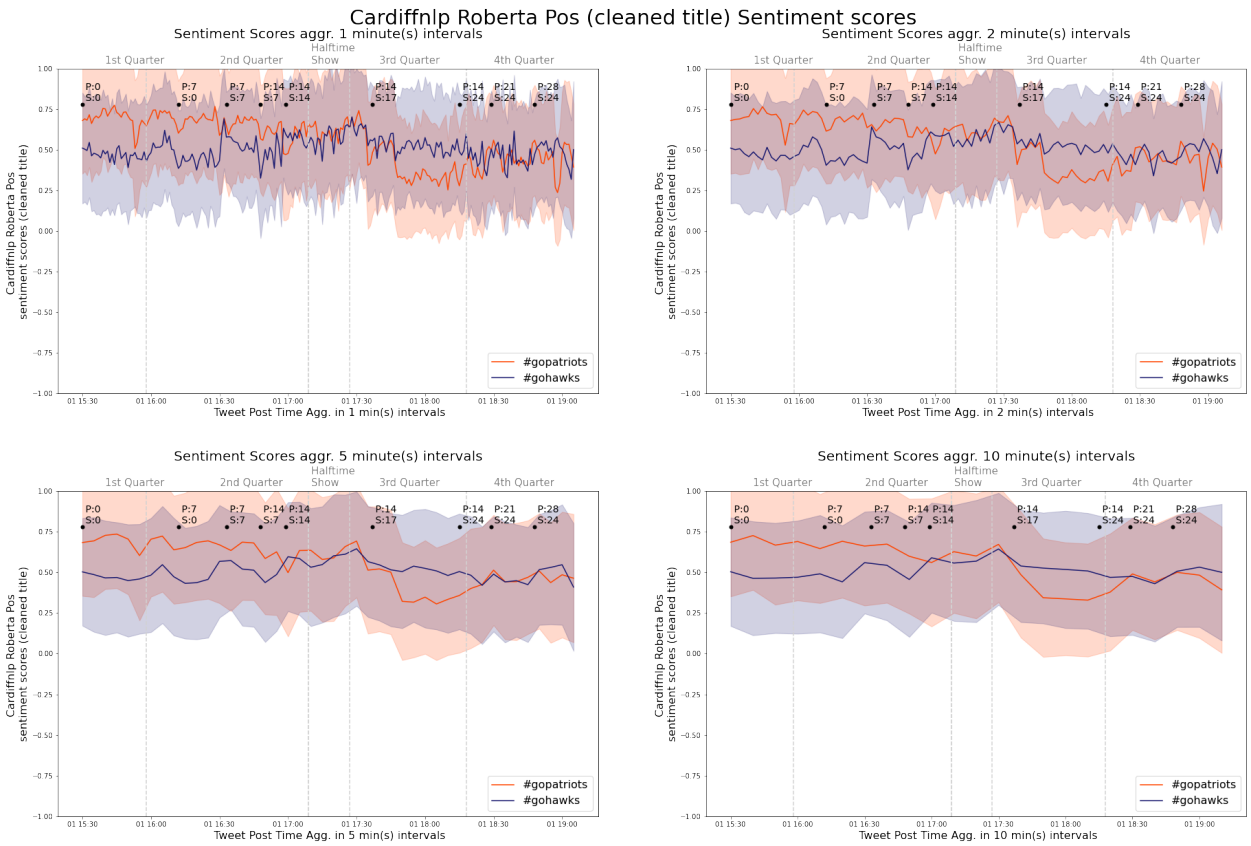
NLTK Vader Fans' emotion during game¶ Plot:



Observations:

- We can see the spikes and downs in the teams when scores happened, for example when Patriots scored a touch down and game score is 7-0, the orange line peaks, showing increase in positive emotion, we can see the huge decrease on Patriots emotions when hawks makes score 14-17, hawks emotions get more positive than patriots when the score is 14-24.
- With nltk vader features, in 2 and 5 minute intervals plot we see same emotion patteerns for patriots and hawks from beginning until midlle of the 3rd quarter in the game. PAtriots have more positive emotions compared to seahawks, though, after the middle of 3rd quarter, at first hawks emotion gets higher and then gets almost similar with Patriots one.

CardiffNLP Fans' emotion during game¶ Same plot drawn for CardiffNLP sentiment features:

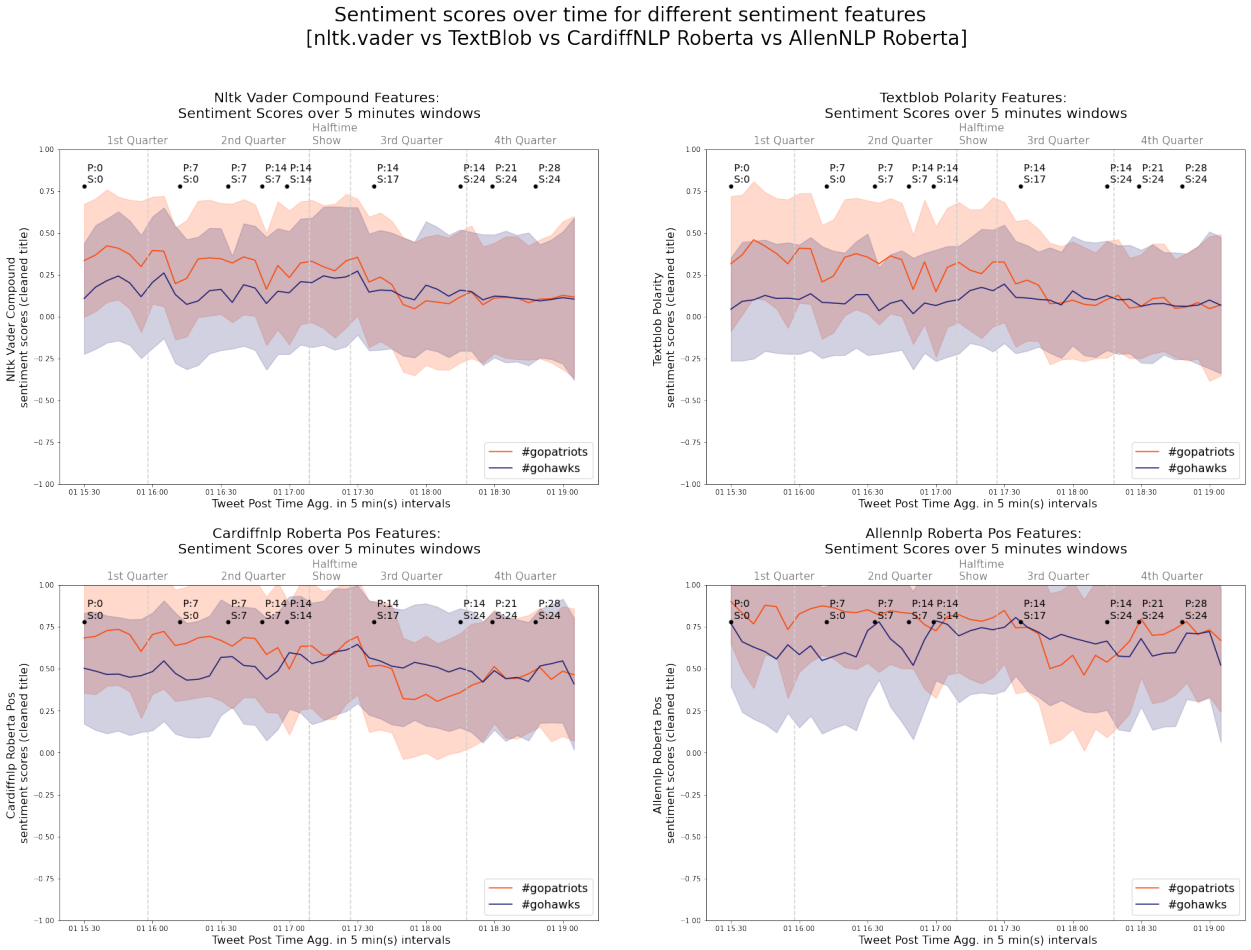


Observations:

- Positive scores with CardiffNLP seems to be much higher compared to NLTK in overall.
- The trends are more clear as well. By looking at the lines we can mostly tell which team is winning. We can see that orange line, patriots, overall emotions is more positive until the end of 2nd quarter, the periods where Patriots are leading the game. Whenever the score difference decrease or a tie happens between the teams the curve of each hashtag becomes closer to each other and the gap decreases as well. When Seahawks, starts to lead in the 3rd quarter, we can also see that the line for seahawks surpasses the patriots as well.
- We can also see big spikes and down emotion trends also correlated with scores.
- As a first intuition, Cardiff NLP emotion distribution seems to show the correlation between scores and perceived emotions on teams more clearly than NLTK. The gaps between lines are more telling especially in the 3rd quarter in CardiffNLP.

Fans' Emotion towards each team: Comparison of different sentiment feature types¶

In this section, I compare the 4 extracted sentiment features in a fixed interval time of 5 minute windows during thee game.



Observations:

- For NLTK in the first half of game the trends for each team is very close, on the other hand, for Textblob the closeness is less and for the 2 Roberta features set the overall line shapes are different.
- For NLTK and Textblob, rule-based approaches, we can see that Patriots have more positive emotions, in the first half of the game, then in the second half the emotions are more flat and lower for both teams, the gap is also very small. For CardiffNLP and AllenNLP Roberta models, we can see better trends.
- If we assume higher positive emotions corresponds to that team is leading the game, then we can see the game status clearly with line trends through out the game with AllenNLP:
  - P:7-S:0, orange line higher -> patriots winning
  - P:7-S:7, orange and purple line gap decrease -> tie
  - P:14-S:7, blue line curve decrease -> patriots score, tie break, patriots leading again
  - P:14-S:14, seahawks scores, gap decrease -> tie
  - P:14-S:24, purple line surpasses orange -> seahawks winning
  - P:21-S:24, orange line surpasses purple -> patriots winning
  - P:28-S:24, orange line surpasses purple -> patriots winning
- The gap between the lines, can even tell the score difference. Notice that when the score difference is higher the gap is bigger, whereas when there is a tie the two lines either merged or get very close.

Fans' Emotion for Top 8 tweeted players¶

Who are the most tweeted players?

	#gopatriots	#gohawks	Tom Brady	Russell Wilson	Rob Gronkowski
tweet_counts	357785.000000	115377.000000	32383.000000	8710.00000	5395.000000
tweet_percentage	0.649418	0.209421	0.058779	0.01581	0.009792

	Marshawn Lynch	Richard Sherman	Chris Matthews	Julian Edelman	Earl Thomas
tweet_counts	4845.000000	4314.00000	2753.000000	2439.000000	2130.000000
tweet_percentage	0.008794	0.00783	0.004997	0.004427	0.003866

	LeGarrette Blount	Doug Baldwin	Darrelle Revis	Kam Chancellor	Vince Wilfork
tweet_counts	1948.000000	1541.000000	1342.000000	1010.000000	949.000000
tweet_percentage	0.003536	0.002797	0.002436	0.001833	0.001723

Who are the top players for each hashtag? Are the players play in defense or offense? Are the 10 players of each hashtag is for that team or the rival team?

Below table shows top 10 players for each hashtag:

#gopatriots

	0	1	2	3	4
hashtag	#gopatriots	#gopatriots	#gopatriots	#gopatriots	#gopatriots
player2	Tom Brady	Rob Gronkowski	Julian Edelman	LeGarrette Blount	Richard Sherman
Team	Patriots	Patriots	Patriots	Patriots	Seahawks
Category	Offense	Offense	Offense	Offense	Defense
tweeet_count	28360	4327	2401	1907	1457

	5	6	7	8	9
hashtag	#gopatriots	#gopatriots	#gopatriots	#gopatriots	#gopatriots
player2	Marshawn Lynch	Russell Wilson	Darrelle Revis	Chris Matthews	Earl Thomas
Team	Seahawks	Seahawks	Patriots	Seahawks	Seahawks
Category	Offense	Offense	Defense	Offense	Defense
tweeet_count	1306	1246	1245	1156	1037

#gohawks

	39	40	41	42	43
hashtag	#gohawks	#gohawks	#gohawks	#gohawks	#gohawks
player2	Russell Wilson	Tom Brady	Marshawn Lynch	Richard Sherman	Chris Matthews
Team	Seahawks	Patriots	Seahawks	Seahawks	Seahawks
Category	Offense	Offense	Offense	Defense	Offense
tweeet_count	7464	4023	3539	2857	1597

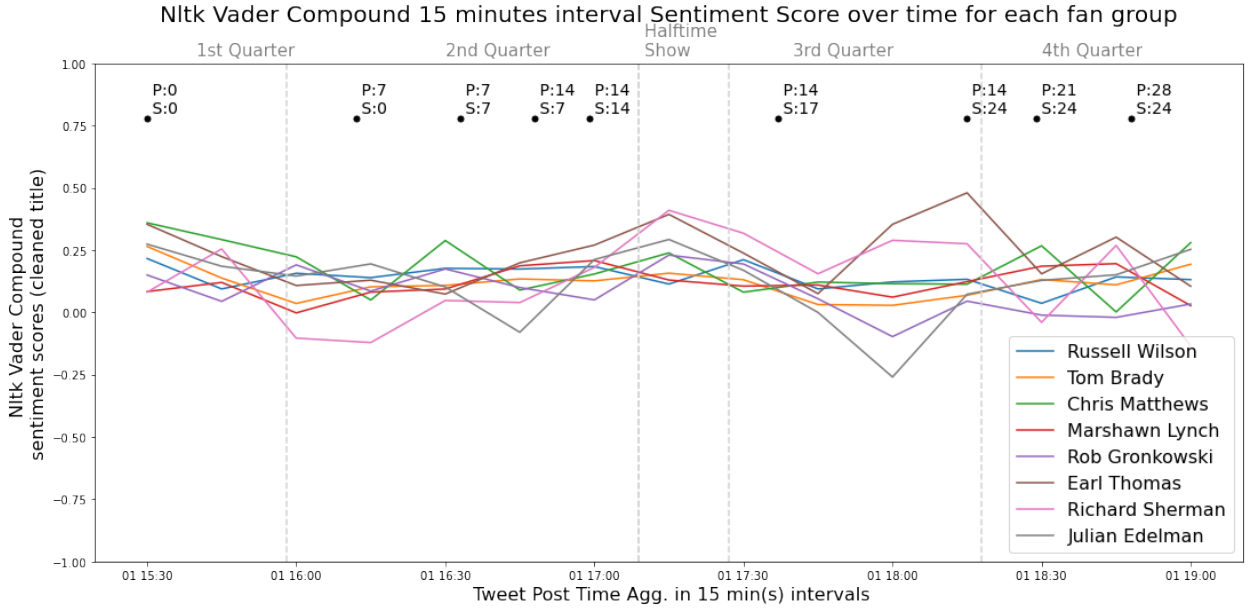
	44	45	46	47	48
hashtag	#gohawks	#gohawks	#gohawks	#gohawks	#gohawks
player2	Doug Baldwin	Earl Thomas	Rob Gronkowski	Kam Chancellor	Jermaine Kearse
Team	Seahawks	Seahawks	Patriots	Seahawks	Seahawks
Category	Offense	Defense	Offense	Defense	Offense
tweeet_count	1307	1093	1068	843	518

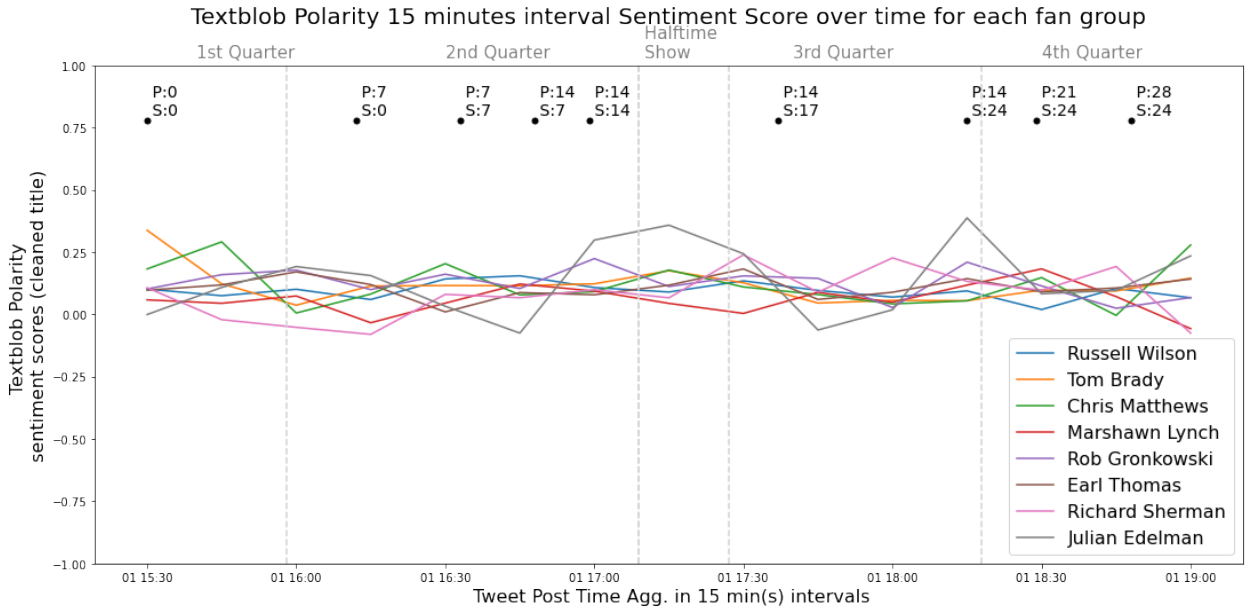
Tom Brady gets the bigger number of tweets. We can see that hawks supporters tends to talk more about Seahawks players whilst patriots fan talks for players of both teams 5/10 top players mentioned by patriots are SeaHawks players. One might suggest that this might be because for patriots we merged #gopatriots and #patriots tags, however, this was the case with 5/10 ratio before merging #gopatriots and #patriots tag as well.

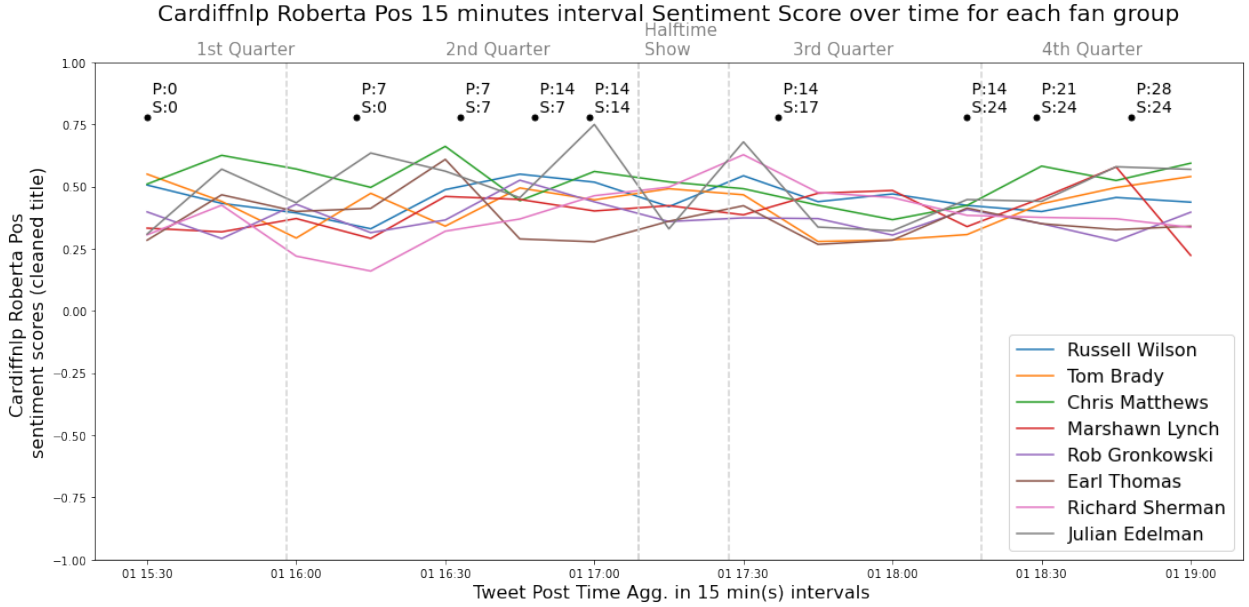
In both of the hashtags majority of the comments are for offense players than defense players. #gopatriots mainly talks about opposite team's defense players when the player's category is defense. This potentially means that even talking about defense players, patriots fan are more focusing on the offense moments of their team. We may conclude that fans are more interested talking about attack moments of their team (deffense of opposing team).

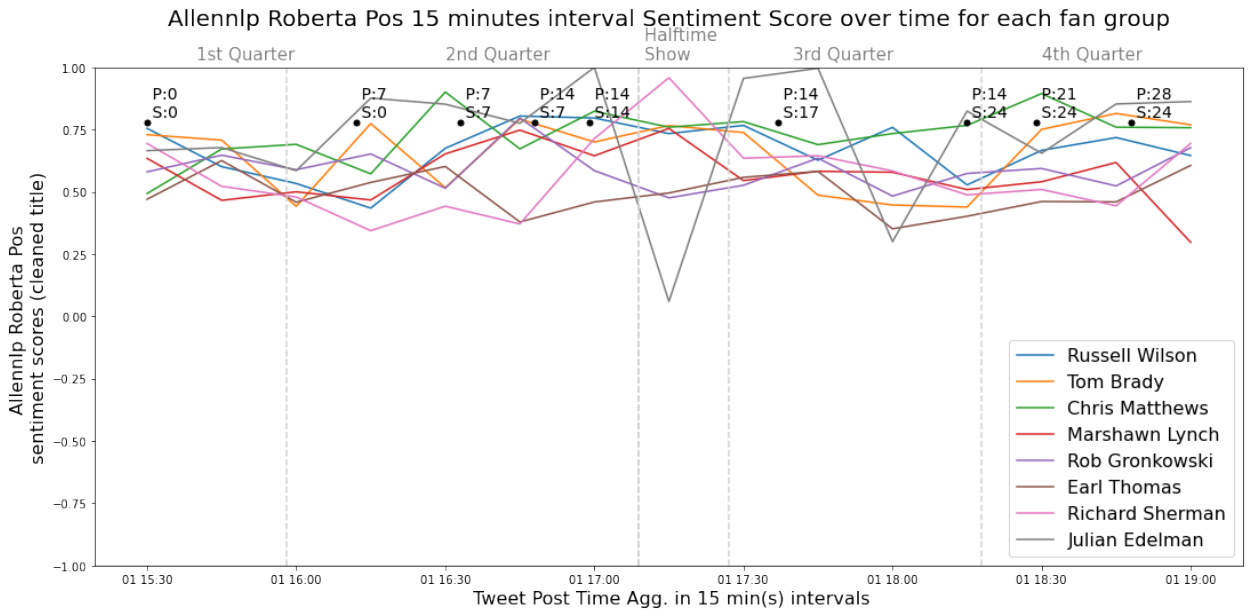
I plotted the perceived emotion about players over 15 minutes time inteerval for each feature set. I chose the most tweeted 8 players and increaseed time interval to 15 minutes for convenience of reading plots more easily. The perceived emotions in these plots are combined for each fan group. Given that the data is unbalanced against each hashtag, the results might be towards more Patriots' fan feeling.

```
Tom Brady, Russell Wilson, Rob Gronkowski,
Marshawn Lynch, Richard Sherman, Chris Matthews,
Julian Edelman, Earl Thomas
```




Textblob Polarity 15 minutes interval Sentiment Score over time for each fan group
1st Quarter
2nd Quarter
Halftime Show
3rd Quarter
4th Quarter
P:0 S:0
P:7 S:0
P:7 S:7
P:14 S:7
P:14 S:14
P:14 S:17
P:14 S:24
P:21 S:24
P:28 S:24
Textblob Polarity sentiment scores (cleaned title)
Tweet Post Time Agg. in 15 min(s) intervals
Russell Wilson
Tom Brady
Chris Matthews
Marshawn Lynch
Rob Gronkowski
Earl Thomas
Richard Sherman
Julian Edelman


Cardiffnlp Roberta Pos 15 minutes interval Sentiment Score over time for each fan group
1st Quarter
2nd Quarter
Halftime Show
3rd Quarter
4th Quarter
P:0 S:0
P:7 S:0
P:7 S:7
P:14 S:7
P:14 S:14
P:14 S:17
P:14 S:24
P:21 S:24
P:28 S:24
Cardiffnlp Roberta Pos sentiment scores (cleaned title)
Tweet Post Time Agg. in 15 min(s) intervals
Russell Wilson
Tom Brady
Chris Matthews
Marshawn Lynch
Rob Gronkowski
Earl Thomas
Richard Sherman
Julian Edelman



Observations:

- Notice that 5 of the 8 top tweeted players are from Seahawks. (Russell Wilson, Chris Matthews, Marshawn Lynch, Earl Thomas, Richard Sherman)
- First two players are passing leaders of each team, the next 2 are receiver for Seahawks, Rob is receiver for Patriots, Earl and Richard are defense players for Seahawks, and Julian is receiver for Patriots.
- We can see that emotions for players, varies a lot during the game, the most sharp and bigger changes seeem to happen in AllenNLP and NLTK. Julian Edelman seems to have the biggest change of heart from fans during the game, where it has the biggest peaks and downs.The emotions perceived for Tom Brady seems rather stale in the first 3 approaches, but changes a lot in AllenNLP.
- For CardiffNLP, we can see clear peaks on the players who involved plays during Touch Down and Field Goal times. This also makes sense since it is very likely that team's praise the player who take role in scoring or shades the players who fail to defense.

Individual Players' Emotion Plots over time during the game¶

In this part, I plotted individual players' perceived emotion during game. Each plot has emotion trends for 4 sentiment feature types and two of the fan base. Again plotted for 15 minute intervals for convenience.

```
Players selected: #gopatriots, #gohawks, Tom Brady, Russell Wilson,
Rob Gronkowski, Marshawn Lynch, Richard Sherman, Chris Matthews,
Julian Edelman, Earl Thomas
```

Observations:

- #gohawks, #gopatriots: these emotion groups belong to the tweets where I can't identiy a player. It also corresponds to the majority of the tweets. #gopatriots is mostly on highly positve stable curve up to 3rd quarter, and has a huge decrease in 3rd quarter. These makes perfect sense since Patriots lead thee game in first half and Seahawks, lead the 3rd quarter. I cannot clearly see trends in Touch Downs though. For seahawks, we can see smooth little peaks when Seahawks ties the game twice in thee second quarteer. Textblob and NLTK curves are almost same. The trends of Roberta models are very similar. AllenNLP assigns higher emotion scores than CardiffNLP.
- Tom Brady: The emotions are keep changing for Tom Brady during the game for both fans. Patriots

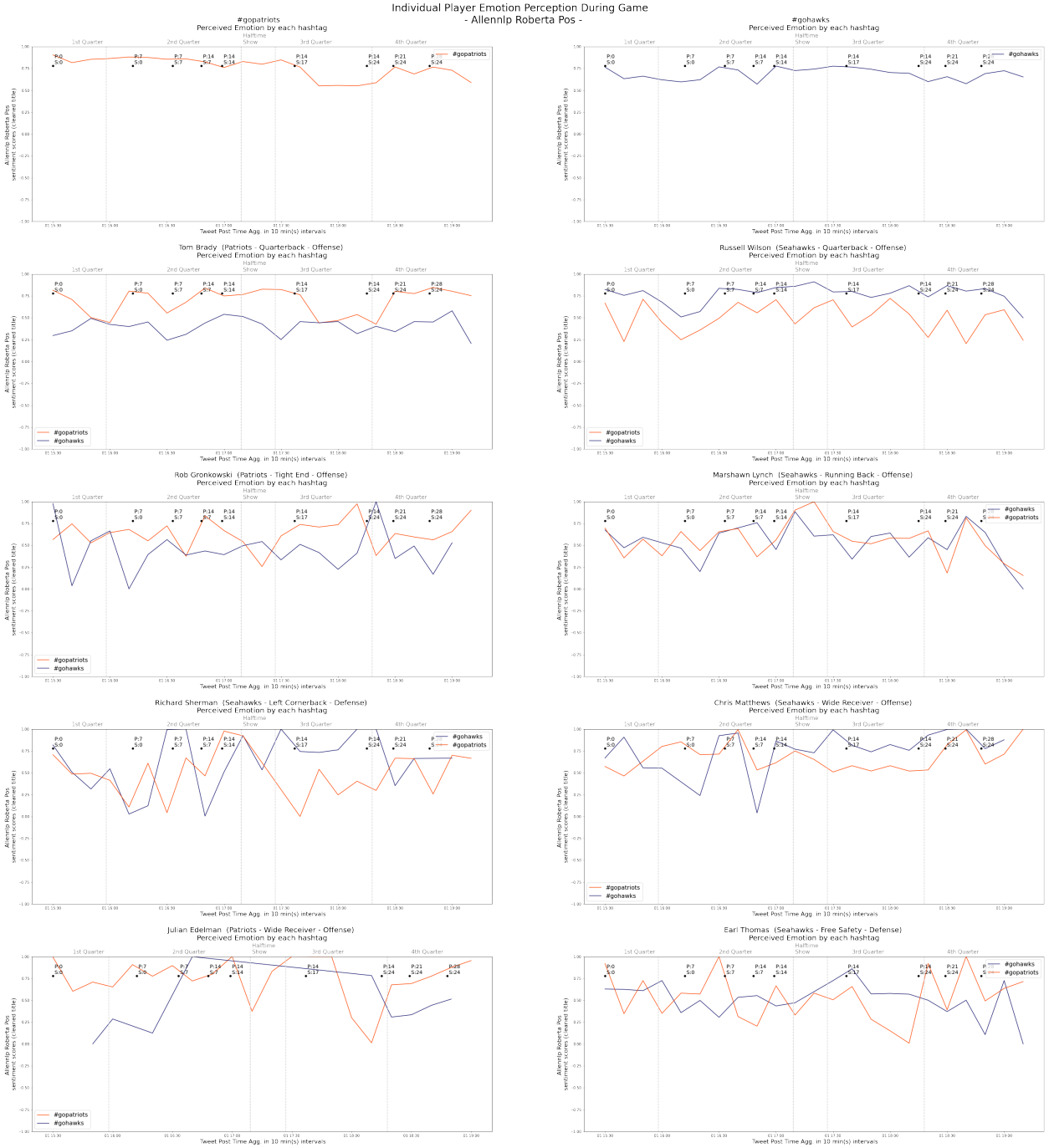
fan's positive emotion increases towards Tom Brady when Patriots score, and decrease considerably when Seahawks scores. Again we see that AllenNLP assigns higher values, then CardiffNLP and then the rule based features.

- Russell Wilson: The zigzag trend is more towards Russell Wilson from both fans, compared to Tom Brady.
- The trends of different feature types throughout the game per fan base seems to be similar but differs in the score amount.
- The more lower towards the subplots, players with lower tweet number, the data becomes more sparse throughout time or fan base, and the more chaotic lines seem to get. Patterns are harder to interpret, and for different feature set, the trends are no longer simiilar either.

Individual Players' Emotion Plots over time during the game for AllenNLP¶

Since it is harder to read the previous plots, I also plotted the same thing on only AllenNLP Roberta Feeatures. I chose AllenNLP since the polarity seem to be highhtened in this feature set and previous results were more interesting for this one than the other feature sets. I decreased the intervals to 10 minutes, since we have less lines, and want to see more granular, accurate patterns.

```
Players selected: #gopatriots, #gohawks, Tom Brady, Russell Wilson,
Rob Gronkowski, Marshawn Lynch, Richard Sherman, Chris Matthews,
Julian Edelman, Earl Thomas
```



Observations:

- Tom Brady: Patriots have a clear pattern for Tom, when Patriots score huge jump in emotion towards positive, when Seahawks scores huge decrease. This makes sense since Tom Brady is the main player. Patriots overall emotion is more positive for Tom.
- Russell Wilson: Again, as stated in the previous plot, the emotion is way zigzagy and fragile for Russell Wilson than Tom Brady. Seahawks fans emotion is more stable, and has an upward trend when SeaHawks scores. Seahawks emotions are more positive overall for Russell.
- Rob Gronkowski: He is an offense player for Patriots and the most interesting thing is that sometimes hawks fans' emotions are higher to a Patriot player than from patriot fans (halftime show, beegining

of 4th quarter). Could positive emotions be more sarcasm? Especially in 4th quarter the positive emotions come after hawks score. It is also possible that more than 1 player is listed in the tweet and we see these results, ie. hawks fan may be talking about a specific moment between the players of the teams such as Rob and his counterpart in the game.

- Marshawn Lynch: Perceived emotion by each fan group is again more complicated, we can't tell which fan group as an overall higher/lower emotion for him during the game. Two clear peaks are seahawks scored at the end of second quarter and when Patriots scored 21-24 in 4th quarter. He seems to make a comeback in the second half of the game, since hawks fan emotion is way higher in the second half of the game for him.
- Richard Sherman & Chris Matthews: I see huge peaks in emotion when Seahawks scores, which makes sense. Looks like more seahawks emotion towards the whol game rather than player itself.
- Julian Edelman: Very few hawks fan tweets. There is a huge PAtriots' fan emotion decrease twice during game, both which seahawks scored. He might have been interrupted or played a role on loosing the ball? (excuse my lousy american football terms).

For most of the players, I believe there might be more than one player stated in the tweet, and the tweet emotion might not directly correlate to the player itself but rather a significant position/event happening in the game where name of the player is stated. Such as "Wow, such a comeback Tom Brady and Russell Wilson are playing big. Way to go Seahawks I hope you win.": if this would have been a real tweet we would assign positive scores both Tom and Russell no matter which team's fan I am. The problem is the emotion in the tweet is not directly towards either player but rather team itself. There might be more interesting inferences from these plot about players but my american football knowledge is limited to the things I learned in the last two weeks.

Pairwise Players' Perceived Emotion Change During Game¶

In this part, I coupled the leaders of the game by their counterparts in the match, such as Tom Brady and Russell Wilson and check the overall perceived emotion for both players oveer time during game. Note that at this part perceived emotion is calculated for both fans together. Time interval over 5 minutes. I used AllenNLP Roberta feature for this part.

Patriots

Top 10 players tweet counts in all dataset:

	20	21	22	23	24
Team	Patriots	Patriots	Patriots	Patriots	Patriots
player2	Tom Brady	Rob Gronkowski	Julian Edelman	LeGarrette Blount	Darrelle Revis
Category	Offense	Offense	Offense	Offense	Defense
tweeet_count	32383	5395	2439	1948	1342

	25	26	27	28	29
Team	Patriots	Patriots	Patriots	Patriots	Patriots
player2	Vince Wilfork	Brandon LaFell	Michael Hoomanawanui	Chandler Jones	Danny Amendola
Category	Defense	Offense	Offense	Defense	Offense
tweeet_count	949	868	737	507	491

Saahawks

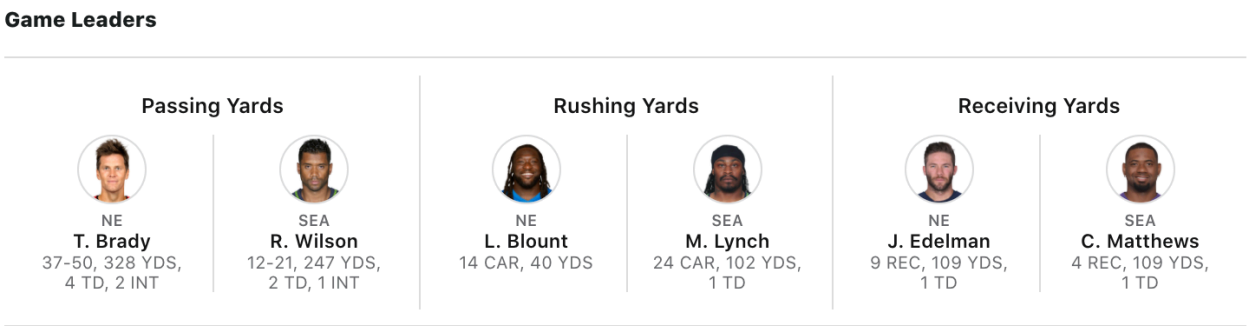
Top 10 players tweet counts in all dataset:

	0	1	2	3	4
Team	Seahawks	Seahawks	Seahawks	Seahawks	Seahawks
player2	Russell Wilson	Marshawn Lynch	Richard Sherman	Chris Matthews	Earl Thomas
Category	Offense	Offense	Defense	Offense	Defense
tweeet_count	8710	4845	4314	2753	2130

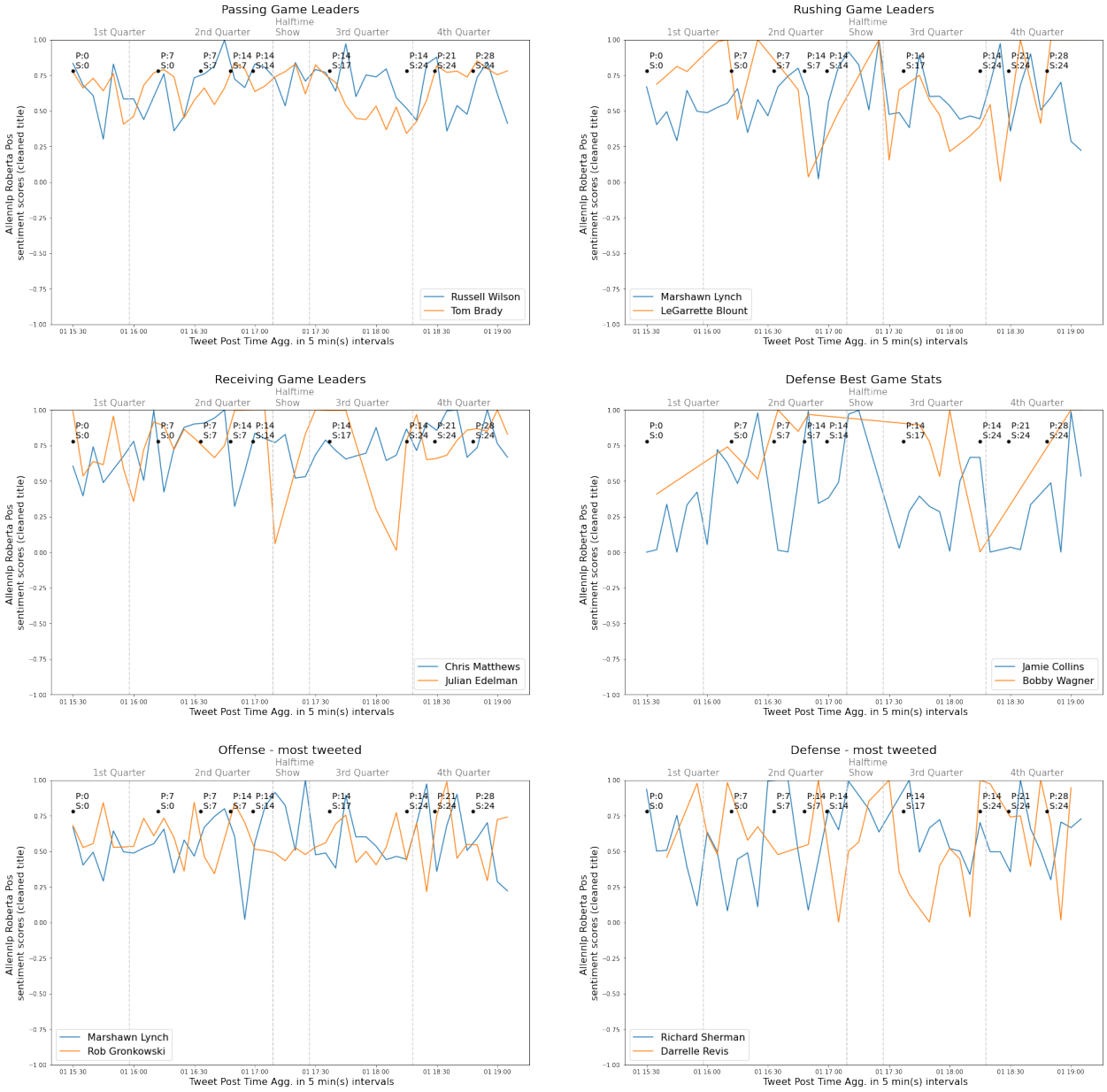
	5	6	7	8	9
Team	Seahawks	Seahawks	Seahawks	Seahawks	Seahawks
player2	Doug Baldwin	Kam Chancellor	Jermaine Kearse	Bryan Walters	Tony McDaniel
Category	Offense	Defense	Offense	Offense	Defense
tweeet_count	1541	1010	646	585	430

For offense and defense use players other than Tom Brady and Russell Wilson since we already compared them in Passing Leaders.

Image shows the game leaders listed in ESPN website for the game:



ref: https://www.espn.com/nfl/game/_/gameId/400749027



Passing Game Leaders: Tom Brady vs Russell Wilson

- There is a pick in both players emotion positively whenever a team is scored. Sometimes the emotion with each player is related based on their team's status only, ie. peak in Russell 3rd quarter when Seahawks scores, peak in Tom's curve when Patriots score in 2nd quarter P14-S7. Though sometimes emotions are rather merged, see peaks in P7-S0, P21-S24, P28-S24, both player's get peak, irrespective of which team is scored. I believe the reason behind this is that both players are mentioned in the time of these touch down tweets.

Rushing Game Leaders: Marshawn Lynch vs LeGarette Blount

- The curve patterns mostly have same high peaks and low drop moments for both players.

Receiving Game Leaders: Chris Matthews vs Julian Edelman

- The emotions are motly opposite for each player when touch downs happened. When Patriots score, Julian's emotion raise towards more positive and Chris' drops, and when Seahawks scores reverse happens. This is interesting, we can see the expected opposite team emotions for these player throughout the game.

Defense Best Game Stats: Jamie Collins vs Bobby Wagner

- Bobby Wagner's emotion features are more sparse than Jamie Collins. Again wee see the opposite feeling pattern we observed above with these two players.

Offense - most tweeted: Marshawn Lynch vs Rob Gronkowski & Defense - most tweeted: Richard Sherman - Darnell Revis

Again we see the opposite patterns, when one player peaks other's emotion drops in the touch down times and most likely in other significant events during game.

Prediction Task¶

From tweet emotions expressed to players in 1-minute time windows can we predict who is winning?

I aggregated the emotions about each player in 1-minute time windows during the game, for aggregation I used mean of the given sentiment scores. For mapping tweets to players I used the same starategy described in previous parts. I also added 2 more columns: #gohawks and #gopatriots where I get the overall sentiment for each team in 1-minute intervals. There is two main reason I added these two columns: 1- around 75% of the data I couldn't idendity player name, so if I don't I would be loosing too much information that might help my prediction task. 2- The final matrix created is a sparse matrix, not many players have tweets posted with their name each minute of the game. These two columns are dense one, that might increase the prediction power of the task.

The classes of this task is Tie, Patriots or Seahawks. This was also unable in the raw tweet data and for the labels I used the 'leading' column of the score_df I created in the previous parts. To map the game time to the tweets real time, I used the approximated score times (described in detail Data Exploration & Feature Engineering: Mapping Game Time to Real Time section), with the use of these time mappings I assigned the labels of who is leading the game to every 1-minute interval.

For each 4 feature set type, I created different datasets for the prediction task and applied 3 ML models to all 4 of them.

Feature sets:

- Nltk.sentiment.vader,
- TextBlob,
- CardiffNLP Roberta and
- AllenNLP Roberta

ML Models:

- Logistic Regression
- Random Forest
- Neural Network

I choose these 3 models, because I want to see the performance of the models in a simple baseline model (LR), a model that is performed well in unbalanced dataseet and a more complex model (NN).

The steps followed:

- Create data for the modelling, get labels and prepare data for modelling
- Split data to train-test set, do 10-fold cross validation for model parameter tuning using train dataset to select best estimators for each model type
- Apply test set on the refitted ML models with best parameters on the all training dataset

- Check Feature Importance for Random Forest and coeff weights for Logistic Regression models.

I only used the data tweeted during the superbowl 49 game and English tweets.

In addition to the prediction task, I want to check the importance of the models I created to see whether I can say anything additional about the players. Who is the most important player for the game. Can we tell who might be the MVP of the game? Along with player's position can we tell who could be the best Receiver, Defense player?

Creating the data to be used for prediction task:¶ How many tweeets each player received during game?

```
#gopatriots             57138
#gohawks                19203
Tom Brady                5977
Russell Wilson           1302
Chris Matthews           1100
Marshawn Lynch            864
Julian Edelman            646
Rob Gronkowski            571
Doug Baldwin              354
Brandon LaFell            272
Richard Sherman           251
Earl Thomas               240
Jermaine Kearse           221
Danny Amendola            220
Darrelle Revis            154
LeGarrette Blount         137
Kyle Arrington            129
Shane Vereen              113
Jamie Collins             106
Bobby Wagner              103
Vince Wilfork              77
Chandler Jones             71
Michael Hoomanawanui       69
Tony McDaniel              64
Kevin Williams             55
Kam Chancellor             54
Bruce Irvin                46
Cliff Avril                39
K.J. Wright                38
Bryan Walters              33
Rob Ninkovich              30
Ricardo Lockette           30
Dont'a Hightower           26
Brandon Browner            26
Patrick Chung              15
Byron Maxwell              12
Michael Bennett            10
Devin McCourty              5
Robert Turbin               3
Name: player2, dtype: int64
```

How Data Looks Like?¶ Game took 216 minutes, that is why we have 216 rows, samples in our dataset and I found tweets posted on 40 players during game + 2 team columns, 42 columns. Below, we can see the

data created for AllenNLP Roberta Positive Score Features for the first 5 minute of the game. Notice that the data seems very sparse for some of the players.

Data Preview for the first 6 columns of first 5 minutes in the game:

```
(216, 42)
```

player2	post_time_1_intervals	#gohawks	#gopatriots	Bobby Wagner	Brandon Browner	Brandon LaFell
0	2015-02-01 15:30:00-08:00	0.765169	0.915447	NaN	NaN	NaN
1	2015-02-01 15:31:00-08:00	0.815970	0.918031	NaN	NaN	NaN
2	2015-02-01 15:32:00-08:00	0.781914	0.905448	NaN	NaN	0.951774
3	2015-02-01 15:33:00-08:00	0.733644	0.895856	NaN	NaN	NaN
4	2015-02-01 15:34:00-08:00	0.657131	0.878550	NaN	NaN	NaN

As expected, data becomes less sparse for the most tweeted players. Below you can see only the most tweeted players and randomly selected 10 minutes of the game:

(post_time_1_intervals won't be used in the modelling, I put in here just for showing what time in the game the selected 1-minute interval corresponds to)

player2	post_time_1_intervals	#gohawks	#gopatriots	Tom Brady	Russell Wilson	Chris Matthews
201	2015-02-01 18:51:00-08:00	0.768218	0.672990	0.713158	0.709041	0.781835
213	2015-02-01 19:03:00-08:00	0.548149	0.745636	0.819526	0.460369	0.000856
138	2015-02-01 17:48:00-08:00	0.700695	0.559546	0.433796	0.999411	0.809007
177	2015-02-01 18:27:00-08:00	0.532102	0.788759	0.660526	NaN	0.999224
15	2015-02-01 15:45:00-08:00	0.630571	0.881566	0.666181	NaN	0.285398
111	2015-02-01 17:21:00-08:00	0.765330	0.791283	0.999210	0.943161	NaN
182	2015-02-01 18:32:00-08:00	0.700104	0.748749	0.829745	0.947678	0.996738
73	2015-02-01 16:43:00-08:00	0.684693	0.853567	0.571063	0.998351	NaN
205	2015-02-01 18:55:00-08:00	0.640510	0.675349	0.702878	0.647346	0.999822
139	2015-02-01 17:49:00-08:00	0.737002	0.507526	0.465867	0.602391	0.757743

player2	Marshawn Lynch	Julian Edelman	Rob Gronkowski	leading
201	0.678363	0.892279	0.990402	Patriots
213	0.163377	0.828680	0.000495	Patriots
138	NaN	NaN	0.382201	Seahawks
177	NaN	0.606261	NaN	Seahawks
15	0.201507	0.999847	0.786160	Tie
111	0.998131	NaN	0.572241	Tie
182	0.999836	0.462827	0.995795	Patriots
73	0.997983	NaN	0.320715	Tie
205	0.757766	0.990173	NaN	Patriots
139	NaN	NaN	NaN	Seahawks

In the prediction task I will only use a subset of the players, since for some players the data is very very sparse, it might hurt prediction rather than help. As I experimented with the full dataset, I realized that those sparse columns are creating noise to confuse model than to increase prediction power. For example, for Rober Turbin I could only match 3 tweets in the entire game, in the best case if every tweet is posted in separate minutes, we only have 3 rows filled out of 216.

That is why, I will only use a subset of these 42 columns and select the best ones based on their density level,

by dropping the most sparse columns.

Parameter Tuning Experiments:¶ For parameter tuning I applied 10-fold cross validation on training dataset (contains 80% of samples). For each model I tuned the following parameters:

Logistic Regression:

- penalty: ['none','l1','l2'],
- class_weight: [None, 'balanced'],
- C: [10**(x) for x in range(-2,2)],

Random Forest:

- max_features: ['auto', None, 1, 2, 3, 4, 6] ,
- n_estimators: [5, 10, 20, 40, 60],
- max_depth: [2, 4, 6, None],
- class_weight: [None, 'balanced', 'balanced_subsample'],

Neural Network:

- hidden_layer_sizes: [(5,), (8,), (10,), (30,),(50,),(100,), (120,), (30,30), (50, 50)],
- alpha: [10**(x) for x in range(-5,3)] ,
- learning_rate_init:[0.001, 0.01, 0.1],
- activation:['relu','logistic','tanh']

For Logistic Regression and Neural Network, I applied 10-fold cross validation on gridsearch, for Random forest I used 10-fold cross validation on randomized search with 100 samples.

In this experiment I only kept selected denser columns that have at least 50% of the rows that are non-nulls. This corresponds to 8 out of 42 columns. The columns kept are: ['#gohawks', '#gopatriots', 'Chris Matthews', 'Earl Thomas', 'Marshawn Lynch', 'Rob Gronkowski', 'Russell Wilson', 'Tom Brady']

I also filled the remaining null values with 0s as 0 is the neutral score in this case:

```
-- Info about data:

X shape: (216, 8)

'Class Distribution:'

Tie         98
Patriots    67
Seahawks    51
Name: leading, dtype: int64

'Class Distribution (normalized):'

Tie         0.453704
Patriots    0.310185
Seahawks    0.236111
Name: leading, dtype: float64

   Dropped columns where less than 50.0% is empty
    X shape after dropping columns: (216, 8)
        Remaining columns:                  ['#gohawks',
'#gopatriots', 'Chris Matthews', 'Earl Thomas', 'Marshawn Lynch', 'Rob
Gronkowski', 'Russell Wilson', 'Tom Brady']
    Filling NaN emotions scores with 0..
Model Experiments for : sent_nltk_vader_compound_1_mins data:
Model Experiments for : sent_textblob_polarity_1_mins data:
Model Experiments for : sent_cardiffnlp_roberta_pos_1_mins data:
```

Model Experiments for : sent_allennlp_roberta_pos_1_mins data:

Top 3 Best Features Found for each feature set and model pairs:¶ Below, I shared the top 3 best tuned parameter combination for each feature set, model pairs along with 10-fold training and validation scores.

sent_nltk_vader_compound_1_mins Features - LR Model CV Results

	mean_train_score	mean_validation_score	param_model__C	param_model__class_weight
9	0.709271	0.655229	0.10	balanced
22	0.712501	0.655229	10.00	balanced
3	0.709271	0.655229	0.01	balanced

	param_model__penalty	rank_test_score
9	none	1
22	l1	1
3	none	1

sent_nltk_vader_compound_1_mins Features - RF Model CV Results

	mean_train_score	mean_validation_score	param_model__n_estimators	param_model__max_features
77	0.950930	0.743137	40	3
74	0.788726	0.738562	20	3
80	0.880494	0.738235	20	3

	param_model__max_depth	param_model__class_weight	rank_test_score
77	6.0	NaN	1
74	2.0	NaN	2
80	4.0	NaN	3

sent_nltk_vader_compound_1_mins Features - NN Model CV Results

	mean_train_score	mean_validation_score	param_model__hidden_layer_sizes	param_model__alpha
101	0.754512	0.609150	(120,)	0.0100
575	0.642145	0.604575	(10,)	1.0000
38	0.708638	0.604248	(30,)	0.0001

sent_textblob_polarity_1_mins Features - LR Model CV Results

	mean_train_score	mean_validation_score	param_model__C	param_model__class_weight
9	0.647294	0.633333	0.10	balanced
3	0.647294	0.633333	0.01	balanced
21	0.647294	0.633333	10.00	balanced

	param_model___penalty	rank_test_score
9	none	1
3	none	1
21	none	1

`sent_textblob_polarity_1_mins Features - RF Model CV Results`

	mean_train_score	mean_validation_score	param_model___n_estimators	param_model___max_features
87	0.891487	0.726797	40	2
58	0.898580	0.715033	20	6
15	0.974805	0.709477	20	6

	param_model___max_depth	param_model___class_weight	rank_test_score
87	4.0	balanced_subsample	1
58	4.0	balanced_subsample	2
15	6.0	balanced_subsample	3

`sent_textblob_polarity_1_mins Features - NN Model CV Results`

	mean_train_score	mean_validation_score	param_model___hidden_layer_sizes	param_model___alpha
119	0.709305	0.628758	(30,)	0.100
131	0.679535	0.623203	(30, 30)	0.100
65	0.714491	0.622876	(30,)	0.001

`sent_cardiffnlp_roberta_pos_1_mins Features - LR Model CV Results`

	mean_train_score	mean_validation_score	param_model___C	param_model___class_weight
20	0.748710	0.697386	10.00	NaN
19	0.746125	0.697386	10.00	NaN
0	0.741605	0.685621	0.01	NaN

	param_model___penalty	rank_test_score
20	l2	1
19	l1	1
0	none	3

`sent_cardiffnlp_roberta_pos_1_mins Features - RF Model CV Results`

	mean_train_score	mean_validation_score	param_model___n_estimators	param_model___max_features
34	0.892765	0.750980	10	NaN
60	0.770670	0.745752	40	NaN

	mean_train_score	mean_validation_score	param_model___n_estimators	param_model___max_features
64	0.983192	0.745098	40	4

	param_model___max_depth	param_model___class_weight	rank_test_score
34	4.0	balanced	1
60	2.0	balanced	2
64	6.0	NaN	3

`sent_cardiffnlp_roberta_pos_1_mins Features - NN Model CV Results`

	mean_train_score	mean_validation_score	param_model___hidden_layer_sizes	param_model___alpha
11	0.658140	0.646732	(30,)	0.00001
149	0.676984	0.634314	(50,)	1.00000
92	0.649074	0.629412	(30,)	0.01000

`sent_allennlp_roberta_pos_1_mins Features - LR Model CV Results`

	mean_train_score	mean_validation_score	param_model___C	param_model___class_weight
20	0.724801	0.668954	10.00	NaN
0	0.724835	0.657516	0.01	NaN
6	0.724835	0.657516	0.10	NaN

	param_model___penalty	rank_test_score
20	l2	1
0	none	2
6	none	2

`sent_allennlp_roberta_pos_1_mins Features - RF Model CV Results`

	mean_train_score	mean_validation_score	param_model___n_estimators	param_model___max_features
25	0.998056	0.703595	20	6
17	0.991609	0.698693	40	3
72	0.970268	0.693137	10	3

	param_model___max_depth	param_model___class_weight	rank_test_score
25	NaN	NaN	1
17	6.0	balanced	2
72	6.0	balanced_subsample	3

`sent_allennlp_roberta_pos_1_mins Features - NN Model CV Results`

	mean_train_score	mean_validation_score	param_model___hidden_layer_sizes	param_model___alpha
68	0.594378	0.557516	(50,)	0.001
97	0.625987	0.552288	(100,)	0.010
583	0.617495	0.547059	(100,)	1.000

Evaluation Results¶

For evaluating I used accuracy as my metric.

Below table shows 10-fold average train and validation scores and final test score for each model and feature types:

	Mean Train Score			Mean Validation Score			Mean Test Score		
Model Name Sentiment Feature	LR	NN	RF	LR	NN	RF	LR	NN	RF
Nltk Vader Compound	0.709	0.755	0.951	0.655	0.609	0.743	0.773	0.432	0.727
Textblob Polarity	0.647	0.709	0.891	0.633	0.629	0.727	0.750	0.705	0.705
Cardiffnlp Roberta Pos	0.746	0.658	0.893	0.697	0.647	0.751	0.750	0.477	0.795
Allennlp Roberta Pos	0.725	0.594	0.998	0.669	0.558	0.704	0.705	0.568	0.659

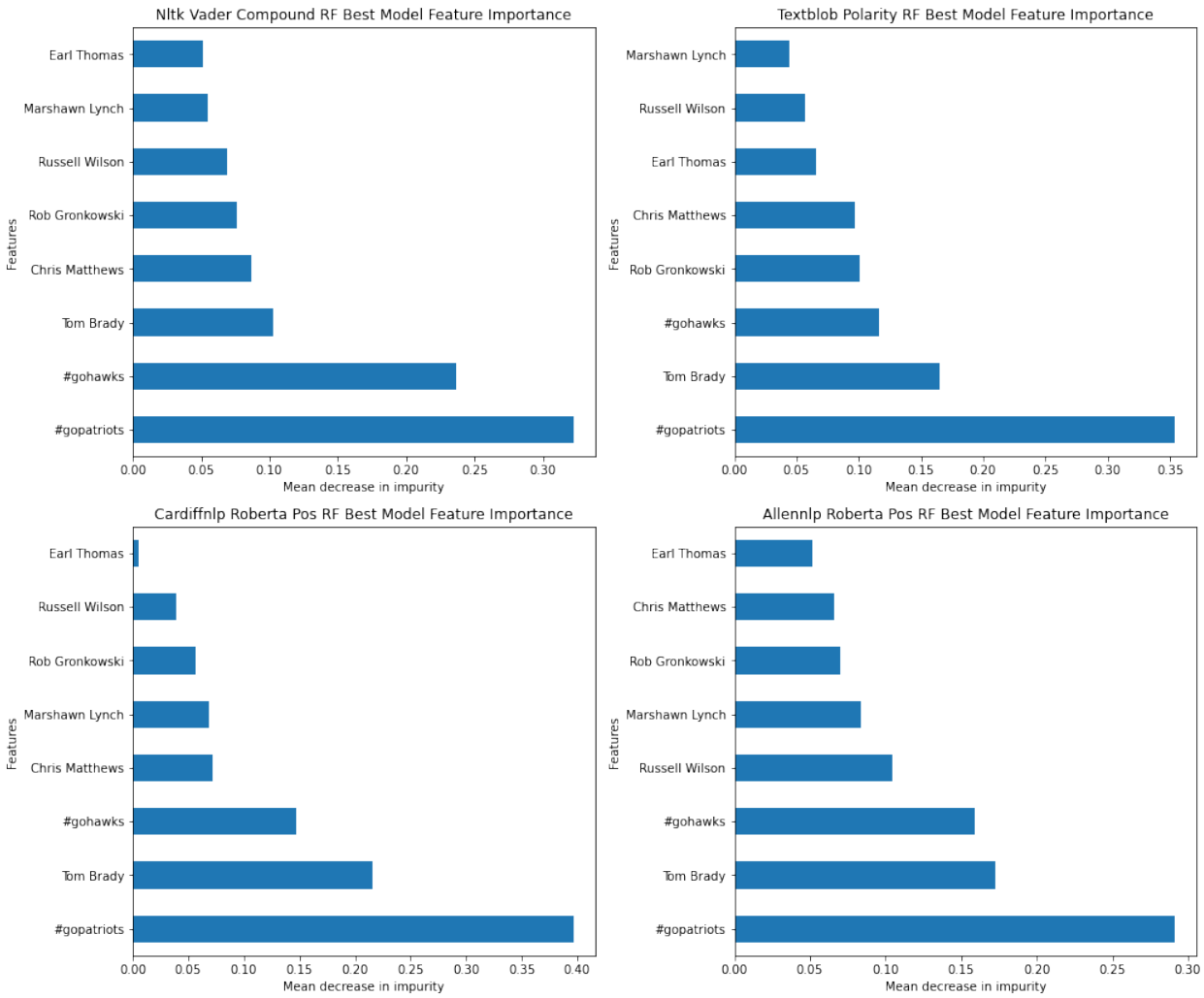
We need to be mindful while checking these results given that we have 1-minute intervals, the samples in the data is 216, very small dataset. Test sample corresponds only 20% of the data, which is around 43 samples. Hence, test samples are small and might not reflect the overall game status distribution, and should be taken into account with caution.

I will mainly focus on validation scores: the best feature set for this prediction task seems to be Cardiffnlp Roberta Pos and best model is Random Forest. With Random Forest on CardiffNLP features, we can reach up to 75% mean validation accuracy and 79.5% test accuracy.

I also did the same experimentation on different subset of columns (Players) based on different density levels with fewer and more columns than I had in this data. 30% and 80% density selected feature experiment results are shared in the appendix.

Interpreting Models¶

RF feature importances and LR coeff plots for each feat type¶ Below you can seee the feature importances for each feature set on the best estimator found from parameter tuning.




```
Index(['Player', 'Pos', 'Team', 'espn_team_list', 'Abbreviation', 'Position',
       'Category', 'player_first_name', 'player_last_name'],
      dtype='object')
```

	Player	Pos	Team	espn_team_list	Abbreviation	Position	Category
0	Tom Brady	QB	Patriots	True	QB	Quarterback	Offense
4	Rob Gronkowski	TE	Patriots	True	TE	Tight End	Offense
21	Russell Wilson	QB	Seahawks	True	QB	Quarterback	Offense
22	Marshawn Lynch	RB	Seahawks	True	RB	Running Back	Offense
42	Earl Thomas	FS	Seahawks	True	FS	Free Safety	Defense
46	Chris Matthews	WR	Seahawks	True	WR	Wide Receiver	Offense

Overall, we see that #gohawks and #gopatriots are the two columns that have high influence on the model results. We can see that Tom Brady is the player who has the most importance in all 4 models independent of which feature set we use.

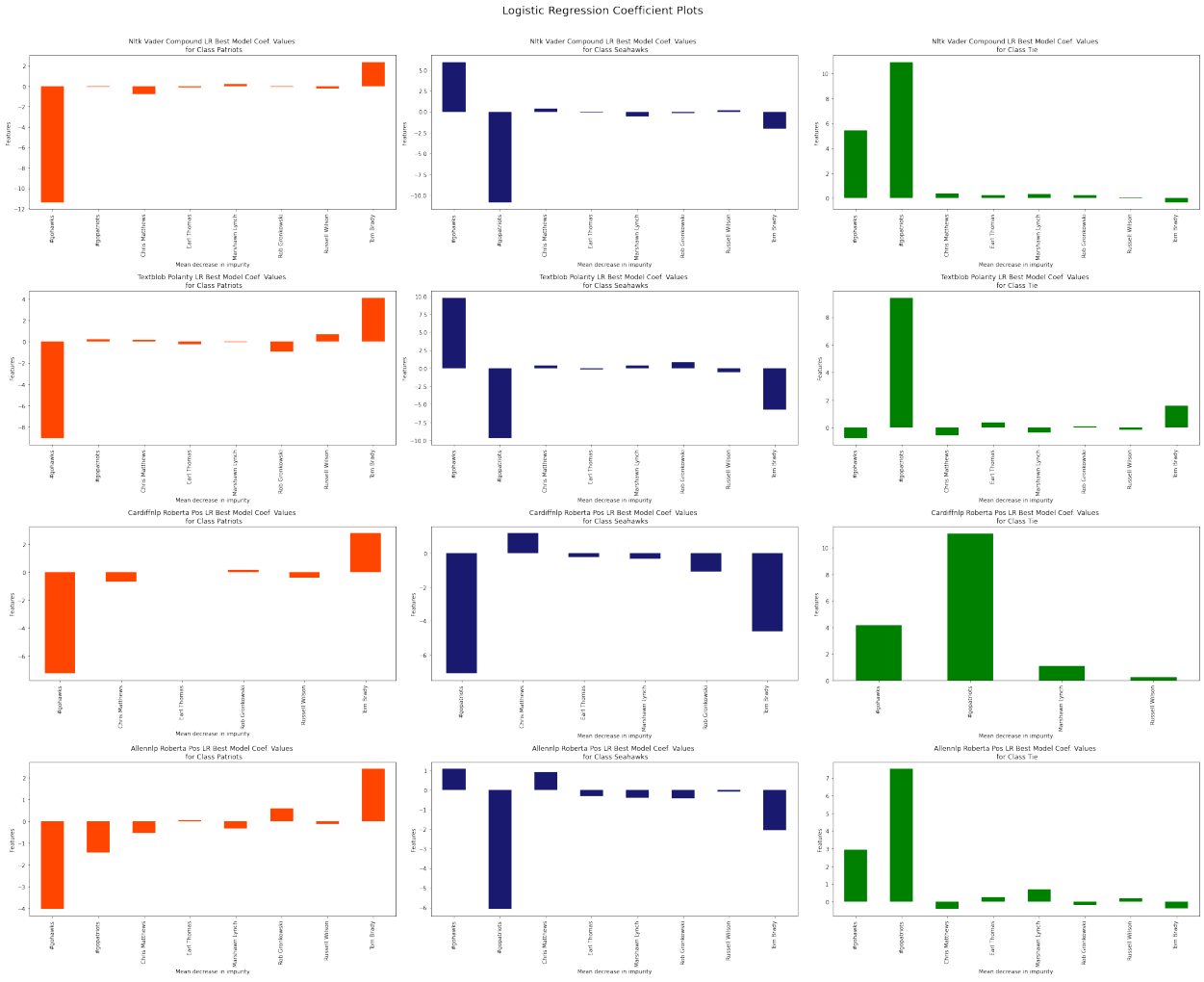
When we check AllenNLP RF feature importances, what I see is that the paired most important players shown in order. The first two most important players are Tom Brady and his counterpart Russell Wilson,

following them Marshawn Lynch and his Patriots counterpart Rob Gronkowski. TexBlob, seems to put more importance on Patriots' players.

Best Model we found was CardiffNLP's Random Forest: We can see that for this model the second most important feature and most important player is Tom Brady. Followed by, Chris Matthews, Marshawn Lynch, Rob Gronkowski and Russell Wilson.

Can we guess who might be the MVP of the game by looking at the Random Forests' importance for each model? All seems to point Tom Brady and indeed he was the MVP. If we look at the department of each player can we also guess who might be best Receiver and Defense player? For the CardiffNLP feature set, the second most important player is Chris Matthews, he is a receiver from Seahawks, when I check the Team Stats from ESPN site, he is indeed the Game Leader for Receiving Yards for Seahawks, and has one of the best stats makes him a good candidate for best receiver. The third importan player is Marshawn Lynch, who is also selected as Rushing Leader of the Game for his team and has better game stats than his Patriots counterpart, L. Blount.

LR coeff plots for each Sentiment type¶ In this part, we show the coefficient plots of each class for best LR models' on each feature type. I only show non-zero coefficients in the plot.



- We can see that in all 4 feature sets, Tom Brady get the highest positive coeff value for assigning Patriots as leading game, and highest negative coeff value for class Seahawks. For Tie assignment, Tom Brady is not as important.

Patriots Coefficients:

- In all 4 models, negative #gohawks and positive Tom Brady coeffs are found the most relevant informations for assigning Patriots as class. This means there is a negative correlation on positive overall positive Seahawks team and positive correlation on positive Tom Brady sentiments to predict Patriots leading the game. One interesting thing is that AllenNLP was able to catch correct players for each team: assigned positive coeff. values to Patriots players and negative coeff. values to Seahawks players

Seahawks Coefficients:

- We can see Tom Brady as the highest negative coeff. value for the players for all 4 LR model coefficients on assigning Seawhawks class. In all 4 models, the prediction is mainly driven on overall team emotions and Tom Brady. One interesting thing to notice is that both of the Roberta feature sets gave a higher positive value to Chris Matthews (and also only positive coeff. to a player). We can interpret this as perceived emotion against Chris Matthews is positively correlated in predicting the times when Seahawks leading the game and Chris Matthews played an important role as a receiver for Seahawks team for them to score.

Overall, from these plots, if we would guess and MVP it would be Tom Brady, who had high absolute value of coefficient amongs all players for each class. Chris Matthews can potentially be selected as best receiver for the Seahawks team.

Conclusion¶

I learned more about American Football, Patriots and Seahawks than I could ever imagine. Probably now I should go and try to see a game. The results for both Character-Centric Time-Series Tracking and Winner Prediction tasks were very interesting and full of insights. Fan based emotions about players in time windows are indeed good signals for understanding game status, scores and even the MVP of the game. Can we tell which feature type is better for the two tasks? For the Character-Centric Time-Series Tracking, both of the Roberta models offer more accurate and relevant correlations between scores and the emotions perceived by fans for players and teams. Rule-based nltk and textblob, seems to underperform compared the other two feature types, textblob seem to be the worst one, in this context and dataset. For Winner Prediction task in a given time interval, CardiffNLP seem to be the best performing feature set combined with a Random Forest model that leads almost 75% validation and 79.5% test accuracy.

This project should be taken as an preliminary exploration for the given tasks and can be improved in many different ways, we can deep dive even further in almost every step.

Appendix¶

Winner Prediction Experiments with different number of players¶

Results for experiment with 30% density requirement on columns¶

```
-- Info about data:

X shape: (216, 11)

'Class Distribution:'

Tie         98
Patriots    67
Seahawks    51
Name: leading, dtype: int64

'Class Distribution (normalized):'

Tie         0.453704
Patriots    0.310185
```

```
Seahawks    0.236111
Name: leading, dtype: float64

   Dropped columns where less than 30.0% is empty
    X shape after dropping columns: (216, 11)
        Remaining columns:                  ['#gohawks',
'#gopatriots', 'Chris Matthews', 'Earl Thomas', 'Jamie Collins',
'Julian Edelman', 'Marshawn Lynch', 'Richard Sherman', 'Rob
Gronkowski', 'Russell Wilson', 'Tom Brady']
    Filling NaN emotions scores with 0..
Model Experiments for : sent_nltk_vader_compound_1_mins data:
Model Experiments for : sent_textblob_polarity_1_mins data:
Model Experiments for : sent_cardiffnlp_roberta_pos_1_mins data:
Model Experiments for : sent_allennlp_roberta_pos_1_mins data:
```

Model Name Sentiment Feature	Mean Train Score LR	NN	RF	Mean Validation Score LR	NN	RF	Mean Test Score LR	NN	RF
Nltk Vader Compound	0.716	0.711	1.000	0.656	0.616	0.750	0.773	0.682	0.727
Textblob Polarity	0.660	0.713	0.988	0.623	0.605	0.727	0.705	0.614	0.705
Cardiffnlp Roberta Pos	0.738	0.650	0.983	0.675	0.583	0.750	0.727	0.727	0.773
Allennlp Roberta Pos	0.735	0.615	0.990	0.669	0.528	0.722	0.682	0.500	0.659

Results for experiment with 80% density requirement on columns¶

```
-- Info about data:

X shape: (216, 5)

'Class Distribution:'

Tie         98
Patriots    67
Seahawks    51
Name: leading, dtype: int64

'Class Distribution (normalized):'

Tie         0.453704
Patriots    0.310185
Seahawks    0.236111
Name: leading, dtype: float64

   Dropped columns where less than 80.0% is empty
    X shape after dropping columns: (216, 5)
        Remaining columns:                  ['#gohawks',
'#gopatriots', 'Rob Gronkowski', 'Russell Wilson', 'Tom Brady']
    Filling NaN emotions scores with 0..
Model Experiments for : sent_nltk_vader_compound_1_mins data:
Model Experiments for : sent_textblob_polarity_1_mins data:
Model Experiments for : sent_cardiffnlp_roberta_pos_1_mins data:
Model Experiments for : sent_allennlp_roberta_pos_1_mins data:
```

Model Name Sentiment Feature	Mean Train Score			Mean Validation Score			Mean Test Score		
	LR	NN	RF	LR	NN	RF	LR	NN	RF
Nltk Vader Compound	0.702	0.709	0.767	0.685	0.644	0.732	0.773	0.750	0.705
Textblob Polarity	0.686	0.694	0.860	0.669	0.658	0.692	0.727	0.682	0.682
Cardiffnlp Roberta Pos	0.745	0.682	0.770	0.721	0.645	0.751	0.750	0.523	0.682
Allennlp Roberta Pos	0.715	0.645	0.921	0.693	0.617	0.710	0.727	0.523	0.682