ichael Galietti, David Kotaev, Milad Mirghahari, Jesse Polanco
Professor Watson
CS301-006

https://github.com/flexadecimal/cs301-project

A) Project Title: Using Data to Build the Best Drag-Racing Car

B) Overview/ Summary: We have decided to delve deep into the understanding and correlation between torque and power in specific builds designed for drag racing and other automotive related activities. Drag racing is considered an expensive hobby and one should get an idea of exactly what they are getting themselves into, and at least be able to develop a plan. This development would allow the user to find exactly what would be available and limit the amount of options needed for comparison when going into such an expensive hobby. The goal of this group is to become proficient in data science techniques by utilizing multiple tools (i.e. pandas, numpy, etc), as well as getting the experience of taking a real world scenario and performing empirical analysis to design the perfect car for drag-racing while staying within an economic budget. The data set encompasses runs of data in relation to torque and horsepower of specific builds/mods in relation to turbos and fuel types. This project hopes to solidify the learning process for all involved, as well as inspire one's in the drag-racing industry to utilize data to achieve their goals more simply.

C) Problem Definition;

A. Question 1 - What data set would have the best 1/8th mile? 1/2 mile?
    a. In a short 1/8th mile drag race, you want the car with the most torque at the low end to pull off the line.

   b. A ½ mile, because of the longer distance traveled, horse power is more important than it was in a shorter, 1/2th mile, race.
   c. There are different factors that determine whether or not a car will perform well in an 1/8 mile race and a 1/2 mile race. Observing these differences will reveal the importance of torque and horsepower with certain race distances.

B. Question 2 - Which fuel type is the best "bang for the buck" in terms of power vs cost?
   a. Observing the statistics of HP/torque on gas types will reveal this.
   b. Pump vs e85 vs 100 octane/race fuel will all have different effects on power and cost, but will all cost different amounts per gallon.

C. Question 3 - Which of the most popular turbochargers have the most lag?
   a. There are many different turbos, but we will filter out the 10 most popular (amount of runs that were tested with X turbo)
   b. "Turbo lag" describes the wait time between a turbo spooling and horsepower delivery - bigger jump in power is more lag. Therefore, taking derivatives will let you rigorously compare lag. The output of this will be a list - #1 has least lag, #10 has the most.

D) Period of Analysis: February 14, 2020 to April 24, 2020

E/F) Contact Information and Documentation for Data Sets:

A. Dynomite - a collection of 3S dyno charts
   a. Original visualization of 3SI forum dyno charts

B. Dyno chart collection thread take three | Mitsubishi 3000GT & Dodge Stealth Forum
   a. 3SI forums dyno chart collection thread

G) Links to Data Sets:

A. Sunoco Race Fuels | Leaded & Unleaded for Sale

   a. 100 octane race fuel prices

B. Public E85 Ethanol stations and prices in the USA

   a. E85 prices and availability

H) Code:

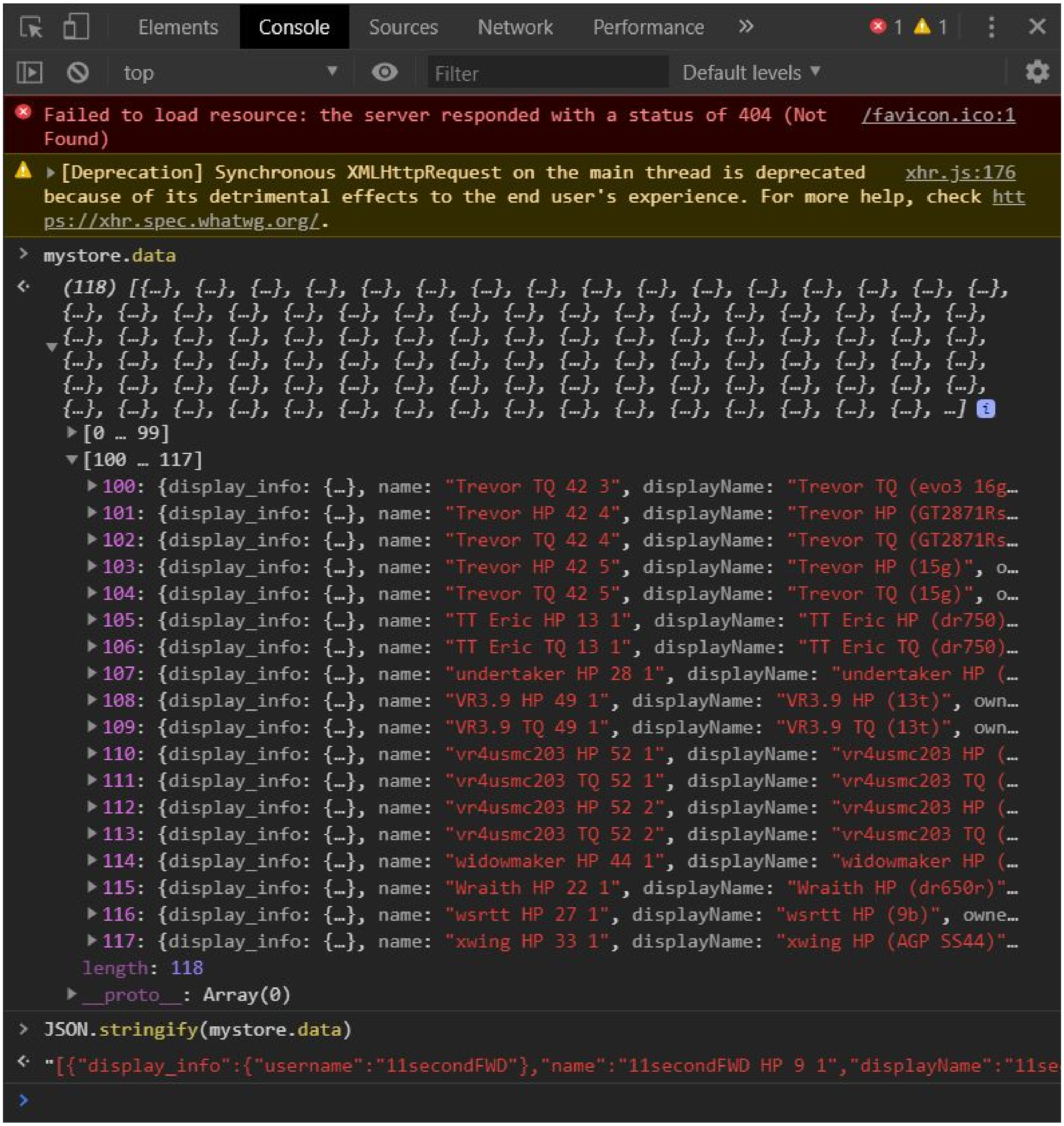
A. https://github.com/flexadecimal/cs301-project

   a. JSON data for all runs

2) Data Documentation Section 2: Methodology (Due 3/8 11:59PM EST)

a) Data Collection

i) The data was pulled from the dynomite site (http://www.primaryboost.com/3s/dynocharts/) by inspecting the local JavaScript data object in the console and converting it to JSON. The mods were manually typed in from number index to name as a dictionary.



The mods were received with a similar process.

```
> modstore.data
(12) [{…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}]
  0: {mod_type_id: "1", mod_name: "Boost", mod_default: "0", mod_type_type: "range", mod_checked: null}
  1: {mod_type_id: "2", mod_name: "Fuel", mod_default: "pump gas", mod_type_type: "fixed", mod_checked: null}
  2: {mod_type_id: "3", mod_name: "Liter", mod_default: "3.0", mod_type_type: "fixed", mod_checked: null}
  3: {mod_type_id: "4", mod_name: "Turbo Class", mod_default: "0", mod_type_type: "fixed", mod_checked: null}
  4: {mod_type_id: "5", mod_name: "Turbo Name", mod_default: "0", mod_type_type: "fixed", mod_checked: null}
  5: {mod_type_id: "6", mod_name: "Other", mod_default: "0", mod_type_type: "fixed", mod_checked: null}
  6: {mod_type_id: "7", mod_name: "Injection", mod_default: "0", mod_type_type: "boolean", mod_checked: null}
  7: {mod_type_id: "8", mod_name: "Nitrous", mod_default: "0", mod_type_type: "boolean", mod_checked: null}
  8: {mod_type_id: "9", mod_name: "Correction", mod_default: "0", mod_type_type: "fixed", mod_checked: null}
  9: {mod_type_id: "10", mod_name: "Heads", mod_default: "0", mod_type_type: "fixed", mod_checked: null}
  10: {mod_type_id: "11", mod_name: "Max HP", mod_default: "0", mod_type_type: "hidden", mod_checked: null}
  11: {mod_type_id: "12", mod_name: "Max TQ", mod_default: "0", mod_type_type: "hidden", mod_checked: null}
  length: 12
  __proto__: Array(0)
> JSON.stringify(modstore.data)
"[{"mod_type_id":"1","mod_name":"Boost","mod_default":"0","mod_type_type":"range","mod_checked":null},{"mod_type_id
```

ii) Screenshots of the df.head() of the DataFrame(s) as seen below;

```
trials = pd.read_json('./dynomite_trials.json')
# mods table
mods = pd.read_json('./dynomite_mods.json')
# split into hp/torque trials
hp_trials = trials.loc[trials['data_type_id'] == 1]
torque_trials = trials.loc[trials['data_type_id'] == 2]
trials.head()
```

	display_info	name	displayName	owner_id	car_id	run_id	data_type_id	mods	run_data
0	{'username': '11secondFWD'}	11secondFWD HP 9 1	11secondFWD HP (17g)	9	9	1	1	{'4': 'td04', '5': '17g'}	[{'x': 2510, 'y': 53}, {'x': 2921, 'y': 107}, ...
1	{'username': '11secondFWD'}	11secondFWD TQ 9 1	11secondFWD TQ (17g)	9	9	1	2	{'4': 'td04', '5': '17g'}	[{'x': 2514, 'y': 118}, {'x': 2619, 'y': 143},...
2	{'username': '2root4u'}	2root4u HP 18 1	2root4u HP (gt368)	18	18	1	1	{'1': '25', '4': 'td04', '5': 'gt368'}	[{'x': 3564, 'y': 210}, {'x': 3942, 'y': 262},...
3	{'username': '3sx'}	3sx HP 6 1	3sx HP (GT35)	6	6	1	1	{'4': 'td05', '5': 'GT35', '8': '100'}	[{'x': 4200, 'y': 200}, {'x': 4800, 'y': 370},...
4	{'username': '97vr4'}	97vr4 HP 15 1	97vr4 HP (19t)	15	15	1	1	{'1': '25', '4': 'td04', '5': '19t'}	[{'x': 3457, 'y': 201}, {'x': 3686, 'y': 244},...

```
trials = pd.read_json('./dynomite_trials.json')
# mods table
mods = pd.read_json('./dynomite_mods.json')
# split into hp/torque trials
hp_trials = trials.loc[trials['data_type_id'] == 1]
torque_trials = trials.loc[trials['data_type_id'] == 2]
mods.head()
```

	mod_type_id	mod_name	mod_default	mod_type_type	mod_checked
0	1	Boost	0	range	NaN
1	2	Fuel	pump gas	fixed	NaN
2	3	Liter	3.0	fixed	NaN
3	4	Turbo Class	0	fixed	NaN
4	5	Turbo Name	0	fixed	NaN

iii) The data set chosen was found via online as previously stated in section i, but there were minor custom changes to the data we have incorporated. The initial data frame though informative had some slight aesthetic issues i.e. the mods column of our data frame. The mods were also retrieved using a similar process as we did for the horsepower and torque trials.

b) **Column Descriptions**

(1) **Display_Info:** A key value pair that shows the value of the username this will coincide with other columns with provide unique designators. Example: {"username", username}

Percent of data populated: (100.0%)

Data Type: Categorical

(2) **Name:** Direct designator for the username coupled with designators of the columns displayName, owner_id and run_id. Example: 11secondFWD HP 9 1

Percent of data populated: (100.0 %)

Data Type: Categorical

(3) **displayName:** Designator which provides the username then either a HP or TQ for horsepower or torque followed by the mod type. Example: 11secondsFWD HP (17g)

Percent of data populated: (100.0 %)

Data Type: Categorical

(4) **owner_id:** 51 unique designations to represent the username and is shown in the name column and can be used to quickly reference a certain user when looking at either the horsepower data set or the torque data set per run.

Percent of data populated: (100.0 %)

Data Type: Ordinal

(5) **car_id:** 55 unique designations to represent the car used and is shown in the name column and can be used to quickly reference a certain car when looking at either the horsepower data set or the torque data set per run as well as the mod difference i.e. type of gas used. The difference in unique values represents the fact unique users have multiple cars.

Percent of data populated: (100.0 %)

Data Type: Ordinal

(6) **run_id:** Depicts the iteration of a run with the same configuration, though not many but aren't considered duplicates. This allows us to see and confirm multiple values per user as well as the amount of times they tested via dyno.

Percent of data populated: (100.0 %)

Data Type: Ordinal

(7) **data_type_id:** A binary data based column with mutually exclusive values of either 1 for horsepower(HP) data or 2 for torque(TQ) data.

Percent of data populated: (100.0 %)

Data Type: Ordinal

(8) **mods:** Column with a dictionary of key value pairs of the mod_type_id and the value being the name of the specific mod_type_id. Example:{'4':'td04', '5',17g'} would be the Turbo Class for the '4' and the Turbo Name for the '5' designation. Below is a reference chart for the other possibilities of mod_type_id and some further default information associated with it.

Percent of data populated: (100.0 %)

Data Type: Qualitative

	mod_type_id	mod_name	mod_default	mod_type_type
0	1	Boost	0	range
1	2	Fuel	pump gas	fixed
2	3	Liter	3.0	fixed
3	4	Turbo Class	0	fixed
4	5	Turbo Name	0	fixed
5	6	Other	0	fixed
6	7	Injection	0	boolean
7	8	Nitrous	0	boolean
8	9	Correction	0	fixed
9	10	Heads	0	fixed
10	11	Max HP	0	hidden
11	12	Max TQ	0	hidden

(9) **run_data:** Lists of x,y points that show the progression of either horsepower or torque in regards to the runs. It depicts the first two points of the run. Example [{‘x’:2510, ‘y’:53},{‘x’:2921,’y’:107},....

Percent of data populated: (100.0 %)

Data Type: Qualitative

c) Data Processing

Within the data set we’ve received we faced a few specific issues that made it more difficult to determine the necessary statistics needed. To perform statistical summary on our data set, we had to find an aggregate function that made sense for each dyno chart. We chose to

integrate these in order to achieve the goal of determining power behind each automobile. We had to create bound-points for our data entries. We chose the median start and end points of ~3000 RPM and ~7000 RPM, and then integrated each run, tossing out data points that were outside the bounds.

```
# for torque
torque_start, torque_end = summarize_trials_bounds(torque_trials['run_data'], 'torque')
print('Integration bounds (median q2): ({}, {})'.format(torque_start, torque_end))
torque_trials[integral_col_name] = torque_trials['run_data'].apply(lambda r: definite_discrete_int
egral(r, torque_start, torque_end))
torque_trials.head()
```

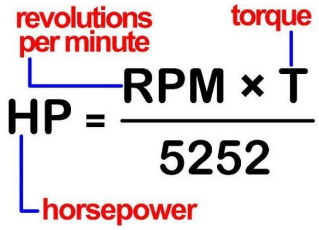
```
torque start RPM summary: {'min': 1916, 'q1': 2400.0, 'q2': 2536.0, 'q3': 2882.0, 'max': 3300, 'ou
tliers': [1012]}
torque end RPM summary: {'min': 6399, 'q1': 6758.0, 'q2': 6961.0, 'q3': 7100.0, 'max': 7563, 'outl
iers': [7950, 6103]}
Integration bounds (median q2): (2536.0, 6961.0)
```

```
integral_col_name = "total_power_normalized"

# for horsepower
hp_start, hp_end = summarize_trials_bounds(hp_trials['run_data'], 'horsepower')
print('Integration bounds (median q2): ({}, {})'.format(hp_start, hp_end), end='\n\n')
# now do the integrals and add to dataframe
hp_trials[integral_col_name] = hp_trials['run_data'].apply(lambda r: definite_discrete_integral(r,
hp_start, hp_end))
hp_trials.head()
```

```
horsepower start RPM summary: {'min': 1020, 'q1': 2510.0, 'q2': 3065.0, 'q3': 3603.0, 'max': 4672,
'outliers': None}
horsepower end RPM summary: {'min': 5988, 'q1': 6639.0, 'q2': 6965.0, 'q3': 7158.0, 'max': 7544, '
outliers': [8500, 7950, 8011, 8463, 8468, 8072, 8015]}
Integration bounds (median q2): (3065.0, 6965.0)
```

Although we lose some crucial data by performing this process (outliers, start and end rates of change, etc.), these lost data points do not contribute to the overall objective we seek to deliver. There are slight differences in the bounds for torque and horsepower, as both have a different median start/end value. There is a direct relationship between the horsepower and torque as seen below;

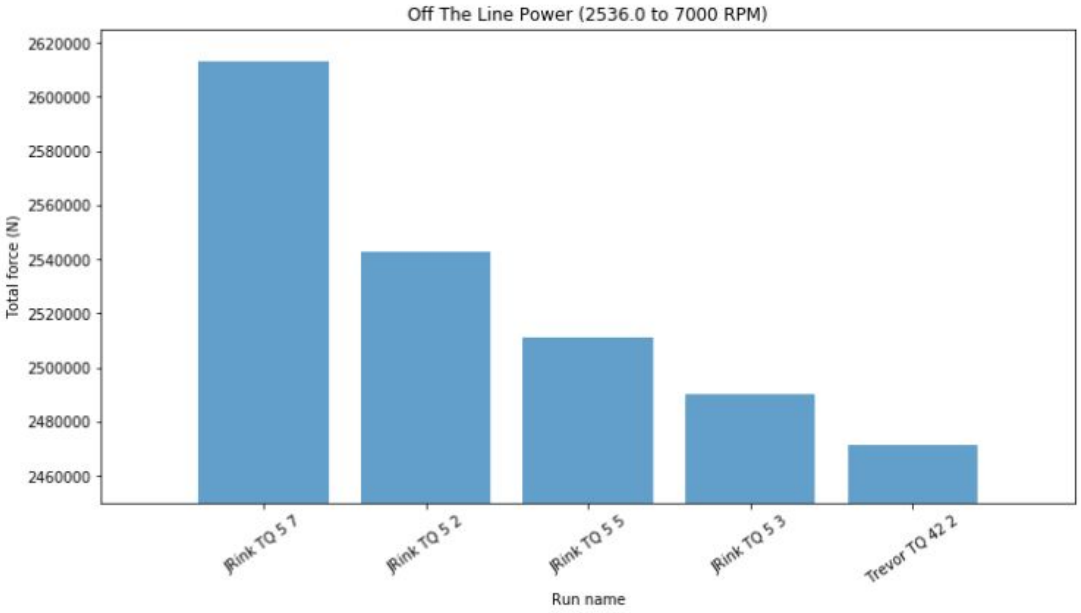


This equation will not be used throughout (as we are using integrations for our data), although it is essential to understand how torque and horsepower are directly related. Each car will start and end at a different RPM, but in drag-racing we care specifically about the power of the car. The data received does not face an issue in being outdated, as all of the cars are relative cars that are seen on the streets today. The data set also contains no missing entries, although it does contain entries that could be argued as duplicates. These entries are indeed not duplicates, as the car will not perform at the same exact metrics at any given run-time. Just as a runner will never get their exact sprint time twice, a car will perform at different performance metrics for each run. There were no incorrect values, spelling errors, missing entries, or column errors found within the data set. Overall, the dataset we received seems to have been a complete dataset, although adjusting the bounds and including multiple runs per vehicle type is necessary as it allows us to have a much more concentrated form of data.

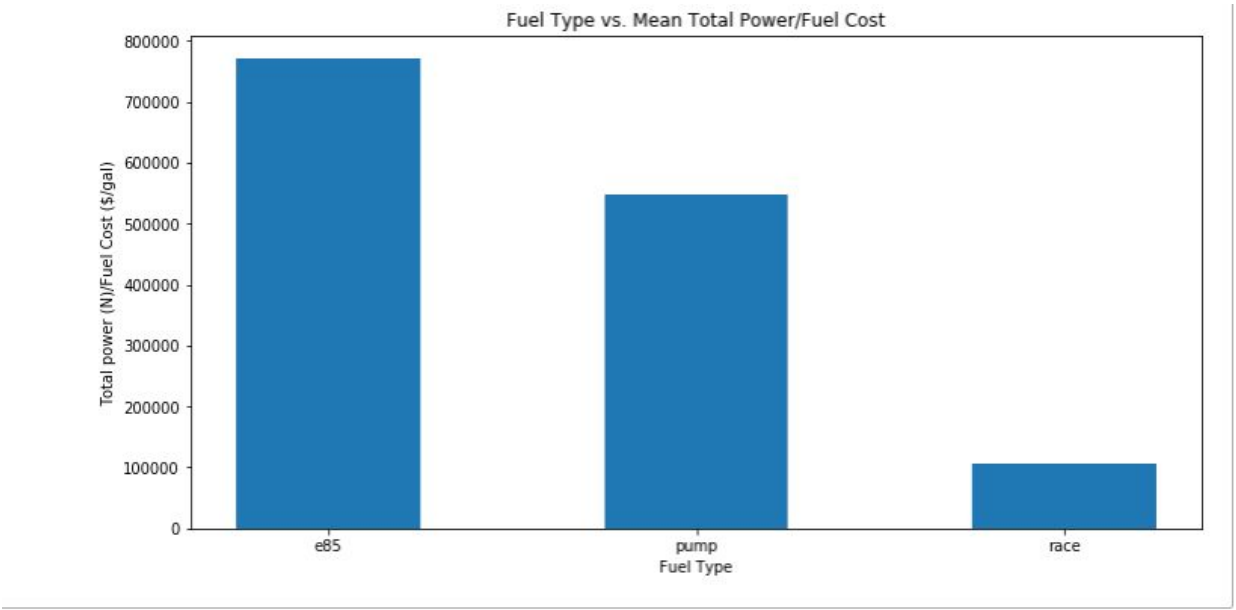
Deliverable #3

To answer the questions we posed earlier, filtered the relevant trial data to specification using a filter_by_mods function. Then we used aggregate functions like the integral to calculate total power over an RPM range for the drag racing ⅛ and ½ mile comparisons.

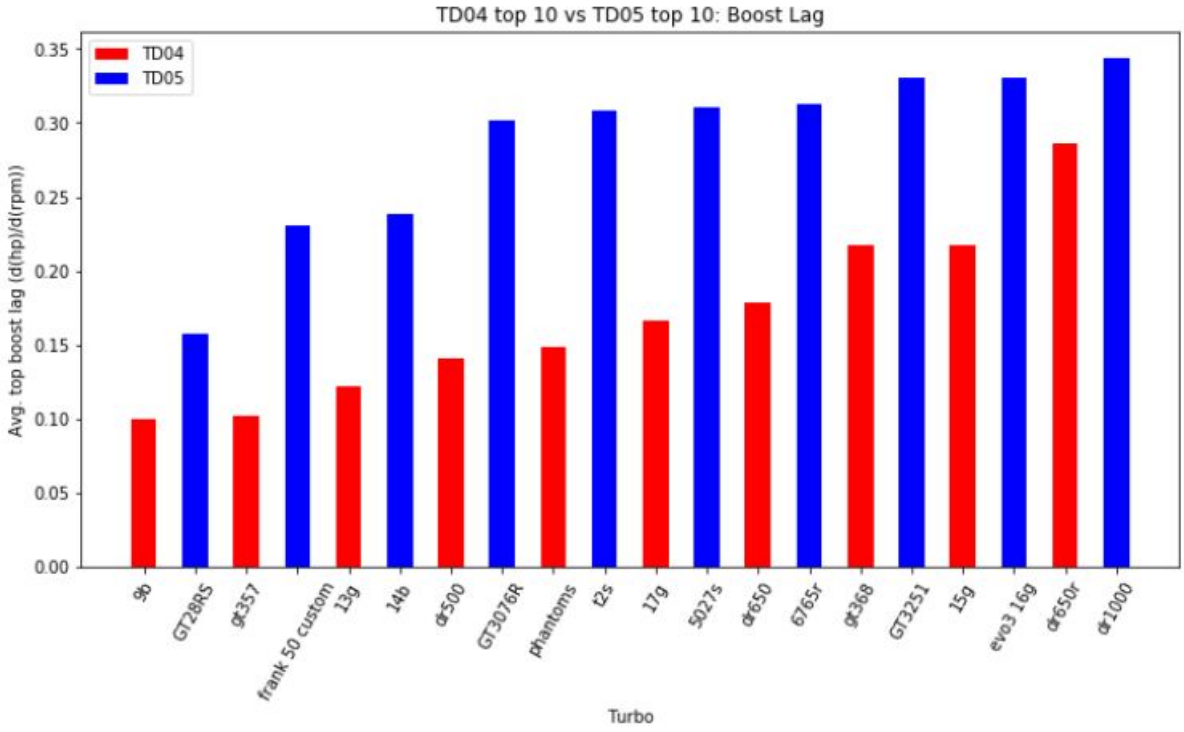
To answer which trial setup would be the best in an 1/8th and 1.2 mile drag race we calculated the total power over a range from stating, as a quantitative measure of how much power each car would make at the beginning of a race.



For the second question, we rated the fuels by taking a ratio of the total power that set of comparable runs had (similar boost, displacement specs, stock engine) over the fuel cost.



For the third question, we took a gradient of the torque curve - a set of derivatives at each given point - and took the maximum derivative. This is a quantitative measure of the jump in power delivery commonly known as “turbo lag”. Turbochargers in different classes are compared to the average worst “bump” in the power curve - lower values are better.



See https://github.com/flexadecimal/cs301-project/blob/master/deliverables.ipynb for source and full statistical values (min/max/quartile/outliers) for each step.