CA4010 Data Warehousing & Data Mining Final Report - Group 5

Adam Gray - 20364103

Nacé van Wyk - 20467546

Idea and dataset description

The idea for our project is to predict the Average Pit Stop Time's for each track during the 2019 season of the Formula 1 World Championship. To do this we found a dataset on Kaggle called **Formula 1 Race Data** which contains information relating to the pit stops during the seasons from 2011-2018, which is also extremely beneficial to use as this period saw little to no changes in the pit stop regulations, and such should provide a consistent basis for our data analysis.

The dataset consists for 13 `.csv` files each containing all different information about aspects of a given Formula 1 season, containing different data from season's from the first in 1950 all the way to 2018, however as stated, the pit stop information was only available from 2011 to 2018. Some files included were `circuits.csv`, containing information about all the circuits that have held a race since 1950 and `pitStops.csv` which contains the times of each pit stop during the given season.

Looking at the dataset initially we were unsure as to how we would be able to extract the data we needed, as certain `.csv` files within the dataset did not contain all the fields we would need to effectively use an algorithm to make a prediction. However, upon closer inspection we realised with some data integration we could combine certain `.csv` files to achieve the refined dataset we were looking for.

We decided that by using 3 of the `.csv` files, (`circuits.csv`, `races.csv` and `pitStops.csv`) we could firstly tie all races to their given track, so one track would link to many races, and from here, each race would link to all of it's pit stops, again one to many respectively. By doing this we would then have each race contained to itself and help in having no data crossover.

circuits.csv

The circuits.csv contains useful information such as the `circuitId` and `circuitName` that can be used to help identify which track we are working with. We also need it to extract the name of the circuit at the end to visualise our result.

races.csv

The races.csv file will be used to help us link the `raceId` to the `circuitId` as the file contains both columns for us to use.

pitStops.csv

This is our most important `.csv` file as it contains the actual times of the pit stops for each race, both in seconds and milliseconds. It also will let us link all of the pit stops to the races they happened at due to it also containing the `raceId` column.

Data Preparation

Data Integration

The 3 main `.csv` files we're using are shown below using a snippet of each:

circuits.csv

```
circuitId,circuitRef,name,location,country,lat,lng,alt,url
1,albert_park,Albert Park Grand Prix Circuit,Melbourne,Australia,-37.8497,144.968,10,http://en.wikipedia.org/wiki/Melbourne_Grand_Prix_Circuit
2,sepang,Sepang International Circuit,Kuala Lumpur,Malaysia,2.76083,101.738,,http://en.wikipedia.org/wiki/Sepang_International_Circuit
3,bahrain,Bahrain International Circuit,Sakhir,Bahrain,26.0325,50.5106,,http://en.wikipedia.org/wiki/Bahrain_International_Circuit
4,catalunya,Circuit de Barcelona-Catalunya,Montmeló,Spain,41.57,2.26111,,http://en.wikipedia.org/wiki/Circuit_de_Barcelona-Catalunya
5,istanbul,Istanbul Park,Istanbul,Turkey,40.9517,29.405,,http://en.wikipedia.org/wiki/Istanbul_Park
6,monaco,Circuit de Monaco,Monte-Carlo,Monaco,43.7347,7.42056,,http://en.wikipedia.org/wiki/Circuit_de_Monaco
7,villeneuve,Circuit Gilles Villeneuve,Montreal,Canada,45.5,-73.5228,,http://en.wikipedia.org/wiki/Circuit_Gilles_Villeneuve
8,magny_cours,Circuit de Nevers Magny-Cours,Magny Cours,France,46.8642,3.16361,,http://en.wikipedia.org/wiki/Circuit_de_Nevers_Magny-Cours
9,silverstone,Silverstone Circuit,Silverstone,UK,52.0786,-1.01694,,http://en.wikipedia.org/wiki/Silverstone_Circuit
10,hockenheimring,Hockenheimring,Hockenheim,Germany,49.3278,8.56583,,http://en.wikipedia.org/wiki/Hockenheimring
11,hungaroring,Hungaroring,Budapest,Hungary,47.5789,19.2486,,http://en.wikipedia.org/wiki/Hungaroring
12,valencia,Valencia Street Circuit,Valencia,Spain,39.4589,-0.331667,,http://en.wikipedia.org/wiki/Valencia_Street_Circuit
13,spa,Circuit de Spa-Francorchamps,Spa,Belgium,50.4372,5.97139,,http://en.wikipedia.org/wiki/Circuit_de_Spa-Francorchamps
14,monza,Autodromo Nazionale di Monza,Monza,Italy,45.6156,9.28111,,http://en.wikipedia.org/wiki/Autodromo_Nazionale_Monza
15,marina_bay,Marina Bay Street Circuit,Marina Bay,Singapore,1.2914,103.864,,http://en.wikipedia.org/wiki/Marina_Bay_Street_Circuit
```

From this `.csv` file we're going to be using the `circuitId` field as well as eventually using the `name` field too to help with displaying our data.

races.csv

```
raceId,year,round,circuitId,name,date,time,url
1,2009,1,1,Australian Grand Prix,2009-03-29,06:00:00,http://en.wikipedia.org/wiki/2009_Australian_Grand_Prix
2,2009,2,2,Malaysian Grand Prix,2009-04-05,09:00:00,http://en.wikipedia.org/wiki/2009_Malaysian_Grand_Prix
3,2009,3,17,Chinese Grand Prix,2009-04-19,07:00:00,http://en.wikipedia.org/wiki/2009_Chinese_Grand_Prix
4,2009,4,3,Bahrain Grand Prix,2009-04-26,12:00:00,http://en.wikipedia.org/wiki/2009_Bahrain_Grand_Prix
5,2009,5,4,Spanish Grand Prix,2009-05-10,12:00:00,http://en.wikipedia.org/wiki/2009_Spanish_Grand_Prix
6,2009,6,6,Monaco Grand Prix,2009-05-24,12:00:00,http://en.wikipedia.org/wiki/2009_Monaco_Grand_Prix
7,2009,7,5,Turkish Grand Prix,2009-06-07,12:00:00,http://en.wikipedia.org/wiki/2009_Turkish_Grand_Prix
8,2009,8,9,British Grand Prix,2009-06-21,12:00:00,http://en.wikipedia.org/wiki/2009_British_Grand_Prix
9,2009,9,20,German Grand Prix,2009-07-12,12:00:00,http://en.wikipedia.org/wiki/2009_German_Grand_Prix
10,2009,10,11,Hungarian Grand Prix,2009-07-26,12:00:00,http://en.wikipedia.org/wiki/2009_Hungarian_Grand_Prix
11,2009,11,12,European Grand Prix,2009-08-23,12:00:00,http://en.wikipedia.org/wiki/2009_European_Grand_Prix
12,2009,12,13,Belgian Grand Prix,2009-08-30,12:00:00,http://en.wikipedia.org/wiki/2009_Belgian_Grand_Prix
13,2009,13,14,Italian Grand Prix,2009-09-13,12:00:00,http://en.wikipedia.org/wiki/2009_Italian_Grand_Prix
14,2009,14,15,Singapore Grand Prix,2009-09-27,12:00:00,http://en.wikipedia.org/wiki/2009_Singapore_Grand_Prix
15,2009,15,22,Japanese Grand Prix,2009-10-04,05:00:00,http://en.wikipedia.org/wiki/2009_Japanese_Grand_Prix
16,2009,16,18,Brazilian Grand Prix,2009-10-18,16:00:00,http://en.wikipedia.org/wiki/2009_Brazilian_Grand_Prix
17,2009,17,24,Abu Dhabi Grand Prix,2009-11-01,11:00:00,http://en.wikipedia.org/wiki/2009_Abu_Dhabi_Grand_Prix
18,2008,1,1,Australian Grand Prix,2008-03-16,04:30:00,http://en.wikipedia.org/wiki/2008_Australian_Grand_Prix
19,2008,2,2,Malaysian Grand Prix,2008-03-23,07:00:00,http://en.wikipedia.org/wiki/2008_Malaysian_Grand_Prix
20,2008,3,3,Bahrain Grand Prix,2008-04-06,11:30:00,http://en.wikipedia.org/wiki/2008_Bahrain_Grand_Prix
```

From the `races.csv` file we will need to use both the `raceId`, to keep track of which race the pit stops take place at. As well as this, we will need to use the `circuitId` field to map every race to every track it happened at.

pitStops.csv

```
raceId,driverId,stop,lap,time,duration,milliseconds
841,153,1,1,17:05:23,26.898,26898
841,30,1,1,17:05:52,25.021,25021
841,17,1,11,17:20:48,23.426,23426
841,4,1,12,17:22:34,23.251,23251
841,13,1,13,17:24:10,23.842,23842
841,22,1,13,17:24:29,23.643,23643
841,20,1,14,17:25:17,22.603,22603
841,814,1,14,17:26:03,24.863,24863
841,816,1,14,17:26:50,25.259,25259
841,67,1,15,17:27:34,25.342,25342
841,2,1,15,17:27:41,22.994,22994
841,1,1,16,17:28:24,23.227,23227
841,808,1,16,17:28:39,24.535,24535
841,3,1,16,17:29:00,23.716,23716
841,155,1,16,17:29:06,24.064,24064
841,16,1,16,17:29:08,25.978,25978
841,15,1,16,17:29:49,24.899,24899
841,18,1,17,17:30:24,16.867,16867
841,153,2,17,17:31:06,24.463,24463
```

Finally, and arguably the most important file, we need to use the `raceId` to map the pit stop times in `milliseconds` to the race that the pit stop happened at.

We now simply want to merge all the 3 fields together, firstly merging `pitStops.csv` and `races.csv` on the `raceId`, followed by merging the result of that with the `circuits.csv` on the `circuitId` field. To do this we used python, as well as the pandas library which is very popular for data analysis, as seen below, we simply merged all 3 sets of data, and then only kept the relevant fields of data.

```
import pandas as pd

"""
we want to read in the 3 datasets and merge them to keep the following columns:
    raceId, milliseconds, circuitId, name_y
"""

mergedData = (
    pd.read_csv("pitStops.csv", encoding="ISO-8859-1")
    .merge(pd.read_csv("races.csv", encoding="ISO-8859-1"), on="raceId")
    .merge(pd.read_csv("circuits.csv", encoding="ISO-8859-1"), on="circuitId")
)

mergedData = mergedData[["raceId", "milliseconds", "circuitId", "name_y"]]
```

By doing this we get the following initial result as a merged dataset we can work with:

	raceId	milliseconds	circuitId	name_y
0	841	26898	1	Albert Park Grand Prix Circuit
1	841	25021	1	Albert Park Grand Prix Circuit
2	841	23426	1	Albert Park Grand Prix Circuit
3	841	23251	1	Albert Park Grand Prix Circuit
4	841	23842	1	Albert Park Grand Prix Circuit
...	...	...	...	...
6246	976	29317	73	Baku City Circuit
6247	976	20446	73	Baku City Circuit
6248	976	14951	73	Baku City Circuit
6249	976	14943	73	Baku City Circuit
6250	976	24486	73	Baku City Circuit

As seen above, we now have all 6250 pit stops during the 2011-2018 season, with each containing the race it happened at, the time in milliseconds the pit stop took, as well as which track the race was at, and finally the name of the track so we can use it later on to display the information visually.

Measuring the Central Tendency of the Data

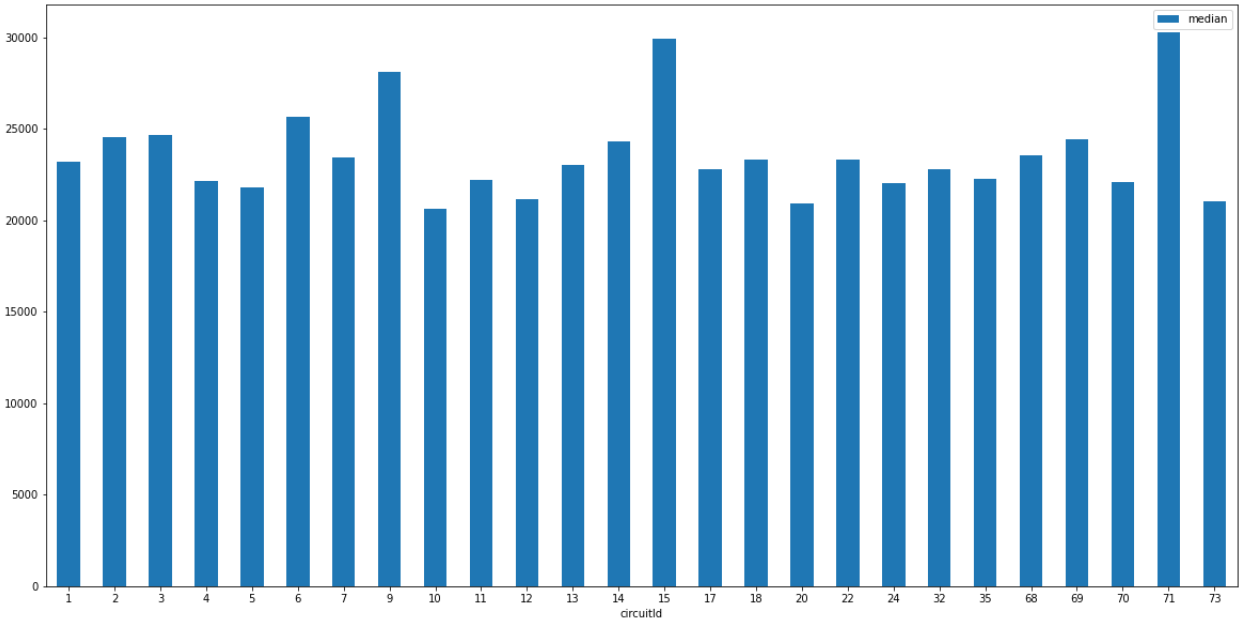
Once we had integrated our data, we decided it would be good to measure the central tendency of the data, to see if our data contained many outliers, as we expected the majority of pit stops to fall within the 20 second to 30 second range, with most being closer to the 20 second side. We did this again using python this time using the `mathplotlib` library which is "***a comprehensive library for creating static, animated, and interactive visualizations in Python".***

We firstly decided to take a measure of the mean and the median, and plot them visually to give us a better understanding of what our data would look like when trying to find out where the mid point of each track's timing's would lie, to do this we did the following:

```
import sys
!{sys.executable} -m pip install matplotlib
import matplotlib.pyplot as plt

groupedMed = mergedData.groupby('circuitId')['milliseconds']
  .agg(['median']).reset_index()

groupedMed.plot
  .bar(x='circuitId', y='median', rot=0, figsize=(20,10))
```

Note: the use of a command to install matplotlib in case it is not installed on the machine it is ran on inside of our Jupyter Notebook.

We firstly grouped the dataset we has created before using the `circuitId` column, so within each group we could then calculate the median of the `milliseconds` values. After that, we then used the `plot()` function to show all the data visually labelling both axis appropriately. After running this code we received the following graph as output:

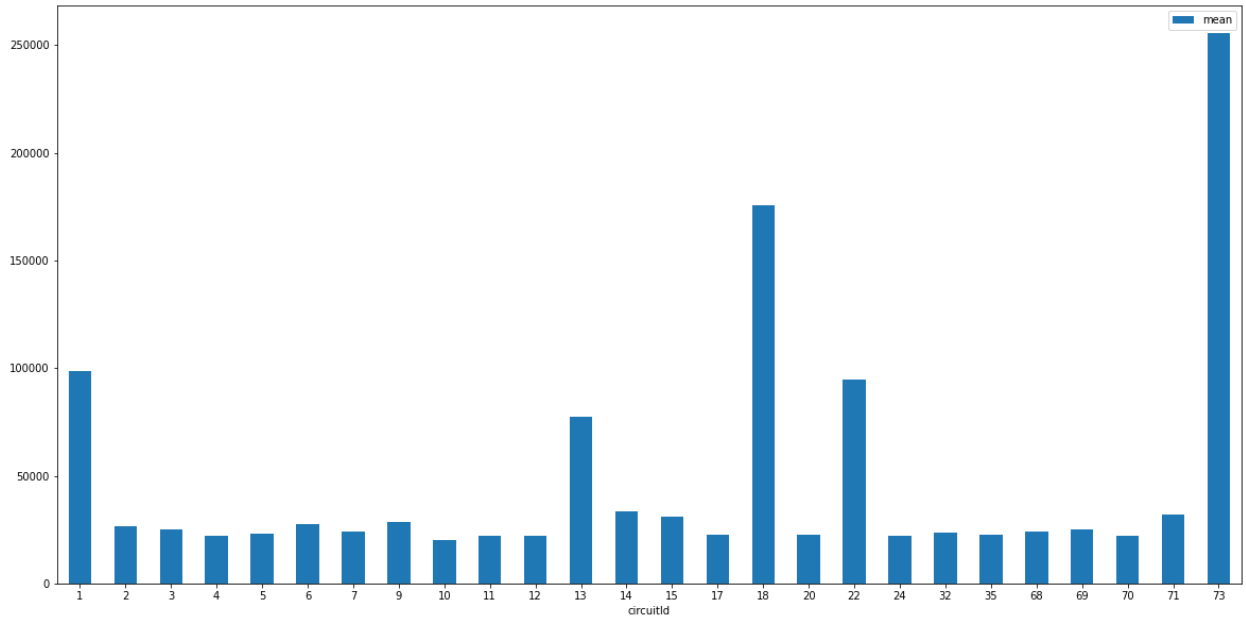


As can be seen from this above graph, as expected, we found that, (although there were some tracks falling a little close to below the 20 second mark, and some being very close to the 30 second mark, indicating some outliers were present) a majority of the tracks were exactly within the range we expected, with the majority being between 20 to 25 seconds in duration. At first, we thought that this would mean our dataset was very well cleaned and corrected, however, we soon found this was not the case.

Upon taking the measure of the mean, again to give us an understanding of where the central point of the data for each track lay, and so we had one holistic and one arithmetic measure of the central tendency of our data. We again used some python code to find the mean:

```
groupedMean = mergedData.groupby('circuitId')['milliseconds'].agg(['mean']).reset_index()

groupedMean.plot.bar(x='circuitId', y='mean', rot=0, figsize=(20,10))
```

Much like with the median code, we again grouped the data by the `circuitId` column and got the mean of the `milliseconds` column, and finally displayed the data again for us to see, as shown in the diagram below:



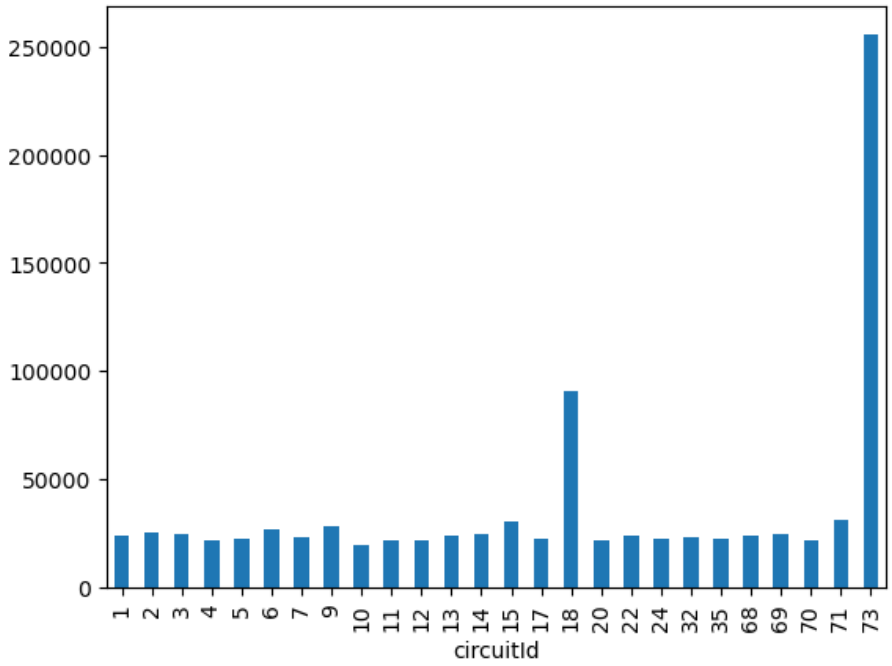
Clearly, as can be seen this time from the diagram, we have some extreme outliers, with the `milliseconds` column no longer being in the tens of thousand's, but rather the hundreds of thousands (over 4 minute average pit stop time). From this we knew immediately we had some investigation to do to find out what was causing this issue to occur, and upon doing this, we noticed we had some skewed data in our dataset.

Firstly we examined our code quickly to make sure we had no errors in our approach to the solution, and after reading some documentation and confirming we were correct we then could deduce that the data was definitely skewed and needed to be investigated further. Upon opening the `pitStops.csv` file and navigating to the races with `circuitId` 73, we found in our search one of the tracks had a major issue with on of it's races, as seen in the image below:

```
5474    967,1,1,20,14:47:37,33:30.361,2010361
5475    967,3,1,20,14:47:40,33:29.747,2009747
5476    967,830,2,20,14:47:41,33:29.643,2009643
5477    967,807,1,20,14:47:44,33:28.504,2008504
5478    967,815,1,20,14:47:46,33:28.398,2008398
5479    967,832,1,20,14:47:48,33:28.828,2008828
5480    967,831,1,20,14:47:50,33:28.206,2008206
5481    967,817,2,20,14:47:52,33:28.600,2008600
5482    967,839,1,20,14:47:53,33:29.052,2009052
5483    967,836,1,20,14:47:56,33:28.464,2008464
5484    967,4,2,20,14:47:57,33:28.789,2008789
```

As can be seen by the final two columns, showing `duration` and `milliseconds` respectively, the pit stop times are all over 30 minutes, which when we check the actual pit stop times for that race (information can be found here), the 2016 Brazilian Grand Prix, and we calculate the mean, again using the method above, but with this smaller set of data, we find that it is approximately 25.5 seconds.

After these revelations proved to us we had some dirty data, we tried to find the trimmed mean of the data, using a value of 2%, as we did not want to reduce our data much to affect our other results, as well as the course book suggesting this, we hoped that this may solve our issue, the result is seen below:

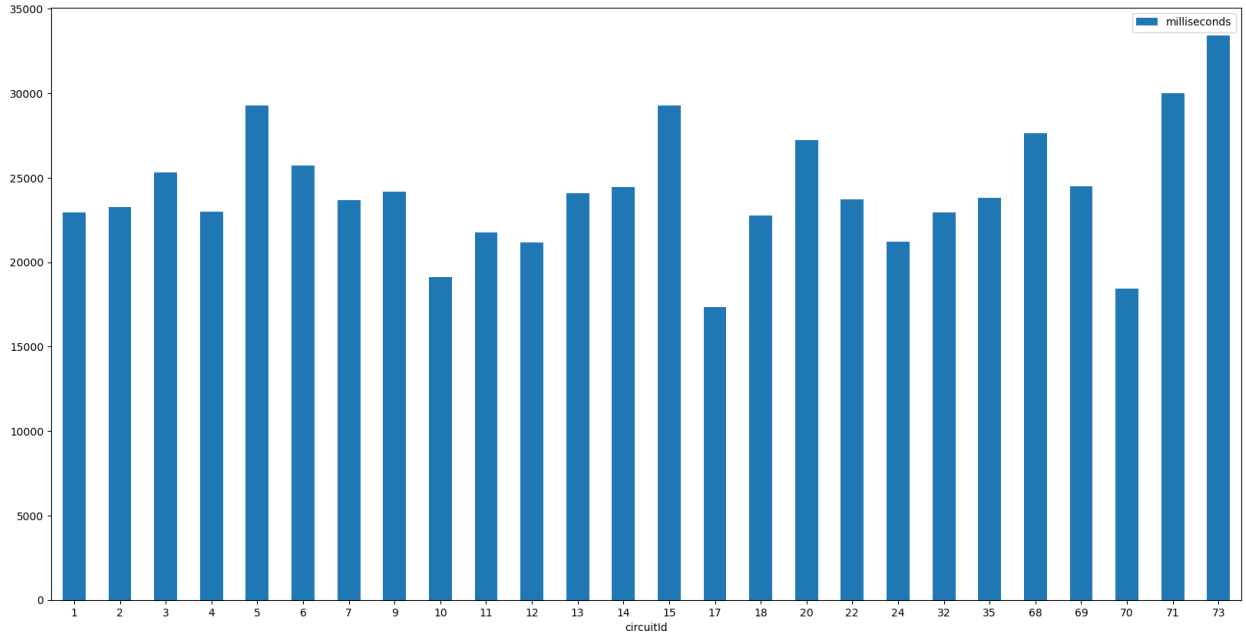


As can be seen by the result, this method did not entirely work, as too many of the values were of too high a number for certain tracks meaning that even by reducing the size of our data by 2%, we only removed some outliers, but not all, because of this, we did not think this was a good solution, and decided that although getting the trimmed mean helped us see this, we would not continue with the trimmed dataset, and would instead focus on cleaning the dataset instead going forward (more about this in the Data Cleaning section of the report).

Finally we went about finding the mode of each of the circuits, again with python I found the mode of the data and graphed it:

```
groupedMode = mergedData.groupby('circuitId')['milliseconds'].agg(lambda x: x.value_counts().index[0]).reset_index()

groupedMode.plot.bar(x='circuitId', y='milliseconds', rot=0, figsize=(20,10))
```

As seen, this time we needed to use the lambda function to count the most common occurrence of a given time in milliseconds, and extracting it to use in the graph, which is shown below:



Much like finding the median, we see that a majority of values fall between the 20 to 30 second range, however this time we see that we have slightly more extreme values on both ends, having a definite lower and higher max of our times across the tracks. We proposed this was due to longer stop times for the changing of a part of the car, or on the other end, the stop time being lower perhaps due to some value error in the dataset, again showing to us, it will be important to clean the data as well as setting a threshold for outliers to be ignored past, as to not affect the final result of the prediction.

Data Cleaning

As mentioned previously, we found that we would need to clean our data before we could even begin to write an algorithm to predict the average pit stop time. To start we began by looking into replacing the data fields that were incorrect previously, specifically trying to target the races that were causing major issues, being races at tracks with `cirquitId` 18 and 73.

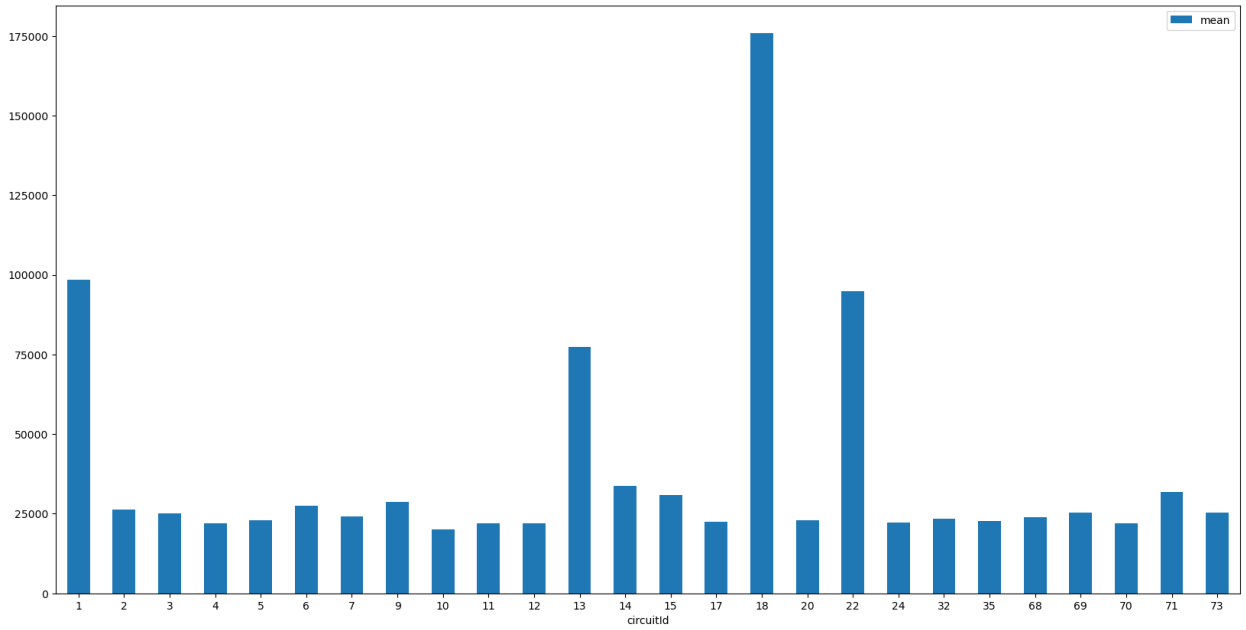
cirquitId 73

Looking at the resource from before, found here, we can see all of the pit stops that took place during that race. However, our dataset has not considered the fact that a majority of tyre changes took place under red flag conditions, meaning that these changes in tyres did not count as a regular pit stop, but rather a pit stop under non racing conditions. After some discussion, we decided to not allow these pit stops to occur, and as such, needed to modify the dataset to account for this. To do this, we decided the fairest way to do this would be by ignoring any tyre changes that took place between the beginning and end of the red flag period, using the `time` value to determine if the pit stop happened under a red flag.

```
condition1 = (mergedData['raceId'] == 967) & (mergedData['time_x'] > '14:42:00') & (mergedData['time_x'] < '15:25:00')
condition2 = (mergedData['raceId'] == 967) & (mergedData['time_x'] > '15:37:00') & (mergedData['time_x'] < '16:08:00')

removedRedFlagData = mergedData.drop(mergedData[condition1 | condition2].index)

store73 = removedRedFlagData[removedRedFlagData['raceId'] == 967]
```

```
#get the mean of store73
store73['milliseconds'].mean()

#get the mean of the mergedData, then, replace the circuitId with Id 73 with the store73 mean
groupedMean = mergedData.groupby('circuitId')['milliseconds'].agg(['mean']).reset_index()

groupedMean.loc[groupedMean['circuitId'] == 73, 'mean'] = store73['milliseconds'].mean()

groupedMean.plot.bar(x='circuitId', y='mean', rot=0, figsize=(20,10))
```

To do this we needed to handle 2 instances of red flags in that race, so we dropped 2 conditions from the data set. After this we got the mean of the newly updated dataset and stored the value. Finally we took the mean of the data and again displayed it in a bar chart again, as seen below:



As can be seen, we successfully have found the real mean of pit stops under racing conditions for `circuitId` 73.

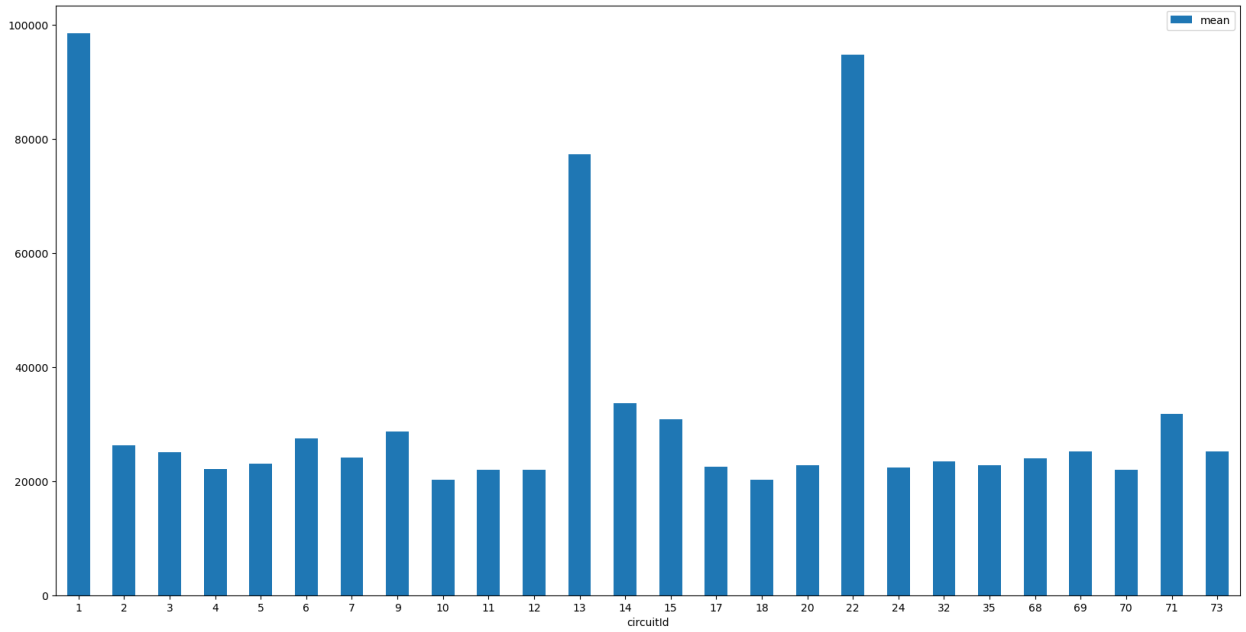
cirquitId 18

Similarly for `cirquitId` 18, we had a race again with a red flag period where tyre changes happened, and have been added to the dataset as pit stops, but again, we will not be taking red flag tyre changes as pit stops, as they do not reflect a real pit stop. To do this, we again created a condition where the red flag happened, an dropped this from the dataset. We then again took the mean of the dataset and replaced the `cirquitId` 18 value with the real value from the stored value, as can be seen in the updated code below:

```
condition1 = (mergedData['raceId'] == 967) & (mergedData['time_x'] > '14:42:00') & (mergedData['time_x'] < '15:25:00')
condition2 = (mergedData['raceId'] == 967) & (mergedData['time_x'] > '15:37:00') & (mergedData['time_x'] < '16:08:00')
conditon3 = (mergedData['raceId'] == 976) & (mergedData['time_x'] > '17:40:00') & (mergedData['time_x'] < '18:30:00')

removedRedFlagData73 = mergedData.drop(mergedData[condition1 | condition2].index)

store73 = removedRedFlagData73[removedRedFlagData73['raceId'] == 967]

#get the mean of store73
```

```
store73['milliseconds'].mean()

removedRedFlagData18 = mergedData.drop(mergedData[conditon3].index)

store18 = removedRedFlagData18[removedRedFlagData18['raceId'] == 976]

store18['milliseconds'].mean()

groupedMean = mergedData.groupby('circuitId')['milliseconds'].agg(['mean']).reset_index()

groupedMean.loc[groupedMean['circuitId'] == 73, 'mean'] = store73['milliseconds'].mean()
groupedMean.loc[groupedMean['circuitId'] == 18, 'mean'] = store18['milliseconds'].mean()

groupedMean.plot.bar(x='circuitId', y='mean', rot=0, figsize=(20,10))
```

This code produced the following result as a bar plot:

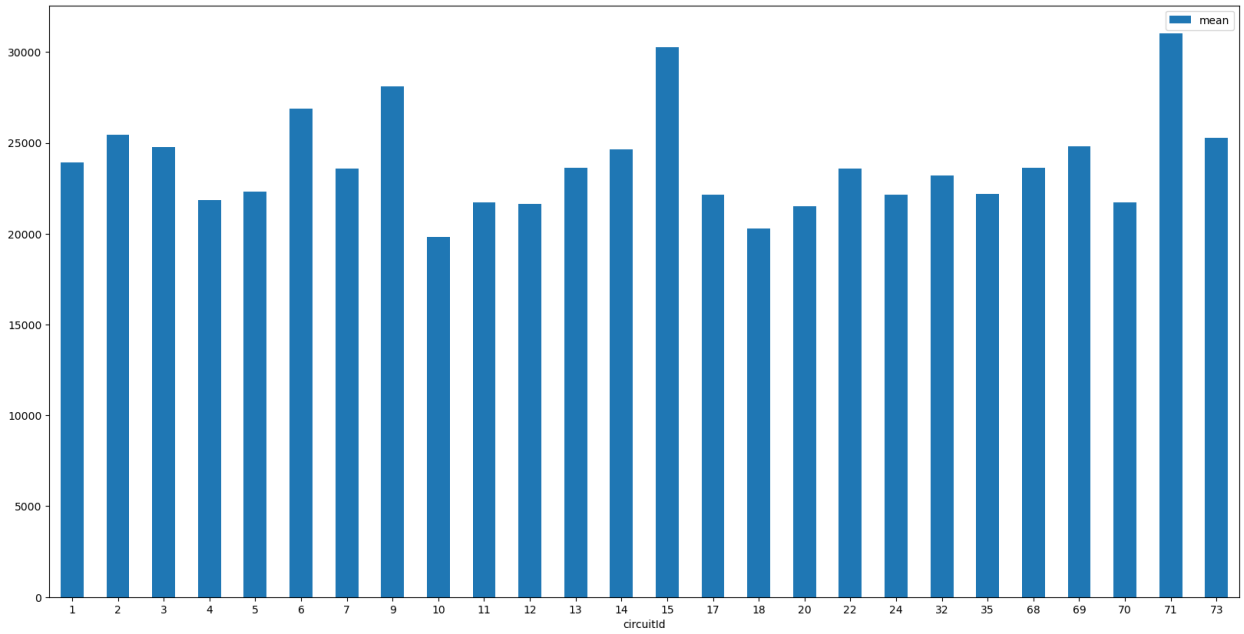


As we can now see, the only remaining outliers are those `cirquitId`'s which were resolved by trimming the data earlier in the report, so knowing this we decided to try our hand at removing outliers both on the upper extreme and the lower extreme of the data, as to have a fair way of classifying outliers. To do this we decided to use the method of removing any result that are more or less than 3 standard deviations from the mean, as these data points are likely to be significantly different from the mean, and may distort the mean, which, we can clearly see in the above graph, if we did not remove these, it could have a large impact on our final prediction. The code to do this is displayed below:

```
grouped = mergedData.groupby('circuitId')['milliseconds'].agg(['mean', 'std']).reset_index()

mergedData = mergedData.merge(grouped, on='circuitId')

# Define a function to remove outliers
def remove_outliers(row):
    mean = row['mean']
```

```
    std = row['std']
    if mean - 3 * std <= row['milliseconds'] <= mean + 3 * std:
        return True
    return False

mergedData = mergedData[mergedData.apply(remove_outliers, axis=1)]

# Drop 'mean' and 'std' columns
mergedData = mergedData.drop(['mean', 'std'], axis=1)
```

Once we ran this code, the result we got was satisfactory to us, and was very similar to what we had expected to see originally, it is seen below:



From this graph, we can see that, a majority of the average pit stop times fell within the expected 20 to 25 second range, with some being just above, but still within the 30 second window of pit stop times.

Algorithm description

When we had finally cleaned our dataset it was then ready for us to begin making a prediction based off of our data. To do this we needed to chose a learning algorithm to allow us to predict based on our data. Our idea was to predict for our current data set and then, from there, compare our prediction to the actual average of the 2019 season pit stops to see how accurate our model was after we had cleaned it and removed any outliers.

After some careful consideration, we made the deliberate choice to work with a Random Forest model for our predictive analysis. The reason behind this decision lies in the nature of our dataset, primarily composed of numerical variables. Numerical datasets often exhibit intricate, non-linear relationships between the features, and Random Forests are exceptionally well-suited for unveiling these complex patterns. Their strength lies in their ability to uncover hidden interactions and dependencies between variables.

In our initial deliberations, we also contemplated the use of a Decision Tree model. Decision Trees have their advantages, including simplicity and interpretability. However, they are known to be somewhat sensitive to outliers or noisy data, potentially leading to overfitting. This can result in individual Decision Trees making erratic predictions by giving undue weight to outliers.

To address this concern and enhance the overall robustness of our predictive model, we ultimately settled on the Random Forest algorithm. Random Forest is an ensemble technique that combines multiple Decision Trees, effectively reducing sensitivity to outliers. By aggregating predictions from a diverse set of trees, the Random Forest achieves more stable and accurate predictions while minimizing the influence of any remaining outliers. This ensemble approach not only bolsters predictive performance but also improves the model's ability to generalize to new data, making it a compelling choice for our task.

With this in mind, we researched the popular machine learning library for Python **scikit-learn**, which can be used to implement a Random Forest model. We knew we needed to do the following:

- Group all of the data on its `circuitId`, so each track would have its own predictive model
- Each `circuitId` would then need its own Random Forest models
- We would then need the average of these models

```
# Group data by 'circuitId' and initialize dictionaries
groups = finalData.groupby('circuitId')
track_random_forest_models = {}
track_averages_random_forest = {}

# Iterate through groups
for circuit_id, group_data in groups:
    X = group_data[['raceId']]
    y = group_data['milliseconds']

    # Create, train, and store the Random Forest model for the current track
    random_forest_model = RandomForestRegressor(n_estimators=100, random_state=42).fit(X, y)
    track_random_forest_models[circuit_id] = random_forest_model

    # Calculate and store the average for the current track using the Random Forest
    track_averages_random_forest[circuit_id] = random_forest_model.predict(X).mean()

track_averages = track_averages_random_forest
```

Above shows our implementation of the Random Forest algorithm using the **scikit-learn** library. As can be seen, we firstly grouped all the data by its `circuitId`, then with that done, we then split our data into a feature value (X) and a target variable (y), to be used within the model. Hence, the model will be trained to predict `milliseconds` based on `raceId`.

We then fit and trained the Random Forest model with these values, using 100 estimators. We chose 100, as from our testing, there was no noticeable difference between 100 and 1000 estimators, hence we chose 100 to allow the code to still run efficiently. Then we store the predicted mean of each `circuitId` to be use later for the analysis. Finally after this we simply converted our dictionary into a dataframe, and converted to seconds from milliseconds.

We now had all we needed in terms of our predicted data from our original dataset, we could now move onto validating our results of our prediction.

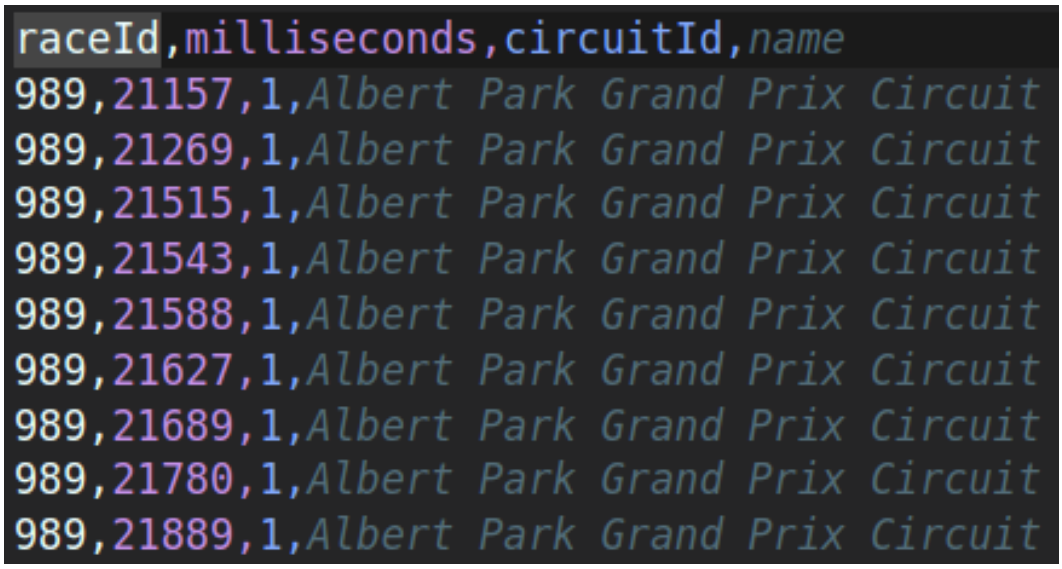
Creating our own dataset

Firstly the dataset for the 2019 season had to be obtained, however, as the full database is not yet available up to 2019, and only each race timings are available individually, we needed to then manually merge these datasets and only keep information relevant to us. While yes this would be a tedious process, it would allow for an extremely accurate comparison between what we have predicted and what actually happened. To do this looked at the RaceFans website to obtain each races information, as it is one of the most reliable sources of 3rd party Formula 1 data available currently, as the official Formula 1 website is yet to release the 2019 database in full.

The tables for the pit stop information on RaceFans looked like the following:

	Driver	Team	Pit stop time	Gap	On lap
1	Max Verstappen	Red Bull	19.062		46
2	George Russell	Williams	19.287	0.225	24
3	Max Verstappen	Red Bull	19.327	0.265	41
4	George Russell	Williams	19.345	0.283	41
5	Sebastian Vettel	Ferrari	19.586	0.524	47
6	Lewis Hamilton	Mercedes	19.610	0.548	56
7	Robert Kubica	Williams	19.626	0.564	31
8	Max Verstappen	Red Bull	19.640	0.578	25
9	Robert Kubica	Williams	19.825	0.763	57

So from here we knew which race we were looking at, so we would simple note down the `circuitId` for each stop, as well as the `raceId`, as it just continued from the end of the 2018 season, so the only changeable variable in the `.csv` file for each race would be each individual stop time. We also added the race name for use in displaying the information if needed. Below shows how our final 2019 `.csv` turned out:



Once this was finished, we could finally begin the last stage which was verifying our accuracy against the 2019 season.

Finding the real average of the 2019 season data

After reading in the finalised `2019.csv` file, we again needed to get the mean and the std, and then grouped the data by the `circuitId`, so we could apply our previously used remove-outlier function.

```
data2019 = pd.read_csv("2019.csv", encoding="ISO-8859-1")

grouped2019 = data2019.groupby('circuitId')['milliseconds'].agg(['mean', 'std', 'median']).reset_index()

data2019 = data2019.merge(grouped2019, on='circuitId')

data2019 = data2019[data2019.apply(remove_outliers, axis=1)]
```

After successfully cleaning our dataset, our next step was to derive a reliable measure of central tendency to characterize the pitstop times for the 2019 season. To accomplish this, we chose to calculate both the mean and the median of the pitstop times. These statistical measures provide us with different perspectives on the central tendency of the data.

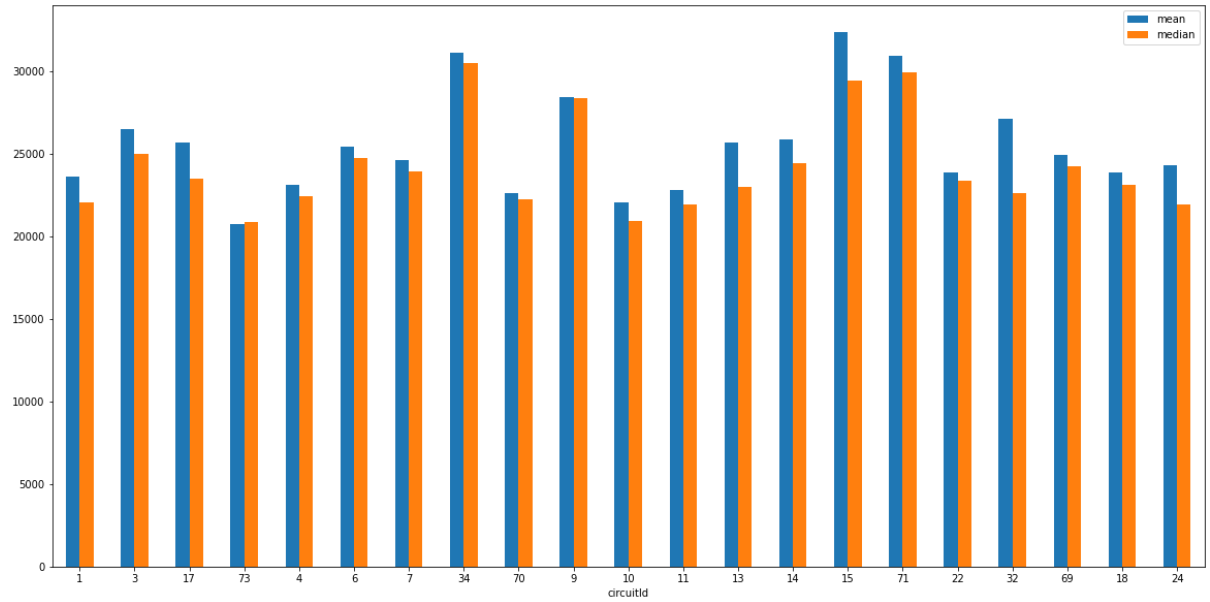
The mean, often referred to as the arithmetic average, serves as a common representation of the dataset's overall pitstop time. It is obtained by summing all the pitstop times and dividing by the total count. This measure is sensitive to outliers, which can sometimes distort the average if extreme values are present in the data.

In contrast, the median is another valuable measure of central tendency. It represents the middle value of the dataset when all values are arranged in ascending order. Unlike the mean, the median is less affected by outliers, making it a robust statistic.

To arrive at a comprehensive average for the 2019 season, we didn't simply rely on one of these measures. Instead, we took the average of both the mean and median results. This approach provides a balanced perspective, combining the sensitivity of the mean with the robustness of the median. The resultant average can be considered as a representative measure that appropriately captures the essence of the pitstop times throughout the 2019 season in Formula 1.

```
mean2019 = data2019['mean'].div(1000).round(2)
median2019 = data2019['median'].div(1000).round(2)

#real average is the average of the mean and median
data2019['real-average'] = (mean2019 + median2019) / 2

data2019['2019-average'] = data2019['real-average']
```

The results of the above code is displayed below in the bar graph:

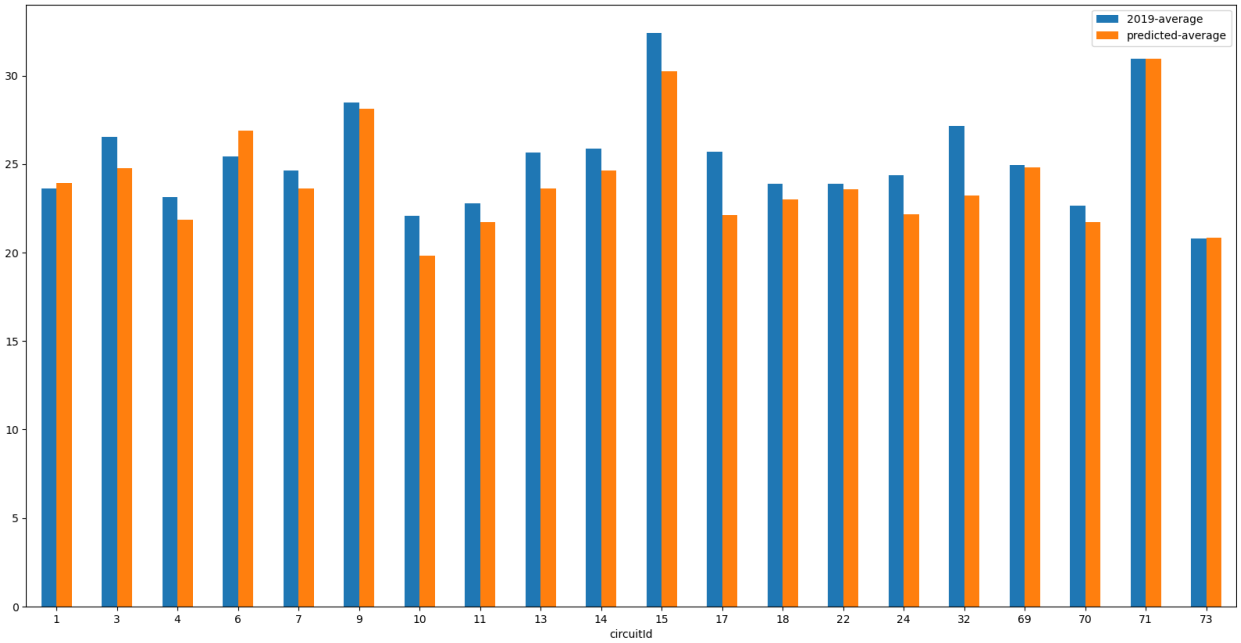


Comparing this graph to the one we found from cleaning our data, shows us that again, `circuitId` 15 and 71 are some of the highest in this graph too. We found this extremely encouraging as it meant both datasets were tending towards the same overall results, meaning we were ready to validate the accuracy of our prediction we had made, and see if it was again similar to these 2 graphs we had made.

Results and Analysis

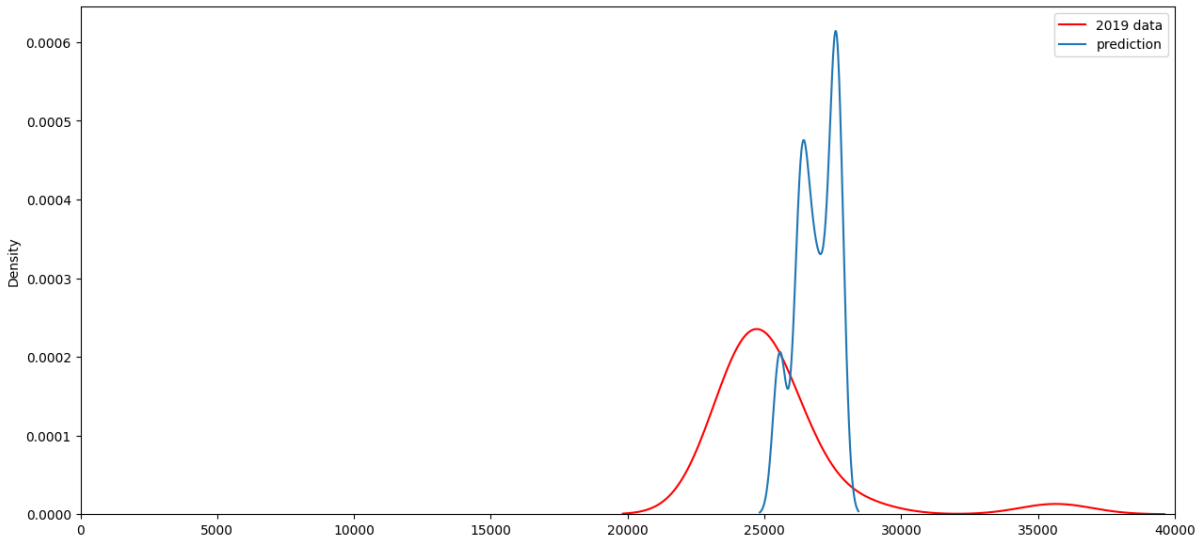
Our Results

We now finally had all the information we needed from both our prediction as well as the real average of the 2019 season. Lets first look at how our prediction matched the real average of the 2019 data, we can see in the below bar graph each track common to both side by side:



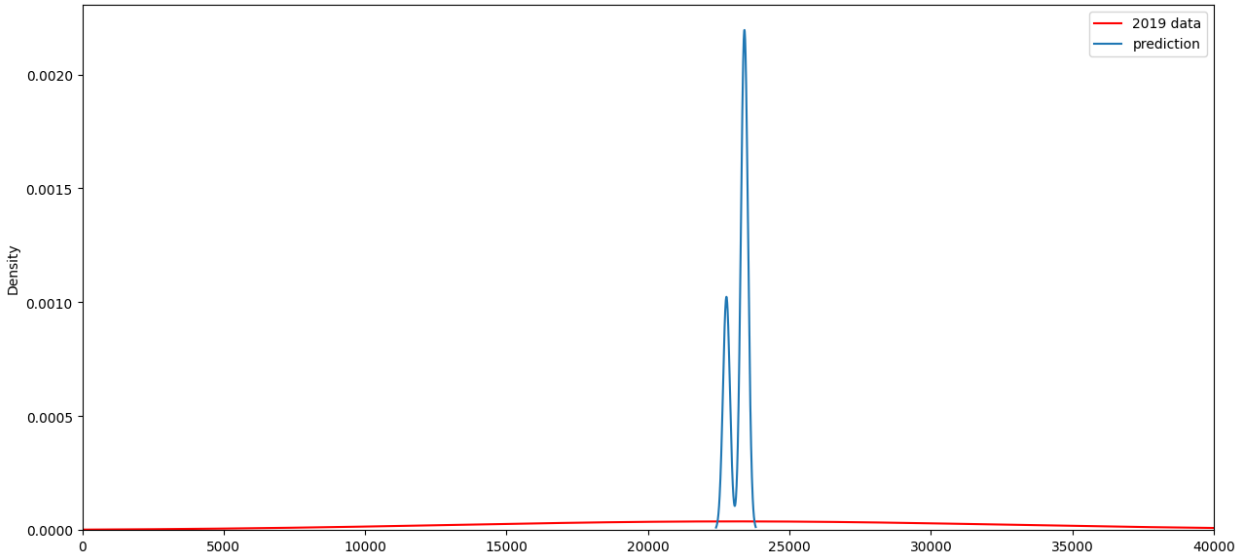
When analysing the data in a graphical display such as the above, we immediately noticed the values appear to be close together. However, on further analysis the majority of the predicted values generated by our algorithm appear to be below the actual average pit stop time of the 2019 season. On further inspection we observed the following tracks to stand out for various different reasons. These circuits included `circuitId` 6 which seems to have a noticeably longer predicted pit stop than the actual pit stop time in 2019. `CircuitId` 32 was also included in these tracks and this seemed to have the opposite observation to `circuitId` 6 where the predicted pit stop time seemed to be significantly shorter than the actual pit stop time in 2019. Another striking result was the accuracy of `circuitId` 71 where both the predicted and average pit stop times seemed to be the exact same.

CircuitId 6



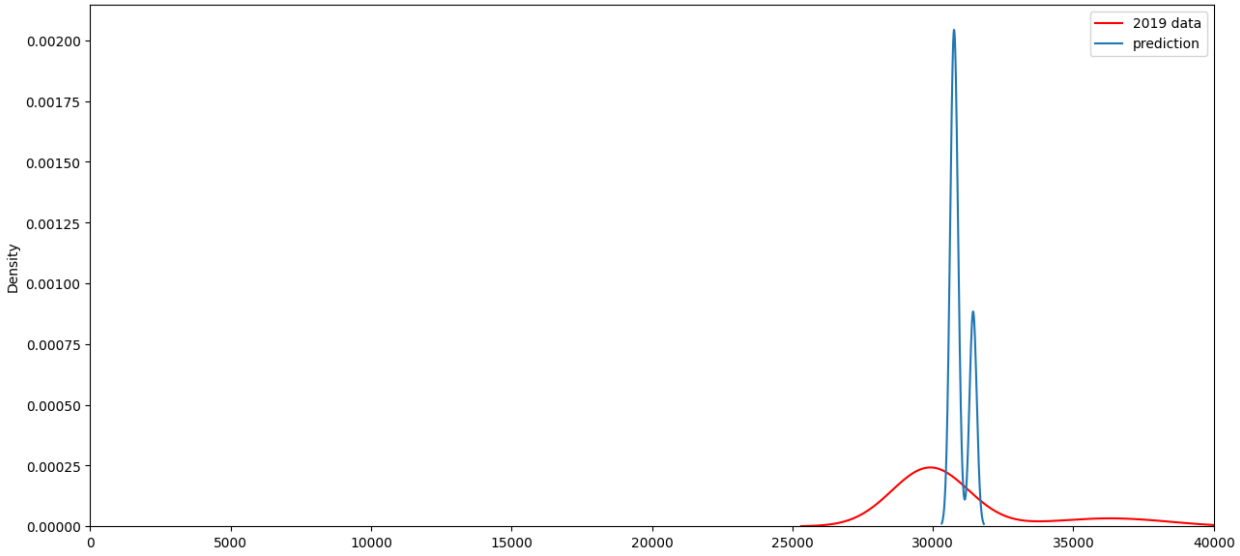
Looking at the above KDE graph for `circuitId` 6, we see how the density of the values used for this circuit to calculate both the real average of the 2019 season, and the predicted average of our algorithm, we can see how we ended with a prediction that was unlike the rest of the predictions vs their real average counterparts. We see that the centre of distribution for the predicted data would approximately fall around the 27 second mark. And so, despite the data for the 2019 data being pulled by the slight density pull from the bulge in the line near the 35 second mark, the rough centre of density for the 2019 data only falling around the 26 second mark, we ended with the outlier of a result showing the real average being shorter than the predicted average.

CircuitId 32



When we analyse the KDE graph for the data for `circuitId` 32, again used for the prediction as well as the real average of the 2019 season, we notice a huge change in how the data is distributed. The data for the 2019 season seems to be heavily spread between the 15 and 35 second mark, with little clustering of the data points at all, with just a small rise between the 20 to 25 second range. Conversely, we see how the clustering of the significantly larger set of data for the 2011-18 seasons begins to cluster highly between the 23-24 second mark. Due to this, it is very clear that the average for the predictions data is roughly 23-24 seconds. This however, is not so clear for the 2019 data, thus leading to the average being found to be higher than 25 seconds when the true average is found, hence we get the result with the prediction being noticeably shorter than the true average.

CircuitId 71



Finally looking at the KDE for `circuitId` 71, we can see for the predicted data, the centre of the distribution is clearly around the 31 second mark, seen by the two large spikes around this point on the x-axis. However, unlike before, when we inspect the distribution for the 2019 data, we quickly realise, the centre of distribution for this set of data as well was visually identical to the predicted data average, with both being around the 31 second mark. We decided to pull the exact numbers for both, and found that the predicted average was 30.95 seconds, while the real average was only 0.02 seconds higher at 30.97 seconds. This is then why we could not see any visual difference in the data, as the values are so incredibly close to each other.

Why predicted averages are lower?

As we have discussed above, the overall average of the predictions is lower than the 2019 data averages, which did not come as a surprise to use for several reasons. The main being, the 2019 dataset is significantly smaller than the dataset from 2011-2018 used for the predictions, as it is only a single season of pit stops, as opposed to 8 seasons worth of pit stops.

Due to this, when we are examining the data to remove outliers, and we have a race that only has, for example 20 pitstops, and 2 of these are extremely slow, we cannot say, lets drop both of the slow pitstops, as this would be dropping 10% of the data, and, for a fair result, we are using the same percentage of outlier removal for every `circuitId` , and such, if we were to remove 10% every time we could end up dropping a large number of useful data points, especially if a race has no outlier pit stops, as is often the case.

Now that we know this, it is clear why on average the 2019 data produced higher overall averages compared to the predicted data, due to less outliers being dropped in the 2019 dataset.

Our Result Accuracy

In this section we will be examining the accuracy of our prediction. We will look at the accuracy of our prediction, how we determined the accuracy percentage and whether this accuracy percentage is good or bad.

How we determined the accuracy

To do this, we used the following code:

```
merged['accuracy'] = (merged['2019-average'] / merged['predicted-average']).round(2)
merged['accuracy'] = merged['accuracy'].apply(lambda x: 1 - abs(1 - x))
totalAccuracy = merged['accuracy'].mean() * 100
(str(totalAccuracy.round(3)) + '%')
```

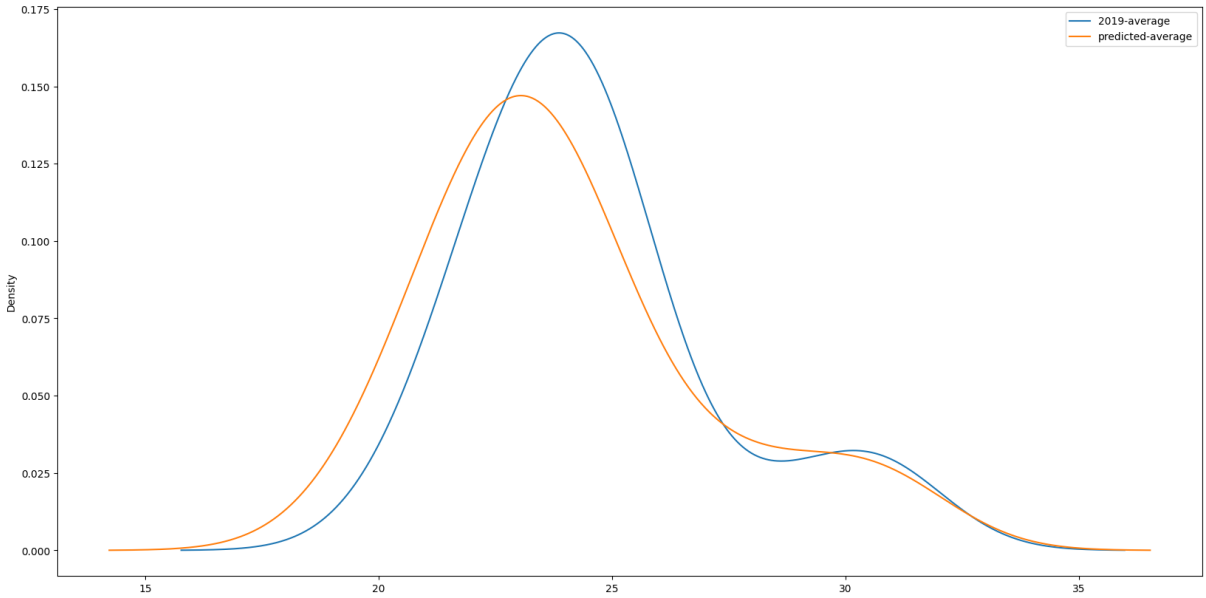
Firstly we measured the accuracy of by dividing the 2019-average by the predicted-average, and then rounding it, just to keep it readable. After this, we then used a lambda function that takes a single argument 'x'. For each element 'x' in the 'accuracy' column, it computes the absolute difference between 1 and 'x', subtracts that difference from 1, and returns the result. This will effectively find the how far each x value is from 1. From here, we find the mean of all the values to be represented as a single percentage value.

How accurate was our prediction

Below shows the accuracy of each track, the max is 100%, with the minimum value only being as low as 89. After this we found the total average of the entire season.

```
0      95.0
1      96.0
2      89.0
3     100.0
4      96.0
5      93.0
6      97.0
7      97.0
8      99.0
9      91.0
10     97.0
11     97.0
12     98.0
13     98.0
14     98.0
15    100.0
16     93.0
17     99.0
18     98.0
19     96.0
```

Our final result returned a predicted accuracy for the entire season of 96.35%, this can be shown again using a KDE, seen below:



As seen, the distribution of the data is extremely similar between both the 2019-average and the predicted-average, and such it is clear that our predicted final percentage is indeed true, as our 2 graphed lines are almost identical. We are confident in saying that our prediction of the 2019 seasons pit stops is above 90% accurate.

Can our prediction be improved?

Although there are definitely ways to improve our final prediction, one such way would be by having a larger set of data, but, as is obvious, this is not a simple as it sounds, as it is impossible to predict the amount of pitstops in a season, and, no matter which season we chose, no one season will be able to ever be fully accurate compared to several seasons worth of pitstops.