Predecting Qualifying Times in F1

Husam Alkhateeb
0193431
Computer Engineering Department
University of Jordan
Amman, Jordan
Hsa0193431@ju.edu.jo

***Abstract*—This electronic document is a "live" template and already defines the components of your paper [title, text, heads, etc.] in its style sheet. **CRITICAL: Do Not Use Symbols, Special Characters, Footnotes, or Math in Paper Title or Abstract.* (*Abstract*)**

***Keywords—component, formatting, style, styling, insert** (key words)*

I. Introduction

Formula 1 is the highest class of international racing in the world, each weekend of an F1 race consists of 3 practice sessions, 3 qualifying sessions and a race. The first 2 practice session are used so drivers can get accumulated to the track and teams can collect data for the strategy for the race, the third qualifying session is used for drivers to push the limits so they can prepare for qualifying. Qualifying sessions are to set the grid for the race, whoever has the fastest time will start from the first grid slot, second fastest time will start from the second grid slot and so on… the first Qualifying session eliminate the bottom 5 drivers with the worst times then the 2nd qualifying session they also eliminate the bottom 5, so the top 10 can have a shot at pole position in Q3, the data from the third Free Practice session and the third qualifying session will be used to predict the times that will be achieved during Q3.

II. Data Collection

A. Web-scraping F1 website for FP3 times:

Data-sets that contained practice session times was searched for but there weren't any so it was decided web-scraping and dataframe manipulation to create a dataset of FP3 times so it can be trained on these times to predict Q3 times.

A list of URL links had to be generated so it can take show the webpage of the FP3 session of each race weekend,a lot of problems were faced with this method because if a link doesnt open an FP3 session page, it will be redirected to another page which can generate a lot of errors, so another list of URL links was generated that contain only working links that open actual FP3 webpages, then Beautifulsoup and Selenium were used to collect the data from each page.

```
for i in range(0,len(array_from_file)):
    url = array_from_file[i]
    print(url)
driver2 = webdriver.Chrome("/usr/lib/chromium-browser/chromedriver")
    driver2.get(url)
    sleep(randint(10,20))

soup = BeautifulSoup(driver2.page_source, 'html.parser')
    my_table2 = soup.find('table', class_='resultsarchive-table')

    for j in my_table2.find_all('tr')[1:]:
        data.append(j.text.strip())
```

Figure 1
the code used to extract the data from the F1 website

This code above is used to extract data from every cell in the the table that has the class "resultsarchive-table" (we got the table name by using inspect element on the F1 website), the code ran for 1 hour to run through all the links and collect all the data and import it into a numpy array:

which was converted later to a dataframe using pd.Dataframe function in pandas after np.reshape() was used on the numpy array the number of columns was obtained need.

And after cleaning up the data for a bit this is the data-set that was obtained:

https://drive.google.com/file/d/1AGzw5BVXJIA-R4dLnu43Hn4MHQYuWTrA/view?usp=sharing

```
print(data)
```

['1\n44\n\nLewis\nHamilton\nHAM\n\nMercedes\n1:31.699\n1:42.890\n1:44.231\n22', '2\n3\n\nDaniel\nRicciardo\nRIC\n\nRed Bull Racing Renault\n1:30.775\n1:42.295\n1:44.548\n20',
'4\n20\n\nKevin\nMagnussen\nMAG\n\nMcLaren Mercedes\n1:30.949\n1:43.247\n1:45.745\n19', '5\n14\n\nFernando\nAlonso\nALO\n\nFerrari\n1:31.388\n1:42.885\n1:45.819\n21', '6\n25\
'7\n27\n\nNico\nHulkenberg\nHUL\n\nForce India Mercedes\n1:33.893\n1:43.658\n1:46.030\n20', '8\n26\n\nDaniil\nKvyat\nKVY\n\nSTR Renault\n1:33.777\n1:44.331\n1:47.368\n20', '9
'10\n77\n\nValtteri\nBottas\nBOT\n\nWilliams Mercedes\n1:31.601\n1:43.852\n1:48.147\n19', '11\n22\n\nJenson\nButton\nBUT\n\nMcLaren Mercedes\n1:31.396\n1:44.437\n\n13', '12\n

	A	B	C	D	E	F	G	H	I
1	POS	DRIVERID	FIRSTNAME	LASTNAME	3LETTERS	CAR	TIME	GAP	LAPS
2	1	6	Nico	Rosberg	ROS	Mercedes	1:29.375		15
3	2	22	Jenson	Button	BUT	McLaren Mercedes	1:30.766	+1.391s	20
4	3	14	Fernando	Alonso	ALO	Ferrari	1:30.876	+1.501s	11
5	4	44	Lewis	Hamilton	HAM	Mercedes	1:30.919	+1.544s	13
6	5	3	Daniel	Ricciardo	RIC	Red Bull Racing Renault	1:30.970	+1.595s	13
7	6	27	Nico	Hulkenberg	HUL	Force India Mercedes	1:30.978	+1.603s	16
8	7	7	Kimi	Räikkönen	RAI	Ferrari	1:31.156	+1.781s	12
9	8	20	Kevin	Magnussen	MAG	McLaren Mercedes	1:31.251	+1.876s	22
10	9	11	Sergio	Perez	PER	Force India Mercedes	1:31.665	+2.290s	17
11	10	19	Felipe	Massa	MAS	Williams Mercedes	1:31.723	+2.348s	20

Figure 2
This is a sample of the FP3 dataset that was obtained using Web Scraping techniques

B. Web-scraping F1 website for Q3 times:

Q3 times were available through the data-set that is available on kaggle (Formula 1 World Championship (1950-2022)) but the data-set was too messy and couldn't be used efficiently without some heavy modification so it was decided to create another data-set by web-scarping the F1 website for Q3 times. The same method was applied on the Q3 URL links as the FP3 links which created the following dataset:

https://drive.google.com/file/d/1pHsqP4mOWe7ezbS6wWL4RD_RRFE9iozK/view?usp=sharing

POS	DRIVERID	k0	FIRSTNAME	LASTNAME	3LETTERS	k1	CAR	Q1	Q2	Q3	LAPS
1	44	k	Lewis	Hamilton	HAM	k	Mercedes	1:31.699	1:42.890	1:44.231	22
2	3	k	Daniel	Ricciardo	RIC	k	Red Bull Racing Renault	1:30.775	1:42.295	1:44.548	20
3	6	k	Nico	Rosberg	ROS	k	Mercedes	1:32.564	1:42.264	1:44.595	21
4	20	k	Kevin	Magnussen	MAG	k	McLaren Mercedes	1:30.949	1:43.247	1:45.745	19
5	14	k	Fernando	Alonso	ALO	k	Ferrari	1:31.388	1:42.805	1:45.819	21
6	25	k	Jean-Eric	Vergne	VER	k	STR Renault	1:33.488	1:43.849	1:45.864	21
7	27	k	Nico	Hulkenberg	HUL	k	Force India Mercedes	1:33.893	1:43.658	1:46.030	20
8	26	k	Daniil	Kvyat	KVY	k	STR Renault	1:33.777	1:44.331	1:47.368	20
9	19	k	Felipe	Massa	MAS	k	Williams Mercedes	1:31.228	1:44.242	1:48.079	21
10	77	k	Valtteri	Bottas	BOT	k	Williams Mercedes	1:31.601	1:43.852	1:48.147	19

Figure 3 This is a sample of the Qualifying Dataset that was obtained using Web Scraping techniques

III. Data Preparation:

Now that the dataset got created the data needed to be prepared then merged into two data-sets together so it can be easily trained. Only the top 10 positions were taken as there was a lot on null values in the bottom 20 drivers in each session as sometime there would be mechanical trouble so some cars would stay in the garage and not set any lap time whether it was FP3 or Q3. For the practice data-set the POS, DriverId, Firstname, Lastname, 3Letters, car, and Gap can be removed as all of them are irrelevant to the times set in practice.as for the qualifying data-set the POS, DriverId, k0, Firstname, Lastname, 3Letters, k1, car, Q1 and Q2 columns can be removed as they are irrelevant to the actual qualifying times. Then the Practice and Qualifying times had to converted to seconds and thats by multiplying them by 86400. then the two datasets were merged and aligned each practice session with its qualifying session to get the following data-set:

https://drive.google.com/file/d/1IHiQ8VVZQQYRsCELx0LpJniE_uPsPXPF/view?usp=sharing

POS	LAPS	WETPRAC	WETQUALI	PRAC3SECONDS	Q3SECONDS
1	22	0	2	89.375	104.231
2	20	0	2	90.766	104.548
3	21	0	2	90.876	104.595
4	19	0	2	90.919	105.745
5	21	0	2	90.97	105.819
6	21	0	2	90.978	105.864
7	20	0	2	91.156	106.03
8	20	0	2	91.251	107.368
9	21	0	2	91.665	108.079

Figure 4 This is a sample of the merged dataset

It can be seen that there are 2 extra columns (WETPRAC, WETQUALI), those columns were added because they affect the times drastically, the wet practice columns indicates if the FP3 session was wet which can explain the high times in some practice sessions compared to Q3 times, the times where there was +3 seconds difference in between the FP3 time and Q3 time had to be gone through and checked the weather during those sessions through wikipedia, so the sessions were classified into 5 classes:

0	Cars used dry tires and the track was dry
0.5	Cars used dry tires, but the track had less grip due to light rain
1	Cars used intermediate tires with the rain
2	Cars used intermediate with < heavy rain
3	Cars used full wet tires due to heavy rain

and this will help the regressor to identify the if the Q3 time will be a lot less or a lot more than the FP3 time.

IV. Visualizing the Data:

There is not a lot of things that can be gained from visualizing the data other than the relation between Q3SECONDS and PRAC3SECONDS, as can be seen from the graph the relation between the 2 is almost linear but there are a lot of outliers due to having either wet practice sessions or wet qualifying sessions which will affect the predictor heavily.

```
<AxesSubplot:xlabel='Q3SECONDS', ylabel='PRAC3SECONDS'>
```

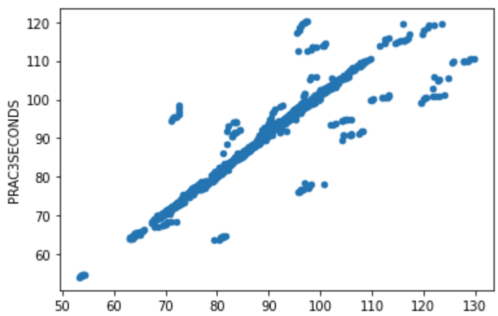


Figure 5 A plot describing the relation between Q3SECONDS and PRAC3SECONDS

As there wasn't any categorical data SimpleImpuer was used on the numerical features, and scale the features using the StandardScaler, then these components will be joined into a Pipeline to preprocess the attributes.

V. Training The Model

A. Linear regression:

using the LinearRegression() on the training set RMSE was found to be 2.27 and 2.27 using the cross_val_score on tree_reg with cv=10.

```
lin_mae = mean_squared_error(trainin_labels, trainin_predictions)
lin_train_rmse = np.sqrt(lin_mae)
print(lin_train_rmse)
✓ 0.4s

2.270553927736319
```

On the test set the mean squared error was found to be 2.24.

```
final_mse = mean_squared_error(y_test, final_predictions)
lin_rmse = np.sqrt(final_mse)
lin_rmse

✓ 0.1s

2.2444874850419243
```

as it was said before that the data has a lot of outliers because sometimes there were wet practice or qualifying sessions, so the linear regression model will be affected the most by it.

B. *Decision Tree Regressor*

using the DecisionTreeRegressor() on the training set RMSE was found to be 0.042 and 1.60 using the cross_val_score on tree_reg with cv=10.

```
tree_train_rmse = np.sqrt(tree_mse)
tree_train_rmse

✓ 0.4s

0.04193530694478645
```

On the test set the RMSE was found to be 1.05.

```
final_mse = mean_squared_error(y_test, final_predictions)
tree_rmse = np.sqrt(final_mse)
tree_rmse

✓ 0.3s

1.0502935610891886
```

looking at the rmse of both the training set and the test set it can be concluded that there was an overfit because the training set has a very small rmse and the test set rmse is more than 20 times worse.

C. *Random Forest Regressor*

using the RandomForestRegressor() on the training set the RMSE was found to be 0.561 and scored 1.44 using the cross_val_score on forest_reg with cv=10.

```
forest_mse = mean_squared_error(trainin_labels, trainin_predictions)
forest_train_rmse = np.sqrt(forest_mse)
forest_train_rmse

✓ 0.3s

0.5609167779858018
```

On the test set the RMSE was found to be 0.914.

```
final_mse = mean_squared_error(y_test, final_predictions)
forest_rmse = np.sqrt(final_mse)
forest_rmse

✓ 0.2s

0.9141122245539044
```

D. *Grid Search*

When applying GridSearchCV() on the RandomForestRegressor() we get the best combination (4 max_features and 30 n_estimators)

```
grid_search.best_params_

✓ 0.2s

{'max_features': 4, 'n_estimators': 30}
```

with an rmse of 1.52 on the test set it has given a 1.0.

```
1.8893741648717435 {'max_features': 2, 'n_estimators': 3}
1.5808850184342267 {'max_features': 2, 'n_estimators': 10}
1.5437800386468397 {'max_features': 2, 'n_estimators': 30}
1.5384924442471877 {'max_features': 4, 'n_estimators': 3}
1.543270650922739 {'max_features': 4, 'n_estimators': 10}
1.5240752628650178 {'max_features': 4, 'n_estimators': 30}
1.7935074473633 {'bootstrap': False, 'max_features': 2, 'n_estimators': 3}
1.598576337833882 {'bootstrap': False, 'max_features': 2, 'n_estimators': 10}
1.7123339892935068 {'bootstrap': False, 'max_features': 3, 'n_estimators': 3}
1.6554628099886182 {'bootstrap': False, 'max_features': 3, 'n_estimators': 10}
1.7310738673243926 {'bootstrap': False, 'max_features': 4, 'n_estimators': 3}
1.6871418033405912 {'bootstrap': False, 'max_features': 4, 'n_estimators': 10}
```

E. *Random Forest Regressor*

with the RandomForestRegressor() we got a 1.43 rmse which was the best out of the rest, and it had 3 max_features and 22 n_estimators,

```
1.4865753300311644 {'max_features': 3, 'n_estimators': 52}
1.8375520221160215 {'max_features': 1, 'n_estimators': 15}
1.4836438533224705 {'max_features': 3, 'n_estimators': 72}
1.790561633062259 {'max_features': 1, 'n_estimators': 21}
1.4898879518386565 {'max_features': 3, 'n_estimators': 83}
1.4840124207971865 {'max_features': 3, 'n_estimators': 75}
1.49129472625436 {'max_features': 3, 'n_estimators': 88}
1.7560171335284236 {'max_features': 1, 'n_estimators': 24}
1.4353432295959143 {'max_features': 3, 'n_estimators': 22}
2.3038365717516793 {'max_features': 1, 'n_estimators': 2}
```

as for the test set it had an rmse of 1.05.

F. *Keras Training*

when the data was given to the keras training model with 100 layers and 50 epchos the rmse was 1.83 on the test set.

```
Epoch 50/50
28/28 [==============================] - 0s 1ms/step - loss: 3.5721 - val_loss: 3.2821
13/13 [==============================] - 0s 494us/step - loss: 3096103.5000
13/13 [==============================] - 0s 493us/step

1.8368717454928687
```

G. *SVR*

And finally when applied the SVR() on the training set the RMSE was found to be 2.46 and 2.37 using the cross_val_score on svm_reg with cv=10.

VI. Analyzing The Output

looking at the graphs it can be clearly seen that the linear regression model is the worst one as the rmse was highest, also the decision tree model is clearly overfitted as it almost memorised all the data and it performed much worse on the test set, and finally the random forest tree gave the best results as the rmse for the training set and the test set is the closest one and the lowest out of all the other models.

A. Training

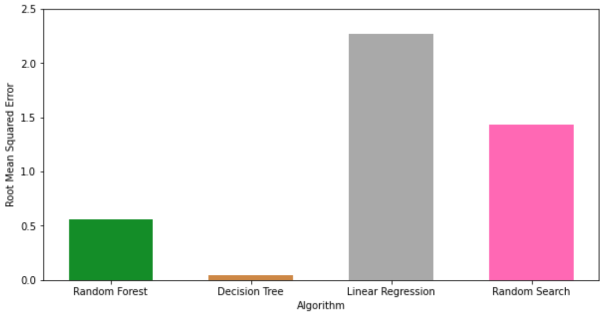


Figure 6 RMSE for training set

B. Testing

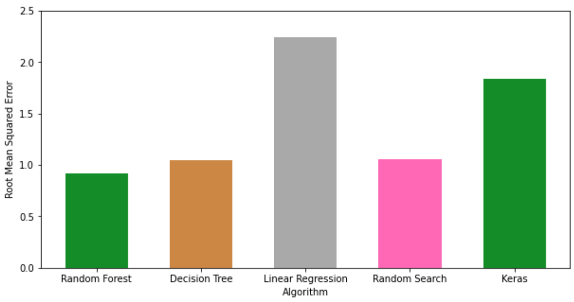


Figure 7 RMSE for testing set

VII. Removing Attributes

trying to make it better the rows were there were wet sessions in qualifying or in FP3 can be removed. The wet practice and qualifying sessions can be removed using the following code:

this code checks every value of WETPRAC and QUALIPRAC and removes the row that contains anything other than 0 which means it removes the wet sessions and their times from the dataset. Then the algorithms can be applied on this new dataset.

```
df = pd.read_csv("TRAIN12.csv")



df.drop(df[(df.WETPRAC > 0) | (df.WETQUALI >
0)].index, inplace=True)

df.to_csv('TRAINwoWET.csv', index=False)
```

And here is the corrolation matrix:

```
corr_matrix = train_set.corr()
corr_matrix["Q3SECONDS"].sort_values(ascending=False)
✓ 0.1s

Q3SECONDS       1.000000
PRAC3SECONDS    0.997657
LAPS           -0.327982
```

Figure 4 Correlation Matrix

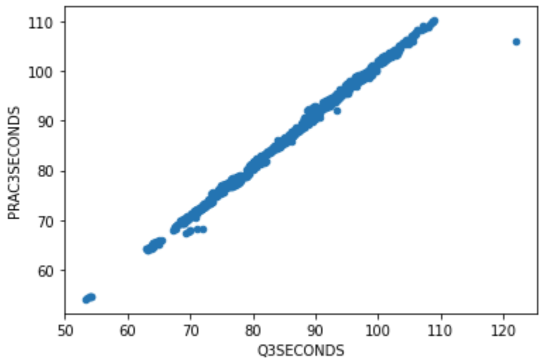
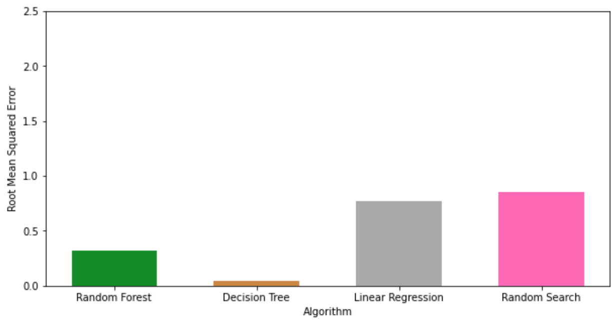


Figure 8 Plot showing the relation between Qualifying times and practice times excluding wet sessions

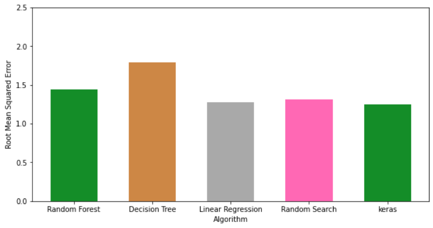
It makes sense that the laps in practice give us less times for qualifying because the drivers are more accumulated to the track with more laps, also its more correlated than when we used the wet sessions because with wet sessions the laps dont really matter that much as qualifying weather will most likely be different so the drivers will have to relearn the track.

The same piplining, scalering and imputing will be applied to the dataset now looking at the graphs that represent the rmse for the training set and test set for each model its clear that the linear regression model is much better in predicting qualifying times when there is no wet sessions, although surprisngly the rest of the algortihms all performed considerbly worse when the dry dataset was applied to them.

A. Training



B. Test



VIII. Conclusion

looking at the graphs of the rmse for each one of algorithms used on the datasets it can be concluded that almost every time the model gets overfitted (the test rmse is much higehr than the train rmse), the accuracy of the model can be increased with a more accurate wet session classification, more features for the algorithms to train on such as: the type of the dry tire used for qualifying and practice (C1,C2,C3,C4,C5), the temperature of the track for qualifying and practice as temperature plays a huge part in the tires and the engine efficiency and maybe adding the previous year's qualifying time in the training set for each row. Overall this model can be improved immensely if more data was more easily available as scraping the data from the f1 website was very hard and challenging.

References

[1] F1 Website archive
https://www.formula1.com/

APPENDIX

In [175]:

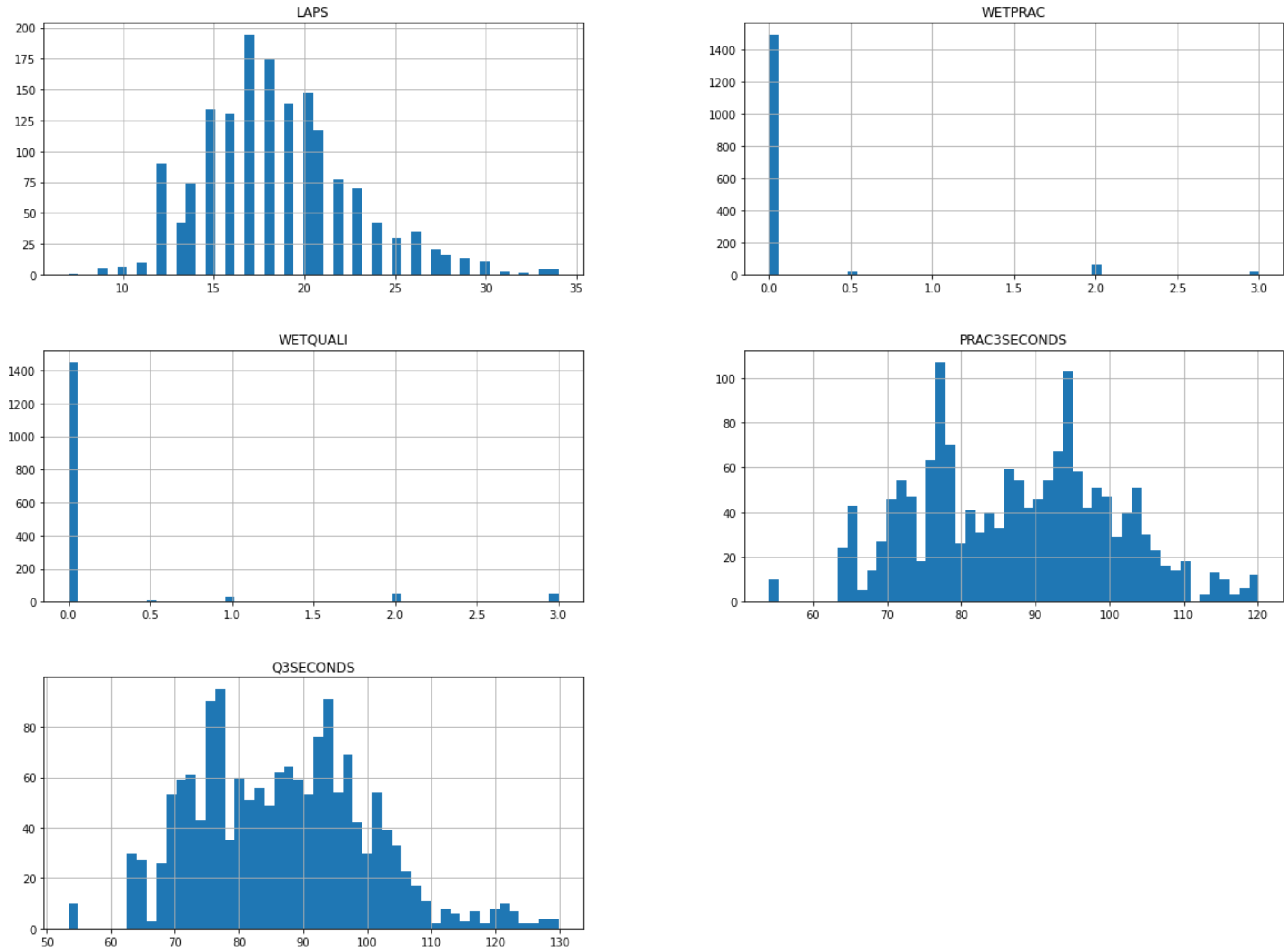
```
import numpy as np
import os
import matplotlib as mpl
import matplotlib.pyplot as plt
import pandas as pd
training = pd.read_csv('TRAIN1.csv')

print(training.describe())
```

```
              LAPS     WETPRAC    WETQUALI  PRAC3SECONDS   Q3SECONDS
count  1590.000000  1590.000000  1590.000000  1590.000000  1590.000000
mean     18.579874     0.119497     0.179874    87.533692    86.798224
std       4.161872     0.503165     0.632682    12.936957    13.359347
min       7.000000     0.000000     0.000000    54.064000    53.377000
25%      16.000000     0.000000     0.000000    77.122250    76.236250
50%      18.000000     0.000000     0.000000    88.062500    86.621500
75%      21.000000     0.000000     0.000000    96.621500    95.730750
max      34.000000     3.000000     3.000000   120.158000   129.776000
```

In [176]:

```
training.hist(bins=50, figsize=(20,15))
plt.show()
```



In [177]:

```
np.random.seed(42)
def split_train_test(data, test_ratio):
    shuffled_indices = np.random.permutation(len(data))
    test_set_size = int(len(data) * test_ratio)
    test_indices = shuffled_indices[:test_set_size]
```

```
    train_indices = shuffled_indices[test_set_size:]
    return data.iloc[train_indices], data.iloc[test_indices]
```

In [178]:

```
from sklearn.model_selection import train_test_split
train_set, test_set = train_test_split(training, test_size=0.2, random_state=42)

test_set
```

Out[178]:

	LAPS	WETPRAC	WETQUALI	PRAC3SECONDS	Q3SECONDS
1079	19	0.0	0.0	65.391	64.200
405	22	0.0	0.0	97.985	97.125
1493	24	0.0	1.0	94.154	104.050
239	16	0.0	0.0	87.809	86.770
610	20	0.0	0.0	94.001	93.194
...	...	...	...	...	...
1023	15	0.0	0.0	103.064	101.069
700	29	3.0	3.0	100.660	95.554
486	12	0.0	0.0	108.742	107.543
672	24	0.0	0.0	88.137	87.356
1303	18	0.0	0.0	86.896	86.035

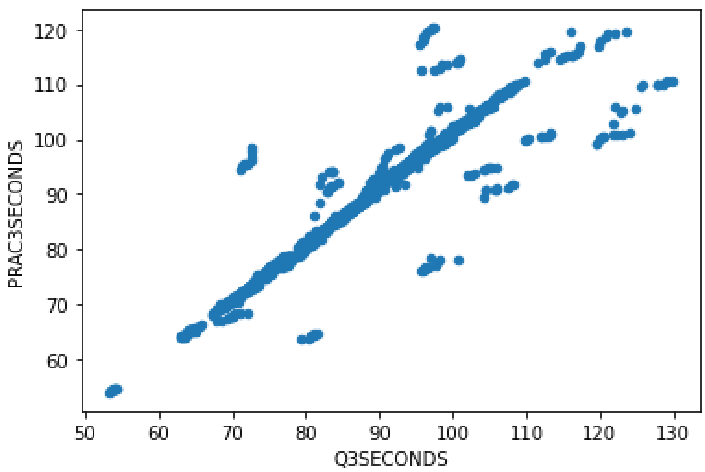
318 rows × 5 columns

In [179]:

```
relation = train_set.copy()
relation.plot(kind="scatter", x="Q3SECONDS", y="PRAC3SECONDS")
```

Out[179]:

<AxesSubplot:xlabel='Q3SECONDS', ylabel='PRAC3SECONDS'>



In [180]:

```
corr_matrix = train_set.corr()
corr_matrix["Q3SECONDS"].sort_values(ascending=False)
```

Out[180]:

```
Q3SECONDS       1.000000
PRAC3SECONDS    0.923711
WETQUALI        0.356625
WETPRAC         0.177442
LAPS           -0.142827
Name: Q3SECONDS, dtype: float64
```

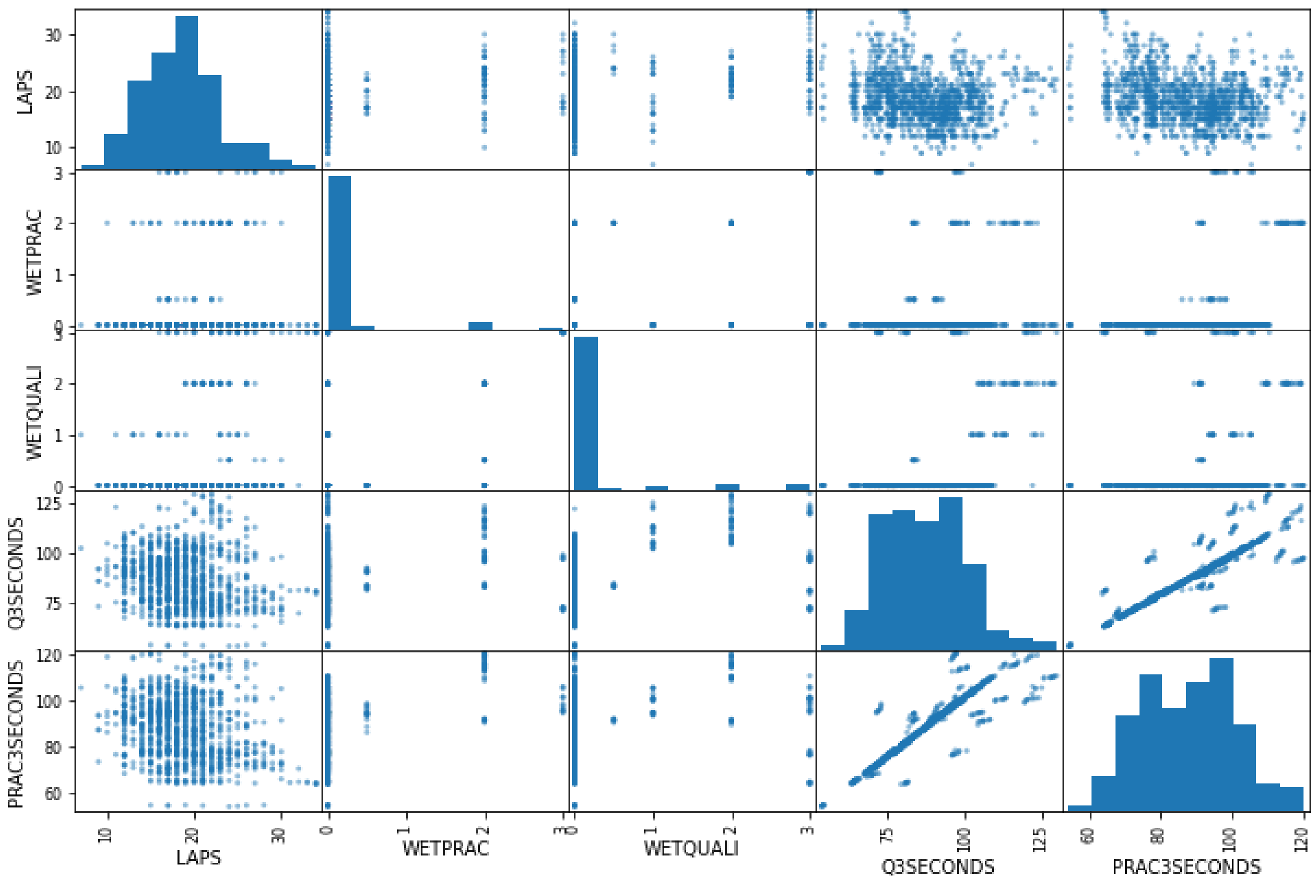
In [181]:

```
from pandas.plotting import scatter_matrix

attributes = ["LAPS", "WETPRAC",
              "WETQUALI","Q3SECONDS","PRAC3SECONDS"]
scatter_matrix(train_set[attributes], figsize=(12, 8))
```

Out[181]:

```
array([[<AxesSubplot:xlabel='LAPS', ylabel='LAPS'>,
        <AxesSubplot:xlabel='WETPRAC', ylabel='LAPS'>,
        <AxesSubplot:xlabel='WETQUALI', ylabel='LAPS'>,
        <AxesSubplot:xlabel='Q3SECONDS', ylabel='LAPS'>,
        <AxesSubplot:xlabel='PRAC3SECONDS', ylabel='LAPS'>],
       [<AxesSubplot:xlabel='LAPS', ylabel='WETPRAC'>,
        <AxesSubplot:xlabel='WETPRAC', ylabel='WETPRAC'>,
        <AxesSubplot:xlabel='WETQUALI', ylabel='WETPRAC'>,
        <AxesSubplot:xlabel='Q3SECONDS', ylabel='WETPRAC'>,
        <AxesSubplot:xlabel='PRAC3SECONDS', ylabel='WETPRAC'>],
       [<AxesSubplot:xlabel='LAPS', ylabel='WETQUALI'>,
        <AxesSubplot:xlabel='WETPRAC', ylabel='WETQUALI'>,
        <AxesSubplot:xlabel='WETQUALI', ylabel='WETQUALI'>,
        <AxesSubplot:xlabel='Q3SECONDS', ylabel='WETQUALI'>,
        <AxesSubplot:xlabel='PRAC3SECONDS', ylabel='WETQUALI'>],
       [<AxesSubplot:xlabel='LAPS', ylabel='Q3SECONDS'>,
        <AxesSubplot:xlabel='WETPRAC', ylabel='Q3SECONDS'>,
        <AxesSubplot:xlabel='WETQUALI', ylabel='Q3SECONDS'>,
        <AxesSubplot:xlabel='Q3SECONDS', ylabel='Q3SECONDS'>,
        <AxesSubplot:xlabel='PRAC3SECONDS', ylabel='Q3SECONDS'>],
       [<AxesSubplot:xlabel='LAPS', ylabel='PRAC3SECONDS'>,
        <AxesSubplot:xlabel='WETPRAC', ylabel='PRAC3SECONDS'>,
        <AxesSubplot:xlabel='WETQUALI', ylabel='PRAC3SECONDS'>,
        <AxesSubplot:xlabel='Q3SECONDS', ylabel='PRAC3SECONDS'>,
        <AxesSubplot:xlabel='PRAC3SECONDS', ylabel='PRAC3SECONDS'>]],
      dtype=object)
```



In [182]:

```
trainin = train_set.drop("Q3SECONDS", axis=1)
trainin_labels = train_set["Q3SECONDS"].copy()
```

```
trainin_labels
```

Out[182]:

```
1174    92.321
701     96.702
1479    70.166
528     92.142
987     96.237
         ...
1130    96.217
1294    92.364
860     63.130
1459    76.750
1126    80.455
Name: Q3SECONDS, Length: 1272, dtype: float64
```

In [183]:

```
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler
from sklearn.impute import SimpleImputer

num_pipeline = Pipeline([
        ('imputer', SimpleImputer(strategy="median")),
        ('std_scaler', StandardScaler()),
    ])

trainin_num_tr = num_pipeline.fit_transform(trainin)
trainin_num_tr

from sklearn.compose import ColumnTransformer

num_attribs = list(trainin)

full_pipeline = ColumnTransformer([("num", num_pipeline, num_attribs)])

trainin_prepared = full_pipeline.fit_transform(trainin)

trainin_prepared
```

Out[183]:

```
array([[-0.38047487, -0.24213262, -0.28951069,  0.48007869],
       [ 2.52621601,  5.6998485 ,  4.37197788,  1.01533388],
       [-0.38047487, -0.24213262, -0.28951069, -1.28115299],
       ...,
       [-0.13825063, -0.24213262, -0.28951069, -1.81410271],
       [-0.86492335, -0.24213262, -0.28951069, -0.73974985],
       [ 0.34619785, -0.24213262, -0.28951069, -0.53210311]])
```

In [184]:

```
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_absolute_error
from sklearn.metrics import mean_squared_error
lin_reg = LinearRegression()
lin_reg.fit(trainin_prepared, trainin_labels)
```

Out[184]:

```
LinearRegression()
```

In [185]:

```
some_data = trainin.iloc[:5]
some_labels = trainin_labels.iloc[:5]
some_data_prepared = full_pipeline.transform(some_data)

# print("Predictions:", lin_reg.predict(some_data_prepared))

trainin_predictions = lin_reg.predict(trainin_prepared)
# print("Labels:", list(some_labels))
```

```
lin_mae = mean_squared_error(trainin_labels, trainin_predictions)
lin_train_rmse = np.sqrt(lin_mae)
print(lin_train_rmse)
```

```
2.270553927736319
```

In [186]:

```
from sklearn.tree import DecisionTreeRegressor

tree_reg = DecisionTreeRegressor(random_state=42)
tree_reg.fit(trainin_prepared, trainin_labels)
```

Out[186]:

```
DecisionTreeRegressor(random_state=42)
```

In [212]:

```
from sklearn.metrics import mean_squared_error
trainin_predictions = tree_reg.predict(trainin_prepared)
# print(trainin_predictions)
# print(trainin_labels)
tree_mse = mean_squared_error(trainin_labels, trainin_predictions)
tree_train_rmse = np.sqrt(tree_mse)
tree_train_rmse
```

Out[212]:

```
0.04193530694478645
```

In [188]:

```
from sklearn.ensemble import RandomForestRegressor

forest_reg = RandomForestRegressor(n_estimators=100, random_state=42)
forest_reg.fit(trainin_prepared, trainin_labels)
```

Out[188]:

```
RandomForestRegressor(random_state=42)
```

In [189]:

```
trainin_predictions = forest_reg.predict(trainin_prepared)
print(trainin_predictions)
print(trainin_labels)
```

```
[92.52729 97.70577 70.28498 ... 63.15821 76.92424 80.22815]
1174    92.321
701     96.702
1479    70.166
528     92.142
987     96.237
         ...
1130    96.217
1294    92.364
860     63.130
1459    76.750
1126    80.455
Name: Q3SECONDS, Length: 1272, dtype: float64
```

In [190]:

```
forest_mse = mean_squared_error(trainin_labels, trainin_predictions)
forest_train_rmse = np.sqrt(forest_mse)
forest_train_rmse
```

Out[190]:

```
0.5609167779858018
```

In [191]:

```python
def display_scores(scores):
    print("Scores:", scores)
    print("Mean:", scores.mean())
    print("Standard deviation:", scores.std())
```

In [192]:

```python
from sklearn.model_selection import cross_val_score

tree_scores = cross_val_score(tree_reg, trainin_prepared, trainin_labels,
                              scoring="neg_mean_squared_error", cv=10)
tree_rmse_scores = np.sqrt(-tree_scores)
display_scores(tree_rmse_scores)
```

```
Scores: [0.77127077 1.87100041 2.64592845 1.65957357 1.36436877 2.06278512
 0.85066368 1.13602927 1.93072371 1.6947688 ]
Mean: 1.598711254376181
Standard deviation: 0.5497778905846179
```

In [193]:

```python
from sklearn.model_selection import cross_val_score

forest_scores = cross_val_score(forest_reg, trainin_prepared, trainin_labels,
                              scoring="neg_mean_squared_error", cv=10)
forest_rmse_scores = np.sqrt(-forest_scores)
display_scores(forest_rmse_scores)
```

```
Scores: [0.72499733 2.05863792 1.90695698 1.64422999 1.04663353 1.78886521
 0.99724042 1.13919325 1.90929284 1.23366104]
Mean: 1.4449708496566465
Standard deviation: 0.44488534463053275
```

In [194]:

```python
from sklearn.model_selection import cross_val_score

lin_scores = cross_val_score(lin_reg, trainin_prepared, trainin_labels,
                              scoring="neg_mean_squared_error", cv=10)
lin_rmse_scores = np.sqrt(-lin_scores)
display_scores(lin_rmse_scores)
```

```
Scores: [1.86185864 1.6625982  2.45913801 2.83621271 2.79913169 2.42852398
 1.68897711 2.41287877 2.72411467 1.86208121]
Mean: 2.27355149941091
Standard deviation: 0.43903191315891066
```

In [195]:

```python
scores = cross_val_score(lin_reg, trainin_prepared, trainin_labels, scoring="neg_mean_sq
uared_error", cv=10)
pd.Series(np.sqrt(-scores)).describe()
```

Out[195]:

```
count    10.000000
mean      2.273551
std       0.462780
min       1.662598
25%       1.861914
50%       2.420701
75%       2.657871
max       2.836213
dtype: float64
```

In [196]:

```python
from sklearn.svm import SVR

svm_reg = SVR(kernel="linear")
svm_reg.fit(trainin_prepared, trainin_labels)
```

```
trainin_predictions = svm_reg.predict(trainin_prepared)
svm_mse = mean_squared_error(trainin_labels, trainin_predictions)
svm_rmse = np.sqrt(svm_mse)
svm_rmse
svm_mse
```

Out[196]:

```
6.037202588684238
```

In [197]:

```
scores = cross_val_score(svm_reg, trainin_prepared, trainin_labels, scoring="neg_mean_sq
uared_error", cv=10)
pd.Series(np.sqrt(-scores)).describe()
```

Out[197]:

```
count    10.000000
mean      2.376633
std       0.810618
min       1.421286
25%       1.510119
50%       2.646530
75%       2.959227
max       3.463912
dtype: float64
```

In [198]:

```
from sklearn.model_selection import GridSearchCV

param_grid = [
    {'n_estimators': [3, 10, 30], 'max_features': [2, 4]},
    {'bootstrap': [False], 'n_estimators': [3, 10], 'max_features': [2, 3, 4]},
  ]

forest_reg = RandomForestRegressor(n_estimators=100,random_state=42)


grid_search = GridSearchCV(forest_reg, param_grid, cv=5,
                           scoring='neg_mean_squared_error',
                           return_train_score=True)
grid_search.fit(trainin_prepared, trainin_labels)
```

Out[198]:

```
GridSearchCV(cv=5, estimator=RandomForestRegressor(random_state=42),
             param_grid=[{'max_features': [2, 4], 'n_estimators': [3, 10, 30]},
                         {'bootstrap': [False], 'max_features': [2, 3, 4],
                          'n_estimators': [3, 10]}],
             return_train_score=True, scoring='neg_mean_squared_error')
```

In [199]:

```
grid_search.best_params_
```

Out[199]:

```
{'max_features': 4, 'n_estimators': 30}
```

In [200]:

```
grid_search.best_estimator_
```

Out[200]:

```
RandomForestRegressor(max_features=4, n_estimators=30, random_state=42)
```

In [201]:

```
cvres = grid_search.cv_results_
for mean_score, params in zip(cvres["mean_test_score"], cvres["params"]):
```

```
    print(np.sqrt(-mean_score), params)
```

```
1.8893741648717435 {'max_features': 2, 'n_estimators': 3}
1.5808850184342267 {'max_features': 2, 'n_estimators': 10}
1.543780038646839 7 {'max_features': 2, 'n_estimators': 30}
1.5384924442471877 {'max_features': 4, 'n_estimators': 3}
1.54327065092273 9 {'max_features': 4, 'n_estimators': 10}
1.5240752628650178 {'max_features': 4, 'n_estimators': 30}
1.7935074473633 {'bootstrap': False, 'max_features': 2, 'n_estimators': 3}
1.598576337833882 {'bootstrap': False, 'max_features': 2, 'n_estimators': 10}
1.7123339892935068 {'bootstrap': False, 'max_features': 3, 'n_estimators': 3}
1.6554628099886182 {'bootstrap': False, 'max_features': 3, 'n_estimators': 10}
1.7310738673243926 {'bootstrap': False, 'max_features': 4, 'n_estimators': 3}
1.6871418033405912 {'bootstrap': False, 'max_features': 4, 'n_estimators': 10}
```

In [202]:

```
from sklearn.model_selection import RandomizedSearchCV
from scipy.stats import randint

param_distribs = {
        'n_estimators': randint(low=1, high=100),
        'max_features': randint(low=1, high=4),
    }

forest_reg = RandomForestRegressor(random_state=42)
rnd_search = RandomizedSearchCV(forest_reg, param_distributions=param_distribs,
                                n_iter=10, cv=5, scoring='neg_mean_squared_error', rando
m_state=42)
rnd_search.fit(trainin_prepared, trainin_labels)
```

Out[202]:

```
RandomizedSearchCV(cv=5, estimator=RandomForestRegressor(random_state=42),
                   param_distributions={'max_features': <scipy.stats._distn_infrastructur
e.rv_frozen object at 0x7f39a81623a0>,
                                        'n_estimators': <scipy.stats._distn_infrastructu
re.rv_frozen object at 0x7f39dc6738e0>},
                   random_state=42, scoring='neg_mean_squared_error')
```

In [203]:

```
cvres = rnd_search.cv_results_
for mean_score, params in zip(cvres["mean_test_score"], cvres["params"]):
    print(np.sqrt(-mean_score), params)
```

```
1.4865753300311644 {'max_features': 3, 'n_estimators': 52}
1.8375520221160215 {'max_features': 1, 'n_estimators': 15}
1.4836438533224705 {'max_features': 3, 'n_estimators': 72}
1.79056163306225 9 {'max_features': 1, 'n_estimators': 21}
1.4898879518386565 {'max_features': 3, 'n_estimators': 83}
1.4840124207971865 {'max_features': 3, 'n_estimators': 75}
1.49129472625436 {'max_features': 3, 'n_estimators': 88}
1.7560171335284236 {'max_features': 1, 'n_estimators': 24}
1.4353432295959143 {'max_features': 3, 'n_estimators': 22}
2.3038365717516793 {'max_features': 1, 'n_estimators': 2}
```

In [214]:

```
rnd_train_rmse = 1.43
final_model = rnd_search.best_estimator_

X_test = test_set.drop("Q3SECONDS", axis=1)
y_test = test_set["Q3SECONDS"].copy()

X_test_prepared = full_pipeline.transform(X_test)
forest_reg = RandomForestRegressor(n_estimators=100, random_state=42)
# forest_reg.fit(trainin_prepared, trainin_labels)
final_predictions = final_model.predict(X_test_prepared)

# trainin_predictions = forest_reg.predict(X_test_prepared)
# print(X_test_prepared[:2])
```

```
# print(final_predictions[:20])
# print(y_test[:20])
final_mse = mean_squared_error(y_test, final_predictions)
rnd_rmse = np.sqrt(final_mse)
rnd_rmse
```

Out[214]:

```
1.0558769035970659
```

In [213]:

```
X_test = test_set.drop("Q3SECONDS", axis=1)
y_test = test_set["Q3SECONDS"].copy()

X_test_prepared = full_pipeline.transform(X_test)
forest_reg = RandomForestRegressor(n_estimators=100, random_state=42)
forest_reg.fit(trainin_prepared, trainin_labels)
final_predictions = forest_reg.predict(X_test_prepared)
# print(final_predictions[:20])
# print(y_test[:20])
final_mse = mean_squared_error(y_test, final_predictions)
forest_rmse = np.sqrt(final_mse)
forest_rmse
```

Out[213]:

```
0.9141122245539044
```

In [211]:

```
X_test = test_set.drop("Q3SECONDS", axis=1)
y_test = test_set["Q3SECONDS"].copy()

X_test_prepared = full_pipeline.transform(X_test)
tree_reg = DecisionTreeRegressor(random_state=42)
tree_reg.fit(trainin_prepared, trainin_labels)
final_predictions = tree_reg.predict(X_test_prepared)
# print(final_predictions[:20])
# print(y_test[:20])
final_mse = mean_squared_error(y_test, final_predictions)
tree_rmse = np.sqrt(final_mse)
tree_rmse
```

Out[211]:

```
1.0502935610891886
```

In [207]:

```
X_test = test_set.drop("Q3SECONDS", axis=1)
y_test = test_set["Q3SECONDS"].copy()

X_test_prepared = full_pipeline.transform(X_test)
lin_reg = LinearRegression()
lin_reg.fit(trainin_prepared, trainin_labels)
final_predictions = lin_reg.predict(X_test_prepared)
# print(final_predictions[:20])
# print(y_test[:20])
final_mse = mean_squared_error(y_test, final_predictions)
lin_rmse = np.sqrt(final_mse)
lin_rmse
```

Out[207]:

```
2.2444874850419243
```

In [208]:

```
from scipy import stats

confidence = 0.95
squared_errors = (final_predictions - y_test) ** 2
```

```
np.sqrt(stats.t.interval(confidence, len(squared_errors) - 1,
                         loc=squared_errors.mean(),
                         scale=stats.sem(squared_errors)))
```

Out[208]:

```
array([1.54205053, 2.77444198])
```

In [209]:

```
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
import pandas as pd
import numpy as np
import tensorflow as tf
from tensorflow import keras
training = pd.read_csv('TRAIN1.csv')
trainin = training.drop("Q3SECONDS", axis=1)
X_train_full, X_test, y_train_full, y_test = train_test_split(trainin, training.Q3SECOND
S, random_state=42)
X_train, X_valid, y_train, y_valid = train_test_split(X_train_full, y_train_full, random
_state=42)
# print(X_train_full)
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_valid = scaler.transform(X_valid)
X_test1 = scaler.transform(X_test)


np.random.seed(42)
tf.random.set_seed(42)
model = keras.models.Sequential([
    keras.layers.Dense(100, activation="relu", input_shape=X_train.shape[1:]),
    keras.layers.Dense(1)
])
model.compile(loss="mean_squared_error", optimizer=keras.optimizers.SGD(learning_rate=1e-
3))
history = model.fit(X_train, y_train, epochs=50, validation_data=(X_valid, y_valid))
mse_test = model.evaluate(X_test, y_test)
X_new = X_test1
y_pred = model.predict(X_new)
# print(y_pred)
# print(X_test[:10])
from sklearn.metrics import mean_squared_error
listofTrue = y_test.values.tolist()
final_mse = mean_squared_error(listofTrue,y_pred)
final_rmse = np.sqrt(final_mse)
final_rmse
```

```
Epoch 1/50
28/28 [==============================] - 0s 3ms/step - loss: 3972.8516 - val_loss: 36.975
4
Epoch 2/50
28/28 [==============================] - 0s 1ms/step - loss: 21.3568 - val_loss: 15.0190
Epoch 3/50
28/28 [==============================] - 0s 1ms/step - loss: 13.7094 - val_loss: 12.5989
Epoch 4/50
28/28 [==============================] - 0s 2ms/step - loss: 11.7783 - val_loss: 11.3939
Epoch 5/50
28/28 [==============================] - 0s 1ms/step - loss: 10.5105 - val_loss: 9.3386
Epoch 6/50
28/28 [==============================] - 0s 1ms/step - loss: 9.2135 - val_loss: 9.7567
Epoch 7/50
28/28 [==============================] - 0s 1ms/step - loss: 9.1301 - val_loss: 7.9897
Epoch 8/50
28/28 [==============================] - 0s 1ms/step - loss: 7.8995 - val_loss: 7.3820
Epoch 9/50
28/28 [==============================] - 0s 2ms/step - loss: 7.3705 - val_loss: 6.8640
Epoch 10/50
28/28 [==============================] - 0s 1ms/step - loss: 6.9956 - val_loss: 6.7019
Epoch 11/50
28/28 [==============================] - 0s 1ms/step - loss: 6.6881 - val_loss: 6.0571
Epoch 12/50
```

```
Epoch 12/50
28/28 [==============================] - 0s 1ms/step - loss: 6.1788 - val_loss: 5.7141
Epoch 13/50
28/28 [==============================] - 0s 1ms/step - loss: 6.0304 - val_loss: 5.2004
Epoch 14/50
28/28 [==============================] - 0s 1ms/step - loss: 5.6594 - val_loss: 4.9974
Epoch 15/50
28/28 [==============================] - 0s 2ms/step - loss: 5.4017 - val_loss: 5.1497
Epoch 16/50
28/28 [==============================] - 0s 2ms/step - loss: 5.2700 - val_loss: 4.7796
Epoch 17/50
28/28 [==============================] - 0s 1ms/step - loss: 5.4485 - val_loss: 4.7712
Epoch 18/50
28/28 [==============================] - 0s 1ms/step - loss: 5.2646 - val_loss: 4.6529
Epoch 19/50
28/28 [==============================] - 0s 1ms/step - loss: 4.8939 - val_loss: 4.5103
Epoch 20/50
28/28 [==============================] - 0s 1ms/step - loss: 4.9116 - val_loss: 4.3805
Epoch 21/50
28/28 [==============================] - 0s 1ms/step - loss: 4.8861 - val_loss: 4.3111
Epoch 22/50
28/28 [==============================] - 0s 1ms/step - loss: 4.6229 - val_loss: 4.7995
Epoch 23/50
28/28 [==============================] - 0s 1ms/step - loss: 4.8250 - val_loss: 4.3168
Epoch 24/50
28/28 [==============================] - 0s 1ms/step - loss: 4.7296 - val_loss: 4.1153
Epoch 25/50
28/28 [==============================] - 0s 1ms/step - loss: 4.5931 - val_loss: 4.1624
Epoch 26/50
28/28 [==============================] - 0s 1ms/step - loss: 4.5077 - val_loss: 4.0055
Epoch 27/50
28/28 [==============================] - 0s 1ms/step - loss: 4.3661 - val_loss: 4.0426
Epoch 28/50
28/28 [==============================] - 0s 1ms/step - loss: 4.2536 - val_loss: 4.3859
Epoch 29/50
28/28 [==============================] - 0s 1ms/step - loss: 4.4908 - val_loss: 4.1042
Epoch 30/50
28/28 [==============================] - 0s 1ms/step - loss: 4.2972 - val_loss: 3.9055
Epoch 31/50
28/28 [==============================] - 0s 1ms/step - loss: 4.2585 - val_loss: 3.7921
Epoch 32/50
28/28 [==============================] - 0s 1ms/step - loss: 4.2372 - val_loss: 3.7442
Epoch 33/50
28/28 [==============================] - 0s 2ms/step - loss: 3.9303 - val_loss: 4.4576
Epoch 34/50
28/28 [==============================] - 0s 1ms/step - loss: 4.3091 - val_loss: 3.6791
Epoch 35/50
28/28 [==============================] - 0s 1ms/step - loss: 4.0793 - val_loss: 3.6280
Epoch 36/50
28/28 [==============================] - 0s 1ms/step - loss: 4.1464 - val_loss: 3.6964
Epoch 37/50
28/28 [==============================] - 0s 1ms/step - loss: 3.8067 - val_loss: 3.9472
Epoch 38/50
28/28 [==============================] - 0s 1ms/step - loss: 3.9222 - val_loss: 3.5610
Epoch 39/50
28/28 [==============================] - 0s 1ms/step - loss: 3.9819 - val_loss: 3.5404
Epoch 40/50
28/28 [==============================] - 0s 1ms/step - loss: 4.0824 - val_loss: 3.5892
Epoch 41/50
28/28 [==============================] - 0s 1ms/step - loss: 4.0211 - val_loss: 3.4693
Epoch 42/50
28/28 [==============================] - 0s 1ms/step - loss: 3.9082 - val_loss: 3.5109
Epoch 43/50
28/28 [==============================] - 0s 1ms/step - loss: 3.8392 - val_loss: 3.4456
Epoch 44/50
28/28 [==============================] - 0s 1ms/step - loss: 3.7881 - val_loss: 3.4674
Epoch 45/50
28/28 [==============================] - 0s 1ms/step - loss: 3.8355 - val_loss: 3.3719
Epoch 46/50
28/28 [==============================] - 0s 1ms/step - loss: 3.7454 - val_loss: 3.5015
Epoch 47/50
28/28 [==============================] - 0s 1ms/step - loss: 3.7784 - val_loss: 3.3838
Epoch 48/50
```

```
Epoch 48/50
28/28 [==============================] - 0s 1ms/step - loss: 3.8415 - val_loss: 3.3922
Epoch 49/50
28/28 [==============================] - 0s 1ms/step - loss: 3.6843 - val_loss: 3.4433
Epoch 50/50
28/28 [==============================] - 0s 1ms/step - loss: 3.5721 - val_loss: 3.2821
13/13 [==============================] - 0s 494us/step - loss: 3096103.5000
13/13 [==============================] - 0s 493us/step
```

Out[209]:

```
1.8368717454928687
```

In [210]:

```python
train_acc=np.array([forest_train_rmse,tree_train_rmse,lin_train_rmse,rnd_train_rmse])

test_acc=np.array([forest_rmse,tree_rmse,lin_rmse,rnd_rmse,final_rmse])
print(train_acc)
print(test_acc)
fig = plt.figure(figsize = (10, 5))
plt.ylim(0,2.5)
plt.ylabel("Root Mean Squared Error")
plt.xlabel("Algorithm")
plt.bar(height=test_acc,x=["Random Forest","Decision Tree","Linear Regression","Random Se
arch","Keras"], color =["forestgreen","peru","darkgrey","hotpink"],
        width = 0.6)
plt.show()
```

```
[0.56091678 0.04193531 2.27055393 1.43      ]
[0.91411222 1.05029356 2.24448749 1.0558769  1.83687175]
```