Introduction to Machine Learning

EE-323

Project Report

Prediction of Steering Angle using Throttle and Road Angle Values

Submitted to:
Mr. Rohan Pillai
Assistant Professor
Department of Electrical Engineering

Made by:

Yatharth Ahuja
2K18/EE/249

Shreyan Sood
2K18/MC/110

CONTENTS

INTRODUCTION

The main objective of this project is to create and compare different models for the prediction of steering angle for a vehicle using the values of Throttle and Road angles as input.

Road images and corresponding throttle and steering angle values were obtained from a Unity3D simulation of an unmanned autonomous ground vehicle being run in a virtual road environment. Each of the 2018 data points obtained contains road image along with a tuple having steering angle and throttle values. The images were processed using Gaussian Blur, Sobel Filter, Non-Maxima Suppression in progression. Then geometric formulas are deployed to obtain the angle of edges and hence angle of the road in image. All these corresponding angles are recorded to provide a parameter vector. All the parameter values were stored in convenient format and analysed further for

For the prediction models we used the following algorithms: Polynomial Regression, Decision Tree Regression, Random Forest.
The results produced by models upon training on the obtained dataset were then evaluated on the metrics: R-square and RMSE. The evaluations were then compared to provide a conclusion.

This study helped in practical exploration of mathematical basics and intricacies of analysing a dataset. Finding patterns in the data provides avenues in automation on existing and abundant data. Thus, the aforementioned Steering Angle prediction project tries to help in realising the true intentions of this course.

DATASET BUILDING

2018 raw data points were obtained by recording different instances in the track run of an off-the-shelf autonomous ground vehicle bot in Unity3D simulation.

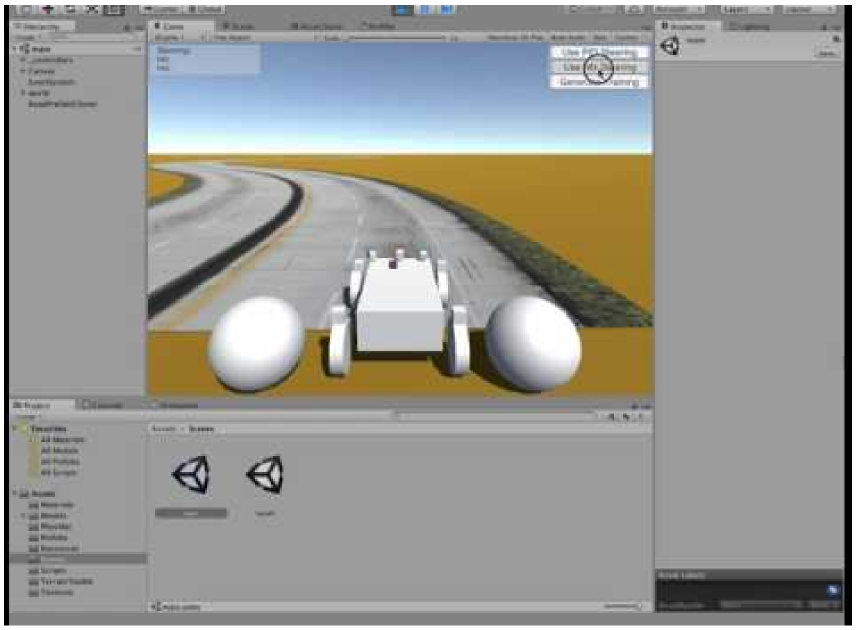
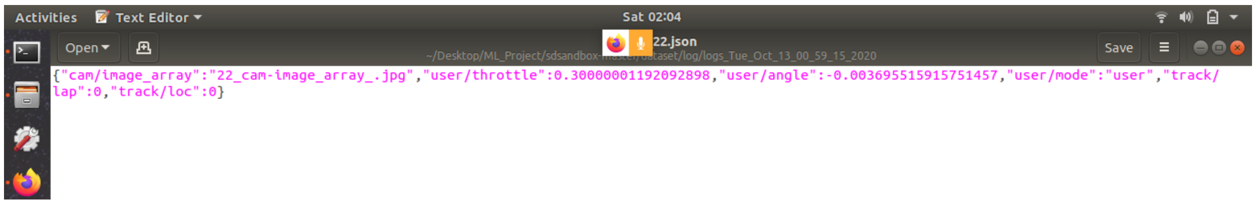


Figure: Track simulation and bot environment

Each individual sample includes an image of the road and a corresponding JSON file containing throttle values and steering angle. Following entails the remaining process of dataset building dine in this project:

A. *Combining Multiple JSON records containing throttle and steering angle into a single CSV file*

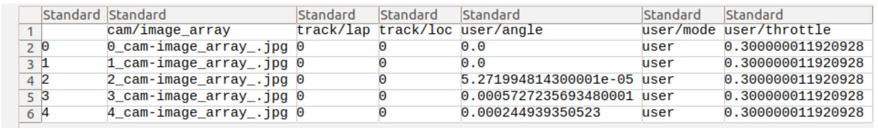
The default format in which steering angle and throttle values were obtained directly from simulation is JSON, which is inherently difficult to process in *python* or *MATLAB*.


Activities
Text Editor
Sat 02:04
Open
22.json
~/Desktop/ML_Project/sdsandbox-master/dataset/log/logs_Tue_Oct_13_00_59_15_2020
Save

```
{"cam/image_array":"22_cam-image_array_.jpg","user/throttle":0.30000001192092898,"user/angle":-0.003695515915751457,"user/mode":"user","track/
lap":0,"track/loc":0}
```

Figure: Individual raw data sample in JSON

Hence, these 2018 samples were individually converted into CSV format.
And then combined into a single CSV file.



	Standard	Standard	Standard	Standard	Standard	Standard	Standard
1		cam/image_array	track/lap	track/loc	user/angle	user/mode	user/throttle
2	0	0_cam-image_array_.jpg	0	0	0.0	user	0.300000011920928
3	1	1_cam-image_array_.jpg	0	0	0.0	user	0.300000011920928
4	2	2_cam-image_array_.jpg	0	0	5.271994814300001e-05	user	0.300000011920928
5	3	3_cam-image_array_.jpg	0	0	0.0005727235693480001	user	0.300000011920928
6	4	4_cam-image_array_.jpg	0	0	0.000244939350523	user	0.300000011920928

Figure: Individual data samples in JSON as combined CSV

B. Image processing each frame to extract road angles

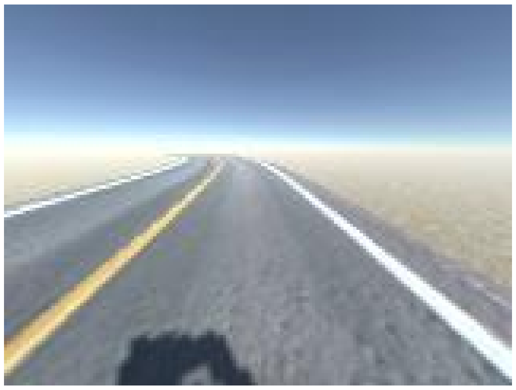
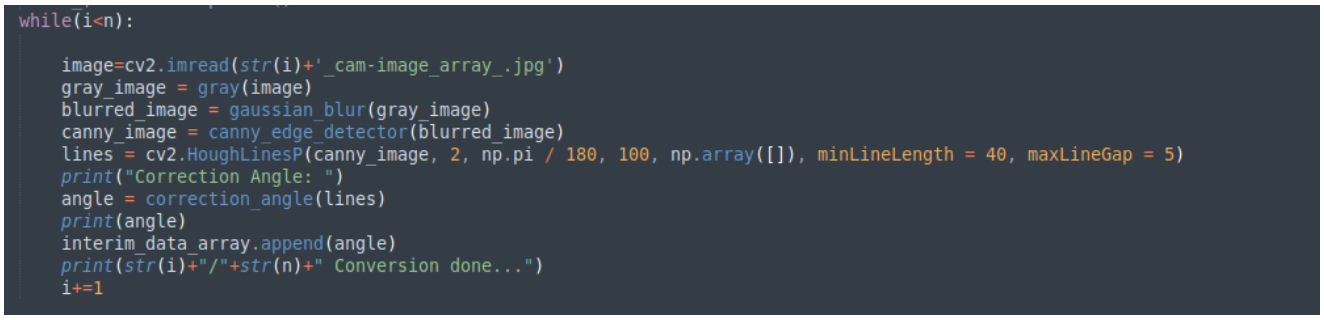


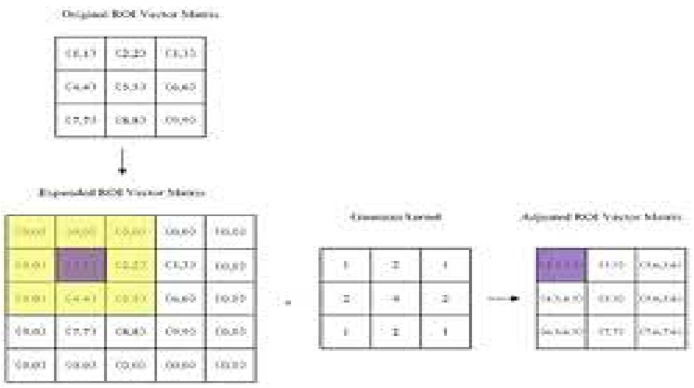
Figure: Track image sample for road angle

Each image associated with 2018 data samples were processed in following manner to extract road angle feature:


```
while(i<n):

    image=cv2.imread(str(i)+'_cam-image_array_.jpg')
    gray_image = gray(image)
    blurred_image = gaussian_blur(gray_image)
    canny_image = canny_edge_detector(blurred_image)
    lines = cv2.HoughLinesP(canny_image, 2, np.pi / 180, 100, np.array([]), minLineLength = 40, maxLineGap = 5)
    print("Correction Angle: ")
    angle = correction_angle(lines)
    print(angle)
    interim_data_array.append(angle)
    print(str(i)+"/"+str(n)+" Conversion done...")
    i+=1
```
Figure: Code snippet depicting steps involved in image processing for feature extraction

1. <u>Graying and Gaussian Blur:</u>

Graying helps us by increasing the contrast of the colours, making it easier to identify changes in pixel intensity.
The purpose of the Gaussian Blur filter is to reduce noise in the image. We do this because the gradients in Canny are really sensitive to noise, so we want to eliminate the most noise possible. It is done by applying a kernel across the image with weights obeying gaussian distribution.


Original ROI Vector Matrix
Expanded ROI Vector Matrix
Gaussian Kernel
Adjusted ROI Vector Matrix
1 2 1
2 4 2
1 2 1

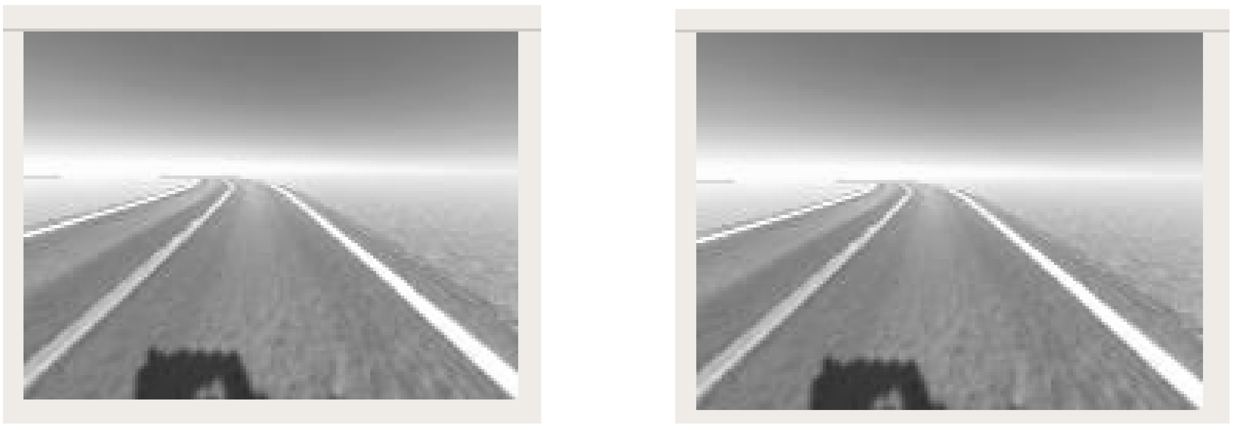


FIgure: Greyscaling and Gaussian Blurring ___

2. <u>Canny Edge Detector:</u>

Canny Edge Detection is used to detect the edges in an image. It accepts a gray scale image as input and it uses a multistage algorithm. It deploys intensity gradient based filters to detect changes and hence concludes an edge.

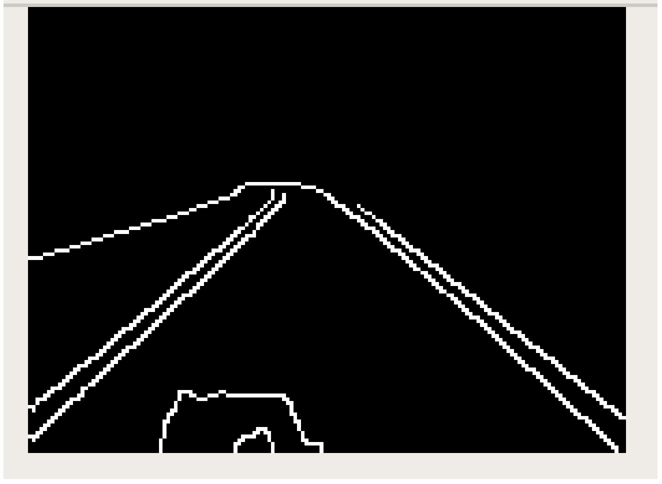
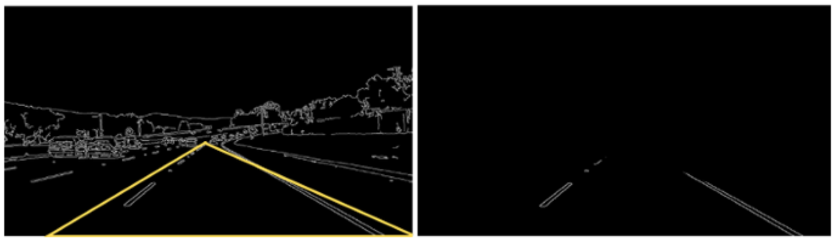


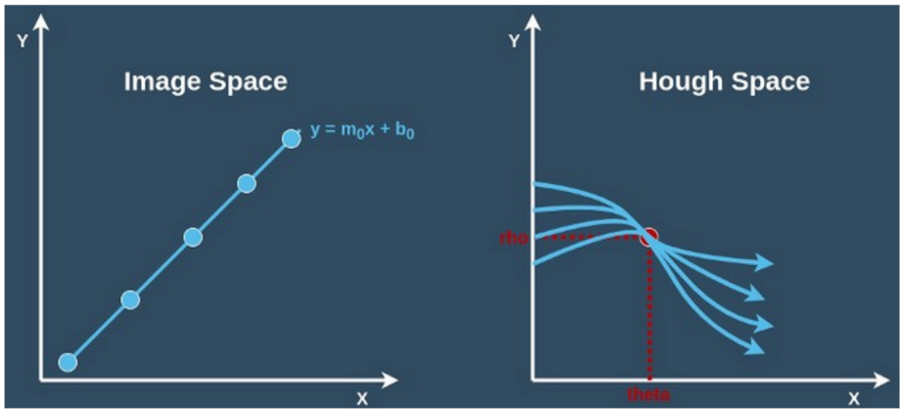
Figure: Image after applying Canny edge detector

3. <u>Masking and Hough Line Transform:</u>

Masking is done to isolate a certain hard-coded region in the image where the lane lines are. It takes one parameter, the Canny image and outputs the isolated region. It helps in increasing the computation speed and removing false positives for the detectors.



Hough transform is a feature extraction method for detecting simple shapes such as circles, lines, etc in an image. It does so by plotting the illuminated points in the hough space, where it deduces the most suitable line shape depicted and outputs these ascertained lines in tuple format.



4. <u>Calculating Road Angle:</u>

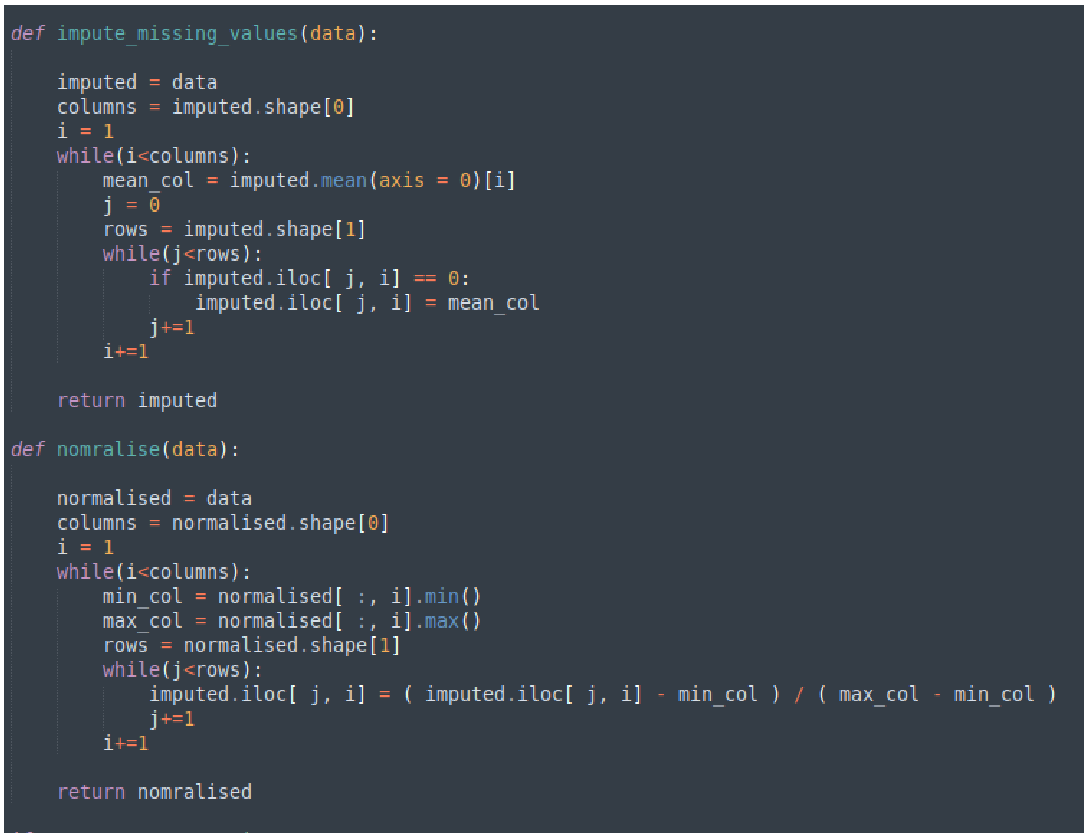
The line coordinate tuples obtained from Hough Transform are processed to output a mean slope of aggregate coordinates of all the road lines in the processed image. These road angle values are stored as intermediate CSV to be saved in the master dataset.

C. ***Combining all the obtained road angles and throttle and steering angle into a single master CSV file***

The road angles CSV file and steering angle+throttle value CSV files are merged together removing redundant columns to give a penultimate dataset.

D. ***Normalisation and Imputation of the dataset obtained***

Final dataset is obtained by normalizing the column data and imputing the missing values by placing column means in their stead.

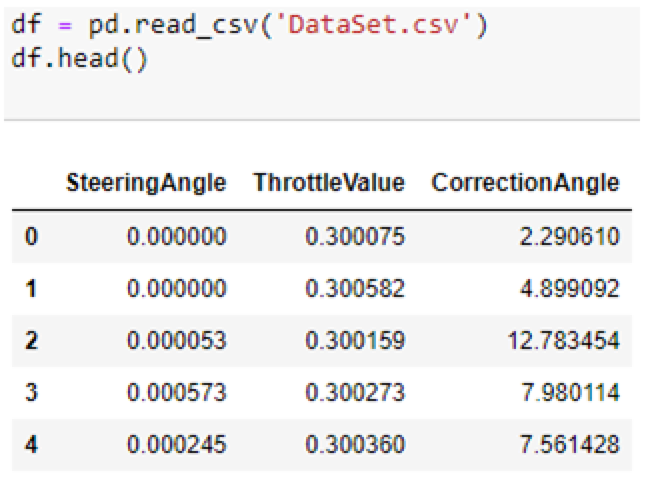


```
def impute_missing_values(data):

    imputed = data
    columns = imputed.shape[0]
    i = 1
    while(i<columns):
        mean_col = imputed.mean(axis = 0)[i]
        j = 0
        rows = imputed.shape[1]
        while(j<rows):
            if imputed.iloc[ j, i] == 0:
                imputed.iloc[ j, i] = mean_col
            j+=1
        i+=1

    return imputed

def nomralise(data):

    normalised = data
    columns = normalised.shape[0]
    i = 1
    while(i<columns):
        min_col = normalised[ :, i].min()
        max_col = normalised[ :, i].max()
        rows = normalised.shape[1]
        while(j<rows):
            imputed.iloc[ j, i] = ( imputed.iloc[ j, i] - min_col ) / ( max_col - min_col )
            j+=1
        i+=1

    return nomralised
```

Figure: Code snippet used for normalisation and imputation

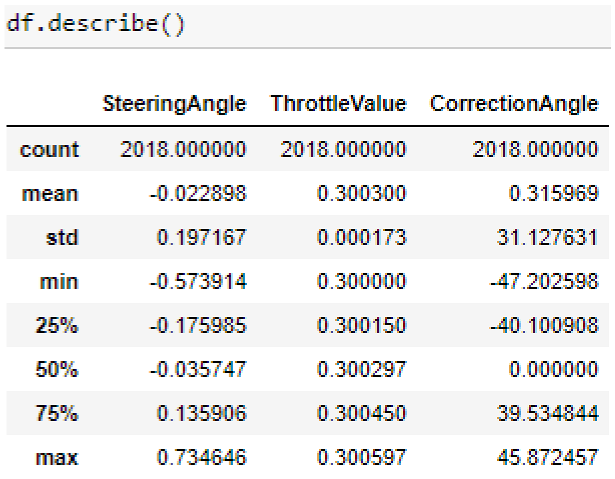
Hence, we obtain the dataset used in the proceeding sections.

DATASET ANALYSIS



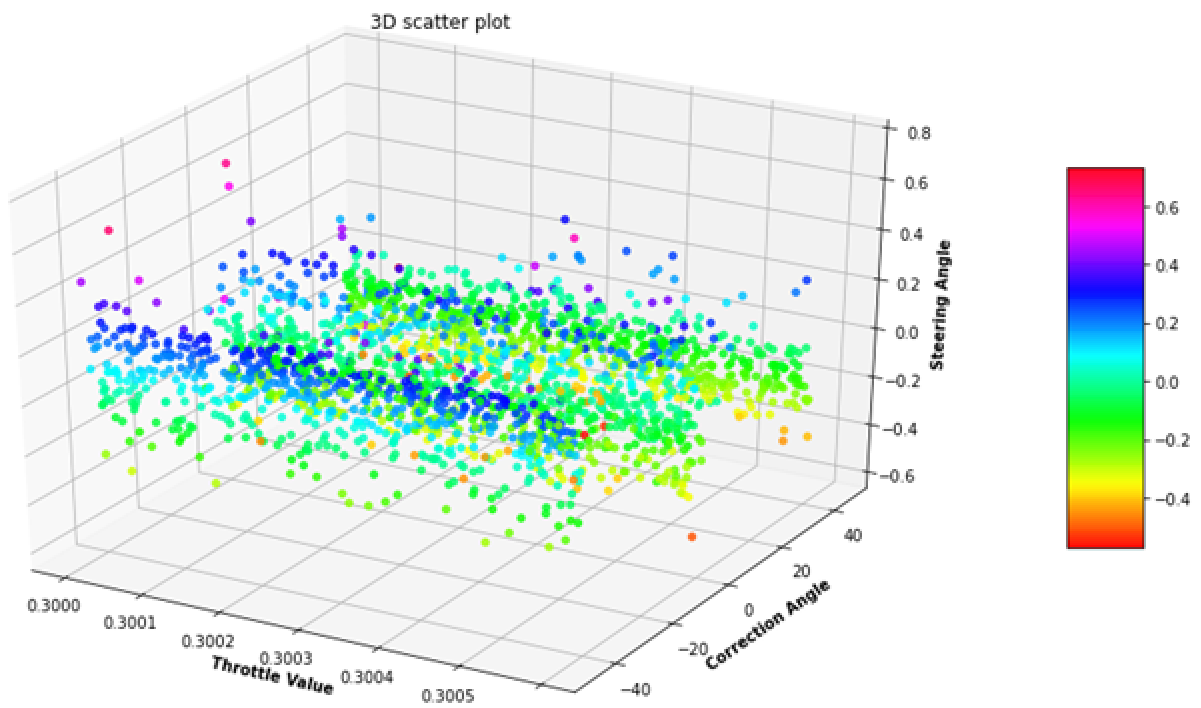
```
df = pd.read_csv('DataSet.csv')
df.head()
```

	SteeringAngle	ThrottleValue	CorrectionAngle
0	0.000000	0.300075	2.290610
1	0.000000	0.300582	4.899092
2	0.000053	0.300159	12.783454
3	0.000573	0.300273	7.980114
4	0.000245	0.300360	7.561428

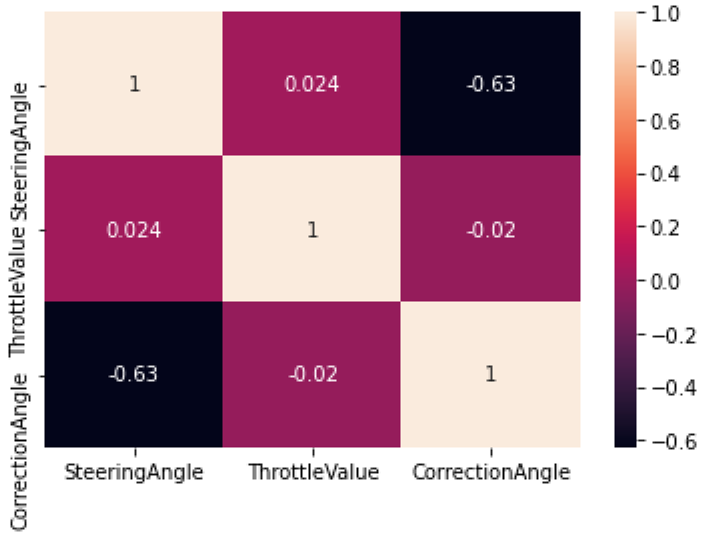


```
df.describe()
```

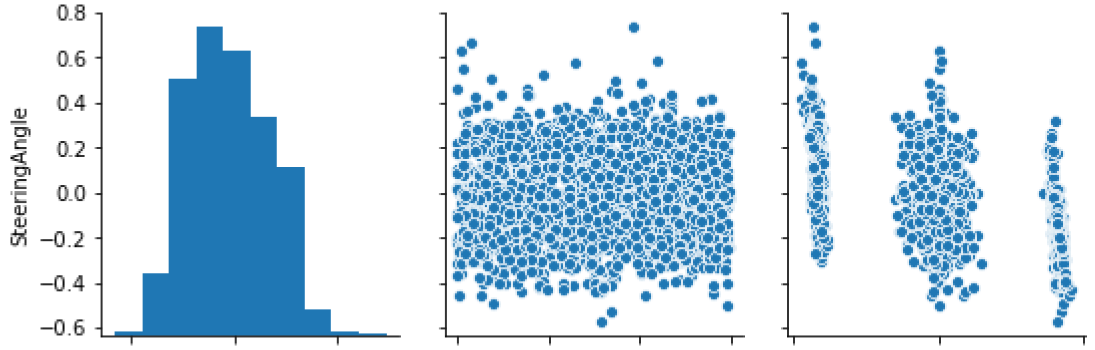
	SteeringAngle	ThrottleValue	CorrectionAngle
count	2018.000000	2018.000000	2018.000000
mean	-0.022898	0.300300	0.315969
std	0.197167	0.000173	31.127631
min	-0.573914	0.300000	-47.202598
25%	-0.175985	0.300150	-40.100908
50%	-0.035747	0.300297	0.000000
75%	0.135906	0.300450	39.534844
max	0.734646	0.300597	45.872457



```
sns.heatmap(df.corr(),annot=True)
plt.show()
```



```
sns.pairplot(data=df)
plt.show()
```



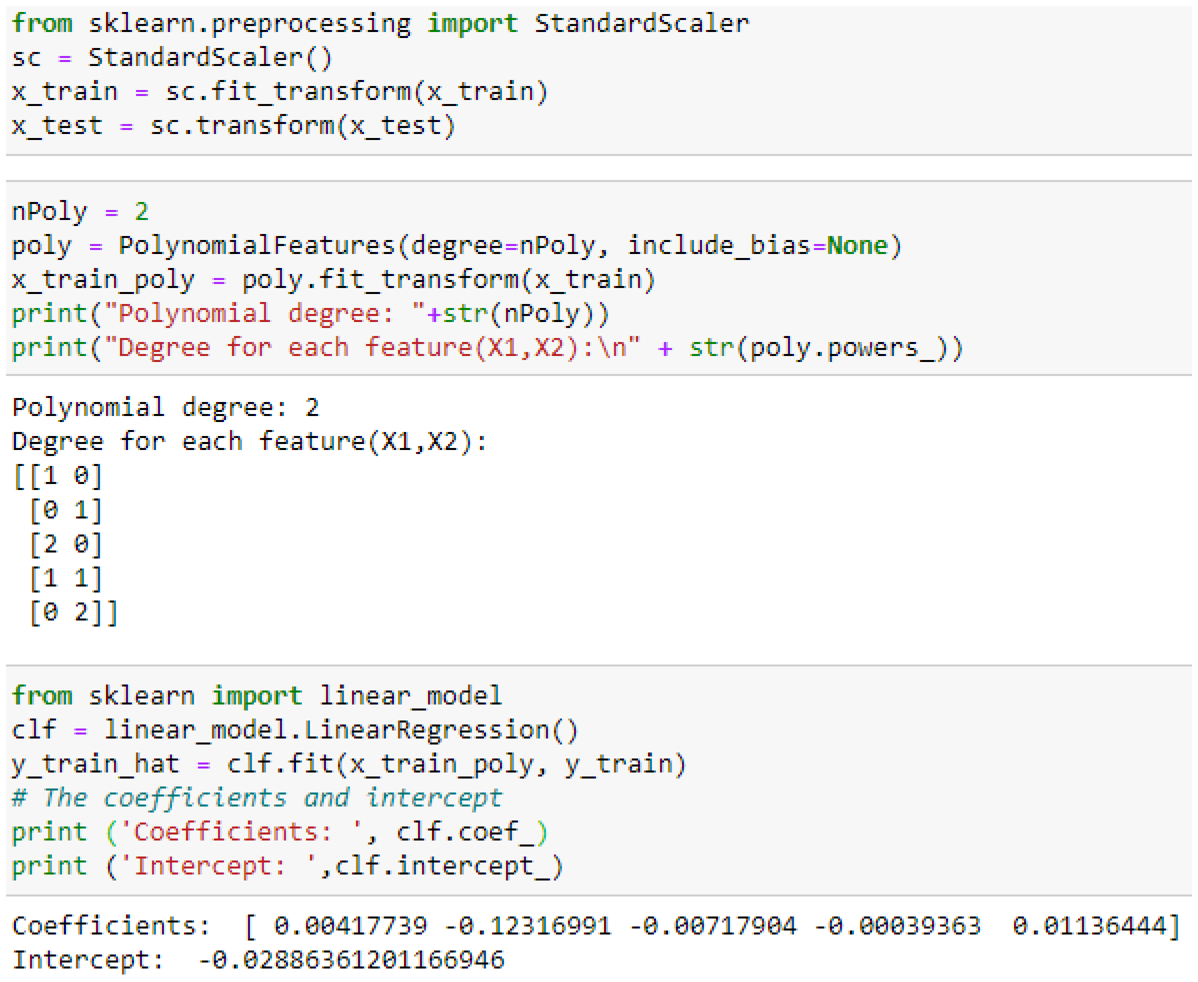
- In the dataset, Steering Angle acts as the output variable while Throttle Value and our self calculated Correction (Road) Angle are the two input variables.

- Data analysis revealed that our dataset is highly cluttered and there is minimal direct correlation between the input and output variables.

- Because of the highly random dataset, we expect the RMSE and R squared values to vary for different prediction models.

APPLYING VARIOUS ALGORITHM MODELS

1. Polynomial Regression (Degree 2)



```
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
x_train = sc.fit_transform(x_train)
x_test = sc.transform(x_test)
```

```
nPoly = 2
poly = PolynomialFeatures(degree=nPoly, include_bias=None)
x_train_poly = poly.fit_transform(x_train)
print("Polynomial degree: "+str(nPoly))
print("Degree for each feature(X1,X2):\n" + str(poly.powers_))
```

```
Polynomial degree: 2
Degree for each feature(X1,X2):
[[1 0]
 [0 1]
 [2 0]
 [1 1]
 [0 2]]
```

```
from sklearn import linear_model
clf = linear_model.LinearRegression()
y_train_hat = clf.fit(x_train_poly, y_train)
# The coefficients and intercept
print ('Coefficients: ', clf.coef_)
print ('Intercept: ',clf.intercept_)
```

```
Coefficients:  [ 0.00417739 -0.12316991 -0.00717904 -0.00039363  0.01136444]
Intercept:  -0.02886361201166946
```

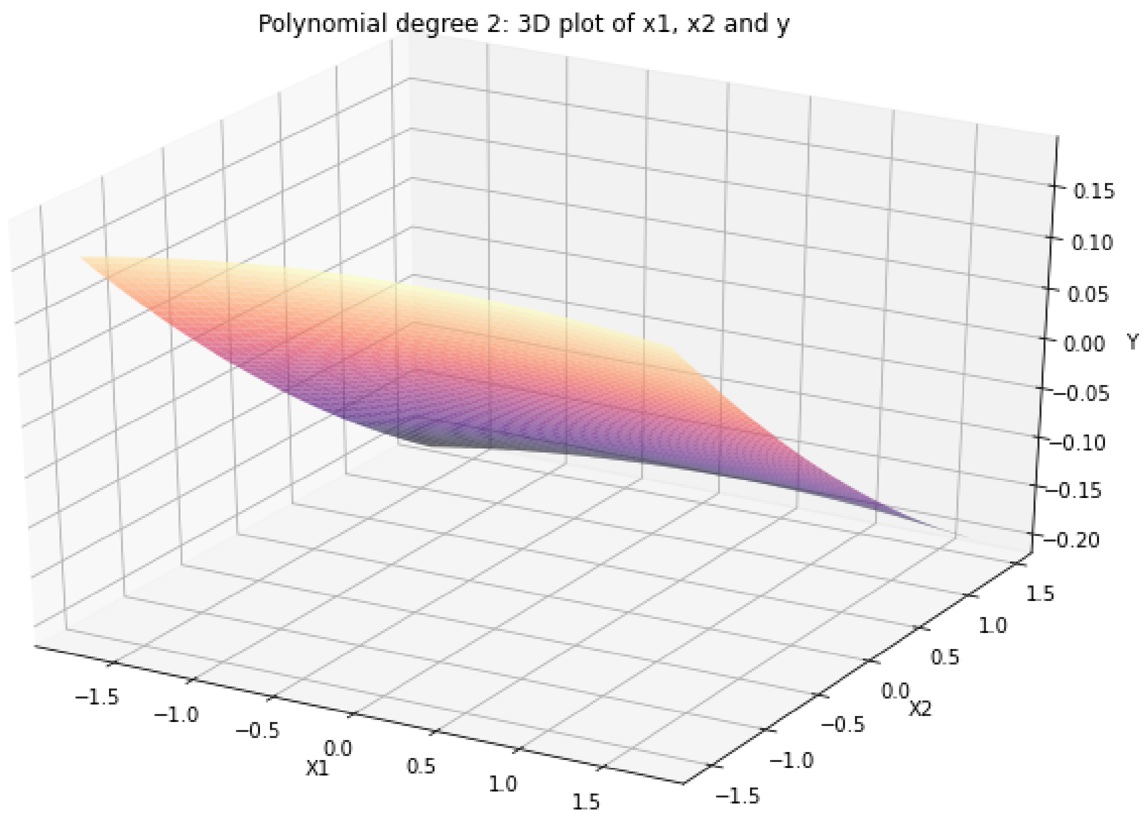
```
print("Equation: " + Poly_equation(['X1','X2'], clf.coef_, clf.intercept_, poly.powers_))
```

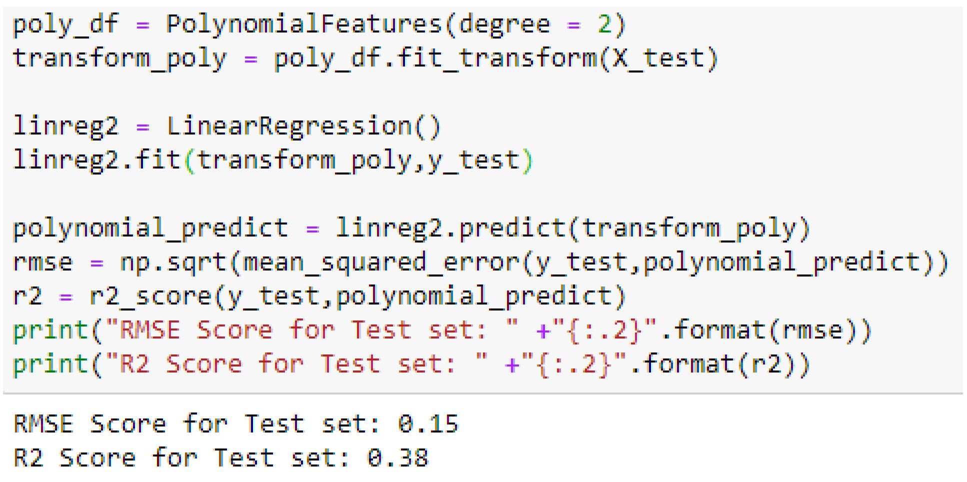
```
Equation: y = 4.177E-03*X1-1.232E-01*X2-7.179E-03*X1^2-3.936E-04*X1*X2+1.136E-02*X2^2-2.886E-02
```

```
x1 = x_train[:,0]
x2 = x_train[:,1]
y = y_train
```

```
x1_lin = np.linspace(min(x1),max(x1))
x2_lin = np.linspace(min(x2),max(x2))
x1_grid, x2_grid = np.meshgrid(x1_lin, x2_lin)
y_grid =y = (0.004177*x1_grid)-(0.1232*x2_grid)-(0.007179*x1_grid**2)-(0.00039*x1_grid*x2_grid)
            +(0.01136*x2_grid**2)-0.0288
```

```
import matplotlib.cm as cm
fig = plt.figure(figsize = (12, 8))
ax = fig.add_subplot(111, projection='3d')
#Line plot
ax.plot_surface(x1_grid, x2_grid, y_grid, cmap = cm.magma, alpha = 0.7)
#Labels
ax.set_xlabel("X1")
ax.set_ylabel("X2")
ax.set_zlabel("Y")
ax.set_title("Polynomial degree 2: 3D plot of x1, x2 and y")
plt.show()
```

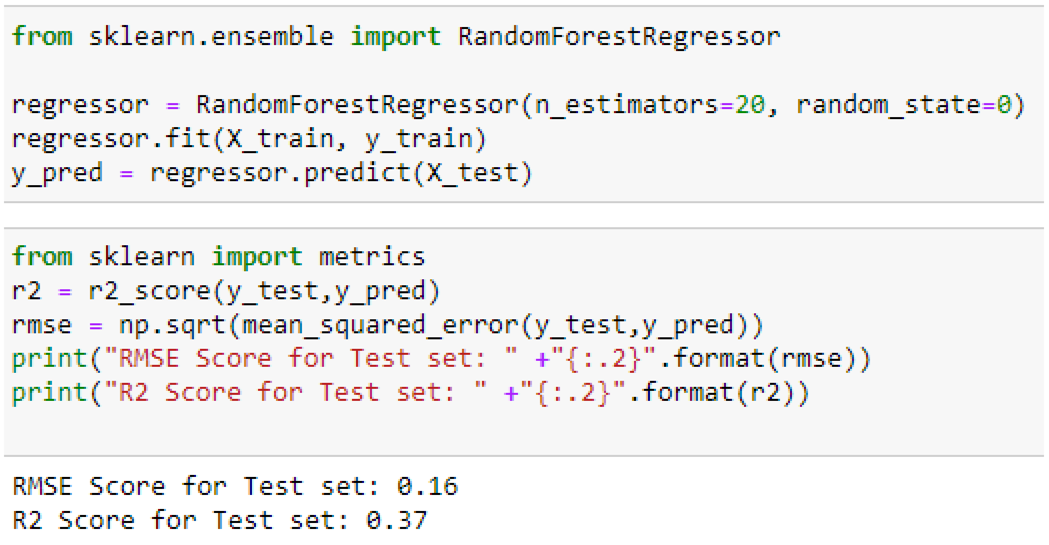




```python
poly_df = PolynomialFeatures(degree = 2)
transform_poly = poly_df.fit_transform(X_test)

linreg2 = LinearRegression()
linreg2.fit(transform_poly,y_test)

polynomial_predict = linreg2.predict(transform_poly)
rmse = np.sqrt(mean_squared_error(y_test,polynomial_predict))
r2 = r2_score(y_test,polynomial_predict)
print("RMSE Score for Test set: " +"{:.2}".format(rmse))
print("R2 Score for Test set: " +"{:.2}".format(r2))
```

```
RMSE Score for Test set: 0.15
R2 Score for Test set: 0.38
```

2. Random Forest

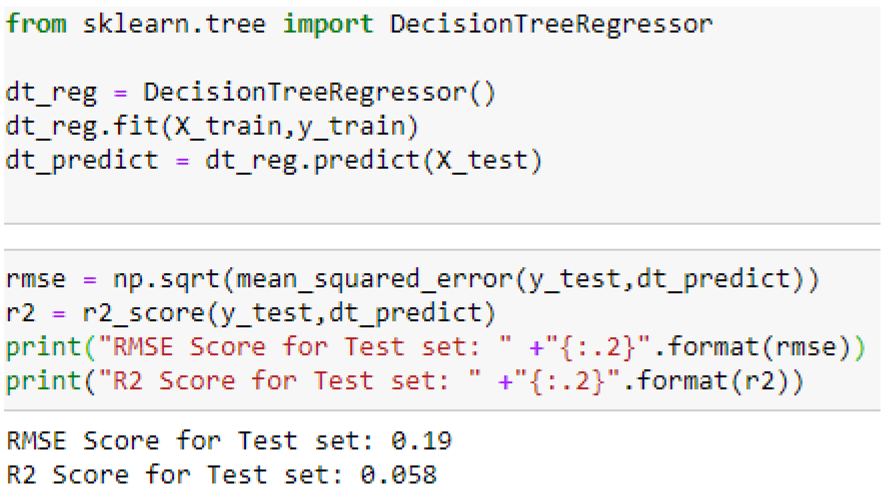


```python
from sklearn.ensemble import RandomForestRegressor

regressor = RandomForestRegressor(n_estimators=20, random_state=0)
regressor.fit(X_train, y_train)
y_pred = regressor.predict(X_test)
```

```python
from sklearn import metrics
r2 = r2_score(y_test,y_pred)
rmse = np.sqrt(mean_squared_error(y_test,y_pred))
print("RMSE Score for Test set: " +"{:.2}".format(rmse))
print("R2 Score for Test set: " +"{:.2}".format(r2))
```

```
RMSE Score for Test set: 0.16
R2 Score for Test set: 0.37
```

3. Decision Tree Regression



```python
from sklearn.tree import DecisionTreeRegressor

dt_reg = DecisionTreeRegressor()
dt_reg.fit(X_train,y_train)
dt_predict = dt_reg.predict(X_test)
```

```python
rmse = np.sqrt(mean_squared_error(y_test,dt_predict))
r2 = r2_score(y_test,dt_predict)
print("RMSE Score for Test set: " +"{:.2}".format(rmse))
print("R2 Score for Test set: " +"{:.2}".format(r2))
```

```
RMSE Score for Test set: 0.19
R2 Score for Test set: 0.058
```

<u>RESULTS & CONCLUSION</u>

Table showing the results of the evaluation metrics used for the models

S.No.	Algorithm Model	RMSE	R squared
1.	Polynomial Regression	0.15	0.38
2.	Random Forest	0.16	0.37
3.	Decision Tree Regression	0.19	0.058

Based on the results, we can conclude that the Decision Tree Regression Model was the worst model for our dataset as it has the minimum R squared score and the maximum RMSE value.
The results also signify that Polynomial Regression and Random Forest models are suitable for our dataset and are almost equally as good prediction models for the dataset..

REFERENCES

- A Deep Dive into Lane Detection with Hough Transform
- Understanding & Implementing Shape Detection using Hough Transform with OpenCV & Python
- Car Data from Unity Simulation
- SdSandbox Simulation
- LR, MLR, PR, DT, RF predict data CalCOFI
- Polynomial Regression in Python
- Self-Driving Car Steering Angle Prediction Based on Image Recognition
- Multivariate Polynomial Regression Python

fin.