This is just my notes, writing up a proper document.
https://github.com/TheHarlander/704_2_DevLog
Will be working on proper one and updating to github.

https://medium.com/all-things-ai/in-depth-parameter-tuning-for-random-forest-d67bb7e920d

N_estimators

n_estimators represents the number of trees in the forest. Usually the higher the number of trees the better to learn the data. However, adding a lot of trees can slow down the training process considerably, therefore we do a parameter search to find the sweet spot.

max_depth

max_depth represents the depth of each tree in the forest. The deeper the tree, the more splits it has and it captures more information about the data. We fit each decision tree with depths ranging from 1 to 32 and plot the training and test errors.

min_samples_leaf

min_samples_leaf is The minimum number of samples required to be at a leaf node. This parameter is similar to min_samples_splits, however, this describe the minimum number of samples of samples at the leafs, the base of the tree.

max_features

max_features represents the number of features to consider when looking for the best split.

https://medium.com/all-things-ai/in-depth-parameter-tuning-for-gradient-boosting-3363992e9bae another info one

((((((

Section idea: Data

What data

How much data

Problems throughout

```
Traceback (most recent call last):
  File "C:/Users/Dean/Documents/University/Comp704Training/GBRegression.py", line 15, in <module>
    items_df = pd.get_dummies(df, columns=['ObstacleXPos','ObstacleY', 'CoinXPos','CoinYPos'])
  File "C:\Users\Dean\Documents\University\Comp704Training\venv\lib\site-packages\pandas\core\reshape\reshape.py", line 915, in get_dummies
    dummy = _get_dummies_1d(
  File "C:\Users\Dean\Documents\University\Comp704Training\venv\lib\site-packages\pandas\core\reshape\reshape.py", line 1035, in _get_dummies_1d
    dummy_mat = np.eye(number_of_cols, dtype=dtype).take(codes, axis=0)
MemoryError: Unable to allocate 917. MiB for an array with shape (88763, 10838) and data type uint8
```

Too big?

Solutions

references

)))))

Regression, moving player dependent on obstacle and coin

Week 1 introduction week (Started 27th January)

At the beginning I thought of doing a 2d pinball machine as it would have been interesting to see how an ai would be able to abuse certain physics to get the best score during the game. Having scoring objects and objects that produce bounce back.

Week 2
I scrapped the pinball idea as I thought the physics part of it would take too long compared to my new idea of a 2D driving game.

Decided on 2D driving game
Did research to find a 2d car tutorial

One found used velocity and seemed more complex that needed

Thought about what type of info i want to collect
Will need to know the players position, the obstacles position, coins position

Planned game
Designing it, what will be needed
Draw plans[insert image]

Supervised or unsupervised learning

Week 3
Started developing game,
https://www.youtube.com/watch?v=FfWpgLFMI7w&t=180s

Try different preparations of your data using these heuristics and see what works best for your problem.

- **Linear Assumption**. Linear regression assumes that the relationship between your input and output is linear. It does not support anything else. This may be obvious, but it is good to remember when you have a lot of attributes. You may need to transform data to make the relationship linear (e.g. log transform for an exponential relationship).
- **Remove Noise**. Linear regression assumes that your input and output variables are not noisy. Consider using data cleaning operations that let you better expose and clarify the signal in your data. This is most important for the output variable and you want to remove outliers in the output variable (y) if possible.
- **Remove Collinearity**. Linear regression will over-fit your data when you have highly correlated input variables. Consider calculating pairwise correlations for your input data and removing the most correlated.
- **Gaussian Distributions**. Linear regression will make more reliable predictions if your input and output variables have a Gaussian distribution. You may get some benefit using transforms (e.g. log or BoxCox) on you variables to make their distribution more Gaussian looking.
- **Rescale Inputs**: Linear regression will often make more reliable predictions if you rescale input variables using standardization or normalization.

https://machinelearningmastery.com/linear-regression-for-machine-learning/
http://setosa.io/ev/ordinary-least-squares-regression/ O Square thing

First attempt at writing to an excel sheet. playpos, coinpos, obstacle pos
I implemented it into the coin collision logic so that the player will not only be alive, but scoring a point. Which is not in a obstacle that can kill the player..

Time	PlayerPos	ObstacleX	CoinXPos
1	375	506	547
1	555.9	506	547
1	587.1	182	587
1	546.3	525	546
1	451.8	372	452
1	469.8	372	470
1	440.1	539	440
1	341.7	308	341
1	423.9	222	424
1	483.9	279	484
1	286.2	374	286
1	537.9	616	538
1	431.7	193	432
1	483	187	483
1	577.8	187	578
1	429	603	429
1	453	342	453
1	433.2	219	433
1	528.9	457	554

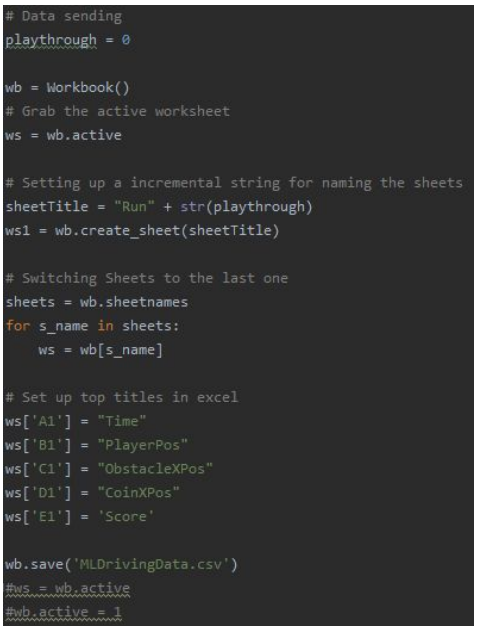
```
# Data sending
wb = Workbook()
# grab the active worksheet
ws = wb.active
ws['A1'] = "Time"
ws['B1'] = "PlayerPos"
ws['C1'] = "ObstacleXPos"
ws['D1'] = "CoinXPos"

ws.append([1,playerX,obstacleX, coinX])

wb.save('MLDrivingData.csv')
```

Will need to add more data such as score to see if its actullly doing well, maybe fix time aswell.

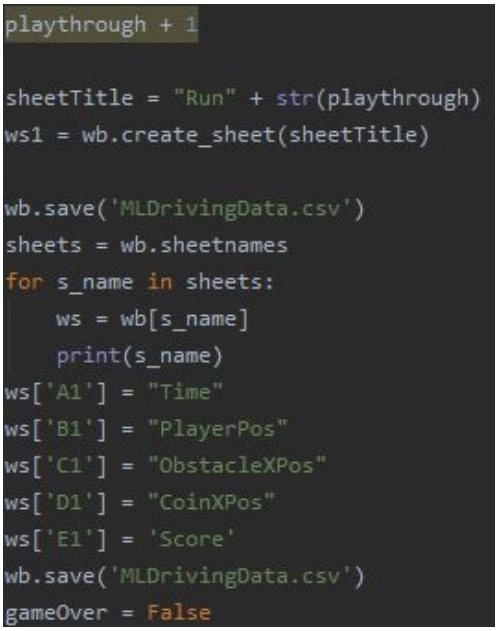
Discuss in great depth here for that sweeet sweet grade

Maybe need to get it everytime the coin or obstacle hits the bottom. As the AI can miss alot of coins before actually getting one which gives no data for a while

https://towardsdatascience.com/a-beginners-guide-to-linear-regression-in-python-with-scikit-learn-83a8f7ae2b4f



```
# Data sending
playthrough = 0

wb = Workbook()
# Grab the active worksheet
ws = wb.active

# Setting up a incremental string for naming the sheets
sheetTitle = "Run" + str(playthrough)
ws1 = wb.create_sheet(sheetTitle)

# Switching Sheets to the last one
sheets = wb.sheetnames
for s_name in sheets:
    ws = wb[s_name]

# Set up top titles in excel
ws['A1'] = "Time"
ws['B1'] = "PlayerPos"
ws['C1'] = "ObstacleXPos"
ws['D1'] = "CoinXPos"
ws['E1'] = 'Score'

wb.save('MLDrivingData.csv')
#ws = wb.active
#wb.active = 1
```



```
playthrough + 1

sheetTitle = "Run" + str(playthrough)
ws1 = wb.create_sheet(sheetTitle)

wb.save('MLDrivingData.csv')
sheets = wb.sheetnames
for s_name in sheets:
    ws = wb[s_name]
    print(s_name)
ws['A1'] = "Time"
ws['B1'] = "PlayerPos"
ws['C1'] = "ObstacleXPos"
ws['D1'] = "CoinXPos"
ws['E1'] = 'Score'
wb.save('MLDrivingData.csv')
gameOver = False
```

I wanted to save data in a new worksheet every time the player died from colliding with an obstacle. So I created a naming convention for each new sheet to be Run(playthrough); play through being the number of restarts.

I then let the program run for around 19 runs to collect some data to test out.

Creating a duplicate project and then removed all the symbolic AI out of it so that it is playable by a person.

While attempting to ready my data to implement into regression I came across a problem with pandas loading the file.
I thought that the way I saved the excel sheet was a CSV file, however after some help I found out that it was saving as an xlxs. This also meant that the way I stored my data into separate sheets would become nullified as CSV's do not support multiple sheet.

Gradient Boosting

https://medium.com/mlreview/gradient-boosting-from-scratch-1e317ae4587d
Gradient boosting is a machine learning technique for regression and classification problems, which produces a prediction model in the form of an ensemble of weak prediction models, typically decision trees. (Wikipedia definition)

So, the intuition behind gradient boosting algorithm is to repetitively leverage the patterns in residuals and strengthen a model with weak predictions and make it better.

https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.GradientBoostingRegressor.html
Has more info on each of the things, such as loss=huber

https://blog.paperspace.com/implementing-gradient-boosting-regression-python/
^ HAS BETTER EXPLANATIONS OF THE VARIABLES FOR MODELING

Boosting

The prediction accuracy of decision trees can be further improved by using Boosting algorithms.

The basic idea behind boosting is converting many weak learners to form a single strong learner. What do we mean by weak learners?

Weak learner is a learner that will always do better than chance, when it tries to label the data, no matter what the distribution over the training data is. Doing better than chance means we are always going to have an error rate which is less than 1/2. *This means that the learner algorithm is always going to learn something, and will not always be completely accurate i.e., it is weak and poor when it comes to learning the relationships between inputs and target.* It also means a rule formed using a single predictor/classifier is not powerful individually.

We start finding weak learners in the dataset by making some distributions and forming small decision trees from them. The size of the tree is tuned using number of splits it has. Often 1 works well, where each tree consists of a single split. Such trees are known as **Decision Stumps.**

https://towardsdatascience.com/boosting-the-accuracy-of-your-machine-learning-models-f878d6a2d185

So instead I used the last remaining sheet in excel and used it to try out regression.

	A	B	C	D	E
1	Time	PlayerPos	ObstacleX	CoinXPos	Score
2	1	225.9	389	226	0
3	1	180.9	517	181	2
4	1	440.7	265	441	4
5	1	439.2	265	439	5
6	1	325.2	561	325	7
7	1	325.2	284	325	9
8	1	290.1	343	290	11
9	1	264.3	343	264	12
10	1	297.9	179	298	14
11	1	207	591	207	16
12	1	351	526	351	19
13	1	369.9	526	370	20
14	1	460.2	555	460	22
15	1	582.9	479	583	24
16	1	446.4	534	437	26
17	1	482.7	225	483	28
18	1	367.8	225	368	29
19	1	534.9	408	535	31
20	1	432	287	432	34
21	1	315.3	267	315	36
22	1	225.9	504	226	38
23	1	271.2	619	271	40
24	1	276.3	610	276	42
25	1	417.3	531	393	45
26	1	253.8	413	244	47

```
# Testing Data

df = pd.read_csv("MLDrivingData.csv")


items_df = pd.get_dummies(df, columns=['ObstacleXPos', 'CoinXPos'])

# Create the X and Y arrays
X = items_df.values
y = df['PlayerPos'].values

# Split the data set in a traning set (70%) and test set (30%)
X_train, X_test, y_train, y_test = train_test_split(X,y, test_size=0.3, random_state=0)

# Fit regression model
model = ensemble.GradientBoostingRegressor(
    n_estimators=1000,
    learning_rate=0.1,
    max_depth=6,
    min_samples_leaf=9,
    max_features=0.1,
    loss='huber',
    random_state=0
)

print('doing training ...')
model.fit(X_train, y_train)
print('done training ...')

# Save trained model to pkl file
joblib.dump(model,'testTrainData.pkl')

# Find error rate
mse = mean_absolute_error(y_train, model.predict(X_train))
print("Training set mean absolute error: %.4f" % mse)
```

```
Training set mean absolute error: 97.9202
```

Because I would now need to go back and change my code it was a good point to rethink my data and collection methods. I found that my data lacked depth due to the fact it was only collecting data when the coins were collected. This meant that when the ai missed a few coins, there would be gaps in the data. To change this I have made it so when either the coin or obstacle hits the bottom of the screen and respawns it will also append to excel.

	A	B	C	D	E
1	Time	PlayerPos	ObstacleX	CoinXPos	Score
2	1	609.9	560	596	0
3	4	463.2	340	463	3
4	8	181.8	313	182	8
5	10	477	310	477	12
6	10	438.9	281	439	14
7	12	459.9	367	460	18
8	14	291.9	176	292	21
9	15	403.2	371	401	25
10	17	383.1	489	383	29
11	6	421.2	477	421	1
12	7	423.9	303	424	3
13	8	301.8	282	287	5
14	10	528	350	528	7
15	12	205.8	419	206	10
16	14	330.9	201	331	13
17	15	529.2	417	549	15
18	15	486.9	417	487	16
19	16	381	245	381	18
20	17	479.1	278	479	20
21	18	495	548	495	22
22	19	429.9	509	430	24
23	20	255.6	509	251	25
24	22	494.1	504	507	28
25	24	444	573	444	32
26	25	498	569	498	34

	A	B	C	D	E
1	Time	PlayerPos	ObstacleX	CoinXPos	Score
2	1	404.7	494	405	0
3	3	444	622	574	2
4	4	573.9	622	574	2
5	5	402.9	546	403	4
6	6	403.2	546	403	4
7	6	249.9	546	250	5
8	7	231.6	335	224	7
9	7	224.1	335	224	7
10	7	271.8	390	538	9
11	8	343.8	390	446	9
12	8	344.1	454	446	10
13	9	422.4	454	446	10
14	9	360.3	229	360	12
15	9	360.3	229	360	12
16	10	273	440	240	14
17	10	256.5	440	240	14
18	11	396.9	390	576	16
19	11	404.4	390	242	16
20	11	269.7	390	242	16
21	11	288.3	280	517	18
22	12	420.9	280	577	18
23	12	448.2	360	577	19
24	13	552.3	360	577	19
25	13	497.4	517	182	21

Before After

The data already looks fuller and you can tell by the time that it is already being collected more often.

Another problem I faced was how slow the game actually was, It is alright for human players and their reaction times however it did not matter much for the AI. I could not simply increase the speed of everything I would also need to change the players left and right movespeed.

```
# Player
playerImage = pygame.image.load('playercar.png')
playerX = 375
playerY = 750
playerXSpeed = 0
leftSpeed = -0.3
rightSpeed = 0.3

# Obstacle
obstacleImage = pygame.image.load('cone.png')
obstacleX = random.randint(200, 600)
obstacleY = 0
obstacleYSpeed = 0.4

# Coins
coinImage = pygame.image.load('coin.png')
coinX = random.randint(200, 600)
coinY = 0
coinSpeed = 0.5
```

To improve this I will times each of the speed variables by a set number. This number now being 2.

Tested for 30 minutes, usewd data again

```
Training set mean absolute error: 2.3364
```

```
Test set mean absolute error: 2.9506
```

5. Evaluate performance, as in is this good or bad for the errors?

```
Traceback (most recent call last):
  File "C:/Users/Dean/Documents/University/Comp704MLTesting/main.py", line 77, in <module>
    pickleModel = pickle.load(open("testTrainData.pkl", 'rb'))
_pickle.UnpicklingError: invalid load key, '4'.
```

```
pickleModel = pickle.load(open("testTrainData.pkl", 'rb'))
```

Had this error that did allow me to load the PKL file.

Saves as CSV instead of Excl

https://machinelearningmastery.com/save-load-machine-learning-models-python-scikit-learn/

```
# load the model from disk
loadedModel = pickle.load(open('testTrainData.pkl', 'rb'))
result = loadedModel.score(X_test, y_test)
print(result)
```

```
0.9985797042915066
```

Load the saved model and evaluating it provides an estimate of accuracy of the model on unseen data.

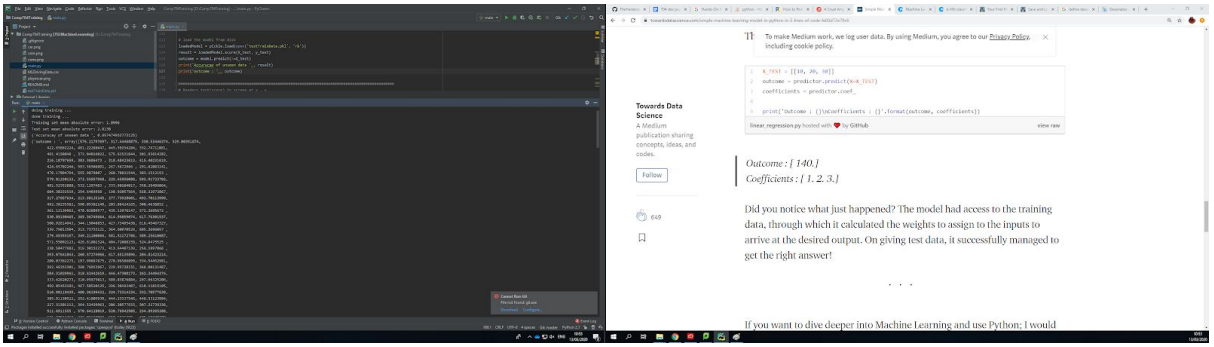
Changed out how i was dumping the training data so that it was just pickle doing it.
Load the saved model and evaluating it provides an estimate of accuracy of the model on unseen data.

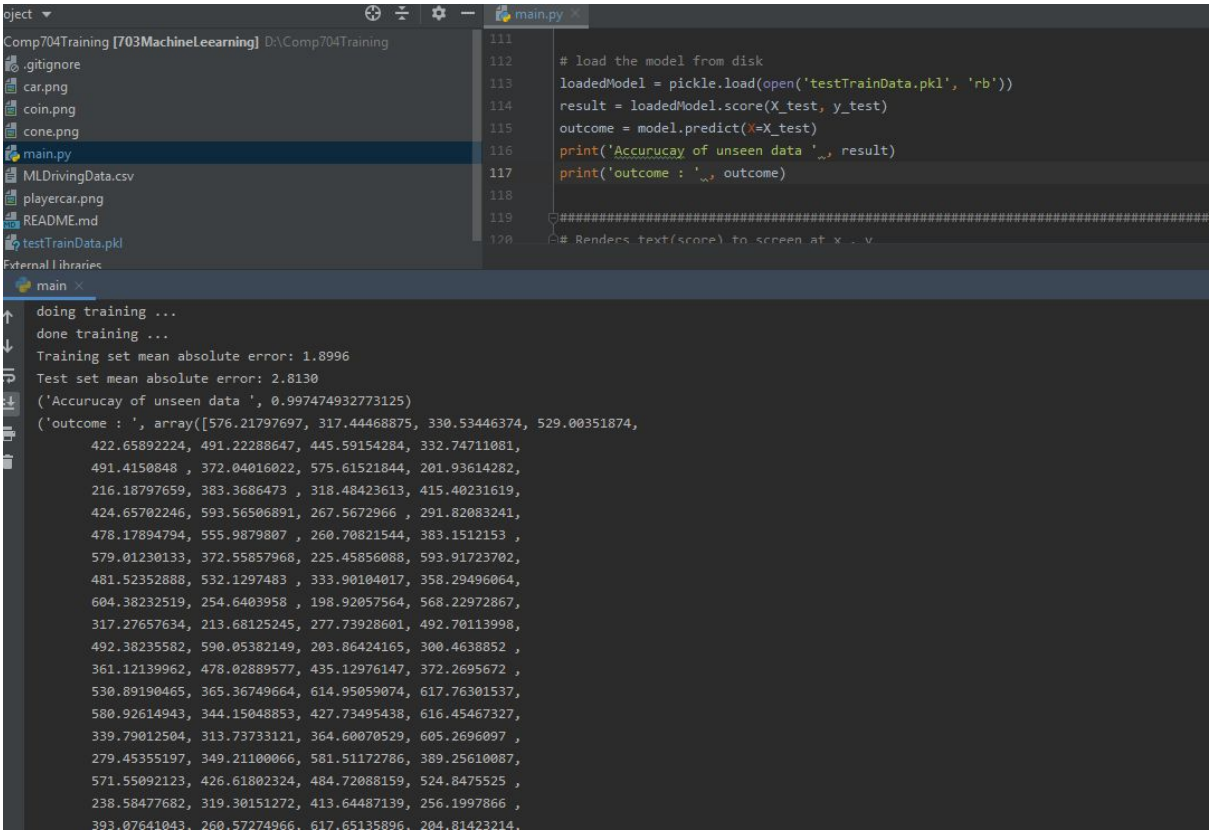
```python
# Fit regression model
model = ensemble.GradientBoostingRegressor(
    n_estimators=2000,
    learning_rate=0.1,
    max_depth=16,
    min_samples_leaf=9,
    max_features=0.1,
    loss='huber',
    random_state=0
)
```

```
Training set mean absolute error: 1.8996
Test set mean absolute error: 2.8130
0.997474932773125
```

https://blog.paperspace.com/implementing-gradient-boosting-regression-python/

Trying to ploit graph





```
doing training ...
done training ...
Training set mean absolute error: 1.8996
Test set mean absolute error: 2.8130
('Accurucay of unseen data ', 0.997474932773125)
('outcome : ', array([576.21797697, 317.44468875, 330.53446374, 529.00351874,
       422.65892224, 491.22288647, 445.59154284, 332.74711081,
       491.4150848 , 372.04016022, 575.61521844, 201.93614282,
       216.18797659, 383.3686473 , 318.48423613, 415.40231619,
       424.65702246, 593.56506891, 267.5672966 , 291.82083241,
       478.17894794, 555.9879807 , 260.70821544, 383.1512153 ,
       579.01230133, 372.55857968, 225.45856088, 593.91723702,
       481.52352888, 532.1297483 , 333.90104017, 358.29496064,
       604.38232519, 254.6403958 , 198.92057564, 568.22972867,
       317.27657634, 213.68125245, 277.73928601, 492.70113998,
       492.38235582, 590.05382149, 203.86424165, 300.4638852 ,
       361.12139962, 478.02889577, 435.12976147, 372.2695672 ,
       530.89190465, 365.36749664, 614.95059074, 617.76301537,
       580.92614943, 344.15048853, 427.73495438, 616.45467327,
       339.79012504, 313.73733121, 364.60070529, 605.2696097 ,
       279.45355197, 349.21100066, 581.51172786, 389.25610087,
       571.55092123, 426.61802324, 484.72088159, 524.8475525 ,
       238.58477682, 319.30151272, 413.64487139, 256.1997866 ,
       393.07641043, 260.57274966, 617.65135896, 204.81423214,
```

Learn about pickle files

Put notes on it

https://www.datacamp.com/community/tutorials/pickle-python-tutorial

Im stuck on how to use the generated pickel file to do anything.

I think I have to use the pickle file to create some form of prediction for the player.

Comparing the

```
Training set mean absolute error: 1.0094
Test set mean absolute error: 1.2668
Accurucay of unseen data  0.9990619953856031
```

16000 data training pkl

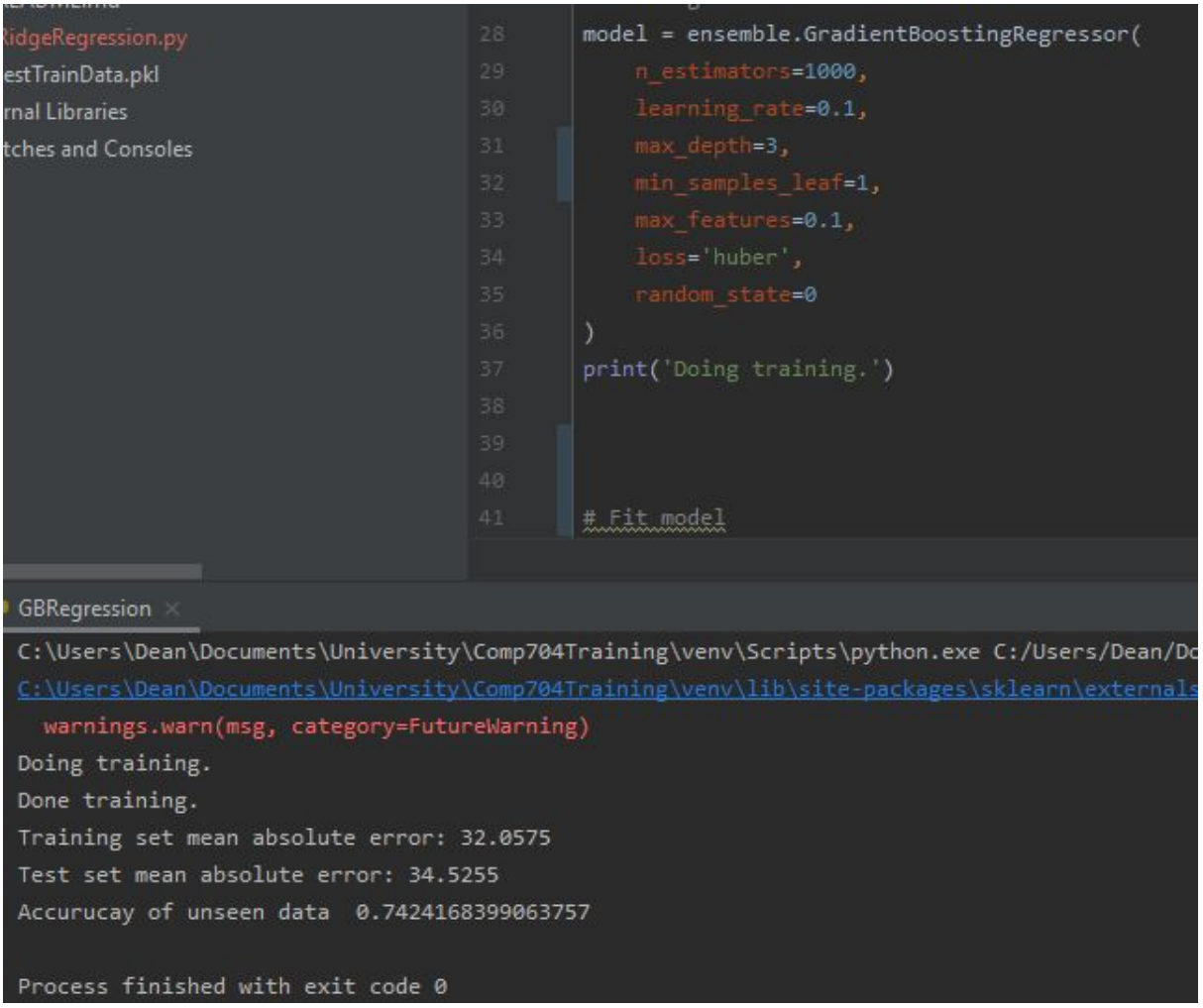
```
ValueError: Number of features of the model must match the input. Model n_features is 903 and input n_features is 2
```

Tried doing these:
https://towardsdatascience.com/simple-machine-learning-model-in-python-in-5-lines-of-code-fe03d72e78c6

https://towardsdatascience.com/linear-regression-using-least-squares-a4c3456e8570

Memory Error is exactly what it means, you have run out of memory in your RAM for your code to execute.
When this error occurs it is likely because you have loaded the entire data into memory. For large datasets you will want to use batch processing.
Please try using a smaller array that your system can handle and run the code.

The problem was with the get dummies.
IT is giving catergoircal vartiables instead of conitnours variables.
So every obstacle X would have 0.1 is a variables



50k lines of data
After increasing the n_esitmators the player has a more consistent time getting the coins. Where as before with n_estimators = 40 the layer would be moving left and right continuously as it had a bigger error range.
88k lines of data
Slightly increased error ranges

```
Training set mean absolute error: 33.1095
Test set mean absolute error: 34.7712
Accurucay of unseen data  0.7379881963345105
```

Although it is hard to tell I feel like the player is dodging the obstacle at some points even if it is by a few millimeters

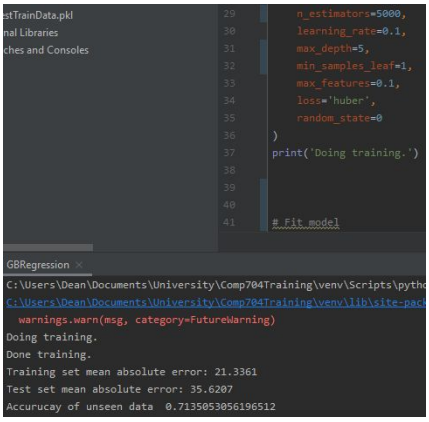
N_estimators = 5000
Still has the problem with obstacles however there was one point where it performed like the symbolic AI, aligning itself to the right of the obstacle before going and getting the coin

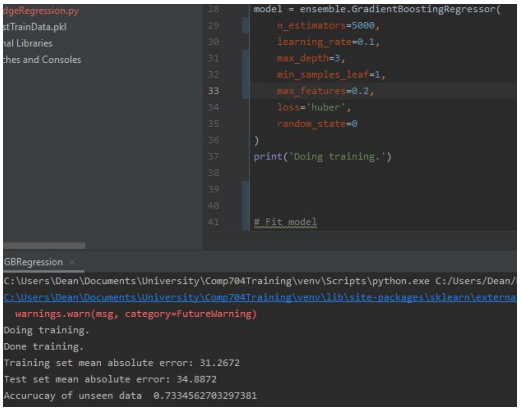
Seems to be fine if the obstacle and player are the same side as the coin, the player will move to the side of the obstacle and then move to the coin. But if it needs to cross over it causes problems.

```
Training set mean absolute error: 22.3042
Test set mean absolute error: 35.2035
Accurucay of unseen data  0.7138074425702461
```

0.4 test

Can survive up to around 40 score.





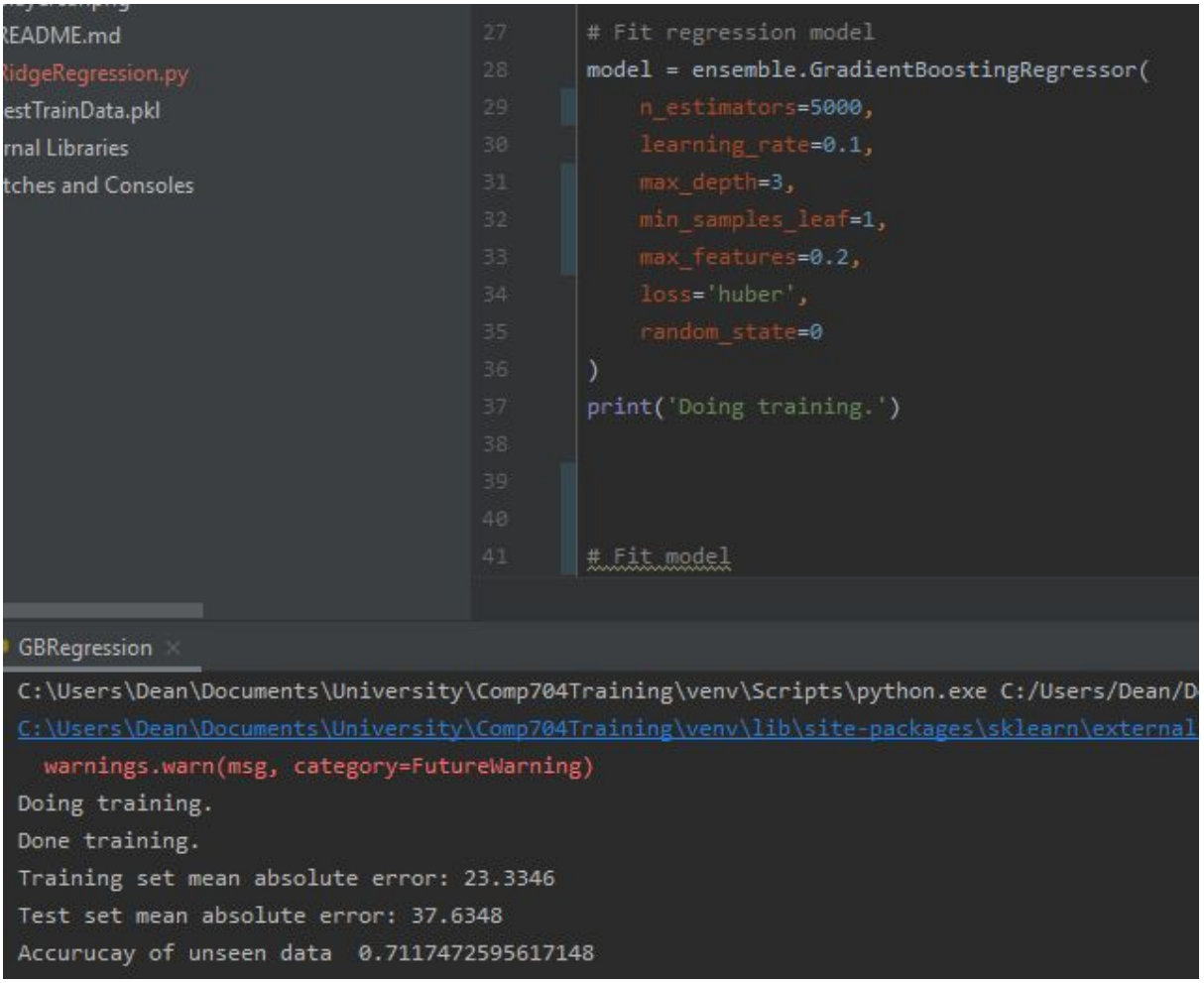
Before I go any further into the algorithm I want to see if adding a variable in for when the obstacle is below a certain point. If this will help the player identify that it needs to be more careful during this time.

Had some things wrong in the recording of my data:

```
# Extra data for obstacle
if (obstacleY >= 400) and (obstacleY <= 400.7):
    ws.append([playerX, obstacleX, obstacleY, coinX, coinY, nearBool])
if (obstacleY >= 500) and (obstacleY <= 500.7):
    ws.append([playerX, obstacleX, obstacleY, coinX, coinY, nearBool])
if (obstacleY >= 600) and (obstacleY <= 600.7):
    ws.append([playerX, obstacleX, obstacleY, coinX, coinY, nearBool])
if (obstacleY >= 700) and (obstacleY <= 700.7):
    ws.append([playerX, obstacleX, obstacleY, coinX, coinY, nearBool])
```

I had it so it was top half of the screen recorded not the bottom half, so i also added in an extra one so that it will gather more data each time the obstacle gets 100 Y closer from 400Y.

Some of my appends were the wrong way around as well such as the coin being recorded after its respawned.

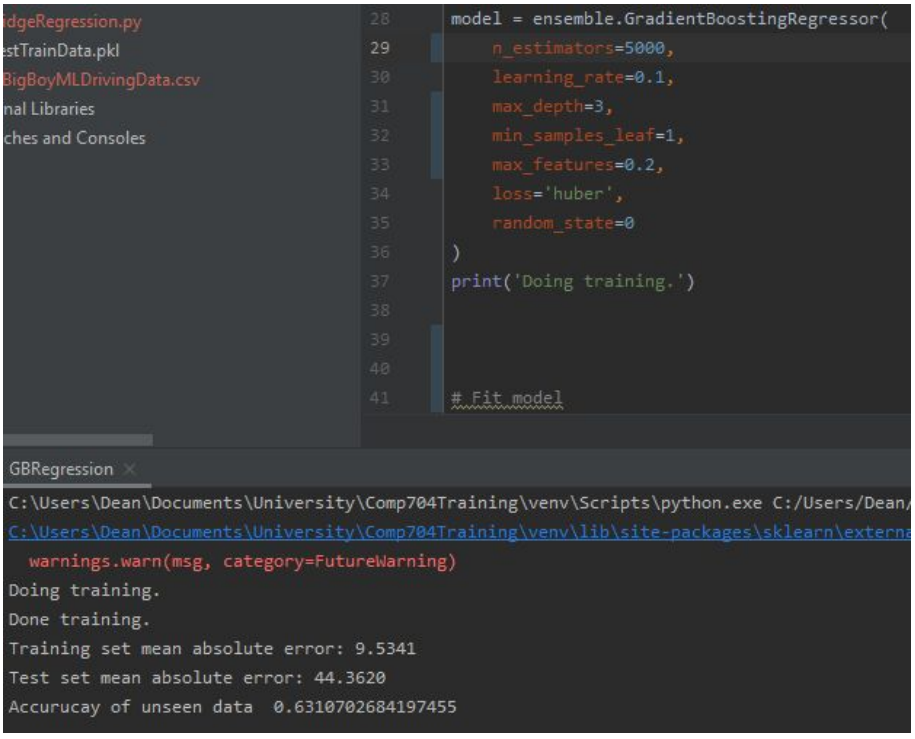


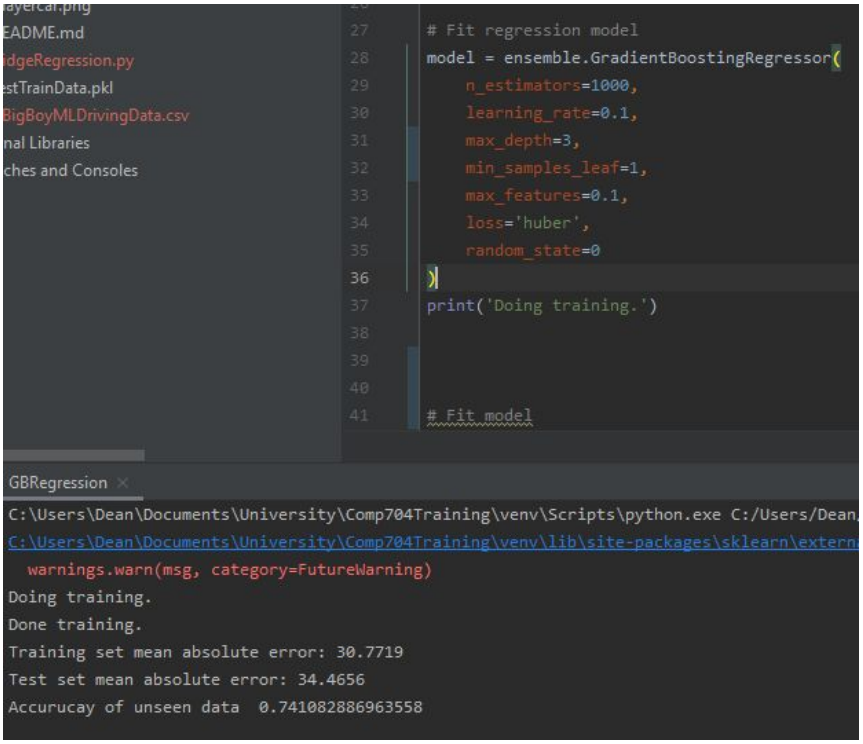
14k lines of data
Slight imrpovement. Player sometimes moves around the obstacle.

Does my symbloic ai make the the positioning too close?
If it is training it to be right next to the obstacle and decreases the distance every time, even a 10 error range could cause it to fail

```
# Decrease error size
x = randint(1, 100)
if x >=75:
    errorSize = errorSize - errorRate
```

Just to be safe I added in this piece of code so now thee player will play better for longer and theres a bit more variation in the game play.





New data at 23k

Actually tries to avoid obastacles although not very good

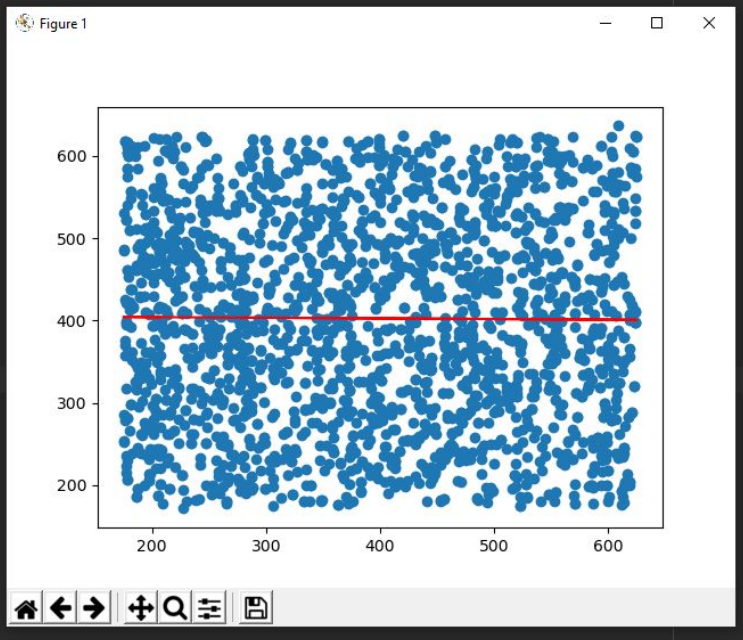
```
README.md
BigBoyMLDrivingData.csv
nal Libraries
tches and Consoles
```

```
# Fit regression model
model = ensemble.GradientBoostingRegressor(
    n_estimators=5000,
    learning_rate=0.1,
    max_depth=3,
    min_samples_leaf=1,
    max_features=0.1,
    loss='huber',
    random_state=0
)
print('Doing training.')



# Fit model
```

```
GBRegression
C:\Users\Dean\Documents\University\Comp704Training\venv\Scripts\python.exe C:/Users/Dean/D
C:\Users\Dean\Documents\University\Comp704Training\venv\lib\site-packages\sklearn\external
  warnings.warn(msg, category=FutureWarning)
Doing training.
Done training.
Training set mean absolute error: 30.3976
Test set mean absolute error: 34.0676
Accurucay of unseen data  0.7367940334193437
```

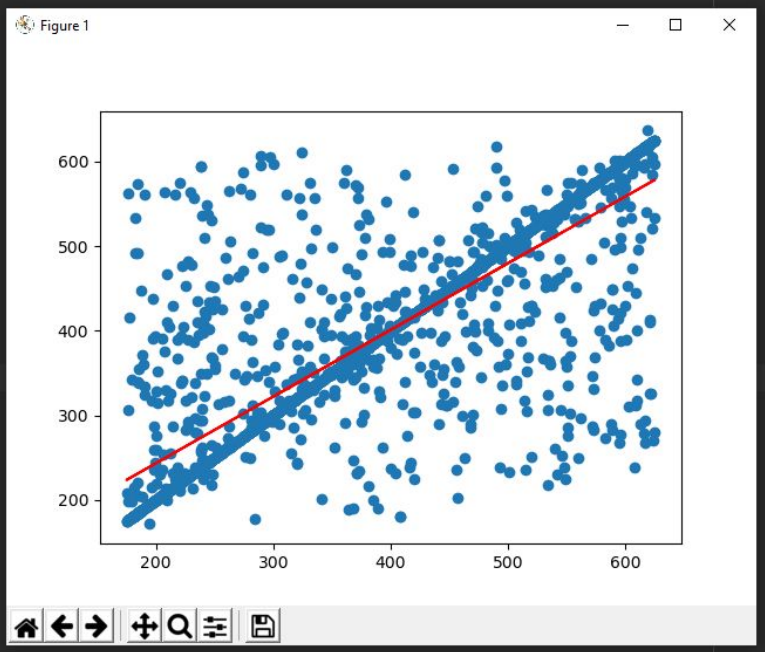
Linear regression plot graphs

```
from sklearn import linear_model
from sklearn.linear_model import LinearRegression
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt


data = pd.read_csv("MLDrivingData.csv")
X = data.iloc[:,0].values.reshape(-1,1)
Y = data.iloc[:,1].values.reshape(-1,1)
lr = LinearRegression()
lr.fit(X,Y)
Y_pred = lr.predict(X)

plt.scatter(X,Y)
plt.plot(X,Y_pred, color='red')
plt.show()
```



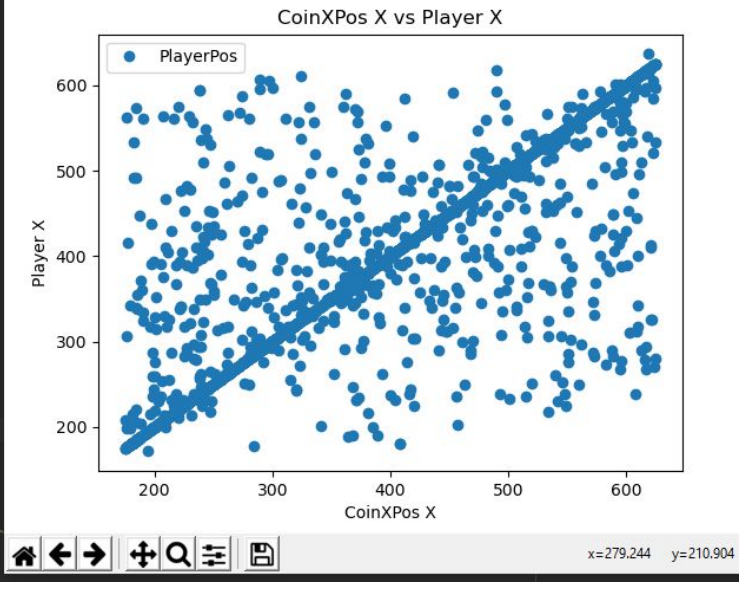
This one was with obstacle pos as input and player pos as output

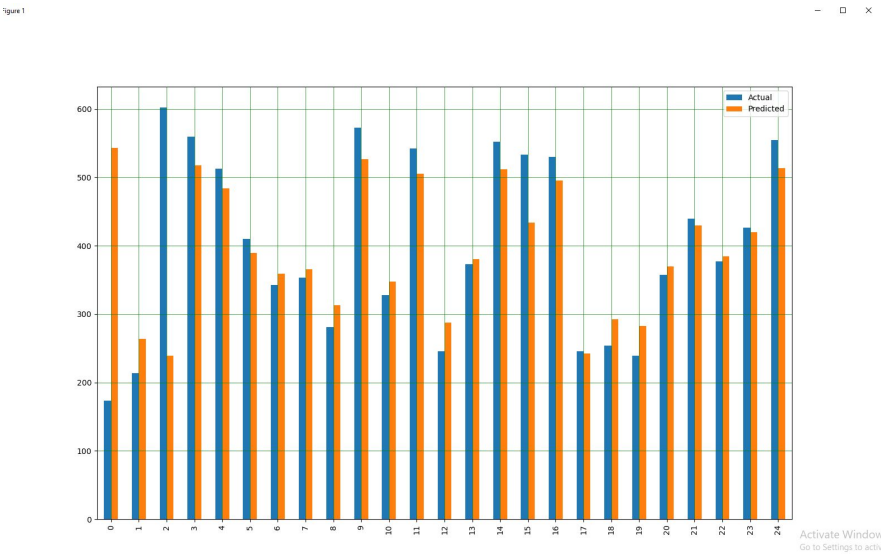
```python
from sklearn import linear_model
from sklearn.linear_model import LinearRegression
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt


data = pd.read_csv("MLDrivingData.csv")
X = data.iloc[:,0].values.reshape(-1,1)
Y = data.iloc[:,1].values.reshape(-1,1)
lr = LinearRegression()
lr.fit(X,Y)
Y_pred = lr.predict(X)

plt.scatter(X,Y)
plt.plot(X,Y_pred, color='red')
plt.show()
```

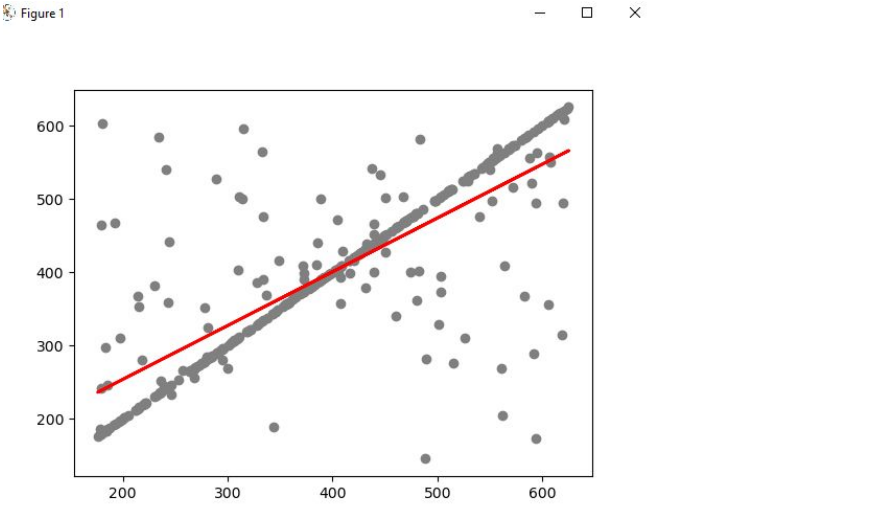


This one was with coin pos as input and player pos as output
https://towardsdatascience.com/linear-regression-in-6-lines-of-python-5e1d0cd05b8d

```python
from sklearn.linear_model import LinearRegression
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

#now got 3 columns in excel sheet
from sklearn.model_selection import train_test_split

data = pd.read_csv("MLDrivingData.csv")

data.shape

print(data.describe())

X = data['CoinXPos']
y = data['PlayerPos']

data.plot(x='CoinXPos', y='PlayerPos', style='o')
plt.title('CoinXPos X vs Player X')
plt.xlabel('CoinXPos X')
plt.ylabel('Player X')
plt.show()

#X_train, X_test, y_train, y_test = train_test_split(X,
```








```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=0)

regressor = LinearRegression()
regressor.fit(X_train, y_train) #training the algorithm

#To retrieve the intercept:
print(regressor.intercept_)
#For retrieving the slope:
print(regressor.coef_)

y_pred = regressor.predict(X_test)

df = pd.DataFrame({'Actual': y_test.flatten(), 'Predicted': y_pred.flatten()})
print(df)

print('Mean Absolute Error:', metrics.mean_absolute_error(y_test, y_pred))
print('Mean Squared Error:', metrics.mean_squared_error(y_test, y_pred))
print('Root Mean Squared Error:', np.sqrt(metrics.mean_squared_error(y_test, y_pred)))
```

```
Mean Absolute Error: 44.98215746754988
Mean Squared Error: 5261.369161892627
Root Mean Squared Error: 72.53529597301322
```

The final step is to evaluate the performance of the algorithm. This step is particularly important to compare how well different algorithms perform on a particular dataset. For regression algorithms, three evaluation metrics are commonly used:

1. **Mean Absolute Error** (MAE) is the mean of the absolute value of the errors. It is calculated as:

$$MAE = \frac{1}{n}\sum_{j=1}^{n}|y_j - y_j|$$

Mean Absolute Error

2. **Mean Squared Error** (MSE) is the mean of the squared errors and is calculated as:

$$MSE = \frac{1}{N}\sum_{i}^{n}(Y_i - y_i)^2$$

Mean Squared Error

3. **Root Mean Squared Error** (RMSE) is the square root of the mean of the squared errors:

$$\mathrm{RMSE} = \sqrt{\frac{1}{n}\sum_{j=1}^{n}(y_j - \hat{y}_j)^2}$$

Root Mean Squared Error

https://towardsdatascience.com/a-beginners-guide-to-linear-regression-in-python-with-scikit-learn-83a8f7ae2b4f
https://towardsdatascience.com/a-beginners-guide-to-linear-regression-in-python-with-scikit-learn-83a8f7ae2b4f

```
infile = open("LRtestTrainData.pkl", 'rb')
pickleModel = pickle.load(infile)
infile.close()
```

```
while running:

    X = [[coinX]]
    ynew = pickleModel.predict(X)

    if playerX > ynew:
        playerXSpeed = leftSpeed

    elif playerX < ynew:
        playerXSpeed = rightSpeed
```

How did it do?>
It moves towards the coin most of the time. It sometimes just stops before it gets to it though.
Has no perception of obstacles so crashes and loses alot.
Using more data would help its accuracy.

Was 2300 lines of data
Now 8203

```
Mean Absolute Error: 44.45610252890953
Mean Squared Error: 4943.013770880863
Root Mean Squared Error: 70.30656989841606
0.6623320381713609
```

Copy pasted excel data to test
16k lines of data

```
Mean Absolute Error: 43.455636651656114
Mean Squared Error: 4428.526697054458
Root Mean Squared Error: 66.54717647695098
0.7037951655237005
```

Talk about how much you think you need

Multi linear regression

```
data = pd.read_csv("MLDrivingData.csv")
#df = DataFrame(data, columns=['PlayerX', 'ObstacleXPos', 'CoinXPos'])

X = data[['ObstacleXPos', 'CoinXPos']] #should be able to add more variables to
y = data['PlayerPos']

lr = linear_model.LinearRegression()
lr.fit(X,y)

print('Intercept: ', lr.intercept_)
print('Coefficients: ', lr.coef_)

#predictioon
print(lr.predict([[100,650]]))
```

```
Intercept:  88.70237983883464
Coefficients:  [0.00846393 0.77047664]
[590.35858581]
```

Moves with poor accuracy, but still does not dodge the obstacles.
Is this because of the lack of data again?
Theres not much data for it if when the coin and obstacle are on the same X

Recoreded 26000 lines of data and the player still would easily crash into the obstacles

```
Intercept:  93.17337927134764
Coefficients:  [0.00417657 0.76249124]
[476.40704692]
0.6577013141710023
```

Maybe i need Y coordinates to take into account where the obstacle is compared to the player

Have to re record data as i thought I would not need the Y and now i do

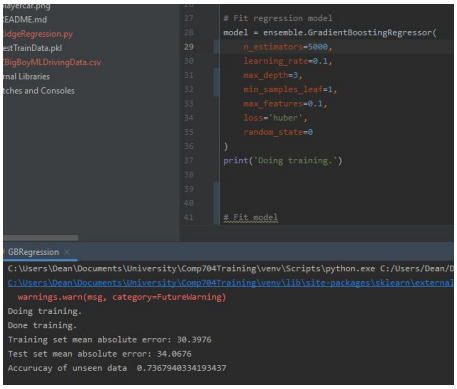
```
Intercept:  91.22093323068594
Coefficients:  [ 0.0134497   0.00625221  0.77237415 -0.01245168]
[442.99237581]
0.676382865688584
```
temp

```
# Extra data for obstacle
if (obstacleY >= 400) and (obstacleY <= 400.7):
    ws.append([playerX, obstacleX, obstacleY, coinX, coinY])
if (obstacleY >= 500) and (obstacleY <= 500.7):
    ws.append([playerX, obstacleX, obstacleY, coinX, coinY])
```

I wanted more data around the obstacle to see if this helps with it knowing not to collide with them.
This means that it will grab data when the obstacle is very close to the bottom of the screen

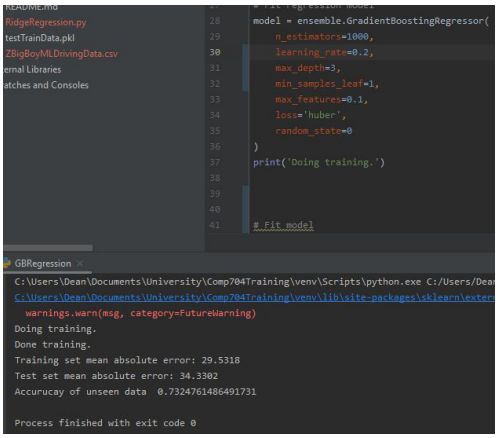
2 models?
One for obstacle and player and,
One for coin and player and run them both?

Gradient boosting regression pictures of changin around variables



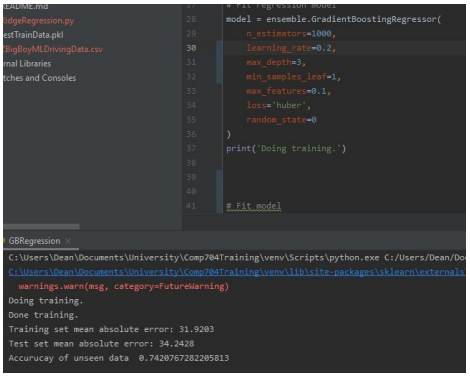
```
# Fit regression model
model = ensemble.GradientBoostingRegressor(
    n_estimators=5000,
    learning_rate=0.1,
    max_depth=3,
    min_samples_leaf=1,
    max_features=0.1,
    loss='huber',
    random_state=0
)
print('Doing training.')


# Fit model
```

```
GBRegression
C:\Users\Dean\Documents\University\Comp704Training\venv\Scripts\python.exe C:/Users/Dean/D
C:\Users\Dean\Documents\University\Comp704Training\venv\lib\site-packages\sklearn\externa
  warnings.warn(msg, category=FutureWarning)
Doing training.
Done training.
Training set mean absolute error: 30.3976
Test set mean absolute error: 34.0676
Accuracy of unseen data  0.7367940334193437
```



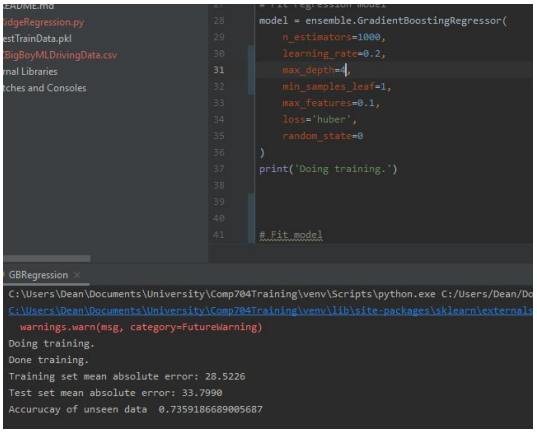
```
model = ensemble.GradientBoostingRegressor(
    n_estimators=1000,
    learning_rate=0.2,
    max_depth=3,
    min_samples_leaf=1,
    max_features=0.1,
    loss='huber',
    random_state=0
)
print('Doing training.')


# Fit model
```

```
GBRegression
C:\Users\Dean\Documents\University\Comp704Training\venv\Scripts\python.exe C:/Users/Dean
C:\Users\Dean\Documents\University\Comp704Training\venv\lib\site-packages\sklearn\extern
  warnings.warn(msg, category=FutureWarning)
Doing training.
Done training.
Training set mean absolute error: 29.5318
Test set mean absolute error: 34.3302
Accuracy of unseen data  0.7324761486491731

Process finished with exit code 0
```



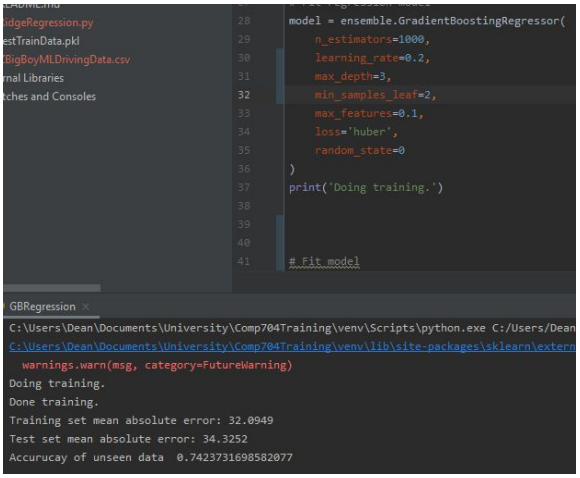
```
model = ensemble.GradientBoostingRegressor(
    n_estimators=1000,
    learning_rate=0.2,
    max_depth=3,
    min_samples_leaf=1,
    max_features=0.1,
    loss='huber',
    random_state=0
)
print('Doing training.')


# Fit model
```

```
GBRegression
C:\Users\Dean\Documents\University\Comp704Training\venv\Scripts\python.exe C:/Users/Dean/Do
C:\Users\Dean\Documents\University\Comp704Training\venv\lib\site-packages\sklearn\externals
  warnings.warn(msg, category=FutureWarning)
Doing training.
Done training.
Training set mean absolute error: 31.9203
Test set mean absolute error: 34.2428
Accuracy of unseen data  0.7420767282205813
```



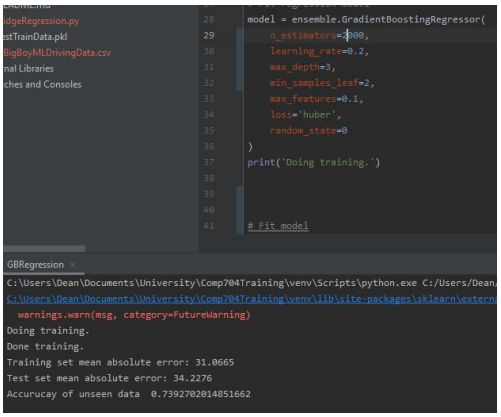
```
model = ensemble.GradientBoostingRegressor(
    n_estimators=1000,
    learning_rate=0.2,
    max_depth=4,
    min_samples_leaf=1,
    max_features=0.1,
    loss='huber',
    random_state=0
)
print('Doing training.')


# Fit model
```

```
GBRegression
C:\Users\Dean\Documents\University\Comp704Training\venv\Scripts\python.exe C:/Users/Dean/Do
C:\Users\Dean\Documents\University\Comp704Training\venv\lib\site-packages\sklearn\externals
  warnings.warn(msg, category=FutureWarning)
Doing training.
Done training.
Training set mean absolute error: 28.5226
Test set mean absolute error: 33.7990
Accuracy of unseen data  0.7359186689005687
```



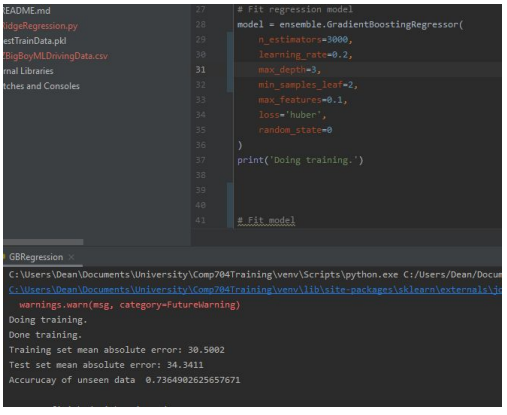
```
model = ensemble.GradientBoostingRegressor(
    n_estimators=1000,
    learning_rate=0.2,
    max_depth=3,
    min_samples_leaf=2,
    max_features=0.1,
    loss='huber',
    random_state=0
)
print('Doing training.')


# Fit model
```

```
GBRegression
C:\Users\Dean\Documents\University\Comp704Training\venv\Scripts\python.exe C:/Users/Dean
C:\Users\Dean\Documents\University\Comp704Training\venv\lib\site-packages\sklearn\extern
  warnings.warn(msg, category=FutureWarning)
Doing training.
Done training.
Training set mean absolute error: 32.0949
Test set mean absolute error: 34.3252
Accuracy of unseen data  0.7423731698582077
```



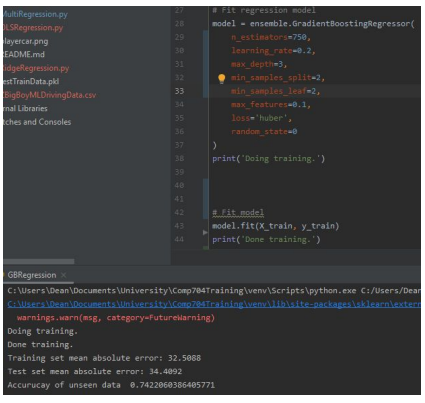
```
model = ensemble.GradientBoostingRegressor(
    n_estimators=1000,
    learning_rate=0.2,
    max_depth=3,
    min_samples_leaf=2,
    max_features=0.1,
    loss='huber',
    random_state=0
)
print('Doing training.')


# Fit model
```

```
GBRegression
C:\Users\Dean\Documents\University\Comp704Training\venv\Scripts\python.exe C:/Users/Dean/
C:\Users\Dean\Documents\University\Comp704Training\venv\lib\site-packages\sklearn\extern
  warnings.warn(msg, category=FutureWarning)
Doing training.
Done training.
Training set mean absolute error: 31.0665
Test set mean absolute error: 34.2276
Accuracy of unseen data  0.7392702014851662
```



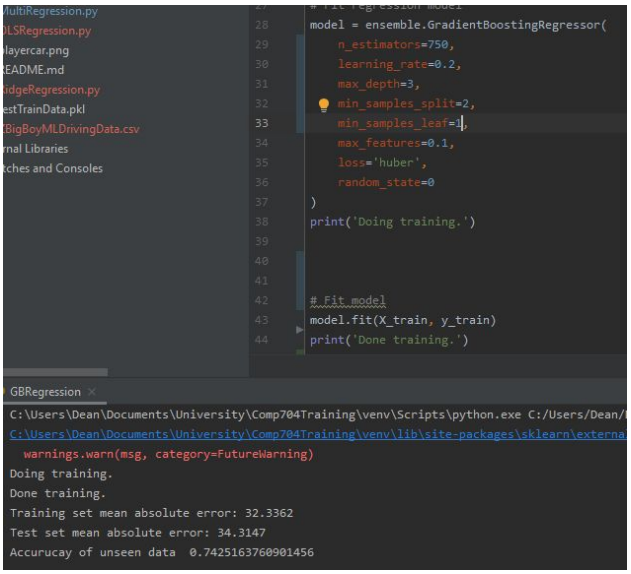
```
# Fit regression model
model = ensemble.GradientBoostingRegressor(
    n_estimators=3000,
    learning_rate=0.2,
    max_depth=3,
    min_samples_leaf=2,
    max_features=0.1,
    loss='huber',
    random_state=0
)
print('Doing training.')


# Fit model
```

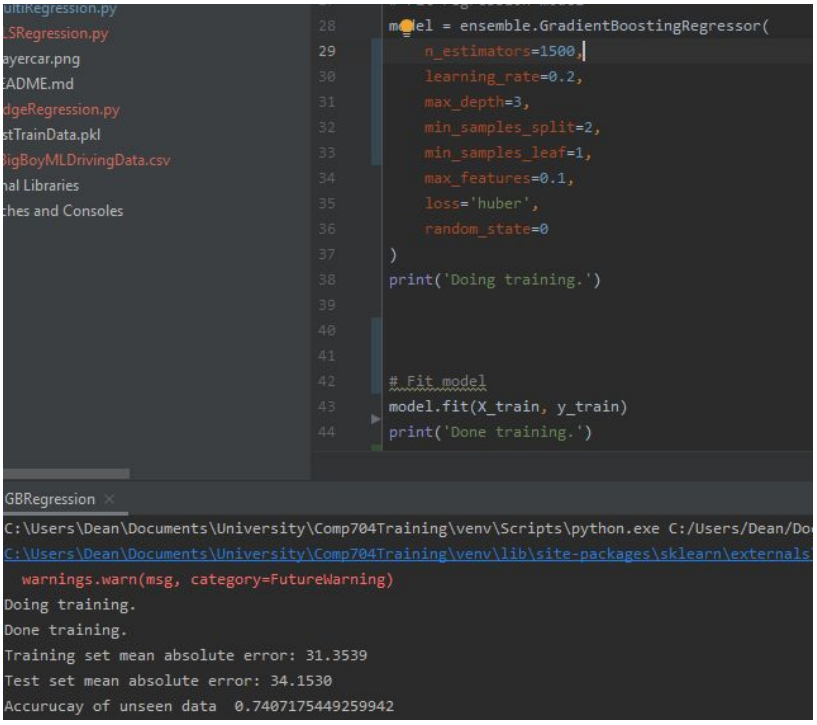
```
GBRegression
C:\Users\Dean\Documents\University\Comp704Training\venv\Scripts\python.exe C:/Users/Dean/Docu
C:\Users\Dean\Documents\University\Comp704Training\venv\lib\site-packages\sklearn\externals\jo
  warnings.warn(msg, category=FutureWarning)
Doing training.
Done training.
Training set mean absolute error: 30.5002
Test set mean absolute error: 34.3411
Accuracy of unseen data  0.7364902625657671
```

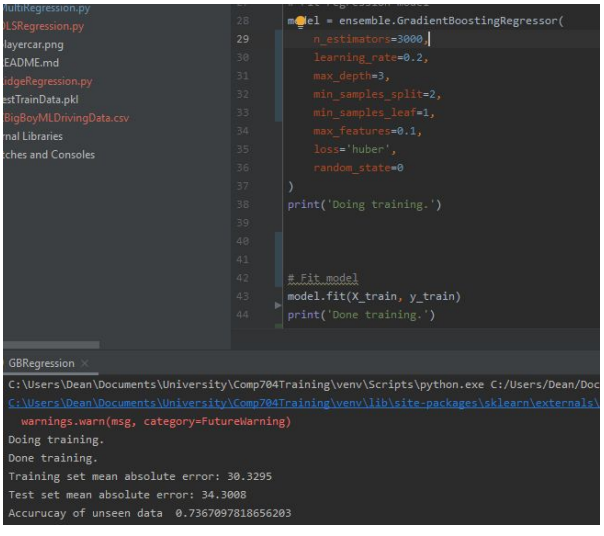


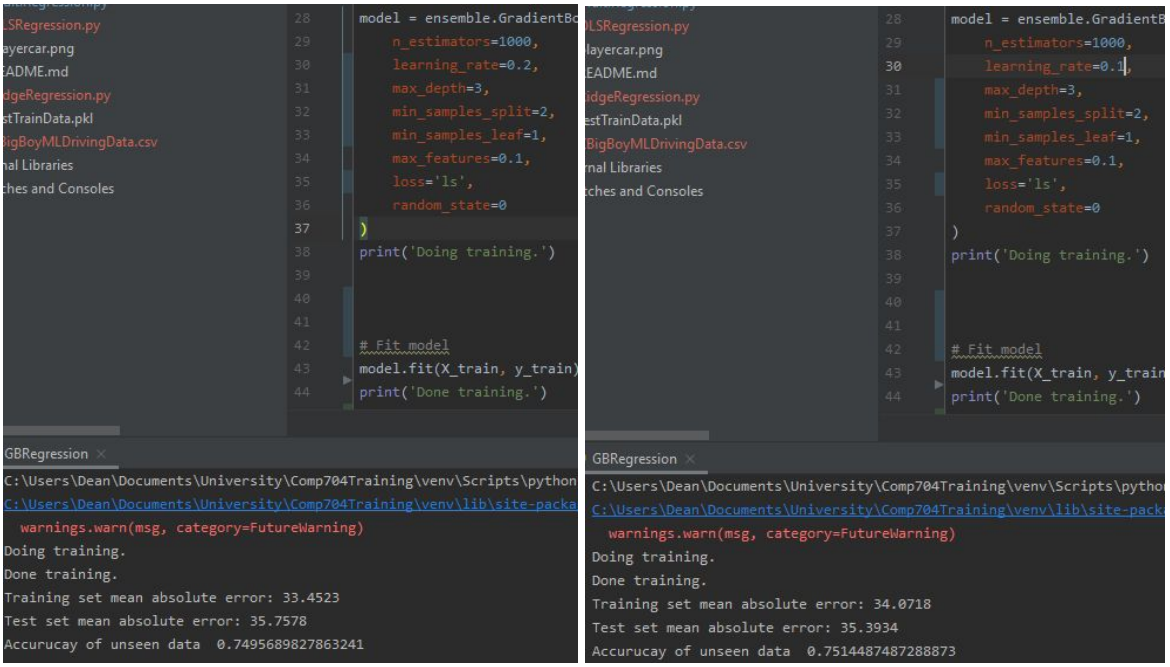
```
# Fit regression model
model = ensemble.GradientBoostingRegressor(
    n_estimators=750,
    learning_rate=0.2,
    max_depth=3,
    min_samples_split=2,
    min_samples_leaf=2,
    max_features=0.1,
    loss='huber',
    random_state=0
)
print('Doing training.')


# Fit model
model.fit(X_train, y_train)
print('Done training.')
```

```
GBRegression
C:\Users\Dean\Documents\University\Comp704Training\venv\Scripts\python.exe C:/Users/Dean
C:\Users\Dean\Documents\University\Comp704Training\venv\lib\site-packages\sklearn\extern
  warnings.warn(msg, category=FutureWarning)
Doing training.
Done training.
Training set mean absolute error: 32.5088
Test set mean absolute error: 34.4092
Accuracy of unseen data  0.7422060306405771
```

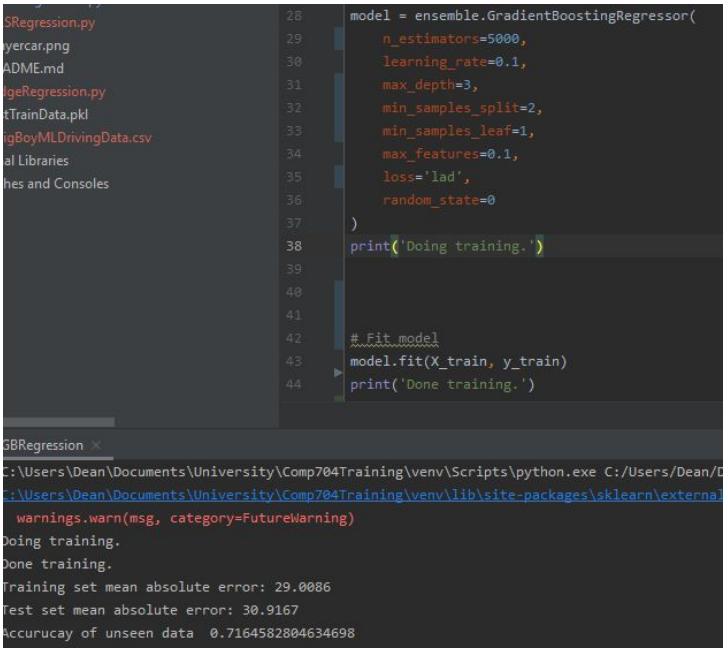

MultiRegression.py
LSRegression.py
layercar.png
README.md
idgeRegression.py
estTrainData.pkl
BigBoyMLDrivingData.csv
rnal Libraries
tches and Consoles
model = ensemble.GradientBoostingRegressor(
n_estimators=750,
learning_rate=0.2,
max_depth=3,
min_samples_split=2,
min_samples_leaf=1,
max_features=0.1,
loss='huber',
random_state=0
)
print('Doing training.')
# Fit model
model.fit(X_train, y_train)
print('Done training.')
GBRegression
C:\Users\Dean\Documents\University\Comp704Training\venv\Scripts\python.exe C:/Users/Dean/
C:\Users\Dean\Documents\University\Comp704Training\venv\lib\site-packages\sklearn\externa
warnings.warn(msg, category=FutureWarning)
Doing training.
Done training.
Training set mean absolute error: 32.3362
Test set mean absolute error: 34.3147
Accurucay of unseen data 0.7425163760901456


model = ensemble.GradientBoostingRegressor(
n_estimators=1500,
learning_rate=0.2,
max_depth=3,
min_samples_split=2,
min_samples_leaf=1,
max_features=0.1,
loss='huber',
random_state=0
)
print('Doing training.')
# Fit model
model.fit(X_train, y_train)
print('Done training.')
GBRegression
C:\Users\Dean\Documents\University\Comp704Training\venv\Scripts\python.exe C:/Users/Dean/Do
C:\Users\Dean\Documents\University\Comp704Training\venv\lib\site-packages\sklearn\externals
warnings.warn(msg, category=FutureWarning)
Doing training.
Done training.
Training set mean absolute error: 31.3539
Test set mean absolute error: 34.1530
Accurucay of unseen data 0.7407175449259942

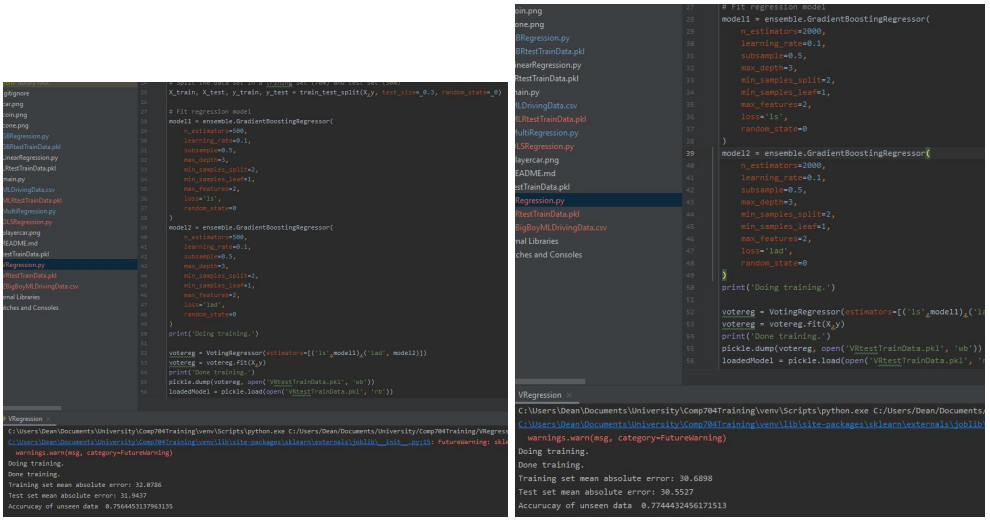

model = ensemble.GradientBoostingRegressor(
n_estimators=3000,
learning_rate=0.2,
max_depth=3,
min_samples_split=2,
min_samples_leaf=1,
max_features=0.1,
loss='huber',
random_state=0
)
print('Doing training.')
# Fit model
model.fit(X_train, y_train)
print('Done training.')
GBRegression
C:\Users\Dean\Documents\University\Comp704Training\venv\Scripts\python.exe C:/Users/Dean/Doc
warnings.warn(msg, category=FutureWarning)
Doing training.
Done training.
Training set mean absolute error: 30.3295
Test set mean absolute error: 34.3008
Accurucay of unseen data 0.7367097818656203



Least squares performs a little better and performance is also increased



Lad was also alright but still did silly mistakes



```
# Fit regression model
model1 = ensemble.GradientBoostingRegressor(
    n_estimators=2000,
    learning_rate=0.1,
    subsample=0.5,
    max_depth=3,
    min_samples_split=2,
    min_samples_leaf=1,
    max_features=2,
    loss='ls',
    random_state=0
)
model2 = ensemble.GradientBoostingRegressor(
    n_estimators=2000,
    learning_rate=0.1,
    subsample=0.5,
    max_depth=3,
    min_samples_split=2,
    min_samples_leaf=1,
    max_features=2,
    loss='lad',
    random_state=0
)
print('Doing training.')

votereg = VotingRegressor(estimators=[('ls',model1),('lad', model2)])
votereg = votereg.fit(X,y)
print('Done training.')
pickle.dump(votereg, open('VRtestTrainData.pkl', 'wb'))
loadedModel = pickle.load(open('VRtestTrainData.pkl', 'rb'))
```

```
C:\Users\Dean\Documents\University\Comp704Training\venv\Scripts\python.exe C:/Users/Dean/Documents/University/Comp704Training/VRegression.py
C:\Users\Dean\Documents\University\Comp704Training\venv\lib\site-packages\sklearn\externals\joblib\__init__.py:15: FutureWarning: sklearn.externals.joblib is deprecated in 0.21 and will be removed in 0.23. Please import this functionality directly from joblib, which can be installed with: pip install joblib. If this warning is raised when loading pickled models, you may need to re-serialize those models with scikit-learn 0.21+.
  warnings.warn(msg, category=FutureWarning)
Doing training.
Done training.
Training set mean absolute error: 32.0786
Test set mean absolute error: 31.9437
Accuracy of unseen data  0.7564453137963135
```

```
C:\Users\Dean\Documents\University\Comp704Training\venv\Scripts\python.exe C:/Users/Dean/Documents/University/Comp704Training/VRegression.py
C:\Users\Dean\Documents\University\Comp704Training\venv\lib\site-packages\sklearn\externals\joblib\__init__.py:15: FutureWarning: sklearn.externals.joblib is deprecated in 0.21 and will be removed in 0.23. Please import this functionality directly from joblib, which can be installed with: pip install joblib. If this warning is raised when loading pickled models, you may need to re-serialize those models with scikit-learn 0.21+.
  warnings.warn(msg, category=FutureWarning)
Doing training.
Done training.
Training set mean absolute error: 30.6898
Test set mean absolute error: 30.5527
Accuracy of unseen data  0.7744432456171513
```

Voting regression works alot better

N esitmators slows down the game and plays worse that less n estimaros