This is Linear Regression Model to predict solar wind.

This was part of my Capastone project for CSE 485-486

Project Name: Predicting Solar Wind Conditions with Machine Learning – Team Helios

https://psyche.asu.edu/get-involved/capstone-projects/predicting-solar-wind-conditions-with-machine-learning-team-helios/ (https://psyche.asu.edu/get-involved/capstone-projects/predicting-solar-wind-conditions-with-machine-learning-team-helios/)

James Niroomand, Arizona State University, Spring 2020

Project Overview

We took the OMNI and ARTEMIS Spacecraft mission dataset to build a machine learning model The purpose of the model is to predict ARTEMIS Ion density, given OMNI Ion Density and other features such as Omni Latitude, Longitude, magnitude average and date/time. Also, we trained our model with the difference in latitude and longitude between OMNI and ARTEMIS dataset.

Datasets

Dataset: from NASA https://spdf.gsfc.nasa.gov/data_orbits.html (https://spdf.gsfc.nasa.gov/data_orbits.html)

OMNI and ARTEMIS dataset is more complete and clean compare to other datasets that are available and this was one of the reasons we chose these datasets.

```
1. First I took the combined hourly dataset from March to October o
f 2017 and March to October of 2018. This is a large dataset with 3
172 observations.It is a great way to start building and training o
ur model.
2. After I completed the training of my model with a large dataset,
I focused on taking a smaller sample. Therefore, I took the dataset
for May and June of 2018 and trained my model.
```

Machine Learning Algorithm

For this model we will be using Linear Regression model which we import from sklearn Library

In [134]:
```
# libraries to import
import pandas as pd
import numpy as np
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error, r2_score, mean_absolute
import matplotlib.pyplot as plt
from scipy import stats
import statsmodels.api as sm
import statsmodels.formula.api as smf
import math
from matplotlib.pyplot import figure
import seaborn as seabornInstance
import datetime as dt
from sklearn.tree import DecisionTreeRegressor
```

Hourly Data from March 2017 to October 2017 and March 2018 to October 2018 from OMNI and ARTEMIS Spacecraft missions

In [135]:
```
# Dataset from 2017 and 2018
combinedDataFrom2017and2018 = pd.read_csv("../Artemis1and2YearRedux.csv
```

In [136]:
```
count_row = combinedDataFrom2017and2018.shape[0]
print(count_row)
```

```
3172
```

In [137]:
```
# the start of our dataset
combinedDataFrom2017and2018.head()
```

Out[137]:

	EPOCH_TIME_yyyy-mm-ddThh:mm:ss.sssZ	OMNI_LAT_deg	OMNI_LONG_deg	OMNI_MAG_AVG_nT	OMNI_SPEED_kms	C
0	2017-03-17 09:00:00+00:00	-7.1	99.9	2.6	347	
1	2017-03-17 10:00:00+00:00	-7.1	99.9	2.7	348	
2	2017-03-17 11:00:00+00:00	-7.1	100.0	2.7	345	
3	2017-03-17 12:00:00+00:00	-7.1	100.0	2.6	345	
4	2017-03-17 13:00:00+00:00	-7.1	100.1	2.9	344	

In [138]:

```python
# end of our dataset
combinedDataFrom2017and2018.tail()
```

Out[138]:

	EPOCH_TIME_yyyy-mm-ddThh:mm:ss.sssZ	OMNI_LAT_deg	OMNI_LONG_deg	OMNI_MAG_AVG_nT	OMNI_SPEED_kms
3167	2018-10-19 23:00:00+00:00	5.6	309.5	2.1	307
3168	2018-10-28 19:00:00+00:00	4.8	318.3	3.5	330
3169	2018-10-28 20:00:00+00:00	4.8	318.4	3.8	329
3170	2018-10-28 21:00:00+00:00	4.8	318.4	3.9	326
3171	2018-10-28 22:00:00+00:00	4.8	318.5	3.9	320

In [139]:

```python
# here I am deleting some variables which i do not need for this model.
del combinedDataFrom2017and2018['Time_offset_hours']
del combinedDataFrom2017and2018['new_time']
del combinedDataFrom2017and2018['EPOCH_TIME__yyyy-mm-ddThh:mm:ss.sssZ']
del combinedDataFrom2017and2018['ARTEMIS_DIST_AU']
del combinedDataFrom2017and2018['ARTEMIS_LAT_DEG']
del combinedDataFrom2017and2018['ARTEMIS_LONG_DEG']
del combinedDataFrom2017and2018['SCALED_ARTEMIS_DENSITY']
del combinedDataFrom2017and2018['SCALED_ARTEMIS_MAG_AVG']
```

In [140]:

```python
combinedDataFrom2017and2018.head()
```

Out[140]:

	EPOCH_TIME_yyyy-mm-ddThh:mm:ss.sssZ	OMNI_LAT_deg	OMNI_LONG_deg	OMNI_MAG_AVG_nT	OMNI_SPEED_kms	C
0	2017-03-17 09:00:00+00:00	-7.1	99.9	2.6	347	
1	2017-03-17 10:00:00+00:00	-7.1	99.9	2.7	348	
2	2017-03-17 11:00:00+00:00	-7.1	100.0	2.7	345	
3	2017-03-17 12:00:00+00:00	-7.1	100.0	2.6	345	
4	2017-03-17 13:00:00+00:00	-7.1	100.1	2.9	344	

In [141]:
```
# renaming the dataset
# here the latittude and longitude are the differences between the lat
combinedDataFrom2017and2018.columns = [ "Date/Time",'Omni Latitude', 'O

```

In [142]:
```
# setting the data and time to represent hours and removing extra stuff
combinedDataFrom2017and2018['Date/Time'] = combinedDataFrom2017and2018[

```

```
/opt/anaconda3/lib/python3.7/site-packages/ipykernel_launcher.py:2: Depre
cationWarning: parsing timezone aware datetimes is deprecated; this will
raise an error in the future
```

Pulling the date and time and assigning it to its column

In [143]:
```
combinedDataFrom2017and2018['Year'] = combinedDataFrom2017and2018['Date
combinedDataFrom2017and2018['Month'] = combinedDataFrom2017and2018['Dat
combinedDataFrom2017and2018['Day'] = combinedDataFrom2017and2018['Date/
combinedDataFrom2017and2018['Hour'] = combinedDataFrom2017and2018['Date
```

Here we are taking our ranamed and cleaned data set and storing it into CSV

In [144]:
```
combinedDataFrom2017and2018.to_csv('cleanedCombinedData.csv', index = F
combinedDataFrom2017and2018.head()
```

Out[144]:

	Date/Time	Omni Latitude	Omni Longitude	Omni Mag Average	Omni speed	Omni Ion Density	Artemis Mag Average	Artemis Ion Densitity	Artemis Speed	La Differ
0	2017-03-17 09:00:00	-7.1	99.9	2.6	347	6.7	2.946000	5.779000	337.230000	2.660
1	2017-03-17 10:00:00	-7.1	99.9	2.7	348	6.6	2.745000	5.808846	337.357692	4.440
2	2017-03-17 11:00:00	-7.1	100.0	2.7	345	6.6	2.773077	5.672308	337.242308	4.440
3	2017-03-17 12:00:00	-7.1	100.0	2.6	345	6.5	2.824815	5.600741	334.555556	5.330
4	2017-03-17 13:00:00	-7.1	100.1	2.9	344	6.6	3.205769	6.133462	335.007692	4.440

We only need to pull certain features for our model to train on

1.We will only pull OMNI and OMNI Speed columns and make a Linear Regression Model based on these two variables and see how good our model is doing
2. Next, I will pull OMNI Speed, Latitude, and Longitude differences between Omni and ARTEMIS Spacecraft, Omni Ion Density, and ARTEMIS Ion Density. We will be using these variables to build a multiple Linear Regression model to predict ARTEMIS Ion Density.
4. Furthermore, we will focus on training our model with OMNI latitude and longitude instead of their differences.
5. Finally, we will take Omni Speed, Omni Mag Average, Omni Ion Density, Latitude, Longitude along with Year, Month, Day and hour to predict Artemis Ion Density

In [49]:

```
# Omni Speed and Artemis Speed in Km
Omni_Speed_and_Artemis_Speed = pd.read_csv("cleanedCombinedData.csv", u

# Pulling Omni Latitude, Longitude, Speed and Ion Density to predict Ar
Omni_long_lat_speed_ion_to_predict_Artemis_Ion_Density = pd.read_csv("c
# Omni Features to predict Artemis Ion Density
Omni_Features_with_Artemis_Ion_Density = pd.read_csv("cleanedCombinedDa
# Omni Features with dates to predict Artemis Ion Density
Omni_long_lat_speed_ion_mag_to_predict_Artemis_Ion_Density = pd.read_cs

_Omni_long_lat_speed_ion_mag_to_predict_Artemis_Ion_Density_ = pd.read_
```

In [145]:

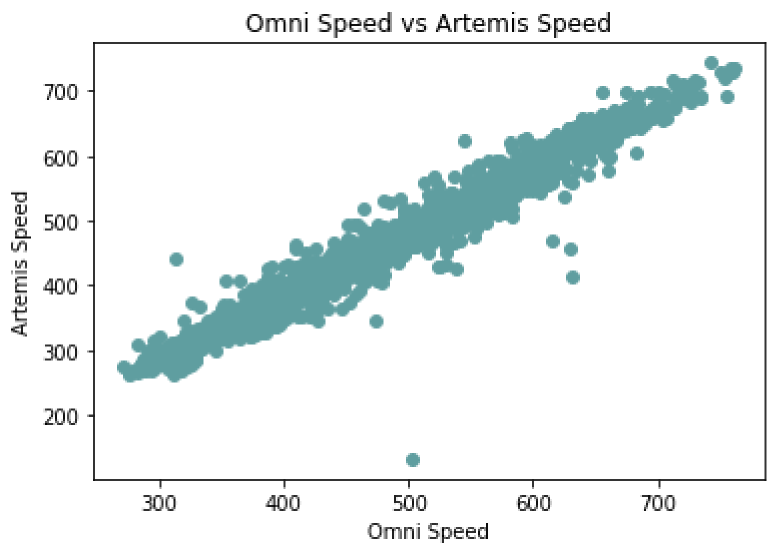
```
Omni_long_lat_speed_ion_mag_to_predict_Artemis_Ion_Density.head()
```

Out[145]:

	Omni Mag Average	Omni speed	Omni Ion Density	Artemis Ion Densitity	Latitude Differences	Longitude Differences	Year	Month	Day	Hour
0	4.3	324	6.7	5.761538	0.1	1.3	2018	5	5	0
1	3.4	325	7.9	5.191905	0.1	1.3	2018	5	5	1
2	9.0	377	25.7	12.820000	0.1	0.9	2018	5	5	11
3	9.3	375	28.8	12.593636	0.1	0.8	2018	5	5	12
4	10.4	376	28.3	15.904000	0.1	0.8	2018	5	5	13

Ploting the relationship between Omni and Artemis Speed

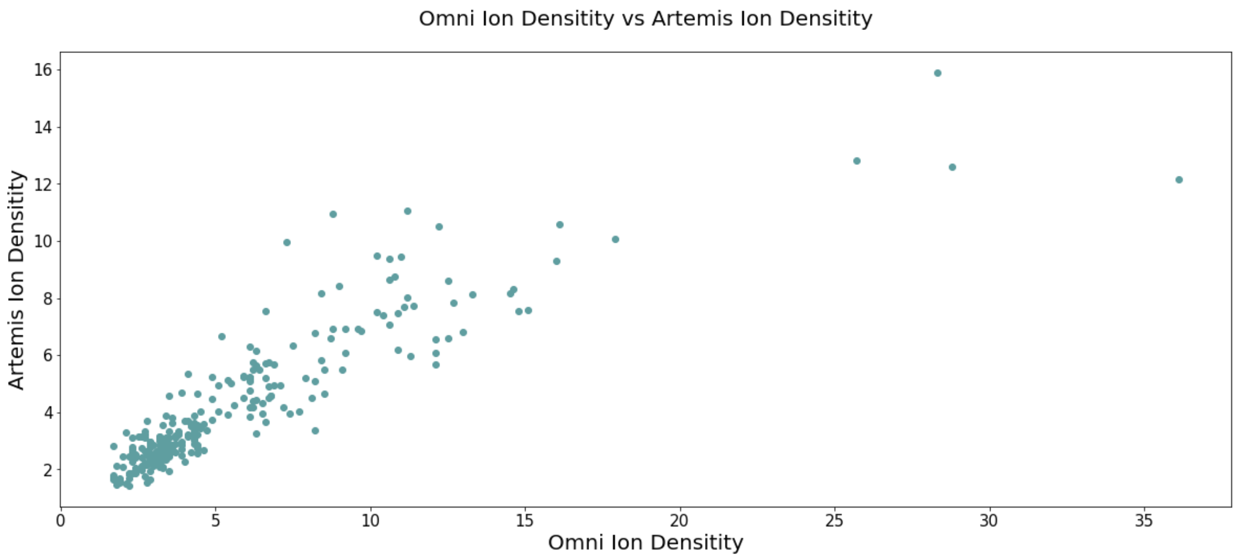
In [15]:
```
# ploting the ion density of Omni dataset vs Ion Density Artemis
plt.plot(Omni_Speed_and_Artemis_Speed['Omni speed'], Omni_Speed_and_Art
plt.title('Omni Speed vs Artemis Speed')
plt.xlabel('Omni Speed')
plt.ylabel('Artemis Speed')
plt.show()
```



Ploting the relationship between Omni and Artemis Ion Density

In [146]:
```
speed_ion_mag_to_predict_Artemis_Ion_Density['Omni Ion Density'], Omni_long_
sitity vs Artemis Ion Densitity')
nsitity')
 Densitity')

```



Measuring the Correlation in our data

In [17]:
```
Omni_Speed_and_Artemis_Speed.corr()
```

Out[17]:

	Omni speed	Artemis Speed
Omni speed	1.000000	0.985268
Artemis Speed	0.985268	1.000000

In [147]:

```
# finding relationship in our data
Omni_long_lat_speed_ion_mag_to_predict_Artemis_Ion_Density.corr()
```

Out[147]:

	Omni Mag Average	Omni speed	Omni Ion Density	Artemis Ion Densitity	Latitude Differences	Longitude Differences	Year	Month	
Omni Mag Average	1.000000	0.253567	0.367032	0.450457	-0.207920	-0.039226	NaN	-0.181581	-0.
Omni speed	0.253567	1.000000	-0.373477	-0.407115	0.107803	-0.148841	NaN	-0.242343	-0.
Omni Ion Density	0.367032	-0.373477	1.000000	0.889513	-0.166208	0.043942	NaN	0.170449	0.
Artemis Ion Densitity	0.450457	-0.407115	0.889513	1.000000	-0.203114	0.038264	NaN	0.138099	0.
Latitude Differences	-0.207920	0.107803	-0.166208	-0.203114	1.000000	0.621193	NaN	-0.088705	0.
Longitude Differences	-0.039226	-0.148841	0.043942	0.038264	0.621193	1.000000	NaN	-0.097831	0.
Year	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	
Month	-0.181581	-0.242343	0.170449	0.138099	-0.088705	-0.097831	NaN	1.000000	-0.
Day	-0.229337	-0.584919	0.019431	0.129506	0.065344	0.026778	NaN	-0.112764	1.
Hour	0.138633	0.365113	-0.105141	-0.139192	-0.395571	-0.614392	NaN	0.075286	-0.

- Very Strong relationship(|r| >0.8)
- Strong Relationship (0.6 <= |r|)
- Moderate Relatuin
- Weak Relationship (|r| >= 0.2)
- Very weak relationship (|r|)

Creating a Statistical Summary

In [148]:
```
1 Omni_long_lat_speed_ion_mag_to_predict_Artemis_Ion_Density.describe()
```

Out[148]:

	Omni Mag Average	Omni speed	Omni Ion Density	Artemis Ion Densitity	Latitude Differences	Longitude Differences	Year	Month
count	263.000000	263.000000	263.000000	263.000000	263.000000	263.000000	263.0	263.000000
mean	4.921293	448.961977	5.625475	4.187141	0.112167	0.805703	2018.0	5.258555
std	2.316668	111.169145	4.444449	2.406352	0.051728	0.292407	0.0	0.438675
min	1.500000	294.000000	1.700000	1.425000	0.000000	0.200000	2018.0	5.000000
25%	3.400000	358.500000	3.000000	2.569167	0.100000	0.600000	2018.0	5.000000
50%	4.300000	426.000000	3.900000	3.235714	0.100000	0.700000	2018.0	5.000000
75%	5.500000	528.500000	6.650000	5.219167	0.100000	1.100000	2018.0	6.000000
max	15.900000	707.000000	36.100000	15.904000	0.200000	1.400000	2018.0	6.000000

In [20]:
```
1 Omni_Speed_and_Artemis_Speed.describe()
```

Out[20]:

	Omni speed	Artemis Speed
count	3172.000000	3172.000000
mean	445.127680	426.535276
std	104.254016	102.998489
min	272.000000	132.900000
25%	361.750000	344.235714
50%	420.000000	402.043091
75%	517.000000	494.448095
max	761.000000	743.750000

Building our Model

In [149]:
```
Omni_long_lat_speed_ion_mag_to_predict_Artemis_Ion_Density.tail()
```

Out[149]:

	Omni Mag Average	Omni speed	Omni Ion Density	Artemis Ion Densitity	Latitude Differences	Longitude Differences	Year	Month	Day	Hour
258	1.7	394	3.3	2.771538	0.1	0.6	2018	6	21	10
259	2.5	390	3.2	3.102727	0.1	0.5	2018	6	21	11
260	2.7	361	4.6	3.595000	0.1	0.6	2018	6	22	10
261	3.3	351	4.3	3.607308	0.1	0.6	2018	6	22	11
262	3.4	352	5.1	4.036667	0.1	0.6	2018	6	22	12

In [152]:
```
# Spliting the data
X = Omni_long_lat_speed_ion_mag_to_predict_Artemis_Ion_Density[['Omni M
# Y is our output
Y = Omni_long_lat_speed_ion_mag_to_predict_Artemis_Ion_Density['Artemis
# X = Omni_Speed_and_Artemis_Speed[['Omni speed']].values
# Y = Omni_Speed_and_Artemis_Speed[['Artemis Speed']].values

# X = Omni_Features_with_Artemis_Ion_Density[['Omni speed','Omni Ion De
# Y = Omni_Features_with_Artemis_Ion_Density[['Artemis Ion Densitity']]

```

In [153]:
```
# spliting the data into training and testing
X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size = 0
```

Creating and Fitting the Model

Linear Regression Equation

Y = B0 + B1*X1* + *B2*X2 + e

The variables in the model are:

1. Y, the response variable;
2. X1, the first predictor variable;
3. X2, the second predictor variable; and
4. e, the residual error, which is an unmeasured variable.

The parameters in the model are:

1. B0, the Y-intercept;
2. B1, the first regression coefficient;
3. B2, the second regression coefficient.

In [154]:
```
regr = LinearRegression()
regr.fit(X_train, y_train)
```

Out[154]:
```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None, normalize=
False)
```

In [155]:
```
print(len(X_test))
```

```
53
```

In [161]:

```
Here I am pulling the year and month from our t
onth = []
ear = []
our = []
ay = []

for i in range(200,250):
month.append(X_test[i][6])

for i in range(200,250):
day.append(X_test[i][7])

for i in range(200,250):
hour.append(X_test[i][8])

for i in range(len(X_test)):
year.append(X_test[i][5])
ndex = []

or i in range(52):
index.append(str(X_test[i][5]) + ',' + str(X_test[i][6]) + ',' + str(X_te

middle = endDate = str(X_test[25][5]) + ',' + str(X_test[25][6]) + ',' + st

endDate = str(X_test[52][5]) + ',' + str(X_test[52][6]) + ',' + str(X_test[

print(startDate)
print(middle)



print(endDate)

rint(index)
```

```
['2018.0,5.0,23.0,12.0:00', '2018.0,5.0,13.0,5.0:00', '2018.0,6.0,6.0,21.0:00', '2018.0,5.0,8.0,17.0:00', '2018.0,5.0,5.0,23.0:00', '2018.0,5.0,19.0,0.0:00', '2018.0,5.0,5.0,21.0:00', '2018.0,5.0,9.0,19.0:00', '2018.0,5.0,15.0,1.0:00', '2018.0,5.0,24.0,9.0:00', '2018.0,5.0,14.0,23.0:00', '2018.0,5.0,18.0,8.0:00', '2018.0,6.0,14.0,4.0:00', '2018.0,5.0,24.0,12.0:00', '2018.0,5.0,20.0,4.0:00', '2018.0,6.0,3.0,20.0:00', '2018.0,5.0,11.0,20.0:00', '2018.0,5.0,20.0,5.0:00', '2018.0,5.0,10.0,23.0:00', '2018.0,5.0,16.0,22.0:00', '2018.0,5.0,22.0,10.0:00', '2018.0,5.0,24.0,11.0:00', '2018.0,6.0,13.0,0.0:00', '2018.0,6.0,16.0,4.0:00', '2018.0,5.0,6.0,23.0:00', '2018.0,5.0,9.0,22.0:00', '2018.0,5.0,8.0,22.0:00', '2018.0,5.0,12.0,23.0:00', '2018.0,5.0,15.0,5.0:00', '2018.0,5.0,13.0,3.0:00', '2018.0,6.0,7.0,22.0:00', '2018.0,5.0,8.0,3.0:00', '2018.0,5.0,8.0,2.0:00', '2018.0,5.0,16.0,2.0:00', '2018.0,5.0,7.0,14.0:00', '2018.0,6.0,22.0,12.0:00', '2018.0,5.0,11.0,21.0:00', '2018.0,5.0,15.0,6.0:00', '2018.0,5.0,10.0,3.0:00', '2018.0,6.0,21.0,10.0:00', '2018.0,5.0,13.0,20.0:00', '2018.0,5.0,19.0,7.0:00', '2018.0,5.0,5.0,15.0:00', '2018.0,5.0,18.0,6.0:00', '2018.0,5.0,15.0,2.0:00', '2018.0,5.0,12.0,22.0:00', '2018.0,5.0,21.0,8.0:00', '2018.0,5.0,24.0,7.0:00', '2018.0,6.0,18.0,7.0:00', '2018.0,6.0,18.0,8.0:00', '2018.0,5.0,20.0,8.0:00', '2018.0,5.0,16.0,7.0:00']
```

In [162]:
```
# indexDate = "2018, 05, 18, 2:00 AM"
# endIndex = "2018, 05, 23, 10:00 AM"
index  = [s.replace('.0', '') for s in index] # remove all the 8s

```

In [158]:
```
# making prediction with our test data
# We trained our model with 80 percent of our sample size and here we a
# with 20 percent of the sample size
y_pred = regr.predict(X_test)
predictedData = pd.DataFrame({'Actual Ion Density': y_test.flatten(), '
predictedData[:10]

# predictedData = pd.DataFrame({'Actual Speed (km)': y_test.flatten(),
# predictedData[:10]

```

Out[158]:

	Actual Ion Density	Predicted Ion Density
0	6.910000	6.398556
1	5.697500	4.716791
2	2.713333	4.551232
3	2.152500	2.105379
4	5.096667	4.453908
5	2.367500	3.035096
6	6.758333	5.788880
7	2.711429	2.589757
8	5.025714	3.899237
9	2.848571	3.368171

In [159]:
```
#To retrieve the intercept:
print("Intercept %.2f" % regr.intercept_)
#For retrieving the slope:
print("slope: ",  regr.coef_)
```

```
Intercept 1.41
slope:  [ 2.28445622e-01 -2.78710911e-03  3.98822867e-01 -8.51171329e-01
 -2.00409855e-01  1.56819002e-15  9.71491915e-02  3.44916356e-02
 -5.16051642e-03]
```

In [160]:
```
# getting the prediction of the model

from sklearn.metrics import mean_absolute_error, median_absolute_error
print("The Explained Variance and Accuracy of our model is: %.2f" % reg
```

```
The Explained Variance and Accuracy of our model is: 0.88
```

Plotting

In [170]:

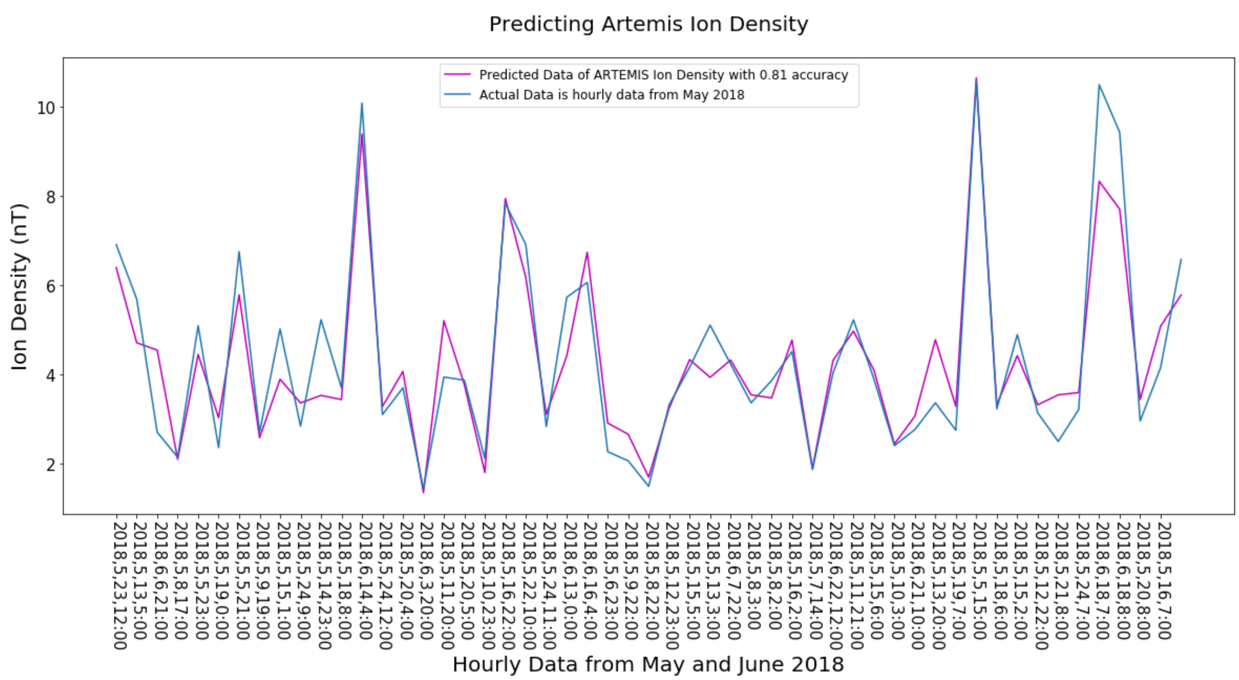
```
%matplotlib inline
# X_test is the test data set.

size=20
params = {'legend.fontsize': 'large',
          'figure.figsize': (20,8),
          'axes.labelsize': size,
          'axes.titlesize': size,
          'xtick.labelsize': size*0.75,
          'ytick.labelsize': size*0.75,
          'axes.titlepad': 25}
plt.rcParams.update(params)

plt.xticks( range(52), index)
plt.xticks(rotation=270)
plt.title("Predicting Artemis Ion Density")
plt.plot(y_pred, 'm', label='Predicted Data of ARTEMIS Ion Density with
plt.plot(y_test, label='Actual Data is hourly data from May 2018 ')
plt.ylabel("Ion Density (nT)")
plt.xlabel("Hourly Data from May and June 2018" )

plt.legend()
#plt.xlim([0,53])
#plt.xlim([400,450])
plt.show()

```



Evaluating our Model

Using the Statsmodel

In [171]:
```
X = sm.add_constant(X) # adding a constant

model = sm.OLS(Y, X).fit()
predictions = model.predict(X)

```

Confidence Interval

In [172]:
```
model.conf_int()
```

Out[172]:
```
array([[ 1.68008316e-01,  2.99570475e-01],
       [-3.95315492e-03, -2.85152436e-04],
       [ 3.72524951e-01,  4.42928800e-01],
       [-5.19711966e+00,  1.45300071e+00],
       [-5.44940210e-01,  8.59714760e-01],
       [-1.18833985e-03,  1.30986166e-03],
       [-9.57281270e-02,  5.17532681e-01],
       [ 1.26005378e-02,  7.64400299e-02],
       [-2.66330665e-02,  2.05925137e-02]])
```

Hypothesis Testing

```
- Null Hypothesis: There is no relationship between our data
- Alternative Hypothesis: There is relationship between our data
```

In [173]:
```
model.pvalues
```

Out[173]:
```
array([2.28648229e-11, 2.37067283e-02, 2.02899724e-63, 2.68577012e-01,
       6.59357736e-01, 9.23757826e-01, 1.76771222e-01, 6.44946114e-03,
       8.01324932e-01])
```

The p-value represents the probability coefficent equals to 0. We want a p-value that is less than 0.05, if it is we can reject the null hypothesis.

Next we can evaluate how well our model is doing

- Mean Absolute Error(MAE): Is the mean of the absolute value of the errors. This metric gives an idea of magnitude but no idea about direction
- Mean Squared Error(MSE): Is the mean of the squared errors.
- Root Mean Squared Error(RMSE): Is the square root of the mean of the squared errors.

In [174]:
```
# Mean Squared Error
model_mse = mean_squared_error(y_test, y_pred)
# Mean Absolute Error
model_mae = mean_absolute_error(y_test, y_pred)
# Rppt mean Squared Error
model_rmse = math.sqrt(model_mse)

print("MSE {:.3}".format(model_mse))
print("MAE {:.3}".format(model_mae))
print("RMSE {:.3}".format(model_rmse))
```

```
MSE 0.603
MAE 0.578
RMSE 0.777
```

R-Squared

- The R-Squared metric provides us a way to measure the goodness of fit or how well our data fits the model. The highest the R-squared metric the better the data fit our model.

In [175]:
```
r2 = r2_score(y_test,y_pred)
print("R2 Score {:.2}".format(r2))
```

```
R2 Score 0.88
```

Create a summar of the model output

In [2093]:
```
print(model.summary())
```

```
                            OLS Regression Results
==========================================================================
=====
Dep. Variable:                      y   R-squared:
0.735
Model:                            OLS   Adj. R-squared:
0.735
Method:                 Least Squares   F-statistic:
976.9
Date:                Sun, 26 Apr 2020   Prob (F-statistic):
0.00
Time:                        19:30:32   Log-Likelihood:                 -5
869.7
No. Observations:                3172   AIC:                          1.17
6e+04
Df Residuals:                    3162   BIC:                          1.18
2e+04
Df Model:                           9
Covariance Type:            nonrobust
==========================================================================
=====
                 coef    std err          t      P>|t|      [0.025
0.975]
--------------------------------------------------------------------------
-----
const        262.9178    140.824      1.867      0.062     -13.198      53
9.034
x1            -0.2415      0.028     -8.558      0.000      -0.297       -
0.186
x2            -0.4079      0.080     -5.124      0.000      -0.564       -
0.252
x3             0.1446      0.011     13.186      0.000       0.123
0.166
x4            -0.0052      0.000    -17.226      0.000      -0.006       -
0.005
x5             0.4259      0.006     67.283      0.000       0.414
0.438
x6            -0.1309      0.070     -1.871      0.061      -0.268
0.006
x7            12.4744      2.378      5.246      0.000       7.812        1
7.137
x8             0.4001      0.078      5.157      0.000       0.248
0.552
x9             0.0141      0.005      2.709      0.007       0.004
0.024
==========================================================================
=====
Omnibus:                     2339.953   Durbin-Watson:
0.958
Prob(Omnibus):                  0.000   Jarque-Bera (JB):            27287
8.689
Skew:                           2.699   Prob(JB):
0.00
Kurtosis:                      48.117   Cond. No.                      1.0
7e+07
```

```
==============================================================================
=====

Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is co
rrectly specified.
[2] The condition number is large, 1.07e+07. This might indicate that the
re are
strong multicollinearity or other numerical problems.
```

Once you have obtained your error metric/s, take note of which X's have minimal impacts on y. Removing some of these features may result in an increased accuracy of your model.

In [1512]:

```
# we save our model
import pickle
# save the model
with open("Linear_Regression_Model", 'wb') as f:
    pickle.dump(regr,f)
```

In [1513]:

```
# this is how we open our model for fututre use.
with open('Linear_Regression_Model', 'rb') as pickle_file:
    reg_mode_2 = pickle.load(pickle_file)
```

In [ ]:

```

```