CSEE 5590/490: Big Data Programming

Project Report

Due Date: Monday, May 10, 2021

Project Title: Analysis for determination of a relationship between energy demand and weather.

Team Members: Joe Goldsich, Anna Johnson, Kyle Son, and Bill Yerkes

Introduction: Information overload can make it difficult to understand issues and solve problems. The appearance of excessive quantity of information prevents the understanding of the problem and the ability to construct a solution to the problem. Big Data technologies takes this paradigm and turns it inside out. The students working on this project are going to attempt to construct a model which will be able to take weather and energy related data to be able to forecast energy demand.

Background:

There are numerous articles and papers about leveraging weather data, weather forecast, to be able to forecast energy demand and energy production via renewable sources. An article in Wired talks about how this relates to energy pricing and how the technology has moved from using spreadsheets to the using of machine learning.[1] The company BAXENERGY has a web page about the benefits of this field.[2] Hugo Ferreira wrote a paper about predicting wind and solar generation using machine learning in 2018.[3] They all talk about the vast amount of data which gets generated and stored. How this data can be leveraged to help improve the energy sector, and in turn help improve society.

Goals and Objectives: Utilize the tools and technologies learned from CSEE 5590 to be able to analyze collected data so that it will be possible to determine if there is a relationship between weather and energy consumption and if a relationship exists determine the possibilities of using that relationship to predict future energy needs.

Motivation: The global population continues to increase, and the weather patterns seem to be getting more extreme, from extending periods of both above and below normal temperatures in various parts of the world and in the United States. The demand and consumption of energy increases with the population and with the extreme weather, the need for air conditioning in the summer and for heating in the winter. The recent crisis in Texas has demonstrated what can happen if the energy providers are not able to meet the demands of the consumers. Being able to

forecast accurately future demand and plan accordingly can help prevent or mitigate such crises in the future.

Significance: Better planning of resources for Utility Companies can result in reduced cost to the consumers and more reliable service. This also dips into the area of public safety, as loss of power during extreme weather with no warning can be dangerous for vulnerable groups.

Objectives:

What weather measurements and cities influence most the electrical demand, prices, generation capacity?

Forecast intraday price or electrical demand hour-by-hour.

What is the next generation source to be activated on the load curve?

Analyze the change of the electricity price yearly and hourly and get some insights

Create ML or DL model to predict the electricity price

Create Clustering model by the weather features

Could We perform Map Reduce on the datasets to get some refined data

Plotting some graphs for visualization with meaningful insights

Approaches/Methods:

The team's initial form of communication was via Email. The discussion over email resulted in the team deciding to communicate via Discord, which has allowed for the team member to communicate on a regular basis. The team also meets periodically via Zoom to review items and discuss plans.

The team uses Google Docs and Google Slides to collaborate on documentation. The team also uses GitHub as the repository for source code and final version of documentation.

The team met to discuss several ideas on what topic to do the project on and came to a consensus to do the project on Energy prediction based on Weather. The team reviewed the information on the Kaggle site. The team decided to leverage the knowledge we had gained over the first half of

the course to investigate both the technologies, Hadoop, MapReduce, Hive, Sqoop, Cassandra, Solr, that were covered and the data, Weather and Energy datasets.

Each team member worked on their own. They utilized the various tools to do analysis on the data for the project. This allowed the team members to get more practice with the various tools and to get familiar with the data. This approach led to discussions of common issues found and exchanges of ideas between team members. The team shared with each other their findings on various tools and data. The team also started discussion on what approach the team should take for the next iteration.

Our Solution:

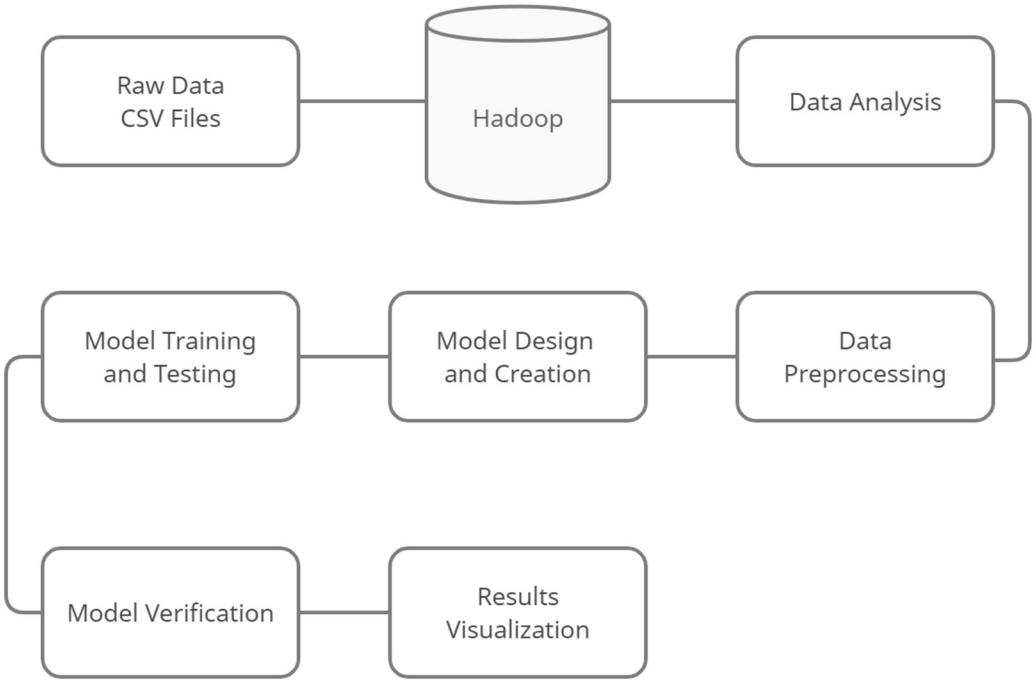
The processes of creating the solution for our project began with a set of CSV files. The data in the CSV files was stored in Hadoop. This process was done both in Cloudera, directly into Hadoop, and later via Spark.

The first half of the project effort was focused on utilizing the tools learned in the first half of the semester to perform data analysis of the data. This was done using Hive, MapReduce, Sqoop with MySQL and Hadoop. Insights into the data gave us an overview of the data that was to be utilized. This research highlighted an issue with the date time field and how it was configured, which we addressed in the second half of the project.

The second half of the project was focused on leveraging Spark. The first step which was taken was to "clean" the data so that it would be ready for analysis. Records with missing values were dropped from the datasets. The date time field, which included a time offset, had to be converted to a timestamp field. The two datasets were joined together based on the newly created timestamp field.

The Machine Learning module of the class was at the end of the semester, which required the team to investigate this topic on their own. The team created several models to predict energy demand based on features present in the weather dataset. These ranged from the straight forward linear regression to the complex Long Short Term Memory model. Ridge, Lasso, and Elastic regression models were created as well.

After a model was created the data was split into training and test sets. The model was trained on the training data and the accuracy of the model was checked on the test set. Finally the model was used to verify the prediction compared to the actual for the entire data set.



Dataset

Hourly energy demand generation and weather | Kaggle

This dataset contains 4 years of electrical consumption, generation, pricing, and weather data for Spain. Consumption and generation data were retrieved from ENTSOE a public portal for Transmission Service Operator (TSO) data. Settlement prices were obtained from the Spanish TSO Red Electric España. Weather data was purchased as part of a personal project from the Open Weather API for the 5 largest cities in Spain and made public on Kaggle..

Analysis of Data:

Data preprocessing began in iteration two. Utilizing Hive, MapReduce, Sqoop with MySQL, and storing the data in Hadoop, the team performed various queries on the data. The preprocessing and analysis of the data continued into the first part of iteration three. Records and Columns with missing data were removed from the dataset. The issue concerning the date time field was resolved in the beginning of iteration three as well.

Weather Data Summary

	temp	temp_min	temp_max	pressure	humidity
Min.	**262.2**	**262.2**	**262.2**	**0**	**0.00**
1st Qu.	**283.7**	**282.5**	**284.6**	**1013**	**53.00**
Median	**289.1**	**288.1**	**290.1**	**1018**	**72.00**
Mean	**289.6**	**288.3**	**291.1**	**1069**	**68.42**
3rd Qu.	**295.1**	**293.7**	**297.1**	**1022**	**87.00**
Max.	**315.6**	**315.1**	**321.1**	**1008371**	**100.00**

	wind_speed	wind_deg	rain_1h	rain_3h	snow_3h	clouds_all
Min.	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**
1st Qu.	**1.00**	**55.00**	**0.00**	**0.00**	**0.00**	**0.00**
Median	**2.00**	**177.00**	**0.00**	**0.00**	**0.00**	**20.00**
Mean	**2.47**	**166.60**	**0.08**	**0.00**	**0.00**	**25.07**
3rd Qu.	**4.00**	**270.00**	**0.00**	**0.00**	**0.00**	**40.00**
Max.	**133.00**	**360.00**	**12.00**	**2.32**	**21.50**	**100.00**

Energy Data Summary

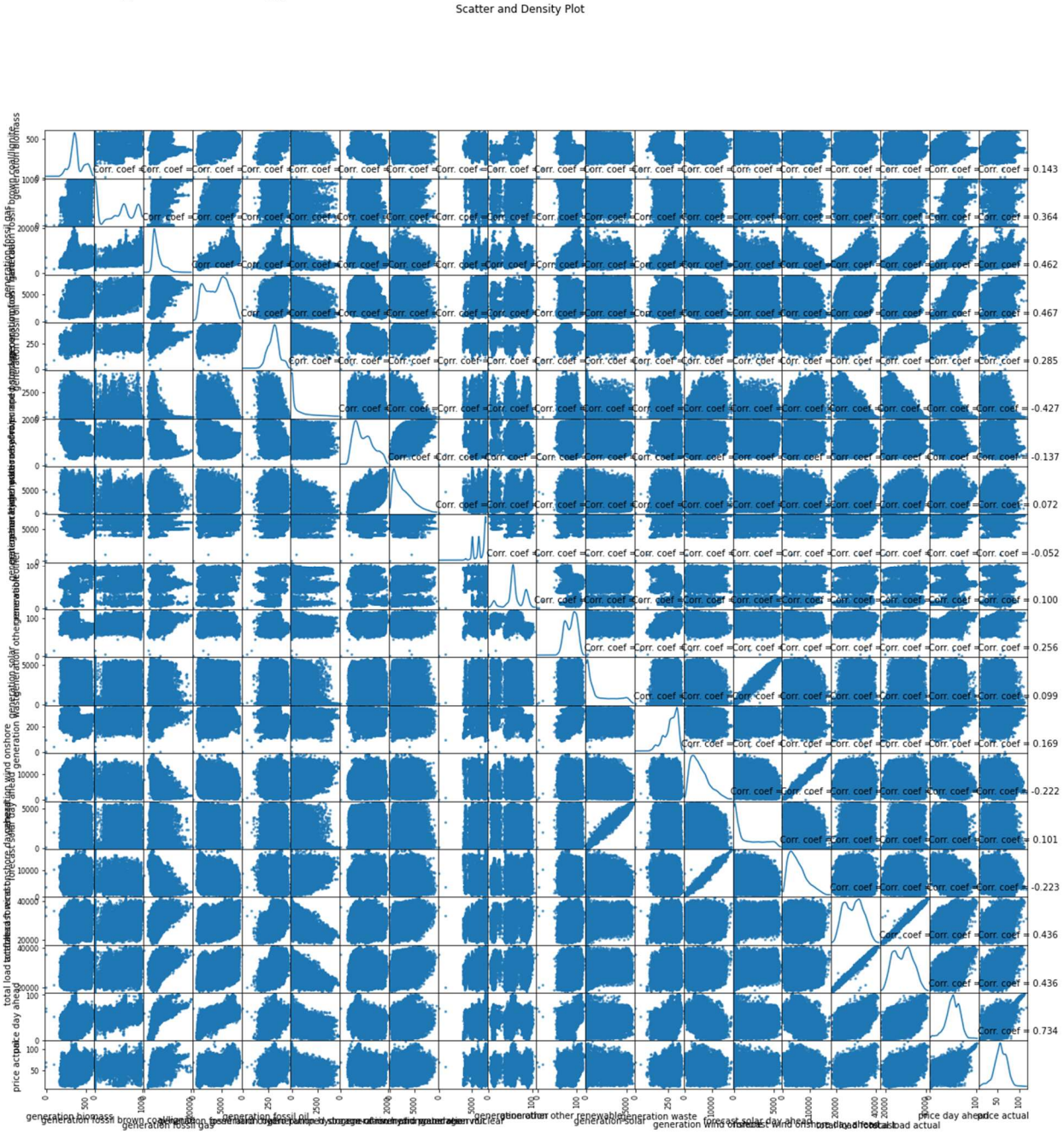
	Generation biomass	Generation Fossil hard coal	Generation hydro pumped storage consumption	Generation hydro water reservoir	Generation fossil brown coal lignite
Min.	**0**	**0**	**0**	**0**	**0**
1st Qu.	**333**	**2527**	**0**	**1077**	**0**
Median	**367**	**4474**	**68**	**2164**	**509**
Mean	**383.5**	**4256**	**475.6**	**2605**	**448.1**
3rd Qu.	**433**	**5839**	**616**	**3757**	**757**
Max.	**592**	**8359**	**4523**	**9728**	**999**
NA's	**19**	**18**	**19**	**18**	**18**

	Generation waste	Forecast wind onshore day ahead	Generation wind onshore	Generation other	Generation fossil oil
Min.	**0**	**237**		**0**	**0**
1st Qu.	**240**	**2979**	**0**	**53**	**263**
Median	**279**	**4855**	**2933**	**57**	**300**
Mean	**269.5**	**5471**	**4849**	**60.23**	**298.3**
3rd Qu.	**310**	**7353**	**5464**	**80**	**330**
Max.	**357**	**17430**	**7398**	**106**	**449**
NA's	**19**		**17436**	**18**	**19**

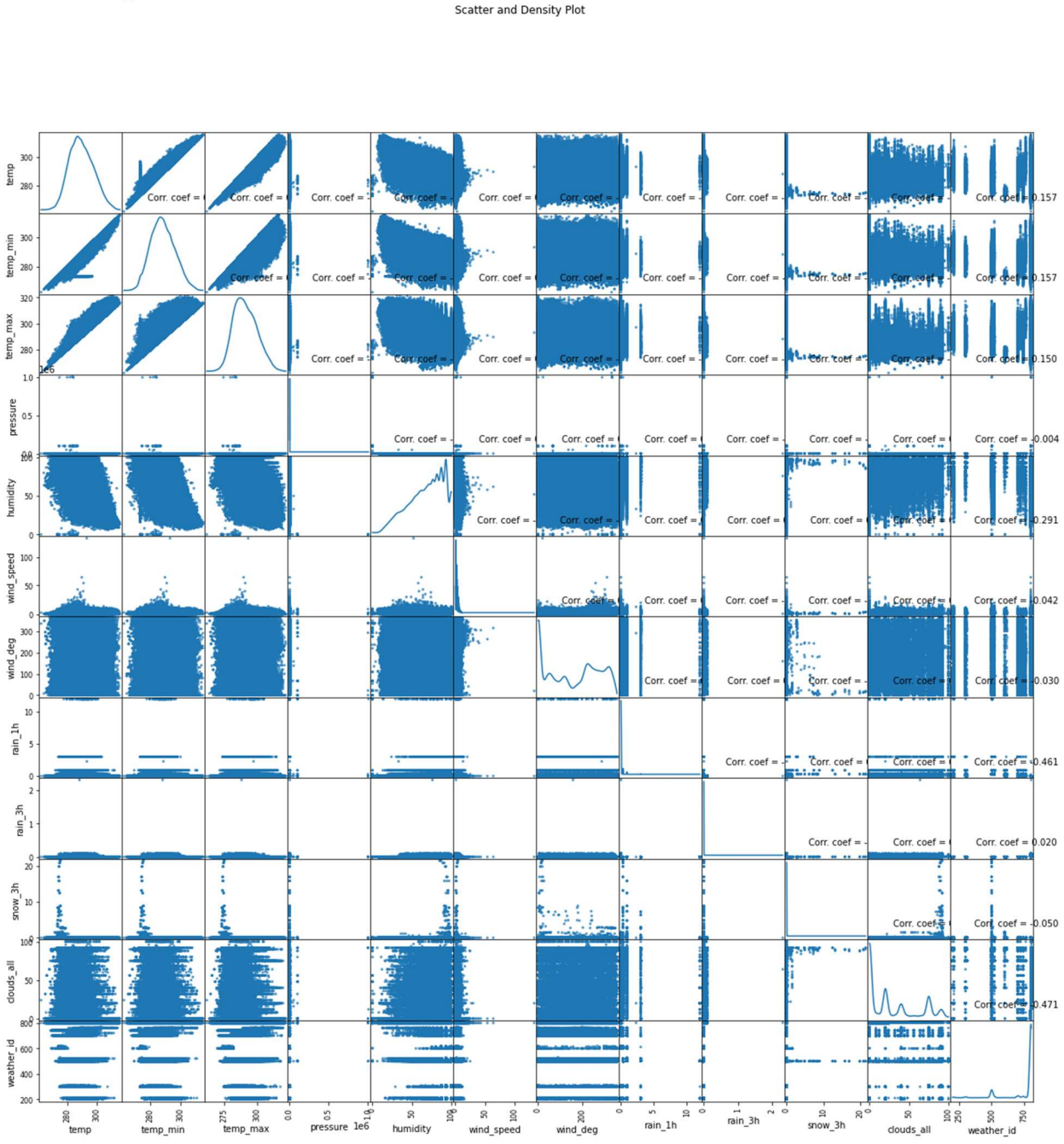
	Forecast solar day ahead	Generation fossil gas	Generation other renewable	Generation nuclear	Generation hydro run of river and poundage
Min.	0	0	0	0	0
1st Qu.	69	4126	73	5760	637
Median	576	4969	88	6566	906
Mean	1439	5623	85.64	6264	972.1
3rd Qu.	2636	6429	97	7025	1250
Max.	5836	20034	119	7117	2000
NA's		18	18	17	19

	Total load forecast	Total load actual	Price day ahead	Price actual
Min.	18105	18041	2.06	9.33
1st Qu.	24794	24808	41.49	49.35
Median	28906	28901	50.52	58.02
Mean	28712	28697	49.87	57.88
3rd Qu.	32263	32192	60.53	68.01
Max.	41390	41015	101.99	116.8
NA's		36		

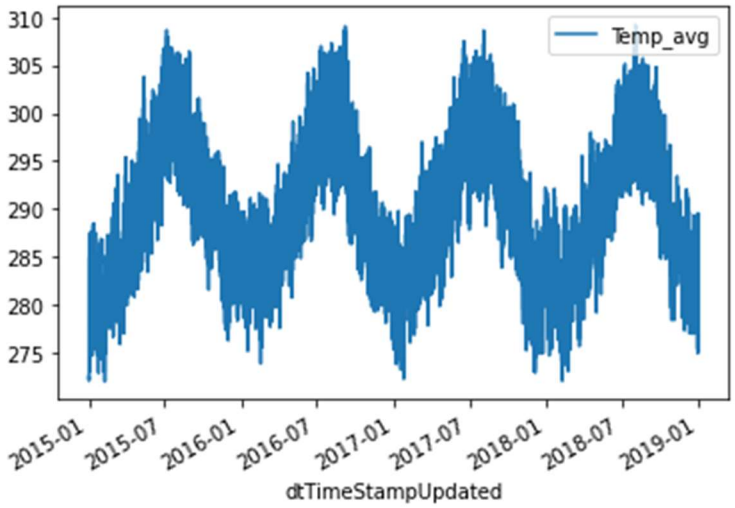
Relationship of the Energy Dataset Features



Relationship of the WeatherDataset Features

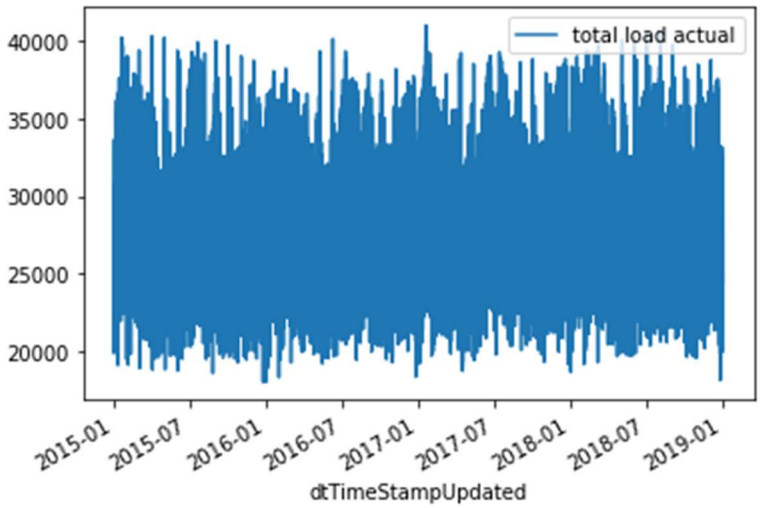


Average Temperature over the five cities:



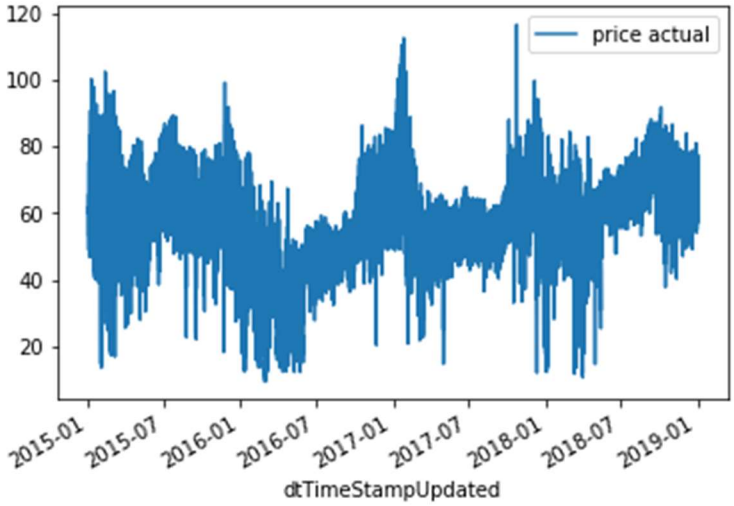
The previous graph shows the average temperature of the five cities over the time frame the data covers. It follows a known pattern of the temperatures being hotter during the summer seasons and colder during the winter seasons. The ten to fifteen degree range of temperature is consistent and reflects the rise and fall of the temperature during a twenty four hour period.

Total Actual Load:



The previous graph shows the Total Actual Load of energy over the time frame the data covers. The data appears very chaotic. It does not appear to follow a pattern. The load appears to fluctuate by a degree of twenty thousand units almost daily. There is some consistency in the data in that the range of Total Actual Load appears to be bounded and does not appear to be either steadily increasing or decreasing over time..

Actual Price:



The previous graph shows the price of energy over the time frame the data covers. It does not appear to be following a pattern. There are several spikes in the data, both on the high and low ends of the range. There does appear to be a trend of the price on average increasing from about May of 2016. The daily range in values also narrows staring at this time as well. There are other economical factors which effect the price of energy over time, such as global economic boom and bust cycles.

Features Implementation

Resolving the issue with the Date Field

Code to Add the time offset into the datefield:

```
from datetime import datetime, timedelta
format = "%Y-%m-%d %H:%M:%S"

def addTimeOffset(v_date, v_hour, v_minute):
  new_datetime = timedelta(minutes = int(v_minute), hours = int(v_hour))
  return_dt = v_date + new_datetime
  date_string = return_dt.strftime(format)
  return date_string
udfaddTimeOffset = f.udf(addTimeOffset, StringType())
```

Code to convert string to timestamp:

```
EnergyDF = EnergyDF.withColumn('dtTimeStamp', to_timestamp(f.substring('time', 1,19), 'yyyy-MM-dd HH:mm:ss')) \
                   .withColumn('dtHourOffset', f.substring('time', 21,2)) \
                   .withColumn('dtMinuteOffset', f.substring('time', 24,2))

WeatherDF = WeatherDF.withColumn('dtTimeStamp', to_timestamp(f.substring('dt_iso', 1,19), 'yyyy-MM-dd HH:mm:ss')) \
                     .withColumn('dtHourOffset', f.substring('dt_iso', 21,2)) \
                     .withColumn('dtMinuteOffset', f.substring('dt_iso', 24,2))
```

Apply the time offset to the timestamp fields

```
EnergyDF = EnergyDF.withColumn("interval", udfaddTimeOffset("dtTimeStamp","dtHourOffset","dtMinuteOffset"))
EnergyDF = EnergyDF.withColumn('dtTimeStampUpdated', to_timestamp("interval", 'yyyy-MM-dd HH:mm:ss'))

WeatherDF = WeatherDF.withColumn("interval", udfaddTimeOffset("dtTimeStamp","dtHourOffset","dtMinuteOffset"))
WeatherDF = WeatherDF.withColumn('dtTimeStampUpdated', to_timestamp("interval", 'yyyy-MM-dd HH:mm:ss'))
```

Linear Regression

```
## Data for perfoming linear regression
linearReg_X = PandasWeatherEnergyDF[['Temp_avg']]
linearReg_y = PandasWeatherEnergyDF[['total load actual']]

body_reg = linear_model.LinearRegression()
body_reg.fit(linearReg_X,linearReg_y)
# parameters
m =body_reg.coef_
c =body_reg.intercept_
plt.scatter(linearReg_X,linearReg_y)
plt.plot(linearReg_X,body_reg.predict(linearReg_X),color='Red')
```

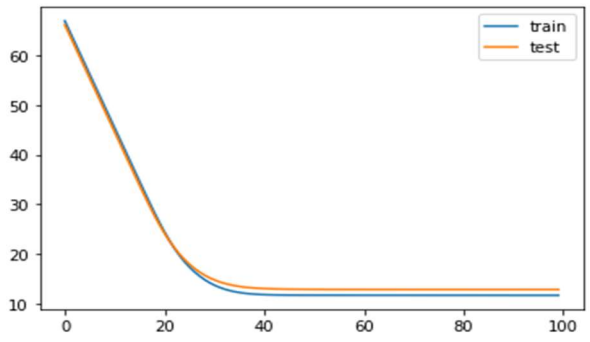
Long Short Term Memory Model:

```
# split into train and test sets
#values = reframed.values
n_train_hours = 365 * 24
train = values[:n_train_hours, :]
test = values[n_train_hours:, :]
# split into input and outputs
train_X, train_y = train[:, :-1], train[:, -1]
test_X, test_y = test[:, :-1], test[:, -1]
# reshape input to be 3D [samples, timesteps, features]
train_X = train_X.reshape((train_X.shape[0], 1, train_X.shape[1]))
test_X = test_X.reshape((test_X.shape[0], 1, test_X.shape[1]))
print(train_X.shape, train_y.shape, test_X.shape, test_y.shape)
```

```
(8760, 1, 3) (8760,) (26268, 1, 3) (26268,)
```

Desiging and creating the model

```
# design network
from keras.layers.core import Dense, Activation, Dropout
from keras.layers.recurrent import LSTM
from keras.models import Sequential

model = Sequential()
model.add(LSTM(100, input_shape=(train_X.shape[1], train_X.shape[2])))
model.add(Dense(1))
model.compile(loss='mae', optimizer='adam')
# fit network
history = model.fit(train_X, train_y, epochs=100, batch_size=72, validation_data=(test_X, test_y), verbose=2, shuffle=False)
# plot history
pyplot.plot(history.history['loss'], label='train')
pyplot.plot(history.history['val_loss'], label='test')
pyplot.legend()
pyplot.show()
```



Ridge Model:

```
[45] rr = Ridge(alpha=0.01)
     rr.fit(X_train, y_train)
     pred_train_rr= rr.predict(X_train)
     print(np.sqrt(mean_squared_error(y_train,pred_train_rr)))
     print(r2_score(y_train, pred_train_rr))

     pred_test_rr= rr.predict(X_test)
     print(np.sqrt(mean_squared_error(y_test,pred_test_rr)))
     print(r2_score(y_test, pred_test_rr))
```

```
4254.827749792242
0.136185478314513
4234.536646970196
0.14060291578775475
```

Lasso Model:

```
[47] model_lasso = Lasso(alpha=0.01)
     model_lasso.fit(X_train, y_train)
     pred_train_lasso= model_lasso.predict(X_train)
     print(np.sqrt(mean_squared_error(y_train,pred_train_lasso)))
     print(r2_score(y_train, pred_train_lasso))

     pred_test_lasso= model_lasso.predict(X_test)
     print(np.sqrt(mean_squared_error(y_test,pred_test_lasso)))
     print(r2_score(y_test, pred_test_lasso))
```

```
4254.827840653484
0.1361854414212399
4234.541855537697
0.14060080163414113
```

Elastic Model:

```
[49] #Elastic Net
     model_enet = ElasticNet(alpha = 0.01)
     model_enet.fit(X_train, y_train)
     pred_train_enet= model_enet.predict(X_train)
     print(np.sqrt(mean_squared_error(y_train,pred_train_enet)))
     print(r2_score(y_train, pred_train_enet))

     pred_test_enet= model_enet.predict(X_test)
     print(np.sqrt(mean_squared_error(y_test,pred_test_enet)))
     print(r2_score(y_test, pred_test_enet))
```

```
4274.636074490374
0.12812378982508799
4255.736646545202
0.1319763168693785
```

Plot Weather Conditions for 2 Weather Variables:

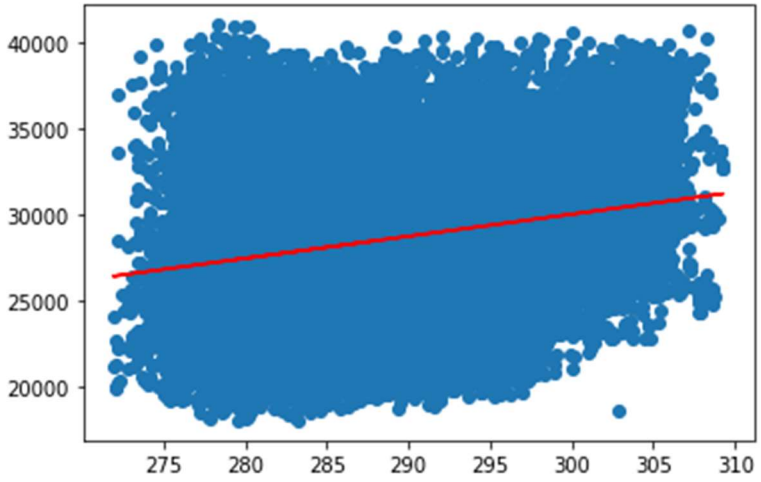
```
def get_condition_list(weather_condition, weather_type, weather_df):

  # Check for proper condition input.
  condition_list = ['temp', 'pressure', 'humidity', 'wind_speed', 'wind_deg']
  if (condition_list.count(weather_condition) == 0):
    print("Improper weather condition passed.")
    print("Condition must be: temp, pressure, humidity, wind_speed, wind_deg")
    return

  # Check for proper weather type
  weather_list = ['clear', 'cloudy', 'rain', 'fog', 'thunderstorm', 'drizzle',
                  'mist', 'dust']
  if (weather_list.count(weather_type) == 0):
    print("Improper weather type passed.")
    print("Weather must be: clear, cloudy, rain, drizzle, fog, mist, dust, or thunderstorm.")
    return

  # Return Create a list of weather conditions when the weather type is present in the DF.
  if isinstance(weather_df, pyspark.sql.dataframe.DataFrame):
    list_for_return = list(weather_df.select(weather_condition) \
                                      .filter(df_w.weather_main == weather_type) \
                                      .toPandas()[weather_condition])
    return list_for_return
  else:
    ## weather_df is not of type dataframe
    print("Must pass a dataframe type")
    return
```

```
def graph_weather_type(condition1, condition2, weather_df):

  # Check for proper condition input.
  condition_list = ['temp', 'pressure', 'humidity', 'wind_speed', 'wind_deg']
  if (condition_list.count(condition1) == 0) or (condition_list.count(condition2) == 0):
    print("Improper weather condition passed.")
    print("Condition must be: temp, pressure, humidity, wind_speed, wind_deg")
    return

  # Use helper condition to get lists of the needed data.
  rain_cond1 = get_condition_list(condition1, 'rain', weather_df)
  rain_cond2 = get_condition_list(condition2, 'rain', weather_df)

  thunderstorm_cond1 = get_condition_list(condition1, 'thunderstorm', weather_df)
  thunderstorm_cond2 = get_condition_list(condition2, 'thunderstorm', weather_df)

  fog_cond1 = get_condition_list(condition1, 'fog', weather_df)
  fog_cond2 = get_condition_list(condition2, 'fog', weather_df)

  drizzle_cond1 = get_condition_list(condition1, 'drizzle', weather_df)
  drizzle_cond2 = get_condition_list(condition2, 'drizzle', weather_df)

  mist_cond1 = get_condition_list(condition1, 'mist', weather_df)
  mist_cond2 = get_condition_list(condition2, 'mist', weather_df)

  dust_cond1 = get_condition_list(condition1, 'dust', weather_df)
  dust_cond2 = get_condition_list(condition2, 'dust', weather_df)

  # Create the graph
  plt.figure(figsize=(18,10))              ## Figure(figsize) changes the size of the output graph
  #plt.plot(clear_pressure, clear_humidity, 'bx', linewidth=1, markersize=12, label='Clear')
  plt.plot(rain_cond1, rain_cond2, 'm.', linewidth=1, markersize=12, label='Rain')
  plt.plot(thunderstorm_cond1, thunderstorm_cond2, 'b.', linewidth=1, markersize=12, label='Thunderstorm')
  plt.plot(drizzle_cond1, drizzle_cond2, 'g.', linewidth=1, markersize=12, label='Drizzle')
```

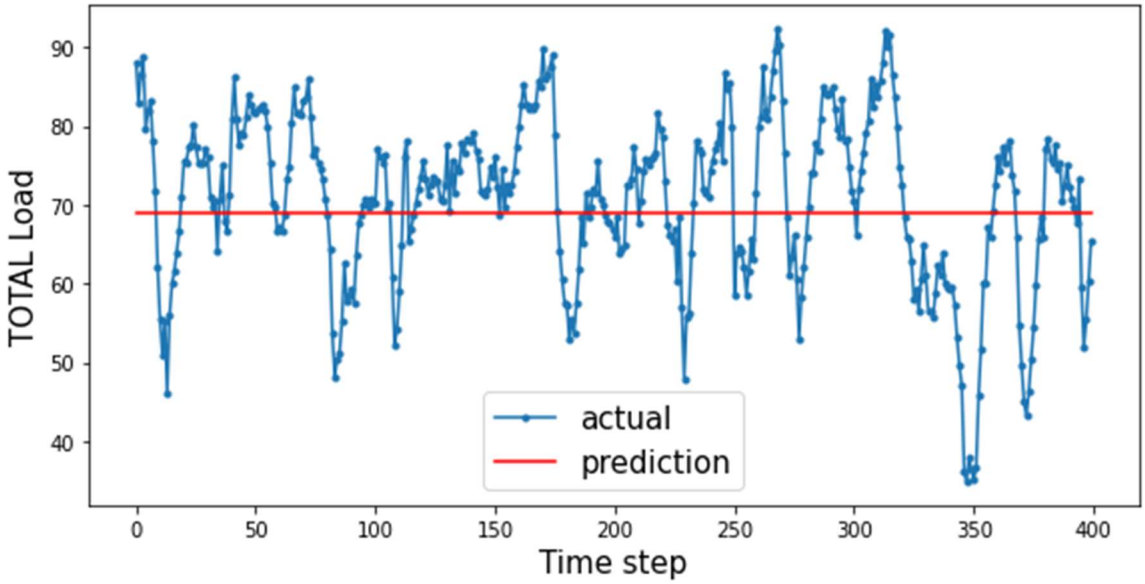
```
# Grab a dataFrame for testing: Seville in April of 2016
df_weather_main = df_energy_weather.select("temp", "pressure", "weather_main") \
                                   .filter((df_w.city_name == 'Bilbao') \
                                   & (df_w.dt_iso.like('2016-06%')))

graph_weather_type('temp', 'pressure', df_weather_main)
```

Results :

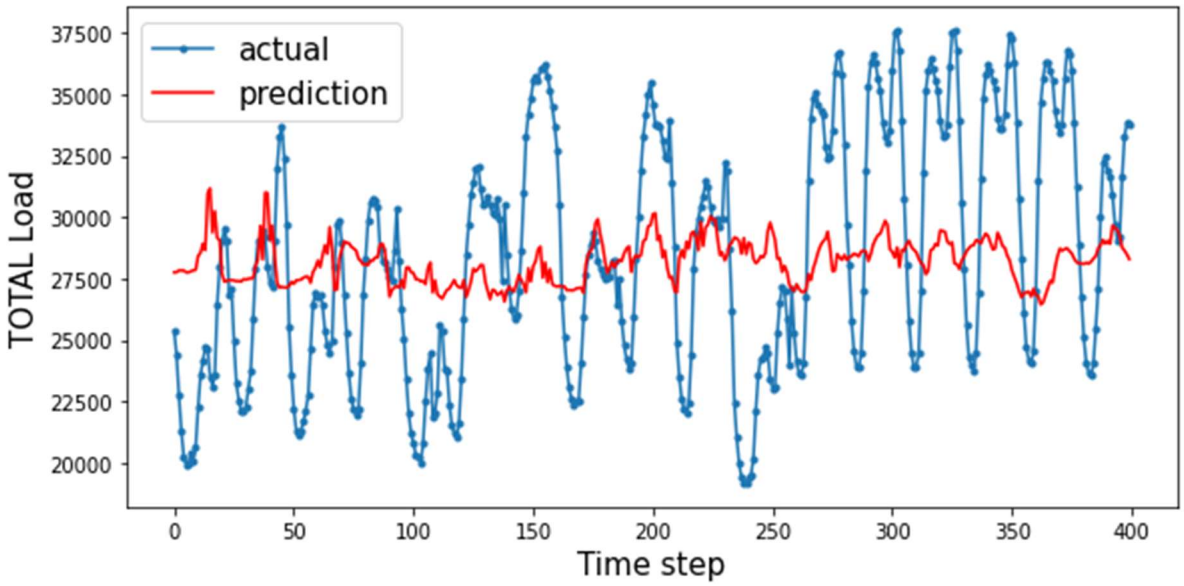
Linear Regression Temperature ~ Total Actual Load:



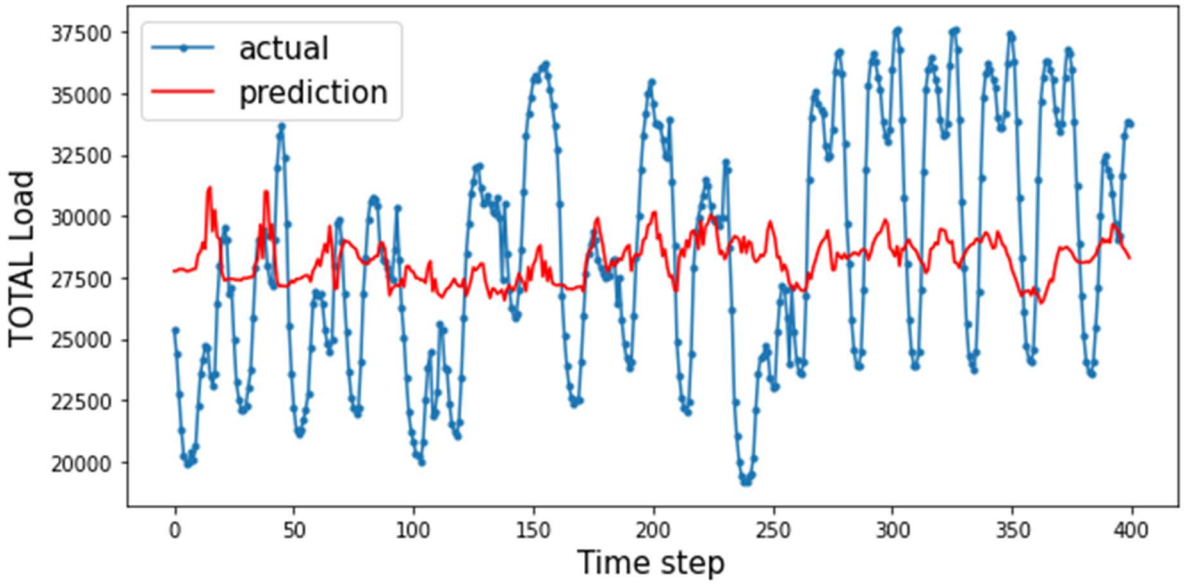
LSTM [Temperature,Pressure,Humidity~Total Actual Load]:



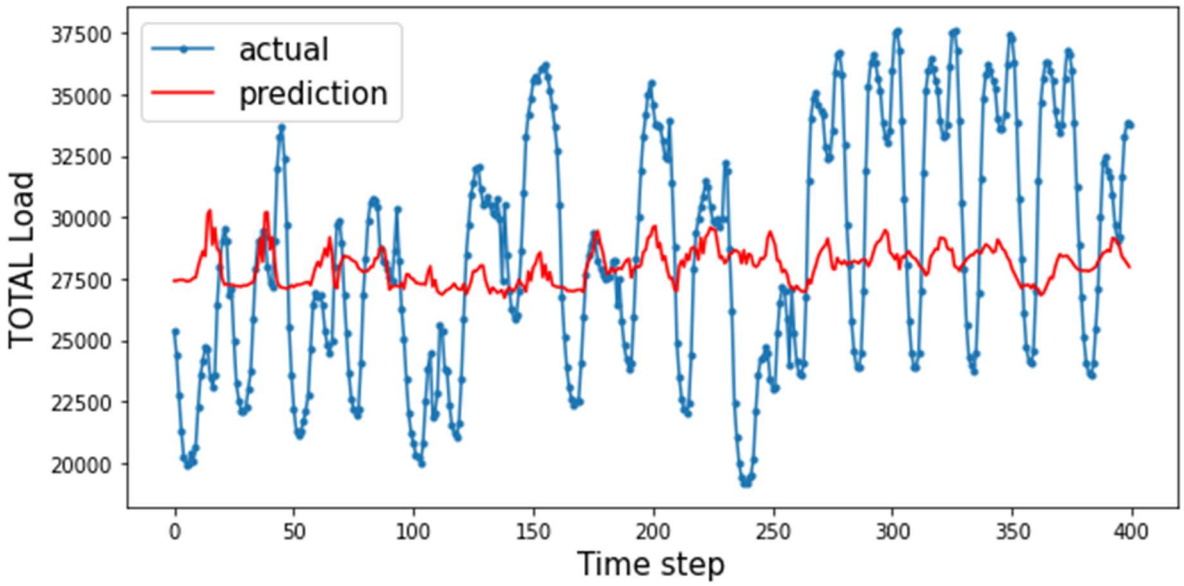
Ridge Regression [Temperature,Pressure,Humidity~Total Actual Load]:



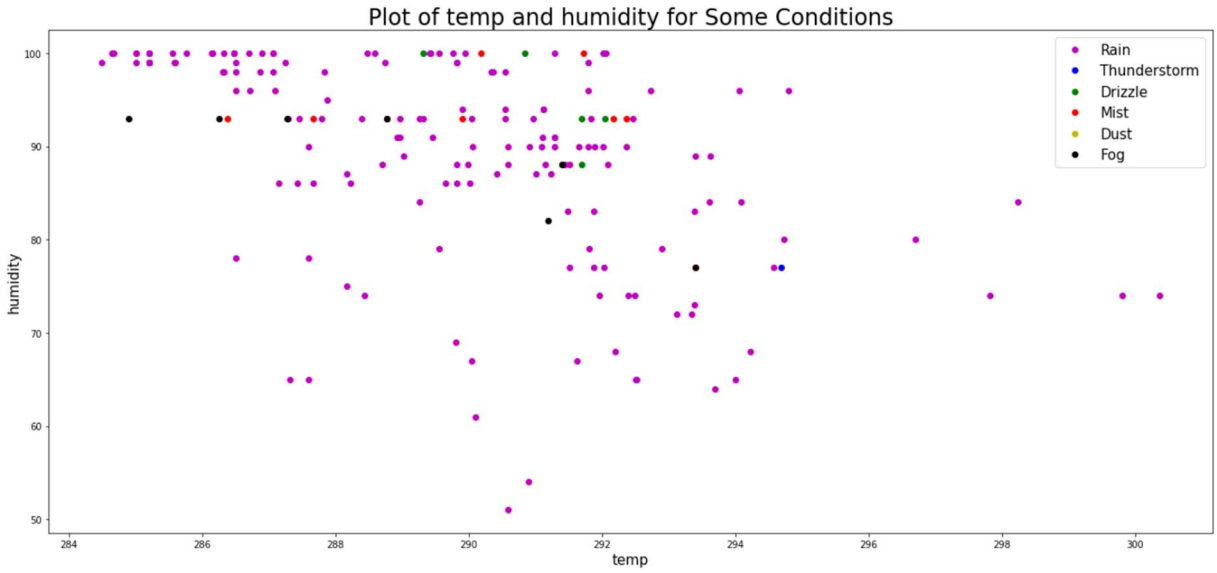
Lasso Regression [Temperature,Pressure,Humidity~Total Actual Load]:



Elastic Regression [Temperature,Pressure,Humidity~Total Actual Load]:



Weather Type Plot:



Future Work

This project afforded the team an excellent opportunity to explore various Big Data tools and Technologies. The concepts of how to store and query the data from the first half of the semester were very straightforward and were relatable to previous knowledge we had gained. The second half of the semester with Spark and the various components of Spark were more complex. The area of Machine Learning and the various types of prediction models is definitely an area the team plans to explore and learn more about. The models we were able to create reflect the "novice" level of understanding of the subject, which is understandable given the amount of exposure the team had to it.

Future work and plans would include to expand on the level of ML knowledge, both via coursework and independent learning. Improve or even create new and different prediction models. Locate other sources of data to be able to build better and more robust models. Explore concepts of semily dissimilar areas being able to predict the behavior of other areas, such as weather and pollution, or movie preferences based upon consumer spending patterns.

Story Telling:

Life

1. **Who** are the people or communities in need of help?

   Consumers of energy utility companies (gas and electricity) are one portion of people who need help, as the recent crisis in Texas has demonstrated. The other people who need help are the producer of the energy being consumed (Utility Companies) and their suppliers. Being able to accurately forecast demand can help them better plan on how much to produce, how much raw materials to keep in stock, and how to better plan to use their resources, including labor, to meet their client's demands. In the case where it is not possible for utility companies to keep up with demand, adequate warning could be given to energy consumers so they can plan for loss of power.

2. **What** problem happened to them?

   Recently in Texas with the severe cold weather, the demand for electricity far exceeded the capacity the Utility Companies were able to provide. In addition, the cold weather caused some producers of gas to halt production. If the Utility Companies could have foreseen the spike in demand, it may have been possible for them to take actions to mitigate the negative impacts which were caused by the shortages in energy as compared to the demand that was required to keep people warm.

3. **When** did the problem take place?

This recent issue occurred in February 2021. There were also rolling blackouts in Kansas City Missouri in February 2021 because of the cold. There have been rolling blackouts in California because of the heat over numerous years.

4. **Where** means two things:

    a. The environment and settings that the people or the community is living in, and

Due to a rapidly changing climate, these problems can affect a wide range of environments.

Communities that are not accustomed to extreme cold or hot weather are particularly vulnerable to energy shortfalls when hit with unexpected demand. Notably in the case of Texas, homes there were designed to shed heat, so when they were hit with uncharacteristically cold weather, the infrastructure was especially unequipped to deal with power outages.

    b. The place/location where the problem takes place.

The problem of energy shortages can occur anywhere, in any country. The recent cases have been in Kansas City Missouri and in Texas. They have occurred in California. They can occur anywhere.

5. **Why** means the possible causes and/origin of the problem.

The inability of the Utility Companies to accurately predict demand and a lack of excess capacity to handle a reduction in production capacity by outside forces results in a failure to meet the demands of the consumer. IT companies have HA and redundancy built into their server farms to help prevent outages of their services. Utility companies need to have the same HA and redundancy built into their systems, and they need to understand and be able to reasonably predict how much demand there will be from their consumers.

6. **How**: If you would like, you can add a dimension of how. How did it happen? Sometimes, the answer to how can be covered by what, when, and where.

In the case of Texas, lawmakers were warned years in advance of the possibility of this situation happening. They ignored recommendations that they winterize their energy infrastructure, and that contributed greatly to the energy shortages as some systems failed. Analysis of energy consumption vs. weather patterns could provide the necessary information to utility companies and lawmakers in advance of disastrous situations and could help hold them accountable for their lack of preparedness.

Workflow:

Gain knowledge and experience with Big Data Programming tools.

Select the problem the team wishes to solve.

Research aspects of the problem, data, tools, why, benefits, issues.

Select the tools the team wishes to use to solve the problem.

[Repeat following steps as necessary]

Refine the data associated with the problem.

Design Model for solving the problem.

Construct Model for solving the problem.

Test/validate Model for solving the problem.

Create a presentation of the solution the team has created.

Project Management:

Implementation Status Report

Task Name	Assigned To	Iteration
Store Weather data set in Hadoop	Joe Goldsich	2
Analyze Weather data set using Hive	Joe Goldsich	2
Analyze Weather data set using MapReduce	Joe Goldsich	2
Store data sets in Hadoop	Anna Johnson	2
Export Data out of Hadoop and into MySQL using Sqoop	Anna Johnson	2
Export Data out of MySQL and into Hadoop using Squoop	Anna Johnson	2
Analyze the data in MySQL	Anna Johnson	2
Store data sets in Hadoop	Kyle Son	2
Analyze Merged data using Hive	Kyle Son	2

Analyze Merged data using Cassandra	Kyle Son	2
Store Energy data set in Hadoop	Bill Yerkes	2
Analyze Energy data set using Hive	Bill Yerkes	2
Analyze Energy data set using MapReduce	Bill Yerkes	2
Convert string type to date-type in datasets	Joe Goldsich	3
Plotting Graphs of Query results on datasets	Joe Goldsich	3
Graphing Madrid's January 2015 actual and predicted energy loads	Joe Goldsich	3
Plot Sevilles' April 2016 weather_main relative to humidity and pressure	Joe Goldsich	3
Altering previous code into more repeatable functions	Joe Goldsich	3
Create KNN classification for Various Column Combinations	Joe Goldsich	3

Implementation Status Report Continued

Task Name	Assigned To	Iteration
Plot the average temp of the city	Kyle Son	3
Plot the count of the city and avg price by city	Kyle Son	3
Plot the graph contain relationship between weather status and average_price	Kyle Son	3
Hourly analysis of average price	Kyle Son	3
Hourly analysis of average load	Kyle Son	3
Hourly analysis of wind speed and humidity	Kyle Son	3
Add Code from Kaggle to Project to review and learn from	Bill Yerkes	3
Figure out how to read Data in Colab directly from GitHub	Bill Yerkes	3
Convert column with string of time to timestamp	Bill Yerkes	3

Create function to Add x hours to timestamp field	Bill Yerkes	3
Create function to add time offset back to timestamp field	Bill Yerkes	3
Implement an LSTM model	Bill Yerkes	3
Implement a Linear regression	Bill Yerkes	3
Implement a Lasso regression	Bill Yerkes	3
Implement a Ridge regression	Bill Yerkes	3
Create Kmeans clustering model and 3D visual	Bill Yerkes	3
Knn analysis of merged data set for energy and weather features	Anna Johnson	3
MapReduce functions for both energy and weather features dataset using RDD's and Pyspark's built-in MapReduce functionality	Anna Johnson	3
Analysis of datasets in Pyspark	Anna Johnson	3
Visualization of data in Pyspark	Anna Johnson	3
Querying energy and weather features datasets in Pyspark to pull useful information to inform our modeling	Anna Johnson	3
Merging the datasets in Pyspark so they could be queried together	Anna Johnson	3
Generalize Graphing weather_main based on two numeric weather conditions with functions	Joe Goldsich	3

Working screens from project:

Anna Johnson:

Create tables in MySQL:

Energy dataset:

```
mysql> create table energy_datas(time VARCHAR(50), biomass FLOAT, lignite FLOAT,
 coal_derived_gas FLOAT, fossil_gas FLOAT, hard_coal FLOAT, fossil_oil FLOAT, oi
l_shale FLOAT, peat FLOAT, geothermal FLOAT, hydro_pumped_agg FLOAT, hydro_pumpe
d_consump FLOAT, hydro_run_of_river FLOAT, hydro_water_res FLOAT, marine FLOAT,
nuclear FLOAT, gen_other FLOAT, gen_other_renew FLOAT, solar FLOAT, waste FLOAT,
 gen_wind_offshore FLOAT, gen_wind_onshore FLOAT, forecast_solar FLOAT, forecast
_wind_offshore FLOAT, forecast_wind_onshore FLOAT, total_load_forecast FLOAT, to
tal_load_actual FLOAT, price_day_ahead FLOAT, price_actual FLOAT);
Query OK, 0 rows affected (0.02 sec)
```

Weather dataset:

```
mysql> create table weather_data(dt_iso VARCHAR(100), city_name VARCHAR(100), te
mp FLOAT, temp_min FLOAT, temp_max FLOAT, pressure INT, humidity INT, wind_speed
 INT, wind_deg INT, rain_1h FLOAT, rain_3h FLOAT, snow_3h FLOAT, clouds_all INT,
 weather_id INT, weather_main VARCHAR(100), description VARCHAR(100), weather_ic
on VARCHAR(20));
Query OK, 0 rows affected (0.04 sec)
```

Load Data into MySQL tables from csv files:

```
mysql> load data local infile '/home/cloudera/Desktop/weather_features.csv' into
 table weather_test
    -> fields terminated by ','
    -> lines terminated by '\n';
Query OK, 178397 rows affected, 12 warnings (2.94 sec)
Records: 178397  Deleted: 0  Skipped: 0  Warnings: 6
```

Run Queries on data in MySQL:

Query on weather dataset to get statistics on weather for each city:

```
mysql> SELECT AVG(temp), MAX(temp), MIN(temp), city_name FROM weather_data GROUP
 BY city_name;
+------------------+-----------+-----------+------------+
| AVG(temp)        | MAX(temp) | MIN(temp) | city_name  |
+------------------+-----------+-----------+------------+
| 289.848244622644 |    309.15 |    262.24 |  Barcelona |
| 286.378488296441 |    312.47 |    266.85 | Bilbao     |
|                0 |         0 |         0 | city_name  |
| 288.061070234593 |    313.33 |   264.132 | Madrid     |
| 293.105430544879 |     315.6 |    271.05 | Seville    |
| 290.780776819068 |    311.15 |   268.831 | Valencia   |
+------------------+-----------+-----------+------------+
6 rows in set (0.94 sec)
```

Queries on energy dataset:

1. Find the average, min, and max for the ‘waste’ column
2. Find times where there was above average waste, or no waste at all
3. Find times where more energy was generated from solar sources than from fossil oil sources

```
mysql> SELECt AVG(waste), MAX(waste), MIN(waste) FROM energy_datas;
+-----------------+------------+------------+
| AVG(waste)      | MAX(waste) | MIN(waste) |
+-----------------+------------+------------+
| 269.298445743619 |        357 |          0 |
+-----------------+------------+------------+
1 row in set (0.04 sec)

mysql> SELECT time, waste FROM energy_datas WHERE waste > 269 OR waste = 0 limit
 10;
+--------------------------+-------+
| time                     | waste |
+--------------------------+-------+
| time                     |     0 |
| 2015-01-05 03:00:00+01:00 |     0 |
| 2015-01-05 12:00:00+01:00 |     0 |
| 2015-01-05 13:00:00+01:00 |     0 |
| 2015-01-05 14:00:00+01:00 |     0 |
| 2015-01-05 15:00:00+01:00 |     0 |
| 2015-01-05 16:00:00+01:00 |     0 |
| 2015-01-05 17:00:00+01:00 |     0 |
| 2015-01-09 00:00:00+01:00 |   273 |
| 2015-01-09 19:00:00+01:00 |   281 |
+--------------------------+-------+
10 rows in set (0.00 sec)

mysql> SELECT time, solar, fossil_oil FROM energy_datas WHERE solar > fossil_oil
 limit 10;
+--------------------------+-------+------------+
| time                     | solar | fossil_oil |
+--------------------------+-------+------------+
| 2015-01-01 09:00:00+01:00 |   743 |        163 |
| 2015-01-01 10:00:00+01:00 |  2019 |        167 |
| 2015-01-01 11:00:00+01:00 |  3197 |        166 |
| 2015-01-01 12:00:00+01:00 |  3885 |        167 |
| 2015-01-01 13:00:00+01:00 |  4007 |        167 |
| 2015-01-01 14:00:00+01:00 |  3973 |        166 |
| 2015-01-01 15:00:00+01:00 |  3818 |        160 |
| 2015-01-01 16:00:00+01:00 |  3088 |        163 |
| 2015-01-01 17:00:00+01:00 |  1467 |        165 |
| 2015-01-01 18:00:00+01:00 |   404 |        164 |
+--------------------------+-------+------------+
10 rows in set (0.00 sec)
```

These queries helped us to gain a better insight into our datasets. We can apply this knowledge in the next steps of our project to identify important features to compare between the energy and weather datasets to determine the relationship between them.

Lastly, it's important that we are able to move these datasets between Hadoop and MySQL so that we can utilize the different functionality of both the Hive and MySQL databases. This can be done using Sqoop.

Exporting and importing tables between Hadoop, Hive, and MySQL:

```
[cloudera@quickstart ~]$ sqoop import --connect jdbc:mysql://localhost/weather -
-username root --password cloudera --table weather_data --m 1
```

```
[cloudera@quickstart ~]$ sqoop export --connect jdbc:mysql://localhost/weather -
-username root --password cloudera --table weather_test --export-dir /user/hive/
warehouse/weather_dbh.db/weather_datah --input-fields-terminated-by ',' --input-
lines-terminated-by '\n' -m 1
```

The MySQL tables in HDFS after importing using sqoop:

```
21/03/22 11:40:45 INFO mapreduce.ImportJobBase: Retrieved 35065 records.
[cloudera@quickstart ~]$ hadoop fs -ls
Found 7 items
drwxr-xr-x   - cloudera cloudera          0 2021-02-25 19:28 acad
drwxr-xr-x   - cloudera cloudera          0 2021-03-22 11:40 energy_datas
drwxr-xr-x   - cloudera cloudera          0 2021-01-27 18:42 johnsona9726
drwxr-xr-x   - cloudera cloudera          0 2021-02-25 17:41 queries
drwxr-xr-x   - cloudera cloudera          0 2021-02-25 16:46 queryresult
drwxr-xr-x   - cloudera cloudera          0 2021-03-21 21:49 weather_data
```

KNN Analysis in Pyspark:

I created the dataframes in Pyspark so that they could be used for KNN modeling.

```
#inferSchema allows different datatypes
energy_df = spark.read.csv('energy_dataset.csv', inferSchema=True, header=True)
energy_df.show(10)

weather_df = spark.read.csv('weather_features.csv', inferSchema=True, header=True)
weather_df.show(10)
```

```
+-------------------+----------------+-----------------------------+--------------------
|               time|generation biomass|generation fossil brown coal/lignite|generation fossil co
+-------------------+----------------+-----------------------------+--------------------
|2015-01-01 00:00:...|           447.0|                        329.0|
|2015-01-01 01:00:...|           449.0|                        328.0|
|2015-01-01 02:00:...|           448.0|                        323.0|
|2015-01-01 03:00:...|           438.0|                        254.0|
|2015-01-01 04:00:...|           428.0|                        187.0|
|2015-01-01 05:00:...|           410.0|                        178.0|
|2015-01-01 06:00:...|           401.0|                        172.0|
|2015-01-01 07:00:...|           408.0|                        172.0|
|2015-01-01 08:00:...|           413.0|                        177.0|
|2015-01-01 09:00:...|           419.0|                        177.0|
+-------------------+----------------+-----------------------------+--------------------
only showing top 10 rows

+-------------------+---------+------------------+------------------+------------------+--------+
|             dt_iso|city_name|              temp|          temp_min|          temp_max|pressure|
+-------------------+---------+------------------+------------------+------------------+--------+
|2015-01-01 00:00:...| Valencia|           270.475|           270.475|           270.475|    1001|
|2015-01-01 01:00:...| Valencia|           270.475|           270.475|           270.475|    1001|
|2015-01-01 02:00:...| Valencia|           269.686|           269.686|           269.686|    1002|
|2015-01-01 03:00:...| Valencia|           269.686|           269.686|           269.686|    1002|
|2015-01-01 04:00:...| Valencia|           269.686|           269.686|           269.686|    1002|
|2015-01-01 05:00:...| Valencia|270.29200000000003|270.29200000000003|270.29200000000003|    1004|
|2015-01-01 06:00:...| Valencia|270.29200000000003|270.29200000000003|270.29200000000003|    1004|
|2015-01-01 07:00:...| Valencia|270.29200000000003|270.29200000000003|270.29200000000003|    1004|
|2015-01-01 08:00:...| Valencia|           274.601|           274.601|           274.601|    1005|
|2015-01-01 09:00:...| Valencia|           274.601|           274.601|           274.601|    1005|
+-------------------+---------+------------------+------------------+------------------+--------+
only showing top 10 rows
```

we can see, the two dataframes have similar entries for the 'dt_iso' and 'time' columns. We want to join the en
lumn so that we can look for correlations between the two datasets.

```
combined_df = weather_df.join(energy_df, weather_df.dt_iso == energy_df.time, 'inner')
combined_df.show(10)
```

Then I used KNN analysis to determine the relationships between various features in the weather and energy datasets. Higher accuracy scores from the model correspond to a stronger correlation between the features.

An example of one of the models:

```
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

# drop rows that contain null values for the selected columns
knn_df = combined_df.select('generation solar', 'generation wind onshore', 'generation wind offshore', 'weather_main')
knn_df_drop = knn_df.dropna()

#create feature and target arrays
knn_data = np.array(knn_df_drop.select('generation solar', 'generation wind onshore', 'generation wind offshore').collect())
knn_target = np.array(knn_df_drop.select('weather_main').collect())

#use np.ravel() to convert the target array to the proper format
knn_target = np.ravel(knn_target)

#split into train and test sets
X_train, X_test, y_train, y_test = train_test_split(knn_data, knn_target, test_size=0.2, random_state=42)

#create the model
knn = KNeighborsClassifier(n_neighbors = 7)
knn.fit(X_train, y_train)

#test the model
print(knn.predict(X_test))

#get the accuracy score for the model
print(knn.score(X_test, y_test))
```

```
['clear' 'clouds' 'clouds' ... 'clouds' 'clouds' 'clear']
0.4859794739498626
```

```
# drop rows that contain null values for the selected columns
knn_df = combined_df.select('price actual', 'generation waste', 'total load actual', 'weather_main')
knn_df_drop = knn_df.dropna()

#create feature and target arrays
knn_data = np.array(knn_df_drop.select('price actual', 'generation waste', 'total load actual').filter((weather_df.city_name == 'Madrid') & (weather_df.dt_iso.like('2017-11%'))).collect())
knn_target = np.array(knn_df_drop.select('weather_main').filter((weather_df.city_name == 'Madrid') & (weather_df.dt_iso.like('2017-11%'))).collect())

#use np.ravel() to convert the target array to the proper format
knn_target = np.ravel(knn_target)

#split into train and test sets
X_train, X_test, y_train, y_test = train_test_split(knn_data, knn_target, test_size=0.2, random_state=42)

#create the model
knn = KNeighborsClassifier(n_neighbors = 7)
knn.fit(X_train, y_train)

#test the model
print(knn.predict(X_test))

#get the accuracy score for the model
print(knn.score(X_test, y_test))
```

```
['clear' 'clear' 'clear' 'clear' 'clear' 'clear' 'clear' 'clear' 'clear'
 'clear' 'clear' 'clear' 'clear' 'clear' 'clear' 'clear' 'clear' 'clear'
 'clear' 'clear' 'clear' 'clear' 'clear' 'clear' 'clear' 'clear' 'clear'
 'clear' 'clear' 'clear' 'clear' 'clear' 'clear' 'clear' 'clouds' 'clear'
 'clear' 'clouds' 'clear' 'clear' 'clear' 'clear' 'clear' 'clear' 'clear'
 'clouds' 'clear' 'clear' 'clear' 'clear' 'clear' 'clear' 'clear' 'clear'
 'clear' 'clouds' 'clear' 'clouds' 'clear' 'clouds' 'clear' 'clear'
 'clear' 'clear' 'clear' 'clear' 'clear' 'clear' 'clear' 'clear' 'clear'
 'clear' 'clear' 'clouds' 'clear' 'clear' 'clear' 'clear' 'clear' 'clear'
 'clear' 'clear' 'clear' 'clear' 'clear' 'clear' 'clear' 'clear' 'clear'
 'clear' 'clear' 'clear' 'clear' 'clear' 'clear' 'clear' 'clear' 'clouds'
 'clear' 'clouds' 'clear' 'clear' 'clear' 'clear' 'clear' 'clear' 'clear'
 'clear' 'clear' 'clear' 'clear' 'clear' 'clear' 'clear' 'clear' 'clear'
 'clear' 'clear' 'clear' 'clouds' 'clear' 'clear' 'clear' 'clear' 'clear'
 'clouds' 'clear' 'clear' 'clear' 'clear' 'clear' 'clear' 'clear' 'clear'
 'clear' 'clear' 'clear' 'clear' 'clear' 'clear' 'clear' 'clear' 'clear'
 'clear' 'clear' 'clear']
0.5958904109589042
```

The KNN analysis showed that there was in fact a correlation between some of the columns in these two datasets. The code and output for the rest of the models can be found in the github repository for the project.

MapReduce:

I wanted to utilize the power of MapReduce functions to run several queries on our large datasets. I ran similar queries to the ones that I worked on in MySQL for increment 2, however I was able to work with larger datasets at much faster speeds by making the dataframe into RDD's and utilizing Pyspark's built-in MapReduce functionality.

Some examples of queries:

Some pyspark queries using map-reduce:

```
[ ] print("Average temperature by city:\n")
    weather_df.rdd.map(lambda x: (x[1], x[3])) \
    .mapValues(lambda x: (x, 1)) \
    .reduceByKey(lambda x, y: (x[0]+y[0], x[1]+y[1])) \
    .mapValues(lambda x: x[0]/x[1]) \
    .take(5)
```

```
Average temperature by city:

[('Bilbao', 284.91666115356213),
 (' Barcelona', 288.59470412145697),
 ('Valencia', 290.2222765464678),
 ('Madrid', 286.82487717417484),
 ('Seville', 291.1841030547889)]
```

```
[ ] weather_df.filter(weather_df['snow_3h'] > 0).groupBy(weather_df['city_name']).count().take(5)
```

```
[Row(city_name='Madrid', count=1),
 Row(city_name='Bilbao', count=257),
 Row(city_name='Valencia', count=9)]
```

```
weather_df.filter(weather_df['weather_main'] == 'clear').groupBy(weather_df['city_name']).count().take(5)
```

```
[Row(city_name='Madrid', count=20356),
 Row(city_name='Seville', count=23581),
 Row(city_name=' Barcelona', count=14760),
 Row(city_name='Bilbao', count=8453),
 Row(city_name='Valencia', count=15535)]
```

```
[ ] print("Average pressure by city:\n")
    weather_df.rdd.map(lambda x: (x[1], x[5])) \
    .mapValues(lambda x: (x, 1)) \
    .reduceByKey(lambda x, y: (x[0]+y[0], x[1],y[1])) \
    .mapValues(lambda x: x[0]/x[1]) \
    .take(5)
```

```
Average pressure by city:

[('Bilbao', 36582567.0),
 (' Barcelona', 45551556.0),
 ('Valencia', 35706399.0),
 ('Madrid', 36696345.0),
 ('Seville', 36214972.0)]
```

```
[ ] print("Average weather id by city: \n")
    weather_df.rdd.map(lambda x: (x[1], x[13])) \
    .mapValues(lambda x: (x, 1)) \
    .reduceByKey(lambda x, y:(x[0]+y[0], x[1]+y[1])) \
    .mapValues(lambda x: x[0]/x[1]) \
    .take(5)
```

```
Average weather id by city:
```

```
Average snow_3h per city:

[('Bilbao', 0.02345514728380295),
 (' Barcelona', 0.0),
 ('Valencia', 0.0001536491677336747),
 ('Madrid', 2.8924366503984343e-05),
 ('Seville', 0.0)]
```

```
from pyspark.sql.functions import *
energy_df.agg(min(col("price actual")), max(col("price actual")), min(col("price day ahead")), max(col("price day ahead"))).show()
```

```
+----------------+----------------+-----------------------+-----------------------+
|min(price actual)|max(price actual)|min(price day ahead)|max(price day ahead)|
+----------------+----------------+-----------------------+-----------------------+
|            9.33|           116.8|                   2.06|                 101.99|
+----------------+----------------+-----------------------+-----------------------+
```

```
energy_df.agg(min(col("generation solar")), max(col("generation solar")), min(col("generation wind onshore")), \
              max(col("generation wind onshore")), min(col("generation nuclear")), max(col("generation nuclear")), \
              min(col("generation other renewable")), max(col("generation other renewable"))).show()
```

```
+--------------------+--------------------+-----------------------------+-----------------------------+-----------------------+-----------------------+-----------------------------+-----------------------------+
|min(generation solar)|max(generation solar)|min(generation wind onshore)|max(generation wind onshore)|min(generation nuclear)|max(generation nuclear)|min(generation other renewable)|max(generation other renewable)|
+--------------------+--------------------+-----------------------------+-----------------------------+-----------------------+-----------------------+-----------------------------+-----------------------------+
|                 0.0|              5792.0|                          0.0|                      17436.0|                    0.0|                 7117.0|                          0.0|                        119.0|
+--------------------+--------------------+-----------------------------+-----------------------------+-----------------------+-----------------------+-----------------------------+-----------------------------+
```

This was useful as it allowed us to get a better idea of what variables we should use in our modeling, as well as what kind of models to build.

Joe Goldsich:

An example of a simple query on a join of the two data sets using HiveQL:

```
SELECT w.city_name, AVG(e.total_load_actual)
        FROM weather AS w
        INNER JOIN energy AS e
        ON w.dt_iso = e.time
        WHERE w.temp > 393.15 AND w.temp < 398.15 /*Between 20 and 25C (68 - 77)*/
        GROUP BY w.city_name;
```

And the results of this simple query:

```
Total MapReduce CPU Time Spent: 8 seconds 150 ms
OK
 Barcelona      28193.09695734459
Bilbao  30800.297554697554
Madrid  28273.834212840808
Seville 28026.180286310766
Valencia        28202.34704
Time taken: 90.513 seconds, Fetched: 5 row(s)
hive>
```

With the exception of Bilbao (which has a very temperate climate) most of the cities experienced an increase in their energy usage at more extreme temperatures (below 10C and over 30C):

```
Total MapReduce CPU Time Spent: 8 seconds 920 msec
OK
 Barcelona      28316.030604196247
Bilbao  28286.30073713927
Madrid  29125.597647667055
Seville 29662.865883807168
Valencia        28404.889145496534
Time taken: 89.553 seconds, Fetched: 5 row(s)
hive>
```

A slightly more complicated Select query was very slow to run (10 minutes slow).

```
SELECT *
    FROM (SELECT w.city_name
            FROM weather AS w
            INNER JOIN energy AS e
            ON w.dt_iso = e.time
            WHERE w.temp < 383.15 OR w.temp > 303.15
            GROUP BY w.city_name)
            AS t1
    INNER JOIN (SELECT w.city_name
            FROM weather AS w
            INNER JOIN energy AS e
            ON w.dt_iso = e.time
            WHERE w.temp > 393.15 AND w.temp < 398.15
            GROUP BY w.city_name)
            AS t2
    ON t1.city_name = t2.city_name
    GROUP BY t1.city_name;
```

Creating and loading partitions into a new table for the weather dataset to look for performance gains:

```
hive> CREATE TABLE weather_partition(
dt_iso STRING,
city_name STRING,
temp FLOAT,
temp_min FLOAT,
temp_max FLOAT,
pressure INT,
humidity INT,
wind_speed INT,
wind_deg INT,
rain_1h FLOAT,
rain_3h FLOAT,
snow_3h FLOAT,
clouds_all INT,
weather_id INT,
weather_main STRING,
weather_description STRING,
weather_icon STRING)
PARTITIONED BY(name STRING)
row format delimited fields terminated
by ',' stored as textfile;
```

```
hive> ALTER TABLE weather_partition ADD PARTITION (name='July');

INSERT OVERWRITE TABLE weather_partition PARTITION (name='July') SELECT * FROM weather WHERE
month(dt_iso) = 7;
```

The result of all the partitions:

```
hive> SHOW PARTITIONS weather_partition;
OK
name=April
name=August
name=Barcelona
name=Bilbao
name=Bilboa
name=December
name=February
name=January
name=July
name=June
name=Madrid
name=March
name=May
name=November
name=October
name=September
name=Seville
name=Valencia
Time taken: 0.186 seconds, Fetched: 18 row(s)
hive>
```

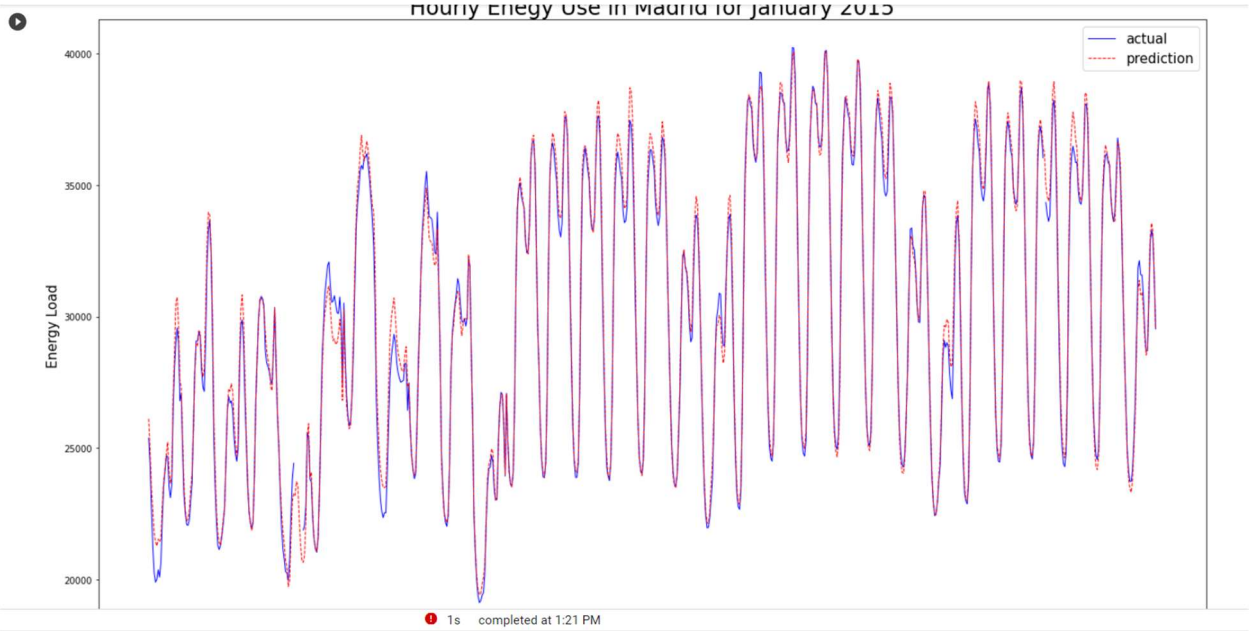
And some queries using the partitions and not using them. This did result in significantly quicker query times (~58seconds down from ~99 seconds).

```
SELECT AVG(temp) FROM weather_partition WHERE city_name = 'Valencia' AND month(dt_iso) = 11;

SELECT AVG(temp) FROM weather_partition WHERE name = 'Valencia' AND month(dt_iso) = 11;
```

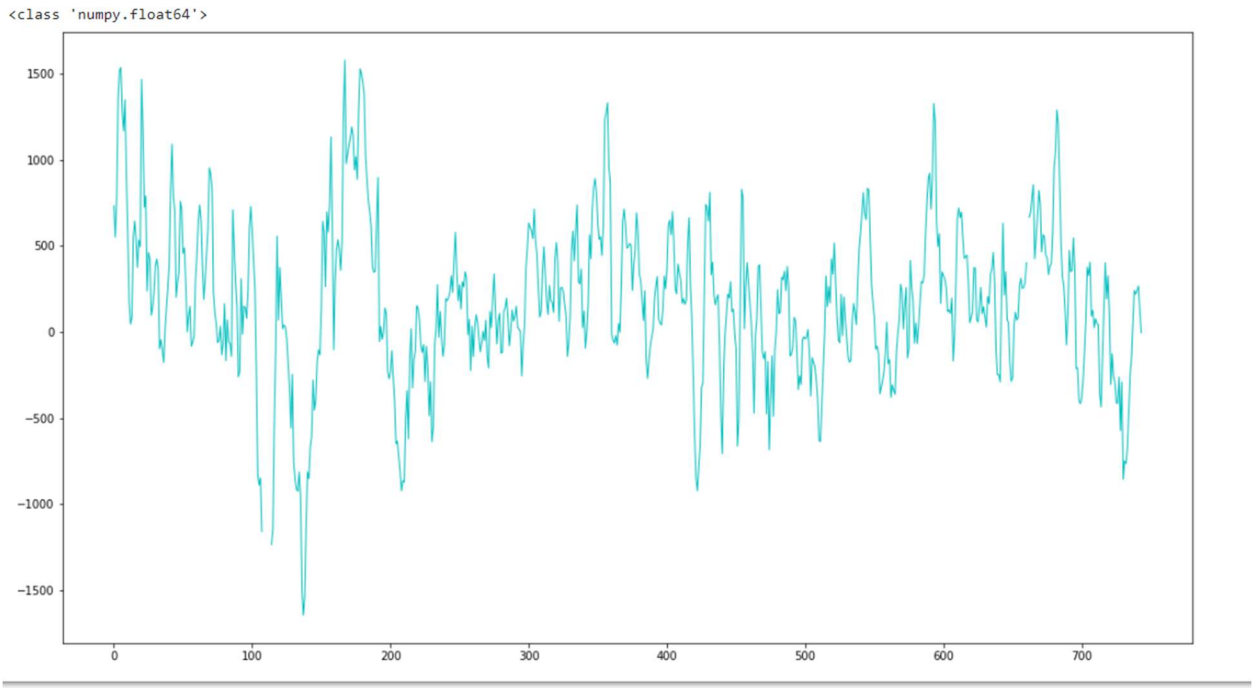
Once we moved onto the portion of the course that taught us Spark. I found using Dataframes was the most efficient way to manipulate the data that we had. The data was visualized with the use of the 'matplotlib.pyplot' library. I would primarily utilize Google Colabs for the rest of the work.

For our goal of trying to better predict energy usage based on weather information, I thought it would be good to get a baseline from which to compare the efficacy of our predictions. I created a subset of the total joined dataset for Madrid in January of 2015 (This period was chosen randomly). I then charted the actual energy load with the forecasted energy load.

```
[10] #collect the data from one column
     #timelist = df_energy_weather.select("time").collectAsList()

     # Get Madrid's actual power usage and time for the month of January 2015
     df_madrid_total_load = df_energy_weather.select("total load actual", "dt_iso", "total load forecast") \
                                             .filter((df_w.city_name == 'Madrid') \
                                             & (df_w.dt_iso.like('2015-01%')))

     # Convert the data in the new DF to values in a list for future graph plot.
     madrid_january_energy = list(df_madrid_total_load.select('total load actual').toPandas()['total load actual'])

     # Convert the data in the new DF to values in a list for future graph plot.
     madrid_january_forecast = list(df_madrid_total_load.select('total load forecast').toPandas()['total load forecast'])


     # Convert the data in the new DF to values in a list for future graph plot.
     madrid_january_time = list(df_madrid_total_load.select('dt_iso').toPandas()['dt_iso'])

     ## Get the number of hours available.  Not sure this is necessary
     mad_w_arr = array('I')
     for i in range(len(madrid_january_time)):
       mad_w_arr.append(i)

     plt.figure(figsize=(18,10))              ## Figure(figsize) changes the size of the output graph
     plt.plot(mad_w_arr, madrid_january_energy, 'b-', linewidth=1, markersize=12, label='actual')  ## Plot the actual energy use
     plt.plot(mad_w_arr, madrid_january_forecast, 'r--', linewidth=1, markersize=12, label='prediction')  ## Plot the forecast energy use
     plt.tight_layout()
     plt.title('Hourly Enegy Use in Madrid for January 2015', size=24)
     plt.ylabel('Energy Load', size=15)
     plt.xlabel('Hours in January', size=15)
     plt.legend(fontsize=15)
     plt.show()
```



Then I charted the difference.



Ideally, we would want to flatten that line.

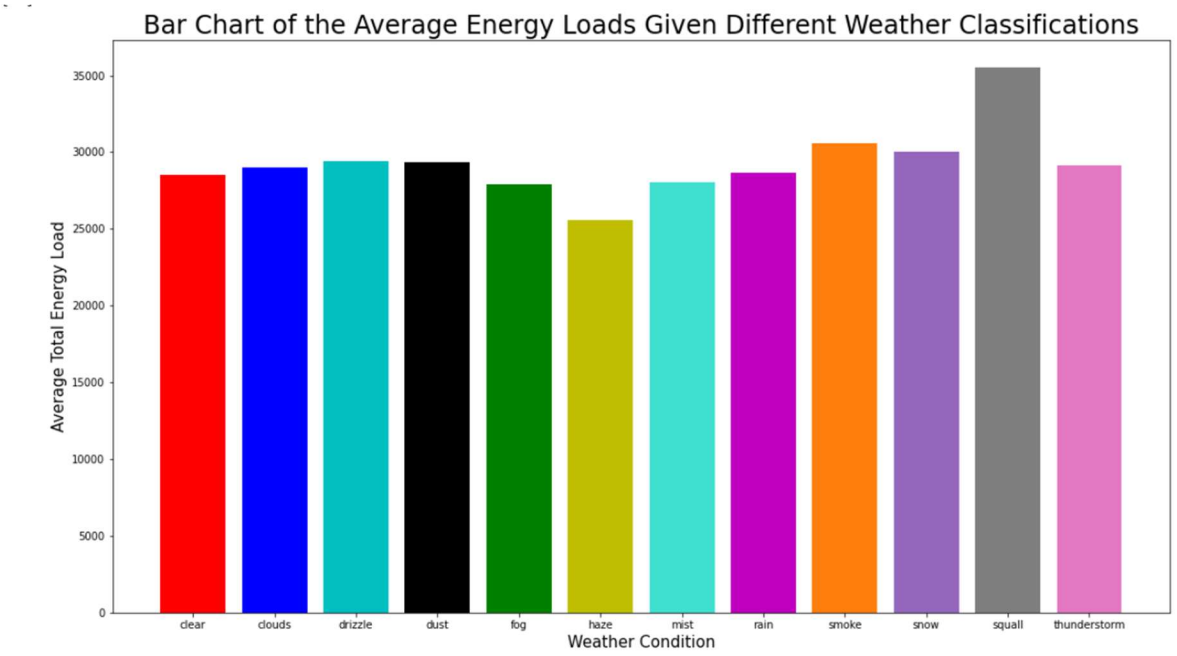
I compared total energy load with weather conditions. The column called ‘weather_main’ in the data used a verbal description to classify the weather each hour. The possibilities were: clear, rain, cloudy, thunderstorm, drizzle, mist, dust, and fog. I created a bar graph to show how each condition related to energy load.

```
example_df = df_energy_weather.groupBy('weather_main').avg('total load actual').orderBy('df_w.weather_main')
example_df.show()
```

```
+------------+----------------------+
|weather_main|avg(total load actual)|
+------------+----------------------+
|       clear|    28522.895244091436|
|      clouds|    28989.780124113997|
|     drizzle|    29405.969257540604|
|        dust|    29365.109510086455|
|         fog|    27938.253391859536|
|        haze|    25578.075862068967|
|        mist|     28038.32702149437|
|        rain|    28673.870887826237|
|       smoke|     30588.78787878788|
|        snow|    30062.196296296297|
|      squall|               35513.0|
|thunderstorm|    29115.577329490876|
+------------+----------------------+
```

```
[23] temp_main_list = list(example_df.select('weather_main').toPandas()['weather_main'])
     avg_energy_total_load_list = list(example_df.select('avg(total load actual)').toPandas()['avg(total load actual)'])
```

```
[24]
     plt.figure(figsize=(18,10))
     plt.title('Bar Chart of the Average Energy Loads Given Different Weather Classifications', size=24)
     plt.ylabel('Average Total Energy Load', size=15)
     plt.xlabel('Weather Condition', size=15)
     plt.bar(temp_main_list, avg_energy_total_load_list, color=['r','b','c','k','g','y','turquoise','m','tab:orange','tab:purple','tab:gray','tab:pink'])
```



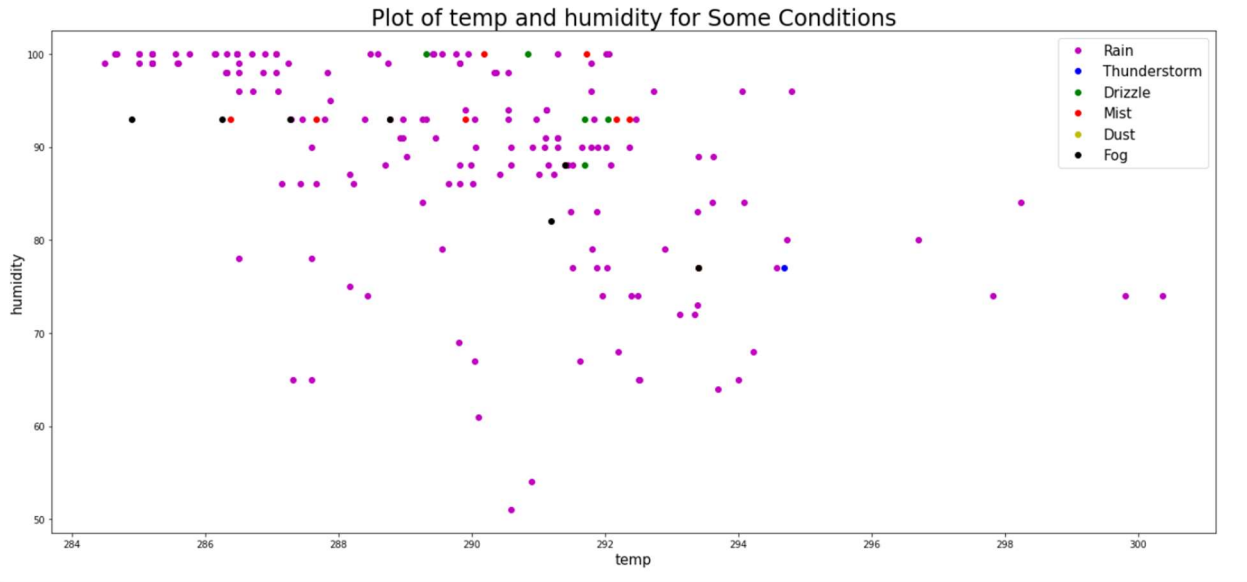
I thought these results indicated that this seemed a promising avenue to pursue. So, next I wanted to see if there was any kind of correlation between the numerically measured weather variables in our data set. So I wrote a couple of functions that together would allow us to print out a scatter plot of the weather condition (from 'weather_main') based on two of the numerical columns in the set ('pressure', 'temp', 'humidity', 'wind_speed', 'wind_deg').

```
def get_condition_list(weather_condition, weather_type, weather_df):

  # Check for proper condition input.
  condition_list = ['temp', 'pressure', 'humidity', 'wind_speed', 'wind_deg']
  if (condition_list.count(weather_condition) == 0):
    print("Improper weather condition passed.")
    print("Condition must be: temp, pressure, humidity, wind_speed, wind_deg")
    return

  # Check for proper weather type
  weather_list = ['clear', 'cloudy', 'rain', 'fog', 'thunderstorm', 'drizzle',
                  'mist', 'dust']
  if (weather_list.count(weather_type) == 0):
    print("Improper weather type passed.")
    print("Weather must be: clear, cloudy, rain, drizzle, fog, mist, dust, or thunderstorm.")
    return

  # Return Create a list of weather conditions when the weather type is present in the DF.
  if isinstance(weather_df, pyspark.sql.dataframe.DataFrame):
    list_for_return = list(weather_df.select(weather_condition) \
                                     .filter(df_w.weather_main == weather_type) \
                                     .toPandas()[weather_condition])
    return list_for_return
  else:
    ## weather_df is not of type dataframe
    print("Must pass a dataframe type")
    return
```

```
def graph_weather_type(condition1, condition2, weather_df):

  # Check for proper condition input.
  condition_list = ['temp', 'pressure', 'humidity', 'wind_speed', 'wind_deg']
  if (condition_list.count(condition1) == 0) or (condition_list.count(condition2) == 0):
    print("Improper weather condition passed.")
    print("Condition must be: temp, pressure, humidity, wind_speed, wind_deg")
    return

  # Use helper condition to get lists of the needed data.
  rain_cond1 = get_condition_list(condition1, 'rain', weather_df)
  rain_cond2 = get_condition_list(condition2, 'rain', weather_df)

  thunderstorm_cond1 = get_condition_list(condition1, 'thunderstorm', weather_df)
  thunderstorm_cond2 = get_condition_list(condition2, 'thunderstorm', weather_df)

  fog_cond1 = get_condition_list(condition1, 'fog', weather_df)
  fog_cond2 = get_condition_list(condition2, 'fog', weather_df)

  drizzle_cond1 = get_condition_list(condition1, 'drizzle', weather_df)
  drizzle_cond2 = get_condition_list(condition2, 'drizzle', weather_df)

  mist_cond1 = get_condition_list(condition1, 'mist', weather_df)
  mist_cond2 = get_condition_list(condition2, 'mist', weather_df)

  dust_cond1 = get_condition_list(condition1, 'dust', weather_df)
  dust_cond2 = get_condition_list(condition2, 'dust', weather_df)

  # Create the graph
  plt.figure(figsize=(18,10))             ## Figure(figsize) changes the size of the output graph
  #plt.plot(clear_pressure, clear_humidity, 'bx', linewidth=1, markersize=12, label='Clear')
  plt.plot(rain_cond1, rain_cond2, 'm.', linewidth=1, markersize=12, label='Rain')
  plt.plot(thunderstorm_cond1, thunderstorm_cond2, 'b.', linewidth=1, markersize=12, label='Thunderstorm')
```

These two allow for the following:

```
# Grab a dataFrame for testing: Seville in April of 2016
df_weather_main = df_energy_weather.select("temp", "pressure", "weather_main") \
                                   .filter((df_w.city_name == 'Bilbao') \
                                   & (df_w.dt_iso.like('2016-06%')))

graph_weather_type('temp', 'pressure', df_weather_main)
```



While this looked promising, a more thorough analysis with knn functions indicated these were poor predictors of the weather condition.

```
# Printing for # of neigbors between 1 and 10
for i in range(1,10):
  knn = KNeighborsClassifier(n_neighbors=i)
  knn.fit(X_train, y_train)
  print(knn.score(X_test, y_test))
```

```
0.5337837837837838
0.5337837837837838
0.49324324324324326
0.527027027027027
0.5540540540540541
0.527027027027027
0.5472972972972973
0.527027027027027
0.5472972972972973
```

Given more time I would like to explore some of the newer machine learning algorithms that we learned near the end of the course. I think that a classification algorithm like Naive Bayes may be better at predicting the weather, and then in turn make for better energy use predictions.

Kyle Son:

Hive(Beeline)

In hive, I used beeline command for the better visualization of the table then, I merged two tables to named merged

```
jdbc:hive2://> create table merged as select * from (select * from Energy e  left join Weather w on e.time = w.dt_iso  )x;
```

Displays avg temp, windspeed and humidity group by city name

```
0: jdbc:hive2://> SELECT city_name, AVG(temp) as AvgTemp, AVG(wind_speed) as Avg
WindSpeed, AVG(humidity) as AvgHumidity from merged group by city_name;
```

```
+-------------+--------------------+--------------------+---------------------+--+
|  city_name  |      avgtemp       |    avgwindspeed    |     avghumidity     |
+-------------+--------------------+--------------------+---------------------+--+
|  Barcelona  | 289.8482446226438  | 2.786588115909347  | 73.99422144548427   |
| Bilbao      | 286.3784882964413  | 1.9574698895719174 | 79.08945509165252   |
| Madrid      | 288.0610702345934  | 2.4416963079383462 | 59.776932197314366  |
| Seville     | 293.1054305448794  | 2.4837865961695305 | 64.14073178277133   |
| Valencia    | 290.7807768190682  | 2.6928154787309717 | 65.14511310285958   |
+-------------+--------------------+--------------------+---------------------+--+
5 rows selected (27.37 seconds)
```

Show the chronological change of the price group by city name

```
0: jdbc:hive2://> SELECT x.city_name AS city_name, x.year AS year, AVG(x.price_actual) AS avg_price_actual, AVG(x.price_day_ahead) AS a
vg_price_ahead
. . . . . > FROM (SELECT city_name, YEAR(time) AS YEAR, price_actual, price_day_ahead FROM merged) AS x
. . . . . > GROUP BY city_name, year;
```

```
+-------------+-------+---------------------+---------------------+--+
|  city_name  | year  |  avg_price_actual   |   avg_price_ahead   |
+-------------+-------+---------------------+---------------------+--+
|  Barcelona  | 2015  | 61.37339986352105   | 50.34858796293164   |
|  Barcelona  | 2016  | 47.408718672313334  | 39.706185255424245  |
|  Barcelona  | 2017  | 59.336550360126544  | 52.264069580019545  |
|  Barcelona  | 2018  | 63.41520793983889   | 57.24723606680608   |
| Bilbao      | 2015  | 61.43668094638449   | 50.40329749004072   |
| Bilbao      | 2016  | 47.504653980321336  | 39.73472841074056   |
| Bilbao      | 2017  | 59.43972139166336   | 52.22414713690373   |
| Bilbao      | 2018  | 63.37481516512984   | 57.217914237181084  |
| Madrid      | 2015  | 61.3486871444867    | 50.32945377248953   |
| Madrid      | 2016  | 47.70215140951364   | 39.848420415028215  |
| Madrid      | 2017  | 59.489210907216496  | 52.3455380426181    |
| Madrid      | 2018  | 63.475907996585576  | 57.38593137540892   |
| Seville     | 2015  | 61.34763784962553   | 50.3336542601593    |
| Seville     | 2016  | 47.44788847544184   | 39.619593737169     |
| Seville     | 2017  | 59.30196663987525   | 52.225143714592235  |
| Seville     | 2018  | 63.368107600977105  | 57.221872207268525  |
| Valencia    | 2015  | 61.36173105451796   | 50.33310753950027   |
| Valencia    | 2016  | 47.43544185960827   | 39.68283255650655   |
| Valencia    | 2017  | 59.32627164795927   | 52.24235188971896   |
| Valencia    | 2018  | 63.45932222534277   | 57.31257539540139   |
+-------------+-------+---------------------+---------------------+--+
20 rows selected (22.771 seconds)
```

Cassandra

Perform some queries in cassandra, There is a limitation for our project because joining the table is not possible in cassandra

```
cqlsh:bigdataproject> select time, total_load_forecast, total_load_actual, price_day_ahead, price_actual from energy where price_
actual >70 LIMIT 10 ALLOW FILTERING;

 time                             | total_load_forecast | total_load_actual | price_day_ahead | price_actual
----------------------------------+---------------------+-------------------+-----------------+--------------
 2015-01-08 19:00:00.000000+0000 |               28124 |             27507 |           52.04 |        90.87
 2015-08-24 12:00:00.000000+0000 |               31216 |             30648 |           62.47 |        70.38
 2015-01-06 21:00:00.000000+0000 |               29307 |             30120 |              57 |        81.65
 2015-02-12 16:00:00.000000+0000 |               32620 |             32607 |            66.4 |        80.05
 2015-03-06 11:00:00.000000+0000 |               32292 |             32003 |           60.21 |        70.29
 2015-07-21 01:00:00.000000+0000 |               32205 |             32737 |              63 |        75.78
 2015-02-05 19:00:00.000000+0000 |               23765 |             24216 |           32.08 |        75.71
 2015-12-23 10:00:00.000000+0000 |               33123 |             32680 |           64.54 |        71.72
 2015-08-03 12:00:00.000000+0000 |               25938 |             25789 |           38.17 |        77.98
 2015-06-27 15:00:00.000000+0000 |               31866 |             31970 |              65 |        74.87
```

```
cqlsh:bigdataproject> SELECT * FROM weather where temp > 290 AND city_name = 'Seville' ALLOW FILTERING;

 dt_iso                           | city_name | clouds_all | humidity | pressure | rain_1h | rain_3h | snow_3h | temp      | temp_
max  | temp_min  | weather_description | weather_icon | weather_id | weather_main | wind_deg | wind_speed
----------------------------------+-----------+------------+----------+----------+---------+---------+---------+-----------+------
----+-----------+---------------------+--------------+------------+--------------+----------+------------
 2018-05-31 12:00:00.000000+0000 |   Seville |          0 |       68 |     1019 |       0 |       0 |       0 | 290.54001 | 291.1
4999 | 290.14999 |        sky is clear |          01d |        800 |        clear |       30 |               2
 2018-07-07 17:00:00.000000+0000 |   Seville |          0 |       26 |     1015 |       0 |       0 |       0 | 306.54001 | 307.1
4999 | 306.14999 |        sky is clear |          01d |        800 |        clear |      310 |               2
 2017-05-30 13:00:00.000000+0000 |   Seville |         20 |       73 |     1016 |       0 |       0 |       0 | 294.54001 | 295.1
4999 | 294.14999 |         few clouds |          02d |        801 |       clouds |       20 |               0
 2016-11-15 10:00:00.000000+0000 |   Seville |          0 |       81 |     1024 |       0 |       0 |       0 |    290.06 | 300.1
4999 | 282.14999 |        sky is clear |          01d |        800 |        clear |       70 |               4
 2016-06-12 07:00:00.000000+0000 |   Seville |          8 |       70 |     1023 |       0 |       0 |       0 |   290.289 |   290
.289 |   290.289 |        sky is clear |          02n |        800 |        clear |        7 |               1
 2017-04-23 15:00:00.000000+0000 |   Seville |          0 |       43 |     1016 |       0 |       0 |       0 |    300.22 | 309.1
4999 | 294.14999 |        sky is clear |          01d |        800 |        clear |      156 |               1
 2018-08-27 00:00:00.000000+0000 |   Seville |          0 |       33 |     1011 |       0 |       0 |       0 | 305.14999 | 305.1
4999 | 305.14999 |        sky is clear |          01n |        800 |        clear |      230 |               3
 2016-03-05 18:00:00.000000+0000 |   Seville |          0 |       36 |     1014 |       0 |       0 |       0 |    292.44 | 298.1
4999 | 289.14999 |        sky is clear |          01d |        800 |        clear |      320 |               5
 2018-10-02 07:00:00.000000+0000 |   Seville |         40 |       83 |     1017 |       0 |       0 |       0 |    292.94 | 294.1
4999 | 292.14999 |    scattered clouds |          03n |        802 |       clouds |       30 |               2
 2017-08-30 13:00:00.000000+0000 |   Seville |         20 |       77 |     1016 |       0 |       0 |       0 | 293.54001 | 294.1
```

Spark(Scala)

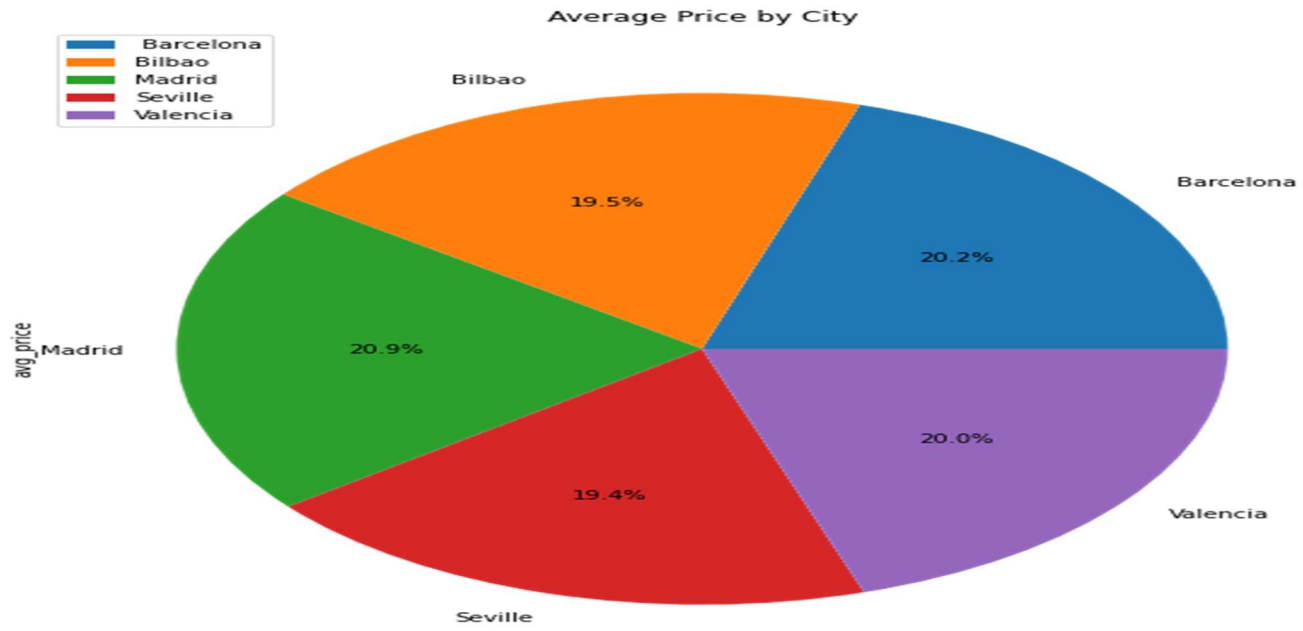
Load the csv file in the spark in the intellij, I merged two table by using spark sql join function and then do some queries based on joined table.

```
object GroupProject {

  def main(args: Array[String]): Unit ={


    val spark: SparkSession = SparkSession.builder()
      .master( master = "local[*]")
      .appName( name = "groupProject")
      .getOrCreate()

    val filepath1 = "energy_dataset.csv"
    val filepath2 = "weather_features.csv"
    val df_energy = spark.read.options(Map("inferSchema"->"true","sep"->",","header"->"true")).csv(filepath1)
    val df_weather = spark.read.options(Map("inferSchema"->"true","sep"->",","header"->"true")).csv(filepath2)

    val df_merge = df_energy.join(df_weather, df_energy("time") === df_weather("dt_iso"), joinType = "inner")
    df_merge.show()

    val df_weatherby = df_merge.groupBy( col1 = "weather_description")
      .avg( colNames = "price actual", "price day ahead", "total load actual")
    df_weatherby.show()
```

```
+--------------------+------------------+--------------------+----------------------+
| weather_description| avg(price actual)|avg(price day ahead)|avg(total load actual)|
+--------------------+------------------+--------------------+----------------------+
|                 fog| 61.98602154828406|   51.95822426177173|    27938.253391859536|
|             drizzle| 56.84691056910567|   51.23531165311652|    29612.173441734416|
|     very heavy rain|  55.3823076923077|  46.016666666666666|     27669.25641025641|
|  ragged shower rain|             68.61|                50.6|               26513.0|
|proximity shower ...|56.882668067226895|   52.44361344537816|    30394.945263157893|
|  light thunderstorm|            58.455|               52.36|               28351.5|
|          few clouds| 57.73782467835898|  50.286180181303074|     29256.7720938932|
|heavy intensity s...| 57.78037037037037|   53.74283950617283|     29858.0987654321|
|proximity moderat...|             62.74|  56.347500000000004|               29026.0|
|                haze| 51.93441379310345|   42.21977011494253|    25578.075862068967|
|        shower sleet|             11.65|                 4.0|               25136.0|
|          light rain| 55.99469234296196|   48.28205226960125|    28280.440106558883|
|                dust| 58.34052173913043|   49.90831884057972|               29306.6|
|light intensity d...| 56.87957292506042|  51.253650282030605|     29254.7751813054|
|light intensity s...| 58.42899543378993|   52.71281582952817|    29459.219178082192|
|proximity thunder...| 62.33487500000001|   55.29085416666669|    28945.922916666666|
|       broken clouds| 56.80642824392482|  49.646100985786504|     28832.29800412939|
|     overcast clouds|  57.0338110113237|   48.5816516985552|      28101.91484375|
|             squalls|             70.53|               62.16|               35513.0|
|   proximity drizzle|              64.9|               57.41|               31462.0|
+--------------------+------------------+--------------------+----------------------+
only showing top 20 rows
```

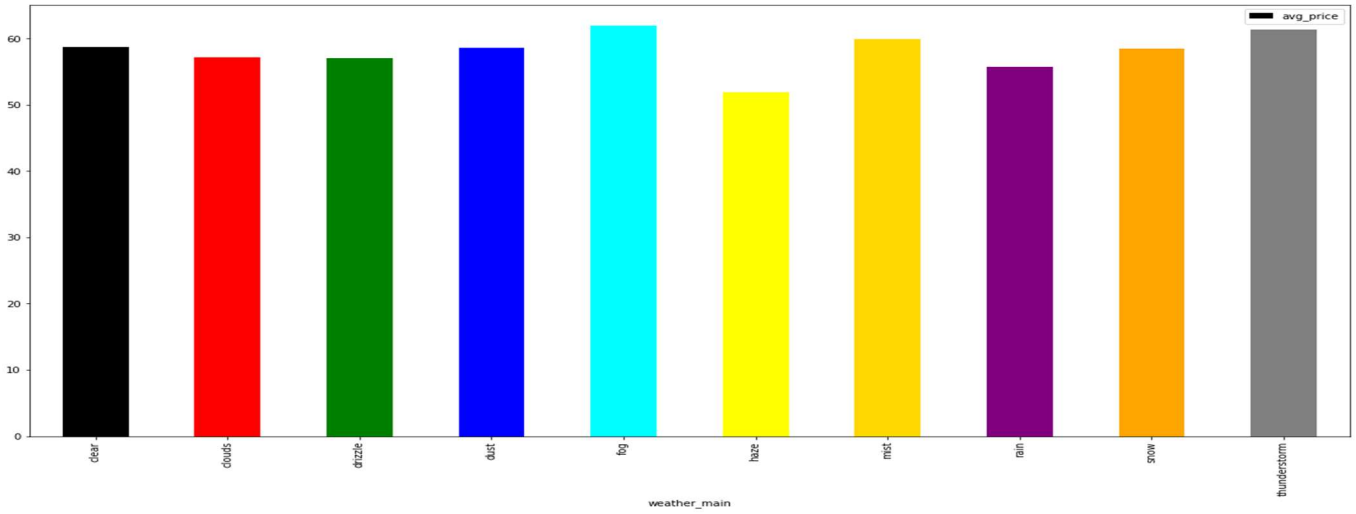
Pyspark Graphs

Average price by city



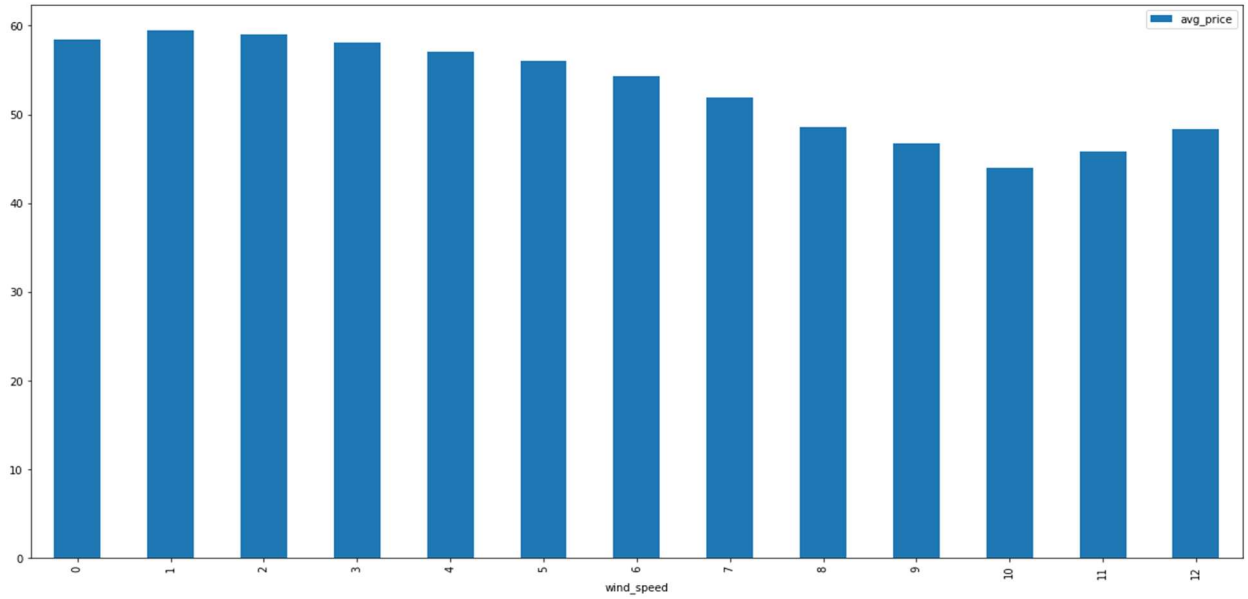
Average price of each city is almost similar to each city. But most highest one is Madrid

Average Price by weather condition



From the graph, We can see the average price by weather status. We can figure out the relationship between weather and price. Fog and the thunderstorm are most relevant weather condition which raise a electricity price

Average wind speed and Average price

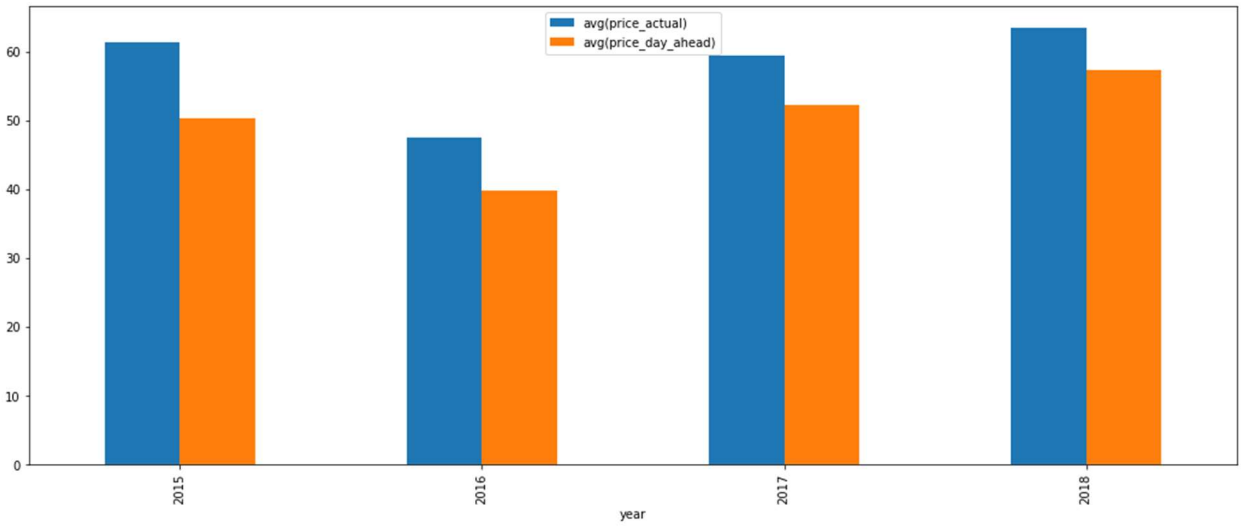


Average price seems to increase when average wind speed is at low level. We can infer that wind is supplied for the electricity, which makes the average price low.

Yearly Analysis

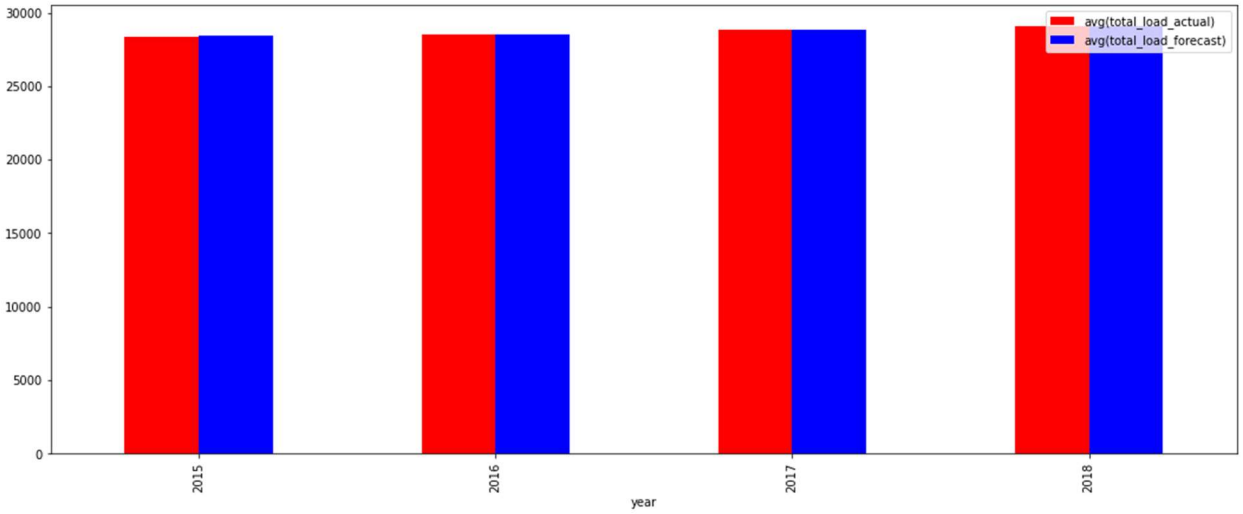
```
test_agg = joined_df.groupBy(f.year('time'))#
    .agg({"price_actual" : "avg", #
          "price_day_ahead" : "avg", #
          "total_load_actual" : "avg",#
          "total_load_forecast" : "avg",#
          "generation_biomass" : "avg",#
          "generation_fossil_brown_coal_lignite" : "avg",#
          "generation_fossil_gas" : "avg",#
          "generation_fossil_hard_coal" : "avg",#
          "generation_fossil_oil" : "avg",#
          "generation_hydro_pumped_storage_consumption" : "avg",#
          "generation_hydro_run_of_river_and_poundage" : "avg",#
          "generation_hydro_water_reservoir" : "avg",#
          "generation_nuclear" : "avg",#
          "generation_other" : "avg",#
          "generation_other_renewable" : "avg",#
          "generation_solar" : "avg",#
          "generation_waste" : "avg",#
          "generation_wind_onshore" : "avg",#
         })#
    .withColumnRenamed('year(time)', 'year')#
    .orderBy('year')
```

Average Price and prediction by Year

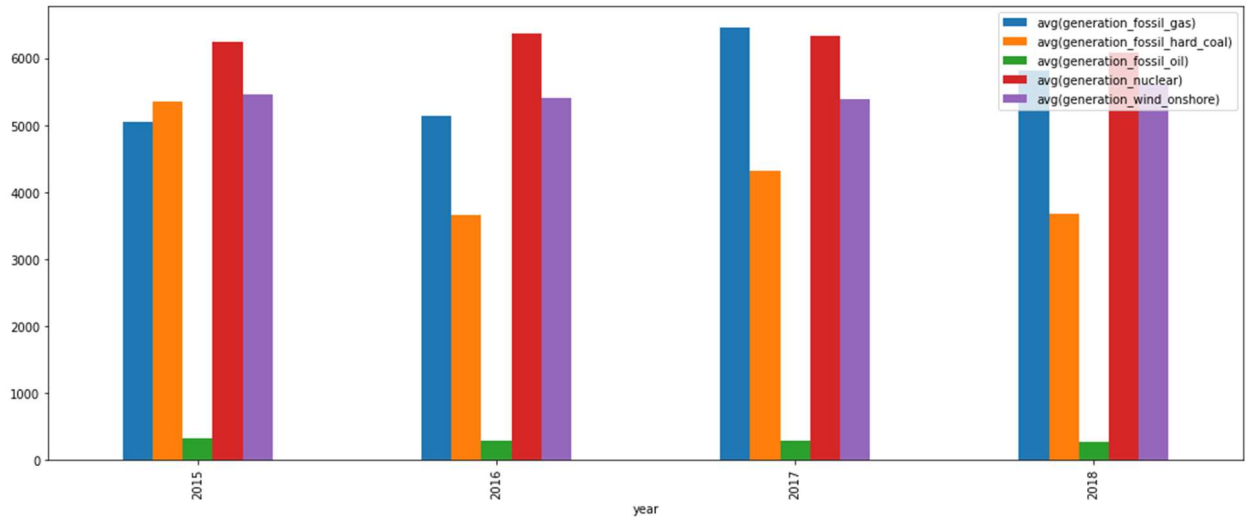


It shows each year's average price and average price prediction. There is some gap between them

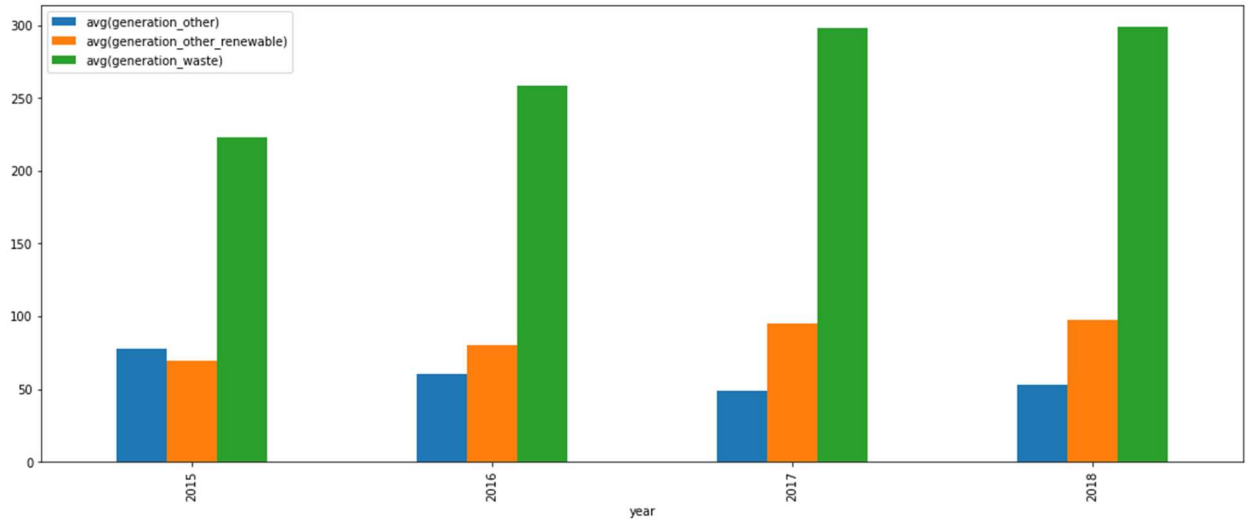
Average Load and prediction by Year



Main Resources generation By Year



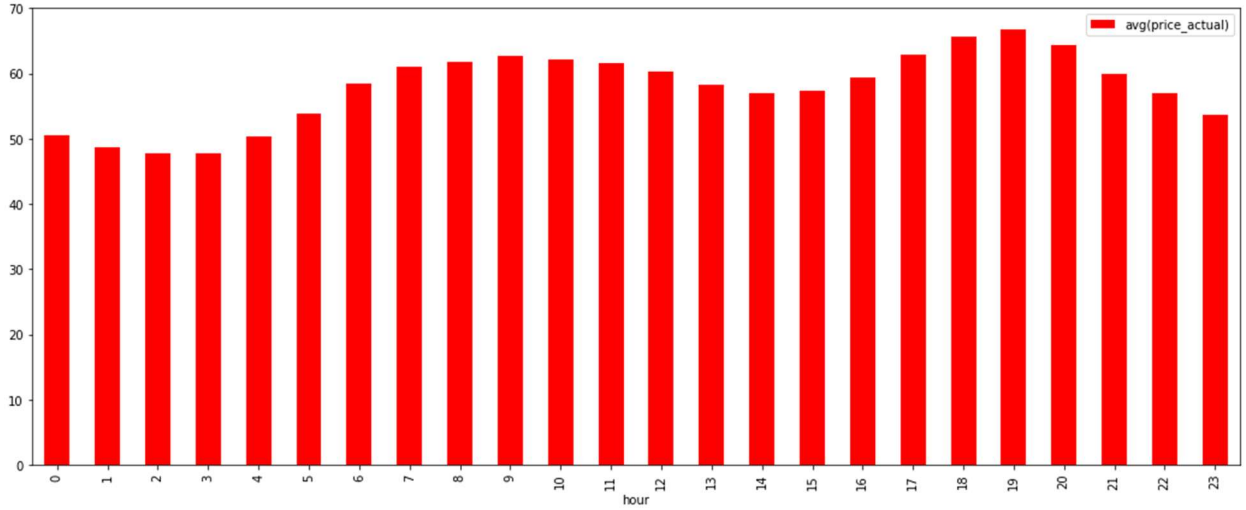
Other Resources generation By Year



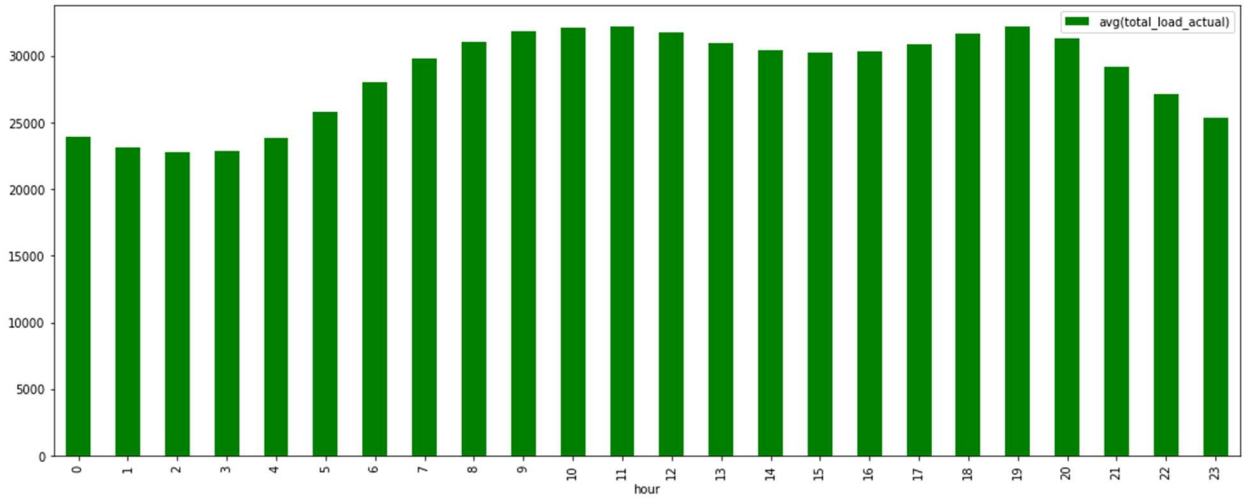
Hourly Analysis

```
test_agg = joined_df.groupBy(f.hour('time'))#
    .agg({"total_load_actual" : "avg", #
          "price_actual" : "avg", #
          "temp" : "avg", #
          "humidity" : "avg",
          "wind_deg" : "avg"})#
    .withColumnRenamed('hour(time)', 'hour')#
    .orderBy('hour(time)')
```

Average price by specific hour

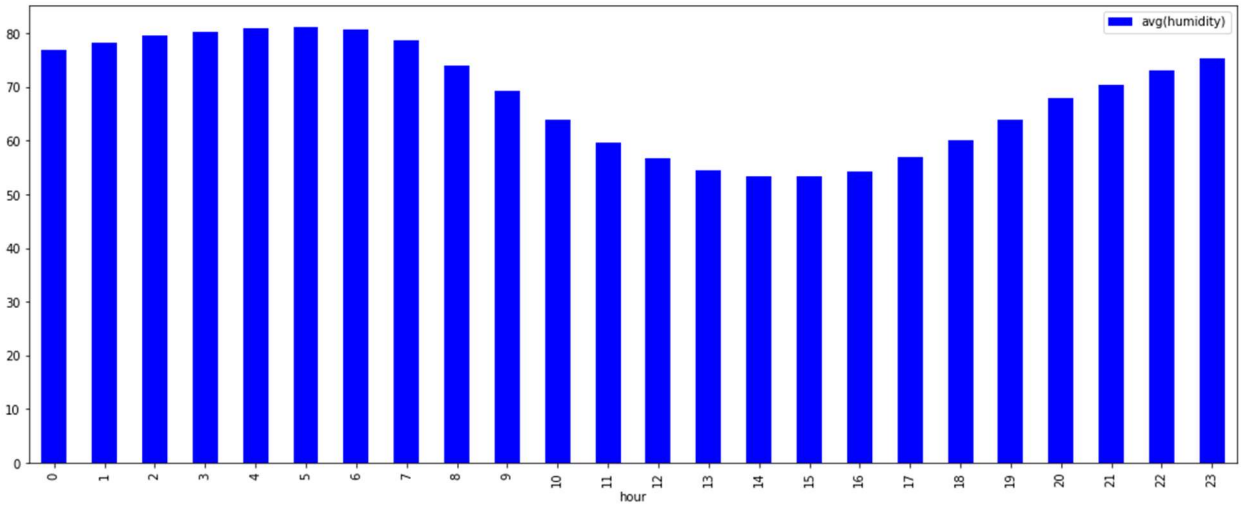


Average Total load by specific hour



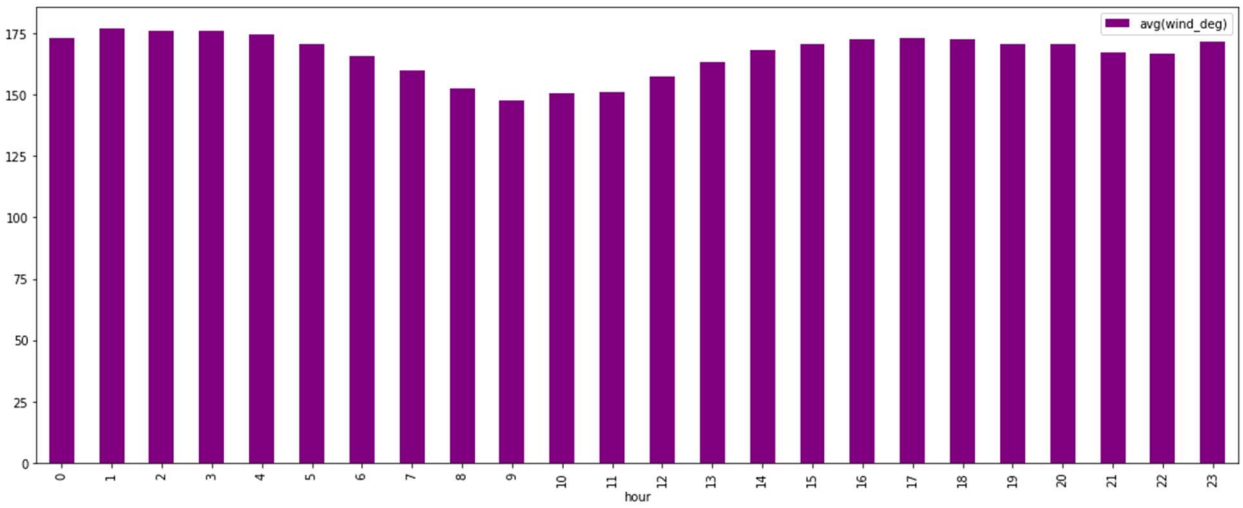
Load seems to increase in after noon time and decrease in morning. It is almost the same with the change of the average price graph.

Avg humidity by hour



When average humidity is going up, the average price is decreased. So We can say that humidity and price is in opposite relation

Average wind degree by hour



Similar to the average humidity. opposite relation with the average price

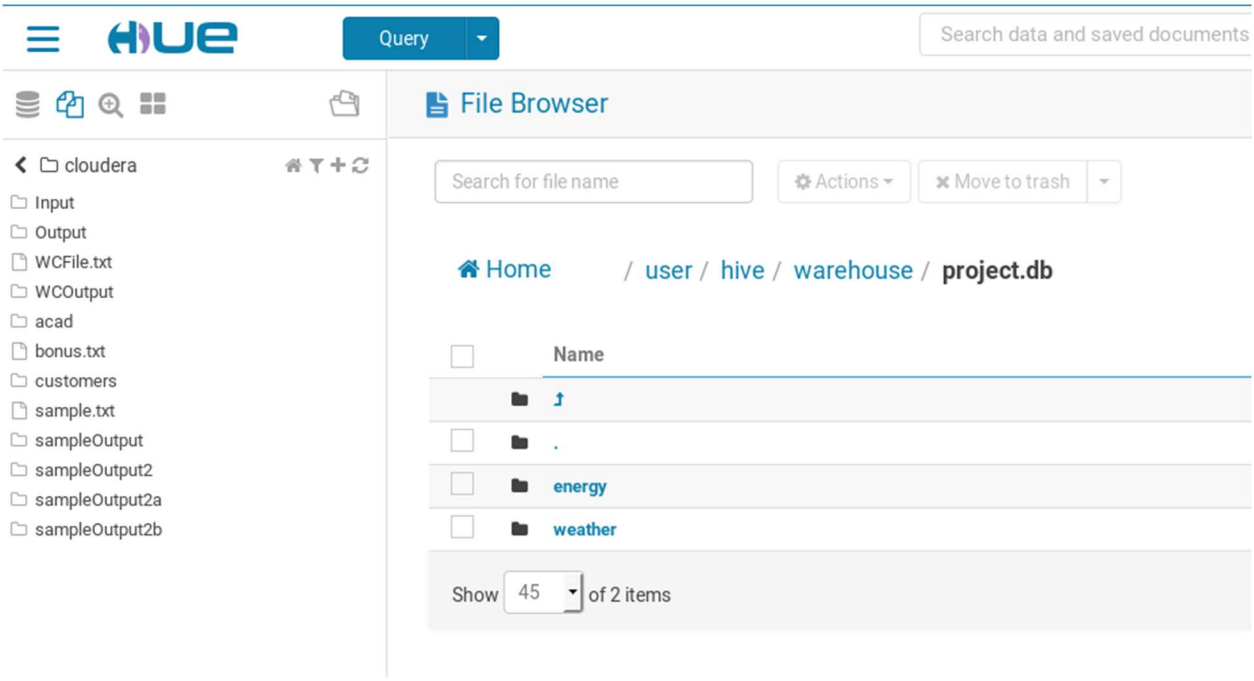
Bill Yerkes:

Create Tables in Hive:

```
CREATE TABLE Weather (
dt_iso TIMESTAMP,
city_name STRING,
temp DOUBLE,
temp_min DOUBLE,
temp_max DOUBLE,
pressure INT,
humidity INT,
wind_speed INT,
wind_deg INT,
rain_1h DOUBLE,
rain_3h DOUBLE,
snow_3h DOUBLE,
clouds_all INT,
weather_id INT,
weather_main STRING,
weather_description STRING,
weather_icon STRING)
row format delimited fields terminated by ',' stored as textfile;
```

```
CREATE TABLE Energy (
time TIMESTAMP,
generation_biomass INT,
generation_fossil INT,
brown_coal_lignite INT,
generation_fossil_coal_derived_gas INT,
generation_fossil_gas INT,
generation_fossil_hard_coal INT,
generation_fossil_oil INT,
generation_fossil_oil_shale INT,
generation_fossil_peat INT,
generation_geothermal INT,
generation_hydro_pumped_storage_aggregated INT,
generation_hydro_pumped_storage_consumption INT,
generation_hydro_run_of_river_and_poundage INT,
generation_hydro_water_reservoir INT,
generation_marine INT,
generation_nuclear INT,
generation_other INT,
generation_other_renewable INT,
generation_solar INT,
generation_waste INT,
generation_wind_offshore INT,
generation_wind_onshore INT,
forecast_solar_day_ahead INT,
forecast_wind_offshore_day_ahead INT,
forecast_wind_onshore_day_ahead INT,
total_load_forecast INT,
total_load_actual INT,
price_day_ahead DOUBLE,
price_actual DOUBLE)
row format delimited fields terminated by ',' stored as textfile;
```

Hive Tables in Hadoop:



Load the Data into Hive:

```
#------------------------------------------------------------------------
#Load the Data into Hive
#------------------------------------------------------------------------
load data local inpath '/home/cloudera/Downloads/weather_features.csv' into table Weather;

load data local inpath '/home/cloudera/Downloads/energy_dataset.csv' into table Energy;
```

Some of the queries on Energy Data Set:

```
Select AVG(generation_biomass),MAX(generation_biomass),MIN(generation_biomass),
STDDEV(generation_biomass) FROM energy;

Select AVG(generation_fossil),MAX(generation_fossil),MIN(generation_fossil),
STDDEV(generation_fossil) FROM energy;

Select AVG(brown_coal_lignite),MAX(brown_coal_lignite),MIN(brown_coal_lignite),
STDDEV(brown_coal_lignite) FROM energy;
```

Some of the queries on Weather Data Set:

```
Select city_name, Count(city_name) from weather group by city_name;

Select city_name, weather_description, Count(city_name)from weather group by
city_name, weather_description order by city_name, weather_description;

Select city_name, weather_description, Count(city_name), AVG(temp),MAX(temp),
MIN(temp), STDDEV(temp)from weather group by city_name, weather_description order
by city_name, weather_description;
```

Some of the metrics on the Data

	Average	Max	Min	STDV
generation biomass	383.51	592.00	0.00	85.35
generation fossil brown coal/lignite	448.06	999.00	0.00	354.57
generation fossil gas	5622.74	20034.00	0.00	2201.83
generation fossil hard coal	4256.07	8359.00	0.00	1961.60
generation fossil oil	298.32	449.00	0.00	52.52
generation hydro pumped storage consumption	475.58	4523.00	0.00	792.41
generation hydro run-of-river and poundage	972.12	2000.00	0.00	400.78
generation hydro water reservoir	2605.11	9728.00	0.00	1835.20
generation nuclear	6263.91	7117.00	0.00	839.67
generation other	60.23	106.00	0.00	20.24
generation other renewable	85.64	119.00	0.00	14.08
generation solar	1432.67	5792.00	0.00	1680.12
generation waste	269.45	357.00	0.00	50.20
generation wind onshore	5464.48	17436.00	0.00	3213.69
forecast wind onshore day ahead	5471.22	17430.00	237.00	3176.31
total load forecast	28712.13	41390.00	18105.00	4594.10
total load actual	28696.94	41015.00	18041.00	4574.99
price day ahead	49.87	101.99	2.06	14.62
price actual	57.88	116.80	9.33	14.20

Query and Results in Pyspark

```
[9]  city = dfWeather.select('city_name')

     city.printSchema()
     city.count()

     city.groupBy("city_name").count().show()
```

```
root
 |-- city_name: string (nullable = true)

+----------+-----+
| city_name|count|
+----------+-----+
|    Madrid|36267|
|   Seville|35557|
| Barcelona|35476|
|    Bilbao|35951|
|  Valencia|35145|
+----------+-----+
```

```
dfWeather.groupBy("city_name","weather_description").count().sort("city_name","weather_description").show()
```

```
+----------+--------------------+-----+
| city_name| weather_description|count|
+----------+--------------------+-----+
| Barcelona|       broken clouds| 2659|
| Barcelona|             drizzle|   65|
| Barcelona|          few clouds| 9502|
| Barcelona|                 fog|   74|
| Barcelona|heavy intensity d...|    4|
| Barcelona|heavy intensity rain|  527|
| Barcelona|heavy intensity s...|   14|
| Barcelona|          heavy snow|    2|
| Barcelona|light intensity d...|  224|
| Barcelona|light intensity d...|    5|
| Barcelona|light intensity s...|  171|
| Barcelona|          light rain| 1910|
| Barcelona|          light snow|    7|
| Barcelona|               mist|  443|
| Barcelona|       moderate rain|  576|
| Barcelona|     overcast clouds|  525|
| Barcelona|   proximity drizzle|    1|
| Barcelona|proximity moderat...|    2|
| Barcelona|proximity shower ...|  122|
| Barcelona|proximity thunder...|  188|
+----------+--------------------+-----+
only showing top 20 rows
```

Corrected issue with timestamp, converted from string and added offset

▾ Function so we can add offset back to time stamp

We are saving it as a string, we will later convert it back to a timestamp

```
[9]  from datetime import datetime, timedelta
     format = "%Y-%m-%d %H:%M:%S"

     def addTimeOffset(v_date, v_hour, v_minute):
       new_datetime = timedelta(minutes = int(v_minute), hours = int(v_hour))
       return_dt = v_date + new_datetime
       date_string = return_dt.strftime(format)
       return date_string
     udfaddTimeOffset = f.udf(addTimeOffset, StringType())
```

▾ Converting the string to a timestamp

also pulling out the hours and minute offset

```
[10] EnergyDF = EnergyDF.withColumn('dtTimeStamp', to_timestamp(f.substring('time', 1,19), 'yyyy-MM-dd HH:mm:ss')) \
                        .withColumn('dtHourOffset', f.substring('time', 21,2)) \
                        .withColumn('dtMinuteOffset', f.substring('time', 24,2))

     WeatherDF = WeatherDF.withColumn('dtTimeStamp', to_timestamp(f.substring('dt_iso', 1,19), 'yyyy-MM-dd HH:mm:ss')) \
                          .withColumn('dtHourOffset', f.substring('dt_iso', 21,2)) \
                          .withColumn('dtMinuteOffset', f.substring('dt_iso', 24,2))
```

Add new timestamp to DF with offset added to them

```
[12] EnergyDF = EnergyDF.withColumn("interval", udfaddTimeOffset("dtTimeStamp","dtHourOffset","dtMinuteOffset"))
     EnergyDF = EnergyDF.withColumn('dtTimeStampUpdated', to_timestamp("interval", 'yyyy-MM-dd HH:mm:ss'))

     WeatherDF = WeatherDF.withColumn("interval", udfaddTimeOffset("dtTimeStamp","dtHourOffset","dtMinuteOffset"))
     WeatherDF = WeatherDF.withColumn('dtTimeStampUpdated', to_timestamp("interval", 'yyyy-MM-dd HH:mm:ss'))

     EnergyDF.select('dtTimeStamp','dtTimeStampUpdated').show()
     WeatherDF.select('dtTimeStamp','dtTimeStampUpdated').show()
```

```
+-------------------+-------------------+
|        dtTimeStamp| dtTimeStampUpdated|
+-------------------+-------------------+
|2015-01-01 00:00:00|2015-01-01 01:00:00|
|2015-01-01 01:00:00|2015-01-01 02:00:00|
|2015-01-01 02:00:00|2015-01-01 03:00:00|
|2015-01-01 03:00:00|2015-01-01 04:00:00|
|2015-01-01 04:00:00|2015-01-01 05:00:00|
|2015-01-01 05:00:00|2015-01-01 06:00:00|
|2015-01-01 06:00:00|2015-01-01 07:00:00|
|2015-01-01 07:00:00|2015-01-01 08:00:00|
|2015-01-01 08:00:00|2015-01-01 09:00:00|
|2015-01-01 09:00:00|2015-01-01 10:00:00|
|2015-01-01 10:00:00|2015-01-01 11:00:00|
|2015-01-01 11:00:00|2015-01-01 12:00:00|
|2015-01-01 12:00:00|2015-01-01 13:00:00|
|2015-01-01 13:00:00|2015-01-01 14:00:00|
|2015-01-01 14:00:00|2015-01-01 15:00:00|
|2015-01-01 15:00:00|2015-01-01 16:00:00|
|2015-01-01 16:00:00|2015-01-01 17:00:00|
|2015-01-01 17:00:00|2015-01-01 18:00:00|
|2015-01-01 18:00:00|2015-01-01 19:00:00|
|2015-01-01 19:00:00|2015-01-01 20:00:00|
+-------------------+-------------------+
```

Joining Weather and Energy Datasets

▾ Inner Join between the weather and energy dataframes

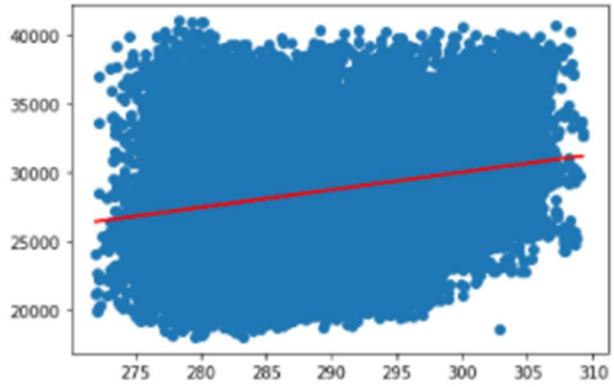
```
[14] WeatherPartialDF = WeatherDF.groupBy('dtTimeStampUpdated').agg(f.avg("temp").alias("Temp_avg"), \
                                                                   f.avg("pressure").alias("Pressure_avg"),\
                                                                   f.avg("humidity").alias("humidity_avg") )

     WeatherTempDF = WeatherDF.groupBy('dtTimeStampUpdated').agg(f.avg("temp").alias("Temp_avg"))
     EnergyDFTotalLoad = EnergyDF.select('dtTimeStampUpdated','total load actual')

     WeatherEnergyDF = WeatherTempDF.join(EnergyDF, on=['dtTimeStampUpdated'], how='inner')
     EnergyWeatherDF = EnergyDFTotalLoad.join(WeatherPartialDF, on=['dtTimeStampUpdated'], how='inner')

     PandasWeatherEnergyDF = WeatherEnergyDF.toPandas()
     PandasEnergyWeatherDF = EnergyWeatherDF.orderBy("dtTimeStampUpdated").toPandas()

     ## Data for perfoming linear regression
     linearReg_X = PandasWeatherEnergyDF[['Temp_avg']]
     linearReg_y = PandasWeatherEnergyDF[['total load actual']]
```

Linear Regression between temperature and total energy load:

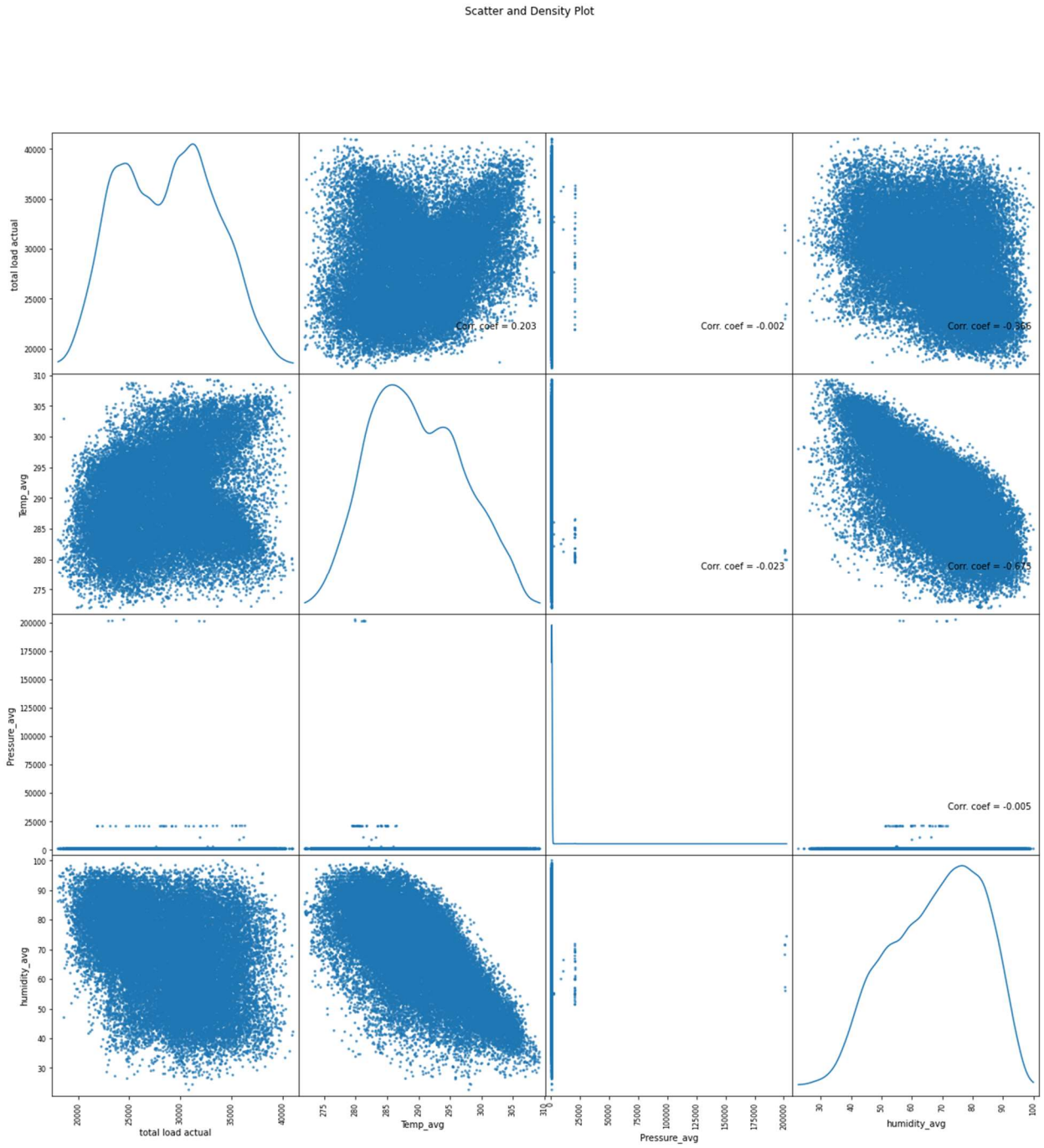
Perform a Linear Regression

```
body_reg = linear_model.LinearRegression()
body_reg.fit(linearReg_X,linearReg_y)
# parameters
m =body_reg.coef_
c =body_reg.intercept_
plt.scatter(linearReg_X,linearReg_y)
plt.plot(linearReg_X,body_reg.predict(linearReg_X),color='Red')
```

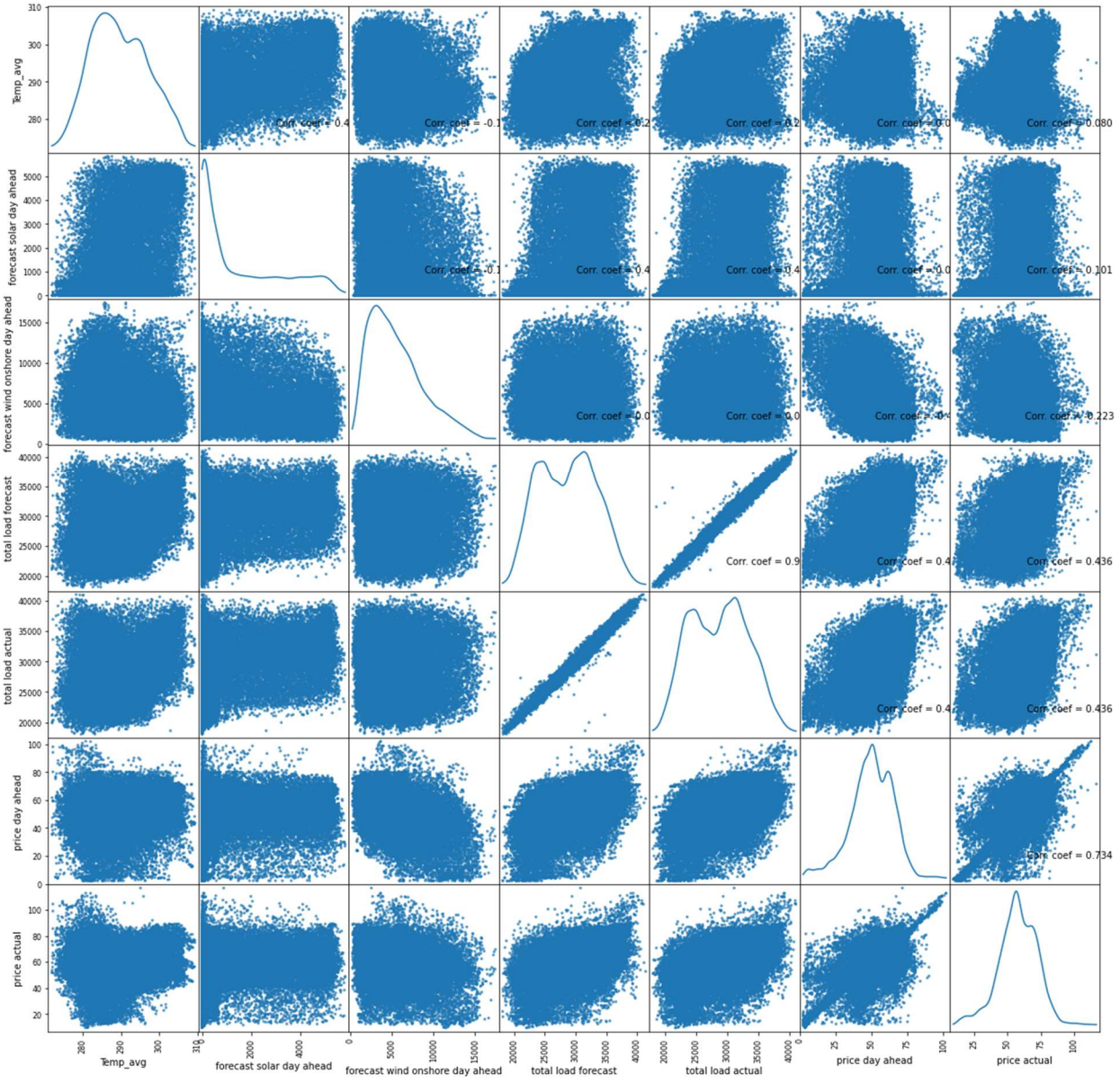
```
[<matplotlib.lines.Line2D at 0x7febbc22ba10>]
```



Relationship between: Total energy load, Temperature, Pressure, and Humidity:

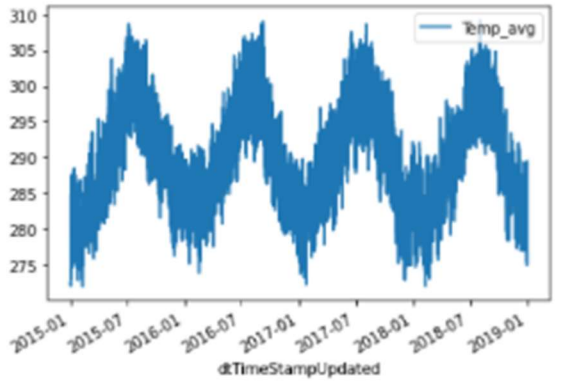


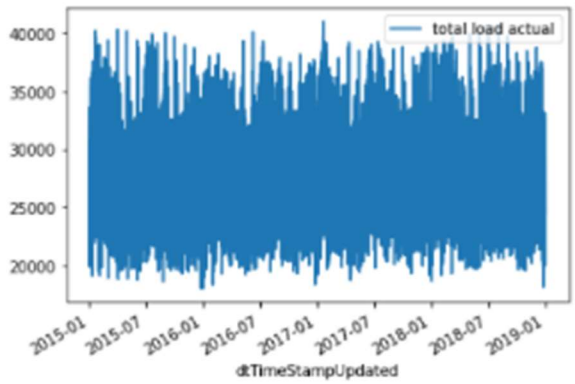
Scatter and Density Plot

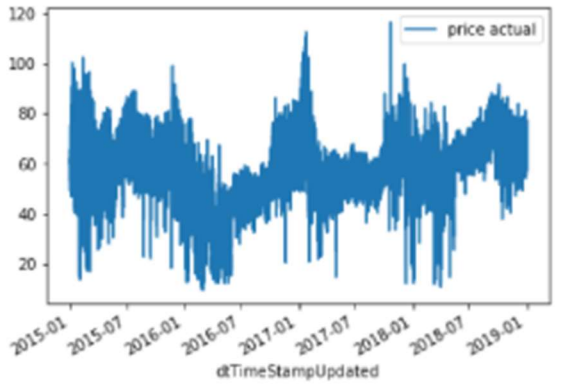


View of Temperature, Total Load, and Price over time

```
PandasWeatherEnergyDF.plot(x="dtTimeStampUpdated", y=["Temp_avg"])
plt.show()

PandasWeatherEnergyDF.plot(x="dtTimeStampUpdated", y=["total load actual"])
plt.show()

PandasWeatherEnergyDF.plot(x="dtTimeStampUpdated", y=["price actual"])
plt.show()
```







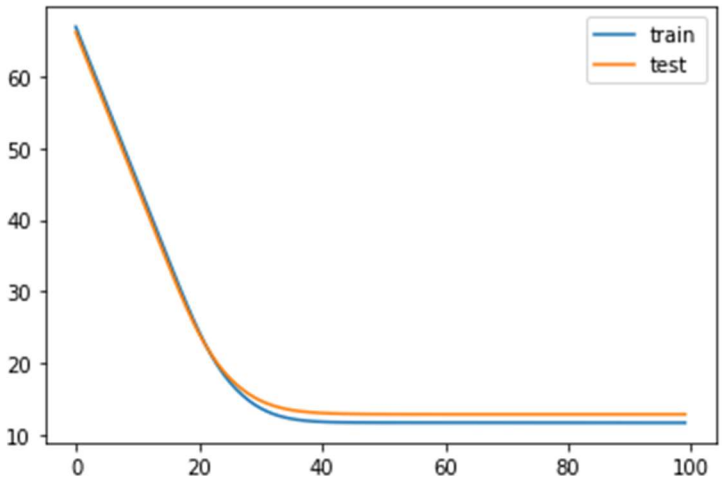
Long Short Term Memory Model:

```
# split into train and test sets
#values = reframed.values
n_train_hours = 365 * 24
train = values[:n_train_hours, :]
test = values[n_train_hours:, :]
# split into input and outputs
train_X, train_y = train[:, :-1], train[:, -1]
test_X, test_y = test[:, :-1], test[:, -1]
# reshape input to be 3D [samples, timesteps, features]
train_X = train_X.reshape((train_X.shape[0], 1, train_X.shape[1]))
test_X = test_X.reshape((test_X.shape[0], 1, test_X.shape[1]))
print(train_X.shape, train_y.shape, test_X.shape, test_y.shape)
```

```
(8760, 1, 3) (8760,) (26268, 1, 3) (26268,)
```

Desiging and creating the model

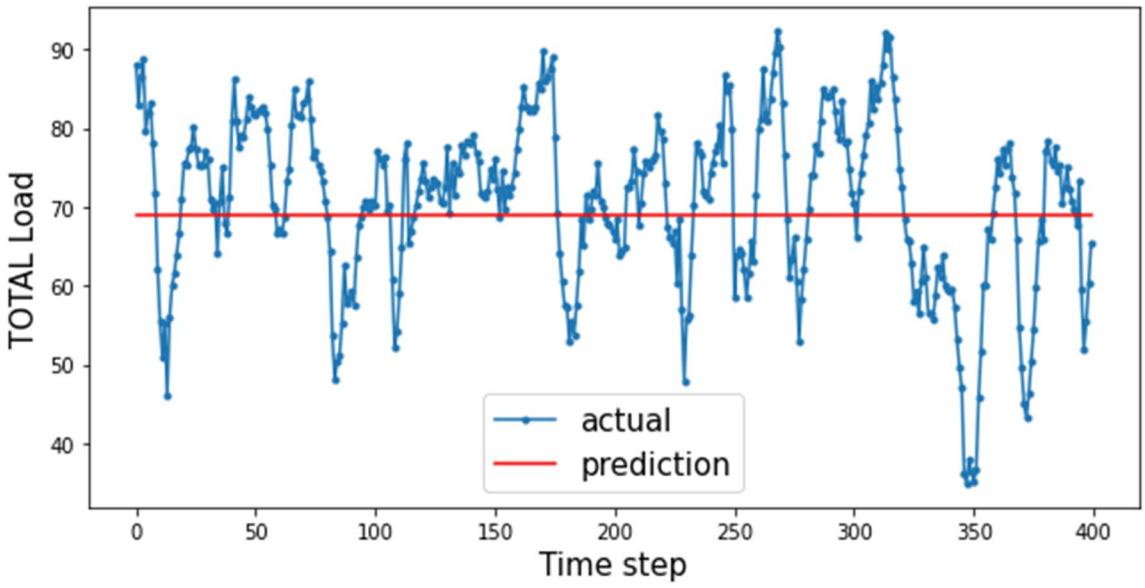
```
# design network
from keras.layers.core import Dense, Activation, Dropout
from keras.layers.recurrent import LSTM
from keras.models import Sequential

model = Sequential()
model.add(LSTM(100, input_shape=(train_X.shape[1], train_X.shape[2])))
model.add(Dense(1))
model.compile(loss='mae', optimizer='adam')
# fit network
history = model.fit(train_X, train_y, epochs=100, batch_size=72, validation_data=(test_X, test_y), verbose=2, shuffle=False)
# plot history
pyplot.plot(history.history['loss'], label='train')
pyplot.plot(history.history['val_loss'], label='test')
pyplot.legend()
pyplot.show()
```



LSTM Results:

```
# make a prediction
yhat = model.predict(test_X)
test_X1 = test_X.reshape((test_X.shape[0], test_X.shape[2]))
# invert scaling for forecast

# invert scaling for actual
test_y = test_y.reshape((len(test_y), 1))

# calculate RMSE

print(mean_squared_error(test_y, yhat))

rmse = math.sqrt(mean_squared_error(test_y, yhat))
print('Test RMSE: %.3f' % rmse)
```

```
232.48802374826883
Test RMSE: 15.248
```

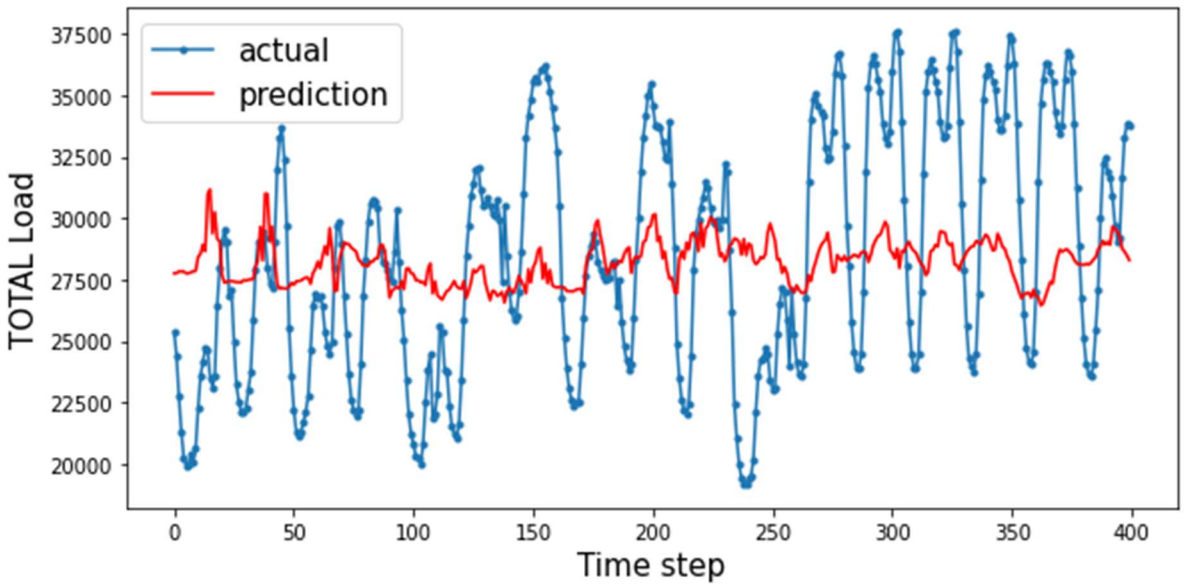


Ridge Model:

```
[45] rr = Ridge(alpha=0.01)
     rr.fit(X_train, y_train)
     pred_train_rr= rr.predict(X_train)
     print(np.sqrt(mean_squared_error(y_train,pred_train_rr)))
     print(r2_score(y_train, pred_train_rr))

     pred_test_rr= rr.predict(X_test)
     print(np.sqrt(mean_squared_error(y_test,pred_test_rr)))
     print(r2_score(y_test, pred_test_rr))
```

```
4254.827749792242
0.136185478314513
4234.536646970196
0.14060291578775475
```

Results:

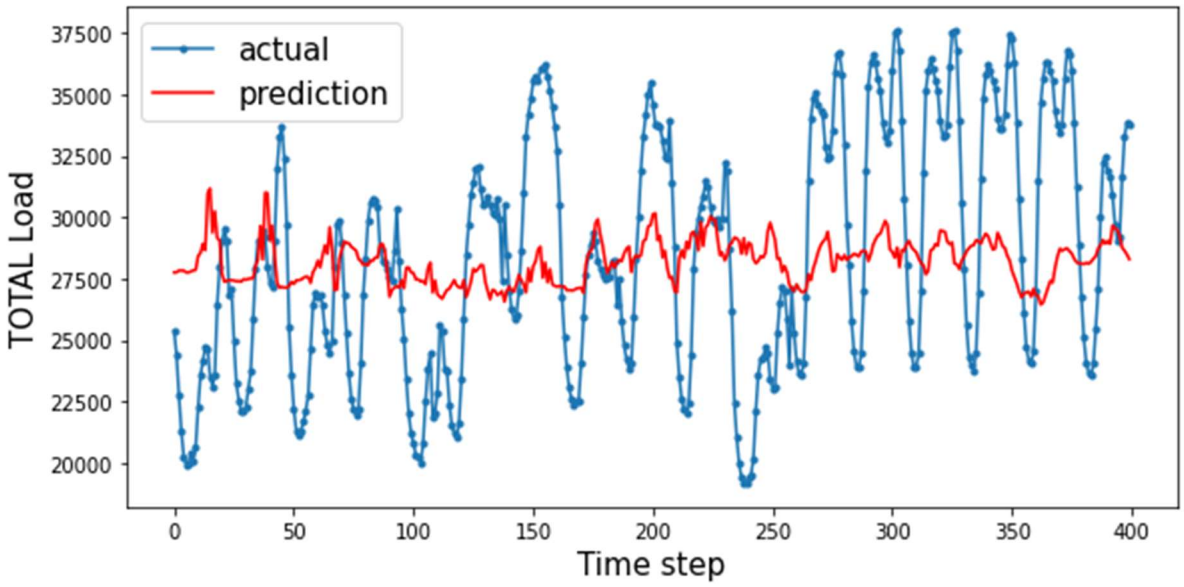


Lasso Model:

```
[47] model_lasso = Lasso(alpha=0.01)
     model_lasso.fit(X_train, y_train)
     pred_train_lasso= model_lasso.predict(X_train)
     print(np.sqrt(mean_squared_error(y_train,pred_train_lasso)))
     print(r2_score(y_train, pred_train_lasso))

     pred_test_lasso= model_lasso.predict(X_test)
     print(np.sqrt(mean_squared_error(y_test,pred_test_lasso)))
     print(r2_score(y_test, pred_test_lasso))
```

```
4254.827840653484
0.1361854414212399
4234.541855537697
0.14060080163414113
```

Results:

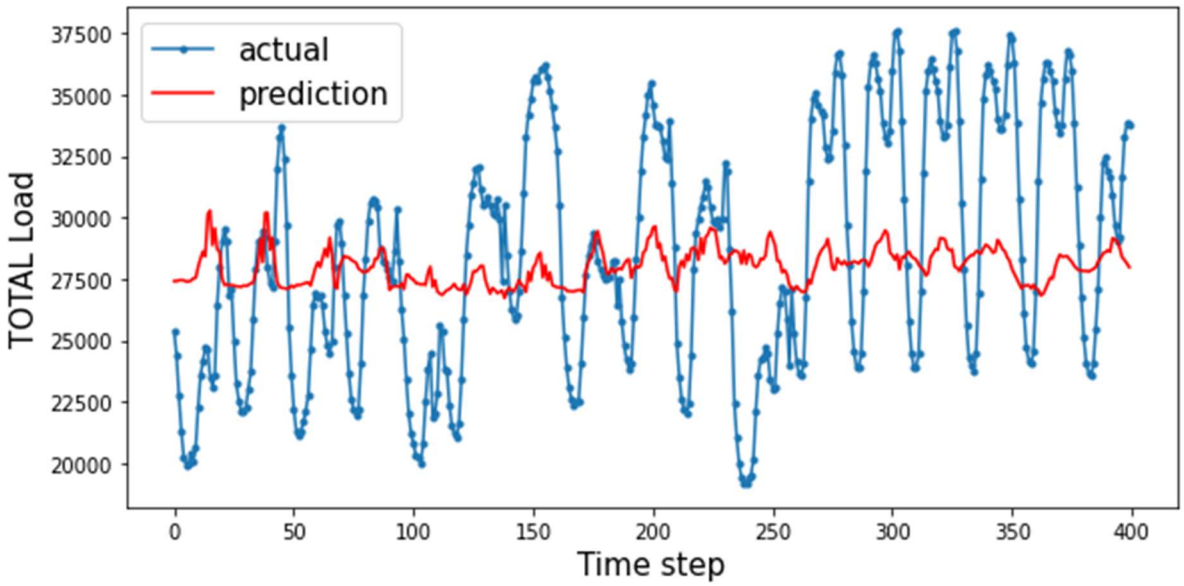


Elastic Model:

```
[49] #Elastic Net
     model_enet = ElasticNet(alpha = 0.01)
     model_enet.fit(X_train, y_train)
     pred_train_enet= model_enet.predict(X_train)
     print(np.sqrt(mean_squared_error(y_train,pred_train_enet)))
     print(r2_score(y_train, pred_train_enet))

     pred_test_enet= model_enet.predict(X_test)
     print(np.sqrt(mean_squared_error(y_test,pred_test_enet)))
     print(r2_score(y_test, pred_test_enet))
```

```
4274.636074490374
0.12812378982508799
4255.736646545202
0.1319763168693785
```

Results:



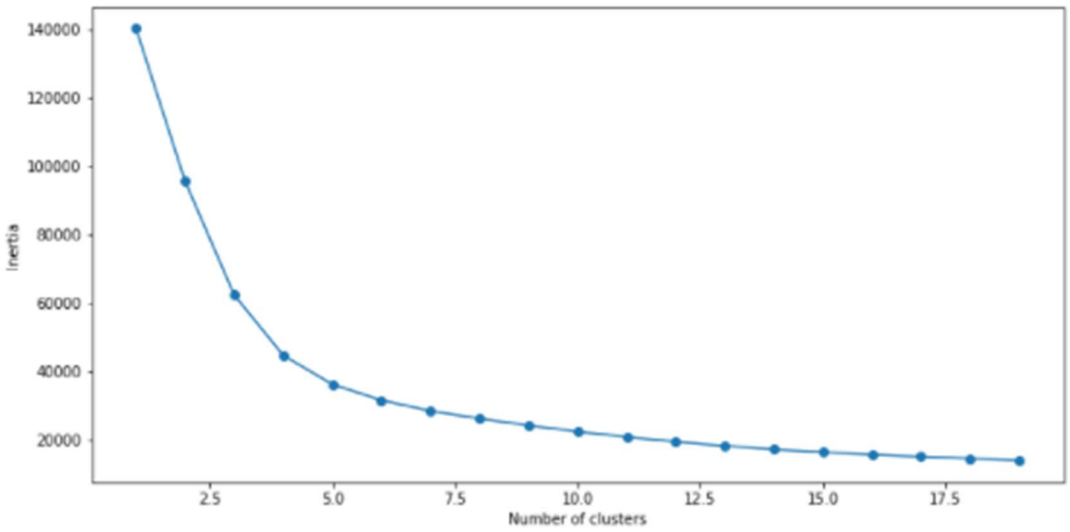
KMeans Cluster

```
scaler = StandardScaler()
data_scaled = scaler.fit_transform(dfKmeans)

# defining the kmeans function with initialization as k-means++
kmeans = KMeans(n_clusters=2, init='k-means++')

# fitting the k means algorithm on scaled data
kmeans.fit(data_scaled)

# inertia on the fitted data
kmeans.inertia_
```

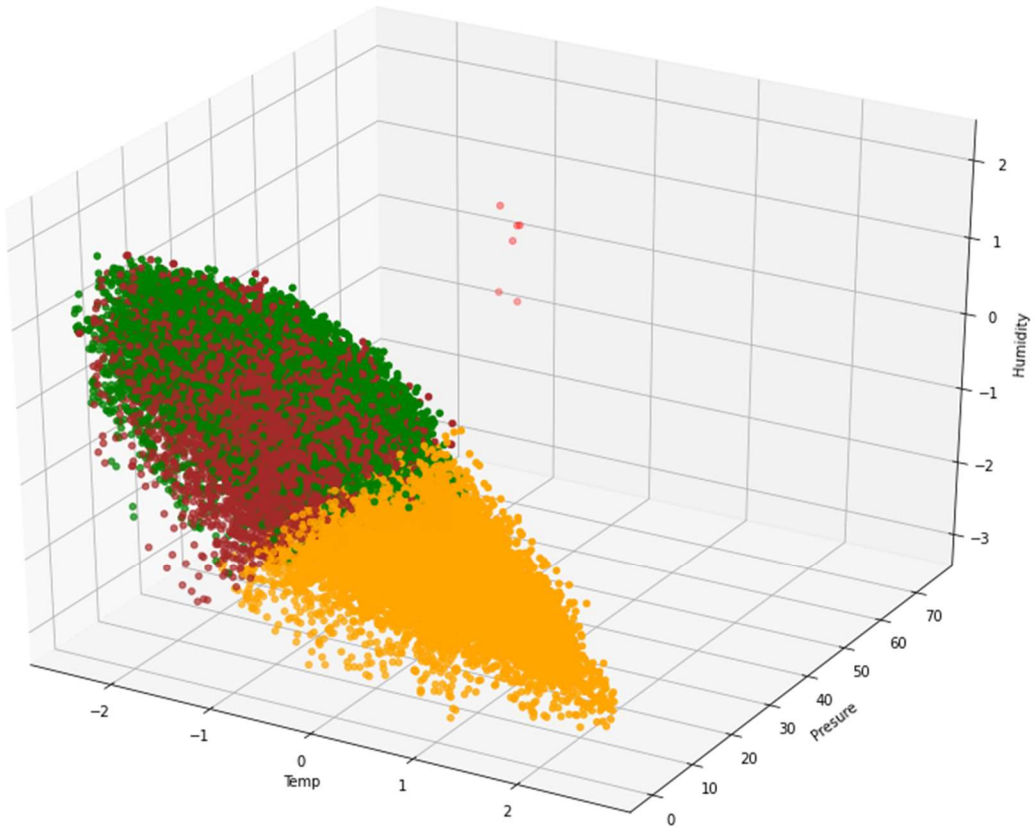
```
95425.55825878611
```

```
# fitting multiple k-means algorithms and storing the values in an empty list
SSE = []
for cluster in range(1,20):
    kmeans = KMeans(n_jobs = -1, n_clusters = cluster, init='k-means++')
    kmeans.fit(data_scaled)
    SSE.append(kmeans.inertia_)

# converting the results into a dataframe and plotting them
frame = pd.DataFrame({'Cluster':range(1,20), 'SSE':SSE})
plt.figure(figsize=(12,6))
plt.plot(frame['Cluster'], frame['SSE'], marker='o')
plt.xlabel('Number of clusters')
plt.ylabel('Inertia')
```

```
Text(0, 0.5, 'Inertia')
```



Visualization:



Data and Parameters:

Hourly energy demand generation and weather | Kaggle

This dataset contains 4 years of electrical consumption, generation, pricing, and weather data for Spain. Consumption and generation data were retrieved from ENTSOE a public portal for Transmission Service Operator (TSO) data. Settlement prices were obtained from the Spanish TSO Red Electric España. Weather data was purchased as part of a personal project from the Open Weather API for the 5 largest cities in Spain and made public here.

GitHub Link:

BillYerkes/CSEE5590_GroupProject (github.com)

Issues or Concerns:

Big Data Programming comprises a large area. Spending one or two weeks on a topic offered the team the ability to get exposure to the various components. The subject of Machine Learning is one area the team could have used more time on.

Working remotely from team members during a pandemic offers its own set of challenges and stress.

One aspect of Big Data is the volume of data available to perform analysis on, our weather data could have been a bit larger to help improve the development of models. Finding data sets that had both weather and energy information was a bit difficult, learn how to manipulate the data set to get information relevant to the problem we were trying to solve.

Conclusion:

The team has familiarized themselves with the tools learned over the semester. The team has also investigated the two datasets to be able to better understand how to construct our solution. The team has received exposure to how Big Data can be used to solve real world problems. The team learned how to work together remotely using different technologies to help facilitate communication and collaboration, such as Discord, Zoom, GitHub and Email..

The scientist:

UMKC Students / CSEE 5590 Big Data Programming. .

Anna Johnson, Joe Goldsich, Jongkook Son, and Bill Yerkes

Users

There are two main users for our application. Consumers of energy utility companies and producers of the energy being consumed.

The Society

Thanks to our application, the overall total utility/energy cost for our society would decrease because each subject would be able to act appropriately according to the prediction of the application. Producers would be able to expand or decrease their production line based on the weather forecast. Consumers of energy would be able to avoid huge amounts of electricity bills because of a more efficient system.

Video Presentations:

Iteration 2

Final Iteration

References:

1. **Betting on the Weather: Predicting Tomorrow's Energy Prices Using Machine Learning**
2. **Predict the Unpredictable: Power Forecasting and Energy Trading (baxenergy.com)**
3. **Predicting wind and solar generation from weather data using Machine Learning | by Hugo Ferreira | Hugo Ferreira's blog | Medium**
4. **https://www.kaggle.com/nicholasjhana/energy-consumption-generation-prices-and-weather**
5. **Thousands caught off guard by rolling blackouts in Kansas City metro Monday (fox4kc.com)**
6. **https://www.texastribune.org/2021/02/17/texas-power-grid-failures/**
7. **https://www.forbes.com/sites/arielcohen/2021/02/19/texas-energy-crisis-is-an-epic-resilience-and-leadership-failure/?sh=46d08806eee8**
8. **California rolling blackouts during summer heat wave caused by 3 main factors, report says | Fox Business**