CSEE5590/490: Big Data Programming

Final Report

Project Title: Energy Demand Analysis in Spain

Team Members: Claire Ndofor, Wes McNall, Scott Howard, Shelby Mohar

Introduction:

Forecasting in energy markets is one exceedingly helpful tool in making the transition to a renewable-based electrical infrastructure (Rolnick et al, 2019). By improving forecasting, we can also increase the efficiency of a power grid and help reduce the usage of peak demand on power plants, which are generally less efficient than their counterparts. While the short-term results have the potential to improve 24-hour and hour-by-hour predictions, this work also has the potential predict energy prices for consumers.

Background:

The data is collected from the five largest cities in Spain: Madrid, Barcelona, Valencia, Seville, and Bilbao between the years of 2015 and 2019. This data has the ability to impact every community that uses an electrical grid. Not only is it advantageous at the individual level to be able to predict the cost of an electric bill, but it is also extremely helpful to be able to predict energy usage at a macro level as communities across the globe begin to make the transition to renewable energies in response to climate change. As stated in the introduction, forecasting in energy markets is an exceedingly helpful tool in making the transition to a renewable-based electrical infrastructure (Rolnick et al, 2019).

Goals and Objectives:

- Motivation:
    - Forecasting in energy markets is one exceedingly helpful tool in making the transition to a renewable-based electrical infrastructure, as stated in "Tackling Climate Change with Machine Learning" (see resources for link to paper). Our goal is to demonstrate this by leveraging Big Data analysis tools on a dataset that consists of energy usage and weather data for five large cities in Spain.
- Significance:
    - Predict energy usage to increase efficiency of electrical production
    - Predict energy price
    - Locate areas that would benefit from renewable energies
- Objectives:
    - Predict energy usage based on the weather
    - Predict energy prices by:
        - Time of day
        - Day of the week
        - Time of year

- Analyze the factors that affect the fluctuations in energy usage, as well as the sources of energy

- Features:
    - dt_iso (datetime index localized to CET)
    - generation biomass (in MW)
    - generation fossil brown coal/lignite (in MW)
    - generation fossil coal-derived gas (in MW)
    - generation fossil gas (in MW)
    - generation fossil hard coal (in MW)
    - generation fossil oil (in MW)
    - generation fossil oil shale (in MW)
    - generation fossil peat (in MW)
    - generation geothermal (in MW)
    - city_name
    - temp (in kelvin)
    - temp_min (in kelvin)
    - temp_max (in kelvin)
    - pressure (in hPa)
    - humidity (in %)
    - wind_speed (in m/s)
    - wind_deg (wind direction)
    - rain_1h (rain in last hour in mm)

Dataset

Our dataset is comprised of two .csv files:

- weather_features.csv – contains information about the weather
- energy_dataset.csv – contains information about the production, price, and variation of energy resources

The two files can be joined by a timestamp. The dataset can be found on Kaggle with the heading "Hourly energy demand generation and weather". See resources for link.

Features Developed:

This section is dedicated to the features developed in this increment, and a guide to the files within the team repo.

HiveQL: (Wes)

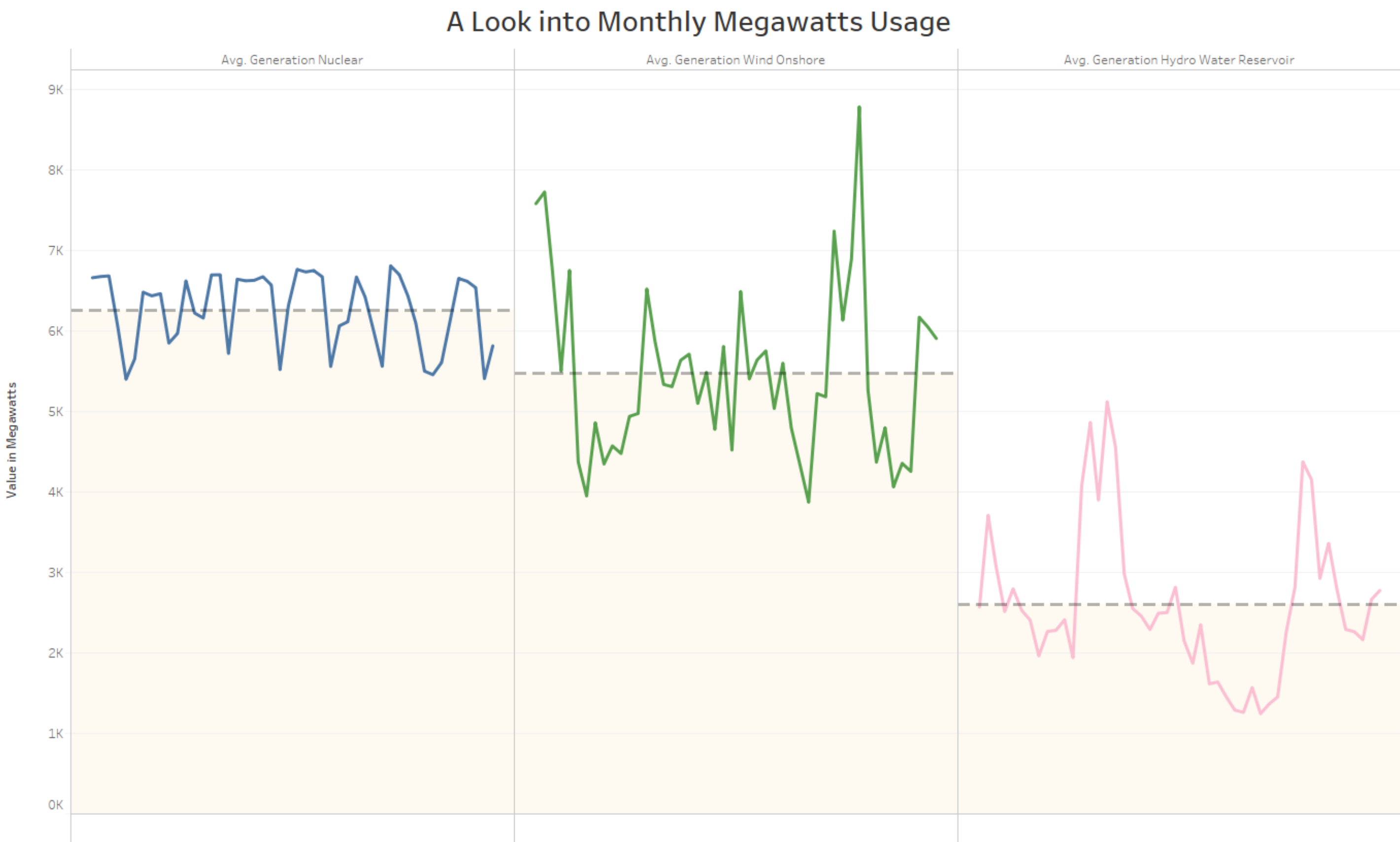
During a past class our professor mentioned using Graphs in this project and as soon as I heard that I knew I wanted to use Tableau to visualize some key aspects of the data, not only to learn more about it but to show key findings.

With the data already loaded from the previous Increment I took to asking some questions about the data and then visualizing the data to see what was interesting about it. Because we are dealing with trends of prices over time, that was a key aspect that I wanted to be able to visualize. Using Tableau I could add some extra visuals without having to calculate, such as trend lines for each particular year. Looking at the Average Electricity Price per Year Graph we can see there are clear lows and highs between the years, which will require some further investigation as to why those trends exist

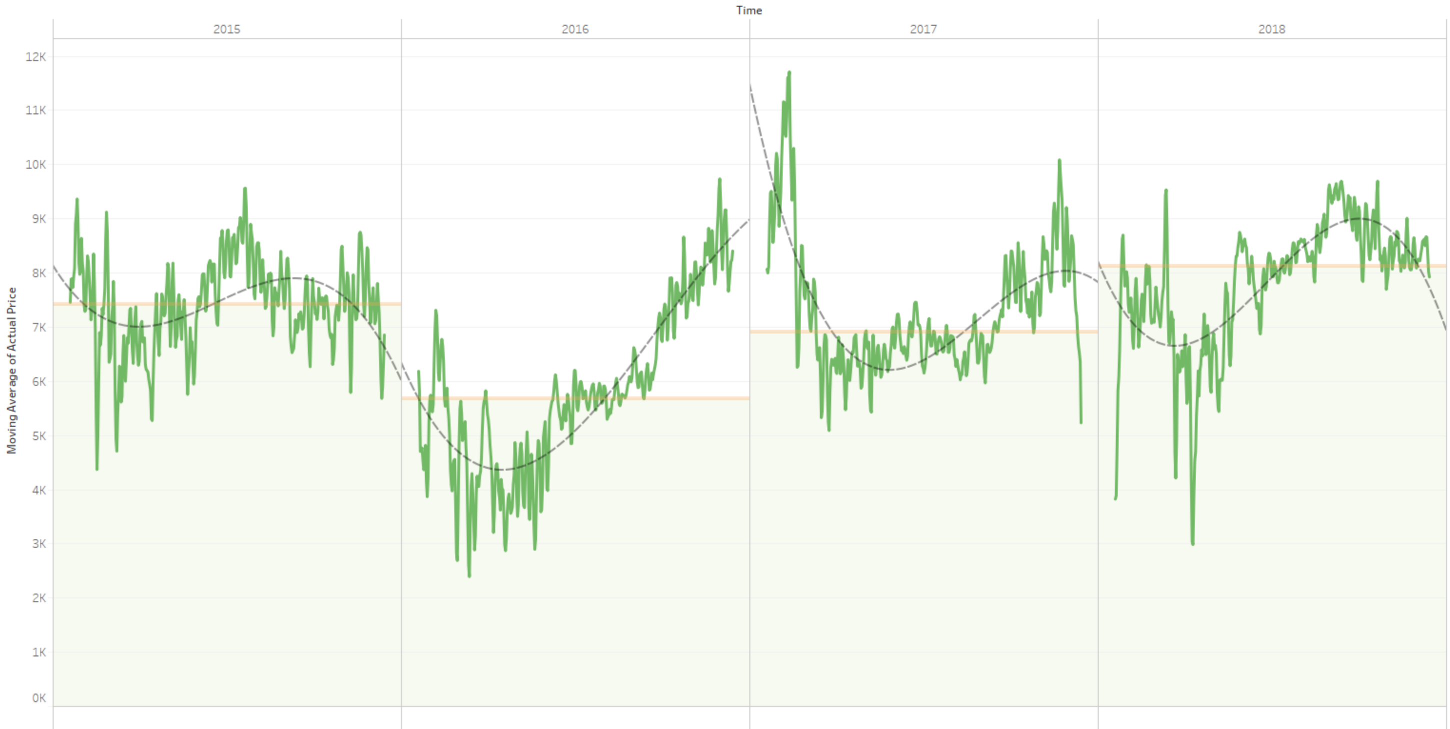
With a wide dataset, part of what I wanted to accomplish this Increment was to determine what columns had interesting and worthwhile data and what columns could be more or less ignored. By writing a large HiveQL query that included summary statistics over time, it would allow us to look for trends and determine which were worth investigating further. All separate Megawatts Usage graphs were trends I found interesting enough to highlight, and all others within the dataset I left ignored from the graphs and queries

Something I wanted to see was not only trends over months and years, but just over the course of a day. Specifically the average prices over different times of day. The Prices by Hour of Day Graph shows that there is a fluctuation of the cost throughout the course of the day. This is to be expected and the highs and lows also match times that make sense for what time most people will be working and most people will be sleeping

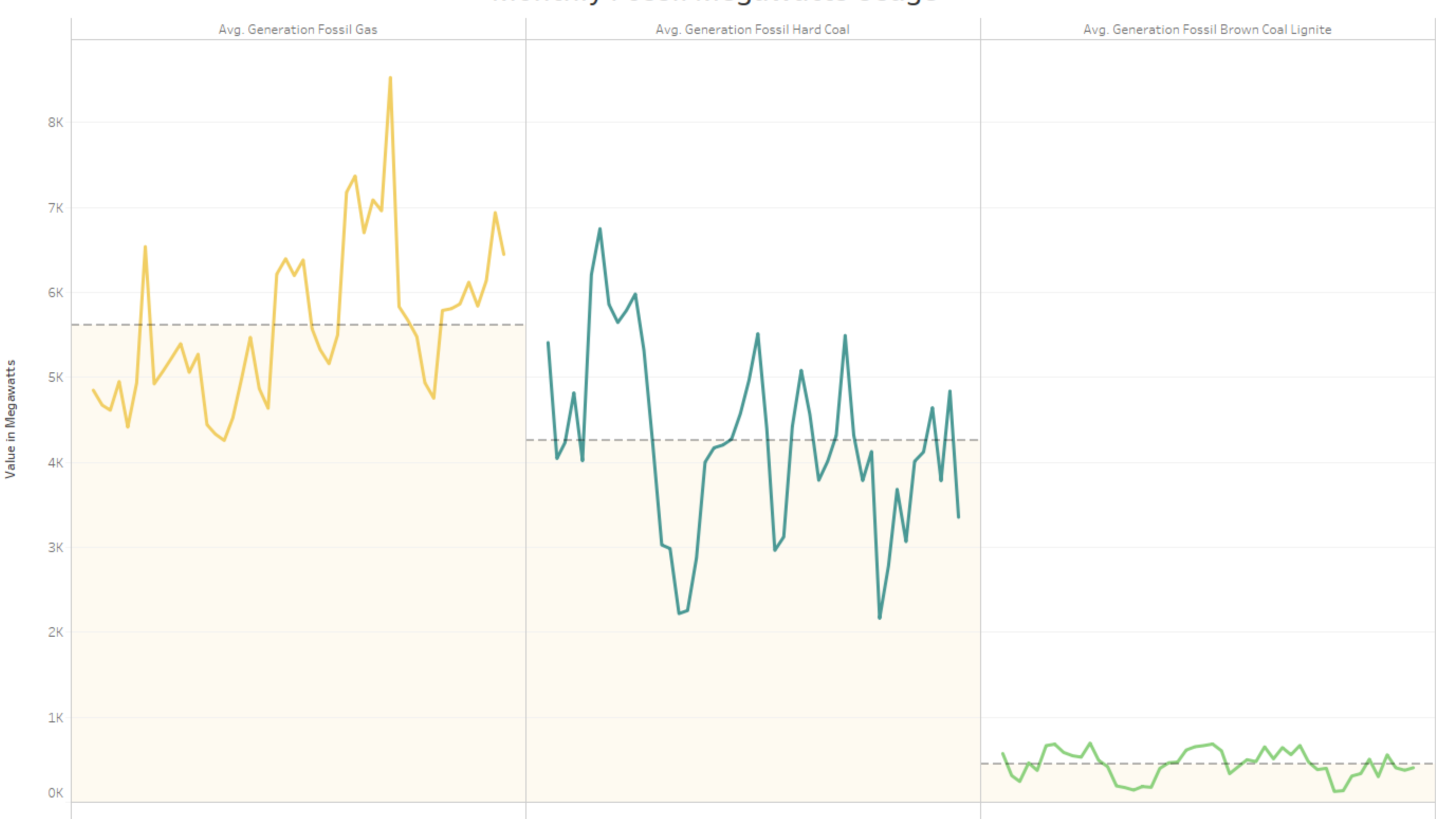
With the 5 different cities in the dataset I wanted to explore the quantitative difference between the locations and see if there was any interesting information that varied between them. Well, the answer was that there isn't, but this wasn't an unfortunate discovery it was a happy one! This means that these prices were being fairly priced between all of the different locations within the region, meaning that the pricing is independent of location which was a good thing to learn.

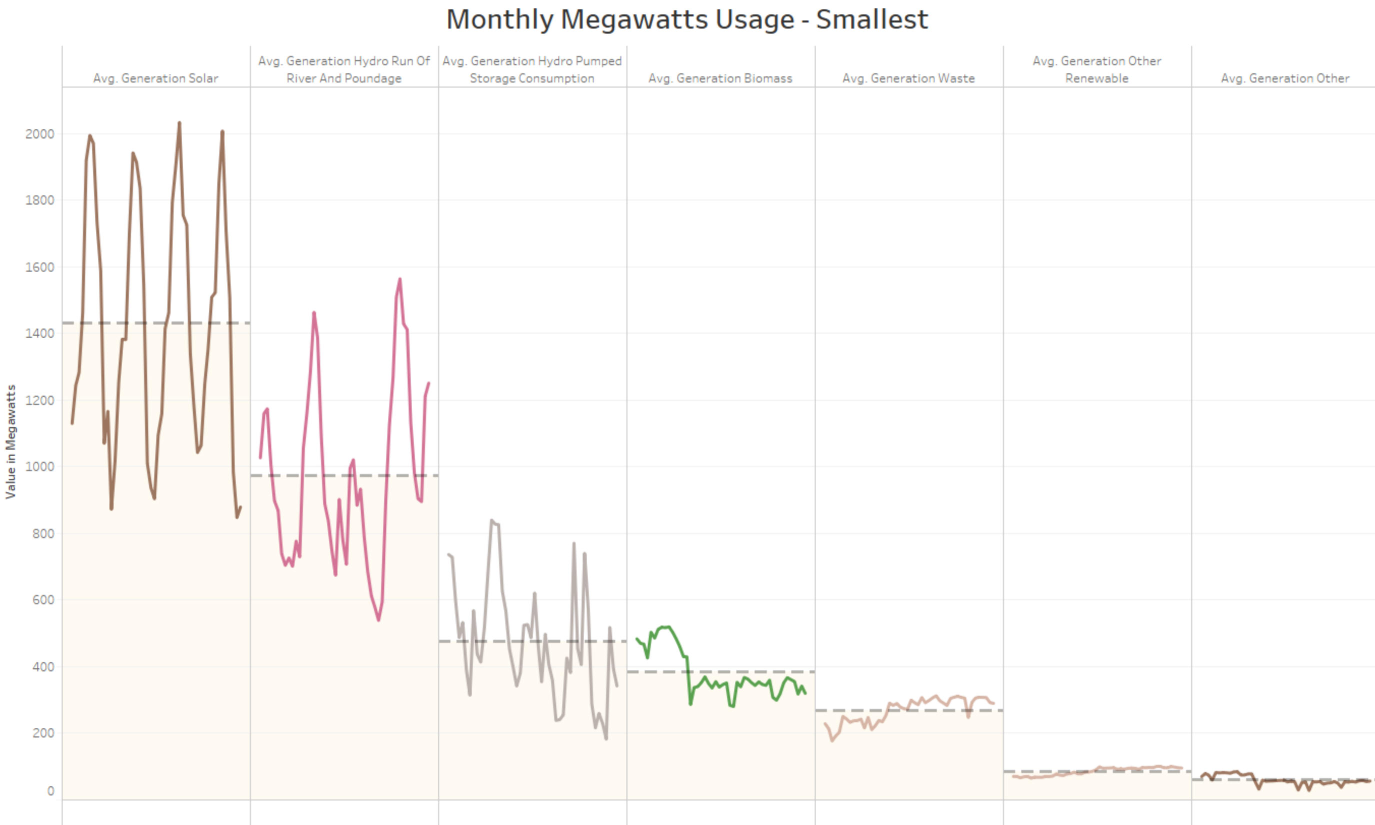


Average Electricity Price by Year



Monthly Fossil Megawatts Usage





Sqoop: (Shelby/Claire)

Within Cloudera, we used Sqoop to transfer the merged dataset from Hive to mySQL. However, we eventually ran all of our SQL queries using Pyspark instead of doing them in Cloudera.

```
mysql> SHOW TABLES;
+------------------------+
| Tables_in_group_project |
+------------------------+
| energy                 |
+------------------------+
1 row in set (0.00 sec)

mysql> DESCRIBE energy;
+-------------------------------------------+-----------+------+-----+--------
----------+---------------------------+
| Field                                     | Type      | Null | Key | Default
          | Extra                     |
+-------------------------------------------+-----------+------+-----+--------
----------+---------------------------+
| time                                      | timestamp | NO   |     | CURRENT
_TIMESTAMP | on update CURRENT_TIMESTAMP |
| generation_biomass                        | float     | YES  |     | NULL
          |                           |
| generation_fossil_brown_coal_lignite      | float     | YES  |     | NULL
          |                           |
| generation_fossil_coal_derived_gas        | float     | YES  |     | NULL
          |                           |
| generation_fossil_gas                     | float     | YES  |     | NULL
          |                           |
| generation_fossil_hard_coal               | float     | YES  |     | NULL
          |                           |
| generation_fossil_oil                     | float     | YES  |     | NULL
          |                           |
| generation_fossil_oil_shale               | float     | YES  |     | NULL
          |                           |
| generation_fossil_peat                    | float     | YES  |     | NULL
          |                           |
| generation_geothermal                     | float     | YES  |     | NULL
          |                           |
| generation_hydro_pumped_storage_aggregated | float     | YES  |     | NULL
          |                           |
| generation_hydro_pumped_storage_consumption | float     | YES  |     | NULL
          |                           |
| generation_hydro_run_of_river_and_poundage | float     | YES  |     | NULL
```

Spark: (Claire)

With the successful upload of both tables, now I can see what to visualize. The Weather dataset has columns that a layman can easily understand what to visualize but as for the energy dataset, this will be more helpful for users in the electrical field as the terms would be familiar to them.

a) Viewing the maximum and minimum temperatures in the city of Madrid this can be helpful for a weather channel to forecast how the temperatures might look like for the next few hours during a football match for example.

```
Spark.sql("select * from Weather where city_name = 'Madrid' or weather_main = 'clouds'").createTempView("madrid")
```

```
Spark.sql("select dt_iso,city_name,temp_max,temp_min from Weather where city_name = 'Madrid' ORDER BY temp_max desc").show()
```

```
+-------------------+---------+--------+--------+
|             dt_iso|city_name|temp_max|temp_min|
+-------------------+---------+--------+--------+
|2016-09-06 14:00:00|   Madrid|  316.48|  307.04|
|2016-09-06 15:00:00|   Madrid|  316.48|  308.15|
|2016-09-07 15:00:00|   Madrid|  315.37|  309.05|
|2017-07-13 18:00:00|   Madrid|  315.15|  310.15|
|2016-09-06 13:00:00|   Madrid|  314.82|  303.71|
|2016-09-07 14:00:00|   Madrid|  314.82|  307.04|
|2015-07-07 12:00:00|   Madrid|  314.75|  305.15|
|2015-06-28 12:00:00|   Madrid|  314.65|  304.15|
|2015-07-08 12:00:00|   Madrid|  314.55|  305.15|
|2015-07-07 11:00:00|   Madrid|  314.25|  303.15|
|2015-07-14 12:00:00|   Madrid|  314.25|  303.71|
|2015-07-06 19:00:00|   Madrid|  314.15|  310.75|
|2015-07-06 20:00:00|   Madrid|  314.15|  309.75|
|2017-07-14 18:00:00|   Madrid|  314.15|  310.15|
|2017-07-13 20:00:00|   Madrid|  314.15|  310.15|
|2017-07-13 17:00:00|   Madrid|  314.15|  310.15|
|2017-07-14 19:00:00|   Madrid|  314.15|  310.15|
|2018-08-03 18:00:00|   Madrid|  314.15|  310.15|
|2017-07-13 19:00:00|   Madrid|  314.15|  310.15|
|2018-08-03 17:00:00|   Madrid|  314.15|  310.15|
+-------------------+---------+--------+--------+
```

b) This next visualization, I want to see how the generation_hydro_water_reservoir affects the price_actual column

```
Spark.sql("select generation_hydro_water_reservoir,price_actual from Energy ORDER BY price_actual").show()
```

```
+--------------------------------+------------+
|generation_hydro_water_reservoir|price_actual|
+--------------------------------+------------+
|                          3270.0|       10.07|
|                          3765.0|       10.18|
|                          3077.0|       10.66|
|                          3869.0|       10.77|
|                          3835.0|      100.02|
|                          4243.0|      100.03|
|                          4128.0|      100.04|
|                          3965.0|      100.05|
|                          4135.0|      100.24|
|                          4492.0|      100.29|
|                          4346.0|      100.39|
|                          3674.0|      100.45|
|                          1686.0|      100.46|
|                          4741.0|      100.49|
|                          5884.0|      100.52|
|                          3864.0|      100.55|
|                          1176.0|      100.58|
|                          3866.0|      100.76|
|                          5501.0|      100.95|
|                          5307.0|      101.15|
+--------------------------------+------------+
only showing top 20 rows
```

Looking at the outcome of this query, we notice that the generation_hydro_wter_reservoir doesn't really affect the price in the sense that, you will think that the higher the value, the higher the price bur I sorted my output in order of price ascending and we notice that a generation_hydro_water_reservoir of 3270.0 incures a price of $10.07 but a value of 1176.0 which is obviously lower incures of price of up to 100.58.

```
spark.sql("select generation_hydro_water_reservoir,price_actual from Energy ORDER BY generation_hydro_water_reservoir desc").show()
```

```
+--------------------------------+------------+
|generation_hydro_water_reservoir|price_actual|
+--------------------------------+------------+
|                           999.0|       46.29|
|                           999.0|       42.88|
|                           999.0|       44.59|
|                           999.0|       76.86|
|                           999.0|       66.82|
|                           999.0|       51.29|
|                           999.0|       54.81|
|                           999.0|       60.24|
|                           999.0|       61.96|
|                           999.0|       59.07|
|                           999.0|       45.42|
|                           999.0|       58.44|
|                           999.0|       79.33|
|                           999.0|       59.43|
|                           999.0|       69.52|
|                           999.0|       73.05|
|                           999.0|       53.18|
|                           999.0|       75.81|
|                           998.0|       75.95|
|                           998.0|       52.66|
+--------------------------------+------------+
only showing top 20 rows
```

The screenshot above, I sorted my results in ascending order of the values of generation_hydro_water and we notice that even though the values are the same for most of the rows, the price_actual is still not the same. Hence analyzing this I will say the generation_hydro_water_reservoir does not have a high impact on the price_actual column

c) You notice above that I created a tempview of Madrid and Barcelona, this can be helpful when you want to filter out some cities and work with. For my case I am using Madrid and Barcelona, I created a tempview for both cities and my first query will be merging these two tables together with the help of union operation and I am sorting it in ascending order of temperature.

```
spark.sql("select * from madrid union select * from barcelonatable order by temp").show(50)
```

```
+-------------------+---------+--------+--------+--------+--------+--------+----------+--------+-------+-------+-------+----------+----------+------------+----------------
|             dt_iso|city_name|    temp|temp_min|temp_max|pressure|humidity|wind_speed|wind_deg|rain_1h|rain_3h|snow_3h|clouds_all|weather_id|weather_main|weather_descript
+-------------------+---------+--------+--------+--------+--------+--------+----------+--------+-------+-------+-------+----------+----------+------------+----------------
|2015-02-23 10:00:00|Barcelona|  262.24|  262.24|  262.24|    1007|       0|         3|     335|    0.0|    0.0|    0.0|        24|       801|      clouds|          few clo
|2015-02-08 06:00:00|   Madrid| 264.132| 264.132| 264.132|     961|      80|         1|       8|    0.0|    0.0|    0.0|         0|       800|       clear|        sky is cl
|2015-01-24 05:00:00|   Madrid| 264.428| 264.428| 264.428|     965|      64|         1|     348|    0.0|    0.0|    0.0|         0|       800|       clear|        sky is cl
|2015-01-24 06:00:00|   Madrid| 264.428| 264.428| 264.428|     965|      64|         1|     348|    0.0|    0.0|    0.0|         0|       800|       clear|        sky is cl
|2015-02-07 02:00:00|   Madrid| 265.091| 265.091| 265.091|     954|      73|         1|     295|    0.0|    0.0|    0.0|        32|       802|      clouds|    scattered clo
|2015-02-07 03:00:00|   Madrid| 265.091| 265.091| 265.091|     954|      73|         1|     295|    0.0|    0.0|    0.0|        32|       802|      clouds|    scattered clo
|2015-01-24 04:00:00|   Madrid| 265.261| 265.261| 265.261|     964|      69|         1|     276|    0.0|    0.0|    0.0|         0|       800|       clear|        sky is cl
|2015-01-24 02:00:00|   Madrid| 265.261| 265.261| 265.261|     964|      69|         1|     276|    0.0|    0.0|    0.0|         0|       800|       clear|        sky is cl
|2015-01-24 03:00:00|   Madrid| 265.261| 265.261| 265.261|     964|      69|         1|     276|    0.0|    0.0|    0.0|         0|       800|       clear|        sky is cl
|2015-01-01 05:00:00|   Madrid| 265.442| 265.442| 265.442|     972|      64|         0|     240|    0.0|    0.0|    0.0|         0|       800|       clear|        sky is cl
|2015-01-01 06:00:00|   Madrid| 265.442| 265.442| 265.442|     972|      64|         0|     240|    0.0|    0.0|    0.0|         0|       800|       clear|        sky is cl
|2015-01-01 07:00:00|   Madrid| 265.442| 265.442| 265.442|     972|      64|         0|     240|    0.0|    0.0|    0.0|         0|       800|       clear|        sky is cl
|2015-02-08 05:00:00|   Madrid|265.6245|265.6245|265.6245|     993|      87|         2|      15|    0.0|    0.0|    0.0|         0|       800|       clear|        sky is cl
|2015-02-07 04:00:00|   Madrid| 265.638| 265.638| 265.638|     954|      75|         1|     277|    0.0|    0.0|    0.0|         8|       800|       clear|        sky is cl
|2015-01-02 05:00:00|   Madrid| 265.902| 265.902| 265.902|     975|      62|         1|      13|    0.0|    0.0|    0.0|         0|       800|       clear|        sky is cl
|2015-01-02 07:00:00|   Madrid| 265.902| 265.902| 265.902|     975|      62|         1|      13|    0.0|    0.0|    0.0|         0|       800|       clear|        sky is cl
|2015-01-02 06:00:00|   Madrid| 265.902| 265.902| 265.902|     975|      62|         1|      13|    0.0|    0.0|    0.0|         0|       800|       clear|        sky is cl
|2015-01-24 01:00:00|   Madrid|266.0235|266.0235|266.0235|     964|      69|         1|     138|    0.0|    0.0|    0.0|         0|       800|       clear|        sky is cl
|2015-01-26 05:00:00|   Madrid| 266.024| 266.024| 266.024|     967|      69|         1|      31|    0.0|    0.0|    0.0|         0|       800|       clear|        sky is cl
|2015-01-26 07:00:00|   Madrid| 266.024| 266.024| 266.024|     967|      69|         1|      31|    0.0|    0.0|    0.0|         0|       800|       clear|        sky is cl
|2015-01-26 06:00:00|   Madrid| 266.024| 266.024| 266.024|     967|      69|         1|      31|    0.0|    0.0|    0.0|         0|       800|       clear|        sky is cl
|2015-01-24 07:00:00|   Madrid|266.1065|266.1065|266.1065|     966|      68|         1|     177|    0.0|    0.0|    0.0|         0|       800|       clear|        sky is cl
|2015-02-07 06:00:00|   Madrid| 266.149| 266.149| 266.149|     954|      76|         1|     260|    0.0|    0.0|    0.0|        56|       803|      clouds|       broken clo
|2015-02-07 05:00:00|   Madrid| 266.149| 266.149| 266.149|     954|      76|         1|     260|    0.0|    0.0|    0.0|        56|       803|      clouds|       broken clo
|2015-02-07 07:00:00|   Madrid| 266.149| 266.149| 266.149|     954|      76|         1|     260|    0.0|    0.0|    0.0|        56|       803|      clouds|       broken clo
|2015-01-01 03:00:00|   Madrid| 266.186| 266.186| 266.186|     971|      64|         1|     273|    0.0|    0.0|    0.0|         0|       800|       clear|        sky is cl
|2015-01-01 04:00:00|   Madrid| 266.186| 266.186| 266.186|     971|      64|         1|     273|    0.0|    0.0|    0.0|         0|       800|       clear|        sky is cl
|2015-01-01 02:00:00|   Madrid| 266.186| 266.186| 266.186|     971|      64|         1|     273|    0.0|    0.0|    0.0|         0|       800|       clear|        sky is cl
```

d) Our next visualization is one between the city, temperatiure and he weather. My idea here is to find out how the temperature is related to the weather_main column which basically just tells you how the weather is.

```
spark.sql("select city_name,temp,weather_main from Weather order by temp desc").show(50)
```

```
+---------+------+------------+
|city_name|  temp|weather_main|
+---------+------+------------+
|  Seville| 315.6|       clear|
|  Seville|315.54|       clear|
|  Seville|315.15|       clear|
|  Seville|315.15|       clear|
|  Seville|315.15|       clear|
|  Seville|315.15|       clear|
|  Seville|315.03|       clear|
|  Seville|314.76|       clear|
|  Seville|314.76|       clear|
|  Seville|314.76|       clear|
|  Seville|314.76|       clear|
|  Seville|314.76|       clear|
|  Seville|314.76|       clear|
|  Seville| 314.7|       clear|
|  Seville|314.63|       clear|
|  Seville|314.54|       clear|
|  Seville|314.54|       clear|
|  Seville|314.54|       clear|
|  Seville|314.51|       clear|
|  Seville|314.33|       clear|
|  Seville|314.33|       clear|
|  Seville| 314.3|       clear|
|  Seville|314.29|       clear|
|  Seville|314.26|       clear|
```

```
spark.sql("select city_name,temp,weather_main from Weather order by temp").show(50)
```

```
+----------+--------+------------+
| city_name|    temp|weather_main|
+----------+--------+------------+
| Barcelona|  262.24|      clouds|
|    Madrid| 264.132|       clear|
|    Madrid| 264.428|       clear|
|    Madrid| 264.428|       clear|
|    Madrid| 265.091|      clouds|
|    Madrid| 265.091|      clouds|
|    Madrid| 265.261|       clear|
|    Madrid| 265.261|       clear|
|    Madrid| 265.261|       clear|
|    Madrid| 265.442|       clear|
|    Madrid| 265.442|       clear|
|    Madrid| 265.442|       clear|
|    Madrid|265.6245|       clear|
|    Madrid| 265.638|       clear|
|    Madrid| 265.902|       clear|
|    Madrid| 265.902|       clear|
|    Madrid| 265.902|       clear|
|    Madrid|266.0235|       clear|
|    Madrid| 266.024|       clear|
|    Madrid| 266.024|       clear|
|    Madrid| 266.024|       clear|
|    Madrid|266.1065|       clear|
|    Madrid| 266.149|      clouds|
|    Madrid| 266.149|      clouds|
|    Madrid| 266.149|      clouds|
|    Madrid| 266.186|       clear|
|    Madrid| 266.186|       clear|
|    Madrid| 266.186|       clear|
```

From the screenshot above I could come up with a conclusion that , when the temperature is high, the weather is clear and also the city of Seville has high temperatures compared to the other cities also Madrid city has lower temperature compared to the other cities. In order for me to make this conclusion, I viewed the data in ascending order of temp and descending order of temp.

e) My next query is a simple one where I want to view those cities that had mist weather. This can be helpful when a user wants to know if a particular city has a history of this weather type

```
Spark.sql("select * from Weather where weather_description = 'mist'").show()
```

```
+-------------------+---------+------+--------+--------+--------+--------+----------+--------+-------+-------+-------+----------+----------+------------+-------------------+-
|             dt_iso|city_name|  temp|temp_min|temp_max|pressure|humidity|wind_speed|wind_deg|rain_1h|rain_3h|snow_3h|clouds_all|weather_id|weather_main|weather_description|w
+-------------------+---------+------+--------+--------+--------+--------+----------+--------+-------+-------+-------+----------+----------+------------+-------------------+-
|2015-03-04 08:00:00| Valencia|287.19|  286.15|  288.05|    1024|      87|         1|       0|    0.0|    0.0|    0.0|        90|       701|        mist|               mist|
|2015-03-04 09:00:00| Valencia| 286.7|  285.15|  288.05|    1025|      93|         1|       0|    0.0|    0.0|    0.0|        90|       701|        mist|               mist|
|2015-03-04 10:00:00| Valencia|287.19|  286.15|  288.05|    1026|      87|         0|       0|    0.0|    0.0|    0.0|        90|       701|        mist|               mist|
|2015-03-04 11:00:00| Valencia| 288.1|  288.05|  288.15|    1026|      82|         0|     104|    0.0|    0.0|    0.0|        40|       701|        mist|               mist|
|2015-11-07 20:00:00| Valencia|292.15|  292.15|  292.15|    1028|      88|         0|       0|    0.0|    0.0|    0.0|         0|       701|        mist|               mist|
|2015-11-07 22:00:00| Valencia|289.15|  289.15|  289.15|    1029|      93|         2|     320|    0.0|    0.0|    0.0|         0|       701|        mist|               mist|
|2015-11-08 02:00:00| Valencia|286.15|  286.15|  286.15|    1029|     100|         1|      54|    0.0|    0.0|    0.0|         0|       701|        mist|               mist|
|2015-11-08 04:00:00| Valencia|286.15|  286.15|  286.15|    1029|      93|         1|       0|    0.0|    0.0|    0.0|         0|       701|        mist|               mist|
|2015-11-09 01:00:00| Valencia|287.15|  287.15|  287.15|    1031|      93|         1|       0|    0.0|    0.0|    0.0|        48|       701|        mist|               mist|
|2015-11-09 03:00:00| Valencia|285.15|  285.15|  285.15|    1031|     100|         1|     290|    0.0|    0.0|    0.0|         8|       701|        mist|               mist|
|2015-11-09 22:00:00| Valencia|287.15|  287.15|  287.15|    1032|      93|         0|       0|    0.0|    0.0|    0.0|         0|       701|        mist|               mist|
|2015-11-10 00:00:00| Valencia|285.15|  285.15|  285.15|    1031|     100|         1|       0|    0.0|    0.0|    0.0|         0|       701|        mist|               mist|
|2015-11-10 02:00:00| Valencia|284.15|  284.15|  284.15|    1031|      93|         1|     301|    0.0|    0.0|    0.0|         0|       701|        mist|               mist|
|2015-11-10 04:00:00| Valencia|283.15|  283.15|  283.15|    1030|     100|         1|       0|    0.0|    0.0|    0.0|         0|       701|        mist|               mist|
|2015-11-11 22:00:00| Valencia|286.15|  286.15|  286.15|    1025|      93|         1|       0|    0.0|    0.0|    0.0|         0|       701|        mist|               mist|
|2015-11-12 00:00:00| Valencia|285.15|  285.15|  285.15|    1025|      93|         1|       0|    0.0|    0.0|    0.0|         0|       701|        mist|               mist|
|2015-11-12 02:00:00| Valencia|284.15|  284.15|  284.15|    1025|      93|         0|       0|    0.0|    0.0|    0.0|        20|       701|        mist|               mist|
|2015-11-12 04:00:00| Valencia|285.15|  285.15|  285.15|    1025|      93|         1|       0|    0.0|    0.0|    0.0|        75|       701|        mist|               mist|
|2015-11-14 02:00:00| Valencia|286.15|  286.15|  286.15|    1032|      93|         1|     307|    0.0|    0.0|    0.0|        20|       701|        mist|               mist|
|2015-11-14 04:00:00| Valencia|285.15|  285.15|  285.15|    1032|      93|         1|     300|    0.0|    0.0|    0.0|        32|       701|        mist|               mist|
+-------------------+---------+------+--------+--------+--------+--------+----------+--------+-------+-------+-------+----------+----------+------------+-------------------+-
only showing top 20 rows
```

f) using avg and max function, I am able to get the average temperature and maximum temperature form my dataset and the maximu pressure. This kind of analysis can be useful to users who want to compare the difference in temperature values for the previous year and current year

```
Spark.sql("select avg(temp_max) from Weather").show()
```

```
+----------------------------+
|avg(CAST(temp_max AS DOUBLE))|
+----------------------------+
|            291.0912665669784|
+----------------------------+
```

```
[ ] Spark.sql("select max(pressure) from Weather").show()
```

```
+-------------+
|max(pressure)|
+-------------+
|        99915|
+-------------+
```

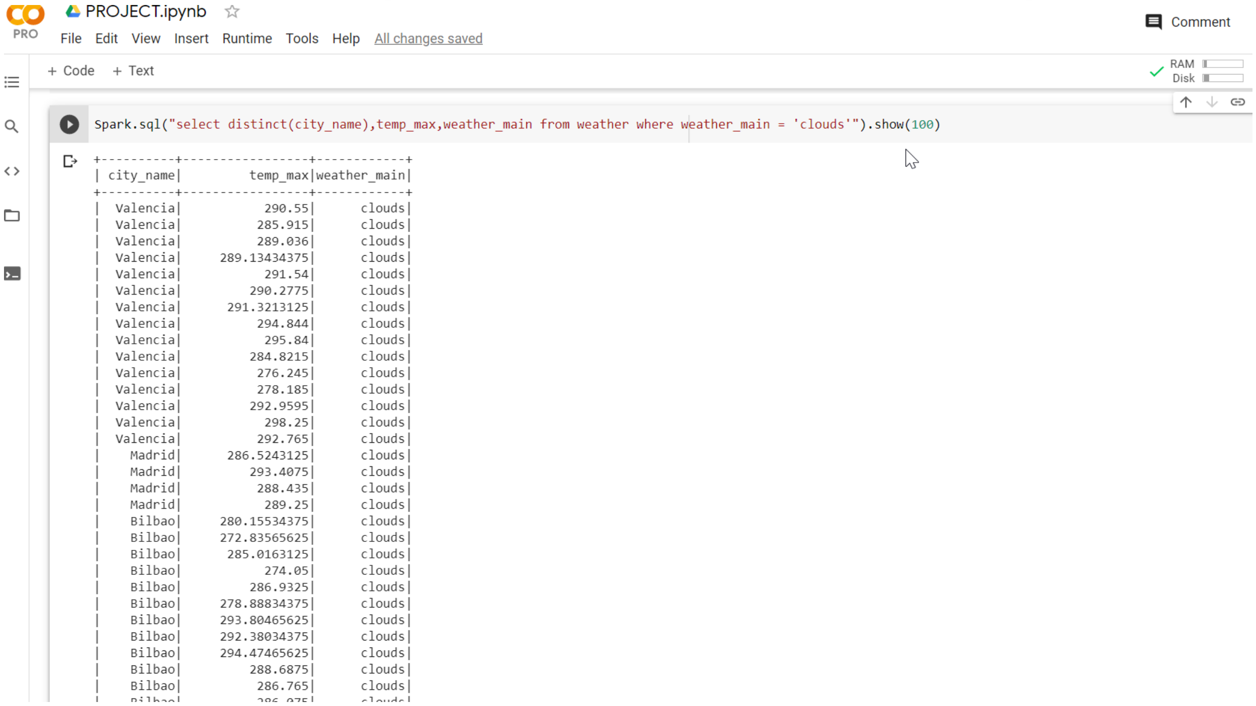
e) with the Energy dataset, I could view solar focast for a day ahead and the price

```
Spark.sql("select forecast_solar_day_ahead,price_day_ahead from Energy ORDER by price_day_ahead").show()
```

```
+------------------------+---------------+
|forecast_solar_day_ahead|price_day_ahead|
+------------------------+---------------+
|                     5.0|           10.0|
|                  4632.0|           10.0|
|                   577.0|           10.0|
|                     0.0|           10.0|
|                     5.0|           10.0|
|                     1.0|           10.0|
|                   241.0|           10.0|
|                     0.0|           10.0|
|                  4507.0|           10.0|
|                  4947.0|           10.0|
|                   527.0|           10.0|
|                    72.0|           10.0|
|                     0.0|           10.0|
|                     2.0|           10.0|
|                   127.0|           10.0|
|                     5.0|           10.0|
|                  3451.0|           10.0|
|                  3452.0|           10.0|
|                  2070.0|           10.0|
|                     2.0|           10.0|
+------------------------+---------------+
only showing top 20 rows
```

From our result , we notice that the forecast_solar_day_ahead doesn’t have a high impact in the price_day_ahead as the values of the forecast are quite different but the price is same

g) Using the distinct function as our dataset is very large and for this query I do not want repetitions, I can view cities , their maximum temperature and how the weather looks like with such temperatures



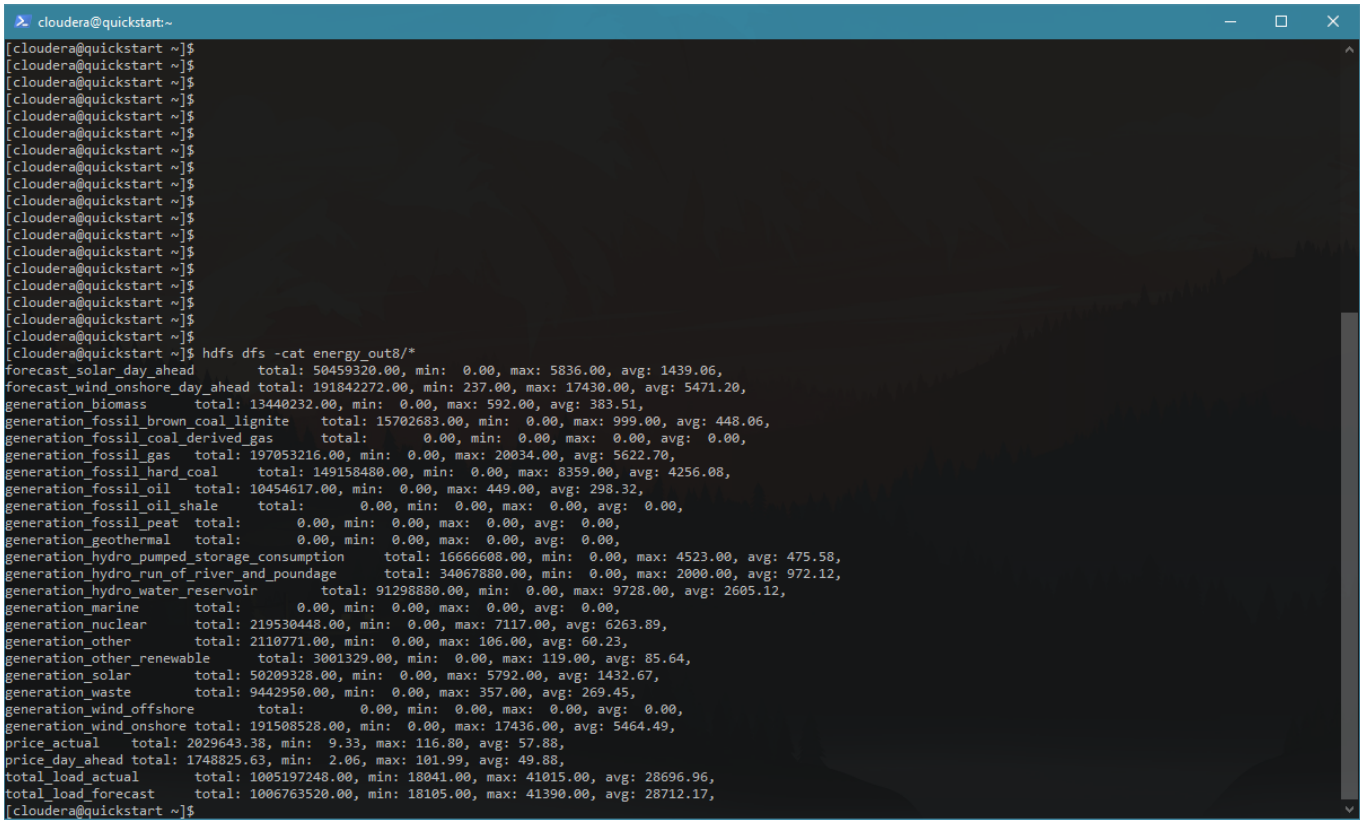
MapReduce: (Scott)

Starting with the basics in MapReduce I wanted to get some descriptive statistics for each column in our dataset. Fortunately, our dataset makes this easy since nearly all the columns are of the same type. Since I only have one datatype to worry about, I can get away with creating only one reducer to find the mean, min, and max. Each row is split up by column and written to the reducer with the field name as the key.

I wish to calculate a few more descriptive statistics, such as the median, quartiles, standard deviation, and variance but found them difficult to calculate due to the nature of MapReduce. I believe I can overcome a few of these limitations by using a secondary sort, changing the algorithm used to calculate the statistic or only calculating an approximation. In addition, I also want experiment with joining the weather dataset to start doing some complex grouping. A mapper-side join currently is not be possible without some preprocessing since the datasets are different lengths and some rows may be missing from the energy dataset.

cloudera@quickstart:~/project/Energy_Demand_Analysis/MapReduce/output

```
[cloudera@quickstart output]$ cat job_output.log
20/10/30 04:27:44 INFO mapreduce.Job:  map 0% reduce 0%
20/10/30 04:27:58 INFO mapreduce.Job:  map 100% reduce 0%
20/10/30 04:28:17 INFO mapreduce.Job:  map 100% reduce 100%
20/10/30 04:28:17 INFO mapreduce.Job: Job job_1604032870144_0008 completed successfully
20/10/30 04:28:17 INFO mapreduce.Job: Counters: 49
        File System Counters
                FILE: Number of bytes read=28038648
                FILE: Number of bytes written=56365263
                FILE: Number of read operations=0
                FILE: Number of large read operations=0
                FILE: Number of write operations=0
                HDFS: Number of bytes read=6062761
                HDFS: Number of bytes written=2170
                HDFS: Number of read operations=6
                HDFS: Number of large read operations=0
                HDFS: Number of write operations=2
        Job Counters
                Launched map tasks=1
                Launched reduce tasks=1
                Data-local map tasks=1
                Total time spent by all maps in occupied slots (ms)=10581
                Total time spent by all reduces in occupied slots (ms)=15436
                Total time spent by all map tasks (ms)=10581
                Total time spent by all reduce tasks (ms)=15436
                Total vcore-milliseconds taken by all map tasks=10581
                Total vcore-milliseconds taken by all reduce tasks=15436
                Total megabyte-milliseconds taken by all map tasks=10834944
                Total megabyte-milliseconds taken by all reduce tasks=15806464
        Map-Reduce Framework
                Map input records=35065
                Map output records=911263
                Map output bytes=26216116
                Map output materialized bytes=28038648
                Input split bytes=136
                Combine input records=0
                Combine output records=0
                Reduce input groups=26
                Reduce shuffle bytes=28038648
                Reduce input records=911263
                Reduce output records=26
                Spilled Records=1822526
                Shuffled Maps =1
                Failed Shuffles=0
                Merged Map outputs=1
                GC time elapsed (ms)=221
                CPU time spent (ms)=18370
                Physical memory (bytes) snapshot=714465280
                Virtual memory (bytes) snapshot=3139694592
                Total committed heap usage (bytes)=643825664
        Shuffle Errors
                BAD_ID=0
                CONNECTION=0
                IO_ERROR=0
                WRONG_LENGTH=0
                WRONG_MAP=0
                WRONG_REDUCE=0
        File Input Format Counters
                Bytes Read=6062625
        File Output Format Counters
                Bytes Written=2170

[cloudera@quickstart output]$
```



```
cloudera@quickstart:~
[cloudera@quickstart ~]$
[cloudera@quickstart ~]$
[cloudera@quickstart ~]$
[cloudera@quickstart ~]$
[cloudera@quickstart ~]$
[cloudera@quickstart ~]$
[cloudera@quickstart ~]$
[cloudera@quickstart ~]$
[cloudera@quickstart ~]$
[cloudera@quickstart ~]$
[cloudera@quickstart ~]$
[cloudera@quickstart ~]$
[cloudera@quickstart ~]$
[cloudera@quickstart ~]$
[cloudera@quickstart ~]$
[cloudera@quickstart ~]$
[cloudera@quickstart ~]$
[cloudera@quickstart ~]$
[cloudera@quickstart ~]$ hdfs dfs -cat energy_out8/*
forecast_solar_day_ahead        total: 50459320.00, min:  0.00, max: 5836.00, avg: 1439.06,
forecast_wind_onshore_day_ahead total: 191842272.00, min: 237.00, max: 17430.00, avg: 5471.20,
generation_biomass      total: 13440232.00, min:  0.00, max: 592.00, avg: 383.51,
generation_fossil_brown_coal_lignite    total: 15702683.00, min:  0.00, max: 999.00, avg: 448.06,
generation_fossil_coal_derived_gas      total:       0.00, min:  0.00, max:  0.00, avg:  0.00,
generation_fossil_gas   total: 197053216.00, min:  0.00, max: 20034.00, avg: 5622.70,
generation_fossil_hard_coal     total: 149158480.00, min:  0.00, max: 8359.00, avg: 4256.08,
generation_fossil_oil   total: 10454617.00, min:  0.00, max: 449.00, avg: 298.32,
generation_fossil_oil_shale     total:       0.00, min:  0.00, max:  0.00, avg:  0.00,
generation_fossil_peat  total:       0.00, min:  0.00, max:  0.00, avg:  0.00,
generation_geothermal   total:       0.00, min:  0.00, max:  0.00, avg:  0.00,
generation_hydro_pumped_storage_consumption     total: 16666608.00, min:  0.00, max: 4523.00, avg: 475.58,
generation_hydro_run_of_river_and_poundage      total: 34067880.00, min:  0.00, max: 2000.00, avg: 972.12,
generation_hydro_water_reservoir        total: 91298880.00, min:  0.00, max: 9728.00, avg: 2605.12,
generation_marine       total:       0.00, min:  0.00, max:  0.00, avg:  0.00,
generation_nuclear      total: 219530448.00, min:  0.00, max: 7117.00, avg: 6263.89,
generation_other        total: 2110771.00, min:  0.00, max: 106.00, avg: 60.23,
generation_other_renewable      total: 3001329.00, min:  0.00, max: 119.00, avg: 85.64,
generation_solar        total: 50209328.00, min:  0.00, max: 5792.00, avg: 1432.67,
generation_waste        total: 9442950.00, min:  0.00, max: 357.00, avg: 269.45,
generation_wind_offshore        total:       0.00, min:  0.00, max:  0.00, avg:  0.00,
generation_wind_onshore total: 191508528.00, min:  0.00, max: 17436.00, avg: 5464.49,
price_actual    total: 2029643.38, min:  9.33, max: 116.80, avg: 57.88,
price_day_ahead total: 1748825.63, min:  2.06, max: 101.99, avg: 49.88,
total_load_actual       total: 1005197248.00, min: 18041.00, max: 41015.00, avg: 28696.96,
total_load_forecast     total: 1006763520.00, min: 18105.00, max: 41390.00, avg: 28712.17,
[cloudera@quickstart ~]$
```

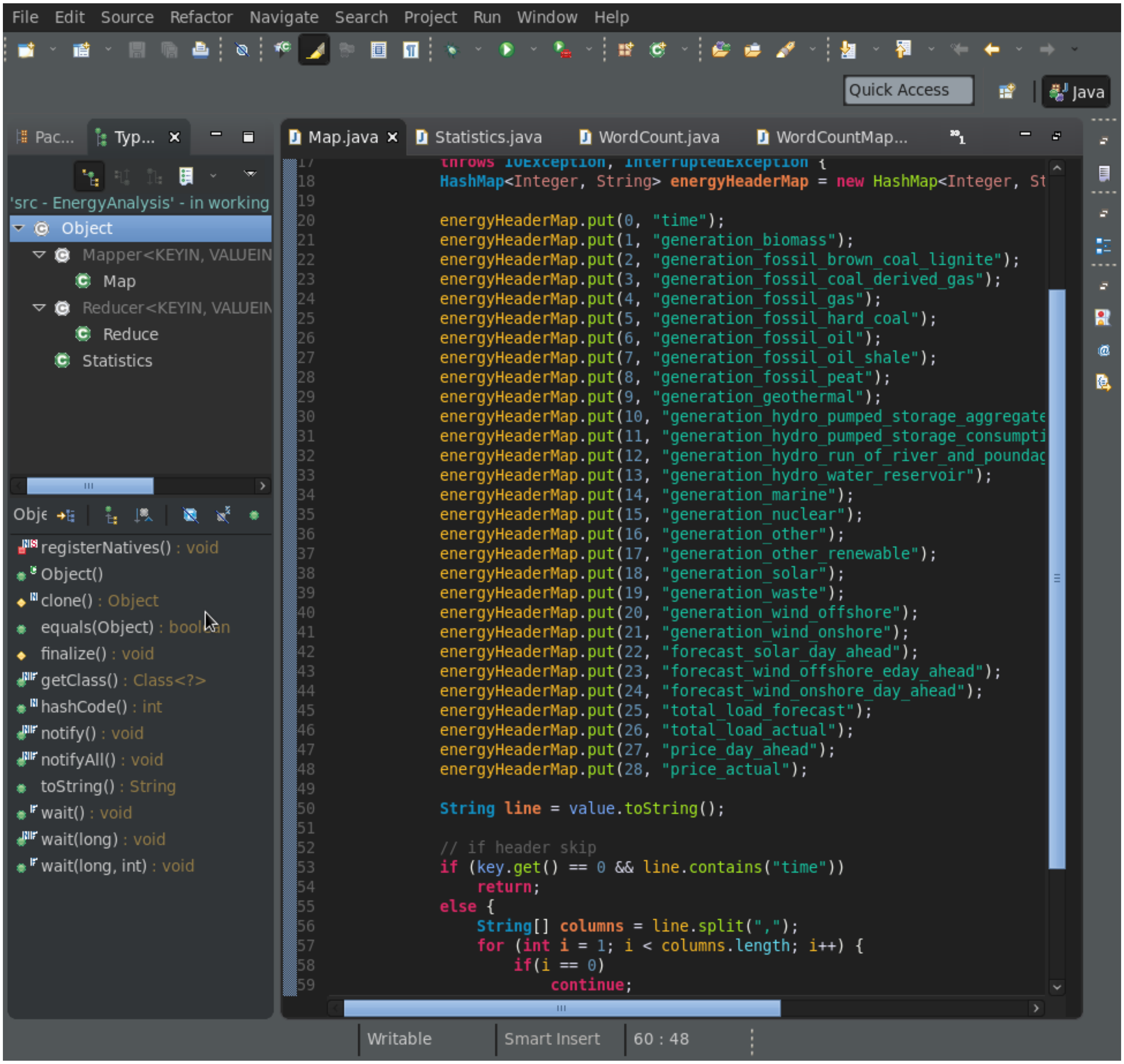
I explored our dataset using Hadoop MapReduce. In the first iteration I figured out how to parse and calculate some basic statistics for each column, but in this iteration, I wanted to explore a bit deeper. With large amounts of data algorithm efficiency becomes a primary goal. Data gathered in an analysis has an expiration date of usefulness. If it takes you a week to calculate the next week trend your analysis is useless. While our dataset is small enough that this isn't the case I wanted to explore how to do some more advanced statistics without it taking a long time to compute.

I found during my research that many of the algorithms require at least 2 passes over the dataset to compute the wanted statistic. I did not like that, so I went searching for algorithms that could approximate the value and I found the Welford's method. This is what I implemented in this iteration. It can compute the variance and in-turn the standard deviation piece by piece as the data comes in from the mapper. This method would also work great for spark streaming!

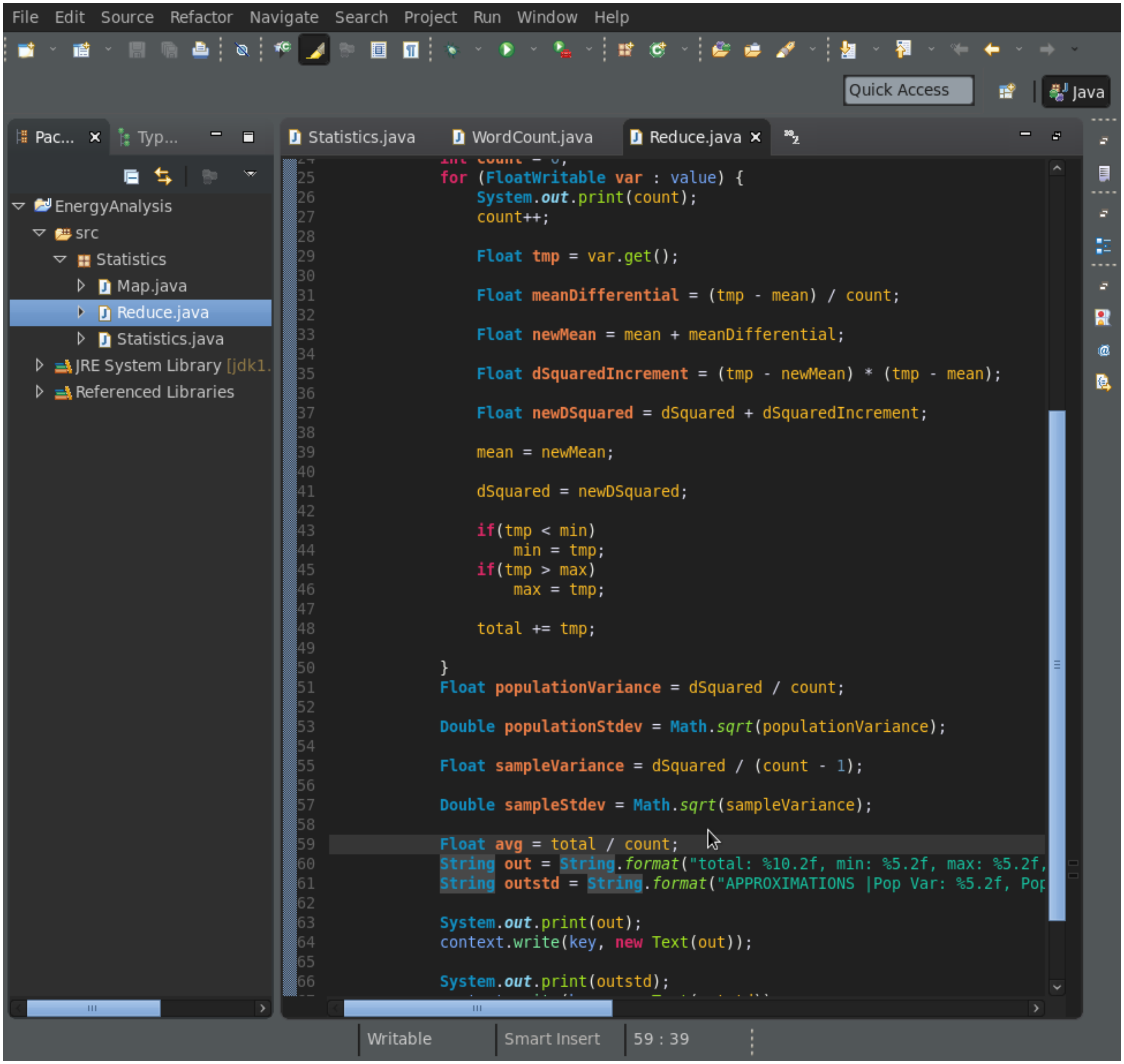
To start I placed our dataset into hdfs.

```
hdfs dfs -mkdir Energy_Demand_Analysis
hdfs dfs -cp Energy_Demand_Analysis/dataset/* Energy_Demand_Analysis/
```

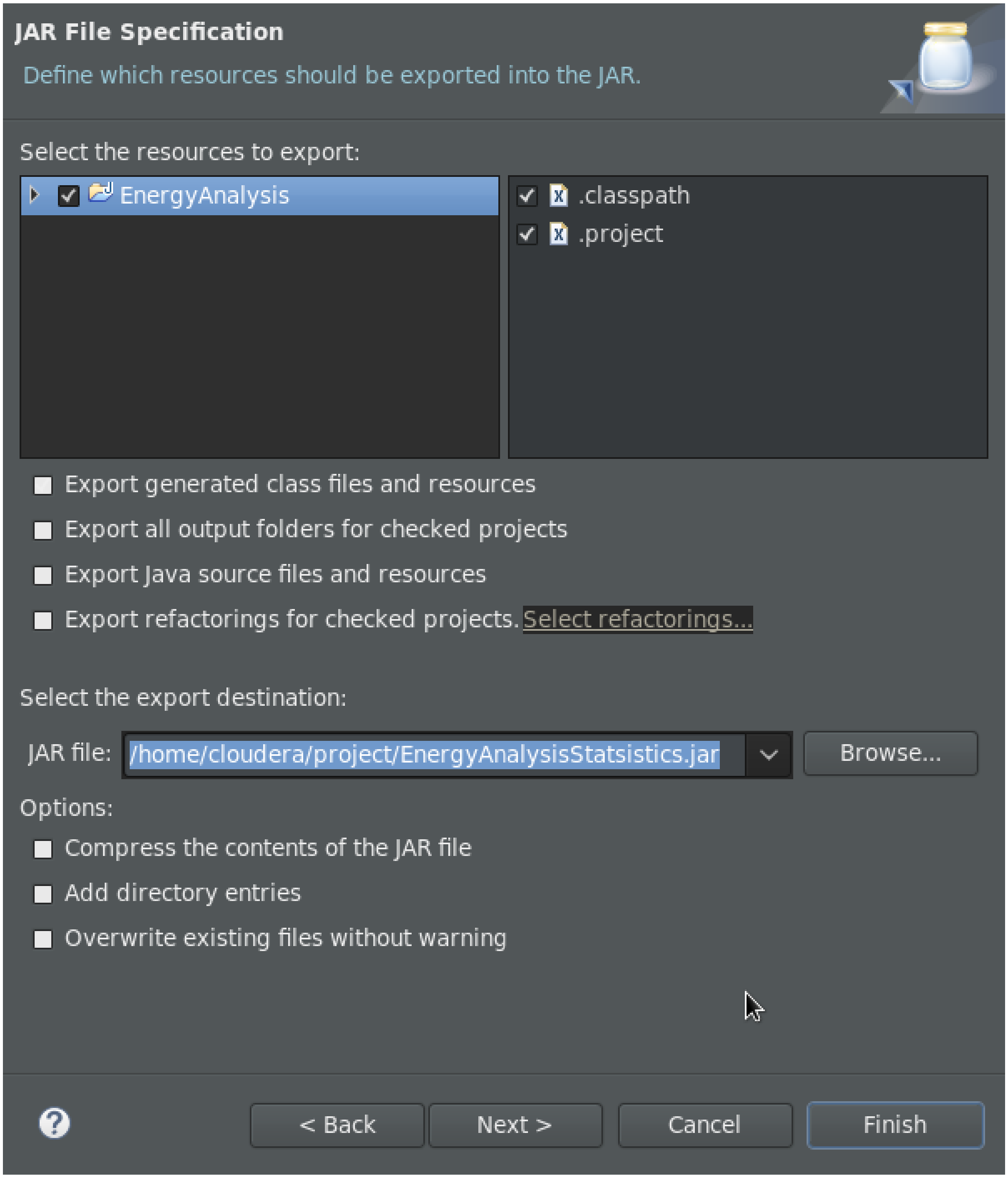
Then I wrote a Mapper that essentially takes the csv file and splits it by column and passes it off to the reducer.



The Reducer is where the meat and potatoes exist. The reducers take a feed of values and begins to calculate true values mean, min, max and approximates the value of mean and variance.



From there we export to a JAR so Hadoop can run our fantastic program.



Finally, we can run our jar on the local machine (or cluster). You can see the output to that above.

```
hadoop jar EnergyAnalysisStatsistics.jar Energy_Demand_Analysis/ energy_out
```

Cassandra: (Shelby)

Because joins aren’t possible in Cassandra, it was necessary to keep the two tables separate. Furthermore, since Cassandra operates by a query-first approach, I created several tables within Cassandra such that each table was designed for a specific query. Though it did result it duplication of data, this design is good for high-load queries that usually happened in big data. The insights gleaned from these queries seemed rather unhelpful compared to the query capabilities of HQL and mySQL. Whereas HQL/mySQL can perform direct analysis on the data (such as calculating averages, join functions, etc.), it seems like there would have to be some secondary analysis step performed with any data returned from a Cassandra query.

‘Cassandra Tables Creation.cql’ – This file contains the script that was used to create and load data into five different Cassandra tables. Because the data is just text, the class used was SimpleStrategy. A replication factor of 3 was arbitrarily decided upon.

‘Cassandra Queries.cql’ – This file contains the queries used for each table. The result of the queries was stored into a unique txt file.

‘Cassandra Results” – This folder contains the results of the five .cql queries used for each of the Cassandra tables, as well as screenshots of the successfully created tables.

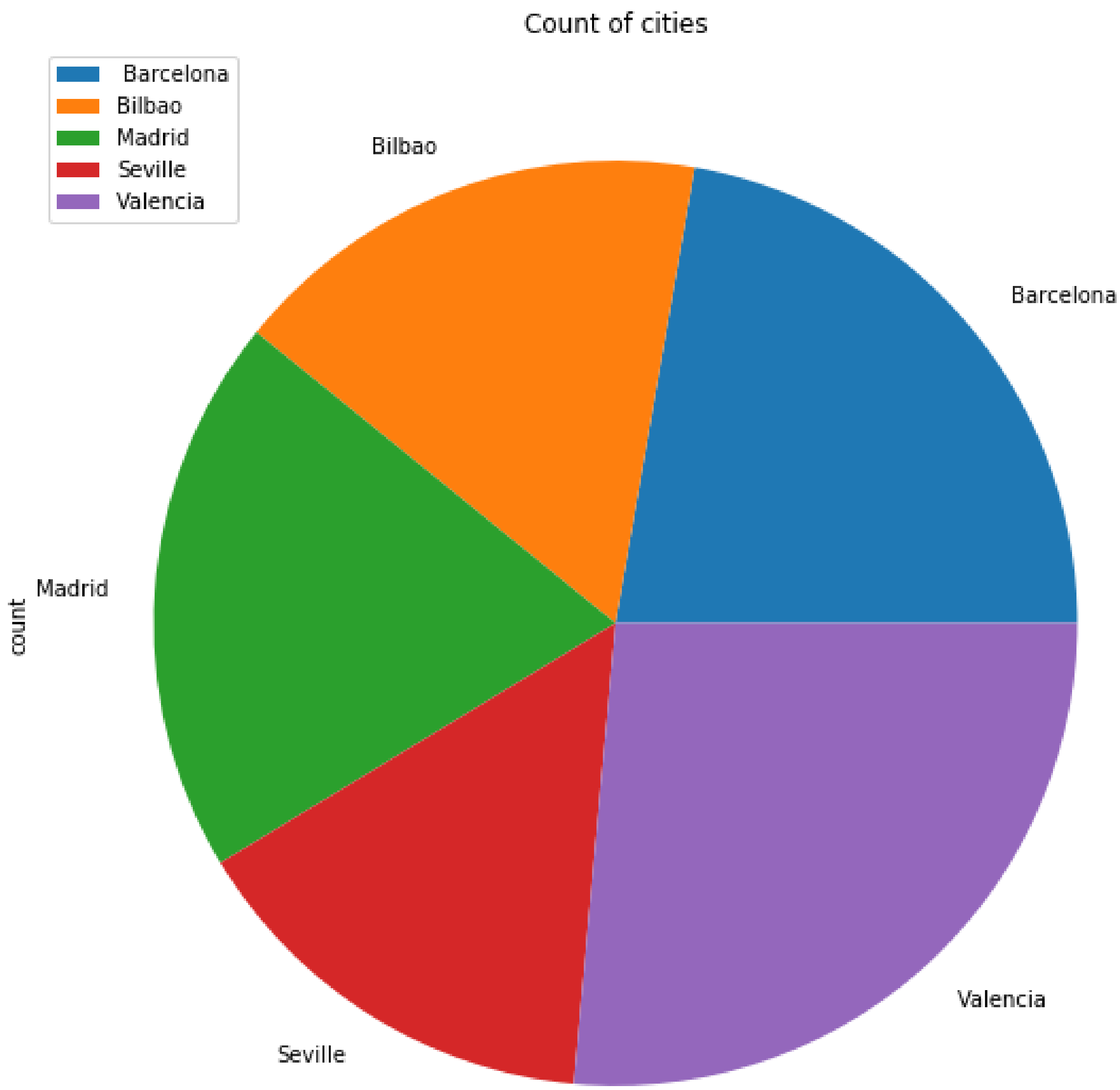
```
cqlsh:group_project> DESCRIBE TABLES;

energy_by_price_actual  temp_by_time_and_city  energy_renewable_by_time
energy_fossil_by_time   weather_by_time
```

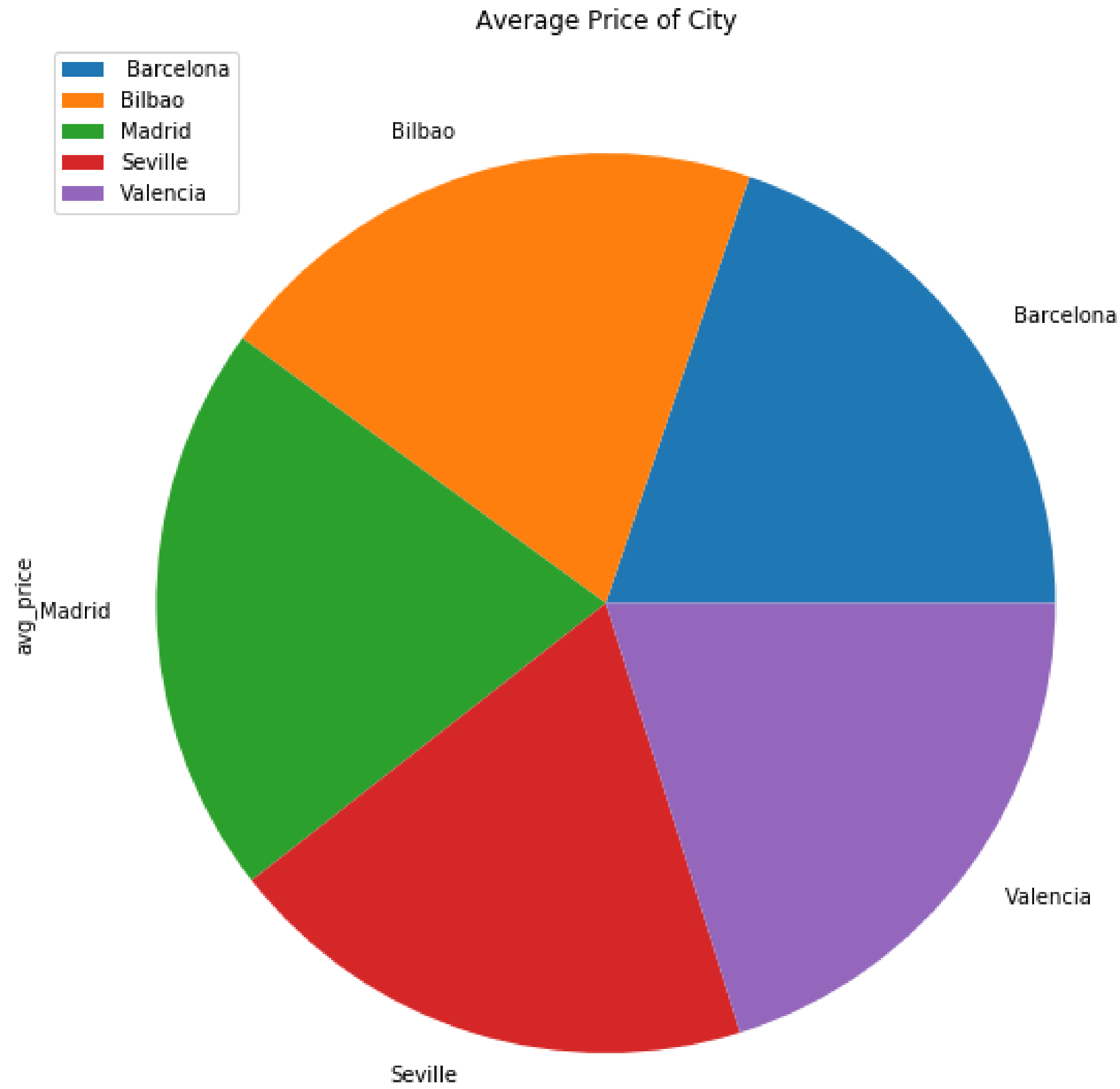
dt_iso	temp	city_name	clouds_all	humidity	pressure	rain_1h	rain_3h	snow_3h	temp_max	temp_min	weather_description	weather_icon	weather_id	weather_main	wind_deg	wind_speed
2018-05-31 12:00:00.000000+0000	292.04001	Bilbao	75	68	1018	0	0	0	293.14999	291.14999	broken clouds	04d	803	clouds	40	2
2018-05-31 12:00:00.000000+0000	295.32999	Madrid	40	43	1018	0	0	0	297.14999	293.14999	scattered clouds	03d	802	clouds	220	2
2018-05-31 12:00:00.000000+0000	296.14999	Barcelona	20	57	1017	0	0	0	297.14999	295.14999	few clouds	02d	801	clouds	130	5
2018-05-31 12:00:00.000000+0000	298.32999	Seville	0	34	1017	0	0	0	300.14999	297.14999	sky is clear	01d	800	clear	300	2
2018-05-31 12:00:00.000000+0000	299.14999	Valencia	20	39	1016	0	0	0	299.14999	299.14999	few clouds	02d	801	clouds	100	4
2016-12-20 20:00:00.000000+0000	276.26001	Madrid	0	70	1024	0	0	0	279.14999	274.14999	sky is clear	01n	800	clear	340	2
2016-12-20 20:00:00.000000+0000	280.51999	Bilbao	88	100	1026	0.3	0	0	282.14999	279.14999	light rain	10n	500	rain	0	1
2016-12-20 20:00:00.000000+0000	282.14999	Valencia	0	70	1021	0	0	0	282.14999	282.14999	sky is clear	01n	800	clear	300	3
2016-12-20 20:00:00.000000+0000	282.14999	Barcelona	75	87	1020	0.3	0	0	282.14999	282.14999	light intensity shower rain	09n	520	rain	0	0
2016-12-20 20:00:00.000000+0000	283.20999	Seville	0	93	1025	0	0	0	291.14999	278.14999	sky is clear	01n	800	clear	177	0
2015-01-08 19:00:00.000000+0000	269.29401	Madrid	0	65	978	0	0	0	269.29401	269.29401	sky is clear	01n	800	clear	353	1
2015-01-08 19:00:00.000000+0000	275.10599	Bilbao	58	88	1041	0	0	0	275.10599	275.10599	broken clouds	04	803	clouds	192	1
2015-01-08 19:00:00.000000+0000	276.95001	Valencia	0	83	1040	0	0	0	276.95001	276.95001	sky is clear	01n	800	clear	294	1
2015-01-08 19:00:00.000000+0000	278.944	Seville	0	90	1046	0	0	0	278.944	278.944	sky is clear	01n	800	clear	54	3
2015-01-08 19:00:00.000000+0000	283.45001	Barcelona	0	60	1036	0	0	0	283.45001	283.45001	sky is clear	01n	800	clear	315	2
2018-07-07 17:00:00.000000+0000	293.95001	Bilbao	40	88	1021	0.3	0	0	295.14999	293.14999	light rain	10n	500	rain	290	1
2018-07-07 17:00:00.000000+0000	298.64999	Barcelona	20	54	1018	0	0	0	299.14999	298.14999	few clouds	02n	801	clouds	0	1
2018-07-07 17:00:00.000000+0000	299.14999	Valencia	0	74	1018	0	0	0	299.14999	299.14999	sky is clear	01n	800	clear	120	1
2018-07-07 17:00:00.000000+0000	301.67001	Madrid	0	24	1017	0	0	0	305.14999	300.14999	sky is clear	01n	800	clear	270	2
2018-07-07 17:00:00.000000+0000	302.32999	Seville	0	31	1014	0	0	0	304.14999	301.14999	sky is clear	01n	800	clear	230	4

Pyspark: (Wes)

I did an exploration of the data using Pyspark and Jupyter Notebooks.

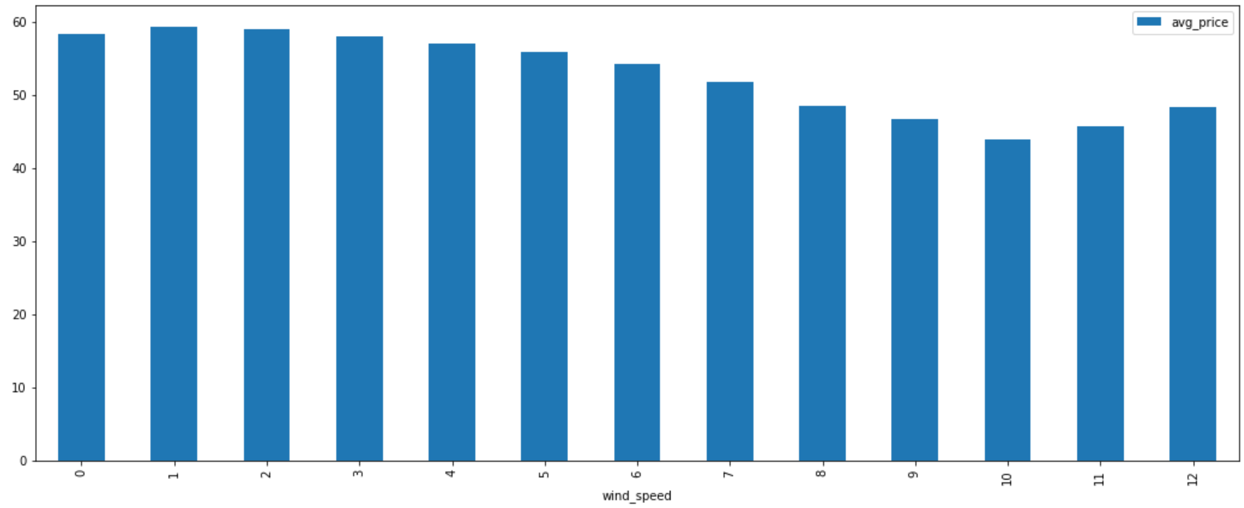
Visualization of the count of cities, to show that each of them are well represented within the dataset. Valencia was the most represented and Seville being the least represented



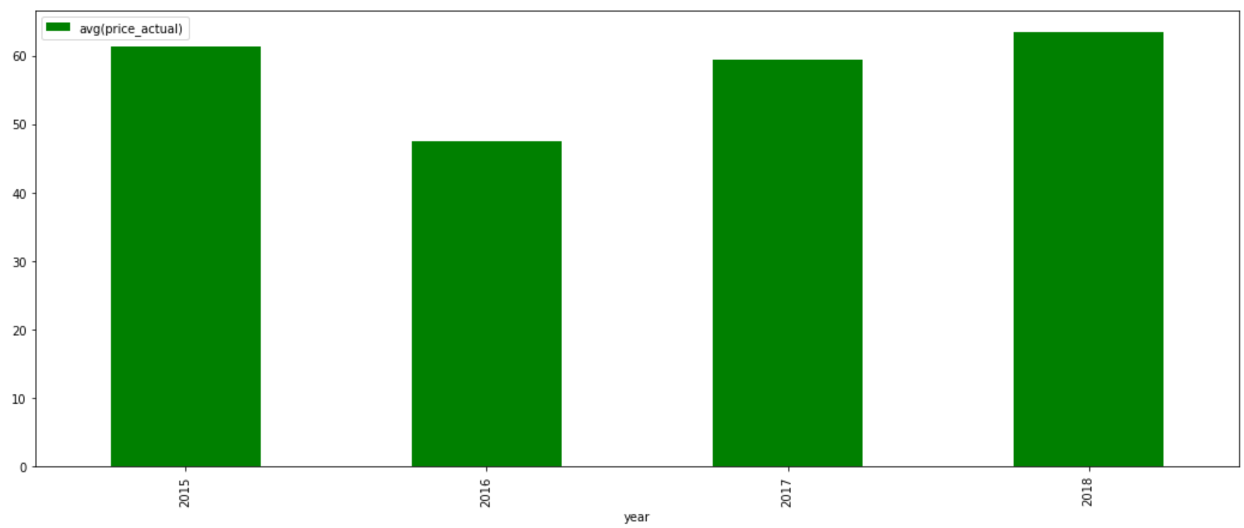


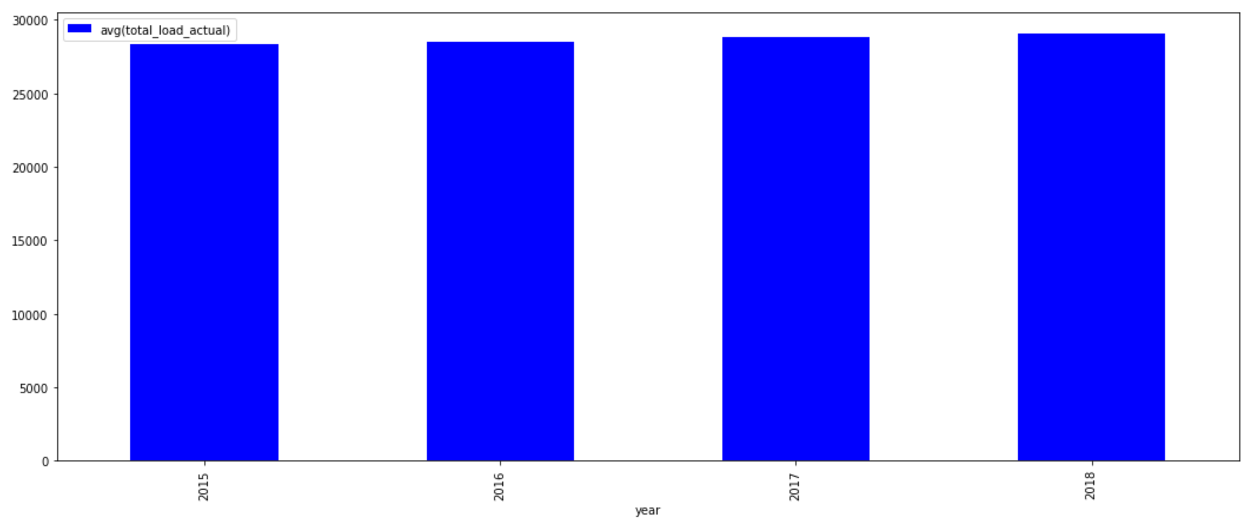
The average price between each city is nearly identical as well. This is something we had hoped to see, as it means the pricing is fair between each city and no one customer is paying more than another just for living in a different city

Exploring more relationships in the data I looked at Wind Speeds and was surprised to a semi-strong correlation between that and price. I don’t have the domain knowledge of the field to know if this is expected behavior or not. It should be noted that higher wind speeds have less data associated with them so those should not be seen as strongly correlated

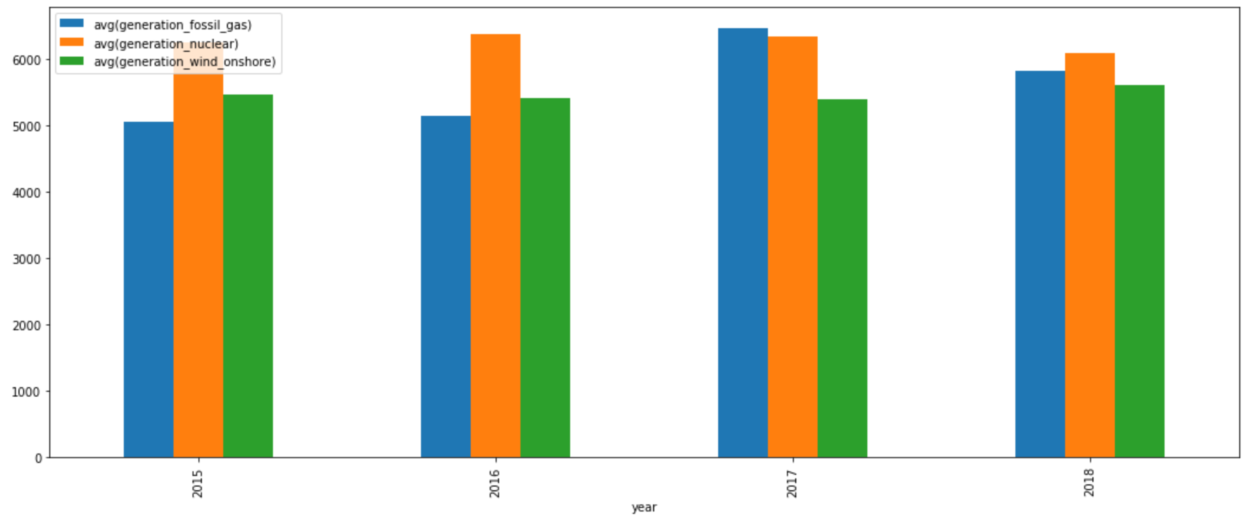


Looking year by year 2016 was a noticeable low point for average price while the load on the system stayed very consistent

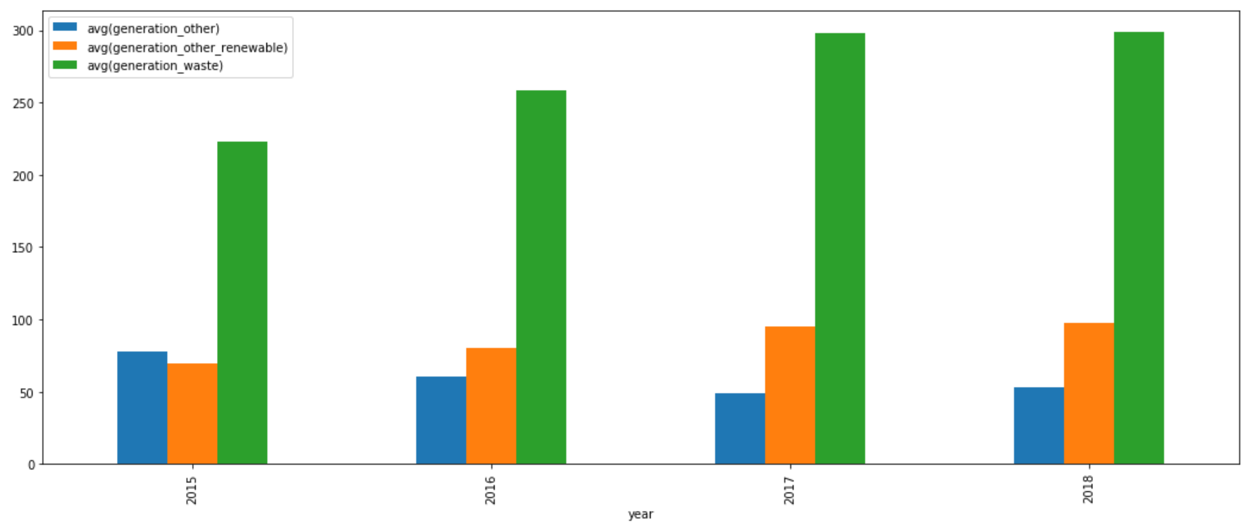




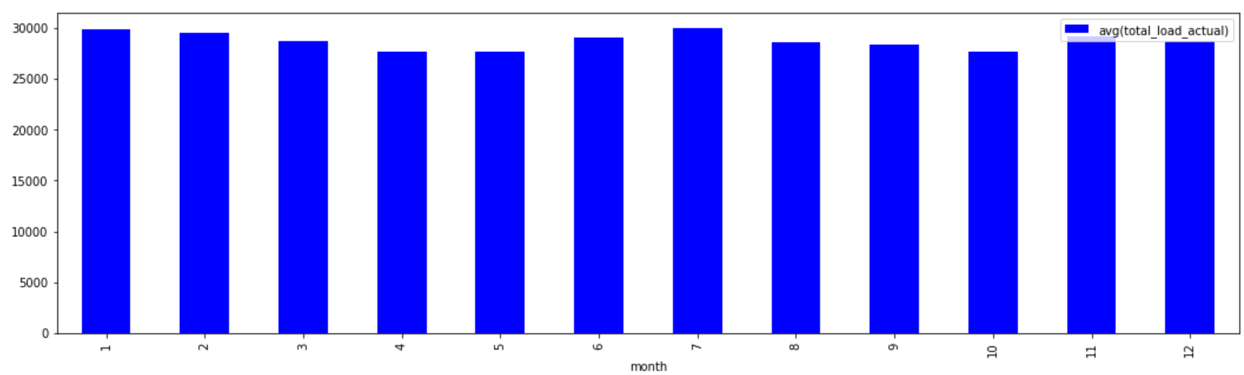
And while the total load on the system remained consistent, the individual energy generations have varied. These are the top three Generations (measured in MegaWatts) and how they varied through the years

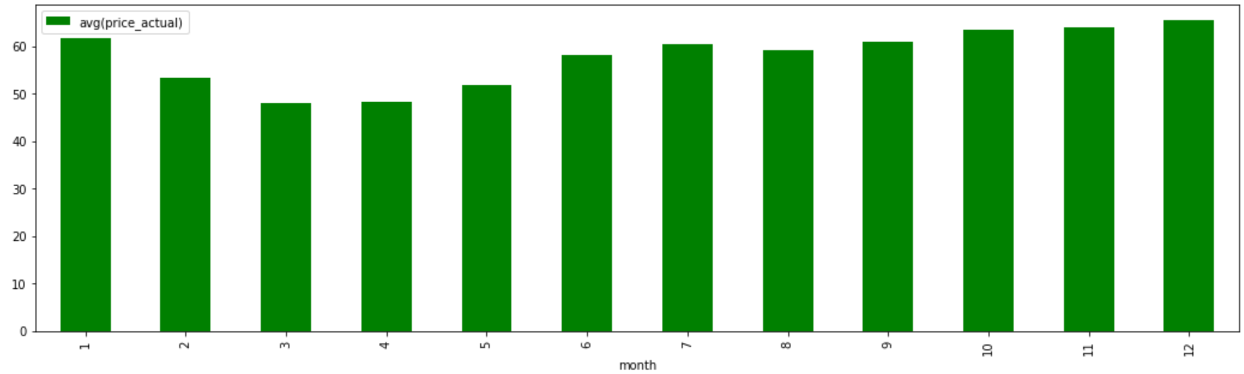


The three lowest (that still follow trends and are not 0) are

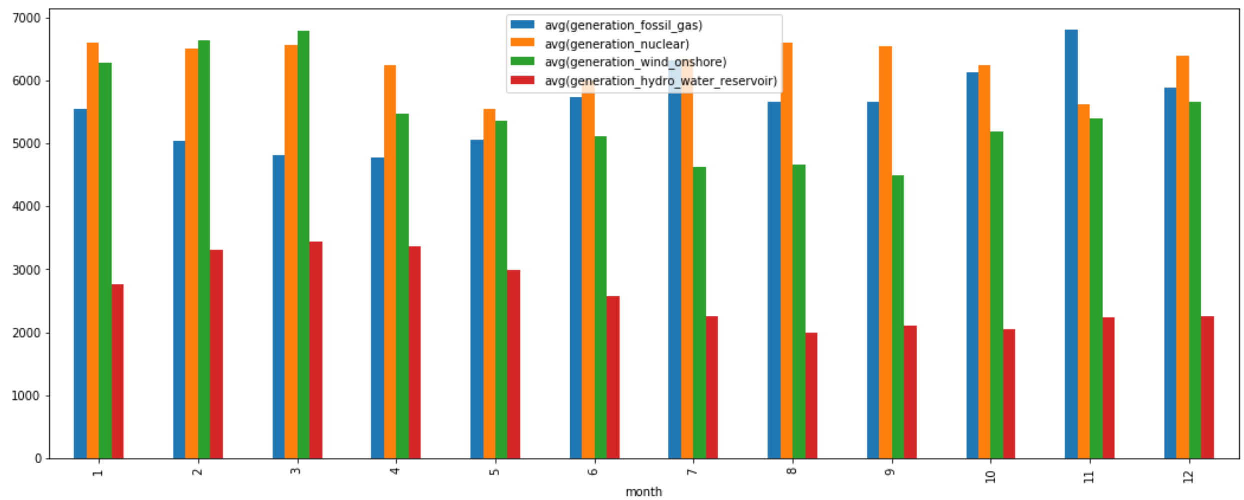


Looking at trends through the different months of the year we can see that the total load is much more consistent than the average price. We can also see that average price is less expensive in the warm months and more expensive in the cold months, as we expect

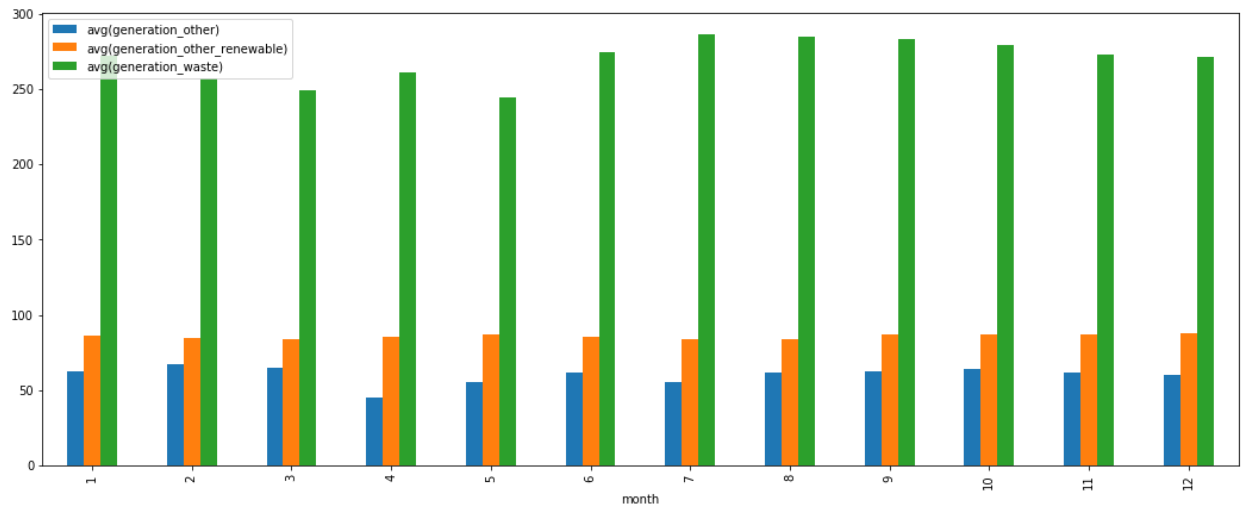




High generation trends by month

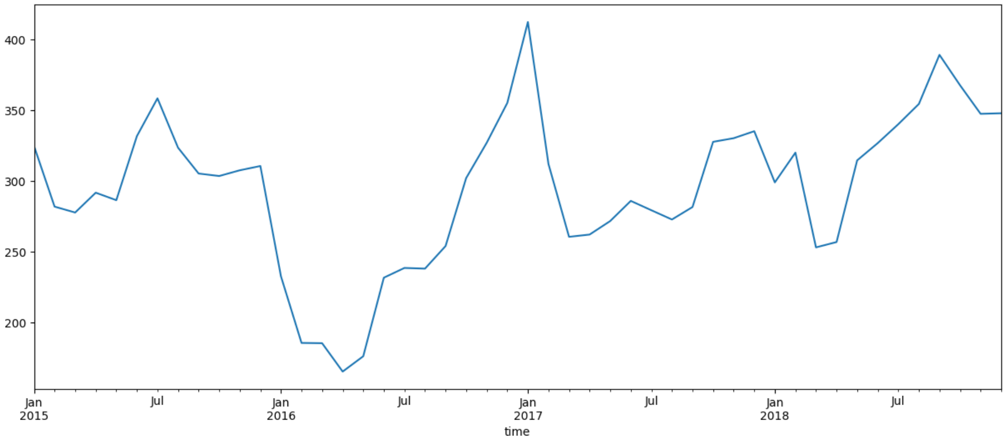


Low generation trends by month

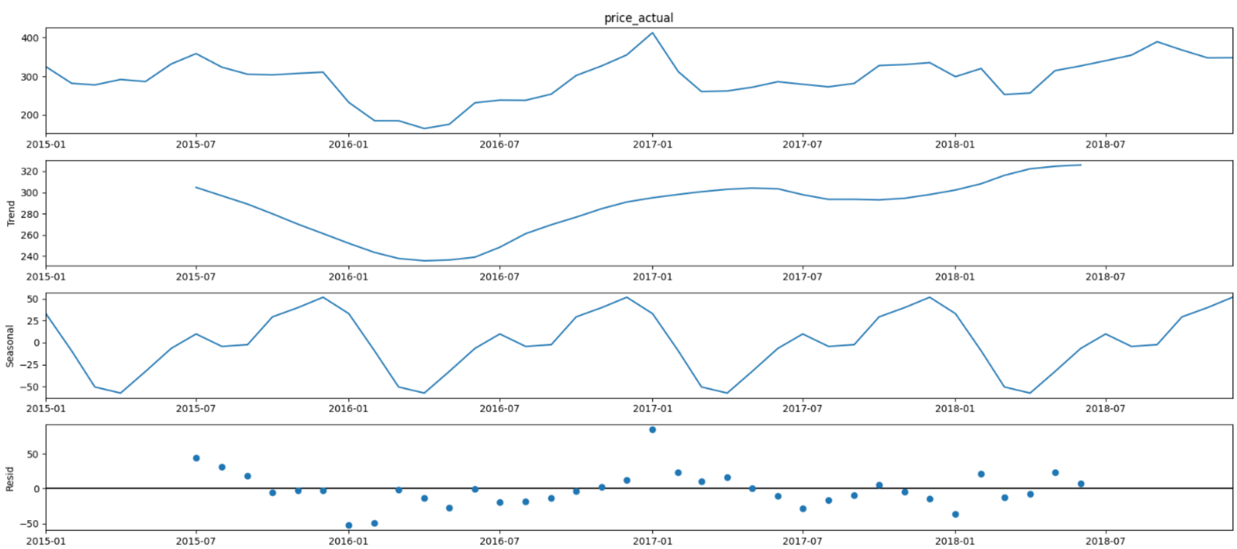


I also did some Time Series analysis using https://towardsdatascience.com/an-end-to-end-project-on-time-series-analysis-and-forecasting-with-python-4835e6bf050b as a guide. I wanted to try and implement some simple machine learning / statistical learning techniques.

Looking at the actual price data broken down by months, it follows a general trend line like this from 2015-2018

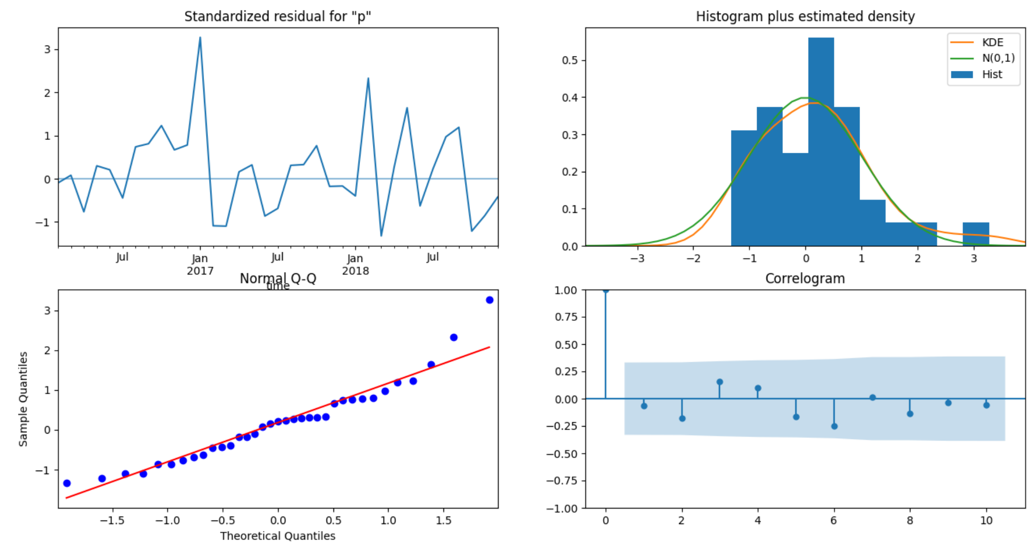


Looking at the Decomposition analysis we can look at the Trend, Seasonals and Residuals. The Trend is essentially the trend line as a quadratic function, simplifying the flow of the data and showing us the general highs, lows and path it traveled. Seasonal is the variation we can expect for a given season, which is why it follows a consistent pattern. Residuals are the timeseries data subtracted by the Trend and the Seasonal data, which we want as close to 0 as possible. The farther away from 0, the more exceptional the data is to be seen at that point.



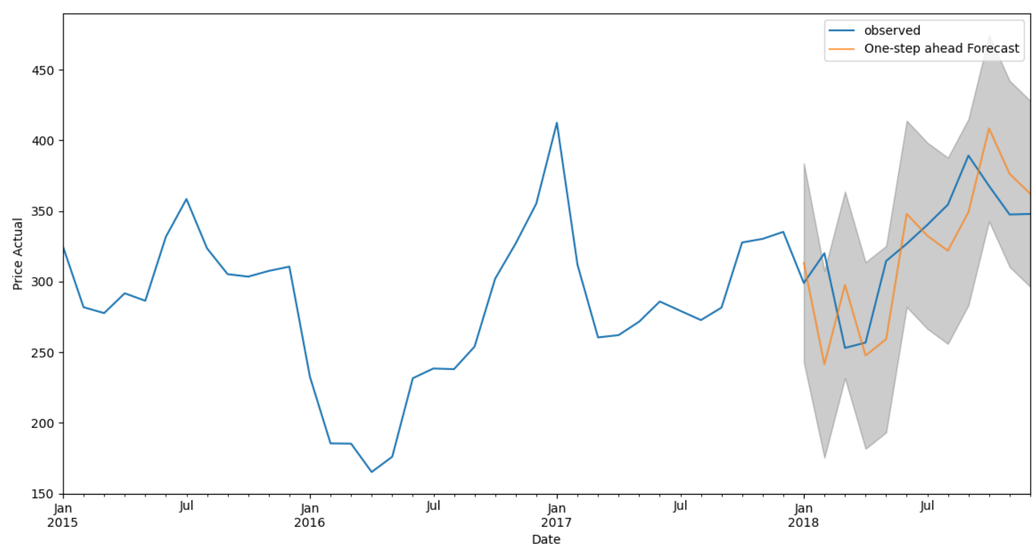
We can see that high point in 2017 was unexpected data, having the highest residual

ARIMA is a commonly used method for time-series analysis. It takes in parameters for the seasonlity, trend and noise within the data. You can decide what values to this by doing an exhaustive search and comparing the AIC and choosing the lowest possible value. Following that methodology and running the diagnostics we get

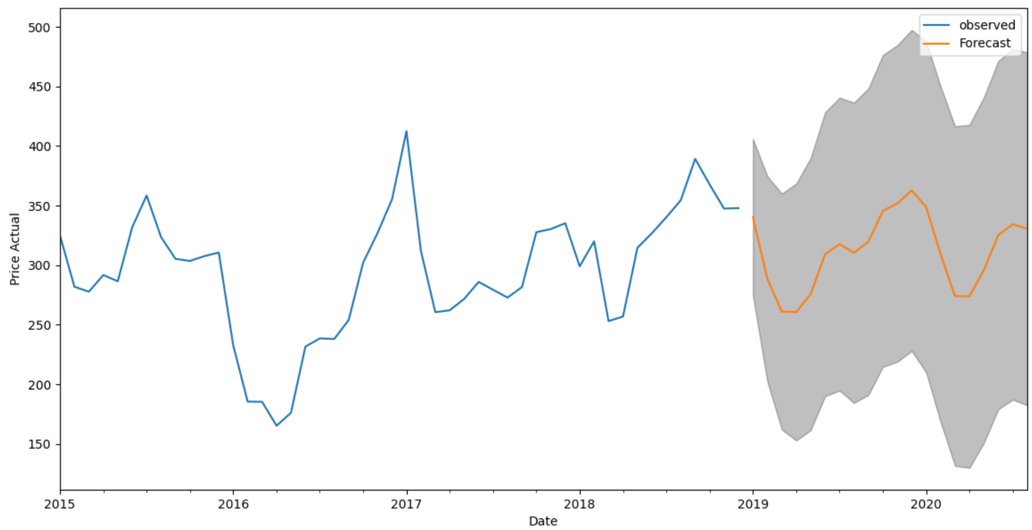


The data isn't perfect, as any real world data won't be, but we can through the Normal Q-Q, Histogram, and Correlogram that the data is normally distributed enough to run some predictive statistics on it

This is a rolling forecast prediction, using the current data to try and predict the tail end of the current data. It's not perfect but we can see we are lying in the same general range of the expected price



Using the same model we built before but trying to predict the next few years of data, we can see a general trend and confidence interval that we can expect the actual price to lie in between 2019 and 2020



SparkStreaming: (Shelby)

Using Spark and SparkStreaming, my goal was to find a way to incorporate new data into our existing dataset. To do this, I found the same API service that was used to populate the data of our original dataset. Unfortunately, due to a paywall, I was only able to use the service that provided information about the current weather rather than historical weather, which would have made querying the data interesting. However, I was still able to make it work using this API. There are two primary ways I did this: using Spark, and using a mock-version of SparkStreaming. The content and results of my work are expanded upon below:

'REST API to Spark DataFrame - Single Call, Save Portions.ipynb' – This Jupyter notebook contains the original attempt to use Spark to process data received from an API call. While it did successfully retrieve data, converting the json to an rdd and then to a Spark datafile, the complex data types of the Spark dataframe made it impossible to conveniently save the data to a .csv.

> 'current_weather_text' – Folder containing output of 'REST API…'. This contains the text version of the saved RDD.
>
> 'current_weather_csv'- Folder containing output of 'REST API…'. This contains the .csv version of selected columns of the Spark dataframe.
>
> 'rel_data_csv'- Folder containing output of 'REST API…'. This contains the .csv version of selected columns of the RDD after they have been parallelized.

'SparkStreaming with REST API - Save and Show DStream Results.ipynb' – This Jupyter notebook contains the SparkStreaming work. Because the data source is an API dependent upon a discrete GET call rather than being from an object that is constantly sending data (say an IoT device), I had to first create an object to contain several calls to the API that I would then use to mimic a streaming data source. This called the API every 10 seconds in an attempt to get varying current data and stored the response into an rddQueue. I then used a DStream to read the data from the rddQueue once every second, and then processed this data by converting the JSON responses into Spark dataframes. I then printed these dataframes to the stream using .foreachRDD and saved each to a folder.

> 'streaming_weather' – Folder containing the results of the DStream processing. Each folder represents the output of one batch interval.

‘JSON to CSV - Get New Info for Multiple Cities.ipynb’ – This Jupyter notebook is contains the finalized work using Spark to get information from an API. In this notebook, I was able to make one call to the API service per city, convert the returned json to a Pandas dataframe, and then concatenate the results in one dataframe that could be saved as a .csv file. By defining the schema of the dataframe, expanding the components of the nested JSON, and making new columns for nested elements, I was able to produce help, relevant information that could then be added to our existing dataset.

‘new_weather_data_5_cities’ - .csv file containing current weather data for all five cities in Spain.

	city_name	feels_like	humidity	pressure	temp	temp_max	temp_min
0	barcelona	277.49	61	1005	281.46	282.59	279.82
0	seville	281.40	81	1018	282.88	283.71	281.48
0	madrid	270.88	86	1011	278.15	278.15	278.15
0	barcelona	277.49	61	1005	281.46	282.59	279.82
0	valencia	275.90	61	1010	282.57	283.15	282.04

Project Management:

- Work completed:
    - Description: We have analyzed our dataset using Hive, mySQL, MapReduce, Cassandra, Spark, and Sparkstreaming, as well as created some robust data visualizations using Tableau.
    - Contributions:
        - Claire: mySQL queries, Sqoop transfer, Spark queries
        - Wes: Hive table creation, HQL queries, Pyspark, Time Series analysis, visualizations
        - Scott: MapReduce queries, advanced algorithms
        - Shelby: Sqoop transfer from Hive to mySQL, Cassandra analysis, SparkStreaming, report composition

Final Assignment Questions:

- Who:
    - This dataset is about the people who use energy in Spain, whose energy production and grid was sampled for this dataset. There is no identifiable information on the individual level, meaning that there is little personal risk with this dataset.
- What:
    - The energy usage and sources of energy production of the people of Spain are what is being recorded by the data set. This addresses all of our questions in Assignment 1.
- When:
    - This data was collected between 2015 – 2019, meaning that the data is recent and therefore relevant. It is cross-sectional since the data was collected from several cities in Spain. This dataset contains real-time data.
- Where:

- ○ The data is collected from the five largest cities in Spain: Madrid, Barcelona, Valencia, Seville, and Bilbao. It could possible be extrapolated that the energy usage would be similar in the surrounding European countries with similar populations and weather as these five cities, and it is certainly possible that larger generalizations about predicting energy usage could be used for non-European locations.
- Why:
  - ○ The data was collected by ENTSOE, a public portal for Transmission Service Operator (TSO) data and is publicly available.

References:

"Tackling Climate Change with Machine Learning"

https://arxiv.org/abs/1906.05433

"Hourly energy demand generation and weather – Electrical demand, generation by type, prices and weather in Space"

https://www.kaggle.com/nicholasjhana/energy-consumption-generation-prices-and-weather?select=weather_features.csv

"Chapter 4. The Cassandra Query Language"

https://www.oreilly.com/library/view/cassandra-the-definitive/9781491933657/ch04.html

"Defining Application Queries"

https://cassandra.apache.org/doc/latest/data_modeling/data_modeling_queries.html

"LanguageManual Select"

https://cwiki.apache.org/confluence/display/Hive/LanguageManual+Select

MapReduce:

https://nestedsoftware.com/2018/03/27/calculating-standard-deviation-on-streaming-data-253l.23919.html

https://hadoop.apache.org/docs/r2.6.0/api/org/apache/hadoop/mapred/lib/ChainMapper.html

https://hadoop.apache.org/docs/r2.6.0/api/org/apache/hadoop/mapred/lib/ChainReducer.html

https://hadoop.apache.org/docs/r2.6.0/api/org/apache/hadoop/mapred/Mapper.html

https://hadoop.apache.org/docs/r2.6.0/api/org/apache/hadoop/mapred/Reducer.html

https://hadoop.apache.org/docs/r2.6.0/api/org/apache/hadoop/mapreduce/Job.html

https://hadoop.apache.org/docs/r2.6.0/api/org/apache/hadoop/io/package-summary.html