Michael Lee

CSCI E-63 Big Data Analytics Spring 2021

**Smart Meter Analysis Project**

**Problem Statement:** Smart Meters are a useful tool to measure energy consumption in households. This project aims to take the energy consumption data provided by Smart Meters and use Spark and Python to analyze and visualize factors which impact household energy consumption, such as socioeconomic factors, time, and weather conditions. In addition, a series of regression models is built to predict the energy consumption of a given household.

**Data Set:** Smart Meters in London dataset from Kaggle. This is a dataset which contains the daily and hourly energy consumption readings for 5,561 London households between November 2011 and February 2014. The demographic information and method of billing for the households is also provided, as well as the weather conditions during that time period in London.

**Technology/Feature:** This project uses Python and Spark to load and join data from different worksheets. Data visualization is performed using matplotlib. The Spark ML API is used to perform Linear Regression, Decision Tree Regression, Random Forest Regression, and Gradient Boosted Regression.

**Uses/Benefits:** The insights from the Smart Meter analysis will allow electricity utilities to understand what factors have an impact of electricity demand. A predictive model would allow utilities to better plan their power generation resources to be online during times of peak demand. This would allow the utilities to improve operational efficiency and reliability.

**Drawbacks/Challenges:** One of the challenges of this project was handling and processing the large amount of data, especially the hourly meter dataset which was over 10GB in size. Even using Spark, simple computations took an excessive amount of time to compute. Therefore, I decided to focus the main analysis with the daily energy consumption dataset and only use a subset of the hourly energy consumption dataset for analysis.

**Results:** Average energy consumption is higher during winter months, during weekends, and during evenings hours. More affluent socioeconomic groups have a higher average energy consumption. Regression analysis on the entire dataset did not provide meaningful results due to the large variance in household energy consumption. However, once I narrowed the scope of the regression analysis to focus on a single household’s energy consumption, I could build regression models to predict energy consumption.

**Project Files:**

Link to Dataset: <https://www.kaggle.com/jeanmidev/smart-meters-in-london>

Link to 2 minute Presentation Video:<http://bit.ly/CSCIE-63SmartMeterShortVideo>

Link to 15 minute Presentation Video: <http://bit.ly/CSCIE-63SmartMeterLongVideo>

**Introduction**

Problem Statement

As part of efforts to promote energy conservation, Smart Meters have been employed to measure household energy consumption. One of the benefits of Smart Meters include a real-time measurement of energy consumption, allowing utilities to better match power generation and power consumption. This project aims to take the energy consumption data provided by Smart Meters and use Spark and Python to analyze and visualize factors which impact household energy consumption, such as socioeconomic factors, time, and weather conditions. In addition, a series of regression models is built to predict the energy consumption of a given household. By understanding and predicting the energy consumption of households, the electric utility can ensure that sufficient power capacity is available during peak demand while minimizing operating other power plants during periods of low power demand.

Data Set

As stated in the abstract, the dataset used for this project is the Smart Meters in London dataset from Kaggle. This dataset originates from the UK Power Networks SmartMeter Energy Consumption Data source (<https://data.london.gov.uk/dataset/smartmeter-energy-use-data-in-london-households>) and includes energy consumption for a sample of 5,567 London households as part of the Low Carbon London project. The households were categorized based on the CACI Acorn group classifications ([https://www.caci.co.uk/sites/default/files/resources/Acorn User Guide 2020.pdf](https://www.caci.co.uk/sites/default/files/resources/Acorn%20User%20Guide%202020.pdf)). A sample of the households were subjected to dynamic Time-of-Use billing where the price of electricity was charged based on the time-of-day and a higher rate was applied during hours of peak demand.

The Kaggle dataset also included a list of official UK holidays and the weather data during the timeframe of the project (November 2011 to February 2014). The weather data comes from Dark Sky API (<https://darksky.net/dev>).

**Software/Tools**

Since the provided dataset is large (approximately 10GB), we will need to use a big data application to load, process, and visualize the data. Spark is a fast general processing engine and it handles big data applications well because it runs on distributed memory. To run Spark in a Python environment, we use PySpark. The Spark API also includes machine learning features (Spark ML), which we will use for regression analysis (Linear Regression, Decision Tree Regression, Random Forest Regression, Gradient Boosted Regression).

For visualizations of the data located in a Spark DataFrame, we first convert the data selected for plotting into a Pandas DataFrame. We then use the Matplotlib Python plotting library to prepare visualizations such as scatter plots, line plots, or histograms. In some cases, the Spark DataFrames are converted to Pandas DataFrames to before plotting in Matplotlib.

**Installation and Configuration**

The requirements to run the code include installation of Spark, as well as the following packages: pyspark, pandas, numpy, matplotlib.

To install numpy, pandas, and matplotlib:

* pip install numpy pandas matplotlib

To install Spark (based on CSCI E-63 Lecture 1 Notes):

* Check if Java 8 or 11 is installed:
  + If Java is not installed, install Java from <https://www.java.com/en/download/help/download_options.html> based on your operating system
  + Set JAVA\_HOME environmental variable to Java directory
* Install Spark:
  + If using Windows, you need to install Hadoop binaries for Windows using <https://github.com/steveloughran/winutils>
  + Set HADOOP\_HOME environmental variable to Java directory
  + Download latest version of Spark from <https://spark.apache.org/downloads.html>
  + Set SPARK\_HOME environmental variable to Java directory

To install pyspark:

* pip install pyspark

Running Project Code

* The Demo Notebook (SmartMeter\_Demo\_Notebook.ipynb) is the main code repository for this project and contains all the code needed to perform the Smart Meter analysis.
* Follow the instructions below to download the data set and run the Demo Notebook in Jupyter Notebook.

*Downloading Data Set*

* When downloading the data set (<https://www.kaggle.com/jeanmidev/smart-meters-in-london>), please ensure the **“archive”** directory is unzipped and stored in the same directory as SmartMeter\_Demo\_Notebook.ipynb. Otherwise, the user will need to update the paths for the data folders in the “Load in Daily Meter Data” and “Load in Hourly Meter Data” sections of the Demo Notebook.

*Run Demo Notebook in Jupyter Notebook*

* Set the following environmental variables in your terminal or command prompt:
  + PYSPARK\_DRIVER\_PYTHON=jupyter
  + PYSPARK\_DRIVER\_PYTHON\_OPTS=notebook
* Run pyspark in your terminal or command prompt
* A Jupyter Notebook running pyspark will open in your browser and you can run the SmartMeter\_Demo\_Notebook.ipynb file.

**Code with Comments**

The main components of the project code for loading, joining, and cleaning the data are listed below, along with comments describing the code functionality. Highlights of code showing how to plot data, perform aggregations, and to build regression models are also included below. Please refer to the Demo Notebook for the full project code.

Import packages and libraries

|  |
| --- |
| from pyspark.sql.functions import countDistinct, col, dayofweek, month, hour, minute, when  from pyspark.sql.functions import isnan, when, count, spark\_partition\_id, asc, desc, avg, to\_date  from pyspark.sql.types import StructType, StructField, StringType, TimestampType, FloatType, IntegerType, StringType, DateType  from pyspark.ml.feature import RFormula, VectorAssembler, OneHotEncoder, StringIndexer  from pyspark.ml.regression import LinearRegression, DecisionTreeRegressor, RandomForestRegressor, GBTRegressor  from pyspark.ml.evaluation import RegressionEvaluator  from pyspark.sql.functions import udf  import numpy as np  import pandas as pd  import matplotlib.pyplot as plt  %matplotlib inline |

* Set SparkSession configuration

|  |
| --- |
| sc.setLogLevel("ERROR")  appName = "SmartMeter"  master = "local[8]"  spark = SparkSession.builder \  .appName(appName) \  .master(master) \  .getOrCreate() |

Data Cleaning

* Set Schemas for Imported Data

|  |
| --- |
| meter\_daily\_schema = StructType([StructField('LCLid', StringType(), True),  StructField('time', DateType(), True),  StructField('energy\_median', FloatType(), True),  StructField('energy\_mean', FloatType(), True),  StructField('max', FloatType(), True),  StructField('energy\_count', IntegerType(), True),  StructField('std', FloatType(), True),  StructField('energy\_sum', FloatType(), True),  StructField('min', FloatType(), True)])  meter\_hourly\_schema = StructType([StructField('LCLid', StringType(), True),  StructField('time', TimestampType(), True),  StructField('energy', FloatType(), True)])  household\_schema = StructType([StructField('LCLid', StringType(), True),  StructField('stdorToU',StringType(), True),  StructField('Acorn',StringType(), True),  StructField('Socialeconomic',StringType(), True),  StructField('File', StringType(), True)])  weather\_daily\_schema = StructType([StructField('daily', FloatType(), True),  StructField('tempMaxTime', TimestampType(), True),  StructField('windDir', IntegerType(), True),  StructField('icon', StringType(), True),  StructField('dew', FloatType(), True),  StructField('tempMinTime', TimestampType(), True),  StructField('cloudCover', FloatType(), True),  StructField('windSpd', FloatType(), True),  StructField('pres', FloatType(), True),  StructField('appTempMinTime', TimestampType(), True),  StructField('appTempHigh', FloatType(), True),  StructField('precip', StringType(), True),  StructField('vis', FloatType(), True),  StructField('humid', FloatType(), True),  StructField('appTempHighTime', TimestampType(), True),  StructField('appTempLow', FloatType(), True),  StructField('appTempMax', FloatType(), True),  StructField('uv', StringType(), True),  StructField('time', DateType(), True),  StructField('sunset', TimestampType(), True),  StructField('tempLow', FloatType(), True),  StructField('tempMin', FloatType(), True),  StructField('tempHigh', FloatType(), True),  StructField('sunrise', TimestampType(), True),  StructField('tempHighTime', TimestampType(), True),  StructField('uvIndexTime', TimestampType(), True),  StructField('summary', StringType(), True),  StructField('tempLowTime', TimestampType(), True),  StructField('appTempMin', FloatType(), True),  StructField('appTempMaxTime', TimestampType(), True),  StructField('appTempLowTime', TimestampType(), True),  StructField('moonPhase', FloatType(), True)])  weather\_hourly\_schema = StructType([StructField('vis', FloatType(), True),  StructField('windDir', IntegerType(), True),  StructField('temp', FloatType(), True),  StructField('time', TimestampType(), True),  StructField('dew', FloatType(), True),  StructField('pres', FloatType(), True),  StructField('apptemp', FloatType(), True),  StructField('windSpd', FloatType(), True),  StructField('precip', StringType(), True),  StructField('icon', StringType(), True),  StructField('humid', FloatType(), True),  StructField('summary', StringType(), True)])  holiday\_schema = StructType([StructField('holiday', DateType(), True),  StructField('Type', StringType(), True)]) |

Load in Data

|  |
| --- |
| # load in daily energy consumption data  meter\_daily\_df = spark.read.option("header", "true").schema(meter\_daily\_schema) \  .csv(r"archive\daily\_dataset\daily\_dataset\block\_\*.csv")  # load in hourly energy consumption data  meter\_hourly\_df = spark.read.option("header", "true").schema(meter\_hourly\_schema) \  .csv(r"archive\halfhourly\_dataset\halfhourly\_dataset\block\_0.csv")  meter\_hourly\_df = meter\_hourly\_df.filter(col("LCLid") == "MAC003597")  # load in household information  household\_df = spark.read.option("header", "true").schema(household\_schema) \  .csv(r"archive\informations\_households.csv")  # load in daily weather data  weather\_daily\_df = spark.read.option("header", "true").schema(weather\_daily\_schema).csv(r"archive\weather\_daily\_darksky.csv")  # load in hourly weather data  weather\_hourly\_df = spark.read.option("header", "true").schema(weather\_hourly\_schema).csv(r"archive\weather\_hourly\_darksky.csv")  # load in holiday data  holiday\_df = spark.read.option("header", "true").schema(holiday\_schema).csv(r"archive\uk\_bank\_holidays.csv") |

Consolidate Data

* Joining and cleaning data for daily energy consumption dataset

|  |
| --- |
| # Join the meter and the household DataFrames into meter\_household\_daily\_df  joinType = "inner"  joinExpression = ["LCLid"]  meter\_household\_daily\_df = meter\_daily\_df.join(household\_df, joinExpression, joinType)  # Join meter\_household\_daily\_df and the daily weather DataFrame into met\_hou\_wea\_daily\_df  joinType = "inner"  joinExpression = ["time"]  met\_hou\_wea\_daily\_df = meter\_household\_daily\_df.join(weather\_daily\_df, joinExpression, joinType)  # Add a binary variable to indicate if the date is a holiday  combined\_daily\_df = met\_hou\_wea\_daily\_df.withColumn("holiday", col("time")) \  .join(holiday\_df.withColumn("holiday", col("holiday")), on="holiday", how="left") \  .select("\*", when(col("Type").isNotNull(), "1").otherwise("0").alias("holiday\_binary")) \  .drop("Type")  # check to see if there are any null values  daily\_timestamp\_cols\_filtered = [c for c, t in combined\_daily\_df.dtypes if t != 'date' and t != 'timestamp']  daily\_data\_nullcheck = combined\_daily\_df.select(\*daily\_timestamp\_cols\_filtered)  daily\_data\_nullcheck.select([count(when(isnan(c) | col(c).isNull() | (col(c) == "NA") | (col(c) == "NULL"),c)) \  .alias(c) for c in daily\_data\_nullcheck.columns]).show()  # remove all rows which have null values  combined\_daily\_df = combined\_daily\_df.na.drop() |

* Joining and cleaning data for Hourly energy consumption dataset

|  |
| --- |
| # Join the meter and the household DataFrames into meter\_household\_hourly\_df  joinType = "inner"  joinExpression = ["LCLid"]  meter\_household\_hourly\_df = meter\_hourly\_df.join(household\_df, joinExpression, joinType)  # Join meter\_household\_hourly\_df and the hourly weather DataFrame into met\_hou\_wea\_hourly\_df  joinType = "inner"  joinExpression = ["time"]  met\_hou\_wea\_hourly\_df = meter\_household\_hourly\_df.join(weather\_hourly\_df, joinExpression, joinType)  # Add a binary variable to indicate if the date is a holiday  combined\_hourly\_df = met\_hou\_wea\_hourly\_df.withColumn("holiday", col("date")) \  .join(holiday\_df.withColumn("holiday", col("holiday")), on="holiday", how="left") \  .select("\*", when(col("Type").isNotNull(), "1").otherwise("0").alias("holiday\_binary")) \  .drop("Type")  # check to see if there are any null values  hourly\_timestamp\_cols\_filtered = [c for c, t in combined\_hourly\_df.dtypes if t != 'date' and t != 'timestamp']  hourly\_data\_nullcheck = combined\_hourly\_df.select(\*hourly\_timestamp\_cols\_filtered)  hourly\_data\_nullcheck.select([count(when(isnan(c) | col(c).isNull() | (col(c) == "NA") | (col(c) == "NULL"),c)) \  .alias(c) for c in hourly\_data\_nullcheck.columns]).show()  # remove all rows which have null values  combined\_hourly\_df = combined\_hourly\_df.na.drop() |

Explore Data

|  |
| --- |
| # count unique meters in the database  combined\_daily\_df.select(countDistinct("LCLid")).show() |

|  |
| --- |
| # count how many observations have no consumption  combined\_daily\_df.select(count(when(col("energy\_sum") < 0, 1)).alias("Energy Cons. < 0")).show()  combined\_daily\_df.select(count(when(col("energy\_sum") == 0, 1)).alias("Energy Cons. == 0")).show()  combined\_daily\_df.select(count(when(col("energy\_sum") > 0, 1)).alias("Energy Cons. > 0")).show()  # remove data points with no daily energy consumption as it could suggest power outage or meter misreading  combined\_daily\_df = combined\_daily\_df.filter(combined\_daily\_df.energy\_sum > 0)  # check to make sure that there are no remaining data points with 0 daily energy consumption.  combined\_daily\_df.select(count(when(col("energy\_sum") == 0, 1)).alias("Energy Cons. == 0")).show() |

* Plotting a scatterplot using matplotlib

|  |
| --- |
| # convert Spark DataFrame to Pandas dataframe for plotting  energy\_daily = combined\_daily\_df.select("time","energy\_mean","min","max", "energy\_sum").toPandas()  energy\_daily.set\_index("time")  # save holiday dates as a list for plotting  holiday\_daily\_list = holiday\_df.select("holiday").filter(col("holiday") < "2014-04-01").collect()  holiday\_daily\_list  # create a scatter plot using matplotlib  plt.figure(figsize=(12,6))  plt.scatter(energy\_daily["time"], energy\_daily["energy\_sum"], label="Energy Consumption (kWh)")  plt.xlabel("Date")  plt.ylabel("Daily Energy Consumption (kWh)")  plt.title("Daily Energy Consumption")  for h in holiday\_daily\_list:  plt.axvline(x=h, color='red')  # red vertical lines indicate holiday  plt.show() |

* Perform aggregation and create line plot using matplotlib

|  |
| --- |
| energy\_month = combined\_daily\_df.groupBy("month") # aggregating by month  .avg("energy\_sum").orderBy(col("month").cast(IntegerType())) # calculate mean energy\_sum  energy\_month.show()  energy\_month\_pd = energy\_month.toPandas()  # create a line plot in matplotlib  plt.figure(figsize=(12,6))  plt.plot(energy\_month\_pd["month"], energy\_month\_pd["avg(energy\_sum)"], label="Energy Consumption (kWh)")  plt.xlabel("Month")  plt.ylabel("Average Daily Energy Consumption (kWh)")  plt.title("Daily Energy Consumption by Month")  plt.show() |

* Perform aggregation and create bar plot using matplotlib

|  |
| --- |
| # aggregating by Acron group and calculating the mean energy\_sum per Acorn group  energy\_acorn\_df = combined\_daily\_df.groupBy("Acorn") # aggregating by Acorn group  .avg("energy\_sum").orderBy(col("Acorn")) # calculate mean energy\_sum  energy\_acorn\_df.show()  energy\_acorn\_pd = energy\_acorn\_df.toPandas()  # create a bar plot in matplotlib  plt.figure(figsize=(18,6))  plt.bar(energy\_acorn\_pd["Acorn"], energy\_acorn\_pd["avg(energy\_sum)"])  plt.xlabel("Acorn Group")  plt.xticks(rotation=45)  plt.ylabel("Average Daily Energy Consumption (kWh)")  plt.title("Energy Consumption by Acorn Group")  plt.show() |

Regression Models

|  |
| --- |
| # prepare data  supervised\_daily = RFormula(formula="energy\_sum ~.")  fittedRF\_daily = supervised\_daily.fit(model\_daily\_df)  # perform 80% train - 20% test split  train\_daily, test\_daily = model\_daily\_df.randomSplit([0.8, 0.2], seed=63)  prepared\_daily\_train\_df = fittedRF\_daily.transform(train\_daily)  prepared\_daily\_test\_df = fittedRF\_daily.transform(test\_daily)  prepared\_daily\_train\_df.show(5)  prepared\_daily\_test\_df.show(5) |

* Create linear regression model

|  |
| --- |
| # create linear regression model  lr\_daily = LinearRegression(featuresCol="features", labelCol="label", maxIter=20, regParam=0.1, elasticNetParam=0.1)  lr\_daily\_model = lr\_daily.fit(prepared\_daily\_train\_df)  # view model coefficients and intercept  print("Coefficients: " + str(np.round(lr\_daily\_model.coefficients,2)))  print("Intercept: " + str(np.round(lr\_daily\_model.intercept,2)))  # view index-to-feature mapping to identify which index position corresponds with which feature  coef\_daily = lr\_daily\_model.coefficients  print("Maximum model coefficient:", np.round(max(np.abs(coef\_daily)),2))  print("Index position of maximum model coefficient:", np.argmax(coef\_daily))  numeric\_coeff\_daily = prepared\_daily\_train\_df.select("features").schema[0].metadata.get(  'ml\_attr').get('attrs').get('numeric')  categorical\_coeff\_daily = prepared\_daily\_train\_df.select("features").schema[0].metadata.get(  'ml\_attr').get('attrs').get('binary')  model\_coeff\_daily = numeric\_coeff\_daily + categorical\_coeff\_daily  model\_coeff\_daily  # view training R2 and RMSE  trainingSummary\_daily = lr\_daily\_model.summary  print("RMSE: %.2f" % trainingSummary\_daily.rootMeanSquaredError)  print("r2: %.3f" % trainingSummary\_daily.r2)  # view test predictions  lr\_daily\_predictions = lr\_daily\_model.transform(prepared\_daily\_test\_df)  lr\_daily\_predictions\_df = lr\_daily\_predictions.select("prediction","energy\_sum").toPandas()  lr\_daily\_predictions\_df['prediction'].describe()  # view test R2 and RMSE  lr\_daily\_evaluator = RegressionEvaluator(predictionCol="prediction", labelCol="energy\_sum",  metricName="r2")  lr\_daily\_test\_r2 = lr\_daily\_evaluator.evaluate(lr\_daily\_predictions)  print("R Squared (R2) on test data = %.2g" % lr\_daily\_test\_r2) |

* Create decision tree model

|  |
| --- |
| # build decision tree model and fit on the training dataset used for linear regression  daily\_dt = DecisionTreeRegressor(featuresCol='features')  daily\_dt\_model = daily\_dt.fit(prepared\_daily\_train\_df)  # making predictions on test dataset  daily\_dt\_predictions = daily\_dt\_model.transform(prepared\_daily\_test\_df)  daily\_dt\_predictions.select("prediction", "label", "features").show(5)  # Show decision tree performance  # Select (prediction, true label) and compute test error  daily\_dt\_evaluator = RegressionEvaluator(  labelCol="label", predictionCol="prediction", metricName="rmse")  daily\_dt\_rmse = daily\_dt\_evaluator.evaluate(daily\_dt\_predictions)  print("Root Mean Squared Error (RMSE) on test data = %.3g" % daily\_dt\_rmse)  # View decision tree feature importance  # Check with linear regression index-to-feature mapping to identify features with highest magnitude.  daily\_dt\_model.featureImportances |

* Create random forest model

|  |
| --- |
| # build random forest model and fit on the training dataset used for linear regression  daily\_rf = RandomForestRegressor(featuresCol="features")  daily\_rf\_model = daily\_rf.fit(prepared\_daily\_train\_df)  # making predictions on test dataset  daily\_rf\_predictions = daily\_rf\_model.transform(prepared\_daily\_test\_df)  daily\_rf\_predictions.select("prediction", "label", "features").show(5)  # Show decision tree performance  # Show random forest tree performance  # Select (prediction, true label) and compute test error  daily\_rf\_evaluator = RegressionEvaluator(  labelCol="label", predictionCol="prediction", metricName="rmse")  daily\_rf\_rmse = daily\_rf\_evaluator.evaluate(daily\_rf\_predictions)  print("Root Mean Squared Error (RMSE) on test data = %.3g" % daily\_rf\_rmse)  # View random forest feature importance.  # Check with linear regression index-to-feature mapping to identify features with highest magnitude.  daily\_rf\_model.featureImportances |

* Create gradient boosted tree model

|  |
| --- |
| # build gradient boosted tree model and fit on the training dataset used for linear regression  daily\_gbt = GBTRegressor(featuresCol="features")  daily\_gbt\_model = daily\_gbt.fit(prepared\_daily\_train\_df)  # making predictions on test dataset  daily\_gbt\_predictions = daily\_gbt\_model.transform(prepared\_daily\_test\_df)  daily\_gbt\_predictions.select("prediction", "label", "features").show(5)  # Show gradient tree performance  # Select (prediction, true label) and compute test error  daily\_gbt\_evaluator = RegressionEvaluator(  labelCol="label", predictionCol="prediction", metricName="rmse")  daily\_gbt\_rmse = daily\_gbt\_evaluator.evaluate(daily\_gbt\_predictions)  print("Root Mean Squared Error (RMSE) on test data = %.3g" % daily\_gbt\_rmse)  # View gradient boosted tree feature importance.  # Check with linear regression index-to-feature mapping to identify features with highest magnitude.  daily\_gbt\_model.featureImportances |

* Create a model summary

|  |
| --- |
| # create a table to summarize the results across all models  result\_data = {'Overall Daily Model': [mean\_energy\_daily, lr\_daily\_test\_r2, lr\_daily\_test\_result.rootMeanSquaredError, daily\_dt\_rmse, daily\_rf\_rmse, daily\_gbt\_rmse],  'Single Meter Hourly Model': [mean\_energy\_hourly, lr\_hourly\_test\_r2, lr\_hourly\_test\_result.rootMeanSquaredError, hourly\_dt\_rmse, hourly\_rf\_rmse, hourly\_gbt\_rmse],  'Model': ['Mean Energy Consumption (kWh or kWh/hr)','Linear Regression Test R2','Linear Regression Test RMSE', 'Decision Tree Test RMSE', 'Random Forest Test RMSE', 'Gradient Boosted Test RMSE'],  }  model\_results = pd.DataFrame(result\_data).set\_index('Model')  model\_results |

**Results**

Daily Energy Usage

The daily energy consumption for all SmartMeters are plotted in Figure 1. The red lines indicate the official UK holidays during the time period from November 2011 to February 2014.

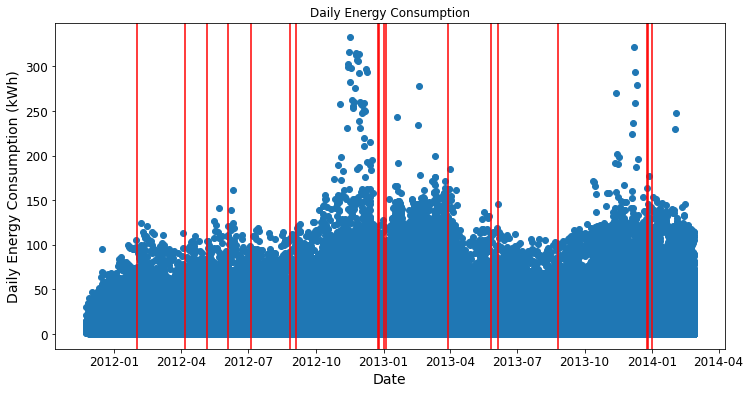


Figure 1. Daily Energy Consumption

The mean daily energy consumption for all households is 10.20 kWh. For UK holidays, the mean daily energy consumption is 10.34 kWh and for non holiday dates , the mean daily energy consumption is 10.19 kWh, this suggests that there is a very slight increase in energy consumption during holidays.

Figure 2 shows the average daily energy consumption across all households by the month of year. The winter months see the highest energy consumption, while the summer months see the lowest energy consumption.

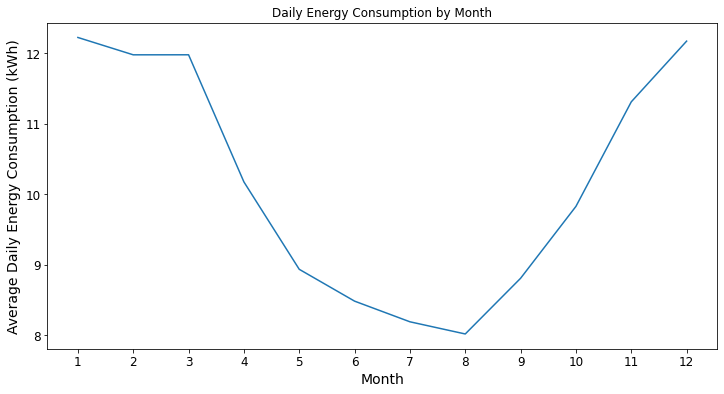


Figure 2. Daily Energy Consumption by Month

Figure 3 shows the average daily consumption by the day of week. The highest energy consumption occurs on Sunday, followed by Saturday. This makes sense since people tend to be at home more during the weekends as compared to the weekdays.

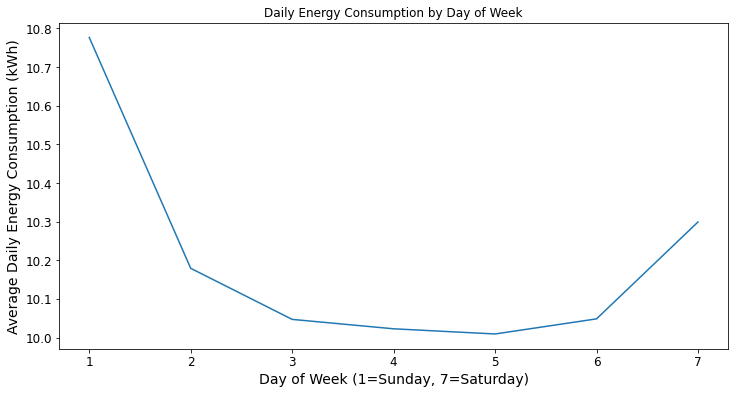


Figure 3. Daily Energy Consumption by Day of Week

Hourly Energy Usage

We plot the hourly energy usage of a single SmartMeter in Figure 4. The red lines indicate the official UK holidays.

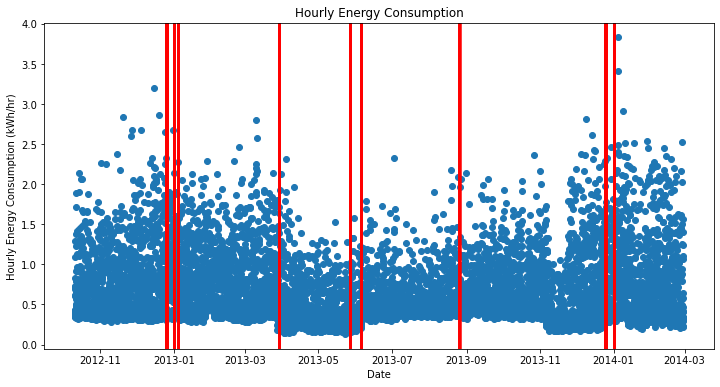


Figure 4. Hourly Energy Consumption for a single SmartMeter

This household uses an average of 0.601 kWh per hour, or 14.4 kWh a day. During the holidays, this household uses an average of 0.691 kWh/hr, while using an average of 0.599 kWh/hr on non-holiday dates. In this household, there is an increase in energy consumption during the holidays.

Figure 5 shows that in this household, the energy consumption is the lowest during the overnight hours, progressively increasing during the morning and afternoon, and finally peaking during the evening hours.

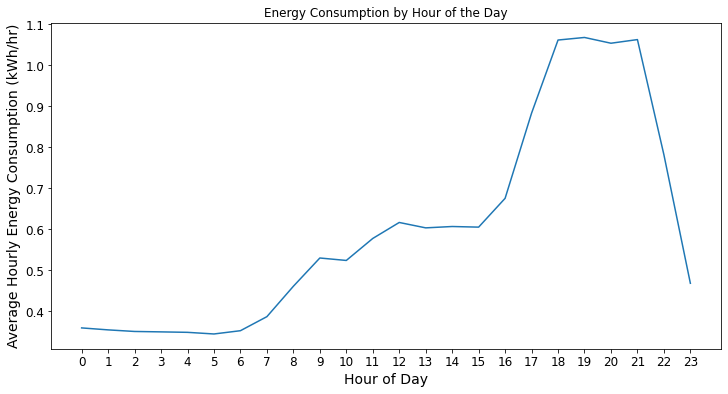


Figure 5. Energy Consumption by Hour of Day

Energy Consumption by ACORN/Socialeconomic Groups

ACORN is a consumer classification system in the United Kingdom which categorizes the households into demographic groups based on social factors and population behavior. There are 17 different categories ranging from ACORN-A to ACORN-Q, where ACORN-A is the most affluent group and ACORN-Q is the poorest group. Figure 6 shows daily energy consumption plot from Figure 1 segmented into six different ACORN categories while Figure 7 shows the average daily energy consumption for each ACORN group. If we exclude the undefined ACORN groups (ACORN- and ACORN-U), the energy consumption is generally higher for the more affluent ACORN groups (ACORN-A through ACORN-D), while lower for the least affluent ACORN groups (ACORN-P and ACORN-Q).

Figure 8 shows the same message while reducing the ACORN groups into three main socioeconomic groups: affluent, comfortable, and adversity. Although the groups with unknown ACORN status have the highest energy consumption, it is clear that the affluent socioeconomic group consume more energy than the comfortable socioeconomic groups, and the adversity socioeconomic group consume the least amount of energy.

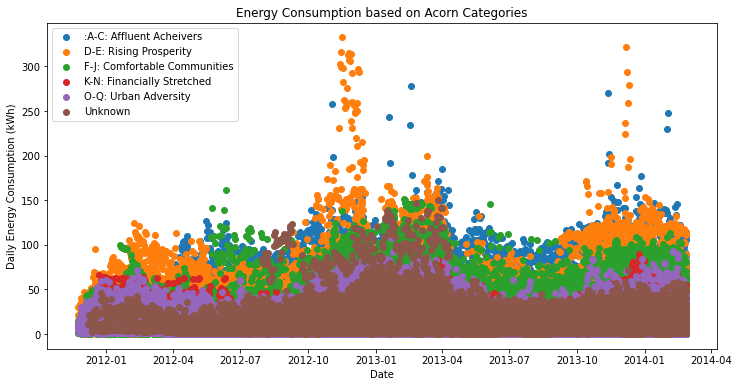


Figure 6. Energy Consumption based on Grouped Acorn Categories

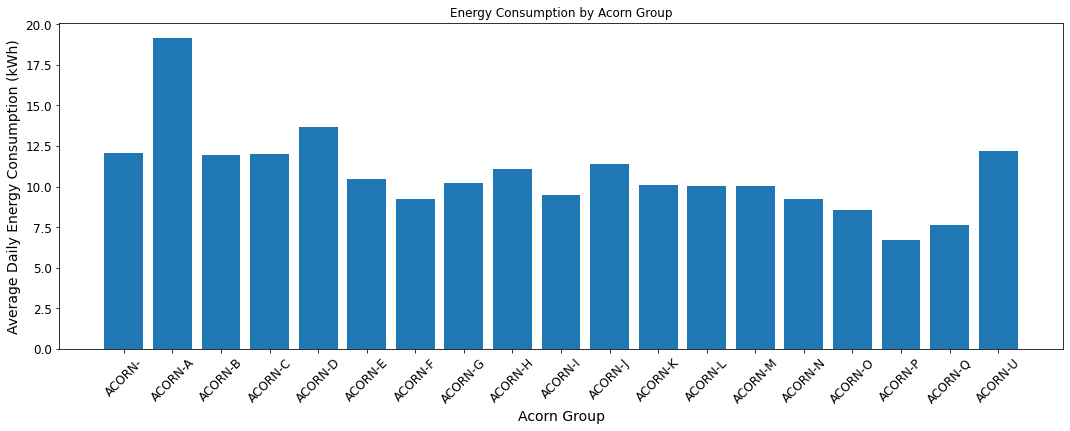


Figure 7. Average Energy Consumption based on Acorn Categories.

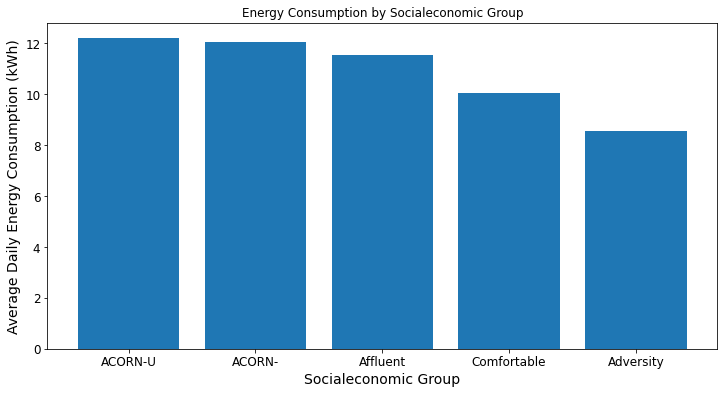


Figure 8. Average Energy Consumption based on Socialeconomic Group

Energy Consumption by Billing Method

There are two billing methods reported in this dataset: standard billing or time-of-use (ToU). The standard billing method charges the same price at all hours of the day, while the ToU method charges a higher rate during peak demand and a lower rate during off-peak hours. Figure 9 shows that households which use a standard billing method have a higher energy consumption (average of 10.36 kWh per day) than households which use the ToU billing method (average of 9.55 kWh per day).

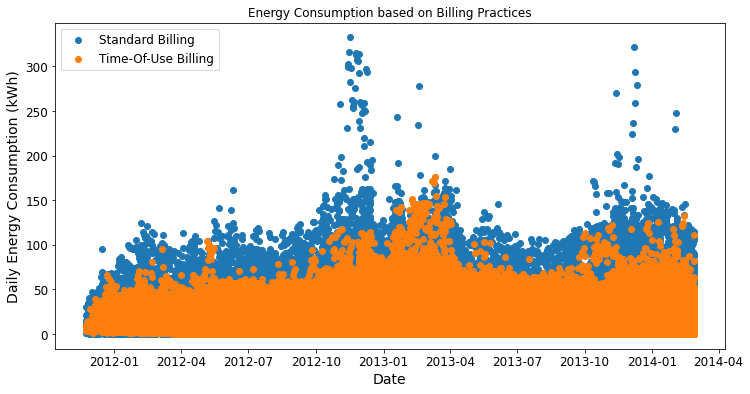


Figure 9. Energy Consumption based on Billing Method

Energy Consumption by Weather

Figure 10 shows the high and low daily temperatures from November 2011 to February 2014.

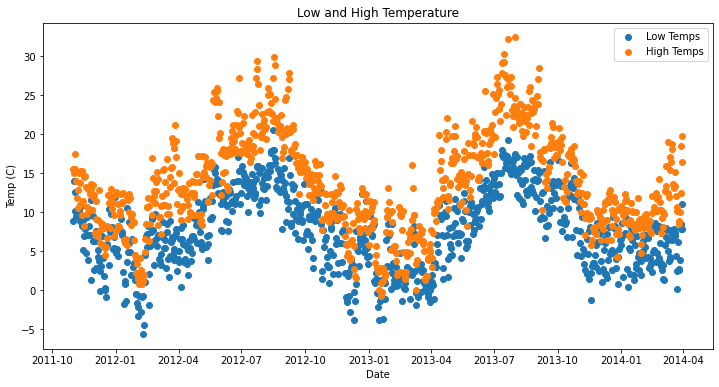


Figure 10. Daily High and Low Temperatures

The correlation between the daily high temperature and the energy consumption is -0.17.

Figure 11 shows the average daily energy consumption based on the weather outlook for that day. The highest energy consumptions was observed when the weather forecast was a partly cloudy evening, while the lowest energy consumption was observed on a clear day.

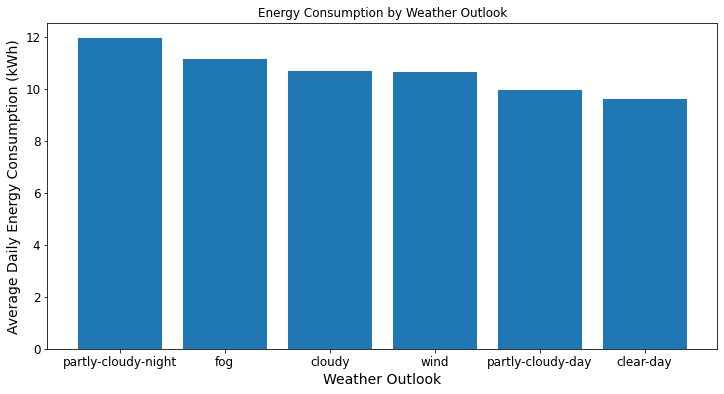


Figure 11. Daily energy consumption by weather outlook.

Figure 12 shows the daily energy consumption based on wind speed. With a correlation of 0.03 between daily energy consumption and wind speed, it does not appear that wind speed has a strong impact on the energy consumption.

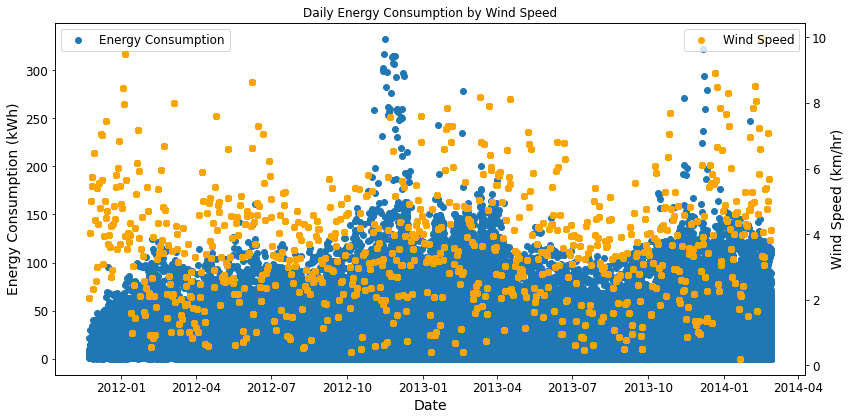


Figure 12. Energy Consumption by wind speed.

Figure 13 shows the average daily energy consumption based on the UV Index. A lower UV index suggests a cloudy day while the higher UV Index indicates increased sunshine. The figure suggests that the energy consumption is the highest when the UV Index is low while the energy consumption is lowest with a high UV Index.

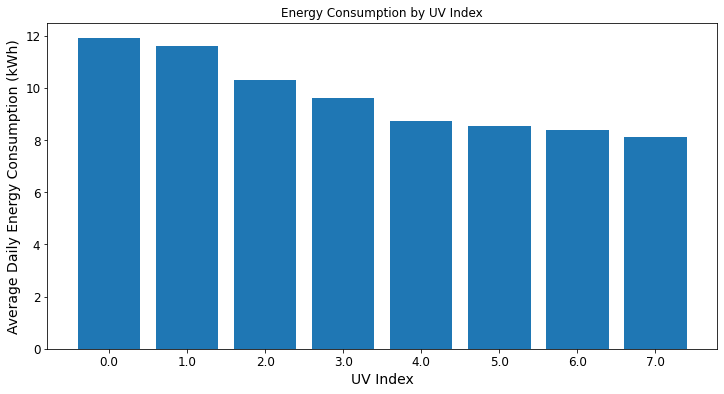
****

Figure 13. Energy Consumption by UV Index.

Regression Models

*Daily Energy Consumption Dataset*

A series of regression analysis was performed on the daily energy consumption dataset using Spark ML API. The following input variables were used for all models: stdorTou (standing billing or Time-of-Use billing), Acorn category, socioeconomic category, wind direction, wind speed, weather outlook, dew point, cloud cover, atmospheric pressure, precipitation, visibility, humidity, UV Index, daily high temperature, daily low temperature, moon phase, month, day, and holiday.

* Linear Regression
  + The linear regression model is: Daily Energy Consumption = 13.89 + 1.07 (standard billing) -0.49 (ACORN-E) -1.74 (ACORN-Q) -0.86 (ACORN-F) + 0.7 (ACORN-H) + 0.57 (ACORN-L) + 2.68 (ACORN-D) + 0.62 (ACORN-G) + 7.97 (ACORN-A) -0.13 (ACORN-N) + 0.83 (ACORN-C) -2.54 (ACORN-P) + 0.4 (ACORN-M) + 1.09 (ACORN-J) – 0.81 (ACORN-O) – 0.7 (ACORN-I) + 0.93 (ACORN-U) + 0.65 (ACORN-B) + 0.78 (Affluent) – 0.83 (Adversity) + 0.93 (ACORN-U) + 0.01 (wind speed) -0.03 (fog) + 0.19 (partly cloudy night) -0.22 (cloudy) -0.1 (dew point) + 0.53 (cloud cover) -0.6 (rain) – 0.03 (visibility) + 1.12 (humidity) + 0.54 (UV Index 1) -0.35 (UV Index 4) -0.35 (UV Index 5) + 0.27 (UV Index 2) -0.19 (UV Index 6) + 0.45 (UV Index 0) -0.05 (High temp) + 0.58 (December) + 0.4 (January) -0.01 (November) -0.77 (October) -0.79 (August) + 0.12 (February) – 0.62 (July) – 0.99 (September) – 0.94 (June) -0.81 (May) -0.27 (April) -0.54 (Thursday) -0.47 (Tuesday) -0.34 (Monday) -0.23 (Saturday) -0.53 (Friday) -0.51 (Wednesday)
  + The most important feature (as measured by the highest magnitude) is the Acorn-A group.
  + The test R2 is 0.089, which suggests that the linear regression model does not do a good job at all of explaining the variance in the energy consumption data. The Test RMSE (root mean squared error) is 8.71. Given that the mean daily energy consumption value is 10.20, such a high RMSE suggests that the linear regression model does not do a good job of accurately predicting the data.
  + Figure 14 shows the predicted daily energy consumptions as compared to the actual energy consumptions. The model predictions fall between 2.77 and 23.41 kWh while the actual energy consumptions range from 0.001 kWh to 332.556 kWh.

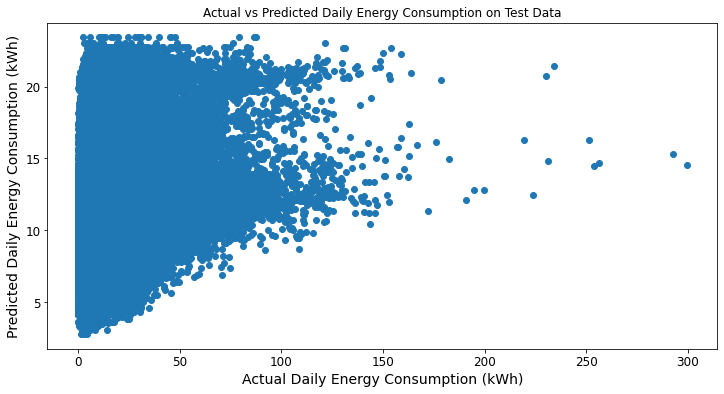


Figure 14. Actual vs Predicted Daily Energy Consumption

* Decision Tree
  + The Test RMSE is 8.75, which is not an improvement when compared to the Linear Regression Model
  + The most important features are the daily high temperature, ACORN-A, and the adversity socioeconomic group.
* Random Forest
  + The Test RMSE is 8.74, which is a slight improvement from the linear regression and decision tree models.
  + The most important features are the daily high temperature, the adversity socioeconomic group, and the daily low temperature.
* Gradient Boosted Tree
  + The Test RMSE is 8.69, which is an improvement compared to the other regression models but is still too high to be considered a useful model.
  + The most important features are standard billing, ACORN-P, and the adversity socioeconomic group.

With such high test RMSE values, we cannot accurately use the models we have built to predict a household’s energy consumption based on the input variables that we have. We may need additional information, such as the size of the house and the number of people living inside the house, to be able to better predict energy consumption.

*Hourly Energy Consumption Dataset*

Given the failure to accurately predict energy consumption in the daily energy consumption dataset, I decided to focus the next set of regression analysis for a single Smart Meter instead of the entire dataset of Smart Meters. I wanted to see if the weather data in the hourly energy consumption dataset and the month, day of week, and hour information could be used to predict the energy consumption for a single household. For this model, the following input variables were included: wind direction, wind speed, weather outlook, dew point, precipitation, visibility, humidity, UV Index, temperature, month, day, hour, and holiday.

* Linear Regression
  + The linear regression model is: Hourly Energy Consumption = 0.5 + 0.01 (wind speed) - 0.02 (clear night) + 0.06 (partly cloudy night) + 0.04 (humidity) + 0.07 (December) + 0.08 (January) - 0.01 (November) + 0.02 (February) - 0.17 (May) – 0.02 (July) -0.12 (April) - 0.04 (June) - 0.01 (Friday) -0.14 (3AM) + 0.05 (4PM) + 0.21 (5PM) + 0.34 (6PM) + 0.35 (7PM) + 0.34 (8PM) + 0.35 (9PM) + 0.13 (10PM) - 0.14 (12AM) -0.13 (1AM) -0.04 (11PM) -0.14 (4AM) -0.15 (5AM) -0.15 (6AM) -0.12 (7AM) -0.14 (2AM).
  + The most important features (features with the highest magnitude) are the evening hours: 7PM, 9PM, 6PM, and 8PM.
  + The test R2 is 0.45, which suggests that the linear regression model explains 44% of the variance in the hourly energy consumption data. The Test RMSE (root mean squared error) is 0.29. This is an improvement from the previous linear regression model on the overall daily energy consumption dataset.
  + Figure 15 shows the predicted daily energy consumptions as compared to the actual energy consumptions. The model predictions fall between 0.21 and 1.09 kWh/hr while the actual hourly energy consumptions range from 0.13 to 2.68 kWh/hr.

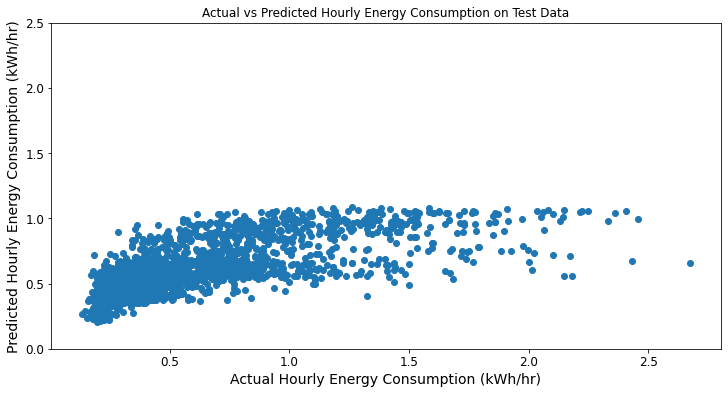


Figure 15. Actual vs. Predicted Hourly Energy Consumption

* Decision Tree
  + The Test RMSE is 0.31, which is slightly higher than the Linear Regression Model
  + The most important features are the evening hours: (6PM, 9PM, 8PM, 7PM).
* Random Forest
  + The Test RMSE is 0.30, which is a slightly higher than the Linear Regression Model.
  + The most important features are the evening hours (7PM, 6PM, 8PM, 9PM).
* Gradient Boosted Tree
  + The Test RMSE is 0.26, which is an improvement compared to the other regression models.
  + The most important features are 10PM, temperature, and April.

**Conclusions**

What Worked

In this project, I have successfully demonstrated how to use Spark to efficiently load in multiple datasets. All the daily meter data contained in 112 csv files can be loaded into a single Spark DataFrame. Information from the weather and household csv files can also be successfully loaded in and merged into a single Spark DataFrame with the meter data using a series of join operations.

After consolidating the multiple datasets into a single daily energy consumption Spark DataFrame, I could successfully select data to plot using matplotlib. Using the plotting functionality, I was able to get insight into some key factors which affect energy consumption: socioeconomic factors, time factors such as month, day of week, and hour, and the method of billing. I also gained insight as to how weather affects energy consumption as well.

What Didn’t Work

Performing regression analysis on the entire dataset of Smart Meters provided to be unsuccessful as shown by the high root mean squared errors for all the regression models which I tried. The variance in the data was too large to effectively predict energy consumption for the overall dataset. There could be additional input variables, such as house size and number of residents living in the household, which could improve the accuracy of the regression models. However, focusing the regression analysis to a single meter appeared to work although this is not a feasible process for estimating energy consumption across an entire city, as a regression model would have to be built for every single Smart Meter.

Lessons Learned

When performing an analysis on a large dataset such as this one, it is important to review the data and narrow the focus of the analysis to what is necessary instead of including data which is available but not essential to the underlying analysis. For example, the hourly dataset includes readings from a half hourly interval. While there may be a specific case for such data, in most cases the hourly or even daily dataset is sufficient. By filtering out what information is critical for the analysis and what is not, one can minimize the computation time.

When plotting the data using matplotlib, it is quicker to plot aggregated such as average energy consumptions by binary class instead of plotting the raw datapoints. Using the collect() function or converting a Spark DataFrame to a Pandas dataframe takes computation time.

Next Steps

I would like to try using time series regression to improve the predictive models, especially since our energy consumption data shows a seasonal trend. I would like to use statistical tests to determine whether the difference between two binary classes are statistically significant or not. Java’s version of Spark ML lib has a student t-test function but PySpark’s version does not.

Finally, to optimize the performance of Spark, I would try repartitioning the Spark DataFrames. The suggested partition size for optimal Spark performance is between 100-200MB[[1]](#footnote-1) and it has been suggested that one should repartition before performing multiple join operations[[2]](#footnote-2).

1. <https://nealanalytics.com/blog/databricks-spark-jobs-optimization-techniques-shuffle-partition-technique-part-1/> [↑](#footnote-ref-1)
2. <https://www.analyticsvidhya.com/blog/2020/11/8-must-know-spark-optimization-tips-for-data-engineering-beginners/> [↑](#footnote-ref-2)