Cairo University - Faculty of Engineering

Computer Engineering Department

Machine learning

**Bike\_Sharing\_Demand\_Prediction**

**Team 10**

**Team members:**

|  |  |  |
| --- | --- | --- |
| **Name** | **Section** | **B.N.** |
| **Aya Adel Hassan** | **1** | **17** |
| **Dina Alaa Ahmad** | **1** | **25** |
| **Dai Alaa Hassan** | **1** | **28** |
| **Nerdeen Ahmad Shawqi** | **2** | **28** |

**Submitted to: Eng. Yahia Zakaria**

**Team members’ contribution:**

**All of the team members worked in understanding and preprocessing of the data.**

* **Aya Adel** 
  + **SVM**
  + **Logistic regression**
* **Dina Alaa**
  + **Linear regression**
  + **Lasso**
  + **ridge**
  + **SGDRegressor**
* **Dai Alaa**
  + **AdaBoost**
  + **XGBoost**
* **Nerdeen Ahmad**
  + **Decision Tree**
  + **Random Forest**

**Problem definition and motivation:**

**Recently, renting bikes has been widely spread because of how easy the process became after Bike sharing systems were introduced. Through these systems, user is able to easily rent a bike from a particular position and return back at another position. This results in increase in the average number of rented bikes. The problem is to predict the total count of bikes rented during each hour, using only information available prior to the rental period.**

**Evaluation metrics:**

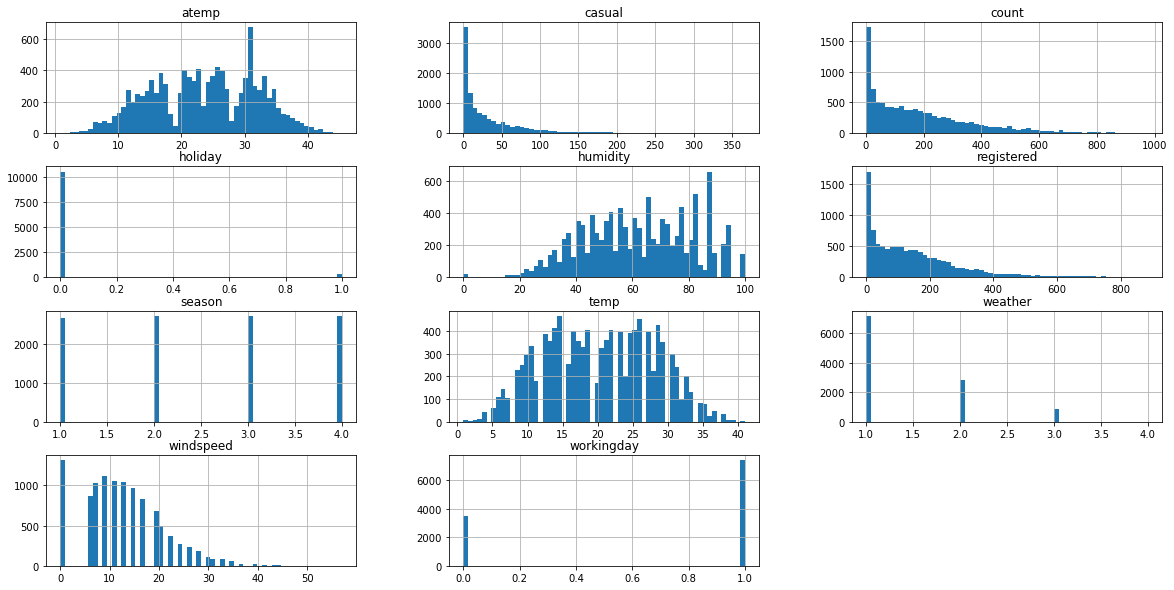
* **the Root Mean Squared Logarithmic Error (RMSLE)Precession**
* **root mean squared error (RMSE)**

**Dataset and references:**

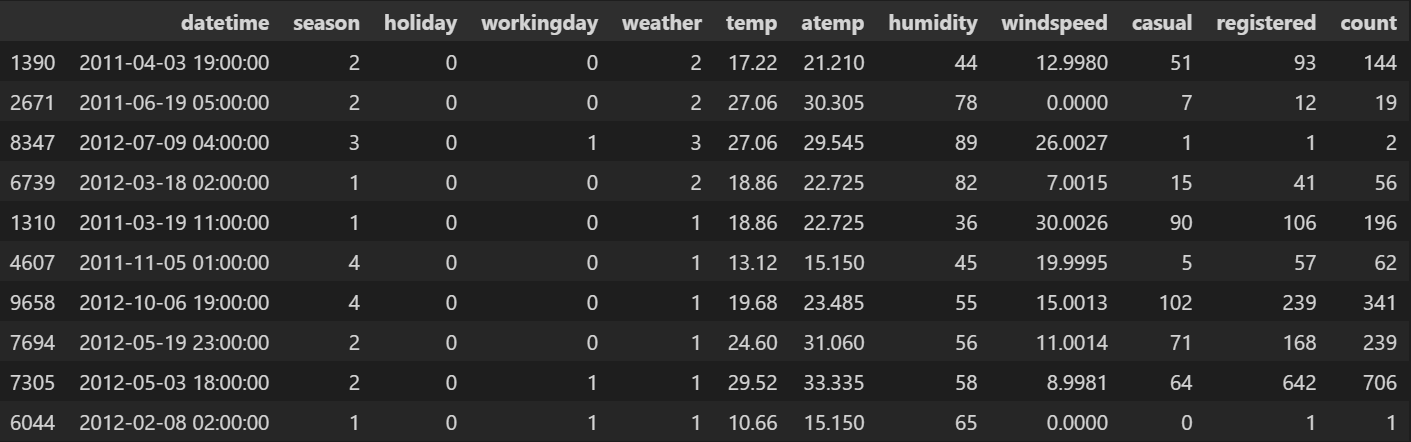
* **Dataset**
  + <https://www.kaggle.com/competitions/bike-sharing-demand/data>
* **Paper**
  + <https://www.kaggle.com/competitions/csce5300-competition/overview>
  + <https://www.researchgate.net/publication/259382357_Bike-Sharing_Dataset>
  + <http://arno.uvt.nl/show.cgi?fid=156914>
  + <https://www.researchgate.net/publication/348974351_Machine_Learning_Approaches_to_Bike-Sharing_Systems_A_Systematic_Literature_Review>
* **Analyze Dataset:**
* **Columns and datatypes**

|  |  |
| --- | --- |
| **Column** | **Data Type** |
| **Date time** | **Object** |
| **Season** | **Int64** |
| **Holiday** | **Int64** |
| **Working day** | **Int64** |
| **Weather** | **Int64** |
| **Temp** | **Float64** |
| **Atemp** | **Float64** |
| **Humidity** | **Int64** |
| **Wind speed** | **Float64** |
| **Causal** | **Int64** |
| **Registered** | **Int64** |
| **Count** | **Int64** |

* **Rows->10886**
* **Data histogram**

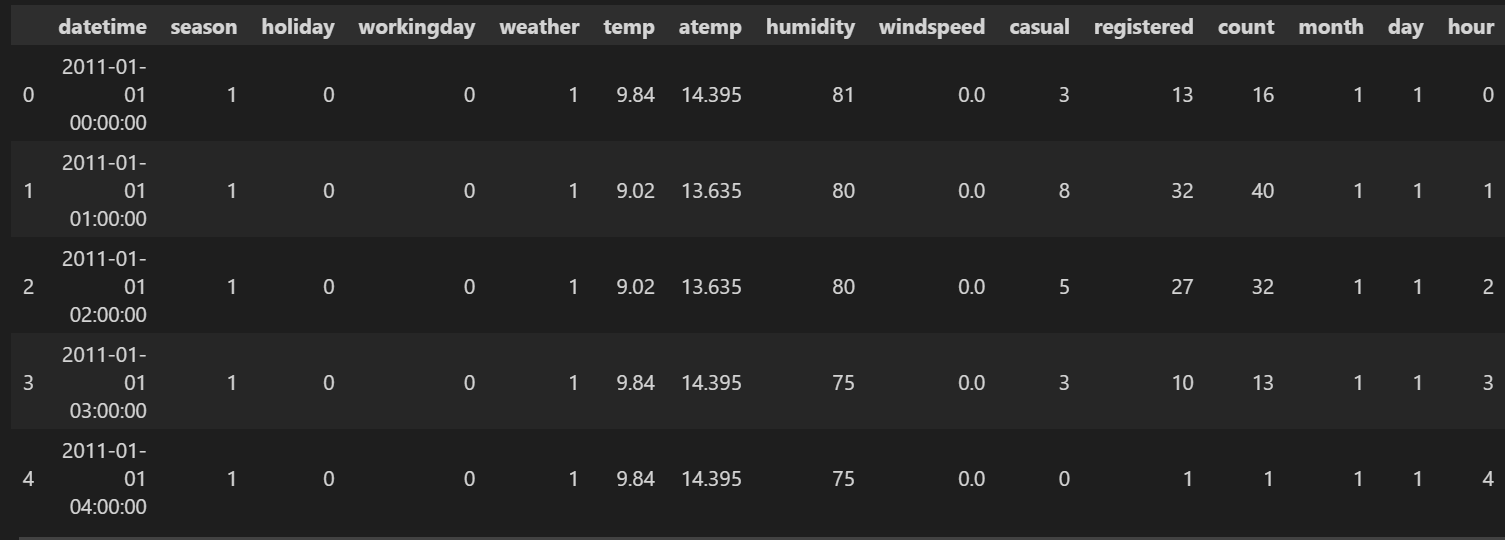
****

**Random samples of the data**

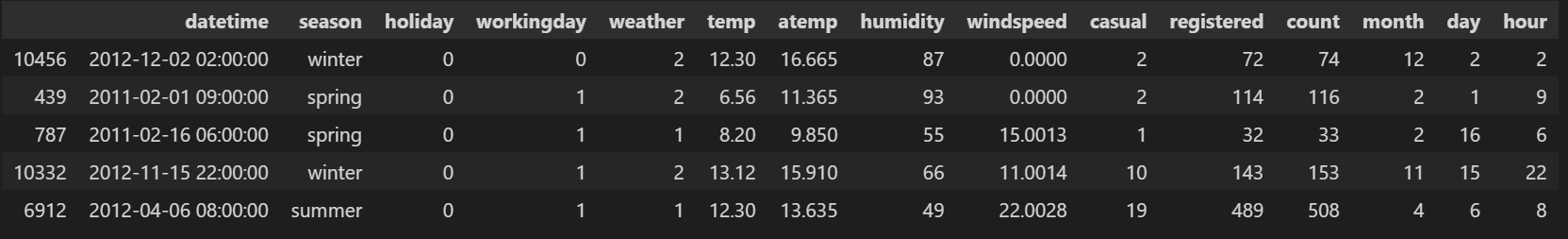
****

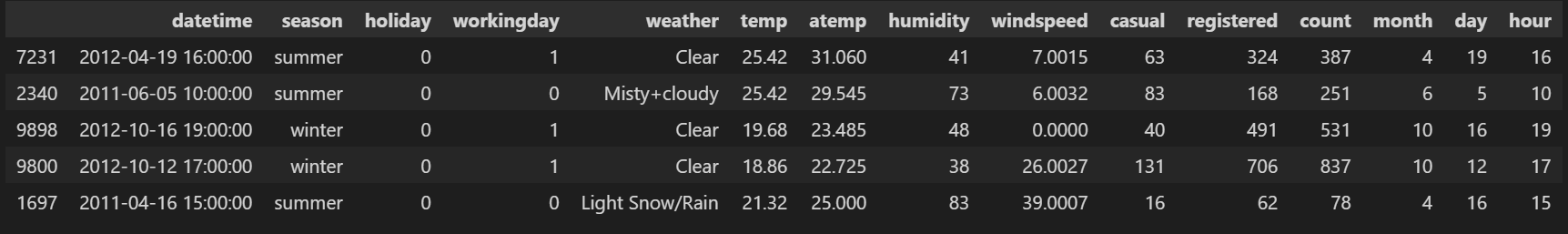
**Dataset preprocessing**

* **Checking for null data and removing it from the data.**
* **Splitting the date time column to separate columns (hour-day-month) to convert the object data type to int64 data type to deal with it.**

****

* **Replace the values of the season {1: Spring, 2: Summer, 3: Fall, 4: Winter} and weather {1: Clear, 2: Misty + Cloudy, 3: Light Snow/Rain, 4: Heavy Snow/Rain } with appropriate categorical values to understand them better**

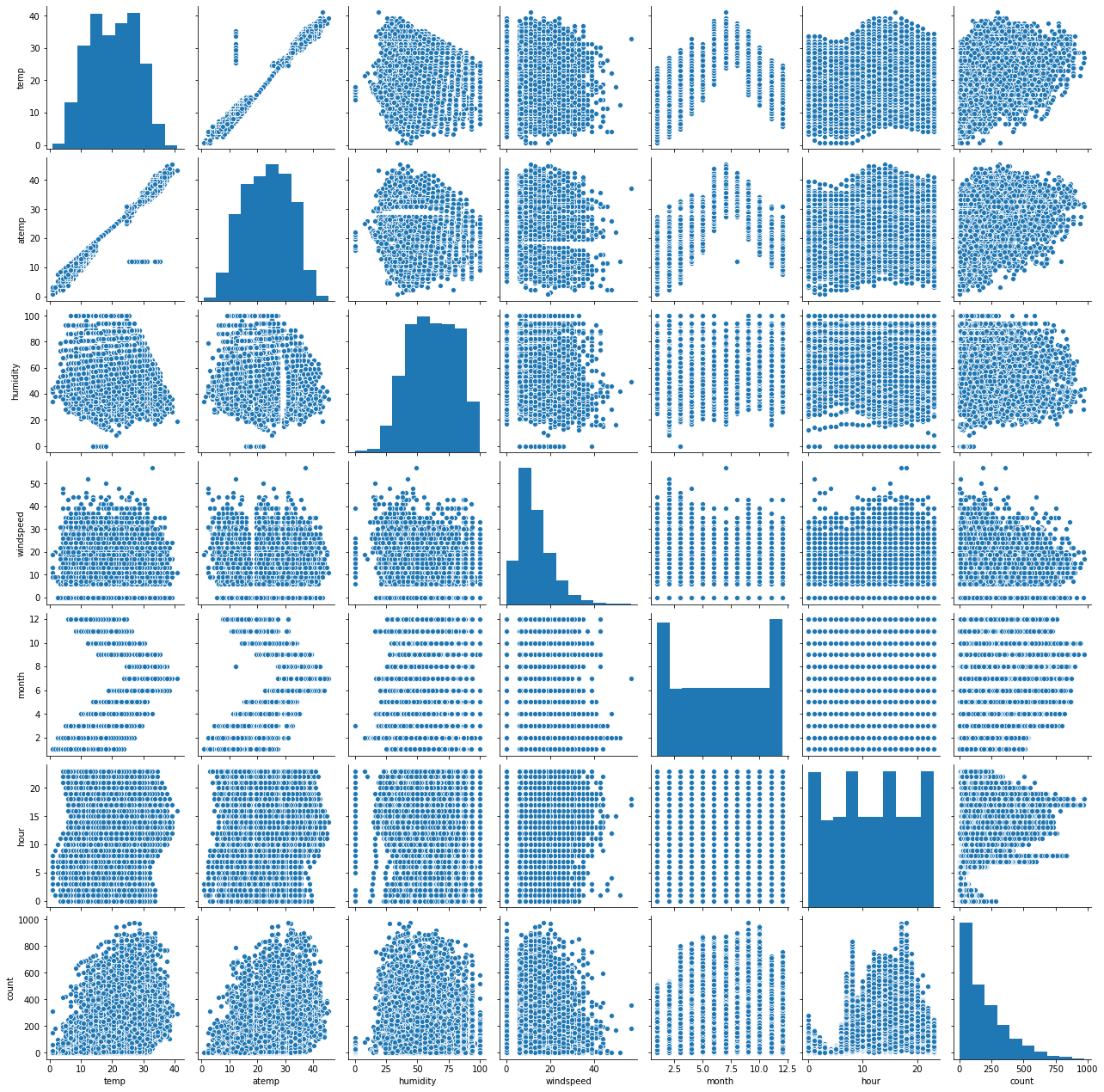
****

****

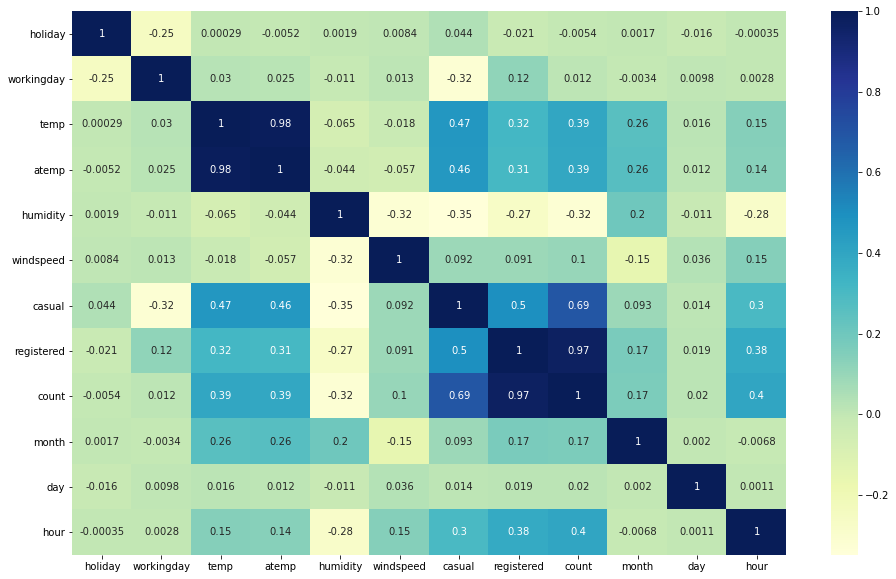
* **Dataset columns after modification**

|  |  |
| --- | --- |
| **Column** | **Data Type** |
| **Date time** | **Object** |
| **Season** | **Object** |
| **Holiday** | **Int64** |
| **Working day** | **Int64** |
| **Weather** | **Object** |
| **Temp** | **Float64** |
| **Atemp** | **Float64** |
| **Humidity** | **Int64** |
| **Wind speed** | **Float64** |
| **Causal** | **Int64** |
| **Registered** | **Int64** |
| **Count** | **Int64** |
| **Month** | **Int64** |
| **Day** | **Int64** |
| **Hour** | **Int64** |

* **Pair plot to check for any correlation between variables**

****

* **Heat map to check for correlation between variables**

****

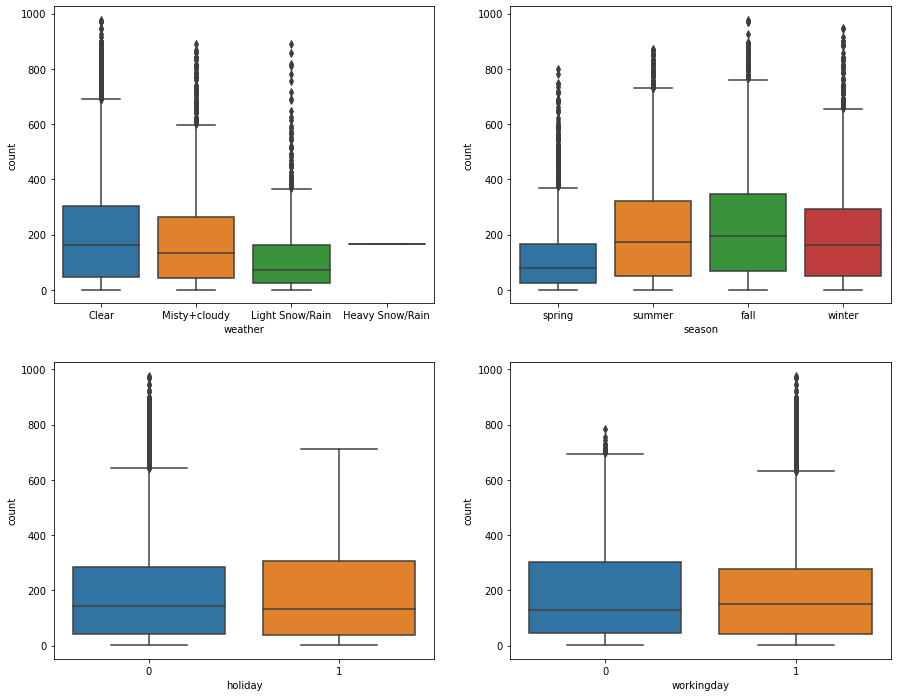
**From the pair plot and heat map we deduce that atemp and temp are highly correlated to each other**

**Pair plot -> linear relationship**

**Heat map -> 0.98**

**So we dropped the temp column from our dataset**

* **Outliers observation of the dataset**

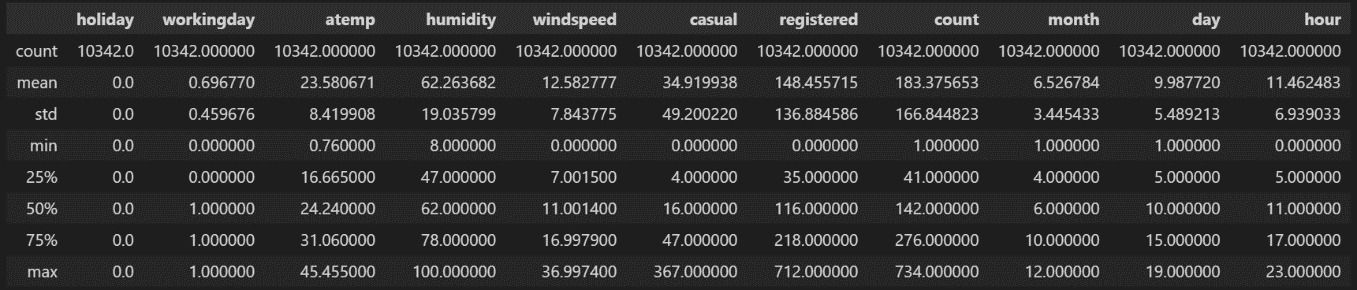
****

**We used zscore to remove outliers**

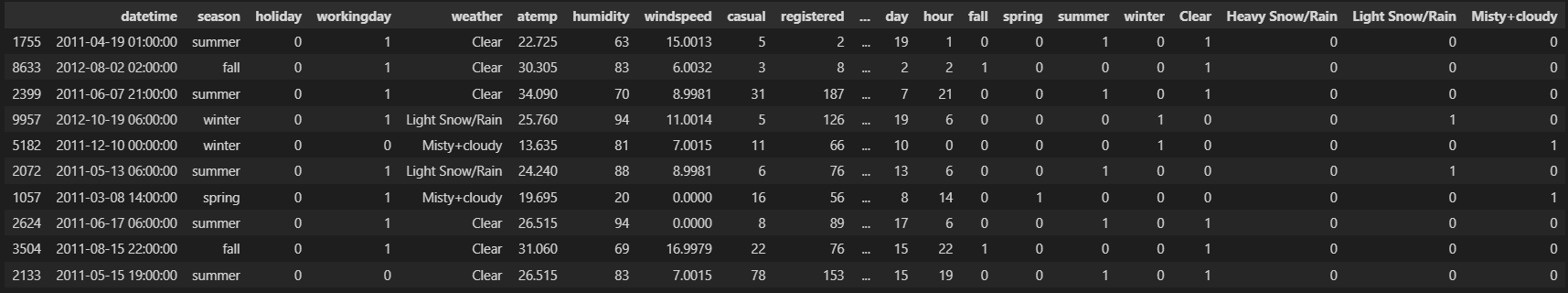
**The dataset size was 10886**

**After removing outliers became 10342**

* **Dataset description after removing outliers**

****

* **Convert categorical variable into dummy (one hot encoding) indicator variables to deal with it.**

****

* **Dropped unnecessary data** 
  + - Date time as we separate it into { month, day, hour }
    - Season as we get its data in separate columns (dummy data)
    - Weather as we get its data in separate columns (dummy data)
    - Casual and Registered as they are redundant for count
* **Data set after dropping unnecessary columns**

|  |  |
| --- | --- |
| **Column** | **Data Type** |
| **Holiday** | **Int64** |
| **Working day** | **Int64** |
| **Weather** | **Object** |
| **Atemp** | **Float64** |
| **Humidity** | **Int64** |
| **Wind speed** | **Float64** |
| **Causal** | **Int64** |
| **Registered** | **Int64** |
| **Count** | **Int64** |
| **Month** | **Int64** |
| **Day** | **Int64** |
| **Hour** | **Int64** |
| **Fall** | **Uint8** |
| **Spring** | **Uint8** |
| **Summer** | **Uint8** |
| **Winter** | **Uint8** |
| **Clear** | **Uint8** |
| **Heavy Snow/Rain** | **Uint8** |
| **Light Snow/Rain** | **Uint8** |
| **Misty + Cloudy** | **Uint8** |

* **Separating dataset to training data 80% and testing data 20%**
* **After separating the data we normalize each of them separately**
* **We didn’t use the test data (leaving it for last evaluation).**
* **We then separate the training data into training 80% and validation 20%**

**Training the models**

**Before Regularization**

1. **Linear Regression**
   1. **Evaluation metrics for training data**
      1. Root mean squared error -> 0.185939
      2. R squared value -> 0.340961
   2. **Evaluation metrics for validation data**
      1. Root mean squared error -> 0.179420
      2. R squared value -> 0.352218

**The values shows that there is under fitting in the model**

1. **Decision Trees**
   1. **Evaluation metrics for training data**
      1. Root mean squared error -> 0
      2. R squared value -> 1
   2. **Evaluation metrics for validation data**
      1. Root mean squared error -> 0.111459
      2. R squared value -> 0.750011

**Overfitting**

1. **Ensemble Learning (Bagging->Random Forest)**
   1. **Evaluation metrics for training data**
      1. Root mean squared error -> 0.032052
      2. R squared value -> 0.980416
   2. **Evaluation metrics for validation data**
      1. Root mean squared error -> 0.082713
      2. R squared value -> 0.862332

**Overfitting**

1. **Ensemble** **Learning (Boosting->AdaBoost)**
   1. **Evaluation metrics for training data**
      1. Root mean squared error -> 0.160657
      2. R squared value is -> 0.507994
   2. **Evaluation metrics for validation data**
      1. Root mean squared error -> 0.162524
      2. R squared value is -> 0.468472
2. **Ensemble** **Learning (Boosting->XGBoost)**
   1. **Evaluation metrics for training data**
      1. Root mean squared error -> 0.043910
      2. R squared value is -> 0.963246
   2. **Evaluation metrics for validation data**
      1. Root mean squared error -> 0.078710
      2. R squared value is -> 0.875334
3. **Support Vector Machine (SVM)**
   1. **Evaluation metrics for training data**
      1. Root mean squared error -> 0.160659
      2. R squared value is -> 0.507982
   2. **Evaluation metrics for validation data**
      1. Root mean squared error -> 0.156724
      2. R squared value is -> 0.505736

**From results:**

* Linear regression -> there is under fitting.
* Decision tree -> there is overfitting.
* Random Forest-> there is overfitting.
* AdaBoost -> very poor results.
* XGBoost -> there is overfitting
* SVM -> very poor results.

**So we tried parameter tuning as:**

* Simple model will result in a very poor generalization of data. At the same time, complex model may not perform well in test data due to over fitting.
* We need to choose the right parameters in between simple and complex model.
* Regularization helps to choose preferred model complexity, so that model is better at predicting.
* Regularization is nothing but adding a penalty term to the objective function and control the model complexity using that penalty term.
* It can be used for many machine learning algorithms.

**After parameter tuning**

1. **Linear Regression**
   1. **Hyper parameters tuning: Alpha and solver type in ridge LR**
      1. **Alpha = 1 and solver type=”svd”**
         1. **Evaluation metrics for training data**
            1. Root mean squared error -> 0.185941
            2. R squared value -> 0.340942
         2. **Evaluation metrics for validation data**
            1. Root mean squared error -> 0.179405
            2. R squared value -> 0.352320
   2. **Hyper parameters tuning: Alpha in Lasso LR**
      1. **Alpha = 0.0001**
         1. **Evaluation metrics for training data**
            1. Root mean squared error -> 0.185959
            2. R squared value -> 0.340819
         2. **Evaluation metrics for validation data**
            1. Root mean squared error -> 0.179415
            2. R squared value -> 0.352252
2. **Decision Trees**

**Max depth -> decides the height of the tree**

**Splitter -> decides how to split the data for training**

* 1. **hyper parameter tuning: max depth & splitter** 
     1. **Max depth = 8 and splitter=”best”**
        1. **Evaluation metrics for training data**
           1. Root mean squared error -> 0.105975
           2. R squared value -> 0.785919
        2. **Evaluation metrics for validation data**
           1. Root mean squared error -> 0.105999
           2. R squared value -> 0.773961

1. **Ensemble Learning (bagging->Random forest)**

**Max depth -> decides the height of the tree**

**N estimator -> decides the number of trees that we use**

* 1. **hyper parameter tuning: max depth & N estimator** 
     1. **max depth= 8 and N estimators=50**
        1. **Evaluation metrics for training data**
           1. Root mean squared error -> 0.092272
           2. R squared value -> 0.837703
        2. **Evaluation metrics for validation data**
           1. Root mean squared error -> 0.094154
           2. R squared value -> 0.821613

1. **Ensemble Learning (boosting->AdaBoost)**

**Loss ->The loss function to use when updating the weights after each boosting iteration.**

**Learning rate -> Weight applied to each regressor at each boosting iteration. A higher learning rate increases the contribution of each regressor.**

**N estimator -> decides the number of trees that we use**

* 1. **hyper parameter tuning: loss & learning rate & N estimator** 
     1. **N estimators=10 & learning rate=1 & loss=”exponential”**
        1. **Evaluation metrics for training data**
           1. Root mean squared error -> 0.153477
           2. R squared value -> 0.550986
        2. **Evaluation metrics for validation data**
           1. Root mean squared error -> 0.151833
           2. R squared value -> 0.536103

1. **Ensemble Learning (boosting->XGBoost)**

**Max depth ->** **decides the height of the model.**

**Learning rate -> Weight applied to each regressor at each boosting iteration. A higher learning rate increases the contribution of each regressor.**

**N estimator -> decides the number of trees that we use**

**Alpha -> for regularization**

* 1. **hyper parameter tuning: loss & learning rate & N estimator** 
     1. **N estimators=20& learning rate=1 & max depth=6 & alpha=5**
        1. **Evaluation metrics for training data**
           1. Root mean squared error -> 0.086703
           2. R squared value -> 0.856700
        2. **Evaluation metrics for validation data**
           1. Root mean squared error -> 0.856700
           2. R squared value -> 0.828530

1. **Support Vector Machine (SVM)**

**Kernel ->** **Specifies the kernel type to be used in the algorithm.**

**C -> Regularization parameter.**

**Gamma -> Kernel coefficient for ‘rbf’, ‘poly’ and ‘sigmoid’.**

* 1. **hyper parameter tuning: Kernel & C & Gamma** 
     1. **Kernel=”rbf” & C=1 & Gamma=0.1**
        1. **Evaluation metrics for training data**
           1. Root mean squared error -> 0.088564
           2. R squared value -> 0.850484
        2. **Evaluation metrics for validation data**
           1. Root mean squared error -> 0.099394
           2. R squared value -> 0.801203

**From the results:**

* Linear regression didn’t get better
* Decision tree, random forest, AdaBoost, XGBoost, SVM achieved better results.

**Final evaluation for the best model:**

1. **The chosen model:**
   * XGBoost
2. **Evaluation metrics for testing data**
   * Root mean squared error -> 0.096741
   * R squared value -> 0.819206

**Conclusion:**

**XGBoost is better as:**

* It has the highest test set accuracy (better generalization).
* It works well with small and medium data.
* It's stable, i.e. doesn't get affected by noise.
* It drives quick learning through parallel and distributed computing.
* XGBoost uses decision trees as base learners; combining many weak learners to make a strong learner.
* It uses the output of many models in the final prediction.