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:

- **R-squared**
- **root mean squared error (RMSE)**

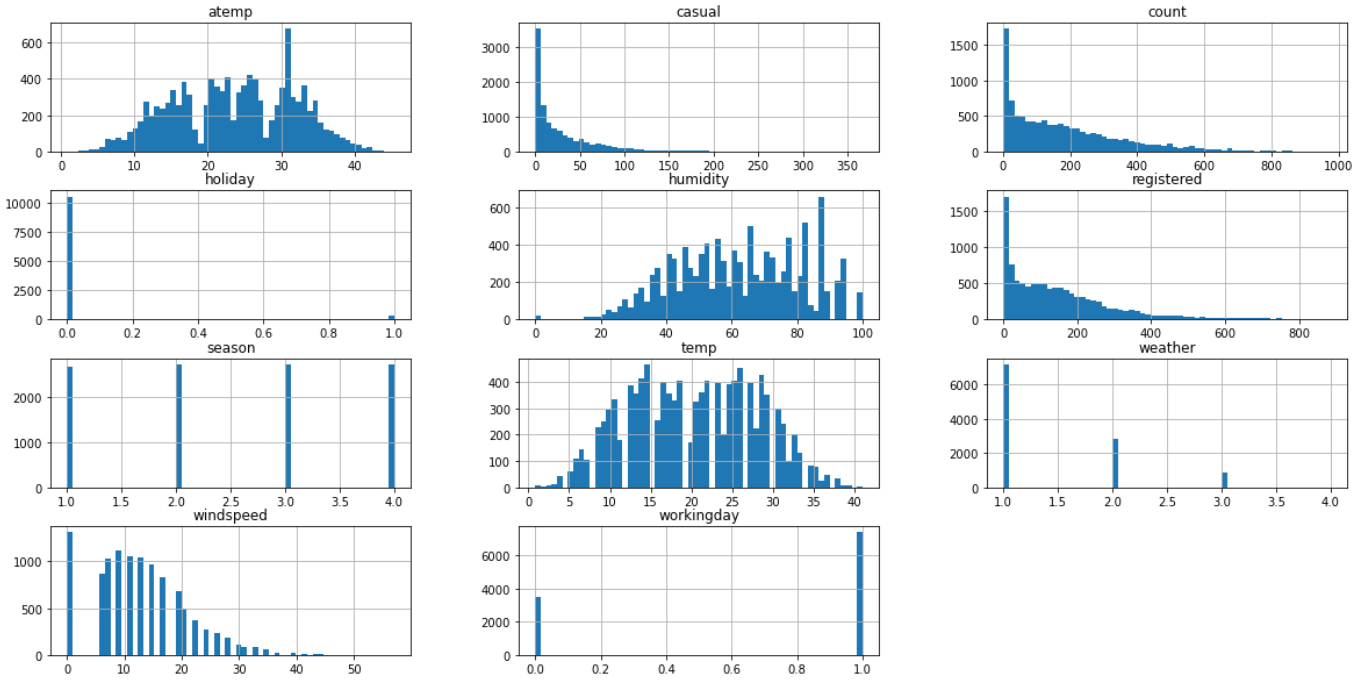
Dataset and references:

- **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

	datetime	season	holiday	workingday	weather	temp	atemp	humidity	windspeed	casual	registered	count
1390	2011-04-03 19:00:00	2	0	0	2	17.22	21.210	44	12.9980	51	93	144
2671	2011-06-19 05:00:00	2	0	0	2	27.06	30.305	78	0.0000	7	12	19
8347	2012-07-09 04:00:00	3	0	1	3	27.06	29.545	89	26.0027	1	1	2
6739	2012-03-18 02:00:00	1	0	0	2	18.86	22.725	82	7.0015	15	41	56
1310	2011-03-19 11:00:00	1	0	0	1	18.86	22.725	36	30.0026	90	106	196
4607	2011-11-05 01:00:00	4	0	0	1	13.12	15.150	45	19.9995	5	57	62
9658	2012-10-06 19:00:00	4	0	0	1	19.68	23.485	55	15.0013	102	239	341
7694	2012-05-19 23:00:00	2	0	0	1	24.60	31.060	56	11.0014	71	168	239
7305	2012-05-03 18:00:00	2	0	1	1	29.52	33.335	58	8.9981	64	642	706
6044	2012-02-08 02:00:00	1	0	1	1	10.66	15.150	65	0.0000	0	1	1

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.**

	datetime	season	holiday	workingday	weather	temp	atemp	humidity	windspeed	casual	registered	count	month	day	hour
0	2011-01-01 00:00:00	1	0	0	1	9.84	14.395	81	0.0	3	13	16	1	1	0
1	2011-01-01 01:00:00	1	0	0	1	9.02	13.635	80	0.0	8	32	40	1	1	1
2	2011-01-01 02:00:00	1	0	0	1	9.02	13.635	80	0.0	5	27	32	1	1	2
3	2011-01-01 03:00:00	1	0	0	1	9.84	14.395	75	0.0	3	10	13	1	1	3
4	2011-01-01 04:00:00	1	0	0	1	9.84	14.395	75	0.0	0	1	1	1	1	4

- **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.**

	datetime	season	holiday	workingday	weather	temp	atemp	humidity	windspeed	casual	registered	count	month	day	hour
10456	2012-12-02 02:00:00	winter	0	0	2	12.30	16.665	87	0.0000	2	72	74	12	2	2
439	2011-02-01 09:00:00	spring	0	1	2	6.56	11.365	93	0.0000	2	114	116	2	1	9
787	2011-02-16 06:00:00	spring	0	1	1	8.20	9.850	55	15.0013	1	32	33	2	16	6
10332	2012-11-15 22:00:00	winter	0	1	2	13.12	15.910	66	11.0014	10	143	153	11	15	22
6912	2012-04-06 08:00:00	summer	0	1	1	12.30	13.635	49	22.0028	19	489	508	4	6	8

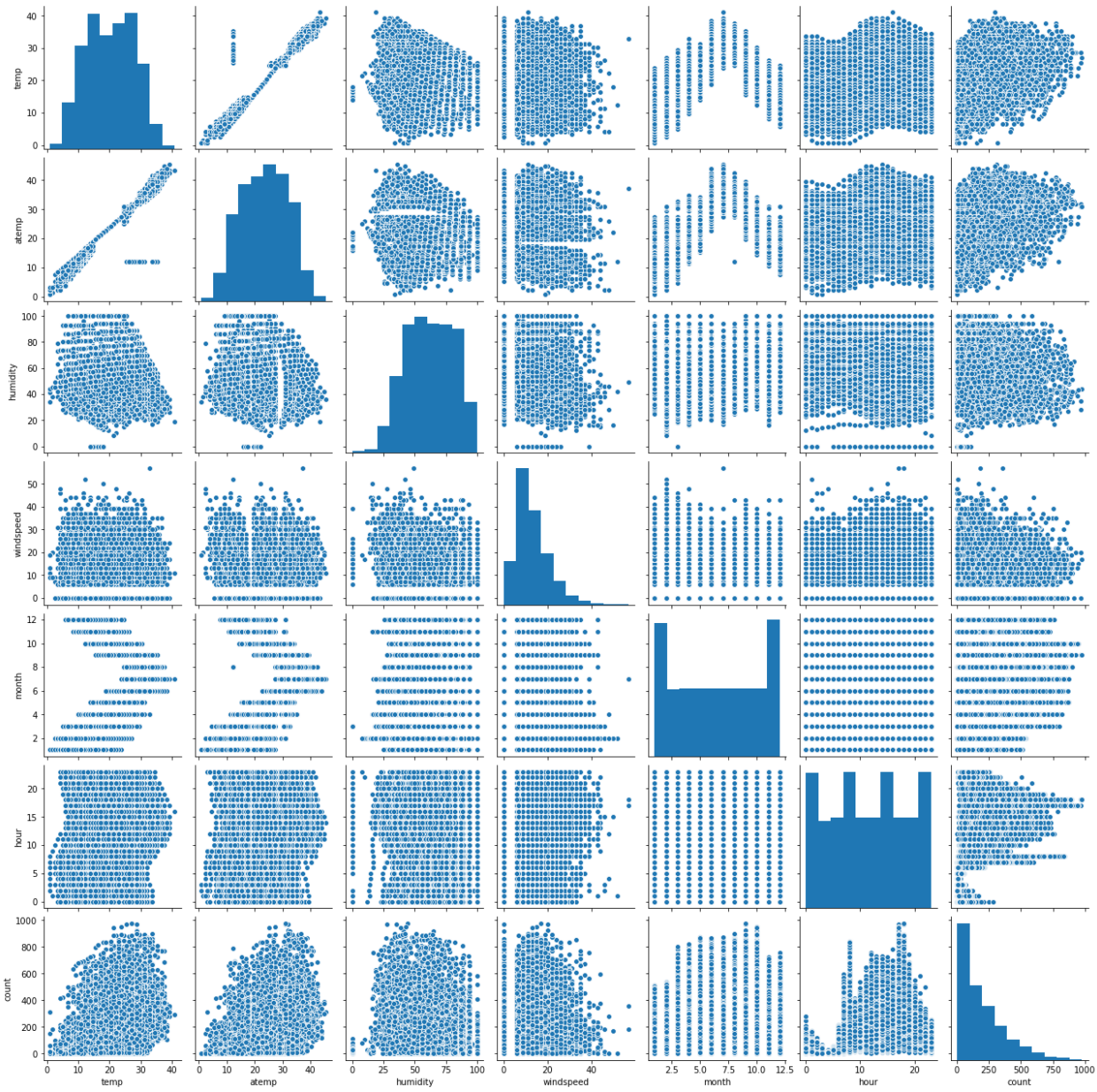
	datetime	season	holiday	workingday	weather	temp	atemp	humidity	windspeed	casual	registered	count	month	day	hour
7231	2012-04-19 16:00:00	summer	0	1	Clear	25.42	31.060	41	7.0015	63	324	387	4	19	16
2340	2011-06-05 10:00:00	summer	0	0	Misty+cloudy	25.42	29.545	73	6.0032	83	168	251	6	5	10
9898	2012-10-16 19:00:00	winter	0	1	Clear	19.68	23.485	48	0.0000	40	491	531	10	16	19
9800	2012-10-12 17:00:00	winter	0	1	Clear	18.86	22.725	38	26.0027	131	706	837	10	12	17
1697	2011-04-16 15:00:00	summer	0	0	Light Snow/Rain	21.32	25.000	83	39.0007	16	62	78	4	16	15

- **Dataset columns after modification**

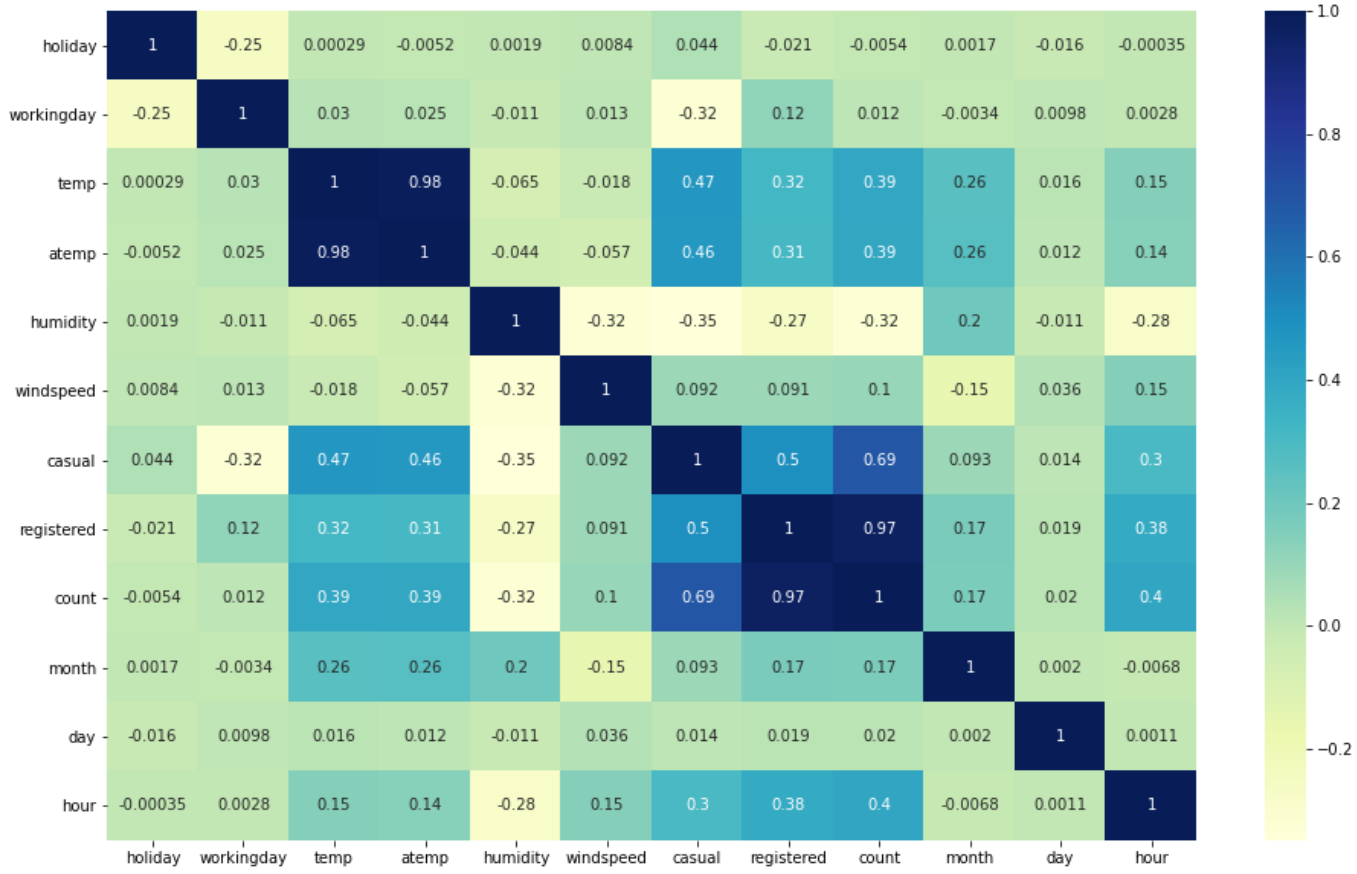
Column	Data Type
Date time	Object
Season	Object
Holiday	Int64
Working day	Int64
Weather	Object
Temp	Float64
Atemp	Float64
Humidity	Int64
Wind speed	Float64
Causal	Int64
Registered	Int64
Count	Int64

Month	**Int64**
Day	**Int64**
Hour	**Int64**

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



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



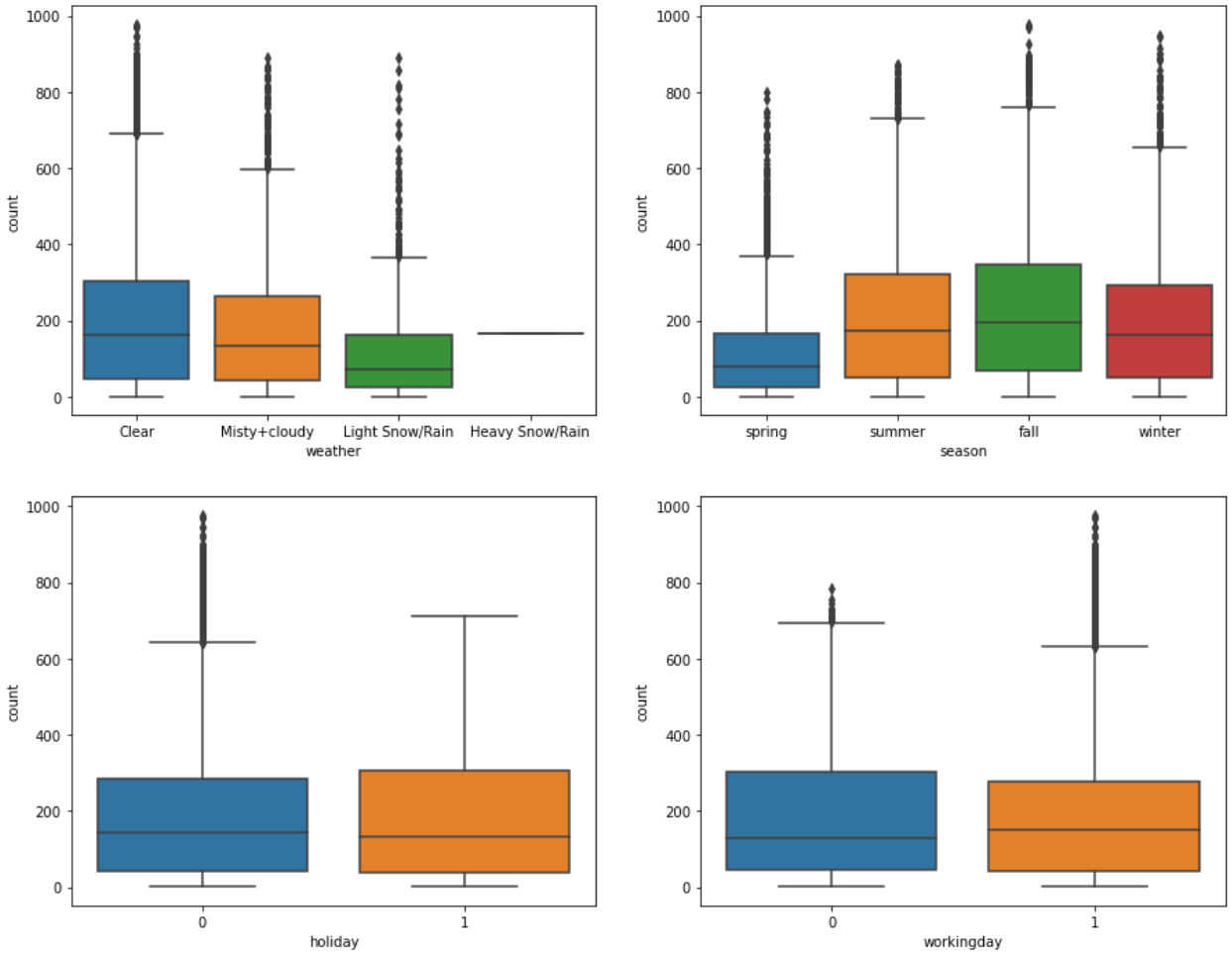
From the pair plot and heat map we deduce that atemp and temp are highly correlated to each other.

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**

	holiday	workingday	atemp	humidity	windspeed	casual	registered	count	month	day	hour
count	10342.0	10342.000000	10342.000000	10342.000000	10342.000000	10342.000000	10342.000000	10342.000000	10342.000000	10342.000000	10342.000000
mean	0.0	0.696770	23.580671	62.263682	12.582777	34.919938	148.455715	183.375653	6.526784	9.987720	11.462483
std	0.0	0.459676	8.419908	19.035799	7.843775	49.200220	136.884586	166.844823	3.445433	5.489213	6.939033
min	0.0	0.000000	0.760000	8.000000	0.000000	0.000000	0.000000	1.000000	1.000000	1.000000	0.000000
25%	0.0	0.000000	16.665000	47.000000	7.001500	4.000000	35.000000	41.000000	4.000000	5.000000	5.000000
50%	0.0	1.000000	24.240000	62.000000	11.001400	16.000000	116.000000	142.000000	6.000000	10.000000	11.000000
75%	0.0	1.000000	31.060000	78.000000	16.997900	47.000000	218.000000	276.000000	10.000000	15.000000	17.000000
max	0.0	1.000000	45.455000	100.000000	36.997400	367.000000	712.000000	734.000000	12.000000	19.000000	23.000000

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

	datetime	season	holiday	workingday	weather	atemp	humidity	windspeed	casual	registered	...	day	hour	fall	spring	summer	winter	Clear	Heavy Snow/Rain	Light Snow/Rain	Misty+cloudy
1755	2011-04-19 01:00:00	summer	0	1	Clear	22.725	63	15.0013	5	2	...	19	1	0	0	1	0	1	0	0	0
8633	2012-08-02 02:00:00	fall	0	1	Clear	30.305	83	6.0032	3	8	...	2	2	1	0	0	0	1	0	0	0
2399	2011-06-07 21:00:00	summer	0	1	Clear	34.090	70	8.9981	31	187	...	7	21	0	0	1	0	1	0	0	0
9957	2012-10-19 06:00:00	winter	0	1	Light Snow/Rain	25.760	94	11.0014	5	126	...	19	6	0	0	0	1	0	0	1	0
5182	2011-12-10 00:00:00	winter	0	0	Misty+cloudy	13.635	81	7.0015	11	66	...	10	0	0	0	0	1	0	0	0	1
2072	2011-05-13 06:00:00	summer	0	1	Light Snow/Rain	24.240	88	8.9981	6	76	...	13	6	0	0	1	0	0	0	1	0
1057	2011-03-08 14:00:00	spring	0	1	Misty+cloudy	19.695	20	0.0000	16	56	...	8	14	0	1	0	0	0	0	0	1
2624	2011-06-17 06:00:00	summer	0	1	Clear	26.515	94	0.0000	8	89	...	17	6	0	0	1	0	1	0	0	0
3504	2011-08-15 22:00:00	fall	0	1	Clear	31.060	69	16.9979	22	76	...	15	22	1	0	0	0	1	0	0	0
2133	2011-05-15 19:00:00	summer	0	0	Clear	26.515	83	7.0015	78	153	...	15	19	0	0	1	0	1	0	0	0

- **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%**

Baseline Model

Our baseline is simply a model that predicts the mean of the target variable always (outputs the average of the target variable).

a. **Evaluation metrics for training data**
    i. Root mean squared error -> 0.229049
    ii. R squared value -> -0.000069
b. **Evaluation metrics for validation data**
    i. Root mean squared error -> 0.223054
    ii. R squared value -> -0.001171

The goal is to train different models to beat the evaluation of the baseline model.

Training the models

Before tuning

2. Linear Regression

a. Evaluation metrics for training data

i. Root mean squared error -> 0.185939

ii. R squared value -> 0.340961

b. Evaluation metrics for validation data

i. Root mean squared error -> 0.179420

ii. R squared value -> 0.352218

The values shows that there is under fitting in the model

3. Decision Trees

a. Evaluation metrics for training data

i. Root mean squared error -> 0

ii. R squared value -> **1**

b. Evaluation metrics for validation data

i. Root mean squared error -> 0.111459

ii. R squared value -> 0.750011

Overfitting

4. Ensemble Learning (Bagging->Random Forest)

a. Evaluation metrics for training data

i. Root mean squared error -> 0.032052

ii. R squared value -> 0.980416

b. Evaluation metrics for validation data

i. Root mean squared error -> 0.082713

ii. R squared value -> 0.862332

Overfitting

5. **Ensemble Learning (Boosting->AdaBoost)**
    a. **Evaluation metrics for training data**
        i. Root mean squared error -> 0.160657
        ii. R squared value is -> 0.507994
    b. Evaluation metrics for validation data
        i. Root mean squared error -> 0.162524
        ii. R squared value is -> 0.468472
6. **Ensemble Learning (Boosting->XGBoost)**
    a. **Evaluation metrics for training data**
        i. Root mean squared error -> 0.043910
        ii. R squared value is -> 0.963246
    b. Evaluation metrics for validation data
        i. Root mean squared error -> 0.078710
        ii. R squared value is -> 0.875334

7. **Support Vector Machine (SVM)**
    a. **Evaluation metrics for training data**
        i. Root mean squared error -> 0.160659
        ii. R squared value is -> 0.507982
    b. Evaluation metrics for validation data
        i. Root mean squared error -> 0.156724
        ii. 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 -> 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(Lasso & Ridge)**
    a. **Hyper parameters tuning: Alpha and solver type in ridge LR**
        i. **Alpha = 1 and solver type=”svd”**
            1. **Evaluation metrics for training data**
                a. Root mean squared error -> 0.185941
                b. R squared value -> 0.340942
            2. **Evaluation metrics for validation data**
                a. Root mean squared error -> 0.179405
                b. R squared value -> 0.352320
    b. **Hyper parameters tuning: Alpha in Lasso LR**
        i. **Alpha = 0.0001**
            1. **Evaluation metrics for training data**
                a. Root mean squared error -> 0.185959
                b. R squared value -> 0.340819
            2. **Evaluation metrics for validation data**
                a. Root mean squared error -> 0.179415
                b. 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

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

2. **Evaluation metrics for validation data**
    a. Root mean squared error -> 0.105999
    b. R squared value -> 0.773961

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

Max depth -> decides the height of the tree

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

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

4. **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.

a. **hyper parameter tuning: loss & learning rate & N estimator**
    i. **N estimators=10 & learning rate=1 & loss="exponential"**
        1. **Evaluation metrics for training data**
            a. Root mean squared error -> 0.153477
            b. R squared value -> 0.550986

2. **Evaluation metrics for validation data**
    a. Root mean squared error -> 0.151833
    b. R squared value -> 0.536103

5. 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.

a. hyper parameter tuning: loss & learning rate & N estimator

i. N estimators=20& learning rate=1 & max depth=6 & alpha=5

1. **Evaluation metrics for training data**
    a. Root mean squared error -> 0.086703
    b. R squared value -> 0.856700
2. **Evaluation metrics for validation data**
    a. Root mean squared error -> 0.856700
    b. R squared value -> 0.828530

6. 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'.

a. hyper parameter tuning: Kernel & C & Gamma

i. Kernel="rbf" & C=1 & Gamma=0.1

1. **Evaluation metrics for training data**
    a. Root mean squared error -> 0.088564

b. R squared value -> 0.850484

2. **Evaluation metrics for validation data**
   a. Root mean squared error -> 0.099394
   b. 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.