Bike Hiring

Week – 1 Report (Exploratory Data Analysis)

Bike Hiring

Week - 1 (Exploratory Data Analysis)

In [1]:
```
# Importing required libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

In [3]:
```
# Importing train.csv from S3 bucket bike-hiring
dataframe_bike =  pd.read_csv('s3://bike-hiring/train.csv')
dataframe_bike.head()
```

Out[3]:

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

Connecting the jupyter notebook to train.csv in S3 bucket named bike-hiring

In [5]:
```
dataframe_bike.shape
```

Out[5]: (10886, 12)

There are 10886 rows and 12 columns train.csv

In [6]:
```
dataframe_bike.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10886 entries, 0 to 10885
Data columns (total 12 columns):
 #   Column      Non-Null Count  Dtype
---  ------      --------------  -----
 0   datetime    10886 non-null  object
 1   season      10886 non-null  int64
 2   holiday     10886 non-null  int64
 3   workingday  10886 non-null  int64
 4   weather     10886 non-null  int64
 5   temp        10886 non-null  float64
 6   atemp       10886 non-null  float64
 7   humidity    10886 non-null  int64
 8   windspeed   10886 non-null  float64
 9   casual      10886 non-null  int64
 10  registered  10886 non-null  int64
 11  count       10886 non-null  int64
dtypes: float64(3), int64(8), object(1)
memory usage: 1020.7+ KB
```

There are float datatype (3 columns), int datatype (8 columns) and object datatype (1 column). The column ‘datetime’ is an object datatype. We need to convert it to float datatype in order to perform mathematical operations on it.

In [7]:
```
# Creating columns 'date' and 'time'

from datetime import datetime

time=[]
month=[]

for i in dataframe_bike['datetime']:
    date = datetime.strptime(i,"%Y-%m-%d %H:%M:%S")
    time.append(date.hour)
    month.append(date.month)

dataframe_bike['time'] = pd.DataFrame(time)
dataframe_bike['time'] = dataframe_bike['time'].astype(float)

dataframe_bike['month'] = pd.DataFrame(month)
dataframe_bike['month'] = dataframe_bike['month'].astype(float)
```

In [8]:
```
# Dropping datetime column
dataframe_bike.drop('datetime',axis=1,inplace=True)
```

In [9]:
```
dataframe_bike.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10886 entries, 0 to 10885
Data columns (total 13 columns):
 #   Column      Non-Null Count  Dtype
---  ------      --------------  -----
 0   season      10886 non-null  int64
 1   holiday     10886 non-null  int64
 2   workingday  10886 non-null  int64
 3   weather     10886 non-null  int64
 4   temp        10886 non-null  float64
 5   atemp       10886 non-null  float64
 6   humidity    10886 non-null  int64
 7   windspeed   10886 non-null  float64
 8   casual      10886 non-null  int64
 9   registered  10886 non-null  int64
 10  count       10886 non-null  int64
 11  time        10886 non-null  float64
 12  month       10886 non-null  float64
dtypes: float64(5), int64(8)
memory usage: 1.1 MB
```

'datetime' column was split into 2 float type columns of 'date' and 'type'. Now there are 5 float datatype columns and 8 int datatype columns.

In [11]:
```
dataframe_bike.describe()
```

Out[11]:

	season	holiday	workingday	weather	temp	atemp	humidity	windspeed	casual	registered	co
count	10886.000000	10886.000000	10886.000000	10886.000000	10886.00000	10886.000000	10886.000000	10886.000000	10886.000000	10886.000000	10886.000
mean	2.506614	0.028569	0.680875	1.418427	20.23086	23.655084	61.886460	12.799395	36.021955	155.552177	191.574
std	1.116174	0.166599	0.466159	0.633839	7.79159	8.474601	19.245033	8.164537	49.960477	151.039033	181.144
min	1.000000	0.000000	0.000000	1.000000	0.82000	0.760000	0.000000	0.000000	0.000000	0.000000	1.000
25%	2.000000	0.000000	0.000000	1.000000	13.94000	16.665000	47.000000	7.001500	4.000000	36.000000	42.000
50%	3.000000	0.000000	1.000000	1.000000	20.50000	24.240000	62.000000	12.998000	17.000000	118.000000	145.000
75%	4.000000	0.000000	1.000000	2.000000	26.24000	31.060000	77.000000	16.997900	49.000000	222.000000	284.000
max	4.000000	1.000000	1.000000	4.000000	41.00000	45.455000	100.000000	56.996900	367.000000	886.000000	977.000

In [12]:
```
# Checking for null values
dataframe_bike.isnull().sum()
```

Out[12]:
```
season        0
holiday       0
workingday    0
weather       0
temp          0
atemp         0
humidity      0
windspeed     0
casual        0
registered    0
count         0
time          0
month         0
dtype: int64
```

No null values

train.csv does not contain any null values.

In [13]:
```
# Checking for duplicates
duplicates = dataframe_bike.duplicated()
print(duplicates.sum())
```

```
4
```

In [14]:
```
# Removing duplicates
dataframe_bike.drop_duplicates(inplace=True)
```

In [16]:
```
duplicates = dataframe_bike.duplicated()
print(duplicates.sum())
```

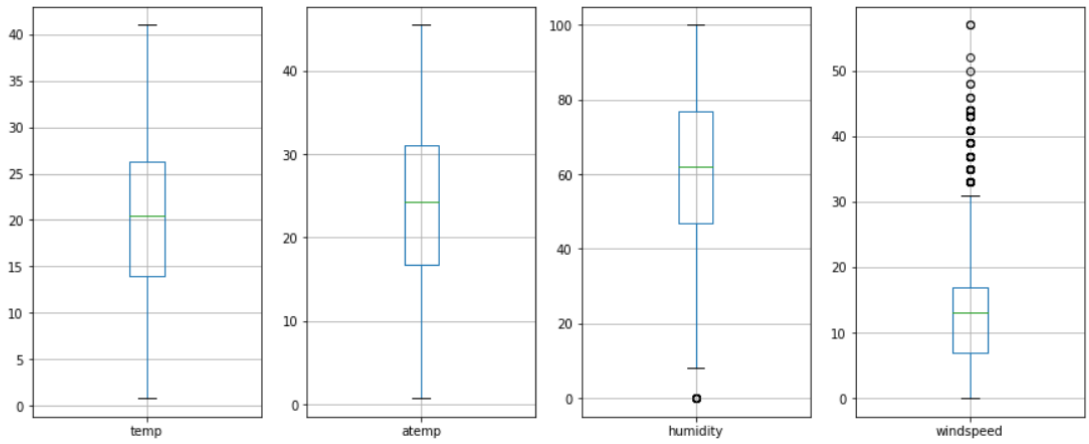
```
0
```

Duplicates removed

train.csv has 4 duplicate rows. These need to be removed so that we can get accurate results while analyzing the data.

In [17]:
```
# Checking for outliers using boxplot

plt.figure(figsize=(15,6))
plt.subplot(141)
dataframe_bike.boxplot(column=["temp"])
plt.subplot(142)
dataframe_bike.boxplot(column=["atemp"])
plt.subplot(143)
dataframe_bike.boxplot(column=["humidity"])
plt.subplot(144)
dataframe_bike.boxplot(column=["windspeed"])
plt.show
```

Out[17]: <function matplotlib.pyplot.show(close=None, block=None)>



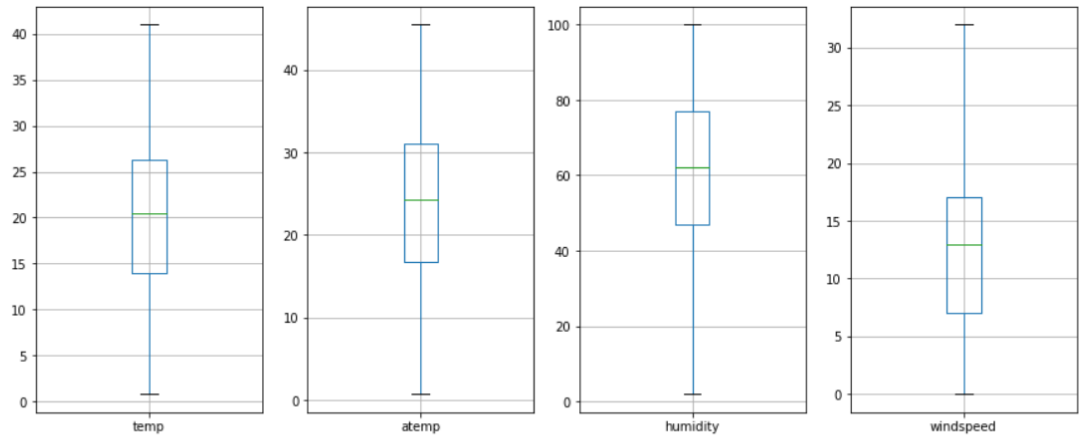
Outliers is present in the columns ‘humidity’ and ‘windspeed’.

In [18]:
```
def remove_outliers(col):
    sorted(col)
    Q1,Q3 = col.quantile([0.25,0.75])
    IQR = Q3-Q1
    lower_range = Q1-(1.5*IQR)
    upper_range = Q3+(1.5*IQR)

    return lower_range,upper_range
```

In [20]:
```
# Removing outliers for humidity and windspeed

low_humidity,up_humidity = remove_outliers(dataframe_bike['humidity'])

dataframe_bike['humidity'] = np.where(dataframe_bike['humidity']>up_humidity,up_humidity,dataframe_bike['humidity'])
dataframe_bike['humidity'] = np.where(dataframe_bike['humidity']<low_humidity,low_humidity,dataframe_bike['humidity'])

low_windspeed,up_windspeed = remove_outliers(dataframe_bike['windspeed'])

dataframe_bike['windspeed'] = np.where(dataframe_bike['windspeed']>up_windspeed,up_windspeed,dataframe_bike['windspeed'])
dataframe_bike['windspeed'] = np.where(dataframe_bike['windspeed']<low_windspeed,low_windspeed,dataframe_bike['windspeed'])
```

```
plt.figure(figsize=(15,6))
plt.subplot(141)
dataframe_bike.boxplot(column=["temp"])
plt.subplot(142)
dataframe_bike.boxplot(column=["atemp"])
plt.subplot(143)
dataframe_bike.boxplot(column=["humidity"])
plt.subplot(144)
dataframe_bike.boxplot(column=["windspeed"])
plt.show
```

Out[21]: <function matplotlib.pyplot.show(close=None, block=None)>

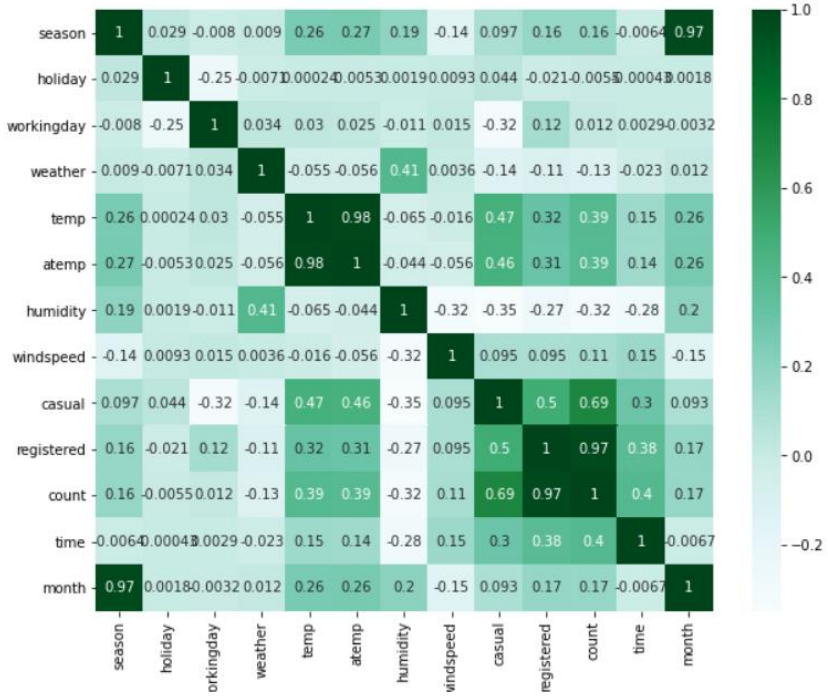


Outliers removed

Heatmap plotted for correlation

In [23]:
```
#Plotting Heatmap for correlation
plt.figure(figsize=(10,8))
sns.heatmap(dataframe_bike.corr(),cmap='BuGn',annot=True)
```

Out[23]: <AxesSubplot:>



From the above plotted heatmap, it is found that ‘season’ and ‘month’ column, ‘registered’ and ‘count’ columns and ‘temp’ and ‘atemp’ columns are highly correlated with a correlation of 0.97,0.97 and 0.98 respectively.

Week – 2 Report

Feature Selection

The techniques for feature selection in machine learning can be broadly classified into the following categories:

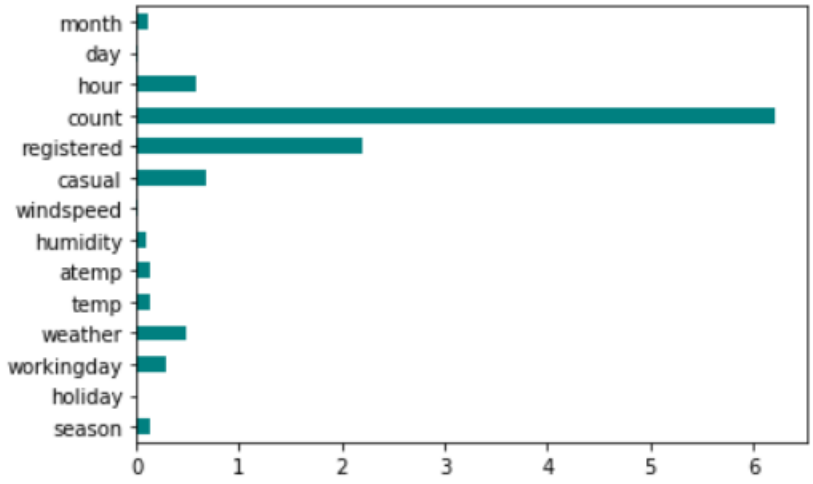
Supervised Techniques: These techniques can be used for labeled data, and are used to identify the relevant features for increasing the efficiency of supervised models like classification and regression.

Unsupervised Techniques: These techniques can be used for unlabeled data.

1. Filter Methods

a) Information Gain

Information gain calculates the reduction in entropy from the transformation of a dataset. It can be used for feature selection by evaluating the Information gain of each variable in the context of the target variable.



b) Chi – square test

The Chi-square test is used for categorical features in a dataset. We calculate Chi-square between each feature and the target and select the desired number of features with the best Chi-square scores. In order to correctly apply the chi-squared in order to test the relation between various features in the dataset and the target variable, the following conditions have to be met: the variables have to be *categorical*, sampled *independently* and values should have an *expected frequency greater than 5*.

	Parameter	Value
0	season	6.100552e+02
1	holiday	6.813270e+02
2	workingday	3.200576e+02
3	weather	2.342817e+02
4	temp	8.172345e+03
5	atemp	8.150863e+03
6	humidity	1.229426e+04
7	windspeed	5.057659e+03
8	casual	4.624429e+05
9	registered	1.528535e+06
10	count	1.864079e+06
11	hour	1.811747e+04
12	day	1.252994e+03
13	month	2.188255e+03

```
print(featureScores.nlargest(5,'Value'))
```

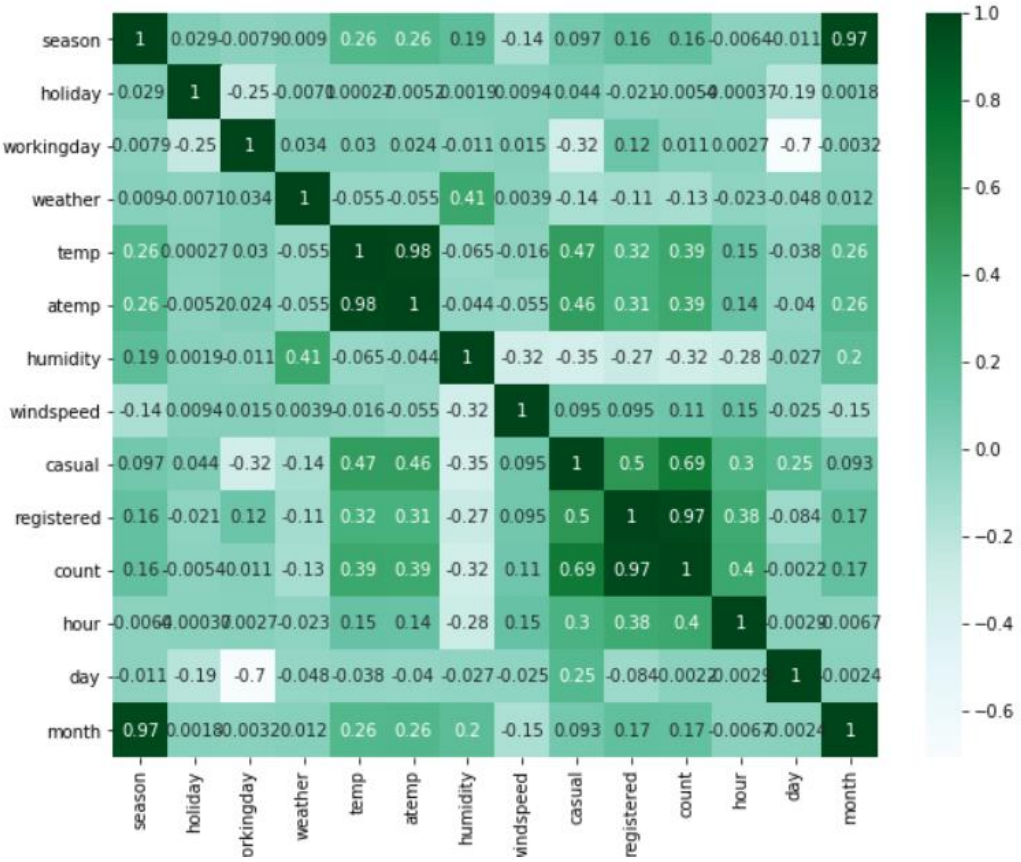
```
     Parameter         Value
10       count  1.864079e+06
9    registered  1.528535e+06
8        casual  4.624429e+05
11         hour  1.811747e+04
6      humidity  1.229426e+04
```

c) Fisher's Score

The algorithm which we will use returns the ranks of the variables based on the fisher's score in descending order. We can then select the variables as per the case.

d) Correlation Coefficient

Correlation is a measure of the linear relationship of 2 or more variables. Through correlation, we can predict one variable from the other. The logic behind using correlation for feature selection is that the good variables are highly correlated with the target. Furthermore, variables should be correlated with the target but should be uncorrelated among themselves.


season 1 0.029 -0.007 9 0.009 0.26 0.26 0.19 -0.14 0.097 0.16 0.16 -0.006 4 0.011 0.97
holiday 0.029 1 -0.25 -0.007 1 0.000 27 0.005 2 0.001 9 0.009 4 0.044 -0.021 0.005 4 0.000 37 -0.19 0.0018
workingday -0.007 9 -0.25 1 0.034 0.03 0.024 -0.011 0.015 -0.32 0.12 0.011 0.0027 -0.7 -0.0032
weather 0.009 -0.007 1 0.034 1 -0.055 -0.055 0.41 0.003 9 -0.14 -0.11 -0.13 -0.023 -0.048 0.012
temp 0.26 0.000 27 0.03 -0.055 1 0.98 -0.065 -0.016 0.47 0.32 0.39 0.15 -0.038 0.26
atemp 0.26 -0.005 2 0.024 -0.055 0.98 1 -0.044 -0.055 0.46 0.31 0.39 0.14 -0.04 0.26
humidity 0.19 0.001 9 -0.011 0.41 -0.065 -0.044 1 -0.32 -0.35 -0.27 -0.32 -0.28 -0.027 0.2
windspeed -0.14 0.009 4 0.015 0.003 9 -0.016 -0.055 -0.32 1 0.095 0.095 0.11 0.15 -0.025 -0.15
casual 0.097 0.044 -0.32 -0.14 0.47 0.46 -0.35 0.095 1 0.5 0.69 0.3 0.25 0.093
registered 0.16 -0.021 0.12 -0.11 0.32 0.31 -0.27 0.095 0.5 1 0.97 0.38 -0.084 0.17
count 0.16 -0.005 4 0.011 -0.13 0.39 0.39 -0.32 0.11 0.69 0.97 1 0.4 -0.002 2 0.17
hour -0.006 4 0.000 37 0.002 7 -0.023 0.15 0.14 -0.28 0.15 0.3 0.38 0.4 1 -0.002 9 0.006 7
day -0.011 -0.19 -0.7 -0.048 -0.038 -0.04 -0.027 -0.025 0.25 -0.084 0.002 2 0.002 9 1 0.002 4
month 0.97 0.001 8 0.003 2 0.012 0.26 0.26 0.2 -0.15 0.093 0.17 0.17 -0.006 7 0.002 4 1
season
holiday
orkingday
weather
temp
atemp
humidity
windspeed
casual
registered
count
hour
day
month
1.0
0.8
0.6
0.4
0.2
0.0
−0.2
−0.4
−0.6

2. Wrapper Methods

a) Forward Selection Feature

This is an iterative method wherein we start with the best performing variable against the target. Next, we select another variable that gives the best performance in combination with the first selected variable. This process continues until the preset criterion is achieved.

```
sfs1 = sfs1.fit(X, y)
```

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurre
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    0.0s remainin
[Parallel(n_jobs=1)]: Done  13 out of  13 | elapsed:    0.2s finished

[2021-06-27 12:59:12] Features: 1/4 -- score: -1958.7561616488529[Par
oncurrent workers.
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    0.0s remainin
[Parallel(n_jobs=1)]: Done  12 out of  12 | elapsed:    0.3s finished

[2021-06-27 12:59:13] Features: 2/4 -- score: -1624.5759983211933[Par
oncurrent workers.
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    0.0s remainin
[Parallel(n_jobs=1)]: Done  11 out of  11 | elapsed:    0.2s finished

[2021-06-27 12:59:13] Features: 3/4 -- score: -1330.6158702209689[Par
oncurrent workers.
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    0.0s remainin
[Parallel(n_jobs=1)]: Done  10 out of  10 | elapsed:    0.1s finished

[2021-06-27 12:59:13] Features: 4/4 -- score: -1224.6071198596778
```

```
feat_names = list(sfs1.k_feature_names_)
print(feat_names)
# these are the parameters compared
```

```
['workingday', 'temp', 'humidity', 'registered']
```

b) Backward Selection Feature

Here, we start with all the features available and build a model. Next, we the variable from the model which gives the best evaluation measure value. This process is continued until the preset criterion is achieved.

```
sfs2 = sfs2.fit(X, y)
```

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent
[Parallel(n_jobs=1)]: Done  13 out of  13 | elapsed:    0.5s finished
Features: 12/4[Parallel(n_jobs=1)]: Using backend SequentialBackend wit
[Parallel(n_jobs=1)]: Done  12 out of  12 | elapsed:    0.4s finished
Features: 11/4[Parallel(n_jobs=1)]: Using backend SequentialBackend wit
[Parallel(n_jobs=1)]: Done  11 out of  11 | elapsed:    0.3s finished
Features: 10/4[Parallel(n_jobs=1)]: Using backend SequentialBackend wit
[Parallel(n_jobs=1)]: Done  10 out of  10 | elapsed:    0.3s finished
Features: 9/4[Parallel(n_jobs=1)]: Using backend SequentialBackend with
[Parallel(n_jobs=1)]: Done   9 out of   9 | elapsed:    0.3s finished
Features: 8/4[Parallel(n_jobs=1)]: Using backend SequentialBackend with
[Parallel(n_jobs=1)]: Done   8 out of   8 | elapsed:    0.1s finished
Features: 7/4[Parallel(n_jobs=1)]: Using backend SequentialBackend with
[Parallel(n_jobs=1)]: Done   7 out of   7 | elapsed:    0.1s finished
Features: 6/4[Parallel(n_jobs=1)]: Using backend SequentialBackend with
[Parallel(n_jobs=1)]: Done   6 out of   6 | elapsed:    0.0s finished
Features: 5/4[Parallel(n_jobs=1)]: Using backend SequentialBackend with
[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:    0.0s finished
Features: 4/4
```

```
feat_names = list(sfs2.k_feature_names_)
print(feat_names)
```

```
['workingday', 'atemp', 'humidity', 'registered']
```

Week - 3

Things to do:

1. Decide what kind of problem it is? (Supervised/Unsupervised, Discrete/Continuous/ Categorical, etc.)
2. Explore Algorithms such as Random Forest, Decision Tree, XGBBoost, etc. (Relevant to this problem)
3. Create extra features or remove .
4. Do research on evaluation metrics for the prediction given by the model.

1: Since our dataset is labelled we would require supervised learning techniques to train our model. Also our problem type is Regression (Prediction) with discrete output.

Supervised learning : Techniques which is used for labeled data, and is used to identify the relevant features for increasing the efficiency of supervised models like classification and regression.

2: Algorithms that can be applied are:

- Linear Regression
- Polynomial Regression
- Exponential Regression
- Logistic Regression
- Logarithmic Regression
- Decision Trees
- Random Forest
- XGBoost

3: Extra features that could be added are:

- peak (0/1 if that hour is a peak hour or not)
- weekend (0/1 if that day is a weekend or not)
- Hot encode season -> is_season_1, is_season_2, is_season_3 & is_season_4

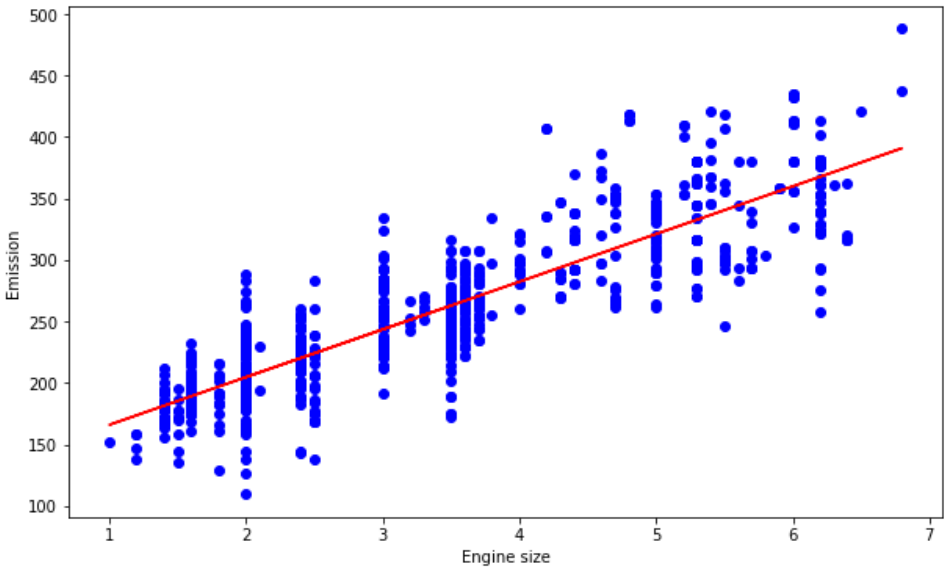
4: Types of different evaluation matrix:

- Classification Accuracy
- Area Under Curve
- F1 Score
- Logarithmic Loss
- Mean Absolute Error
- Mean Squared Error
- Confusion Matrix

Linear Regression

Linear Regression is one of the most fundamental algorithms used to model relationships between a dependent variable and one or more independent variables. In simpler terms, it involves finding the 'line of best fit' that represents two or more variables.

The line of best fit is found by minimizing the squared distances between the points and the line of best fit — this is known as minimizing the sum of squared residuals. A residual is simply equal to the predicted value minus the actual value.



Parameters

- fit_interceptbool, default=True
  Whether to calculate the intercept for this model. If set to False, no intercept will be used in calculations (i.e. data is expected to be centered).

- normalizebool, default=False
  This parameter is ignored when fit_intercept is set to False. If True, the regressors X will be normalized before regression by subtracting the mean and dividing by the l2-norm. If you wish to standardize, please use StandardScaler before calling fit on an estimator with normalize=False.

- copy_Xbool, default=True
  If True, X will be copied; else, it may be overwritten.

- n_jobsint, default=None
  The number of jobs to use for the computation. This will only provide speedup for n_targets > 1 and sufficient large problems. None means 1 unless in a joblib.parallel_backend context. -1 means using all processors. See Glossary for more details.

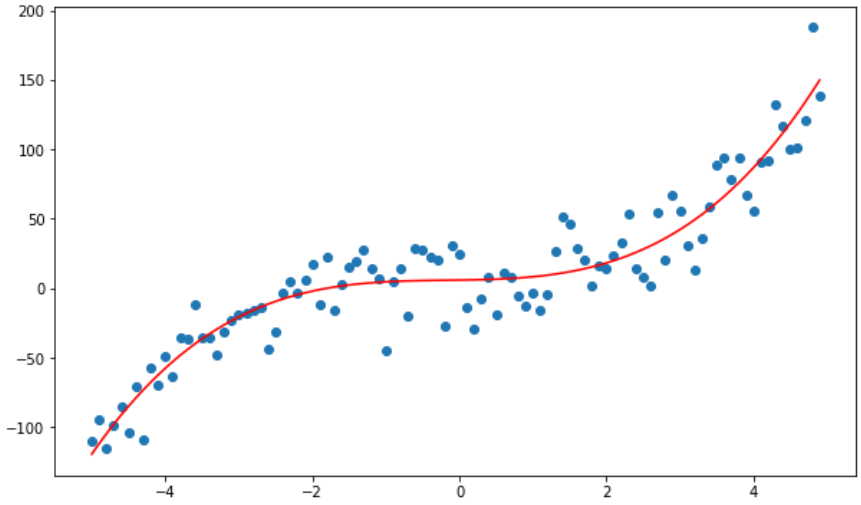
- positivebool, default=False
  When set to True, forces the coefficients to be positive. This option is only supported for dense arrays.

Some use cases:

- Sales of a product; pricing, performance, and risk parameters
- Generating insights on consumer behavior, profitability, and other business factors
- Evaluation of trends; making estimates, and forecasts

Polynomial Regression

Sometimes we have data that does not merely follow a linear trend. We sometimes have data that follows a polynomial trend. Therefore, we are going to use polynomial regression.

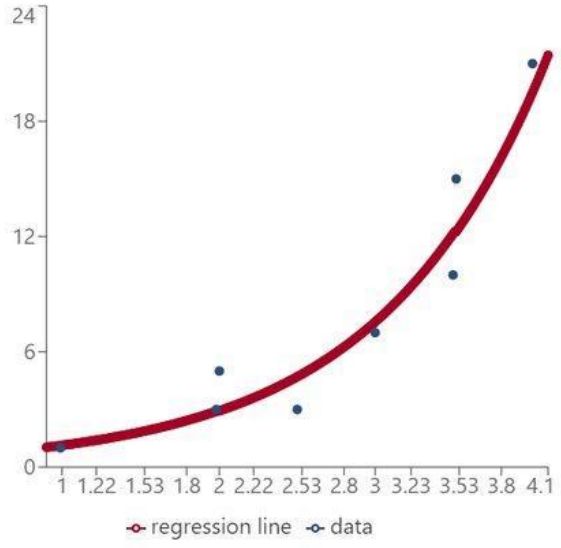


Exponential Regression

An exponential regression is the process of finding the equation of the exponential function that fits best for a set of data. As a result, we get an equation of the form

Y = ab^x where a ≠ 0

The relative predictive power of an exponential model is denoted by R^2. The value of R^2 varies between 0 and 1. The closer the value is to 1 , the more accurate the model is.



Logistic Regression

Logistic regression is similar to linear regression but is used to model the probability of a discrete number of outcomes, typically two.

First, you calculate a score using an equation similar to the equation for the line of best fit for linear regression.

$$y = ax + b$$
$$y = \Theta^T X$$

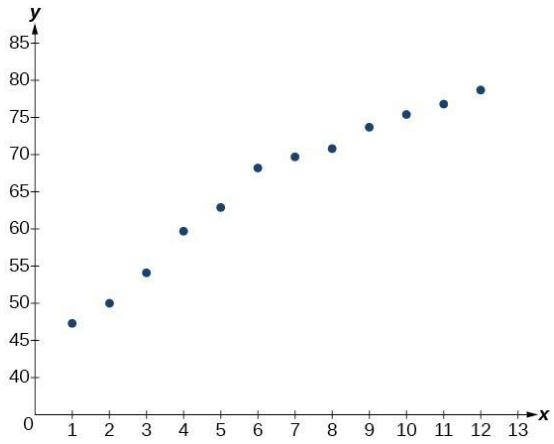
The extra step is feeding the score that you previously calculated in the sigmoid function below so that you get a probability in return. This probability can then be converted to a binary output, either 1 or 0.

$$S(x) = \frac{1}{1+e^{-x}}$$

Logarithmic Regression

Logarithmic regression is used to model situations where growth or decay accelerates rapidly at first and then slows over time.
It is like the normal Logarithmic function which increases or decreases drastically at first and then normalizes.

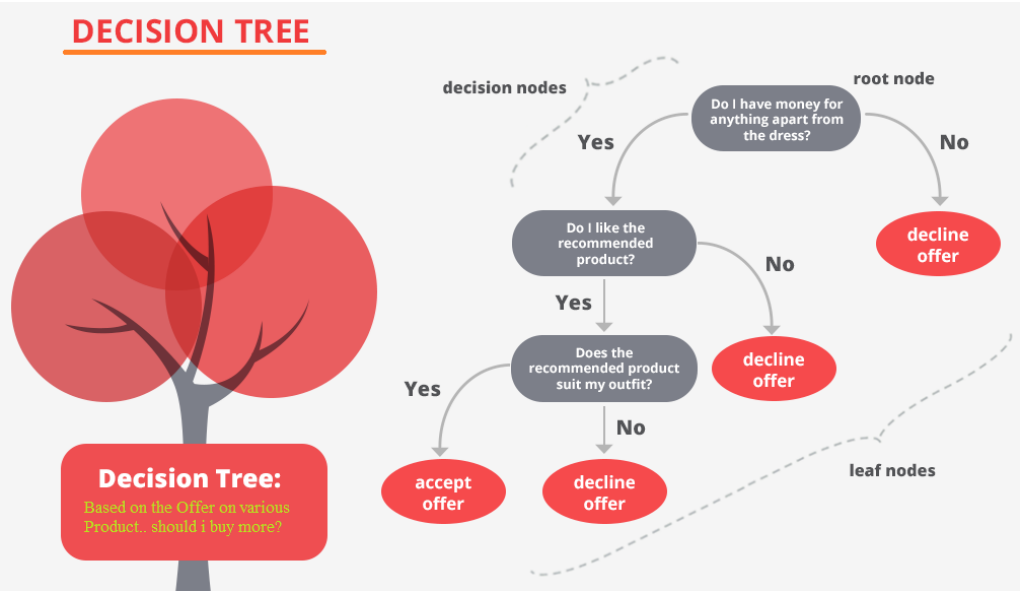
y=a+bln(x)



Decision Trees

Decision Trees are supervised learning method used for classification and regression. The goal is to create a model that predicts the value of a target variable by learning simple decision rules inferred from the data features. A tree can be seen as a piecewise constant approximation.

In Decision Trees, for predicting a class label for a record we start from the root of the tree. We compare the values of the root attribute with the record's attribute. On the basis of comparison, we follow the branch corresponding to that value and jump to the next node.



Some advantages of decision trees are:

- Requires little data preparation. Other techniques often require data normalization, dummy variables need to be created and blank values to be removed.
- The cost of using the tree (i.e., predicting data) is logarithmic in the number of data points used to train the tree.
- Able to handle both numerical and categorical data.
- Able to handle multi-output problems.

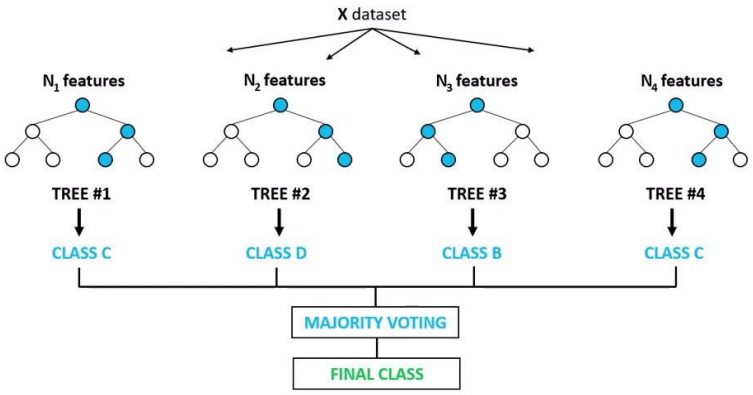
Some use cases –

- Building knowledge management platforms for customer service that improve first call resolution, average handling time, and customer satisfaction rates
- In finance, forecasting future outcomes and assigning probabilities to those outcomes
- Product planning; for example, Gerber Products, Inc. used decision trees to decide whether to continue planning PVC for manufacturing toys or not
- General business decision-making

Random Forest

Random forests are an ensemble learning technique that builds off of decision trees. Random forests involve creating multiple decision trees using bootstrapped datasets of the original data and randomly selecting a subset of variables at each step of the decision tree. The model then selects the mode of all of the predictions of each decision tree (bagging). What's the point of this? By relying on a "majority wins" model, it reduces the risk of error from an individual tree.

Random Forest Classifier



Use Cases –

- **Banking Sector:** With the help of a random forest algorithm in machine learning, we can easily determine whether the customer is fraud or loyal. A system uses a set of a random algorithm which identifies the fraud transactions by a series of the pattern.
- **Medicines:** With the help of Random Forest algorithm, it has become easier to detect and predict the drug sensitivity of a medicine. Also, it helps to identify the patient's disease by analyzing the patient's medical record.
- **Stock Market:** When you want to know the behavior of the stock market, with the help of Random forest algorithm, the behavior of the stock market can be analyzed. Also, it can show the expected loss or profit which can be produced while purchasing a particular stock.
- **E-Commerce:** Using Random Forest, you can suggest the products which will be more likely for a customer. Using a certain pattern and following the product's interest of a customer, you can suggest similar products to your customers.

XG Boost

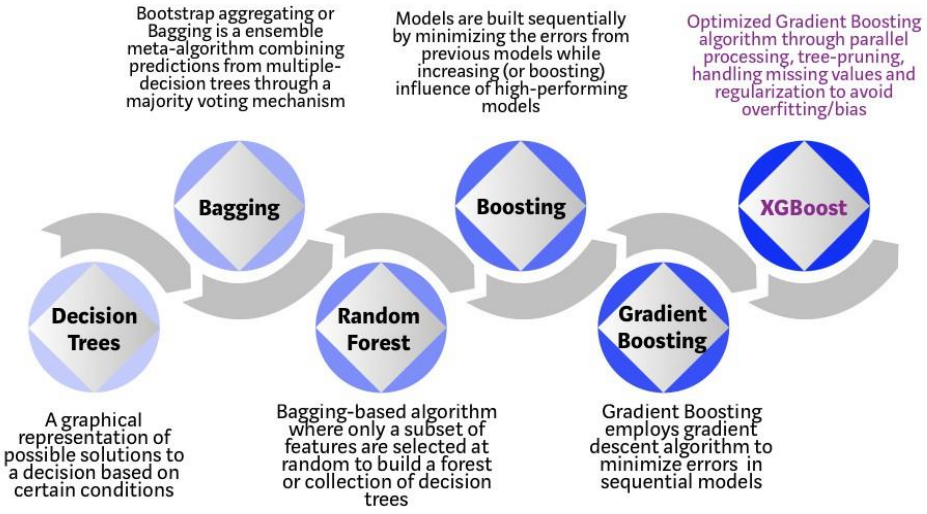
XGBoost is a decision-tree-based ensemble Machine Learning algorithm that uses a gradient boosting framework.

Boosting : N new training data sets are formed by random sampling with replacement from the original dataset, during which some observations may be repeated in each new training data set. The observations are weighted and therefore some of them may get selected in the new datasets more often.

Gradient Boosting*:* It is an additive and sequential model where trees are grown in sequential manner which converts weak learners into strong learners by adding weights to the weak learners and reduce weights of the strong learners. So each tree learns and boosts from the previous tree grown.

Use Cases :

1. Classification problems, especially those related **to** real-world business problems.
2. Problems in which the range or distribution of target values present in the training set can be expected **to** be similar **to** that of real-world testing data.
3. Situations in which there are many categorical variables.



Features of XGBoost:

1. **Handling sparse data**
2. **Block structure for parallel learning**
3. **Cache Awareness**
4. **Out-of-the-computing**

Evaluation Metrics

Confusion matrix

A Confusion matrix is an N x N matrix used for evaluating the performance of a classification model, where N is the number of target classes. The matrix compares the actual target values with those predicted by the machine learning model. This gives us a holistic view of how well our classification model is performing and what kinds of errors it is making.

For a binary classification problem, we would have a 2 x 2 matrix as shown below with 4 values:

		Actual	
		Positive	Negative
Predicted	Positive	True Positive	False Positive
	Negative	False Negative	True Negative

- The target variable has two values: Positive or Negative
- The columns represent the actual values of the target variable
- The rows represent the predicted values of the target variable

$$Sensitivity = \frac{TP}{TP + FN}$$ (Recall)

$$FNR = \frac{FN}{TP + FN}$$ (False Negative Rate)

Precision = TP / (TP+FP)

F1 Score

F1 Score is the Harmonic Mean between precision and recall. The range for F1 Score is [0, 1]. It tells you how precise your classifier is (how many instances it classifies correctly), as well as how robust it is (it does not miss a significant number of instances).The greater the F1 Score, the better is the performance of our model.

$$F1 = 2 * \frac{1}{\frac{1}{precision} + \frac{1}{recall}}$$

F1 Score tries to find the balance between precision and recall.

Precision : It is the number of correct positive results divided by the number of positive results predicted by the classifier.

Recall : It is the number of correct positive results divided by the number of all relevant samples (all samples that should have been identified as positive).

Logarithmic Loss

Logarithmic Loss or Log Loss, works by penalising the false classifications. It works well for multi-class classification. When working with Log Loss, the classifier must assign probability to each class for all the samples.
In general, minimising Log Loss gives greater accuracy for the classifier.
Suppose, there are N samples belonging to M classes, then the Log Loss is calculated as below :

$$LogarithmicLoss = \frac{-1}{N}\sum_{i=1}^{N}\sum_{j=1}^{M} y_{ij} * \log(p_{ij})$$

Classification Accuracy

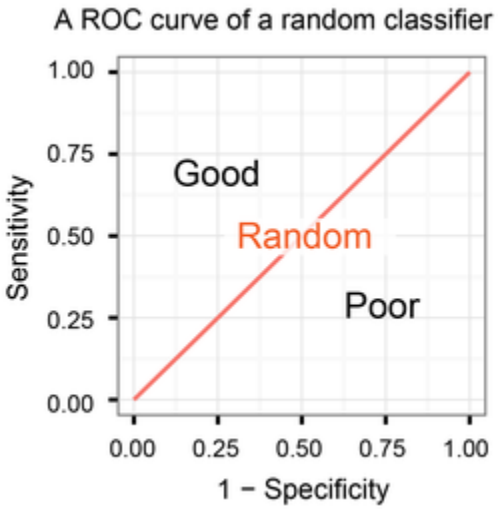
When we make predictions by classifying the observations, the result is either correct (True) or incorrect (False). The classification accuracy measures the percentage of the correct classifications with the formula below:

Accuracy = # of correct predictions / # of total predictions

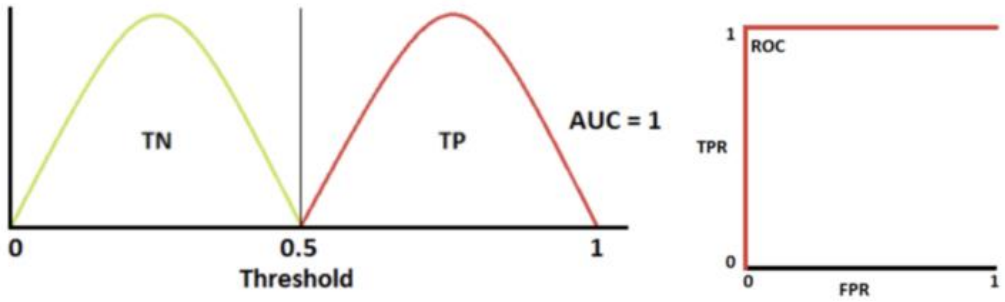
The Higher the Accuracy, more accurate the model.
Yet, accuracy doesn't tell the full story, especially for imbalanced datasets.

Area Under Curve

It is an area under the curve calculated in the ROC space. It is the metric we consider when we want to evaluate a model's performance when using the ROC curve, also called AUROC.



ROC : It is a graph showing the performance of a classification model at all classification thresholds.
The curve separates the space into two areas for good and poor performance levels.



The higher the AUC, the better the performance of the model at distinguishing between the positive and negative classes.

- When AUC = 1, then the classifier is able to perfectly distinguish between all the Positive and the Negative class points correctly. If, however, the AUC had been 0, then the classifier would be predicting all Negatives as Positives, and all Positives as Negatives.

- When 0.5<AUC<1, there is a high chance that the classifier will be able to distinguish the positive class values from the negative class values. This is so because the classifier is able to detect more numbers of True positives and True negatives than False negatives and False positives.
- When AUC=0.5, then the classifier is not able to distinguish between Positive and Negative class points. Meaning either the classifier is predicting random class or constant class for all the data points.

Mean Absolute Error Method

Mean Absolute Error is the average of the difference between the Original Values and the Predicted Values. It gives us the measure of how far the predictions were from the actual output. However, they don't gives us any idea of the direction of the error i.e. whether we are under predicting the data or over predicting the data. Mathematically, it is represented as :

$$MeanAbsoluteError = \frac{1}{N}\sum_{j=1}^{N}|y_j - \hat{y}_j|$$

Mean Squared Error Method

Mean Squared Error(MSE) is quite similar to Mean Absolute Error, the only difference being that MSE takes the average of the square of the difference between the original values and the predicted values. The advantage of MSE being that it is easier to compute the gradient, whereas Mean Absolute Error requires complicated linear programming tools to compute the gradient. As, we take square of the error, the effect of larger errors become more pronounced then smaller error, hence the model can now focus more on the larger errors.

$$MeanSquaredError = \frac{1}{N}\sum_{j=1}^{N}(y_j - \hat{y}_j)^2$$

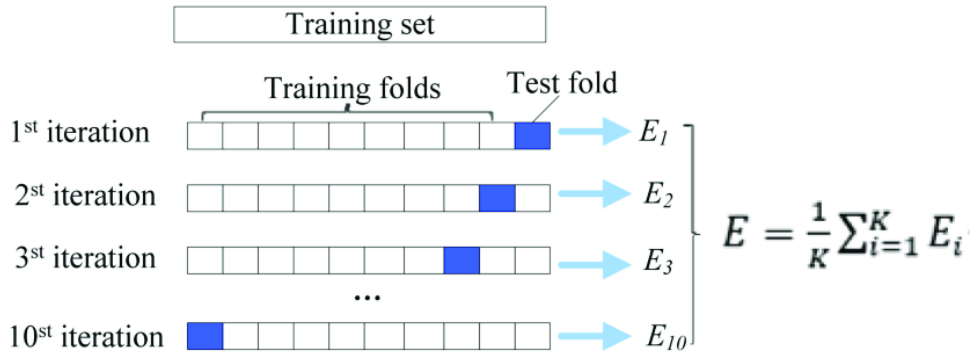
Week – 4

k-Fold Cross-Validation

Cross-validation is a resampling procedure used to evaluate machine learning models on a limited data sample.

The procedure has a single parameter called k that refers to the number of groups that a given data sample is to be split into. As such, the procedure is often called k-fold cross-validation. When a specific value for k is chosen, it may be used in place of k in the reference to the model, such as k=10 becoming 10-fold cross-validation.

It is a popular method because it is simple to understand and because it generally results in a less biased or less optimistic estimate of the model skill than other methods, such as a simple train/test split.



Root Mean Square Error (RMSE) –

It is the standard deviation of the residuals (prediction errors). Residuals are a measure of how far from the regression line data points are; RMSE is a measure of how spread out these residuals are. In other words, it tells you how concentrated the data is around the line of best fit.

$$RMSE = \sqrt{\frac{\sum_{i=1}^{N} (Predicted_i - Actual_i)^2}{N}}$$

Linear Regression –

Linear Regression is one of the most fundamental algorithms used to model relationships between a dependent variable and one or more independent variables. In simpler terms, it involves finding the 'line of best fit' that represents two or more variables.

The line of best fit is found by minimizing the squared distances between the points and the line of best fit — this is known as minimizing the sum of squared residuals. A residual is simply equal to the predicted value minus the actual value.

```
#Linear Regression

scores = []
for train_index, test_index in kf.split(x):
    x_train, x_test = x.iloc[train_index], x.iloc[test_index]
    y_train, y_test = y.iloc[train_index], y.iloc[test_index]

    model = LinearRegression(fit_intercept=True)
    scores.append(check(x_train, x_test, y_train, y_test, model))

print(min(scores))
results.append((str(model).split('(')[0], min(scores), list(x.columns), False))
```

```
32.53810500152216
```

Decision Trees Regression –

Decision tree builds regression or classification models in the form of a tree structure. For predicting a class label for a record we start from the root of the tree. We compare the values of the root attribute with the record's attribute. On the basis of comparison, we follow the branch corresponding to that value and jump to the next node. It breaks down a dataset into smaller and smaller subsets while at the same time an associated decision tree is incrementally developed. The final result is a tree with **decision nodes** and **leaf nodes**.

```
#Decision Tree Regression
from sklearn.tree import DecisionTreeRegressor
scores = []
for train_index, test_index in kf.split(x):
    x_train, x_test = x.iloc[train_index], x.iloc[test_index]
    y_train, y_test = y.iloc[train_index], y.iloc[test_index]

    model = DecisionTreeRegressor(criterion = 'mse',random_state=0)
    scores.append(check(x_train, x_test, y_train, y_test, model))

print(min(scores))
results.append((str(model).split('(')[0], min(scores), list(x.columns), False))
```

```
22.55191275586383
```

Adaptive Boost Regression –

AdaBoost is best used to boost the performance of decision trees on binary classification problems. The most suited and therefore most common algorithm used with AdaBoost are decision trees with one level. Because these trees are so short and only contain one decision for classification, they are often called decision stumps.

Each instance in the training dataset is weighted. The initial weight is set to:

weight(xi) = 1/n

Where xi is the i'th training instance and n is the number of training instances.

```
#Adaptive Boost Regressor

from sklearn.ensemble import AdaBoostRegressor
from sklearn.datasets import make_regression
scores = []
for train_index, test_index in kf.split(x):
    x_train, x_test = x.iloc[train_index], x.iloc[test_index]
    y_train, y_test = y.iloc[train_index], y.iloc[test_index]

    model = AdaBoostRegressor(n_estimators=100, loss = 'square',random_state=0)
    scores.append(check(x_train, x_test, y_train, y_test, model))

print(min(scores))
results.append((str(model).split('(')[0], min(scores), list(x.columns), False))
```

```
29.017032722075243
```

Gradient Boosting Regression –

Gradient boosting algorithm can be used to train models for both regression and classification problem. Gradient Boosting Regression algorithm is used to fit the model which predicts the continuous value.

Gradient boosting builds an additive mode by using multiple decision trees of fixed size as weak learners or weak predictive models. The parameter, n_estimators, decides the number of decision trees which will be used in the boosting stages. Gradient boosting differs from AdaBoost in the manner that decision stumps (one node & two leaves) are used in AdaBoost whereas decision trees of fixed size are used in Gradient Boosting.

```
#Gradient Boost Regressor

from sklearn.ensemble import GradientBoostingRegressor
scores = []
for train_index, test_index in kf.split(x):
    x_train, x_test = x.iloc[train_index], x.iloc[test_index]
    y_train, y_test = y.iloc[train_index], y.iloc[test_index]

    model = GradientBoostingRegressor()
    scores.append(check(x_train, x_test, y_train, y_test, model))

print(min(scores))
results.append((str(model).split('(')[0], min(scores), list(x.columns), False))
```

```
19.86790292973056
```

Extreme Gradient Boosting (XGBoosting) –

Extreme Gradient Boosting is an efficient open-source implementation of the gradient boosting algorithm. It is designed to be both computationally efficient (e.g. fast to execute) and highly effective, perhaps more effective than other open-source implementations. XGBoost and Gradient Boosting Machines are both ensemble tree methods that apply the principle of boosting weak learners (CARTs generally) using the gradient descent architecture. However, XGBoost improves upon the base GBM framework through systems optimization and algorithmic enhancements.

```
import xgboost
from xgboost import XGBRegressor

scores = []
for train_index, test_index in kf.split(x):
    x_train, x_test = x.iloc[train_index], x.iloc[test_index]
    y_train, y_test = y.iloc[train_index], y.iloc[test_index]

    model = XGBRegressor()
    scores.append(check(x_train, x_test, y_train, y_test, model))

print(min(scores))
results.append((str(model).split('(')[0], min(scores), list(x.columns), False))
```

```
15.547370607629134
```

Results –

For Casual

	0	1
	XGBRegressor	15.547371
	GradientBoostingRegressor	19.867903
	DecisionTreeRegressor	22.551913
	AdaBoostRegressor	29.017033
	LinearRegression	32.538105

For Registered –

	0	1
	XGBRegressor	51.163880
	GradientBoostingRegressor	62.110846
	DecisionTreeRegressor	73.917667
	AdaBoostRegressor	94.331863
	LinearRegression	100.237459

Week – 5

Hyperparameters Tuning

Hyperparameter tuning is the process of tuning the parameters present as the tuples while we build machine learning models. These parameters are defined by us which can be manipulated according to programmer wish. Machine learning algorithms never learn these parameters. These are tuned so that we could get good performance by the model. Hyperparameter tuning aims to find such parameters where the performance of the model is highest or where the model performance is best and the error rate is least.

Tuning your hyperparameters is absolutely critical in obtaining a high-accuracy model. Many machine learning models have various knobs, dials, and parameters that you can set. The difference between a very low-accuracy model versus a high-accuracy one is sometimes as simple as tuning the right dial.

Grid Search –

- We start by defining a set of all hyperparameters and the associated values we want to explore
- The grid search then examines all combinations of these hyperparameters
- For each possible combination of hyperparameters, we train a model on them
- The hyperparameters associated with the highest accuracy are then returned

The drawback of this method is that it is time-consuming and it is hard to keep track of hyperparameters we tried and we still have to try.

Random Search –

- Define the hyperparameters we want to search over
- Set a lower and upper bound on the values (if it's a continuous variable) or the possible values the hyperparameter can take on (if it's a categorical variable) for each hyperparameter
- A random search then randomly samples from these distributions a total of N times, training a model on each set of hyperparameters
- The hyperparameters associated with the highest accuracy model are then returned

Random Search (XGBoost)

```
xgb_r = xgboost.XGBRegressor()

xgb_params = {
    "learning_rate":[0.05,0.10,0.15,0.20,0.25,0.30],
    "max_depth":[3,4,5,6,8,10,12,15],
    "min_child_weight":[1,3,5,7],
    "gamma":[0.0,0.1,0.2,0.3,0.4],
    "colsample_bytree":[0.3,0.4,0.5,0.7],
}
xgb_model = RandomizedSearchCV(estimator = xgb_r, param_distributions = xgb_params,cv=5, n_iter=100, verbose=2)
xgb_model.fit(x_train,y_train)
```

When tuning hyperparameters, there isn't just one "golden set of values" that will give you the highest accuracy. Instead, it's a distribution — there are ranges for each hyperparameter that will obtain the best accuracy. If you land within that range/distribution, you'll still enjoy the same high accuracy without the requirement of exhaustively tuning your hyperparameters with a grid search.