

PROJECT REPORT

PREDICTION OF BIKE RENTAL COUNTS BASIS SEASONAL SETTINGS

Author: Prashant

INDEX

Chapter 1

- 1.1 Problem Statement
- 1.2 Data

Chapter 2

Methodology

- Pre-Processing
- Modelling
- Model Selection

Chapter 3

Pre-Processing

- 3.1 Looking into the data and analyzing various variables
- 3.2 Uniqueness in Variable
- 3.3 Missing Value Analysis
- 3.4 Outlier Analysis
- 3.5 Feature Selection
- 3.6 Features Scaling
- 3.7 Dummy variable analysis

Chapter 4

Modelling

- 4.1 Decision Tree
- 4.2 Random Forest
- 4.3 Linear Regression
- 4.4 Gradient Boosting
- 4.5 Principal Component Analysis
- 4.6 Hyper Parameters Tunings on PCA results

Chapter 5

Conclusion

- 5.1 Model Evaluation
- 5.2 Model Selection

References

Appendix

Chapter 1

Introduction

1.1 Problem Statement

The objective of this Case is to Prediction of bike rental count on daily based on the environmental and seasonal settings.

1.2 Data

Understanding of data is the very first and important step in the process of finding solution of any business problem. Here we will proceed with two years data which is 2011 and 2012 with 15 variable including one dependent variable.

Dataset Characteristics: Timeseries Multivariant

Size of Dataset Provided: - 731 rows, 15 Columns (including dependent variable)

Missing Values: Yes

Below mentioned is a list of all the variable names with their meanings:

Variables	Description
Dteday	Date of renting
Season	Season (1: springer, 2: summer, 3: fall, 4: winter)
Yr	Year (0: 2011, 1:2012)
Mnth	Month (1 to 12)
Holiday	Weekday weather day is holiday or not.
Weekday	Day of the week (starting from Sunday= 0 to Saturday= 6)
Workingday	If day is neither weekend nor holiday is 1, otherwise is 0.
Weathersit	extracted from Freemeteo
Temp	Normalized temperature in Celsius.
Atemp	Normalized feeling temperature in Celsius.
Hum	Normalized humidity. The values are divided to 100 (max)
Windspeed	Normalized wind speed. The values are divided to 67 (max)
Casual	Count of casual users
Registered	Count of registered users
Cnt	Count of total rental bikes including both casual and registered

Chapter 2

Methodology

➤ Pre-Processing

When we required to build a predictive model, we require to look and manipulate the data before we start modelling which includes multiple preprocessing steps such as exploring the data, cleaning the data as well as visualizing the data through graph and plots, all these steps is combined under one shed which is **Exploratory Data Analysis**, which includes following steps:

- Looking into the data and analyzing various variables
- Missing Value Analysis
- Outlier Analysis
 - Box plot to detect outliers
 - Elimination and imputation of outliers presented
- Feature Selection
 - Correlation Analysis
 - ANOVA (Two way Anova)
- Features Scaling
 - Normalization
- Dummy variable analysis

➤ Modelling

Once all the Pre-Processing steps has been done on our data set, we will now further move to our next step which is modelling. Modelling plays an important role to find out the good inferences from the data. Choice of models depends upon the problem statement and data set. As per our problem statement and dataset, we will try some models on our preprocessed data and post comparing the output results we will select the best suitable model for our problem. As per our data set following models need to be tested:

- Linear regression
 - Decision Tree
 - Random forest,
 - Gradient Boosting
- ❖ We have also used **Principal component analysis** is also used and post reducing the dimensions using PCA above models are again tested.
- ❖ Post applying PCA we have used hyper tuning technique with aim to improve the results.

➤ **Model Selection**

The final step of our methodology will be the selection of the model based on the different output and results shown by different models. We have multiple parameters which we will study further in our report to test whether the model is suitable for our problem statement or not.

Chapter 3

Pre-Processing

3.1 Looking into the data and analyzing various variables

In this report we will try to predict of bike rental count based on the environmental and seasonal settings. So here we have a data set of 15 variable of two years data which is 2011 and 2012.

Independent variables: Dteday , Season, Yr, Mnth, Holiday, Weekday, Workingday, Weathersit, Temp, Attemp, Hum, Windspeed, Casual, Registered.\

Our Dependent variable: **Cnt (count of total rental bikes including both casual and registered)**

We have also tried to find out the category of the variables:

<u>Variable Name</u>	<u>Variable Type</u>
instant	int64
dteday	object
season	int64
yr	int64
mnth	int64
holiday	int64
weekday	int64
workingday	int64
weathersit	int64
temp	float64
attemp	float64
hum	float64
windspeed	float64
casual	int64
registered	int64
cnt	int64

Here **int64** means categorical or non-numeric variable and **float64** means numeric variables.

3.2 Uniqueness in Variable

We need to look at the unique number in the variables which help us to decide whether the variable is categorical or numeric. So, by using python script ‘nunique’ we tried to find out the unique values in each variable. We have also added the table below:

Variable Name	Unique Counts
instant	731
Dteday	731
Season	4
yr	2
Mnth	12
Holiday	2
Weekday	7
Workingday	2
Weathersit	3
Temp	499
Atemp	690
Hum	595
Windspeed	650
Casual	606
Registered	679
cnt	696

Categorical Variables - season, year, month, holiday, weekday, working day, weathersit

3.3 Missing Value Analysis

Missing values in data is a common phenomenon in real world problems. It can have a significant effect on the conclusions that can be drawn from the data.

The most common reason for missing values in our data could be:

- Human error
- Refuse to answer
- Option to fill

As per the industry standards if a column has more than 30% of data as missing value either we ignore the entire column, or we ignore those observations.

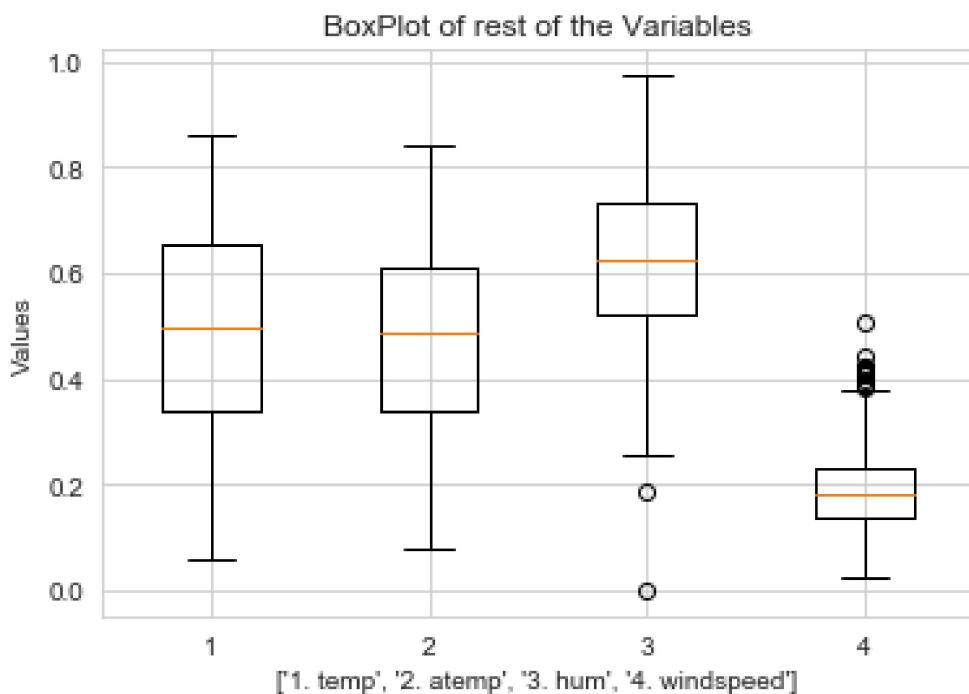
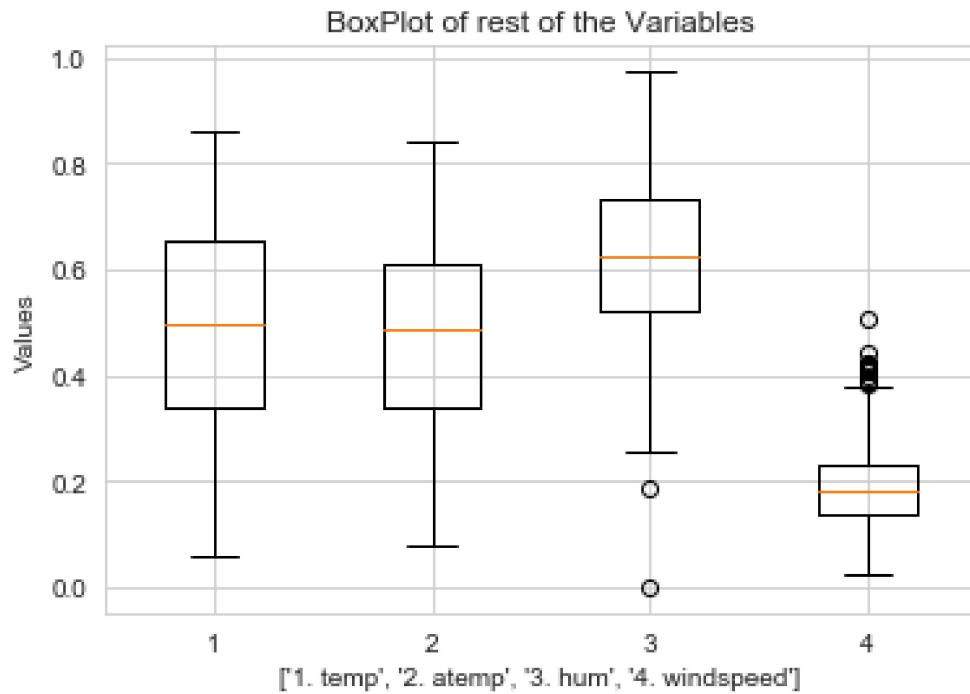
As per our data set there is no missing values present so will skip this step and move to our next step.

3.4 Outlier Analysis

The next step of Preprocessing Technique is **Outliers Analysis**, An Outlier is a rare chance of occurrence within a given data set. In Data Science, an Outlier is an observation point that is distant from other observations. An Outlier may be due to variability in the measurement or it may indicate experimental error.

We will use boxplot technique to see whether outliers is present in continuous variables or not.

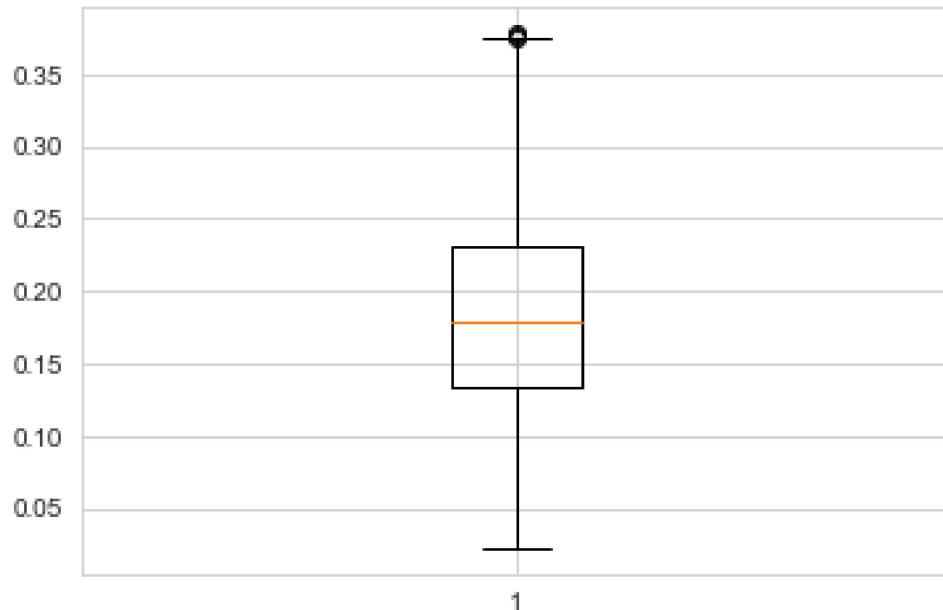
- ❖ **Boxplot:** - It is a method for graphically depicting groups of numerical data through their quartiles. Box plots may also have lines extending vertically from the boxes (whiskers) indicating variability outside the upper and lower quartiles.



From the above boxplots we can see that variables “windspeed” have outliers.

Treatment of outliers: remove the rows containing outlier values using the outlier formula.

Below are the box plot of windspeed after treating the outliers:



As we can see all the outlier figures has been successfully imputed.

3.5 Feature Selection

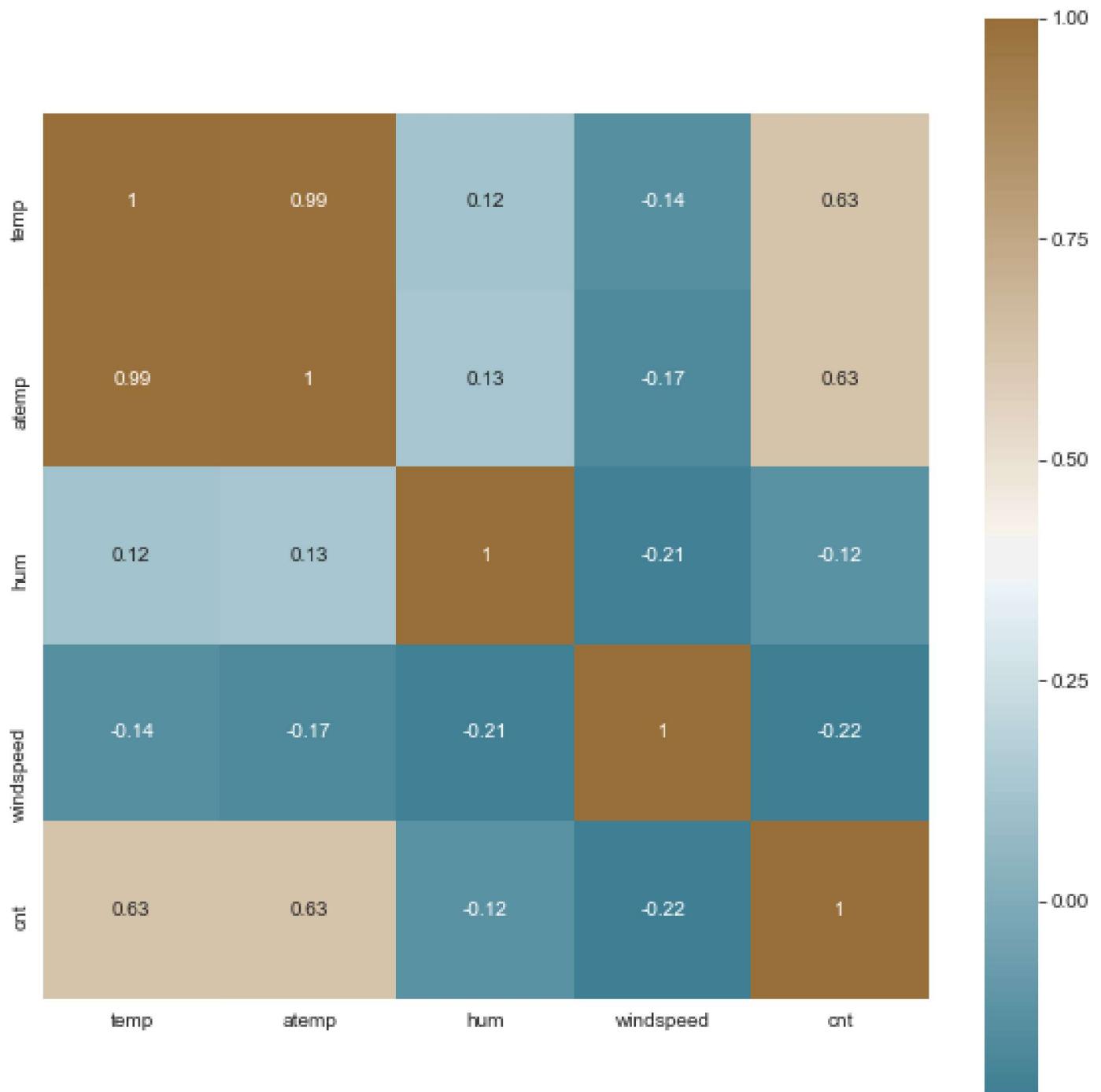
Feature selection is also called variable selection or attribute selection. Basic rule on which Machine learning works is— *if you put garbage in, you will only get garbage to come out*. By garbage here, we mean to say noise in the data.

When the number of features is very large, this rule becomes even more important. We need not use every feature at our disposal for creating an algorithm. We can assist our algorithm by feeding in only those features that are really important. We have witnessed feature subsets giving better results than complete set of features for the same algorithm or – “*Sometimes, less is better!*”.

The selection of features or variables is based on following two conditions:

1. The relationship between two independent variables should be less and
2. The relationship between Independent and Target variables should be high.

A correlogram or correlation matrix allows to analyze the relationship between each pair of numerical variables of a matrix.



The variable which we are going to drop is '**temp**' as it is showing high correlation with '**atemp**' variable which means both the variables are carrying the similar information so there would be no need to continue with both the variable, we can proceed further with only one variable out of these two. So, we are proceeding with **atemp** variable.

In this project we have selected Correlation Analysis for numerical variable and ANOVA testing categorical variable. So as per our hypothesis which is:

H0 – all the variables are independent of each other

H1 – all the variables are not independent of each other

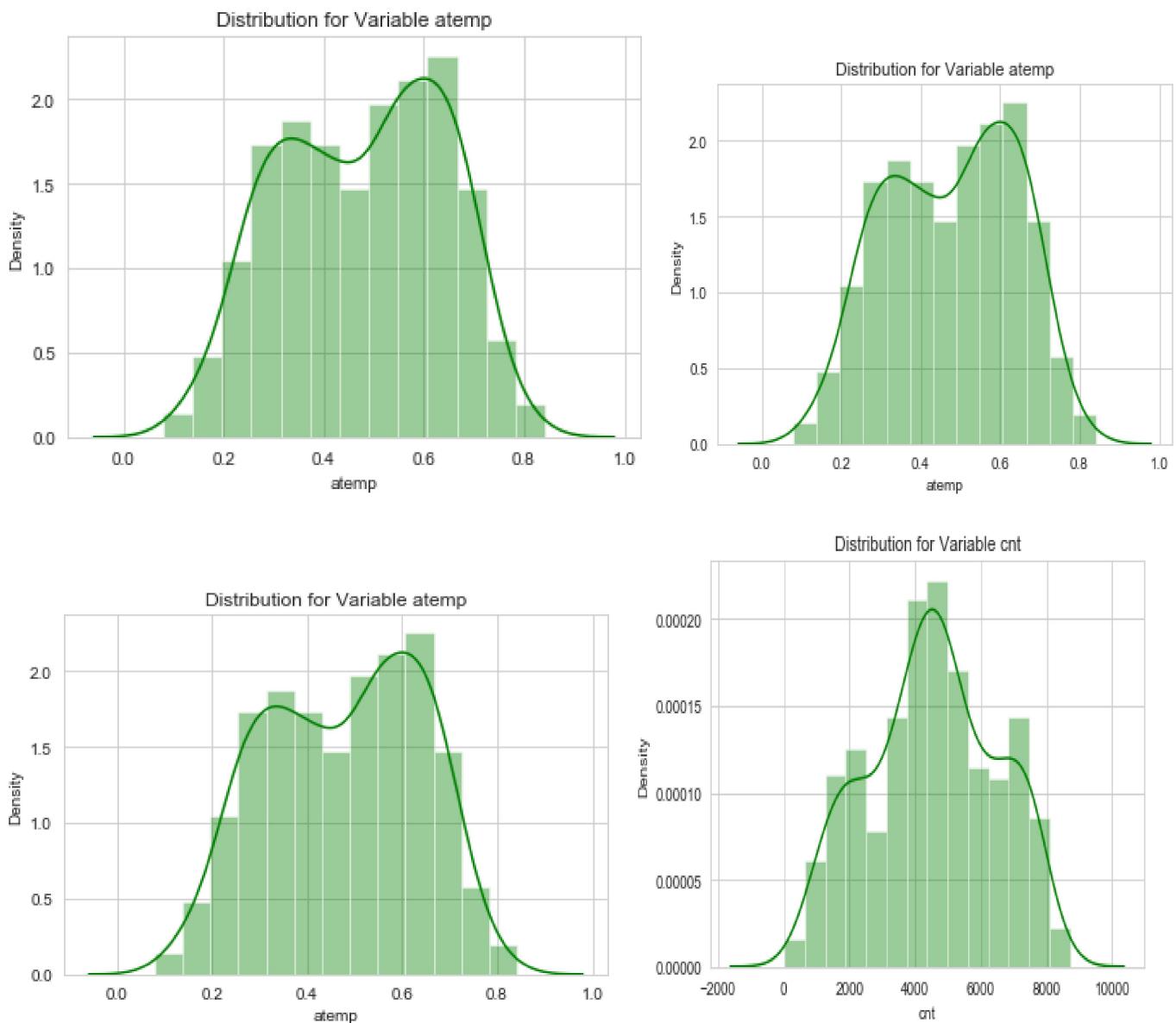
We will drop categorical variables 'holiday', 'weekday', 'workingday', 'instant' and 'dteday' for our further modelling steps basis our ANOVA results.

3.6 Features Scaling

Skewness is asymmetry in a statistical distribution, in which the curve appears distorted or skewed either to the left or to the right. Skewness can be quantified to define the extent to which a distribution differs from a normal distribution. Here we tried to show the skewness of our variables and we find that our target variable absenteeism in hours having is one sided skewed so by using log transform technique we tried to reduce the skewness of the same.

In the simplest cases, **normalization** of ratings means adjusting values measured on different scales to a notionally common scale, often prior to averaging. In more complicated cases, normalization may refer to more sophisticated adjustments where the intention is to bring the entire probability distributions of adjusted values into alignment. In the case of normalization of scores in educational assessment, there may be an intention to align distributions to a normal distribution. A different approach to normalization of probability distributions is quantile normalization, where the quantiles of the different measures are brought into alignment.

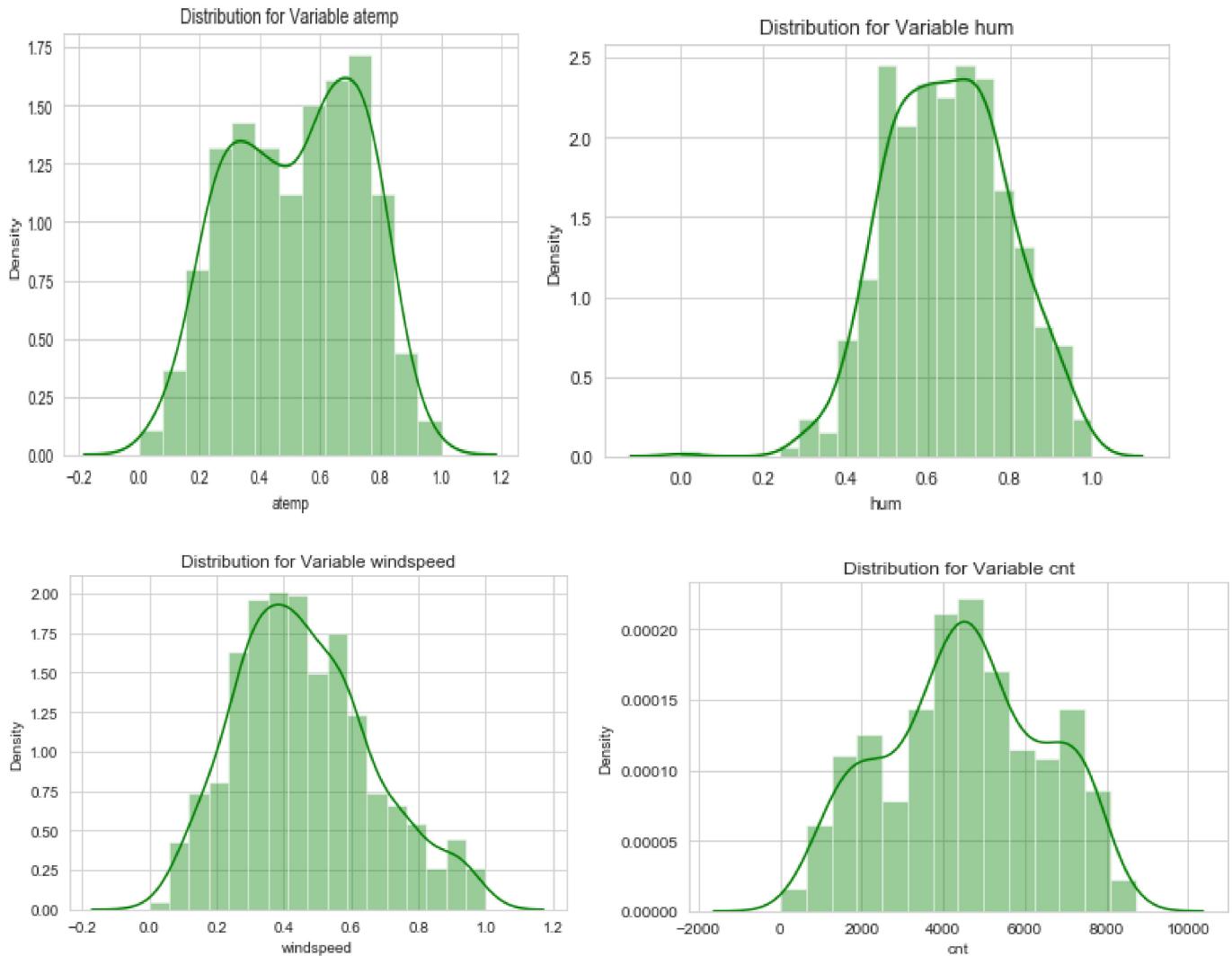
Below mentioned graphs shows the probability distribution plot to check distribution:



As we can see that our continuous variables are not normally distributed which means that we have to use normalization technique to normalize the values of each continuous variable to proceed for modelling stage. Rescaling data to have values between **0 and 1**. This is usually called feature scaling. One possible formula to achieve this is:

$$x_{new} = \frac{x - x_{min}}{x_{max} - x_{min}}$$

After normalization plot we have created distribution below:



3.7 Dummy variable analysis

A dummy variable is a numerical variable used in regression analysis to represent the importance of categorical variables. Here we can see there are 6 categorical variables so we have created dummies for each categorical variable (there unique values). These dummy variables will interact with our model and show the significance of categorical variables in quantitative terms.

Chapter 4

Modelling

After a thorough preprocessing, we will use some regression models on our processed data to predict the target variable. Following are the models which we have built –

- Linear Regression
- Decision Tree
- Random Forest
- Gradient Boosting

Before we start building our models, we would like to know how our model will be evaluated.

4.1 Linear Regression

Multiple linear regression is the most common form of linear regression analysis. Multiple regression is an extension of simple linear regression. It is used as a predictive analysis, when we want to predict the value of a variable based on the value of two or more other variables. The variable we want to predict is called the dependent variable (or sometimes, the outcome, target or criterion variable).

Before running any model, we will split our data into two parts which is train and test data. Here in our case we have taken 80% of the data as our train data. Below is the snipped image of the split of train test.

Machine Learning algorithms

```
####train test split
```

```
In [83]: from sklearn.tree import DecisionTreeRegressor  
from sklearn.metrics import mean_squared_error
```

```
In [88]: ##train test split for further modelling  
X_train, X_test, y_train, y_test = train_test_split( dummy_df1.iloc[:, dummy_df1.columns != 'cnt'],  
dummy_df1.iloc[:, 3], test_size = 0.20, random_state = 1)
```

```
In [89]: #checking the shape of the train and test  
print (X_train.shape, y_train.shape)  
print (X_test.shape, y_test.shape)
```

```
(574, 24) (574,)  
(144, 24) (144,)
```

Below is the screenshot of the query we executed and the result shown, we will compare the results of each model in a combined table later on.

```
In [97]: # Building model on top of training dataset
fit_LR = LinearRegression().fit(X_train , y_train)

In [98]: #prediction on train data
pred_train_LR = fit_LR.predict(X_train)

#prediction on test data
pred_test_LR = fit_LR.predict(X_test)

In [99]: ##calculating RMSE for train data
RMSE_train_LR = np.sqrt(mean_squared_error(y_train, pred_train_LR))

##calculating RMSE for test data
RMSE_test_LR = np.sqrt(mean_squared_error(y_test, pred_test_LR))

In [100]: print("Root Mean Squared Error For Training data = "+str(RMSE_train_LR))
print("Root Mean Squared Error For Test data = "+str(RMSE_test_LR))

Root Mean Squared Error For Training data = 748.0128017406815
Root Mean Squared Error For Test data = 901.6852393342887

In [101]: #calculate R^2 for train data
r2_score(y_train, pred_train_LR)

Out[101]: 0.845245138638133

In [102]: #calculate R^2 for test data
r2_score(y_test, pred_test_LR)

Out[102]: 0.810161927005014
```

Below is the screenshot of the Linear regression model OLS summary

OLS Regression Results							
Dep. Variable:	cnt	R-squared:	0.831				
Model:	OLS	Adj. R-squared:	0.804				
Method:	Least Squares	F-statistic:	30.30				
Date:	Thu, 02 May 2019	Prob (F-statistic):	4.00e-38				
Time:	12:23:45	Log-Likelihood:	-1175.6				
No. Observations:	144	AIC:	2393.				
DF Residuals:	123	BIC:	2456.				
DF Model:	20						
Covariance Type:	nonrobust						
	coef	std err	t	P> t	[0.025	0.975]	
atemp	4588.1305	942.856	4.866	0.000	2721.804	6454.457	
hum	-1444.7384	690.151	-2.093	0.038	-2810.850	-78.627	
windspeed	-962.5245	453.138	-2.124	0.036	-1859.483	-65.566	
season_1	-190.8677	363.973	-0.524	0.601	-911.331	529.595	
season_2	777.1909	336.352	2.311	0.023	111.403	1442.979	
season_3	431.6207	366.056	1.179	0.241	-292.966	1156.207	
season_4	1353.4568	363.639	3.722	0.000	633.656	2073.258	
yr_0	216.4115	284.412	0.761	0.448	-346.564	779.387	
yr_1	2154.9893	311.034	6.928	0.000	1539.316	2770.663	
mnth_1	-26.6017	480.272	-0.055	0.956	-977.270	924.067	
mnth_2	168.0041	439.412	0.382	0.703	-701.786	1037.794	
mnth_3	520.4274	333.416	1.561	0.121	-139.550	1180.405	
mnth_4	230.2353	441.791	0.521	0.603	-644.264	1104.734	
mnth_5	222.3173	429.507	0.518	0.606	-627.866	1072.501	
mnth_6	540.8332	396.535	1.364	0.175	-244.083	1325.750	
mnth_7	-292.7100	495.294	-0.591	0.556	-1273.115	687.695	
mnth_8	95.1684	487.998	0.195	0.846	-870.793	1061.130	
mnth_9	855.9332	369.631	2.316	0.022	124.272	1587.595	
mnth_10	437.0390	425.696	1.027	0.307	-405.600	1279.678	
mnth_11	200.1014	470.520	0.020	0.100	1221.805	510.012	

4.2 Decision Tree

A tree has many analogies in real life, and turns out that it has influenced a wide area of machine learning, covering both classification and regression. In decision analysis, a decision tree can be used to visually and explicitly represent decisions and decision making. As the name goes, it uses a tree-like model of decisions.

To say it in simple words: Random forest builds multiple decision trees and merges them together to get a more accurate and stable prediction.

Below is a screenshot of the model we build and its output:

```
###Decision Tree
```

```
In [90]: fit_DT = DecisionTreeRegressor(max_depth = 2).fit(x_train,y_train)

In [91]: #prediction on train data
pred_train_DT = fit_DT.predict(x_train)

#prediction on test data
pred_test_DT = fit_DT.predict(x_test)

In [92]: ##calculating RMSE for train data
RMSE_train_DT = np.sqrt(mean_squared_error(y_train, pred_train_DT))

##calculating RMSE for test data
RMSE_test_DT = np.sqrt(mean_squared_error(y_test, pred_test_DT))

In [93]: print("Root Mean Squared Error For Training data = "+str(RMSE_train_DT))
print("Root Mean Squared Error For Test data = "+str(RMSE_test_DT))

Root Mean Squared Error For Training data = 1030.2928796742729
Root Mean Squared Error For Test data = 1282.4613468501855

In [94]: ## R^2 calculation for train data
r2_score(y_train, pred_train_DT)

Out[94]: 0.7064056888207673

In [95]: ## R^2 calculation for test data
r2_score(y_test, pred_test_DT)

Out[95]: 0.6159728183399968
```

4.3 Random forests

Random forests or random decision forests are an ensemble learning method for classification, regression and other task, that operate by constructing a multitude of decision trees at training time and outputting the class that is the mode of the classes (classification) or mean prediction (regression) of the individual trees. Random decision forests correct for decision trees' habit of overfitting to their training set.

Below is a screenshot of the model we build and its output:

```
##random forest model

In [104]: # Importing Libraries for Random Forest
from sklearn.ensemble import RandomForestRegressor

In [105]: #here we have chose n_estimators = 200, we will further try to evaluate the performance of the model by tuning
fit_RF = RandomForestRegressor(n_estimators = 200).fit(X_train,y_train)

In [106]: #prediction on train data
pred_train_RF = fit_RF.predict(X_train)
#prediction on test data
pred_test_RF = fit_RF.predict(X_test)

In [107]: ##calculating RMSE for train data
RMSE_train_RF = np.sqrt(mean_squared_error(y_train, pred_train_RF))
##calculating RMSE for test data
RMSE_test_RF = np.sqrt(mean_squared_error(y_test, pred_test_RF))

In [108]: print("Root Mean Squared Error For Training data = "+str(RMSE_train_RF))
print("Root Mean Squared Error For Test data = "+str(RMSE_test_RF))

Root Mean Squared Error For Training data = 259.9264030257324
Root Mean Squared Error For Test data = 812.026249578066

In [109]: ## calculate R^2 for train data
r2_score(y_train, pred_train_RF)

Out[109]: 0.9813135401056612

In [110]: #calculate R^2 for test data
r2_score(y_test, pred_test_RF)

Out[110]: 0.8460380038515609
```

4.4 Gradient Boosting

Gradient boosting is a machine learning technique for regression and classification problems, which produces a prediction model in the form of an ensemble of weak prediction models, typically decision trees. It builds the model in a stage-wise fashion like other boosting methods do, and it generalizes them by allowing optimization of an arbitrary differentiable loss function.

Below is a screenshot of the model we build and its output:

```

    ###gradient boosting model

[112]: # Importing Library for GradientBoosting
from sklearn.ensemble import GradientBoostingRegressor

[113]: # Building model on top of training dataset
fit_GB = GradientBoostingRegressor().fit(x_train, y_train)

[114]: #prediction on train data
pred_train_GB = fit_GB.predict(x_train)
#prediction on test data
pred_test_GB = fit_GB.predict(x_test)

[115]: ##calculating RMSE for train data
RMSE_train_GB = np.sqrt(mean_squared_error(y_train, pred_train_GB))
##calculating RMSE for test data
RMSE_test_GB = np.sqrt(mean_squared_error(y_test, pred_test_GB))

[116]: print("Root Mean Squared Error For Training data = "+str(RMSE_train_GB))
print("Root Mean Squared Error For Test data = "+str(RMSE_test_GB))

Root Mean Squared Error For Training data = 437.9280086306251
Root Mean Squared Error For Test data = 751.108268821793

[119]: #calculate R^2 for train data
r2_score(y_train, pred_train_GB)

[119]: 0.9469565596823101

[120]: #calculate R^2 for test data
r2_score(y_test, pred_test_GB)

[120]: 0.8682718834344123

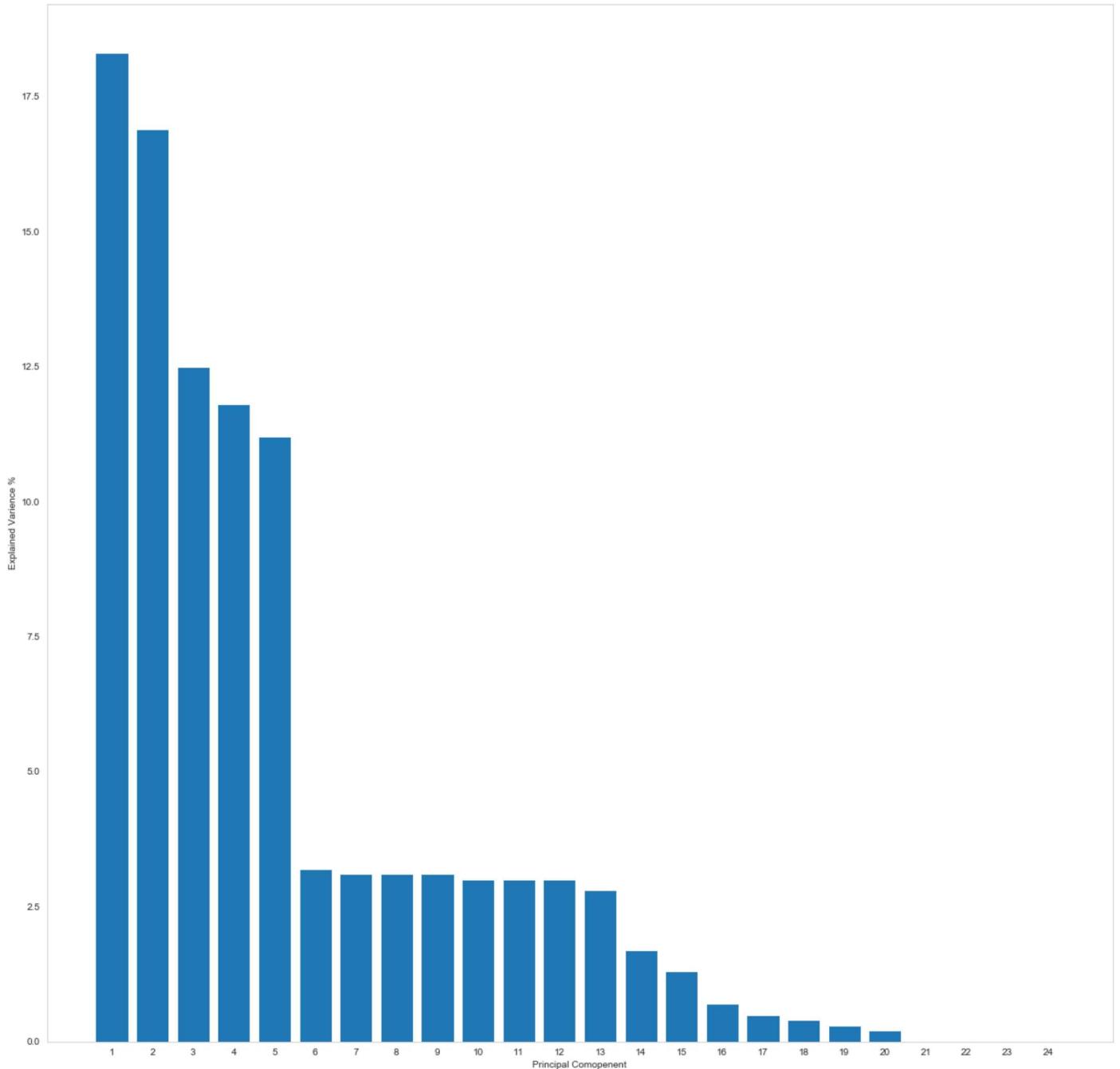
```

4.5 Principal Component Analysis

Principal Component Analysis (PCA) is a dimension-reduction tool that can be used to reduce a large set of variables to a small set that still contains most of the information in the large set.

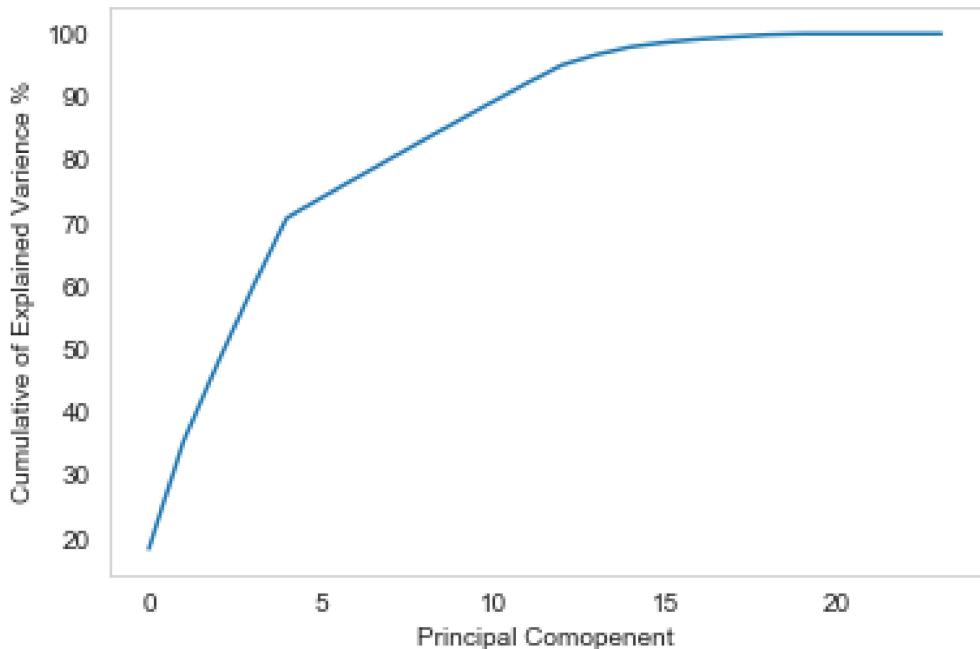
- Principal component analysis (PCA) is a mathematical procedure that transforms a number of (possibly) correlated variables into a (smaller) number of uncorrelated variables called principal components.
- The first principal component accounts for as much of the variability in the data as possible, and each succeeding component accounts for as much of the remaining variability as possible.
- Principal components analysis is similar to another multivariate procedure called Factor Analysis. They are often confused and many scientists do not understand the difference between the two methods or what types of analyses they are each best suited.

Below is a graph shows the number of variables explains the dependent variable:



This curve shows the % increase in the explained variance by increasing the number of factors or variables. Here we can see more than 95% of the variance is explained by 15 principal components or variables.

So here we can reduce our model dimensions from 25 variables to 15 variables with the focus of optimization of model.



We will mention the model output post using the Principal Component Analysis and check whether there is any significant change in the result is noticed in a combined table while evaluation of model.

4.6 Hyper Parameters Tunings on PCA results

Model hyperparameters are set by the data scientist ahead of training and control implementation aspects of the model. The weights learned during training of a linear regression model are parameters while the number of trees in a random forest is a model hyperparameter because this is set by the data scientist. Hyperparameters can be thought of as model settings. These settings need to be tuned for each problem because the best model hyperparameters for one particular dataset will not be the best across all datasets. The process of hyperparameter tuning (also called hyperparameter optimization) means finding the combination of hyperparameter values for a machine learning model that performs the best - as measured on a validation dataset - for a problem.

Here we have used two hyper parameters tuning techniques

- Random Search CV
- Grid Search CV

1. **Random Search CV:** This algorithm set up a grid of hyperparameter values and select random combinations to train the model and score. The number of search iterations is set based on time/resources.
2. **Grid Search CV:** This algorithm set up a grid of hyperparameter values and for each combination, train a model and score on the validation data. In this approach, every single combination of hyperparameters values is tried which can be very inefficient.

Check results post hyper parameter tuning on post PCA models in the model valuation section.

Chapter 5

Conclusion

5.1 Model Evaluation

The main concept of looking at what is called residuals or difference between our predictions $f(x[I,:])$ and actual outcomes $y[i]$.

In general, most data scientists use two methods to evaluate the performance of the model:

- I. **RMSE** (Root Mean Square Error): is a frequently used measure of the difference between values predicted by a model and the values actually observed from the environment that is being modelled.

$$RMSE = \sqrt{\frac{\sum_{i=1}^n (X_{obs,i} - X_{model,i})^2}{n}}$$

- II. **R Squared(R²):** is a statistical measure of how close the data are to the fitted regression line. It is also known as the coefficient of determination, or the coefficient of multiple determination for multiple regression. In other words, we can say it explains as to how much of the variance of the target variable is explained.
- III. We have shown both train and test data results, the main reason behind showing both the results is to check whether our data is overfitted or not.

Below table shows the model results before applying PCA:

Model Name	RMSE		R Squared	
	Train	Test	Train	Test
Linear Regression	748.0	901.6	0.84	0.81
Decision Tree	1030.29	1282.4	0.70	0.61
Random Forest model	259.92	812.02	0.98	0.84
Gradient Boosting	437.92	751.10	0.94	0.86

Below table shows the model results after applying PCA:

Model Name	RMSE		R Squared	
	Train	Test	Train	Test
Linear Regression (PCA)	802.6	961.03	0.82	0.78
Decision Tree (PCA)	1198.11	1336.35	0.60	0.58
Random Forest (PCA)	271.008	917.75	0.97	0.80
Gradient Boosting (PCA)	419.15	885.69	0.95	0.81

Below table shows results post using hyper parameter tuning techniques:

Model Name	Random Search CV		Grid Search CV	
	RMSE (Test)	R Squared (Test)	RMSE (Test)	R Squared (Test)
Random Forest (PCA)	936.51	0.80	954.82	0.79
Gradient Boosting (PCA)	1018.74	0.76	957.86	0.79

Above table shows the results after tuning the parameters of our two best suited models i.e. Random Forest and Gradient Boosting. For tuning the parameters, we have used Random Search CV and Grid Search CV under which we have given the range of n_estimators, depth and CV folds.

5. 2 Model Selection

On the basis RMSE and R Squared results a good model should have least RMSE and max R Squared value. So, from above tables we can see:

- Gradient Boosting Regressor model shows best result among all before applying the PCA.
- After applying PCA also gradient boosting model shows best results compared to rest of three.
- And also, after tuning the parameters of both random forest and gradient boost models, we get random forest shows better results compared to gradient boosting after both parameter tuning techniques.
- So finally, we can say that Gradient Boosting Regressor is the best method to solve our problem with highest explained variance of the target variables and lowest error chances.
- And if we want to optimize the parameters using Random Search CV then Random forest will be the best suitable model.

End of Report