Bike Renting

Abhishek Hupele

5 Sep 2018

Contents

Introduction 3

1.1 Problem Statement 3

1.2 Data 3

Methodology 4

2.1 Exploratory Data Analysis 4

2.1.1 Data Visualisation 6

2.2 Pre Processing 10

2.2.1 Assigning Levels/Categories to Categorical Variables 10

2.2.2 Outlier Analysis 10

2.2.3 Feature Scaling 11

2.2.4 Dummy Variable/One hot encoding of factor variables 11

2.2.5 Correlation Analysis 11

2.2.5 Feature Engineering: 13

2.4. Model Selection 16

2.5 Model Evaluation 19

Python : 19

R : 23

2.5.1 Comparing results of the models 26

Python Code 27

Expl Data Analysis : 27

Code : 31

R Code : 38

```{r} 38
```

Introduction

1.1 Problem Statement

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

1.2 Data

File : 'day.csv'

The details of data attributes in the dataset are as follows -
instant: Record index
dteday: Date
season: Season (1:springer, 2:summer, 3:fall, 4:winter)
yr: Year (0: 2011, 1:2012)
mnth: Month (1 to 12)
hr: Hour (0 to 23)
holiday: weather day is holiday or not (extracted fromHoliday Schedule)
weekday: Day of the week
workingday: If day is neither weekend nor holiday is 1, otherwise is 0.
weathersit: (extracted fromFreemeteo)
1: Clear, Few clouds, Partly cloudy, Partly cloudy
2: Mist + Cloudy, Mist + Broken clouds, Mist + Few clouds, Mist
3: Light Snow, Light Rain + Thunderstorm + Scattered clouds, Light Rain + Scattered clouds
4: Heavy Rain + Ice Pallets + Thunderstorm + Mist, Snow + Fog
temp: Normalized temperature in Celsius. The values are derived via (t-t_min)/(t_max-t_min),
t_min=-8, t_max=+39 (only in hourly scale)
atemp: Normalized feeling temperature in Celsius. The values are derived via (t-t_min)/(t_max-
t_min), t_min=-16, t_max=+50 (only in hourly scale)
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

day

instant	dteday	season	yr	mnth	holiday	weekday	workingday	weathersit	temp	atemp	hum	windspeed	casual	registered	cnt
1	2011-01-01	1	0	1	0	6	0	2	0.344167	0.363625	0.805833	0.160446	331	654	985
2	2011-01-02	1	0	1	0	0	0	2	0.363478	0.353739	0.696087	0.248539	131	670	801
3	2011-01-03	1	0	1	0	1	1	1	0.196364	0.189405	0.437273	0.248309	120	1229	1349
4	2011-01-04	1	0	1	0	2	1	1	0.2	0.212122	0.590435	0.160296	108	1454	1562
5	2011-01-05	1	0	1	0	3	1	1	0.226957	0.22927	0.436957	0.1869	82	1518	1600
6	2011-01-06	1	0	1	0	4	1	1	0.204348	0.233209	0.518261	0.0895652	88	1518	1606
7	2011-01-07	1	0	1	0	5	1	2	0.196522	0.208839	0.498696	0.168726	148	1362	1510
8	2011-01-08	1	0	1	0	6	0	2	0.165	0.162254	0.535833	0.266804	68	891	959
9	2011-01-09	1	0	1	0	0	0	1	0.138333	0.116175	0.434167	0.36195	54	768	822

Methodology

2.1 Exploratory Data Analysis

Data needs to be examined so the analyst knows what is there. Without Data Understanding, the analyst doesn't know what problems may arise in modelling.

- Examine key summary characteristics about the data to be used for modelling, including how many records are available, and how many target variables are included in the data.
- Begin to enumerate problems with the data, including inaccurate or invalid values, missing values , unexpected distributions, and outliers.
- Visualize data to gain further insights into the characteristics of the data, especially those masked by summary statistics.

The data has dimensions (731 x 16), 731 observations and 16 attributes .

Dataset Characteristics: Timeseries Multivariant
Number of Attributes: 16
Missing Values : No

Target/Dependent Variable : 'cnt'
Independent Variable : 'instant', 'dteday', 'season', 'yr', 'mnth', 'holiday', 'weekday', 'workingday', 'weathersit', 'temp', 'atemp', 'hum', 'windspeed', 'casual', 'registered'
Size : (731, 16)

Given below is a sample of the data set that we are using :

Continuos Variables : 'temp', 'atemp', 'hum', 'windspeed', 'casual', 'registered', 'cnt'
Discrete Variables : 'instant', 'season', 'yr', 'mnth', 'holiday', 'weekday', 'workingday', 'weathersit',

Datetime Variable : 'dteday'
Normalised Variables : 'temp', 'atemp', 'hum', 'windspeed'

Data Type -
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 731 entries, 0 to 730
Data columns (total 16 columns):
instant 731 non-null int64
dteday 731 non-null object
season 731 non-null int64
yr 731 non-null int64
mnth 731 non-null int64
holiday 731 non-null int64
weekday 731 non-null int64
workingday 731 non-null int64
weathersit 731 non-null int64
temp 731 non-null float64
atemp 731 non-null float64
hum 731 non-null float64
windspeed 731 non-null float64
casual 731 non-null int64

registered 731 non-null int64
cnt 731 non-null int64
dtypes: float64(4), int64(11), object(1)
memory usage: 91.5+ KB

	temp	atemp	hum	windspeed	casual	registered	cnt
count	731	731	731	731	731	731	731
mean	0.495385	0.474354	0.627894	0.190486	848.176471	3656.172367	4504.348837
std	0.183051	0.162961	0.142429	0.077498	686.622488	1560.256377	1937.211452
min	0.05913	0.07907	0	0.022392	2	20	22
25%	0.337083	0.337842	0.52	0.13495	315.5	2497	3152
50%	0.498333	0.486733	0.626667	0.180975	713	3662	4548
75%	0.655417	0.608602	0.730209	0.233214	1096	4776.5	5956
max	0.861667	0.840896	0.9725	0.507463	3410	6946	8714

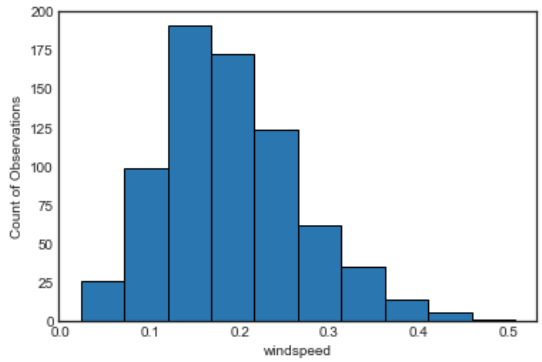
Summary Statistics :

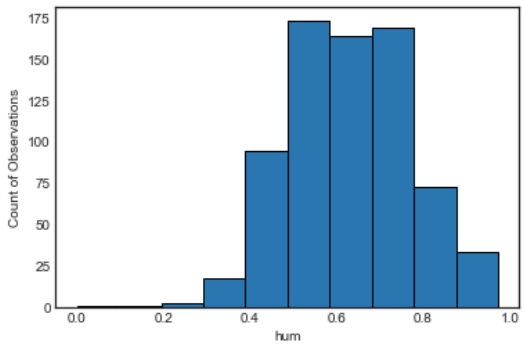
	instant	season	yr	mnth	holiday	weekday	workingday	weathersit
count	731	731	731	731	731	731	731	731
mean	366	2.49658	0.500684	6.519836	0.028728	2.997264	0.683995	1.395349
std	211.165812	1.110807	0.500342	3.451913	0.167155	2.004787	0.465233	0.544894
min	1	1	0	1	0	0	0	1
25%	183.5	2	0	4	0	1	0	1
50%	366	3	1	7	0	3	1	1
75%	548.5	3	1	10	0	5	1	2
max	731	4	1	12	1	6	1	3

2.1.1 Data Visualisation

Showing some of the visualisations used for exploratory data analysis.

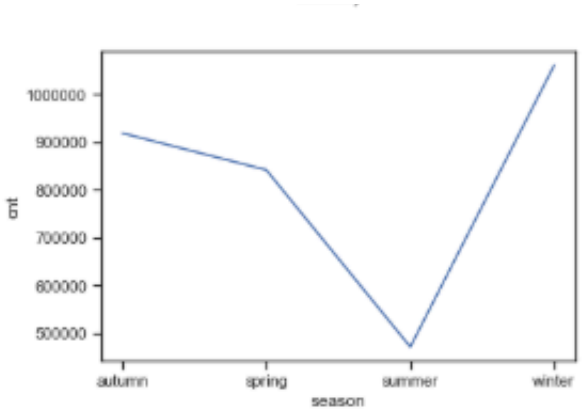
Histograms :

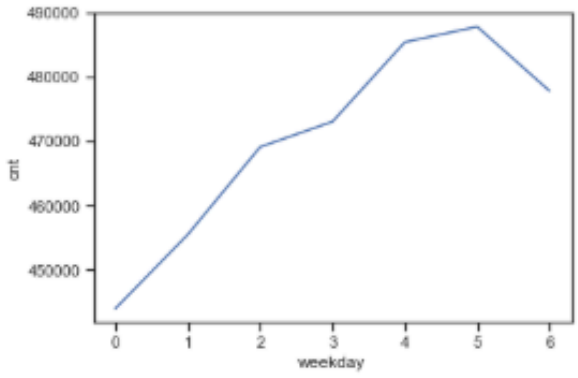


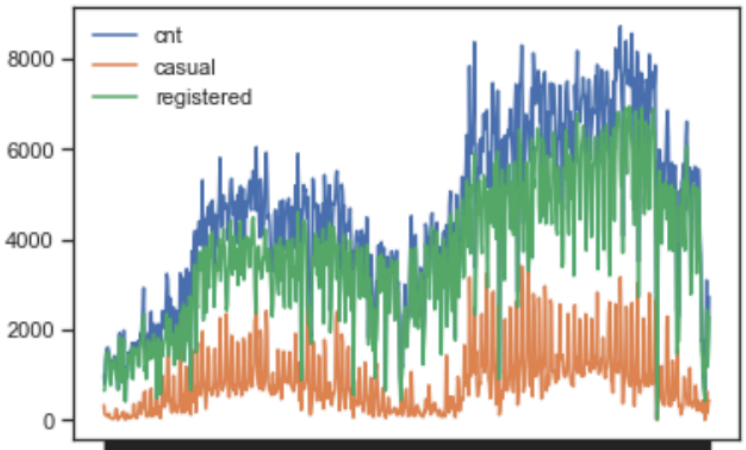


Line Graph :

Visible trend in ‘cnt’ visible with ‘season’, ‘weekday’. Also, ‘cnt’ shows shows trend with ‘dteday’ with noise. However, no cyclic pattern can be seen evident and need further investigation on time series analysis.

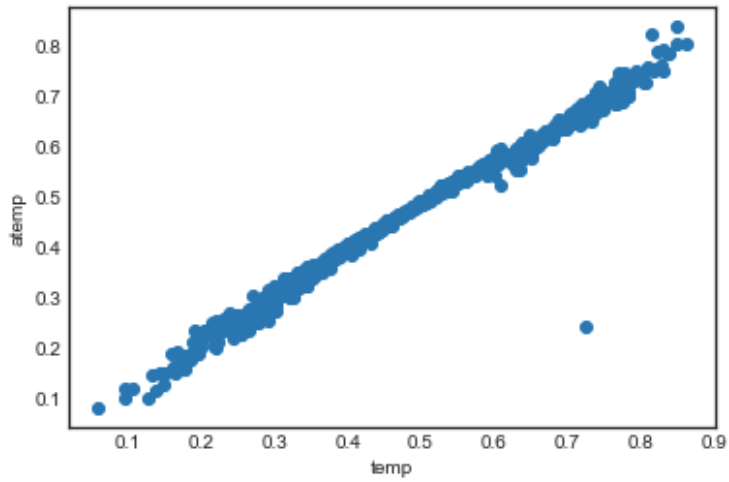


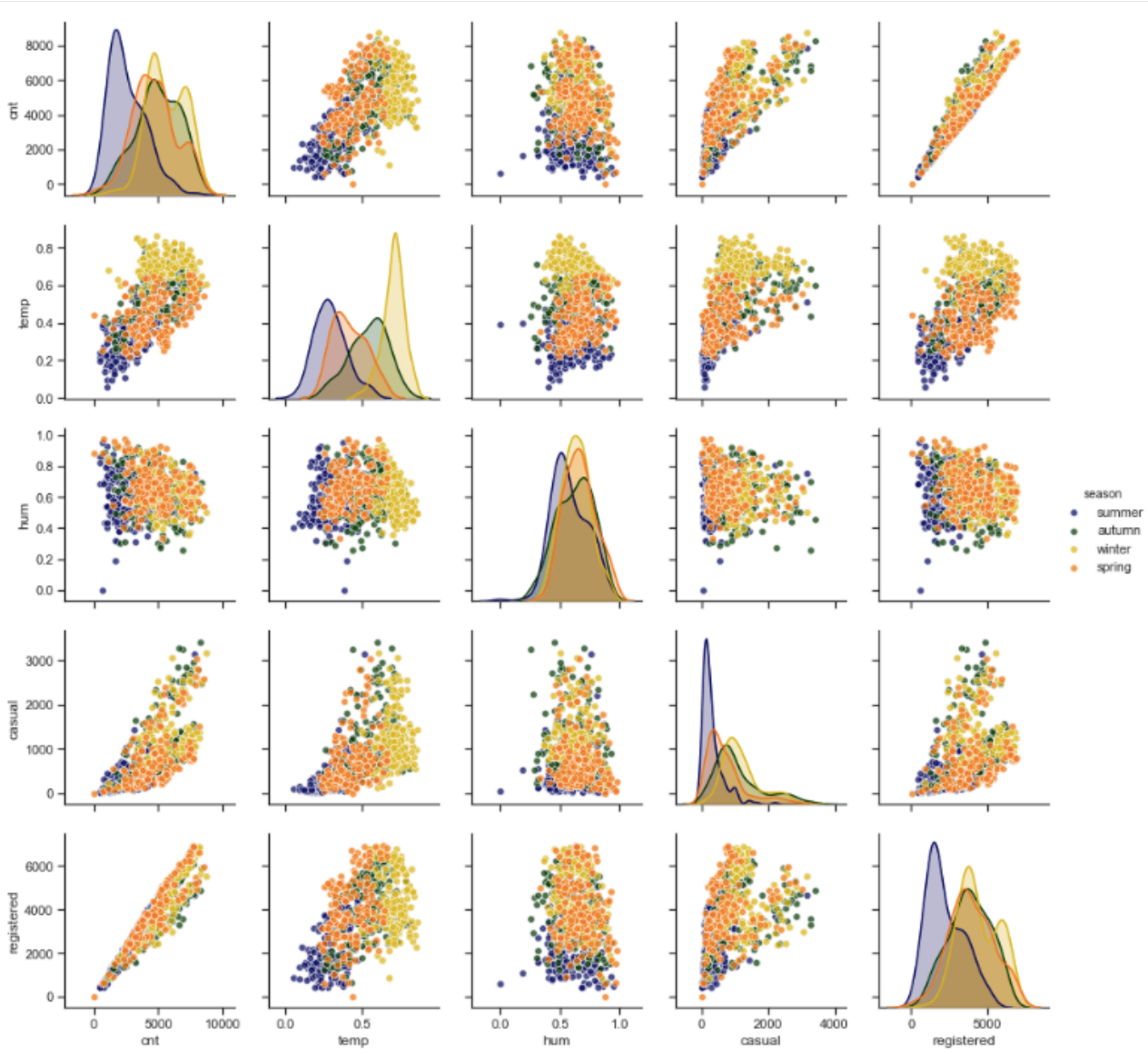


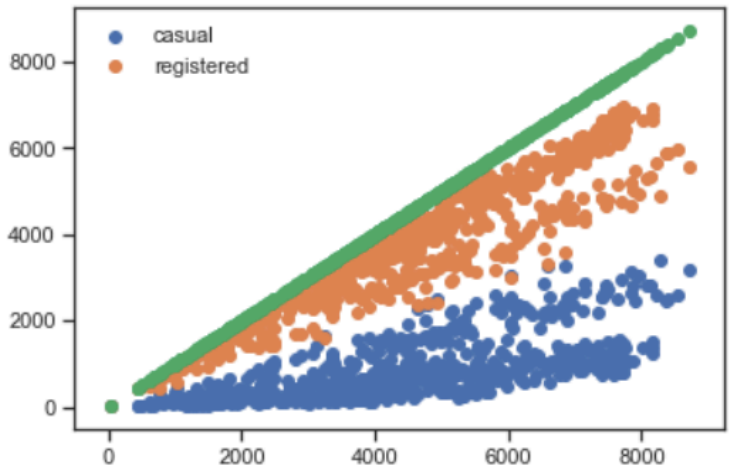


Scatter Plots :

‘atemp’ and ‘temp’ are correlated.

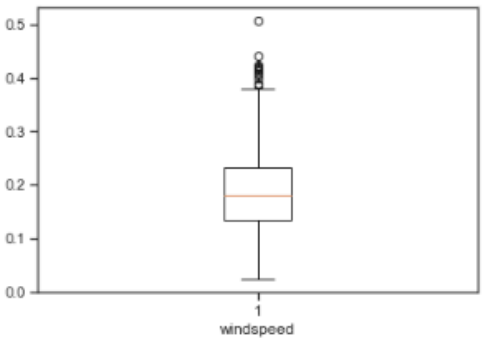


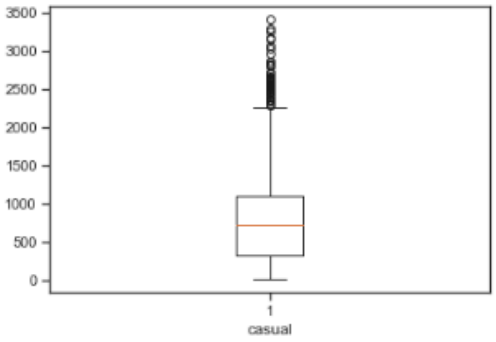


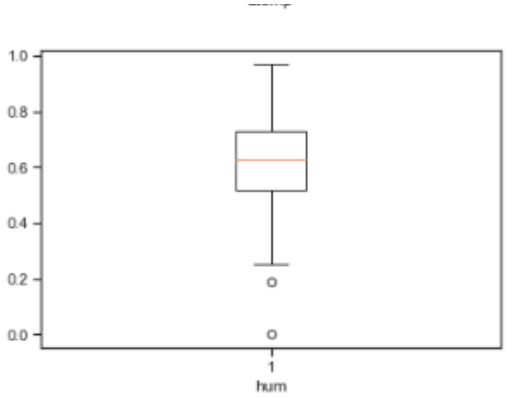


Above plot depicts scatter plot of ‘casual’, ‘registered’ and ‘casual’+’registered’ with ‘cnt’.
Target variable ‘cnt’ is summation of ‘casual’, ‘registered’ variables.
We can treat all three as ‘cnt’, ‘casual’, ‘registered’ variables as target variable to predict the most accurate model for regression analysis.

Boxplot -


0.5
0.4
0.3
0.2
0.1
0.0
1
windspeed


3500
3000
2500
2000
1500
1000
500
0
1
casual


1.0
0.8
0.6
0.4
0.2
0.0
1
hum

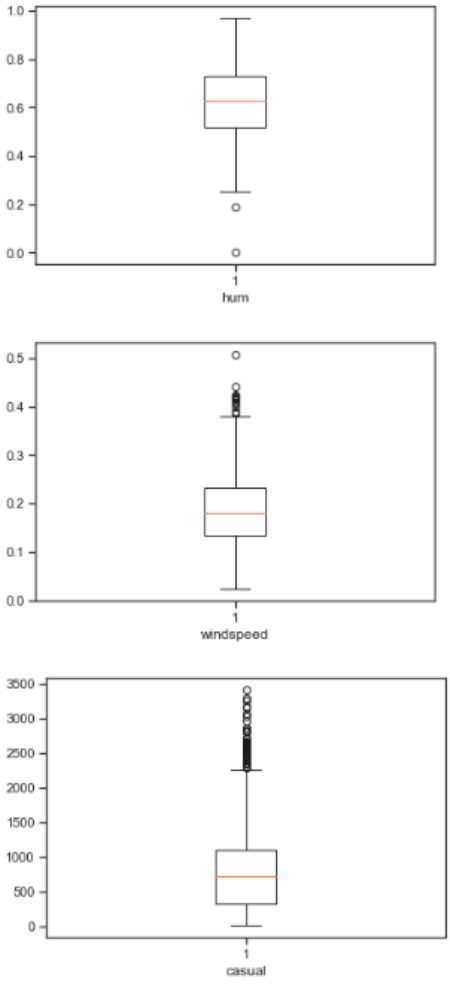
2.2 Pre Processing

2.2.1 Assigning Levels/Categories to Categorical Variables

We observe that all the variables are numeric in data set. But, we have also seen categorical variables in data as mentioned in 'Data Summary' represented by the numbers. Therefore, we need to convert these variables into categorical.

2.2.2 Outlier Analysis

An outlier is a pattern which is dissimilar with respect to the rest of the patterns in the data set.
Outliers are observed in 'hum', 'casual', 'windspeed'. Regression analysis is significantly affected by outlier values and as we are using 'casual' as target variable which has considerable outliers as visible from box plot, removing outliers may lead to significant loss of information. so, in order to preserve the information as well as prevent undue alteration due to some extreme outliers,
We will go for imputing the outliers by capping and flooring. We have replaced values greater than 0.99 quantile with value of 0.99 quantile and replaced values less than 0.01 quantile with value of 0.01 quantile.



2.2.3 Feature Scaling

Data is already normalised , so further scaling is not required.

In order to reduce unwanted variation either within or between variables, we perform either normalisation or standardisation. As all the variables are not normally distributed, we bring the continuous variables within range(0,1) by performing normalisation.

$$z = \frac{\mathrm{x} - min(x)}{[\max(x) - min(x)]}$$

Where x is an original value, z is the normalised value

Used min-max scaling for each independent variable column and have brought all of them in to same range(0,1), excluded target variable.

2.2.4 Dummy Variable/One hot encoding of factor variables

By far the most common way to represent categorical variables is using the one-hot- encoding or one-out-of-N encoding, also known as dummy variables.

The idea behind dummy variables is to replace a categorical variable with one or more new features that can have the values 0 and 1. we can represent any number of categories by introducing one new feature per category as follows.

For categorical variables where no such ordinal relationship exists, the integer encoding is not enough.

In fact, using this encoding and allowing the model to assume a natural ordering between categories may result in poor performance or unexpected results (predictions halfway between categories).

In this case, a one-hot encoding can be applied to the integer representation. This is where the integer encoded variable is removed and a new binary variable is added for each unique integer value.

2.2.5 Correlation Analysis

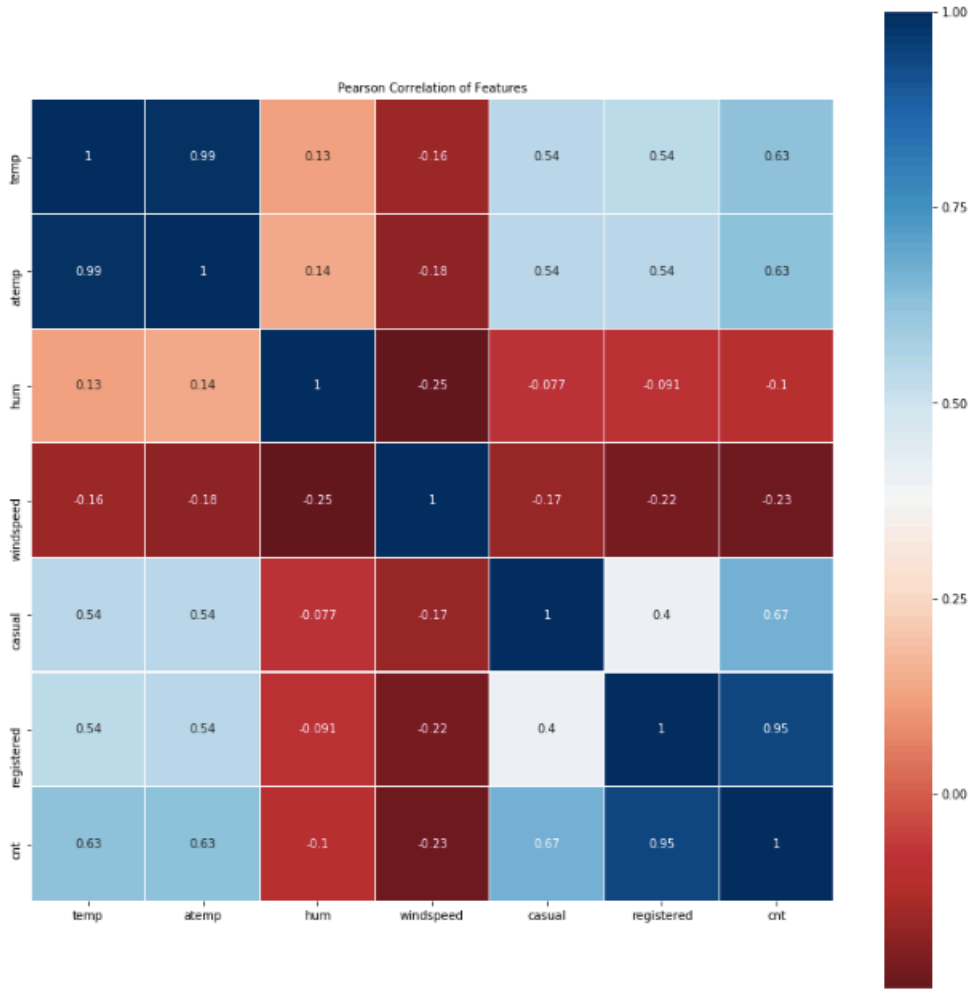
Correlation is a statistical technique that can show whether and how strongly pairs of variables are related.

The main result of a correlation is called the **correlation coefficient** . It ranges from -1.0 to +1.0. The closer r is to +1 or -1, the more closely the two variables are related.If r is close to 0, it means there is no relationship between the variables. If r is positive, it means that as one variable gets larger the other gets larger. If r is negative it means that as one gets larger, the other gets smaller (usally called an inverse correlation).

While correlation coefficients are normally reported as r = (a value between -1 and +1), squaring them makes then easier to understand. The square of the coefficient (or r square) is equal to the percent of the variation in one variable that is related to the variation in the other. After squaring r,

ignore the decimal point. An r of .5 means 25% of the variation is related (.5 squared =.25). An r value of .7 means 49% of the variance is related (.7 squared = .49).

```
51]: <matplotlib.axes._subplots.AxesSubplot at 0x1a161be0f0>
```



Removing highly correlated features (having high correlation coefficient [>0.95])
"atemp" is variable is not taken into since "atemp" and "temp" has got strong correlation with each other.

During model building any one of the variable has to be dropped since they will exhibit multicollinearity in the data.

2.2.5 Feature Engineering:

Feature engineering is the process of using **domain knowledge** of the data to create features that make machine learning algorithms work. If feature engineering is done correctly, it increases the predictive power of machine learning algorithms by creating features from raw data that help facilitate the machine learning process. Feature Engineering is an art.
Steps which are involved while solving any problem in machine learning are as follows:

- Gathering data.
- Cleaning data.
- Feature engineering.
- Defining model.
- Training, testing model and predicting the output.

Feature engineering is the most important art in machine learning which creates the huge difference between a good model and a bad model. Let's see what feature engineering covers.
Suppose, we are given a data "flight date time vs status". Then, given the date-time data, we have to predict the status of the flight.

	Date_Time_Combined	Status
0	2018-02-14 20:40	Delayed
1	2018-02-15 10:30	On Time
2	2018-02-14 07:40	On Time
3	2018-02-15 18:10	Delayed
4	2018-02-14 10:20	On Time

Flight Date Time Data

As the status of the flight depends on the hour of the day, not on the date-time. We will create the new feature "Hour_Of_Day". Using the "Hour_Of_Day" feature, the machine will learn better as this feature is directly related to the status of the flight.

	Hour_Of_Day	Status
0	20	Delayed
1	10	On Time
2	7	On Time
3	18	Delayed
4	10	On Time

Flight Hour Of Day Data

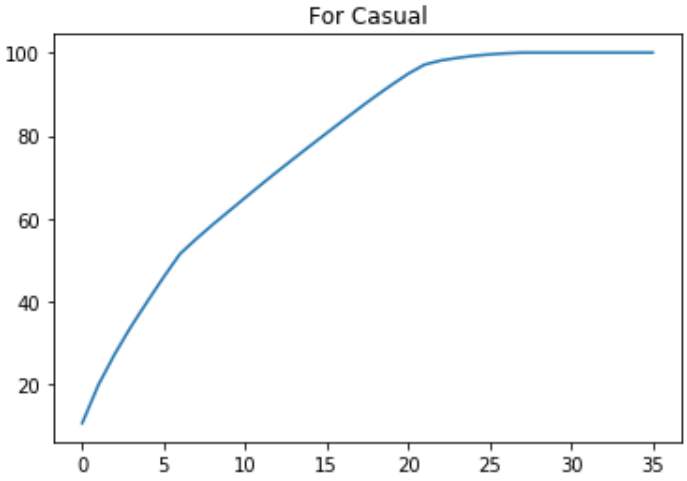
Here, creating the new feature "Hour_Of_Day" is the feature engineering.

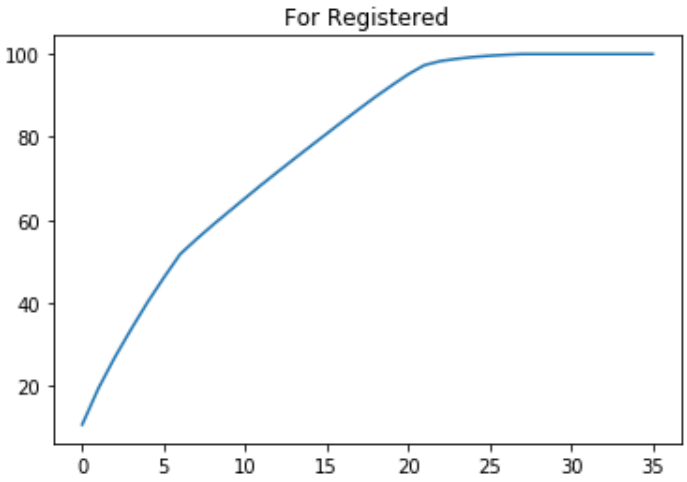
Using :

a) Boruta package in R
b) SeleckKBest package in Python

Principal component analysis :

It is a fast and flexible unsupervised method for dimensionality reduction in data, Using PCA for dimensionality reduction involves zeroing out one or more of the smallest principal components, resulting in a lower-dimensional projection of the data that preserves the maximal data variance.
A vital part of using PCA in practice is the ability to estimate how many components are needed to describe the data. This can be determined by looking at the cumulative *explained variance ratio* as a function of the number of components:





From the above plot 25 variables clearly explains the ~100% of variance in the data

Boruta is a feature selection algorithm. Precisely, it works as a wrapper algorithm around Random Forest. This package derive its name from a demon in Slavic mythology who dwelled in pine forests.
We know that feature selection is a crucial step in predictive modeling. This technique achieves supreme importance when a data set comprised of several variables is given for model building.
Boruta can be your algorithm of choice to deal with such data sets. Particularly when one is interested in understanding the mechanisms related to the variable of interest, rather than just building a black box predictive model with good prediction accuracy.

How does it work?
Below is the step wise working of boruta algorithm:

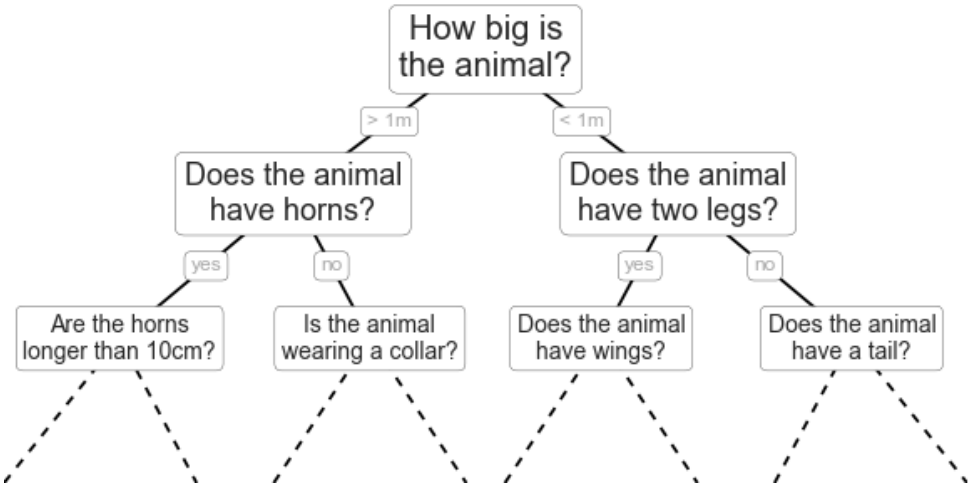
1. Firstly, it adds randomness to the given data set by creating shuffled copies of all features (which are called shadow features).
2. Then, it trains a random forest classifier on the extended data set and applies a feature importance measure (the default is Mean Decrease Accuracy) to evaluate the importance of each feature where higher means more important.
3. At every iteration, it checks whether a real feature has a higher importance than the best of its shadow features (i.e. whether the feature has a higher Z score than the maximum Z score of its shadow features) and constantly removes features which are deemed highly unimportant.
4. Finally, the algorithm stops either when all features gets confirmed or rejected or it reaches a specified limit of random forest runs.

Sampling methods - K-fold repeated CV

2.4. Model Selection

(i) Decision Tree , Random Forest :

Random forests are an example of an *ensemble learner* built on decision trees. For this reason we'll start by discussing decision trees themselves.
Decision trees are extremely intuitive ways to classify or label objects: you simply ask a series of questions designed to zero-in on the classification. For example, if you wanted to build a decision tree to classify an animal you come across while on a hike, you might construct the one shown here:



The binary splitting makes this extremely efficient: in a well-constructed tree, each question will cut the number of options by approximately half, very quickly narrowing the options even among a large number of classes. The trick, of course, comes in deciding which questions to ask at each step. In machine learning implementations of decision trees, the questions generally take the form of axis-aligned splits in the data: that is, each node in the tree splits the data into two groups using a cutoff value within one of the features.
A simple decision tree built on this data will iteratively split the data along one or the other axis according to some quantitative criterion, and at each level assign the label of the new region according to a majority vote of points within it.
This notion—that multiple overfitting estimators can be combined to reduce the effect of this overfitting—is what underlies an ensemble method called *bagging*. Bagging makes use of an ensemble (a grab bag, perhaps) of parallel estimators, each of which over-fits the data, and averages the results to find a better classification. An ensemble of randomized decision trees is known as a *random forest*.

(ii) Regression

Regression analysis is a form of predictive modelling technique which investigates the relationship between a dependent (target) and independent variable (s) (predictor). This technique is used for forecasting, time series modelling and finding the causal effect relationship between the variables. For example, relationship between rash driving and number of road accidents by a driver is best studied through regression.

Regression analysis is an important tool for modelling and analyzing data. Here, we fit a curve / line to the data points, in such a manner that the differences between the distances of data points from the curve or line is minimized. I'll explain this in more details in coming sections.

There are multiple benefits of using regression analysis. They are as follows:

1. It indicates the significant relationships between dependent variable and independent variable.

2. It indicates the strength of impact of multiple independent variables on a dependent variable.

There are various kinds of regression techniques available to make predictions. These techniques are mostly driven by three metrics (number of independent variables, type of dependent variables and shape of regression line).

1. Linear Regression

It is one of the most widely known modeling technique. Linear regression is usually among the first few topics which people pick while learning predictive modeling. In this technique, the dependent variable is continuous, independent variable(s) can be continuous or discrete, and nature of regression line is linear.

Linear Regression establishes a relationship between dependent variable (Y) and one or more independent variables (X) using a best fit straight line (also known as regression line).

It is represented by an equation Y=a+b*X + e, where a is intercept, b is slope of the line and e is error term. This equation can be used to predict the value of target variable based on given predictor variable(s).

2. Ridge Regression

Ridge Regression is a technique used when the data suffers from multicollinearity (independent variables are highly correlated). In multicollinearity, even though the least squares estimates (OLS) are unbiased, their variances are large which deviates the observed value far from the true value. By adding a degree of bias to the regression estimates, ridge regression reduces the standard errors.

Above, we saw the equation for linear regression. Remember? It can be represented as:

$$y=a+ b*x$$

This equation also has an error term. The complete equation becomes:

y=a+b*x+e (error term), [error term is the value needed to correct for a prediction error between the observed and predicted value]

=> y=a+y= a+ b1x1+ b2x2+....+e, for multiple independent variables.

In a linear equation, prediction errors can be decomposed into two sub components. First is due to the biased and second is due to the variance. Prediction error can occur due to any one of these two or both components. Here, we'll discuss about the error caused due to variance.

Ridge regression solves the multicollinearity problem through shrinkage parameter λ (lambda). Look at the equation below.

$$= \underset{\beta \in \mathbf{R}^p}{\operatorname{argmin}} \underbrace{\|y - X\beta\|_2^2}_{\text{Loss}} + \lambda \underbrace{\|\beta\|_2^2}_{\text{Penalty}}$$

3. Lasso Regression

Similar to Ridge Regression, Lasso (Least Absolute Shrinkage and Selection Operator) also penalizes the absolute size of the regression coefficients. In addition, it is capable of reducing the variability and improving the accuracy of linear regression models. Look at the equation below:

$$= \underset{\beta \in \mathbf{R}^p}{\operatorname{argmin}} \underbrace{\|y - X\beta\|_2^2}_{\text{Loss}} + \lambda \underbrace{\|\beta\|_1}_{\text{Penalty}}$$

Lasso regression differs from ridge regression in a way that it uses absolute values in the penalty function, instead of squares. This leads to penalizing (or equivalently constraining the sum of the absolute values of the estimates) values which causes some of the parameter estimates to turn out exactly zero. Larger the penalty applied, further the estimates get shrunk towards absolute zero. This results to variable selection out of given n variables.

4.L**east-angle regression (LARS)**

It is an algorithm for fitting linear regression models to high-dimensional data, developed by Bradley Efron, Trevor Hastie, Iain Johnstone and Robert Tibshirani.[1]
Suppose we expect a response variable to be determined by a linear combination of a subset of potential covariates. Then the LARS algorithm provides a means of producing an estimate of which variables to include, as well as their coefficients.
Instead of giving a vector result, the LARS solution consists of a curve denoting the solution for each value of the L1 norm of the parameter vector. The algorithm is similar to forward stepwise regression, but instead of including variables at each step, the estimated parameters are increased in a direction equiangular to each one's correlations with the residual.

2.5 Model Evaluation

Python :

Decision Tree :

```
Casual :
RMSE :  898.8995580311359
Model Score : -0.5486222028315206
Registered :
RMSE :  1137.3832552784286
Model Score : 0.29777669111901467
Count :
RMSE :  1437.8240285165607
```

Random Forest :

```
Casual :
RMSE :  475.4592320766723
Model Score : 0.566738881140593
Registered :
RMSE :  1055.4875541566523
Model Score : 0.39526122786718 02
Count :
RMSE :  1117.892245351008
```

Linear Regression :

```
Casual :
RMSE :  424.9209000388145
Model Score : 0.6539496178882476
Registered :
RMSE :  903.1730955328001
Model Score : 0.5572041270279229
Count :
RMSE :  1051.715014306732
```

In [61]:

Ridge Regression :

```
Casual :
RMSE :  418.29190291508917
Model Score : 0.6646625455000048
Registered :
RMSE :  910.5746161979094
Model Score : 0.5499169484223905
```

Count :
RMSE : 1058.751163972777

Lasso Regression :

Casual :
RMSE : 418.3696542090059
Model Score : 0.6645378701524947
Registered :
RMSE : 901.2962307577742
Model Score : 0.5590425442001858
Count :
RMSE : 1049.5571529310841

Lars Regression

Casual :
RMSE : 438.0828673911513
Model Score : 0.632179707138155
Registered :
RMSE : 988.5553438873961
Model Score : 0.4695266669733078
Count :
RMSE : 1153.1921703661671

PCA_Decision Tree:

Casual :
RMSE : 42.08063299748933
Model Score : 0.9966061888211043
Registered :
RMSE : 291.1492218566883
Model Score : 0.9539856970046925
Count :
RMSE : 294.74245693209417

PCA_Random Forest :

Casual :
RMSE : 27.25476728642907
Model Score : 0.9985732321694418
Registered :
RMSE : 353.3949622583682
Model Score : 0.931932063554032
Count :

RMSE : 354.5823505413759

PCA_Linear Regression :

Casual :
RMSE : 3.554218989297293e-06
Model Score : 1.0
Registered :
RMSE : 3.774453326802237e-06
Model Score : 1.0
Count :
RMSE : 0.11790792879159628

PCA_Ridge Regression :

Casual :
RMSE : 8.927623672309348e-06
Model Score : 0.9999999999999999
Registered :
RMSE : 9.051972215133674e-06
Model Score : 1.0
Count :
RMSE : 0.11790779467315071

PCA_Lasso Regression :

Casual :
RMSE : 0.0024465882591219006
Model Score : 0.9999999999885029
Registered :
RMSE : 0.0017872732012681805
Model Score : 0.999999999998259
Count :
RMSE : 0.1174453713641168

PCA_Lars Regression :

Casual :
RMSE : 1.3417316582397942e-05
Model Score : 0.9999999999999997
Registered :
RMSE : 1.8645142146288997e-05
Model Score : 0.9999999999999998
Count :

RMSE : 0.11791011095004408

R :

Linear Regression :
"Casual"
rmse
369.9694
"Registered"
rmse
770.7711
"Count"
rmse
926.1262
Random Forest :
[1] "Casual"
rmse
129.4328
[1] "Registered"
rmse
317.1956
[1] "Count"
rmse
366.3585

Decision Tree :
[1] "Casual"
rmse
414.8738
rmse
1178.428
[1] "Count"
rmse
1393.001

Ridge :
[1] "Casual"
rmse
369.9537
[1] "Registered"
rmse
770.7651
[1] "Count"
rmse
926.1067
Lasso :
[1] "Casual"
rmse
372.2932
[1] "Registered"
rmse
776.5481
[1] "Count"
rmse

932.3251
Lars :
[1] "Casual"
rmse
369.9694
[1] "Registered"
rmse
770.7711
[1] "Count"
rmse
926.1262

PCA _ Linear Regression :
[1] "Casual"
rmse
4.266233e-06
[1] "Registered"
rmse
3.627236e-06
[1] "Count"
rmse
7.352193e-06

PCA_Decision Tree
[1] "Casual"
rmse
30.79164
[1] "Registered"
rmse
368.1081
[1] "Count"
rmse
368.4049

PCA_Random Forest
[1] "Casual"
rmse
32.90786
[1] "Registered"
rmse
372.5362
[1] "Count"
rmse
373.0246

PCA_Ridge Regression

[1] "Casual"
rmse
4.266233e-06
[1] "Registered"
rmse

3.627234e-06
[1] "Count"
rmse
7.352191e-06

PCA_Lasso Regression

[1] "Casual"
rmse
86.03056
[1] "Registered"
rmse
243.1225
[1] "Count"
rmse
295.7792

PCA_Lars Regression

```
[1] "Casual"
        rmse
4.266233e-06
[1] "Registered"
        rmse
3.627233e-06
[1] "Count"
        rmse
7.352191e-06
```

2.5.1 Comparing results of the models

The results obtained , shown as below, with PCA gives significant improvement over without applying PCA.

'cnt'	SelectKBest	PCA
Model	**RMSE**	**RMSE**
Decision Tree	**1437**	**286**
RF	**1117**	**354**
Linear R	**1051**	**0.11**
Ridge R	**1056**	**0.11**
Lasso R	**1047**	**0.11**
Lars R	**1151**	**0.11**

'cnt'	Boruta	PCA
Model	**RMSE**	**RMSE**
Decision Tree	1393.001	368.4049
RF	366.3585	**354**
Linear R	926.1262	7.352193E-06
Ridge R	926.1067	7.352191E-06
Lasso R	932.3251	295.7792
Lars R	926.1262	**7.352191E-06**

Python Code

Expl Data Analysis :

```
# coding: utf-8

# In[1]:


# Load Libraries
import pandas as pd
import numpy as np
get_ipython().run_line_magic('matplotlib', 'inline')
import matplotlib.pyplot as plt
import seaborn as sns


# In[2]:


data = pd.read_csv('day.csv')
data_ord = data.copy()


# In[3]:


#plt.style.use('seaborn-white')
for i in data.columns :
    plt.figure()
    plt.hist(data[i], alpha = 1, edgecolor = 'black')
    plt.ylabel('Count of Observations')
    plt.xlabel(i)


# In[4]:


d = data.pivot_table(index='weekday',aggfunc={'cnt':sum})
d = d.reset_index()
plt.plot(d['weekday'],d['cnt'])
plt.xlabel('weekday')
plt.ylabel('cnt')

plt.figure()
d = data.pivot_table(index='season',aggfunc={'cnt':sum})
d = d.reset_index()
plt.plot(d['season'],d['cnt'])
```

```
plt.xlabel('season')
plt.ylabel('cnt')


plt.figure()
d = data.pivot_table(index='dteday',aggfunc={'cnt':sum})
d = d.reset_index()
plt.plot(d['dteday'],d['cnt'])
plt.xlabel('dteday')
plt.ylabel('cnt')


# In[5]:


data.pivot_table(index='mnth', columns = 'weekday', aggfunc={'cnt':sum})


# In[6]:


data.columns


# In[7]:


# Scatter Plot
import matplotlib.pyplot as plt
get_ipython().run_line_magic('matplotlib', 'inline')
plt.style.use('seaborn-white')

plt.figure()
plt.plot(data['temp'],data['weathersit'],'o')
plt.xlabel('temp')
plt.ylabel('weathersit')

plt.figure()
plt.plot(data['casual'],data['registered'],'o')
plt.xlabel('casual')
plt.ylabel('registered')

plt.figure()
plt.plot(data['hum'],data['temp'],'o')
plt.xlabel('hum')
plt.ylabel('temp')

plt.figure()
plt.plot(data['hum'],data['windspeed'],'o')
plt.xlabel('hum')
plt.ylabel('windspeed')
```

```
plt.figure()
plt.plot(data['weekday'],data['hum'],'o')
plt.xlabel('weekday')
plt.ylabel('hum')

plt.figure()
plt.plot(data['temp'],data['atemp'],'o')
plt.xlabel('temp')
plt.ylabel('atemp')


# In[8]:


data['season'].unique()


# In[9]:


# Scatter Pair Plots
# Create columns of seasons for pair plotting colors
# Seasons (summer (1), autumn (2), winter (3), spring (4))
seasons = data['season']
for i in range(len(seasons)):
    if seasons[i] == 1:
        seasons[i] = 'summer'
    if seasons[i] == 2:
        seasons[i] = 'autumn'
    if seasons[i] == 3:
        seasons[i] = 'winter'
    if seasons[i] == 4:
        seasons[i] = 'spring'

# Will only use six variables for plotting pairs
reduced_features = data[['cnt', 'temp', 'hum',
                         'casual','registered']].copy()
reduced_features['season'] = seasons
# Use seaborn for pair plots
import seaborn as sns
sns.set(style="ticks", color_codes=True);
# Create a custom color palete
palette = sns.xkcd_palette(['dark blue', 'dark green', 'gold', 'orange'])
# Make the pair plot with a some aesthetic changes
sns.pairplot(reduced_features, hue = 'season', diag_kind = 'kde', palette= palette,
plot_kws=dict(alpha = 0.7),
                 diag_kws=dict(shade=True))


# In[10]:
```

```
num_var = ['hum', 'windspeed', 'casual']
for i in num_var:
    plt.figure()
    plt.boxplot(data[i])
    plt.xlabel(i)


# In[11]:


data.columns


# In[12]:


# data summary
data.info()


# In[13]:


plt.scatter(data['cnt'],data['casual'])
plt.scatter(data['cnt'],data['registered'])
plt.scatter(data['cnt'],data['casual']+data['registered'])
plt.legend()


# In[14]:


data.head()


# In[15]:


d = data.pivot_table(index='dteday',aggfunc={'cnt':sum})
d = d.reset_index()
plt.plot(d['dteday'],d['cnt'])
d = data.pivot_table(index='dteday',aggfunc={'casual':sum})
d = d.reset_index()
plt.plot(d['dteday'],d['casual'])
d = data.pivot_table(index='dteday',aggfunc={'registered':sum})
d = d.reset_index()
plt.plot(d['dteday'],d['registered'])
plt.legend()
```

Code :

```
# coding: utf-8

# In[44]:


# Load Libraries
import pandas as pd
import numpy as np
get_ipython().run_line_magic('matplotlib', 'inline')
import matplotlib.pyplot as plt


# In[45]:


# Load Data file
data = pd.read_csv('day.csv')
data_org = data.copy()


# In[46]:


# Numerical and Cateorical Variables
num_var = ['temp', 'atemp', 'hum', 'windspeed', 'casual', 'registered', 'cnt']
cat_var = ['instant', 'season', 'yr', 'mnth', 'holiday', 'weekday', 'workingday', 'weathersit']


# In[47]:


'''# assigining categories/levels to categorical variable
for i in data.columns :
    if data[i].dtypes == 'object' :
        data[i] = pd.Categorical(data[i])
        data[i] = data[i].cat.codes
data.info()'''


# In[48]:


#data.head()


# In[49]:


# Checking for Missing Values
for i in data.columns :
    print(i, data[i].isnull().sum())


# In[50]:
```

```
#Corelation Analysis
data_corr = data.loc[:,num_var]
f,ax = plt.subplots(figsize = (7,5))
corr = data_corr.corr()

import seaborn as sns
sns.heatmap(corr,mask = np.zeros_like(corr, dtype = np.bool),cmap =
sns.diverging_palette(220,10,as_cmap=True),
            square = True, ax =ax)
cmap = sns.diverging_palette(5, 250, as_cmap=True)
#cmap = plt.cm.RdBu

def magnify():
    return [dict(selector="th",
                 props=[("font-size", "7pt")]),
            dict(selector="td",
                 props=[('padding', "0em 0em")]),
            dict(selector="th:hover",
                 props=[("font-size", "12pt")]),
            dict(selector="tr:hover td:hover",
                 props=[('max-width', '200px'),
                        ('font-size', '12pt')])
]

corr.style.background_gradient(cmap, axis=1)    .set_properties(**{'max-width': '80px', 'font-size':
'10pt'})    .set_caption("Hover to magify")    .set_precision(2)    .set_table_styles(magnify())


# In[51]:


#correlation plot
colormap = plt.cm.RdBu
plt.figure(figsize=(15,15))
plt.title('Pearson Correlation of Features', y=1.0, size=10)
sns.heatmap(data_corr.corr(),linewidths=0.2,vmax=1.0, square=True, cmap=colormap,
linecolor='white', annot=True)


# In[52]:


# Outlier Analysis, Capping and Flooring
for i in ['hum','windspeed','casual'] :
    q99, q01 = np.percentile(data[i],[99,1])
    data.loc[data[i] > q99,i] = q99
    data.loc[data[i] < q01,i] = q01
# so we have to update total count variables as it is sum of casual and registered users
data['cnt'] = data['casual']+data['registered']


# In[53]:


# Drop 'atemp'
data = data.drop(['atemp'],axis=1)
data = data.drop(['dteday'],axis=1)
```

In[54]:

```
# Changing data type of categorical variable to object from integer
cat_var = ['season', 'yr', 'mnth', 'holiday', 'weekday', 'workingday', 'weathersit']
for i in cat_var :
    data[i] = data[i].astype('category')
data.info()
```

In[55]:

```
# check category  variables which do not require hot encoding
cat_var = ['season', 'yr', 'mnth', 'holiday', 'weekday', 'workingday','weathersit']
for i in cat_var :
    print(i, data[i].unique())
```

In[56]:

```
#red_cat_var = ['season','mnth','weekday','weathersit']
```

In[57]:

```
num_var = ['temp','hum', 'windspeed','casual','registered']
df1 = data.applymap(int)
for i in cat_var:
    df1[i].astype('str')
temp = pd.DataFrame(df1['cnt'])
temp = temp.join(data[num_var])
for i in cat_var:
    d = pd.get_dummies(df1[i],prefix = i)
    temp = temp.join(d)
data_hotencod = temp
data_hotencod.shape,data_hotencod.columns
```

In[58]:

```
#data_hotencod.info()
```

In[59]:

```
new_data = data_hotencod.drop(['cnt','registered','casual'],axis=1)
cas_data = pd.DataFrame(data_hotencod['casual']).join(new_data)
reg_data = pd.DataFrame(data_hotencod['registered']).join(new_data)
```

In[60]:

```
#selecting topfeatures for 'casual'
from sklearn.feature_selection import SelectKBest, f_regression
```

```
cas_new = SelectKBest(f_regression,k=30).fit_transform(cas_data.iloc[:,1:],cas_data.iloc[:,0])

##selecting top features for 'registered'
from sklearn.model_selection import train_test_split
from sklearn.feature_selection import SelectKBest, f_regression
reg_new = SelectKBest(f_regression,k=30).fit_transform(reg_data.iloc[:,1:],reg_data.iloc[:,0])


# In[61]:


# Selecting train and test sample
# observations for 'casual' and 'registered' should be in same order, so to compare thr
# summation to 'cnt' of data

# train : test, ratio(80:20)

## 'casual' customers
X_train_cas=cas_new[:485:1]
X_test_cas=cas_new[485::1]
y_train_cas=cas_data.iloc[:485,0]
y_test_cas=cas_data.iloc[485:,0]


## 'registered' customers

X_train_reg=reg_new[:485:1]
X_test_reg=reg_new[485::1]
y_train_reg=reg_data.iloc[:485,0]
y_test_reg=reg_data.iloc[485:,0]

y_test_count=data_hotencod.iloc[485:,0]


# In[62]:


def model_pred(model) :
    cas_model = model.fit(X_train_cas,y_train_cas)
    cas_pred = model.predict(X_test_cas)
    print('Casual :')
    print('RMSE : ',np.sqrt(mean_squared_error(y_test_cas,cas_pred)))
    print('Model Score :', model.score(X_test_cas, y_test_cas))

    reg_model = model.fit(X_train_reg,y_train_reg)
    reg_pred = model.predict(X_test_reg)
    print('Registered :')
    print('RMSE : ',np.sqrt(mean_squared_error(y_test_reg,reg_pred)))
    print('Model Score :', model.score(X_test_reg, y_test_reg))

    cnt_pred = cas_pred + reg_pred

    print('Count :')
    print('RMSE : ',np.sqrt(mean_squared_error(y_test_count,cnt_pred)))
    return


# In[63]:
```

```
#Decision Tree
from sklearn.tree import DecisionTreeRegressor
from sklearn.metrics import mean_squared_error
model = DecisionTreeRegressor()
model_pred(model)


# In[64]:


#Random forest
from sklearn.ensemble import RandomForestRegressor
model = RandomForestRegressor()
model_pred(model)


# In[65]:


#Linear Regression
from sklearn.linear_model import LinearRegression
model = LinearRegression()
model_pred(model)


# In[66]:


#Ridge Regression
from sklearn.linear_model import Ridge
model=Ridge()
model_pred(model)


# In[67]:


#Lasso regression
from sklearn.linear_model import Lasso
model=Lasso()
model_pred(model)


# In[68]:


#Lars Regression
from sklearn.linear_model import Lars
model=Lars()
model_pred(model)


# In[69]:


cas_data.shape


# In[79]:
```

```
# PCA feature engineering/dimensionality reduction
from sklearn.decomposition import PCA
from sklearn.preprocessing import scale
y_test_count=data_hotencod.iloc[485:,0]

pca_cas = PCA(n_components=36)
pca_reg=PCA(n_components=36)

pca_cas.fit(scale(cas_data.values))
pca_reg.fit(scale(reg_data.values))

#The amount of variance that each PC explained
var_cas= pca_cas.explained_variance_ratio_
var_reg= pca_reg.explained_variance_ratio_

#Cumulative Variance
var1_cas=np.cumsum(np.round(pca_cas.explained_variance_ratio_, decimals=4)*100)
var1_reg=np.cumsum(np.round(pca_reg.explained_variance_ratio_, decimals=4)*100)


#graph of the variance


plt.plot(var1_cas)
plt.title('For Casual')
plt.figure()
plt.plot(var1_reg)
plt.title('For Registered')


# In[71]:


# 25 componentsaround 99% variance in the data set.


#Looking at above 25 variables to be opted
pca_cas = PCA(n_components=25)
pca_reg = PCA(n_components=25)


#now fitting the selected components to the data
pca_cas.fit(cas_data.values)
pca_cas.fit(reg_data.values)


#PCA selected features
X_cas=pca_cas.fit_transform(cas_data.values)
X_reg=pca_reg.fit_transform(reg_data.values)

#splitting train and test data

X_train_cas=X_cas[:485:1]
X_test_cas=X_cas[485::1]
y_train_cas=cas_data.iloc[:485,0]
y_test_cas=cas_data.iloc[485:,0]

X_train_reg=X_reg[:485:1]
X_test_reg=X_reg[485::1]
```

```
y_train_reg=reg_data.iloc[:485,0]
y_test_reg=reg_data.iloc[485:,0]


# In[72]:


#Decision Tree
from sklearn.tree import DecisionTreeRegressor
from sklearn.metrics import mean_squared_error
model = DecisionTreeRegressor()
model_pred(model)


# In[73]:


#Random forest
from sklearn.ensemble import RandomForestRegressor
model = RandomForestRegressor()
model_pred(model)


# In[74]:


#Linear Regression
from sklearn.linear_model import LinearRegression
model = LinearRegression()
model_pred(model)


# In[75]:


#Ridge Regression
from sklearn.linear_model import Ridge
model=Ridge()
model_pred(model)


# In[76]:


#Lasso regression
from sklearn.linear_model import Lasso
model=Lasso()
model_pred(model)


# In[77]:


#Lars Regression
from sklearn.linear_model import Lars
model=Lars()
model_pred(model)
```

R Code :

```{r}
# Load Libraries
rm(list = ls())
# install.packages("corrgram", lib="/Library/Frameworks/R.framework/Versions/3.4/Resources/
library")
# install.packages("sampling", lib="/Library/Frameworks/R.framework/Versions/3.4/Resources/
library")
# install.packages("DataCombine", lib="/Library/Frameworks/R.framework/Versions/3.4/
Resources/library")
# install.packages("caret", lib="/Library/Frameworks/R.framework/Versions/3.4/Resources/
library")
#install.packages(c("forcats", "DataExplorer",
"ggthemes","grid","gridExtra","factoextra","FactoMineR"))
#install.packages("DataExplorer", lib="/Library/Frameworks/R.framework/Versions/3.4/Resources/
library")
#install.packages("psych")
library(psych)
library(ggplot2)
library(corrgram)
library(sampling)
library(corrgram)
library(class)
library(e1071)
library(caret)
library(DataCombine)
library(caret)
library(randomForest)
library(inTrees)
library(C50)
library(dplyr)
library(forcats)
library(plyr)
library(DataExplorer)
library(ggthemes)
library(grid)
library(gridExtra)
library(factoextra)
library(FactoMineR)
```


```{r}
# Import Data
data = read.csv('day.csv', header = TRUE)
data = data.frame(data)
data_org = data
# list of Columns
col = colnames(train)
# list of numeric and categorical columns
num_var = c('temp', 'atemp', 'hum', 'windspeed', 'casual', 'casistered', 'cnt')
cat_var = c('instant', 'season', 'yr', 'mnth', 'holiday', 'weekday', 'workingday', 'weathersit')
```


```{r}
str(data) # Work.load.Average.day is 'Factor Variable'
summary(data)
```

```
```{r}

# Convert catefory variables from numeric to factor type
for (i in cat_var) {
    data[,i]=as.factor((data[,i]))
}

numeric_index = sapply(data,is.numeric)
numeric_data = data[,numeric_index]
cnames_n = colnames(numeric_data)

factor_index = sapply(data, is.factor)
factor_data = data[,factor_index]
cnames_f = colnames(factor_data)
```


```{r}
multi.hist(data[,numeric_index], main = NA, dcol = c("blue", "red"), dlty = c("solid", "solid"), bcol =
"linen")
```

```{r}
pairs.panels(data)
```

```{r}
# Missing Value Analysis
# Create datframe with missing percentage
missing_val = data.frame(apply(data,2,function(x){sum(is.na(x))}))
# Convert row names into column
missing_val$Columns = row.names(missing_val)
row.names(missing_val) = NULL
# Rename the variable name
names(missing_val)[1] = 'missing_percentage'
#Calculate percentage
missing_val$missing_percentage = (missing_val$missing_percentage/nrow(missing_val))*100
# Arrange in descending order
missing_val = missing_val[order(-missing_val$missing_percentage),]
#Rearranging the columns
missing_val = missing_val[,c(2,1)]
#View(missing_val)
```

```{r}
boxplot(data)
```


```{r}
for (i in 1:length(cnames_n)){
 assign(paste0("plot",i), ggplot(aes_string(y = (cnames_n[i]), x = "cnt"), data =subset(data))+
 stat_boxplot(geom = "errorbar", width = 0.5) +
 geom_boxplot(outlier.colour="red", fill = "blue" ,outlier.shape=18,
 outlier.size=1, notch=FALSE) +
 theme(legend.position="bottom")+
 labs(y=cnames_n[i],x="cnt")+
 ggtitle(paste("Box plot of responded for",cnames_n[i])))
}
```
```

```
#boxplot of outliers
gridExtra::grid.arrange(plot1,plot2,plot3,ncol=3)
gridExtra::grid.arrange(plot4,plot5,plot6,ncol=3)
gridExtra::grid.arrange(plot7,ncol=3)
```

```{r}
# Capping and Flooring Outliers
for (i in cnames_n) {
  percentile = quantile(data[i], c(0.01, 0.99),na.rm = TRUE)
  data[i][data[i]<percentile[1]]=percentile[1]
  data[i][data[i]>percentile[2]]=percentile[2]
}

# so we have to update total count variables as it is sum of casual and registered users
data['cnt'] = data['casual']+data['registered']
```

```{r}
corrgram(data[-1], order = F, upper.panel = panel.pie, text.panel = panel.txt, main =
'CorrelationPlot')
symnum(cor(numeric_data))
high_corr =  findCorrelation(cor(numeric_data), cutoff=0.99)
```
```{r}
#removing highly correlated column 'atemp' and 'instant' & 'dteday'
data = subset(data, select = -c(instant,atemp,dteday))
```

```{r}
#install.packages("dummies", lib="/Library/Frameworks/R.framework/Versions/3.4/Resources/
library")
cat_var = c('season', 'yr', 'mnth', 'holiday', 'weekday', 'workingday','weathersit')
library(dummies)
for (i in cat_var){
  temp = data.frame(dummy(data[,i]))
  data = cbind(data,temp)
  data[,i] = NULL
}
```

```{r}
new_data = subset(data, select = -c(cnt,registered,casual))
cas_data = cbind(data[,'casual'],new_data)
reg_data = cbind(data[,'registered'],new_data)

colnames(cas_data) <- c("casual", colnames(new_data))
colnames(reg_data) <- c("registered", colnames(new_data))
```



```{r}
#install.packages('DAAG')
library(DAAG)
#feature selection using boruta package
install.packages("Boruta", lib="/Library/Frameworks/R.framework/Versions/3.4/Resources/
library")
library(Boruta)
cas_boruta.train=Boruta(casual~., data = cas_data, doTrace = 2)
cas_selected_features=getSelectedAttributes(cas_boruta.train, withTentative = F)
```

```
set.seed(123)
cas_formula=as.formula(paste("casual~",paste(cas_selected_features,collapse = "+")))


reg_boruta.train=Boruta(registered~., data = reg_data, doTrace = 2)
reg_selected_features=getSelectedAttributes(reg_boruta.train, withTentative = F)
reg_formula=as.formula(paste("registered~",paste(reg_selected_features,collapse = "+")))
```

```{r}
# train and test sample
X_train_reg = reg_data[1:485,2:36]
X_test_reg=reg_data[485:731,2:36]
y_train_reg=reg_data[1:485,1]
y_test_reg=reg_data[485:731,1]

X_train_cas = cas_data[1:485,2:36]
X_test_cas=cas_data[485:731,2:36]
y_train_cas=cas_data[1:485,1]
y_test_cas=cas_data[485:731,1]

y_test_count= data[485:731,6]
```


```{r}
# Cross-Validation
train_control <- trainControl(method = "repeatedcv",number = 10)
#options(warn=-1)


# model prediction function
model_pred  <- function(method_model) {
  cas_model <- train(cas_formula,data = cas_data,metric="RMSE",
method=method_model,trControl=train_control)
  cas_pred = predict(cas_model,cas_data[485:731,])
  print('Casual')
  #print(cas_model)
  print(regr.eval(y_test_cas, cas_pred, stats = c('rmse')))

  reg_model <- train(reg_formula,data = reg_data,metric="RMSE",
method=method_model,trControl=train_control)
  reg_pred=predict(reg_model,reg_data[485:731,])
  print('Registered')
  #print(reg_model)
  print(regr.eval(y_test_reg, reg_pred, stats = c('rmse')))

  cnt_pred = cas_pred + reg_pred
  print('Count')
  print(regr.eval(y_test_count,cnt_pred, stats = c('rmse')))
}

```


```{r}
install.packages("DMwR", lib="/Library/Frameworks/R.framework/Versions/3.4/Resources/
library")
library(DMwR)
# Linear Model
model_pred('lm')
```

```

```{r}
#Random Forest
model_pred('rf')
```

```{r}
#Decision Tree
model_pred('rpart')
```
```{r}
# Ridge
model_pred('ridge')
```

```{r}
# Lasso
model_pred('lasso')
```
```{r}
# Lars
model_pred('lars')
```

```{r}
getSelectedAttributes(cas_boruta.train, withTentative = F)
boruta.df <- attStats(cas_boruta.train)
class(boruta.df)
print(boruta.df)
```

```{r}
# PCA
## Registered Customers

prin_comp <- prcomp(reg_data[1:485,]) #outputs the mean of variables
prin_comp$center
#outputs the standard deviation of variables prin_comp$scale
dim(prin_comp$x)
biplot(prin_comp, scale = 0)
#compute standard deviation of each principal component
std_dev = prin_comp$sdev
#compute variance
pr_var = std_dev^2
#proportion of variance explained
prop_varex =pr_var/sum(pr_var)
#scree plot
plot(prop_varex, xlab = "Principal Component",ylab = "Proportion of Variance Explained", type =
"b")
#cumulative scree plot
plot(cumsum(prop_varex), xlab = "Principal Component",ylab = "Cumulative Proportion of
Variance Explained",type = "b")
#add a training set with principal components
reg_train_data = data.frame(registered = reg_data[1:485,'registered'], prin_comp$x)
#we are interested in first 25 PCAs as we have seen from the graph # and the target variable ,so in
total 41(including target variable)
reg_train_data = reg_train_data[,1:26]
#transform test into PCA
reg_test_data=predict(prin_comp, newdata = reg_data[486:731,])
```
```

```
reg_test_data= as.data.frame(reg_test_data)
#select the first 40 components
reg_test_data =  reg_test_data[,1:25]
```

```{r}
## Casual Customers

prin_comp <- prcomp(cas_data[1:485,]) #outputs the mean of variables
prin_comp$center
#outputs the standard deviation of variables prin_comp$scale
dim(prin_comp$x)
biplot(prin_comp, scale = 0)
#compute standard deviation of each principal component
std_dev = prin_comp$sdev
#compute variance
pr_var = std_dev^2
#proportion of variance explained
prop_varex =pr_var/sum(pr_var)
#scree plot
plot(prop_varex, xlab = "Principal Component",ylab = "Proportion of Variance Explained", type =
"b")
#cumulative scree plot
plot(cumsum(prop_varex), xlab = "Principal Component",ylab = "Cumulative Proportion of
Variance Explained",type = "b")
#add a training set with principal components
cas_train_data = data.frame(casual = cas_data[1:485,'casual'], prin_comp$x)
#we are interested in first 25 PCAs as we have seen from the graph # and the target variable ,so in
total 41(including target variable)
cas_train_data = cas_train_data[,1:26]
#transform test into PCA
cas_test_data=predict(prin_comp, newdata = cas_data[486:731,])
cas_test_data= as.data.frame(cas_test_data)
#select the first 40 components
cas_test_data= cas_test_data[,1:25]
```

```{r}
pca_model_pred  <- function(method_model) {
  cas_model <- train(casual~.,data = cas_train_data,metric="RMSE",
method=method_model,trControl=train_control)
  cas_pred = predict(cas_model,cas_test_data)
  print('Casual')
  #print(cas_model)
  print(regr.eval(cas_pred, cas_data[486:731,1], stats = c('rmse')))

  reg_model <- train(registered~.,data = reg_train_data,metric="RMSE",
method=method_model,trControl=train_control)
  reg_pred = predict(reg_model,reg_test_data)
  print('Registered')
  #print(reg_model)
  print(regr.eval(reg_pred, reg_data[486:731,1], stats = c('rmse')))

  cnt_pred = cas_pred + reg_pred
  print('Count')
  print(regr.eval(cnt_pred, data[486:731,6],stats = c('rmse')))
}
```

```{r}
# Linear Model
pca_model_pred('lm')
```

```{r}
#Random Forest
pca_model_pred('rf')
```

```{r}
#Decision Tree
pca_model_pred('rpart')
```

```{r}
# Ridge
pca_model_pred('ridge')
```

```{r}
# Lasso
pca_model_pred('lasso')
```

```{r}
# Lars
pca_model_pred('lars')
```