Project: Bike Demand Prediction

Sailesh Patra

08th January 2020

1.1 Table of Contents

Chapter 1

2. Introduction

2.1 Problem Statement

The objective of this project is to predict the demand of bike rental count based on the environmental and seasonal settings. Our data consists of different environmental parameters based on which we have to make a model which will predict a number for us, denoting the number of bikes will be demanded at a particular place depending upon the weather condition, day of week, month and different other parameters.

2.2 Data

Our task is to build a regressor model that will predict the no. of bikes may get demanded at a place based the parameters given below:

Table 1.1 Bike Rental Data (Columns: 1-9)

instant	dteday	season	yr	mnth	holiday	weekday	workingday	weathersit
1	01-01-2011	1	0	1	0	6	0	2
2	02-01-2011	1	0	1	0	0	0	2
3	03-01-2011	1	0	1	0	1	1	1

Table 1.2 Bike Rental Data (Columns: 10-16)

temp	atemp	hum	windspeed	casual	registered	cnt
0.344167	0.363625	0.805833	0.160446	331	654	985
0.363478	0.353739	0.696087	0.248539	131	670	801
0.196364	0.189405	0.437273	0.248309	120	1229	1349

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 from Holiday schedule)
- **weekday**: day of the week
- **workingday**: if day is neither weekend nor holiday is 1, otherwise is 0.

+ **weathersit**:

- 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
- **registered**: count of registered users
- **cnt**: count of total rental bikes including both casual and registered

Out of these 16 variables mentioned above the last 3 (i.e., casual, registered and cnt) variables are dependent / target variable that we are going to predict. The list of predictors is:

Table 1.3 Predictor Variables in our dataset

instant	dteday	season	yr	mnth	hr	holiday
weekday	workingday	weathersit	temp	atemp	hum	windspeed

Chapter 2

3. Methodology

3.1 Pre-Processing

Pre-processing data is a crucial step and also the initial step in any data science project. Before developing a model and make it ready to understand and learn the hidden patterns from our data, it is necessary to make it noise-free. By cleaning the data, we restrict our model from learning the unclear patterns and cleaning data also helps us in reducing complexity of the data under analysis. Real-world data is often incomplete, inconsistent, and lacking certain behaviours or trends, and is likely to contain many errors which may obstruct in getting successful predictions. Pre-processing techniques transforms these raw data into an understandable format for our data.

Pre-processing is not limited to clean the data only, but also involves in deriving statistical information from the data, visualizing the data to know how they are distributed through graphs and plots. All these steps together well known as **Exploratory Data Analysis** in data science terminology.

3.1.1 Data Exploration

As discussed in section 2.1, we will first analyse the data to know its statistical distributions and then our next step will be univariate and bivariate analysis to know about each variable in detail through different visualization techniques. So before starting any of these we need to transform the variables to an appropriate data type.

Columns / Variables of our dataset:

```
Index(['instant', 'dteday', 'season', 'yr', 'mnth', 'holiday', 'weekday',
       'workingday', 'weathersit', 'temp', 'atemp', 'hum', 'windspeed',
       'casual', 'registered', 'cnt'],
      dtype='object')
```

Original Datatypes:

```
<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)
```

There are 3 types of data in total: object, int and float. Let's analyse what should be each variables datatype in detail.

dteday: This is variable denotes the date which will be unique for every day, hence can't be a category. It can neither a numerical variable as no date can be interpreted as greater or smaller that another date. Hence leave it as an object.

season: From the info given with the dataset, it is mentioned that season has 4 unique values. Hence, it will be a categorical variable.

yr: Our dataset contains observations from year 2011, denoted as 0 and year 2012, denoted as 1. Hence, it'll also be a categorical variable.

mnth: In every year there are 12 months and mnth has 12 categories.

holiday: Either a day is a holiday or it isn't. Hence, a categorical.

workingday: It will also be a categorical.

weekday: Each week has 7 days which are unique and hence a categorical variable.

weathersit: From information attached to our dataset, it has values belonging to 4 categories.

temp, **atemp**, **hum**, **windspeed**: All of these 4 variables will contain a numeric value. Hence float type is appropriate.

casual, **registered**, **cnt**: These are our target variables and are continuous.

After conversion:

```
<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 category
yr            731 non-null category
mnth          731 non-null category
holiday       731 non-null category
weekday       731 non-null category
workingday    731 non-null category
weathersit    731 non-null category
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 float64
dtypes: category(7), float64(5), int64(3), object(1)
memory usage: 57.9+ KB
```

Central tendency and other information about the numerical variables:

```
Descriptive statistics about the numeric columns:
             temp       atemp         hum   windspeed       casual  \
count  731.000000  731.000000  731.000000  731.000000   731.000000
mean     0.495385    0.474354    0.627894    0.190486   848.176471
std      0.183051    0.162961    0.142429    0.077498   686.622488
min      0.059130    0.079070    0.000000    0.022392     2.000000
25%      0.337083    0.337842    0.520000    0.134950   315.500000
50%      0.498333    0.486733    0.626667    0.180975   713.000000
75%      0.655417    0.608602    0.730209    0.233214  1096.000000
max      0.861667    0.840896    0.972500    0.507463  3410.000000

        registered          cnt
count   731.000000   731.000000
mean   3656.172367  4504.348837
std    1560.256377  1937.211452
min      20.000000    22.000000
25%    2497.000000  3152.000000
50%    3662.000000  4548.000000
75%    4776.500000  5956.000000
max    6946.000000  8714.000000
```

Conclusion:

From the above descriptive statistics, we got to know about the central tendency, min & max value of all the numeric variables. It is seen that all independent variables are scaled from 0 to 1.

Count of each category in every categorical variable in our dataset is as follows:

```
3    188
2    184
1    181
4    178
Name: season, dtype: int64

1    366
0    365
Name: yr, dtype: int64

12    62
10    62
8     62
7     62
5     62
3     62
1     62
11    60
9     60
6     60
4     60
2     57
Name: mnth, dtype: int64

0    710
1     21
Name: holiday, dtype: int64

6    105
1    105
0    105
5    104
4    104
3    104
2    104
Name: weekday, dtype: int64

1    500
0    231
Name: workingday, dtype: int64

1    463
2    247
3     21
Name: weathersit, dtype: int64
```

<u>Conclusion:</u>

Total Non-working day = 231
Total Holidays = 21
Total (Saturday+Sunday) = 105+105 = 210
=> Total Non-working day = Total Holiday + (Total Saturday+Sunday)
=> Working Day = 1: denotes it's a working day
& Working day = 0: denotes its either a holiday or Saturday/Sunday.

If working day + (all Saturday & Sunday + holidays) = Total No. of Observations, then holiday & Weekends don't overlap => Hence contain extra information and we can remove any one of holiday & working day.

3.1.1.1 Visualizations showing count of categorical variables

Figure I Count of categories of categorical variables

Conclusion:

Season, year, month, weekday variables are uniformed and balanced and it should be as these are time-based data and count of categories of month, weekday, season are fixed in a year. Beside these, we noticed that holiday variable is highly imbalanced, slight to moderate imbalance is also seen in weathersit and workingday variable as well.

3.1.1.2 Visualizations showing distributions of numeric variables

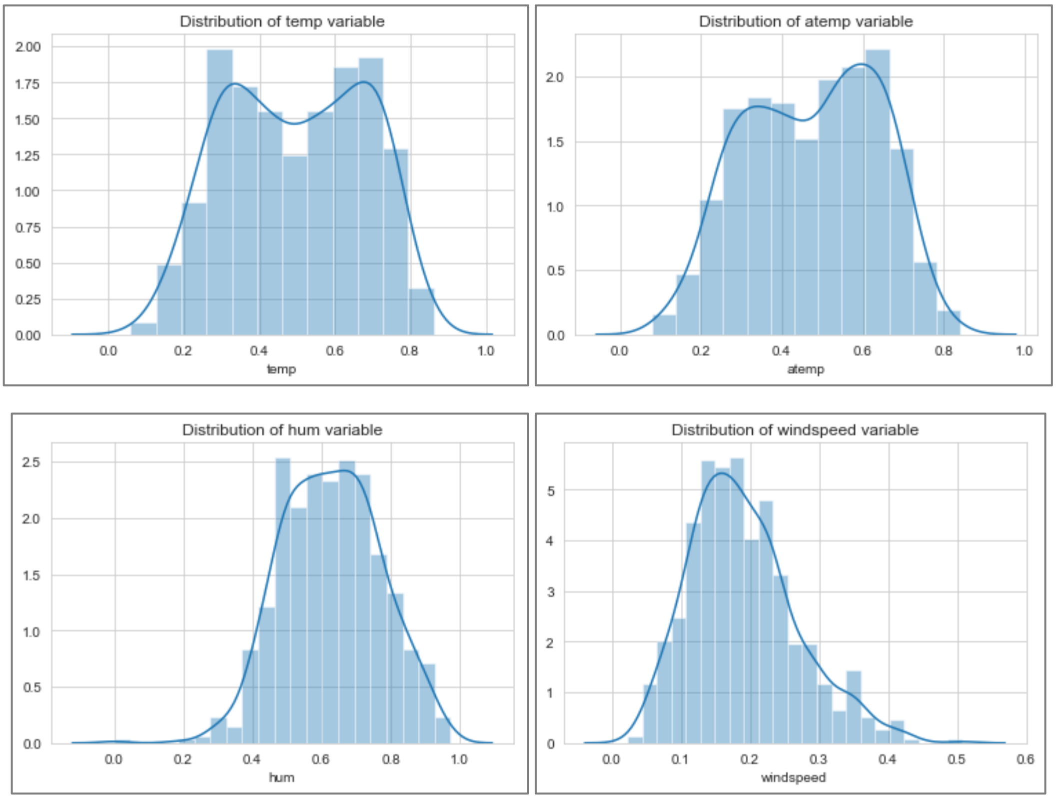

Figure II Probability Distribution of Numerical Variables

Conclusions:

Temp and atemp variables are showing a little bimodal effect. All the variables are normally distributed.

3.1.1.3 Bivariate analysis of numerical variables

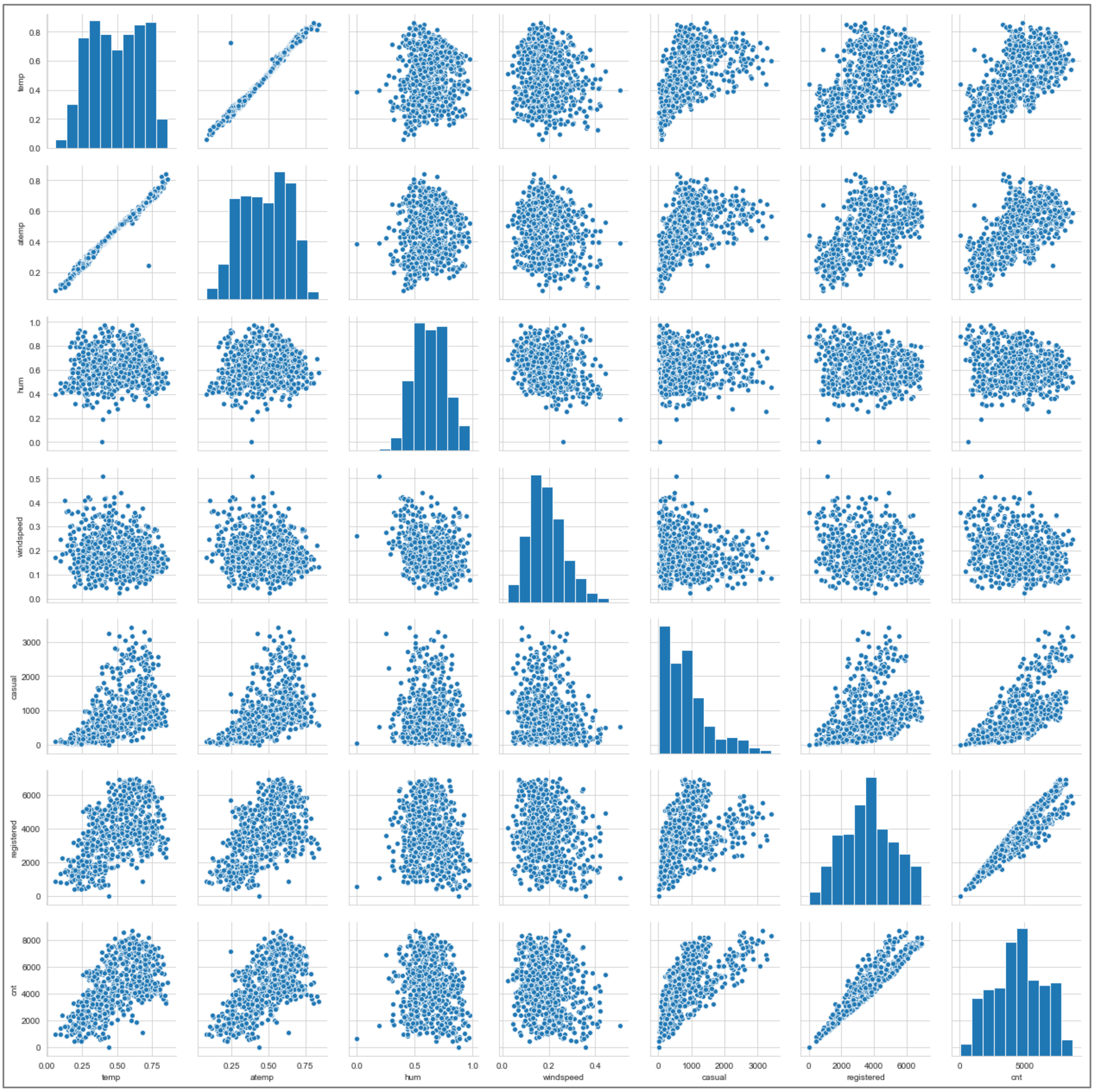

Figure III Bivariate analysis of numeric variables

<u>Conclusion:</u>

High correlation is observed between temp and atemp variables. Registered and cnt variables are also highly correlated.

3.1.1.4 Bivariate Analysis of categorical variables

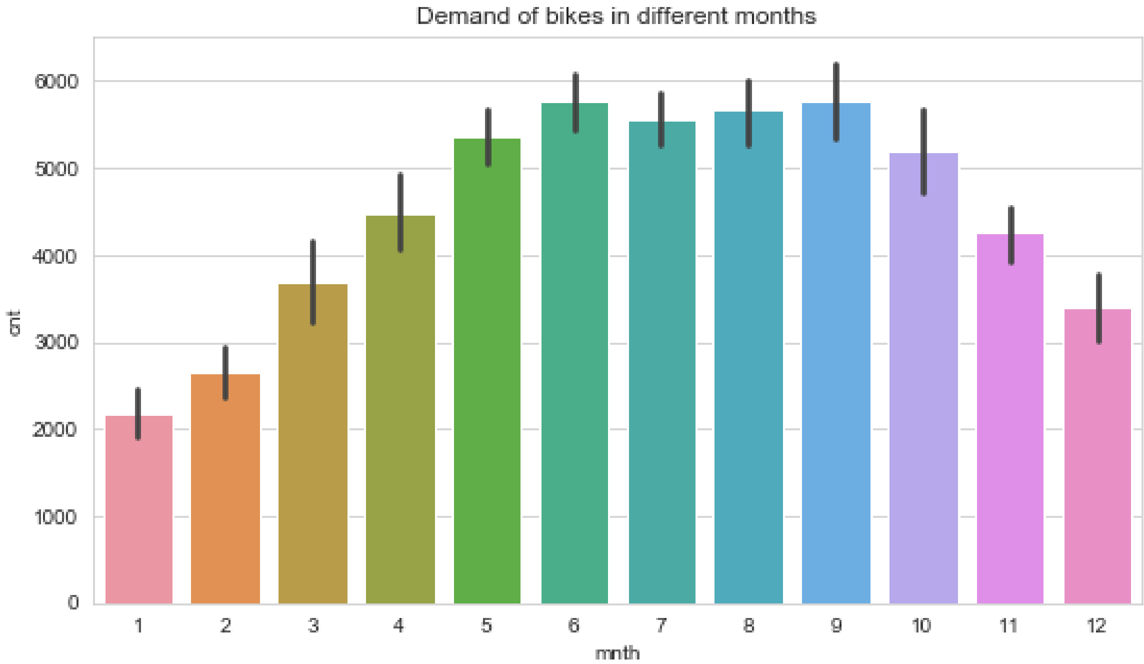

Figure IV Demand Rental Bikes on different months of year 2011 & 2012

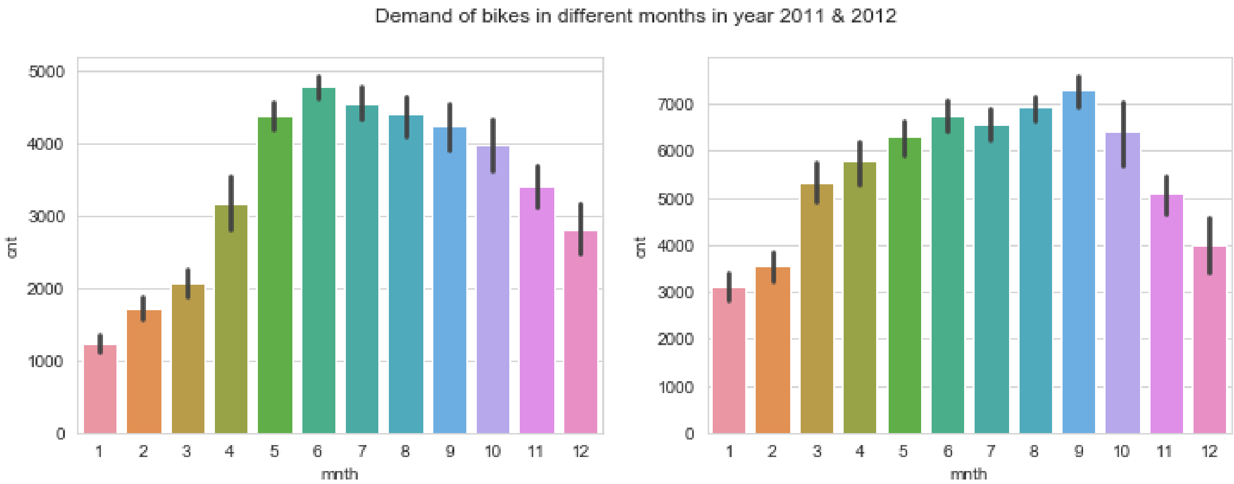

Figure V Demand of rental bikes on different months of 2011 & 2012 separately

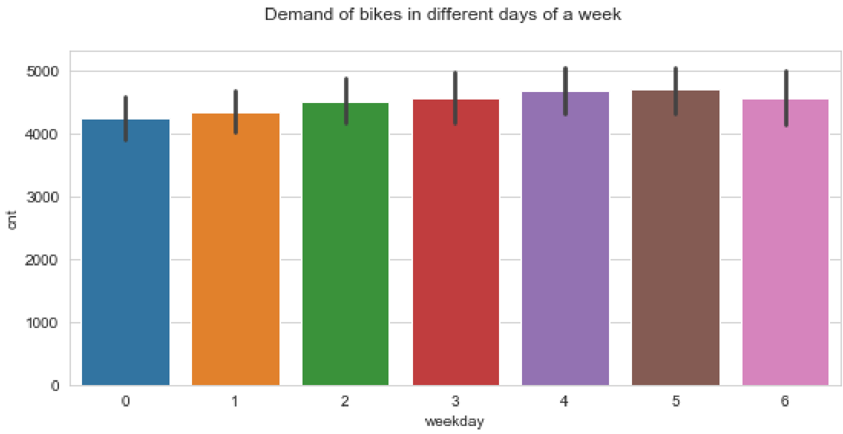

Figure VI Demand of rental bikes on different days of a week

Conclusion:

Both month and weekday variable show a trend that from 5th to 10th month shows a higher demand of rental bikes than other months in both the years 2011 & 2012.

Similarly, weekday 2-6; i.e. working days shows higher demand of rental bikes than on the weekends.

3.1.2 Missing Value Analysis

The data we collect are mostly impure and may contain missing values which will reduce the efficiency of our model. Hence treating missing values in a dataset is important and comes under the EDA process. There are several ways of treating the missing value; e.g. Mean/Median Imputation, Knn imputation etc. We are lucky though there are no missing values exist in our dataset.

instant	0
dteday	0
season	0
yr	0
mnth	0
holiday	0
weekday	0
workingday	0
weathersit	0
temp	0
atemp	0
hum	0
windspeed	0
casual	0
registered	0
cnt	0

3.1.3 Outlier Analysis

An outlier is a data point that differs significantly from other observations. An outlier can affect the mean of a data set by skewing the results so that the mean is no longer representative of the dataset. So, our dataset should be free from outliers for better performance of our model. There are mainly two ways to treat to outliers: Boxplot method or Replace with NAs. Outlier analysis can only be performed on numeric variables.

In our dataset there are few outliers observed in some of the variables.

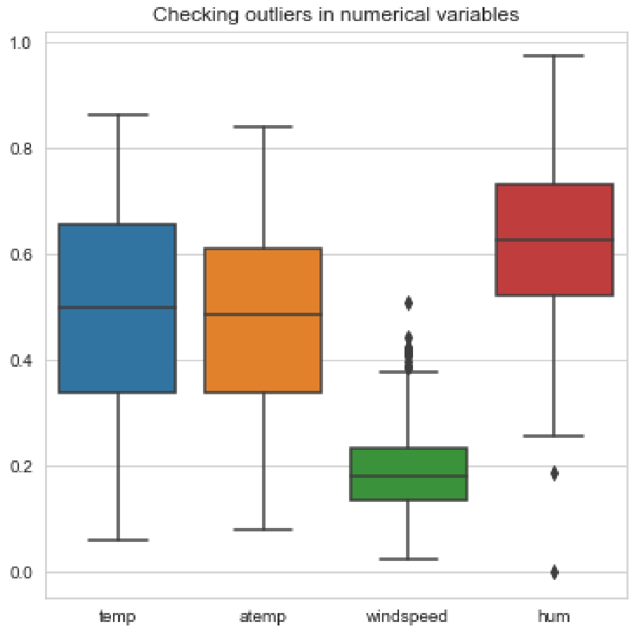

Figure VII Outlier Analysis using boxplot method

The outliers seen in the windspeed and humidity variables are near to 1% data of the whole dataset. None of the mean/median or knn imputation method gave result even nearer to the original data. Hence, we will proceed with removing these outliers.

3.1.4 Feature Engineering

According to Wikipedia:

"Feature engineering is the process of using domain knowledge of the data to create features that make machine learning algorithms work.".

Previously we are considering month and weekday variable as categorical. But as both of these variables have unusual number of categories, it may result in "Curse of dimensionality". Besides this, from the conclusions of "Bivariate analysis of categorical variables", we got a trend that both month and weekday variables are following. From those conclusions, we can replace 5^{th} to 10^{th} month as 1 and rest months as 0. Similarly, we will replace the weekday 2-6 as 1 and rest of the days of the week as 0.

By this we can reduce the no. categories in month variable from 12 to 2 and in weekday variable from 7 to 2. This process is known as **Binning**.

Binning or grouping data (sometimes called quantization) is an important tool in preparing numerical data for machine learning, and is useful in scenarios like these:

A column of continuous numbers has too many unique values to model effectively, so you automatically or manually assign the values to groups, to create a smaller set of discrete ranges.

3.1.5 Feature Selection

We get a lot of observations and variables in our raw data, but it is not necessary that all of those will be helpful in achieving our target. We have to selective while choosing variables that we are going to feed to our model, so that it won't be complex for our model to draw out the patterns from our data. There are so many feature selection techniques. We have applied 3 of them which will be discussed in this section.

During feature selection we have to keep in mind one thing that, there should be high dependency between our independent variables and target variable & there should be very less or no dependency in between the independent variables. To check the dependency between independent and dependent variable you can refer to the section 3.1.1.3. Now to check the inter-dependency between the independent variables we'll apply correlation analysis.

3.1.5.1 Correlation Analysis

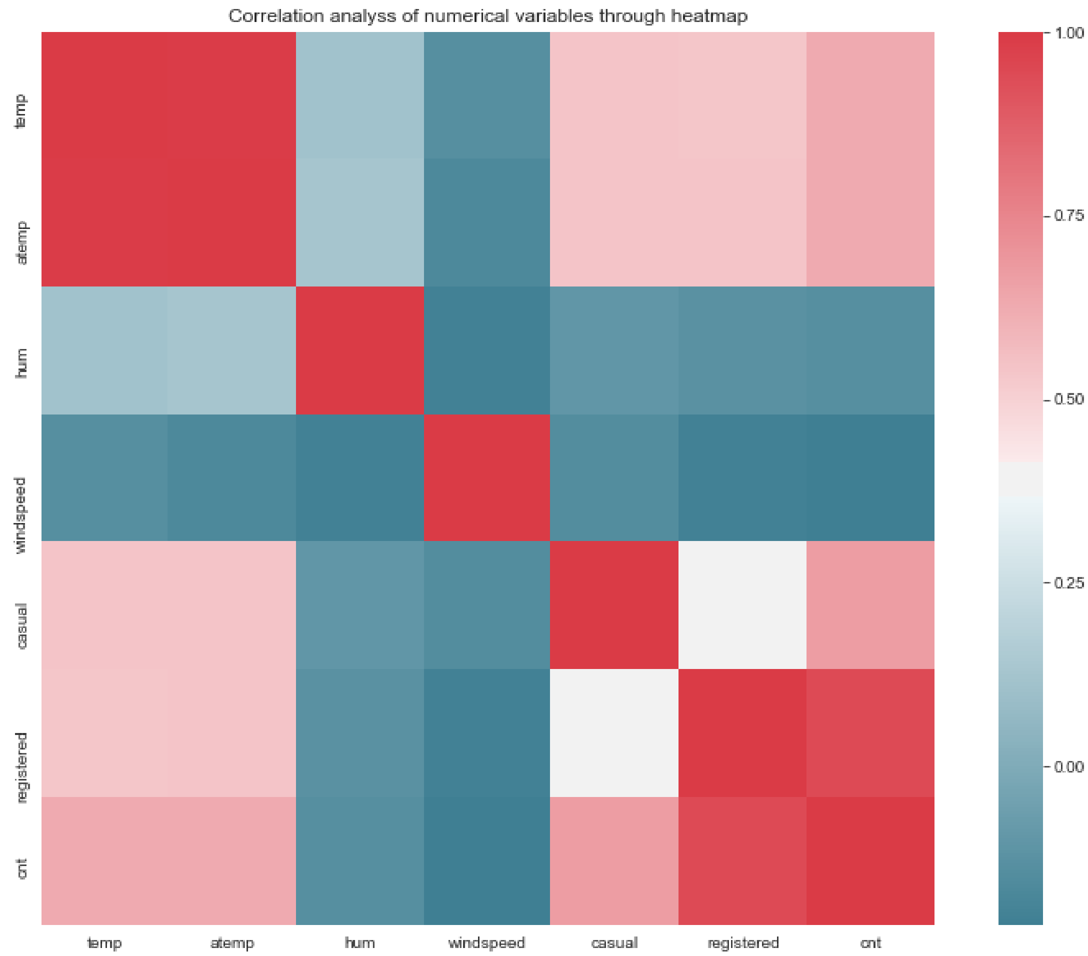

Figure VIII Correlation Matrix

The correlation matrix above depicts: extreme red colour as “both variables are highly positively corelated” & extreme blue colour as “both variables are highly negatively corelated”

Conclusions:

temp & atemp are positively co-related.

3.1.5.2 Chi-square test

There are two hypotheses in Chi-Square test based on which we can check dependency between two variables.

Null Hypothesis, H0: Two variables are independent.

Alternate Hypothesis, H1: Two variables are not independent.

Chi-square test gives us a probability value based on which we’ll decide which hypothesis is going to be rejected and which won’t be rejected. If the p-value < 0.05, we reject the null hypothesis saying “Two variables are not independent”.

We have checked inter-dependency between the continuous independent variables through correlation analysis. Now we’ll check the inter-dependency between the categorical independent variable and we want that interdependency to be minimum, i.e. p-value > 0.05.

In our project we got the following pair of variables who gave p-value < 0.05:

('season', 'weathersit'):	**0.013**
('season', 'month_binned'):	**0.0**
('holiday', 'workingday'):	**0.0**
('holiday', 'weekday_binned'):	**0.0**
('workingday', 'holiday'):	**0.0**
('workingday', 'weekday_binned'):	**0.0**
('weathersit', 'season'):	**0.013**
('month_binned', 'season'):	**0.0**
('weekday_binned', 'holiday'):	**0.0**
('weekday_binned', 'workingday'):	**0.0**

Conclusions:

Season with Weathersit-Month, Holiday with Workingday-Weekday & Workingday with Weekday-Holiday showing dependency.

3.1.5.3 Feature Importance

We can also check the importance of every predictor variable that they contribute in predicting the target variable. For this dataset we got the result as below:

```
holiday    :                0.008
month_binned    :           0.024
weekday_binned    :         0.031
workingday    :             0.033
yr    :                     0.037
weathersit    :             0.04
season    :                 0.045
windspeed    :              0.188
hum    :                    0.191
atemp    :                  0.198
temp    :                   0.202
```

Conclusion:

Holiday variable is the least important variable from all the independent variable.

3.1.6 Multi-collinearity test

Multicollinearity is a state of very high intercorrelations or inter-associations among the independent variables. It is therefore a type of disturbance in the data, and if present in the data the statistical inferences made about the data may not be reliable.

Multicollinearity makes it tedious to assess the relative importance of the independent variables in explaining the variation caused by the dependent variable.

In our dataset it is already seen collinearity between temp and atemp variable, let's check what vif tells us about multi-collinearity.

Variation Inflation Factor (VIF) in presence of atemp variable:

```
const          46.436
temp           63.326
atemp          63.933
hum            1.057
windspeed      1.102
dtype: float64
```

Variation Inflation Factor (VIF) in presence of atemp variable:

```
const          41.644
temp           1.028
hum            1.052
windspeed      1.059
dtype: float64
```

Conclusions:

As after removing atemp, vif is around 1 for all variables, it is safe to apply regression.

3.1.7 Creating Dummies for categorical variable

A dummy variable is a numeric variable that represents categorical data.

Regression results are easiest to interpret when dummy variables are limited to two specific values, 1 or 0. Typically, 1 represents the presence of a qualitative attribute, and 0 represents the absence.

Once a categorical variable has been recoded as a dummy variable, the dummy variable can be used in regression analysis just like any other quantitative variable.

3.1.8 Fetaure Scaling

Most of the times, your dataset will contain features highly varying in magnitudes, units and range. In this case the features with high magnitudes will weigh in a lot more in the distance calculations than features with low magnitudes.

In our dataset, all the values are in a range of 0 to 1. Hence, the data is already scaled.

3.1.9 Removal of unwanted variables

Here, after applying all the pre-processing techniques, we derive many conclusions. One last task remain is to remove all the unwanted variables to make our data clean and feedable to our model. After analysing conclusions from all pre-processing techniques, we decided to remove the following variables:

Instant: This is just an index and contain no useful information in predicting our target.

Dteday: The information this variable sharing is already available to us through year and month variable.

Atemp: It shows high correlation with temp variable.

Casual, registered: These are the target variables.

Holiday: Least important feature among all independent variable and also shows dependency with other independent variables (from conclusion of chi-square test).

After applying EDA, shape of our data is: (717,13)

	yr	workingday	temp	hum	windspeed	month_binned	weekday_binned	season_2	season_3	season_4	weather_2	weather_3	cnt
0	0	0	0.344167	0.805833	0.160446	0	1	0	0	0	1	0	985.0
1	0	0	0.363478	0.696087	0.248539	0	0	0	0	0	1	0	801.0
2	0	1	0.196364	0.437273	0.248309	0	0	0	0	0	0	0	1349.0
3	0	1	0.200000	0.590435	0.160296	0	1	0	0	0	0	0	1562.0
4	0	1	0.226957	0.436957	0.186900	0	1	0	0	0	0	0	1600.0

Figure IX Data after EDA

3.2 Modelling

3.2.1 Model Selection

The target we are chasing to predict is a numerical variable; i.e. cnt: No. of bike to demanded at a particular place. Hence, we have a regression problem. Regression problems can be solved by many algorithms, e.g. Decision tree, Linear regression, Random forest, K-Nearest Neighbours, Support Vector Regressor, Boosting algorithms and many more.

Here we will apply all algorithms mentioned above one by one and compare them to know the best model for this particular problem.

Before applying any model, I would like to show the code structure used for development and evaluation of our model:

```
# Modularizing
from sklearn.model_selection import cross_val_score
from sklearn.metrics import r2_score

def fit_N_predict(model, X_train, y_train, X_test, y_test,model_code=''):

    if(model_code == 'OLS'):
        model = model.OLS(y_train,X_train.astype('float')).fit()
        print(model.summary())
        y_pred = model.predict(X_test.astype('float'))
        print("\n================================")
        print('Score on testing data: ',(r2_score(y_test,y_pred)*100).round(3))
        print("================================")
        return

    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)
    print("================================")
    print("Score on training data: ",(model.score(X_train, y_train)*100.0).round(3))
    print("================================")
    print("Score on testing data: ", (model.score(X_test, y_test)*100.0).round(3)) ## Same as r-squared value
    print("================================")

    if(model_code == "DT"):
        from sklearn import tree
        dotfile = open("pt.dot","w")
        df = tree.export_graphviz(model, out_file=dotfile, feature_names = X_train.columns)
```

fit_N_predict: A single function is made for easy modification and to maintain code reusability, which takes a model object, train data, test data and a unique model code for each model and gives us the model's accuracy for which we are using the R-Squared metric. Based on the model code we can perform some other operations for some specific models as well

cross_validation: Another method which performs K-fold cross validation for us and returns back the mean of 10 scores from 10 folds. It takes the whole dataset as input. The function's code is structured as below.

```python
from sklearn.model_selection import KFold
from sklearn.metrics import r2_score
from statistics import mean
kf = KFold(n_splits=10, shuffle=True, random_state=42)


def cross_validation(model,X,y):
    l = []
    for train_index, test_index in kf.split(X,y):
        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.fit(X_train,y_train)
        y_pred = model.predict(X_test)
        l.append(r2_score(y_test,y_pred))
    print("Mean of 10 cross validation scores = ",(mean(l)*100).round(3))
```

Importance of K-Fold Cross Validation:

In K-Fold cross validation what happens is, we use the whole dataset some during both training and testing. While using the traditional splitting of data, we randomly separate a part of data as our validation set, which may contain some crucial information from which our model can't draw any patterns. Cross validation solved this problem by splitting the data into k-folds and in this process the training and testing takes place k-times. In every iteration of all the k iterations, k-1 folds used for training and 1-fold for testing and this k-1,1 fold used for training and testing respectively are different for every iteration.

Grid Search CV

After developing any model, we don't know it is performing to its optimum level or not. There are many parameters that a model takes as input. It is not possible for a human to test our model with all possible combinations of all parameters that a function takes. With the help of grid search we can determine the best combination of parameters by inputting which we can get the best of any model. This technique is known as **hyper-parameter tuning**.

So, after you are introduced with the code structure used in our project, let's apply each model one by one and analyse the results.

We'll go from applying from simpler models to complex models. For every model we show the scores 1st when model is applied with train data, 2nd when applied with test data and lastly the cross validation score.

Linear Regression:

```
######################## LINEAR REGRESSION ##################
===================================
Score on training data:    81.263
===================================
Score on testing data:    87.42
===================================
Mean of 10 cross validation scores =    81.57
```

```
In [74]: import statsmodels.api as sm
         model = sm
         fit_N_predict(model,X_train, y_train, X_test, y_test,model_code="OLS")

                                 OLS Regression Results
=======================================================================================
Dep. Variable:                    cnt   R-squared (uncentered):                   0.970
Model:                            OLS   Adj. R-squared (uncentered):              0.970
Method:                 Least Squares   F-statistic:                              1533.
Date:                Tue, 07 Jan 2020   Prob (F-statistic):                        0.00
Time:                        15:29:18   Log-Likelihood:                         -4671.9
No. Observations:                 573   AIC:                                      9368.
Df Residuals:                     561   BIC:                                      9420.
Df Model:                          12
Covariance Type:            nonrobust
==================================================================================
                     coef    std err          t      P>|t|      [0.025      0.975]
----------------------------------------------------------------------------------
yr              2132.5260     70.874     30.089      0.000    1993.315    2271.737
workingday       111.1339     81.672      1.361      0.174     -49.287     271.555
temp            4724.7662    404.637     11.677      0.000    3929.977    5519.555
hum              244.8524    249.110      0.983      0.326    -244.450     734.155
windspeed       -778.5416    434.626     -1.791      0.074   -1632.234      75.151
month_binned     310.6609    120.701      2.574      0.010      73.580     547.741
weekday_binned   368.0815     83.653      4.400      0.000     203.771     532.392
season_2        1104.2190    131.957      8.368      0.000     845.028    1363.410
season_3         736.0110    174.951      4.207      0.000     392.372    1079.650
season_4        1489.8955    114.025     13.066      0.000    1265.927    1713.864
weather_2       -588.5358     91.709     -6.417      0.000    -768.671    -408.401
weather_3      -2465.6840    245.016    -10.063      0.000   -2946.945   -1984.423
==============================================================================
Omnibus:                       85.095   Durbin-Watson:                   1.812
Prob(Omnibus):                  0.000   Jarque-Bera (JB):              225.228
Skew:                          -0.748   Prob(JB):                     1.24e-49
Kurtosis:                       5.683   Cond. No.                         22.5
==============================================================================
```

K Nearest Neighbour:

```
################### K NEAREST NEIGHBOUR REGRESSOR ###########
===================================
Score on training data:    85.281
===================================
Score on testing data:    87.095
===================================
Mean of 10 cross validation scores =    79.448
```

<u>Support Vector Regressor:</u>

```
######################### SVR #########################
==================================
Score on training data:   9.268
==================================
Score on testing data:   9.741
==================================
Mean of 10 cross validation scores =   9.232
```

<u>Decision Tree:</u>

```
######################### Decision Tree #########################
==================================
Score on training data:   100.0
==================================
Score on testing data:   82.819
==================================
Mean of 10 cross validation scores =   74.95
```

<u>Random Forest:</u>

```
######################### Random Forest #########################
==================================
Score on training data:   96.95
==================================
Score on testing data:   91.274
==================================
Mean of 10 cross validation scores =   84.933
```

<u>XGBoost Regressor:</u>

```
######################### XGB Regressor #########################
==================================
Score on training data:   94.004
==================================
Score on testing data:   92.043
==================================
Mean of 10 cross validation scores =   87.5
```

So, we got the best results from random forest and XGBoost regressor models, as they showed the least variance on test data with accuracy values 91.274% and 92.043% respectively; we'll further try to improve the performance of these two models by tuning their hyperparameters.

Random Forest on hyperparameter tuning:

```
################### Tuning Random Forest ########################

Best parameters for Random Forest {'max_depth': 14, 'max_features': 'auto', 'min_samples_leaf': 2, 'min_samples_split': 2, 'n_estimators': 500, 'random_state': 1}
=================================
Score on training data:    96.263
=================================
Score on testing data:    91.706
=================================
Mean of 10 cross validation scores =    86.835
```

XGBoost on hyperparameter tuning:

```
###################### Tuning XGBoost ########################
Best parameters for XGBoost {'gamma': 0, 'learning_rate': 0.045, 'max_depth': 3, 'n_estimators': 300, 'random_state': 1, 'subsample': 0.7}
=================================
Score on training data:    95.519
=================================
Score on testing data:    92.928
=================================
Mean of 10 cross validation scores =    88.34
```

Chapter 3

4. Conclusion

4.1 Model Selection

Hence, after seeing the scores from all the model, we came to a conclusion that, XGBoost algorithms works best on our data and will predict with an accuracy of 92.928% when fed with a new data. Other than this, it gave 95.519% accuracy with train data.

5. Appendix A :: Python Code

```
import os
import numpy as np
import pandas as pd
import scipy.stats as stats
from scipy.stats import chi2_contingency
import matplotlib as plt
import matplotlib.pyplot as plt2
import seaborn as sns
import sys


if not sys.warnoptions:
    import warnings
    warnings.simplefilter("ignore")



os.chdir("G:/Project/Bike-Sharing-Dataset")


bike_sharing_train = pd.read_csv("day.csv")

print('Shape of our dataset:')
print(bike_sharing_train.shape,'\n')


# ## Exploratory Data Analysis

print('*'*25,'Exploratory Data Analysis: ','*'*25,'\n')

print('Column / Variable Names:')
print(bike_sharing_train.columns)

# Showing 1st few rows of our dataset
print('Showing 1st few rows of our dataset: \n')
print(bike_sharing_train.head(5))

print("Basic info about dataset:\n")
print(bike_sharing_train.info())

print("Checking the data types of the variables:\n")
print(bike_sharing_train.dtypes,'\n')

print("Converting the varibales to it's proper data type: \n\nAfter Convertion:\n")

bike_sharing_train['season'] = bike_sharing_train['season'].astype('category')
bike_sharing_train['yr'] = bike_sharing_train['yr'].astype('category')
bike_sharing_train['mnth'] = bike_sharing_train['mnth'].astype('category')
bike_sharing_train['weekday'] = bike_sharing_train['weekday'].astype('category')
bike_sharing_train['workingday'] = bike_sharing_train['workingday'].astype('category')
bike_sharing_train['weathersit'] = bike_sharing_train['weathersit'].astype('category')
bike_sharing_train['holiday'] = bike_sharing_train['holiday'].astype('category')

bike_sharing_train['temp'] = bike_sharing_train['temp'].astype('float')
bike_sharing_train['atemp'] = bike_sharing_train['atemp'].astype('float')
bike_sharing_train['hum'] = bike_sharing_train['hum'].astype('float')
bike_sharing_train['windspeed'] = bike_sharing_train['windspeed'].astype('float')
```

```
bike_sharing_train['cnt'] = bike_sharing_train['cnt'].astype('float')

print(bike_sharing_train.dtypes,'\n')


categorical = ['season','yr','mnth','holiday','weekday','workingday','weathersit']

print('Count of each categorical variable in our data is as follows:\n')
[print(bike_sharing_train[i].value_counts(),'\n\n') for i in categorical]


# Count of each category of a categorical variable
print("Checking count of each category of categorical variables in dataset\n\n")
sns.set_style("whitegrid")

def check_count_of_category(categorical_var):
    ax = sns.factorplot(data=bike_sharing_train, x=categorical_var, kind= 'count',size=3,aspect=2)
    title = "Count of each category of "+categorical_var+" variable in 2years"
    plt2.title(title)
    plt2.show()
[check_count_of_category(i) for i in categorical]


# ##  Univariate & Bivariate analysis


numeric = ['temp','atemp','hum','windspeed','casual','registered','cnt']
print("Descriptive statistics about the numeric columns:")
print(bike_sharing_train[numeric].describe(),'\n')


print("Univariate analysis of numerical variables")
def dist_plot(i):
    sns.distplot(bike_sharing_train[i])
    title = "Distribution of "+ i + " variable"
    plt2.title(title)
    plt2.show()

num = ['temp','atemp','hum','windspeed']

[dist_plot(i) for i in num]

print("Bivariate analysis of numerical variables")
sns.pairplot(bike_sharing_train[numeric])
plt2.show()


# ### For Month


# Month
fig, ax = plt2.subplots(nrows = 1, ncols = 1, figsize= (9,5), squeeze=False)

x1 = 'mnth'
y1='cnt'

sns.barplot(x= x1, y = y1, data = bike_sharing_train, ax=ax[0][0])
title = "Demand of bikes in different months"
plt2.title(title)
plt2.show()

yr_0 = bike_sharing_train.loc[bike_sharing_train['yr'] == 0]
yr_1 = bike_sharing_train.loc[bike_sharing_train['yr'] == 1]

```

```

fig, ax = plt2.subplots(nrows = 1, ncols = 2, figsize= (12,4), squeeze=False)
fig.suptitle("Demand of bikes in different months in year 2011 & 2012")
x1 = 'mnth'
y1='cnt'
sns.barplot(x= x1, y = y1, data = yr_0, ax=ax[0][0])
sns.barplot(x= x1, y = y1, data = yr_1, ax=ax[0][1])
plt2.show()


# ### For Weekday


fig, ax = plt2.subplots(nrows = 1, ncols = 1, figsize= (9,4), squeeze=False)
fig.suptitle("Demand of bikes in different days of a week")
x1 = 'weekday'
y1='cnt'

sns.barplot(x= x1, y = y1, data = bike_sharing_train, ax=ax[0][0])


print("From figures we can categorize 5-10th month as one category and rest months as another category.\n")
print("Similarly,in weekday variables; workindays can be categorized as one and weekends as another category. As in working days demand of bikes found high than weekends.\n")


# Keep on adding the unwanted variables (that we will get by applying different techniques) to remove list and
# will finally we will remove from our dataset
remove = ['instant','dteday']


# ## Missing Value Analysis


print('*'*25,'Missing Value Analysis: ','*'*25,'\n')


missing_val = pd.DataFrame(bike_sharing_train.isnull().sum())


print('Missing values in our dataset: \n')
print(missing_val)

print("No missing values present in our dataset")


# ## Outlier Analysis


print('*'*25,'Outlier Analysis','*'*25,'\n')

# Check for outliers in data using boxplot
sns.boxplot(data=bike_sharing_train[['temp','atemp','windspeed','hum']])
fig=plt2.gcf()
title = "Checking outliers in numerical variables"
plt2.title(title)
fig.set_size_inches(6,6)


print("Outliers found in windspeed and humidity variable")

```

```python

# Numeric Variables
num = ['temp','atemp','hum','windspeed']


# Removing the outliers
for i in num:
    q75, q25 = np.percentile(bike_sharing_train[i], [75, 25])
    iqr = q75 - q25
    minimum = q25 - (iqr*1.5)
    maximum = q75 + (iqr*1.5)

    bike_sharing_train = bike_sharing_train.drop(bike_sharing_train[bike_sharing_train.loc[:,i] < minimum].index)
    bike_sharing_train = bike_sharing_train.drop(bike_sharing_train[bike_sharing_train.loc[:,i] > maximum].index)
print("Outliers removed")


# ## Feature Engineering


bike_sharing_train.head()

categorical = ['season','yr','mnth','holiday','weekday','workingday','weathersit']


# Creating new variables  through binning
def binned_month(row):
    if row['mnth'] <= 4 or row['mnth'] >=11:
        return(0)
    else:
        return(1)

def binned_weekday(row):
    if row['weekday'] < 2:
        return(0)
    else:
        return(1)


bike_sharing_train['month_binned'] = bike_sharing_train.apply(lambda row : binned_month(row), axis=1)
bike_sharing_train = bike_sharing_train.drop(columns=['mnth'])
bike_sharing_train['weekday_binned'] = bike_sharing_train.apply(lambda row : binned_weekday(row), axis=1)
bike_sharing_train = bike_sharing_train.drop(columns=['weekday'])


categorical.remove('mnth')
categorical.remove('weekday')
categorical.append('month_binned')
categorical.append('weekday_binned')


# ## Feature Selection

bike_sharing_train.columns

# ### Correlation Analysis

df_corr = bike_sharing_train[numeric]



```

```
# Correlation Analysis
f, ax = plt2.subplots(figsize=(14, 10))
corr = df_corr.corr()
sns.heatmap(corr, mask=np.zeros_like(corr, dtype=np.bool), cmap=sns.diverging_palette(220, 10, as_cmap=True),
            square=True, ax=ax)
plt2.title("Correlation analyss of numerical variables through heatmap")



print("From correlation analysis we found,\n    1.temp and atemp are highly correlated.\n    2.registered and cnt also showing high correlation.\n")


remove.extend(['atemp','casual','registered'])


# ### Chi-square test


print("Chi-square Test\n1. Null Hypothesis: Two variables are independent\n2. Alternate Hypothesis: Two variables are not independent\n3. p-value < 0.05 , can not accept null hypothesis\n")
print("That means p < 0.05 two categorical variables are dependent, so we will remove one of variable from that pair to avoid sending the same information to our model through 2 variables")


# Create all combinations
factors_paired = [(i,j) for i in categorical for j in categorical]



# Calculating p-values for each pair
p_values = []
from scipy.stats import chi2_contingency
for factor in factors_paired:
    if factor[0] != factor[1]:
        chi2, p, dof, ex = chi2_contingency(pd.crosstab(bike_sharing_train[factor[0]],
                                                    bike_sharing_train[factor[1]]))
        if(p<0.05):
            p_values.append({factor:p.round(3)})
    else:
        p_values.append('-')

[print(i,'\n') for i in p_values if i != '-']


print("Season with Weathersit-Month\nHoliday with Worikingday-Weekday\nWorkingday with Weekday-Holiday\n")

bike_sharing_train.columns


# ### Importance of Features


from sklearn.ensemble import RandomForestClassifier
clf = RandomForestClassifier(random_state=0, n_jobs=-1)
X = bike_sharing_train.drop(columns=['cnt','casual','registered','instant','dteday'])
y = bike_sharing_train['cnt']
model = clf.fit(X, y)
```

```
importances = model.feature_importances_


X.columns


print("Checking feature importance: \n")
l = list(zip(X,importances))
l.sort(key = lambda x: x[1])
[print(i[0]," : ",i[1].round(3)) for i in l]


remove.append('holiday')


# ### Multi-colinearity test


from statsmodels.stats.outliers_influence import variance_inflation_factor as vf
from statsmodels.tools.tools import add_constant
numeric_df = add_constant(bike_sharing_train[['temp', 'atemp', 'hum', 'windspeed']])
vif = pd.Series([vf(numeric_df.values, j) for j in range(numeric_df.shape[1])],index = numeric_df.columns)
vif.round(3)


print("After removing atemp variable , VIF:\n")
numeric_df = add_constant(bike_sharing_train[['temp', 'hum', 'windspeed']])

vif = pd.Series([vf(numeric_df.values, i) for i in range(numeric_df.shape[1])],
                 index = numeric_df.columns)
print(vif.round(3))


# ### Dummy for categorical


season_dm = pd.get_dummies(bike_sharing_train['season'], drop_first=True, prefix='season')
bike_sharing_train = pd.concat([bike_sharing_train, season_dm],axis=1)
bike_sharing_train = bike_sharing_train.drop(columns = ['season'])
weather_dm = pd.get_dummies(bike_sharing_train['weathersit'], prefix= 'weather',drop_first=True)
bike_sharing_train = pd.concat([bike_sharing_train, weather_dm],axis=1)
bike_sharing_train = bike_sharing_train.drop(columns= ['weathersit'])


remove


# Removing unwanted variables
bike_sharing_train.drop(columns=remove, inplace=True)


# Reshaping
cnt = bike_sharing_train['cnt']
bike_sharing_train = bike_sharing_train.drop(columns=['cnt'])
bike_sharing_train['cnt'] = cnt

bike_sharing_train.shape

print(bike_sharing_train.head(5), '\n')
```

```
print('shape of dataset after all pre-processing\n',bike_sharing_train.shape)


# ### Model Development


# Modularizing
from sklearn.model_selection import cross_val_score
from sklearn.metrics import r2_score

def fit_N_predict(model, X_train, y_train, X_test, y_test,model_code=''):

    if(model_code == 'OLS'):
        model = model.OLS(y_train,X_train.astype('float')).fit()
        print(model.summary())
        y_pred = model.predict(X_test.astype('float'))
        print("\n================================")
        print('Score on testing data: ',(r2_score(y_test,y_pred)*100).round(3))
        print("================================")
        return

    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)
    print("================================")
    print("Score on training data: ",(model.score(X_train, y_train)*100.0).round(3))
    print("================================")
    print("Score on testing data: ", (model.score(X_test, y_test)*100.0).round(3)) ## Same as r-squared value
    print("================================")

    if(model_code == "DT"):
        from sklearn import tree
        dotfile = open("pt.dot","w")
        df = tree.export_graphviz(model, out_file=dotfile, feature_names = X_train.columns)




from sklearn.model_selection import KFold
from sklearn.metrics import r2_score
from statistics import mean
kf = KFold(n_splits=10, shuffle=True, random_state=42)


def cross_validation(model,X,y):
    l = []
    for train_index, test_index in kf.split(X,y):
        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.fit(X_train,y_train)
        y_pred = model.predict(X_test)
        l.append(r2_score(y_test,y_pred))
    print("Mean of 10 cross validation scores = ",(mean(l)*100).round(3))

# Partitioning of dataset
X = bike_sharing_train.drop(columns=['cnt'])
y = bike_sharing_train[['cnt']]
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, random_state = 0)

```

```
        print('######################### LINEAR REGRESSION #########################')
        from sklearn.linear_model import LinearRegression

        model = LinearRegression()
        fit_N_predict(model, X_train, y_train, X_test, y_test, model_code="SK_LR")
        cross_validation(model,X,y)

        import statsmodels.api as sm
        model = sm
        fit_N_predict(model,X_train, y_train, X_test, y_test,model_code="OLS")

        print('######################### K NEIGHBOURS REGRESSOR #########################')
        from sklearn.neighbors import KNeighborsRegressor
        model = KNeighborsRegressor(n_neighbors=5)
        fit_N_predict(model, X_train, y_train, X_test, y_test,model_code="KNN")
        cross_validation(model,X,y)

        print('######################### SVR #########################')
        from sklearn.svm import SVR
        model = SVR(kernel = 'linear')
        fit_N_predict(model, X_train, y_train, X_test, y_test, model_code="SVR")
        cross_validation(model,X,y)


        print('######################### Decision Tree #########################')

        from sklearn.tree import DecisionTreeRegressor
        model = DecisionTreeRegressor(random_state=1)
        fit_N_predict(model, X_train, y_train, X_test, y_test, model_code="DT")
        cross_validation(model,X,y)


        print('######################### Random Forest #########################')

        from sklearn.ensemble import RandomForestRegressor
        model = RandomForestRegressor(random_state=1)
        fit_N_predict(model, X_train, y_train, X_test, y_test, model_code="RF")
        cross_validation(model,X,y)


        # Pre-processing of data for XGBoost
        bike = bike_sharing_train.copy()
        yr_dm = pd.get_dummies(bike['yr'], prefix= 'yr',drop_first=True)
        bike = pd.concat([bike, yr_dm],axis=1)
        bike = bike.drop(columns= ['yr'])

        workingday_dm = pd.get_dummies(bike['workingday'], prefix= 'workingday',drop_first=True)
        bike = pd.concat([bike, workingday_dm],axis=1)
        bike = bike.drop(columns= ['workingday'])

        bike['yr_1'] = bike['yr_1'].astype('int')
        bike['workingday_1'] = bike['workingday_1'].astype('int')

        X1 = bike.drop(columns=['cnt'])
        y1 = bike['cnt']
        from sklearn.model_selection import train_test_split
        X_train1, X_test1, y_train1, y_test1 = train_test_split(X1, y1, test_size = 0.2, random_state = 0)

        print('######################### XGB Regressor #########################')

        from xgboost import XGBRegressor
        model = XGBRegressor(random_state=1)
```

```python
    fit_N_predict(model, X_train1, y_train1, X_test1, y_test1, model_code="XGB")
    cross_validation(model,X1,y1)


    # ## Hyper-parameter tuning for the best models

    print('######################### Tuning Random Forest #########################')

    from sklearn.model_selection import GridSearchCV
    model = RandomForestRegressor(random_state=1)
    params = [{'n_estimators' : [400, 500, 600],
               'max_features': ['auto', 'sqrt', 'log2'],
               'min_samples_split': [2,4,6],
               'max_depth': [10, 12, 14],
               'min_samples_leaf': [2,3,5],
               'random_state' : [1]
              }]
    grid_search = GridSearchCV(estimator=model, param_grid=params,cv = 5,
                               scoring = 'explained_variance', n_jobs=-1)
    grid_search = grid_search.fit(X_train, y_train)
    print('Best parameters for Random Forest',grid_search.best_params_)

    # Developing Random Forest model with best params
    model = RandomForestRegressor(random_state=1, max_depth=14, n_estimators=500,
                                  max_features='auto', min_samples_leaf=2,min_samples_split=2)
    fit_N_predict(model, X_train, y_train, X_test, y_test, model_code="RF")
    cross_validation(model,X,y)

    print('######################### Tuning XGBoost #########################')
    model = XGBRegressor(random_state=1)
    params = [{'n_estimators' : [200, 250, 300,350, 400,450],
               'max_depth':[2, 3, 5],
               'learning_rate':[0.01, 0.045, 0.05, 0.055, 0.1, 0.3],
               'gamma':[0, 0.001, 0.01, 0.03],
               'subsample':[1, 0.7, 0.8, 0.9],
               'random_state' :[1]
              }]
    grid_search = GridSearchCV(estimator=model, param_grid=params,cv = 5,
                               scoring = 'explained_variance', n_jobs=-1)
    grid_search = grid_search.fit(X_train1, y_train1)
    print('Best parameters for XGBoost',grid_search.best_params_)


    # Developing XGBoost model with best params
    model = XGBRegressor(random_state=1, learning_rate=0.045, max_depth=3, n_estimators=300,
                         gamma = 0, subsample=0.7)
    fit_N_predict(model, X_train1, y_train1, X_test1, y_test1, model_code="XGB")
    cross_validation(model,X1,y1)

    # Scatterplot showing the prediction vs actual values for the best model for our dataset

    model = XGBRegressor(random_state=1, learning_rate=0.045, max_depth=3, n_estimators=300,
                       gamma = 0, subsample=0.7)
    model.fit(X_train1,y_train1)
    y_pred = model.predict(X_test1)
    fig, ax = plt2.subplots(figsize=(7,5))
    ax.scatter(y_test1, y_pred)
```

```
    ax.plot([0,8000],[0,8000], 'r--', label='Perfect Prediction')
    ax.legend()
    plt2.title("Scatter plot between y_test and y_pred")
    plt2.xlabel("y_test")
    plt2.ylabel("y_pred")
    plt2.tight_layout()
    plt2.show()
```

R Code:

Link:

https://drive.google.com/drive/folders/1fEhv4E9iAwJ_0kj6tYBUMH9QD2w8dn30?usp=sharing