Predicting Bike Rental Count

Saurav Roy

September 14, 2019

CONTENTS

LIST OF FIGURES

LIST OF TABLES

Note: All code snippets and plots pertain to the codes used in Python. The results obtained in R are almost similar to the results obtained in Python. They are not exactly equal because the split into train and test was randomly done in both R and Python.

CHAPTER I

INTRODUCTION

1.1. Problem Statement

A bike rental service is a system which allows users to rent bicycles and use them for a particular period of time. These bikes can be picked up from any particular “dock” and can then be returned at any other dock belonging to the same system. Users can typically purchase a service using an app on their phones and make the payment. They can use them for travelling to their work, hiking, road trips etc. This holds some advantages over using motorized vehicles like motorbikes, cars or buses, in that air and noise pollutions are reduced. It also leads to less congestions on the roads leading to much less traffic. As a result, people are able to reach their destinations much smoother and more efficiently. According to Wikipedia, roughly about 1000 cities worldwide have a bike-sharing program. Behind-the-scenes, the organizations running these bike-renting services have a lot on their plates, so to speak. The ability to predict the daily counts can certainly be beneficial to them in the long run, allowing them to manage their businesses in a more proficient and cost-effective fashion. This is the goal of the project, i.e., to use Machine Learning algorithms to predict the daily bike rental count depending on the weather and certain seasonal or environmental conditions.

1.2. Dataset

The dataset is a csv file having 731 observations and 16 features. The information is regarding the daily bike rental counts corresponding to years 2011 and 2012. It also has information about the weather conditions, humidity, wind speed etc. corresponding to each day from the aforementioned years. The following are the features:

1. *instant*: Record index
2. *dteday*: Date
3. *season*: Season (1: spring, 2: summer, 3: fall, 4: winter)
4. *yr*: Year (0: 2011, 1: 2012)
5. *mnth*: Month (1 to 12)
6. *holiday*: Whether day is holiday or not (0: not holiday, 1: holiday)
7. *weekday*: Day of the week (0 to 6, starting from Sunday)
8. *wokringday*: If day is neither weekend nor holiday, then 1, otherwise 0
9. *weathersit*: Weather
    a. 1: Clear, Few clouds, Partly cloudy
    b. 2: Mist + Cloudy, Mist + Broken clouds, Mist + Few clouds, Mist
    c. 3: Light Snow, Light Rain + Thunderstorm + Scattered clouds, Light Rain + Scattered clouds
    d. 4: Heavy Rain + Ice Pallets + Thunderstorm + Mist, Snow + Fog
10. *temp:* Normalized temperature in Celsius, using (t – t_min)/(t_max – t_min), where t_min = -8, t_max = +39
11. *atemp:* Normalized feeling temperature in Celsius, using (t – t_min)/(t_max – t_min), where t_min = -16, t_max = +50
12. *hum:* Normalized humidity, the values are divided to 100 (max)
13. *windspeed:* Normalized wind speed, the values are divided to 67 (max)
14. *casual:* Count of casual users
15. *registered:* Count of registered users
16. *cnt:* Count of total rental bikes including both *casual* and *registered*

CHAPTER II

EXPLORATORY DATA ANALYSIS

2.1 Introduction

The very first thing to do before starting work on any dataset is to explore the data. This means a lot in the context of data analysis because looking at data not only means just visualizing the data in terms of graphs, but also cleaning the data and making it ready for analysis. This process is referred to as **exploratory data analysis** in the field of data analytics. To do this, we look at certain features present in the dataset and convert them, if necessary, to their proper forms which can be fed to the machine. Not only this, we also remove a few features which we feel may not contribute to the overall prediction process.

2.2. Removal of features

Talking about removal of certain features, I have decided to go ahead and remove the following features from the dataset for reasons that have been mentioned below:

- *instant:* instant is just the record index or the serial number of the dataset, and thus has no effect on bike rental count.
- *dteday:* This is just the dates for the two years. While compiling this dataset, all dates have been split up into more useful features, like *mnth*, *holiday*, *weekday* and *workingday*. Moreover, keeping the date and using *datetime* to parse and format it only leads to complications in the program, which are totally avoidable.

- *casual* and *registered: casual* refers to the number of bike rentals that were made without any registrations. And *registered* refers to the number of people who actually registered for bikes and then made the rent. Here both casual and registered sum up to give us the total *cnt*. Now this can be approached in two ways:
  - One model can be made to predict *casual* and another separate model could be made to predict *registered* and then both these numbers could be summed up.
  - Both these variables can be dropped and one model could be built to predict the overall count.

  The reason I decided to go the second route is for ease of complexity. At the end of the day, both *casual* and *registered* are just features which can be summed up to get *cnt*. Also, when predicting for the future, a person would NOT think about how many casual and registered bike rentals were made. He would rather go for other important features like *season*, *holiday*, *temp*, *hum*, *weathersit* etc. While casual and registered are important for businesses who would like to analyze how different factors like weather and month affect the casual and registered counts separately, it is not important in this case. Keeping this in mind, I made the decision to drop *casual* and *registered* was made.

After dropping these variables, we are left with 731 observations and 12 features (11 independent + 1 dependent) in the dataset.

2.3. Conversion of features to proper forms

Out of the remaining 11 independent features, we have 7 categorical (season, yr, mnth, holiday, weekday, workingday, weathersit) and 4 continuous (temp, atemp, hum, windspeed) predictors. These 11 predictors need to be converted to their proper forms. The categorical features are in the form of levels and their datatype is of the *int* type, so we need to convert them to their actual forms, of the *object* datatype. As for the continuous variables, they are normalized, so we convert them to their actual values using the normalization formula as shown below in Table 2.1.

Features	Present form	Converted form
season	1, 2, 3, 4	Spring, summer, fall, winter
yr	0, 1	2011, 2012 (as strings)
mnth	1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12	Jan, Feb, Mar, Apr, May, Jun, Jul, Aug, Sep, Oct, Nov, Dec
holiday	0, 1	No, Yes
weekday	0, 1, 2, 3, 4, 5, 6	Sun, Mon, Tue, Wed, Thu, Fri, Sat
workingday	0, 1	No, Yes
weathersit	1, 2, 3, 4	Clear, Mist/Cloudy, Light Snow/Light Rain, Heavy Rain/Thunderstorm/Heavy Snow
temp	Normalized temp	[Normalized temp * ($T_{max} – T_{min}$)] + T_{min}
atemp	Normalized atemp	[Normalized atemp * ($aT_{max} – aT_{min}$)] + aT_{min}
hum	Normalized hum	Normalized hum * 100
windspeed	Normalized windspeed	Normalized windspeed * 67

Table 2.1. Conversion of features

2.1.3. Visualization

Once ***exploratory data analysis*** is done on the features, we can take a look at some of them (*weathersit*, *mnth*, *workingday*, *season*, *holiday*, *weekday)* and see how they each affect the final *cnt*. This is just a test to see if the count explicitly depends, more or less, on any one of the variables. This does not prove anything in the long run, as we are more concerned about finding out how all the variables combine to predict the overall count.

Here, I am using the vertical bars feature in *Tableau Desktop* to obtain the plots:

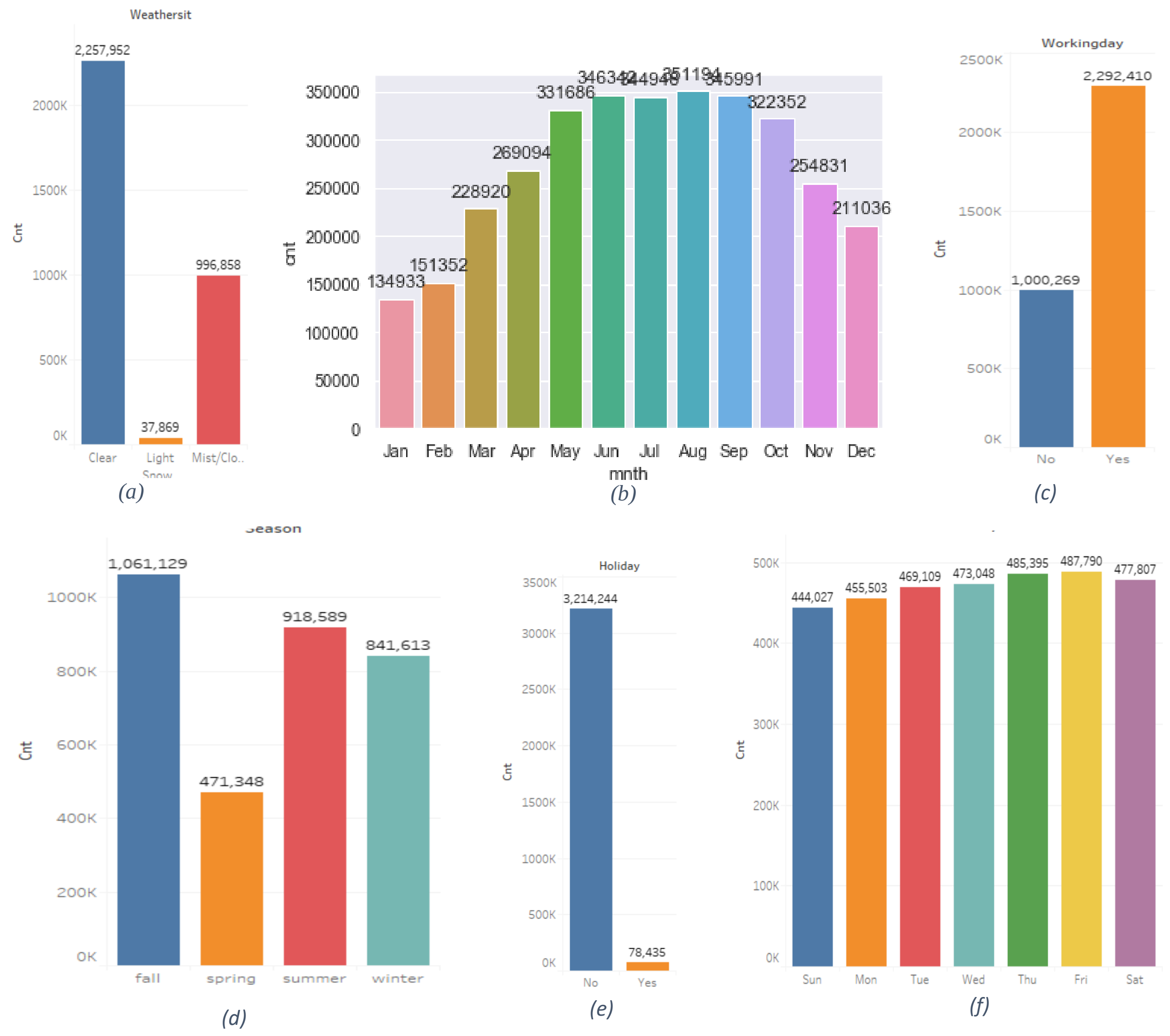
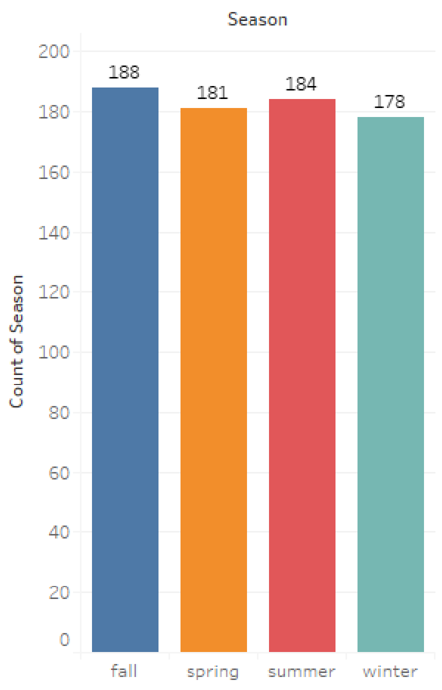


Fig. 2.1. Dependency of (a) weathersit, (b) mnth, (c) workingday, (d) season, (e) holiday, (f) weekday with cnt

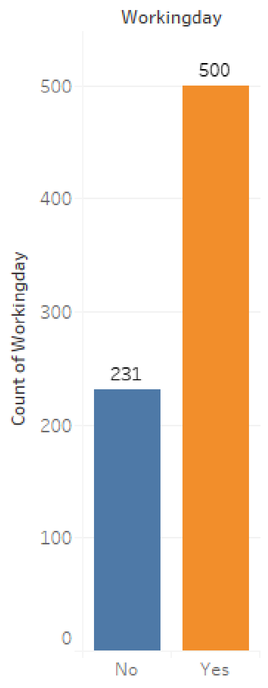
The key takeaways from the above plots are:

- The maximum rentals are made when the weather is *clear*, followed by *mist/cloudy* conditions. And very less people travel by bikes during *light snow/light rain*.
- While the difference in counts is less obvious while moving from months like June to July or August to September, it is more obvious for other pairs like February and March or October and November. Thus in some cases a change in month does not affect the bike rental count, whereas in other cases it does.
- More people are expected to rent bikes on *working* days versus *non-working* days as on working days, people are expected to commute more as compared to non-working days.
- The maximum number of bike rentals come during the *fall* season, followed by *summer*, then *winter* and finally *spring*. This data can be used as a rough estimator of when the bike rental company can do let's say, maximum advertising/promotion, to make maximum profits.
- The pattern of bike renting on holidays v/s non-holidays is very obvious as people generally tend to relax at home during holidays, whereas on non-holidays they are expected to rent bikes for commuting to their place of work, camping trips, grocery shopping, etc.
- The comparison of renting of bikes with the day of the week is more or less the same, when it comes to distribution. Going by the day of the week would be highly inconclusive for bike rental companies, as a change in the day of the week does not make much of a difference in the number of bikes rented.

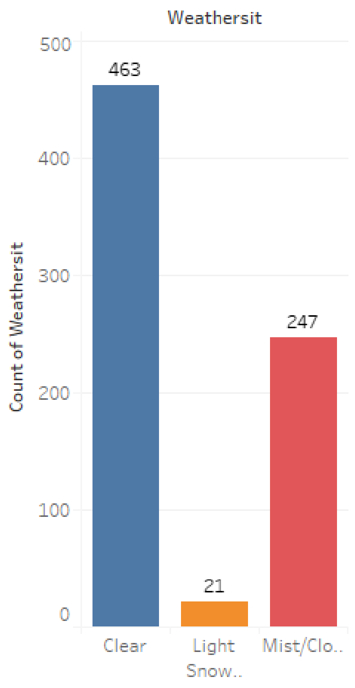
The next step would be to view the distribution of some *categorical* and *continuous* features present in the dataset. This is done to get a fair idea of the variables involved, and see if they are normally distributed or not, or skewed or not. These were plotted using Tableau Desktop.



(a)

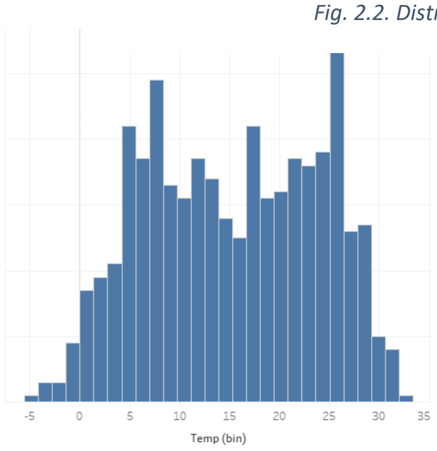


(b)

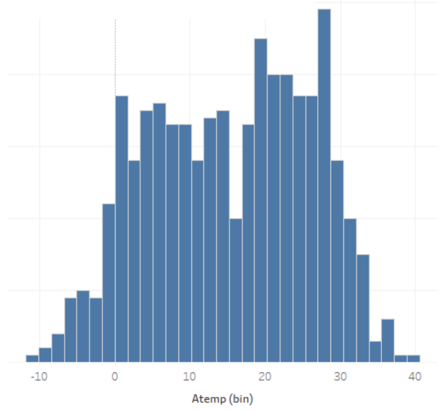


(c)

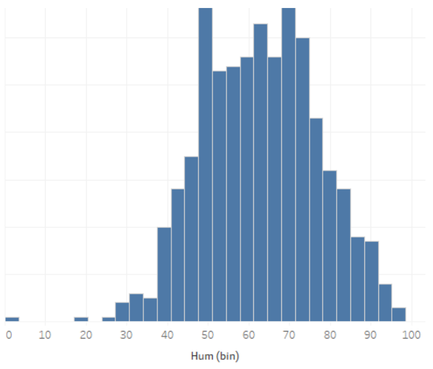
Fig. 2.2. Distribution of (a)season, (b) workingday, (c)weathersit



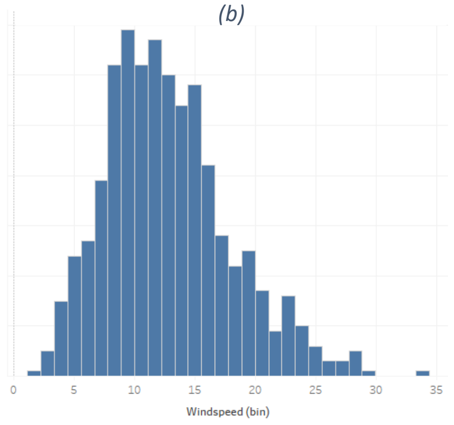
(a)



(b)



(c)



(d)

Fig. 2.3. Distribution of (a)temp, (b) atemp, (c) hum, (d) windspeed

The key takeaways from the above plots are:

- Seasons are fairly equally distributed. This is because data from two full years were chosen for this operation.
- The number of working days is more than the number of non-working days for the two years, which is fairly obvious.
- The number of days having clear weather were far greater in the dataset, followed by mist/cloudy conditions. Light snow/light rain had the lowest frequency in the dataset.
- While *temp* and *atemp* do not show any skewness, *hum* shows some skewness towards the right and *windspeed* shows some skewness towards the left. Also, *temp* and *atemp* are almost equal to each other when it comes to probability distributions, and that makes sense because on any given day, the actual temperature is almost equal to the feel temperature, more or less.

CHAPTER III

DATA PREPROCESSING

3.1. Outlier Analysis

Outliers in a dataset are nothing but data points that differ significantly from other observations. They almost always tend to cause anomalies in the results obtained after the application of Machine Learning algorithms. Outliers in a dataset may arise due to defective apparatus, data transmission errors, changes in system behavior, fraudulent behavior, human errors, etc. They may also arise due to false assumptions in the programmer or researcher’s theory. In statistics, outliers can be best detected by using boxplots to plot the features involved in a dataset. Let us plot the continuous features in our dataset and take a look at the presence, if any, of outliers.

The following boxplots were made using *matplotlib.pypot.boxplot* feature in Python:

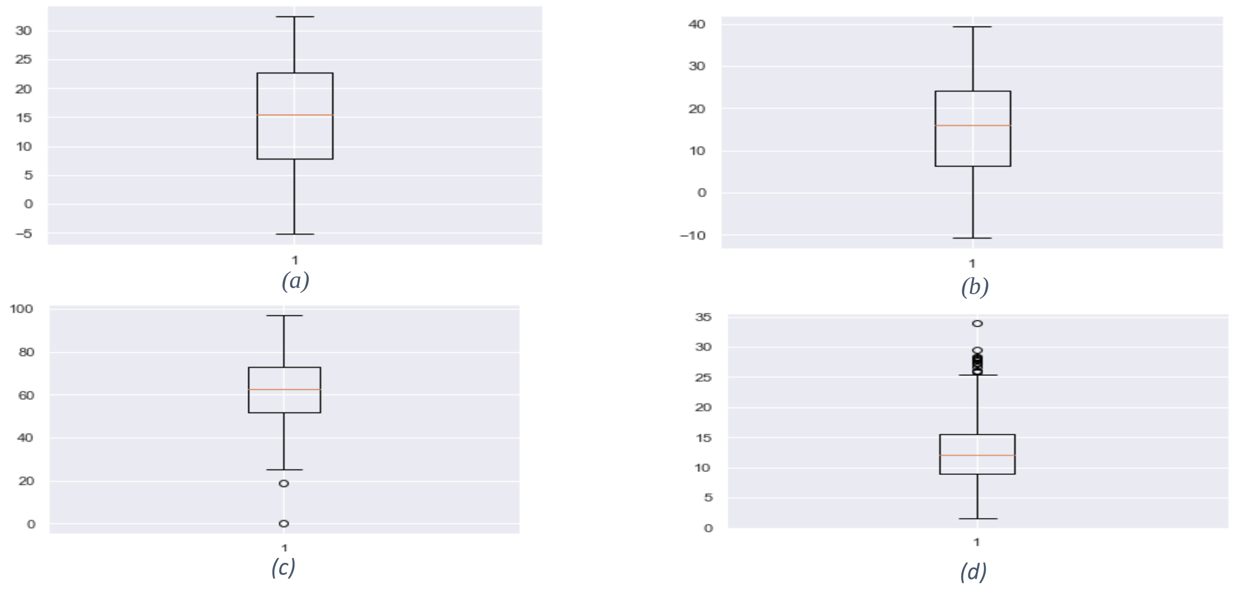


Fig. 3.1. Boxplot for outlier analysis for (a)temp, (b) atemp, (c) hum, (d) windspeed

From the above boxplots for *temp*, *atemp*, *hum* and *windspeed*, we see that *temp* and *atemp* do NOT have any outliers, so these are good to go. On the other hand, *hum* and *windspeed* have outliers in them and thus that needs to be fixed. The decision on whether to remove these outliers or replace them via any imputation method was made after running a few observational tests. Firstly, I used the following code in Python to check out the medians and percentiles for these two variables, and got the following results:

```python
#checking medians and percentiles
for i in ["hum", "windspeed"]:
    print(i)
    q75, q25 = np.percentile(day[i], [75, 25])
    print("q75 = ", q75, "q25 = ", q25)
    iqr = q75 - q25
    print("iqr = ",iqr)
    mini = q25 - (iqr * 1.5)
    maxi = q75 + (iqr * 1.5)
    print("minimum = ",mini, "maximum = ",maxi)
```

```
hum
q75 =  73.02085 q25 =  52.0
iqr =  21.020849999999996
minimum =  20.468725000000006 maximum =  104.55212499999999
windspeed
q75 =  15.6253715 q25 =  9.041649999999999
iqr =  6.583721500000001
minimum =  -0.8339322500000019 maximum =  25.50095375
```

Next, I used the following code to check the unique values present, using bins, and got these results:

```python
#checking unique values using bins
print(day["hum"].value_counts(ascending = True, bins = 5))
print(day["windspeed"].value_counts(ascending = False, bins = 7))
```

```
 (-0.0982, 19.45]      2
 (19.45, 38.9]        21
 (77.8, 97.25]       107
 (38.9, 58.35]       268
 (58.35, 77.8]       333
Name: hum, dtype: int64
 (10.786, 15.429]    253
 (6.143, 10.786]     225
 (15.429, 20.072]    123
 (1.467, 6.143]       59
 (20.072, 24.714]     54
 (24.714, 29.357]     15
 (29.357, 34.0]        2
Name: windspeed, dtype: int64
```

The boxplot for *hum* shows that it has 2 outliers, and both lie below 20.46, which is the minimum value. This fact is confirmed by the bins, as only 2 values lie in the interval (-0.0982, 19.45]. I decided to leave the outliers for *hum* untouched as there are only two of them and they will not massively impact the overall result.

However, *windspeed* has a lot more outliers. I ran a small test to check the number of outliers using the following code:

```
#checking how many outliers windspeed has
j = 0
for i in day["windspeed"]:
    if i > maxi:
        print(i)
        j += 1
print("\nTotal number of outliers =", j)
```

This code gave me all the outliers and they were 13 of them. 13 outliers out of a total number of 731 observations is not a bad thing, but I went ahead and fixed them just to get an improvement. First I removed all the outliers and made them empty values using *numpy.nan*. Next I made a known value *numpy.nan* and imputed using the median method and then method. **The *mean* method gave a more accurate result and so I froze that method**.

Thus, all 13 outliers were made empty values and then the *mean* imputation method was used to fill them out.

3.2. Feature Selection

The next important thing to do before proceeding any further is feature engineering or feature selection. Here we assess the importance of each predictor in our dataset and remove the ones which do not contribute a whole lot to the overall prediction process. There are two important things to keep in mind:

- All independent variables should have low correlation with each other. In layman terms, if two independent variables have high correlation, that means that they convey the same meaning and in such a case, one of them can be dropped without affecting the overall prediction.

- Every independent variable should have a fairly high correlation with the dependent variable. If this happens, that means that the dependent variable can be well explained by that independent variable. So in other words, if any independent variable has a low correlation with the dependent variable, it cannot explain the dependent variable well enough, and thus it can be dropped.

I used a **correlation plot** to check the correlation between the *continuous* variables. I got the following result:

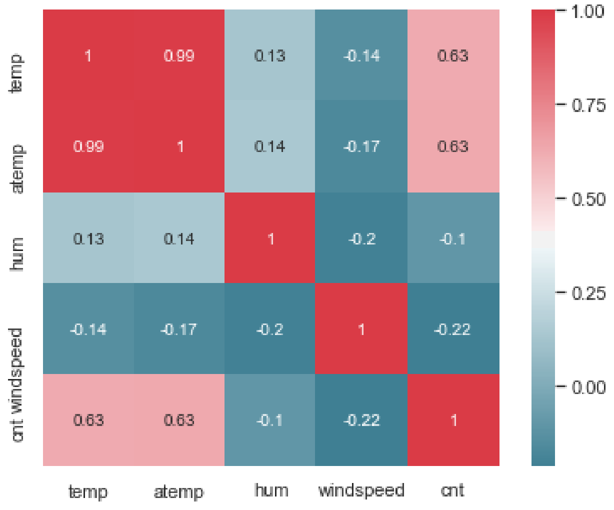


Fig 3.2. Correlation plot for continuous variables

One of the major takeaways from the above correlation plot is that *temp* and *atemp* are highly correlated, as their correlation value is 0.99. This makes sense because the feeling temperature will generally be closer to the actual temperature, and as such the feeling temperature on its own will not determine the bike rental count. **So *atemp* is dropped from the above system.**

Generally speaking, all other independent variables (*temp*, *hum*, *windspeed*) do not have high collinearity with each other. This indicates that these three do not depend on each other and can stay in that regard.

However, another aspect to consider here is the dependency of each independent variable with the dependent variable *cnt*. Here, *temp* has a dependency of 0.63 with *cnt*, which means that *temp* is an actual influencer on *cnt*, and it needs to stay. But the same cannot be said for *hum* and *windspeed*. Both these variables have very low correlation with *cnt*. This essentially means that *hum* and *windspeed* cannot really explain *cnt* well. **Owing to this, *hum* and *windspeed* can be dropped**.

Next I used the **chi-square test** to check the correlation between the c*ategorical* variables. I got the following results for the p-values:

```
Chi-square test for:  season
season : 0.0
yr : 0.9999288084245916
mnth : 0.0
holiday : 0.6831687073042003
weekday : 0.9999999999988407
workingday : 0.8865568364019545
weathersit : 0.02117930104473366


Chi-square test for:  yr
season : 0.9999288084245916
yr : 4.0118539920135064e-160
mnth : 0.9999999999999892
holiday : 0.9949247279855291
weekday : 0.9999995848511959
workingday : 0.9799434134248637
weathersit : 0.12737941480418666


Chi-square test for:  mnth
season : 0.0
yr : 0.9999999999999892
mnth : 0.0
holiday : 0.5593083086035064
weekday : 1.0
workingday : 0.9933495217919545
weathersit : 0.014637111771019139
```

```
Chi-square test for:  holiday
season : 0.6831687073042005
yr : 0.9949247279855291
mnth : 0.5593083086035064
holiday : 2.70694538395451e-153
weekday : 8.567055344615697e-11
workingday : 4.033370935452143e-11
weathersit : 0.6008572213448835


Chi-square test for:  weekday
season : 0.9999999999988407
yr : 0.9999995848511959
mnth : 1.0
holiday : 8.567055344615667e-11
weekday : 0.0
workingday : 6.775030505809736e-136
weathersit : 0.27845933074505175


Chi-square test for:  workingday
season : 0.8865568364019545
yr : 0.9799434134248637
mnth : 0.9933495217919545
holiday : 4.033370935452143e-11
weekday : 6.775030505809736e-136
workingday : 5.484935161027171e-160
weathersit : 0.2537639982644043


Chi-square test for:  weathersit
season : 0.02117930104473366
yr : 0.12737941480418657
mnth : 0.014637111771019139
holiday : 0.6008572213448835
weekday : 0.27845933074505175
workingday : 0.2537639982644043
weathersit : 0.0
```

When it comes to p-values, if p < 0.05 between any two variables, there is dependency between them. And when it comes to independent variables, we do NOT want that. We want two independent variables to be as independent from each other as possible. The following is a list of pairs of variables whose p-values which are less than 0.05.

```
mnth v/s season: 0.0
season v/s weathersit: 0.021
mnth v/s weathersit: 0.014
holiday v/s weekday: 8.57e-11
holiday v/s workingday: 4.03e-11
weekday v/s workingday: 6.77e-136
```

Judging from the above p-values, *weathersit* has dependency with *season* and *mnth*. It makes sense because based on the season and month, one can make a prediction without really knowing about the weather. However, I would like to keep *weathersit* because sometimes in real life, we can expect some kind of weird behavior in the weather. For example, here in Bengaluru during this time of the year, the weather can really be unpredictable at times. One moment it is as clear as day, the other moment it just rains cats and dogs. This kind of behavior can really impact the moods of people who are renting bikes, and bring about a vast change in the overall count. Therefore I would be keeping *weathersit*. I would also keep *mnth* because as can be seen in the plot of *mnth* vs *cnt* (Fig 2.1 (b)), the number of bike rentals of two consecutive months cannot really be explained well as in some cases the change is more as compared to other cases where the change is less.

Another important takeaway is that *holiday*, *weekday* and *workingday* seem to be highly dependent on each other. One or two of these may need to be dropped. Taking a look at how each of these three variables affect overall *cnt*, from Fig 2.1 (c), (e), (f) , we see that *holiday* and *workingday* seem to have a higher impact on *cnt* as opposed to *weekday*. So *weekday* can be eliminated out of these three as it does not have any conclusive influence on *cnt*. Therefore I would be keeping *holiday* and *workingday* but ***weekday* needs to be dropped from the dataset**.

Thus after feature selection, we are left with 7 predictor variables (*season, yr, mnth, holiday, workingday, weathersit, temp*) and 1 dependent variable (*cnt*).

3.3. Feature Scaling

The next step in data preprocessing is known as Feature Scaling. Typically, this is done when we have continuous independent variables having different ranges. For example, let us say we have an *age* variable in our dataset, ranging from 0-99 and an *income* variable which ranges from 10000 – 400000. This biases the output heavily towards *income* because it has higher ranges, as compared to *age*. In such a scenario we scale both the variables to have an equal range.

In this dataset, we are actually left with one continuous variable. However, it is good practice to scale the data, so based on the distribution of the data, I would need to normalize or standardize the variable *temp*. As can be seen from Fig 2.3 (a), *temp* does NOT seem to be normally distributed. **Therefore I have chosen to go with normalization on *temp*.**

3.4. One Hot Encoding

One Hot Encoding refers to the process of splitting up non-binary categorical features (*season, yr, mnth, holiday, workingday, weathersit*) into multiple binary sub-features, with each sub-feature representing whether a certain category in the original feature is True or False (0 for False and 1 for True). In our case, this technique is better than *label encoding* where each category present in a non-binary categorical feature gets converted to levels. The problem with label encoding is that, since different numbers are present in the same column once the features have been encoded, the model misunderstands the data as being in some kind of order ($0 < 1 < 2$ for example), which is erroneous as the categories involved in our dataset are nominal in nature. While in theory, one hot encoding makes the dataset "fat", and the dimensionality and the computational time are increased, in our case this is not a problem as the algorithms used here are fairly non-complex.

After one hot encoding is done, we are left with **27 variables**. The justification for this is:

- **season** has 4 types (fall, spring, summer, winter)
- **yr** has 2 types (2011, 2012)
- **mnth** has 12 types (Apr, Aug, Dec, Feb, Jan, Jul, Jun, Mar, May, Nov, Oct, Sep)
- **holiday** has 2 types (No, Yes)
- **workingday** has 2 types (No, Yes)
- **weathersit** has 3 types (Clear, Light Snow/Light Rain, Mist/Cloudy)
- **temp** has 1 type
- **cnt** has 1 type

Adding all these up, we get 27 variables (26 predictor variables + 1 dependent variable).

3.5. Train/Test Splitting

This is the final step before applying Machine Learning algorithms on our dataset. The dataset needs to be split into training data and testing data. The reason for this is simple. The training set contains a known output so we train the model on this dataset. Once the model learns on this data, we validate it on the test data by predicting the output of the test data and checking its accuracy by comparing the output with the actual output of the test data.

Here, I have used train_test_split from sklearn.model_selection and used simple random sampling to split up the dataset into 80% train and 20% test data.

CHAPTER IV

APPLICATION OF MACHINE LEARNING ALGORITHMS

4.1 Introduction

Problems in data science can be broadly classified into two types – classification and regression. This project predicts the overall bike count in a bike rental service, so this falls under the umbrella of a regression problem. The dependent variable (*cnt*) is ordinal in nature because order matters. 250 is definitely higher than 150. It is also interval in nature because there is a fixed interval between any two points. For example, the difference between 3 and 6 is the same as the difference between 23 and 26, i.e., 3 units. It can also be considered as being of a ratio type because it also has an absolute zero – 0 in this case means no bike rentals were made on a particular date. The following Machine Learning Regression algorithms were applied on the dataset:

- Decision Tree Regression
- Random Forest Regression
- Linear Regression

Let us consider each of the above algorithms and see the accuracies involved.

4.2. Decision Tree Regression

Decision tree-algorithms fall under the category of supervised machine learning algorithms. Using the CART algorithm (Classification and Regression trees), it can be used for both classification and regression models. It calculates the entropy for every predictor variable at each stage and then splits the predictor which has the maximum information gain. In theory, this holds true because the predictor variable that gives out the most information is chosen for splitting. This process continues until no further splits can be done.

The *sklearn.tree* module was imported and *DecisionTreeRegressor* was imported from there. Two decision tree models were made, one with all default parameters and another with max_depth of 2 (this means that splitting was done only twice), just to see the differences. Once the model was used to learn on the training data, it was applied on the testing data. This is the code I used in Python:

```
#importing from library
from sklearn.tree import DecisionTreeRegressor

#making the 1st model
DT_model = DecisionTreeRegressor().fit(train.iloc[:, 0:26], train.iloc[:, 26])

#applying 1st model on test
DT_pred = DT_model.predict(test.iloc[:, 0:26])

#making the 2nd model
DT_2_model = DecisionTreeRegressor(max_depth = 2).fit(train.iloc[:, 0:26], train.iloc[:
, 26])

#applying 2nd model on test
DT_2_pred = DT_2_model.predict(test.iloc[:, 0:26])
```

Finally the MAPE (Mean Absolute Percentage Error) was calculated for each model. The decision tree with default parameters gave an MAPE of 20.10% (accuracy = 79.90%) and the decision tree with max_depth of 2 gave an MAPE of 26.41% (accuracy = 73.59%).

4.3. Random Forest Regression

A random forest algorithm works on the principle of ensembling, in that a group of "weak" learners come together to form a "strong" learner. This essentially means that multiple learning algorithms are used to attain better performance as opposed to results from one single algorithm. The output from one tree is fed as input into the next tree. Thus multiple decision trees are used to learn on the training data and the results obtained from all of them are combined to get maximum possible accuracy.

The *sklearn.ensemble* module was imported and *RandomForestRegressor* was imported from there. Initially I started out with 200 trees (*n_estimators = 200*), and then continued by increasing the number of trees. I got the optimal accuracy at 400 trees (*n_estimators = 400*). This is the code I used in Python:

```python
#importing from library
from sklearn.ensemble import RandomForestRegressor

#making the model
RF_model = RandomForestRegressor(n_estimators = 400).fit(train.iloc[:, 0:26], train.iloc[:, 26])

#applying model on test
RF_pred = RF_model.predict(test.iloc[:, 0:26])
```

Upon calculations, the MAPE (Mean Absolute Percentage Error) turned out to be 16.58% (accuracy = 83.42%). This is better than using a Decision Tree Regressor but let us see if the results could be improved by using Linear Regression.

4.4. Linear Regression

A linear regression model is another supervised machine learning algorithm which inspects the linear relationship between two or more variables. Our dataset has multiple independent variables, so a Multiple Linear Regression model is applied here. The regression equation behind this is:

$$y = b_0 + b_1x_1 + b_2x_2 + \ldots$$

where, y is the dependent variable, x_1, x_2,... are the dependent variables, b_1, b_2... are their respective slopes, and b_0 is the y-intercept. Each slope represents the effect of the respective variable on the dependent variable, in other words, if x_1 increases (or decreases) by 1 unit, y increases (or decreases) by b_1 units. And if all the dependent variables are 0, then y is equal to the intercept, b_0.

Using this equation, we build a model on the training data and try to find the line of best fit, which minimizes the distance between the actual values and the predicted values of the dependent variable.

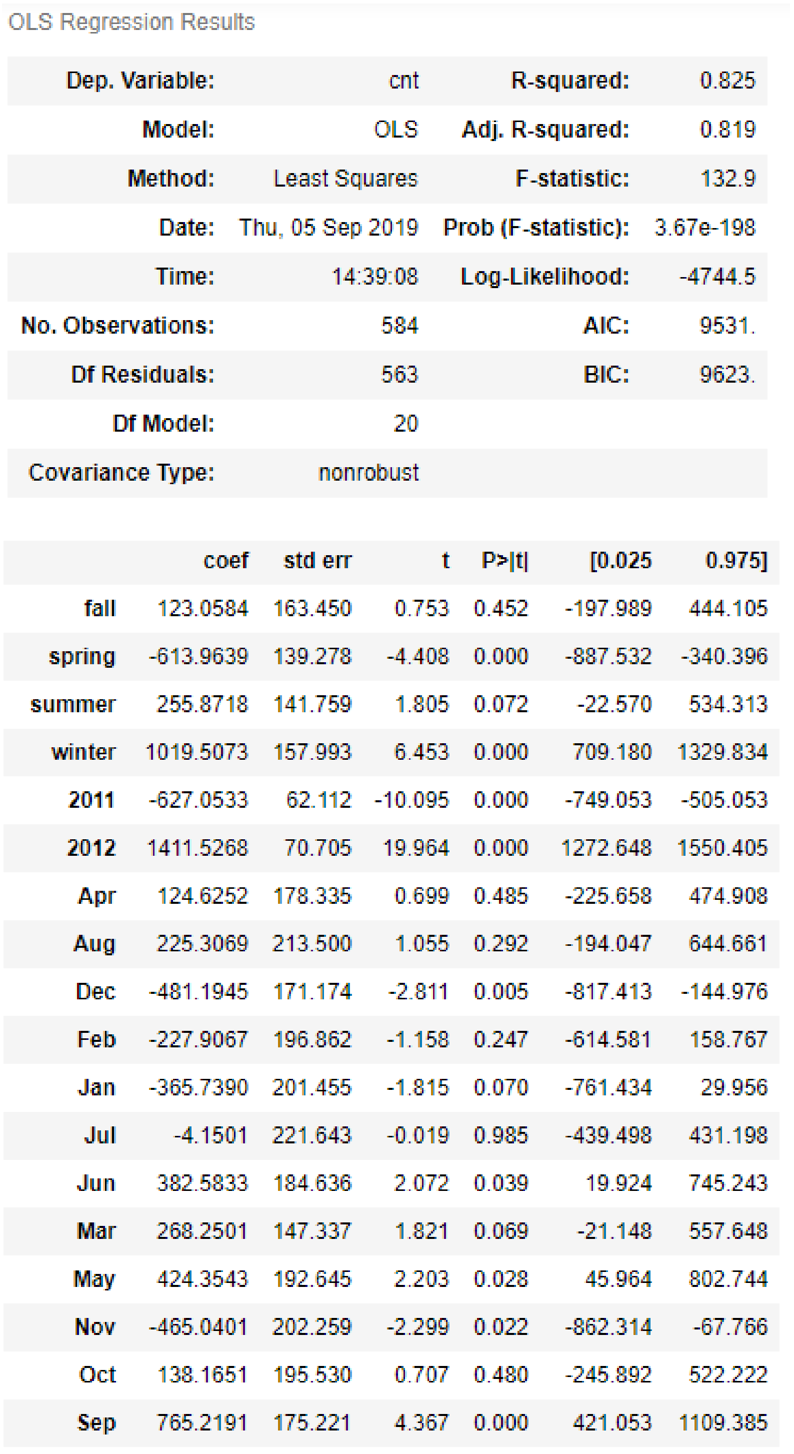
The *statsmodels.api* module was imported for the regression. This is the code I used in Python:

```python
#importing from required library
import statsmodels.api as sm

#building model on train
LR_model = sm.OLS(train.iloc[:, 26], train.iloc[:, 0:26]).fit()

#predicting on the test values
LR_pred = LR_model.predict(test.iloc[:, 0:26])
```

The MAPE (Mean Absolute Percentage Error) turned out to be 15.62 % (accuracy = 84.38%). This algorithm gave us the best results in terms of accuracy and so we are going to explore the summary of the model and try to interpret the output table.

4.5. Interpretation of Linear Regression Results

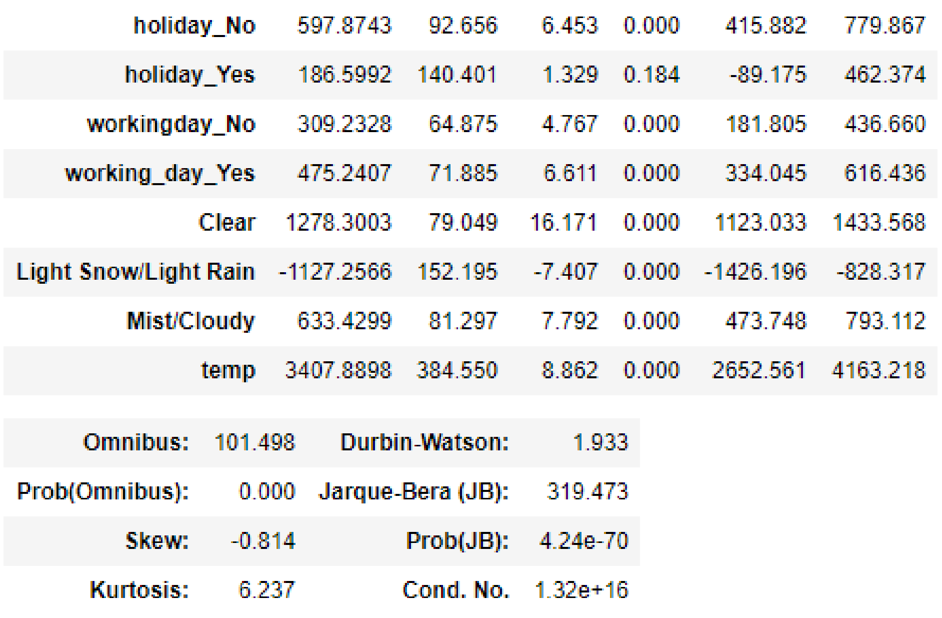
Now that we have frozen the Linear Regression algorithm, let us take a look at the summary of the model. This is the table that comes up in Python after using the summary() function:



OLS Regression Results

Dep. Variable:	cnt	**R-squared:**	0.825
Model:	OLS	**Adj. R-squared:**	0.819
Method:	Least Squares	**F-statistic:**	132.9
Date:	Thu, 05 Sep 2019	**Prob (F-statistic):**	3.67e-198
Time:	14:39:08	**Log-Likelihood:**	-4744.5
No. Observations:	584	**AIC:**	9531.
Df Residuals:	563	**BIC:**	9623.
Df Model:	20		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
fall	123.0584	163.450	0.753	0.452	-197.989	444.105
spring	-613.9639	139.278	-4.408	0.000	-887.532	-340.396
summer	255.8718	141.759	1.805	0.072	-22.570	534.313
winter	1019.5073	157.993	6.453	0.000	709.180	1329.834
2011	-627.0533	62.112	-10.095	0.000	-749.053	-505.053
2012	1411.5268	70.705	19.964	0.000	1272.648	1550.405
Apr	124.6252	178.335	0.699	0.485	-225.658	474.908
Aug	225.3069	213.500	1.055	0.292	-194.047	644.661
Dec	-481.1945	171.174	-2.811	0.005	-817.413	-144.976
Feb	-227.9067	196.862	-1.158	0.247	-614.581	158.767
Jan	-365.7390	201.455	-1.815	0.070	-761.434	29.956
Jul	-4.1501	221.643	-0.019	0.985	-439.498	431.198
Jun	382.5833	184.636	2.072	0.039	19.924	745.243
Mar	268.2501	147.337	1.821	0.069	-21.148	557.648
May	424.3543	192.645	2.203	0.028	45.964	802.744
Nov	-465.0401	202.259	-2.299	0.022	-862.314	-67.766
Oct	138.1651	195.530	0.707	0.480	-245.892	522.222
Sep	765.2191	175.221	4.367	0.000	421.053	1109.385



holiday_No	597.8743	92.656	6.453	0.000	415.882	779.867
holiday_Yes	186.5992	140.401	1.329	0.184	-89.175	462.374
workingday_No	309.2328	64.875	4.767	0.000	181.805	436.660
working_day_Yes	475.2407	71.885	6.611	0.000	334.045	616.436
Clear	1278.3003	79.049	16.171	0.000	1123.033	1433.568
Light Snow/Light Rain	-1127.2566	152.195	-7.407	0.000	-1426.196	-828.317
Mist/Cloudy	633.4299	81.297	7.792	0.000	473.748	793.112
temp	3407.8898	384.550	8.862	0.000	2652.561	4163.218

Omnibus:	101.498	Durbin-Watson:	1.933
Prob(Omnibus):	0.000	Jarque-Bera (JB):	319.473
Skew:	-0.814	Prob(JB):	4.24e-70
Kurtosis:	6.237	Cond. No.	1.32e+16

Let us take a look at few of the important values here:

- **Dep variable** tells us what the dependent variable is. It is *cnt* in our case. **Model** is *OLS* which stands for Ordinary Least Squares. **Method** is *Least Squares* here, and it basically means that the "line of best fit" here tries to minimize the squares of the distances between the predicted values and the actual values.
- **No. Observations** is the number of rows or observations in our training data after splitting, which is 584. **Df Residuals** represents the degrees of freedom of the residuals, in other words, it is the difference between number of observations and number of parameters. This is 563. **Df Model** is the number of parameters in the model, not including the constant term if present. Here it is 20.
- **R-squared** is the coefficient of determination here. It is a measure of how well the model can explain the variance in the predicted values. It lies between 0 and 100%. A higher value of R-squared means that the model fits the data very well. This is 0.825 in our case, so this means that 82.5% of the variations in the output is explained by out input variables. **Adjusted R-squared** is similar to the R-squared but the difference is that it accounts for the addition of new variables or the presence of existing variables. So for example, if there is only one independent variable, both

R-squared and adjusted R-squared will be same. However, if new variables are added, R-squared might stay the same or it might increase but adjusted R-squared value will decrease if the new variables are non-significant ones. So basically, adjusted R-squared acts as a penalizer for adding new variables that do not contribute to the improvement of the existing model. Thus, adjusted R-squared is always a better judgement for goodness of model than R-squared. In our case it is 0.819, which is fairly close to the R-squared value of 0.825, indicating that the presence of non-significant variables are close to none.

- **F-Statistic** is a measure of the significance of the fit. It comes from the F-test which compares the model that we specify to a model with no predictors, i.e., an intercept-only model. While the R-squared value establishes a measure of the strength of the relationship between our model and the target variable, the F-test says whether this relationship is statistically significant. The F-Statistic is the ratio of mean regression sum of the squares to the mean error sum of scores. And in our case this is 132.9. **Prob(F-Statistic)** is the probability of getting the above F-Statistic value, provided that the null hypothesis is true, i.e., all regression coefficients are zero. Here, it is very low (3.67e-198) meaning that there is almost zero chance that all regression coefficients are 0. This is a good thing as it ensures that almost all coefficients are non-zero and our regression equation validates the fit of the data.
- Also, **coef** represents the coefficient of each variable, **std err** is the standard error, **t** is the t-statistic value, **P>|t|** is its probability and then we have the **95% Confidence Intervals**. Let us take one variable *fall* to understand each term.

  So, **coef** for *fall* is 123.0584. This means that for 1 unit increase (or decrease) in *fall*, the dependent variable *cnt* increases (or decreases) by 123.0584 units. The **std err** (163.450) represents the level of accuracy in determing the respective coefficient. Also, **t** (0.753) is the t-statistic value which is a measure of how statistically significant each variable is (*fall* in this case), as compared to the F-Statistic which is a measure of statistical significance of the entire model. **P>|t|** (0.452) is the probability that the null hypothesis is true (coeffiecient = 0) for *fall*. Here, the confidence level used is 95%, so if the P value is less than 0.05, that would indicate that the variable is statistically significant for the model. Here, it is more than 0.05, so *fall* would be a statistically insignificant variable. And finally we have the **confidence intervals**, which determine the range in which our coefficient is likely to fall (with 95% probability). So 123.0584 lies between, as we see, -197.989 and 444.105.

- Finally, we have the lowermost table:
    - **Omnibus** (101.498) is a measure of the skewness and kurtosis of the data, combined. **Prob(Omnibus)** (0) depicts the probability that the errors or residuals (a measure of skewness and kurtosis) are normally distributed.
    - **Skew** (-0.814) is a measure of the symmetry of the data about the mean. It is generally low if normally-distributed errors are distributed symmetrically about their mean. **Kurtosis** (6.237) represents the peakiness or curvature of the data, with higher curvature implying that the residuals are clustered tightly around zero, making the model a good one with as few outliers as possible.
    - **Durbin-Watson** (1.933) is a test for the homoscedasticity in the model. In layman terms, it checks for the presence of correlation in the residuals. It is expected to lie between 1 and 2.
    - **Jarque-Bera (JB)** (319.473) is another test for the presence of skewness and kurtosis. **Prob(JB)** (4.24e-70) is the probability for the JB test.
    - **Cond. No.** (1.32e+16) is a test for multicollinearity among independent variables. Higher values indicate that input variables are collinear and very small changes in the input variables have huge effects on the dependent variable. We do not want that as we want our independent variables to be as independent from each other as possible.

CHAPTER V

CONCLUSION

Initially a dataset having 16 variables (15 independent +1 dependent) was narrowed down to 8 variables (7 independent + 1 dependent) after some exploratory data analysis and data preprocessing. One hot encoding was done on the categorical variables and thus the dataset ended up with 27 variables (26 independent + 1 dependent). The continuous variables were also converted to their actual values, and then scaled by normalization. Then the dataset was split 80-20 into train and test. Three different models were applied on the train data and then validated on the test data. Out of these, the Linear Regression model gave the most accurate result (MAPE = 15.62%).

While the work seems to be mostly done as accuracy, R-squared and adjusted R-squared were all above 80% indicating a good model and a good “line of best fit”, there is still a lot left to desire. For example, the omnibus value needs to be as close to zero as possible as that would indicate normalcy, or less skewness and kurtosis, but it is 101.498 in this model. After that, the probability of omnibus should be as close to 1 as possible, so that there is a higher probability of the omnibus being closer to zero. Skew should also be closer to 0, so our model could be helped somewhat by some changes. Also, higher kurtosis implies the model is good with few outliers, so there is another scope for improvement in that regard. These are but only a few examples where our model needs to be improved.

Thus the Ordinary Least Squares method and the Linear Regression model gave good results but there is certainly room for improvement. It would be better to make some changes in the parameters involved and try to improve the aforementioned features. Also, 8 variables were dropped from the dataset for reasons that have been mentioned, so another way of going about it could be to make some changes there. For example, dropping *holiday* or *workingday* instead of *weekday* could be tried, or dropping *mnth* could also be tried. During feature selection, *hum* and *windspeed* were dropped too. So another model could be made where these variables could be kept and some others could be dropped.

All in all, understanding how the data behaves and making appropriate changes as we go along is how proper analyzing should be done, and therefore changes could be made here and there in bits and pieces to improve the overall model accuracy.

References

- McKinney, W. (2012). *Python for data analysis: Data wrangling with Pandas, NumPy, and IPython*. "O'Reilly Media, Inc.".
- Grolemund, G., & Wickham, H. (2017). *R for data science: import, tidy, transform, visualize, and model data*. "O'Reilly Media, Inc.".