Mid-Term Project Report - Bike Sharing Demand

(Kaggle Competition)

SUBMITTED BY:

Anubhav Bhatti

Student ID: 20169150

STAT 857 Statistical Learning- Winter 2020

Mid Term Project

Table of Contents

1 Bike Sharing Demand (Kaggle Competition)

1.1 Background and Problem Definition

Bike Sharing Demand is a Kaggle competition, in which we need to forecast bike rental demand by analyzing and combining historical usage patterns with weather and season data. According to Kaggle competition, “Bike sharing systems are a means of renting bicycles where the process of obtaining membership, rental, and bike return is automated via a network of kiosk locations throughout a city. Using these systems, people are able rent a bike from a one location and return it to a different place on an as-needed basis. Currently, there are over 500 bike-sharing programs around the world.” [1]

Bike sharing services consists of a large number of bikes located at multiple locations in a city. Riders can rent these bikes from docking stations and can return to a docking station after usage. Due to a large number of daily renters and dynamic nature of bike demand, it is very important to forecast hourly bike demand to maintain the sufficient amount bikes at a station. It will also help in designing and expanding the bike share service.

However, bike demands are very dynamic in nature and depend hugely on external factors like weather, holidays, and office timings. So, in the competition, we need to design n machine learning algorithm that can accurately predict the demand of bikes at a particular hour on a particular day. According to the rules, the machine learning algorithm can only use information which was available prior to the time for which it is forecasting. For this competition, Kaggle given us the dataset provided by Hadi Fanaee Tork using data from Capital Bikeshare. Captial Bikeshare is bike share service in Washington DC. They also provide real-time trip history data to developers, statisticians, and other interested members for analysis, development, and visualization.

1.2 Objective

The objective of my project is to train a model on a challenging mid to large-scale dataset that can predict the number of rented bikes on a day at a particular time. Based problem statement, I understand that it is regression problem on a time series data. Based on my preliminary analysis, regression algorithms like support vector regression, random forest regression, and boosting techniques can be used to forecast the count of rented bikes.

1.3 Data Description

As mentioned earlier, the dataset is created using the data provided by from ‘Captial Bikeshare’ by Hadi Fanaee [2]. The dataset is hosted at UCI machine learning repository. The dataset contains hourly and daily count of rental bikes between the year 2011 and 2012 in Captial Bikeshare system with corresponding weather and seasonal information. The dataset has 17389 number of rows (i.e., number of instances), and has 12 attributes (features). The training dataset comprises of 10887 entries and 12 attributes. The test dataset comprises of 6494 entries and 9 attributes. The training dataset comprises of first 19 days of each month while the test dataset comprises entries from 20th day to the end of the month.

As per the rules, we must predict the total count of bikes to be rented during each hour covered by the test dataset, by only using information available to the rental period.

The train dataset has the following attributes [6]:

1. Datetime: Datetime attribute contains the hourly date and timestamp information of each entry in the data set. The values of this attributes are of format: “dd-mm-yyyy hh:mm:ss”
2. Season: This attribute contains the season information in a year i.e., 1 = Spring, 2 = Summer, 3 = Fall, 4 = Winter.
3. Holiday: This attribute contains information about if the day is considered a holiday or not. This attribute has binary values (0: not holiday, 1: holiday)
4. Workingday: This attribute contains information about if the day is neither a holiday nor a weekend. This attribute has binary values (0: not working day, 1: working day).
5. Weather: It contains information about the weather conditions.
6. Temp: Contains temperature information in Celsius. The values are 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)
7. Atemp: Contains “feels like” temperature values in Celsius. 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)
8. Humidity: Contains normalized relative humidity values. The values are divided by 100 (max).
9. Windspeed: Contains normalized windspeeds. The values are divided by 67 (max)
10. Casual (Dependent Variable): Contains the count of non-registered user rentals initiated.
11. Registered (Dependent Variable): Contains the count of registered user rentals initiated.
12. Count (Dependent Variable): Contains the count of total number of user rentals initiated (casual + registered).

Since, we have good idea of the dataset attributes and its independent and dependent variables, we can start exploring the dataset further. For this, we will try to identify individual and combined relationship between the dependent variables and dependent variables. From the above description of the dataset, we can understand that attributes ‘casual’, ‘registered’, and ‘count’ are the dependent attributes while all other are independent attributes. By analyzing the values of ‘casual’ and ‘registered’ we came to know that attribute ‘count’ is just the sum of two attributes. So, we would have to consider these three attributes as dependent variables.

1.4 Exploratory Data Analysis and Feature Engineering

Now, I would try to identify important patterns and would try to extract insights from the dataset so that we can better understand the role and significance of each attribute and its effect on the dependent variables. Before starting the data analysis, lets have a quick look at how our dataset is arranged. For convenience, I have merged the training and test datasets so that each transformation on every attribute is applied to training and test datasets equivalently.

1.4.1 Extracting Important Information from ‘Datetime’ Column

As we can observe from the first 5 rows of our dataset that the ‘datetime’ column has a lot of information that we could exploit. So, as a first step, we would split the ‘datetime’ and extract information like date, hour, month, and weekday (name of the day of a week e.g., Saturday).

By splitting the ‘datetime’ column we could extract the following new columns:

1. Hour: This column represents the hours on a particular date. The value of this variable varies from 0 – 23. Based on its nature, we can say that this variable is categorical and cyclic in nature since, ‘0’ and ‘23’ are closely related i.e., 12:00am and 11:00pm. So, we would either consider it as a categorical feature or will apply ‘sine’ and ‘cosine’ transformations such that our machine learning algorithm takes this attribute as a cyclic attribute.

2. Month: This column contains the name of the month on which the entry was registered e.g., 1: January, 2: February, etc. This is also a categorical variable.

3. Weekday: This column contains the name of day e.g., Saturday, Sunday, etc. This is a categorical variable.

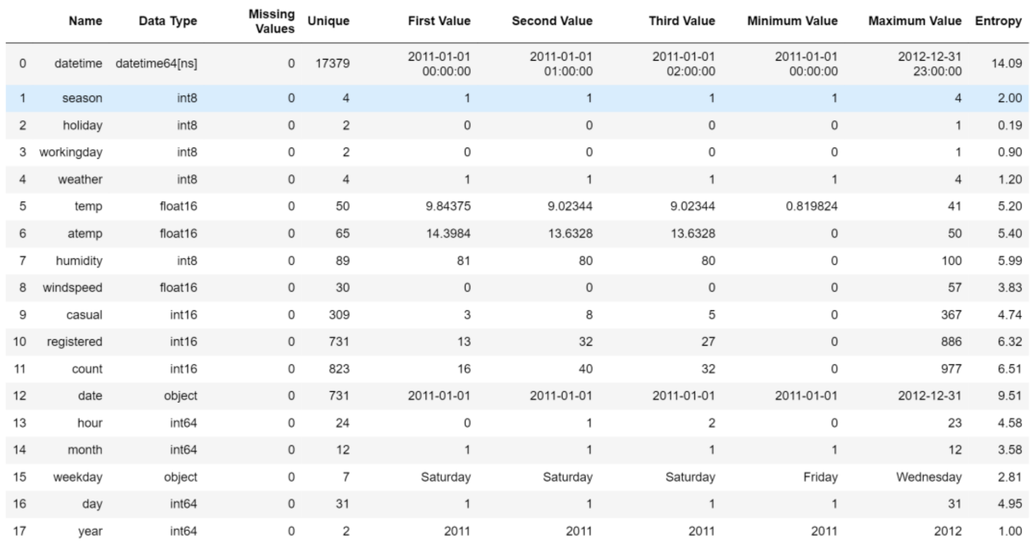
4. Year: This column contains the year in which the entry was registered.

1.4.2 Summary of Columns and Checking for Missing and Null Values in the dataset

Now that we have split the ‘datetime’ columns, we will have to check if there are any missing or null values in the dataset. Along with it, we can also, quickly analyze first, last, maximum, and minimum values of each column and their entropy.

Based on the table below, we can observe that there are no missing values in this dataset. However, there are couple of more interesting insights that we can extract from this. These points are mentioned below:

1. There are no missing values in this dataset. However, column ‘windspeed’ as a lot of values as ‘0’. This is unusual since, it is very rare when we have zero windspeed.
2. The following variables has a limited ‘unique values’ and hence they should be considered as categorical: season, holiday, workingday, weather, hour, month, weekday, day, year.
3. There are no negative values in the dataset i.e., there are no false entries in the dataset.



	Name	Data Type	Missing Values	Unique	First Value	Second Value	Third Value	Minimum Value	Maximum Value	Entropy
0	datetime	datetime64[ns]	0	17379	2011-01-01 00:00:00	2011-01-01 01:00:00	2011-01-01 02:00:00	2011-01-01 00:00:00	2012-12-31 23:00:00	14.09
1	season	int8	0	4	1	1	1	1	4	2.00
2	holiday	int8	0	2	0	0	0	0	1	0.19
3	workingday	int8	0	2	0	0	0	0	1	0.90
4	weather	int8	0	4	1	1	1	1	4	1.20
5	temp	float16	0	50	9.84375	9.02344	9.02344	0.819824	41	5.20
6	atemp	float16	0	65	14.3984	13.6328	13.6328	0	50	5.40
7	humidity	int8	0	89	81	80	80	0	100	5.99
8	windspeed	float16	0	30	0	0	0	0	57	3.83
9	casual	int16	0	309	3	8	5	0	367	4.74
10	registered	int16	0	731	13	32	27	0	886	6.32
11	count	int16	0	823	16	40	32	0	977	6.51
12	date	object	0	731	2011-01-01	2011-01-01	2011-01-01	2011-01-01	2012-12-31	9.51
13	hour	int64	0	24	0	1	2	0	23	4.58
14	month	int64	0	12	1	1	1	1	12	3.58
15	weekday	object	0	7	Saturday	Saturday	Saturday	Friday	Wednesday	2.81
16	day	int64	0	31	1	1	1	1	31	4.95
17	year	int64	0	2	2011	2011	2011	2011	2012	1.00

1.4.3 Predicting Zero Values of 'windspeed'

As we know that there are zero values in windspeed, we can use Random Forest Regressor with default values to predict these zero values and then replace zero values with them. To predict the values, we would use the attribute that can directly affect the values of windspeed. Hence, we would use the following attributes: "season", "holiday", "workingday", "weather", "weekday", "month", "year", "hour".

We would use the following hyperparameters for our random forest regressors: n_estimators=1200, max_depth=180, max_features='auto'

1.4.4 Analysis of Outliers and Skewness in the column 'count'

After predicting the values in windspeed, we should now check if there are any outliers in the 'count' dependent variable. Also, we would check the distribution of the 'count' variable.

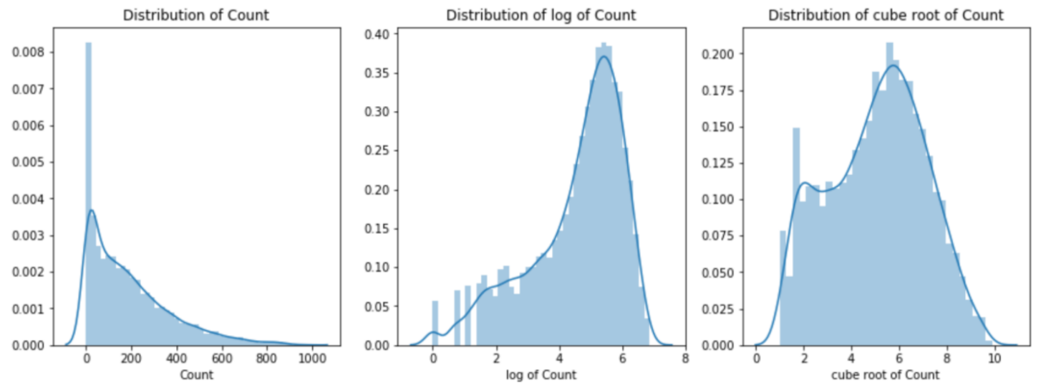


Fig. 1 a. Distribution of Count variable, b. Distribution of Log transformed Count variable, c. Distribution of Cube root transformed Count variable

The above fig. 1 (a) shows that the dependent variable 'count' is highly right skewed i.e., there are a lot of values on the right side of the curve making a tail on the right side. Skewness is an important issue that we need to address as this can make our algorithm make wrong predictions. To deal with the right skewed data we can apply the following transformations:

1. Log Transformation
2. Cube Root Transformation

As we can see, in the fig. 1 (a) and (b), that taking a log transformation improves the distribution of variable, but there is still skewness in the data, meanwhile, the cube root transformation greatly reduce the skewness in the data. However, during modelling, we got better results with log transformation of ‘count’ rather than cube root transformation of ‘count’. Hence, we dropped the cube root transformation root.

Now, we would analyze the outliers in the dependent variable ‘count’. To do that, we would plot the boxplots of count vs different independent variables. As we can see in the plots below that there are outliers in the ‘count’ variable. Also, we can observe following points from the plots below:

1. There are outliers in the variable ‘count’. However, the effect of outliers is greatly reduced when we apply log transformation on ‘count’.
2. The season ‘Spring’ has a smaller number of counts since there is a significant dip in the median. So, we can make a hypothesis that ‘temperature’ would play an important role in forecasting ‘count’ since, in spring outside temperature would not be favorable for bike renting and riders would prefer other modes of transport.
3. There are more outliers on a working day. From this we can conclude that these outliers are due to high demand of bikes on a working day and are not entered in the data erroneously. Hence, we would not remove these outliers as removing them will lead to more information loss. We would use log transformation to reduce the effect of these outliers.

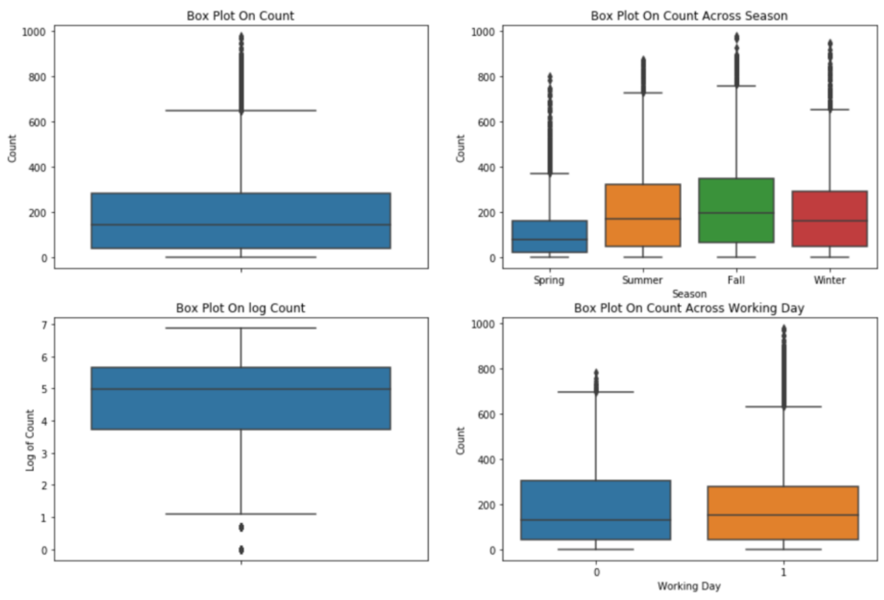


Fig. 2: Boxplot of ‘count’ and ‘log of count’ with ‘season’ and ‘working day’

1.4.5 Visualization of Count During a Day with respect to Independent Variables

Now, we would try to visualize that how does the count changes during a day with respect to different independent variables like season, day of a week, holiday, working day. Also, we would try to extract essential patterns and trends from this analysis and engineer new features based on those patterns. These engineered features would be very useful while training our algorithm and would help our algorithm to model these patterns to predict count vales more accurately.

Firstly, we would check how does the ‘season’ affects the count of bike rentals during a day. Note: Complete code of EDA and Feature Engineering can be found in the attached Html file named “Bike Data 4 EDA and Feature Engineering”. Based on the fig. 3, we can say that there is an increase in values of bike rentals during 7-8am and 5-7pm. This could mean that there is an increase in bike rentals during office/school timings. We can further investigate this hypothesis by studying the count with respect to ‘casual’ and ‘registered’ users as well as analyzing the count on weekdays and weekends. Based on our hypothesis, the count should go down on weekends during office/school timings.

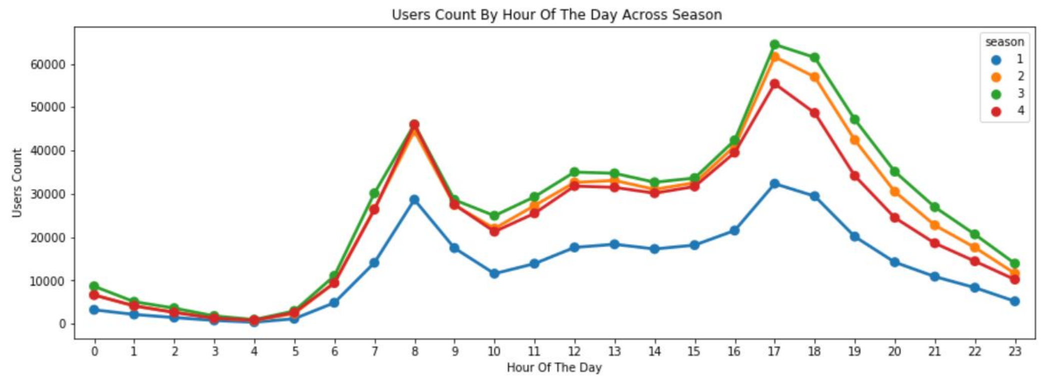


Fig. 3: Line plot showing User Counts by hour of a Day across Seasons

Also, as mentioned earlier (and confirming our hypothesis about spring season), it can be seen from the fig. 3 the count is low for season 1 i.e., spring season. Hence, the weather, especially, the temperature and humidity would play an important role in predicting the number of rental bikes since, people would prefer to rent a bike when the weather and temperature is warm or favorable.

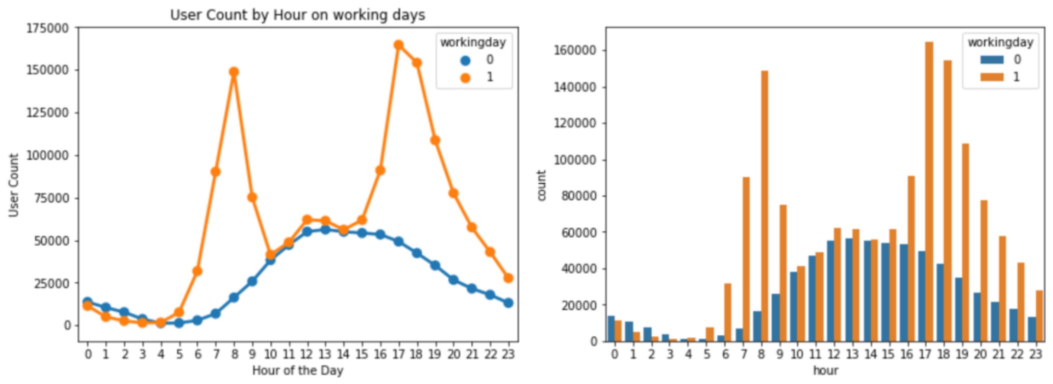


Fig. 4: plots showing the distribution of counts with working day (and non-working day) during a day.

Based on the fig. 4, we can say that there is peak in bike rentals at 7-8am and 5-7pm on a working day as well as there is a peak between 10am-3pm on non-working days. This is a very important pattern and we can extract this feature into a different column that would have value '1' during peak timings i.e., during 7-8am and 5-7pm on a working day and during 10am-3pm on non-working day. Same insight can be drawn from fig. 5, which shows that the peak timings on weekends and weekdays.

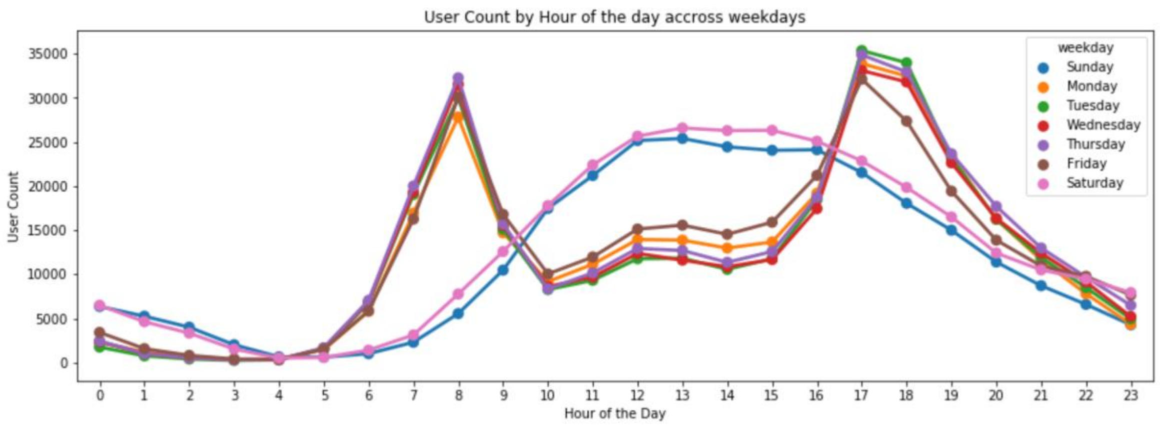


Fig. 5: Distribution of Bike rentals on Days of Week

Based on our hypothesis that the weather conditions like temperature, humidity, and windspeed would greatly impact the number of bike rentals on that day. Hence, we tried to visualize the distribution of bike rental count with respect to the windspeeds. We observed that, fig. 6, the count of bike rentals decreases significantly if the windspeeds are higher than 30-35. Hence, we created a new feature 'best condition' that has value '1' for each entry that has temperature less that 27 and windspeed less than of equal to 30. Also, for a working day, 'not favorable' when humidity is more than 60.

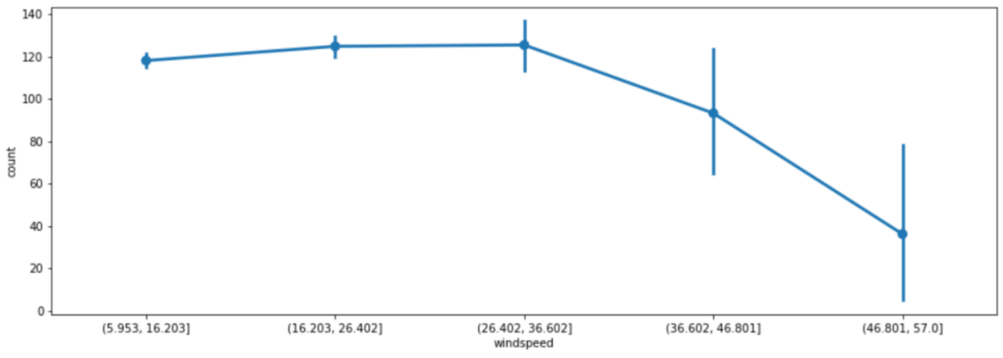


Fig. 6: Line plot showing distribution of 'count' with 'windspeed'

1.4.6 Feature Engineering based on Extreme Weather conditions and Holidays

- Major Holidays

Since, the major holidays in North America are usually not of a single day but are carried to previous as well as next day, for example, Christmas holiday is on 25th of December; however, people start preparing for it a week before. So, there would be a decrease in the count of registered users during that time. This happens similarly for other major holidays as well such as Thanksgiving, and New Year. Also, Tax Day in US is always working so I took that into account as well and marked 24th and 26th as holiday and as non-working day. Also, marked Tax Day as working day.

- Sandy Hurricane:

Also, since the dataset is collected for the city of Washington DC, by checking the weather conditions during the period of 2011 to 2012, I was able to identify that there was a state emergency going-on during the period of October 27- October 30 due to Sandy hurricane. This means that the bike rental would be at minimum during this time. Hence, for this time I marked this as a holiday and a non-working day.

With the above-mentioned feature engineering along with some more minor adjustments in the dataset, I was able to complete the Exploratory Data Analysis and Feature Engineering. Now, our dataset was ready to be used for modelling different machine learning algorithms.

1.4.7 Correlational Matrix

Based on the correlational matrix, we could draw the following insights:

1. We can observe that the 'count' and 'temp' are positively correlated, while 'humidity' and 'count' are negatively correlated. This is in line with our hypothesis that higher temperature means favorable conditions while higher humidity means not favorable conditions.
2. 'temp' and 'atemp' have very high correlation; hence, we can drop one of the columns during modelling.

1.5 Feature Selection

Now for feature selection, I deployed two different techniques to get an idea of what would be the most significant features in predicting the number of bike rentals i.e., count variable. Based, on these techniques I tried training different models to predict the 'count' variable and then selected the ones that help me get better scores on Kaggle. Techniques that I used for feature selection are mentioned below:

1.5.1 From Sklearn's Feature Selection "SELECTKBEST"

This technique is used to determine those features in our data that contribute most to the target variable. This technique is used for supervised machine learning algorithm to improve the estimators' accuracy scores and reduce overfitting in high dimensional data [3].

In this approach, initially, I used the following set of columns for subset selection:

```
fs_cols = ['season', 'holiday', 'workingday', 'weather', 'temp', 'humidity', 'windspeed', 'month', 'year', 'peak_time','
weekoff_count', 'best_condition', 'not_fav', 'sine_hr', 'cos_hr']
```

Using the algorithm in loop for k = 1 to 10, I tried to understand the best features at each level of K. From the analysis, I could conclude that following are the best 10 features for predicting the number of bike rentals:

Best parameters: ['season', 'temp', 'humidity', 'month', 'year', 'peak_time', 'weekoff_count', 'best_condition', 'sin e_hr', 'cos_hr']

Also, from the analysis, I could understand that feature 'hour' plays an important role in the prediction of 'count' variable i.e., it is the most significant feature.

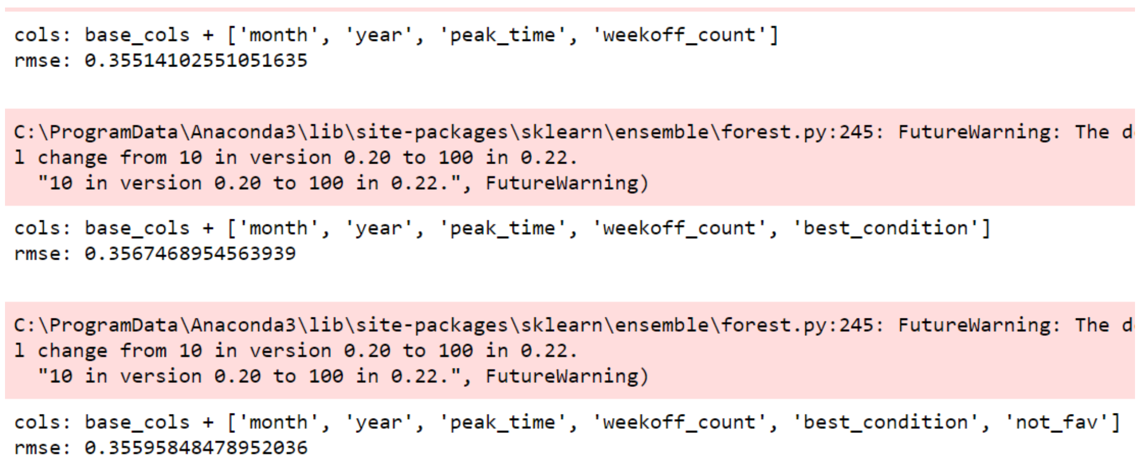
1.5.2 Forward Approach

In this, I used a random forest regressor to estimate the 'count' variable while adding columns to the base columns with forward approach. It is important to note here that to feature selection and fine tuning the hyperparameters I have used the complete train dataset for training and validation instead of just using the data that is available before the time at which the prediction is to be made. The reason for this was that using month batches and then fine tuning the models was very computationally expensive.

For base columns, I choose the following features:

```
base_cols = ['weather', 'temp', 'atemp', 'humidity', 'windspeed', 'holiday', 'workingday', 'season', 'sine_hr', 'cos_hr']
```

After this, I sequentially added one feature at time to the above base columns and then calculated the Root Mean Squared Logarithmic Error.



```
cols: base_cols + ['month', 'year', 'peak_time', 'weekoff_count']
rmse: 0.35514102551051635

C:\ProgramData\Anaconda3\lib\site-packages\sklearn\ensemble\forest.py:245: FutureWarning: The d
l change from 10 in version 0.20 to 100 in 0.22.
  "10 in version 0.20 to 100 in 0.22.", FutureWarning)

cols: base_cols + ['month', 'year', 'peak_time', 'weekoff_count', 'best_condition']
rmse: 0.3567468954563939

C:\ProgramData\Anaconda3\lib\site-packages\sklearn\ensemble\forest.py:245: FutureWarning: The d
l change from 10 in version 0.20 to 100 in 0.22.
  "10 in version 0.20 to 100 in 0.22.", FutureWarning)

cols: base_cols + ['month', 'year', 'peak_time', 'weekoff_count', 'best_condition', 'not_fav']
rmse: 0.35595848478952036
```

Fig. 7: Output of Forward Approach for Feature Selection

From the output of forward approach, I observed that by adding 'best_condition' to features increases the rmsle error for random forest regressor, while adding 'not_fav' feature to the feature set improves the rmsle error. Hence, I would keep 'not_fav' in my feature set for training Random Forest Regressor.

1.6 Modelling

For modelling, I have first trained the models on the complete training dataset (without considering only historical data). The reason for doing so was to get a clear picture of the accuracy of the models. Along with it, it was easier for performing hyperparameter tuning for different models. After selecting the best parameters for different models, I trained my models on the historical data only (as per the rules of Kaggle competition). Then I tried to fine tune my hyperparameters using the historical data. However, as anticipated, it was computationally very expensive, and it gave me memory error every time. So, I used the earlier best parameters only!

1.6.1 Hyperparameter tuning using Grid Search

1. **Model 1: Random Forest Regressor**:

Since, Random Forest Regressor is a very powerful, yet simple, ensemble learning method for regression problems I choose this as my first model to predict ‘count’ variable.

Firstly, I tried the random forest regressor with default parameters on my complete dataset (as mentioned above) to check how my model is performing on my features (base features along with engineered features). With default parameters and the features mentioned below I got a rmsle error of 0.466571.

```
rf_cols = ['weather', 'temp', 'atemp', 'windspeed', 'workingday', 'season', 'holiday', 'not_fav', 'hour', 'pea
k_time']
```

Then I performed the grid search to tune my hyperparameters using the following random grid around my default parameters:

```
n_estimators = [500, 1000, 1500]
min_samples_splits = [6, 8, 10, 12, 14]
best_score, best_params = np.inf, None
```

The parameters came out to be the following for random forest regressor:

```
best params: {'n_estimators': 1500, 'min_samples_split': 14, 'n_jobs': -1, 'random_state': 123}, rmse: 0.35832
28852913084
```

Then, I trained my model with best parameters and got the ‘rmsle error as 0.3581244629806633’. Hyperparameter tuning greatly reduced the rmsle error.

2. **Model 2: Gradient Boosting Regressor**:

After using a bagging technique (random forest regressor), I use a boosting technique called Gradient Boosting Regressor.

Same as before, I tried grid search with the following parameters:

```
x_cols = ['weather', 'temp', 'atemp', 'humidity', 'windspeed', 'holiday', 'workingday', 'season', 'hour', 'year', 'b
est_condition', 'not_fav']

rf_model = GradientBoostingRegressor()
# gradient boosting parameter grid
```

```
n_estimators = [150, 200, 500, 1000, 1500, 1800]
min_samples_leaf = [6, 8, 10, 12, 14]
learning_rate = [0.1, 0.01]
subsample = [0.5, 0.6, 0.7, 0.8]
```

The parameters came out to be the following for gradient boosting regressor:

```
best params: {'n_estimators': 500, 'min_samples_leaf': 8, 'learning_rate': 0.1, 'subsample': 0.7, 'random_state
': 123}, rmse: 0.3325215678106417
```

After training my model with these parameters and predicting it on the actual test set, the Kaggle score was 0.42099. However, after dropping the column 'not_fav' from the feature set, **the Kaggle score improved to 0.41872.** This means that our parameter tuning, and feature engineering were good. Now, I try improving this score by combining the predictions of random forest and gradient boosting.

3. **Model 3: Support Vector Regression (SVR):**

For training an SVR model, I used dummy encoded variables with default parameters. With default parameters, the SVR did not perform well on the test dataset, as I got a score of 1.10676. Then I tried tuning the parameters of SVR using the grid search. However, since SVR takes a lot of time to train, a comprehensive grid search was too computationally expensive. Hence, I could only tune the model for the parameter of C.

Then, I performed the **randomizedCV search** using the following parameters:

```
svr_cols = ['weather', 'temp', 'atemp', 'humidity', 'windspeed', 'holiday', 'workingday', 'season', 'hour', 'year', '
best_condition', 'not_fav']

regre_svr = SVR(verbose=True)
C = [float(x) for x in np.linspace(start = 0.5, stop = 2, num = 5)]
kernelstring = ['rbf']
gamma = ['auto']
epsilon = [0.2]
Best parameters: {'kernel': 'rbf', 'gamma': 'auto', 'epsilon': 0.2, 'C': 0.5}
```

However, the SVR model still could not perform well on the test data set, as Kaggle score I got with best parameters was 0.98756.

1.6.2 Best Model – A Combination of Gradient Boosting (80%) & Random Forest (20%)

Now after performing the grid search, I have the best parameters for gradient boosting regressor, random forest, and SVR.

Now, as per the rules, I defined a function that would only select the training data prior to date at which the prediction is being made. After that I trained and made predictions on the test data set using this model. The best score I could get was by training the random forest and gradient boosting individually with their own best parameters and their own set of features (different). Then I combined the predictions of gradient boosting and random forest in 80%-20% fashion. By doing so, I could get the **Kaggle score of 0.41490.** The code can be found in the html file attached named “Bike Data 4 Feature Selection and Model Training” under the section “Final Code”.

Since, SVR did not perform good on the test data set I did not combine the predictions of SVR with other models to get the final submission.

The parameters of the best models are as follows:

```
For Gradient Boosting:

gbm_cols = ['weather', 'temp', 'atemp', 'humidity', 'windspeed', 'holiday', 'workingday', 'season', 'hour', 'year',
'best_condition']
best_params = {'n_estimators': 500, 'min_samples_leaf': 8, 'learning_rate': 0.1, 'subsample': 0.7, 'random_sta
te': 123}

For Random Forest Regressor:
params = {'n_estimators': 1500, 'max_depth': 15, 'random_state': 123, 'min_samples_split': 14, 'n_jobs': -1}

rf_cols = ['weather', 'temp', 'atemp', 'windspeed', 'workingday', 'season', 'holiday', 'not_fav', 'hour', 'peak_tim
e']
```

1.6.3 Cross Validation

I tried to further train my model by using cross validation using the library RandomizedSearchCV from sci-kit learn. However, to train on different batches of historical data (using only prior training data) using cross validation was computationally very expensive and after hours of training I got “memory error” for each model. Hence, I could not further fine tune the models. The code along with errors that I received are included at the bottom of the html file named “Bike Data 4 Feature Selection and Model Training”.

1.6.4 Training with Dummy Encoding

Another approach that I deployed was to dummy encode the categorical values in the feature set and then feed it to the models. However, interestingly, the Kaggle score of both the models individually and combined could not surpass the Kaggle score of Best Model with best parameters.

With Dummy Encoding I got Kaggle score for each Model as following:

1. Random Forest Regressor: Kaggle Score: 0.47550
2. Gradient Boosting Regressor: Kaggle Score: 0.42257

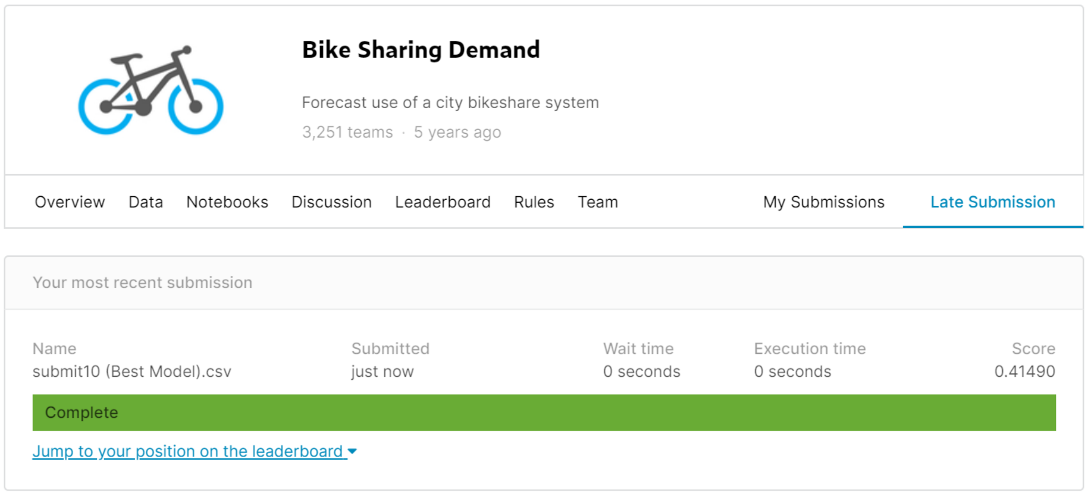
Note: All the code used to implement the above-mentioned algorithms along with other models with their predictions and Kaggle score in included in the attached html files. Other models with different parameters are not included in this report due to page limit restrictions.

1.6.5 Score Table

The Kaggle score table for each model individually and combine can be found below:

Serial No.	Machine Learning Model	Kaggle Score	Submission File Name
1.	Gradient Boosting and Random Forest Combined (80-20) [Best Parameters + Best Columns]	0.41490	submit10 (Best Model).csv
2.	Gradient Boosting [Best Parameters + Best Columns]	0.41872	submit7_1.csv
3.	Gradient Boosting [Best Parameters + Best Columns + column 'not_fav']	0.42099	submit6_1.csv
4.	Random Forest Regression [Best Parameters + Best Columns]	0.48258	submit9.csv
5.	Gradient Boosting [Best Parameters + Dummy Encoded Columns]	0.42257	submit12.csv
6.	Random Forest [Best Parameters + Dummy Encoded Columns]	0.47550	submit11.csv
7.	Support Vector Regression [Best Parameters + Dummy Encoded Columns]	0.98756	submit14.csv

Screenshot of best score (Kaggle):



Bike Sharing Demand

Forecast use of a city bikeshare system

3,251 teams · 5 years ago

Your most recent submission

Name	Submitted	Wait time	Execution time	Score
submit10 (Best Model).csv	just now	0 seconds	0 seconds	0.41490

Complete

1.7 References:

1. Bike Sharing Demand - Kaggle Competition. Link: https://www.kaggle.com/c/bike-sharing-demand/overview/description
2. Fanaee-T, Hadi, and Gama, Joao, Event labeling combining ensemble detectors and background knowledge, Progress in Artificial Intelligence (2013): pp. 1-15, Springer Berlin Heidelberg
3. Sci-kit learn Library: SelectKbest. Link: https://www.kaggle.com/jepsds/feature-selection-using-selectkbest?utm_campaign=News&utm_medium=Community&utm_source=DataCamp.com
4. LogicalGlass (Github). Link: https://github.com/logicalguess/kaggle-bike-sharing-demand/blob/master/code/main.ipynb
5. EDA & Ensemble Model (Top 10 Percentile). Link: https://www.kaggle.com/viveksrinivasan/eda-ensemble-model-top-10-percentile
6. Bike Sharing Dataset Data Set. UCI Machine Learning Repository. Link: http://archive.ics.uci.edu/ml/datasets/Bike+Sharing+Dataset