2022

BIKE SHARE SYSTEM ANALYSIS

BUSINESS ANALYTICS CAPSTONE PROJECT

THANH TU NGUYEN - 301146738

CENTENNIAL COLLEGE | Summer 2022

Contents

I. Introduction 3
1. Executive summary 3
2. Background 3
3. Problem Statement 4
4. Objectives & Measurement 4
5. Assumptions and Limitations 4
6. Data Set Introduction 5
7. Data Dictionary 5
II. Data Exploration 7
8. Data Exploration Techniques 7
9. Importing Comma-separated values or csv file using Panda 8
10. Data Cleansing 8
11. Data Exploration 8
12. Outliers analysis 16
13. Data Preparation and Feature Engineering 18
14. Feature Engineering 21
15. Summary 21
III. Model Exploration 22
16. Modeling Approach/Introduction 22
17. Train and split data 25
18. Linear Regression 26
18.1 Linear Regression for ‘count’ 26
18.2 Linear Regression of ‘casual’ 31
19.1 Linear Regression of ‘registered’ 37
19. Random Forest 43
19.1 Random Forest of ‘count’ 43
19.2 Random Forest of ‘casual’ 47
19.3 Random Forest of ‘registered’ 49
20. Polynomial Regression 52
20.1 Polynomial Regression of ‘count’ 52
20.2 Polynomial Regression of ‘casual’ 53
20.3 Polynomial Regression of ‘registered’ 55
21. Model Comparison 56

22. Model Selection 57
23. Validation and Governance 57
IV. Conclusion and Recommendations 63
24. Impacts on Business Problem 63
25. Recommended Next Steps 63

I. Introduction

1. Executive summary

Bike share system is a form of using shared public transportation that potential lessens impact on environment especially in urban areas where bicycles are the best alternative in response to climate emergencies. The bike share system allows users to rent the bikes from a dock for a short or long distances and return to another dock for a price. This idea has been spread around the cities in the world with 10 million bikes shared and 3000 bike sharing systems in 50 countries. In fact, according to PSBC Urban Solutions, bike share systems have not been popular until the last 10 years even though they have existed for more than 60 years (PBSC, 2022). There are several benefits and drawbacks of the systems regarding evironment friendly, fitness, and convenience; however, sometimes the riders meet the situation that there is no available bike at the docking station. Bike share is used publicly so the hygiene is a concern for a number of people (pedalergripcovers, 2020).

Bike share is not just a shared transportation system helping people to reach from point A to point B but also helps connect them to explore more places in the cities they are living or working (Bryce). It also has environment and economic implications – bike sharing leads to a reduced carboon footprint, and lower spending on environmentally harmful gases (Stott, 2020). In this report, Washington DC Capital Bikeshare dataset is used to analyze the number of riders in varied weather conditions.

2. Background

Narrowingly, Washington DC Capital Bikeshare is one of the largest bike share system in the USA with 5000 bikes at 600 stations in 7 jurisdictions or District of Columbia, Maryland, Virginia (DMV) area: Washington, DC; Arlington, VA; Alexandria, VA; Montgomery County, MD; Prince

George's County, MD; Fairfax County, VA; and the City of Falls Church, VA (Capital Bikeshare, 2022). Capital Bikeshare is operated by Motivate which was acquired by Lyft in July 2018 (Teale, 2018). The main competitors in the market are Blue Bikes, Bixi, Zagster, Lime, Mobike, Ofo and Spin. There are 3 ways to ride with Capital Bikeshare including single trip with $1 to unlock - only $0.05/min for a classic bike ride and $0.15/min for an ebike, 24-hour pass with $8 a day, and annual membership with $7.92 per month. Additionally, Capital Bikeshare Bike Key - $10, Capital Bikeshare Helmet - $16 are sold online (Capital Bikeshare, 2022).

3. Problem Statement

Given the dataset, there are several factors affecting to rental behaviors including weather conditions such as rain, storm, snow, day of week, season, linked with hour of the day to estimate number of casual and registed rentals and their duration using bike share system. In this situation, less rentals are likely to rent bike share; therefore, the solution for the rentals should be connected with temporary back-up transportations in DMV area including public transits or personal vehicles.

4. Objectives & Measurement

The objective is to increase the Capital Bikeshare meet the goal for 2023 revenue reaching $8,200,000 where the data spans from the years 2011 and 2012, and there has been a lot of changes from those years until now. Moreover, we may want to convert casual riders to registered riders.

5. Assumptions and Limitations

Washington DC Capital bikeshare system dataset is was a collected in 2013 by Laboratory of Artificial Intelligence and Decision Support with a period from 2011 to 2012, so this dataset is outdated and may not turn out good accuracy for analyzing. The dataset could make the model become more biased when we assume the difference of bike culture in Western and Eastern

countries with car-focused and walking distance, respectively. Therefore, bikeshare could be more popular in Eastern countries compared to the Western's.

There are a few hidden problems regarding limitations of this dataset where price of gas is not included. Assumingly, the gas price increases, more people tend to switch from driving cars to riding bikes. Another limitation is that the data only indicates how many rides but not how many people. In this case, the data cannot tell us the records were performed by a specific number of people but only number of rides that it can be performed by the same person. On the other hand, even though the price of bikeshare is available on the official website, the data cannot tell us whether the casual riders would spent more or less if they have converted to annual membership as registered riders.

Data Sources

6. Data Set Introduction

The dataset is donated on 20 December 2013 containing hourly and daily count of rental bikes between years 2011 and 2012 in Capital bikeshare system with the corresponding weather and seasonal information. It has 17389 instances and 16 attributes. The original was collected by Machine Learning Repository, Center for Machine Learning and Intelligent System - Laboratory of Artificial Intelligence and Decision Support (LIAAD), University of Porto (Fanaee-T, 2013).

7. Data Dictionary

The detailed data dictionary is shown in the table below:

Variables	Description
instant	Record index
dteday	Date

season	season (1:winter, 2:spring, 3:summer, 4:fall)
yr	year (0: 2011, 1:2012)
mnth	month (1 to 12)
hr	Hour (0 to 23)
holiday	weather day is holiday or not
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

II. Data Exploration

8. Data Exploration Techniques

The main programming language is used in this project is Python where several software libraries are applied. Numpy has functions for working in domain of linear algebra and arrays, Panda for data manipulation and analysis, matplotlib for plotting, seaborn for data visualization.

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns
```

Figure 1: Required library packages for data exploration

9. Importing Comma-separated values or csv file using Panda

```
hour_df=pd.read_csv('hour.csv')
hour_df.head()
```

✓ 0.1s Python

	instant	dteday	season	yr	mnth	hr	holiday	weekday	workingday	weathersit	temp	atemp	hum	windspeed	casual	registered	cnt
0	1	1/1/2011	1	0	1	0	0	6	0	1	0.24	0.2879	0.81	0.0	3	13	16
1	2	1/1/2011	1	0	1	1	0	6	0	1	0.22	0.2727	0.80	0.0	8	32	40
2	3	1/1/2011	1	0	1	2	0	6	0	1	0.22	0.2727	0.80	0.0	5	27	32
3	4	1/1/2011	1	0	1	3	0	6	0	1	0.24	0.2879	0.75	0.0	3	10	13
4	5	1/1/2011	1	0	1	4	0	6	0	1	0.24	0.2879	0.75	0.0	0	1	1

Figure 2: code for data uploading and first five row of the dataset

10. Data Cleansing

a. Columns in the dataset

```
hour_df.columns
```

✓ 0.4s Python

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

Figure 3: columns in the dataset

b. Renaming abbreviation names with full characters in the dataset

```
hour_df = hour_df.rename(columns={'weathersit':'weather_situation',
                                  'yr':'year',
                                  'mnth':'month',
                                  'hr':'hour',
                                  'hum':'humidity',
                                  'cnt':'count'})
hour_df.columns
```

✓ 0.8s Python

```
Index(['instant', 'dteday', 'season', 'year', 'month', 'hour', 'holiday',
       'weekday', 'workingday', 'weather_situation', 'temp', 'atemp',
       'humidity', 'windspeed', 'casual', 'registered', 'count'],
      dtype='object')
```

Figure 4: code for renaming abbreviation names with full characters and underscore for ‘weathersit’ column

11. Data Exploration

11. 1 Information of dataset

```
hour_df.info()
✓ 0.6s
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 17379 entries, 0 to 17378
Data columns (total 17 columns):
 #   Column      Non-Null Count  Dtype
---  ------      --------------  -----
 0   instant     17379 non-null  int64
 1   dteday      17379 non-null  object
 2   season      17379 non-null  int64
 3   yr          17379 non-null  int64
 4   mnth        17379 non-null  int64
 5   hr          17379 non-null  int64
 6   holiday     17379 non-null  int64
 7   weekday     17379 non-null  object
 8   workingday  17379 non-null  int64
 9   weathersit  17379 non-null  int64
 10  temp        17379 non-null  float64
 11  atemp       17379 non-null  float64
 12  hum         17379 non-null  float64
 13  windspeed   17379 non-null  float64
 14  casual      17379 non-null  int64
 15  registered  17379 non-null  int64
 16  cnt         17379 non-null  int64
dtypes: float64(4), int64(11), object(2)
memory usage: 2.3+ MB
```

Figure 5: code for data types of 17 columns in the dataset

11.2 Missing values

Checking missing values is very imporatant step in data exploration because the data is not complete without handling missing values and many leaning algrorithms do now allow any missing values. Fortunately, there is no missing value in this datatset (V, 2021).

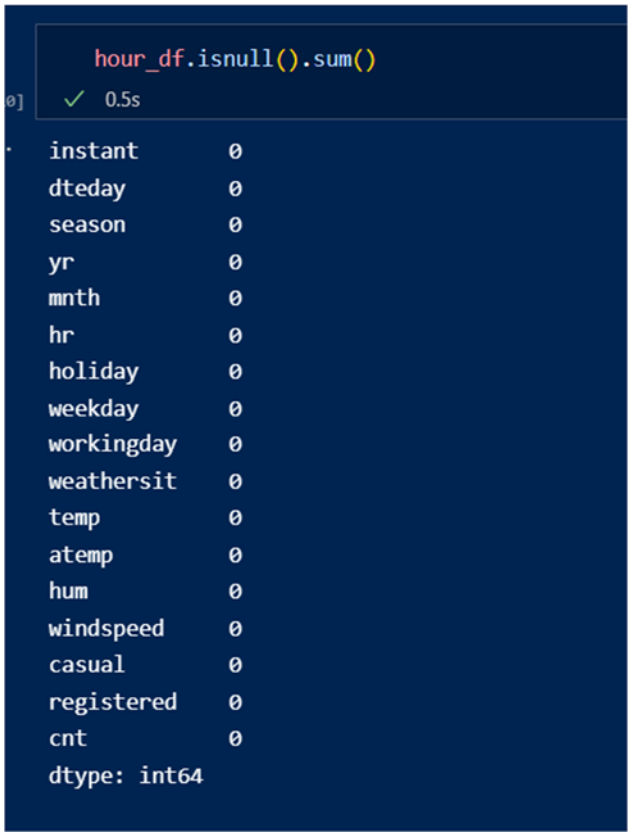


Figure 6: code for detecting missing values in variables

11.3 Convert working day and weekday from integer to string

As noticed, weekday is set with number from 0 to 6. Therefore, in order to to make it easy to see, I will convert the numbers to string: Monday, Tuesday, Wednesday, Thursday, Friday, Saturday and Sunday.

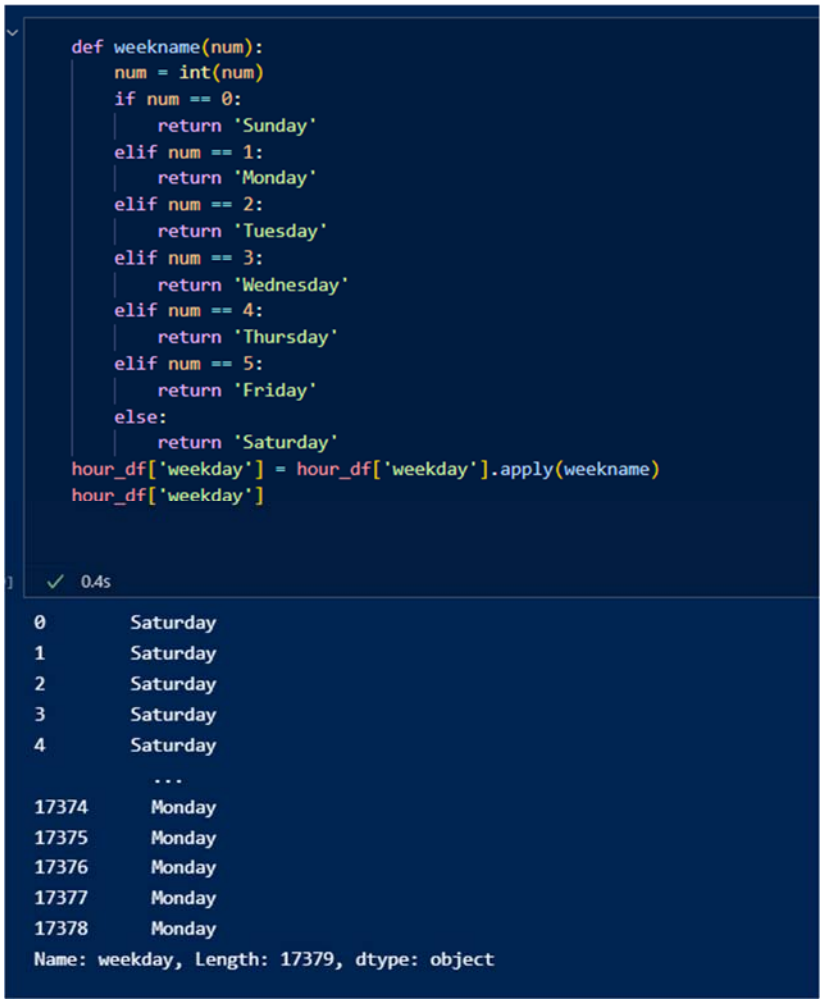
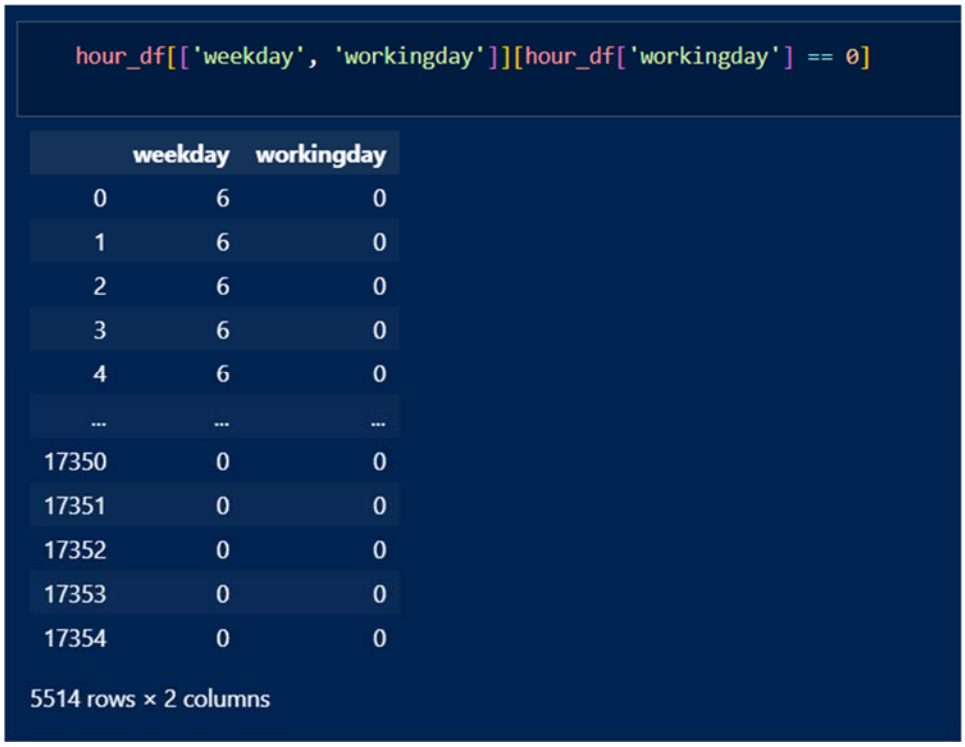


Figure 7: code for defining date names for weekday column

11.4 Logic of weekday and working day

As can be seen from data dictionary, workingday is when day is neither weekend nor holiday is 1, otherwise is 0 so 0 or 6 would be Saturday or Sunday. The Python code below will show how to identify those days.



```
hour_df[['weekday', 'workingday']][hour_df['workingday'] == 0]
```

	weekday	workingday
0	6	0
1	6	0
2	6	0
3	6	0
4	6	0
...	...	...
17350	0	0
17351	0	0
17352	0	0
17353	0	0
17354	0	0

5514 rows × 2 columns

Figure 7. Code for identifying weekend

11.5 Correlation matrix

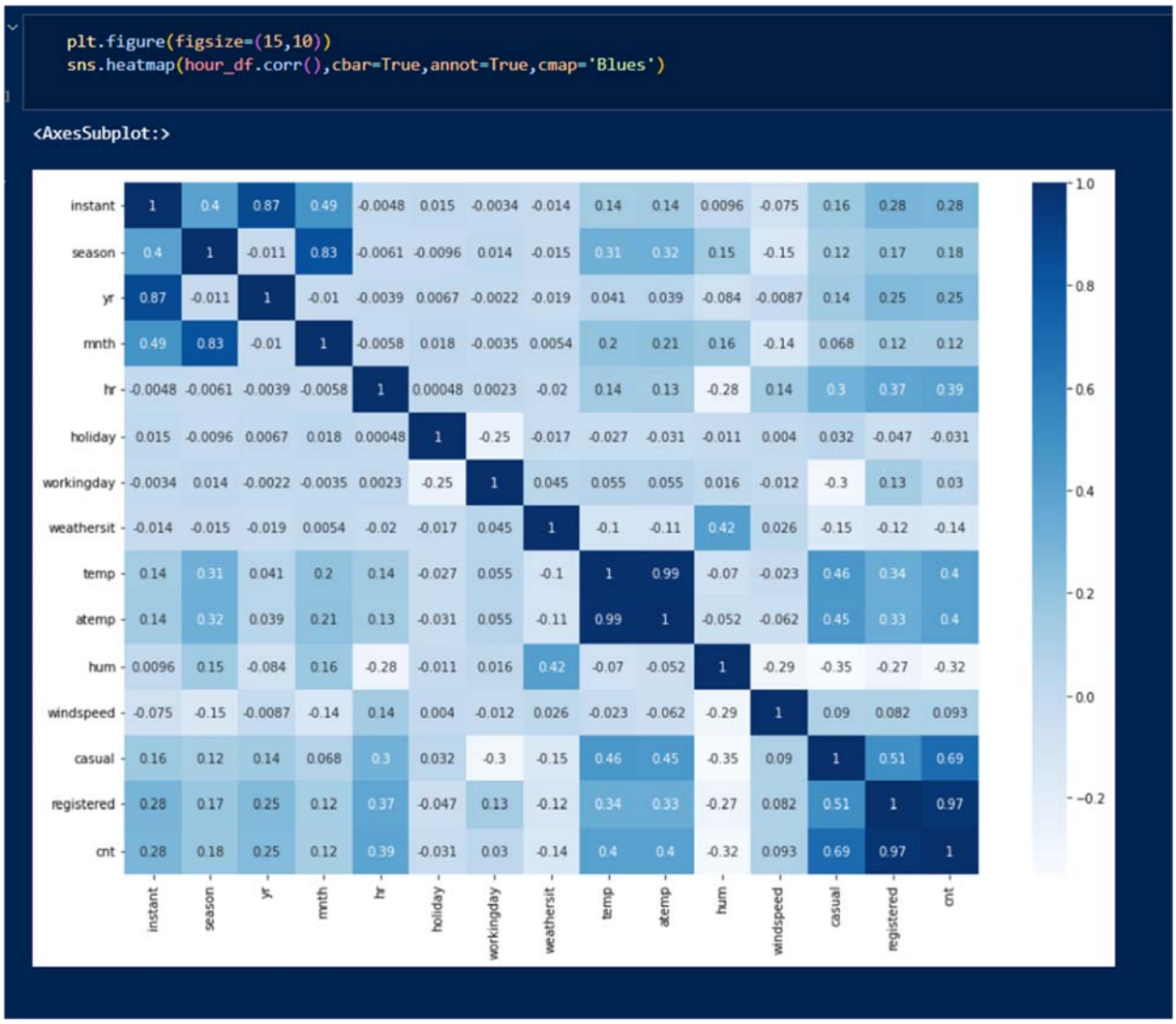


Figure 8. Heat map of a Correlation table

Positive correlation:

1. count- instant, season, year, hour, temp, atemp, casual, registered

2. temp - season, month, hour, casual, registered, count

3. atemp - season, month, hour, casual, registered, count

4. casual - hour, temp, atemp, registered, count

5. registered - hour, temp, atemp, casual, count

Negative correlation:

1. registered - holiday, weather_situation, humidity

2. casual - workingday, weather_situation, humidity

3. holiday - season, weekday, workingday, weather_situation, temp, atemp, humid, registered, count

4. weather_situation - instant, season, year, hour, holiday, temp, atemp, casual, registered, count

5. humidity - year, hour, holiday, weekeday, temp, atemp, windspeed, casual, registered, count

11.6 Compute mean, standard deviation, min, max, median, length of all variables

Compute mean, standard deviation, min, max, median, length and coefficient of ‘casual’ variable

```
    print('mean:' , hour_df.casual.mean())
    print('std.dev:' , hour_df.casual.std())
    print('min:' , hour_df.casual.min())
    print('max:' , hour_df.casual.max())
    print('length:' , len(hour_df.casual))
    cv = lambda x: np.std(x, ddof=1) / np.mean(x) * 100
    print ('coeffcient:',cv(hour_df.casual))
 ✓ 0.4s
mean: 35.67621842453536
std.dev: 49.30503038705308
min: 0
max: 367
length: 17379
coeffcient: 138.20139175161253
```

Figure 9. code for computing mean, standard deviation, min, max, median, length and coefficient of ‘casual’

Compute mean, standard deviation, min, max, median, length and coefficient of ‘registered’ variable

```
print('mean:' , hour_df.registered.mean())
print('std.dev:' , hour_df.registered.std())
print('min:' , hour_df.registered.min())
print('max:' , hour_df.registered.max())
print('length:' , len(hour_df.registered))
cv = lambda x: np.std(x, ddof=1) / np.mean(x) * 100
print ('coeffcient:',cv(hour_df.registered))
```

✓ 0.7s

```
mean: 153.78686920996606
std.dev: 151.35728591258314
min: 0
max: 886
length: 17379
coeffcient: 98.42016206593959
```

Figure 10. code for computing mean, standard deviation, min, max, median, length and coefficient of ‘registered’

Compute mean, standard deviation, min, max, median, and length of all variables.

```
pd.DataFrame({'mean': hour_df.mean(),
              'std.dev': hour_df.std(),
              'min':hour_df.min(),
              'max':hour_df.max(),
              'median': hour_df.median(),
              'length': len(hour_df),
              })
```

✓ 1.1s

	mean	std.dev	min	max	median	length
atemp	0.475775	0.171850	0.0	1.0	0.4848	17379
casual	35.676218	49.305030	0	367	17.0000	17379
count	189.463088	181.387599	1	977	142.0000	17379
holiday	0.028770	0.167165	0	1	0.0000	17379
hour	11.546752	6.914405	0	23	12.0000	17379
humidity	0.627267	0.192809	0.03	1.0	0.6300	17379
month	6.537775	3.438776	1	12	7.0000	17379
registered	153.786869	151.357286	0	886	115.0000	17379
season	2.501640	1.106918	1	4	3.0000	17379
temp	0.496987	0.192556	0.02	1.0	0.5000	17379
weather_situation	1.425283	0.639357	1	4	1.0000	17379
weekday	NaN	NaN	Friday	Wednesday	NaN	17379
windspeed	0.188691	0.118197	0.0	0.4775	0.1940	17379
workingday	0.682721	0.465431	0	1	1.0000	17379

Figure 11. code for computing mean, standard deviation, min, max, median, and length of all variables

12. Outliers analysis

a. Distribution histogram and boxplots

Identification of potential outliers is important because it may indicate bad data and they should be deleted from the analysis. I used distribution histogram and boxplot to identify outliers with selected numeric data including ‘humidity’, ‘temp’, ‘atemp’, and ‘windspeed’.

```
col_to_plot = ['humidity','temp','atemp','windspeed']
sns.set_style('dark')
hour_df[col_to_plot].plot(subplots=True, layout=(5,5), kind='box', figsize=(25,20), patch_artist=True)
plt.subplots_adjust(wspace=0.5);
```

✓ 0.8s

Figure 10. code for indentiyfing outliers using boxplots

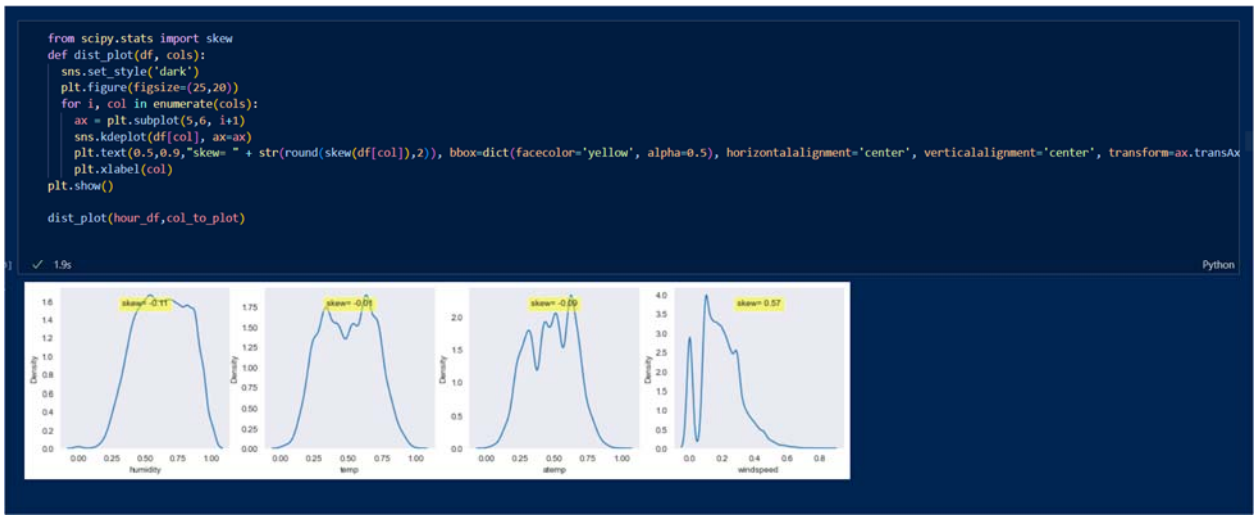


Figure 11. code for identifying skewness using histogram

b. Remove outliers and reduce skewness

```
hour_cap = hour_df
features = ['humidity','temp','atemp','windspeed']
def iqr_capping(df, cols, factor):

    for col in cols:

        q1 = df[col].quantile(0.25)
        q3 = df[col].quantile(0.75)

        iqr = q3 - q1

        upper_whisker = q3 + (factor*iqr)
        lower_whisker = q1 - (factor*iqr)

        df[col] = np.where(df[col]>upper_whisker, upper_whisker,
                           np.where(df[col]<lower_whisker, lower_whisker, df[col]))

iqr_capping(hour_cap, features, 1.5)

sns.set_style('dark')
hour_cap[col_to_plot].plot(subplots=True, layout=(5,5), kind='box', figsize=(25,20), patch_artist=True)
plt.subplots_adjust(wspace=0.5);

dist_plot(hour_cap, col_to_plot)
```

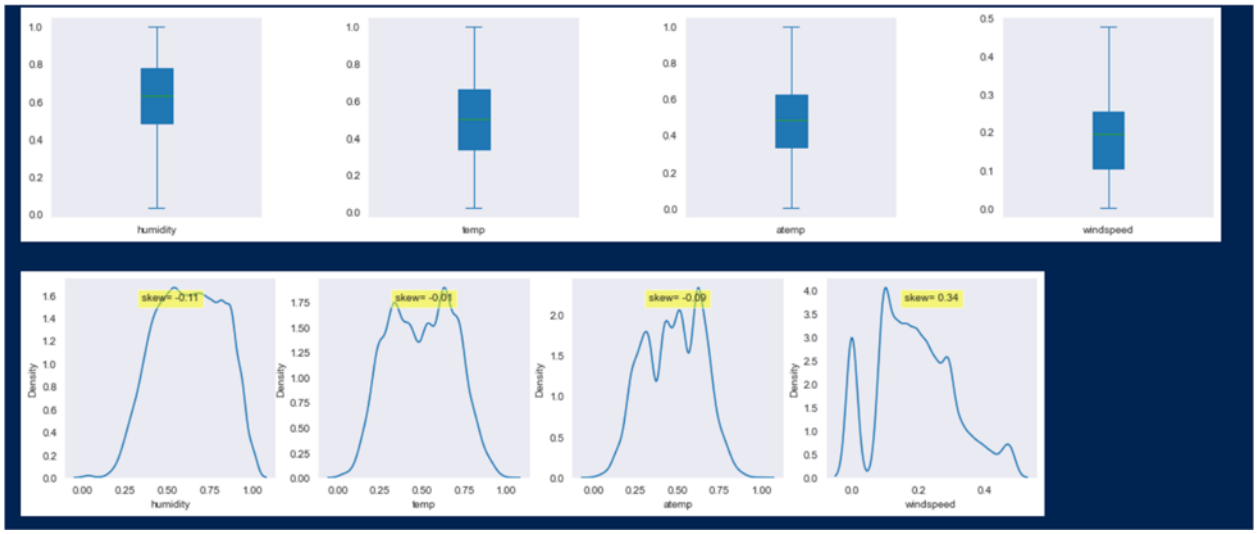


Figure 12. code for removing outliers and reducing skewness

13. Data Preparation and Feature Engineering

a. Data Preparation Needs

There are some columns that are not important in the dataset, so I decided to drop them.

```
hour_df = hour_df.drop(columns = ['instant','dteday','year'])
hour_df.head()
```

✓ 0.9s

	season	month	hour	holiday	weekday	workingday	weather_situation	temp	atemp	humidity	windspeed	casual	registered	count
0	1	1	0	0	Saturday	0	1	0.24	0.2879	0.81	0.0	3	13	16
1	1	1	1	0	Saturday	0	1	0.22	0.2727	0.80	0.0	8	32	40
2	1	1	2	0	Saturday	0	1	0.22	0.2727	0.80	0.0	5	27	32
3	1	1	3	0	Saturday	0	1	0.24	0.2879	0.75	0.0	3	10	13
4	1	1	4	0	Saturday	0	1	0.24	0.2879	0.75	0.0	0	1	1

Figure 12. code for dropping unimportant columns in the dataset

b. Detailed visualization of variables

```
hour_df[['hour', 'casual','registered']].groupby(['hour']).sum().plot(kind = 'bar', figsize = (10,5))
hour_df[['weather_situation', 'casual','registered']].groupby(['weather_situation']).sum().plot(kind='barh', colormap='RdBu')
hour_df[['season', 'casual','registered']].groupby(['season']).sum().plot(kind='barh', colormap='RdPu')
hour_df[['weekday', 'casual','registered']].groupby(['weekday']).sum().plot(kind='barh', colormap='PiYG')
```

✓ 1.2s

Figure 13. code for # Count between hours, weather situation, season, and weekday of casual riders and registered riders

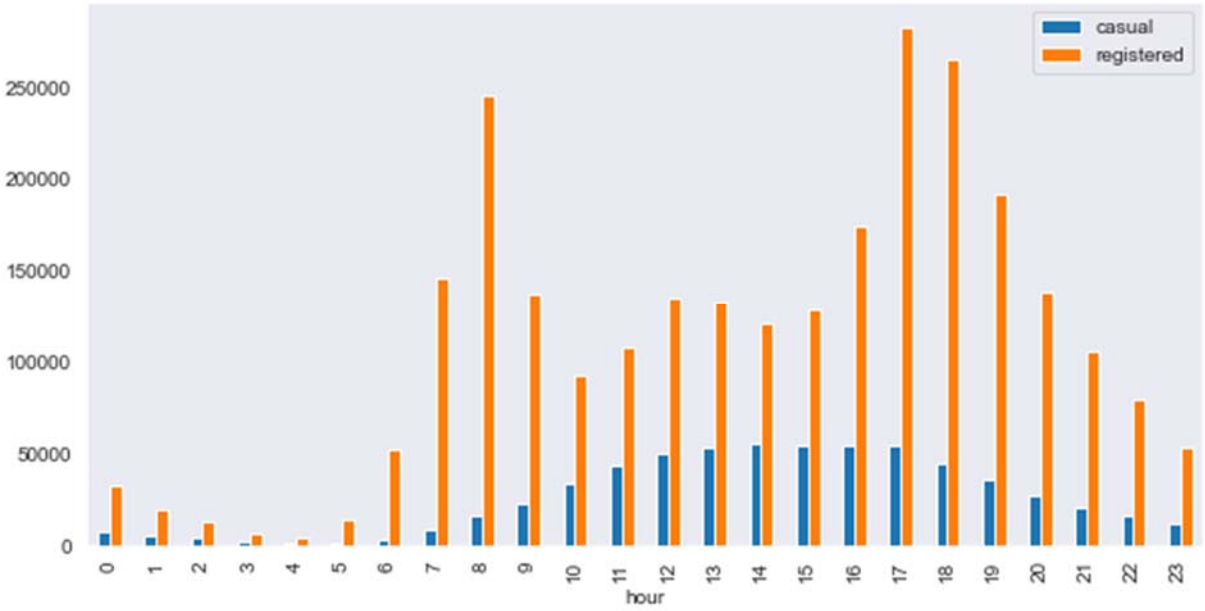


Figure 14. Count on bike rental of casual and registered riders by hours

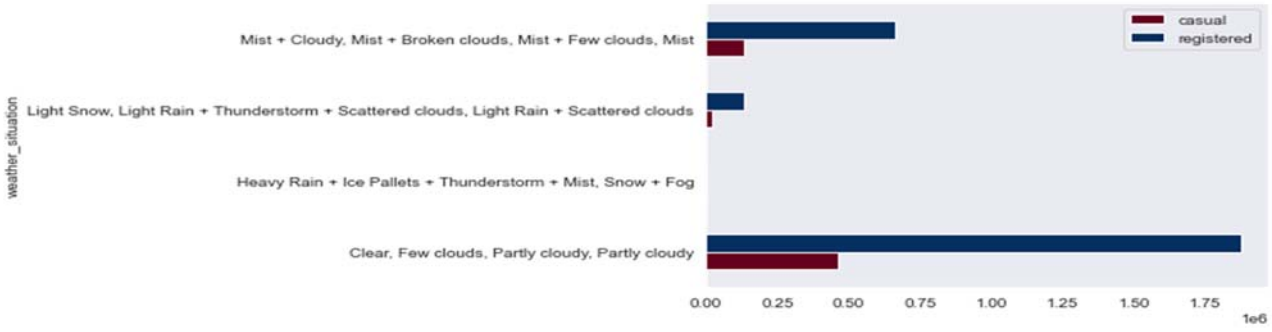


Figure 15. Count on bike rental of casual and registered riders by weather_situation

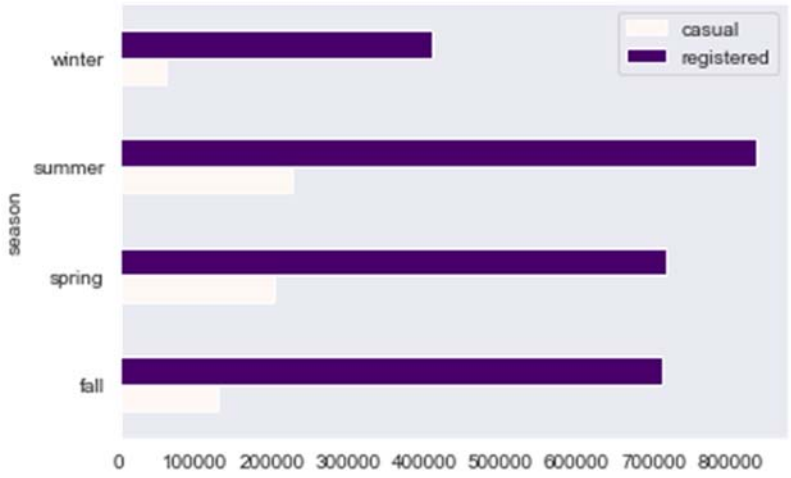


Figure 16. Count on bike rental of casual and registered riders by season

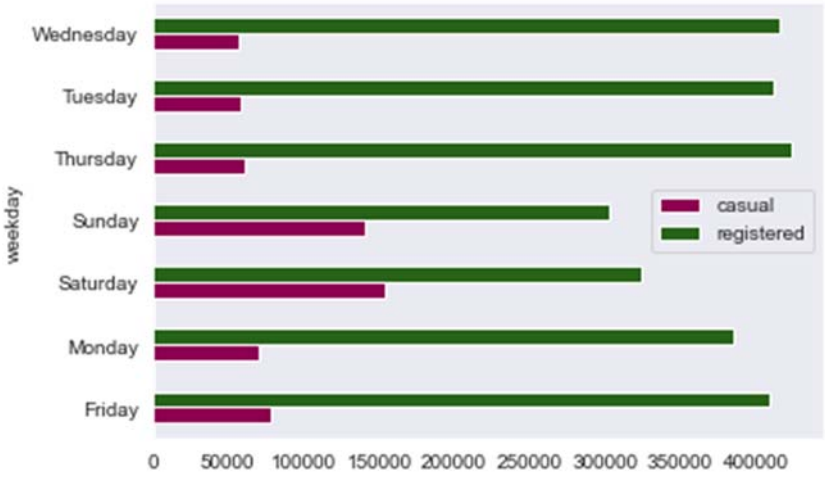


Figure 17. Count on bike rental of casual and registered riders by weekday

14. Feature Engineering

There are few variables that need to be changed to categorial variables including season, month, hour, weekday and weather_situation.

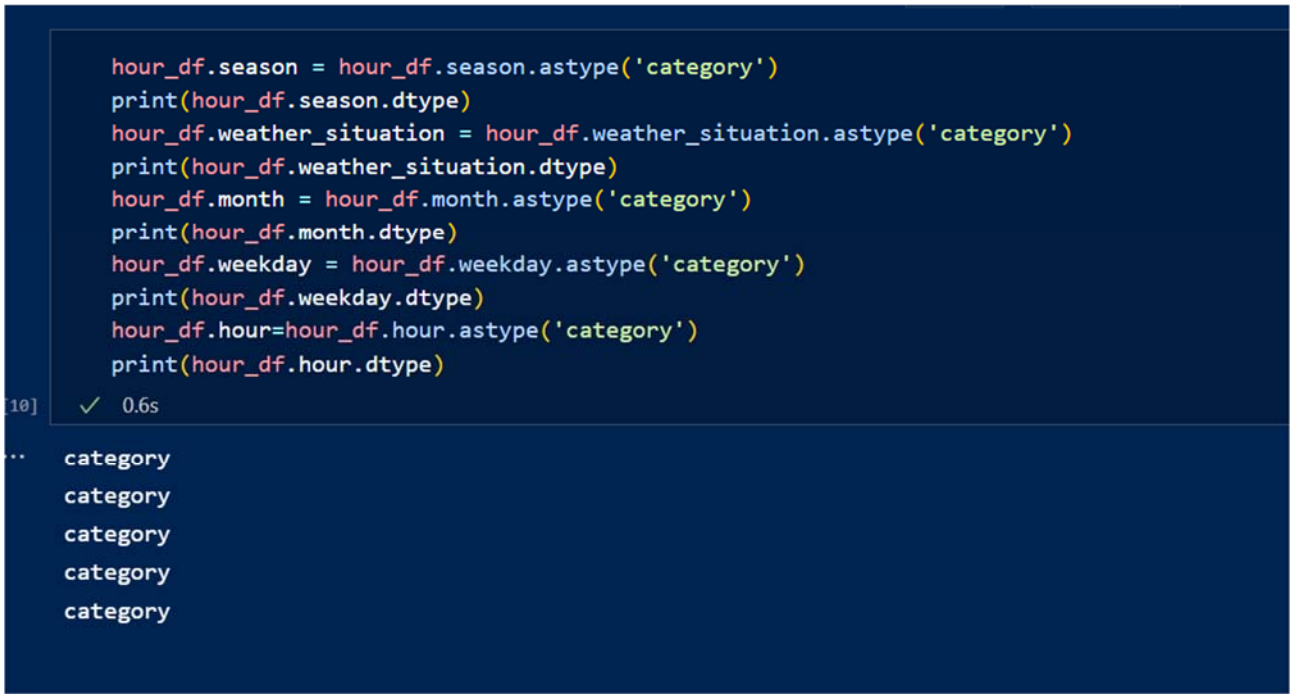


Figure 18. code for changing these variables to categorical variables

15. Summary

a. The dataset has 17379 rows and 17 columns
b. There is correlation between ‘registered’, ‘casual’, ‘count’ with ‘temp’, ‘atemp’, ‘hour’ and ‘season’.
c. There are no missing values
d. Some skewness and outliers have been detected, the action has been taken to reduce skewness and remove outliers.
e. Casual riders tend to rent bikes between 10am to 8pm, when the weather is mist and clear. They are likely to ride during summer, spring and fall. Saturday and Sunday are the dominant days for casual riders.

f. Registered riders tend to rent bikes between 7-9am, 4-7pm, when the weather is mist and clear; they ride all year around for 4 seasons. They literally use bike share everyday.

III. Model Exploration

16. Modeling Approach/Introduction

16.1 Libraries for modeling

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns
import matplotlib.pylab as plt
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.linear_model import LinearRegression
from sklearn.metrics import accuracy_score, mean_squared_error
import matplotlib.pylab as plt
from sklearn.ensemble import RandomForestRegressor
from sklearn.linear_model import LogisticRegression, LogisticRegressionCV
from sklearn.ensemble import GradientBoostingRegressor
from statsmodels.tsa import tsatools
import scikitplot as skplt
from dmba import regressionSummary

✓ 3.7s
```

Figure 19. code for library packages of different algorithms

Before jumping into model techniques, the first thing I need to do is get the correlation heatmap of numeric variables that might have an impact on the analysis.

Correlation heatmap

Figure 20. code for correlation heatmap of numeric variables

As can be seen, temp and atemp are highly correlated with each other, so atemp should be removed from the dataset as well. The target variable is 'count' basically, but besides seeing the general picture of bikeshare riders count, I also want to see count of 'casual' riders and 'registered' riders. Therefore, the models will be applied for three target variables with 'count', 'casual' and 'registered' apparently. The models will be applied in this project are linear regression, polynomial regression and random forest.

In order to predict 'count', besides excluding 'atemp' variable as mentioned above, 'casual' and 'registered' are also needed to be removed. Similar steps will be applied for predicting 'casual' and 'registered'.

Figure 21. code for excluding the high correlation variables for predicting 'count'

Linear Regression is one of the most popular models for making predictions. This model is used to fit the relationship between a numerical outcome variable and a set of predictors (Shmueli et all., 2020). For this model, it is important to do log transform to normalize the data after we check skewness of ‘count’ variable.

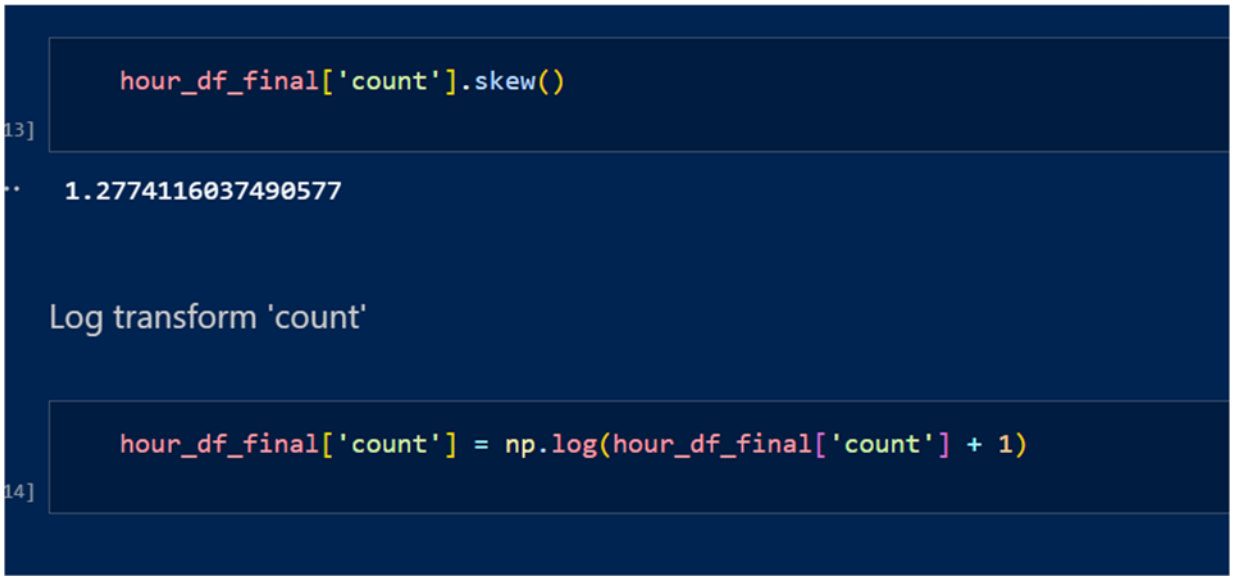


Figure 22. code for checking skewness and log transform ‘count’

16.2 Getting dummies

We will get dummies all categorical variables because for any regression algorithms, strictly require numeric input data. If we try to use under string-based categorical data, they will throw an error. Getting dummies helps us to convert a categorical variable to indicator/dummy variables (columns) (“Pandas Get Dummies Function – get_dummies()”, n.d.)

```
dummified_df = pd.get_dummies(hour_df_final, columns = ['season', 'workingday', 'weather_situation'], drop_first = True)
dummified_df
```

Python

	month	hour	holiday	weekday	temp	humidity	windspeed	count	season_spring	season_summer	season_winter	workingday_1	weather_situation_I Rain + Ice Pall Thunderstorm + Snow
0	1	0	0	6	0.24	0.81	0.0000	2.833213	0	0	1	0	
1	1	1	0	6	0.22	0.80	0.0000	3.713572	0	0	1	0	
2	1	2	0	6	0.22	0.80	0.0000	3.496508	0	0	1	0	
3	1	3	0	6	0.24	0.75	0.0000	2.639057	0	0	1	0	
4	1	4	0	6	0.24	0.75	0.0000	0.693147	0	0	1	0	
...	...	...	...	...	...	...	...	...	...	...	...	...	...
17374	12	19	0	1	0.26	0.60	0.1642	4.787492	0	0	1	1	
17375	12	20	0	1	0.26	0.60	0.1642	4.499810	0	0	1	1	
17376	12	21	0	1	0.26	0.60	0.1642	4.510860	0	0	1	1	
17377	12	22	0	1	0.26	0.56	0.1343	4.127134	0	0	1	1	
17378	12	23	0	1	0.26	0.65	0.1343	3.912023	0	0	1	1	

Figure 23. code for getting dummies categorical variables

17. Train and split data

```
x= dummified_df.drop('count',axis=1)
y= dummified_df['count']
```

Train split based on the final dataset

```
train_x,valid_x, train_y, valid_y = train_test_split(x, y, test_size=.3, random_state=10)
```

```
print(train_x.shape)
valid_x.shape
```

```
(12165, 14)

(5214, 14)
```

Figure 24. code for defining x, y and train split data

18. Linear Regression

18.1 Linear Regression for ‘count’

```
[19]
print(train_y.shape)
valid_y.shape

(12165,)

(5214,)
```

```
[20]
hour_lm_count=LinearRegression()
hour_lm_count.fit(train_x,train_y)

LinearRegression()
```

Figure 25. Code fitting train and split data with Linear Regression of ‘count’

```
[24]
hour_lm_count.intercept_
✓ 0.4s

3.4044240575297926
```

```
[25]
hour_lm_count.coef_
✓ 0.3s

array([-3.47700401e-04,  9.41780660e-02, -2.26694778e-01,  1.28207831e-02,
        2.55719662e+00, -1.43643797e+00,  2.24901011e-01, -3.31612102e-01,
       -5.22008088e-01, -4.84260955e-01, -5.42574915e-02, -1.00504844e-01,
       -2.38857203e-01,  1.66178094e-01])
```

Figure 26. code for checking intercept and coefficient of ‘count’

```
regressionSummary(train_y, hour_lm_count.predict(train_x))
[26] ✓ 0.4s

Regression statistics

                    Mean Error (ME) : -0.0000
         Root Mean Squared Error (RMSE) : 1.0202
              Mean Absolute Error (MAE) : 0.8040
            Mean Percentage Error (MPE) : -11.2648
Mean Absolute Percentage Error (MAPE) : 26.1391
```

Figure 27. code for Regression summary of training data

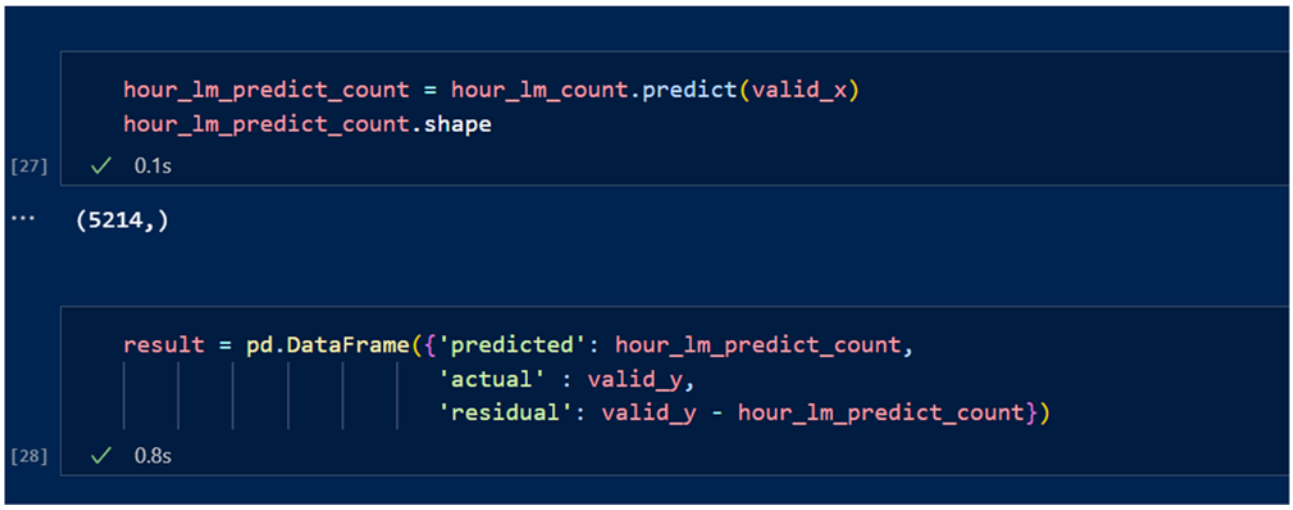
```
pred_y = hour_lm_count.predict(train_x)
pred_y
[27] ✓ 0.3s

array([5.11460218, 4.53619954, 5.54446346, ..., 4.04232231, 4.5333529 ,
       4.90441891])

result = pd.DataFrame({'predicted': pred_y,
                       'actual': train_y,
                       'residuals': train_y - pred_y})
[28] ✓ 0.3s
```

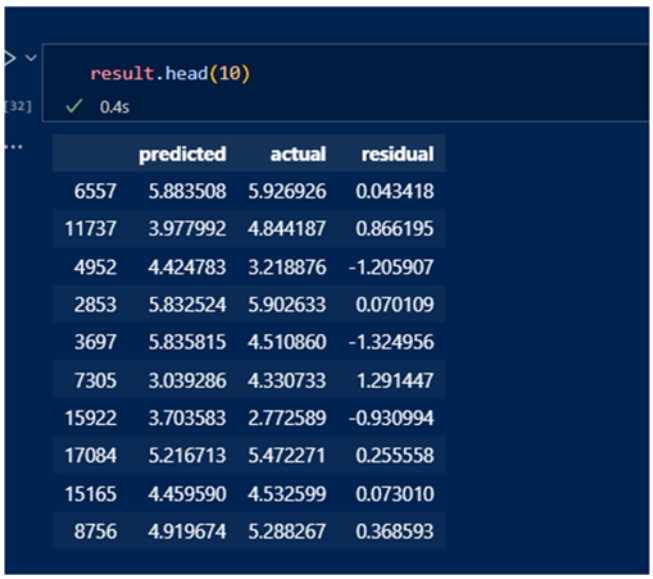
```
result
[29] ✓ 0.4s
```

	predicted	actual	residuals
16745	5.114602	5.451038	0.336436
517	4.536200	4.189655	-0.346545
11296	5.544463	5.662960	0.118497
4726	5.766532	5.736572	-0.029960
8585	4.093299	4.875197	0.781898
...	...	...	...
4829	3.878619	1.609438	-2.269181
10201	2.964532	0.693147	-2.271385
9372	4.042322	4.976734	0.934411
7291	4.533353	5.429346	0.895993
7293	4.904419	5.921578	1.017160

12165 rows × 3 columns

Figure 28. code for define prediction of y and get result of actual and residuals difference of training set





	predicted	actual	residual
6557	5.883508	5.926926	0.043418
11737	3.977992	4.844187	0.866195
4952	4.424783	3.218876	-1.205907
2853	5.832524	5.902633	0.070109
3697	5.835815	4.510860	-1.324956
7305	3.039286	4.330733	1.291447
15922	3.703583	2.772589	-0.930994
17084	5.216713	5.472271	0.255558
15165	4.459590	4.532599	0.073010
8756	4.919674	5.288267	0.368593

Figure 29. code for define prediction of y and get result of actual and residuals difference of validation set

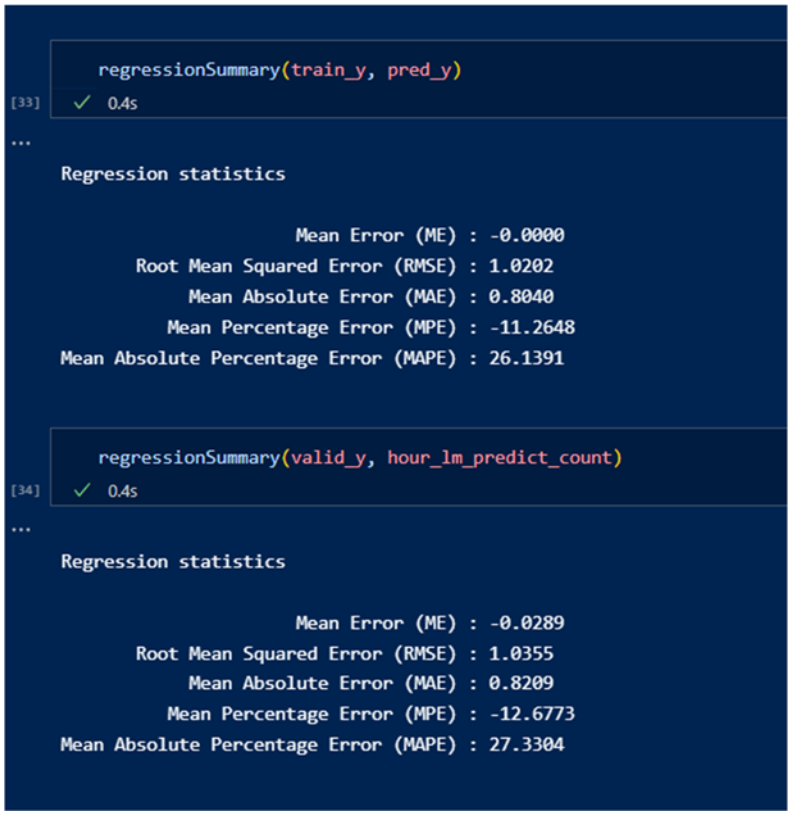


Figure 30. code for Regression Summary for train and validation sets performance

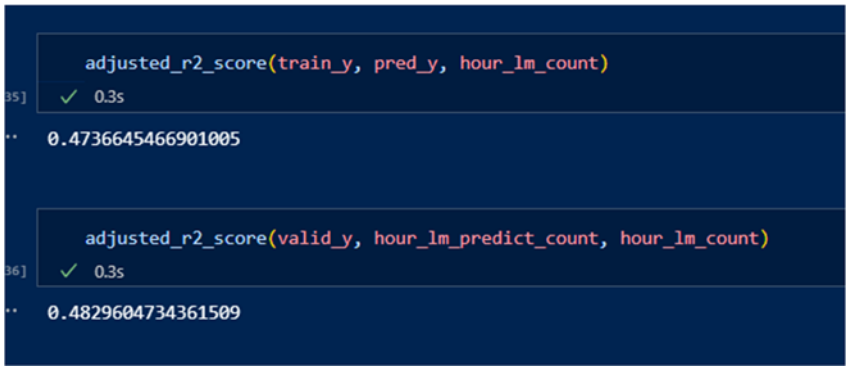


Figure 31. code for adjusted r2 score of training and validation sets

```
feature_names_count = x.columns
model_coefficients = hour_lm_count.coef_

coefficients_df = pd.DataFrame(data = model_coefficients,
                               index = feature_names_count,
                               columns = ['Coefficient value of count'])
print(coefficients_df)
```

[39] ✓ 0.3s

	Coefficient value of count
month	-0.000348
hour	0.094178
holiday	-0.226695
weekday	0.012821
temp	2.557197
humidity	-1.436438
windspeed	0.224901
season_spring	-0.331612
season_summer	-0.522008
season_winter	-0.484261
workingday_1	-0.054257
weather_situation_Heavy Rain + Ice Pallets + Th...	-0.100505
weather_situation_Light Snow, Light Rain + Thun...	-0.238857
weather_situation_Mist + Cloudy, Mist + Broken ...	0.166178

```
coefficients_df.sort_values(by = 'Coefficient value of count', ascending = False, inplace = True)
```

[40] ✓ 0.4s

```
coefficients_df.plot(kind = 'barh', colormap='spring')
```

[41] ✓ 0.3s

```
<AxesSubplot:>
```

Figure 32. Code for setting coefficient value of important features

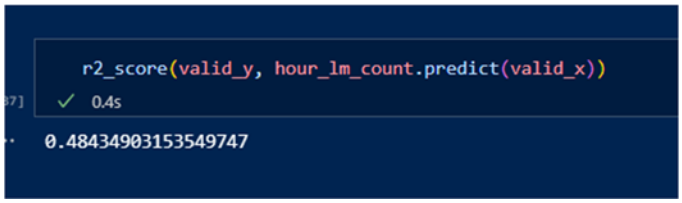


Figure 33 code for R2 score

18.2 Linear Regression of 'casual'

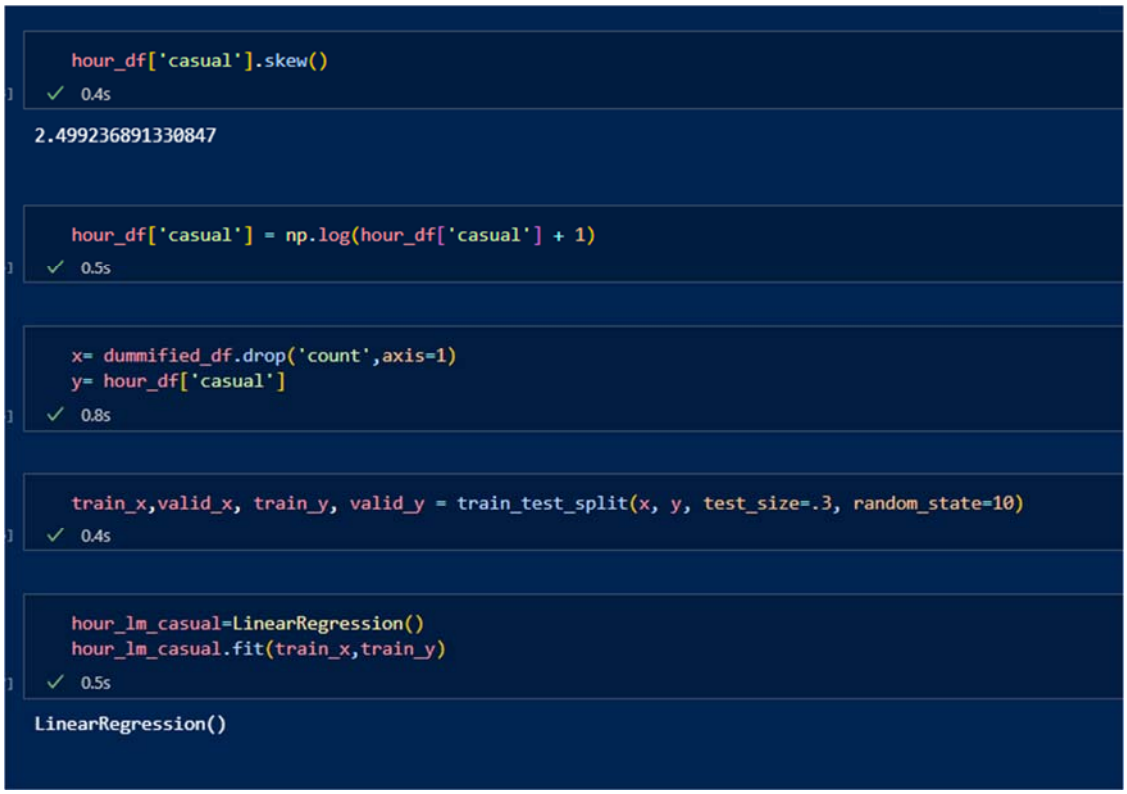


Figure 34. code for fitting Linear Regression model to train and validation sets of 'casual'

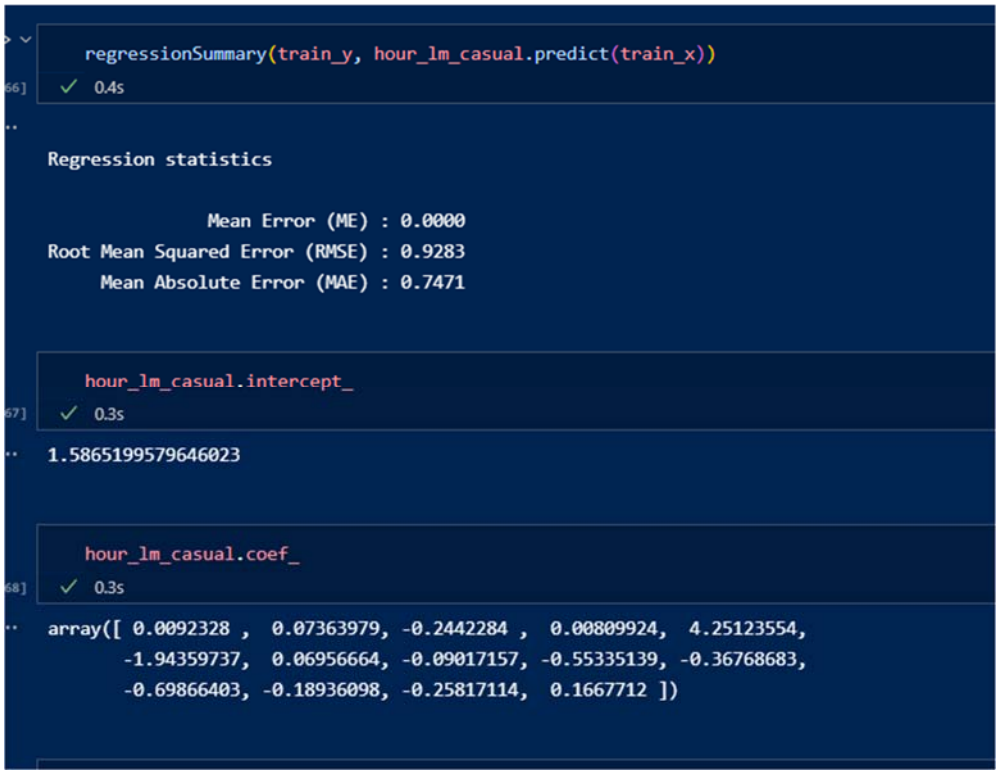
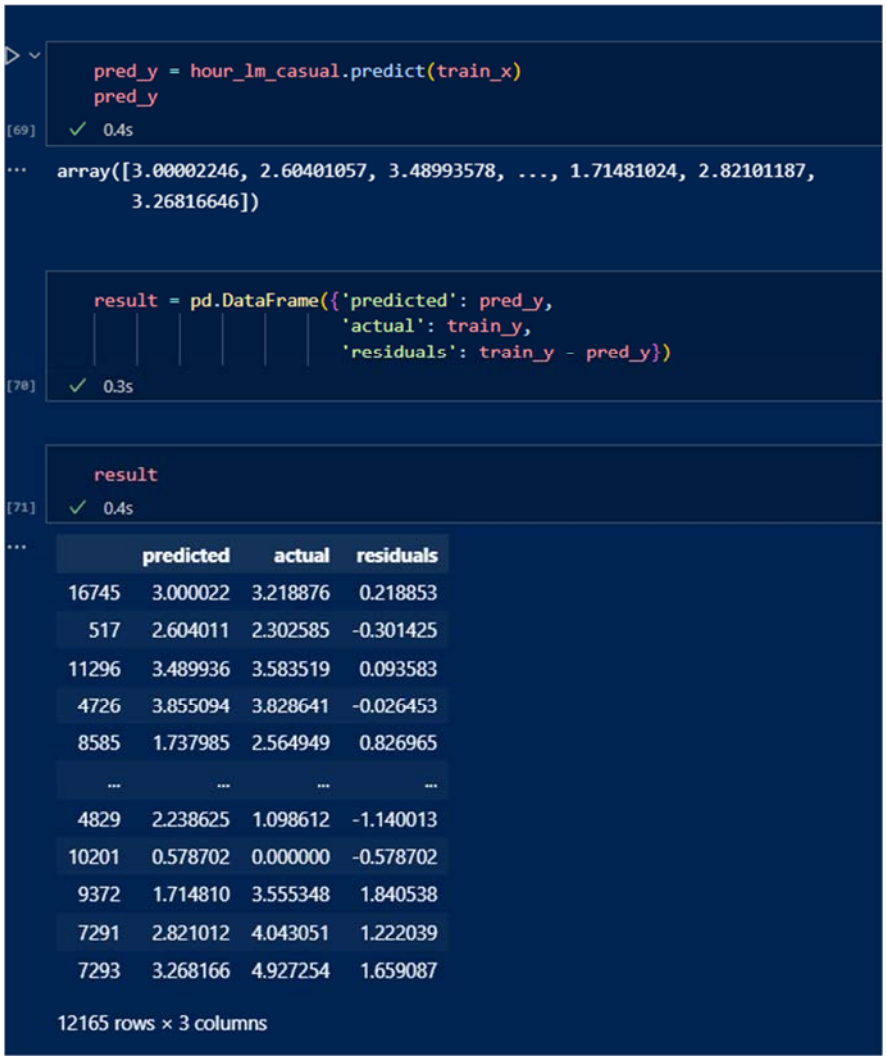


Figure 35. code for training performance, intercept and coefficient



```
pred_y = hour_lm_casual.predict(train_x)
pred_y
```

[69] ✓ 0.4s

```
array([3.00002246, 2.60401057, 3.48993578, ..., 1.71481024, 2.82101187,
       3.26816646])
```

```
result = pd.DataFrame({'predicted': pred_y,
                       'actual': train_y,
                       'residuals': train_y - pred_y})
```

[70] ✓ 0.3s

```
result
```

[71] ✓ 0.4s

	predicted	actual	residuals
16745	3.000022	3.218876	0.218853
517	2.604011	2.302585	-0.301425
11296	3.489936	3.583519	0.093583
4726	3.855094	3.828641	-0.026453
8585	1.737985	2.564949	0.826965
...	...	...	...
4829	2.238625	1.098612	-1.140013
10201	0.578702	0.000000	-0.578702
9372	1.714810	3.555348	1.840538
7291	2.821012	4.043051	1.222039
7293	3.268166	4.927254	1.659087

12165 rows × 3 columns

Figure 36. code for result of difference between residuals and actual in training set

```
hour_lm_predict_casual = hour_lm_casual.predict(valid_x)
hour_lm_predict_casual.shape
```

[72] ✓ 0.4s

```
(5214,)
```

```
result = pd.DataFrame({'predicted': hour_lm_predict_casual,
                       'actual' : valid_y,
                       'residual': valid_y - hour_lm_predict_casual})
```

[73] ✓ 0.4s

```
result.head(10)
```

[74] ✓ 0.4s

	predicted	actual	residual
6557	3.790282	3.713572	-0.076710
11737	2.154352	1.791759	-0.362592
4952	3.309369	1.945910	-1.363459
2853	4.067111	3.761200	-0.305911
3697	3.925663	2.944439	-0.981224
7305	1.420729	2.397895	0.977167
15922	1.511902	1.791759	0.279858
17084	2.960993	3.367296	0.406302
15165	3.255090	2.564949	-0.690141
8756	2.629386	2.995732	0.366346

Figure 37. code for result of difference between residuals and actual in validation set

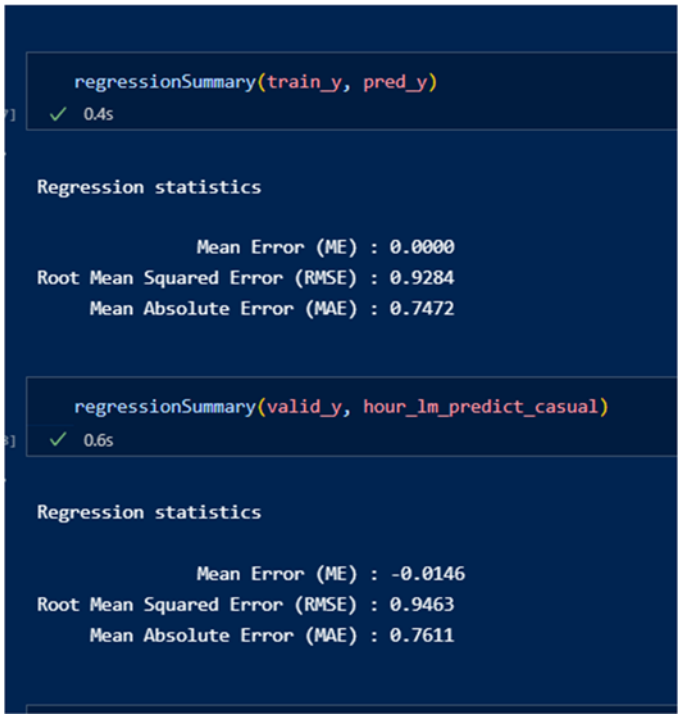


Figure 38. code for Regression Summary of training and validation sets

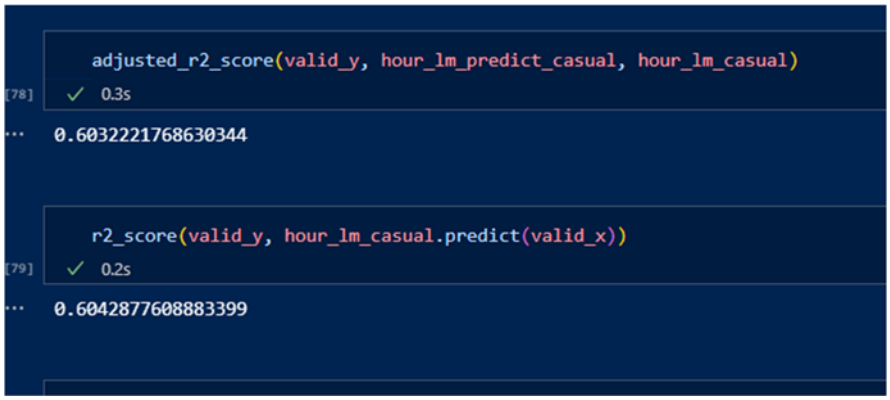


Figure 39. code for adjusted R2 score, r2 score

```
feature_names_casual = x.columns
model_coefficients_1 = hour_lm_casual.coef_

coefficients_df_1 = pd.DataFrame(data = model_coefficients_1,
                                 index = feature_names_casual,
                                 columns = ['Coefficient value of casual'])
print(coefficients_df_1)
```

[81] ✓ 0.4s

	Coefficient value of casual
month	0.009233
hour	0.073640
holiday	-0.244228
weekday	0.008099
temp	4.251236
humidity	-1.943597
windspeed	0.069567
season_spring	-0.090172
season_summer	-0.553351
season_winter	-0.367687
workingday_1	-0.698664
weather_situation_Heavy Rain + Ice Pallets + Th...	-0.189361
weather_situation_Light Snow, Light Rain + Thun...	-0.258171
weather_situation_Mist + Cloudy, Mist + Broken ...	0.166771

```
coefficients_df_1.sort_values(by = 'Coefficient value of casual', ascending = False, inplace = True)
```

[82] ✓ 0.3s

```
coefficients_df_1.plot(kind = 'barh', colormap='winter')
```

[83] ✓ 0.3s

<AxesSubplot:>

Figure 40. code for checking coefficient value of ‘casual’

19.1 Linear Regression of 'registered'

The same steps are applied for registered variable. Python code is displayed below:

```
[108] hour_df['registered'].skew()
✓ 0.6s
1.5579042256481028

[109] hour_df['registered'] = np.log(hour_df['registered'] + 1)
✓ 0.5s

[110] x= dummified_df.drop('count',axis=1)
y= hour_df['registered']
✓ 0.4s

[111] train_x,valid_x, train_y, valid_y = train_test_split(x, y, test_size=.3, random_state=10)
✓ 0.6s

[112] hour_lm_registered=LinearRegression()
hour_lm_registered.fit(train_x,train_y)
✓ 0.4s
LinearRegression()

[113] hour_lm_registered.intercept_
✓ 0.3s
3.2159915810226245

[114] hour_lm_registered.coef_
✓ 0.3s
array([-1.66840668e-03,  9.52056494e-02, -2.24707729e-01,  1.45319980e-02,
        2.16680331e+00, -1.30980959e+00,  2.49282139e-01, -3.73475977e-01,
       -5.01377239e-01, -4.93677450e-01,  9.40064047e-02, -1.38457643e-01,
       -2.29233414e-01,  1.63758716e-01])
```

```
regressionSummary(train_y, hour_lm_registered.predict(train_x))
```

✓ 0.4s

```
Regression statistics

            Mean Error (ME) : 0.0000
 Root Mean Squared Error (RMSE) : 1.0327
      Mean Absolute Error (MAE) : 0.8102
```

```
pred_y = hour_lm_registered.predict(train_x)
pred_y
```

✓ 0.4s

```
array([4.96310883, 4.33193841, 5.35804494, ..., 3.95467767, 4.29930845,
       4.6457678 ])
```

```
result = pd.DataFrame({'predicted': pred_y,
                       'actual': train_y,
                       'residuals': train_y - pred_y})
```

✓ 0.3s

```
result
```

✓ 0.4s

	predicted	actual	residuals
16745	4.963109	5.342334	0.379225
517	4.331938	4.043051	-0.288887
11296	5.358045	5.533389	0.175345
4726	5.551551	5.579730	0.028179
8585	4.016580	4.779123	0.762544
...	...	...	...
4829	3.665776	1.098612	-2.567163
10201	2.918714	0.693147	-2.225567
9372	3.954678	4.709530	0.754853
7291	4.299308	5.147494	0.848186
7293	4.645768	5.463832	0.818064

12165 rows × 3 columns

```
hour_lm_predict_registered = hour_lm_registered.predict(valid_x)
hour_lm_predict_registered.shape
```

[119] ✓ 0.3s

```
(5214,)
```

```
result = pd.DataFrame({'predicted': hour_lm_predict_registered,
                       'actual' : valid_y,
                       'residual': valid_y - hour_lm_predict_registered})
```

[120] ✓ 0.6s

```
result.head(10)
```

[121] ✓ 0.4s

	predicted	actual	residual
6557	5.702548	5.814131	0.111582
11737	3.793596	4.804021	1.010425
4952	4.079004	2.944439	-1.134565
2853	5.582279	5.780744	0.198465
3697	5.610032	4.290459	-1.319573
7305	2.821987	4.189655	1.367668
15922	3.606598	2.397895	-1.208703
17084	5.087507	5.347108	0.259601
15165	4.134733	4.394449	0.259717
8756	4.801481	5.187386	0.385905

```
[122]  regressionSummary(train_y, pred_y)
       ✓ 0.5s

...    Regression statistics

                  Mean Error (ME) : 0.0000
       Root Mean Squared Error (RMSE) : 1.0327
            Mean Absolute Error (MAE) : 0.8102

[123]  regressionSummary(valid_y, hour_lm_predict_registered)
       ✓ 0.4s

...    Regression statistics

                  Mean Error (ME) : -0.0306
       Root Mean Squared Error (RMSE) : 1.0452
            Mean Absolute Error (MAE) : 0.8255

[124]  adjusted_r2_score(train_y, pred_y, hour_lm_registered)
       ✓ 0.4s

...    0.44546729964245524

[125]  adjusted_r2_score(valid_y, hour_lm_predict_registered, hour_lm_registered)
       ✓ 0.4s

...    0.4583613060643249
```

```
feature_names_registered = x.columns
model_coefficients_2 = hour_lm_registered.coef_

coefficients_df_2 = pd.DataFrame(data = model_coefficients_2,
                                 index = feature_names_registered,
                                 columns = ['Coefficient value of registered'])
print(coefficients_df_2)
```

128] ✓ 0.1s

..

	Coefficient value of registered
month	-0.001668
hour	0.095206
holiday	-0.224708
weekday	0.014532
temp	2.166803
humidity	-1.309810
windspeed	0.249282
season_spring	-0.373476
season_summer	-0.501377
season_winter	-0.493677
workingday_1	0.094006
weather_situation_Heavy Rain + Ice Pallets + Th...	-0.138458
weather_situation_Light Snow, Light Rain + Thun...	-0.229233
weather_situation_Mist + Cloudy, Mist + Broken ...	0.163759

```
[129]  coefficients_df_2.sort_values(by = 'Coefficient value of registered', ascending = False, inplace = True)
       ✓ 0.4s

[130]  coefficients_df_2.plot(kind = 'barh')
       ✓ 0.3s

...    <AxesSubplot:>
```


```
[126]  r2_score(valid_y, hour_lm_registered.predict(valid_x))
       ✓ 0.6s

...    0.4598159275327882
```

19. Random Forest

This is the technique performing regression and classification. Random Forest can produce good predictions that can be understood easily and handle dataset efficiently (Mbaabu, 2020). Random Forest works by training a large number of decision trees and then calculating the mean prediction of the individual trees. Notion of random implies to randomly created decision trees. Random decision trees are created on different subsets of the features and datapoints (Jelic, 2021).

19.1 Random Forest of 'count'

```
rf_count=RandomForestRegressor(random_state=1, n_estimators=500)
rf_count.fit(train_x, train_y)
2]  ✓ 21.3s

RandomForestRegressor(n_estimators=500, random_state=1)

rf_count.feature_importances_
3]  ✓ 0.2s

array([1.59426386e-02, 7.46909320e-01, 2.52037483e-03, 1.45786764e-02,
       9.10046512e-02, 2.69529602e-02, 1.68093801e-02, 1.42034255e-03,
       7.53224160e-04, 1.47094141e-02, 4.99483567e-02, 1.76606354e-06,
       1.58837340e-02, 2.56516134e-03])
```

Figure 42. code for fitting Random Forest model to the train and validation set of 'count'

```
importance = rf_count.feature_importances_
std = np.std([tree.feature_importances_ for tree in rf_count.estimators_], axis =0)
```

44] ✓ 0.2s

```
df = pd.DataFrame({'feature' : train_x.columns,
                   'importance': importance,
                   'std'       :std})
print(df.sort_values('importance', ascending = False))
```

45] ✓ 0.4s

```
                                          feature  importance       std
1                                            hour    0.746909  0.008374
4                                            temp    0.091005  0.004288
10                                 workingday_1      0.049948  0.006337
5                                        humidity    0.026953  0.003243
6                                       windspeed    0.016809  0.001015
0                                           month    0.015943  0.001964
12  weather_situation_Light Snow, Light Rain + Thu...    0.015884  0.002586
9                                  season_winter     0.014709  0.003679
3                                         weekday    0.014579  0.001833
13  weather_situation_Mist + Cloudy, Mist + Broken...    0.002565  0.000405
2                                          holiday    0.002520  0.000540
7                                  season_spring     0.001420  0.000512
8                                  season_summer     0.000753  0.000209
11  weather_situation_Heavy Rain + Ice Pallets + T...    0.000002  0.000010
```

Figure 43. code for defining important features and standard deviation

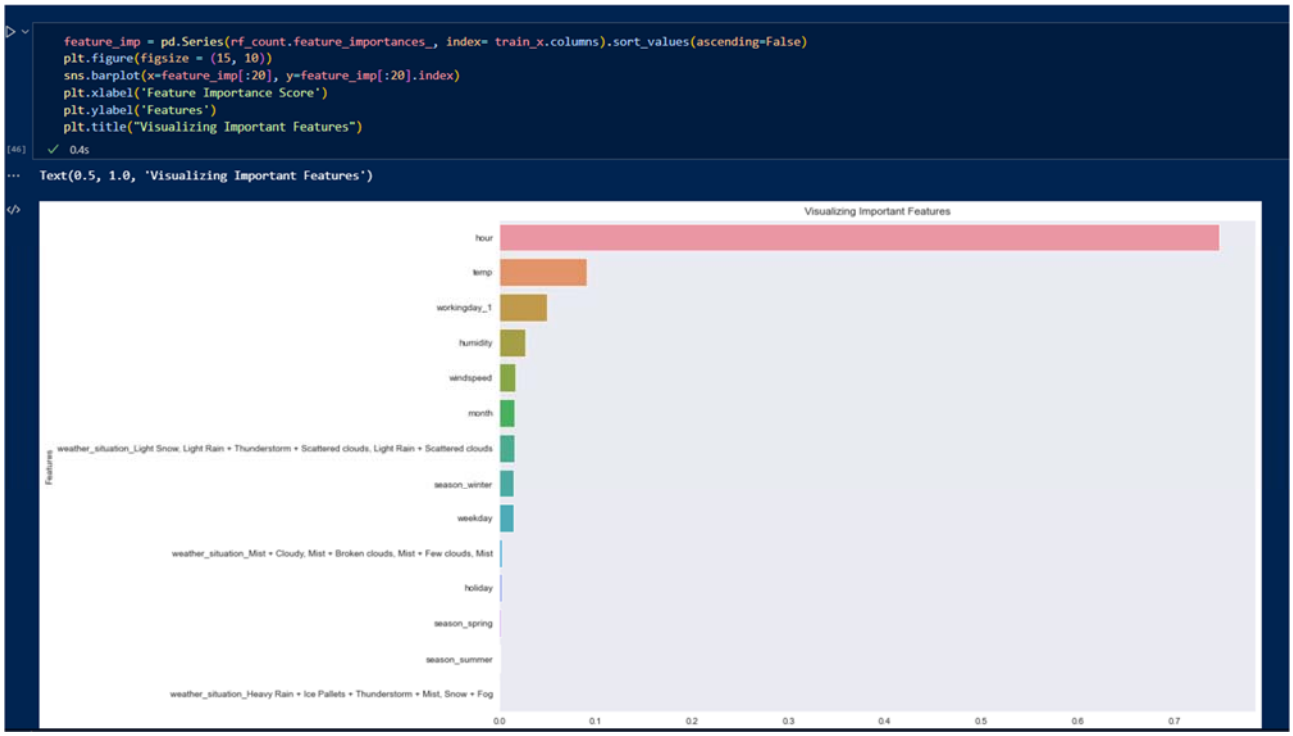


Figure 44. code for plotting important features

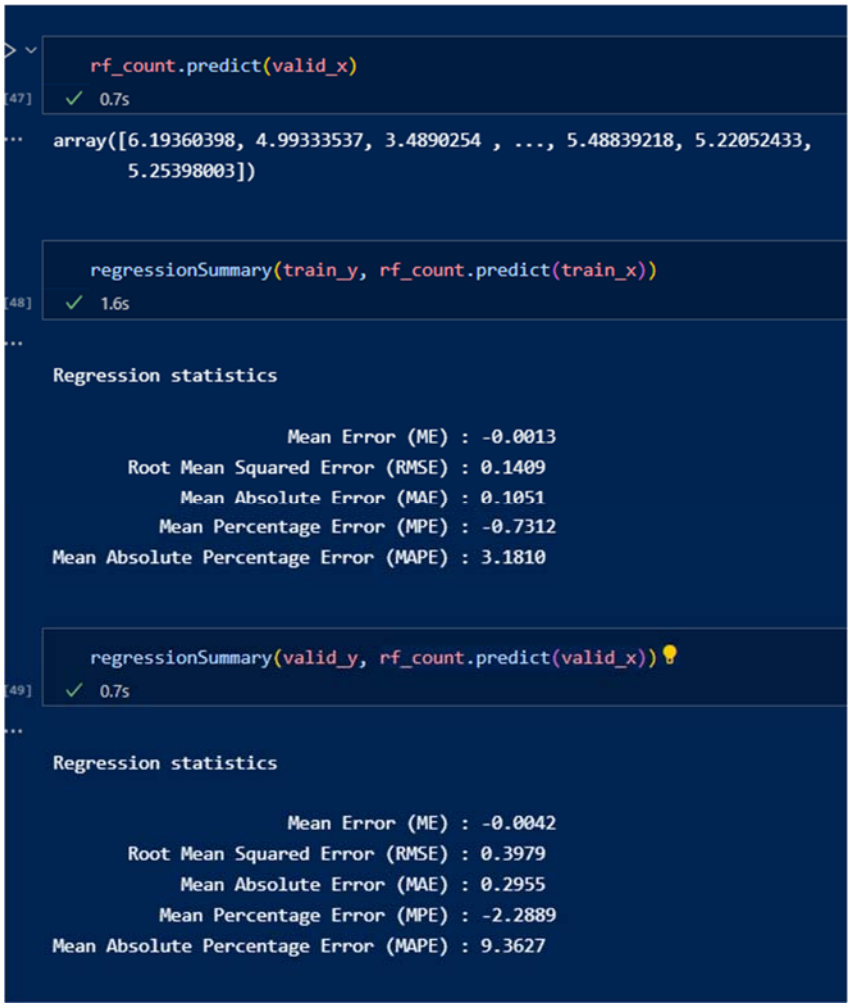


Figure 45. code for Regression Summary of train and validation sets

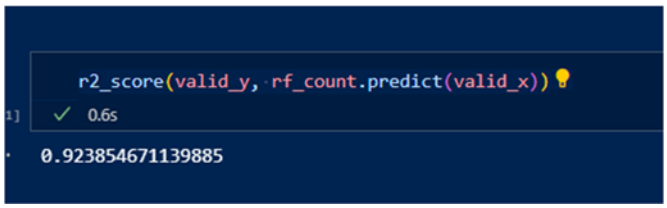


Figure 46. code for r2 score of validation set

19.2 Random Forest of ‘casual’

Python code:

```
excluded_columns_casual = ['atemp','count','registered']
hour_df_final_1 = hour_df.drop(excluded_columns_casual, axis=1)
```

✓ 0.4s Python

```
dummified_df_1 = pd.get_dummies(hour_df_final_1, columns = ['season', 'workingday', 'weather_situation'], drop_first = True)
dummified_df_1
```

✓ 0.9s Python

	month	hour	holiday	weekday	temp	humidity	windspeed	casual	season_spring	season_summer	season_winter	workingday_1	weather_situation_Heavy Rain + Ice Pallets + Thunderstorm + Mist, Snow + Fog	weather_situation_Light Snow, Light Rain + Thunderstorm + Scattered clouds, Light Rain + Scattered clouds	weather_situation_Mist + Cloudy, Mist + Broken clouds, Mist + Few clouds, Mist
0	1	0	0	6	0.24	0.81	0.0000	1.386294	0	0	1	0	0	0	0
1	1	1	0	6	0.22	0.80	0.0000	2.197225	0	0	1	0	0	0	0
2	1	2	0	6	0.22	0.80	0.0000	1.791759	0	0	1	0	0	0	0
3	1	3	0	6	0.24	0.75	0.0000	1.386294	0	0	1	0	0	0	0
4	1	4	0	6	0.24	0.75	0.0000	0.000000	0	0	1	0	0	0	0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
17374	12	19	0	1	0.26	0.60	0.1642	2.484907	0	0	1	1	0	0	1
17375	12	20	0	1	0.26	0.60	0.1642	2.197225	0	0	1	1	0	0	1
17376	12	21	0	1	0.26	0.60	0.1642	2.079442	0	0	1	1	0	0	0
17377	12	22	0	1	0.26	0.56	0.1343	2.639057	0	0	1	1	0	0	0
17378	12	23	0	1	0.26	0.65	0.1343	2.564949	0	0	1	1	0	0	0

17379 rows × 15 columns

```
x= dummified_df_1.drop('casual',axis=1)
y= hour_df_final_1['casual']
```

[86] ✓ 0.3s

```
rf_casual=RandomForestRegressor(random_state=1, n_estimators=500)
rf_casual.fit(train_x, train_y)
```

[87] ✓ 19.9s

```
RandomForestRegressor(n_estimators=500, random_state=1)
```

```
rf_casual.feature_importances_
```

[88] ✓ 0.2s

```
array([2.27677627e-02, 5.52798602e-01, 1.35871736e-03, 1.93202272e-02,
       2.36139294e-01, 5.88211859e-02, 2.66186838e-02, 2.45423393e-03,
       1.62585931e-03, 7.45220346e-03, 5.16453951e-02, 6.12900565e-06,
       1.48942741e-02, 4.09743279e-03])
```

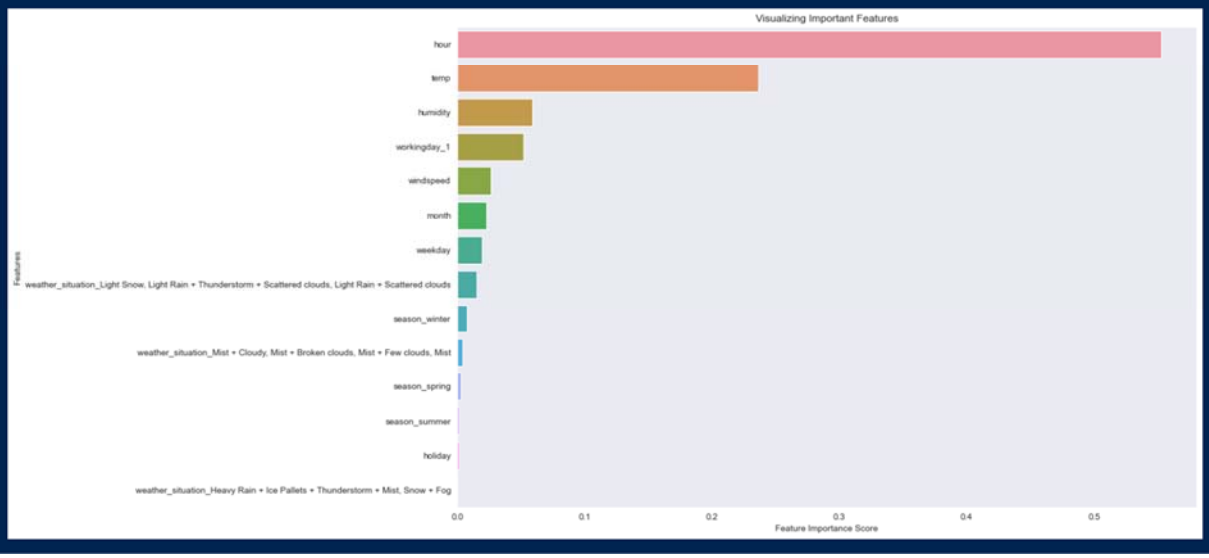
```
importance = rf_casual.feature_importances_
std = np.std([tree.feature_importances_ for tree in rf_casual.estimators_], axis =0)
```
[89] ✓ 0.2s

```
df = pd.DataFrame({'feature' : train_x.columns,
                   'importance': importance,
                   'std'       :std})
print(df.sort_values('importance', ascending = False))
```
[90] ✓ 0.3s

	feature	importance	std
1	hour	0.552799	0.007382
4	temp	0.236139	0.005729
5	humidity	0.058821	0.006179
10	workingday_1	0.051645	0.002828
6	windspeed	0.026619	0.001539
0	month	0.022768	0.001794
3	weekday	0.019320	0.001309
12	weather_situation_Light Snow, Light Rain + Thu...	0.014894	0.002461
9	season_winter	0.007452	0.001467
13	weather_situation_Mist + Cloudy, Mist + Broken...	0.004097	0.000502
7	season_spring	0.002454	0.000512
8	season_summer	0.001626	0.000366
2	holiday	0.001359	0.000388
11	weather_situation_Heavy Rain + Ice Pallets + T...	0.000006	0.000015

```
feature_imp = pd.Series(rf_casual.feature_importances_, index= train_x.columns).sort_values(ascending=False)
plt.figure(figsize = (15, 10))
sns.barplot(x=feature_imp[:20], y=feature_imp[:20].index)
plt.xlabel('Feature Importance Score')
plt.ylabel('Features')
plt.title("Visualizing Important Features")
```
[91] ✓ 0.3s

```
Text(0.5, 1.0, 'Visualizing Important Features')
```



19.3 Random Forest of ‘registered’

Python code

```
excluded_columns_3 = ['atemp','count','casual']
hour_df_final_3 = hour_df.drop(excluded_columns_3, axis=1)
```
✓ 0.2s Python

```
dummified_df_3 = pd.get_dummies(hour_df_final_3, columns = ['season', 'workingday', 'weather_situation'], drop_first = True)
dummified_df_3
```
✓ 0.1s Python

	month	hour	holiday	weekday	temp	humidity	windspeed	registered	season_spring	season_summer	season_winter	workingday_1	weather_situation_Heavy Rain + Ice Pallets + Thunderstorm + Mist, Snow + Fog	weather_situation_Light Snow, Light Rain + Thunderstorm + Scattered clouds, Light Rain + Scattered clouds	weather_situation_Mist + Cloudy, Mist + Broken clouds, Mist + Few clouds, Mist
0	1	0	0	6	0.24	0.81	0.0000	2.639057	0	0	1	0	0	0	0
1	1	1	0	6	0.22	0.80	0.0000	3.496508	0	0	1	0	0	0	0
2	1	2	0	6	0.22	0.80	0.0000	3.332205	0	0	1	0	0	0	0
3	1	3	0	6	0.24	0.75	0.0000	2.397895	0	0	1	0	0	0	0
4	1	4	0	6	0.24	0.75	0.0000	0.693147	0	0	1	0	0	0	0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
17374	12	19	0	1	0.26	0.60	0.1642	4.691348	0	0	1	1	0	0	1
17375	12	20	0	1	0.26	0.60	0.1642	4.406719	0	0	1	1	0	0	1
17376	12	21	0	1	0.26	0.60	0.1642	4.430817	0	0	1	1	0	0	0
17377	12	22	0	1	0.26	0.56	0.1343	3.891820	0	0	1	1	0	0	0
17378	12	23	0	1	0.26	0.65	0.1343	3.637586	0	0	1	1	0	0	0

17379 rows × 15 columns

```
x= dummified_df_3.drop('registered',axis=1)
y= hour_df_final_3['registered']
```
[33] ✓ 0.3s

```
rf_registered=RandomForestRegressor(random_state=1, n_estimators=500)
rf_registered.fit(train_x, train_y)
```
[34] ✓ 20.8s

```
RandomForestRegressor(n_estimators=500, random_state=1)
```

```
rf_registered.feature_importances_
```
[35] ✓ 0.2s

```
array([1.95818314e-02, 7.32272347e-01, 3.01403930e-03, 1.40335154e-02,
       7.44271544e-02, 2.82663398e-02, 1.74359064e-02, 1.51629849e-03,
       7.75515556e-04, 1.19055618e-02, 8.03733152e-02, 3.32522512e-06,
       1.36318268e-02, 2.76302353e-03])
```

```
importance = rf_registered.feature_importances_
std = np.std([tree.feature_importances_ for tree in rf_registered.estimators_], axis =0)
```

[136] ✓ 0.2s

```
df = pd.DataFrame({'feature' : train_x.columns,
                   'importance': importance,
                   'std'       :std})
print(df.sort_values('importance', ascending = False))
```

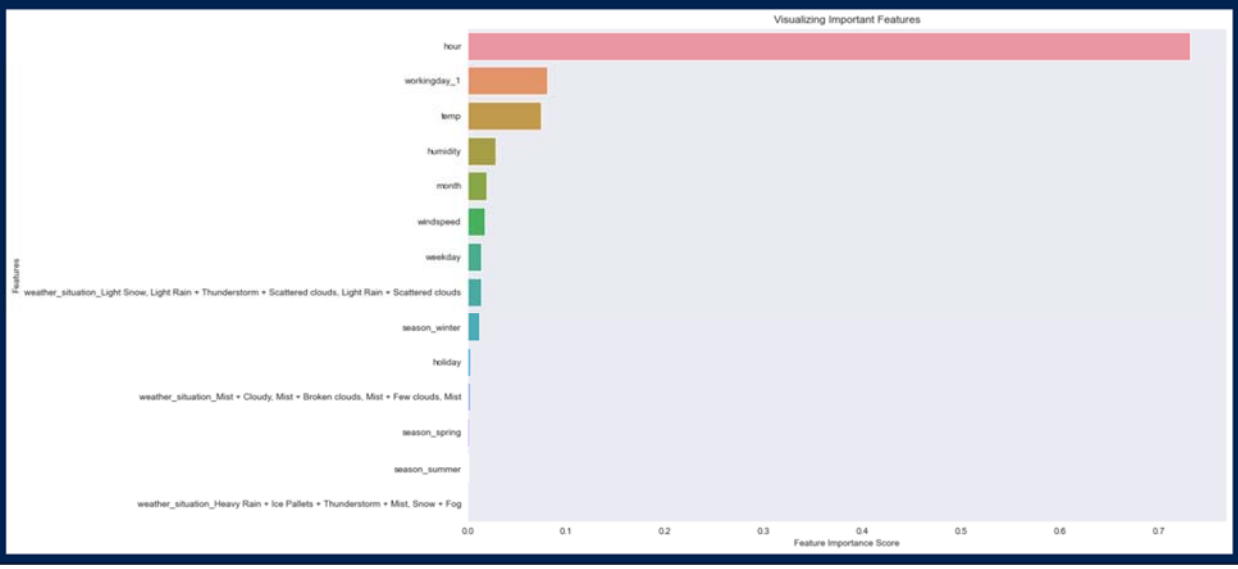
[137] ✓ 0.4s

```
...                                              feature  importance       std
1                                                   hour    0.732272  0.007889
10                                           workingday_1    0.080373  0.007388
4                                                   temp    0.074427  0.005103
5                                               humidity    0.028266  0.004363
0                                                  month    0.019582  0.002284
6                                              windspeed    0.017436  0.001057
3                                                weekday    0.014034  0.001168
12  weather_situation_Light Snow, Light Rain + Thu...    0.013632  0.002207
9                                         season_winter    0.011906  0.003450
2                                                holiday    0.003014  0.000578
13  weather_situation_Mist + Cloudy, Mist + Broken...    0.002763  0.000370
7                                         season_spring    0.001516  0.000448
8                                         season_summer    0.000776  0.000202
11  weather_situation_Heavy Rain + Ice Pallets + T...    0.000003  0.000016
```

```
feature_imp = pd.Series(rf_registered.feature_importances_, index= train_x.columns).sort_values(ascending=False)
plt.figure(figsize = (15, 10))
sns.barplot(x=feature_imp[:20], y=feature_imp[:20].index)
plt.xlabel('Feature Importance Score')
plt.ylabel('Features')
plt.title("Visualizing Important Features")
```

✓ 0.4s

```
Text(0.5, 1.0, 'Visualizing Important Features')
```



20. Polynomial Regression

Polynomial regression is often considered as a special multiple linear regression. The behavior of a dependent variable is explained by a linear, or curvilinear, additive relationship between the dependent variable and a set of k independent variables (voxco, n.d.)

20.1 Polynomial Regression of ‘count’

```
x= dummified_df.drop('count',axis=1)
y= dummified_df['count']
poly = PolynomialFeatures(degree=2, include_bias=True)
poly_features = poly.fit_transform(x)
```

```
train_x,valid_x, train_y, valid_y = train_test_split(poly_features, y, test_size=.3, random_state=10)
```

```
hour_lm_count.fit(train_x, train_y)
```

```
LinearRegression()
```

```
y_predicted_count = hour_lm_count.predict(valid_x)
```

Figure 47. code for fitting polynomial to train and validation data of ‘count’

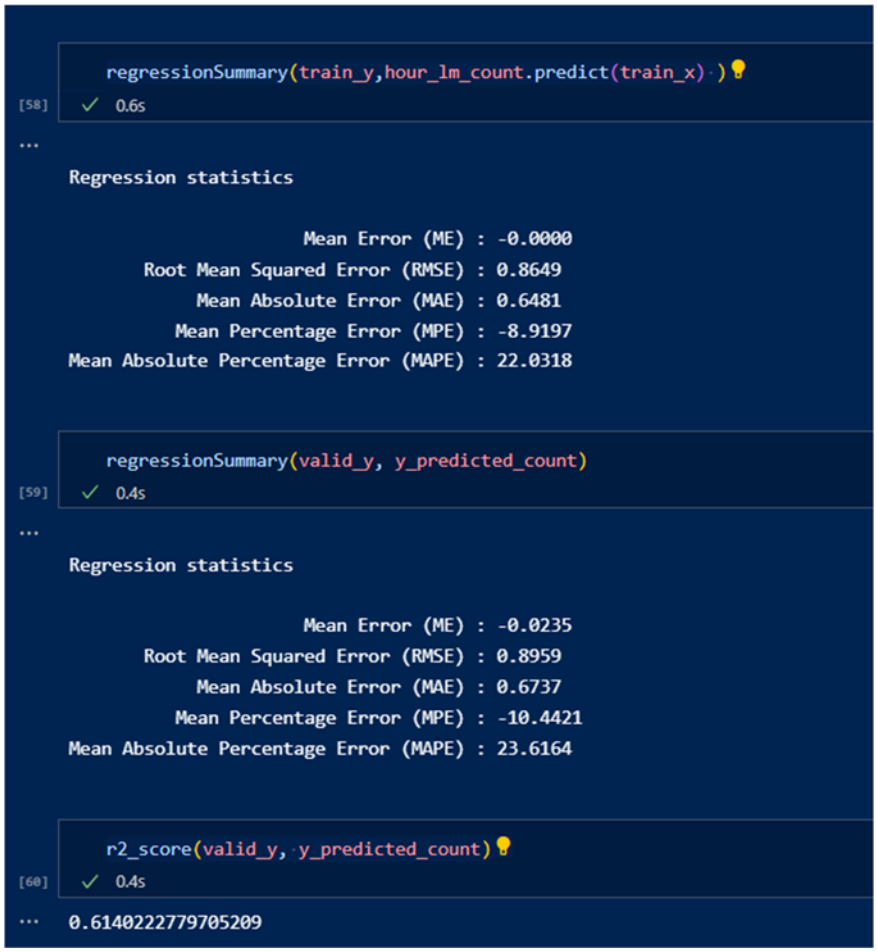
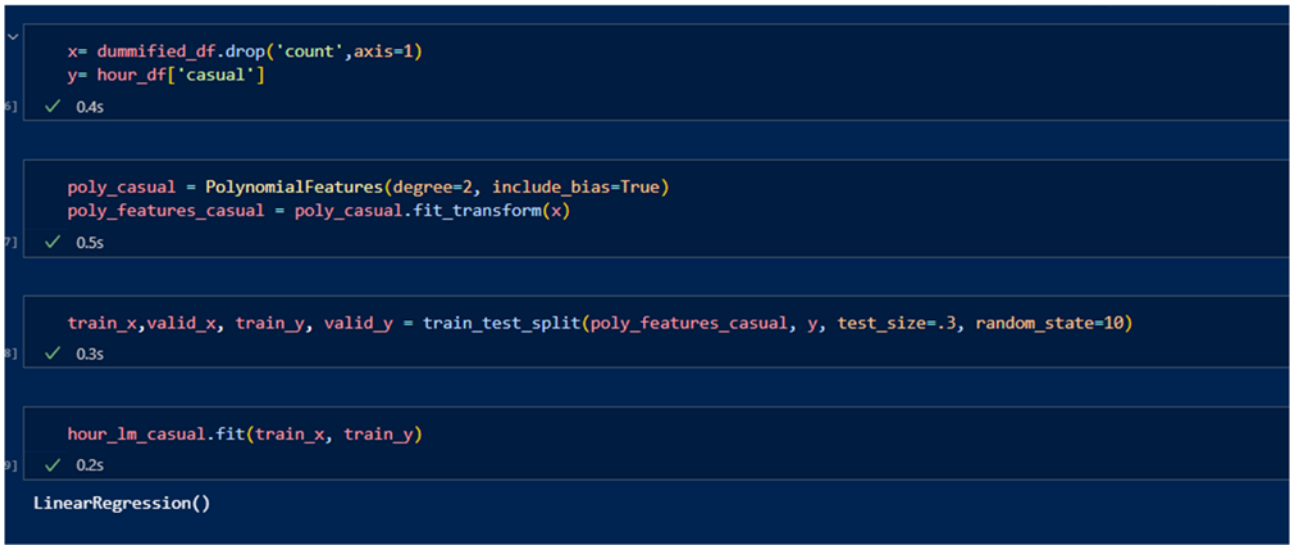


Figure 48. code for regression Summary of train and validation sets performance and r2 score

20.2 Polynomial Regression of 'casual'



```
y_predicted_casual = hour_lm_casual.predict(valid_x)
y_predicted_casual
```

✓ 0.2s

```
array([3.74220877, 2.14460878, 3.30913615, ..., 4.1907431 , 4.11114574,
       3.98025054])
```

```
regressionSummary(train_y,hour_lm_casual.predict(train_x) )
```

✓ 0.4s

```
Regression statistics

                Mean Error (ME) : 0.0000
Root Mean Squared Error (RMSE) : 0.7806
     Mean Absolute Error (MAE) : 0.6001
```

```
regressionSummary(valid_y, y_predicted_casual)
```

✓ 0.3s

```
Regression statistics

                Mean Error (ME) : -0.0108
Root Mean Squared Error (RMSE) : 0.8045
     Mean Absolute Error (MAE) : 0.6160
```

```
r2_score(valid_y, y_predicted_casual)
```

✓ 0.3s

```
0.7139070218812884
```

20.3 Polynomial Regression of 'registered'

```
x= dummified_df.drop('count',axis=1)
y= hour_df['registered']
```

```
poly_registered = PolynomialFeatures(degree=2, include_bias=True)
poly_features_registered = poly_registered.fit_transform(x)
```

```
train_x,valid_x, train_y, valid_y = train_test_split(poly_features_registered, y, test_size=.3, random_state=10)
```

```
hour_lm_registered.fit(train_x, train_y)
```

```
LinearRegression()
```

```
y_predicted_registered = hour_lm_registered.predict(valid_x)
```

```
regressionSummary(train_y,hour_lm_registered.predict(train_x) )
```

```
[51] ✓ 0.4s
```

```
Regression statistics

               Mean Error (ME) : -0.0000
Root Mean Squared Error (RMSE) : 0.8845
     Mean Absolute Error (MAE) : 0.6681
```

```
regressionSummary(valid_y, y_predicted_registered)
```

```
[52] ✓ 0.6s
```

```
Regression statistics

               Mean Error (ME) : -0.0252
Root Mean Squared Error (RMSE) : 0.9134
     Mean Absolute Error (MAE) : 0.6937
```

```
r2_score(valid_y, y_predicted_registered)
```

```
[53] ✓ 0.4s
```

```
0.5874081747170266
```

21. Model Comparison

Model name	Training RMSE	Validation RMSE	R2 value
Linear Regression(count)	1.0202	1.0355	0.4841
Linear Regression(casual)	0.9283	0.9461	0.6041
Linear Regression(registered)	1.0327	1.0452	0.4596
Random Forest (count)	0.1409	0.3979	0.9238
Random Forest (casual)	0.1887	0.5221	0.8794
Random Forest (registered)	0.1433	0.3996	0.9209
Polynomial regression (count)	0.8649	0.8959	0.6143
Polynomial regression (casual)	0.7806	0.8045	0.7142
Polynomial regression (registered)	0.8845	0.9134	0.5877

22. Model Selection

All three models are overfitted but Random Forest for three target variables is the most overfitted because there is huge difference between training and validation sets. Even though this model has very high R2 score (mostly greater than 0.9), it seems to pass through many data points in this regression model which probably makes it less accurate. Therefore, Random Forest is not selected for this analysis.

Linear Regression model turns out almost equal train and validation errors. However, the R2 score is not high enough to be the best model. On the other hand, Polynomial Regression model does not show much difference between train and validation sets, R2 value is fair enough to be the best model for this analysis.

23. Validation and Governance

23.1 Risk management

Model governance is a complex term that can define in words because it is intangible where we cannot touch or feel it. Models are built that they can be changed, modified and adjusted (Eisenstein, 2020). The main point of using models is to indicate the organization needs to improve. The constant changing models, impact business where they rely significantly on data scientist to enhance functions of the business (Datatron, 2022), result in arising risk and decision-making process.

The dataset was collected in 2013 by Hadi Fanaee-T at Laboratory of Artificial Intelligence and Decision Support (LIAAD), University of Porto and extracted from 2011 to 2012 with original source from Capital Bikeshare homepage, weather information and holiday schedule from the extracted links. Therefore, there is high potential of risks:

I am predicting the impact of environmental impact on bike share rental riders, the data is outdated so it could not reflect the most current behaviors of rider accurately when the climate change including weather situation, season, and temperature that fluctuate dynamically over the years.

The dataset can only indicate the total count of riders but not number of people. So it could lead to the bias of the same person who rents the bike more than once that also causes the model less accurate regarding the target variables which are casual and registered riders.

Also, the cost involving in the business cannot be calculated even though the price plan is displayed on company's website. Thus, it is difficult to recommend the business if they need to buy more bikes or just purchase certain number of bikes. The hypothesis occurs with two scenarios:

If Capital Bikeshare buys more bikes, it is unpredictable the number of riders who rent bikes depending on number of different factors (overpredicting).

If Capital Bikeshare buys less bikes, riders will find unsatisfied with empty bike at docking stations (underpredicting).

Moreover, the dataset does not contain price of gas. Thus, let's assume more people would be interested in renting a bike to save money for gas for instance. In this case, it is more biased to predict the number of people who would like to ride for this factor because it is not relevant to environmental impact that I am going to predict in this project.

Because of these hidden problems, the adjusted R2 score of total count (including casual riders and registered riders) of Linear Regression model is not very high.

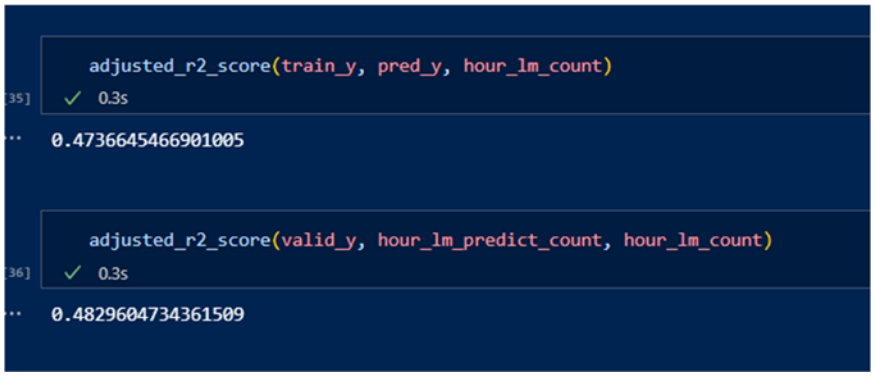


Figure 49. adjusted R2 score of Linear Regression of ‘count’

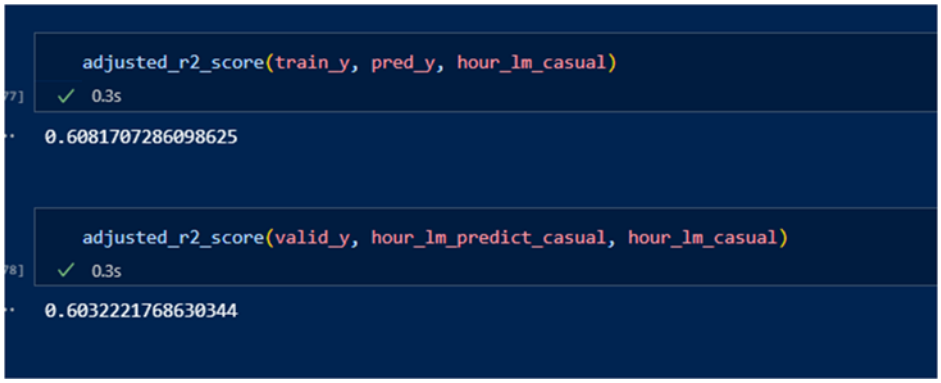


Figure 50. adjusted R2 score of Linear Regression of ‘casual’

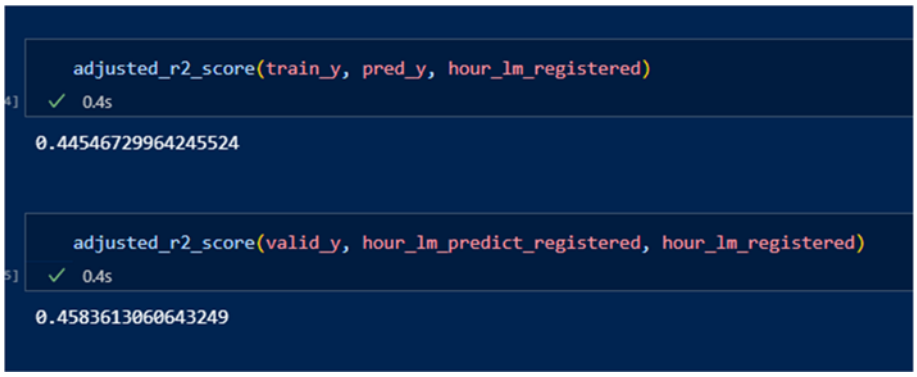


Figure 51. adjusted R2 score of Linear Regression of ‘registered’

Rebalancing is the hardest part of bike sharing (Gossett, 2020) whereas it is also the risk for the organization to optimize the business expenses.

23.2 Variable level monitoring

These are accepted ranges of the data input variables. These variables might change over time, for that reason, we need to re-train the models based on the historical data and future data.

	mean	std.dev	min	max	median	length
temp	0.496987	0.192556	0.02	1.0000	0.5000	17379
atemp	0.475775	0.171850	0.00	1.0000	0.4848	17379
humidity	0.627229	0.192930	0.00	1.0000	0.6300	17379
windspeed	0.190098	0.122340	0.00	0.8507	0.1940	17379
casual	35.676218	49.305030	0.00	367.0000	17.0000	17379
registered	153.786869	151.357286	0.00	886.0000	115.0000	17379
count	189.463088	181.387599	1.00	977.0000	142.0000	17379

Figure 1. The statistics of all variables in the dataset

23.3 Missing values

In this dataset, there is no missing values but there are outliers from numeric variables, so I decided to use cap & floor method to remove outliers and reduce skewness of those variables

23.4 Cap & Floor

I did cap & floor the numeric data including ‘humidity’, ‘temp’, ‘atemp’, and ‘windspeed’. As I have seen ‘windspeed’ has the most outliers while ‘humidity’ has less outliers, so they need to be removed from the dataset. The cap-and-floor operation was carried out on values within 1.5 times the Interquartile Range.

```
hour_cap = hour_df
features = ['humidity','temp','atemp','windspeed']
def iqr_capping(df, cols, factor):

    for col in cols:

        q1 = df[col].quantile(0.25)
        q3 = df[col].quantile(0.75)

        iqr = q3 - q1

        upper_whisker = q3 + (factor*iqr)
        lower_whisker = q1 - (factor*iqr)

        df[col] = np.where(df[col]>upper_whisker, upper_whisker,
                           np.where(df[col]<lower_whisker, lower_whisker, df[col]))

iqr_capping(hour_cap, features, 1.5)
```

23.5 Variable Drift Monitoring

As I mentioned earlier, the variables will drift due to climate change and extreme weather events. Furthermore, changing lifestyles and population demographics will affect my model's stability. As time progresses, my model's stability will deteriorate. Changing weather conditions can be monitored to ensure values are within variable value thresholds determined during EDA. Changing lifestyle and behaviors also monitor through lifestyle survey.

23.6 Model Health & Stability

There are number of methods to measure stability and drift, one of the most popular methods is being used in this project is Root Mean Squared Error (RMSE) which is the average prediction error for the future observations while lower values are better. Recalling from the model comparison table above, Polynomial Regression of three target variables shows better result than Linear Regression and Random Forest. To be noticed, Random Forest model is sensitive to variance and it can generalize the ability of decision tree. In this dataset, there are many categorial variables which make the decision tree fail. As a result, it might make the algorithm too slow and ineffective for real-time predictions. Theoretically, Random Forest only random selects the important features to create regression tree. In this analysis, due to limitation in the dataset, it is hard to pick the right variables for prediction to turn out the accurate results. Polynomial Regression performs well in this dataset because there is no-linear correlation between the variables.

23.7 Risk Tiering

Unacceptable risk	High risk	Limited risk	Minimal risk
•high bias of using bike share in Eastern and Western countries •Climate change and extreme weather events	•cost involvement within business: •overpredicting: buy more bikes, they lose money if not a lot of people are interested in bike share •underpredicting: less bikes, customers are not satisfied with the service	•changing lifestyle, population and demographics •renting and returning bikes to docking stations attitudes	•Use of application to unlock bikes

IV. Conclusion and Recommendations

24. Impacts on Business Problem

Using linear regression, polynomial regression and random forest, we discovered that hour and temperature are the most important factors that drive the demand of bike share system. The results of what the models are telling us are not as expected such as season or weather situation that should have been the key factors on the riders. Hence, according to the results, Capital Bikeshare relies on those two factors to decide number of bikes at each station the riders tend to use. They have to deal with unbalancing problem: available bikes at certain time of a day, so the customers may find poor service quality. Thus, reserving a certain number of bikes at each station is one of the solutions to resolve this problem.

25. Recommended Next Steps

The first recommendation is to add more variables to the dataset that helps us based on the demand to predict the probability of bikes when the riders reach at the recommended station. By doing so, we know the most popular stations for rental and return to refine station performance (Billhardt, 2021).

The second recommendation is to combine the casual riders with social activities such as exploration routes to new stations with low cost. For this purpose, the system not only tries to convert casual riders to membership riders due to long distance trips but also improve the overall distribution of the bikes at the prioritized stations.

The third recommendation is to bring the casual riders to understand and be aware of traffic congestion problem at peak hours. If they are able to track the traffic and save time to commute, they are willing to use bike rentals more often, it also implies that they are thinking to join Capital Bikeshare as memberships (Brandon, 2021)

References

1. PBSC Urban Solutions. (October 27, 2021). *The Meddin Bike-Sharing World Map*. Retrieved August 21, 2022 from https://www.pbsc.com/blog/2021/10/the-meddin-bike-sharing-world-map#:~:text=How%20Many%20Bike%20Sharing%20Systems,in%20cities%20around%20the%20globe.

2. Capital Bikeshare. (n.d.). *About Capital Bikeshare*. Retrieved August 21, 2022 from https://ride.capitalbikeshare.com/about

3. Teale, C. (December 3, 2018). *Deal of the Year: Lyft's acquisition of Motivate*. SMARTCITIESDIVE. Retrieved August 21, 2022 from https://www.smartcitiesdive.com/news/deal-of-the-year-lyft-motivate-acquisition/539476/#:~:text=As%20ride%2Dsharing%20companies%20looked,%2C%20Minneapolis%20and%20Washington%2C%20DC.

4. Bryce, K. (n.d). *The Many Benefits of Bike Sharing Programs*. Commute options. Retrieved July 15, 2022, from https://www.commuteoptions.org/the-many-benefits-of-bike-sharing-programs/

5. Stott, S. (2020, October 30). *How green is cycling? Riding, walking, ebikes and driving ranked*. Bikeradar. Retrieved July 15, 2022, from https://www.bikeradar.com/features/long-reads/cycling-environmental-impact/

6. Fanaee-T, H. (December 20, 2013). Bike Sharing Dataset Data Set. UCI Machine Learning Repository. Retrieved August 21, 2022 from https://archive.ics.uci.edu/ml/datasets/Bike+Sharing+Dataset#

7. V, L. (October 14, 2021). *Dealing with Missing Values for Data Science Beginners*. Analytics Vidhya. Retrieved August 21, 2022 from https://www.analyticsvidhya.com/blog/2021/10/guide-to-deal-with-missing-

values/#:~:text=because%20the%20data%20is%20not,on%20the%20mechanism%20of%20missingness.

8. Pandas Get Dummies Function – get_dummies(). (n.d.). Data Science Parichay. Retrieved August 21, 2022 from https://datascienceparichay.com/article/pandas-get-dummies-function/

9. Mbaabu, O. (December 11, 2020). *Introduction to Random Forest in Machine Learning.* Section. Retrieved August 21, 2022 from https://www.section.io/engineering-education/introduction-to-random-forest-in-machine-learning/

10. Jelic, A. (June 2021). *Predicting bike sharing demand with machine learning.* Tilburg University. Retrieved August 21, 2022 fromhttp://arno.uvt.nl/show.cgi?fid=156914

11. Polynomial regression: Everything you need to know!. (n.d.). voxco. Retrieved August 21, 2022 from https://www.voxco.com/blog/polynomial-regression-everything-you-need-to-know/#:~:text=Polynomial%20regression%20is%20used%20when,like%20a%20non%2Dlinear%20function.

12. Datatron. (2022). *What is Model Governance and Why it's Important*. Retrieved August 13, 2022 from https://datatron.com/model-governance/

13. Eisenstein, L. (2020, January 31). *What Is a Governance Model and Why Does it Matter? Board Effect*. Retrieved August 13, 2022 from https://www.boardeffect.com/blog/what-governance-model-why-does-matter/

14. Gossett, S. (2020, August 17). *Bike-Sharing Rebalancing Is a Classic Data Challenge That Just Got a Lot Harder*. Builtin. Retrieved August 13, 2022 from https://builtin.com/data-science/bike-share-rebalancing

15. Brandon. (September 1, 2021). *How can we convert casual bike-share riders to annual memberships?* RPubs. Retrieved August 21, 2022 from https://rpubs.com/bsixto/807686

16. Billhardt et al. (2021). 2021. *Smart Recommendations for Renting Bikes in Bike-Sharing Systems.* Madrid: MDPI, pp.2-3.