Homework 2

BUSN 41204 - 2023

- Aman Krishna
- Christian Pavilanis
- Jingwen Li
- Yazmin Ramirez Delgado

1 Description

In a bike sharing system the process of obtaining membership, rental, and bike return is automated via a network of kiosk locations throughout a city. In this problem, you will try to combine historical usage patterns with weather data to forecast bike rental demand in the Capital Bikeshare program in Washington, D.C.

You are provided hourly rental data collected from the Capital Bikeshare system spanning two years. The file Bike_train.csv, as the training set, contains data for the first 19 days of each month, while Bike_test.csv, as the test set, contains data from the 20th to the end of the month.

In [ ]:

```
# First we import the necessary libraries
import os
import pandas as pd
pd.set_option("display.precision", 4)
import numpy as np
from datetime import datetime
from datetime import timedelta
from matplotlib import pyplot as plt
from sklearn.model_selection import GridSearchCV
import functools
from scipy import stats
import seaborn as sn
from sklearn.model_selection import LeavePOut
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import LeaveOneOut, cross_val_score
from sklearn.model_selection import train_test_split
from statsmodels.formula.api import ols
import scipy as sp
import plotnine as p9
from sklearn.tree import DecisionTreeClassifier
import warnings
warnings.filterwarnings("ignore")
import random
import math
from sklearn.metrics import mean_squared_error
from sklearn.neighbors import KNeighborsRegressor
import statsmodels.api as sm
pd.options.display.float_format = '{:.4f}'.format
pd.options.mode.chained_assignment = None  # default='warn'
from IPython.display import Markdown, display
def printmd(string):
    display(Markdown(string))
```

In [ ]:

```
# Let's read the data
train = pd.read_csv('Bike_train.csv')
test = pd.read_csv('Bike_test.csv')
```

2 Questions

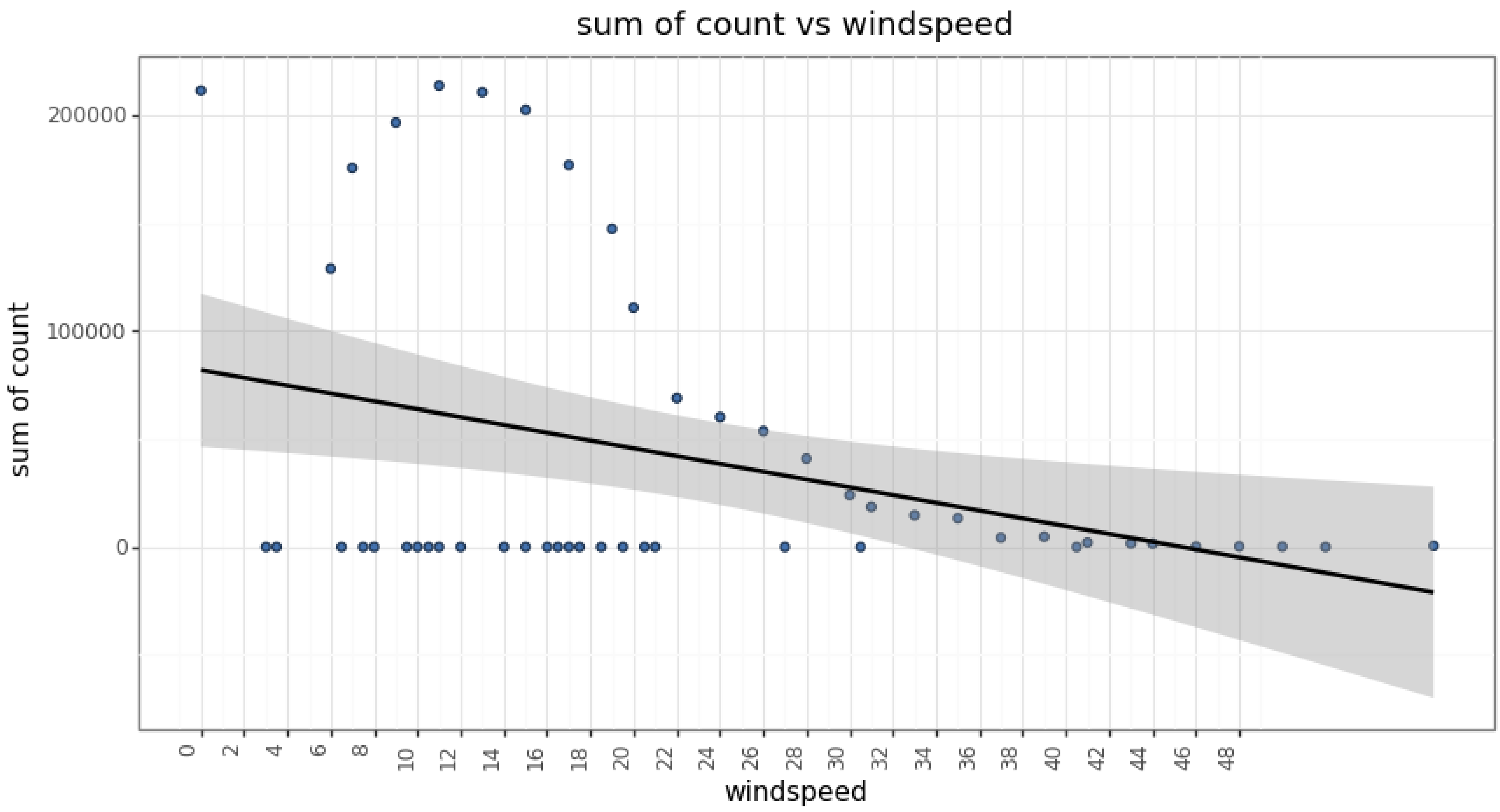
1. Before you build your predictive model, let us first explore the data.

a. Visualize the relationship between count and each one of the following variables on a separate scatter plot: windspeed, humidity, temp, and atemp.

Windspeed

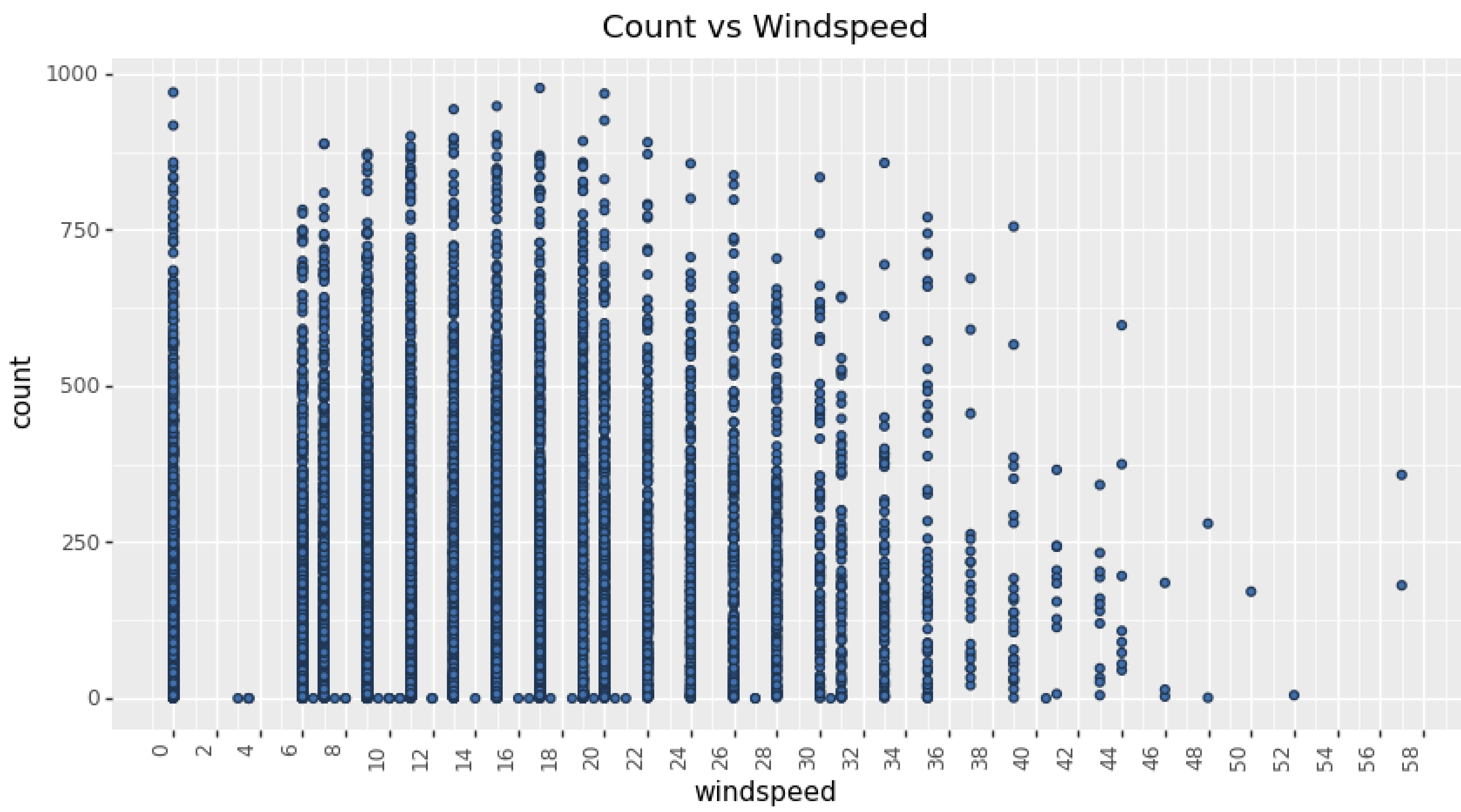
In [ ]:

```
# find the sum of count for each unique wind speed and store it in a new dataframe
wind_speed_count = train.groupby('windspeed').sum()['count']
wind_speed_count = pd.DataFrame(wind_speed_count)
wind_speed_count.reset_index(inplace=True)
wind_speed_count.columns = ['windspeed', 'sum of count']

p9.ggplot(wind_speed_count, p9.aes(x='windspeed', y='sum of count')) + p9.geom_point(colour="#1F3552", fill="#4271AE")
    + p9.geom_smooth(method='lm', se=True) + p9.theme_bw() + p9.theme(figure_size=(10, 5), axis_text_x=p9.element_text
        + p9.scale_x_continuous(breaks=range(0, 50, 2)) + p9.labs(title='sum of count vs windspeed')
```



Out[ ]: <ggplot: (695964558)>

In [ ]:
```
# Scatter plot for windspeed vs count, sum the count for each windspeed
# Path: week3/hw2/bike_codebook.ipynb
p9.ggplot(train, p9.aes(x='windspeed', y='count')) + p9.geom_point(colour="#1F3552", fill="#4271AE") \
    + p9.theme(figure_size=(10, 5), axis_text_x=p9.element_text(angle=90, hjust=1)) + \
        p9.scale_x_continuous(breaks=range(0, 100, 2)) + p9.labs(title='Count vs Windspeed')
```

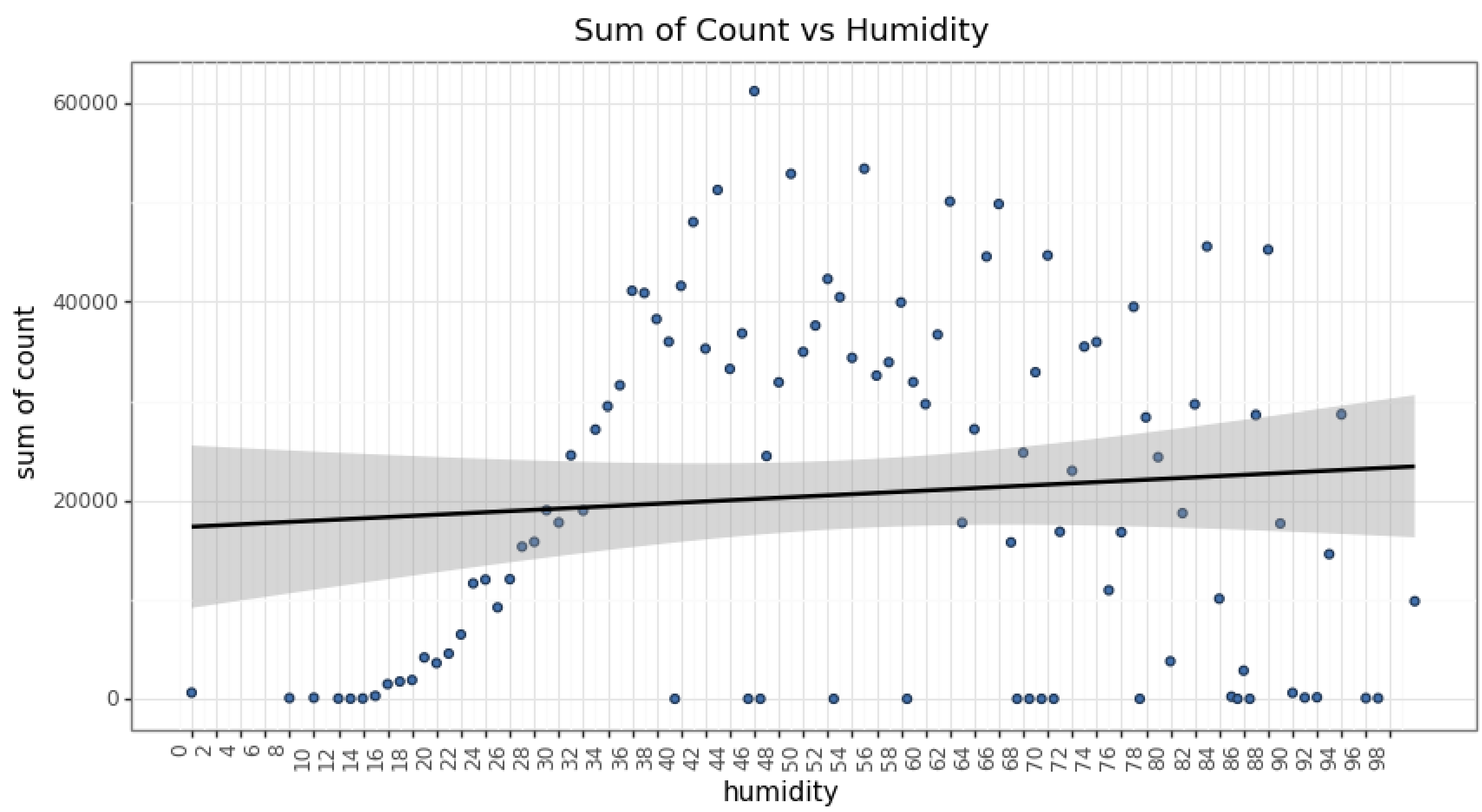


Out[ ]: <ggplot: (378804848)>

- It is observed that when the windspeed is higher the count is lower.
- Furthermore, we see a few outliers in the scatter plot which will be treated later in our analysis before the training of the model.

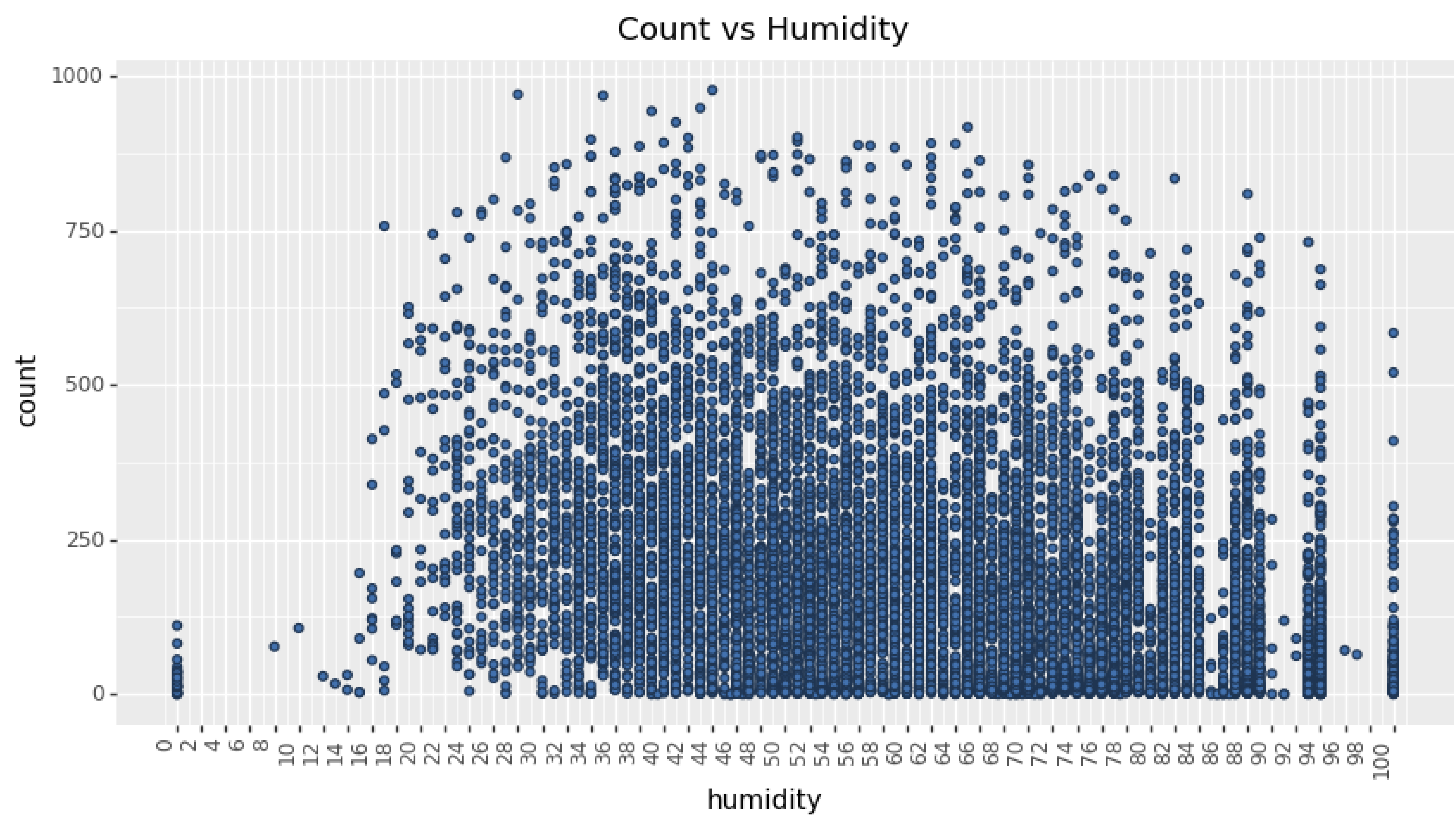
Humidity

In [ ]:
```
# find the sum of count for each unique wind speed and store it in a new dataframe
humudity_count = train.groupby('humidity').sum()['count']
humudity_count = pd.DataFrame(humudity_count)
humudity_count.reset_index(inplace=True)
humudity_count.columns = ['humidity', 'sum of count']

p9.ggplot(humudity_count, p9.aes(x='humidity', y='sum of count')) + p9.geom_point(colour="#1F3552", fill="#4271AE")\
    + p9.geom_smooth(method='lm', se=True) + p9.theme_bw() + p9.theme(figure_size=(10, 5), axis_text_x=p9.element_text
        + p9.scale_x_continuous(breaks=range(0, 100, 2)) + p9.labs(title='Sum of Count vs Humidity')
```



Out[ ]: <ggplot: (355843042)>

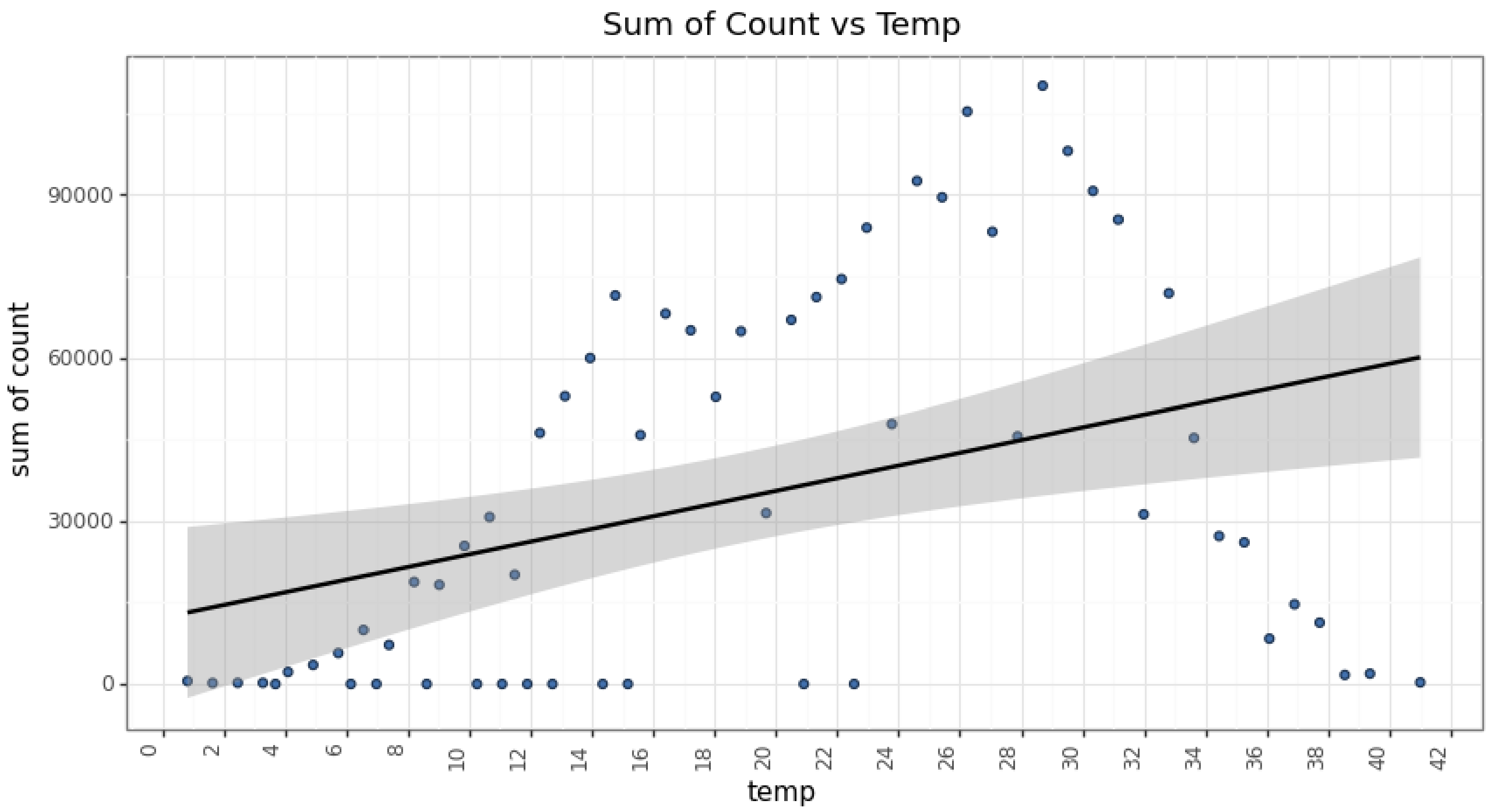
In [ ]:
```
# Scatter plot for humidity vs count, sum the count for each humidity
p9.ggplot(train, p9.aes(x='humidity', y='count')) + p9.geom_point(colour="#1F3552", fill="#4271AE") +\
    p9.theme(figure_size=(10, 5), axis_text_x=p9.element_text(angle=90, hjust=1)) +\
        p9.scale_x_continuous(breaks=range(0, 102, 2)) + p9.labs(title='Count vs Humidity')
```



Out[ ]: <ggplot: (696456569)>

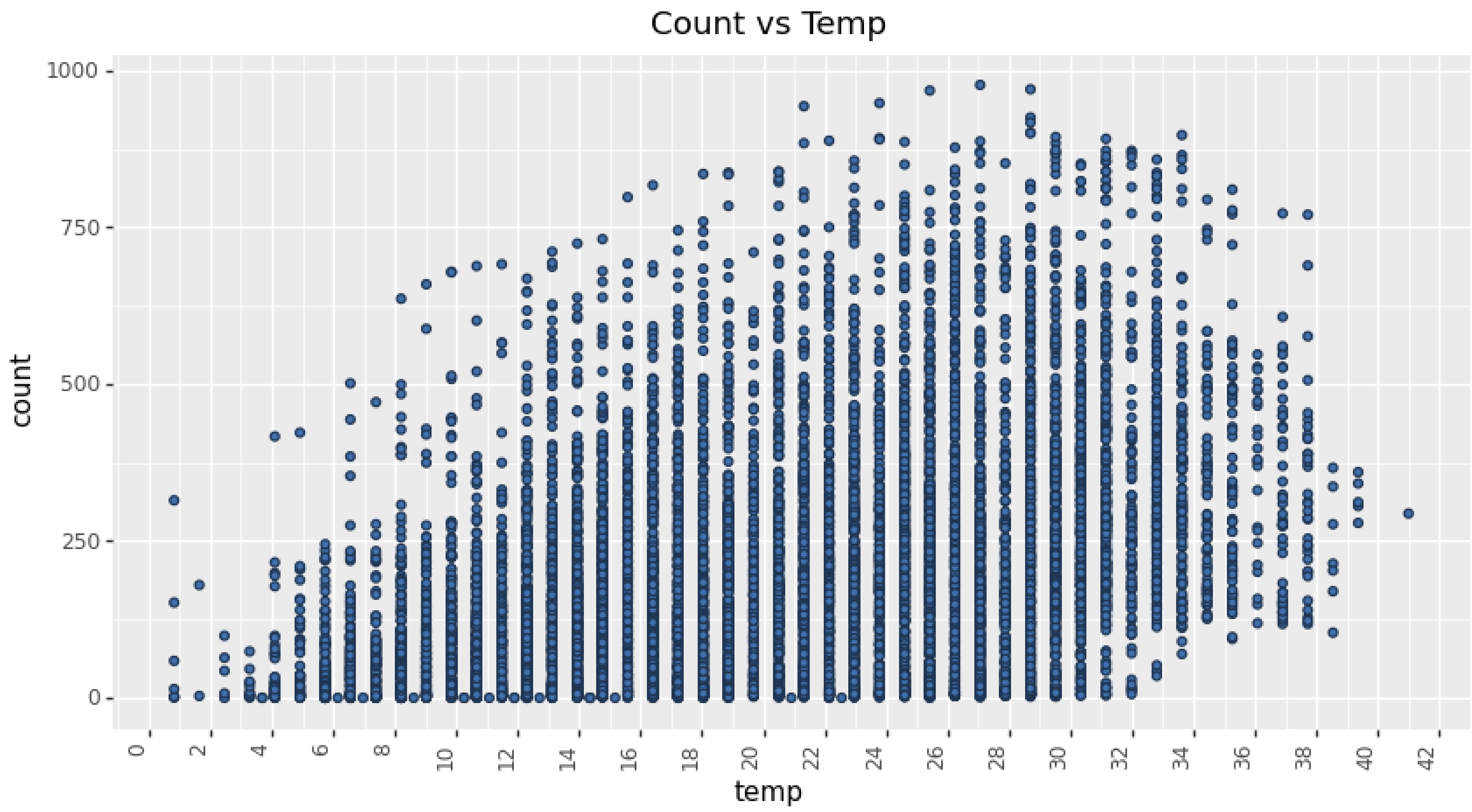
Temperature

In [ ]:
```
# find the sum of count for each unique temp and store it in a new dataframe
temp_count = train.groupby('temp').sum()['count']
temp_count = pd.DataFrame(temp_count)
temp_count.reset_index(inplace=True)
temp_count.columns = ['temp', 'sum of count']

p9.ggplot(temp_count, p9.aes(x='temp', y='sum of count')) + p9.geom_point(colour="#1F3552", fill="#4271AE") +\
    p9.geom_smooth(method='lm', se=True) + p9.theme_bw() + p9.theme(figure_size=(10, 5), axis_text_x=p9.element_text(a
        + p9.scale_x_continuous(breaks=range(-10, 100, 2)) + p9.labs(title='Sum of Count vs Temp')
```



Out[ ]: <ggplot: (381647671)>

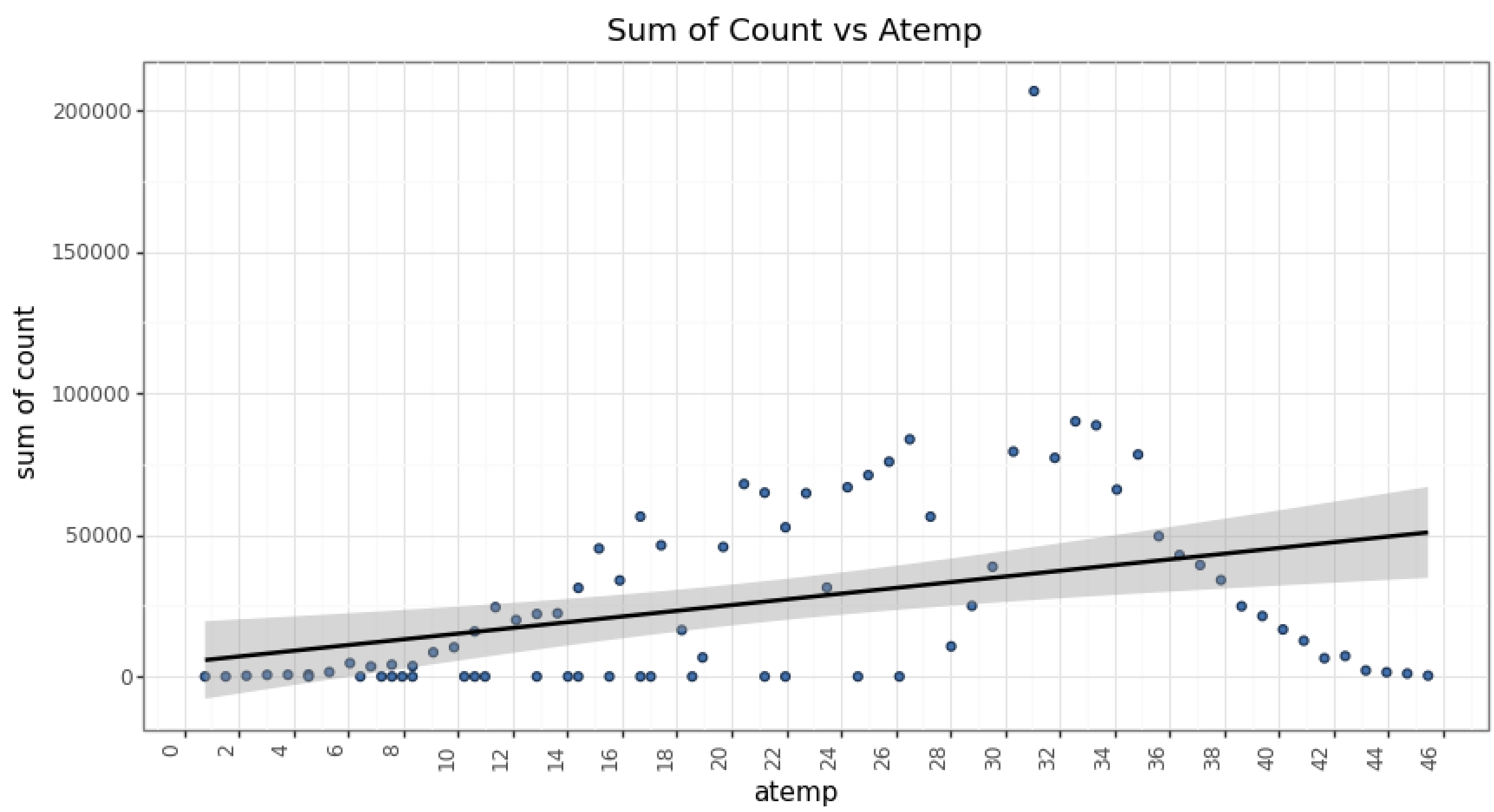
In [ ]:
```
# Scatter plot for temp vs count, sum the count for each temp
p9.ggplot(train, p9.aes(x='temp', y='count')) + p9.geom_point(colour="#1F3552", fill="#4271AE") +\
    p9.theme(figure_size=(10, 5), axis_text_x=p9.element_text(angle=90, hjust=1)) +\
        p9.scale_x_continuous(breaks=range(0, 100, 2)) + p9.labs(title='Count vs Temp')
```



Out[ ]: <ggplot: (696598125)>

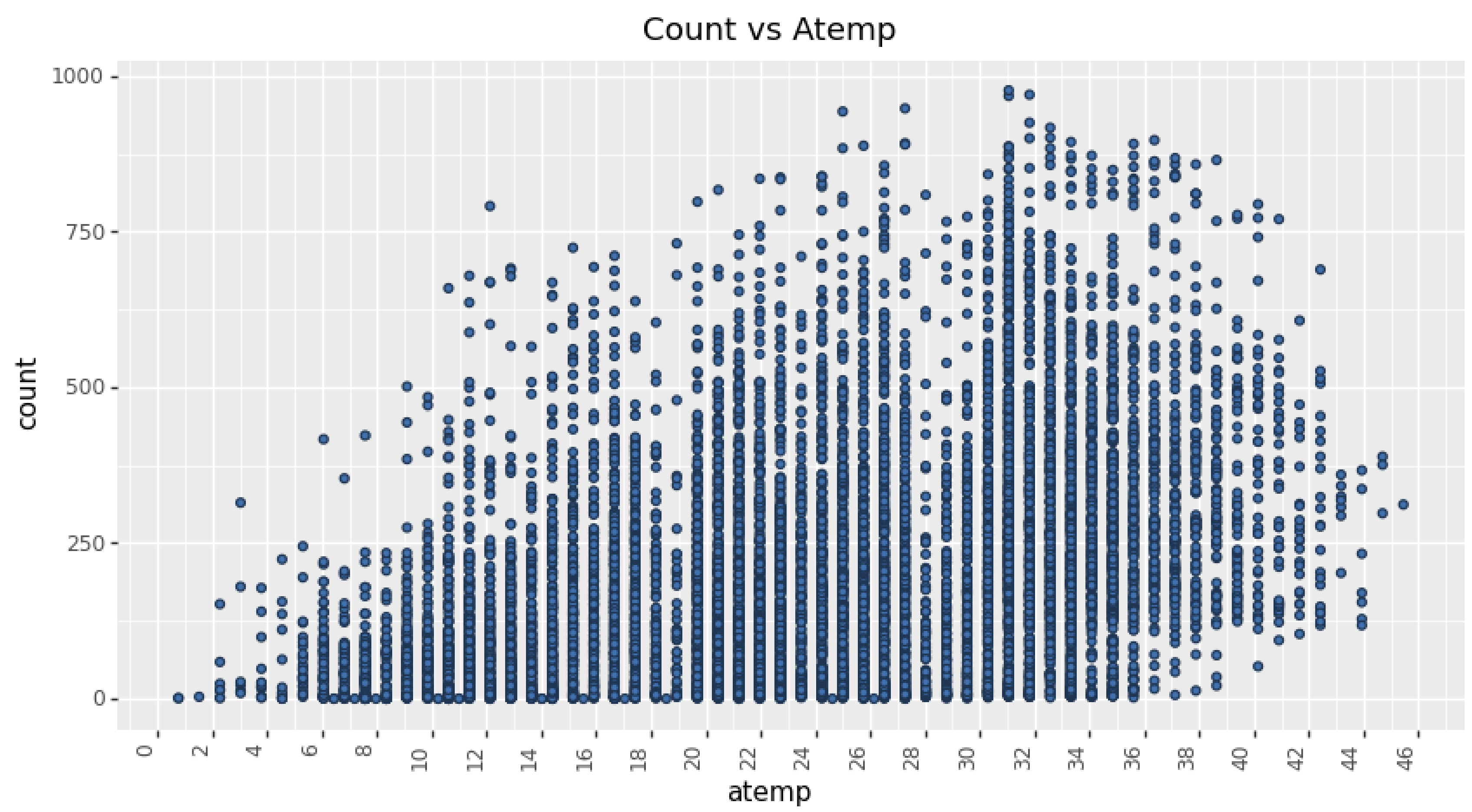
Feel likes temperature

In [ ]:
```
# find the sum of count for each unique wind speed and store it in a new dataframe
atemp_count = train.groupby('atemp').sum()['count']
atemp_count = pd.DataFrame(atemp_count)
atemp_count.reset_index(inplace=True)
atemp_count.columns = ['atemp', 'sum of count']

p9.ggplot(atemp_count, p9.aes(x='atemp', y='sum of count')) + p9.geom_point(colour="#1F3552", fill="#4271AE") + \
    p9.geom_smooth(method='lm', se=True) + p9.theme_bw() + p9.theme(figure_size=(10, 5), axis_text_x=p9.element_text(a
        + p9.scale_x_continuous(breaks=range(0, 50, 2)) + p9.labs(title='Sum of Count vs Atemp')
```



Out[ ]: <ggplot: (381055769)>

In [ ]:
```
# Scatter plot for atemp vs count, sum the count for each atemp
p9.ggplot(train, p9.aes(x='atemp', y='count')) + p9.geom_point(colour="#1F3552", fill="#4271AE") + \
    p9.theme(figure_size=(10, 5), axis_text_x=p9.element_text(angle=90, hjust=1)) +\
        p9.scale_x_continuous(breaks=range(0, 100, 2)) + p9.labs(title='Count vs Atemp')
```



Out[ ]: <ggplot: (694825244)>

- Apparently, both the temperature and feels like temperature have a positive correlation with the count (we will validate this later in the analysis via the correlation matrix).
- The extreme temperatures (either too hot or too cold) have a negative impact on the count.
- But, the count is left skewed vs temperature, which means that people would rather bike in a hotter temperature than a colder one.

b. How does count depend on the season? Consider visualizing this relationship with a boxplot.

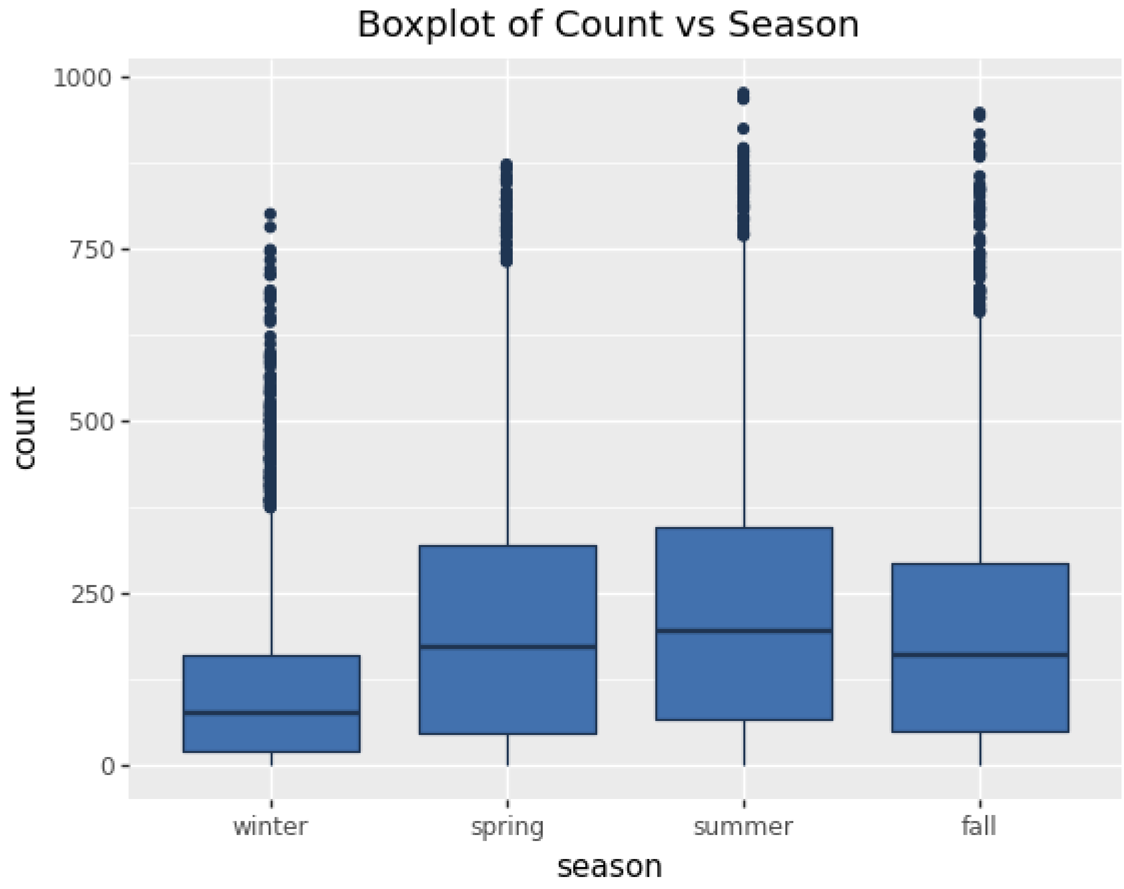
In [ ]:
```
#find the cum of count for each unique season and store it in a new dataframe
season_count = train.groupby('season').sum()['count']
season_count = pd.DataFrame(season_count)
season_count.reset_index(inplace=True)
season_count.columns = ['season', 'sum of count']
season_count.set_index('season', inplace=True)
season_count.index = season_count.index.map({1: 'winter', 2: 'spring', 3: 'summer', 4: 'fall'})
season_count
```

Out[ ]:

season	sum of count
winter	312498
spring	588282
summer	640662
fall	544034

In [ ]:

```
train["season"] = train["season"].astype("category")
train["season"] = train["season"].map({1: 'winter', 2: 'spring', 3: 'summer', 4: 'fall'})
p10 = p9.ggplot(train, p9.aes("season", "count")) + p9.geom_boxplot(colour="#1F3552", fill="#4271AE") + p9.labs(title=
p10
```



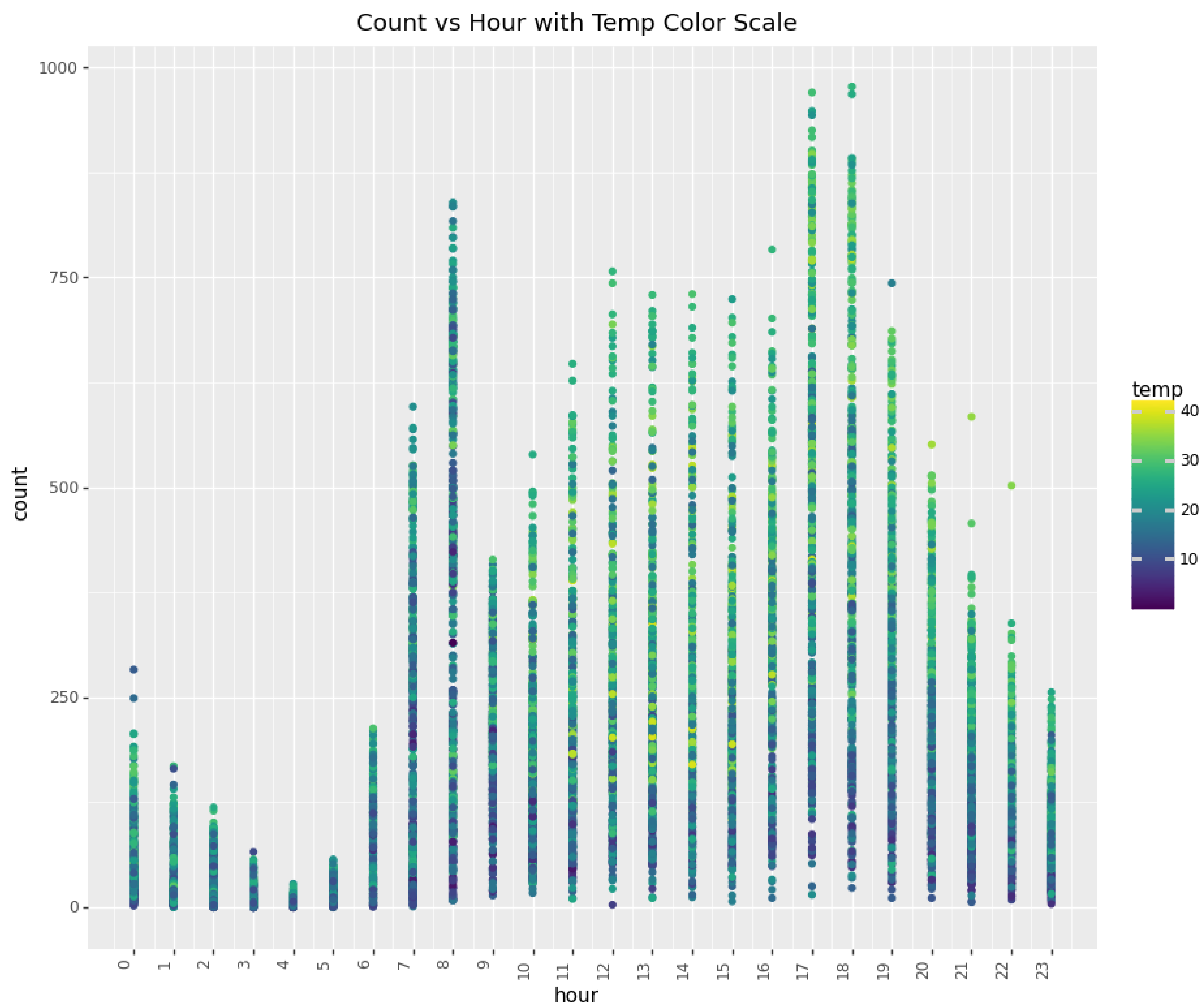
Out[ ]: <ggplot: (695010943)>

It is observed that Winter season has the lowest mean and sum of counts. The Spring and Summer seasons have the higher mean and sum of counts.

c. How does count depend on the time of the day (hour)? Does this relationship change depending on whether it is a workingday or not? A scatterplot could be used to visualize the relationship. You might consider coloring the observations on the scatterplot using the temperature (temp or atemp) do discern how the temperature affects hourly number of rentals.

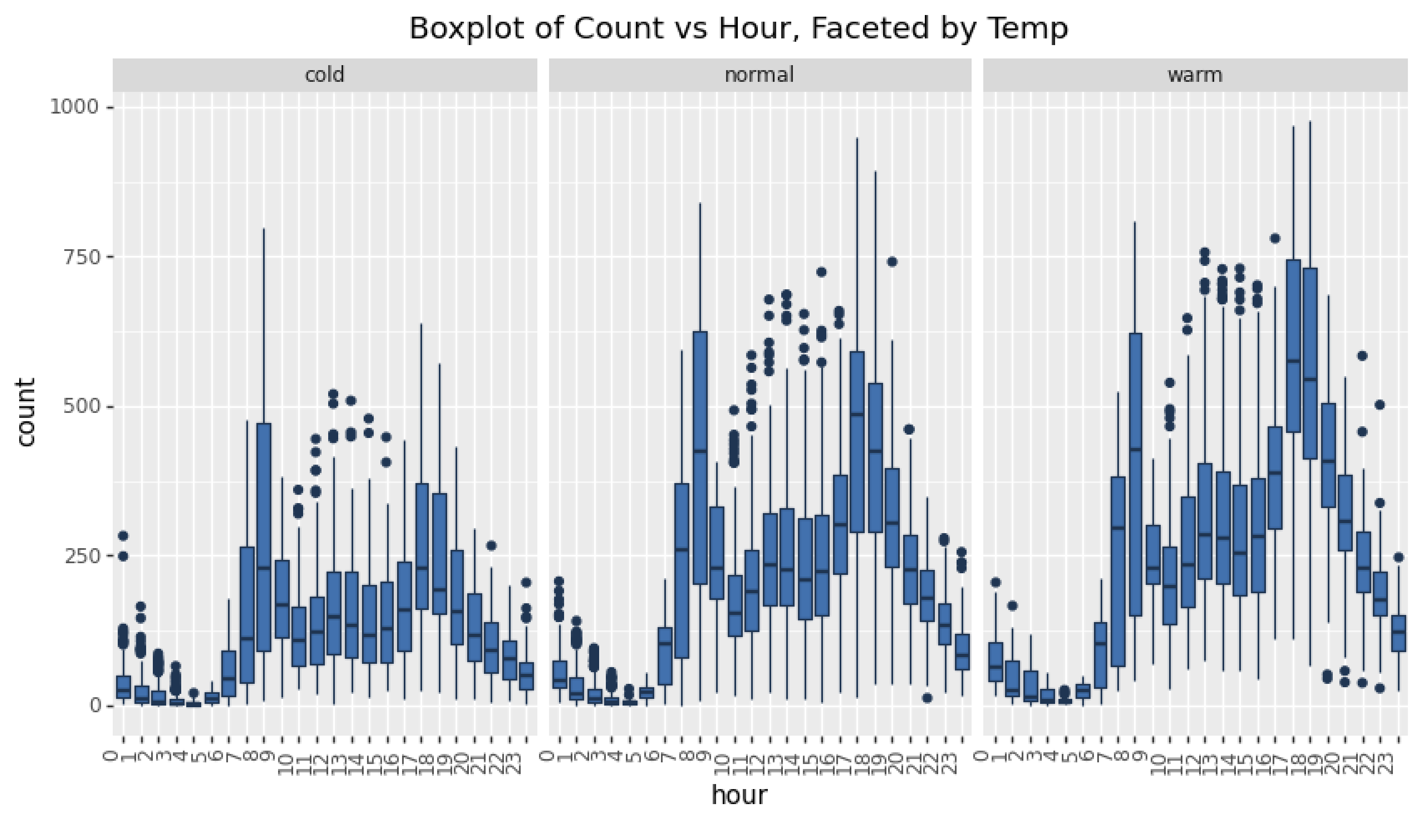
In [ ]:

```
#Create a scatter plot for time of day vs count and color the points by temp
p9.ggplot(train, p9.aes(x='hour', y='count', color='temp')) + p9.geom_point() +\
    p9.theme(figure_size=(10, 9), axis_text_x=p9.element_text(angle=90, hjust=1)) + \
        p9.scale_x_continuous(breaks=range(0, 24, 1)) + p9.labs(title='Count vs Hour with Temp Color Scale')
```



Out[ ]: <ggplot: (381633235)>

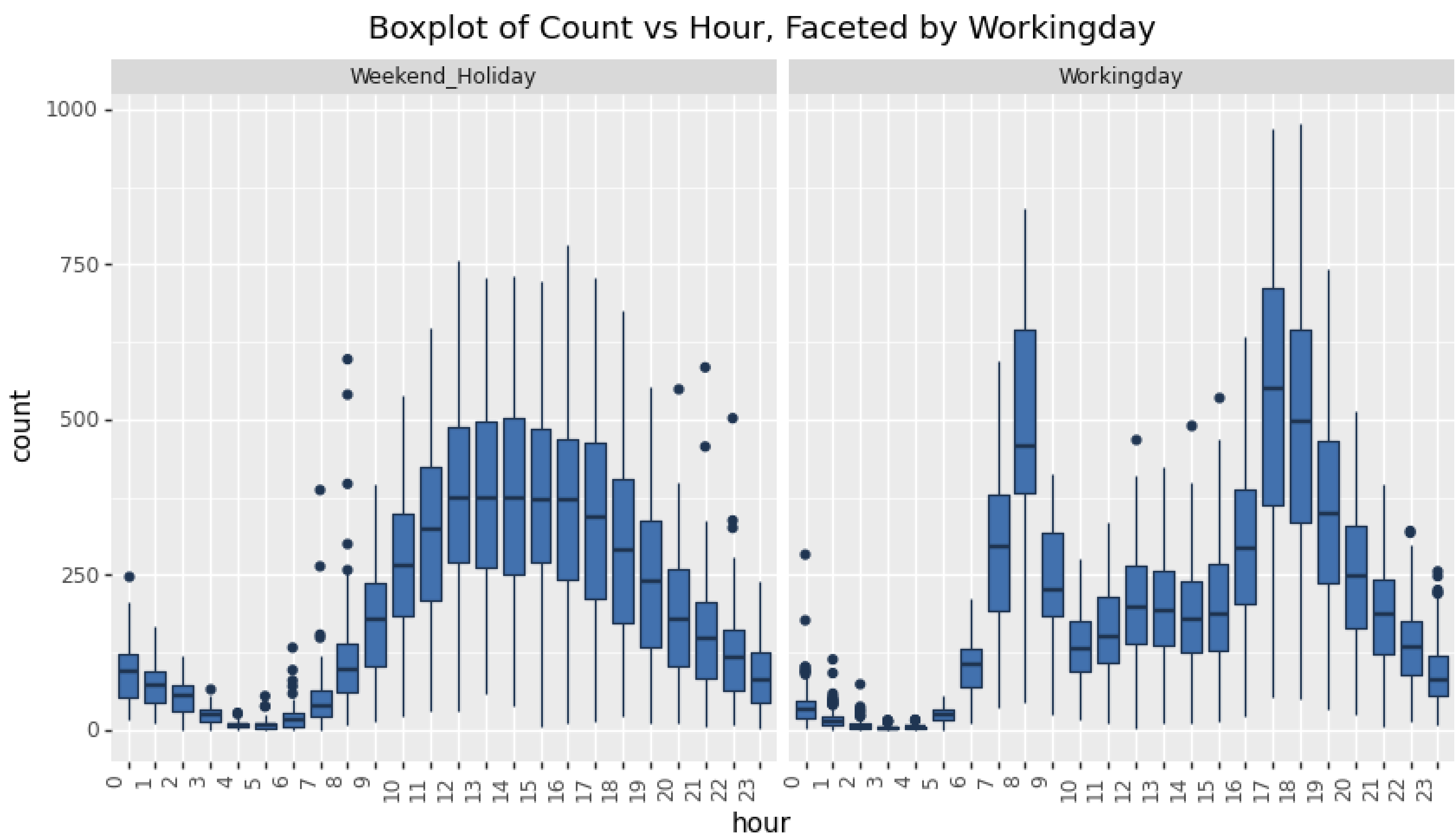
In [ ]:
```
train["hour"] = train["hour"].astype("category")
p13 = p9.ggplot(train, p9.aes("hour", "count")) + p9.geom_boxplot(colour="#1F3552", fill="#4271AE") +\
    p9.theme(figure_size=(10, 5), axis_text_x=p9.element_text(angle=90, hjust=1))
#qcut the temperature into 4 bins
train["temp_bin"] = pd.qcut(train["temp"], 3)
train["temp_bin"] = train["temp_bin"].astype("category")
train["temp_bin"] = train["temp_bin"].cat.rename_categories(["cold", "normal", "warm"])
p13 = p13 + p9.facet_grid(".~temp_bin")
#add labels to the facets
p13 = p13 + p9.labs(x="hour", y="count", title="Boxplot of Count vs Hour, Faceted by Temp")
p13
```



```
Out[ ]: <ggplot: (355942632)>
```

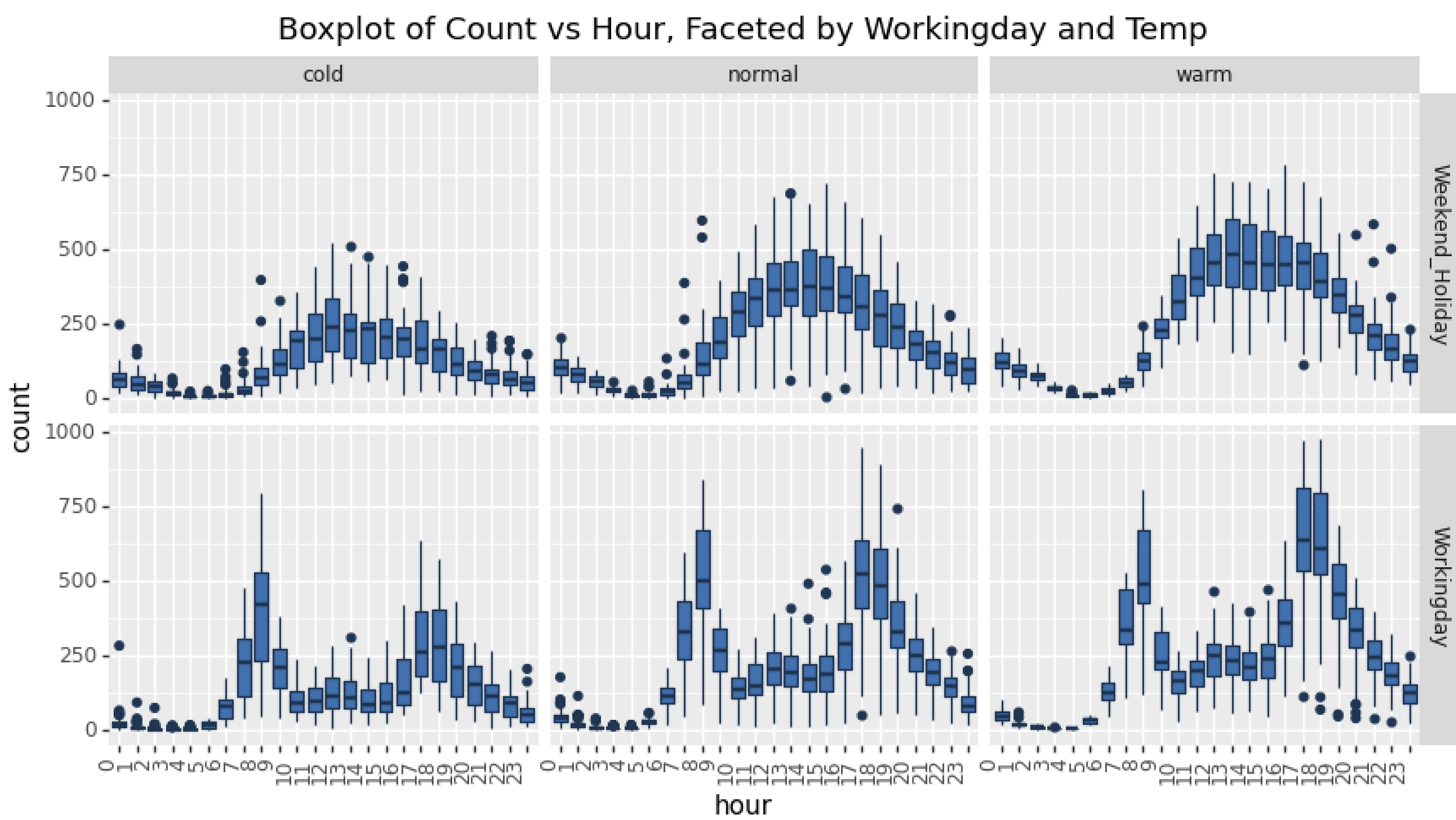
- We see a clear trend that warmer temperatures have more rentals.
- Furthermore, we see that on colder days the number of rentals in the evening is lower than the number of rentals in the morning.

```
In [ ]: train["hour"] = train["hour"].astype("category")
p11 = p9.ggplot(train, p9.aes("hour", "count")) + p9.geom_boxplot(colour="#1F3552", fill="#4271AE") + \
    p9.theme(figure_size=(10, 5), axis_text_x=p9.element_text(angle=90, hjust=1))
train["workingday"] = train["workingday"].astype("category")
#change the 0's to "holiday" and 1's to "workingday"
train["workingday"] = train["workingday"].map({0: "Weekend_Holiday", 1: "Workingday"})
p11 = p11 + p9.facet_grid(".~workingday")
#add labels to the facets
p11 = p11 + p9.labs(x="hour", y="count", title="Boxplot of Count vs Hour, Faceted by Workingday")
p11
```




```
Out[ ]: <ggplot: (695573065)>
```

```
In [ ]: p14 = p9.ggplot(train, p9.aes("hour", "count")) + p9.geom_boxplot(colour="#1F3552", fill="#4271AE") +\
    p9.theme(figure_size=(10, 5), axis_text_x=p9.element_text(angle=90, hjust=1))
p14 = p14 + p9.facet_grid("workingday~temp_bin")
p14 = p14 + p9.labs(x="hour", y="count", title="Boxplot of Count vs Hour, Faceted by Workingday and Temp")
p14
```




```
Out[ ]: <ggplot: (355819813)>
```

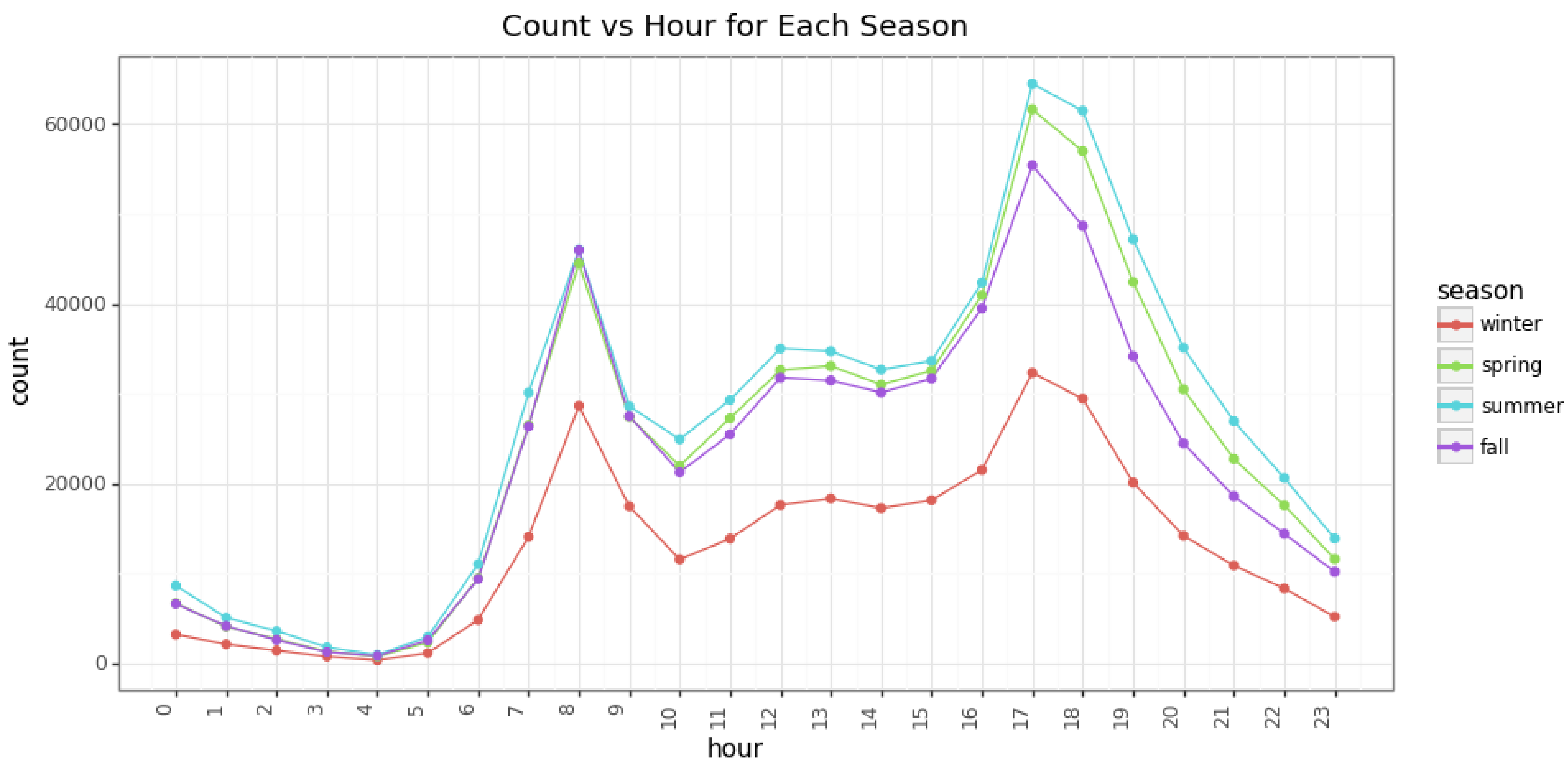
- We obsere that the Working Day hours show a higher count (in terms of mean and sum) during the morning and evening office to-fro hours.
- The Non-Working Day hours show a higher count during the afternoon hours. The temperature does not seem to have a significant effect on the count.
- Further, we note that the non-working days strecth to later hours than the working days. This could be due to the fact that the non-working days are weekends and people tend to stay out later on weekends.

d. Does the relationship between count and hour change by season?

In [ ]:
```
train['hour'] = train['hour'].astype('int')
train['workingday'] = train['workingday'].map({'Workingday': 1, 'Weekend_Holiday': 0})
train["workingday"] = train["workingday"].astype("int")
```

In [ ]:
```
#create a new dataframe with sum of count for each hour by season
hour_season_count = train.groupby(['hour', 'season']).sum()['count']
hour_season_count = pd.DataFrame(hour_season_count)
hour_season_count.reset_index(inplace=True)
```

In [ ]:
```
#plot a line chart the count vs hour for each season and use a dotted line
p15 = p9.ggplot(hour_season_count, p9.aes(x='hour', y='count', color='season')) + p9.geom_line() +\
    p9.theme_bw() + p9.theme(figure_size=(10, 5), axis_text_x=p9.element_text(angle=90, hjust=1)) + \
        p9.scale_x_continuous(breaks=range(0, 24, 1)) + p9.labs(title='Count vs Hour for Each Season') + p9.geom_point
p15
```



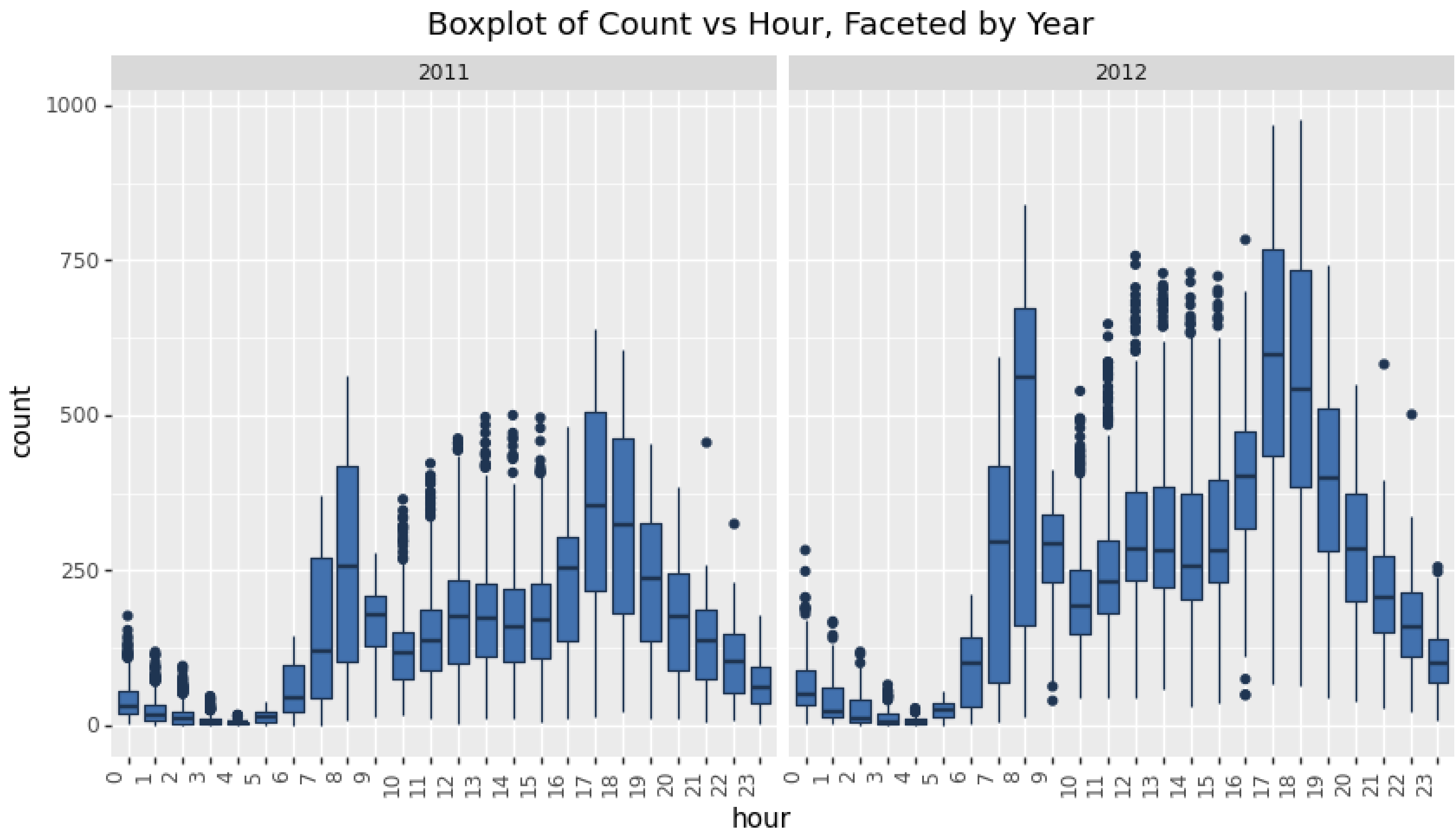
Out[ ]: <ggplot: (379900561)>

- Yes, there is a change in the relationship between count and hour by season.
- As expected, the Spring and Summer seasons have a higher count than the Fall and Winter seasons.
- The Summer season has the highest count across all hours. The Winter season has the lowest count across all hours.

In [ ]:
```
#revert the season back to numeric
train["season"] = train["season"].map({'winter': 1, 'spring': 2, 'summer': 3, 'fall': 4})
train["season"] = train["season"].astype("int")
```

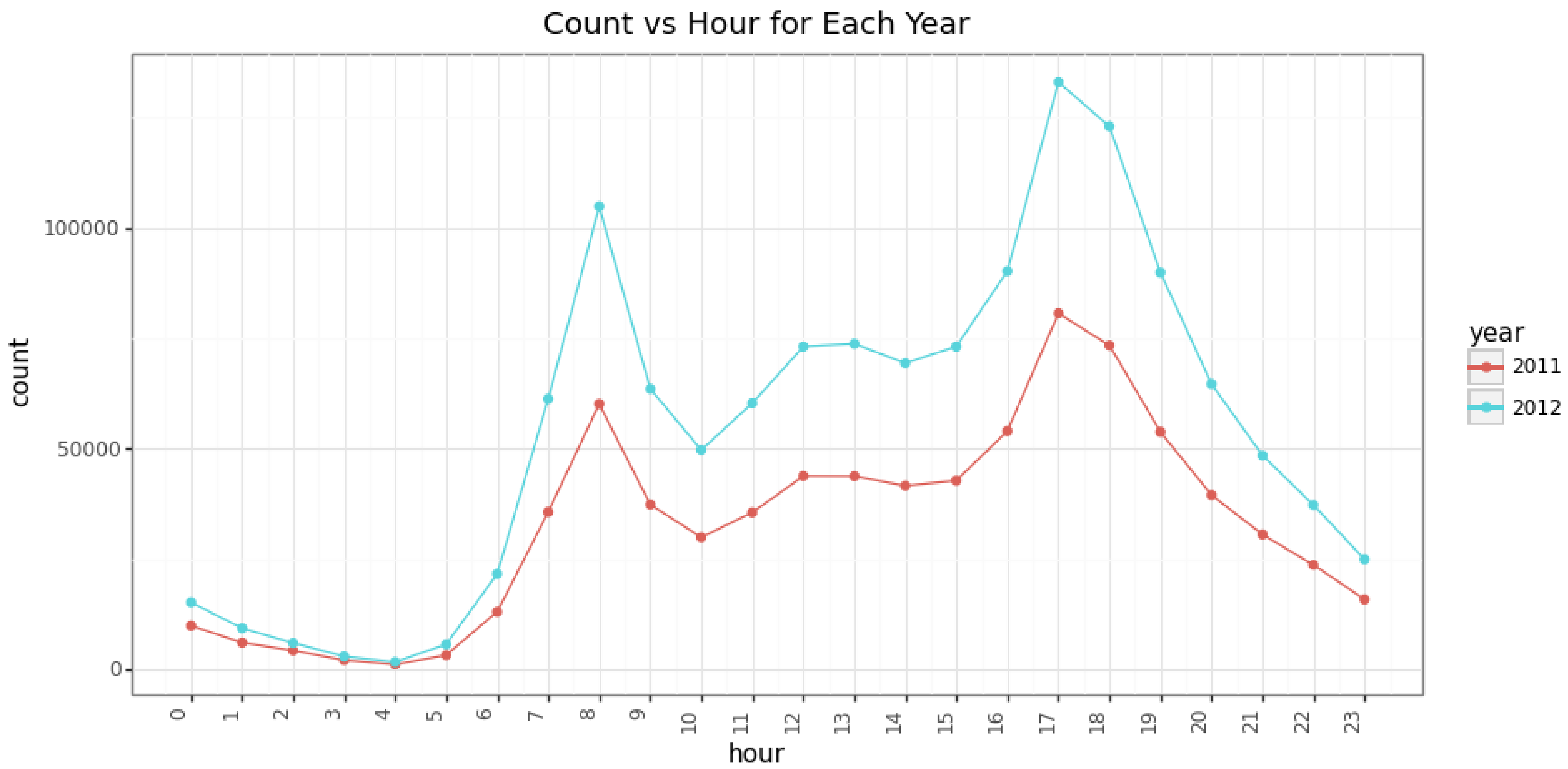
e. Does the distribution of hourly number of rentals change between 2011 and 2012? What does this tell you about the rental business?

In [ ]:
```
train["hour"] = train["hour"].astype("category")
train["year"] = train["year"].astype(str)
p16 = p9.ggplot(train, p9.aes("hour", "count")) + p9.geom_boxplot(colour="#1F3552", fill="#4271AE") +\
    p9.theme(figure_size=(10, 5), axis_text_x=p9.element_text(angle=90, hjust=1))
p16 = p16 + p9.facet_grid(".~year")
#add labels to the facets
p16 = p16 + p9.labs(x="hour", y="count", title="Boxplot of Count vs Hour, Faceted by Year")
p16
```



Out[ ]: <ggplot: (357497070)>

In [ ]:
```
train['hour'] = train['hour'].astype('int')
#create a new dataframe with sum of count for each hour by year
hour_year_count = train.groupby(['hour', 'year']).sum()['count']
hour_year_count = pd.DataFrame(hour_year_count)
hour_year_count.reset_index(inplace=True)
#plot count vs hour for each year
p17 = p9.ggplot(hour_year_count, p9.aes(x='hour', y='count', color='year')) + p9.geom_line() + \
    p9.theme_bw() + p9.theme(figure_size=(10, 5), axis_text_x=p9.element_text(angle=90, hjust=1)) +\
        p9.scale_x_continuous(breaks=range(0, 24, 1)) + p9.labs(title='Count vs Hour for Each Year') + p9.geom_point()
p17
```



Out[ ]: <ggplot: (379450465)>

- It is observed that the distribution of hourly number of rentals is similar between 2011 and 2012.
- The magnitude of the distribution is higher in 2012 than in 2011.
- This tells us that the rental business has grown over the year.

In [ ]:
```
train["year"] = train["year"].astype("int")
train.drop("temp_bin", axis=1, inplace=True)
```

2. Build a model to predict the bikeshare counts for the hours recorded in the test dataset.

Save your predictions to a .csv file that you will submit to Kaggle (see Kaggle instruction below.) Provide a write-up that explains how you went about building your model. Attach the code to create the submission .csv file as an appendix to your homework submission.\ A sample script for generating a valid .csv file for submission is given below. This code builds a simple linear model for predicting log(count) using all the available variables. You will easily do better than this.

In [ ]:
```
# Let's read the data
train = pd.read_csv('Bike_train.csv')
test = pd.read_csv('Bike_test.csv')
```

In [ ]:
```
# Create a table with the columns of training data and their basic statistics
train.describe()
```

Out[ ]:

	daylabel	year	month	day	hour	season	holiday	workingday	weather	temp	atemp
count	10932.0000	10932.0000	10932.0000	10932.0000	10932.0000	10932.0000	10932.0000	10932.0000	10932.0000	10932.0000	10932.000
mean	359.7914	2011.5005	6.5060	9.9912	11.5066	2.5016	0.0285	0.6817	1.4195	20.1908	23.610
std	210.8701	0.5000	3.4493	5.4741	6.9225	1.1176	0.1665	0.4659	0.6345	7.8075	8.492
min	1.0000	2011.0000	1.0000	1.0000	0.0000	1.0000	0.0000	0.0000	1.0000	0.8200	0.760
25%	182.0000	2011.0000	4.0000	5.0000	6.0000	2.0000	0.0000	0.0000	1.0000	13.9400	16.665
50%	366.0000	2012.0000	7.0000	10.0000	12.0000	3.0000	0.0000	1.0000	1.0000	20.5000	24.240
75%	548.0000	2012.0000	10.0000	15.0000	18.0000	4.0000	0.0000	1.0000	2.0000	26.2400	31.060
max	719.0000	2012.0000	12.0000	19.0000	23.0000	4.0000	1.0000	1.0000	4.0000	41.0000	45.455

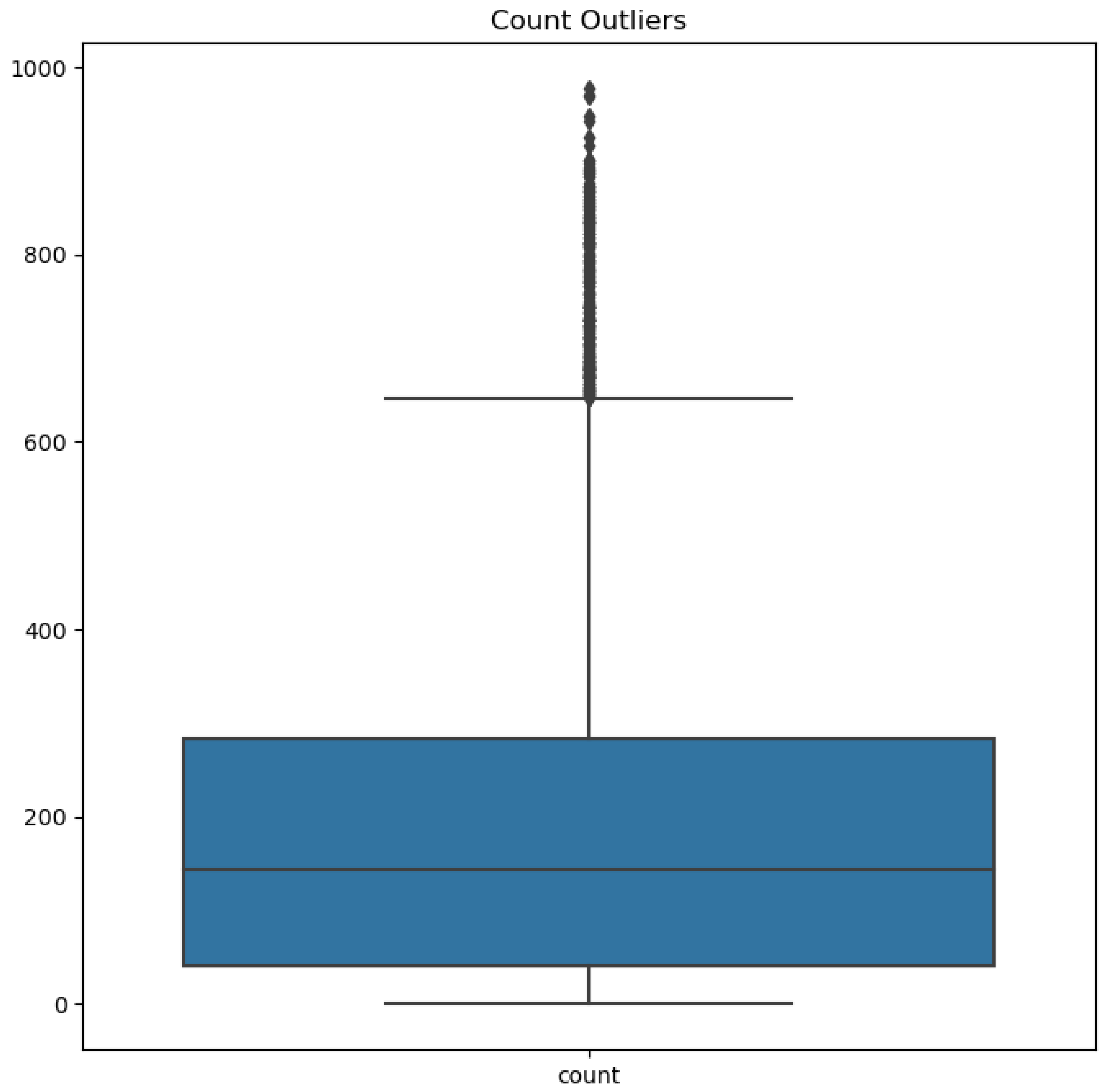
Since the descriptive analysis of the dataframe was done in the previous question, we will skip that part and go straight to the model building.

First, we need to analyze the values of our dataframe to see if there are any missing values. We see that there are no missing values in the dataframe and outliers that have to be treated.

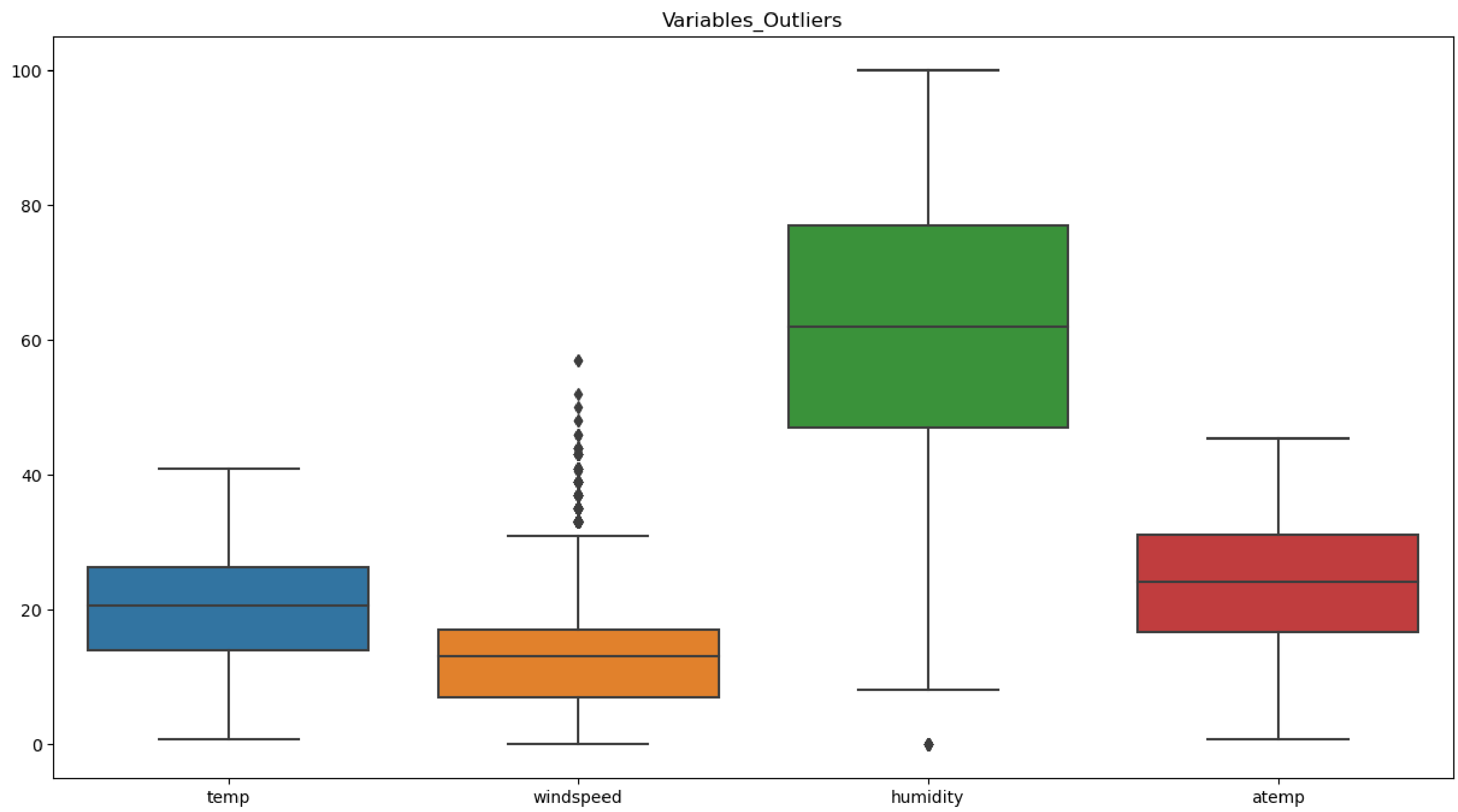
In [ ]:
```
# Validate if there are any missing values in dataset
train.isnull().sum()
```

Out[ ]:
```
daylabel      0
year          0
month         0
day           0
hour          0
season        0
holiday       0
workingday    0
weather       0
temp          0
atemp         0
humidity      0
windspeed     0
count         0
dtype: int64
```

In [ ]:
```
fig,ax=plt.subplots(figsize=(8,8))
#Box plot for Temp_windspeed_humidity_outliers
sn.boxplot(data=train[['count']])
ax.set_title('Count Outliers')
plt.show()
```



In [ ]:
```
fig,ax=plt.subplots(figsize=(15,8))
#Box plot for Temp_windspeed_humidity_outliers
sn.boxplot(data=train[['temp','windspeed','humidity','atemp']])
ax.set_title('Variables_Outliers')
plt.show()
```



- It is observed that the count variable has some outliers as windspeed and humidity. We will treat these outliers by replacing them with a KNN imputer.

In [ ]:
```
def outlier_detection(df, column):
    Q1 = df[column].quantile(0.25)
    Q3 = df[column].quantile(0.75)
    IQR = Q3 - Q1
    lower_bound = Q1 - (1.5 * IQR)
    upper_bound = Q3 + (1.5 * IQR)
```

```
    # Make nan values for outliers
    df.loc[(df[column] < lower_bound) | (df[column] > upper_bound), column] = np.nan
    return df
```

In [ ]:
```
train = outlier_detection(train, 'windspeed')
train = outlier_detection(train, 'humidity')
```

In [ ]:
```
# Let's check the missing values were created
train.isnull().sum()
```

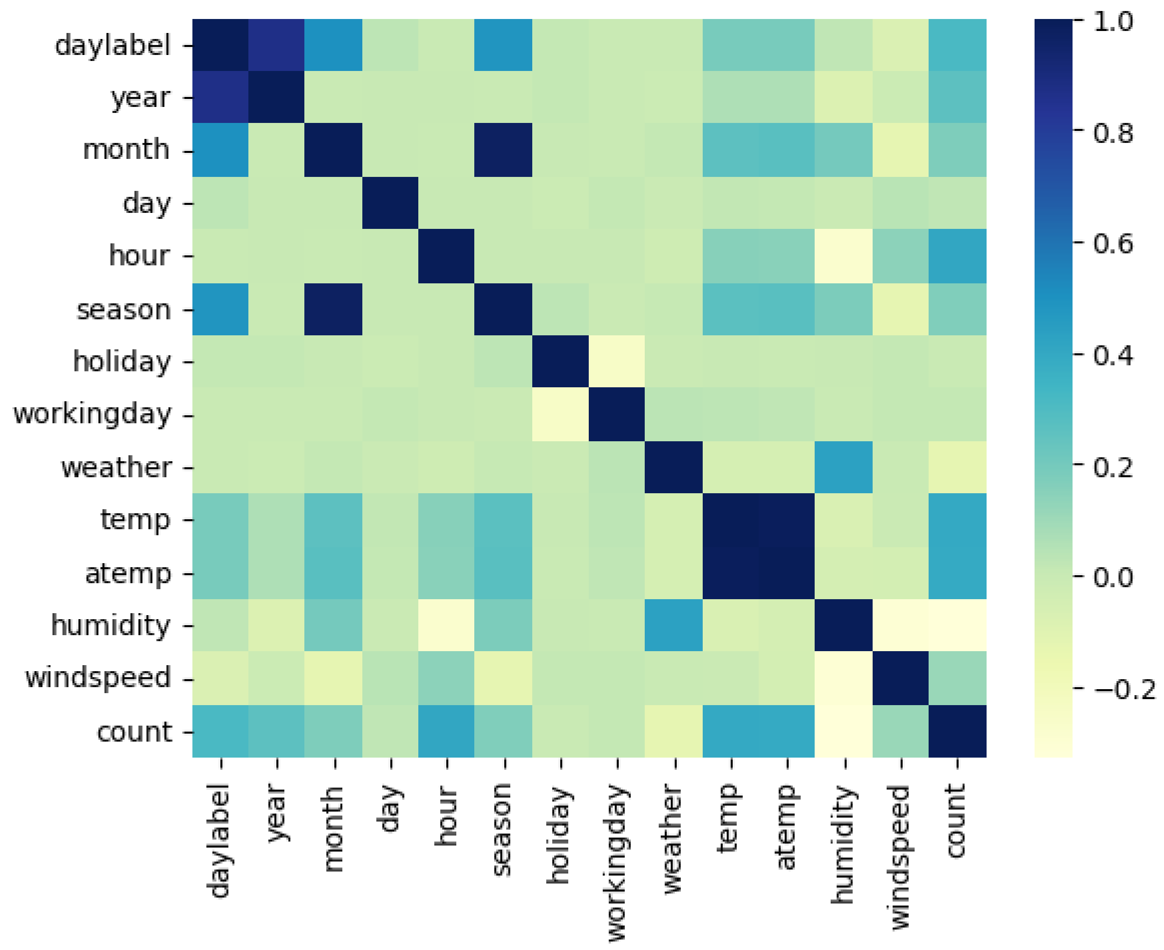
Out[ ]:
```
daylabel        0
year            0
month           0
day             0
hour            0
season          0
holiday         0
workingday      0
weather         0
temp            0
atemp           0
humidity       24
windspeed     228
count           0
dtype: int64
```

In [ ]:
```
from sklearn.impute import KNNImputer

def knnimpute(df):
    knn_imputer = KNNImputer(n_neighbors=5,weights='uniform')
    array_imputed = knn_imputer.fit_transform(df)
    df_imputed = pd.DataFrame(array_imputed, index = df.index, columns = df.columns)
    return df_imputed
```

In [ ]:
```
train = knnimpute(train)
# Let's check the missing values were created
train.isnull().sum()
```

Out[ ]:
```
daylabel      0
year          0
month         0
day           0
hour          0
season        0
holiday       0
workingday    0
weather       0
temp          0
atemp         0
humidity      0
windspeed     0
count         0
dtype: int64
```

In [ ]:
```
# plotting the heatmap
t2 = train.corr()
hm = sn.heatmap(data = t2 , annot=False, cmap="YlGnBu")

# displaying the plotted heatmap
plt.show()
```



- First we used the KNN imputer to replace the outliers in the windspeed and humidity variables.
- These were then input into our XGBoost and Random Forest models to get in-sample RMSE figures.

In [ ]:

```
import xgboost as xgb
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import KFold

# Define the parameter grid
param_grid = {'max_depth': [5],
              'n_estimators': [700],
              'learning_rate': [0.3],
              }

# Split the data into training and test sets
X_train, X_test, y_train, y_test = train_test_split(train.drop(columns=['count']), train['count'], test_size=0.2, rand


# Create the XGBoost model
xgb_model = xgb.XGBRegressor(objective ='reg:squarederror', colsample_bytree = 0.3, alpha = 10)

# Define the KFold cross-validation
kf = KFold(n_splits=5)

# Use GridSearchCV with KFold cross-validation to find the best parameters
grid_search = GridSearchCV(xgb_model, param_grid, cv=kf, verbose=2)
grid_search.fit(train.drop(columns=['count']), train['count'])
```

```
Fitting 5 folds for each of 1 candidates, totalling 5 fits
[CV] END ...learning_rate=0.3, max_depth=5, n_estimators=700; total time=   1.2s
[CV] END ...learning_rate=0.3, max_depth=5, n_estimators=700; total time=   1.2s
[CV] END ...learning_rate=0.3, max_depth=5, n_estimators=700; total time=   1.3s
[CV] END ...learning_rate=0.3, max_depth=5, n_estimators=700; total time=   1.1s
[CV] END ...learning_rate=0.3, max_depth=5, n_estimators=700; total time=   1.1s
```

Out[ ]:

```
▸ GridSearchCV
▸ estimator: XGBRegressor
    ▸ XGBRegressor
```

In [ ]:

```
# Print the best parameters
print("Best parameters: ", grid_search.best_params_)

# Predict the 'count' using the test data
y_pred = grid_search.predict(X_test)

# Evaluate the model using mean squared error
from sklearn.metrics import mean_squared_error
rmse = mean_squared_error(y_test, y_pred, squared=False)
print("Root Mean Squared Error: ", rmse)
```

```
Best parameters:  {'learning_rate': 0.3, 'max_depth': 5, 'n_estimators': 700}
Root Mean Squared Error:  24.36303821834758
```

In [ ]:

```
y_pred = grid_search.predict(test)
#make the negative values to 0
y_pred[y_pred < 0] = 0
```

In [ ]:

```
y_pred_df = pd.DataFrame(y_pred)
#rename the columns to Id and count
y_pred_df.columns = ['count']
y_pred_df.index.name = 'Id'
#Add 1 to the index
y_pred_df.index = y_pred_df.index + 1
y_pred_df.to_csv("Bike_test_predictions_xgboost.csv")
```

In [ ]:

```
from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import KFold

# Define the parameter grid
param_grid = {'n_estimators': [260],
              'max_depth': [15],
              'min_samples_split': [2],
              'min_samples_leaf': [2]
              }

# Create the RandomForestRegressor model
rf_model = RandomForestRegressor(random_state=42)

# Define the KFold cross-validation
kf = KFold(n_splits=5)

# Use GridSearchCV with KFold cross-validation to find the best parameters
grid_search = GridSearchCV(rf_model, param_grid, cv=kf, verbose=2)
grid_search.fit(train.drop(columns=['count']), train['count'])
```

```
Fitting 5 folds for each of 1 candidates, totalling 5 fits
[CV] END max_depth=15, min_samples_leaf=2, min_samples_split=2, n_estimators=260; total time=   4.8s
[CV] END max_depth=15, min_samples_leaf=2, min_samples_split=2, n_estimators=260; total time=   4.6s
[CV] END max_depth=15, min_samples_leaf=2, min_samples_split=2, n_estimators=260; total time=   4.5s
[CV] END max_depth=15, min_samples_leaf=2, min_samples_split=2, n_estimators=260; total time=   4.6s
[CV] END max_depth=15, min_samples_leaf=2, min_samples_split=2, n_estimators=260; total time=   4.6s
```

Out[ ]:

```
▸ GridSearchCV
▸ estimator: RandomForestRegressor
    ▸ RandomForestRegressor
```

In [ ]:

```python
# Print the best parameters
print("Best parameters: ", grid_search.best_params_)

# Predict the 'count' using the test data
y_pred = grid_search.predict(X_test)

# Evaluate the model using mean squared error
from sklearn.metrics import mean_squared_error
rmse = mean_squared_error(y_test, y_pred, squared=False)
print("Root Mean Squared Error: ", rmse)
```

```
Best parameters:  {'max_depth': 15, 'min_samples_leaf': 2, 'min_samples_split': 2, 'n_estimators': 260}
Root Mean Squared Error:  23.358920254811483
```

In [ ]:

```python
#use the model to predict the test data
bike_test = pd.read_csv("Bike_test.csv")

y_pred = grid_search.predict(bike_test)

#make the negative values to 0
y_pred[y_pred < 0] = 0
```

In [ ]:

```python
y_pred_df = pd.DataFrame(y_pred)
#rename the columns to Id and count
y_pred_df.columns = ['count']
y_pred_df.index.name = 'Id'
#Add 1 to the index
y_pred_df.index = y_pred_df.index + 1
y_pred_df.to_csv("Bike_test_predictions_random_forest.csv")
```

Observations

- We see that the XGBoost model has a lower in-sample RMSE than the Random Forest model.
- The current hyper parameters were selected after rigoros grid search and cross validation. for both the models.
- Further, we will run the same models without any changes or update to the input variables to see if the model performance improves.

In [ ]:

```python
train = pd.read_csv("Bike_train.csv")
test = pd.read_csv("Bike_test.csv")
```

In [ ]:

```python
import xgboost as xgb
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import KFold

# Define the parameter grid
param_grid = {'max_depth': [5],
              'n_estimators': [700],
              'learning_rate': [0.3],
              }

# Split the data into training and test sets
X_train, X_test, y_train, y_test = train_test_split(train.drop(columns=['count']), train['count'], test_size=0.2, rand


# Create the XGBoost model
xgb_model = xgb.XGBRegressor(objective ='reg:squarederror', colsample_bytree = 0.3, alpha = 10)

# Define the KFold cross-validation
kf = KFold(n_splits=5)

# Use GridSearchCV with KFold cross-validation to find the best parameters
grid_search = GridSearchCV(xgb_model, param_grid, cv=kf, verbose=2)
grid_search.fit(train.drop(columns=['count']), train['count'])
```

```
Fitting 5 folds for each of 1 candidates, totalling 5 fits
[CV] END ...learning_rate=0.3, max_depth=5, n_estimators=700; total time=   1.1s
[CV] END ...learning_rate=0.3, max_depth=5, n_estimators=700; total time=   1.1s
[CV] END ...learning_rate=0.3, max_depth=5, n_estimators=700; total time=   1.1s
[CV] END ...learning_rate=0.3, max_depth=5, n_estimators=700; total time=   1.1s
[CV] END ...learning_rate=0.3, max_depth=5, n_estimators=700; total time=   1.1s
```

Out[ ]:

```
▸ GridSearchCV
▸ estimator: XGBRegressor
    ▸ XGBRegressor
```

In [ ]:
```
# Print the best parameters
print("Best parameters: ", grid_search.best_params_)

# Predict the 'count' using the test data
y_pred = grid_search.predict(X_test)

# Evaluate the model using mean squared error
from sklearn.metrics import mean_squared_error
rmse = mean_squared_error(y_test, y_pred, squared=False)
print("Root Mean Squared Error: ", rmse)
```

```
Best parameters:  {'learning_rate': 0.3, 'max_depth': 5, 'n_estimators': 700}
Root Mean Squared Error:  23.927441578453593
```

In [ ]:
```
y_pred = grid_search.predict(test)
#make the negative values to 0
y_pred[y_pred < 0] = 0
```

CODE TO GENERATE SUBMISSION FILE

In [ ]:
```
y_pred_df = pd.DataFrame(y_pred)
#rename the columns to Id and count
y_pred_df.columns = ['count']
y_pred_df.index.name = 'Id'
#Add 1 to the index
y_pred_df.index = y_pred_df.index + 1
y_pred_df.to_csv("Bike_test_predictions_xgboost_no.csv")
```

In [ ]:
```
from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import KFold

# Define the parameter grid
param_grid = {'n_estimators': [260],
              'max_depth': [15],
              'min_samples_split': [2],
              'min_samples_leaf': [2]
              }

# Create the RandomForestRegressor model
rf_model = RandomForestRegressor(random_state=42)

# Define the KFold cross-validation
kf = KFold(n_splits=5)

# Use GridSearchCV with KFold cross-validation to find the best parameters
grid_search = GridSearchCV(rf_model, param_grid, cv=kf, verbose=2)
grid_search.fit(train.drop(columns=['count']), train['count'])
```

```
Fitting 5 folds for each of 1 candidates, totalling 5 fits
[CV] END max_depth=15, min_samples_leaf=2, min_samples_split=2, n_estimators=260; total time=   4.7s
[CV] END max_depth=15, min_samples_leaf=2, min_samples_split=2, n_estimators=260; total time=   4.5s
[CV] END max_depth=15, min_samples_leaf=2, min_samples_split=2, n_estimators=260; total time=   4.5s
[CV] END max_depth=15, min_samples_leaf=2, min_samples_split=2, n_estimators=260; total time=   4.6s
[CV] END max_depth=15, min_samples_leaf=2, min_samples_split=2, n_estimators=260; total time=   4.6s
```

Out[ ]:
```
▸ GridSearchCV
▸ estimator: RandomForestRegressor
  ▸ RandomForestRegressor
```

In [ ]:
```
# Print the best parameters
print("Best parameters: ", grid_search.best_params_)

# Predict the 'count' using the test data
y_pred = grid_search.predict(X_test)

# Evaluate the model using mean squared error
from sklearn.metrics import mean_squared_error
rmse = mean_squared_error(y_test, y_pred, squared=False)
print("Root Mean Squared Error: ", rmse)
```

```
Best parameters:  {'max_depth': 15, 'min_samples_leaf': 2, 'min_samples_split': 2, 'n_estimators': 260}
Root Mean Squared Error:  23.37276215446016
```

In [ ]:
```
#use the model to predict the test data
bike_test = pd.read_csv("Bike_test.csv")

y_pred = grid_search.predict(bike_test)

#make the negative values to 0
y_pred[y_pred < 0] = 0
```

CODE TO GENERATE SUBMISSION FILE

In [ ]:
```
y_pred_df = pd.DataFrame(y_pred)
#rename the columns to Id and count
y_pred_df.columns = ['count']
```

```
y_pred_df.index.name = 'Id'
#Add 1 to the index
y_pred_df.index = y_pred_df.index + 1
y_pred_df.to_csv("Bike_test_predictions_random_forest_no.csv")
```

- We see the the models with no changes to the input variables have a lower in-sample RMSE than the models with the outliers replaced.
- The Kaggle submissions were done using these hyperparamters and the output was submitted to Kaggle.
- We observe that the RandomForest model has a lower in-sample RMSE than the XGBoost model.
- We also use a Cross Validation technique to see if the model performance improves, along with an iterative grid search to find the best hyperparameters for the models.
- Further, it would be interesting to see the model performance if we use different outliers.