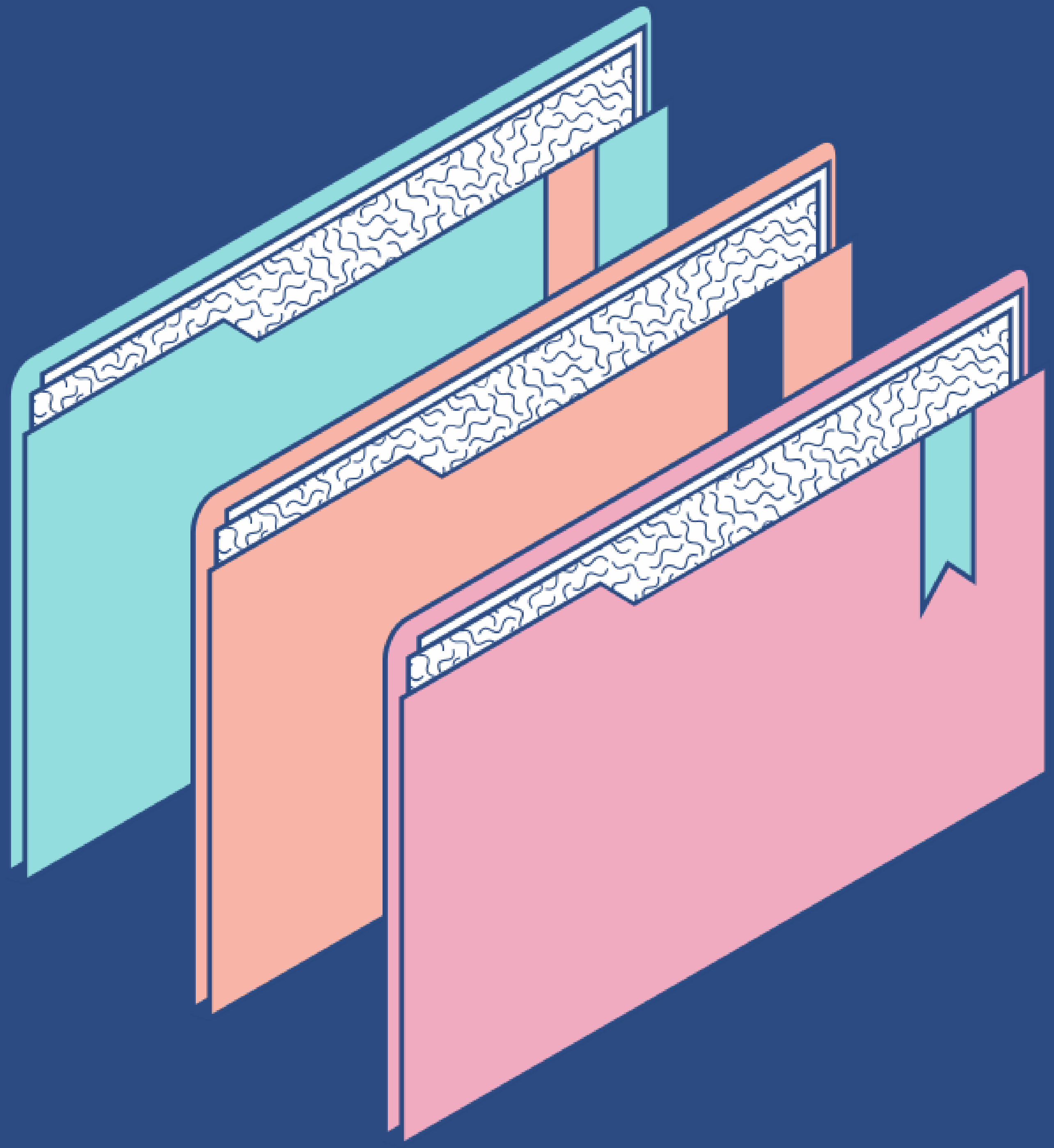
SC1015 Mini Project: Bike Sharing

by Jonathan Chow, Jacob Rossman, Khant Zaw



Practical Motivation

Improve user experience: Data science can help bike sharing companies to improve the user experience for riders. By analyzing user behavior, bike sharing companies can identify pain points in the user journey and make improvements to the service.

Questions

Are we able to forecast the use of a bikeshare system?
Which model would fit our problem the best?

Steps we took in this project

1 — 2 — 3 — 4 — 5

STEP 1

Importing the dataset and studying its features

STEP 2

Cleaning the dataset. Removing outliers, normalising distribution, etc

STEP 3

Exploratory analysis

STEP 4

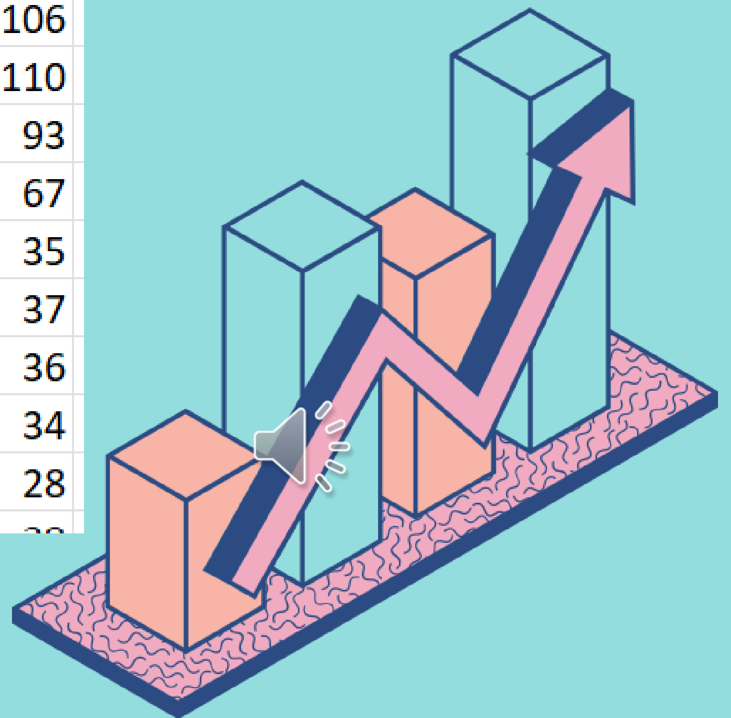
Model Fitting

STEP 5

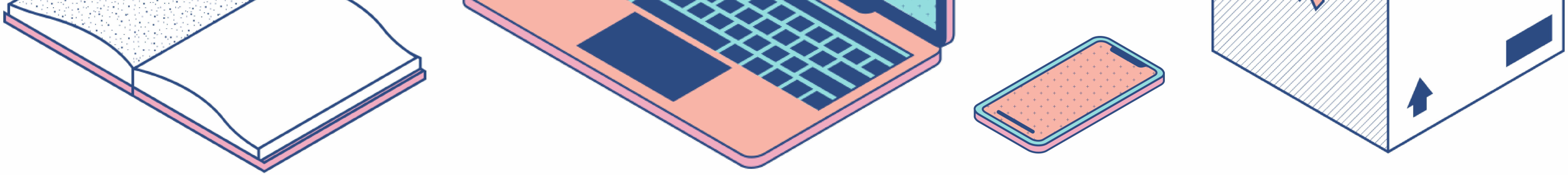
Conclusion.

Our Dataset:
Bike Sharing Demand from Kaggle

datetime	season	holiday	workingda	weather	temp	atemp	humidity	windspee	casual	registered	count
1/1/2011 0:00	1	0	0	1	9.84	14.395	81	0	3	13	16
1/1/2011 1:00	1	0	0	1	9.02	13.635	80	0	8	32	40
1/1/2011 2:00	1	0	0	1	9.02	13.635	80	0	5	27	32
1/1/2011 3:00	1	0	0	1	9.84	14.395	75	0	3	10	13
1/1/2011 4:00	1	0	0	1	9.84	14.395	75	0	0	1	1
1/1/2011 5:00	1	0	0	2	9.84	12.88	75	6.0032	0	1	1
1/1/2011 6:00	1	0	0	1	9.02	13.635	80	0	2	0	2
1/1/2011 7:00	1	0	0	1	8.2	12.88	86	0	1	2	3
1/1/2011 8:00	1	0	0	1	9.84	14.395	75	0	1	7	8
1/1/2011 9:00	1	0	0	1	13.12	17.425	76	0	8	6	14
1/1/2011 10:00	1	0	0	1	15.58	19.695	76	16.9979	12	24	36
1/1/2011 11:00	1	0	0	1	14.76	16.665	81	19.0012	26	30	56
1/1/2011 12:00	1	0	0	1	17.22	21.21	77	19.0012	29	55	84
1/1/2011 13:00	1	0	0	2	18.86	22.725	72	19.9995	47	47	94
1/1/2011 14:00	1	0	0	2	18.86	22.725	72	19.0012	35	71	106
1/1/2011 15:00	1	0	0	2	18.04	21.97	77	19.9995	40	70	110
1/1/2011 16:00	1	0	0	2	17.22	21.21	82	19.9995	41	52	93
1/1/2011 17:00	1	0	0	2	18.04	21.97	82	19.0012	15	52	67
1/1/2011 18:00	1	0	0	3	17.22	21.21	88	16.9979	9	26	35
1/1/2011 19:00	1	0	0	3	17.22	21.21	88	16.9979	6	31	37
1/1/2011 20:00	1	0	0	2	16.4	20.455	87	16.9979	11	25	36
1/1/2011 21:00	1	0	0	2	16.4	20.455	87	12.998	3	31	34
1/1/2011 22:00	1	0	0	2	16.4	20.455	94	15.0013	11	17	28



Variables	Description	Variable	Description
Season	1: spring 2: summer 3: fall 4: winter	Weather	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
Holiday	Whether the day is considered a holiday	Windspeed	wind speed
Workingday	whether the day is neither a weekend nor holiday	Casual	number of non-registered user rentals initiated
Humidity	relative humidity	Registered	number of registered user rentals initiated
Temp	temperature in Celsius	Count	number of total rentals



Formatting the data into correct data types

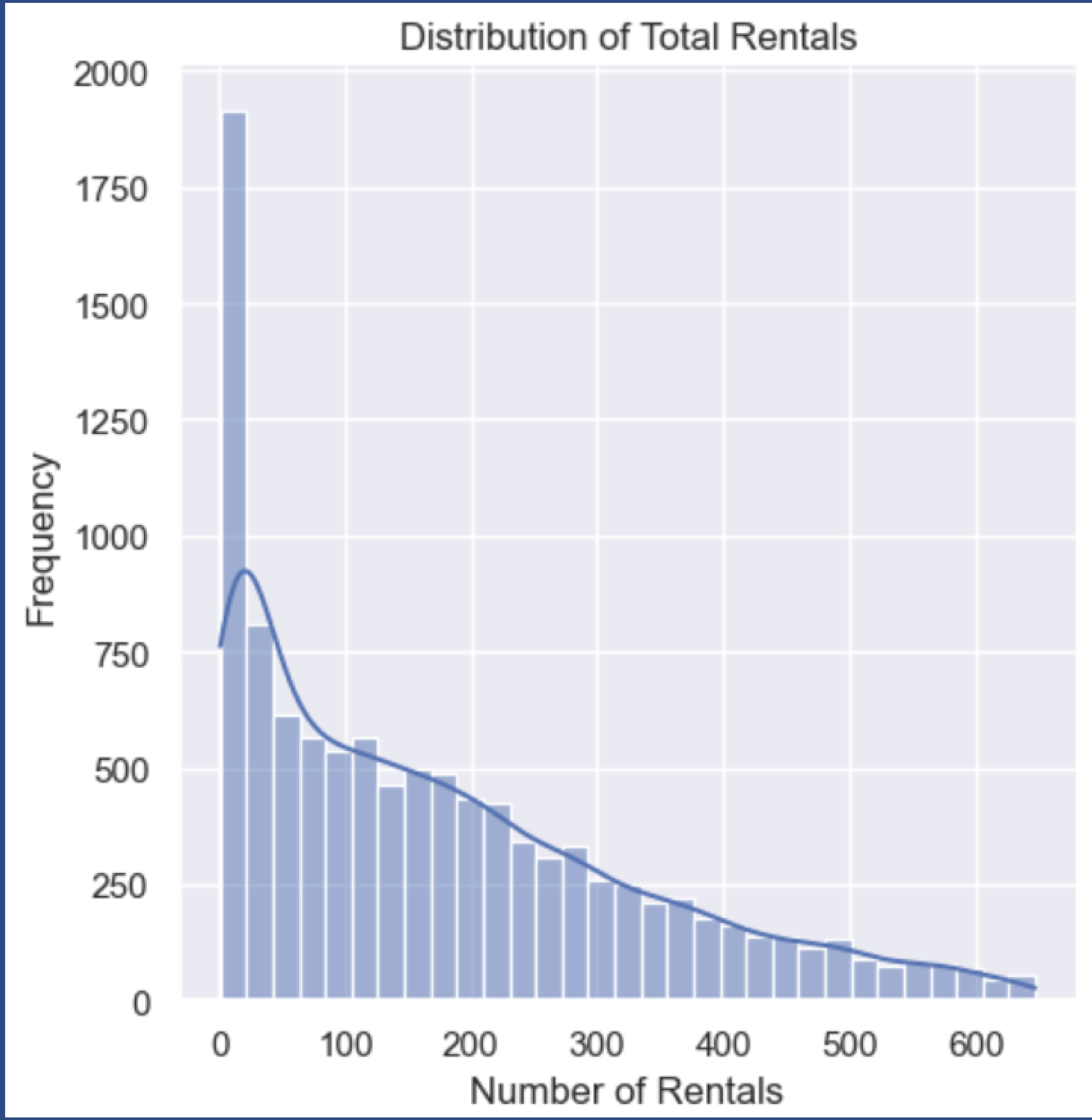
```
Data columns (total 12 columns):
 #   Column  Non-Null Count  Dtype 
---  ------  --------------  ----- 
 0   0       10887 non-null  object
 1   1       10887 non-null  object
 2   2       10887 non-null  object
 3   3       10887 non-null  object
 4   4       10887 non-null  object
 5   5       10887 non-null  object
 6   6       10887 non-null  object
 7   7       10887 non-null  object
 8   8       10887 non-null  object
 9   9       10887 non-null  object
 10  10      10887 non-null  object
 11  11      10887 non-null  object
dtypes: object(12)
```

```
Data columns (total 15 columns):
 #   Column      Non-Null Count  Dtype   
---  ------      --------------  -----   
 0   season      10586 non-null  int64   
 1   holiday     10586 non-null  int64   
 2   workingday  10586 non-null  int64   
 3   weather     10586 non-null  int64   
 4   temp        10586 non-null  float64 
 5   atemp       10586 non-null  float64 
 6   humidity    10586 non-null  int64   
 7   windspeed   10586 non-null  float64 
 8   casual      10586 non-null  int64   
 9   registered  10586 non-null  int64   
 10  count       10586 non-null  int64   
 11  year        10586 non-null  category
 12  month       10586 non-null  category
 13  day         10586 non-null  category
 14  hour        10586 non-null  category
dtypes: category(4), float64(3), int64(8)
```

Extra points on our cleaning

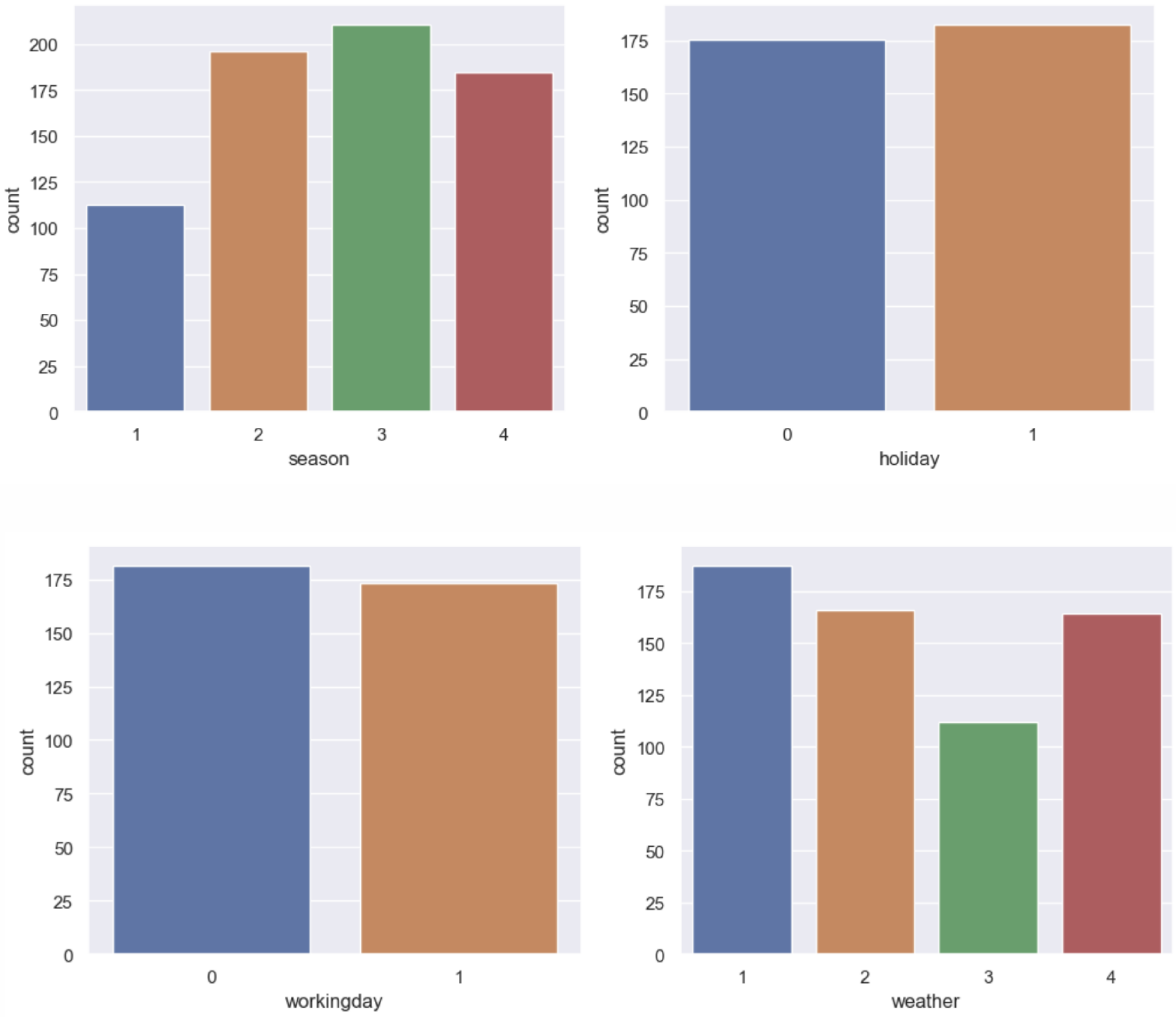
- Usage of Point Biserial Correlation Coefficient (PBCC) and Phi Coefficient
- Also checked for NULL values and outliers and removed them

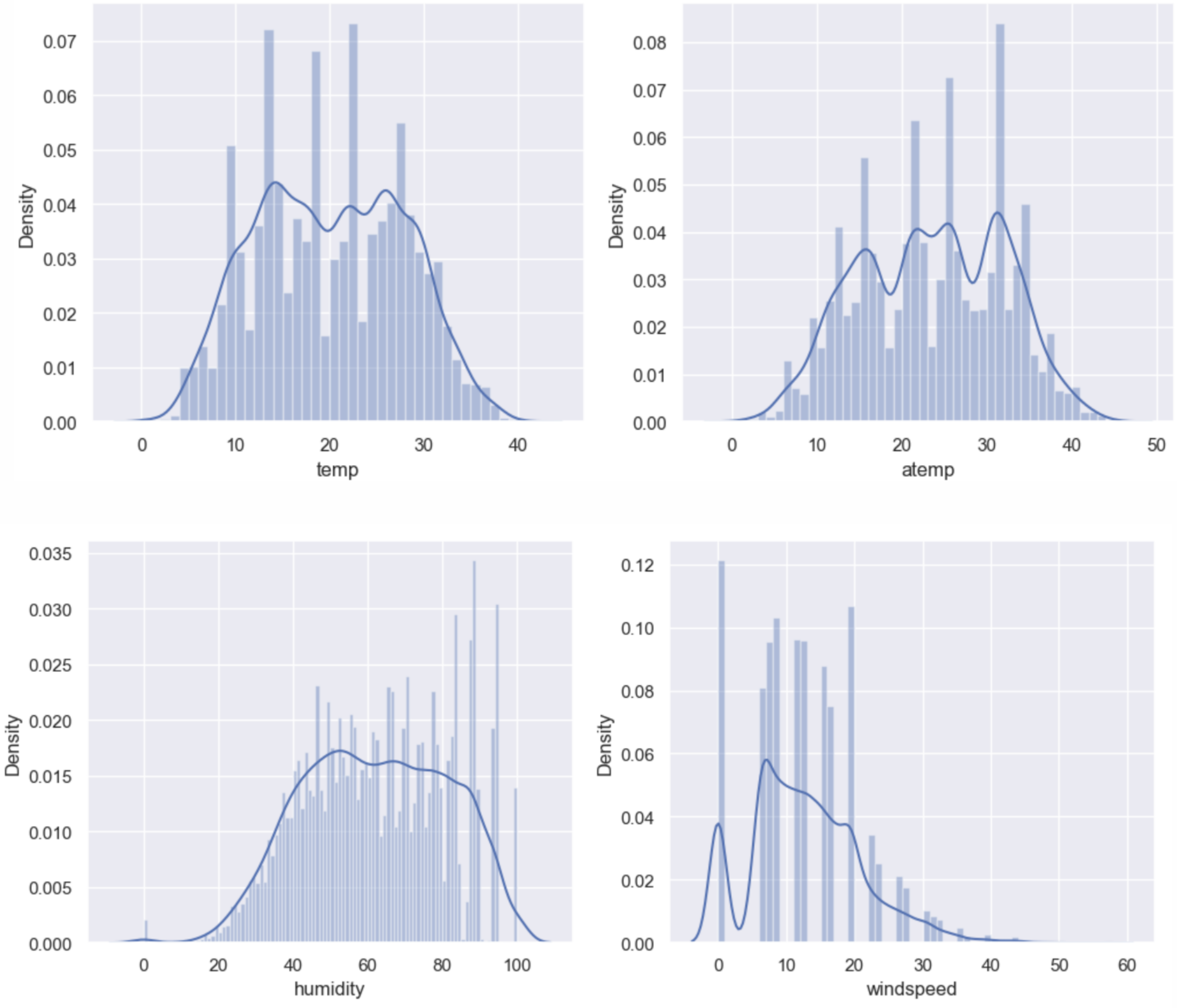
```
Data columns (total 15 columns):
 #   Column      Non-Null Count  Dtype   
---  ------      --------------  -----   
 0   season      10586 non-null  int64   
 1   holiday     10586 non-null  int64   
 2   workingday  10586 non-null  int64   
 3   weather     10586 non-null  int64   
 4   temp        10586 non-null  float64 
 5   atemp       10586 non-null  float64 
 6   humidity    10586 non-null  int64   
 7   windspeed   10586 non-null  float64 
 8   casual      10586 non-null  int64   
 9   registered  10586 non-null  int64   
 10  count       10586 non-null  int64   
 11  year        10586 non-null  category
 12  month       10586 non-null  category
 13  day         10586 non-null  category
 14  hour        10586 non-null  category
dtypes: category(4), float64(3), int64(8)
```



Data Analysis of response variable

- Positively skewed distribution
- Suggest a long tail to the right
- May need to be addressed



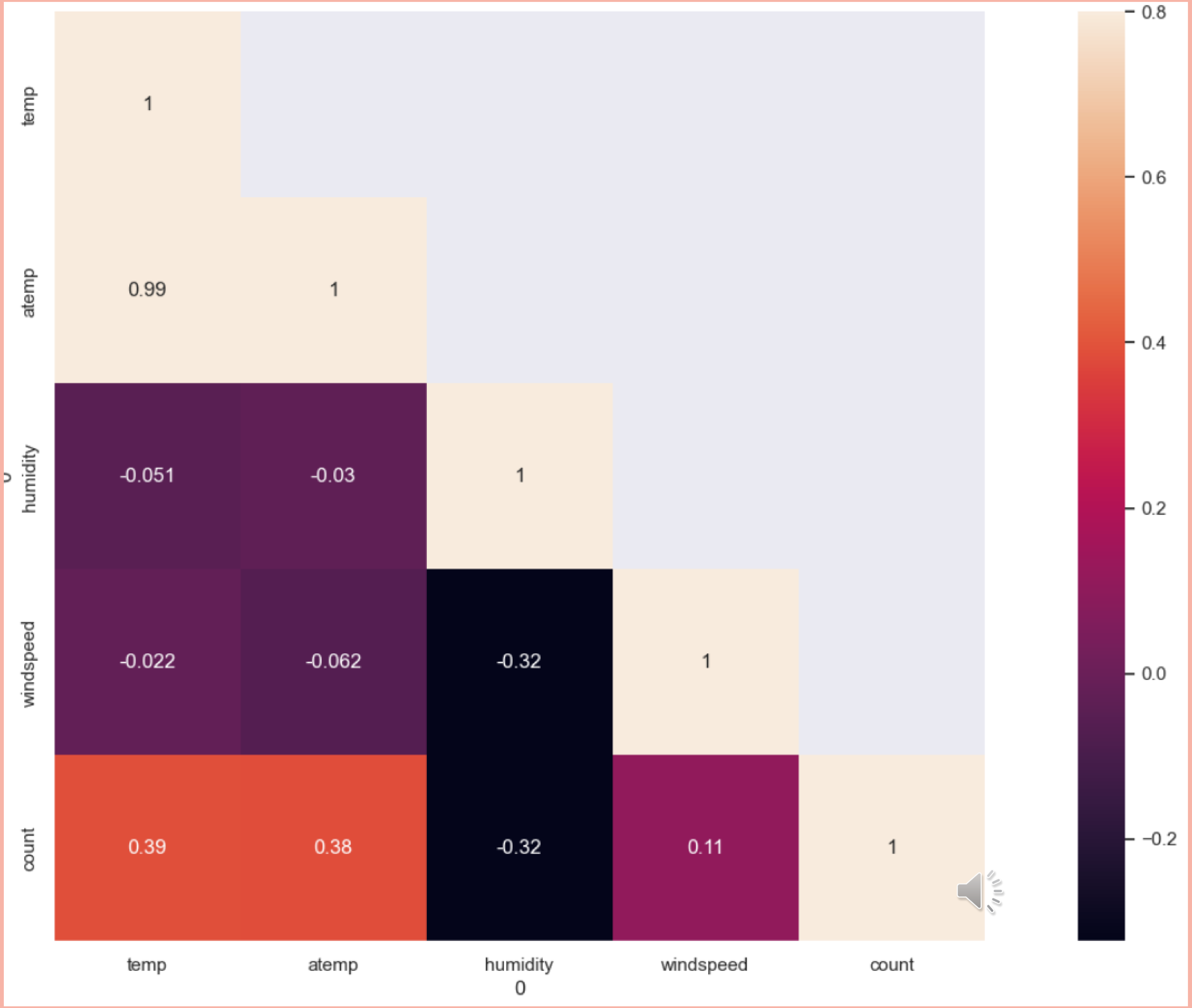


Categorical Variables

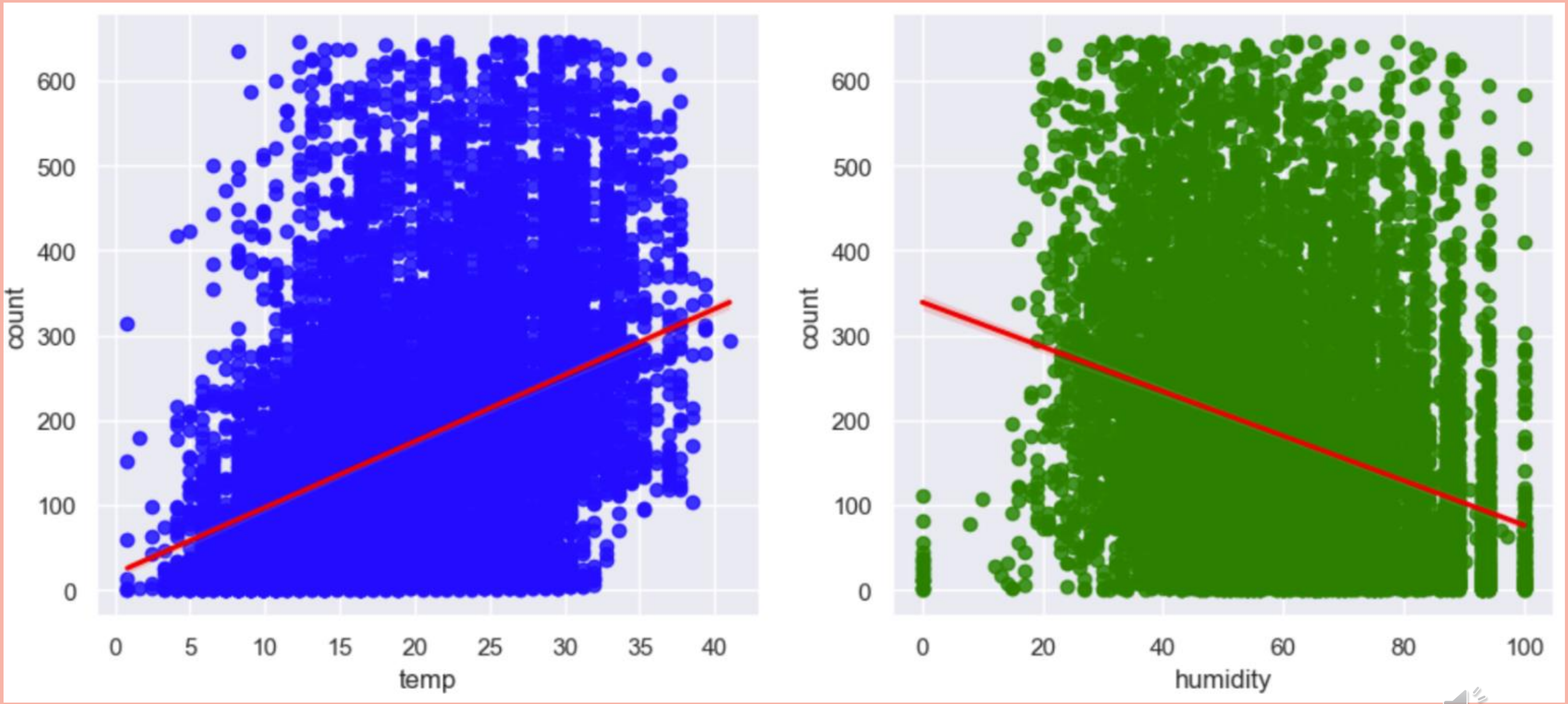
Numerical Variables

Numerical Variable Analysis

- Count has some dependency on temp and humidity
- Omit the relationship of atemp and temp



Temp and Humidity

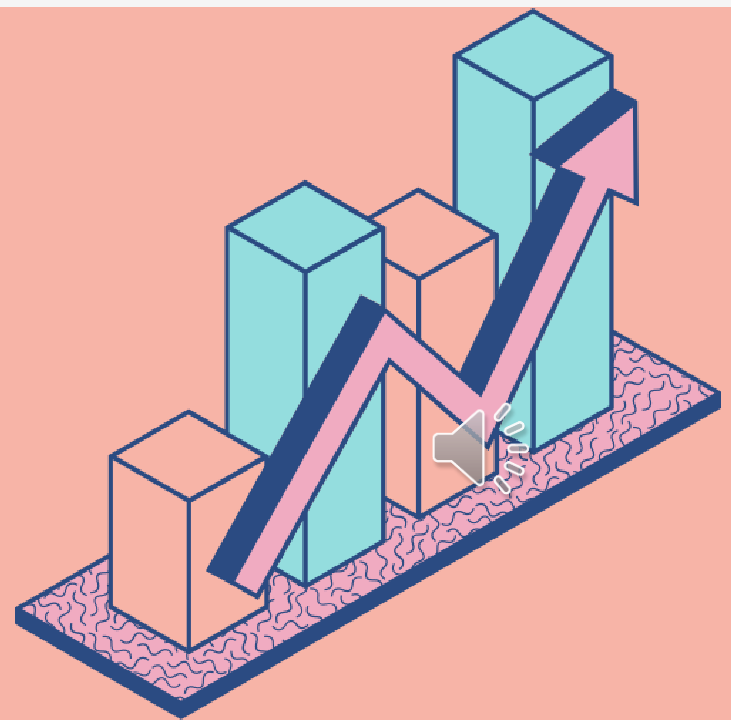


Categorical Variable Analysis

Categorical Variable	Boolean?	Appropriate Measure for Correlation
season	No	Phi Coefficient
holiday	Yes	Point Biserial Correlation Coefficient
workingday	Yes	Point Biserial Correlation Coefficient
weather	No	Phi Coefficient
year	No	Phi Coefficient
month	No	Phi Coefficient
day	No	Phi Coefficient
hour	No	Phi Coefficient

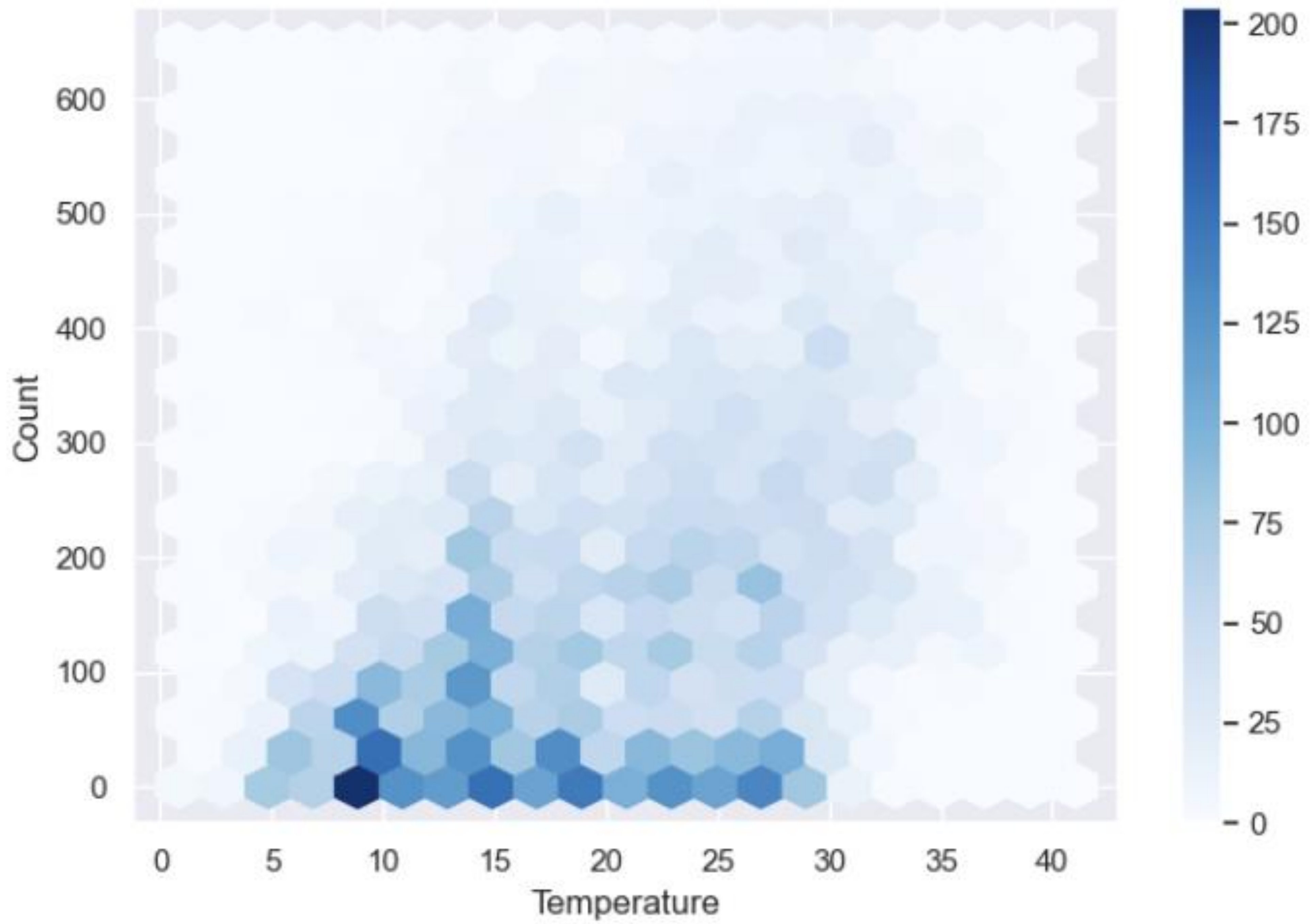
```python
# Define the categorical variables
cat_vars = ['season', 'holiday', 'workingday', 'weather', 'year', 'month', 'day', 'hour']

# Create an empty DataFrame to store the results
results_df = pd.DataFrame(columns=['Variable', 'Correlation', 'P-value'])

# Loop through each categorical variable and compute the correlation
for var in cat_vars:
    if len(df[var].unique()) == 2:
        # For binary variables, compute PBCC
        pbcc, p_value = stats.pointbiserialr(df[var], df['count'])
        results_df = pd.concat([results_df, pd.DataFrame({'Variable': [var], 'Correlation': [pbcc], 'P-value': [p_value]})], ignore_index=True)
    else:
        # For variables with more than two categories, compute Phi Coefficient
        cont_table = pd.crosstab(df[var], df['count'])
        phi_coef, p_value, dof, expected = stats.chi2_contingency(cont_table, correction=False)
        results_df = pd.concat([results_df, pd.DataFrame({'Variable': [var], 'Correlation': [phi_coef], 'P-value': [p_value]})], ignore_index=True)

# Print the results
print(results_df)
```

	Variable	Correlation	P-value
0	season	2964.599153	8.752985e-48
1	holiday	0.007621	4.330150e-01
2	workingday	-0.025021	1.004078e-02
3	weather	1720.455613	9.996342e-01
4	year	0.206398	3.311578e-102
5	month	8385.982619	1.379925e-26
6	day	11497.051305	6.046849e-01
7	hour	28580.237683	0.000000e+00



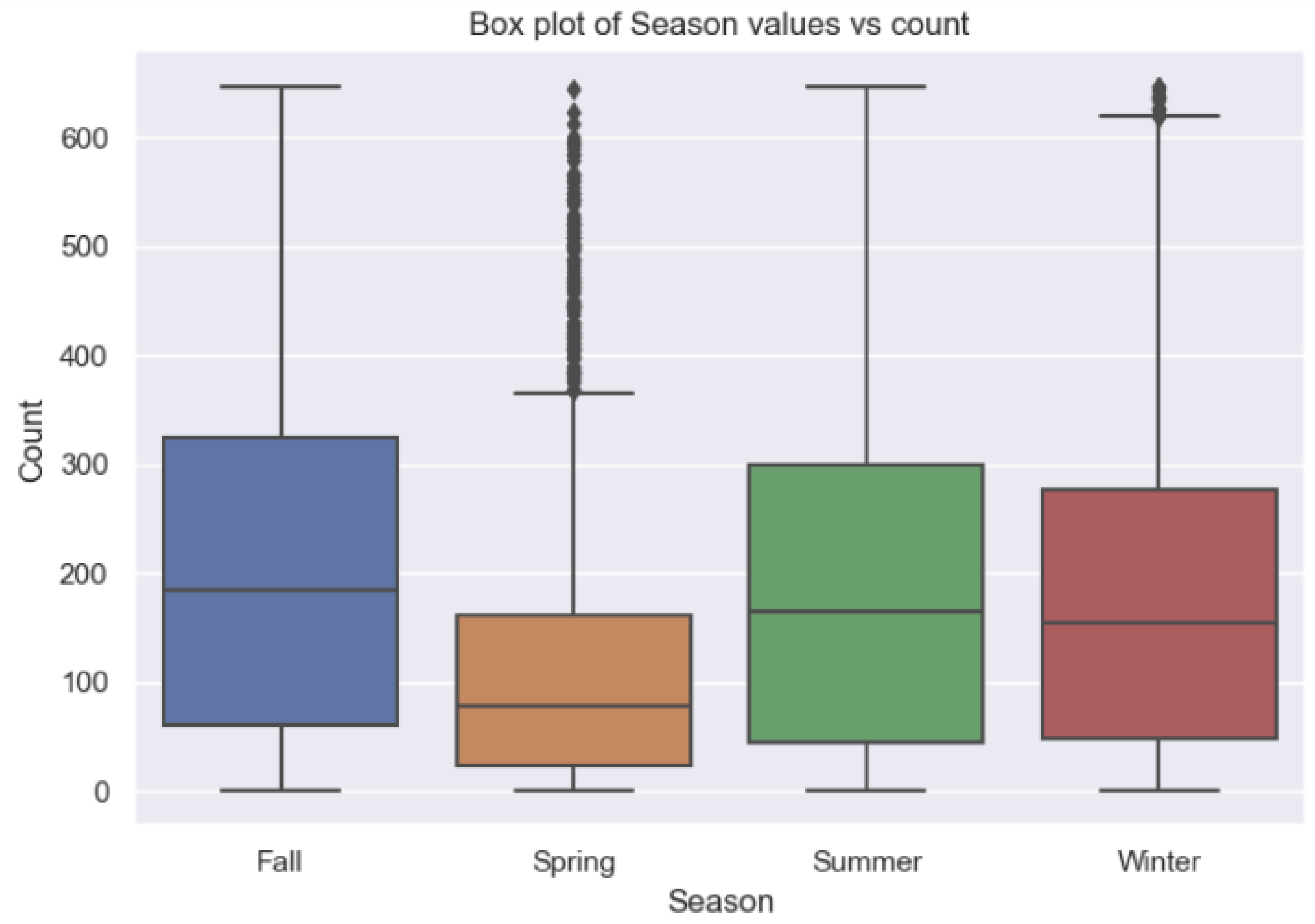
Hexbin Plot

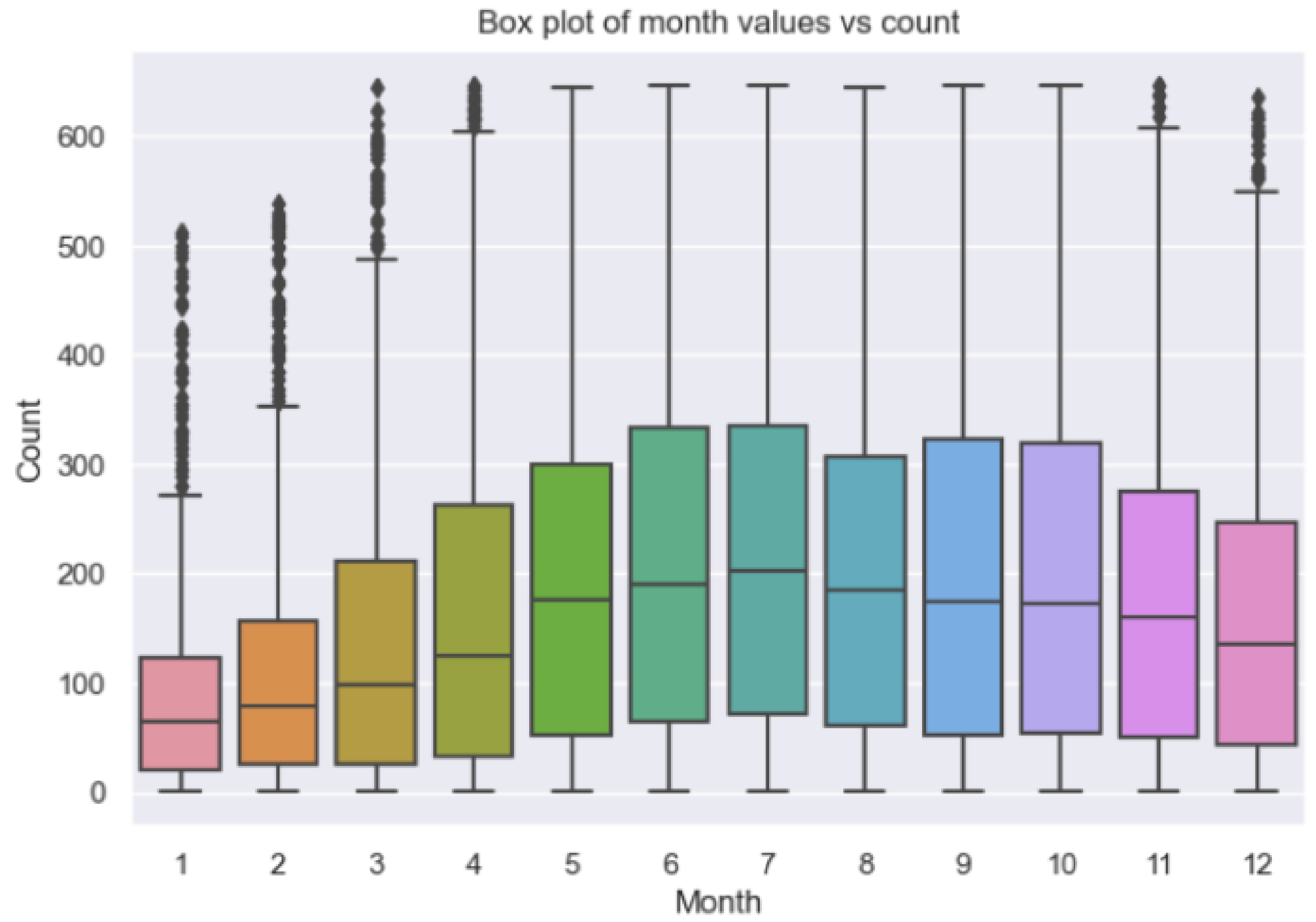
- Colour of density represented by the darkness of the hexes on the graph
- Dark colour suggest strong relationship in that area

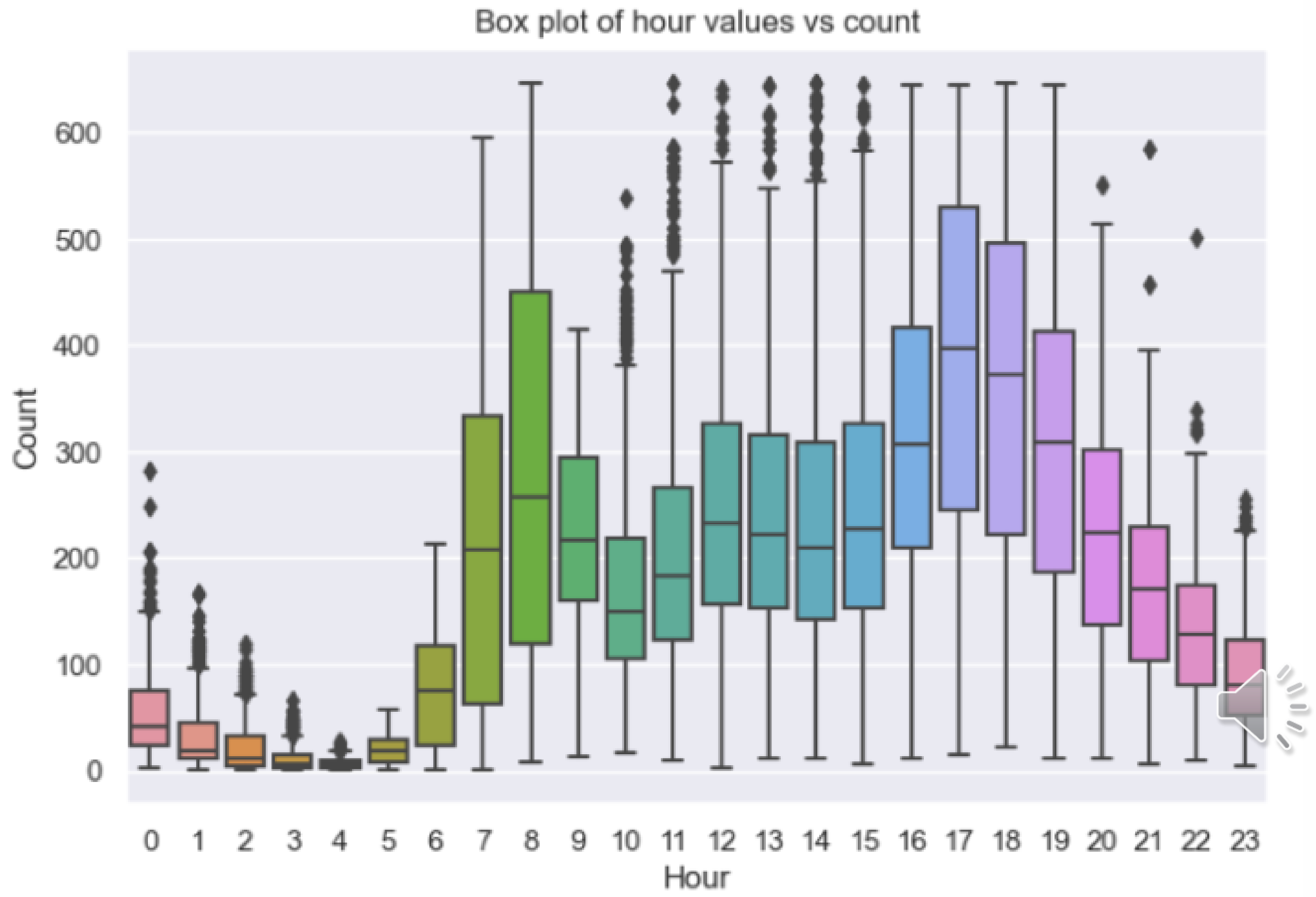


Box Plot (Season vs Count)

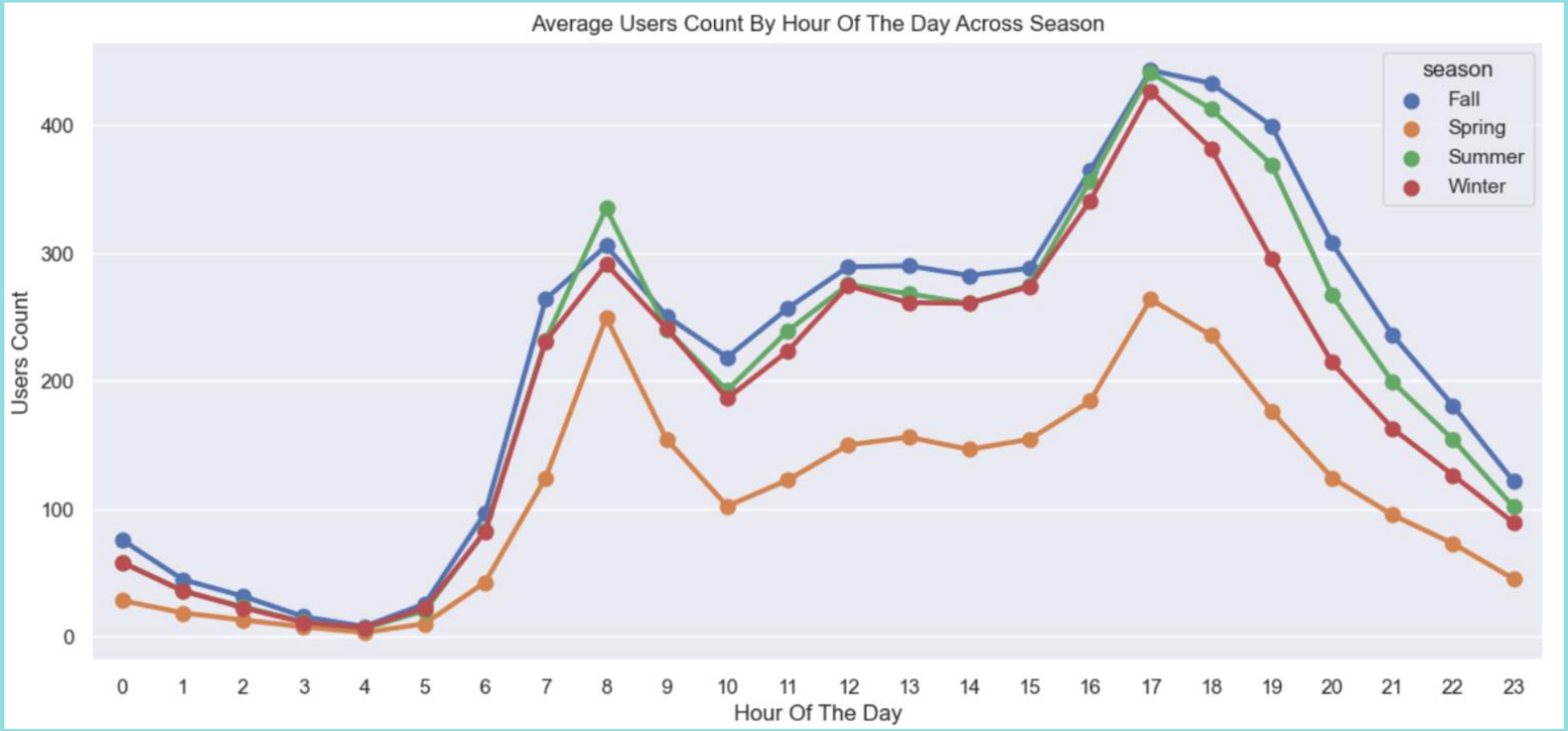
- Spring significantly lower
- Maybe time series will help







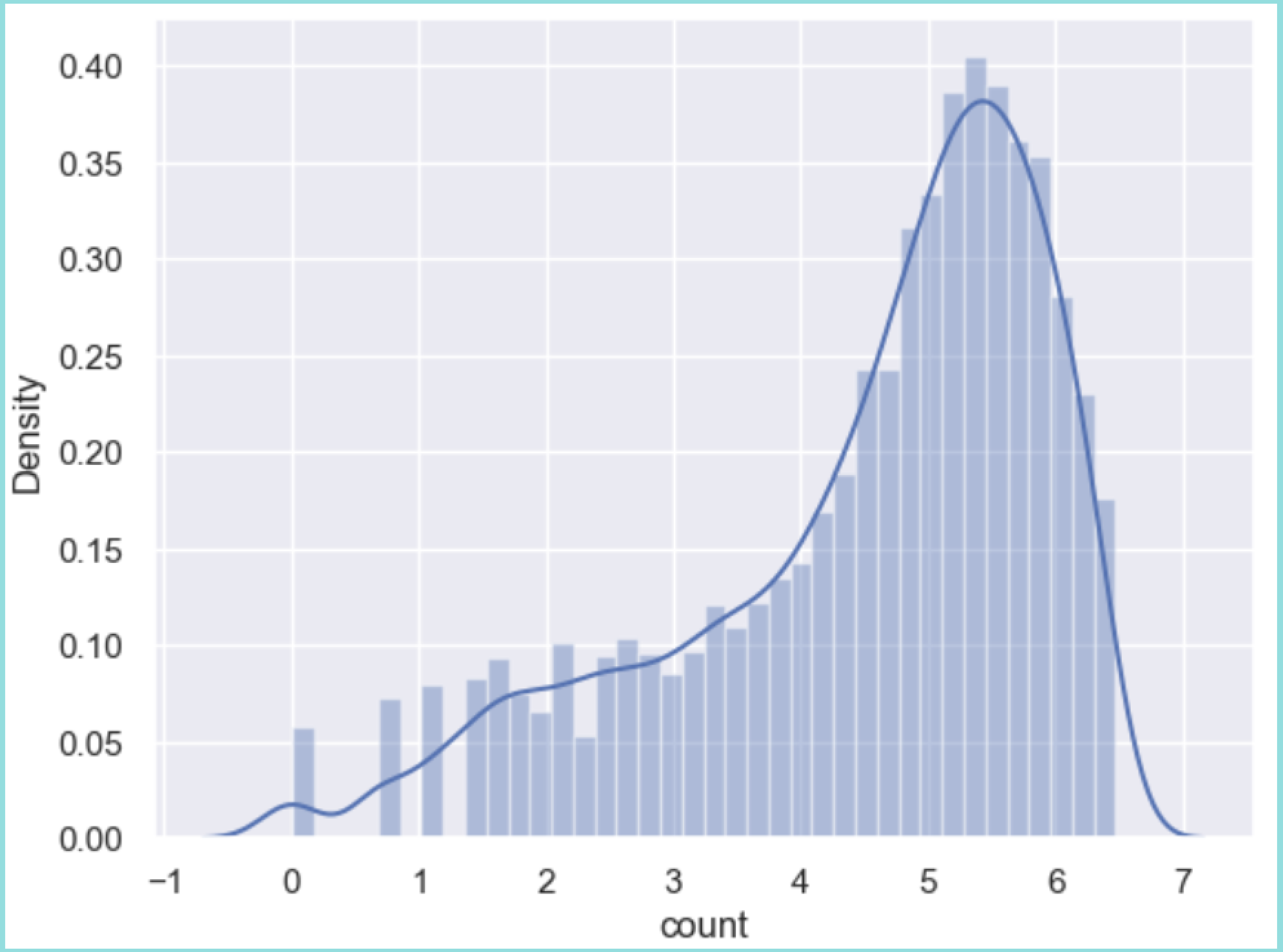
Timeseries



- Spring indeed lowest count
- Spike in count at 0800 and 1700 hours

Normalizing the target variable

- Target variable has highly skewed distribution
- Tested Log, Square-root, Cox-Box
- Log was best

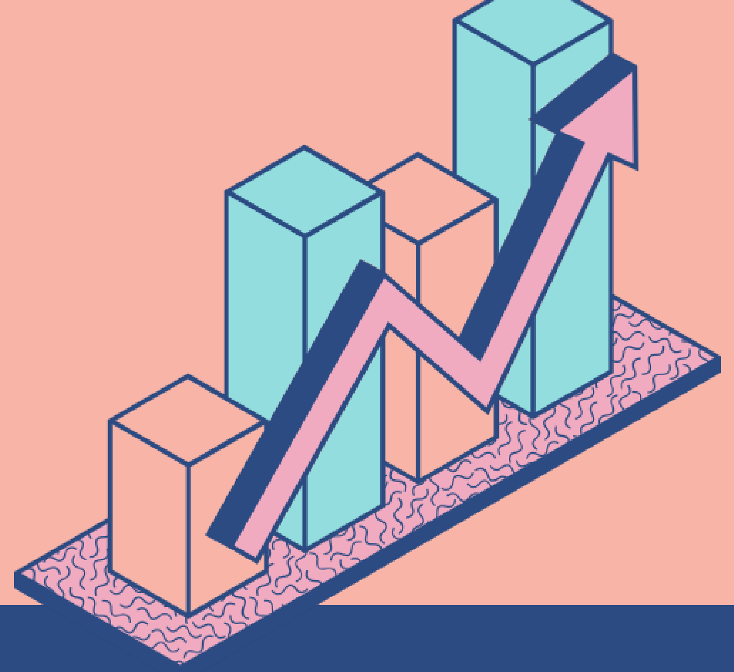


One Hot Encode

```
# Select columns from original dataframe
#selected_cols = ['hour', 'month', 'temp', 'season_Fall', 'season_Spring', 'season_Summer', 'season_Winter', 'count']
selected_cols = ['hour', 'month', 'temp', 'season', 'count']


# Create new dataframe with selected columns
df_selected = df[selected_cols].copy()

# One-hot encode the 'season' variable
season_dummies = pd.get_dummies(df_selected['season'], prefix='season')
df_selected = pd.concat([df_selected, season_dummies], axis=1)

# Drop the original 'season' variable
df_selected.drop('season', axis=1, inplace=True)

df_selected.head()
```

Model	Description
RandomForestRegressor	Fits ts a number of classifying decision trees on various sub-samples of the dataset and uses averaging to improve the predictive accuracy and control over-fitting.
AdaBoostRegressor	Begins by fitting a regressor on the original dataset and then fits additional copies of the regressor on the same dataset but where the weights of instances are adjusted according to the error of the current prediction. As such, subsequent regressors focus more on difficult cases.
BaggingRegressor	A Bagging regressor is an ensemble meta-estimator that fits base regressors each on random subsets of the original dataset and then aggregate their individual predictions (either by voting or by averaging) to form a final prediction.
SVR	Support Vector Regression is a supervised learning algorithm that is used to predict discrete values. Support Vector Regression uses the same principle as the SVMs. The basic idea behind SVR is to find the best fit line. In SVR, the best fit line is the hyperplane that has the maximum number of points
KNeighboursRegressor	Regression based on k-nearest neighbors.The target is predicted by local interpolation of the targets associated of the nearest neighbors in the training set.

Goodness of Fit of our Models



Model	RMSLE	MAE	RMSE	R^2
RandomForestRegressor	0.2099	0.5825	0.7992	0.7051
AdaBoostRegressor	0.2070	0.6427	0.8155	0.6930
BaggingRegressor	0.2125	0.5907	0.8072	0.6992
SVR	0.2519	0.6898	0.9325	0.5985
KNeighboursRegressor	0.2008	0.5618	0.7632	0.7311

- KNeighboursRegressor lowest RMSLE
- Best performing model

Hyperparameter Optimization

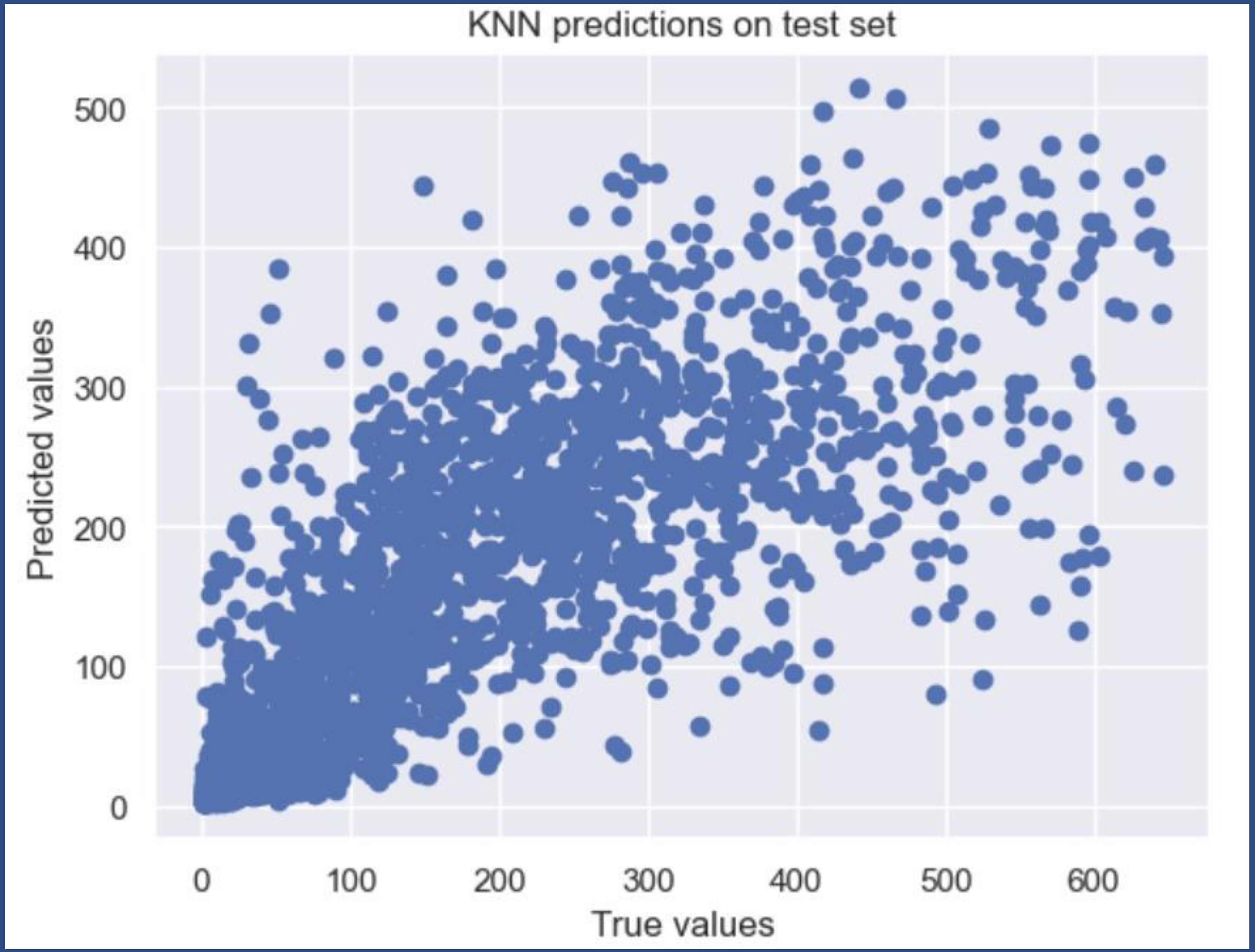
```
#KNN
n_neighbors=[]
for i in range (0,50,5):
    if(i!=0):
        n_neighbors.append(i)
params_dict={'n_neighbors':n_neighbors,'n_jobs':[-1]}
clf_knn=GridSearchCV(estimator=KNeighborsRegressor(),param_grid=params_dict,scoring='neg_mean_squared_log_error')
clf_knn.fit(x_train,y_train)
pred=clf_knn.predict(x_test)
print("RMLSE:", (np.sqrt(mean_squared_log_error(pred,y_test))))
```

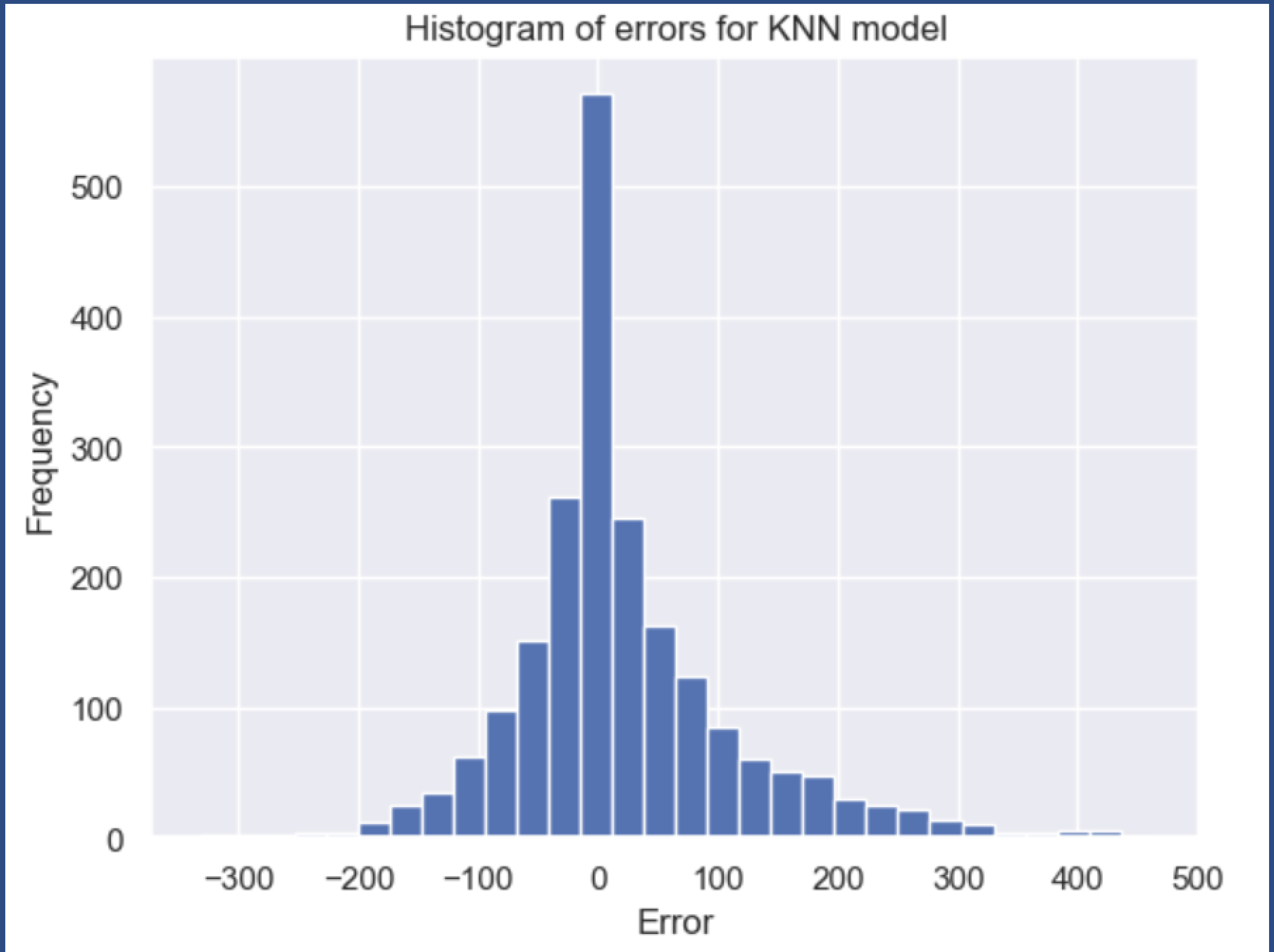
```
RMLSE: 0.18999884381041346
```

```
clf_knn.best_params_
```

```
{'n_jobs': -1, 'n_neighbors': 15}
```

- Code above uses grid search to find the optimal value of the hyperparameter n_neighbors
- Output {'n_jobs': -1, 'n_neighbors': 15} indicates that the optimal hyperparameters for the KNN model, as determined by the grid search, are n_jobs=-1 and n_neighbors=15.





- Dataset may be too large for scatter plot
- Helps to visualize the distribution of errors and identify any patterns or biases in the predictions.
- Model is able to keep a high frequency of predictions with low error, as evidenced by the high concentration of errors near zero in the middle of the histogram.

Overall

- Usage of Kneighbours is the most accurate model for companies
- Demand mainly depends on variables like temperature, season, month and hour.

Thank You