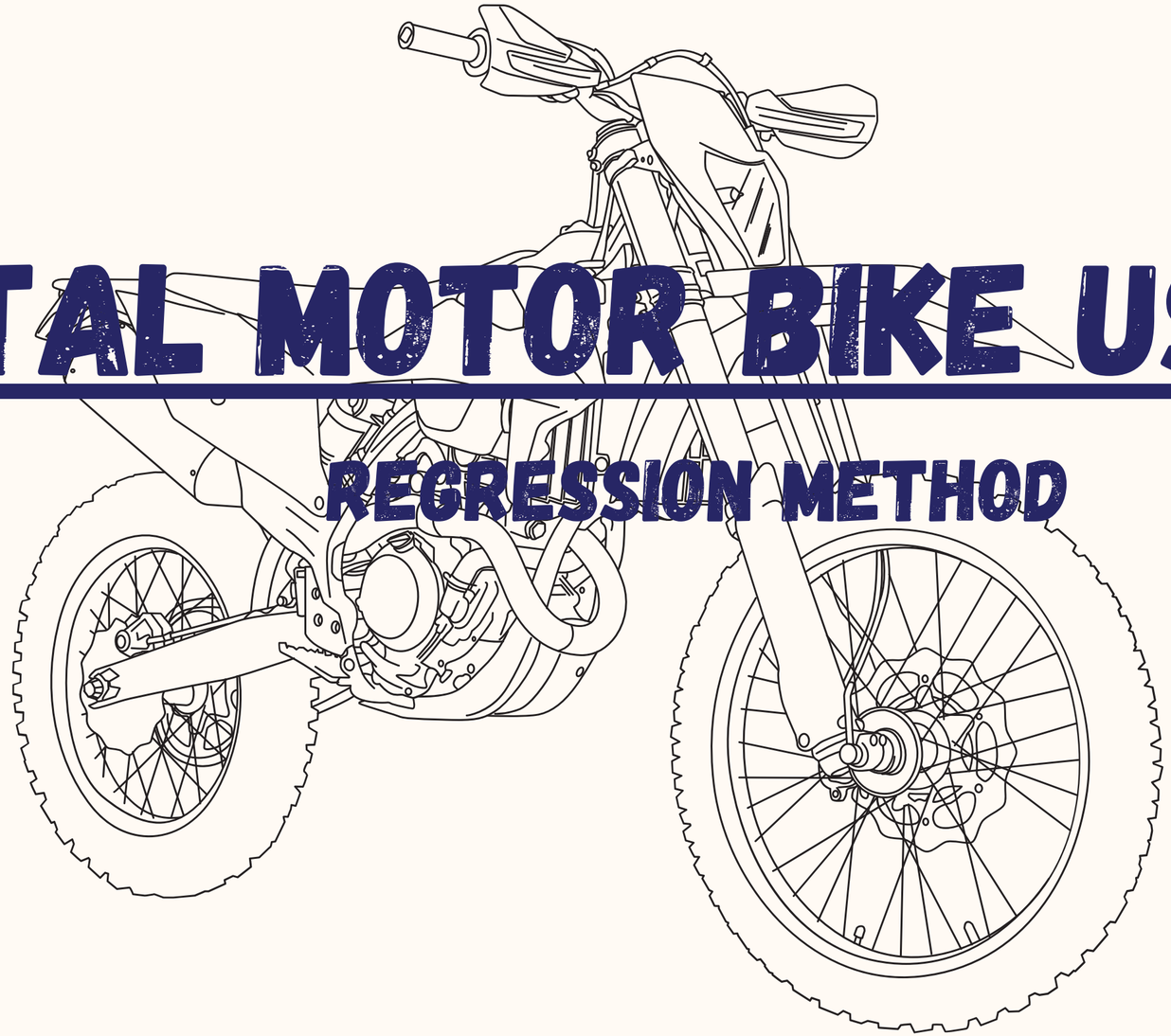
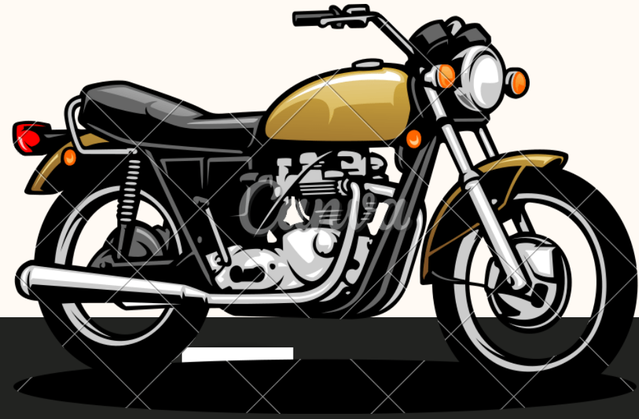
RENTAL MOTOR BIKE USERS

REGRESSION METHOD

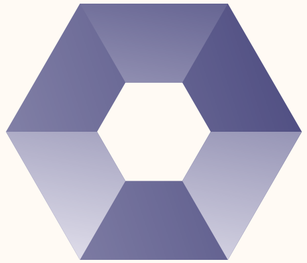


GROUP 10

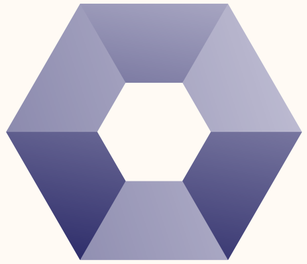
- SHIVAM SINGH
- SYED SUHEL
- NIHARIKA
- MIHIR



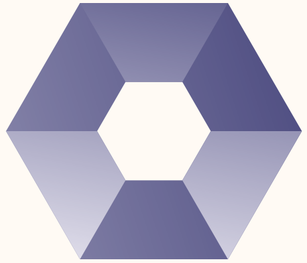
MACHINE LEARNING TIME LINE ANALYSIS



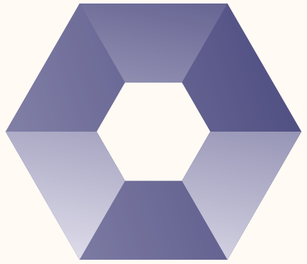
1.DATA PREPROCESSING



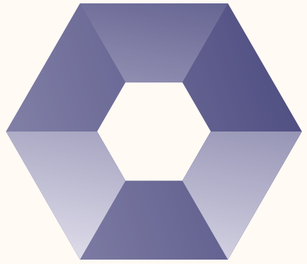
2.EXPLORATORY DATA ANALYSIS



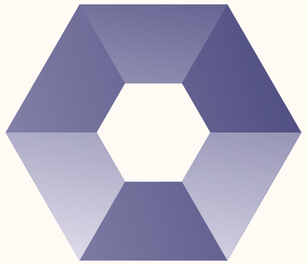
3.TRAINING THE MODEL



4.DETERMINING THE ACCURACY OF MODEL



5.COMPARING RESULTS OF OTHER MODELS



6.CONCLUSION AND INSIGHTS

ABOUT THE DATA SET

- The National Association of City Transportation Officials (NACTO): NACTO is a nonprofit organization that advocates for sustainable transportation. They have a number of resources on bike-sharing, including a report on the economic benefits of bike-sharing programs.

OBJECTIVE

- To increase access to bicycles for transportation and recreation. This can be done by providing a convenient and affordable way for people to rent bikes, and by making bikes available in a variety of locations.
- To reduce traffic congestion and pollution. By encouraging people to bike instead of drive, bike rental projects can help to reduce the number of cars on the road, which can improve air quality and reduce traffic congestion.
- To promote physical activity and health. Biking is a great way to get exercise, and bike rental projects can make it easier for people to get active.

PATH

- Implementing multiple machine learning models to fit the best model for the dataset

DATA PRE-PROCESSING

- There is no missing values
- There is no duplicate values
- Droped columns are instant,dteday,atemp

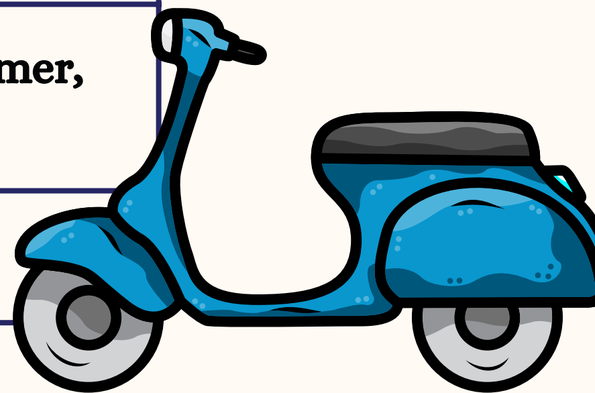
DATA AND DATA QUALITY CHECK

DATA

- This dataset consists of the hourly count of rental bikes with 17 columns and 17379 instances (rows) between the years 2011 and 2012 in the Capital bike share system.

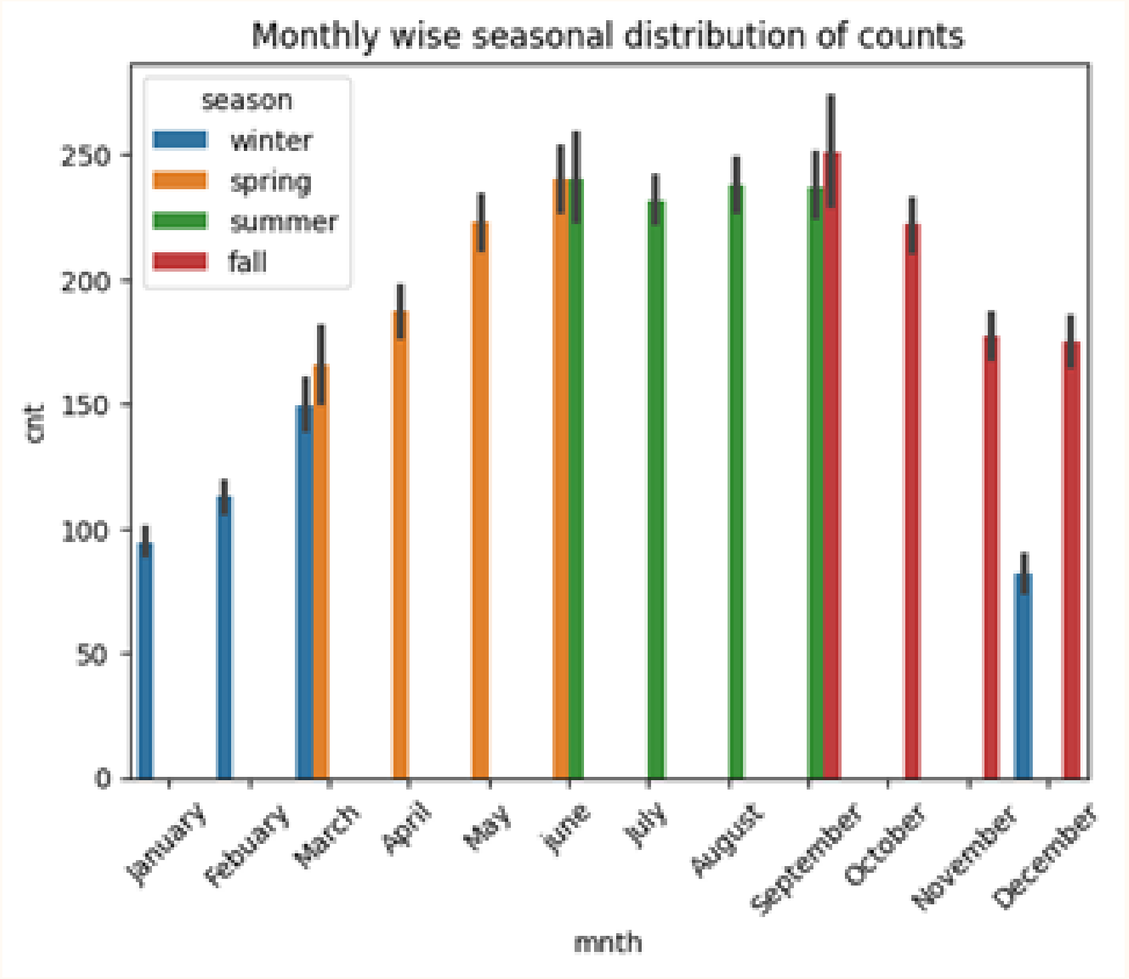
VARIABLES:

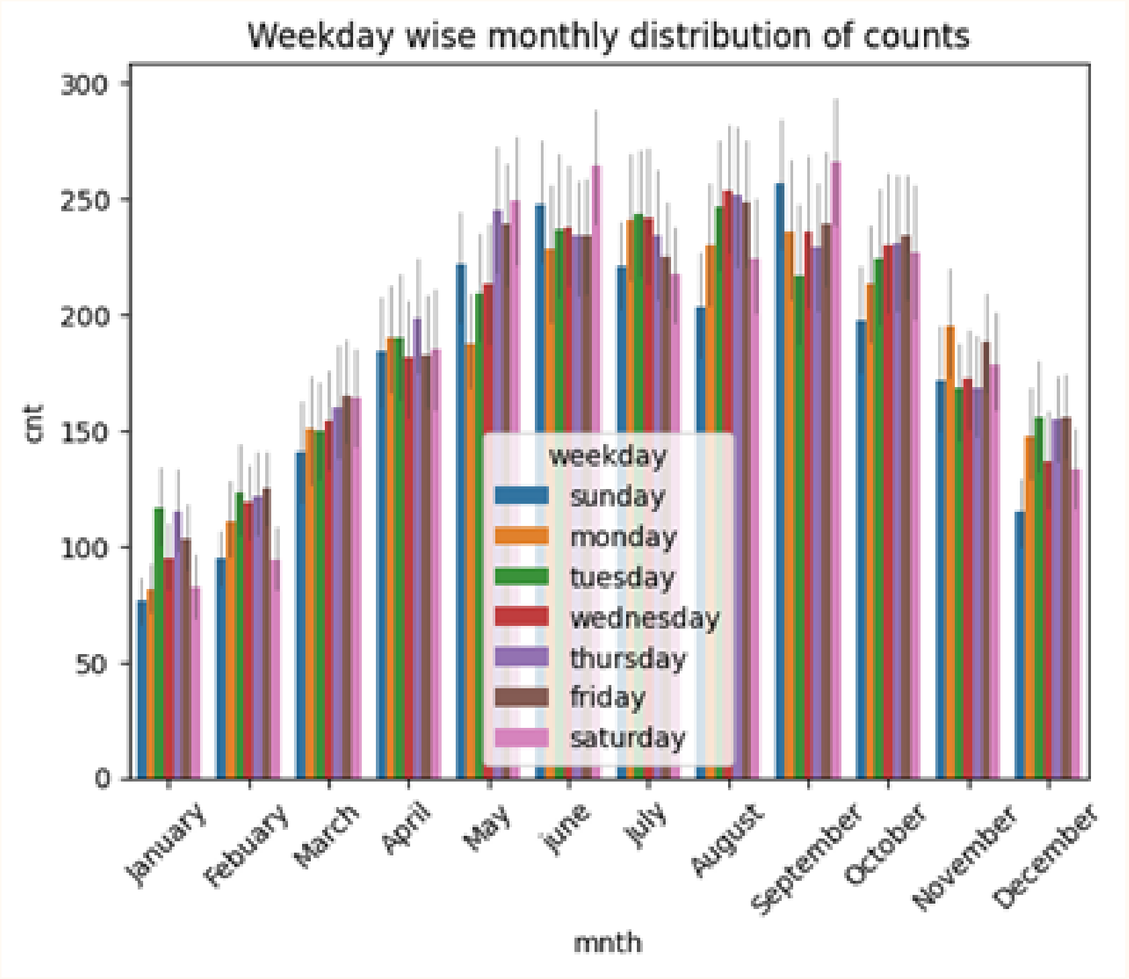
VARIABLES	variable description
• INSTANT	A unique identifier for each record or observation
• DTEDAY	Date of the observation.
• SEASON	Season of the year (e.g., 1 for spring, 2 for summer, etc.).
• YR	Year of the observation.



• MNTH	Month of the year.
• HR	Hour of the day.
• HOLIDAY	Binary indicator (0 or 1) for whether it is a holiday.
• WEEKDAY	Day of the week.
• WEATHERSIT	Weather situation or condition.
• TEMP:	Temperature.
• CASUAL	Count of casual users.
• REGISTERED	registered: Count of registered users.
• ATEMP	Apparent temperature
• HUM	Humidity.

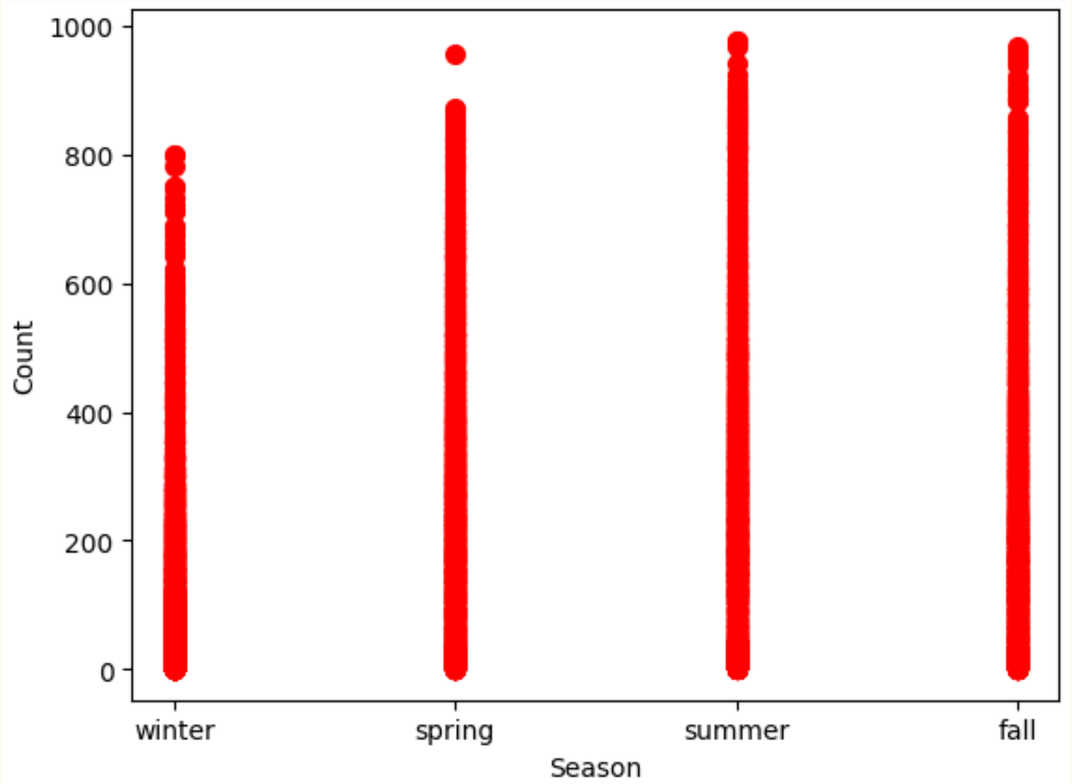
EXPLORATORY DATA ANALYSIS (EDA)



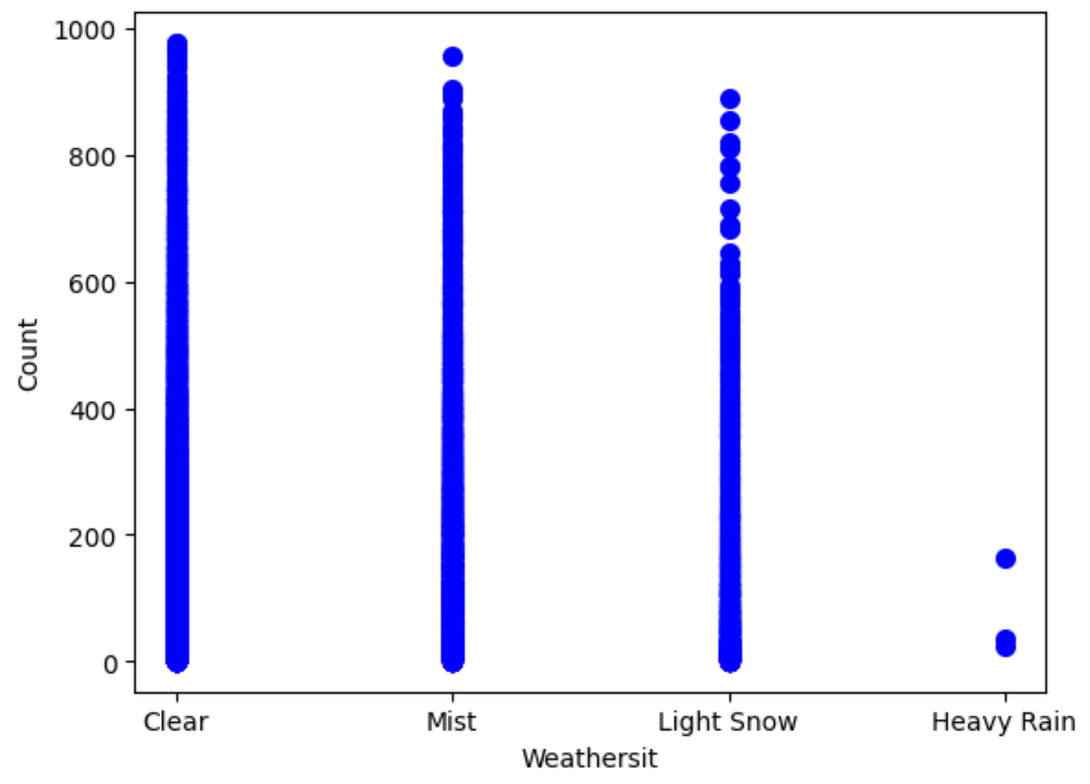


- **The above barplot gives us the idea about monthly wise seasonal distribution of counts of rental bikes**

- **The above barplot gives us the idea about Weekday wise monthly distribution of counts of rental bikes**

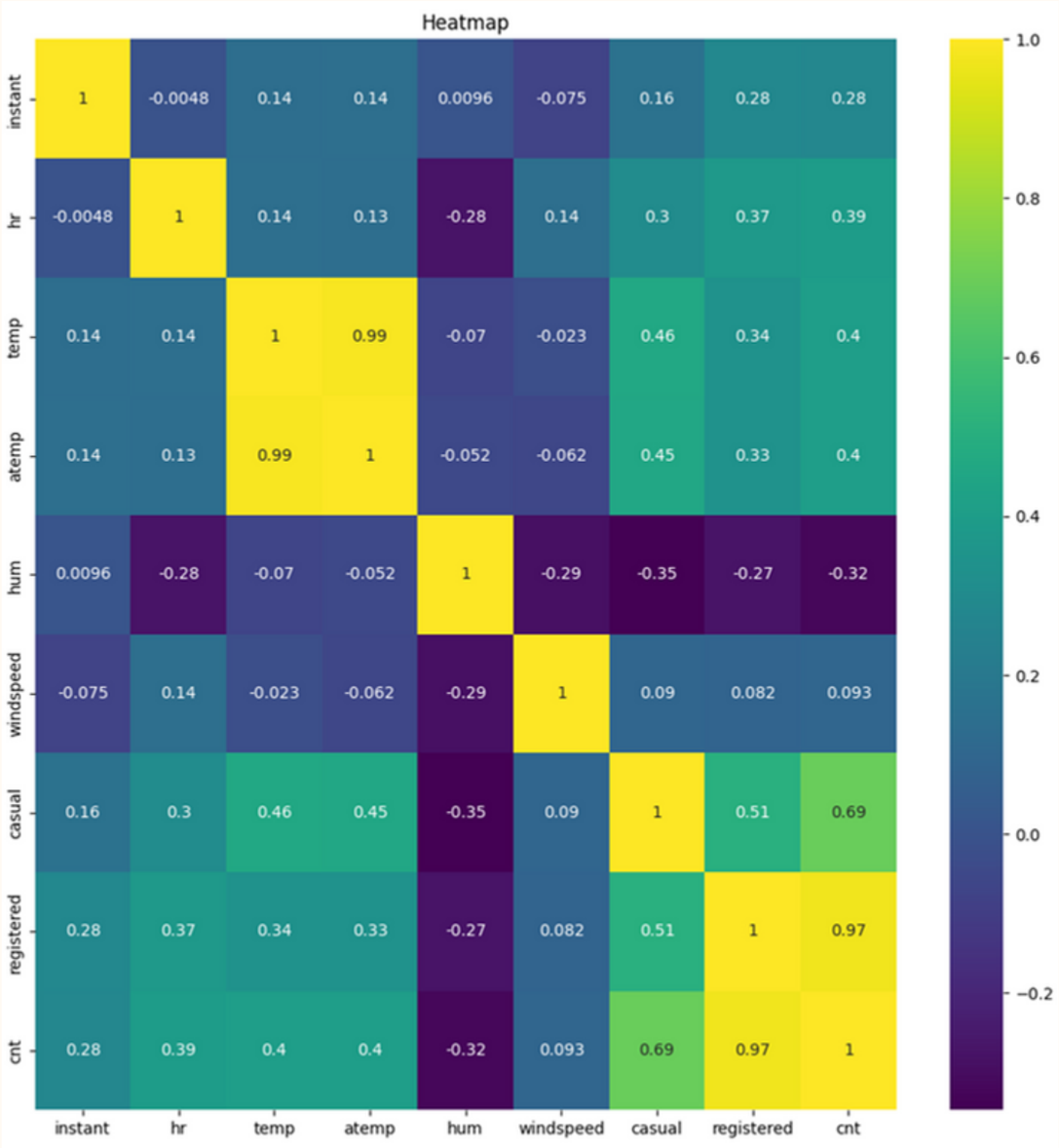


- **The above scatter plot gives us the idea about the trend between season and cnt variables.**



- **The above scatter plot gives us the idea about the trend between weathersit and cnt variables.**

CORRELATION HEAT MAP



- Correlation heatmaps are used to analyze the correlation between different variables in a dataset.
- for example here casual and windspeed are negativelly correlated because there is no relation between the causal and windspeed
- for example here casual and windspeed are positively correlated because there is relation between the temp and atemp- feels like..

VIF(VARIATION INFLATION FACTOR)

variables	VIF	
1	temp	10.569872
2	atemp	388.853624
8	season_summer	14.310397
3	hum	11.638260
7	season_spring	10.051041
9	season_fall	9.800967
16	mnth_July	9.136093
17	mnth_August	8.645864
15	mnth_june	6.969193
14	mnth_May	6.907068
18	mnth_September	6.694761
19	mnth_October	5.820669
13	mnth_April	5.787372
20	mnth_November	5.266458
23	workingday_Working day	4.285767
6	registered	3.521887
4	windspeed	3.501743
5	casual	3.449485
21	mnth_December	3.378809
12	mnth_March	2.619636
0	hr	2.390066
10	yr_2012	2.136861
11	mnth_Febuary	1.804790

Variance Inflation Factor (VIF) is a metric used in regression analysis to assess the extent of multicollinearity among independent variables. Multicollinearity occurs when two or more independent variables are highly correlated, which can lead to unstable and unreliable parameter estimates in the regression model.

ALGORITHMS

1. DECISION TREE

Train-Test Ratio	MAE	R-Squared
65-35	0.037	0.96
70-30	0.032	0.97
75-25	0.023	0.99
80-20	0.020	0.99

2. LINEAR REGRESSION

Train-Test Ratio	MAE	R-Squared
80-20	2.05	1.0
70-30	7.36	1.0
65-35	7.11	1.0
75-25	6.90	1.0

3. RIDGE REGRESSION

Train-Test Ratio	MAE	R-Squared
80-20	4.69	1.0
70-30	5.28	0.93
65-35	5.72	1.0
75-25	5.01	0.99

4. SUPPORT VECTOR MACHINE

Train-Test Ratio	MAE	R-Squared
80-20	0.071	0.939
70-30	0.73	0.89
65-35	0.075	0.79
75-25	0.074	0.99

5. RANDOM FOREST

Train-Test Ratio	MAE	R-Squared
80-20	0.0210	0.99
70-30	0.0201	0.99
65-35	0.0321	0.99
75-25	0.0257	0.99

6. KNN

Train-Test Ratio	MAE	R-Squared
80-20	0.65	0.99
70-30	0.68	0.99
65-35	0.69	0.99
75-25	0.67	0.99

BOOSTING

1. ADA BOOSTING

Train-Test Ratio	n_estimators	Learning Rate	MAE
65-35	2000	0.1	11.39
70-30	2000	0.1	10.73
75-25	2000	0.1	10.56
80-20	2000	0.1	10.06

2. GRADIENT BOOSTING

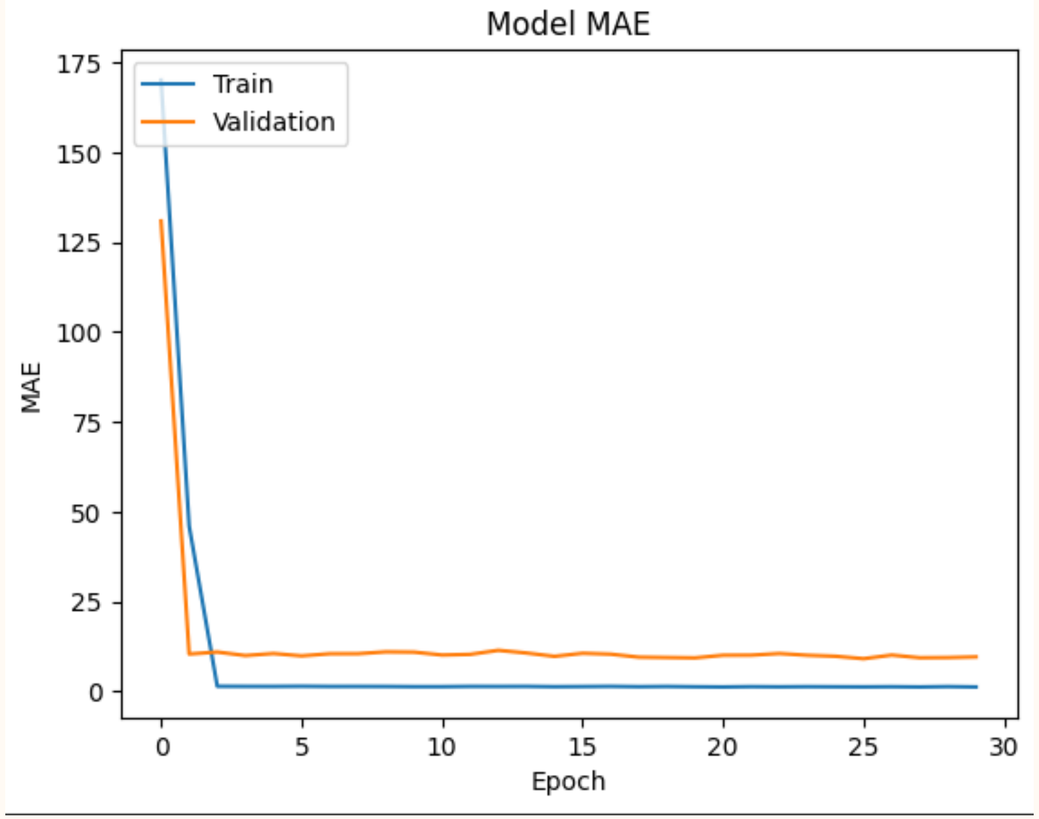
Train-Test Ratio	Learning Rate	n_estimators	MAE
65-35	0.1	100	0.99
70-30	0.1	100	0.98
75-25	0.1	100	1.0
80-20	0.1	100	0.98

3. XG BOOSTING

Train-Test Ratio	n_estimators	Learning Rate	MAE
65-35	1000	0.05	0.864
70-30	1000	0.05	0.861
75-25	1000	0.05	0.852
80-20	1000	0.05	0.85

NEURAL NETWORK

Train-Test Ratio	Architecture	Optimizer	Epochs	MAE
65-35	17-4-1	Adam	30	15.9567
65-35	13-7-1	Adam	100	15.9673
65-35	2-9-1	Adam	30	16.0941
65-35	10-5-1	Adam	100	16.1617
70-30	16-12-10	Adam	30	15.9291
70-30	12-7-3	Adam	100	16.0296
75-25	13-5-3	Adam	100	16.091



Train_Test	**65-35**
Architecture	**13-7-1**
Optimizer	**Adam**
mae	**15.96**
Epochs	**30**

COMPARISION OF IMPLEMENTED MODELS

MODEL	MAE(Test Size 75-25)
Linear Rregression	2.05
Random forest	0.02
K Nearest Regressor	0.65
Support Vector Regressor	0.071
Decision Tree	0.026
Gradient Boosting	1.0
Ada Boost	10.06
XG Boost	0.85
Neural Network	15.92

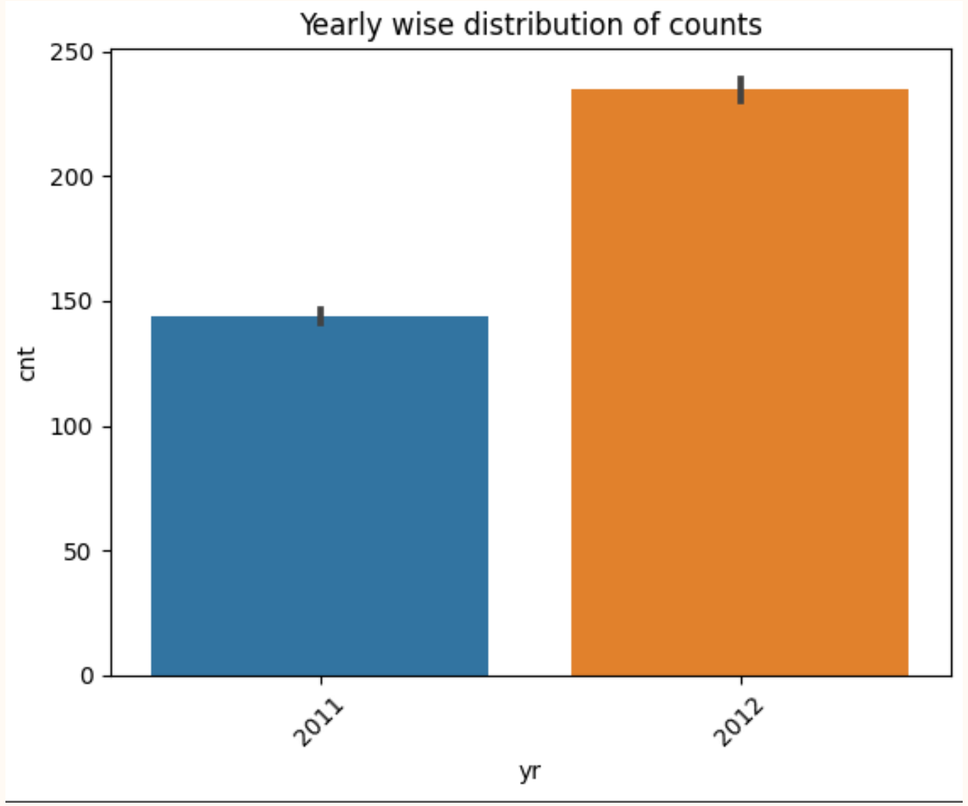
SUMMARY

- Bike-sharing systems have become increasingly popular in recent years as a convenient and eco-friendly mode of transportation. To effectively manage these systems and ensure optimal resource allocation, it is crucial to accurately predict the number of bike users. Bike user prediction involves utilizing historical data and various factors to forecast the demand for bike rentals at a given time and location. The random forest gives us the best result with train and test ratio of 75-25 with MAE of 0.02.
- Key Factors Influencing Bike User Prediction:
- Weather Conditions: Weather plays a significant role in bike usage. Favorable weather conditions, such as warm temperatures and sunny skies, encourage more people to opt for cycling.
- Time of Day and Day of the Week: Bike usage patterns vary throughout the day and week. Rush hour periods and weekends typically experience higher demand for bike rentals.

Seasonality: Bike usage tends to be higher during warmer months and lower during colder months.

Holidays and Events: Special events or holidays can lead to increased bike usage in specific areas.

Availability of Bikes: The availability of bikes at a particular station can influence user behavior.



This graph show that how rental motor bike users are increasing yearly with increase rate of 50% every year.

The bike rental users are increased 7% every year and made a 36.48 billion USD in 2022 and in 2023 it made a revenue of 48.36 billion USD

THANK YOU

APPENDIX

ALGORITHM-1

DECISSION TREE CLASSIFIER

```
# Creating an instance of a LinearRegression() model named model
model1=DecisionTreeRegressor()

# Fitting the lm model with training set
model1.fit(x_train,y_train)
# Predicting the y values by using the .predict() function
y_pred = model1.predict(x_test)
y_pred
```

```
array([586.,  52., 187., ...,   6., 190., 539.])
```

```
model1.fit(x_train,y_train)
```

```
▾ DecisionTreeRegressor
DecisionTreeRegressor()
```

```
# Model performance
print('MAE:',metrics.mean_absolute_error(y_test,y_pred))
print('RMSE:', np.sqrt(metrics.mean_squared_error(y_test, y_pred)))
print('R2 score:', metrics.r2_score(y_test, y_pred))
```

```
MAE: 0.04602991944764097
RMSE: 0.5073302339154703
R2 score: 0.9999921424864819
```

ALGORITHM-2

LINEAR REGRESSION

```
# Importing LinearRegression model from scikit learn library
from sklearn.linear_model import LinearRegression
from sklearn import metrics
```

```
# Creating an instance of a LinearRegression() model named model2
model2 = LinearRegression()
# Fitting the  model with training set
model2.fit(x_train,y_train)
# Predicting the y values by using the .predict() function
y_pred = model2.predict(x_test)
y_pred
```

```
array([586.,  52., 187., ...,   6., 190., 539.])
```

```
model2.fit(x_train,y_train)
```

```
▾ LinearRegression
LinearRegression()
```

```
# Model performance
print('MAE:',metrics.mean_absolute_error(y_test,y_pred))
print('RMSE:', np.sqrt(metrics.mean_squared_error(y_test, y
print('R2 score:', metrics.r2_score(y_test, y_pred))
```

```
MAE: 6.540499719198048e-14
RMSE: 8.518529913107079e-14
R2 score: 1.0
```

ALGORITHM-3

RIDGE REGRESSION

```
# Importing RidgeRegression model from scikit learn library
from sklearn.linear_model import Ridge
from sklearn.model_selection import train_test_split
from sklearn import metrics


#Create Ridge Regression Model
ridge_reg= Ridge()
#Train the model
ridge_reg.fit(x_train,y_train)
#Predict the response for test dataset
y_pred = ridge_reg.predict(x_test)
y_pred
```

```
array([585.99999973,  52.00000002, 187.00000006, ...,   5.99999929,
       189.9999991 , 539.00000018])
```

```
# Model performance
print('MAE:',metrics.mean_absolute_error(y_test,y_pred))
print('RMSE:', np.sqrt(metrics.mean_squared_error(y_test, y_pred)))
print('R2 score:', metrics.r2_score(y_test, y_pred))
```

```
MAE: 5.038579768678448e-07
RMSE: 7.191032176020646e-07
R2 score: 1.0
```

ALGORITHM-4

SUPPORT VECTOR MACHINE(SVM)

```
# Importing SVR model from scikit learn library
from sklearn.svm import SVR
from sklearn import metrics
```

```
# Creating an instance of a SVR() model named svr
svr = SVR(kernel = 'linear')
```

```
# Fitting the svr model with training set
svr.fit(x_train, y_train)
```

```
SVR(kernel='linear')
```

```
# Predicting the y values by using the .predict() function
y_pred = svr.predict(x_test)
y_pred
```

```
array([586.04596299,  52.0932993 , 187.08386211, ...,   6.09935315,
       190.06432745, 539.06598919])
```

```
# Model performance
print('MAE:',metrics.mean_absolute_error(y_test,y_pred))
print('RMSE:', np.sqrt(metrics.mean_squared_error(y_test, y_pred)))
print('R2 score:', metrics.r2_score(y_test, y_pred))
```

```
MAE: 0.08006476782534498
RMSE: 0.08319064776256669
R2 score: 0.9999997887227868
```

ALGORITHM-7

RANDOM FOREST

```
# Importing RandomForestRegressor model from scikit learn library
from sklearn.ensemble import RandomForestRegressor
from sklearn import metrics
```

```
# Creating an instance of a RandomForestRegressor() model named random_forest_regressor
random_forest_regressor = RandomForestRegressor(n_estimators=100, random_state=1)
```

```
# Fitting the clf model with training set
random_forest_regressor.fit(x_train, y_train)
```

```
RandomForestRegressor
RandomForestRegressor(random_state=1)
```

```
# Predicting the y values by using the .predict() function
y_pred = random_forest_regressor.predict(x_test)
y_pred
```

```
array([586.13,  52.  , 187.  , ...,   6.  , 190.  , 538.98])
```

```
# Model performance
print('MAE:',metrics.mean_absolute_error(y_test,y_pred))
print('RMSE:', np.sqrt(metrics.mean_squared_error(y_test, y_pred)))
print('R2 score:', metrics.r2_score(y_test, y_pred))
```

```
MAE: 0.032378212504793634
RMSE: 0.32102085916880885
R2 score: 0.9999968539171642
```

ALGORITHM-5

KNN

```
# Importing KNeighborsRegressor model from scikit learn library
from sklearn.neighbors import KNeighborsRegressor
from sklearn import metrics
```

```
# Creating an instance of a KNeighborsRegressor() model named knn
knn = KNeighborsRegressor(n_neighbors=5)
```

```
# Fit the model to the training data
knn.fit(x_train, y_train)
```

```
KNeighborsRegressor
KNeighborsRegressor()
```

```
# Predicting the y values by using the .predict() function
y_pred = knn.predict(x_test)
y_pred
```

```
array([586.8,  51.6, 187. , ...,   6. , 189.2, 540.6])
```

```
# Model performance
print('MAE:',metrics.mean_absolute_error(y_test,y_pred))
print('RMSE:', np.sqrt(metrics.mean_squared_error(y_test, y_pred)))
print('R2 score:', metrics.r2_score(y_test, y_pred))
```

```
MAE: 0.7018028385116991
RMSE: 1.6701122802879753
R2 score: 0.9999148479737726
```

ALGORITHM-6

XG BOOST

```
# Importing GradientBoostingRegressor model from scikit learn library
from xgboost import XGBRegressor
```

```
# Creating an instance of a XGBRegressor() model named xg
xgboost= XGBRegressor(n_estimators=1000, learning_rate=0.05)
# Fit the model to the training data
xgboost.fit(x_train, y_train)
```

XGBRegressor

```
XGBRegressor(base_score=None, booster=None, callbacks=None,
             colsample_bylevel=None, colsample_bynode=None,
             colsample_bytree=None, device=None, early_stopping_rounds=None,
             enable_categorical=False, eval_metric=None, feature_types=None,
             gamma=None, grow_policy=None, importance_type=None,
             interaction_constraints=None, learning_rate=0.05, max_bin=None,
             max_cat_threshold=None, max_cat_to_onehot=None,
             max_delta_step=None, max_depth=None, max_leaves=None,
             min_child_weight=None, missing=nan, monotone_constraints=None,
             multi_strategy=None, n_estimators=1000, n_jobs=None,
             num_parallel_tree=None, random_state=None, ...)
```

```
# Predicting the y values by using the .predict() function
y_pred = xgboost.predict(x_test)
y_pred
```

```
array([580.5737  ,  52.41975 , 187.57654 , ...,   6.005221, 189.22935 ,
       532.3708  ], dtype=float32)
```

```
# Model performance
print('MAE:',metrics.mean_absolute_error(y_test,y_pred))
print('RMSE:', np.sqrt(metrics.mean_squared_error(y_test, y_pred)))
print('R2 score:', metrics.r2_score(y_test, y_pred))
```

```
MAE: 0.8908840207290869
RMSE: 3.0234314823431623
R2 score: 0.9997209359390519
```

ALGORITHM-7

GRADIANT BOOSTING

```
 Importing GradientBoostingRegressor model from scikit learn library
rom sklearn.ensemble import GradientBoostingRegressor
```

```
 Creating an instance of a GradientBoostingRegressor() model named gbr
br = GradientBoostingRegressor(n_estimators=100, learning_rate=0.1, max_depth=3)
```

```
 Fit the model to the training data
br.fit(x_train, y_train)
```

```
 GradientBoostingRegressor
radientBoostingRegressor()
```

```
 Predicting the y values by using the .predict() function
_pred = gbr.predict(x_test)
_pred
```

```
rray([584.18363385,  52.6326978 , 188.10832962, ...,   6.04324733,
      188.68840761, 537.52990323])
```

```
 Model performance
rint('MAE:',metrics.mean_absolute_error(y_test,y_pred))
rint('RMSE:', np.sqrt(metrics.mean_squared_error(y_test, y_pred)))
rint('R2 score:', metrics.r2_score(y_test, y_pred))
```

```
AE: 0.9907608507170651
MSE: 1.4809049452624634
2 score: 0.9999330488540904
```

ALGORITHM-8

NEURAL NETWORK

```
import tensorflow as tf
from tensorflow.keras import layers, models
from sklearn.preprocessing import StandardScaler


# Normalize the data (important for neural networks)
scaler = StandardScaler()
X_train = scaler.fit_transform(x_train)
X_test = scaler.transform(x_test)



#Build neural network model
def build_model():
    model = models.Sequential([
        layers.Dense(64, activation='linear', input_shape=(X_train.shape[1],)),
        layers.Dense(32, activation='linear'),
        layers.Dense(1) #No activation for Regression
    ])

    #Compile the model
    model.compile(optimizer='rmsprop', loss='mse', metrics=['mae'])
    return model

model = build_model()
```