Daily Based Rental Bike Regressor Using Classical ML Techniques (With Comparative Study)

Iman Hindi
Computer Engineering Department.
University of Jordan.
Amman,Jordan.
aym8220825@ju.edu.jo

Mais Al-Maqableh
Computer Engineering Department.
University of Jordan.
Amman,Jordan.
mys8220956@ju.edu.jo

Prof.Gheith Abandah
Computer Engineering Department.
University of Jordan.
Amman,Jordan.
abandah@ju.edu.jo

***Abstract*— In this project, we studied the performance of the most common classical machine-learning algorithms in predicting daily rental bikes demand. We made a comparison between different ML algorithms using different evaluation metrics, such as RMSE, MSE, R2Score and confidence intervals. These comparisons shows that Random Forest and Gradient Boosting Regressors (GBR) can fit the data perfectly and return a good results (RMSE<15%). We choose the GBR which is the best model (has the lowest RMSE (14.5%) value and the highest R2Score (88%)). Then, we systematically fine-tuned the hyper-parameters of the GBR model using grid search and randomized search. In addition, we did some feature engineering to improve the prediction results. We used cross-validation errors to evaluate the training process and to check overfitting, then we used the test set to evaluate the final model. The final step, we try to enhance the performance of the model by applying the data augmentation technique to enlarge the data set samples. And this tech. significantly improve the system, and enhance the evaluation results compared with model without data augmentation.**

Keywords—ML, SVM, Linear Regression, Gradient Boosting, Decision Tree, Random Forest, Lasso, Ridge, Elastic Net, K-nearest Neighbor, rental bikes, regression.

I. INTRODUCTION

Bike sharing system become widely used in urbans places especially in geographical areas that is suitable for riding due to its mobility comfort for individuals, so they don't need to concern about thefts, parking, storage or even maintenance. Beside its positive impact on the whole communities, since it reduce traffic congestion, car parking, pollution, and it is also a healthy transport. The bicycle can be rented or borrowed from a location and returned to another location.

In the last decade, bicycle renting systems locations and stations were not automated but were run by employees or volunteers. Recently, most of these stations become automated with a powerful system that efficiently control the process and let the users easily and rapidly service themselves. These systems is controlled by regional programs that have been implemented where numerous renting locations are set up at railway stations and at local businesses (usually restaurants, museums and hotels) creating a network of locations where bicycles can be borrowed from and returned.

Since bike-sharing systems provide a discrete and limited number of bikes, whose distribution can vary throughout a city. One person's usage of the good diminishes the ability of others to use the same good. This can be efficiently achieved, by supply the stations with sufficient number of bicycles such that one person's usage does not encroach upon another's use of the good, and thus make the rental bike available and accessible to the public at the right time as it lessens the waiting time. Eventually, providing each station with a stable supply of rental bikes that can cover the varied demand became a major concern. The crucial part here is the prediction of bike count required at each day or hour for the stable supply of rental bikes by taking into considerations all the factors that can affected the rental bikes demand.

Many researches responding to this demand and trying to build a powerful and reliable system able to precisely detect the count of bikes that matches the real counts. Since classical ML algorithm shows high promises in solving such complex problems. Most researchers used them with data set of daily counts others used them in more detailed dataset with hourly counts. Shian, in his theses [1], uses different classical ML algorithms to predict the hourly demand for bicycles in a bike-sharing system by mining data on historical rental patterns and weather information.

In [2] the authors applied four machine learning models, linear regression, the k-nearest neighbors (KNN), random forest, and support vector machine, to predict consumer demand for bike sharing in Seoul, they incorporating features other than weather - such as air pollution, traffic information, Covid-19 cases, and social economic factors- to increase prediction accuracy (29 features) the results shows a good performance for SVM in addition to random forest model.

Lu & Lin [3] used recurrent neural networks (RNN) to timely base prediction for the bikes counts and finding the trend on the number of counts based more informative factors, which depends significantly on the weather forecasts.

In fact, bike sharing modeling entails short-term even almost instantaneous demand prediction. On the other hand, machine learning models need to take into account variance in the stations level of bike demand, which would allow suppliers to deploy bikes efficiently across destination to ensure supply. Such deployment requires modeling and forecasting demand across different docking stations on an hourly basis depending on the degree of demand fluctuation [2].

However, In our case study, we will focus on studying the performance of different ML algorithms on predicting the daily bike rental counts for data set of two-year daily basis historical log corresponding to years 2011 and 2012 from Capital Bike-share system (Washington D.C., USA), comparing the models performances, selecting the best model performance, and then fine tune the best scored model to enhance the predictions and minimizing the RMSE error.

II. DATA SET

Data set in this project is taken from Kaggle [4], which consist of two-year historical log (number of records: 731 days) corresponding to years 2011 and 2012 from Capital Bike share system, Washington D.C., USA, which is concatenated with the corresponding weather and seasonal information. A sample of the data used is listed in TABLE I. below.

TABLE I. BIKE RENTAL DATA SET SAMPLE

	instant	dteday	season	yr	mnth	holiday	weekday	workingday	weathersit	temp	atemp	hum	windspeed	casual	registered	cnt
0	1	2011-01-01	1	0	1	0	6	0	2	0.344167	0.363625	0.805833	0.160446	331	654	985
1	2	2011-01-02	1	0	1	0	0	0	2	0.363478	0.353739	0.696087	0.248539	131	670	801
2	3	2011-01-03	1	0	1	0	1	1	1	0.196364	0.189405	0.437273	0.248309	120	1229	1349
3	4	2011-01-04	1	0	1	0	2	1	1	0.200000	0.212122	0.590435	0.160296	108	1454	1562
4	5	2011-01-05	1	0	1	0	3	1	1	0.226957	0.229270	0.436957	0.186900	82	1518	1600

As shown in the table above the dataset contains 12 features as follows:

A. *Categorical features*

Date of the day, Seasons, Year, Months, Holiday, Weekday, Working day, Weather situation.

B. *Numerical features*

Temperature, Avg-temperature, Wind speed and Humidity.

C. *Rental bikes' counts*

Count of total rental bikes including both casual and registered.

These 12 features are the potential features influencing bike sharing demand. When weather or air quality is bad, people might be reluctant to rent a sharing bike. On the other hand, when traffic is bad, renting a bike will be more efficient.

III. DEVELOPMENT ENVIRONMENT

The models built and tested using VS_Code with Python and Jupiter notebook, the system that is used for development is windows_10 that is run at a CPU with specifications of (Intel(R) Core(TM) i7-10810U CPU @ 1.10GHz 1.61 GHz), With RAM (16.0 GB) and (64-bit) operating system.

We use the guide in "Hands-on Machine Learning with Scikit-Learn, Keras and TensorFlow (3rd edition)" [5] text book and its code repository [6] to derive our project code. We follow its guidelines and its approach to solve such problems, build Models and evaluate the different models using several metrics (RMSE, R2_Score, and Confidence Interval).

IV. DEVELOPMENT PROCESS

A. *Data Set Exploration and Preprocessing*

the dataset has no null values and no duplication on samples, the values of the numerical features is normally distributed and also the categorical features is already encoded using ordinal or one hot encoder, so there is not a lot to do with data, since it's already prepared for training.

Here, a little preprocessing steps are done to explore the data set perfectly and prepare it to fit the ML algorithms.

1. Normalize the data set numerical features using Standard scaler.
2. Perform some feature engineering to derive more useful features, by extracting the day component from the existing dteday column.
3. Drop Outliers in the sample values that may affect the performance of the model and make the data skewed. Fig. 1 shows the outliers values on humidity and wind speed features.

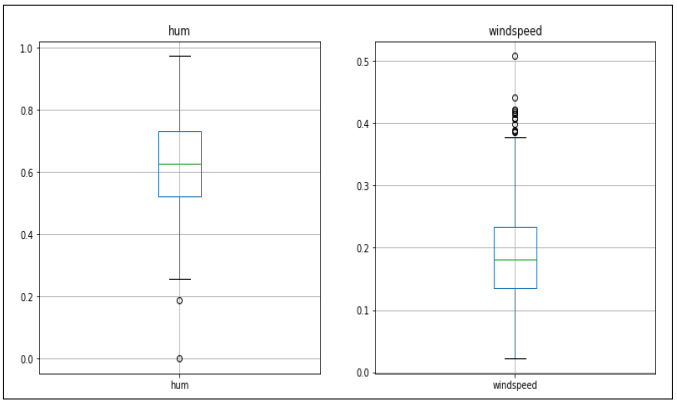


Fig. 1. Outliers on humidity and wind speed features.

4. Calculate the counts of each categorical features values to show how the categorical features distributed and check if there is some imbalances in different categories, Fig. 2 shows a high imbalances in the holiday feature's categories, this difference in the counts of both categories could affect the performance results.

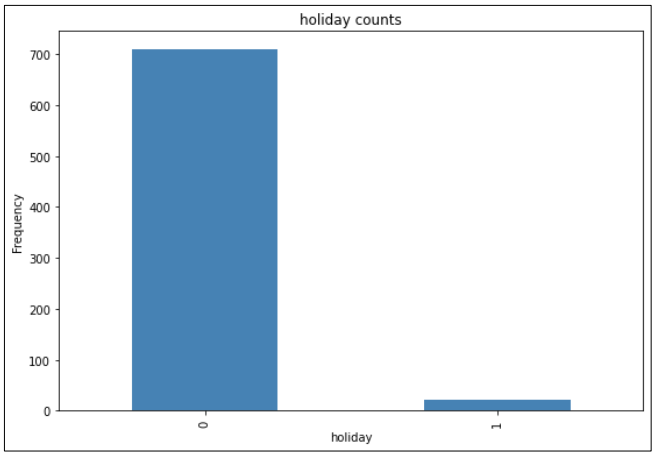


Fig. 2. Imbalances in the holiday feature's categories.

5. Check the relation between number of counts and the categorical features, and here we use boxplot of each feature with the count distribution to give us insights of how the counts of rental bikes in each categorical feature and the most important features that affect the number of bikes rented during the day.

   The boxplot shows that the most significant categorical feature is the year, that's mean that the demand on bike sharing is increased each year and more users is preferred to choose the bike services of this company.

Also, it shows that the users preferred to use the bikes when the weather situation is clear while the demand on bikes is decrease when heavy rain status and that also explain why the use of bikes is highly decreases during the winter season compared with other seasons.

The Fig. 3 shows how the counts is differ in each year, how the seasons is affected the demand on bikes.

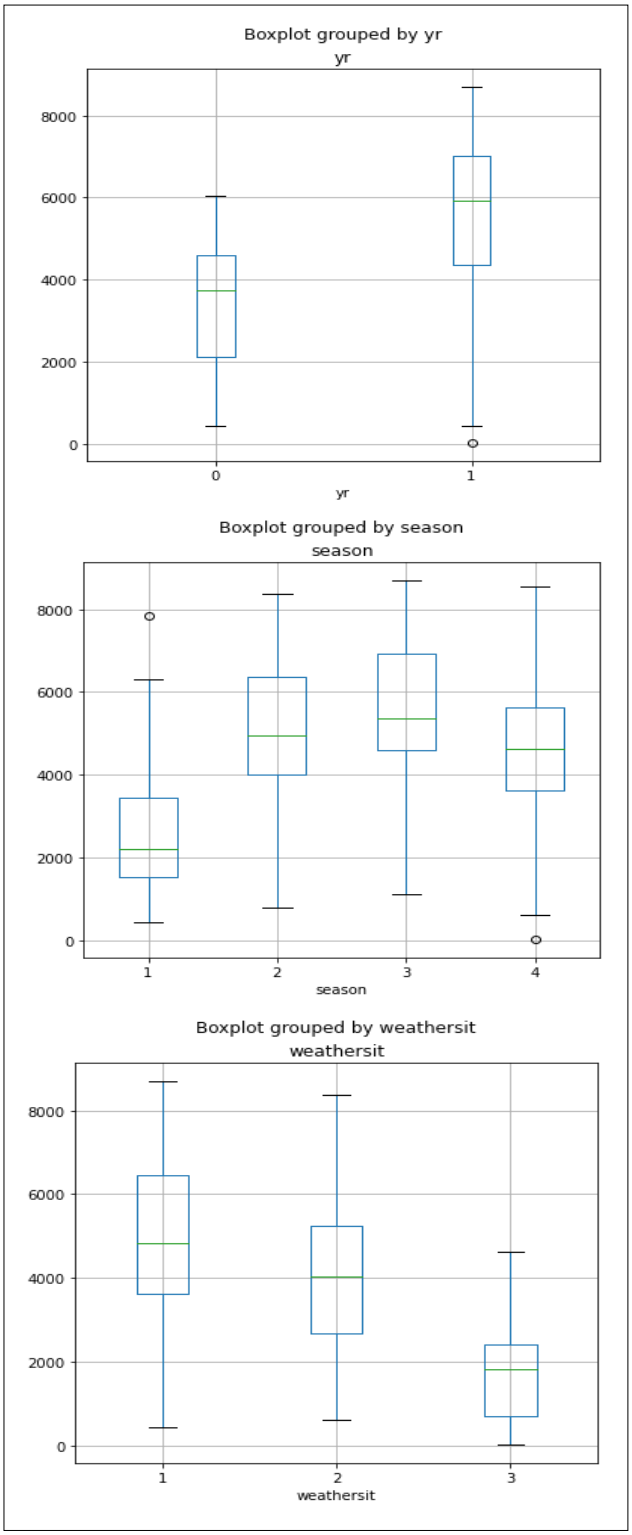


Fig. 3. Boxplot of rental bikes' counts with respect to year, season, and weather situation categorical features.

6. Calculate the Correlation between output and numerical features. Fig. 4 below shows the scatter plot of bike rental counts vs. the temperature, which shows that the number of bikes rented is highly correlated with the temperature and that's also supports the increase the demand on rental bikes during the summer season, in other words when the weather is suitable for riding a bikes.

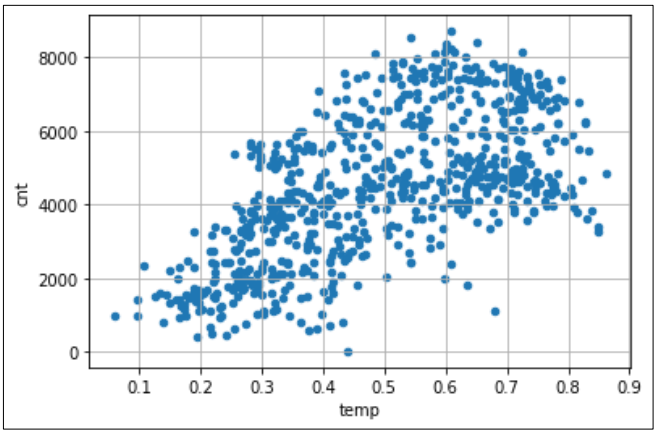


Fig. 4. Scatter plot of bike rental counts vs. temperature.

B. Splitting the Data for Machine Learning Algorithms

Since the data set is imbalanced towards the holiday feature we split the data set to train and test using stratified split with respect to holiday feature to ensure that the data set is representative to all sample values, and thus lead to more accurate performance results.

C. Building and Training the Models

We performed ten machine learning algorithms to predict bike rentals, which are linear regression, Gradient Boosting Regressor, Lasso, Ridge, ElasticNet, Kneighbors Regressor (KNN), Decision Tree Regressor, Random Forest Regressor, Linear SVR, SVR and evaluate them using cross validation technique.

1. Linear regression We utilize linear regression as a baseline against which other more sophisticated models are compared for their predictive power due to its simplicity and clear economic sense in describing the relationship between factors and the outcome.

2. Gradient Boosting Regressor (GBR) Gradient Boosting is a machine learning technique for regression and classification problems, and it is an ensemble learning method.

3. Lasso Regression is a type of linear regression that uses a regularization term called the L1 regularization term. The term is added to the cost function during training, which has the effect of penalizing large coefficients in the model. This can be helpful in preventing overfitting, as it reduces the complexity of the model by forcing some of the coefficients to be equal to zero.

4. Ridge regression is a type of linear regression that includes an L2 regularization term, or "penalty," in the cost function. This term is added to the ordinary least squares cost function and serves to shrink the coefficients of the model toward zero, which can help prevent overfitting and improve the generalization of the model to unseen data.

5. ElasticNet is a type of regularized linear regression that combines the L1 and L2 regularization penalties of the Lasso and Ridge methods. It is defined by the following cost function:

cost = MSE + alpha * (l1_ratio * L1 + (1 - l1_ratio) * L2)

6. K-Nearest Neighbors (KNN). The KNN is a nonparametric technique which provides solution for the curve fitting of unknown shape, because it does not assume specific forms of regression functions. explanatory variables take into account the k (a positive integer) closest instances. The calculations of the KNN are based on distances between an instance to its neighbors.

7. Decision Tree Regressor: The model makes predictions for a new data point by traversing the tree and finding the leaf node that most closely resembles the new data point. The prediction for the new data point is then given by the mean target value of the training examples in the leaf node.

8. Random Forest based model Decision tree (DT).

Random Forest is an ensemble of multiple decision trees in training section and to output the average of every decision tree's prediction.

It improves the accuracy and the overfitting problem in decision trees because each tree in Random Forest won't grow too deep so it won't learn highly irregular patterns.

9. LinearSVR is a specific type of SVR that uses linear functions to make predictions. This means that the prediction function is a linear combination of the input features

10. Support Vector Regression (SVR), SVR algorithms are based on the idea of support vectors, which are the data points that are the most informative and have the largest impact on the prediction function.

D. Evaluate Different Models' Performance

All models are evaluated using different metrics, MSE, RMSE, and R2 Score on the training and testing data, the results in the table below is listed the performance of each model on testing set, TABLE II. shows that the gradient boosting and random forest regressors have the best performance evaluation score, Gradient boosting regressor RMSE (0.150 and 0.145) and in Random forest regressor RMSE (0.154 and 0.152) on training and testing respectively. The GBR also achieved the highest R2_Score (88%).

TABLE II. RMSE, MSE, AND R2_SCORE FOR EACH MODEL ON TESTING DATASET

Model	RMSE	MSE	R2_Score
GradientBoostingRegressor	0.145082	94.825471	0.880503
RandomForestRegressor	0.152231	104.369425	0.868476
Ridge	0.195944	172.954677	0.782047
Lasso	0.197121	175.040506	0.779419
LinearRegression	0.197698	176.059654	0.778134
DecisionTreeRegressor	0.209198	197.119302	0.751595
ElasticNet	0.260748	306.251125	0.614070
KNeighborsRegressor	0.338828	517.150755	0.348300
SVR	0.423857	809.246626	-0.019791
LinearSVR	0.445991	895.924815	-0.129021

In addition, the confidence intervals of RMSE at confidence Coeff=0.95 are also measured on the testing data for each model, the below figure shows that Gradient boosting and Random forest regressors have the lowest RMSE confidence intervals, [.12, .16] and [.13, .17] respectively.

Fig. 5 below also shows that both LinearSVR and SVR regressors have the worst RMSE confidence intervals, which means they are not fitting the data well.

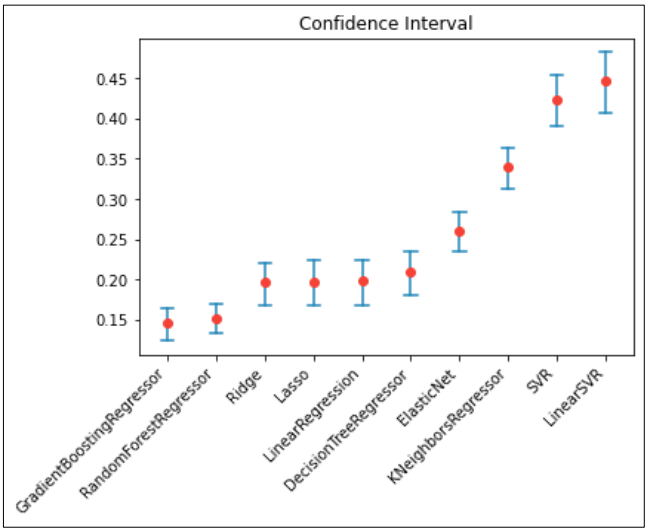


Fig. 5. RMSE confidence interval for each model at CoCoeff=0.95.

E. Select the best Model

As the Gradient boosting regressor score the best values on all performance evaluation metrics, so we performed this regressor as the best model.

Boosting (originally called hypothesis boosting) refers to any ensemble method that can combine several weak learners into a strong learner. Fig. 6 shows the general idea of most boosting methods, which is to train predictors sequentially, each trying to correct its predecessor. There are many boosting methods available, but by far the most popular are AdaBoost (short for adaptive boosting) and gradient boosting [5].

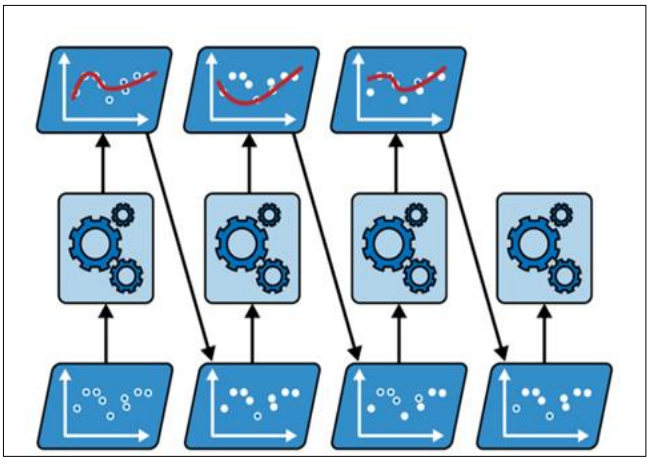


Fig. 6. Boosting Regression Ensemble Model [5].

Gradient boosting works by sequentially adding predictors to an ensemble, each one correcting its predecessor. However, this method tries to fit the new predictor to the residual errors made by the previous predictor [5].

Fig. 7 below represents how GBR works and how it adjust the predictions to reduce the loss. the error predictions are in the left column, and the ensemble's predictions in the right column. In the first row, the ensemble has just one estimator, so its predictions are exactly the same as the first estimator predictions. In the second row, a new estimator is trained on the residual errors of the first estimator. On the right you can see that the ensemble's predictions are equal to the sum of the predictions of the first two estimators. Similarly, in the third row another estimator is trained on the residual errors of the second tree. the ensemble's predictions gradually get better as estimators are added to the ensemble [5].

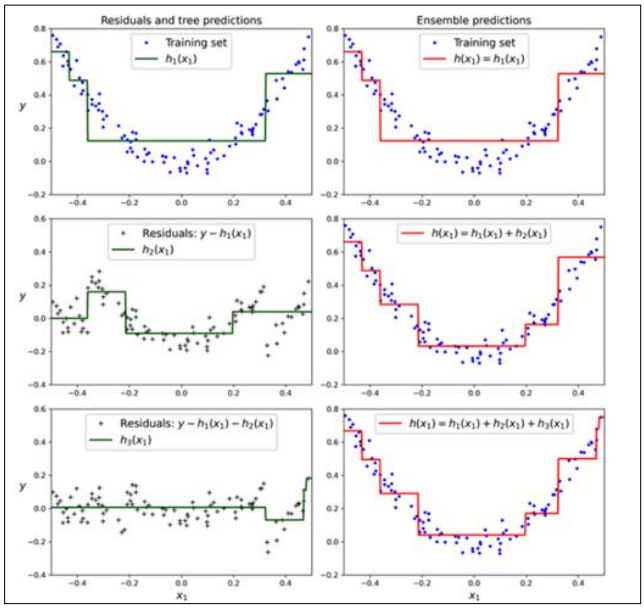


Fig. 7. Adjusting Predictions by Gradient Boosting Regression [5].

F. Fine Tuning the Hyper Parameters of the Best Model

The GBR is a very powerful model and it can promise for a high performance for complex regression problems, however it has a lot of hyper parameters to be tuned. Here we use GridSearchCV from Scikit learn to find the best hyper parameters that can best fit the data set and enhance the model performance.

The n_iter_no_change hyper parameter is adjust to an integer value= 10, this help the Gradient Boosting Regressor to automatically stop adding more trees during training if it sees that the last 10 estimators didn't help. This is simply early stopping, but with a little bit of patience: it tolerates having no progress for a few iterations before it stops [5]

When n_iter_no_change is set, the fit() method automatically splits the training set into a smaller training set and a validation set: this allows it to evaluate the model's performance each time it adds a new estimator. The size of the validation set is controlled by the validation_fraction hyper parameter, which is 10% by default. The tol hyper parameter determines the maximum performance improvement that still counts as negligible. It sets to defaults 0.0001. The Gradient Boosting Regressor class also supports a subsample hyper parameter, which specifies the fraction of training instances to be used for training each tree. If subsample=0.25, then each tree is trained on 25% of the training instances, selected randomly [5].

The best values of hyper parameters found by the GridSearchCv, which give the highest performance, are listed in the following table:

TABLE III. GBR BEST HYPER PARAMETERS VALUES.

	best value
GradientBoostingRegressor__alpha	0.001
GradientBoostingRegressor__criterion	squared_error
GradientBoostingRegressor__learning_rate	0.05
GradientBoostingRegressor__max_depth	4
GradientBoostingRegressor__max_features	6
GradientBoostingRegressor__min_samples_split	4
GradientBoostingRegressor__subsample	0.5
GradientBoostingRegressor__n_estimators_	131

The best GBR model (with the best hyper parameters values found) was fitted using the whole training, then the model is tested on all the evaluation metrics, the result shows an enhance on the performance of the model, the RMSE decrease to (13.4%) and the R2_score (90%)

Also the feature Importance is calculated to find the most significant features that affect the bike counts predictions.

Fig. 8 shows the feature importance ordered from the most significant feature to the least one.

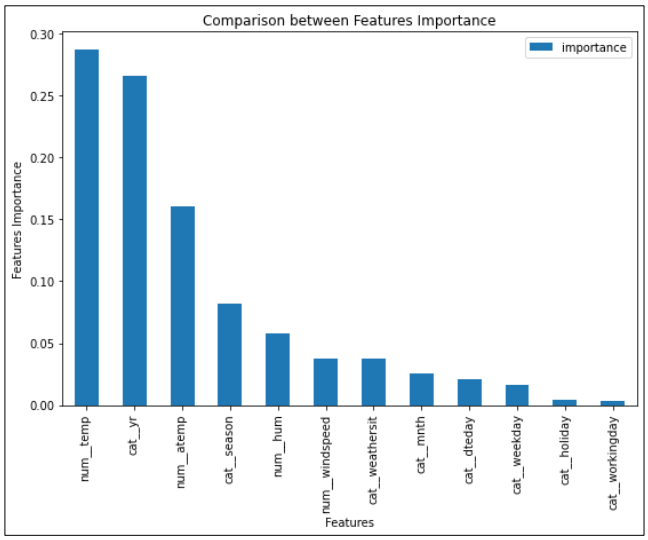


Fig. 8. Features importance for all features.

This support our first insights taken from the data exploration that the most important features that affect the number of count is year and temperature and the least important is holiday and working day features, which make sense.

The learning curves for RMSE and R2 Score is also plotted to find show how the model performance is enhanced by enlarging the data set sample, Fig. 9 shows the RMSE and R2_score learning curves respectively. The learning curves that the performance of the model enhanced in both the training and validation set while the data samples increased.

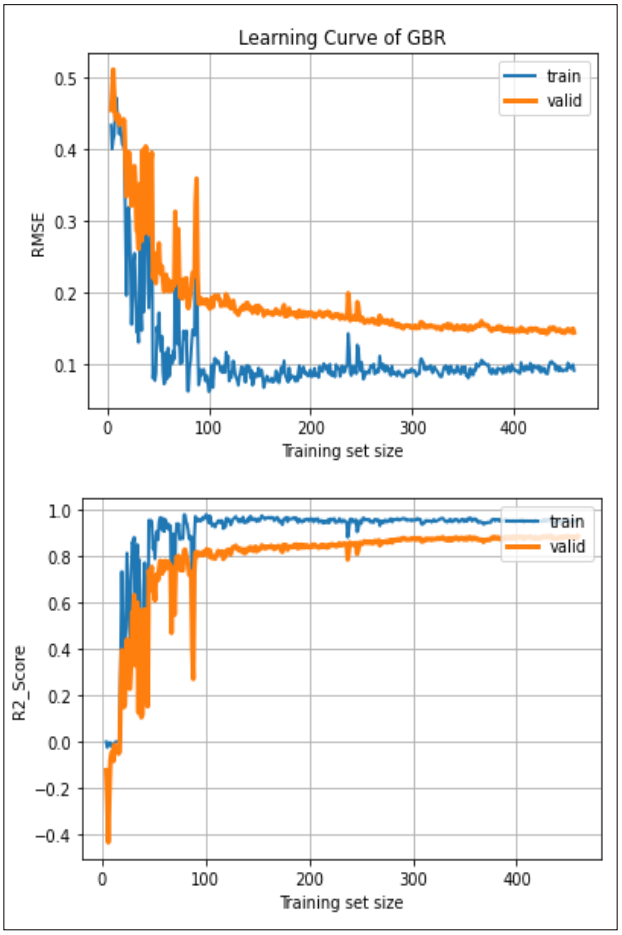


Fig. 9. Learning curves of the GBR model on train and validation set.

V. ENHANCE THE THE MODEL PERFORMANCE

As the performance of the model is increasing by enlarging the data set here we tried to enhance the model by applying data augmentation technique to enlarge the dataset.

A. *Applying Data Augmentation:*

Synthetic Minority Over-sampling Technique (SMOTE) [7] is a very powerful technique that can be used to generate data in a smart way for classification problems (Fig. 10). For regression tasks, where the target variable is continuous, SmoteR [8] method is commonly used, it can be used with any existing regression algorithm turning it into a general tool for addressing problems of forecasting rare extreme values of a continuous target variable.

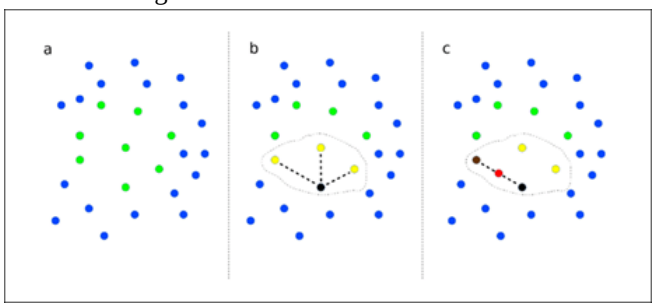


Fig. 10. Synthetic Minority Over-Sampling Technique [9].

In this work we used SMOGN, for tackling imbalanced regression problem. It's a pre-processing method that has the advantage of being versatile because it allows the use of any standard learning algorithm. The method tries to overcome difficulties in SmoteR strategy and in the introduction of Gaussian Noise strategy. It uses the interpolation method of SmoteR for interpolating examples that are closer. This way we tried to eliminate the risk of interpolating examples that, although being among the nearest neighbors of the seed example, are too distant. On the other hand, the use of SmoteR allows to expand the decision boundaries which is only achieved in a limited way with the more conservative method that generates synthetic cases by introducing Gaussian Noise. Fig. 11 shows the synthetic example of the application of SMOGN algorithm. [10]

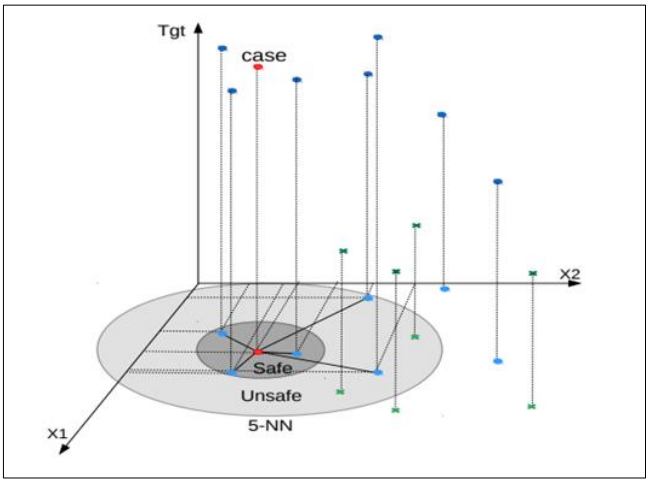


Fig. 11. Synthetic example of the application of SMOGN algorithm [10].

We tuned the model hyper parameter in the enlarged data set by using GridSearchCV and the result of best parameter shown in the table below.

Here we can easily notice that values of the best hyper parameters changed when enlarging the data set to best fit the new data generated, since the hyper parameters is very sensitive to the diversity and distribution of the data used.

B. *A-B Testing and Results*

The new tuned model is tested and evaluated using (RMSE, MSE, and R2) and also the RMSE confidence interval is computed.

The result shows enhancement in the overall performances of the model, TABLE IV. shows a comparison between the system with and without SMOGN.

TABLE IV. PERFOMANCE METRICS OF BOTH GBR MODELS (WITH AND WITHOUT SMOTE).

	Model	R2	RMSE	C1	C2	Confidence Interval
0	GB_Model_With_SMOTE	0.954489	0.078629	0.070195	0.086243	0.016048
1	GB_Model_Without_SMOTE	0.901730	0.131584	0.112897	0.147916	0.035020

By plotting the confidence interval levels for both models Fig. 12, at confidence Coeff=0.95 it can easily noticed the enhancement in performance of the model after using SMOTE.

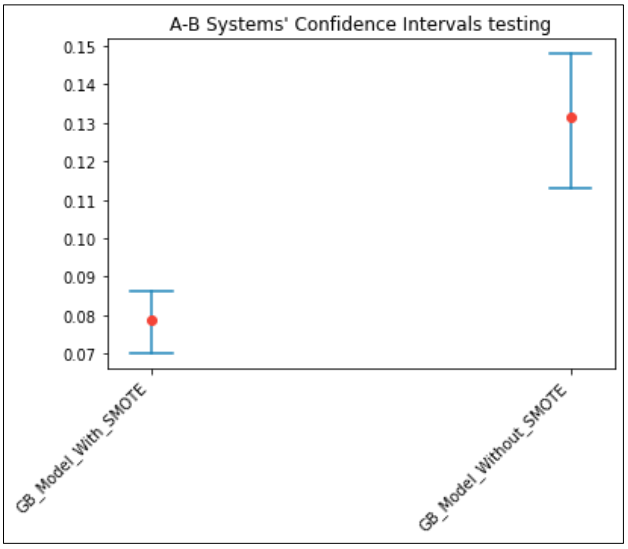


Fig. 12. RMSE confidence intervals of both GBR models (with and without applying SMOTE) with 0.95 confidence Coef.

As shown in Fig. 12 the confidence interval is become lower and shorter which make sense, because the data samples become larger and also the model able to fit the data perfectly, which also can be approved by plot the actuals vs. predictions on the test data set samples for both models, which is represented in Fig. 13 & Fig. 14 below.

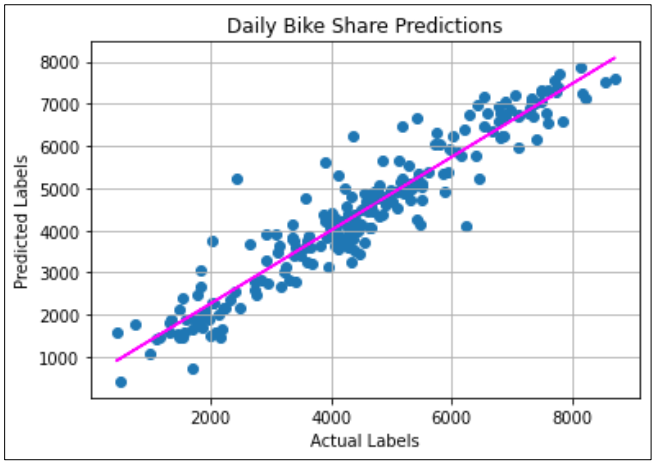


Fig. 13. Predictions Vs. actual labels of GBR model without SMOTE.

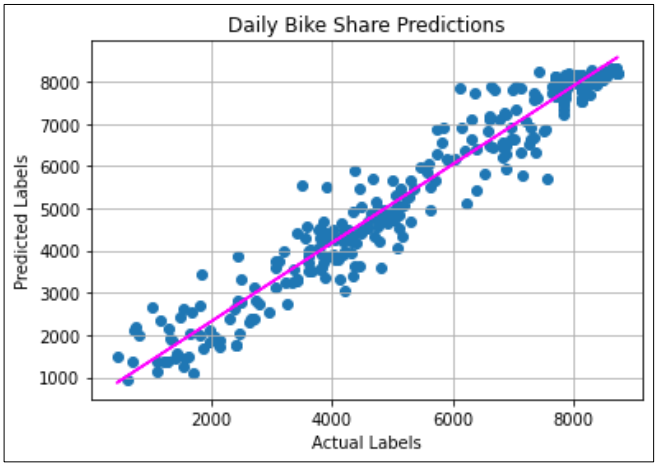


Fig. 14. Predictions Vs. actual labels of GBR model with SMOTE.

VI. Conclusion

In this project, we use a real rental-bike-sharing system data concatenated with the corresponding weather data to study the performance of different ML algorithms in such a problem, and to derive a model that is able to precisely and automatically predict the daily count of rental bikes depending on some input features.

The features that we used to predict bike demand were exclusively weather conditions, ranging from humidity to wind speed and temperature. We aimed to predict bike demand by extending the scope of features on a daily basis. Indeed, some studies have shown that the geography of bike-docking stations has impacts on bike demand, which has a lot to do with social and economic situations in which these stations are located.

We evaluate different ML algorithms (linear regression, Lasso, Ridge, SVR, Linear SVR, Decision tree, random forest, and Gradient Boosting Regressor among different evaluation metrics (RMSE, MSE, R2Score, and confidence intervals). Then, we make a comparison between the nine algorithms. The Random Forest and Gradient Boosting Regression showed a very good results in fit the data and make a good predictions RMSE = (15%), (14.5%) respectively. Where the SVR and Linear SVR were unable to fit the data well and return the worst results. We choose Gradient Boosting regressor since it give us the lowest RMSE value on test set plus the highest R2-Score value (88)%. We used both grid Search and randomized search to fine tune the model to find the best hyper parameters that enhance the performance of the model. We also study the effect of using data augmentation technique by enlarging the data set samples by using SMOGN technique, and how it's significantly enhance the performance of the model. In addition to apply A_B testing to compare both models, which shows enhancement in the prediction output with respect to the actual values.

As a conclusion, there is different factors can affect the performance of the prediction system starting from the complexity of the problem, to the data set which have to be representative, informative, cleaned, processed properly, has a meaning full features, and large enough, to the best model chosen, and finally the hyper parameters' tuning. All these factors must be considered carefully and correctly to reach to the best performance predictor.

As shown in this study, the rental bike sharing system is highly correlated to the weather conditions especially for (temperature degree and the heavy rain conditions in addition to wind speed). These factors may affect the demand on bike renting by users.

As a future work we can try to predict the registered count and the casual count separately and engaged the data set with more features that is affect the rental counts, such as geographical typography, traffic information, and social economic factors which could make the prediction reflects the real world scenarios and thus enhancing the performance results, we can also try to use MLP regression model which may give better results. Also, applying multi label regression to predict the rental bikes for casual and registered users may give us more insights about the behavior of both in different situations and scenarios, this may help to adjust some features in the rental bike systems to fit the different users' requirements.

VII. ACKNOWLEDGMENT

This work done with support and encourage from Prof.Dr.Eng. Gheith Abandah, University of Jordan, Amman/Jordan.

VIII. REFERENCES

[1] S. C. Kang, "Learning to predict demand in a transport-resource sharing task," Calhoun, Naval Postgraduate School, Monterey, California, 2015.

[2] C. C. Y. Gao, "Using Machine Learning Methods to Predict Demand for Bike Sharing," Information and Communication Technologies in Tourism 202, pp. 282--296, 2022.

[3] H.-C. a. L. Z.-Q. Lu, "Rental Prediction in Bicycle-Sharing System Using Recurrent Neural Network," IEEE Access, pp. 1-1, 05 2020.

[4] "Kaggle," Laboratory of Artificial Intelligence and Decision Support (LIAAD), University of Porto, [Online]. Available: https://www.kaggle.com/datasets/imakash3011/rental-bike-sharing?datasetId=1541302&sor. [Accessed 20 Dec. 2022].

[5] A. Géron, Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow, United States of America: y O'Reilly Media, Inc., 1005 Gravenstein Highway North,, 2023.

[6] A. Géron, "handson-ml3," 20 Oct 2022. [Online].Available: https://github.com/ageron/handson-ml3. [Accessed 05 Jan 2023].

[7] N. V. Chawla , K. W. Bowyer, L. O. Hall and W. P. Kegelmeyer, "SMOTE: Synthetic Minority Over-sampling Technique," Journal of Artificial Intelligence Research, vol. 16, pp. 321--357, 2002.

[8] L. Torgo, R. Ribeiro and B. Pfahringer, "SMOTE for Regression," 2013.

[9] M. Schubach, M. Re, P. Robinson and G. Valentini, "Imbalance-Aware Machine Learning for Predicting Rare and Common Disease-Associated Non-Coding Variants," Scientific Reports, vol. 7, 2017.

[10] P. Branco, L. Torgo and R. Ribeiro, "SMOGN: A Pre-Processing Approach for Imbalanced Regression. Proceedings of Machine Learning Research," vol. 74, pp. 36-50, 2017.

IX. APPENDIX: SOURCE CODE

In [ ]:
```
import pandas as pd
import numpy as np
np.random.seed(42)
bikes =pd.read_csv("day.csv")
import dataframe_image as dfi

dfi.export(bikes.head(), "SampleOfDataset.png")
bikes.head()
```

Out[ ]:

	instant	dteday	season	yr	mnth	holiday	weekday	workingday	weathersit	temp	atemp	hum	windspeed	casual	register
0	1	2011-01-01	1	0	1	0	6	0	2	0.344167	0.363625	0.805833	0.160446	331	6
1	2	2011-01-02	1	0	1	0	0	0	2	0.363478	0.353739	0.696087	0.248539	131	6
2	3	2011-01-03	1	0	1	0	1	1	1	0.196364	0.189405	0.437273	0.248309	120	12
3	4	2011-01-04	1	0	1	0	2	1	1	0.200000	0.212122	0.590435	0.160296	108	14
4	5	2011-01-05	1	0	1	0	3	1	1	0.226957	0.229270	0.436957	0.186900	82	15

In [ ]:
```
bikes.drop(columns=['instant','casual','registered'],inplace=True)
bikes.head()
```

Out[ ]:

	dteday	season	yr	mnth	holiday	weekday	workingday	weathersit	temp	atemp	hum	windspeed	cnt
0	2011-01-01	1	0	1	0	6	0	2	0.344167	0.363625	0.805833	0.160446	985
1	2011-01-02	1	0	1	0	0	0	2	0.363478	0.353739	0.696087	0.248539	801
2	2011-01-03	1	0	1	0	1	1	1	0.196364	0.189405	0.437273	0.248309	1349
3	2011-01-04	1	0	1	0	2	1	1	0.200000	0.212122	0.590435	0.160296	1562
4	2011-01-05	1	0	1	0	3	1	1	0.226957	0.229270	0.436957	0.186900	1600

In [ ]:
```
type(bikes['dteday'][0])
```

Out[ ]:
```
str
```

In [ ]:
```
bikes['dteday']=pd.DatetimeIndex(bikes['dteday']).day
type(bikes['dteday'][0])
```

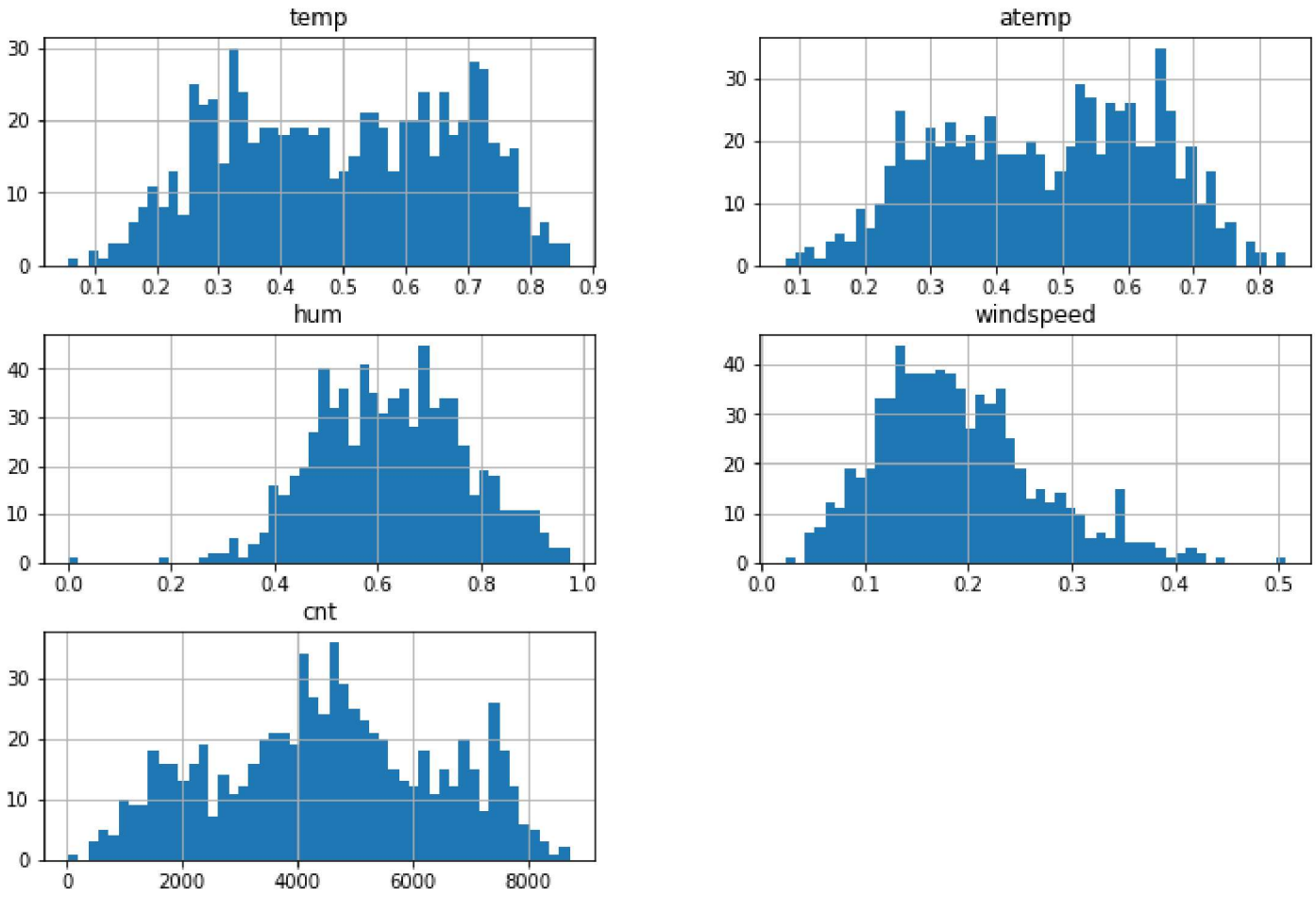
Out[ ]:
```
numpy.int64
```

In [ ]:
```
bikes.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 731 entries, 0 to 730
Data columns (total 13 columns):
 #   Column      Non-Null Count  Dtype  
---  ------      --------------  -----  
 0   dteday      731 non-null    int64  
 1   season      731 non-null    int64  
 2   yr          731 non-null    int64  
 3   mnth        731 non-null    int64  
 4   holiday     731 non-null    int64  
 5   weekday     731 non-null    int64  
 6   workingday  731 non-null    int64  
 7   weathersit  731 non-null    int64  
 8   temp        731 non-null    float64
 9   atemp       731 non-null    float64
 10  hum         731 non-null    float64
 11  windspeed   731 non-null    float64
 12  cnt         731 non-null    int64  
dtypes: float64(4), int64(9)
memory usage: 74.4 KB
```

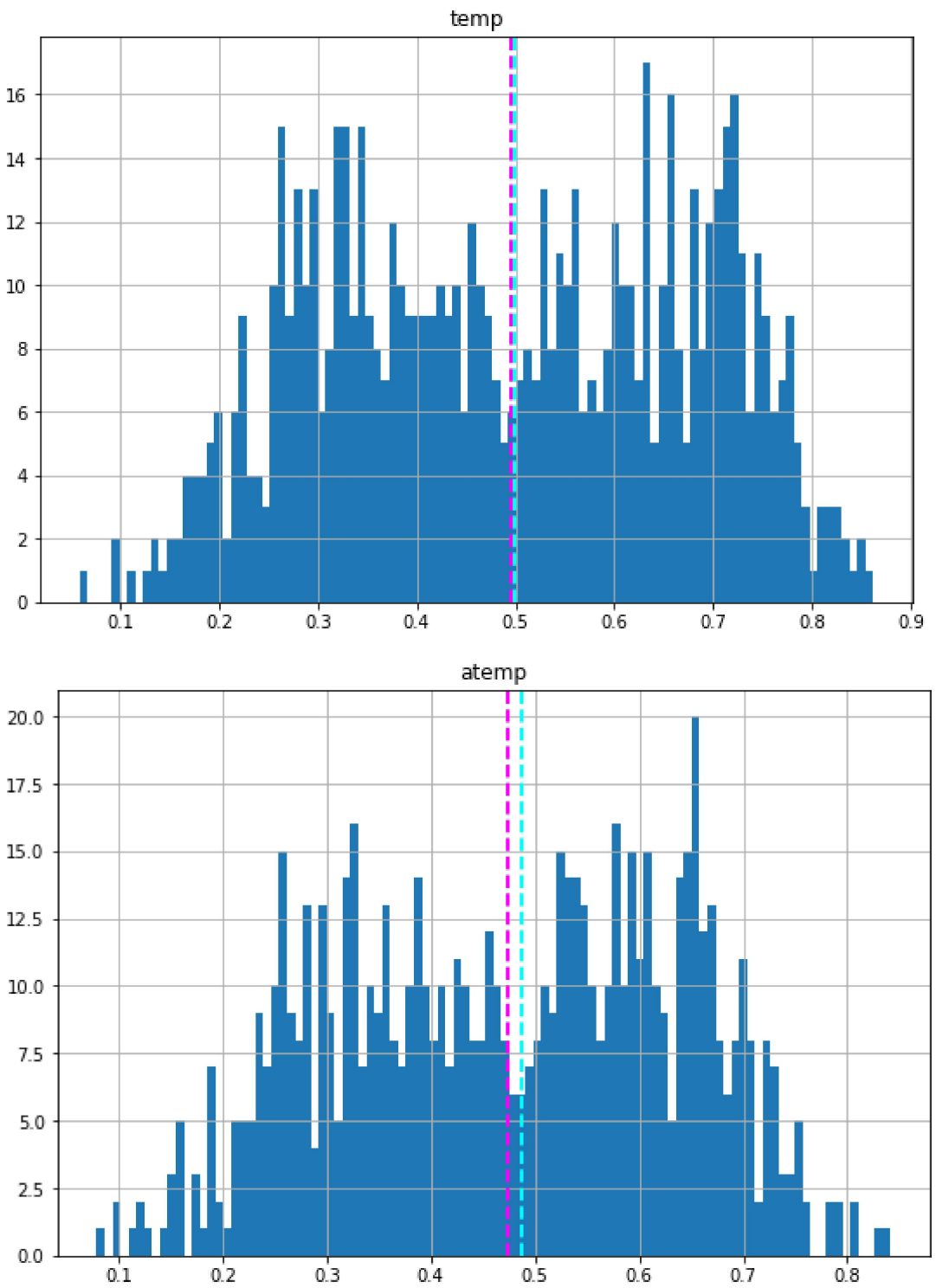
In [ ]:
```
import matplotlib.pyplot as plt

num_attributes=['temp','atemp','hum','windspeed']
cat_attributes=['dteday','season','yr', 'mnth', 'holiday',      'weekday',      'workingday',   'weathersit']
bikes.hist(bins=50,figsize=(12,8),column=['temp','atemp','hum','windspeed','cnt'])
```

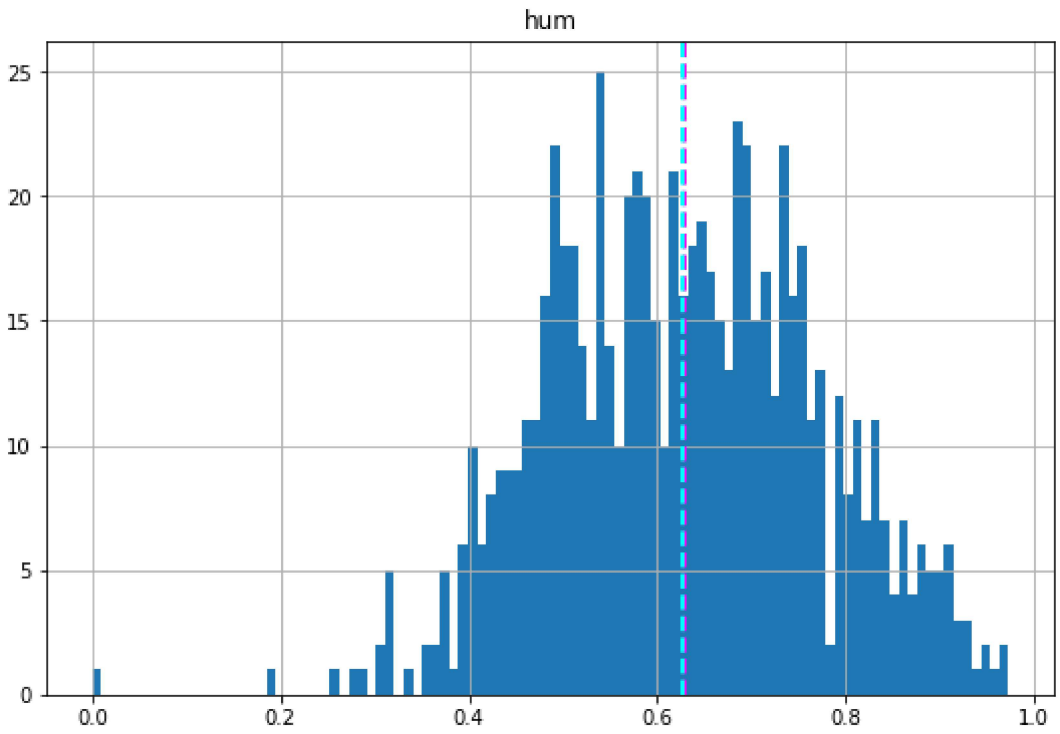
Out[ ]:
```
array([[<AxesSubplot:title={'center':'temp'}>,
        <AxesSubplot:title={'center':'atemp'}>],
       [<AxesSubplot:title={'center':'hum'}>,
        <AxesSubplot:title={'center':'windspeed'}>],
       [<AxesSubplot:title={'center':'cnt'}>, <AxesSubplot:>]],
      dtype=object)
```

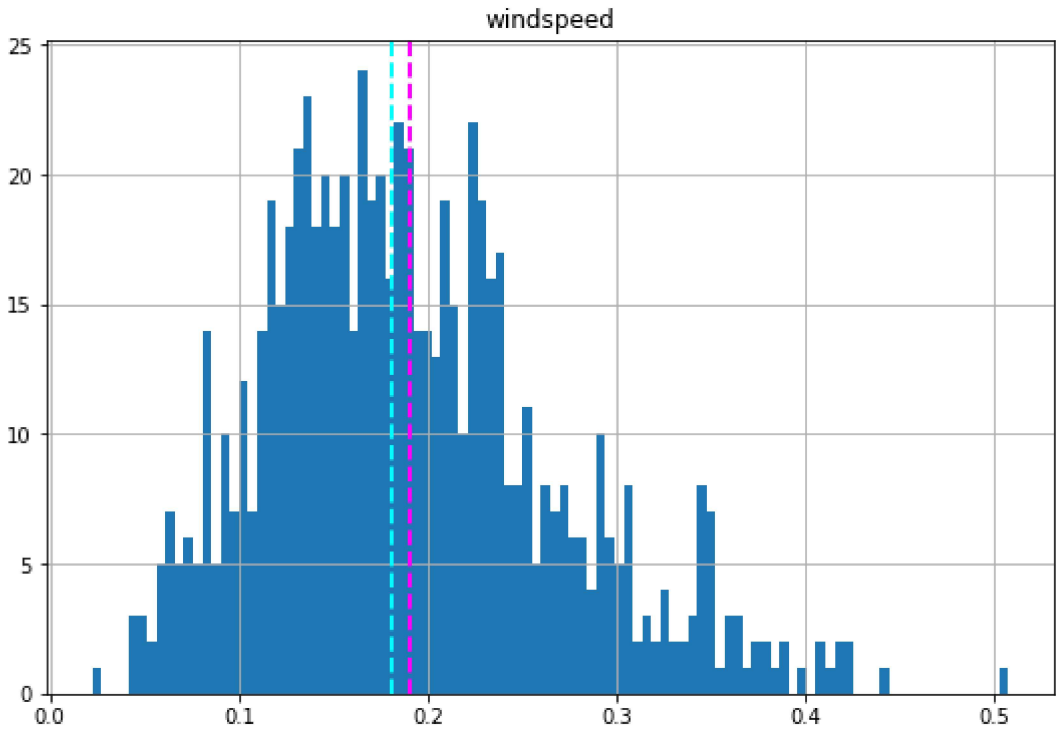


In [ ]:
```
#bikes.drop(bikes[bikes['windspeed']>.5].index,inplace=True)
#bikes.drop(bikes[bikes['hum']<.2].index,inplace=True)
```

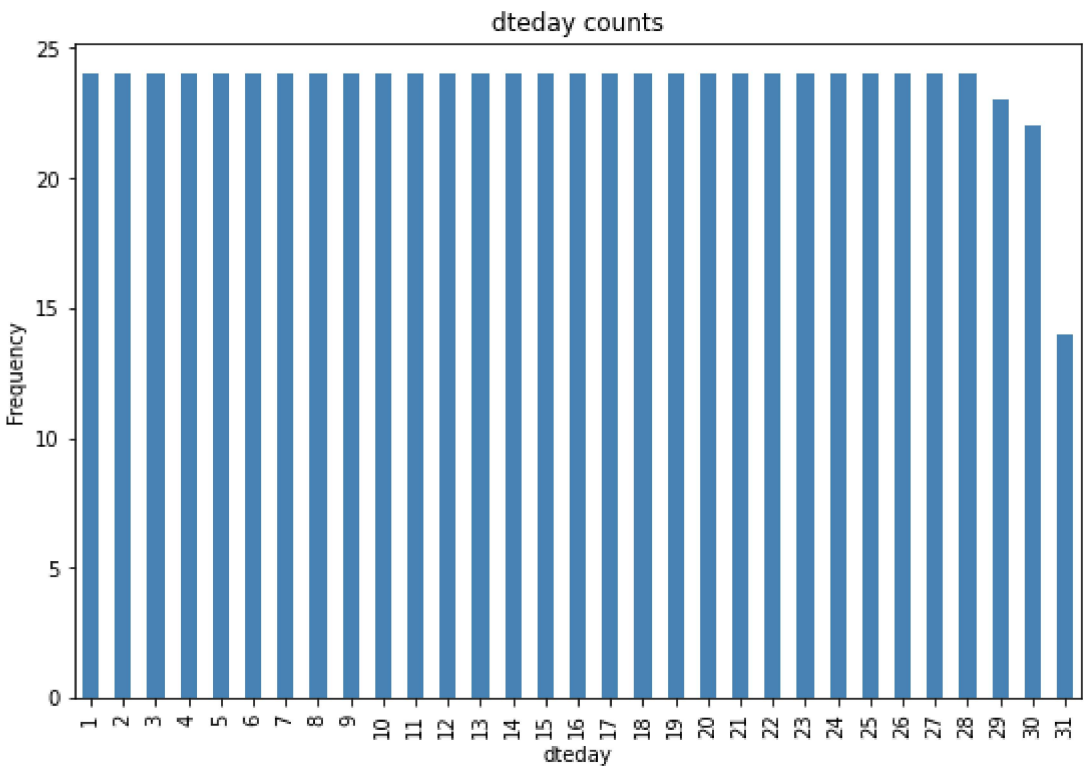
In [ ]:
```
# Plot a histogram for each numeric feature
for col in num_attributes:
    fig = plt.figure(figsize=(9, 6))
    ax = fig.gca()
    feature = bikes[col]
    feature.hist(bins=100, ax = ax)
    ax.axvline(feature.mean(), color='magenta', linestyle='dashed', linewidth=2)
    ax.axvline(feature.median(), color='cyan', linestyle='dashed', linewidth=2)
    ax.set_title(col)
plt.show()
```

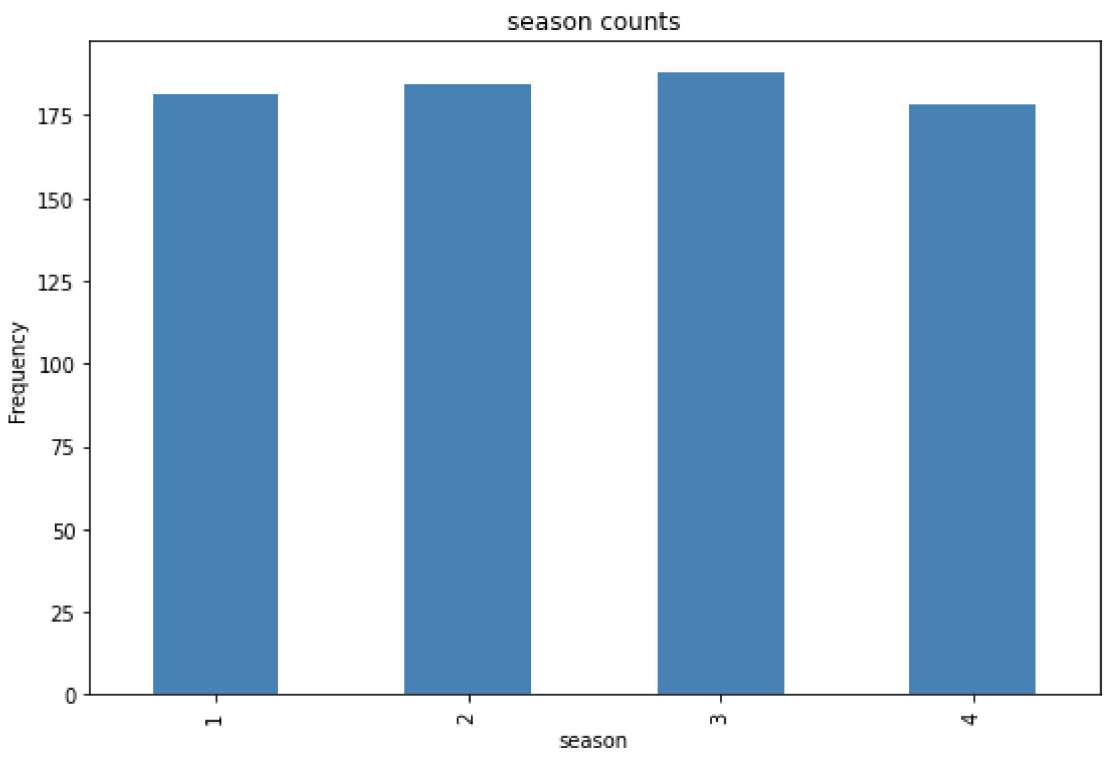


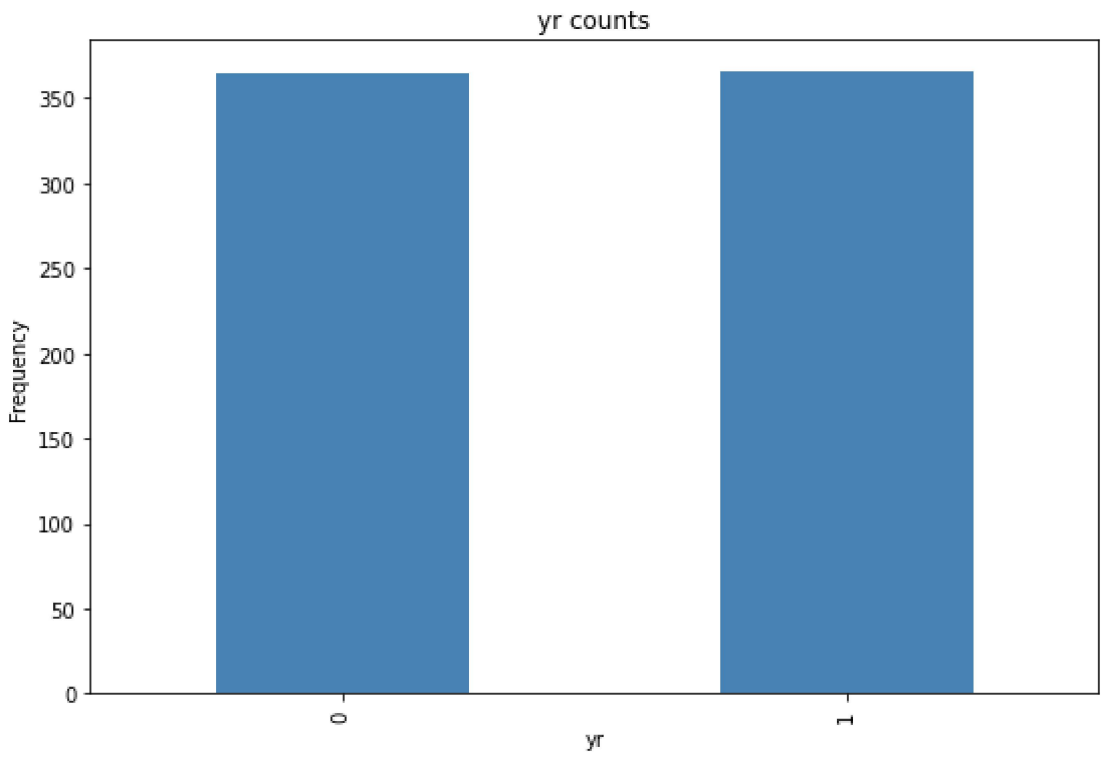


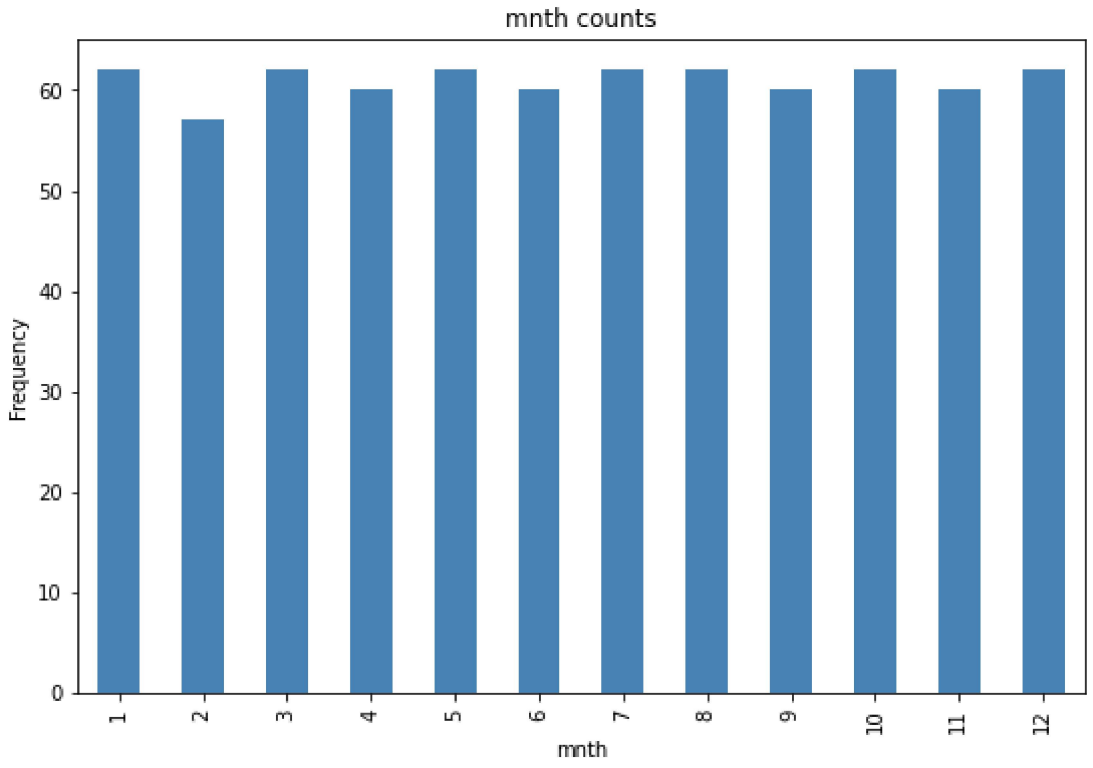


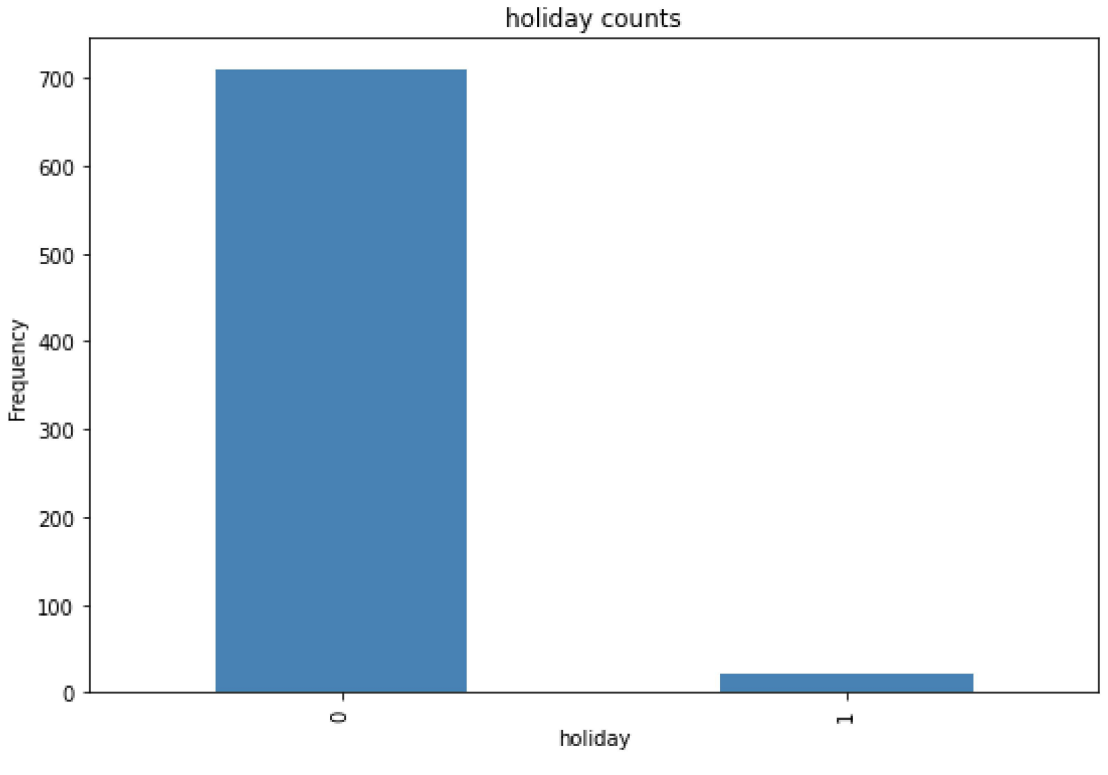
In [ ]:
```
for col in cat_attributes:
    counts = bikes[col].value_counts().sort_index()
    fig = plt.figure(figsize=(9, 6))
    ax = fig.gca()
    counts.plot.bar(ax = ax, color='steelblue')
    ax.set_title(col + ' counts')
    ax.set_xlabel(col)
    ax.set_ylabel("Frequency")
plt.show()
```

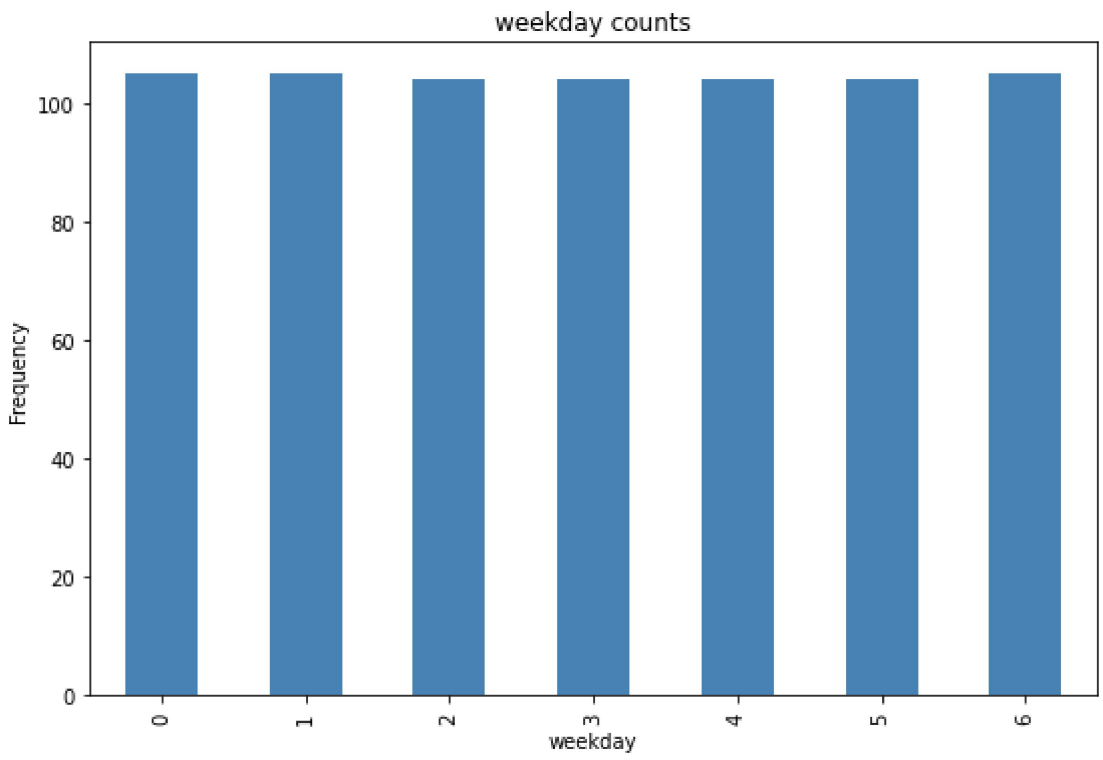


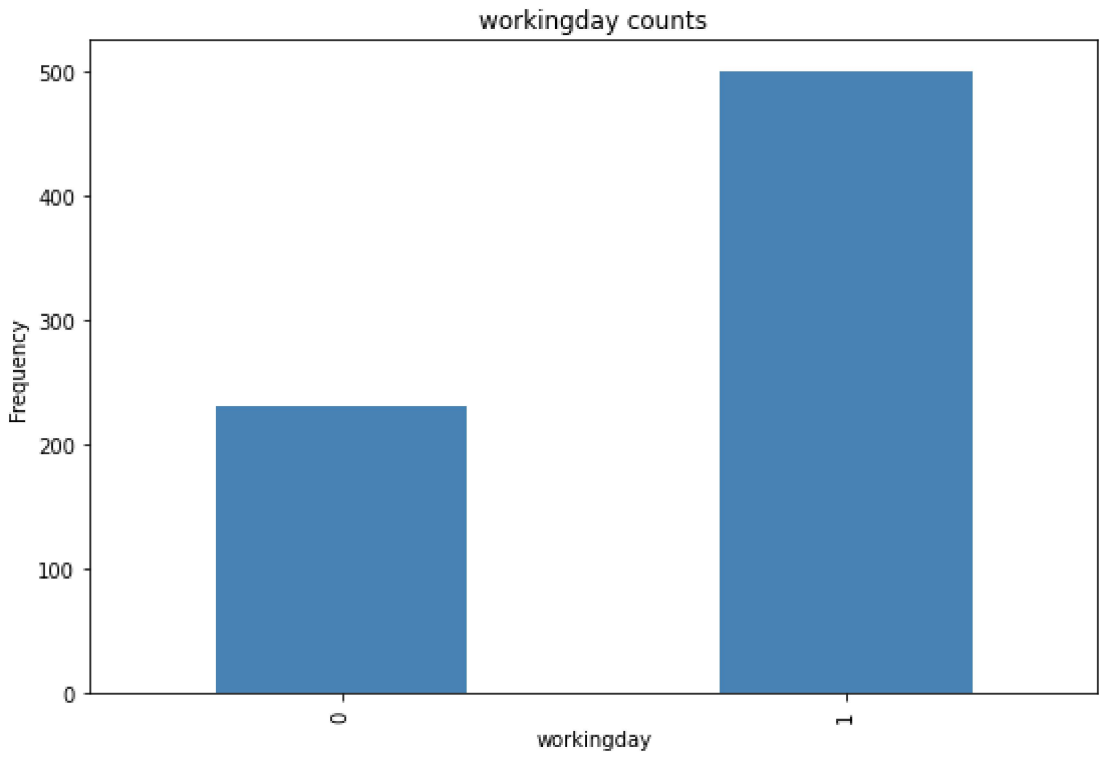

season counts
Frequency
175
150
125
100
75
50
25
0
1
2
3
4
season

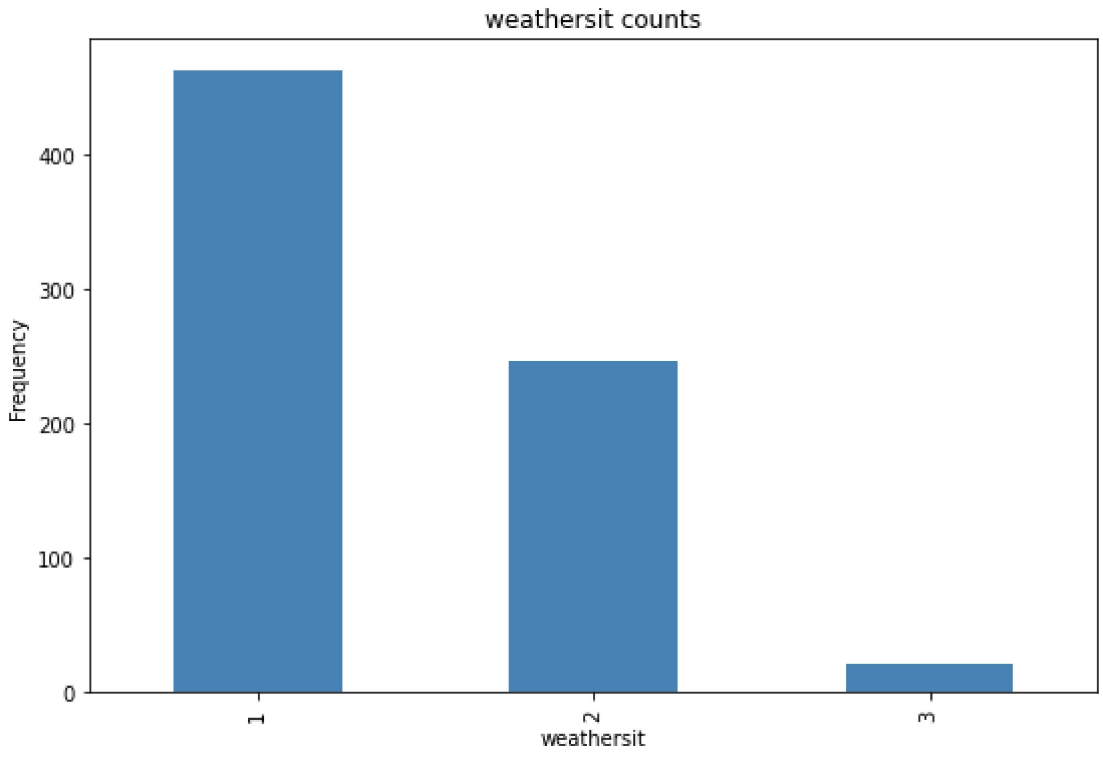

yr counts
Frequency
350
300
250
200
150
100
50
0
0
1
yr


mnth counts
Frequency
60
50
40
30
20
10
0
1
2
3
4
5
6
7
8
9
10
11
12
mnth


holiday counts
Frequency
700
600
500
400
300
200
100
0
0
1
holiday







In [ ]:
```
duplication=bikes.duplicated().value_counts()
#duplication_sum=data.data.duplicated().sum()
duplication
```

Out[ ]:
```
False    731
dtype: int64
```

In [ ]:
```
bikes["weathersit"].value_counts()
```

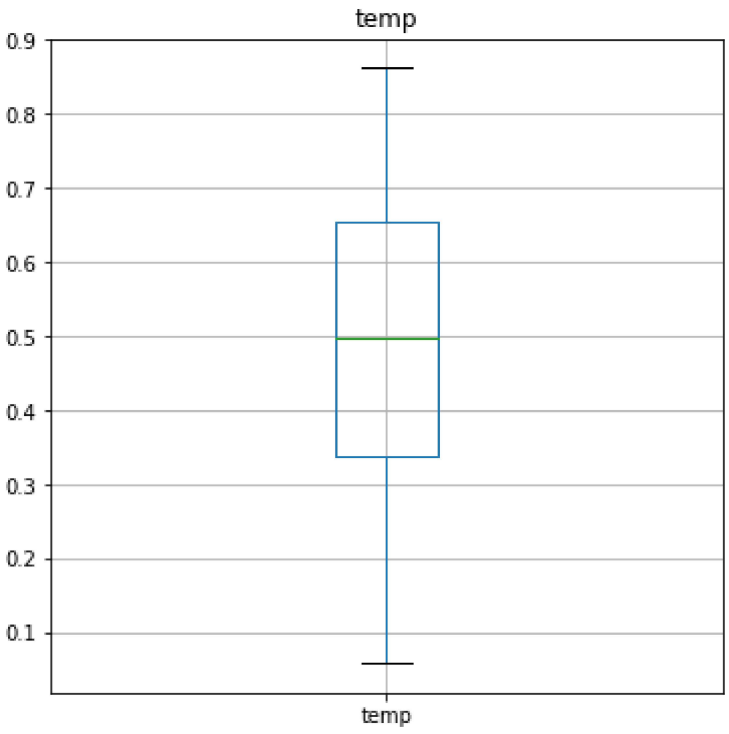
Out[ ]:
```
1    463
2    247
3     21
Name: weathersit, dtype: int64
```

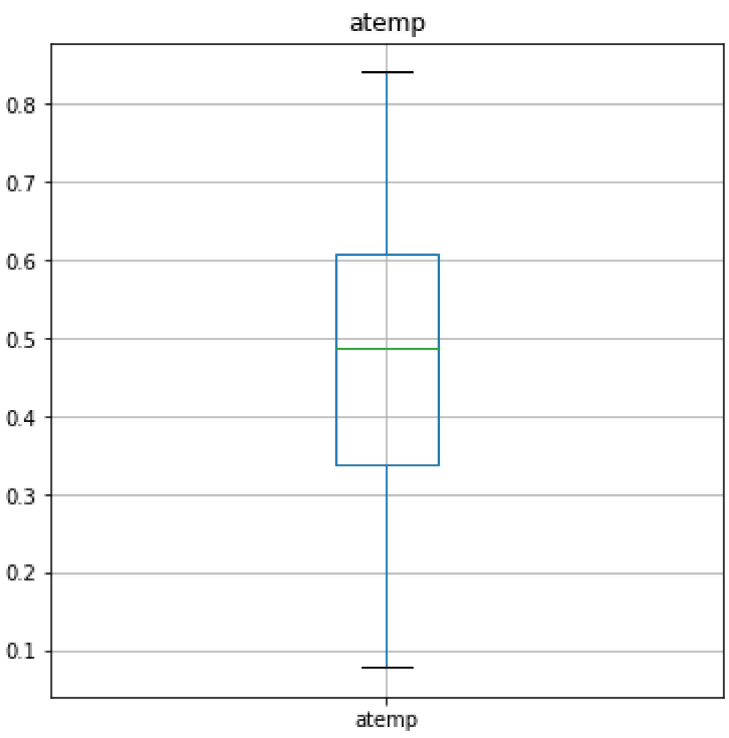
In [ ]:
```
statistics=bikes[num_attributes+['cnt']].describe()
import dataframe_image as dfi

dfi.export(statistics, "statistics.png")
statistics
```

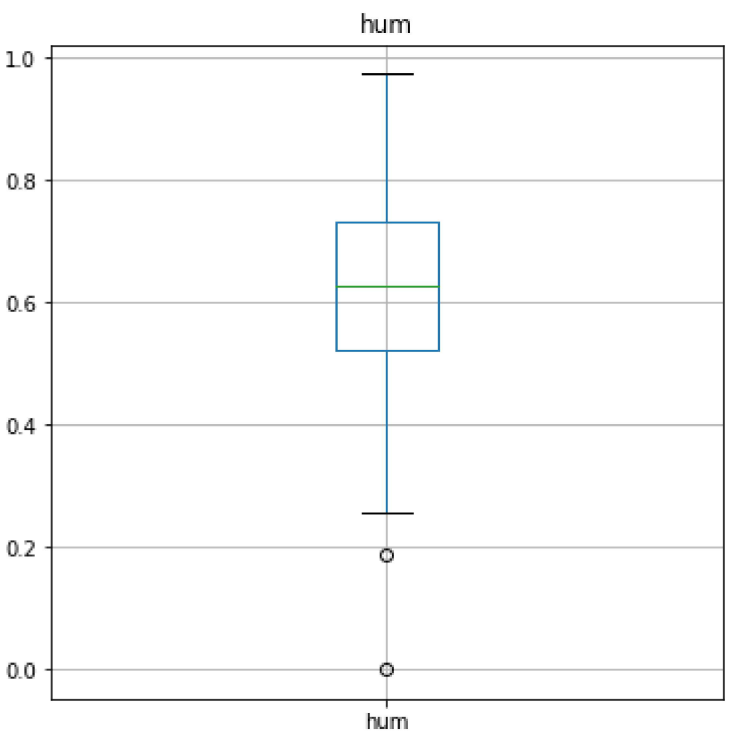
Out[ ]:

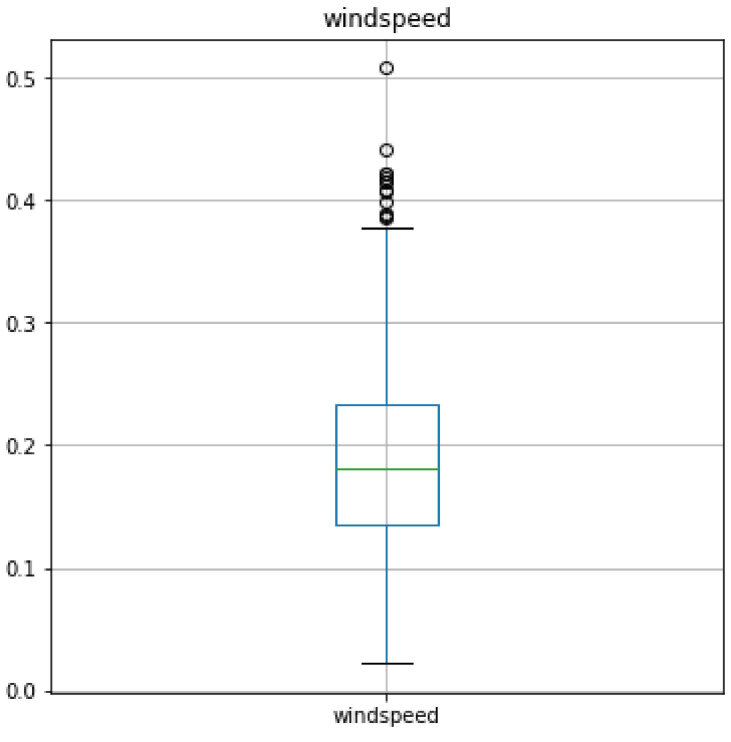
	temp	atemp	hum	windspeed	cnt
count	731.000000	731.000000	731.000000	731.000000	731.000000
mean	0.495385	0.474354	0.627894	0.190486	4504.348837
std	0.183051	0.162961	0.142429	0.077498	1937.211452
min	0.059130	0.079070	0.000000	0.022392	22.000000
25%	0.337083	0.337842	0.520000	0.134950	3152.000000
50%	0.498333	0.486733	0.626667	0.180975	4548.000000
75%	0.655417	0.608602	0.730209	0.233214	5956.000000
max	0.861667	0.840896	0.972500	0.507463	8714.000000

In [ ]:
```
#Check Outliers in numarical features by using boxplot :
from matplotlib import pyplot as plt

type(num_attributes)
for col in num_attributes:
    bikes.boxplot(column=col, figsize=(6,6))
    plt.title(col)
    plt.show()
```

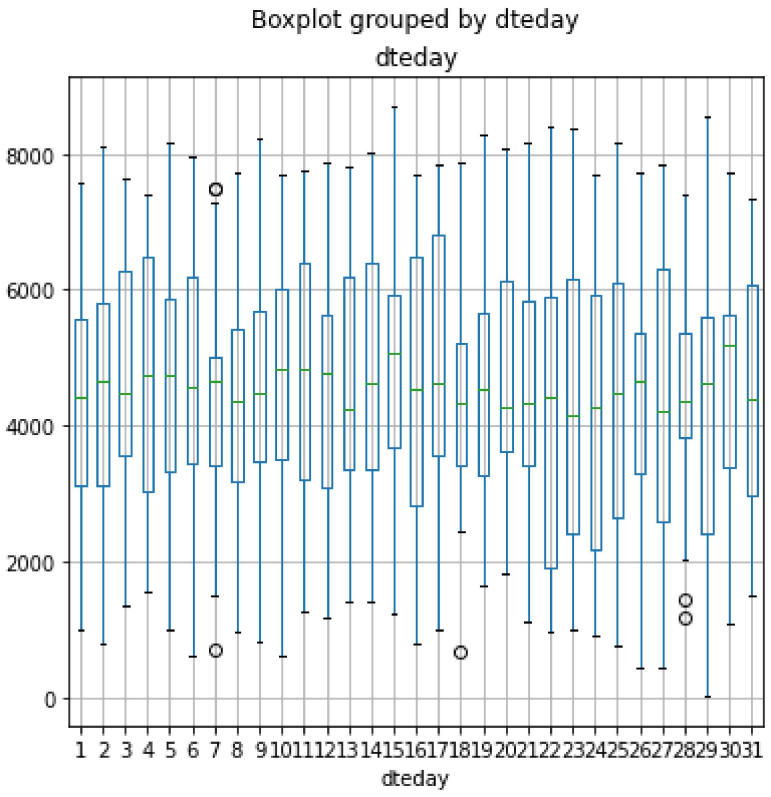


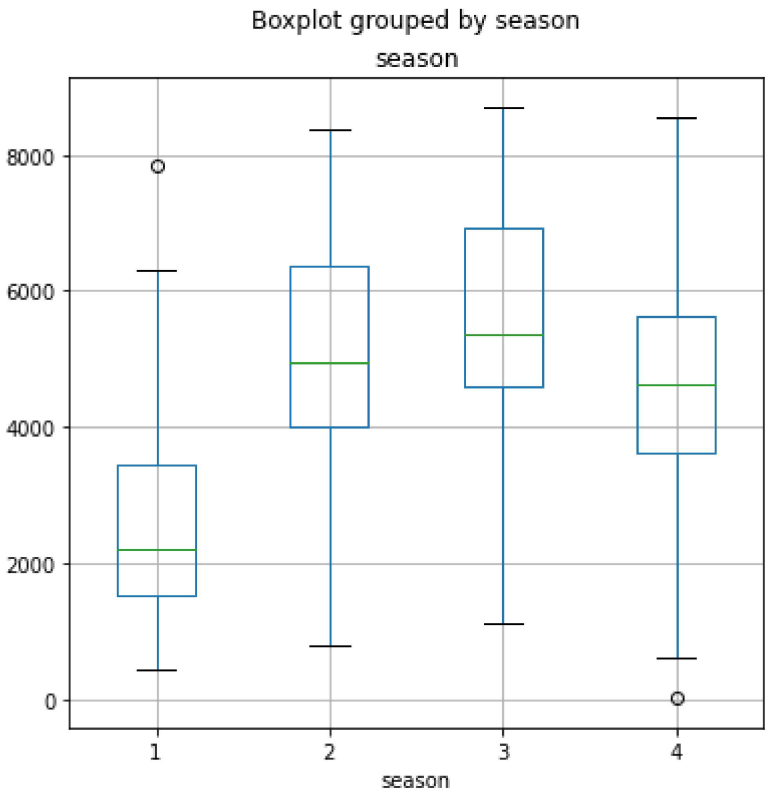


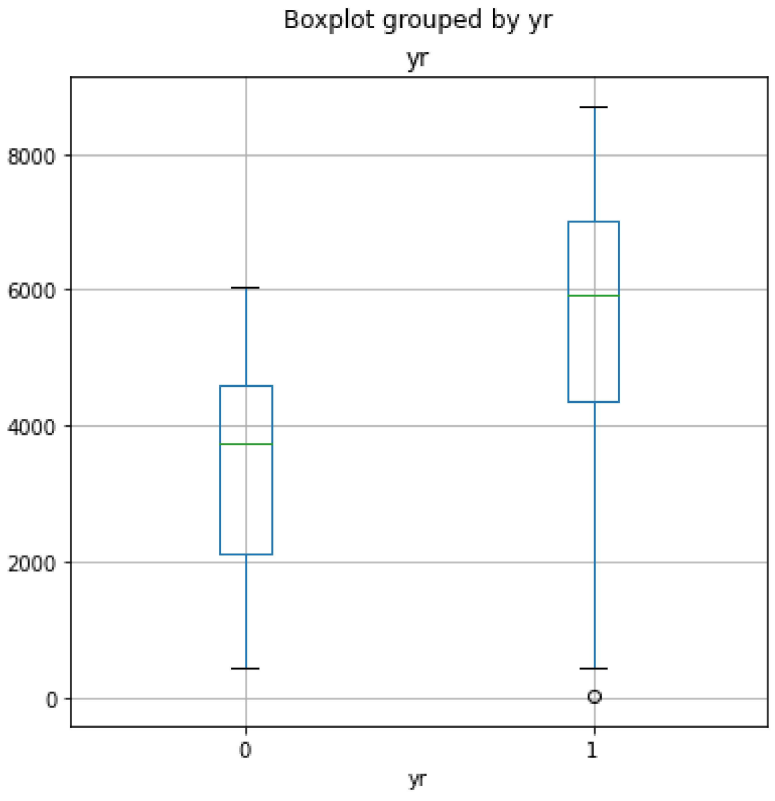


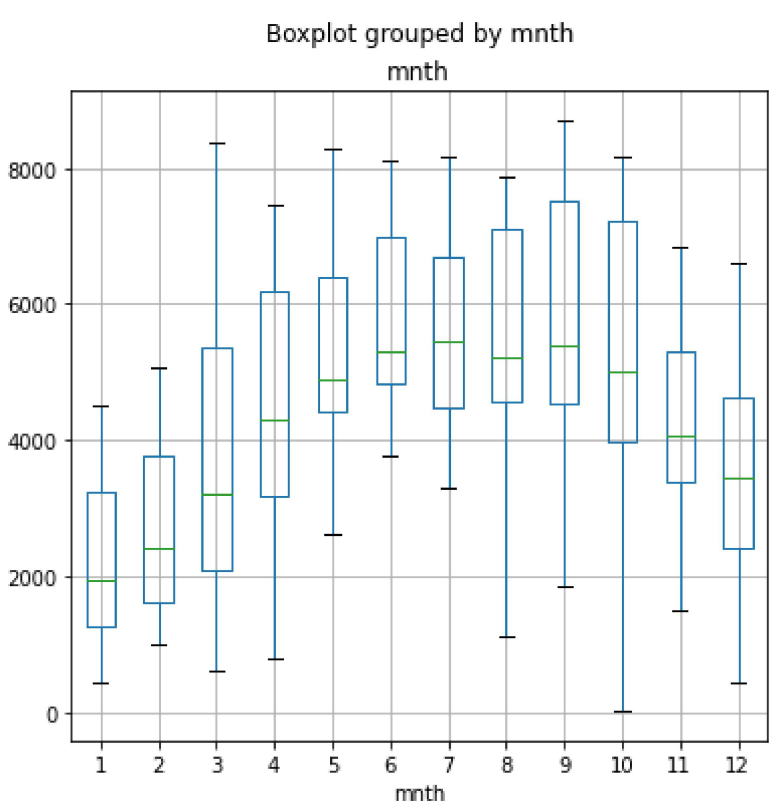


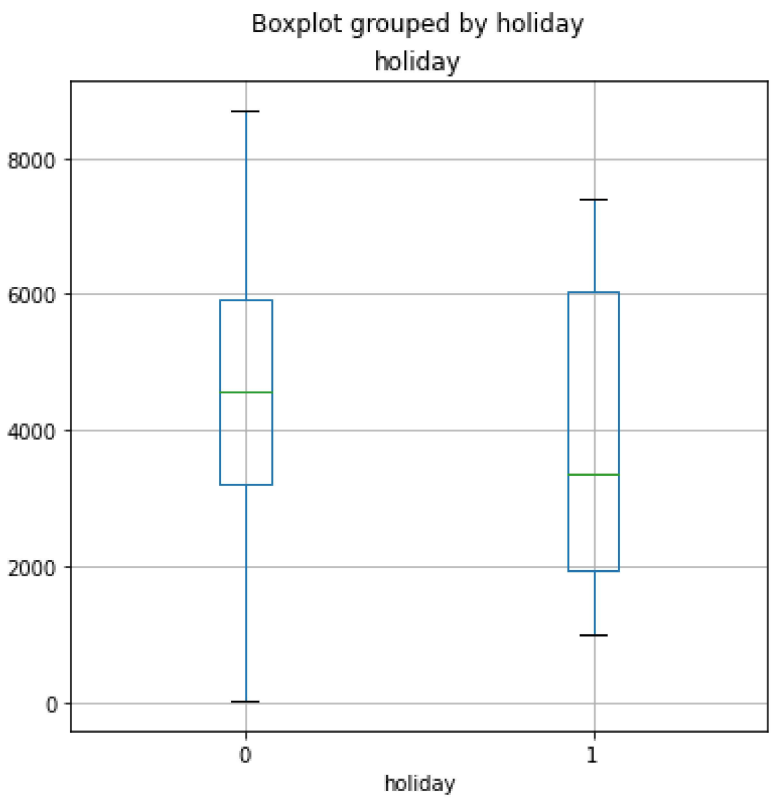
In [ ]:
```
#take insghts for the relation between each categorical feature and the rental counts:
from matplotlib import pyplot as plt
for col in cat_attributes:
    bikes.boxplot(column='cnt',by=col, figsize=(6,6))
    plt.title(col)
    plt.show()
```

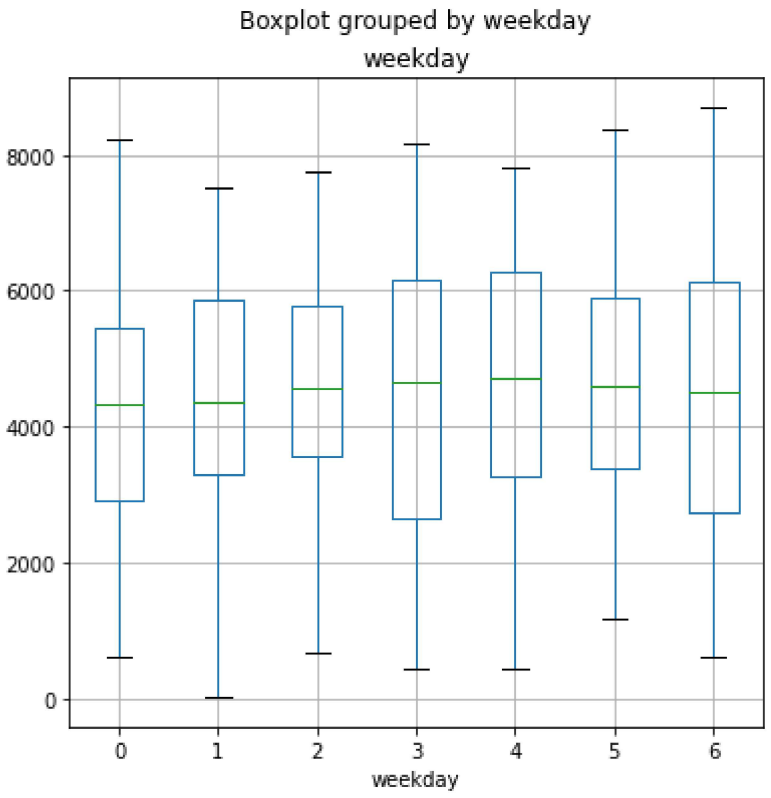


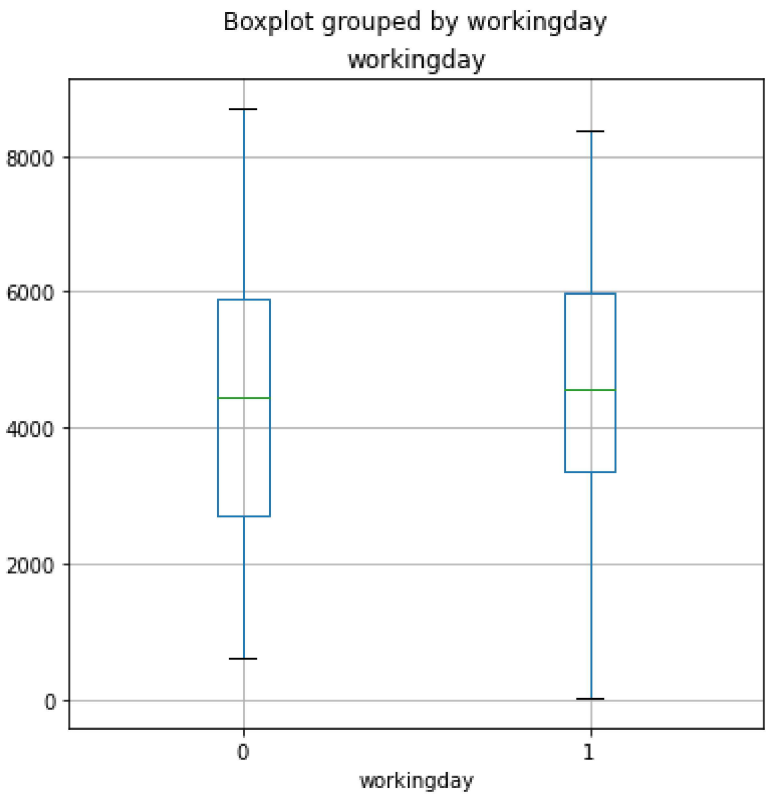


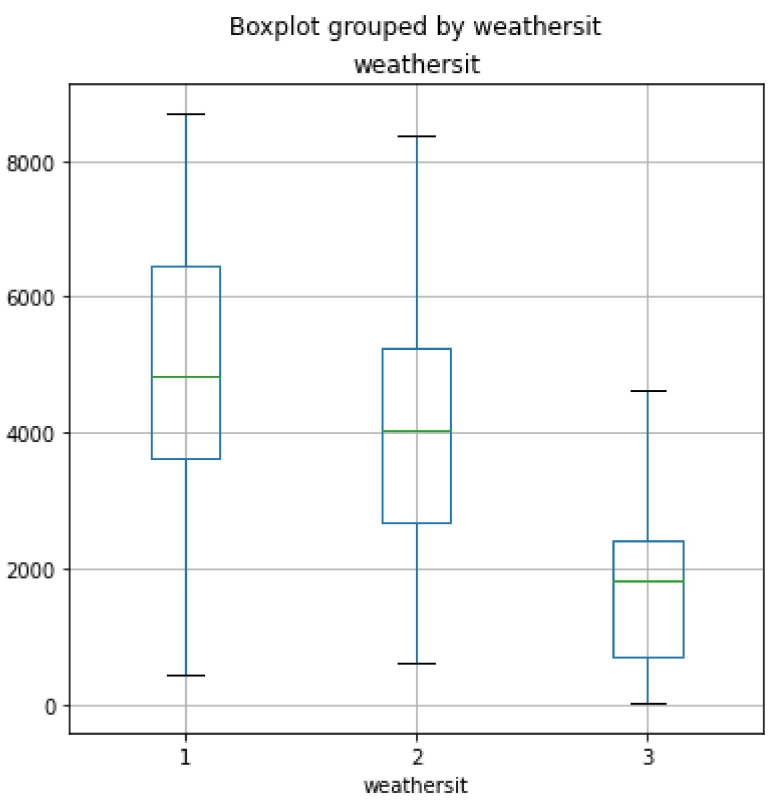

Boxplot grouped by yr
yr
8000
6000
4000
2000
0
0
1
yr


Boxplot grouped by mnth
mnth
8000
6000
4000
2000
0
1
2
3
4
5
6
7
8
9
10
11
12
mnth


Boxplot grouped by holiday
holiday
8000
6000
4000
2000
0
0
1
holiday



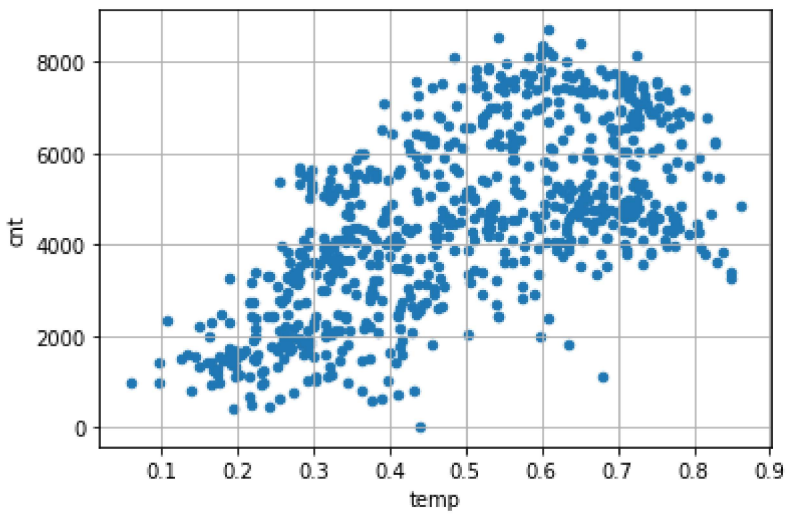


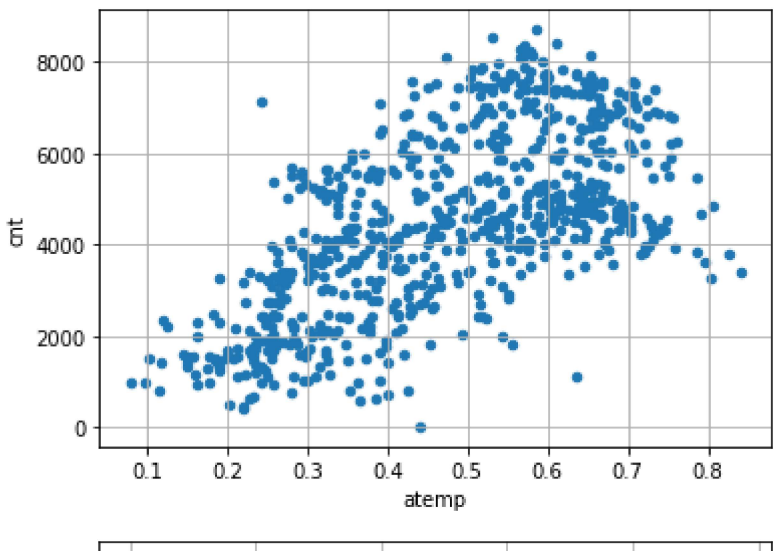


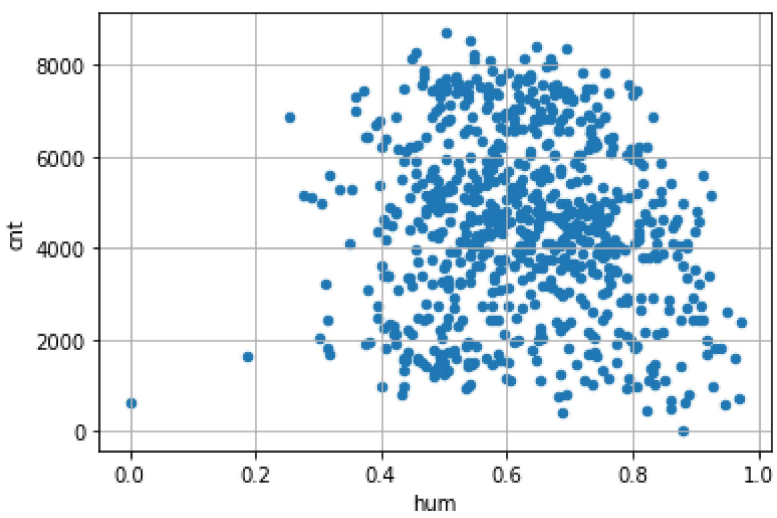
In [ ]:
```
X=bikes.drop(columns='cnt')
y=bikes['cnt']
```

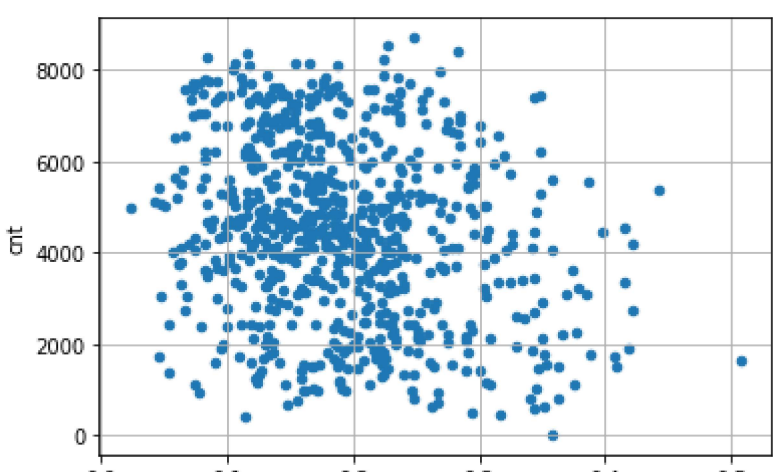
In [ ]:
```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.30, random_state=0,shuffle=True,
                                                    stratify=bikes['holiday'])
```

In [ ]:
```
for num_attribute in num_attributes:
    bikes.plot(kind='scatter',x=num_attribute,y='cnt',grid=True)
    plt.show()
```





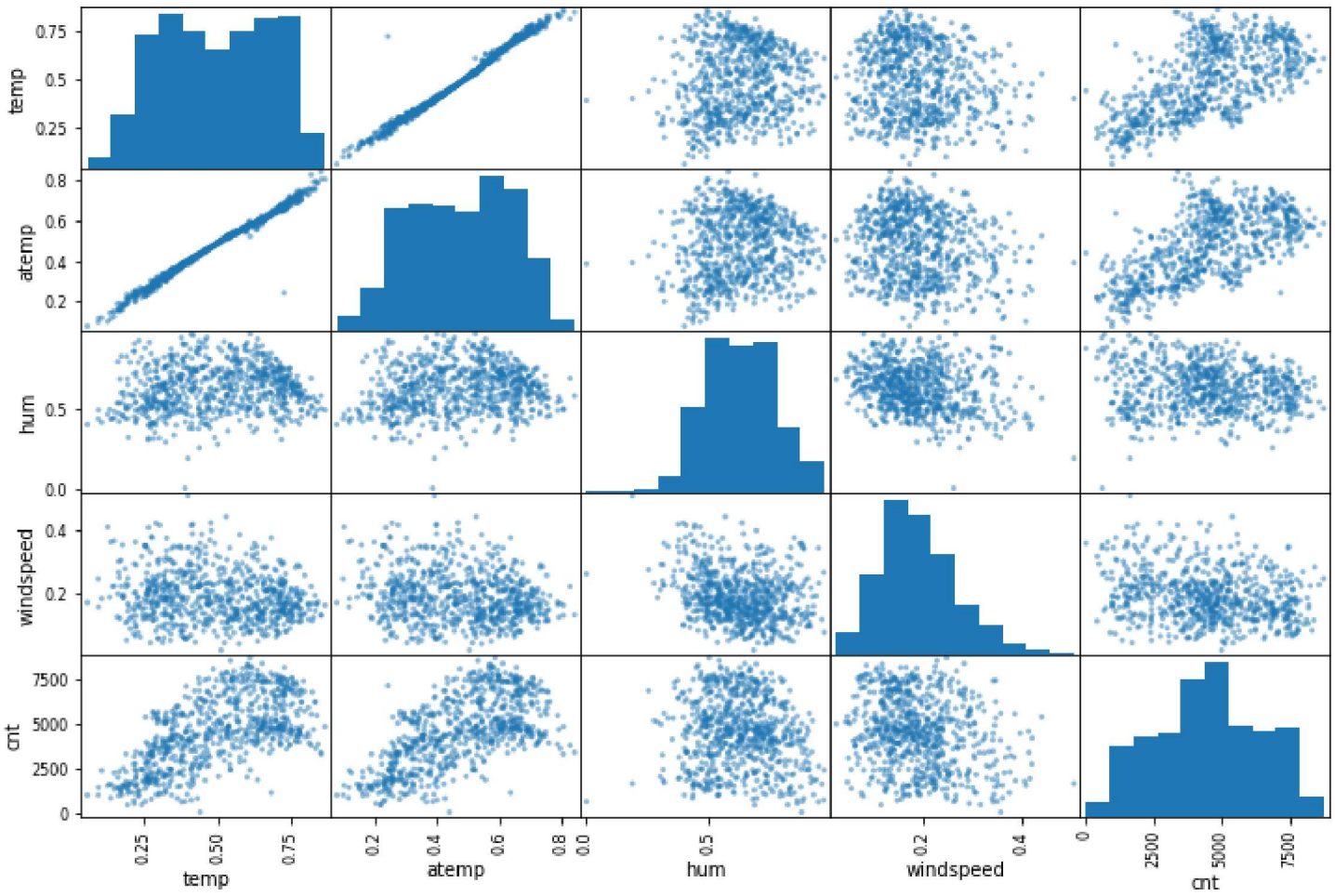




In [ ]:
```
bikes[num_attributes+['cnt']].corr()['cnt'].sort_values(ascending=False)
```

Out[ ]:
```
cnt          1.000000
atemp        0.631066
temp         0.627494
hum         -0.100659
windspeed   -0.234545
Name: cnt, dtype: float64
```

In [ ]:
```
from pandas.plotting import scatter_matrix
scatter_matrix(bikes[num_attributes+['cnt']],figsize=(12,8))
plt.show()
```



In [ ]:
```
from sklearn.pipeline import Pipeline,make_pipeline
from sklearn.linear_model import LinearRegression
from sklearn.neighbors import KNeighborsRegressor
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor
from sklearn.svm import LinearSVR
from sklearn.svm import SVR
from sklearn.model_selection import cross_val_score,cross_val_predict
import numpy as np
from sklearn.compose import ColumnTransformer,make_column_selector
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import StandardScaler
from sklearn.ensemble import GradientBoostingRegressor
from sklearn.linear_model import Lasso,Ridge,ElasticNet
from sklearn.preprocessing import FunctionTransformer


train_results=[]
test_results=[]
confidence_intervals=[]
reg={'LinearRegression':LinearRegression(),
    'GradientBoostingRegressor':GradientBoostingRegressor(random_state=42),
    'Lasso':Lasso(random_state=42),
    'Ridge':Ridge(random_state=42),
    'ElasticNet':ElasticNet(random_state=42),
    'KNeighborsRegressor':KNeighborsRegressor(),
    'DecisionTreeRegressor':DecisionTreeRegressor(random_state=42),
    'RandomForestRegressor':RandomForestRegressor(random_state=42),
    'LinearSVR':LinearSVR(max_iter=50000,random_state=42),
    'SVR':  SVR(max_iter=50000)}


num_pipline=make_pipeline(SimpleImputer(strategy='mean'),StandardScaler())
cat_pipeline=make_pipeline(SimpleImputer(strategy='most_frequent'))

#date_pipeline=make_pipeline(FunctionTransformer(lambda x:pd.DatetimeIndex(x).day))
preprocessing=ColumnTransformer([('num', num_pipline, num_attributes),
                                 ('cat',cat_pipeline,cat_attributes),
                                 #('day',date_pipeline,['dteday'])
                                 ])

#data_prepared=preprocessing.fit_transform(Xtrain)
#data_prepared.shape
```

In [ ]:
```
from sklearn.metrics import mean_squared_error,r2_score
from sklearn.model_selection import cross_val_score,cross_val_predict

for key in reg.keys():
    full_pipeline = Pipeline([
        ('preprocessing', preprocessing),
        ('reg',reg[key]),
    ])
    #train
    score=-cross_val_score(full_pipeline, X_train, y_train, scoring="neg_root_mean_squared_error", cv=10)
    train_rmse=score.mean()/y.mean()
    train_prediction=cross_val_predict(full_pipeline,X_train,y_train,cv=10)
    train_mse=mean_squared_error(y_pred=train_prediction,y_true=y_train)
    train_mse=train_mse.round(1)/y.mean()
    train_r2=r2_score(y_true=y_train,y_pred=train_prediction)
    train_results.append((key,train_rmse,train_mse,train_r2))
    #train_results.append({key: {'train_rmse': train_rmse,'train_mse': train_mse,'train_r2': train_r2}})
    #train_results.append((key,train_rmse,train_mse,train_r2))
```

```
    #test
    test_prediction=cross_val_predict(full_pipeline,X_test,y_test,cv=10)
    test_rmse=mean_squared_error(y_pred=test_prediction,y_true=y_test,squared=False).round(1)/y.mean()
    test_mse=mean_squared_error(y_pred=test_prediction,y_true=y_test)
    test_mse=test_mse.round(1)/y.mean()
    test_r2=r2_score(y_true=y_test,y_pred=test_prediction)
    #test_results.append({key: {'test_rmse':test_rmse,'test_mse': test_mse,'test_r2':test_r2} })

    test_results.append((key,test_rmse,test_mse,test_r2))


    #confidence intervals:
    #*We* can compute a 95% confidence interval for the test RMSE:
    from scipy import stats

    confidence = 0.95
    squared_errors = (test_prediction - y_test) ** 2
    np.sqrt(stats.t.interval(confidence, len(squared_errors) - 1,
                             loc=squared_errors.mean(),
                             scale=stats.sem(squared_errors)))/y.mean()

    # extra code – computes a confidence interval again using a z-score
    zscore = stats.norm.ppf((1 + confidence) / 2)
    zmargin = zscore * squared_errors.std(ddof=1) / np.sqrt(len(squared_errors))
    c1,c2 = np.sqrt(squared_errors.mean() - zmargin)/y.mean(), np.sqrt(squared_errors.mean()
                    + zmargin)/y.mean()
    #confidence_intervals.append({key: {'c1':c1,'c2':c2,'interval':c2-c1}})
    confidence_intervals.append((key,test_rmse,c1,c2,c2-c1))
```

```
c:\Users\user\AppData\Local\Programs\Python\Python310\lib\site-packages\sklearn\linear_model\_coordinate_descent.py:648: ConvergenceWarning: Objective did not converge. You might want to increase the number of iterations, check the scale of the features or consider increasing regularisation. Duality gap: 3.981e+05, tolerance: 7.058e+04
  model = cd_fast.enet_coordinate_descent(
```

In [ ]:
```
#results:
print('models train scores:',train_results)
best_trained_model_idx=np.array(train_results)[:,1].argmin()

print('models test scores:',test_results)
best_tested_model_idx=np.array(test_results)[:,1].argmin()


print('best trained model:',train_results[best_trained_model_idx][0],train_results[best_trained_model_idx][1])
print('best tested model:',test_results[best_tested_model_idx][0],test_results[best_tested_model_idx][1])
```

```
models train scores: [('LinearRegression', 0.2023969926904546, 188.73718060582277, 0.777568233781897), ('GradientBoostingRegressor', 0.1505670579567554, 103.45308874627621, 0.8780777849917398), ('Lasso', 0.20095482015670615, 186.09202579419372, 0.7806856122244701), ('Ridge', 0.20030279274635607, 184.97161967504272, 0.7820060597458061), ('ElasticNet', 0.2588752243298375, 306.37957890823856, 0.6389235620249492), ('KNeighborsRegressor', 0.32726736382524513, 490.0731448161208, 0.42243581861970314), ('DecisionTreeRegressor', 0.19879152256056962, 180.28972207737226, 0.7875237834774442), ('RandomForestRegressor', 0.15433194877233777, 108.68532116249412, 0.8719114807732652), ('LinearSVR', 0.43471003992493584, 862.1508547295379, -0.01606759463811147), ('SVR', 0.4332451407453959, 848.0450422589023, 0.0005564606657268989)]
models test scores: [('LinearRegression', 0.1976978320692664, 176.05965449410647, 0.7781342751688137), ('GradientBoostingRegressor', 0.14508201376447566, 94.825470991858, 0.8805034516769359), ('Lasso', 0.19712061212161888, 175.04050607423318, 0.7794185758355965), ('Ridge', 0.1959439714591067, 172.9546773918745, 0.7820470956405566), ('ElasticNet', 0.2607480109661464, 306.2511252691197, 0.6140704320414441), ('KNeighborsRegressor', 0.33882810926907847, 517.1507545679369, 0.3483003079757677), ('DecisionTreeRegressor', 0.20919782948778182, 197.11930227635307, 0.7515954755947682), ('RandomForestRegressor', 0.1522306608084177, 104.3694254131666, 0.8684764149325555), ('LinearSVR', 0.4459912126265573, 895.9248152947797, -0.129020750171442), ('SVR', 0.42385704771099764, 809.2466262578283, -0.01979117042836398)]
best trained model: GradientBoostingRegressor 0.1505670579567554
best tested model: GradientBoostingRegressor 0.14508201376447566
```

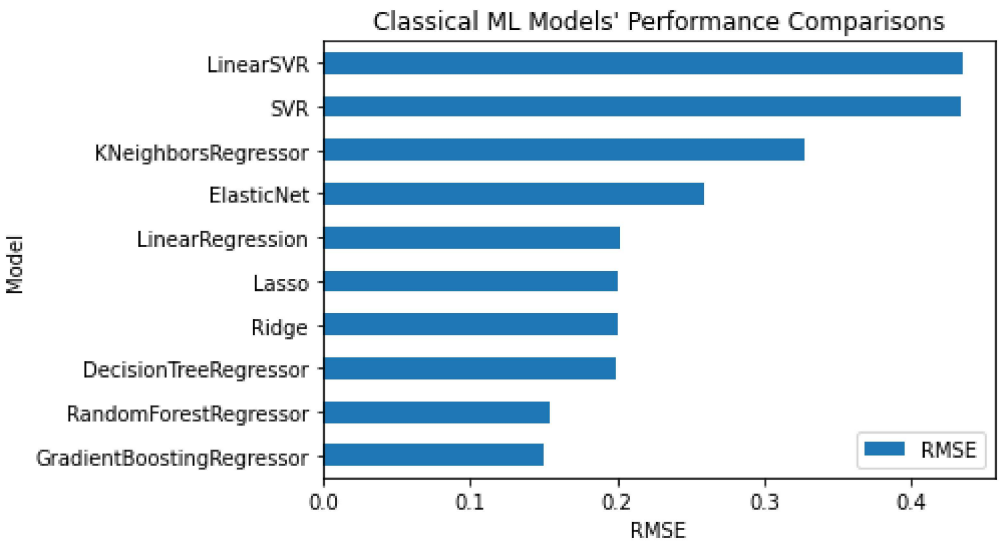
In [ ]:
```
train_result=pd.DataFrame(train_results,
          columns=['Model','RMSE','MSE','R2_Score']).sort_values(by='RMSE')
import dataframe_image as dfi

dfi.export(train_result, "ModelsTrainingResultsTable.png")
train_result
```

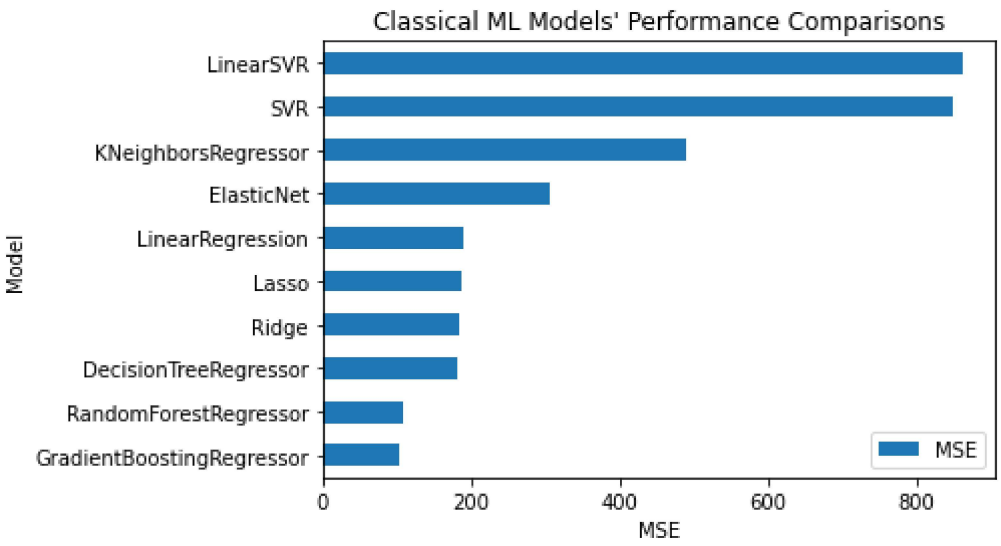
Out[ ]:

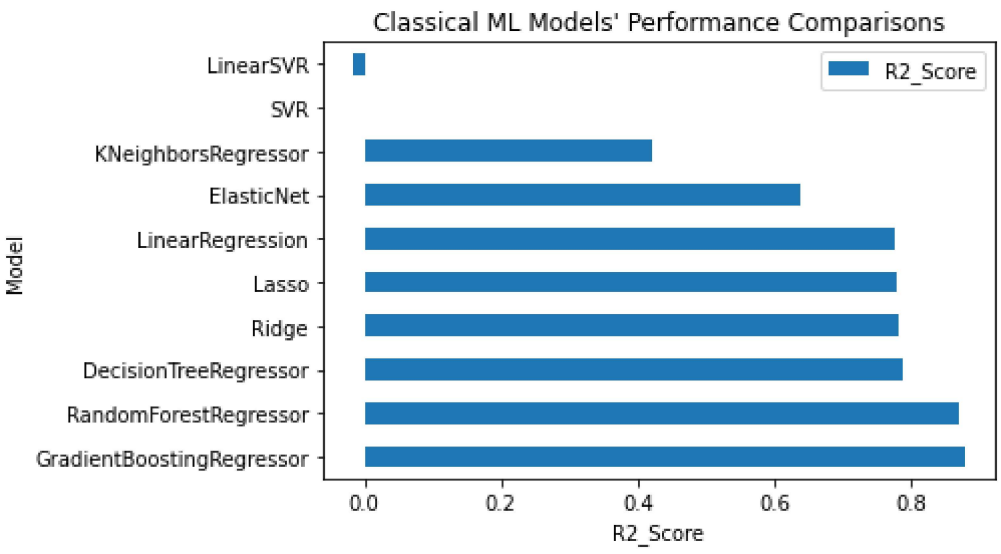
	Model	RMSE	MSE	R2_Score
1	GradientBoostingRegressor	0.150567	103.453089	0.878078
7	RandomForestRegressor	0.154332	108.685321	0.871911
6	DecisionTreeRegressor	0.198792	180.289722	0.787524
3	Ridge	0.200303	184.971620	0.782006
2	Lasso	0.200955	186.092026	0.780686
0	LinearRegression	0.202397	188.737181	0.777568
4	ElasticNet	0.258875	306.379579	0.638924
5	KNeighborsRegressor	0.327267	490.073145	0.422436
9	SVR	0.433245	848.045042	0.000556
8	LinearSVR	0.434710	862.150855	-0.016068

In [ ]:
```
#plt.figure(figsize=(10, 50))
#result.plot.barh(x='Model',subplots=True,sharex=False)
#plt.show()


for score in ['RMSE','MSE','R2_Score']:
```

```
    train_result.plot.barh(x='Model',y=score,sharex=False)
    plt.xlabel(score)
    plt.title('Classical ML Models\' Performance Comparisons')
    plt.show()
```







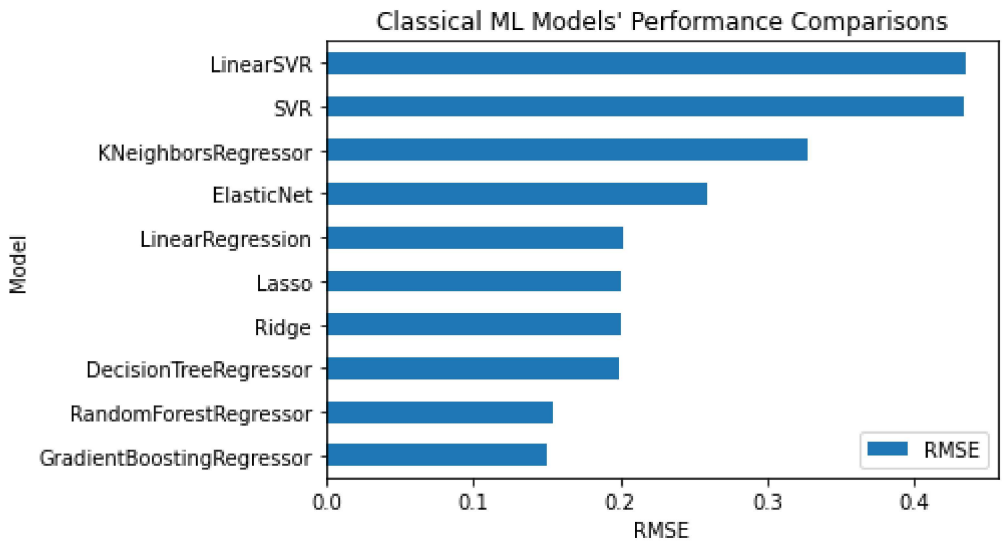
In [ ]:
```
test_result=pd.DataFrame(test_results,
           columns=['Model','RMSE','MSE','R2_Score']).sort_values(by='RMSE')

import dataframe_image as dfi

dfi.export(test_result, "ModelsTestResultsTable.png")
```

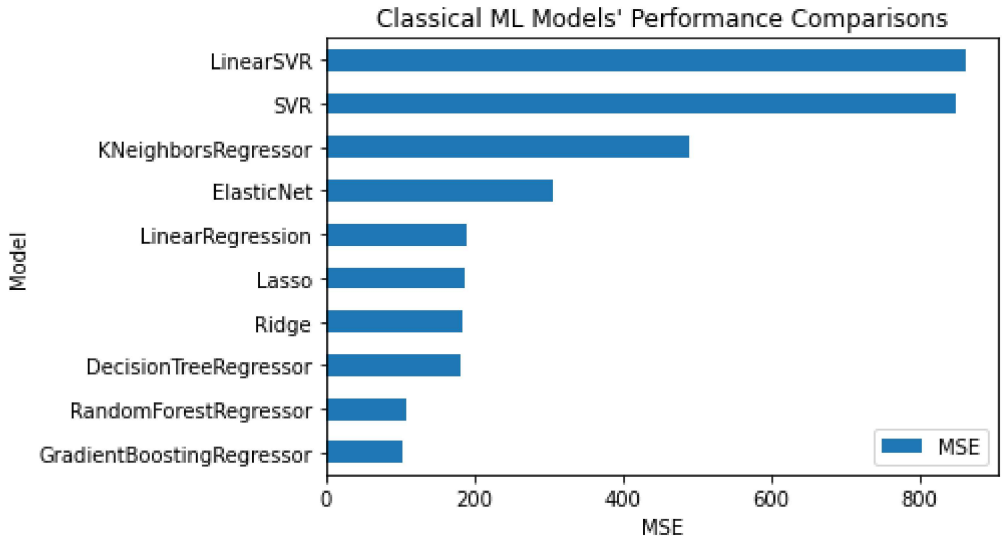
```
test_result
```

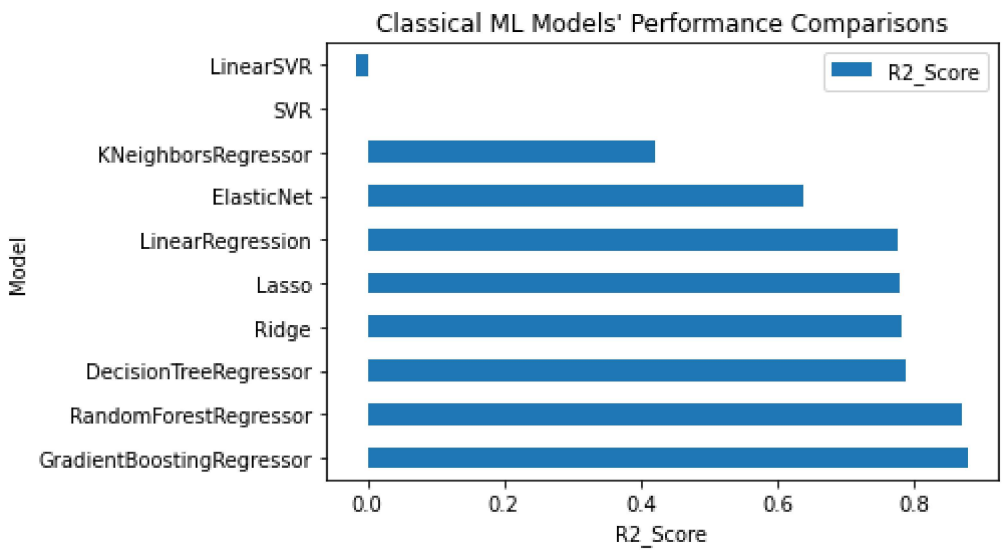
Out[ ]:

	Model	RMSE	MSE	R2_Score
1	GradientBoostingRegressor	0.145082	94.825471	0.880503
7	RandomForestRegressor	0.152231	104.369425	0.868476
3	Ridge	0.195944	172.954677	0.782047
2	Lasso	0.197121	175.040506	0.779419
0	LinearRegression	0.197698	176.059654	0.778134
6	DecisionTreeRegressor	0.209198	197.119302	0.751595
4	ElasticNet	0.260748	306.251125	0.614070
5	KNeighborsRegressor	0.338828	517.150755	0.348300
9	SVR	0.423857	809.246626	-0.019791
8	LinearSVR	0.445991	895.924815	-0.129021

In [ ]:
```
for score in ['RMSE','MSE','R2_Score']:
    train_result.plot.barh(x='Model',y=score,sharex=False)
    plt.xlabel(score)
    plt.title('Classical ML Models\' Performance Comparisons')
    plt.show()
```







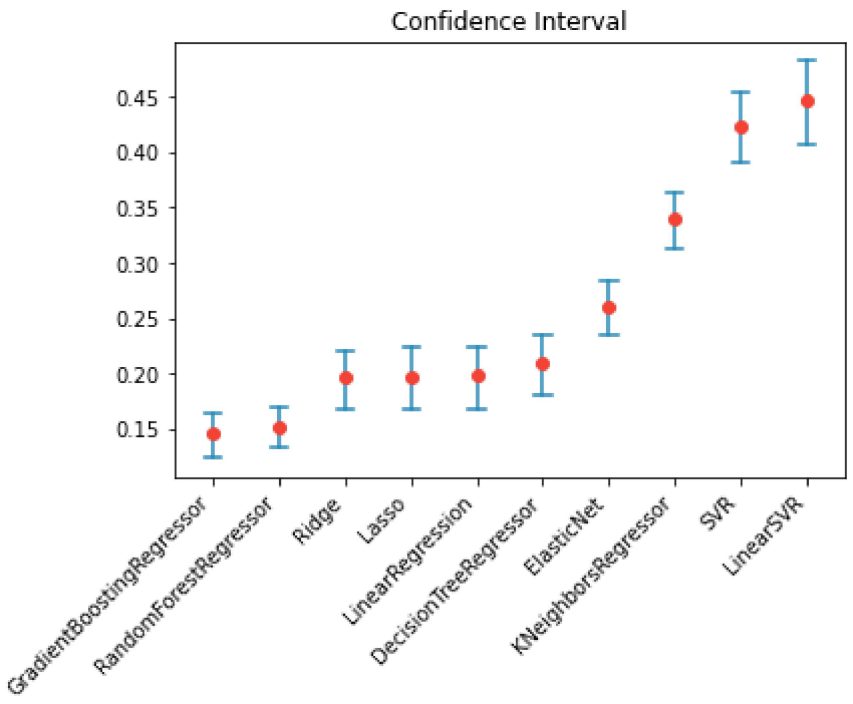
In [ ]:
```
confidence_interval_results=pd.DataFrame(confidence_intervals,
           columns=['Model','RMSE','C1','C2','Confidence Interval']).sort_values(by='C1',ignore_index=True)


import dataframe_image as dfi

dfi.export(confidence_interval_results, "ModelsConfidence_Interval_results.png")

confidence_interval_results
```

Out[ ]:

	Model	RMSE	C1	C2	Confidence Interval
0	GradientBoostingRegressor	0.145082	0.123974	0.163506	0.039532
1	RandomForestRegressor	0.152231	0.132778	0.169445	0.036667
2	Ridge	0.195944	0.167118	0.221056	0.053938
3	Lasso	0.197121	0.167421	0.222914	0.055493
4	LinearRegression	0.197698	0.167517	0.223856	0.056339
5	DecisionTreeRegressor	0.209198	0.179834	0.234913	0.055079
6	ElasticNet	0.260748	0.234856	0.284294	0.049438
7	KNeighborsRegressor	0.338828	0.312297	0.363447	0.051150
8	SVR	0.423857	0.390745	0.454573	0.063828
9	LinearSVR	0.445991	0.406966	0.481853	0.074887

In [ ]:
```
plt.xticks(range(len(confidence_interval_results)),list(confidence_interval_results['Model']),
           rotation = 45, ha="right")
plt.title('Confidence Interval')
for c1,c2,b in zip(confidence_interval_results['C1'],confidence_interval_results['C2'],
                   range(len(confidence_interval_results))):
    color='#2187bb'
    horizontal_line_width=0.25
    left = b - horizontal_line_width / 2
    right = b + horizontal_line_width / 2
    #plt.plot((b,b),(c1,c2),'ro-',color='orange')
```

```
    plt.plot([b, b], [c2, c1], color=color)
    plt.plot([left, right], [c2, c2], color=color)
    plt.plot([left, right], [c1, c1], color=color)
    plt.plot(b, confidence_interval_results['RMSE'][b], 'o', color='#f44336')

#plt.xticks(range(len(confidence_interval_results)),list(confidence_interval_results['Model']))
plt.show()
```



In [ ]:
```
from sklearn.pipeline import Pipeline
from sklearn.model_selection import GridSearchCV,RandomizedSearchCV
from sklearn.ensemble import RandomForestRegressor
from sklearn.compose import TransformedTargetRegressor
from sklearn.metrics import mean_squared_error,r2_score
from scipy.stats import randint


full_pipeline = Pipeline([
    ('preprocessing', preprocessing),
    ('GradientBoostingRegressor',GradientBoostingRegressor(
                            n_estimators = 500, subsample= 0.5, criterion ='squared_error',
                            min_samples_split = 6, min_samples_leaf = 1, min_weight_fraction_leaf =0,
                            max_depth = 3, min_impurity_decrease = 0, init = None, random_state = 42,
                            max_features= 8, alpha = 0.01, max_leaf_nodes = None, warm_start = False,
                            validation_fraction=0.1, n_iter_no_change=10, tol= 0.0001, ccp_alpha= 0))
                        ])


param_grid=[{
                'GradientBoostingRegressor__max_depth':[4],
                'GradientBoostingRegressor__learning_rate':[0.01,.05,.1],
                'GradientBoostingRegressor__criterion':["friedman_mse",'squared_error'],
                'GradientBoostingRegressor__min_samples_split':[4,6,8],
                'GradientBoostingRegressor__subsample':[.5],
                'GradientBoostingRegressor__max_features':[6],
                'GradientBoostingRegressor__alpha':[.001,0.01],
            }]
#param=[{
#                #'GradientBoostingRegressor__n_estimators':randint(low=100, high=200),
#                'GradientBoostingRegressor__min_samples_split':randint(low=2, high=10),
#                'GradientBoostingRegressor__subsample':np.linspace(start=.1,stop=.9),
#                'GradientBoostingRegressor__max_features':randint(low=5, high=10),
#                'GradientBoostingRegressor__alpha':np.linspace(start=0.0001,stop=.1),
#
#            }]
#grid_search=RandomizedSearchCV(full_pipeline,param,cv=10,scoring='neg_root_mean_squared_error',random_state=42,verbose

grid_search=GridSearchCV(full_pipeline,param_grid,cv=10,scoring='neg_root_mean_squared_error',verbose=1)
```

In [ ]:
```
grid_search.fit(X_train,y_train)
print('best score=',-grid_search.best_score_)
rmse=-cross_val_score(grid_search.best_estimator_,X_train,y_train,
                          scoring='neg_root_mean_squared_error',cv=10)
rmse_percentage=rmse.mean().round(1)/y.mean()
print('rmse percentage:',rmse_percentage)
print(grid_search.best_params_)
```

```
Fitting 10 folds for each of 36 candidates, totalling 360 fits
best score= 643.7090628656674
rmse percentage: 0.14290633857718898
{'GradientBoostingRegressor__alpha': 0.001, 'GradientBoostingRegressor__criterion': 'squared_error', 'GradientBoostingR
egressor__learning_rate': 0.05, 'GradientBoostingRegressor__max_depth': 4, 'GradientBoostingRegressor__max_features':
6, 'GradientBoostingRegressor__min_samples_split': 4, 'GradientBoostingRegressor__subsample': 0.5}
```

In [ ]:
```
print('best score=',-grid_search.best_score_/y.mean())
rmse=-cross_val_score(full_pipeline,X_train,y_train,scoring='neg_root_mean_squared_error',cv=10)
rmse.mean()
rmse_percentage=rmse.mean()/y.mean()
print('rmse percentage:',rmse_percentage)
print(grid_search.best_params_)
```

```
best score= 0.142908350602899
rmse percentage: 0.1505154962839044
{'GradientBoostingRegressor__alpha': 0.001, 'GradientBoostingRegressor__criterion': 'squared_error', 'GradientBoostingR
egressor__learning_rate': 0.05, 'GradientBoostingRegressor__max_depth': 4, 'GradientBoostingRegressor__max_features':
6, 'GradientBoostingRegressor__min_samples_split': 4, 'GradientBoostingRegressor__subsample': 0.5}
```

In [ ]:
```
grid_search.best_estimator_['GradientBoostingRegressor'].n_estimators_
```

Out[ ]:
```
131
```

In [ ]:
```
import dataframe_image as dfi
best_params=grid_search.best_params_
best_params['GradientBoostingRegressor__n_estimators_']=grid_search.best_estimator_['GradientBoostingRegressor'].n_esti
GBR_best_param=pd.DataFrame.from_dict(grid_search.best_params_,orient='index',columns=['best value'])
GBR_best_param['best value']

dfi.export(GBR_best_param, "Grid_BestHyperParameters.png")

GBR_best_param
```

Out[ ]:

	best value
GradientBoostingRegressor__alpha	0.001
GradientBoostingRegressor__criterion	squared_error
GradientBoostingRegressor__learning_rate	0.05
GradientBoostingRegressor__max_depth	4
GradientBoostingRegressor__max_features	6
GradientBoostingRegressor__min_samples_split	4
GradientBoostingRegressor__subsample	0.5
GradientBoostingRegressor__n_estimators_	131

In [ ]:
```
from sklearn.metrics import mean_squared_error,r2_score

final_model=grid_search.best_estimator_
final_prediction=final_model.predict(X_test)
final_rmse=mean_squared_error(y_pred=final_prediction,y_true=y_test,squared=False)
final_rmse_percentage=final_rmse.round(1)/y.mean()
print('final rmse percentage:',final_rmse_percentage)
#train the best model in the whole data set including the train and test datat set.
model=final_model.fit(X,y)
```

```
final rmse percentage: 0.13158394729641124
```

In [ ]:
```
feature_importances = final_model["GradientBoostingRegressor"].feature_importances_
feature_importances.round(2)
```

Out[ ]:
```
array([0.29, 0.16, 0.06, 0.04, 0.02, 0.08, 0.27, 0.03, 0.  , 0.02, 0.  ,
       0.04])
```

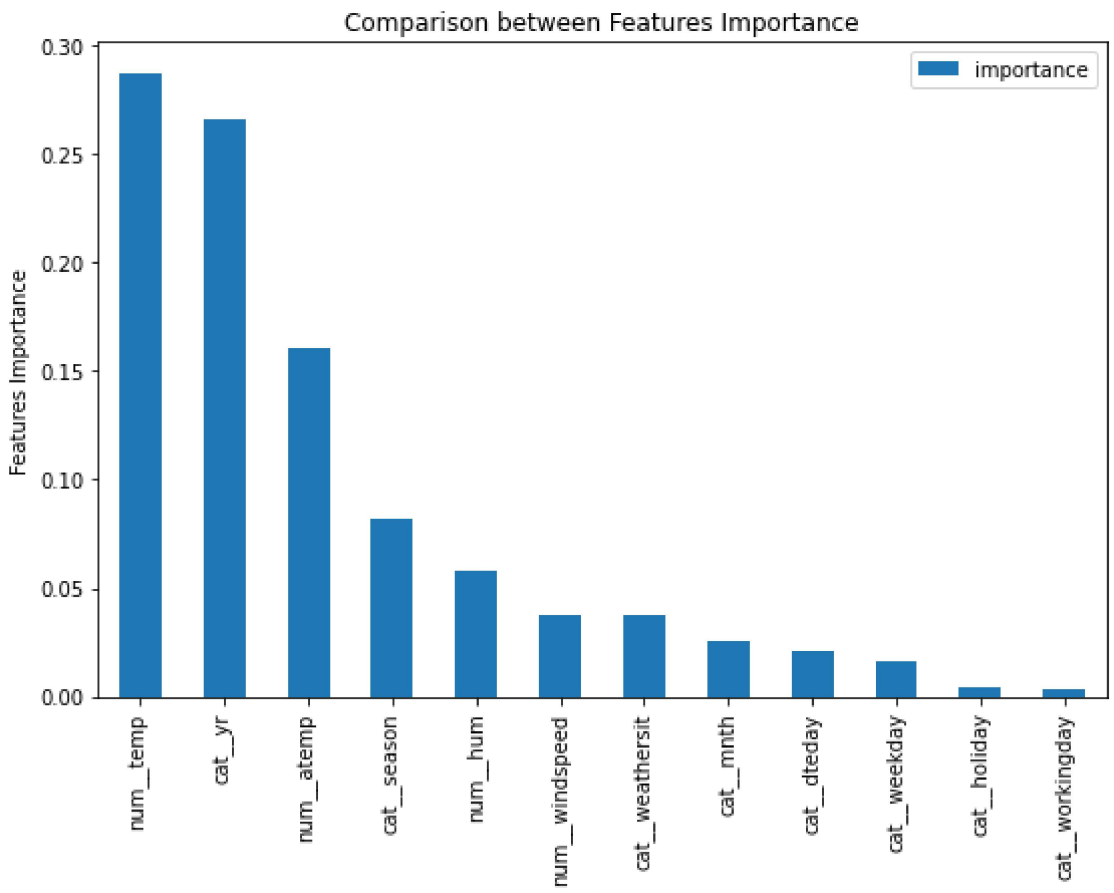
In [ ]:
```
features_imoprtance=sorted(zip(feature_importances,
          final_model["preprocessing"].get_feature_names_out()),reverse=True)

features_imoprtance=pd.DataFrame(features_imoprtance,
          columns=['importance','features'])

import dataframe_image as dfi

dfi.export(features_imoprtance, "FeatureImportanceTable.png")
features_imoprtance
```

Out[ ]:

	importance	features
0	0.287367	num__temp
1	0.265890	cat__yr
2	0.160682	num__atemp
3	0.082350	cat__season
4	0.057751	num__hum
5	0.037859	num__windspeed
6	0.037467	cat__weathersit
7	0.025830	cat__mnth
8	0.020664	cat__dteday
9	0.016149	cat__weekday
10	0.004376	cat__holiday
11	0.003616	cat__workingday

In [ ]:
```
features_imoprtance.index=features_imoprtance['features']

fig = plt.figure(figsize=(9, 6))
ax = fig.gca()

features_imoprtance.plot.bar(ax=ax)
```

```
ax.set_title('Comparison between Features Importance')
ax.set_xlabel('Features')
ax.set_ylabel("Features Importance")
plt.show()
```



In [ ]:
```
#final_prediction=final_model.predict(X_test)
#final_rmse=mean_squared_error(y_pred=final_prediction,y_true=y_test,squared=False)
#final_rmse
#final_rmse_percentage=final_rmse.round(1)/y.mean()
#print('final rmse percentage:',final_rmse_percentage)
```

In [ ]:
```
#*We* can compute a 95% confidence interval for the test RMSE:
from scipy import stats

confidence = 0.95
squared_errors = (final_prediction - y_test) ** 2
np.sqrt(stats.t.interval(confidence, len(squared_errors) - 1,
                         loc=squared_errors.mean(),
                         scale=stats.sem(squared_errors)))/y.mean()
```

Out[ ]:
```
array([0.11278408, 0.14800198])
```

In [ ]:
```
# extra code – computes a confidence interval again using a z-score
zscore = stats.norm.ppf((1 + confidence) / 2)
zmargin = zscore * squared_errors.std(ddof=1) / np.sqrt(len(squared_errors))
final_c1,final_c2=np.sqrt(squared_errors.mean() - zmargin)/y.mean(), np.sqrt(squared_errors.mean()
                          + zmargin)/y.mean()
final_confidence_interval=final_c2-final_c1
```

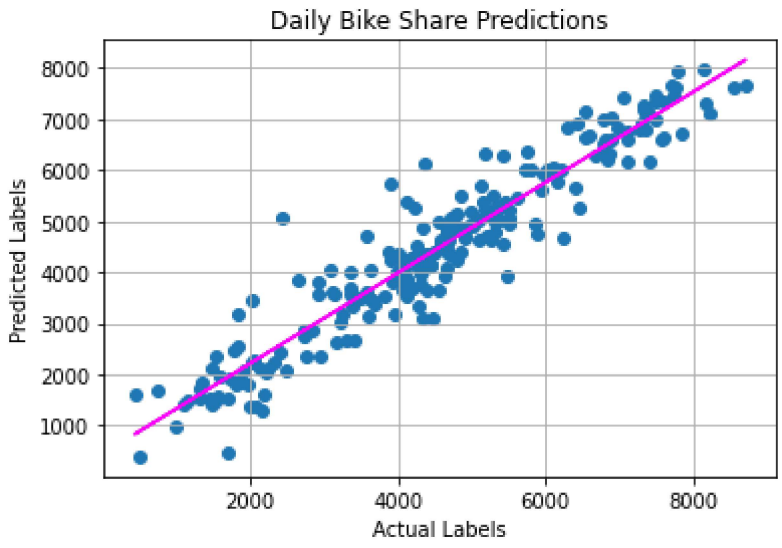
In [ ]:
```
final_r2=r2_score(y_true=y_test,y_pred=final_prediction)
final_r2
```

Out[ ]:
```
0.9017301962234706
```

In [ ]:
```
grid_search.cv_results_
grid_search.return_train_score
```

Out[ ]:
```
False
```

In [ ]:
```
# Plot predicted vs actual
plt.scatter(y_test, final_prediction)
plt.xlabel('Actual Labels')
plt.ylabel('Predicted Labels')
plt.title('Daily Bike Share Predictions')
# overlay the regression line
z = np.polyfit(y_test, final_prediction, 1)
p = np.poly1d(z)
plt.plot(y_test,p(y_test), color='magenta')
plt.grid()
plt.show()
```



In [ ]:
```
#Save the final Model:
import joblib

joblib.dump(final_model,'final_model.pkl')
```

Out[ ]:
```
['final_model.pkl']
```

In [ ]:
```
import joblib
from sklearn.cluster import KMeans
from sklearn.base import BaseEstimator, TransformerMixin
from sklearn.metrics.pairwise import rbf_kernel


loaded_model=joblib.load('final_model.pkl')

new_data=X.iloc[:5]
predictions=final_model.predict(new_data)
predictions
```

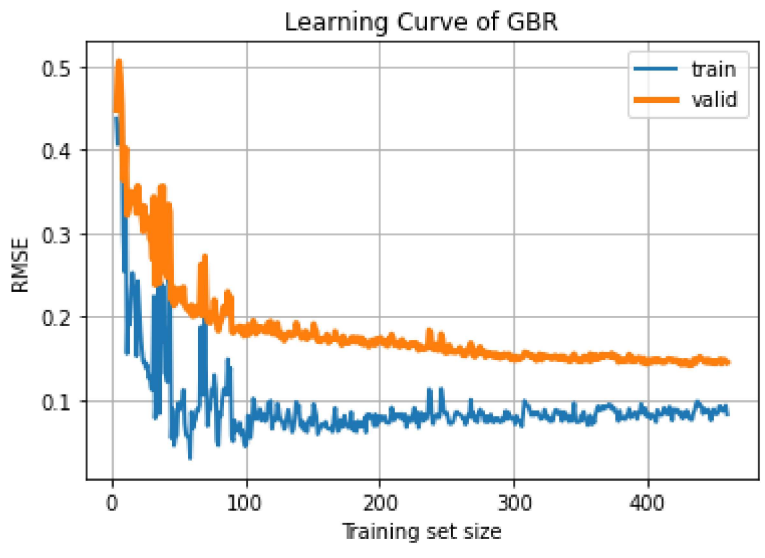
Out[ ]:
```
array([1237.10786125, 1249.49968619, 1451.09773355, 1499.77524497,
       1756.38885036])
```

In [ ]:
```
y.iloc[:5]
```

Out[ ]:
```
0     985
1     801
2    1349
3    1562
4    1600
Name: cnt, dtype: int64
```

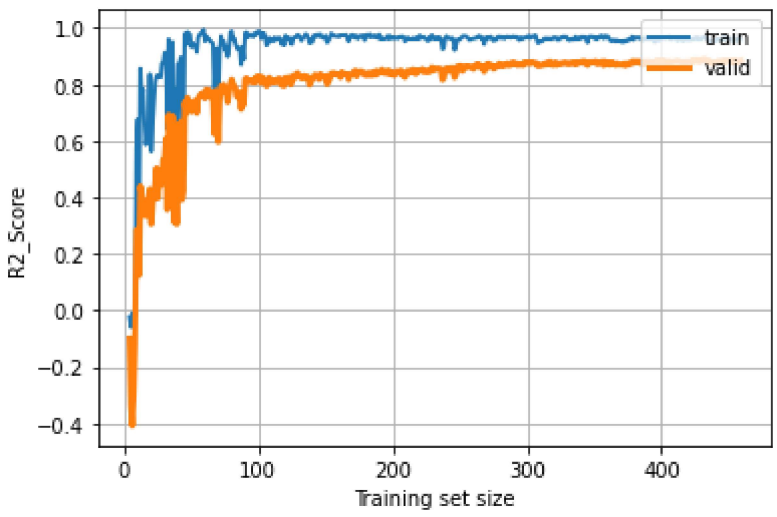
In [ ]:
```
from sklearn.model_selection import learning_curve

train_sizes, train_rmse_scores, valid_rmse_scores = learning_curve(
    final_model, X_train, y_train, train_sizes=np.linspace(0.01, 1, 456), cv=10,
    scoring="neg_root_mean_squared_error",random_state=42)
train_rmse_errors = train_rmse_scores.mean(axis=1)
valid_rmse_errors = valid_rmse_scores.mean(axis=1)


#valid_errors
```

In [ ]:
```
train_rmse_errors = train_rmse_scores.mean(axis=1)/y.mean()
valid_rmse_errors = valid_rmse_scores.mean(axis=1)/y.mean()

plt.figure(figsize=(6, 4))  # extra code – not needed, just formatting
plt.plot(train_sizes, -train_rmse_errors, linewidth=2, label="train")
plt.plot(train_sizes, -valid_rmse_errors, linewidth=3, label="valid")

# extra code – beautifies and saves Figure 4–15
plt.title("Learning Curve of GBR")
plt.xlabel("Training set size")
plt.ylabel("RMSE")
plt.grid()
plt.legend(loc="upper right")
#plt.axis([400, 700, 0, .3])
plt.show()
```



In [ ]:
```
from sklearn.model_selection import learning_curve
from sklearn.metrics import make_scorer

score = make_scorer(r2_score)
train_sizes, train_scores, valid_scores = learning_curve(
    final_model, X_train, y_train, train_sizes=np.linspace(0.01, 1.0, 456), cv=10,
    scoring=score,random_state=42)
train_scores = train_scores.mean(axis=1)
valid_scores = valid_scores.mean(axis=1)
```

In [ ]:
```
plt.figure(figsize=(6, 4))  # extra code – not needed, just formatting
plt.plot(train_sizes, train_scores, linewidth=2, label="train")
plt.plot(train_sizes, valid_scores, linewidth=3, label="valid")

# extra code – beautifies and saves Figure 4–15
plt.xlabel("Training set size")
plt.ylabel("R2_Score")
plt.grid()
plt.legend(loc="upper right")
#plt.axis([400, 700, 0, 1])

plt.show()
#valid_scores
```



In [ ]:
```
#The End! :)
```

In [ ]:
```
# Use SMOGN for oversampling in regression problem
import smogn

bikes_smote = smogn.smoter(
                data=bikes,
                y='cnt',
                k=5,
                samp_method='extreme',
                rel_thres=0.9,
                rel_method='auto',
                rel_xtrm_type='high',
                rel_coef=0.9
            )
```

```
dist_matrix: 100%|##########| 30/30 [00:00<00:00, 37.17it/s]
synth_matrix: 100%|##########| 30/30 [00:03<00:00,  9.95it/s]
r_index: 100%|##########| 11/11 [00:00<00:00, 111.68it/s]
```

In [ ]:
```
bikes_smote.shape
```

Out[ ]:
```
(1402, 13)
```

In [ ]:
```
X=bikes_smote.drop(columns='cnt')
y=bikes_smote['cnt']
```

In [ ]:
```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.30, random_state=42,shuffle=True,
                                                    stratify=bikes_smote['holiday'])
```

In [ ]:
```
from sklearn.pipeline import Pipeline
from sklearn.model_selection import GridSearchCV
from sklearn.ensemble import RandomForestRegressor
from sklearn.compose import TransformedTargetRegressor
from sklearn.metrics import mean_squared_error,r2_score
from sklearn.model_selection import RandomizedSearchCV
from scipy.stats import randint

full_pipeline = Pipeline([
    ('preprocessing', preprocessing),
    ('GradientBoostingRegressor',GradientBoostingRegressor(
                            n_estimators = 500, subsample= 0.5, criterion ='friedman_mse',
                            min_samples_split = 6, min_samples_leaf = 1, min_weight_fraction_leaf =0,
                            max_depth = 3, min_impurity_decrease = 0, init = None, random_state = 42,
                            max_features= 8, alpha = 0.01, max_leaf_nodes = None, warm_start = False,
                            validation_fraction=0.1, tol= 0.0001, ccp_alpha= 0, n_iter_no_change=10))
                        ])

#param={
#                #'GradientBoostingRegressor__n_estimators':randint(low=100, high=200),
```

```
#                 'GradientBoostingRegressor__min_samples_split':randint(low=2, high=10),
#                 'GradientBoostingRegressor__subsample':np.linspace(start=.1,stop=.9),
#                 'GradientBoostingRegressor__max_features':randint(low=5, high=10),
#                 'GradientBoostingRegressor__alpha':np.linspace(start=0.0001,stop=.1),
#
#             }]
#
param=[{
                'GradientBoostingRegressor__max_depth':[4],
                'GradientBoostingRegressor__learning_rate':[0.01,.05,.1],
                'GradientBoostingRegressor__criterion':["friedman_mse",'squared_error'],
                'GradientBoostingRegressor__min_samples_split':[8],
                'GradientBoostingRegressor__subsample':[.5],
                'GradientBoostingRegressor__max_features':[8],
                'GradientBoostingRegressor__alpha':[.001],
            }]

rnd_search=GridSearchCV(full_pipeline,param,cv=10,scoring='neg_root_mean_squared_error',verbose=1)
#rnd_search=GridSearchCV(full_pipeline,param,cv=10,scoring='neg_root_mean_squared_error',verbose=1)

rnd_search.fit(X_train,y_train)
print('best score=',-rnd_search.best_score_)
rnd_rmse=-cross_val_score(rnd_search.best_estimator_,X_train,y_train,
                          scoring='neg_root_mean_squared_error',cv=10)
rnd_rmse_percentage=rnd_rmse.mean().round(1)/y.mean()
print('rmse percentage:',rnd_rmse_percentage)
print(rnd_search.best_params_)
```

```
Fitting 10 folds for each of 6 candidates, totalling 60 fits
best score= 532.964654612481
rmse percentage: 0.08651835035470896
{'GradientBoostingRegressor__alpha': 0.001, 'GradientBoostingRegressor__criterion': 'friedman_mse', 'GradientBoostingRe
gressor__learning_rate': 0.05, 'GradientBoostingRegressor__max_depth': 4, 'GradientBoostingRegressor__max_features': 8,
'GradientBoostingRegressor__min_samples_split': 8, 'GradientBoostingRegressor__subsample': 0.5}
```

In [ ]:
```
rnd_search.best_estimator_['GradientBoostingRegressor'].n_estimators_
```

Out[ ]:
```
125
```

In [ ]:
```
import dataframe_image as dfi
best_params=rnd_search.best_params_
best_params['GradientBoostingRegressor__n_estimators_']=rnd_search.best_estimator_['GradientBoostingRegressor'].n_estim

GBR_rnd_best_param=pd.DataFrame.from_dict(best_params,orient='index',columns=['best value'])
GBR_rnd_best_param['best value']
dfi.export(GBR_rnd_best_param, "RND_BestHyperParameters.png")

GBR_rnd_best_param
```

Out[ ]:

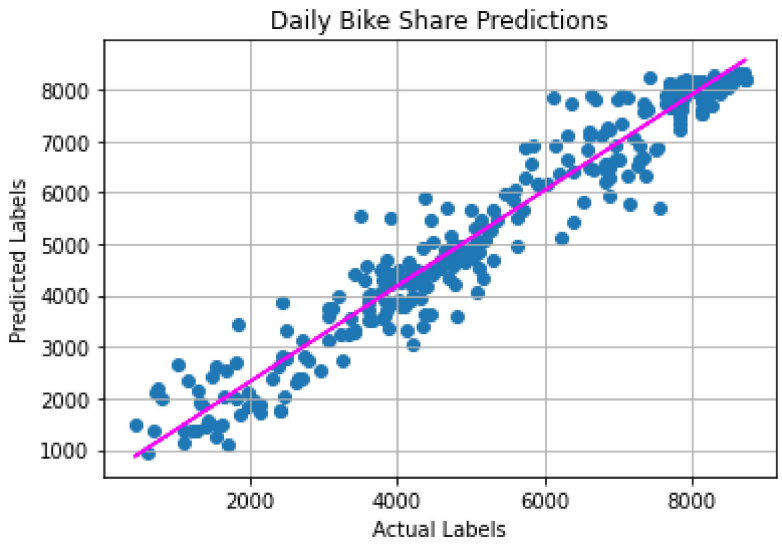
	best value
GradientBoostingRegressor__alpha	0.001
GradientBoostingRegressor__criterion	friedman_mse
GradientBoostingRegressor__learning_rate	0.05
GradientBoostingRegressor__max_depth	4
GradientBoostingRegressor__max_features	8
GradientBoostingRegressor__min_samples_split	8
GradientBoostingRegressor__subsample	0.5
GradientBoostingRegressor__n_estimators_	125

In [ ]:
```
from sklearn.metrics import mean_squared_error,r2_score

final_rnd_model=rnd_search.best_estimator_
final_rnd_prediction=final_rnd_model.predict(X_test)
final_rnd_rmse=mean_squared_error(y_pred=final_rnd_prediction,y_true=y_test,squared=False)
final_rnd_rmse_percentage=final_rnd_rmse.round(1)/y.mean()
print('final rmse percentage:',final_rnd_rmse_percentage)
#train the best model in the whole data set including the train and test datat set.
rnd_model=final_rnd_model.fit(X,y)
```

```
final rmse percentage: 0.07862943510660604
```

In [ ]:
```
#final_rnd_prediction=final_rnd_model.predict(X_test)
#final_rnd_rmse=mean_squared_error(y_pred=final_rnd_prediction,y_true=y_test,squared=False)
#final_rnd_rmse
#final_rnd_rmse_percentage=final_rnd_rmse.round(1)/y.mean()
#print('final rmse percentage:',final_rnd_rmse_percentage)
```

In [ ]:
```
# Plot predicted vs actual
plt.scatter(y_test, final_rnd_prediction)
plt.xlabel('Actual Labels')
plt.ylabel('Predicted Labels')
plt.title('Daily Bike Share Predictions')
# overlay the regression line
z = np.polyfit(y_test, final_rnd_prediction, 1)
p = np.poly1d(z)
plt.plot(y_test,p(y_test), color='magenta')
plt.grid()
plt.show()
```



In [ ]:
```
#*We* can compute a 95% confidence interval for the test RMSE:
from scipy import stats

confidence = 0.95
rnd_squared_errors = (final_rnd_prediction - y_test) ** 2
np.sqrt(stats.t.interval(confidence, len(rnd_squared_errors) - 1,
                         loc=rnd_squared_errors.mean(),
                         scale=stats.sem(rnd_squared_errors)))/y.mean()
```

Out[ ]:
```
array([0.07016885, 0.08626376])
```

In [ ]:
```
# extra code – computes a confidence interval again using a z-score
zscore = stats.norm.ppf((1 + confidence) / 2)
zmargin = zscore * rnd_squared_errors.std(ddof=1) / np.sqrt(len(rnd_squared_errors))
final_rnd_c1,final_rnd_c2=np.sqrt(rnd_squared_errors.mean() - zmargin)/y.mean(), np.sqrt(
                           rnd_squared_errors.mean() + zmargin)/y.mean()
final_rnd_confidence_interval=final_rnd_c2-final_rnd_c1
```

In [ ]:
```
final_rnd_r2=r2_score(y_true=y_test,y_pred=final_rnd_prediction)
final_rnd_r2
```

Out[ ]:
```
0.9544894083024373
```

In [ ]:
```
A_B_testing=[('GB_Model_With_SMOTE',final_rnd_r2,final_rnd_rmse_percentage,final_rnd_c1,final_rnd_c2,
               final_rnd_confidence_interval),('GB_Model_Without_SMOTE',final_r2,final_rmse_percentage,
               final_c1,final_c2,final_confidence_interval)]
```

In [ ]:
```
A_B_testing_results=pd.DataFrame(A_B_testing,
           columns=['Model','R2','RMSE','C1','C2','Confidence Interval']).sort_values(by='RMSE',
           ignore_index=True)
A_B_testing_results
```

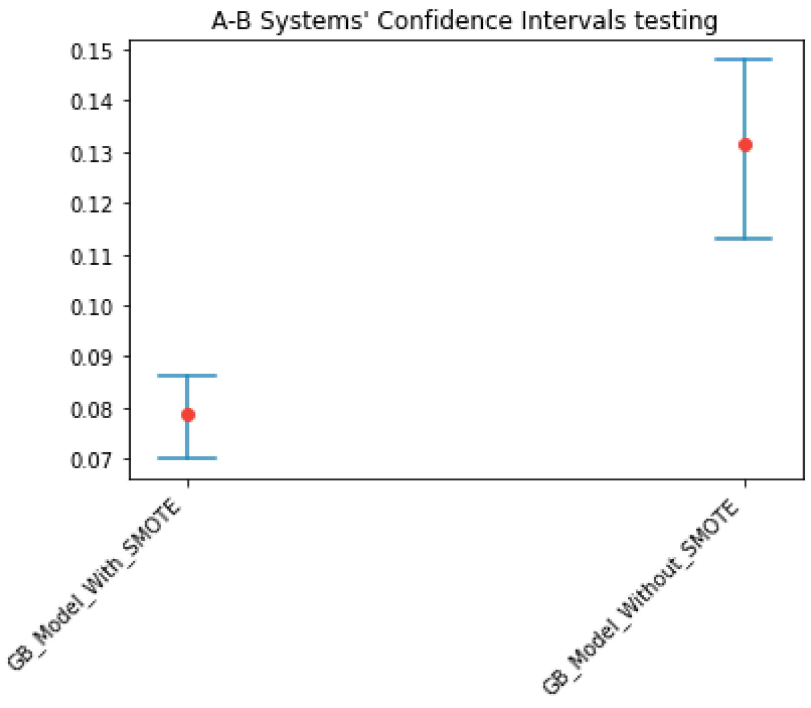
Out[ ]:

	Model	R2	RMSE	C1	C2	Confidence Interval
0	GB_Model_With_SMOTE	0.954489	0.078629	0.070195	0.086243	0.016048
1	GB_Model_Without_SMOTE	0.901730	0.131584	0.112897	0.147916	0.035020

In [ ]:
```
confidence_interval_testing=A_B_testing_results[['Model','C1','C2','Confidence Interval']]
```

In [ ]:
```
import dataframe_image as dfi
dfi.export(A_B_testing_results, "A_B_testing_results.png")
```

In [ ]:
```
plt.xticks(range(len(confidence_interval_testing)),list(confidence_interval_testing['Model']),rotation = 45,
                    ha="right")
plt.title('A-B Systems\' Confidence Intervals testing')
for c1,c2,b in zip(confidence_interval_testing['C1'],confidence_interval_testing['C2'],
                   range(len(confidence_interval_testing))):
    color='#2187bb'
    horizontal_line_width=.1
    left = b - horizontal_line_width / 2
    right = b + horizontal_line_width / 2
    #plt.plot((b,b),(c1,c2),'ro-',color='orange')
    plt.plot([b, b], [c2, c1], color=color)
    plt.plot([left, right], [c2, c2], color=color)
    plt.plot([left, right], [c1, c1], color=color)
    plt.plot(b, A_B_testing_results['RMSE'][b], 'o', color='#f44336')

#plt.xticks(range(len(confidence_interval_results)),list(confidence_interval_results['Model']))
plt.show()
```


A-B Systems' Confidence Intervals testing
0.15
0.14
0.13
0.12
0.11
0.10
0.09
0.08
0.07
GB_Model_With_SMOTE
GB_Model_Without_SMOTE