Project Name -

BIKE RENTING

by

Harsh Agarwal

INTRODUCTION

Project Description:

Bike sharing systems are a means of renting bicycles where the process of obtaining membership, rental, and bike return is automated via a network of kiosk locations throughout a city. Using these systems, people are able rent a bike from a one location and return it to a different place on an as-needed basis. Currently, there are over 500 bike-sharing programs around the world.

Problem Statement:

The objective of this Case is to predict bike rental count on daily basis based on the environmental and seasonal settings.

Data:

Dataset provided contains 731 observations over 2 years with a total 16 variables.

instant	dteday	season	yr	mnth	holiday	weekday	workingday	weathersit
1	2011-01-01	1	0	1	0	6	0	2
2	2011-01-02	1	0	1	0	0	0	2
3	2011-01-03	1	0	1	0	1	1	1
4	2011-01-04	1	0	1	0	2	1	1
5	2011-01-05	1	0	1	0	3	1	1
6	2011-01-06	1	0	1	0	4	1	1
7	2011-01-07	1	0	1	0	5	1	2
8	2011-01-08	1	0	1	0	6	0	2
9	2011-01-09	1	0	1	0	0	0	1
10	2011-01-10	1	0	1	0	1	1	1
11	2011-01-11	1	0	1	0	2	1	2
12	2011-01-12	1	0	1	0	3	1	1
13	2011-01-13	1	0	1	0	4	1	1

temp	atemp	hum	windspeed	casual	registered	cnt
0.344167	0.363625	0.805833	0.160446	331	654	985
0.363478	0.353739	0.696087	0.248539	131	670	801
0.196364	0.189405	0.437273	0.248309	120	1229	1349
0.2	0.212122	0.590435	0.160296	108	1454	1562
0.226957	0.22927	0.436957	0.1869	82	1518	1600
0.204348	0.233209	0.518261	0.0895652	88	1518	1606
0.196522	0.208839	0.498696	0.168726	148	1362	1510
0.165	0.162254	0.535833	0.266804	68	891	959
0.138333	0.116175	0.434167	0.36195	54	768	822
0.150833	0.150888	0.482917	0.223267	41	1280	1321
0.169091	0.191464	0.686364	0.122132	43	1220	1263

We have a total of 13 predictor variables and 3 target variables.

Predictors:
'instant' - index for the observations
'dteday' - date in yyyy:mm:dd format
‘season’- (1:springer, 2:summer, 3:fall, 4:winter)
‘yr’- Year (0: 2011, 1:2012)
‘mnth’- Month (1 to 12)
‘holiday’- weather day is holiday or not
‘weekday’- Day of the week
‘workingday’- If day is neither weekend nor holiday is 1, otherwise is 0.
‘Weathersit’-

- 1: Clear, Few clouds, Partly cloudy, Partly cloudy
- 2: Mist + Cloudy, Mist + Broken clouds, Mist + Few clouds, Mist
- 3: Light Snow, Light Rain+Thunderstorm+Scattered clouds,Light Rain+Scattered clouds
- 4: Heavy Rain + Ice Pallets + Thunderstorm + Mist, Snow + Fog

‘temp’- Normalized temperature in Celsius.
‘atemp’- Normalized feeling temperature in Celsius.
‘hum’- Normalized humidity. The values are divided to 100 (max)
‘windspeed’- Normalized wind speed. The values are divided to 67 (max)

Target Variable:
‘Casual’ (number of casual users)
‘Registered’ (number of registered users)
‘Cnt’ (total number of users)

As the target variables are continuous, its a regression problem
In order to predict the total number of bike rentals, we also need to observe the distributions of both casual and registered users and whether it will be a better approach to predict them individually over directly predicting the total count.

Data Exploration and Preparation

Data exploration and preprocessing can take upto 70% of the total project time and is a very important part as the quality of the input decides the quality of the output. We need to identify relationships between the predictor variables and the target variables, impute missing values, clean and standardize the data.

First we need to import the dataset..

```
dataset= pd.read_csv('day.csv')
```

Univariate Analysis:

Now, we need to explore all the variables one by one. We have a total of 13 predictors out of which 4 are continuous variables whereas 7 are discrete variables.
'Instant' and 'dteday' can be dropped from the dataset as we already have the information like weekday, month, year.

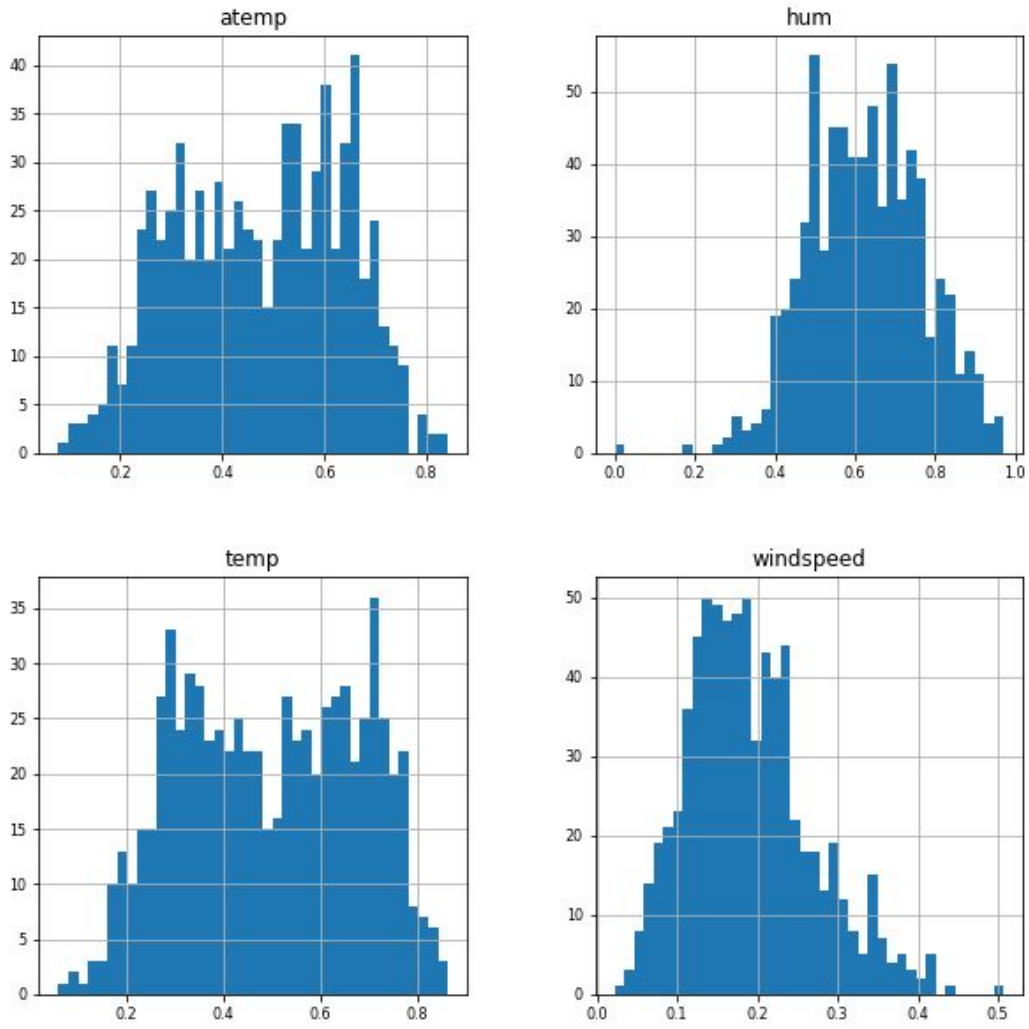
Below is the descriptive summary for the continuous variables including the target variables.

Index	temp	atemp	hum	windspeed	casual	registered	cnt
count	731	731	731	731	731	731	731
mean	0.4953847885	0.4743539886	0.6278940629	0.1904862116	848.1764706	3656.172367	4504.348837
std	0.1830509961	0.1629611784	0.1424290951	0.077497870…	686.6224883	1560.256377	1937.211452
min	0.0591304	0.0790696	0	0.0223917	2	20	22
25%	0.3370835	0.3378425	0.52	0.13495	315.5	2497	3152
50%	0.498333	0.486733	0.626667	0.180975	713	3662	4548
75%	0.6554165	0.608602	0.7302085	0.2332145	1096	4776.5	5956
max	0.861667	0.840896	0.9725	0.507463	3410	6946	8714

From the above table, we can conclude that there are no null values for numeric variables. The target variables have a high range of values with a high standard deviation.

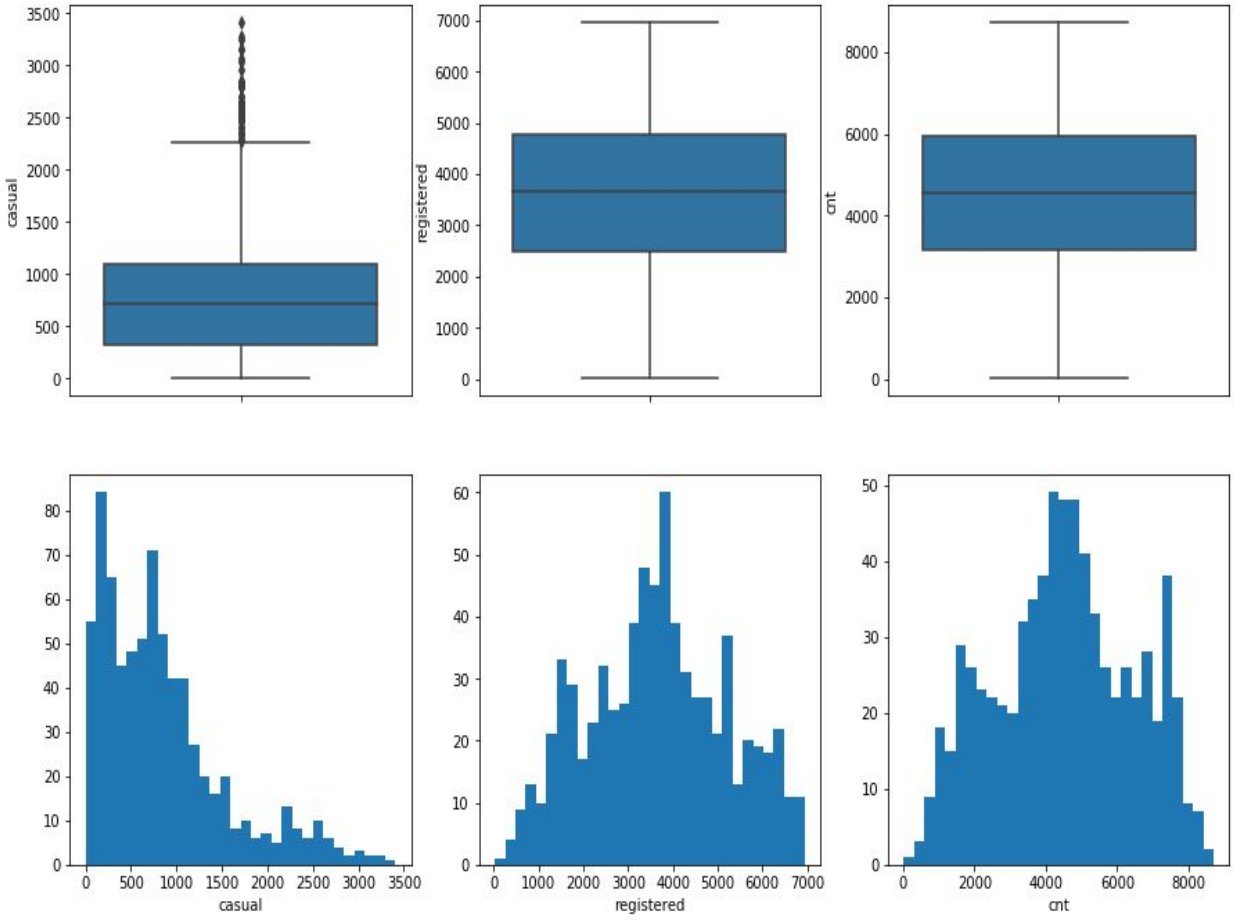
Also there's a minimum value of 0 for humidity variable which is next to impossible so we'll have to impute it or delete the observations with 0 humidity.

Below are the distributions for the numeric variables:



All the variables are normalized to have values between 0 and 1 and are well distributed with humidity and windspeed having a few extreme values. But they do seem to be natural outliers as windspeed can have high values on some days whereas humidity can drop to around 20% except for the one case where it is zero which needs to removed from the dataset.

Below is the distribution of the target variable.



‘Casual’ variable is left skewed with a few outliers as visible on the boxplot whereas both registered and cnt are normally distributed with no outliers.

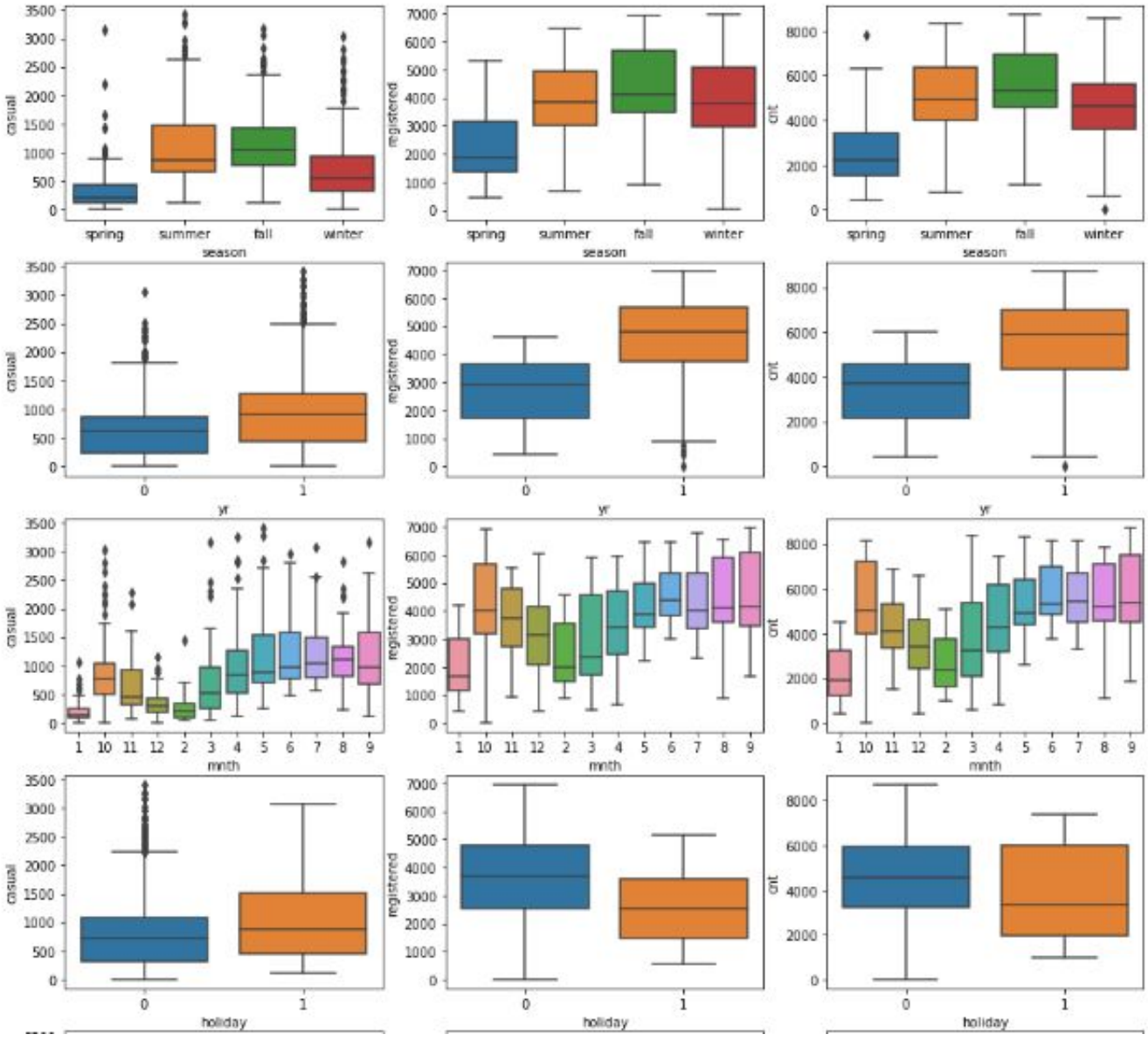
There is a difference between the distribution for both casual and registered users and it will be a better approach to predict them separately. It will become more clear once we observe their relations with predictors.

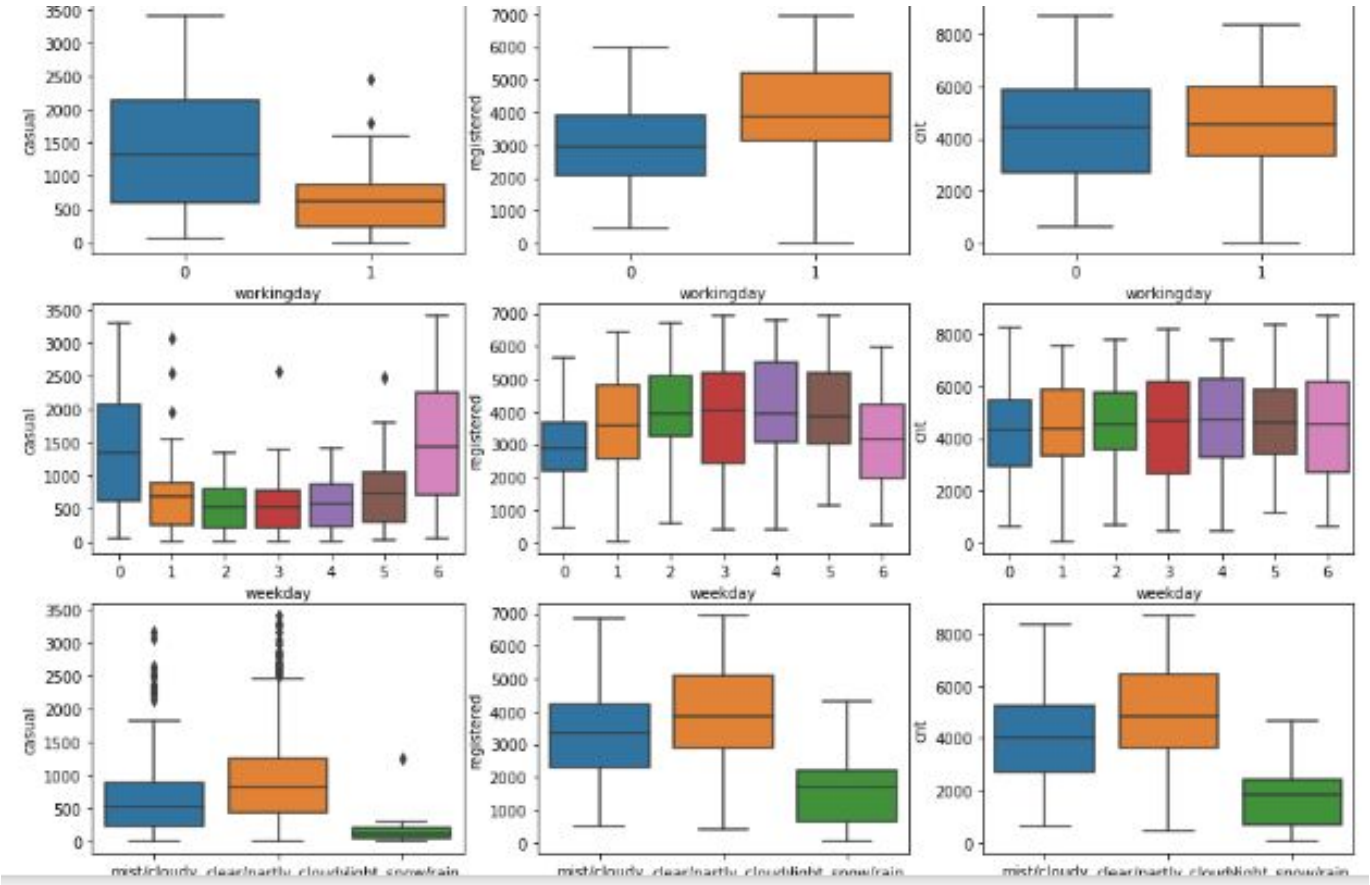
Also,we need to remove the outliers observed in casual users to get more accurate predictions.

Bi-Variate Analysis:

Bivariate analysis is used to find the relations between 2 variables, whether it be continuous or categorical.

First, we find the relation between the categorical variables and the 3 target variables

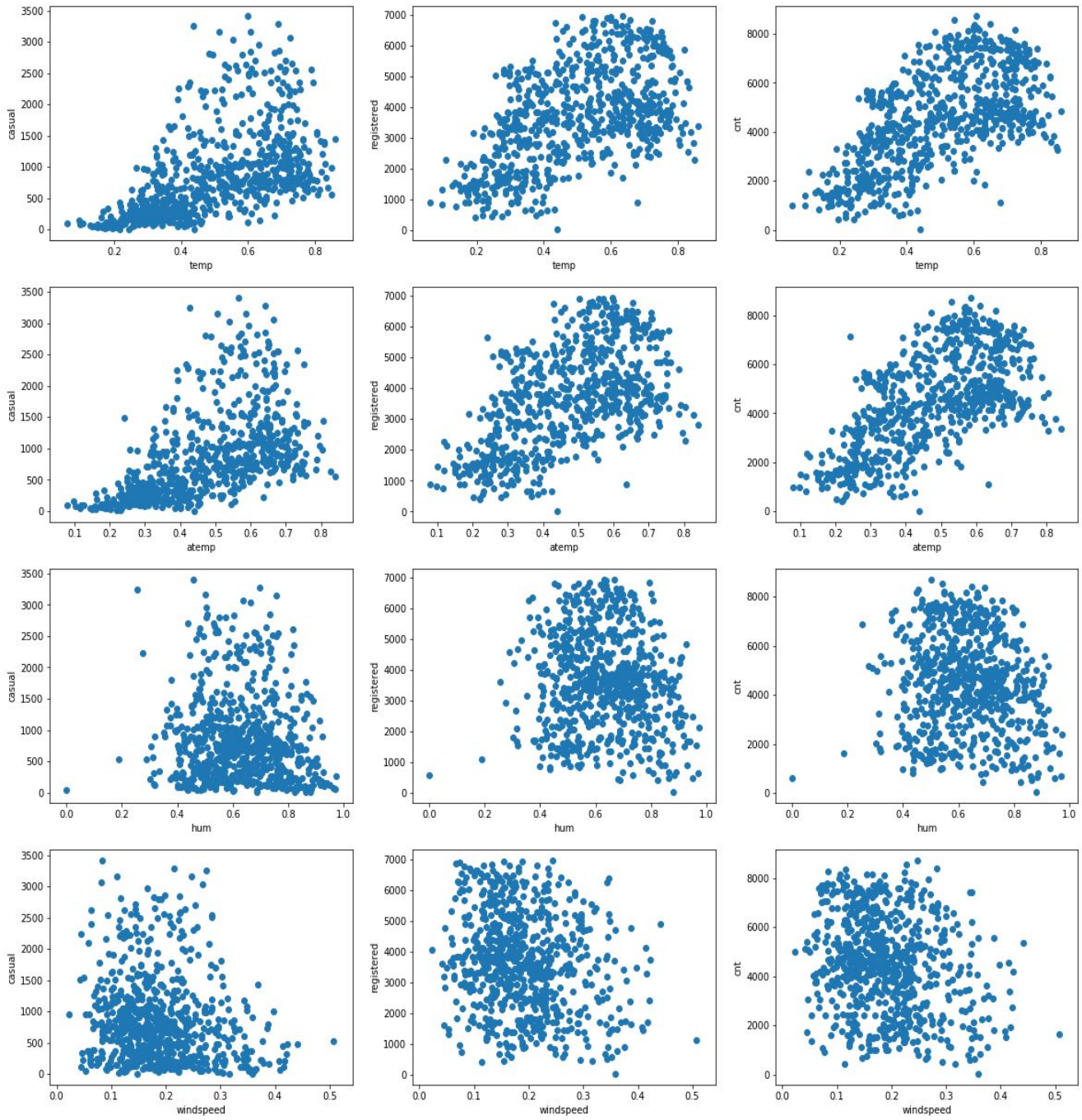




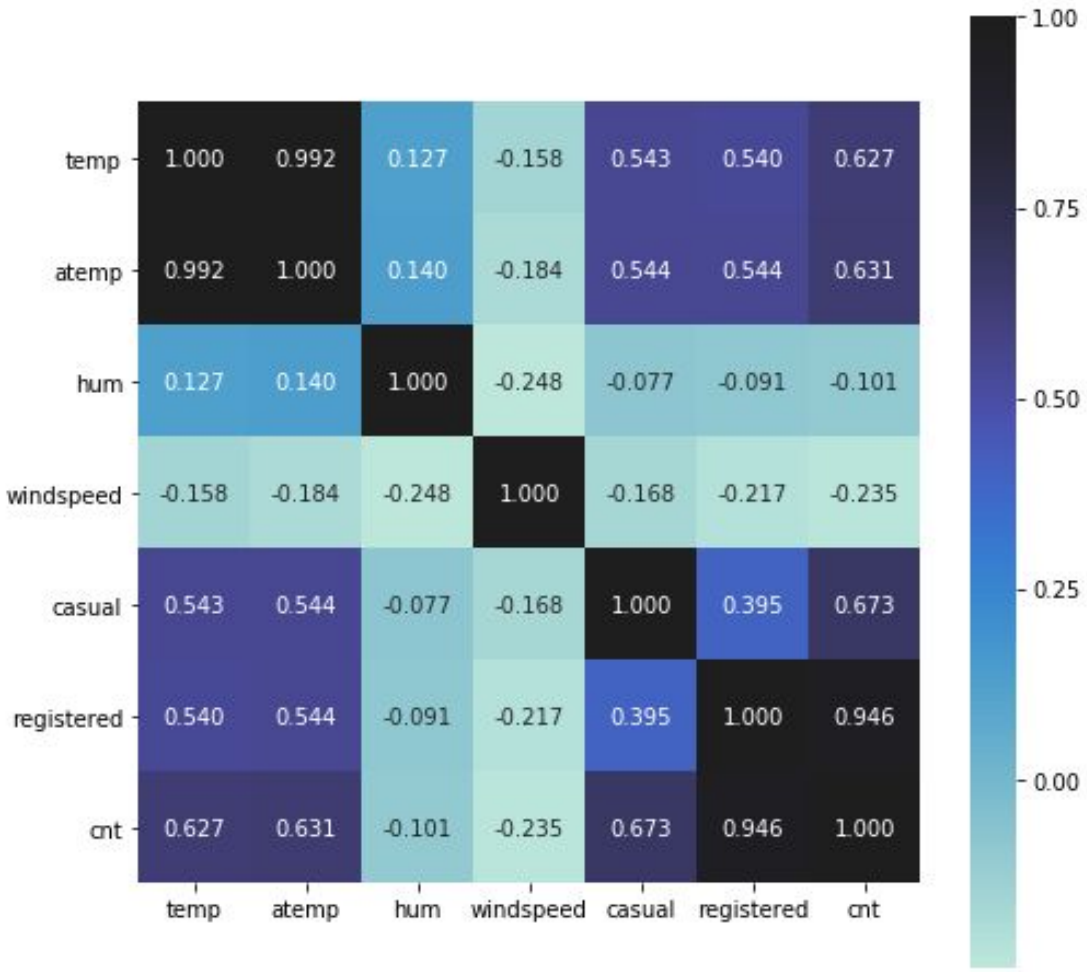
Key Points:

- There is increase in the number of users as we move from year 2011 to 2012.
- Number of casual users peak on holidays and weekends whereas the registered users peak on working days.
- Clear and partly cloudy weather has the highest rental count for both users
- There is a clear difference between the distribution of casual and registered users with most of the predictors and it will be more feasible to predict them separately.

Next we look at the relation between numerical variables and target variables using pairplots.



Both temp and atemp seem to have a positive relation with target variables whereas low windspeeds corresponds to high demand as expected. Humidity has more of a scattered distribution.



The above heatmap confirms the high positive correlation of the target variables with both temp and atemp variables.

Also, there is high multicollinearity between temp and atemp variables.
So, we need to remove one of them from the dataset.

Missing Values:

Next, we find the number of missing values in our dataset:

```
dataset.apply(lambda x: sum(x.isnull()))
```

The above line of code returns 0 for all the columns ie we have a complete dataset without any missing values.

Key points from Data Exploration phase:

- Need to predict casual and registered users separately
- Removed one observation having humidity=0
- Removed observations with outliers in casual variable
- Remove temp feature due to high collinearity with atemp.
- Created dummy variables for categorical features(season, weathersit, mnth, weekday)
- Total predictor variables=28 after creating dummy variables
- No need of feature scaling as all the numeric variables are already normalized.

Modeling:

As there was no test set, so we split the dataset into train and test in 9:1 ratio, with train dataset used to train and tune the model and test dataset to evaluate the model.

Evaluation Metrics:

Root mean squared error (RMSE) will be used to evaluate the model as it is sensitive to large deviations and both variables (casual and registered) have a large range.

K-Cross Validation and Grid Search:

Cross validation technique is used to evaluate the model by splitting the train dataset into K folds where each fold will be evaluated by training the model on the other folds and then a mean RMSE score for all the folds will be calculated.
The test set will be kept aside and will not be used in the training and parameters tuning process.
GridSearch is a tool with a built in cross-validation which is used to get the best combination of hyperparameters. It tries different values of each hyperparameters which we pass to it, perform k cross validation on each combination and returns the mean score. The combination of parameters with the best RMSE mean score are selected and the train data will be refitted with those parameters.

Custom Functions:

To reduce the redundancy in code while building and training different models, soe custom functions are defined in the code to perform repetitive tasks. These are

- Function **model_fit** will be used to build the perform cross validation and grid search on the train dataset and return the best parameters. It will also return the evaluation metric (RMSE) for both train and test datasets.
- Feature Importance function will display the most important features for both casual and registered target variables.
- Test_Set_Report will train the model with the optimal parameters and predict the rental count for casual and registered users separately on the test set. It will then add the result to get the final count which will be used to evaluate the model using RMSE metric.

Linear Regression

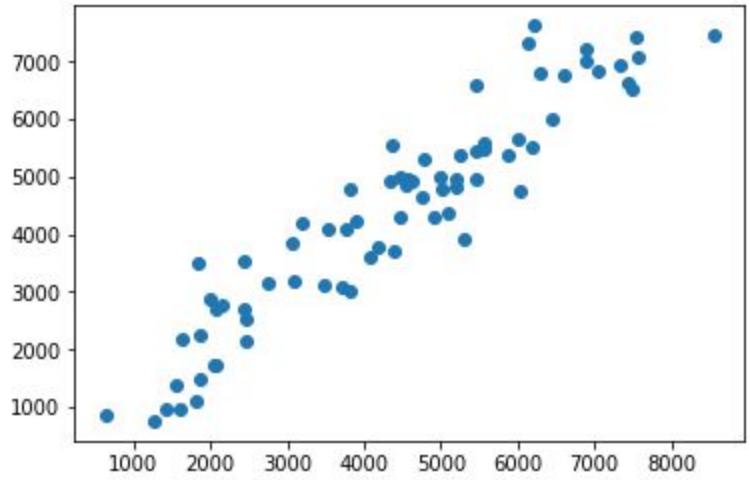
Linear regression algorithm was implemented without any penalization to predict both casual and registered users. RMSE was calculated for train and test fold for both casual and registered users.

Evaluation metrics for best model: Casual users
Train Metrics:
Root Mean Squared Error : 322.1500676631559
Test Metrics:
Root Mean Squared Error : 344.1733552455848

Evaluation metrics for best model: Registered users
Train Metrics:
Root Mean Squared Error : 600.2514611196042
Test Metrics:
Root Mean Squared Error : 635.7641433959247

The model was then implemented on the test set:

Test Set Report
Algorithm used: Linear Regression
RMSE: 644.1345334272082
RMSLE 0.20271328343593273



Linear regression gave an RMSE of about 644 on the test set.

Regularized Linear Regression - Ridge

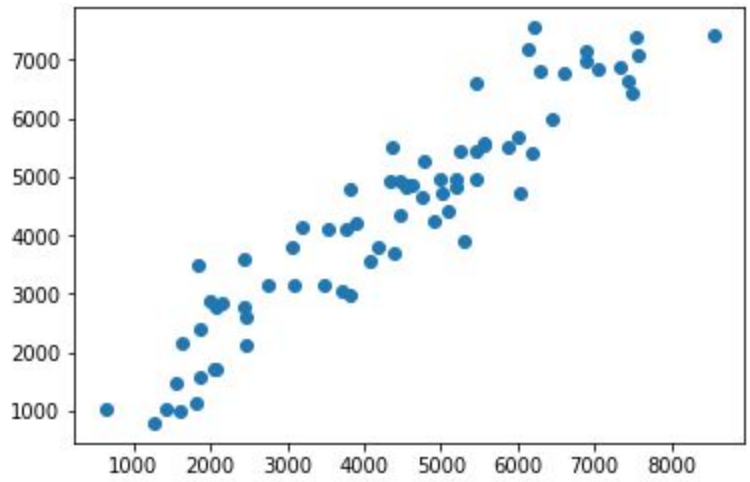
L2 regularization was applied with an optimal value of 0.5 to train for both the target variables. Below are the cross validation scores for casual and registered users.

Evaluation metrics for best model: Casual users
Train Metrics:
Root Mean Squared Error : 322.8342140415673
Test Metrics:
Root Mean Squared Error : 344.2808584073058

Evaluation metrics for best model: Registered users
Train Metrics:
Root Mean Squared Error : 601.5243943767695
Test Metrics:
Root Mean Squared Error : 635.7719982334485

There was no improvement over the basic model in terms of cross validation scores. Below is the summary of the test set.

Algorithm used: Regularized Linear Regression - Ridge
RMSE: 643.6012669018512
RMSLE 0.20465485580008339



The performance was similar to the basic model with a slight improvement of RMSE score from 644.14 to 643.6

Regularized Linear Regression - Lasso

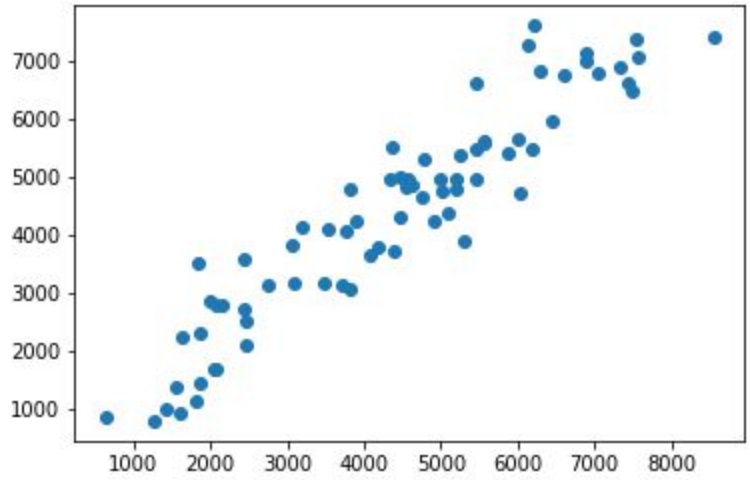
Value of alpha used =0.4

Evaluation metrics for best model: Casual users
Train Metrics:
Root Mean Squared Error : 322.7021886807988
Test Metrics:
Root Mean Squared Error : 344.4322810090644

Evaluation metrics for best model: Registered users
Train Metrics:
Root Mean Squared Error : 600.39738135461
Test Metrics:
Root Mean Squared Error : 634.797824426866

Test set report:

Algorithm used: Regularized Linear Regression - Lasso
RMSE: 649.9581133927114
RMSLE 0.20483939169073664



Lasso model seems to perform worse on the test set as compared to the above 2 models

Decision Trees:

After finding the optimal parameters using grid search, model was trained for both casual and registered variables and below are the cross validated scores:

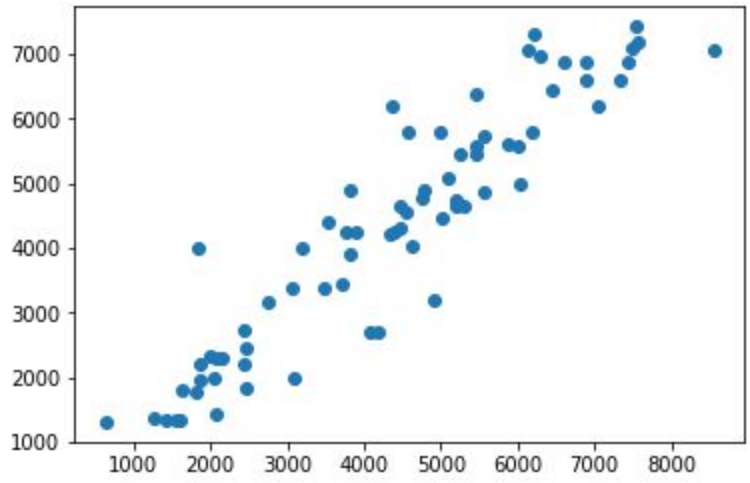
Evaluation metrics for best model: casual users
Train Metrics:
Root Mean Squared Error : 261.0027120456532
Test Metrics:
Root Mean Squared Error : 349.12357556881824

Evaluation metrics for best model: registered users
Train Metrics:
Root Mean Squared Error : 530.4005089096723
Test Metrics:
Root Mean Squared Error : 741.3405693394088

Cross validation scores were the worst for decision trees in comparison to the linear models.

Test Set Report:

Algorithm used: Decision Tree
RMSE: 694.5336155191507
RMSLE 0.20353927323617732



As was the case with cross validation scores, the test set also performed poorly with RMSE of 694 and there quite a few points visible on the scatter plot way off their true value.

Random Forests:

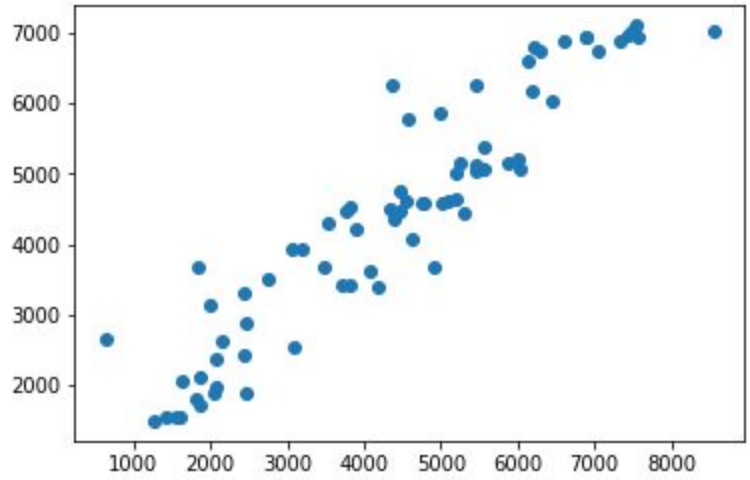
Evaluation metrics for best model: casual users
Train Metrics:
Root Mean Squared Error : 231.0829421560271
Test Metrics:
Root Mean Squared Error : 298.10276717017297

Evaluation metrics for best model: registered users
Train Metrics:
Root Mean Squared Error : 494.126576933345
Test Metrics:
Root Mean Squared Error : 626.9028437874638

Random forests returned the best cross validation scores among all the models so far.

Test Set Report:

Algorithm used: Random Forest
RMSE: 672.9816650618186
RMSLE 0.23692527965779522



Contrary to the cross validation scores, the test set still performs poorly as compared to linear model which had a RMSE of 643.

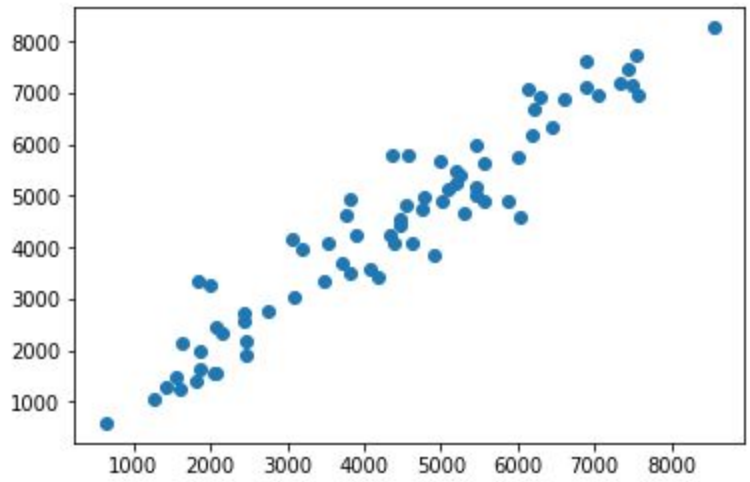
XGBOOST:

Evaluation metrics for best model: casual users
Train Metrics:
Root Mean Squared Error : 90.43749158350722
Test Metrics:
Root Mean Squared Error : 262.1814826791478

Evaluation metrics for best model:registered users
Train Metrics:
Root Mean Squared Error : 193.96347130957616
Test Metrics:
Root Mean Squared Error : 528.2576763743888

Test Set Report:

Algorithm used: XGBoost
RMSE: 574.7812281384278
RMSLE 0.1643569958789054



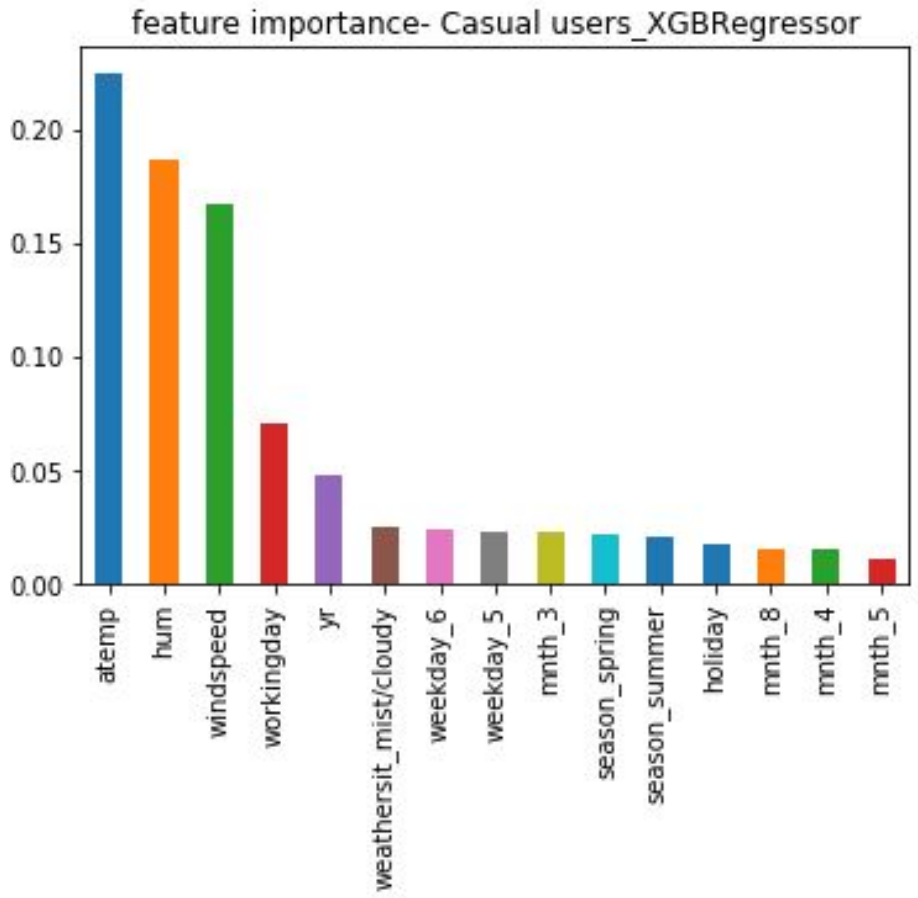
XGboost algorithm returned the best cross validation as well as the test scores so far with an RMSE of 574 which is way better then the next best model which is Ridge Regression with an RMSE of 643.

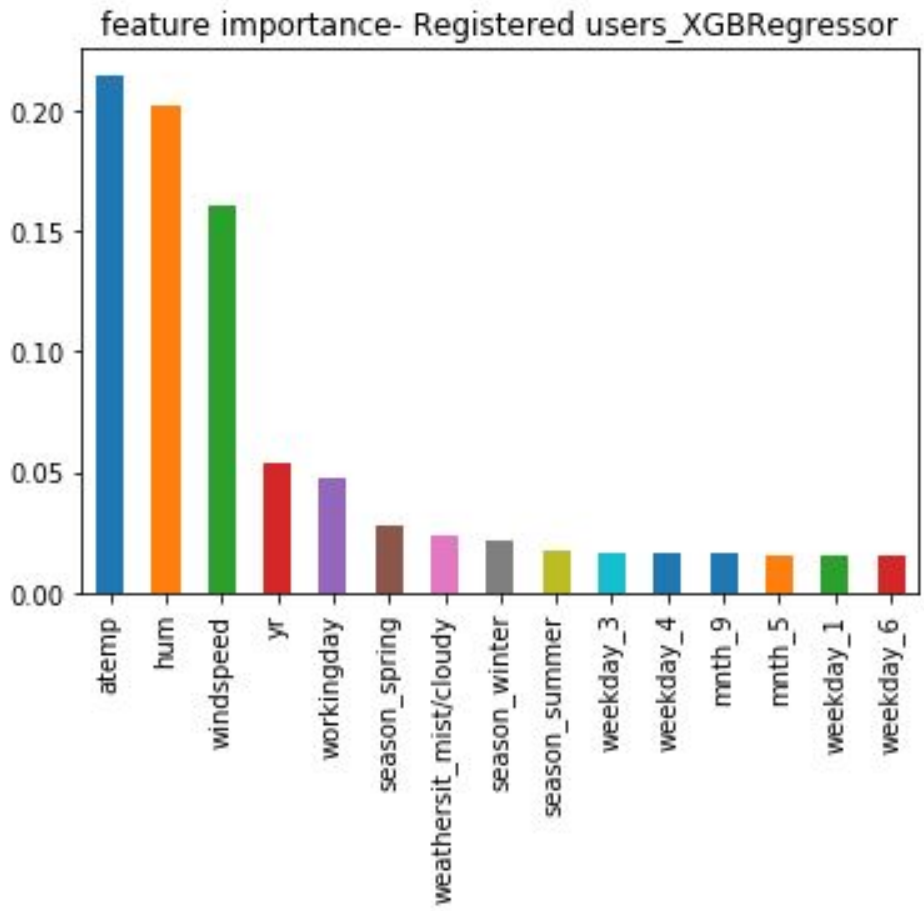
CONCLUSION:

Model Selection:

After comparing the RMSE scores for all the models on the test dataset, model implementing xgboost returned the best numbers with an RMSE of approx 574, therefore, it is selected as the final model for predicting the bike rental count.

Below are the top 15 most important features which were used to predict the casual and registered users respectively.





The top 5 variables in predicting the rental count is same for the both the variables atemp coming out as the most important feature.
The test set along with the predicted rental count are stored in a csv file.

casual	registered	cnt	Predicted Count
795	6898	7693	7485.052
190	2743	2933	2888.323
1338	2482	3820	4093.521
611	5592	6203	6160.992
847	3554	4401	3863.6414
515	2874	3389	3094.2544
1077	6365	7442	7021.003
921	5865	6786	5919.866
1499	2809	4308	5014.956

APPENDIX A - Python code

```
# importing required libraries
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt

from sklearn.model_selection import train_test_split
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import mean_squared_error
from sklearn.metrics import mean_squared_log_error
from sklearn.linear_model import LinearRegression
from sklearn.ensemble import RandomForestRegressor
from sklearn.linear_model import Ridge
from sklearn.linear_model import Lasso
from sklearn.tree import DecisionTreeRegressor
import xgboost as xgb
from xgboost.sklearn import XGBRegressor

import os
os.environ['KMP_DUPLICATE_LIB_OK']='True'

# importing dataset and storing as dataframe
dataset = pd.read_csv("day.csv")
dataset_copy = dataset[:]

# dropping dteday and instant variable
dataset.drop(columns=['instant','dteday'],axis=1, inplace=True)

#convert discreet variables  to categorical:
cat_features = ['season','yr','mnth','holiday','workingday','weekday','weathersit']
for ftr in cat_features:
    dataset[ftr] = dataset[ftr].astype(str)
    print(dataset[ftr].value_counts())

# mapping numerical categories to labels for better representation
dataset['season'] = dataset['season'].map({'1':'spring','2':'summer','3':'fall','4':'winter'})
dataset['weathersit'] =
dataset['weathersit'].map({'1':'clear/partly_cloudy','2':'mist/cloudy','3':'light_snow/rain','4':
'heavy_rain/snow'})
```

```
# Missing value analysis
missing_values = dataset.apply(lambda x: sum(x.isnull()))

# descriptive summary of the numerical variables
data_summary = dataset.describe()

# observing the distribution of target variable using boxplots and histograms
target_var = dataset.select_dtypes(include = ['int64'])

fig, axes = plt.subplots(2,3, figsize=(15,10))
for i,pred in enumerate(target_var.columns):
    bxplt = sns.boxplot(orient='v',data= target_var, y=pred, ax=axes[0,i])
    plt.sca(axes[1,i])
    plt.hist(dataset[pred],bins=30)
    plt.xlabel(pred)

# plotting histograms to observe the distributions for all numeric variables
features_numeric = dataset.select_dtypes(include = ['float64'])
features_numeric.hist(figsize=(10, 10), bins=40, xlabelsize=8, ylabelsize=8);



# plotting scatterplos to observe relation between continous variables
fig, axes = plt.subplots(4,3, figsize=(20,20))
for i,pred in enumerate(features_numeric.columns):
    plt.sca(axes[int(i%4),0])
    plt.scatter(x=pred,y='casual',data=dataset)
    plt.xlabel(pred)
    plt.ylabel('casual')
    plt.sca(axes[int(i%4),1])
    plt.scatter(x=pred,y='registered',data=dataset)
    plt.xlabel(pred)
    plt.ylabel('registered')
    plt.sca(axes[int(i%4),2])
    plt.scatter(x=pred,y='cnt',data=dataset)
    plt.xlabel(pred)
    plt.ylabel('cnt')
```

```
# plotting boxplots for casual and registerd users for categorical variables
fig, axes = plt.subplots(7,3, figsize=(15,25))
for i,pred in enumerate(cat_features):
    bxplt1 = sns.boxplot(data= dataset, x=pred,y=dataset['casual'], ax=axes[int(i%7),0])
    bxplt1.set(xlabel=pred)
    bxplt2 = sns.boxplot(data= dataset, x=pred,y=dataset['registered'], ax=axes[int(i%7),1])
    bxplt2.set(xlabel=pred)
    bxplt3 = sns.boxplot(data= dataset, x=pred,y=dataset['cnt'], ax=axes[int(i%7),2])
    bxplt3.set(xlabel=pred)


# plotting correlation matrix heatmap for numeric predictors
corr= dataset.corr()
f, ax = plt.subplots(figsize=(8, 8))
sns.heatmap(corr, center=1, square=True, annot=True, fmt='0.3f');

# one hot encoding the below features
dataset = pd.get_dummies(dataset,dtype=int, drop_first=True,
columns=['season','weathersit','mnth','weekday'])
dataset[['yr','holiday','workingday']] = dataset[['yr','holiday','workingday']].astype(int)

# dropping temp feature and keeping atemp since both are highly correlated
dataset.drop(columns=['temp'],axis=1, inplace=True)

# dropping observation containing humidity=0 which is next to impossible
temp_data = dataset[dataset['hum']!=0]

# dropping observations containing outliers for casual variable for better accuracy
temp_data =
temp_data.loc[(temp_data['casual']-temp_data['casual'].mean())<=(3*temp_data['casual'].
std())]

# splitting dataset into train and test
train ,test = train_test_split(temp_data, test_size= 0.1, random_state=42)

# further splitting the dataset to separate the target variable from the predictors
FinalResult_Python= dataset_copy.loc[pd.Series(test.index),]
ytrain_casual = train['casual']
ytrain_registered = train['registered']
ytrain_count = train['cnt']
xtrain = train.drop(columns=['casual','registered','cnt'], axis=1)
```

```
ytest_casual = test['casual']
ytest_registered = test['registered']
ytest_count = test['cnt']
ytest_count=ytest_count.reset_index()
ytest_count.drop(columns='index',axis=1, inplace=True)
xtest = test.drop(columns=['casual','registered','cnt'], axis=1)
xtest=xtest.reset_index()
xtest.drop(columns='index',axis=1, inplace=True)

# storing all predictor names in a variable to be used later
predictors = [x for x in xtrain.columns]


''' Generic function to cross validate and tune hyparameters. It will display the
 train and test scores for evaluation metric RMSE and returns best parameters '''
# algo - regression object
# x_train - train dataset
# y_train - target variable corresponding to train dataset
# params - paramgrid containing different combinations to be tested using grid search
# useparams - boolean argument which on False sets param_grid to empty, so that only
# cross validation is performed
def model_fit(algo, x_train, y_train, params, useparams):

    if(useparams==False):
        params={}
    gridsearch= GridSearchCV(algo, params, scoring='neg_mean_squared_error', cv=5,
refit='neg_mean_squared_error', verbose=1, return_train_score=True)
    gridsearch= gridsearch.fit(x_train, y_train)

    print ("\nModel Report-",str(algo)[0:str(algo).find('(')])
    print("estimator:",gridsearch.best_estimator_)
    print()
    print("Best params:",gridsearch.best_params_)
    print()
    print("Evaluation metrics for best model:")
    print("Train Metrics:")
    print("Root Mean Squared Error
:",np.sqrt(np.abs(gridsearch.cv_results_['mean_train_score'].max())),"\n")
    print("Test Metrics:")
    print("Root Mean Squared Error :",np.sqrt(np.abs(gridsearch.best_score_)),"\n")

    return gridsearch.best_estimator_
```

```
''' generic function to print Report for the test dataset for different models after the
 hyperparameters have been tuned '''
# algo - regression object
# xtest - test dataset
# ytest - actual values for target variable corresponding to test dataset
# returns a Series containing predicted count values
def Test_Set_Report(algoname,algo):

    algo.fit(xtrain,ytrain_casual)
    ypred_casual= algo.predict(xtest)

    algo.fit(xtrain,ytrain_registered)
    ypred_registered= algo.predict(xtest)

    ypred_count = ypred_casual + ypred_registered

    print ("\nTest Set Report\n")
    print("Algorithm used:",algoname)
    print("RMSE:",np.sqrt(mean_squared_error(ytest_count, ypred_count)))
    print("RMSLE",np.sqrt(mean_squared_log_error(ytest_count,(ypred_count))))

    plt.scatter(ytest_count,ypred_count)

    return pd.Series(ypred_count)


''' generic function to print feature importance for the best 15 predictors '''
# algo - regression object
# returns names of top 15 predictors
def print_feature_importance(algo):

    algo.fit(xtrain,ytrain_casual)
    ftimp1 = pd.Series(algo.feature_importances_,
predictors).sort_values(ascending=False)
    ftimp1=ftimp1.nlargest(n=15)
    plt.figure()
    ftimp1.plot(kind='bar', title='_'.join(['feature importance- Casual
users',str(algo)[0:str(algo).find('(')]]))

    algo.fit(xtrain,ytrain_registered)
```

```
    ftimp2 = pd.Series(algo.feature_importances_,
predictors).sort_values(ascending=False)
    ftimp2=ftimp2.nlargest(n=15)
    plt.figure()
    ftimp2.plot(kind='bar', title='_'.join(['feature importance- Registered
users',str(algo)[0:str(algo).find('(')]]))

    return pd.concat([pd.Series(ftimp1.index),pd.Series(ftimp2.index)], axis=1)
```

```
''' LINEAR REGRESSION '''

linreg = LinearRegression()

linreg = model_fit(linreg, xtrain, (ytrain_casual),{},False)
linreg = model_fit(linreg, xtrain, (ytrain_registered),{},False)

ypred_lr_count = Test_Set_Report("Linear Regression",linreg)

result_linear = pd.concat([ytest_count,pd.Series(ypred_lr_count)], axis=1)
```

```
''' REGULARIZED LINEAR REGRESSION - RIDGE'''
ridge = Ridge(random_state=42, alpha=0.5)
param_ridge = {'alpha':[0.1,0.2,0.3,0.4,0.5,0.7,1,1.2,1.5,1.8,2]}

ridge = model_fit(ridge,xtrain,ytrain_registered,param_ridge,False)
ridge = model_fit(ridge,xtrain,ytrain_casual,param_ridge,False)

ypred_ridge_count = Test_Set_Report("Regularized Linear Regression - Ridge",ridge)

result_ridge = pd.concat([ytest_count,pd.Series(ypred_ridge_count)], axis=1)
```

```
''' REGULARIZED LINEAR REGRESSION - LASSO'''
lasso = Lasso(random_state=42, alpha=0.4)
param_lasso = {'alpha':[0.1,0.2,0.3,0.4,0.5,0.7,1,1.2,1.5,1.8,2]}

lasso = model_fit(lasso,xtrain,(ytrain_casual),param_lasso,False)
```

```
lasso = model_fit(lasso,xtrain,(ytrain_registered),param_lasso,False)

ypred_lasso_count = Test_Set_Report("Regularized Linear Regression - Lasso",lasso)

result_lasso = pd.concat([ytest_count,pd.Series(ypred_lasso_count)], axis=1)



''' DECISION TREE REGRESSOR '''

dectree =
DecisionTreeRegressor(max_depth=8,random_state=42,min_samples_split=25,max_leaf_
nodes=50, max_features=0.6)
param_dectree= {'max_depth':[4,5,6,7,8,9] }

dectree = model_fit(dectree, xtrain, ytrain_casual, param_dectree, False)
dectree = model_fit(dectree, xtrain, ytrain_registered, param_dectree, False)

ypred_dectree_count = Test_Set_Report("Decision Tree",dectree)

result_dectree = pd.concat([ytest_count,pd.Series(ypred_dectree_count)], axis=1)



''' RANDOM FOREST REGRESSOR '''
ranfor =
RandomForestRegressor(n_estimators=60,criterion='mse',max_features=0.5,max_leaf_n
odes=30, max_depth=6, min_samples_split=20,random_state=42)
param_ranfor= {'max_leaf_nodes':[10,20,30,40,50],'min_samples_split':[20,30,40,50]}

ranfor = model_fit(ranfor, xtrain, ytrain_casual, param_ranfor, False)
ranfor = model_fit(ranfor, xtrain, ytrain_registered, param_ranfor, False)

ypred_ranfor_count = Test_Set_Report("Random Forest",ranfor)

result_ranfor = pd.concat([ytest_count,pd.Series(ypred_ranfor_count)], axis=1)
```

```
''' XGBOOST REGRESSOR '''

xgbc= XGBRegressor(n_estimators=300, min_child_weight=2, max_depth=4,
colsample_bylevel=0.5, gamma=0.1, reg_alpha=1, subsample=0.8, learning_rate=0.05,
random_state=42)

param_xgb={'min_child_weight':[1,2,3,4,5]}
xgbc=model_fit(xgbc, xtrain, ytrain_casual, param_xgb, False)
xgbc=model_fit(xgbc, xtrain, ytrain_registered, param_xgb, False)

ypred_xgb_count = Test_Set_Report("XGBoost",xgbc)

result_xgb = pd.concat([ytest_count,pd.Series(ypred_xgb_count)], axis=1)

# xgb.cv is used to get the actual number of n_estimators required based on the learning
rate,
# it uses early_stopping_rounds to get the optimal value
xdtrain = xgb.DMatrix(xtrain,label=ytrain_casual)
cvresult_xgb=xgb.cv(xgbc.get_xgb_params(),
xdtrain,nfold=5,num_boost_round=5000,metrics='rmse',early_stopping_rounds=50)

bestpred_xgb = print_feature_importance(xgbc)



''' Storing the predicted results along with the actual prediction for each phone number in
a csv file'''

FinalResult_Python= FinalResult_Python.reset_index()
FinalResult_Python.drop(columns=['index'],axis=1,inplace=True)
FinalResult_Python = pd.concat([FinalResult_Python,result_xgb[0]], axis=1)
FinalResult_Python.rename(columns={0:'Predicted Count'}, inplace=True)
FinalResult_Python.to_csv("PredictedRentalCount_Python.csv", index=False)
```

APPENDIX B - R Code

```
## libraries used
#randomForest, rpart, glmnet, lm, xgboost, data.table, mlr, ggplot2, gridextra

# set working directory
setwd("Downloads/Data Science Problem sets/Bike_Renting")

# importing test and train data into dataframes by and removing whitespaces
dataset = read.csv("day.csv", header = T, strip.white= T )
dataset_copy = data.frame(dataset)

# removing dteday and instant variable from the dataset
dataset = within(dataset, rm(dteday,instant))

#convert discreet variables  to categorical:
cat_features = c('season','yr','mnth','holiday','workingday','weekday','weathersit')
for(predictor in cat_features)
{
  dataset[[predictor]]=as.factor(dataset[[predictor]])
}

# mapping numerical categories to labels for better representation
levels(dataset$season) = c('spring','summer','fall','winter')
levels(dataset$weathersit) =
c('clear.partly_cloudy','mist.cloudy','light_snow.rain','heavy_rain.snow')

# getting the descriptive statistics of the dataset
summary(dataset)

# histograms and boxplots to observe distribution for target
variable(casual,registered,count)
for (feature in colnames(dataset)){
  if(class(dataset[[feature]])=='integer' ){
    gg1=ggplot(dataset,aes(x=dataset[[feature]]))+geom_histogram()+labs(x=feature)
    print(gg1)
    gg2=
ggplot(dataset,aes(y=dataset[[feature]]))+geom_boxplot(fill="#4271AE")+labs(y=feature)
    print(gg2)
  }
}
```

```
# histograms to observe distribution for all numeric predictors
for (feature in colnames(dataset)){
  if(class(dataset[[feature]])=='numeric' ){
    gg=ggplot(dataset,aes(x=dataset[[feature]]))+geom_histogram()+labs(x=feature)
    print(gg)
  }
}

# plotting scatterplos to observe relation between continous variables
for (feature in colnames(dataset)){
  if(class(dataset[[feature]])=='numeric' ){

gg1=ggplot(dataset,aes(x=dataset[[feature]],y=dataset$casual))+geom_point()+labs(y='ca
sual',x=feature)

gg2=ggplot(dataset,aes(x=dataset[[feature]],y=dataset$registered))+geom_point()+labs(y
='registered',x=feature)

gg3=ggplot(dataset,aes(x=dataset[[feature]],y=dataset$cnt))+geom_point()+labs(y='count
',x=feature)
    grid.arrange(gg1,gg2,gg3,ncol=1)
  }
}


# plotting boxplots for casual and registerd users for categorical variables
for (feature in colnames(dataset)){
  if(class(dataset[[feature]])=='factor' ){

gg1=ggplot(dataset,aes(x=dataset[[feature]],y=dataset$casual))+geom_boxplot(fill="#427
1AE")+labs(y='casual',x=feature)

gg2=ggplot(dataset,aes(x=dataset[[feature]],y=dataset$registered))+geom_boxplot(fill="#
4271AE")+labs(y='registered',x=feature)

gg3=ggplot(dataset,aes(x=dataset[[feature]],y=dataset$cnt))+geom_boxplot(fill="#4271A
E")+labs(y='count',x=feature)
    grid.arrange(gg1,gg2,gg3,ncol=1)
  }
}
```

```
# finding the count of missing values
missing_values = data.frame(c(sapply(dataset, function(x)  sum(is.na(x)))))
colnames(missing_values) = 'Count'

# correlation matrix of all numeric predictors and target variable
dataset_corr = data.frame(lapply(dataset, function(x) {
  if(is.integer(x)) as.numeric(x) else  x}))
correlation_matrix = cor(dataset_corr[sapply(dataset_corr, is.numeric)], method =
'pearson')

# one hot encoding the below features
dummy_var = c('season','weathersit','mnth','weekday')
for(predictor in dummy_var){
for(unique_value in unique(dataset[[predictor]])){
  dataset[paste(predictor, unique_value, sep = ".")] = ifelse(dataset[[predictor]] ==
unique_value, 1, 0)
}}
dataset = within(dataset,rm(season,weathersit,mnth,weekday))


for(predictor in c('yr','holiday','workingday'))
{
  dataset[[predictor]]=as.integer(dataset[[predictor]])
}

# dropping temp feature and keeping atemp since both are highly correlated
dataset = within(dataset,rm(temp))

# dropping observation containing humidity=0 which is next to impossible
temp_data = dataset[dataset$hum!=0,]

# dropping observations containing outliers for casual variable for better accuracy
temp_data =
temp_data[(temp_data$casual-mean(temp_data$casual))<=(3*sd(temp_data$casual)),]
dataset_copy =
dataset_copy[(dataset_copy$casual-mean(dataset_copy$casual))<=(3*sd(dataset_copy$c
asual)),]


# splitting dataset into train and test
smp_size = floor(0.9 * nrow(temp_data))
set.seed(42)
train_ind = sample(seq_len(nrow(temp_data)), size = smp_size)
```

```
train = temp_data[train_ind, ]
test = temp_data[-train_ind, ]
FinalResult_R <- cbind(NA, NA)
FinalResult_R <- cbind(dataset_copy[-train_ind,])

ytest_count = data.frame(actual_count=test$cnt)
train_casual = within(train,rm(registered,cnt))
test_casual = within(test,rm(registered,cnt))

train_registered = within(train,rm(casual,cnt))
test_registered = within(test,rm(casual,cnt))


# creating training and testing tasks for casual users
traintask_casual = makeRegrTask(data= train_casual , target= 'casual' )
testtask_casual = makeRegrTask(data= test_casual , target= 'casual' )

# creating training and testing tasks for registered users
traintask_registered = makeRegrTask(data= train_registered , target= 'registered' )
testtask_registered = makeRegrTask(data= test_registered , target= 'registered' )

# generic function for tuning hyperparameters using gridsearch
model_tuning <- function(algoname, algo, params_tune, traintask ){
  set.seed(42)
  gridsearch = tuneParams(algo , resampling = makeResampleDesc("CV",iters = 5L,
predict = "both"),
                          task= traintask, par.set = params_tune,
                          measures = list(rmse,setAggregation(rmse, train.mean)) ,
                          control = makeTuneControlGrid()
  )

  cat("\nGrid Search Report-",algoname,"\n")
  cat("Best params:\n")
  print(as.matrix(gridsearch$x))

  cat("\nEvaluation metrics for best model:Train Fold\n")
  cat("RMSE:",gridsearch$y[2],"\n")
  cat("\nEvaluation metrics for best model:Test Fold\n")
  cat("RMSE:",gridsearch$y[1],"\n")

  return(gridsearch$x)
}
```

```
# generic function to display cross validation scores for train dataset
cross_validation <- function(algoname, algo, traintask){
  set.seed(42)
  r=resample(algo, traintask, makeResampleDesc("CV",iters=5, predict="both"),
             measures = list(rmse,setAggregation(rmse, train.mean)) )

  cat("\nResampling Report-",algoname,"\n")
  cat("\nEvaluation metrics for best model:Train Fold\n")
  cat("RMSE:",r$aggr[2],"\n")

  cat("\nEvaluation metrics for best model:Test Fold\n")
  cat("RMSE:",r$aggr[1],"\n")
}


# generic function to evaluate the test set using optimal parameters
Test_Set_Report <- function (algoname, algo){

  model_fit = train(algo, traintask_casual)
  ypred_casual = predict(model_fit , testtask_casual)

  model_fit = train(algo, traintask_registered)
  ypred_registered = predict(model_fit , testtask_registered)

  ypred_count = data.frame(pred_count=ypred_casual$data$response +
ypred_registered$data$response)

  cat("\nTest Set Report-",algoname,"\n")
  cat("\nEvaluation metrics:\n")
  cat("RMSE :",measureRMSE(ytest_count$actual_count,ypred_count$pred_count),"\n")
  cat("RMSLE
:",measureRMSLE(ytest_count$actual_count,ypred_count$pred_count),"\n")

  df = cbind(ytest_count,ypred_count)
  gg=ggplot(df, aes(x=actual_count, y=pred_count)) + geom_point()
  print(gg)
  return(df)
}


# function to generate feature importance graph
featureImportance <- function(algoname, algo, traintask){
  ftimp = data.frame(t((getFeatureImportance(train(algo, traintask)))$res))
```

```
  setDT(ftimp, keep.rownames = TRUE)[]
  colnames(ftimp)= c('Feature','Importance')
  ftimp = ftimp[order(ftimp$Importance, decreasing = T),]
  ftimp = ftimp[1:15,]
  ftimp = ftimp[order(ftimp$Importance, decreasing = F),]
  level_order = ftimp$Feature
  gg = ggplot(ftimp, aes(x=factor(Feature, level =
level_order),y=Importance))+geom_col()+labs(title=algoname, x='Feature')+coord_flip()
  print(gg)
  return(ftimp$Feature)
}


#### LINEAR REGRESSION ####
linreg_model = makeLearner("regr.lm", predict.type='response' )

# cross validation report
cross_validation("Linear Regression", linreg_model, traintask_casual )
cross_validation("Linear Regression", linreg_model, traintask_registered )

# evaluating the test set performance
ypred_count_linreg = Test_Set_Report("Linear Regression", linreg_model)



#### REGULARIZED LINEAR REGRESSION - RIDGE ####
ridge_model = makeLearner("regr.glmnet", predict.type='response' , par.vals =
list(lambda=0.5,alpha=0))

#Search for hyperparameters
params_ridge = makeParamSet(
  makeDiscreteParam("lambda",values=c(10))
)

# perform grid search to get optimal parameters
bestparams_ridge_casual = model_tuning("Ridge Regression", ridge_model,
params_ridge, traintask_casual)
bestparams_ridge_registered = model_tuning("Ridge Regression", ridge_model,
params_ridge, traintask_registered)

# cross validation report using optimal parameters
ridge_model = setHyperPars(ridge_model , par.vals = bestparams_ridge_casual)
cross_validation("Ridge Regression", ridge_model, traintask_casual)
```

```
cross_validation("Ridge Regression", ridge_model, traintask_registered)

# evaluating the test set performance
ypred_count_ridge = Test_Set_Report("Ridge Regression", ridge_model)


#### REGULARIZED LINEAR REGRESSION - LASSO ####
lasso_model = makeLearner("regr.glmnet", predict.type='response' , par.vals =
list(lambda=2,alpha=1))

#Search for hyperparameters
params_lasso = makeParamSet(
  makeDiscreteParam("lambda",values=c(0.1,0.2,0.5,1,2,3,5,6))
)

# perform grid search to get optimal parameters
bestparams_lasso_casual = model_tuning("Lasso Regression", lasso_model,
params_lasso, traintask_casual)
bestparams_lasso_registered = model_tuning("Lasso Regression", lasso_model,
params_lasso, traintask_registered)

# cross validation report using optimal parameters
lasso_model = setHyperPars(lasso_model , par.vals = bestparams_lasso_casual)
cross_validation("Lasso Regression", lasso_model, traintask_casual)
cross_validation("Lasso Regression", lasso_model, traintask_registered)

# evaluating the test set performance
ypred_count_ridge = Test_Set_Report("Ridge Regression", ridge_model)


#### DECISION TREE ####
# creating a learner
dectree_model = makeLearner("regr.rpart" , predict.type = "response", par.vals=
list(minbucket=20,maxdepth=15, minsplit=5) )

#Search for hyperparameters
params_dectree = makeParamSet(
  makeDiscreteParam("minsplit",values=c(3,4,5,6)),
  makeDiscreteParam("minbucket", values=c(15,20,25)),
  makeDiscreteParam("maxdepth", values=c(11,12,14,15))
)

# perform grid search to get optimal parameters
```

```
bestparams_dectree_casual = model_tuning("Decision Tree", dectree_model,
params_dectree, traintask_casual)
bestparams_dectree_registered = model_tuning("Decision Tree", dectree_model,
params_dectree, traintask_registered)

# cross validation report using optimal parameters
dectree_model = setHyperPars(dectree_model , par.vals = bestparams_dectree_casual)
cross_validation("Decison Tree", dectree_model, traintask_casual)
cross_validation("Decison Tree", dectree_model, traintask_registered)

# plot feature importance for decision trees
bestpred_dectree_casual= featureImportance("Decision Tree",dectree_model,
traintask_casual)
bestpred_dectree_registered= featureImportance("Decision Tree",dectree_model,
traintask_registered)

# evaluating the test set performance
ypred_count_dectree = Test_Set_Report("Decision Tree", dectree_model)



#### RANDOM FOREST CLASSIFIER ####
ranfor_model = makeLearner("regr.randomForest" , predict.type = "response", par.vals=
list(ntree= 60,mtry=12, nodesize= 20, maxnodes= 60, importance=T))

#Search for hyperparameters
params_ranfor = makeParamSet(
  #makeDiscreteParam("ntree",values=c(45,55,60,65)),
  #makeDiscreteParam('mtry', values=c(6,9,12,15)),
  makeDiscreteParam("nodesize", values=c(20,25,30)),
  makeDiscreteParam("maxnodes", values=c(50,55,60))
)

# perform grid search to get optimal parameters
bestparams_ranfor_casual = model_tuning("Random Forest", ranfor_model,
params_ranfor, traintask_casual)
bestparams_ranfor_registered = model_tuning("Random Forest", ranfor_model,
params_ranfor, traintask_registered)

# cross validation report using optimal parameters
ranfor_model = setHyperPars(ranfor_model , par.vals = bestparams_ranfor_casual)
cross_validation("Random Forest", ranfor_model, traintask_casual)
cross_validation("Random Forest", ranfor_model, traintask_registered)
```

```
# plot feature importance for random forest
bestpred_rf_casual= featureImportance("Random Forest",ranfor_model,
traintask_casual)
bestpred_rf_registered= featureImportance("Random Forest",ranfor_model,
traintask_registered)


# evaluating the test set performance
ypred_count_ranfor= Test_Set_Report("Random Forest", ranfor_model)




#### XGBOOST CLASSIFIER ####

#make learner with inital parameters
xgb_model <- makeLearner("regr.xgboost", predict.type = "response")
xgb_model$par.vals <- list(
  objective = "reg:linear",
  nrounds = 500,
  eta= 0.05,
  subsample= 0.9,
  colsample_bytree= 0.5,
  eval_metric="rmse",
  early_stopping_rounds=50,
  verbose=0,
  print_every_n = 25,
  max_depth= 4,
  min_child_weight = 2,
  gamma = 0.1,
  alpha = 1
)


#Search for hyperparameters
params_xgb = makeParamSet(
  makeDiscreteParam("max_depth",values=c(5)),
  makeDiscreteParam("gamma",values=c(0)),
  makeDiscreteParam('min_child_weight', values=c(1)),
  makeDiscreteParam("subsample", values=c(1)),
  makeDiscreteParam("colsample_bytree", values=c(0.8)),
  makeDiscreteParam("max_delta_step", values=c(0.32)),
```

```
  makeDiscreteParam("alpha", values=c(0.1,0.5,1)),
  makeDiscreteParam("lambda", values=c(0.1,0.5,1))
)


# perform grid search to get optimal parameters
bestparams_casual_xgb = model_tuning("XGBoost", xgb_model, params_xgb,
traintask_casual)
bestparams_registered_xgb = model_tuning("XGBoost", xgb_model, params_xgb,
traintask_registered)

# cross validation report using optimal parameters
xgb_model = setHyperPars(xgb_model , par.vals = bestparams_casual_xgb)
cross_validation("XGBoost", xgb_model, traintask_casual)
cross_validation("XGBoost", xgb_model, traintask_registered)

# plot feature importance for random forest
bestpred_xgb_casual = data.frame(featureImportance("XGBoost",xgb_model,
traintask_casual))
bestpred_xgb_registered = data.frame(featureImportance("XGBoost",xgb_model,
traintask_registered))


# evaluating the test set performance
ypred_count_xgb = Test_Set_Report("XGBoost", xgb_model)


### Storing the predicted rental count along with the actual count for each day in a csv
file
FinalResult_R = cbind(FinalResult_R, ypred_count_xgb$pred_count)
colnames(FinalResult_R)[17]= c( 'Predicted_Count')
write.csv(FinalResult_R, file="PredictedRentalCount_R.csv", row.names = F)
```