Bike Sharing System

Jieun Lim

https://colab.research.google.com/drive/1Qni4z-2Xsc2X-S5bJG-ISosZNb5nE-XK#scrollTo=IrwV7GXM8JHG

Dataset

```
import pandas as pd
from google.colab import files
uploaded = files.upload()
```

Choose Files 2 files

- **bike_train.csv**(text/csv) - 648353 bytes, last modified: 3/22/2021 - 100% done
- **bike_test.csv**(text/csv) - 323856 bytes, last modified: 3/22/2021 - 100% done

```
Saving bike_train.csv to bike_train (1).csv
Saving bike_test.csv to bike_test (1).csv
```

This data is provided according to the Capital Bikeshare Data License Agreement

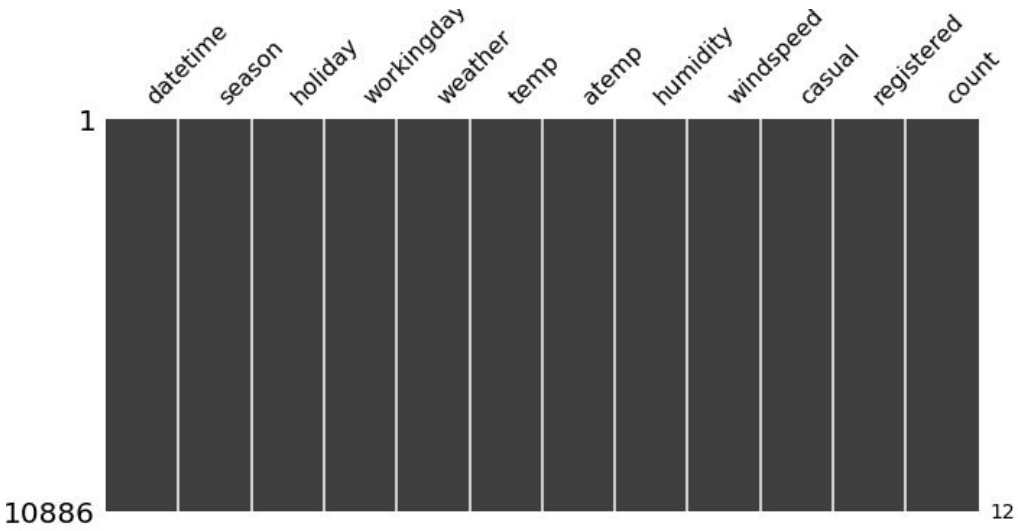
- datetime - hourly date + timestamp
- season - 1 = spring, 2 = summer, 3 = fall, 4 = winter
- holiday - whether the day is considered a holiday
- workingday - whether the day is neither a weekend nor holiday
- weather -
    - 1: Clear, Few clouds, Partly cloudy, Partly cloudy
    - 2: Mist + Cloudy, Mist + Broken clouds, Mist + Few clouds, Mist
    - 3: Light Snow, Light Rain + Thunderstorm + Scattered clouds, Light Rain + Scattered clouds
    - 4: Heavy Rain + Ice Pallets + Thunderstorm + Mist, Snow + Fog

Dataset

- temp - temperature in Celsius
- atemp - "feels like" temperature in Celsius
- humidity - relative humidity
- windspeed - wind speed
- casual - number of non-registered user rentals initiated
- registered - number of registered user rentals initiated
- count - number of total rentals (Dependent Variable)

	datetime	season	holiday	workingday	weather	temp	atemp	humidity	windspeed	casual	registered	count
0	2011-01-01 00:00:00	1	0	0	1	9.84	14.395	81	0.0	3	13	16
1	2011-01-01 01:00:00	1	0	0	1	9.02	13.635	80	0.0	8	32	40
2	2011-01-01 02:00:00	1	0	0	1	9.02	13.635	80	0.0	5	27	32

Dataset

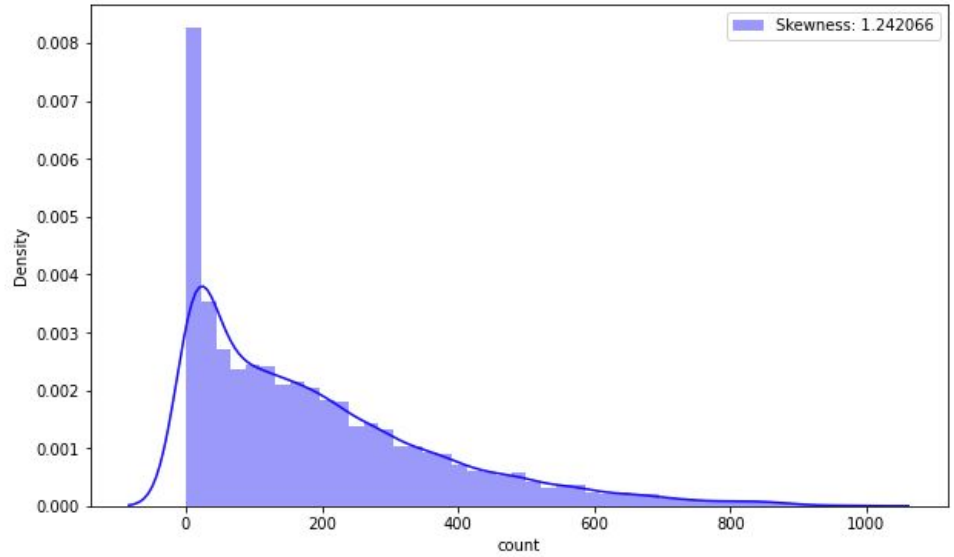



```
import missingno as msno
msno.matrix(train, figsize = (12, 5))
```

No Missing data

Dataset

```
skewness: 1.242066
kurtosis: 1.300093
```



How the data is distributed:
Skewness, Kurtosis

=> will use log scaling for normal distribution

EDA & Feature

```
#2011-01-01 00:00:00
```

```
#Train data
datetime = train['datetime']

train_year = []
train_month = []
train_day = []

train_hour = []
train_minute = []
train_second = []
```

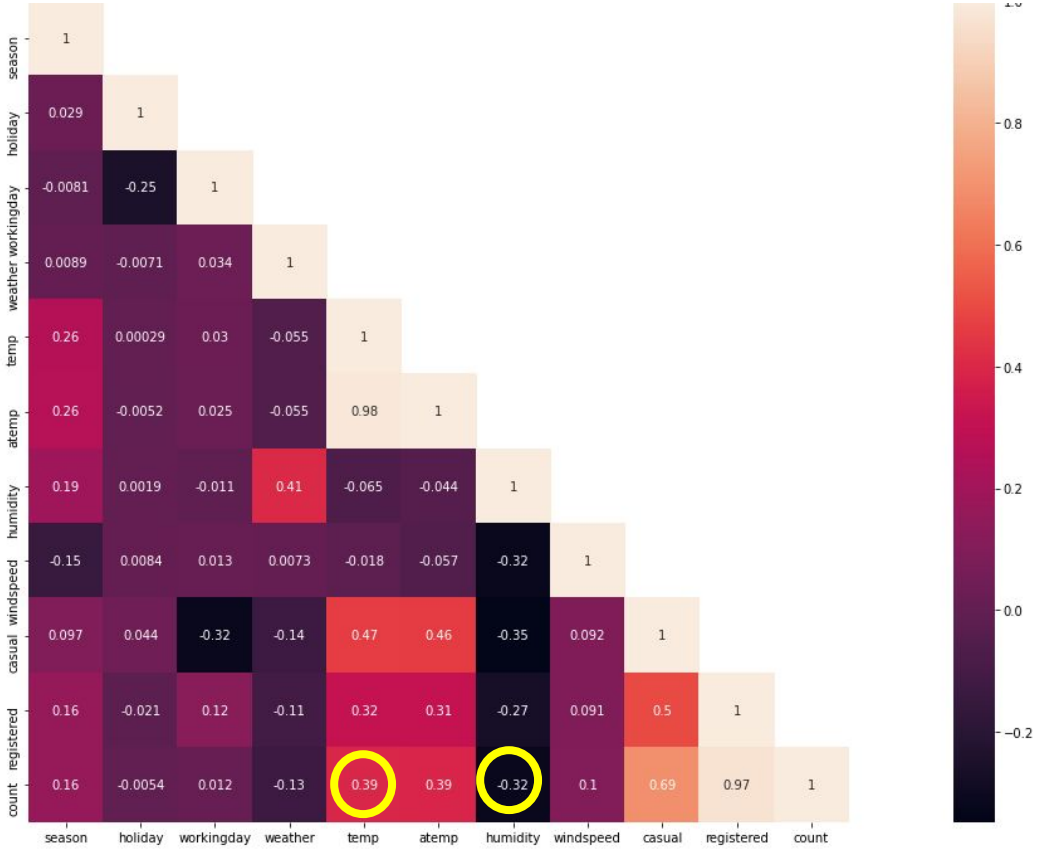
```
for i in datetime:
  data = i.split(" ")

  day = data[0]
  new_date = day.split("-")
  train_year.append(new_date[0])
  train_month.append(new_date[1])
  train_day.append(new_date[2])

  time = data[1]
  new_time = time.split(":")
  train_hour.append(new_time[0])
  train_minute.append(new_time[1])
  train_second.append(new_time[2])

train['year'] = train_year
train['month'] = train_month
train['day'] = train_day
train['hour'] = train_hour
train['minute'] = train_minute
train['second'] = train_second
```

year	month	day	hour	minute	second
2011	01	20	00	00	00
2011	01	20	01	00	00
2011	01	20	02	00	00
2011	01	20	03	00	00
2011	01	20	04	00	00



Correlation

neg : humidity, #bike sharing

pos: temp, #bike sharing

EDA & Feature engineering

```
#Dummy variable Season
season=pd.get_dummies(train['season'],prefix='season')
train=pd.concat([train,season],axis=1)
train.head()
season=pd.get_dummies(test['season'],prefix='season')
test=pd.concat([test,season],axis=1)
test.head()
```

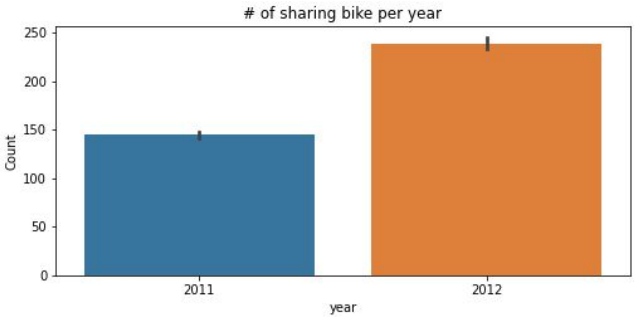
season_1	season_2	season_3	season_4
1	0	0	0
1	0	0	0
1	0	0	0
1	0	0	0
1	0	0	0

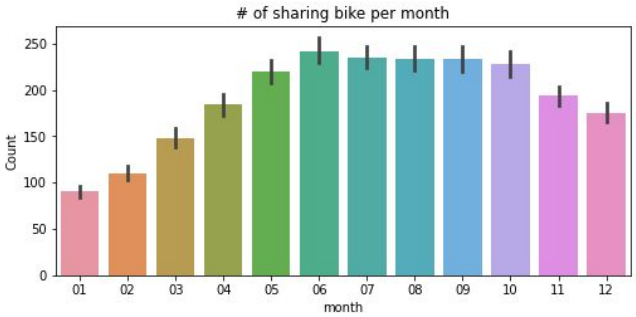
EDA & Feature engineering

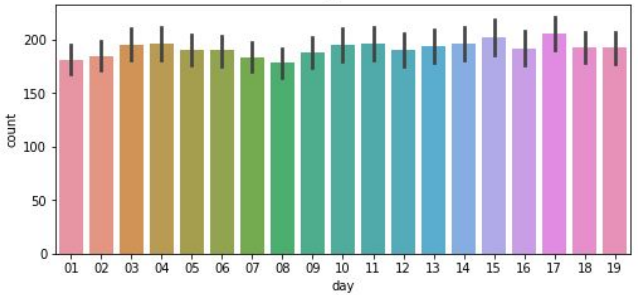
```
# Dummy variable weather
weather=pd.get_dummies(train['weather'],prefix='weather')
train=pd.concat([train,weather],axis=1)
train.head()
weather=pd.get_dummies(test['weather'],prefix='weather')
test=pd.concat([test,weather],axis=1)
test.head()
```

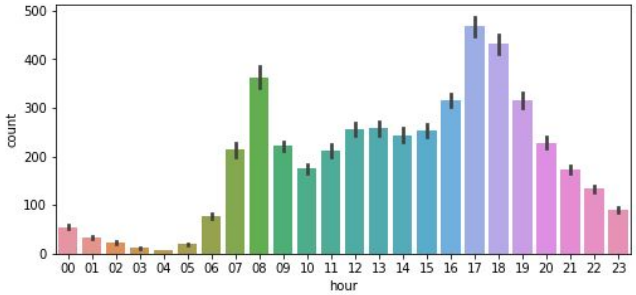
weather_1	weather_2	weather_3	weather_4
1	0	0	0
1	0	0	0
1	0	0	0
1	0	0	0
1	0	0	0

EDA









1) Day is only 1~19th is available on the Train data. Therefore we cannot use it for feature

2) Increase during rush hour (07AM - 08AM, 17 - 18PM). I'd like to compare it between Weekdays and weekands

3) The largest number of bicycle shares in June .

EDA

[16]
```
# In order to compare Weekdays and weekands
# get day-type(monday, tuesday) from datetime

import datetime
train_weekday = []
test_weekday = []

for i in train['datetime']:
    train_weekday.append(datetime.datetime.strptime(i, "%Y-%m-%d %H:%M:%S").strftime("%A"))

for i in test['datetime']:
    test_weekday.append(datetime.datetime.strptime(i, "%Y-%m-%d %H:%M:%S").strftime("%A"))

train['weekday'] = train_weekday
test['weekday'] = test_weekday
```

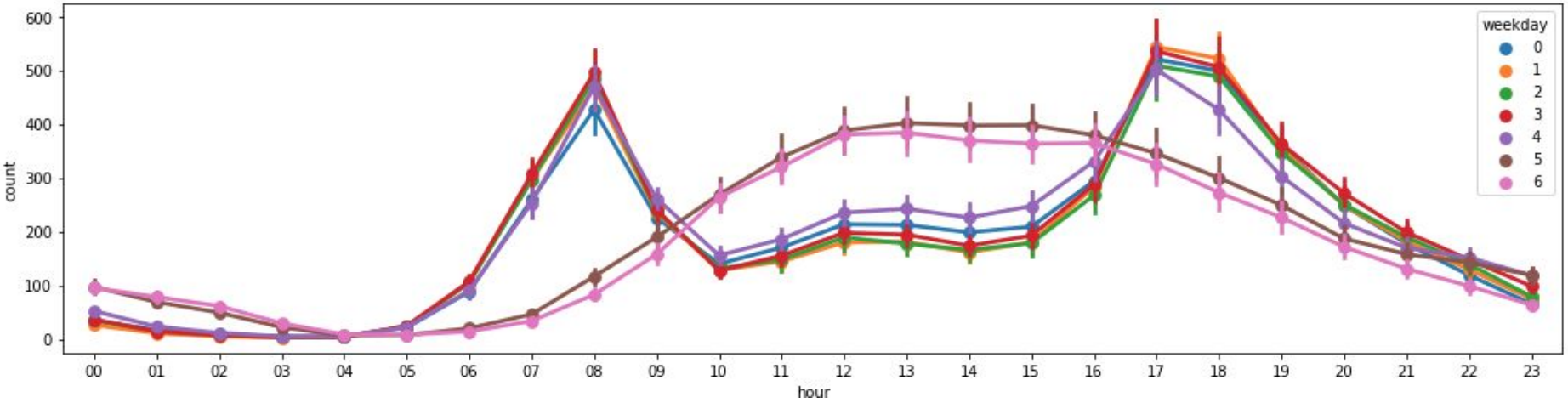
```
# convert weekday to 0-6

train['weekday'] = train['weekday'].map({'Monday': 0,
                                         'Tuesday': 1,
                                         'Wednesday': 2,
                                         'Thursday': 3,
                                         'Friday': 4,
                                         'Saturday': 5,
                                         'Sunday': 6})
test['weekday'] = test['weekday'].map({'Monday': 0,
                                       'Tuesday': 1,
                                       'Wednesday': 2,
                                       'Thursday': 3,
                                       'Friday': 4,
                                       'Saturday': 5,
                                       'Sunday': 6})
```

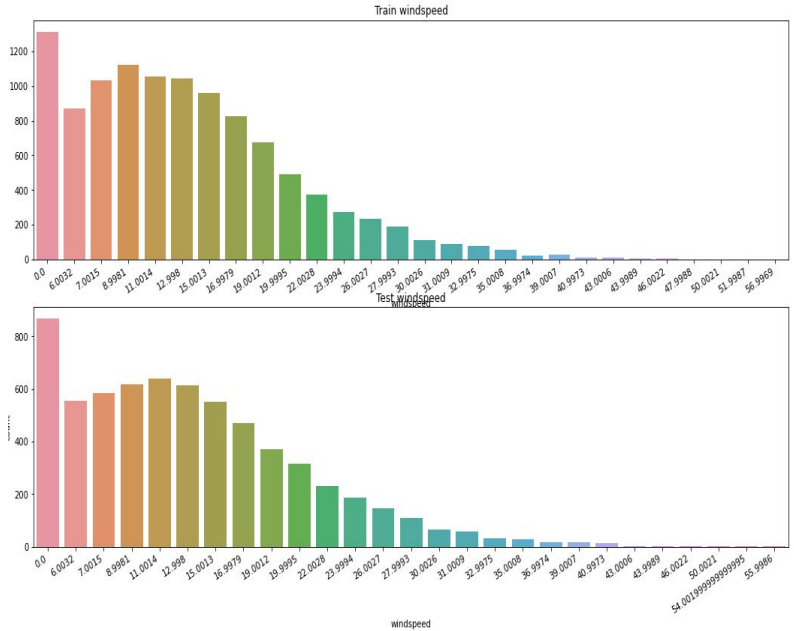
EDA

```
fig, (ax1, ax2, ax3, ax4, ax5) = plt.subplots(nrows = 5)
fig.set_size_inches(18, 25)

sns.pointplot(data = train, x = 'hour', y = 'count', ax=ax1)
sns.pointplot(data = train, x = 'hour', y = 'count', hue = 'workingday', ax=ax2)
sns.pointplot(data = train, x = 'hour', y = 'count', hue = 'weekday', ax=ax3)
sns.pointplot(data = train, x = 'hour', y = 'count', hue = 'weather', ax=ax4)
sns.pointplot(data = train, x = 'hour', y = 'count', hue = 'season', ax=ax5)
```



EDA & Feature engineering



There are too many "0" on the wind speed. (95%) So we will not use it as an feature

```
[18] #train['windspeed']==0.count()
     1-(np.count_nonzero(train['windspeed'])/train.size)

     0.9537165069572701
```

Modeling Work

- Y = Count
- X = Season, weather, "temp", "atemp", "humidity", "year", "hour", "weekday", "holiday", "workingday"
- Use Train data, Test data
- Use RMSLE SCORE
- Compare models:
    - Linear Regression
    - RandomForest
    - GradientBoost

Evaluate Model

Root Mean Squared Logarithmic Error

- Slight modification on MSE
- Penalties for under-predicted estimate rather than over-predicted estimate

The smaller the value of RMSLE,

The better precision

=> score : closer to 0 ?

https://www.kaggle.com/carlolepelaars/understanding-the-metric-rmsle

RMSLE SCORE

```
[23] from sklearn.metrics import make_scorer

     # will use rmsle score
     def rmsle(predicted_values, actual_values, convertExp=True):

         if convertExp:
             predicted_values = np.exp(predicted_values),
             actual_values = np.exp(actual_values)

         # change it to array
         predicted_values = np.array(predicted_values)
         actual_values = np.array(actual_values)

         # add 1 since there are lots of 0
         # apply log to make it normal distribution

         log_predict = np.log(predicted_values + 1)
         log_actual = np.log(actual_values + 1)

         # (yhat - y)^2
         difference = log_predict - log_actual
         difference = np.square(difference)

         # mean of (yhat - y)^2
         mean_difference = difference.mean()

         # sqrt
         score = np.sqrt(mean_difference)

         return score
```

Compare Models

Linear Regression Model

```
from sklearn.linear_model import LinearRegression, Ridge, Lasso
from sklearn.model_selection import GridSearchCV
from sklearn import metrics
import numpy as np
import warnings
pd.options.mode.chained_assignment = None
warnings.filterwarnings("ignore", category=DeprecationWarning)

# initialize
lModel = LinearRegression()

# train
y_train_log = np.log1p(y_train)
lModel.fit(X_train, y_train_log)

# predict and print score
preds = lModel.predict(X_train)
print ("RMSLE Value For Linear Regression: ",
       rmsle(np.exp(y_train_log),np.exp(preds), False))
```

```
RMSLE Value For Linear Regression:  0.9705379002315476
```

Compare Models

Ensemble

- Averaging out biases
- Reduce variance
- Avoid overfitting

Ensemble Models - Random Forest

```
[56] from sklearn.ensemble import RandomForestRegressor
     rfModel = RandomForestRegressor(n_estimators=100)

     y_train_log = np.log1p(y_train)
     rfModel.fit(X_train, y_train_log)

     preds = rfModel.predict(X_train)
     score = rmsle(np.exp(y_train_log),np.exp(preds),False)
     print ("RMSLE Value For Random Forest: ",score)

     RMSLE Value For Random Forest:  0.10744300712309492
```

Compare Models

- Gradient Boost or XG Boost Perform better based on Bagging

Ensemble Model - Gradient Boost

```
[36] from sklearn.ensemble import GradientBoostingRegressor
     gbm = GradientBoostingRegressor(n_estimators=4000, alpha=0.01);

     y_train_log = np.log1p(y_train)
     gbm.fit(X_train, y_train_log)

     preds = gbm.predict(X_train)
     score = rmsle(np.exp(y_train_log),np.exp(preds),False)
     print ("RMSLE Value For Gradient Boost: ", score)

     RMSLE Value For Gradient Boost:  0.21357403727249372
```

Questions?