Problem Statement

In this project, I have combined historical usage patterns with weather data in order to forecast hourly bike rental demand.

```
In [13]:
```

```
#Loading the libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sb
import seaborn as sn
from datetime import datetime
from datetime import date
import calendar
```

```
In [2]:
```

```
#Loading the dataset
train = pd.read_csv("C:\\Users\\Mr Sinha\\Desktop\\internshala\\assigment 4\\train.csv")
test = pd.read_csv("C:\\Users\\Mr Sinha\\Desktop\\internshala\\assigment 4\\test.csv")
```

Let's have a look on the shape of our dataset that is number of rows and columns present in the dataset given.

```
In [3]:
```

```
train.shape , test.shape
```

```
Out[3]:

((12980, 12), (4399, 11))
```

There are 12 columns in train dataset, whereas 11 in the test dataset. The missing column in the test dataset is the target variable and we will train our model to predict that variable.

```
In [4]:
```

```
# printing first five rows
train.head()
```

```
Out[4]:
```

	datetime	season	holiday	workingday	weather	temp	atemp	humidity	windspeed	casual	registered	count
0	2011-01-01 0:00:00	1	0	0	1	9.84	14.395	81.0	0.0	3	13	16
1	2011-01-01 1:00:00	1	0	0	1	9.02	13.635	80.0	0.0	8	32	40
2	2011-01-01 2:00:00	1	0	0	1	9.02	13.635	80.0	0.0	5	27	32
3	2011-01-01 3:00:00	1	0	0	1	9.84	14.395	75.0	0.0	3	10	13
4	2011-01-01 4:00:00	1	0	0	1	9.84	14.395	75.0	0.0	0	1	1

```
In [5]:
```

```
test.head()
```

```
Out[5]:
```

	datetime	season	holiday	workingday	weather	temp	atemp	humidity	windspeed	casual	registered
0	2012-06-30 1:00:00	3	0	0	3	26.24	28.790	89.0	15.0013	3	55
1	2012-06-30 2:00:00	3	0	0	2	26.24	28.790	89.0	0.0000	7	54

	datetime	season	holiday	workingday	weather	temp	atemp	humidity	windspeed	casual	registered
2	2012-06-30 3:00:00	3	0	0	2	26.24	28.790	89.0	0.0000	3	20
3	2012-06-30 4:00:00	3	0	0	2	25.42	27.275	94.0	0.0000	3	15
4	2012-06-30 5:00:00	3	0	0	1	26.24	28.790	89.0	11.0014	3	7

In [6]:

```
# columns in the dataset
train.columns
```

Out[6]:

```
Index(['datetime', 'season', 'holiday', 'workingday', 'weather', 'temp',
       'atemp', 'humidity', 'windspeed', 'casual', 'registered', 'count'],
      dtype='object')
```

In [7]:

```
test.columns
```

Out[7]:

```
Index(['datetime', 'season', 'holiday', 'workingday', 'weather', 'temp',
       'atemp', 'humidity', 'windspeed', 'casual', 'registered'],
      dtype='object')
```

We can infer that "count" is our target variable as it is missing from the test dataset.

In [8]:

```
# Data type of the columns
train.dtypes
```

Out[8]:

```
datetime       object
season          int64
holiday         int64
workingday      int64
weather         int64
temp          float64
atemp         float64
humidity      float64
windspeed     float64
casual          int64
registered      int64
count           int64
dtype: object
```

We can infer that all of the variable in the dataset except datetime are numerical variables. Now Let's look at the distribution of our target variable, i.e. count. As it is a numerical variable, let us look at its distribution.

Univariate Analysis

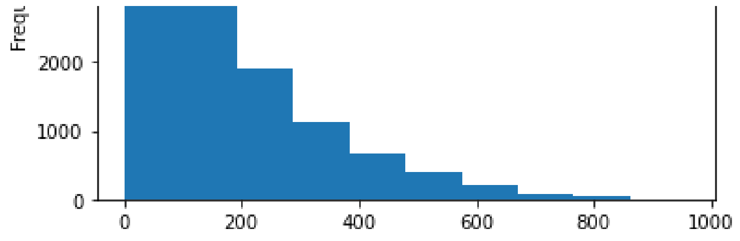
In [9]:

```
train['count'].plot.hist()
```

Out[9]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1ea9b95cd60>
```

5000
4000
3000
uency



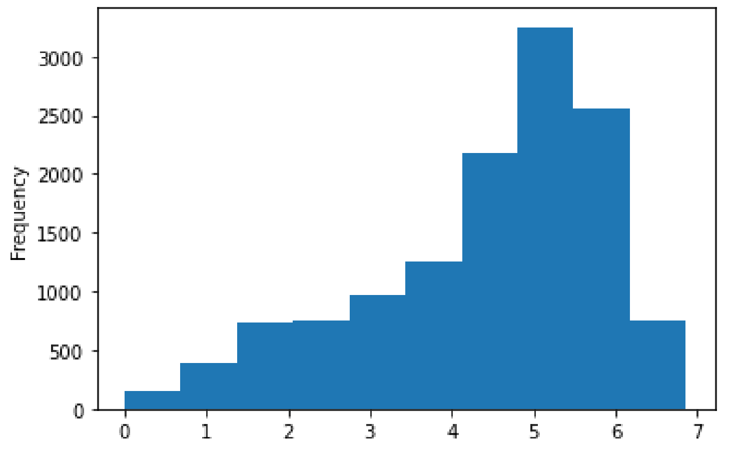
The distribution is skewed towards right and hence we can take log of the variable and see if the distribution becomes normal.

In [10]:

```
np.log(train['count']).plot.hist()
```

Out[10]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1ea9c1573d0>
```



Now the distribution looks less skewed. Let's now explore the variables to have a better understanding of the dataset. We will first explore the variables individually using univariate analysis, then we will look at the relation between various independent variables and the target variable. We will also look at the correlation plot to see which variables affects the target variable most.

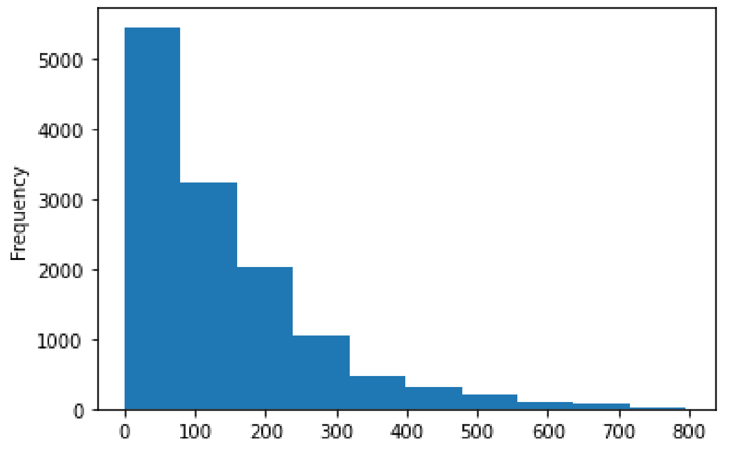
Let's first look at the distribution of registered variable to check the number of registered user rentals initiated.

In [11]:

```
train["registered"].plot.hist()
```

Out[11]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1ea9c1e3850>
```



We can see that most of the registered rentals lies in the range of 0 to 200. The registered users at a particular time step will always be less than or equal to the demand (count) of that particular timestep.

Let's now look at how correlated our numerical variables are.

We will see the correlation between each of these variables and the variable which have high negative or positive values are correlated. By this we can get an overview of the variables which might affect our target variable.

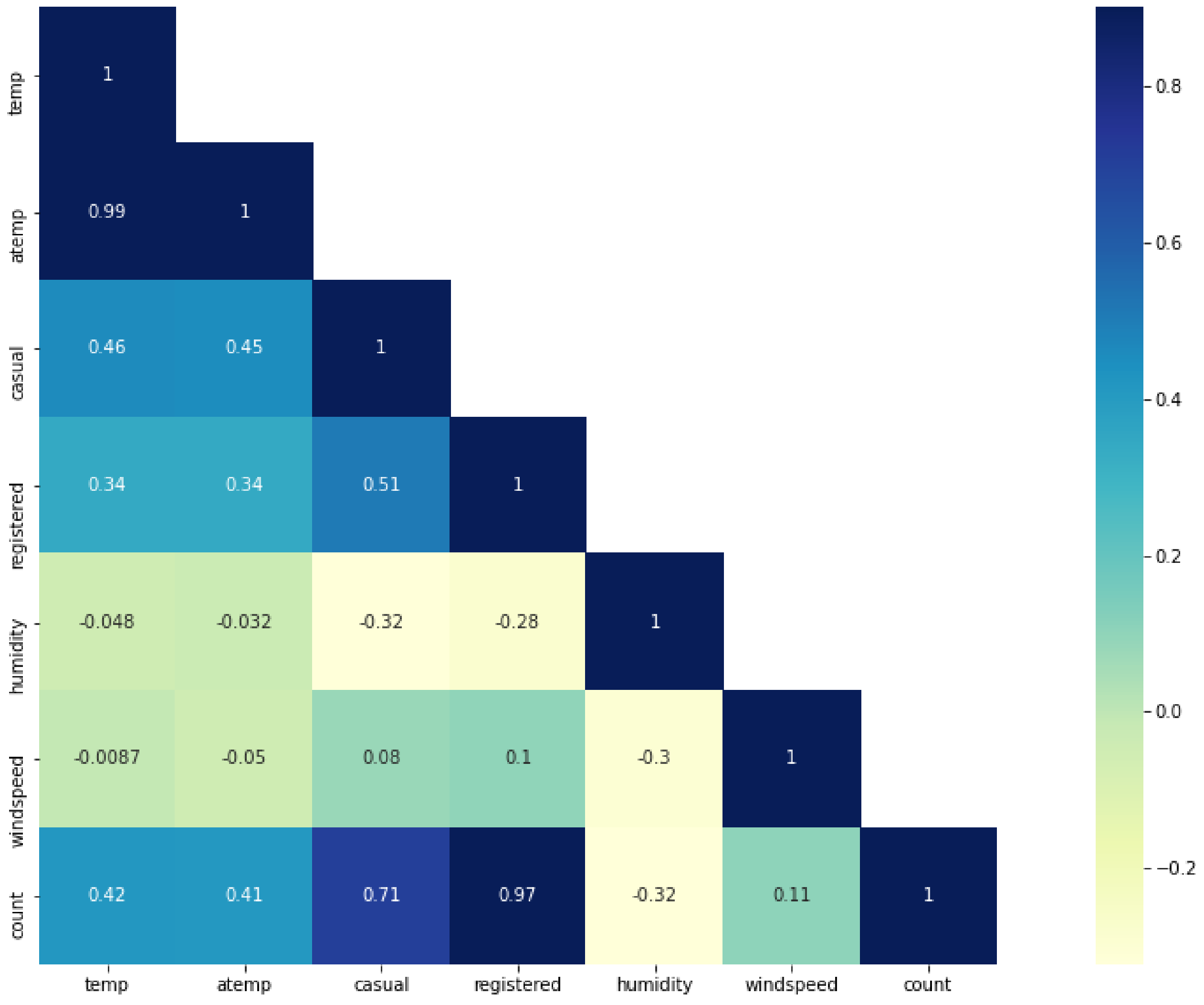
Bivariate Analysis

In [14]:

```
# looking at the correlation between numerical variables
corr = train[["temp","atemp","casual","registered","humidity","windspeed","count"]].corr
()
mask = np.array(corr)
mask[np.tril_indices_from(mask)] = False
fig,ax= plt.subplots()
fig.set_size_inches(20,10)
sn.heatmap(corr, mask=mask,vmax=.9, square=True,annot=True, cmap="YlGnBu")
```

Out[14]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1ea9c388a00>
```



Some of the inferences from the above correlation map are:

1. **temp and humidity features has got positive and negative correlation with count respectively.Although the correlation between them are not very prominent still the count variable has got little dependency on "temp" and "humidity".**
2. **windspeed will not be really useful numerical feature and it is visible from it correlation value with "count"**
3. **Since "atemp" and "temp" has got strong correlation with each other, during model building any one of the variable has to be dropped since they will exhibit multicollinearity in the data.**

Before building the model, let's check if there are any missing values in the dataset.

In [15]:

```
# looking for missing values in the datasaet
train.isnull().sum()
```

Out[15]:

```
datetime      0
season        0
holiday       0
workingday    0
weather       0
temp          0
atemp         0
humidity      0
windspeed     0
casual        0
registered    0
count         0
dtype: int64
```

There are no missing values in the train dataset. Let's look for the missing values in the test dataset.

In [16]:

```
test.isnull().sum()
```

Out[16]:

```
datetime      0
season        0
holiday       0
workingday    0
weather       0
temp          0
atemp         0
humidity      0
windspeed     0
casual        0
registered    0
dtype: int64
```

There are no missing values in the test dataset as well. We can now move further and build our first model. Before that let's first extract some new features using the datetime variable. We can extract the date, hour, month.

In [17]:

```
# extracting date, hour and month from the datetime
train["date"] = train.datetime.apply(lambda x : x.split()[0])
train["hour"] = train.datetime.apply(lambda x : x.split()[1].split(":")[0])
train["month"] = train.date.apply(lambda dateString : datetime.strptime(dateString,"%Y-%
m-%d").month)
```

Let's now build a linear regression model to get the predictions on the test data. We have to make the similar changes in test data as we have done for the training data.

In [18]:

```
test["date"] = test.datetime.apply(lambda x : x.split()[0])
test["hour"] = test.datetime.apply(lambda x : x.split()[1].split(":")[0])
test["month"] = test.date.apply(lambda dateString : datetime.strptime(dateString,"%Y-%m-%
d").month)
```

Now our data is ready. Before making the model, we will create a validation set to validate our model. So, we will divide the train set into training and validation set. We will train the model on the training set and check its performance on the validation set. Since the data is time based, we will split it as per time. Let's take first 15 months for training and remaining 3 months in the validation set.

In [19]:

```
training = train[train['datetime']<='2012-03-30 0:00:00']
validation = train[train['datetime']>'2012-03-30 0:00:00']
```

- **We will drop the datetime, date variable as we have already extracted features from these variables.**
- **We will also drop the atemp variable as we saw that it is highly correlated with the temp variable.**

In [20]:

```
train = train.drop(['datetime','date', 'atemp'],axis=1)
test = test.drop(['datetime','date', 'atemp'], axis=1)
training = training.drop(['datetime','date', 'atemp'],axis=1)
validation = validation.drop(['datetime','date', 'atemp'],axis=1)
```

Model Building

Linear Regression Model

In [21]:

```
from sklearn.linear_model import LinearRegression
```

In [22]:

```
# initialize the linear regression model
lModel = LinearRegression()
```

We will remove the target variable from both the training and validation set and keep it in a separate variable. We saw in the visualization part that the target variable is right skewed, so we will take its log as well before feeding it to the model.

In [23]:

```
X_train = training.drop('count', 1)
y_train = np.log(training['count'])
X_val = validation.drop('count', 1)
y_val = np.log(validation['count'])
```

In [24]:

```
# checking the shape of X_train, y_train, X_val and y_val
X_train.shape, y_train.shape, X_val.shape, y_val.shape
```

Out[24]:

```
((10774, 11), (10774,), (2206, 11), (2206,))
```

In [25]:

```
# fitting the model on X_train and y_train
lModel.fit(X_train,y_train)
```

Out[25]:

```
LinearRegression()
```

Now we have a trained linear regression model with us. We will now make prediction on the X_val set and check the performance of our model. Since the evaluation metric for this problem is RMSLE, we will define a model which will return the RMSLE score.

In [26]:

```
# making prediction on validation set
```

```
prediction = lModel.predict(X_val)
```

In [27]:

```
# defining a function which will return the rmsle score
def rmsle(y, y_):
    y = np.exp(y),    # taking the exponential as we took the log of target variable
    y_ = np.exp(y_)
    log1 = np.nan_to_num(np.array([np.log(v + 1) for v in y]))
    log2 = np.nan_to_num(np.array([np.log(v + 1) for v in y_]))
    calc = (log1 - log2) ** 2
    return np.sqrt(np.mean(calc))
```

Let's now calculate the rmsle value of the predictions

In [28]:

```
rmsle(y_val,prediction)
```

Out[28]:

```
0.8875379204281797
```

In [29]:

```
test_prediction = lModel.predict(test)
```

We got a rmsle value of 0.8875 on the validation set.

Let's use Decision Tree now. Note that rmsle tells us how far the predictions are from the actual value, so we want rmsle value to be as close to 0 as possible. So, we will further try to reduce this value.

Decision Tree

In [30]:

```
from sklearn.tree import DecisionTreeRegressor
```

In [31]:

```
# defining a decision tree model with a depth of 5. You can further tune the hyperparamet
ers to improve the score
dt_reg = DecisionTreeRegressor(max_depth=5)
```

Let's fit the decision tree model now.

In [32]:

```
dt_reg.fit(X_train, y_train)
```

Out[32]:

```
DecisionTreeRegressor(max_depth=5)
```

Its time to make prediction on the validation set using the trained decision tree model.

In [33]:

```
predict = dt_reg.predict(X_val)
```

In [34]:

```
# calculating rmsle of the predicted values
rmsle(y_val, predict)
```

Out[34]:

```
0.17102267300446063
```

```
0.17102267399446963
```

The rmsle value has decreased to 0.171. This is a decent score. Let's now make predictions for the test dataset which you can submit in the excel sheet provided to you to generate your score.

In [36]:

```
test_prediction = dt_reg.predict(test)
```

These are the log values and we have to convert them back to the original scale.

In [37]:

```
final_prediction = np.exp(test_prediction)
```

Let's submit the predictions

In [38]:

```
submission = pd.DataFrame()
```

In [39]:

```
# creating a count column and saving the predictions in it
submission['count'] = final_prediction
```

In [40]:

```
submission.to_csv('submission.csv', header=True, index=False)
```

In [ ]:

```
```