Importing Needed Packages

In [ ]:
```
Vekat Preetham G
```

In [ ]:
```
import warnings
warnings.filterwarnings('ignore')
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

import sklearn
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from sklearn.linear_model import LinearRegression
from sklearn import preprocessing

%matplotlib inline
```

Reading CSV file as weather_df and making date_time column as index of dataframe

In [ ]:
```
weather_df = pd.read_csv('kanpur.csv', parse_dates=['date_time'], index_col='date_time')
weather_df.head(5)
```

Out[ ]:

date_time	maxtempC	mintempC	totalSnow_cm	sunHour	uvIndex	uvIndex.1	moon_illumination	moonrise	moonset	sunrise	su
2009-01-01 00:00:00	24	10	0.0	8.7	4	1	31	09:56 AM	09:45 PM	06:57 AM	
2009-01-01 01:00:00	24	10	0.0	8.7	4	1	31	09:56 AM	09:45 PM	06:57 AM	
2009-01-01 02:00:00	24	10	0.0	8.7	4	1	31	09:56 AM	09:45 PM	06:57 AM	
2009-01-01 03:00:00	24	10	0.0	8.7	4	1	31	09:56 AM	09:45 PM	06:57 AM	
2009-01-01 04:00:00	24	10	0.0	8.7	4	1	31	09:56 AM	09:45 PM	06:57 AM	

Checking columns in our dataframe

In [ ]:
```
weather_df.columns
```

Out[ ]:
```
Index(['maxtempC', 'mintempC', 'totalSnow_cm', 'sunHour', 'uvIndex',
       'uvIndex.1', 'moon_illumination', 'moonrise', 'moonset', 'sunrise',
       'sunset', 'DewPointC', 'FeelsLikeC', 'HeatIndexC', 'WindChillC',
       'WindGustKmph', 'cloudcover', 'humidity', 'precipMM', 'pressure',
       'tempC', 'visibility', 'winddirDegree', 'windspeedKmph'],
      dtype='object')
```

Now shape

In [ ]:
```
weather_df.shape
```

Out[ ]:
```
(96432, 24)
```

In [ ]:
```
weather_df.describe()
```

Out[ ]:

	maxtempC	mintempC	totalSnow_cm	sunHour	uvIndex	uvIndex.1	moon_illumination	DewPointC	96
count	96432.000000	96432.000000	96432.0	96432.000000	96432.000000	96432.000000	96432.000000	96432.000000	
mean	33.400199	22.374564	0.0	11.037805	6.877053	4.465012	46.094077	13.230629	
std	6.994211	7.635253	0.0	2.152973	1.551294	3.414374	31.249725	8.053778	
min	15.000000	3.000000	0.0	4.000000	3.000000	1.000000	0.000000	-14.000000	
25%	28.000000	16.000000	0.0	8.700000	6.000000	1.000000	18.000000	7.000000	
50%	34.000000	24.000000	0.0	11.600000	7.000000	5.000000	46.000000	12.000000	
75%	38.000000	28.000000	0.0	13.000000	8.000000	8.000000	73.000000	21.000000	
max	51.000000	39.000000	0.0	13.900000	11.000000	11.000000	100.000000	31.000000	

Checking is there any null values in dataset

In [ ]:
```
weather_df.isnull().any()
```

Out[ ]:
```
maxtempC             False
mintempC             False
totalSnow_cm         False
sunHour              False
uvIndex              False
uvIndex.1            False
moon_illumination    False
moonrise             False
moonset              False
sunrise              False
sunset               False
DewPointC            False
FeelsLikeC           False
HeatIndexC           False
WindChillC           False
WindGustKmph         False
cloudcover           False
humidity             False
precipMM             False
pressure             False
tempC                False
visibility           False
winddirDegree        False
windspeedKmph        False
dtype: bool
```

Now lets separate the feature (i.e. temperature) to be predicted from the rest of the featured. weather_x stores the rest of the dataset while weather_y has temperature column.

In [ ]:
```
weather_df_num=weather_df.loc[:,['maxtempC','mintempC','cloudcover','humidity','tempC', 'sunHour','HeatIndexC',
weather_df_num.head()
```

Out[ ]:

date_time	maxtempC	mintempC	cloudcover	humidity	tempC	sunHour	HeatIndexC	precipMM	pressure	windspeedKmph
2009-01-01 00:00:00	24	10	17	50	11	8.7	12	0.0	1015	10
2009-01-01 01:00:00	24	10	11	52	11	8.7	13	0.0	1015	11
2009-01-01 02:00:00	24	10	6	55	11	8.7	13	0.0	1015	11
2009-01-01 03:00:00	24	10	0	57	10	8.7	13	0.0	1015	12
2009-01-01 04:00:00	24	10	0	54	11	8.7	14	0.0	1016	11

Shape of new dataframe

In [ ]:
```
weather_df_num.shape
```

Out[ ]:
```
(96432, 10)
```

Columns in new dataframe

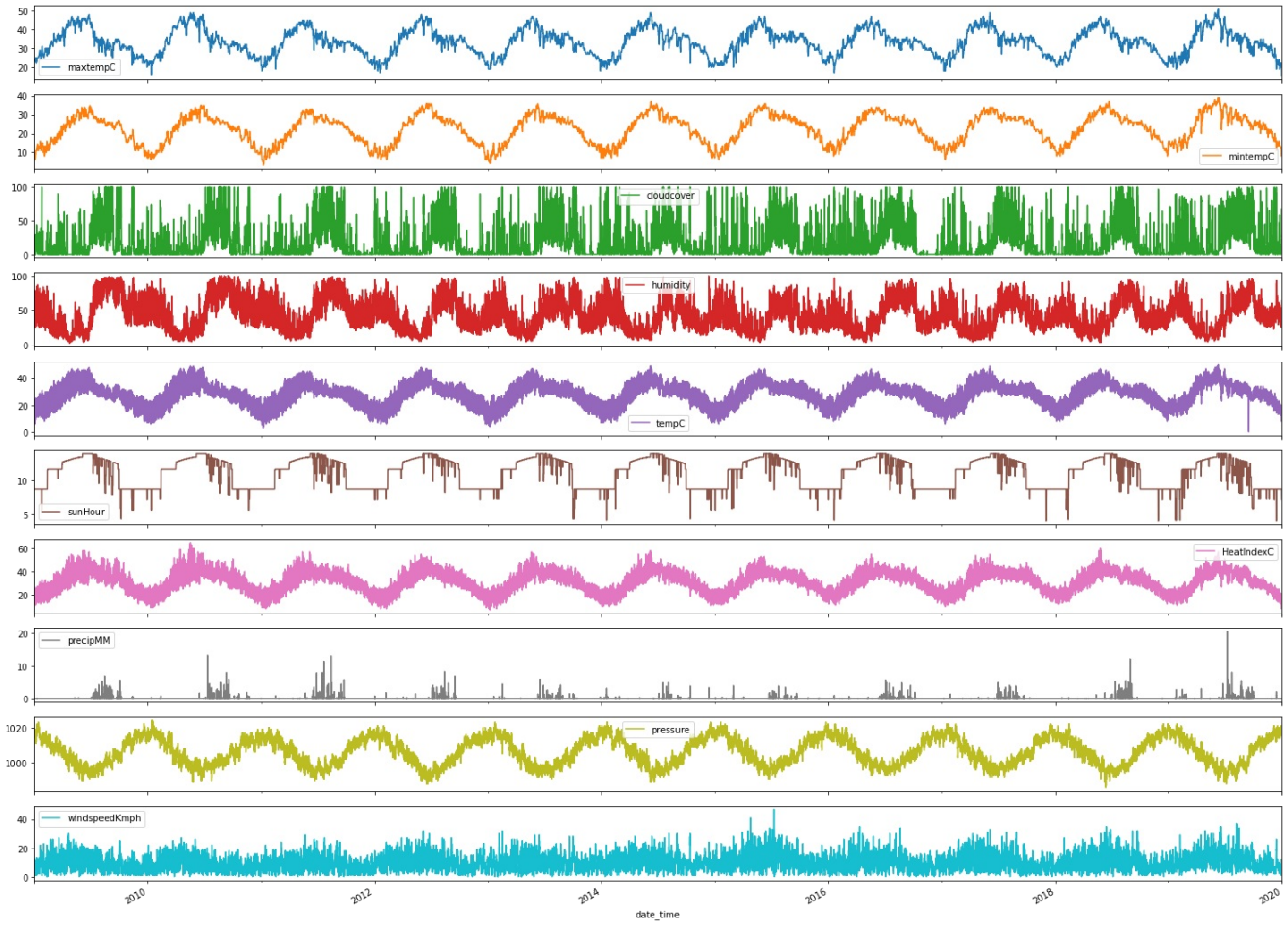
In [ ]:
```
weather_df_num.columns
```

Out[ ]:
```
Index(['maxtempC', 'mintempC', 'cloudcover', 'humidity', 'tempC', 'sunHour',
       'HeatIndexC', 'precipMM', 'pressure', 'windspeedKmph'],
      dtype='object')
```

Ploting all the column values

In [ ]:
```
weather_df_num.plot(subplots=True, figsize=(25,20))
```

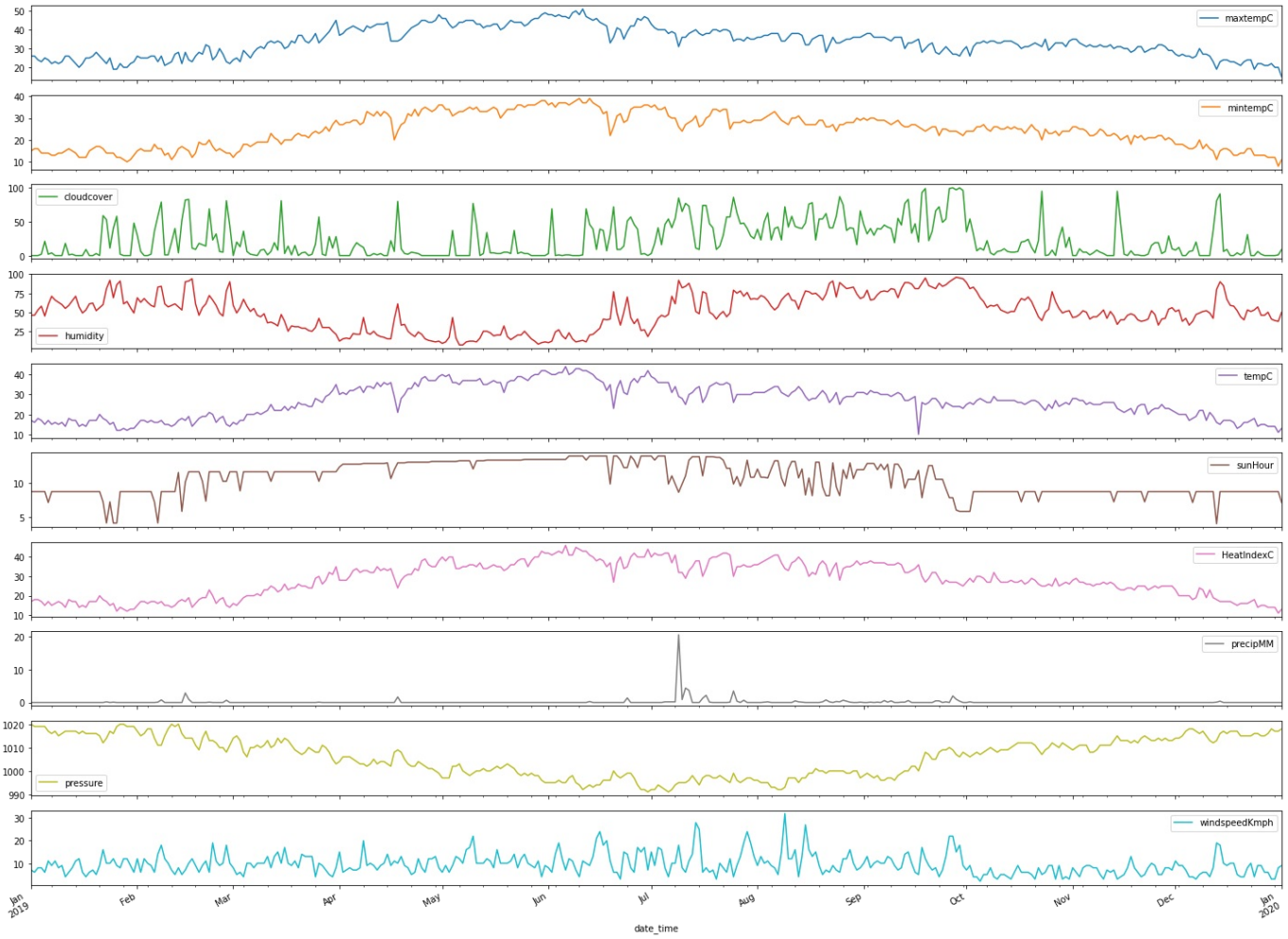
Out[ ]:
```
array([<matplotlib.axes._subplots.AxesSubplot object at 0x7f7a8e2f1f10>,
       <matplotlib.axes._subplots.AxesSubplot object at 0x7f7a8e02e6d0>,
       <matplotlib.axes._subplots.AxesSubplot object at 0x7f7a8e2ee3d0>,
       <matplotlib.axes._subplots.AxesSubplot object at 0x7f7a8dd8ded0>,
       <matplotlib.axes._subplots.AxesSubplot object at 0x7f7a8db37310>,
       <matplotlib.axes._subplots.AxesSubplot object at 0x7f7a8e2c3710>,
       <matplotlib.axes._subplots.AxesSubplot object at 0x7f7a8e252b90>,
       <matplotlib.axes._subplots.AxesSubplot object at 0x7f7a8e288ed0>,
       <matplotlib.axes._subplots.AxesSubplot object at 0x7f7a8e288f10>,
       <matplotlib.axes._subplots.AxesSubplot object at 0x7f7a8e1cc450>],
      dtype=object)
```



Ploting all the column values for 1 year

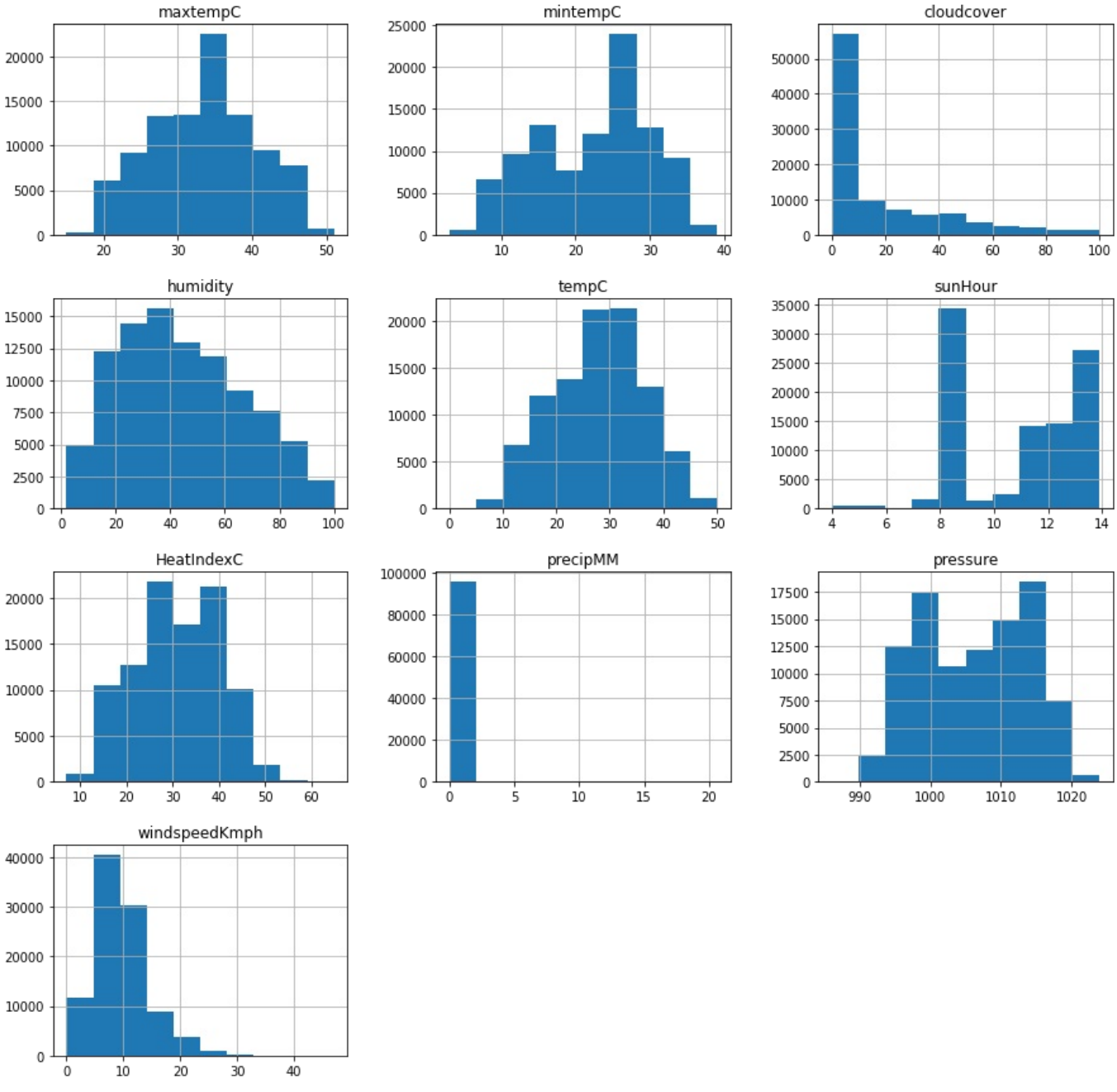
In [ ]:
```
weather_df_num['2019':'2020'].resample('D').fillna(method='pad').plot(subplots=True, figsize=(25,20))
```

Out[ ]:
```
array([<matplotlib.axes._subplots.AxesSubplot object at 0x7f7a8e48b890>,
       <matplotlib.axes._subplots.AxesSubplot object at 0x7f7a86824ed0>,
       <matplotlib.axes._subplots.AxesSubplot object at 0x7f7a8dadfe10>,
       <matplotlib.axes._subplots.AxesSubplot object at 0x7f7a86794650>,
       <matplotlib.axes._subplots.AxesSubplot object at 0x7f7a8674aa10>,
       <matplotlib.axes._subplots.AxesSubplot object at 0x7f7a86780dd0>,
       <matplotlib.axes._subplots.AxesSubplot object at 0x7f7a86745250>,
       <matplotlib.axes._subplots.AxesSubplot object at 0x7f7a866fd550>,
       <matplotlib.axes._subplots.AxesSubplot object at 0x7f7a866fd590>,
       <matplotlib.axes._subplots.AxesSubplot object at 0x7f7a866b5a50>],
      dtype=object)
```



In [ ]:
```
weather_df_num.hist(bins=10,figsize=(15,15))
```

Out[ ]:
```
array([[<matplotlib.axes._subplots.AxesSubplot object at 0x7f7a85e2b850>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f7a85dddb50>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f7a85da4610>],
       [<matplotlib.axes._subplots.AxesSubplot object at 0x7f7a85d56b90>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f7a85d18150>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f7a85ccf6d0>],
       [<matplotlib.axes._subplots.AxesSubplot object at 0x7f7a85d04cd0>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f7a85cc81d0>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f7a85cc8210>],
       [<matplotlib.axes._subplots.AxesSubplot object at 0x7f7a85c7c890>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f7a85bf42d0>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f7a85ba7850>]],
      dtype=object)
```



In [ ]:
```
weth=weather_df_num['2019':'2020']
weth.head()
```

Out[ ]:

date_time	maxtempC	mintempC	cloudcover	humidity	tempC	sunHour	HeatIndexC	precipMM	pressure	windspeedKmph
2019-01-01 00:00:00	26	15	0	46	17	8.7	17	0.0	1020	7
2019-01-01 01:00:00	26	15	0	46	17	8.7	17	0.0	1019	7
2019-01-01 02:00:00	26	15	0	47	16	8.7	16	0.0	1019	7
2019-01-01 03:00:00	26	15	0	48	16	8.7	16	0.0	1019	6
2019-01-01 04:00:00	26	15	0	48	16	8.7	16	0.0	1019	6

In [ ]:
```
weather_y=weather_df_num.pop("tempC")
weather_x=weather_df_num
```

Now our dataset is prepared and it is ready to be fed to the model for training.it's time to split the dataset into training and testing.

In [ ]:
```
train_X,test_X,train_y,test_y=train_test_split(weather_x,weather_y,test_size=0.2,random_state=4)
```

In [ ]:
```
train_X.shape
```

Out[ ]:
```
(77145, 9)
```

In [ ]:
```
train_y.shape
```

Out[ ]:
```
(77145,)
```

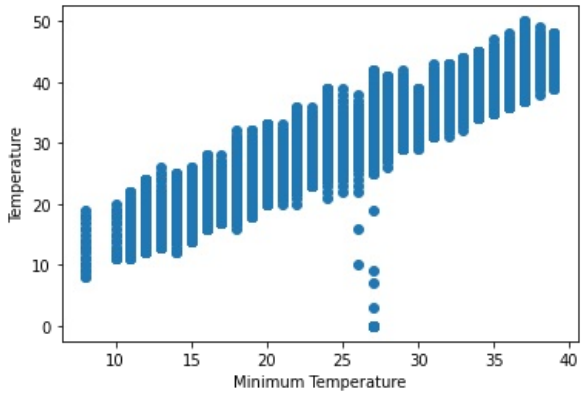
train_x has all the features except temperature and train_y has the corresponding temperature for those features. in supervised machine learning we first feed the model with input and associated output and then we check with a new input.

In [ ]:
```
train_y.head()
```

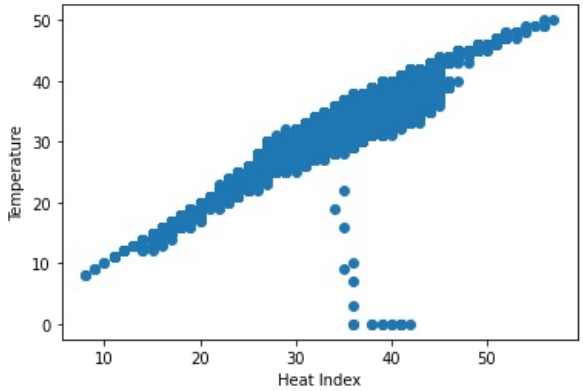
Out[ ]:
```
date_time
2012-03-13 07:00:00    22
2009-11-05 21:00:00    21
2017-10-11 22:00:00    30
2019-06-08 11:00:00    47
2019-03-06 05:00:00    18
Name: tempC, dtype: int64
```

Multiple Linear Regression

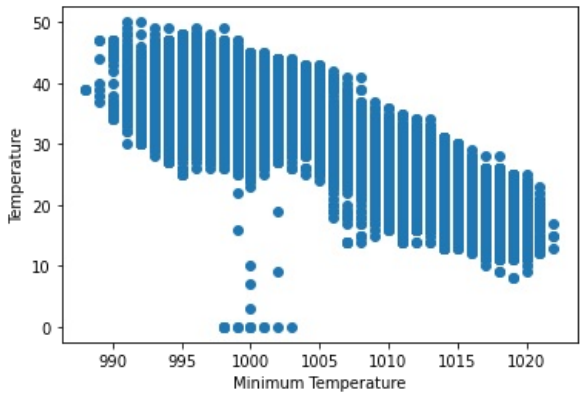
In [ ]:
```
plt.scatter(weth.mintempC, weth.tempC)
plt.xlabel("Minimum Temperature")
plt.ylabel("Temperature")
plt.show()
```



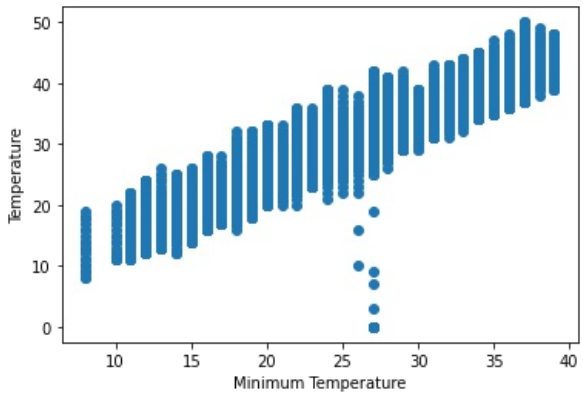
In [ ]:
```
plt.scatter(weth.HeatIndexC, weth.tempC)
plt.xlabel("Heat Index")
plt.ylabel("Temperature")
plt.show()
```



In [ ]:
```
plt.scatter(weth.pressure, weth.tempC)
plt.xlabel("Minimum Temperature")
plt.ylabel("Temperature")
plt.show()
```



In [ ]:
```
plt.scatter(weth.mintempC, weth.tempC)
plt.xlabel("Minimum Temperature")
plt.ylabel("Temperature")
plt.show()
```



In [ ]:
```
model=LinearRegression()
model.fit(train_X,train_y)
```

Out[ ]:
```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None, normalize=False)
```

In [ ]:
```
prediction = model.predict(test_X)
```

In [ ]:
```
#calculating error
np.mean(np.absolute(prediction-test_y))
```

Out[ ]:
```
1.2004735794096681
```

In [ ]:
```
print('Variance score: %.2f' % model.score(test_X, test_y))
```

```
Variance score: 0.96
```

In [ ]:
```
for i in range(len(prediction)):
  prediction[i]=round(prediction[i],2)
pd.DataFrame({'Actual':test_y,'Prediction':prediction,'diff':(test_y-prediction)})
```

Out[ ]:

date_time	Actual	Prediction	diff
2013-07-10 08:00:00	34	34.89	-0.89
2015-11-04 20:00:00	25	24.57	0.43
2015-09-21 09:00:00	34	35.08	-1.08
2017-02-16 11:00:00	28	25.22	2.78
2012-07-21 01:00:00	28	28.04	-0.04
...	...	...	...
2019-03-30 09:00:00	37	33.55	3.45
2015-11-12 12:00:00	32	30.36	1.64
2019-12-31 05:00:00	8	9.13	-1.13
2019-08-02 17:00:00	35	35.92	-0.92
2019-10-22 08:00:00	26	25.77	0.23

19287 rows × 3 columns

Decision Tree Regression

In [ ]:
```
from sklearn.tree import DecisionTreeRegressor
regressor=DecisionTreeRegressor(random_state=0)
regressor.fit(train_X,train_y)
```

Out[ ]:
```
DecisionTreeRegressor(ccp_alpha=0.0, criterion='mse', max_depth=None,
                      max_features=None, max_leaf_nodes=None,
                      min_impurity_decrease=0.0, min_impurity_split=None,
                      min_samples_leaf=1, min_samples_split=2,
                      min_weight_fraction_leaf=0.0, presort='deprecated',
                      random_state=0, splitter='best')
```

In [ ]:
```
prediction2=regressor.predict(test_X)
np.mean(np.absolute(prediction2-test_y))
```

Out[ ]:
```
0.563013083078412
```

In [ ]:
```
print('Variance score: %.2f' % regressor.score(test_X, test_y))
```

```
Variance score: 0.98
```

In [ ]:
```
for i in range(len(prediction2)):
  prediction2[i]=round(prediction2[i],2)
pd.DataFrame({'Actual':test_y,'Prediction':prediction2,'diff':(test_y-prediction2)})
```

Out[ ]:

date_time	Actual	Prediction	diff
2013-07-10 08:00:00	34	34.0	0.0
2015-11-04 20:00:00	25	24.0	1.0
2015-09-21 09:00:00	34	34.0	0.0
2017-02-16 11:00:00	28	27.0	1.0
2012-07-21 01:00:00	28	28.0	0.0
...	...	...	...
2019-03-30 09:00:00	37	32.0	5.0
2015-11-12 12:00:00	32	32.0	0.0
2019-12-31 05:00:00	8	9.0	-1.0
2019-08-02 17:00:00	35	35.0	0.0
2019-10-22 08:00:00	26	26.0	0.0

19287 rows × 3 columns

Random Forest Regression

In [ ]:
```
from sklearn.ensemble import RandomForestRegressor
```

```
regr=RandomForestRegressor(max_depth=90,random_state=0,n_estimators=100)
regr.fit(train_X,train_y)
```

Out[ ]:
```
RandomForestRegressor(bootstrap=True, ccp_alpha=0.0, criterion='mse',
                      max_depth=90, max_features='auto', max_leaf_nodes=None,
                      max_samples=None, min_impurity_decrease=0.0,
                      min_impurity_split=None, min_samples_leaf=1,
                      min_samples_split=2, min_weight_fraction_leaf=0.0,
                      n_estimators=100, n_jobs=None, oob_score=False,
                      random_state=0, verbose=0, warm_start=False)
```

In [ ]:
```
prediction3=regr.predict(test_X)
np.mean(np.absolute(prediction3-test_y))
```

Out[ ]:
```
0.47491654535041405
```

In [ ]:
```
print('Variance score: %.2f' % regr.score(test_X, test_y))
```

```
Variance score: 0.99
```

In [ ]:
```
for i in range(len(prediction3)):
  prediction3[i]=round(prediction3[i],2)
pd.DataFrame({'Actual':test_y,'Prediction':prediction3,'diff':(test_y-prediction3)})
```

Out[ ]:

date_time	Actual	Prediction	diff
2013-07-10 08:00:00	34	33.92	0.08
2015-11-04 20:00:00	25	24.84	0.16
2015-09-21 09:00:00	34	34.25	-0.25
2017-02-16 11:00:00	28	27.00	1.00
2012-07-21 01:00:00	28	27.99	0.01
...	...	...	...
2019-03-30 09:00:00	37	32.79	4.21
2015-11-12 12:00:00	32	31.91	0.09
2019-12-31 05:00:00	8	8.81	-0.81
2019-08-02 17:00:00	35	34.98	0.02
2019-10-22 08:00:00	26	26.32	-0.32

19287 rows × 3 columns

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

Calculating R2-score for Multiple Linear Regression

In [ ]:
```
print("Mean absolute error: %.2f" % np.mean(np.absolute(prediction - test_y)))
print("Residual sum of squares (MSE): %.2f" % np.mean((prediction - test_y) ** 2))
print("R2-score: %.2f" % r2_score(test_y,prediction ) )
```

```
Mean absolute error: 1.20
Residual sum of squares (MSE): 2.51
R2-score: 0.96
```

Calculating R2-score for Decision Tree Regression

In [ ]:
```
print("Mean absolute error: %.2f" % np.mean(np.absolute(prediction2 - test_y)))
print("Residual sum of squares (MSE): %.2f" % np.mean((prediction2 - test_y) ** 2))
print("R2-score: %.2f" % r2_score(test_y,prediction2 ) )
```

```
Mean absolute error: 0.56
Residual sum of squares (MSE): 1.12
R2-score: 0.98
```

Calculating R2-score for Random Forest Regression

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

print("Mean absolute error: %.2f" % np.mean(np.absolute(prediction3 - test_y)))
print("Residual sum of squares (MSE): %.2f" % np.mean((prediction3 - test_y) ** 2))
print("R2-score: %.2f" % r2_score(test_y,prediction3 ) )
```

```
Mean absolute error: 0.47
Residual sum of squares (MSE): 0.63
R2-score: 0.99
```