Date: 14-07-2021

PROBABILITY AND STATISTICS

A report on

Case study on weather temperature

Under the guidance of

Dr. Ramesh Athe
Assistant professor, Humanities and science, IIIT DHARWAD

Report by

SANKEESA ANSHU 20bds047
K.MUNISAI 20bds028
PRAVEEN GUNDYAGOL 20bds042

Introduction:

Now-a-days due to many reasons like global warming, pollution,etc , climate is changing dramatically.It is very important to analyse and predict the changing weather .Inorder to know changes in climate, we have analysed weather using some programming tools and with the knowledge we gained in sem1 and sem 2 in our course probability and statistics.

Objective:

The main objective of this case study is to analyse and predict the temperature and humidity of the present and future.

Methodology:

We have collected data of temperature and humidity of Estes park, Colorado from the year 2005 to 2021 from weather.net website.We used python programming language for our case study.The libraries we used are pandas, matplotlib.pyplot,numpy,etc.And using that data collected we have plotted the distribution of temperature and humidity scatter plot, we fit the data in different curves. We did hypothesis testing, correlation, and regression. We found measures of central tendency and dispersion.We have plotted a pie chart ,histogram,and frequency curve.Finally, we have predicted the temperature in the future. Our prediction has a high chance to be correct.

Extracting data

```
temperatures = []
date = []
humidity = []

def weather_data_extract(code1, code2):
    try:
        source = requests.get('http://www.estesparkweather.net/archive_reports.php?date=' + code1 +
            code2).text
        soup = BeautifulSoup(source, 'lxml')
        tables = soup.find_all('table')
        counter = 0
        for table in tables:
            if counter % 2 == 0:
                tr = table.find('tr', class_='column-light')
                td = list(tr.find_all('td'))
                tr1 = table.find('tr', class_='column-dark')
                td1 = list(tr1.find_all('td'))
                counter += 1
            else:
                tr = table.find('tr', class_='column-dark')
                td = list(tr.find_all('td'))
                tr1 = table.find('tr', class_='column-light')
                td1 = list(tr1.find_all('td'))
                counter += 1
            temp = td[1].text.split('°')
            temperatures.append((temp[0]))
                    date.append(str(counter ) + "/" + code2 + "/" + code1)
            temp1 = td1[1].text.split('%')
            humidity.append(temp1[0])
    except:
        print(" ")
```

Collection of Data & Cleansing of Data for year 2005 to 2021

Code:

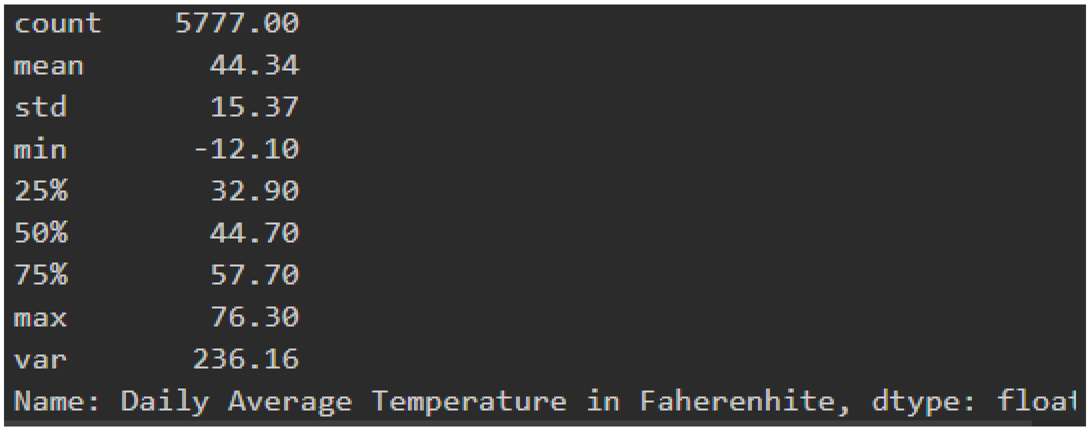
```
year_start = 2005
while year_start <= 2021:
  code = str(year_start)
  weather_data_extract(code, '01')
  weather_data_extract(code, '02')
  weather_data_extract(code, '03')
  weather_data_extract(code, '04')
  weather_data_extract(code, '05')
  weather_data_extract(code, '06')
  if year_start != 2021:
     weather_data_extract(code, '07')
     weather_data_extract(code, '08')
     weather_data_extract(code, '09')
     weather_data_extract(code, '10')
     weather_data_extract(code, '11')
     weather_data_extract(code, '12')
  year_start += 1
```

Finding central tendency and dispersion of temperature and humidity

Temperature:

Code :

```
summary=df['Daily Average Temperature in Faherenhite'].describe()
summary['var']=summary['std']**2
summary.round(2)
```

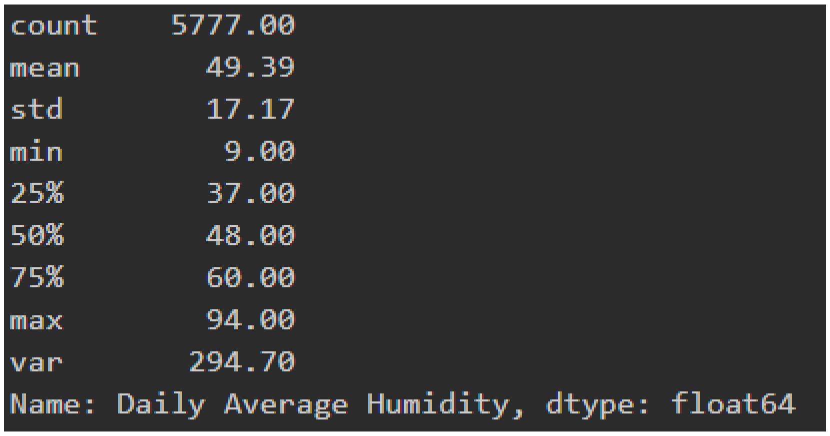


```
count    5777.00
mean       44.34
std        15.37
min       -12.10
25%        32.90
50%        44.70
75%        57.70
max        76.30
var       236.16
Name: Daily Average Temperature in Faherenhite, dtype: floa
```

Humidity:

Code:

```
summary=df['Daily Average Humidity'].describe()
summary['var']=summary['std']**2
summary.round(2)
```



```
count    5777.00
mean       49.39
std        17.17
min         9.00
25%        37.00
50%        48.00
75%        60.00
max        94.00
var       294.70
Name: Daily Average Humidity, dtype: float64
```

Scatter plot:

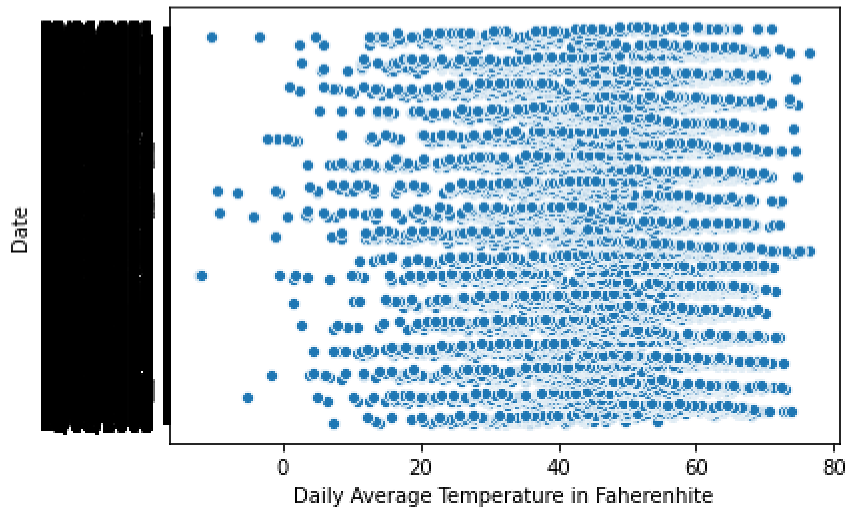
Code:

```
import scipy.stats as stats
import seaborn as sns
```

Temperature:

code:

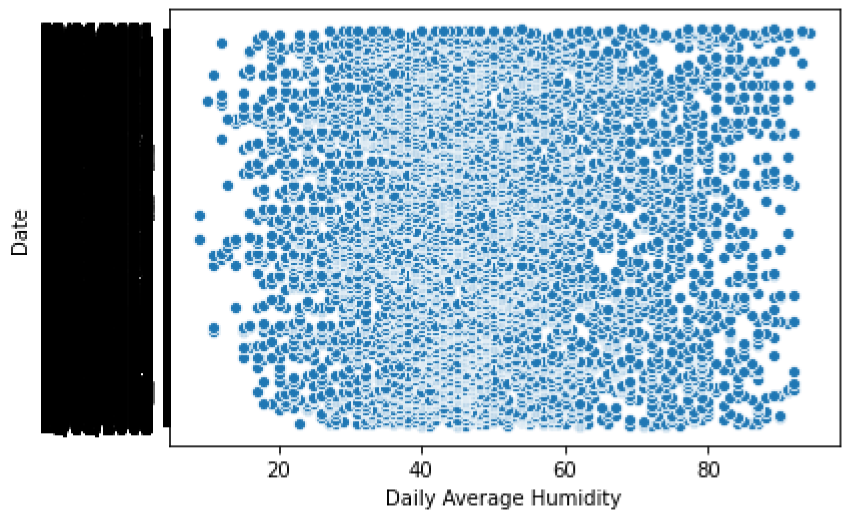
```
sns.scatterplot(df['Daily Average Temperature in Faherenhite'],df['Date'])
```



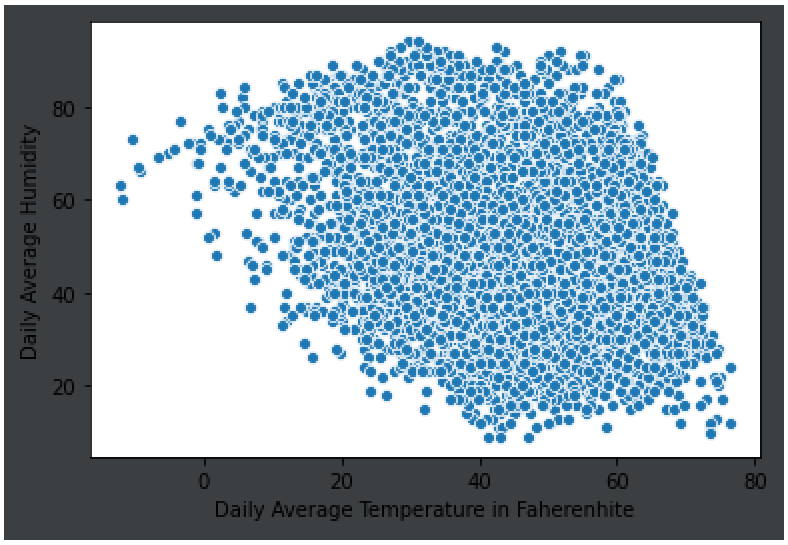
<u>Humidity:</u>

code:

```
sns.scatterplot(df['Daily Average Humidity'],df['Date']
```



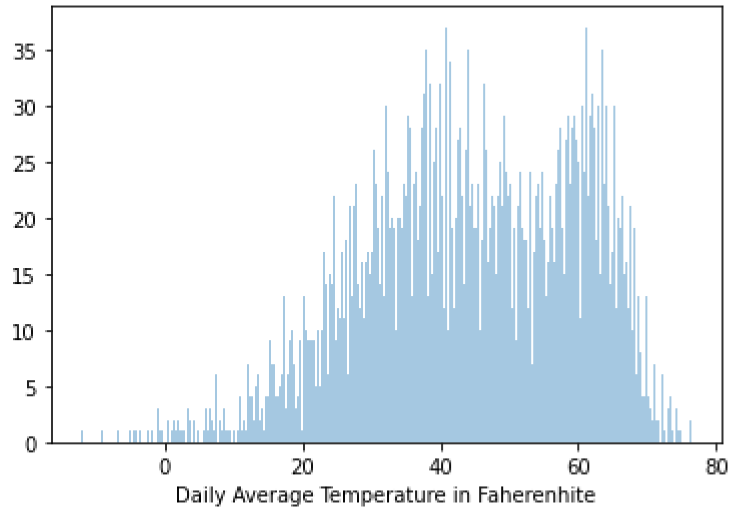
```
sns.scatterplot(df['Daily Average Temperature in Faherenhite'],df['Daily Average Humidity'])
```



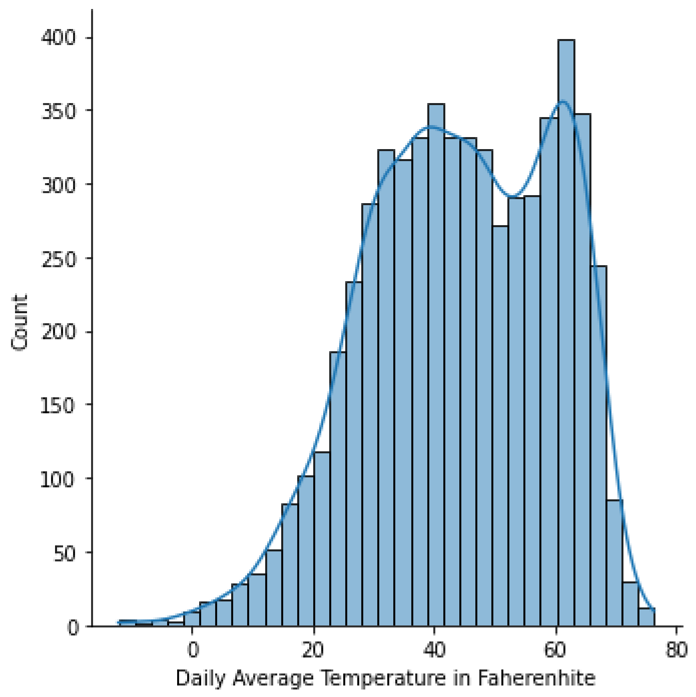
Distribution plot:

code:

```
sns.distplot(df['Daily Average Temperature in Faherenhite'],bins=500,kde=False);
```



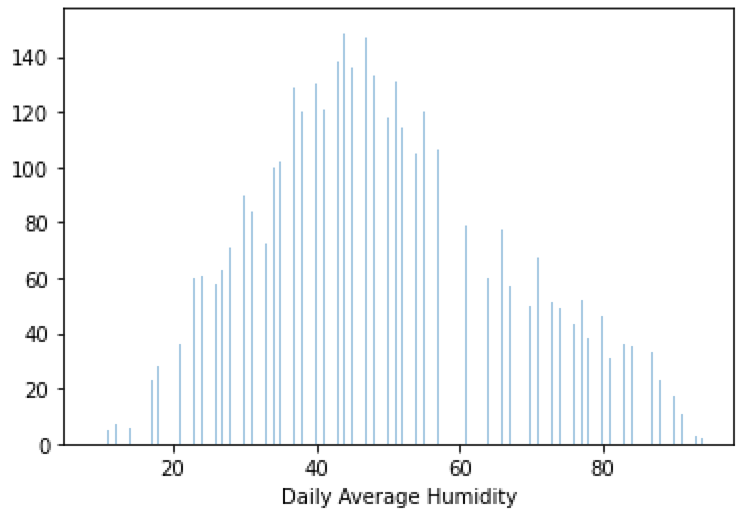
```
sns.displot(df['Daily Average Temperature in Faherenhite'],kde=True)
```



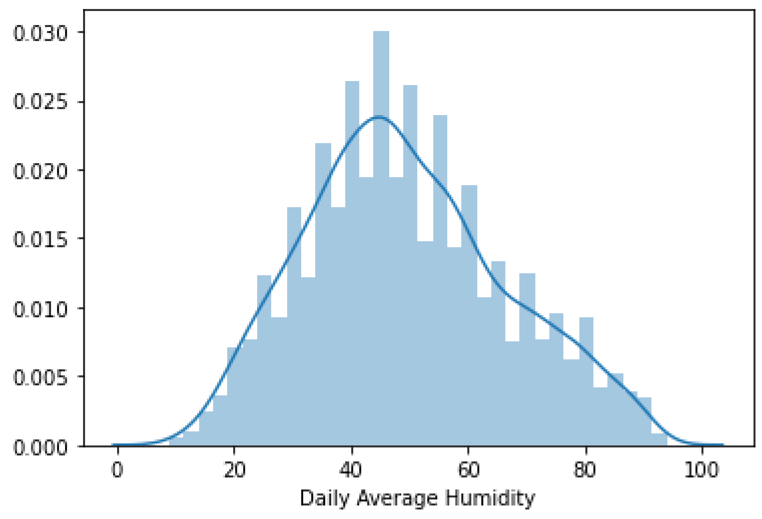
Similarly,<u>Humidity:</u>

<u>code:</u>

```
sns.distplot(df['Daily Average Humidity'],bins=500,kde=False);
```



```
sns.distplot(df['Daily Average Humidity'],kde=True)
```



Skewness and kurtosis

Skewness of humidity:

code:

```
    df['Daily Average Humidity'].skew()
    Skewness is 0.2994621069055353
```

Our data for average humidity is positively skewed.

Skewness of temperature:

code:

```
    df['Daily Average Temperature in Faherenhite'].skew()
    Skewness of temperature is -0.3268096044941174
```

This shows our data for humidity is negatively skewed.

Kurtosis of humidity:

code:

```
    df['Daily Average Humidity'].kurtosis()
    kurtosis for humidity is -0.4772548756204804
```

It shows that our data is platykurtic

Kurtosis of temperature:

code:

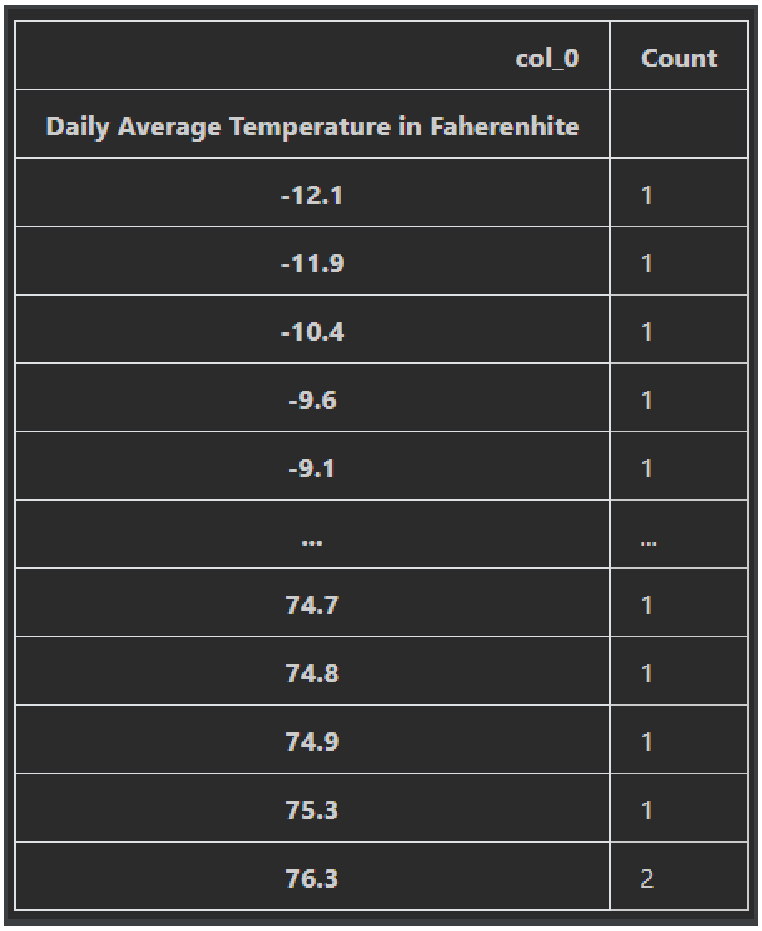
```
    df['Daily Average Temperature in Faherenhite'].kurtosis()
    Kurtosis for temperature is -0.51147507
```
It also platykurtic

Now making frequency table for our data:

code:

For temperature we made frequency table

```
df2 = pd.crosstab(index=df['Daily Average Temperature in Faherenhite'], columns='Count')
Df2
```

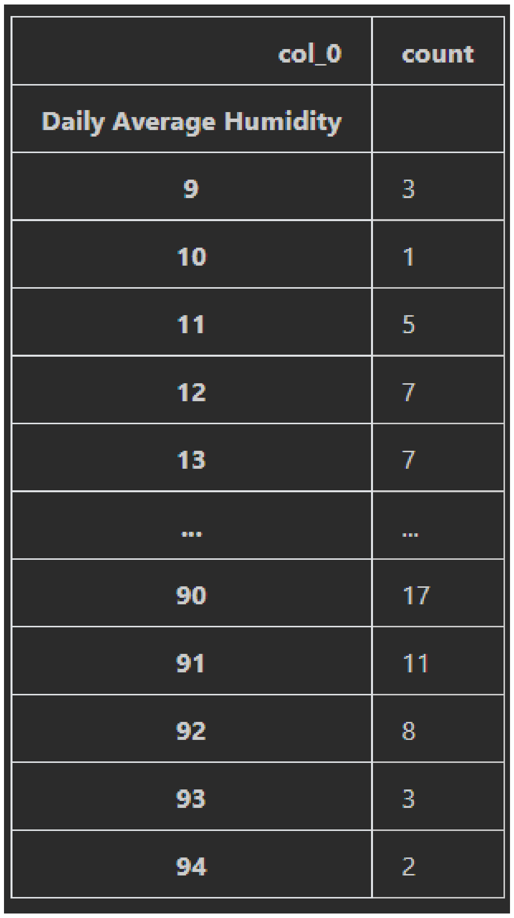


col_0	Count
Daily Average Temperature in Faherenhite	
-12.1	1
-11.9	1
-10.4	1
-9.6	1
-9.1	1
...	...
74.7	1
74.8	1
74.9	1
75.3	1
76.3	2

For humidity

code:

```
df3 =pd.crosstab(index=df['Daily Average Humidity'], columns='count')
df3
```



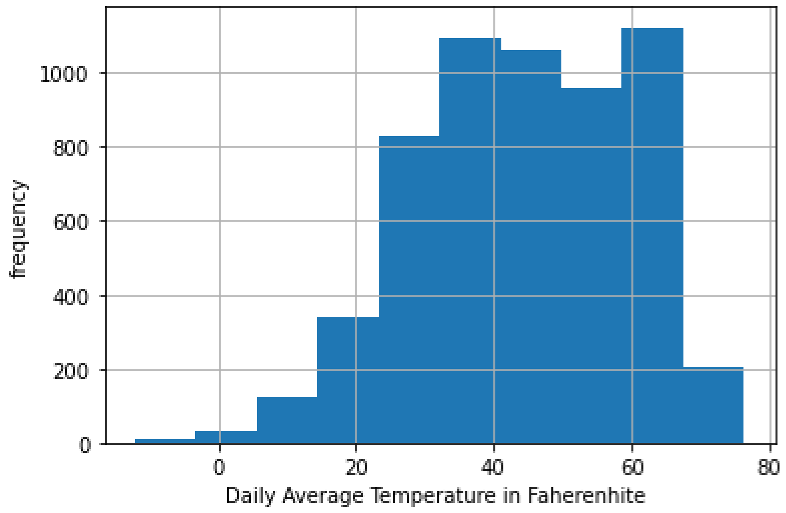
col_0	count
Daily Average Humidity	
9	3
10	1
11	5
12	7
13	7
...	...
90	17
91	11
92	8
93	3
94	2

Now, making histogram, pie chart, frequency polygon graph,ogive

Histogram for temperature:

code:

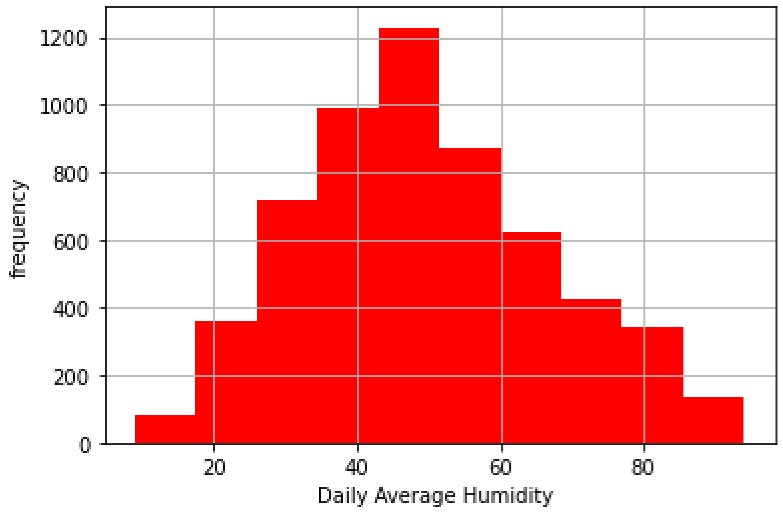
```
ax=df['Daily Average Temperature in Faherenhite'].hist()
ax.set_xlabel("Daily Average Temperature in Faherenhite ")
ax.set_ylabel("frequency")
plt.show()
```



<u>Histogram for humidity</u>:

code:

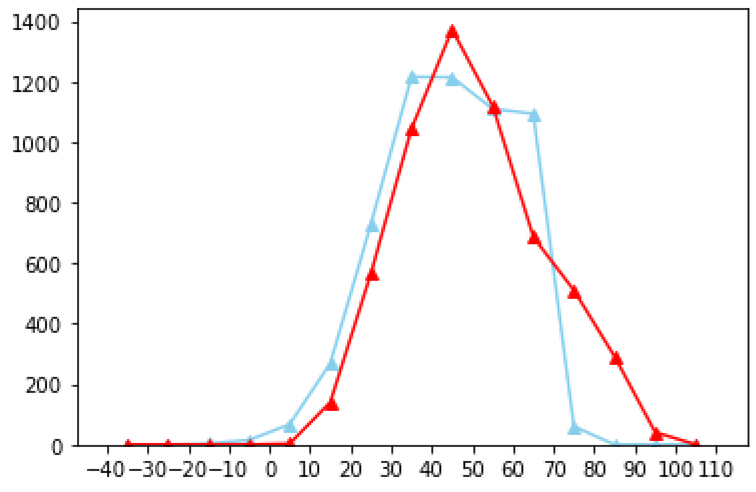
```
    ax=df['Daily Average Humidity'].hist(color = "red")
    ax.set_xlabel("Daily Average Humidity ")
    ax.set_ylabel("frequency")
    plt.show()
```



<u>Frequency polygon for temperature:</u>

code:

```
intervals=[-40,-30,-20,-10,0,10,20,30,40,50,60,70,80,90,100,110]

plt.xticks(intervals)
y,edges,_=plt.hist(df['Daily Average Temperature in Faherenhite'],bins=intervals, histtype
        ='step',edgecolor='white')

midpoints=0.5*(edges[1:]+edges[:-1])
plt.plot(midpoints,y,'r-^',color = "skyblue")
plt.xticks(intervals)
y,edges,_=plt.hist(df['Daily Average Humidity'],bins=intervals, histtype ='step',edgecolor='white')

midpoints=0.5*(edges[1:]+edges[:-1])
plt.plot(midpoints,y,'r-^')
```

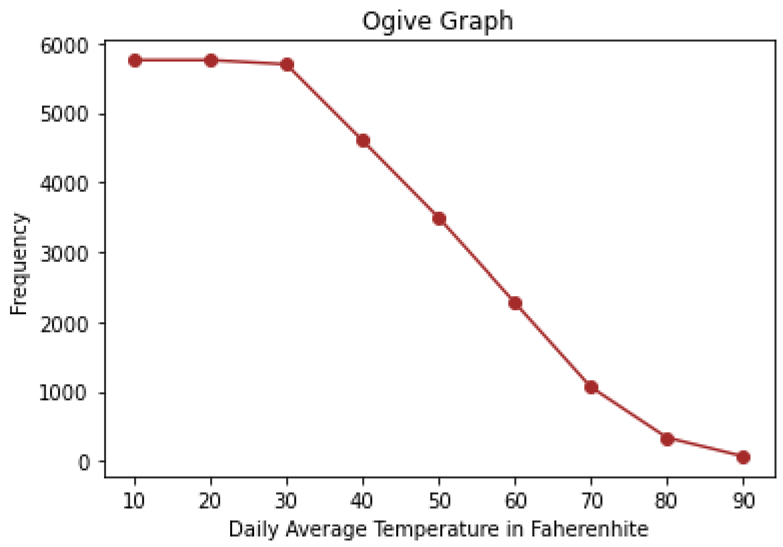


<u>Ogive for temperature:</u>

code:

```
data = df['Daily Average Temperature in Faherenhite']

 # creating class interval
classInterval = [0, 10, 20, 30, 40, 50, 60, 70, 80, 90]

# calculating frequency and intervals
values, base = np.histogram(data, bins=classInterval)
```

```
# calculating cumulative frequency
cumsum = np.cumsum(values)

# reversing cumulative frequency
res = np.flipud(cumsum)

# plotting ogive
plt.plot(base[1:], res, color='brown', marker='o', linestyle='-')

# formatting the graph
plt.title('Ogive Graph')
plt.xlabel('Daily Average Temperature in Faherenhite')
plt.ylabel('Frequency')
```

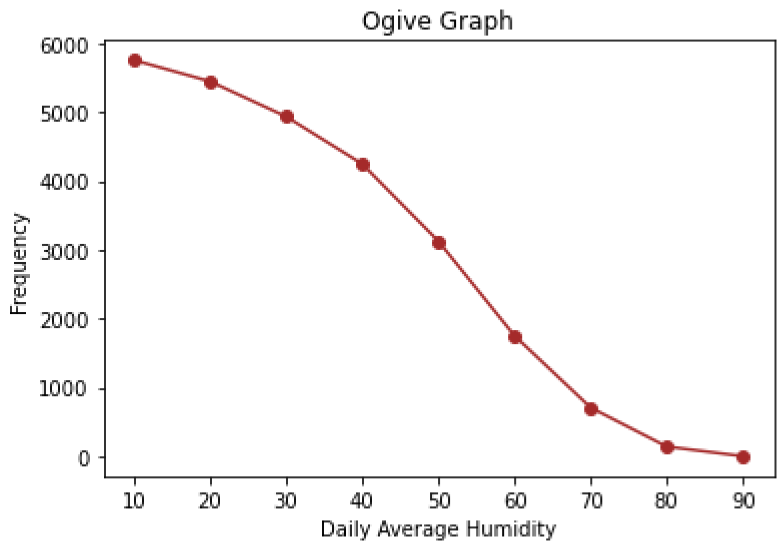


Ogive for humidity:

code:

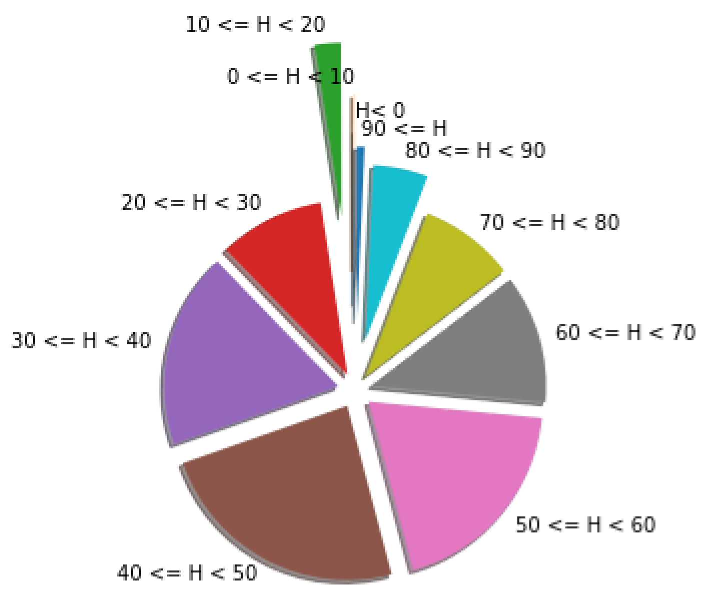
```
data = df['Daily Average Humidity']

# creating class interval
classInterval = [0, 10, 20, 30, 40, 50, 60, 70, 80, 90]

# calculating frequency and intervals
 values, base = np.histogram(data, bins=classInterval)

# calculating cumulative frequency
```

```
 cumsum = np.cumsum(values)

# reversing cumulative frequency
res = np.flipud(cumsum)

# plotting ogive
plt.plot(base[1:], res, color='brown', marker='o', linestyle='-')

# formatting the graph
plt.title('Ogive Graph')
plt.xlabel('Daily Average Humidity')
plt.ylabel('Frequency')
```



Pie chart for Humidity:

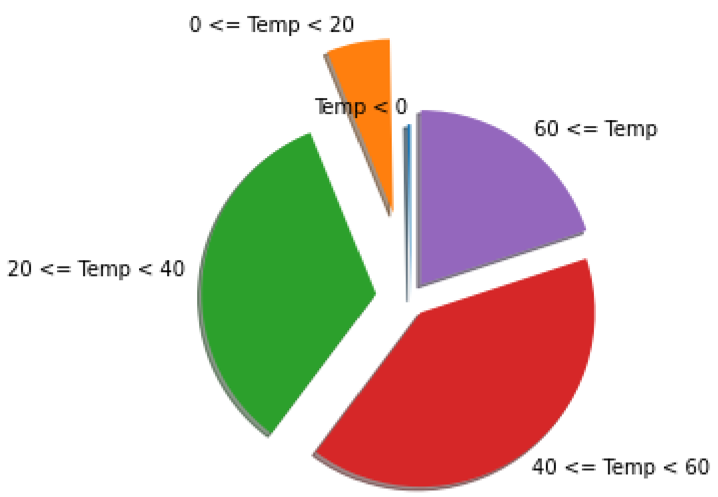
```
import matplotlib.pyplot as plt

labels = 'H< 0', '0 <= H < 10', '10 <= H < 20', '20 <= H < 30', '30 <= H < 40','40 <= H < 50','50 <= H <
    60','60 <= H < 70','70 <= H < 80','80 <= H < 90','90 <= H'
sizes = [size_1, size_2, size_3, size_4, size_5, size_6, size_7, size_8, size_9, size_10, size_11]

fig1, ax1 = plt.subplots()
explode = (0.5, 0.7, 1, 0.1,0.1,0.1, 0.1, 0.1, 0.1,0.3,0.4)
ax1.pie(sizes, explode=explode, labels=labels,
     shadow=True, startangle=90)

  plt.show()
```



<u>Pie chart for Temperature:</u>

Code:

```
import matplotlib.pyplot as plt

labels = 'Temp < 0', '0 <= Temp < 20', '20 <= Temp < 40', '40 <= Temp < 60', '60 <= Temp'
sizes = [size1, size2, size3, size4,size5]

fig1, ax1 = plt.subplots()
explode = (0, 0.5, 0.2, 0.1,0.1)
ax1.pie(sizes, explode=explode, labels=labels,
        shadow=True, startangle=90)

plt.show()
```

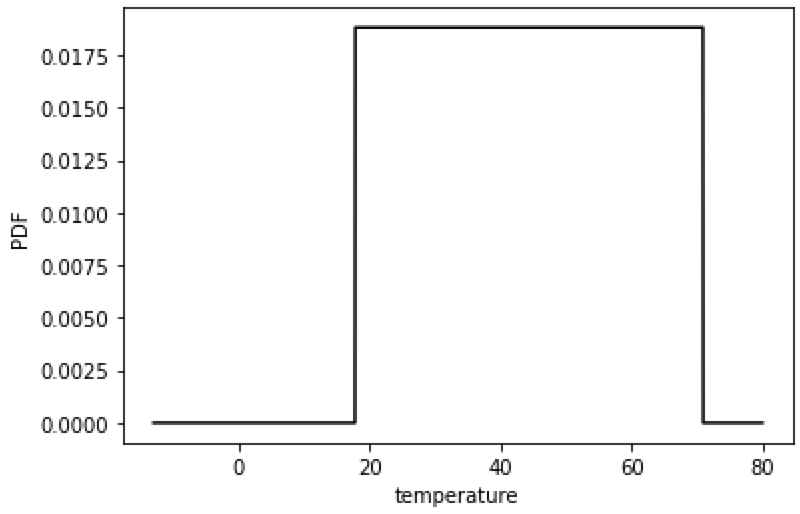


Probability density function:

code:

As our is approx uniformly distributed

```
#as we know that mean =44.35 and s.d 15.34
# calculating alpha and beta values
#mean = (alpha +beta)/2 and variance = ((alpha-beta)^2)/12
#by solving them we get alpha and beta values as
alpha =17.78
beta= 70.92
pdf=stats.uniform.pdf(x,loc=alpha,scale=53.14)
f(x) = 0.011275451252264085 ;if alpha<= x<=beta
     = 0 ; otherwise
plt.plot(x,pdf,color='black')
plt.xlabel('temperature')
plt.ylabel("PDF");
plt.show()
```

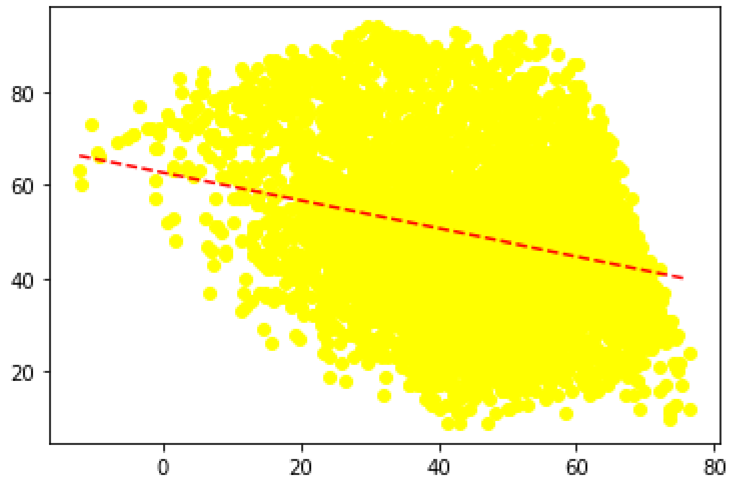


Curve fitting :

Code:

```
from scipy.optimize import curve_fit
Fitting straight line

# define the true objective function
def objective(x, a, b):
  return a * x + b

# curve fit
popt, _ = curve_fit(objective, num, freq)

# summarize the parameter values
a, b = popt
print('y = %.5f * x + %.5f' % (a, b))

# plot input vs output
plt.scatter(num, freq)

# define a sequence of inputs between the smallest and largest known inputs
x_line = nm.arange(min(num), max(num), 1)
```

```
# calculate the output for the range
y_line = objective(x_line, a, b)

# create a line plot for the mapping function
plt.plot(x_line, y_line, '-*', color='red')
plt.show()
Straight line equation is y = -0.29881 * x + 62.63814
```



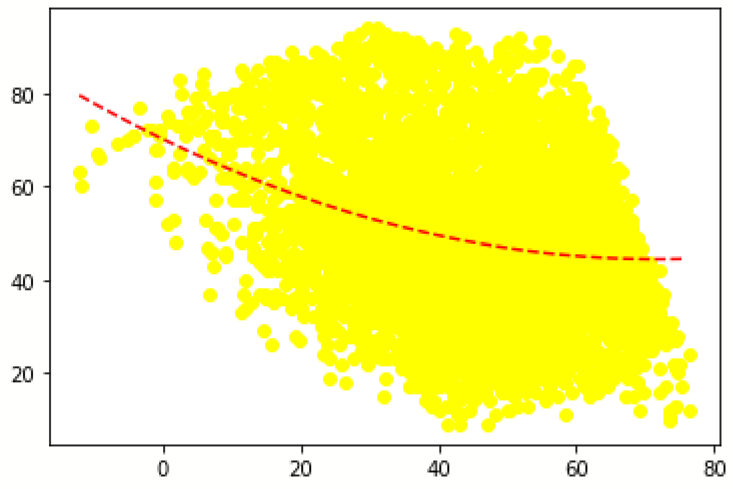
Fitting second degree parabola:

Code:

```
def obj(x, a, b, c):
  return a * x + b * x**2 + c
popt, _ = curve_fit(obj, num, freq)

# summarize the parameter values
a, b, c = popt
print('y = %.5f * x + %.5f * x^2 + %.5f' % (a, b, c))

# plot input vs output
plt.scatter(num, freq)

# define a sequence of inputs between the smallest and largest known inputs
x_line = nm.arange(min(num), max(num), 1)
```

```
# calculate the output for the range
y_line = obj(x_line, a, b, c)

# create a line plot for the mapping function
plt.plot(x_line, y_line, '-^', color='red')
plt.show()
```

Parabola equation is y = -0.71619 * x + 0.00499 * x^2 + 70.15912

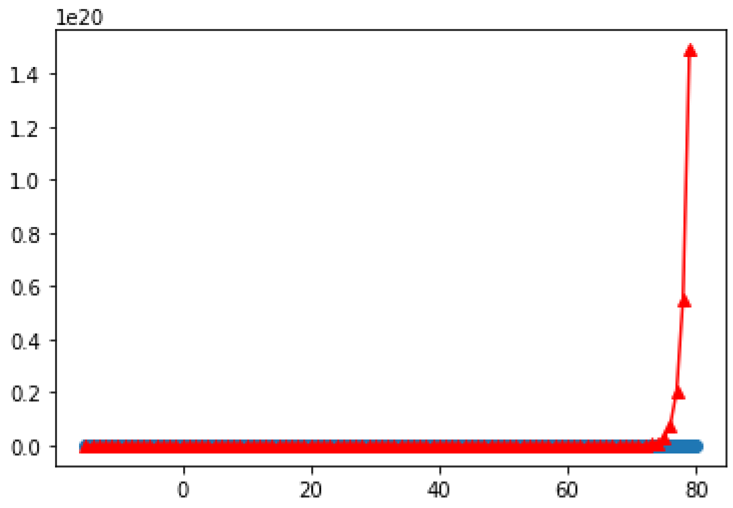


fitting exponential curve:

Code:

```
def exponential(x, a, b):
   return a*(2.718**(b*x))
popt, _ = curve_fit(exponential, num, freq)
# summarize the parameter values
a, b = popt
print('y = %.5f * x^%.5f' % (a, b))
# plot input vs output
plt.scatter(num, freq)


# define a sequence of inputs between the smallest and largest known inputs
x_line = nm.arange(min(num), max(num), 1)
```

```
# calculate the output for the range
y_line = exponential(x_line, a, b)
# create a line plot for the mapping function
plt.plot(x_line, y_line, '-^', color='red')
plt.show()
Exponential curve equation is y = 0.00000 * x^1.00000
```

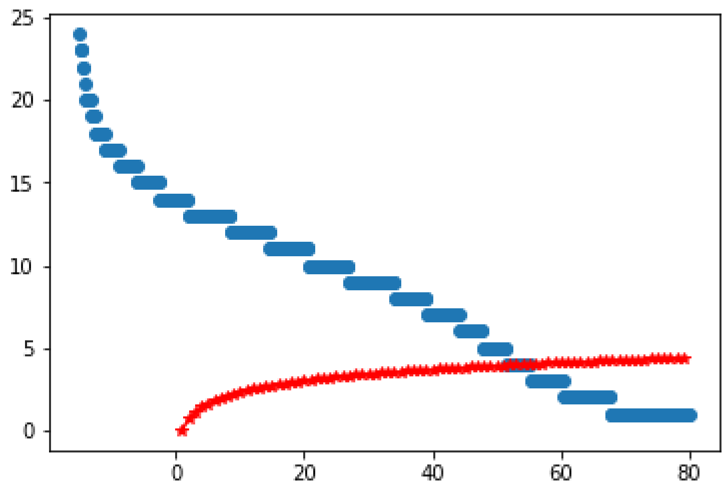


Fitting power curve:

Code:

```
def power(x, a, b):
   return a *(x**b)
popt, _ = curve_fit(power, num, freq)
# summarize the parameter values
a, b = popt
print('y = %.5f * x^%.5f' % (a, b))
# plot input vs output
plt.scatter(num, freq)
# define a sequence of inputs between the smallest and largest known inputs
x_line = nm.arange(min(num), max(num), 1)
# calculate the output for the range
y_line = power(x_line, a, b)
# create a line plot for the mapping function
plt.plot(x_line, y_line, '-^', color='red')
plt.show()
```

Power curve equation is y = 1.00000 * x^1.00000



Hypothesis testing:

Code:

```
# NULL Hypothesis H(0): mean of temperatures = 45 degree Faherenhite
#ALTERNATIVE Hypothesis H(a): mean of temperature != 45 degree faherenhite
import numpy as np
import math
import pandas as pd
from statistics import mean

data1 = pd.read_csv('math_assignment2.csv')
data1['Daily ]Average Temperature in Faherenhite'].mean()
Mean =44.34

data1['Daily Average Humidity'].mean()
mean=49.39
        Let us have the value of alpha set to 0.5 i.e for Type 1 error
        This is a 2 sided test do the value of z at 0.5 making it 0.025 for the 2 sides from the z-table is
        +1.96/-1.96
sampData = data1['Daily Average Temperature in
        Faherenhite'][np.argsort(np.random.random(5777))[:100]]

meanSampData = sampData.mean()
hypMean = 45
```

```
N = 100
stand_Population = np.std(data1['Daily Average Temperature in Faherenhite'])

(meanSampData - hypMean)/(stand_Population/math.sqrt(N))
-1.45
# As the calculated z-score i.e 0.5855006114330584 is less than the tabular z-score i.e +1.96,
#  we accept the NULL hypothesis

# If also we would have got -0.5855006114330584 which was greater than the tabular z-score i.e -1.96,
# we would have accepted the NULL Hypothesis


# Observed Value = + 0.5855006114330584
# Critical Value = + 1.96

# By rejecting the NULL hypothesis when it had to be rejected, we have saved ourselves from making the
        TYPE I Error,
# as the value of the mean is not actually 45 degree Fahrenheit.

# Now let us perform hypothesis testing using P-Value Method (Observed Significance Level)
# The P-Value defines the smallest probability (a) for which the NULL Hypothesis can be rejected.

# Let us define the smallest probability value  a = 0.05

# Now if p is less than or equal to a, we will get a strong evidence against NULL Hypothesis
# Hence, we will reject the NULL Hypothesis
# If p is greater than a, we will get a weak evidence against NULL Hypothesis
# Thus, we will fail to reject the NULL Hypothesis
# As the value of p gets nearer to a,
# the evidence becomes more confusing and we become less sure about rejecting or not rejecting the
        NULL Hypothesis
# Now the Observed z-score value is 0.5855006114330584 which is approximately 0.58
# For 0.58 in the z-score table, we get the standard normal probability of 0.71904

# So, our P-Value is 1-0.71904 = 0.28096
# As the P-Value is much greater than the set a, we have a very weak evidence against the NULL
        Hypothesis
# Thus, we fail to reject the NULL Hypothesis

import scipy.stats as st
# This built in libarary function takes in the sample Data and the hypothised mean value
st.ttest_1samp(sampData,45)

Ttest_1sampResult(statistic=-1.2956164970906812, pvalue=0.198120413636069)
```

```
0.198120413636069> 0.05 # we fail to reject the NULL Hypothesis
True
# Now let us create another NULL Hypothesis
# NULL Hypothesis: mean = 40 degree Farehnite
# We can test this hypothesis using the P-Value Method
st.ttest_1samp(sampData, 37)
Ttest_1sampResult(statistic=3.3690693610215794, pvalue=0.0010759331830419035)
0.0010759331830419035 < 0.05 # we reject the hypothesis
True
```

correlation

It is a measure of degree of relatedness of variables
There are various types of correlation coefficients:

1. Pearson's Coefficient
2. Spearman's Coefficient
3. Kendall Coefficient

```
import seaborn as sns
%matplotlib inline
np.corrcoef(data1['Daily Average Temperature in Faherenhite'], data1['Daily Average Humidity'])
#Here we obtain a matrix and find out the Pearson's correlation between Average Temperature and
            Average Humidity
# Correlation of 1 is between itself and with each other they have a correlation of - 0.26749585
array([[ 1.        , -0.26749585],    [-0.26749585,  1.        ]])
# Now let us calculate all the 3 types of correlations
```

1.Pearson's Coefficient

```
st.pearsonr(data1['Daily Average Temperature in Faherenhite'], data1['Daily Average Humidity'])
(-0.26749585424355415, 3.091294263611235e-95)
```

2. Spearman's Coefficient

```
st.spearmanr(data1['Daily Average Temperature in Faherenhite'], data1['Daily Average Humidity'])

SpearmanrResult(correlation=-0.23285873284950573, pvalue=5.527272991599998e-72)
```

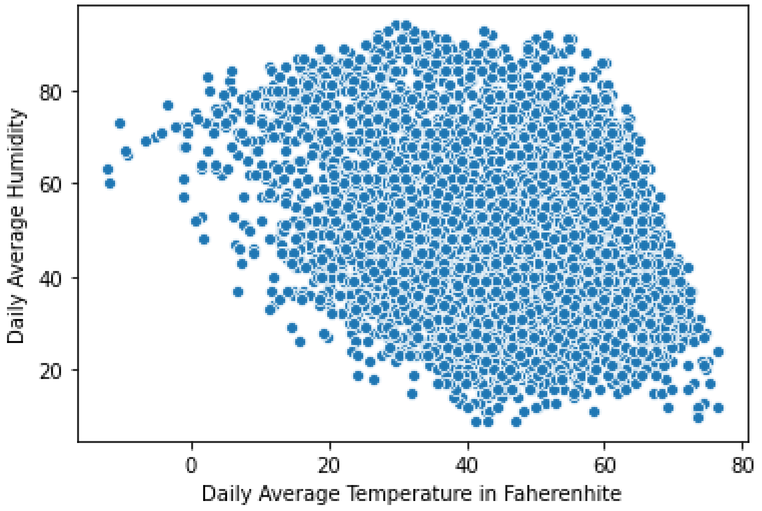
3. Kendall Coefficient

```
st.kendalltau(data1['Daily Average Temperature in Faherenhite'], data1['Daily Average Humidity'])
```

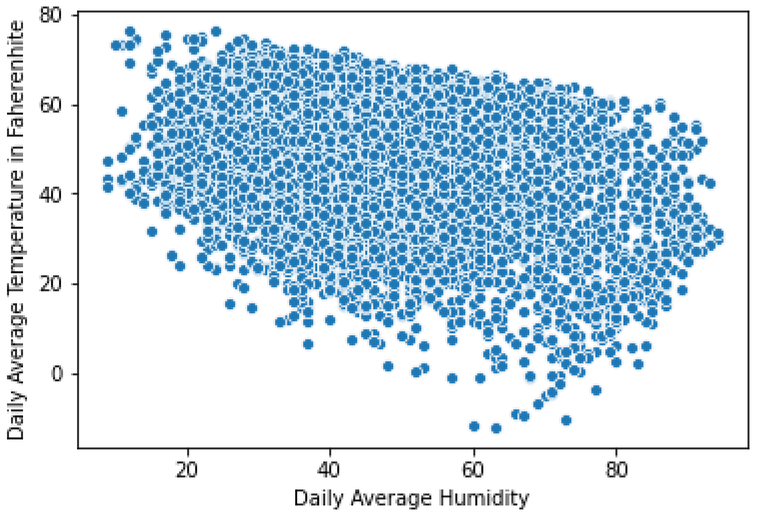
```
KendalltauResult(correlation=-0.15847186888264012, pvalue=1.17299611302675
```

If we see it graphically then we can have a better idea of correlation

```
sns.scatterplot(data1['Daily Average Temperature in Faherenhite'], data1['Daily Average Humidity'])
```

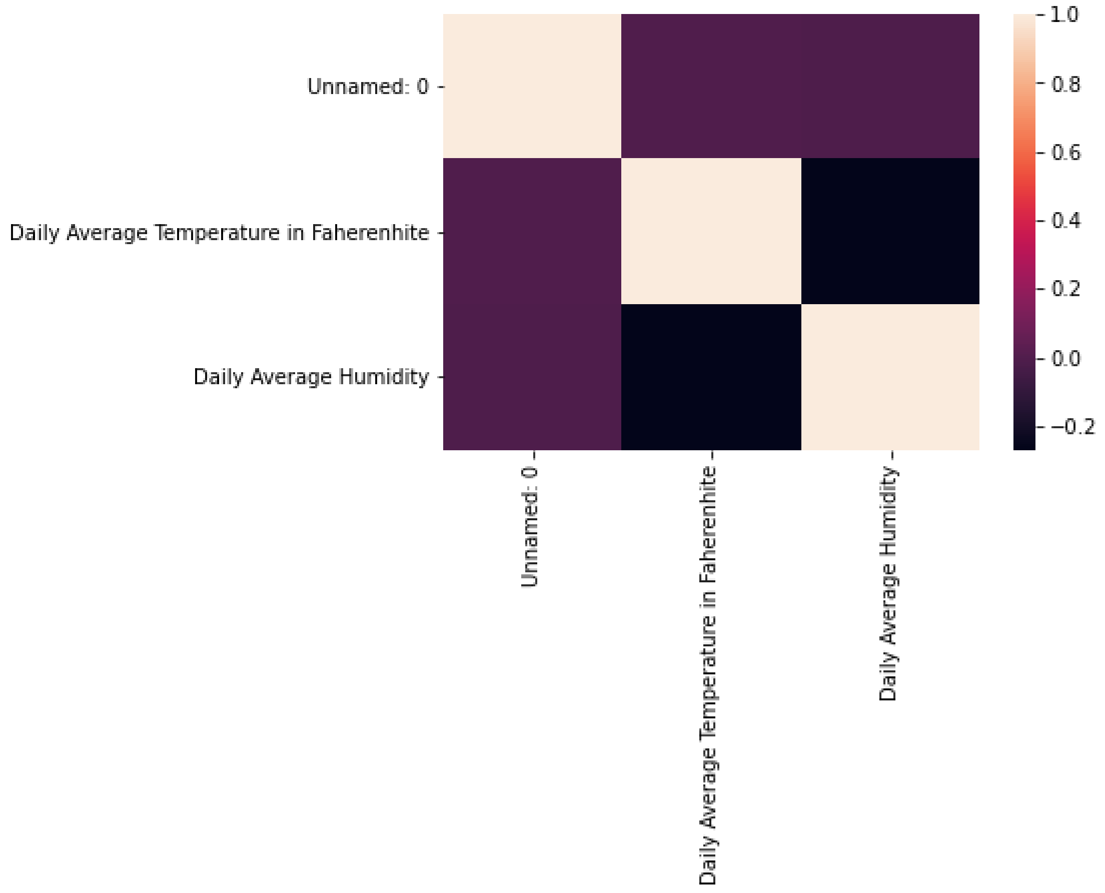


```
sns.scatterplot(data1['Daily Average Humidity'], data1['Daily Average Temperature in Faherenhite'])
```



A Correlation Heatmap also helps us understand Correlation between various variables very easily

```
sns.heatmap(data1.corr())
```



Now let us build a linear regression

```
from sklearn import linear_model

avg_temperature = pd.DataFrame(data1['Daily Average Temperature in Faherenhite'])
avg_humidity = pd.DataFrame(data1['Daily Average Humidity'])

lm = linear_model.LinearRegression()
model = lm.fit(avg_temperature, avg_humidity)

model.coef_
-0.299
model.intercept_
62.64
model.score(avg_temperature, avg_humidity)
0.072
```

Linear regression for temperature:

Code:

```
dataset = pd.read_csv('math_assignment2.csv')
X = dataset.iloc[:, 2:3].values
y = dataset.iloc[:,-1].values
```

Now we split the dataset for training and testing purposes

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X,y,test_size=1/3,random_state=0)
```

Now we fit the regression model using the training set

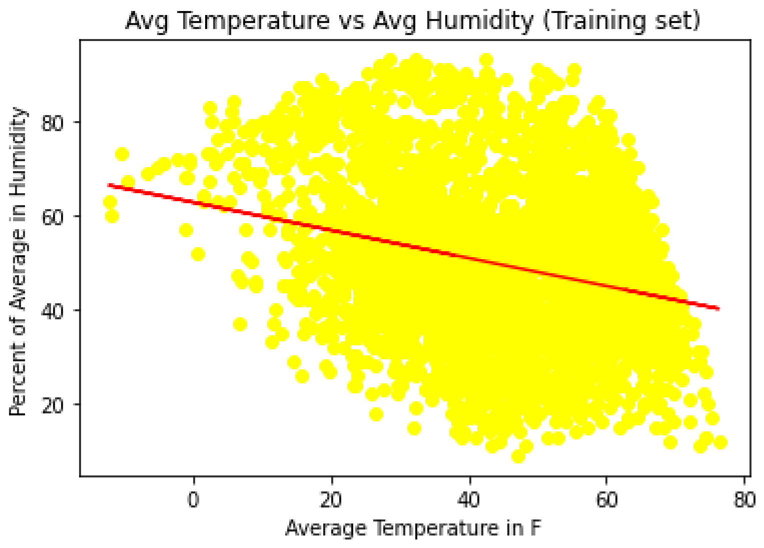
```
from sklearn.linear_model import LinearRegression
regressor = LinearRegression()
regressor.fit(X_train,y_train) #actually produces the linear eqn for the data


# Now as we have trained the model we can predict the results and check from the testing set
y_pred = regressor.predict(X_test)
y_pred

y_test
```

```
array([46, 53, 16, ..., 61, 54, 67], dtype=int64)
```

Graphical representation:

Code:

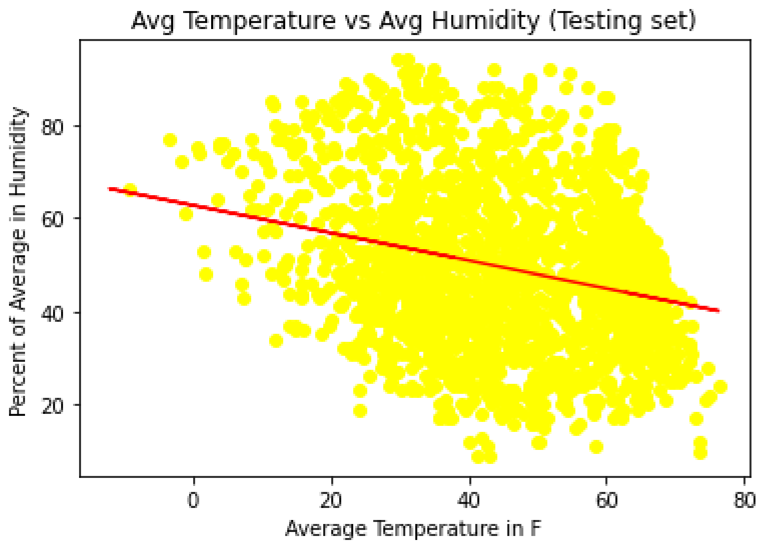
```
plt.scatter(X_train, y_train, color='yellow') # plotting the observation line
plt.plot(X_train, regressor.predict(X_train), color='red') # plotting the regression line
plt.title("Avg Temperature vs Avg Humidity (Training set)") # stating the title of the graph

plt.xlabel("Average Temperature in F") # adding the name of x-axis
plt.ylabel("Percent of Average in Humidity") # adding the name of y-axis
plt.show() # specifies end of graph
```



Now, <u>plotting for the test:</u>

Code:

```
plt.scatter(X_test, y_test, color='yellow')
plt.plot(X_train, regressor.predict(X_train), color='red') # plotting the regression line
plt.title("Avg Temperature vs Avg Humidity (Testing set)")

plt.xlabel("Average Temperature in F")
plt.ylabel("Percent of Average in Humidity")
plt.show()
```



Moments :

```
from scipy import stats
import numpy as np
```

Central moments for temperature

0th central moment

```
stats.moment(data1['Daily Average Humidity'], moment = 0)
1.0
```

1st central moment

```
stats.moment(data1['Daily Average Humidity'], moment = 1)
0.0
```

2nd central moment

```
stats.moment(data1['Daily Average Humidity'], moment = 2)
294.6465687427377
```

3rd central moment

```
stats.moment(data1['Daily Average Humidity'], moment = 3)
1514.1927772196852
```

4th central moment

```
stats.moment(data1['Daily Average Humidity'], moment = 4)
218961.85550016686
```

Non central moment for temperature

```
def raw_moments_T(n,m):
    sum = 0
    N = 0
    for ele in data1['Daily Average Temperature in Faherenhite']:
        sum += (ele-m)**n
        N += 1
    return (sum/N)

print("-----------Raw Moments for Daily Average Temperature in Faherenhite\n")
n = int(input("\nPlease enter the value of 'n' to find the nth raw moment:"))
print(f"\nPlease enter the point about which you will like to find the {n}th raw moment:")
m = int(input())
```

```
print(f"The {n}th raw moment about {m} is {raw_moments_T(n,m)}")
```

```
n=1
m=0
Ist  non central moment about 0
  44.34412324736016
```

Non central moment for humidity.

Code:

```
def raw_moments_H(n,m):
    sum = 0
    N = 0
    for ele in data1['Daily Average Humidity']:
        sum += (ele-m)**n
        N += 1
    return (sum/N)

n = int(input("\nPlease enter the value of 'n' to find the nth raw moment:"))
print(f"\nPlease enter the point about which you will like to find the {n}th raw moment:")
m = int(input())
print(f"The {n}th raw moment about {m} is {raw_moments_H(n,m)}")
```

```
n=1
m=0
Ist  non central moment about 0
49.38757140384283
```

Results:

- The minimum and maximum temperature recorded in Estes park from the year 2005 to 2021 are -12 F and 76.3 F
- The range of temperature and humidity which is most commonly recorded in Estes park is 30-60 F and 20-60 respectively.
- The range of temperature and humidity which is rarely recorded in Estes park is <0 & >75 and <20 & >80
- From correlation and regression graphs, we can observe that the temperature increases as humidity decreases so we can say that they are inversely proportional.and humidity and temperature are found to be moderate.
- There is slight increase in temperature from 2005 to 2020.But, there is a down fall in temperature in 2021 because of decrease in pollution due to lockdown all over the world in 2021 so , there was a huge impact of lockdown on temperature.
- By 2022, we can make a prediction that there will be a low temperature when compared to previous years i.e, before 2020.

Conclusion:

These methods are extremely easy to adopt as they don’t require any specific computational power like Deep Learning methods (RNN, CNN …),machine learning methods and finished predicting using very famous libraries of python and inbuilt functions for some .Nonetheless, predictions perfectly fit in the error range designed by the dataset itself. It is important to consider that we   have examined daily average values while it may not be interesting to consider monthly values and have predictions.

Reference:

https://weather.in/