IDS Project Presentation

2019-20

5/4/2020

RAHIL N MODI PES1201802826

SOORYANATH.I.T PES1201802827

HIMANSHU JAIN PES1201802828

Description about Dataset

- Total Columns :7
- Numerical Columns : 2
  - Average Temperature per month
  - Average Temperature Uncertainty
- Categorical Columns:5
  - Date (observations recorded on 1st of every month)
  - City
  - Country
  - Longitude and Latitude

The Data

```
In [1]: import numpy as np
        import pandas as pd
        import matplotlib.pyplot as plt
        import seaborn as sns
        import statistics as st
        %matplotlib inline

In [2]: df = pd.read_csv('GlobalLandTemperaturesByMajorCity.csv')

In [3]: df.head()
```

Out[3]:

	dt	AverageTemperature	AverageTemperatureUncertainty	City	Country	Latitude	Longitude
0	1849-01-01	26.704	1.435	Abidjan	Côte D'Ivoire	5.63N	3.23W
1	1849-02-01	27.434	1.362	Abidjan	Côte D'Ivoire	5.63N	3.23W
2	1849-03-01	28.101	1.612	Abidjan	Côte D'Ivoire	5.63N	3.23W
3	1849-04-01	26.140	1.387	Abidjan	Côte D'Ivoire	5.63N	3.23W
4	1849-05-01	25.427	1.200	Abidjan	Côte D'Ivoire	5.63N	3.23W

```
In [4]: df.sample(10)
```

Out[4]:

	dt	AverageTemperature	AverageTemperatureUncertainty	City	Country	Latitude	Longitude
156481	1850-12-01	-4.916	1.189	Moscow	Russia	55.45N	36.85E
170424	1798-06-01	21.073	1.807	New York	United States	40.99N	74.56W
194254	1811-08-01	14.590	4.043	Saint Petersburg	Russia	60.27N	29.19E
63351	1883-08-01	20.520	0.635	Chicago	United States	42.59N	87.27W
112050	1992-02-01	17.013	0.307	Kanpur	India	26.52N	80.60E

```
In [5]: df.shape

Out[5]: (239177, 7)

In [6]: df.describe()
```

Out[6]:

	AverageTemperature	AverageTemperatureUncertainty
count	228175.000000	228175.000000
mean	18.125969	0.969343
std	10.024800	0.979644
min	-26.772000	0.040000
25%	12.710000	0.340000
50%	20.428000	0.592000
75%	25.918000	1.320000
max	38.283000	14.037000

```
In [7]: df.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 239177 entries, 0 to 239176
Data columns (total 7 columns):
dt                               239177 non-null object
AverageTemperature               228175 non-null float64
AverageTemperatureUncertainty    228175 non-null float64
City                             239177 non-null object
Country                          239177 non-null object
Latitude                         239177 non-null object
Longitude                        239177 non-null object
```

Data Cleaning using interpolate method


```
In [4]: df['AverageTemperature'].isnull().value_counts()

Out[4]: False    228175
        True      11002
        Name: AverageTemperature, dtype: int64

In [5]: df['AverageTemperatureUncertainty'].isnull().value_counts()

Out[5]: False    228175
        True      11002
        Name: AverageTemperatureUncertainty, dtype: int64

In [6]: df[19945:19949]

Out[6]:
```

	dt	AverageTemperature	AverageTemperatureUncertainty	City	Country	Latitude	Longitude
19945	2013-07-01	25.008	0.416	Bangalore	India	12.05N	77.26E
19946	2013-08-01	25.236	0.520	Bangalore	India	12.05N	77.26E
19947	2013-09-01	NaN	NaN	Bangalore	India	12.05N	77.26E
19948	1816-03-01	27.426	1.793	Bangkok	Thailand	13.66N	99.91E

After Data Cleaning:

```
In [7]: list=df.City.unique()
```

```
In [8]: #Here we clear the Nan values in our data based on city wise analysis.
        for i in list:
            df.loc[df['City']==i]=df.loc[df['City']==i].interpolate()
```

```
In [9]: df[19945:19949]
```

Out[9]:

	dt	AverageTemperature	AverageTemperatureUncertainty	City	Country	Latitude	Longitude
19945	2013-07-01	25.008	0.416	Bangalore	India	12.05N	77.26E
19946	2013-08-01	25.236	0.520	Bangalore	India	12.05N	77.26E
19947	2013-09-01	25.236	0.520	Bangalore	India	12.05N	77.26E
19948	1816-03-01	27.426	1.793	Bangkok	Thailand	13.66N	99.91E

```
In [10]: df['AverageTemperatureUncertainty'].isnull().value_counts()
Out[10]: False    239177
         Name: AverageTemperatureUncertainty, dtype: int64
```

```
In [11]: df['AverageTemperature'].isnull().value_counts()
Out[11]: False    239177
         Name: AverageTemperature, dtype: int64
```

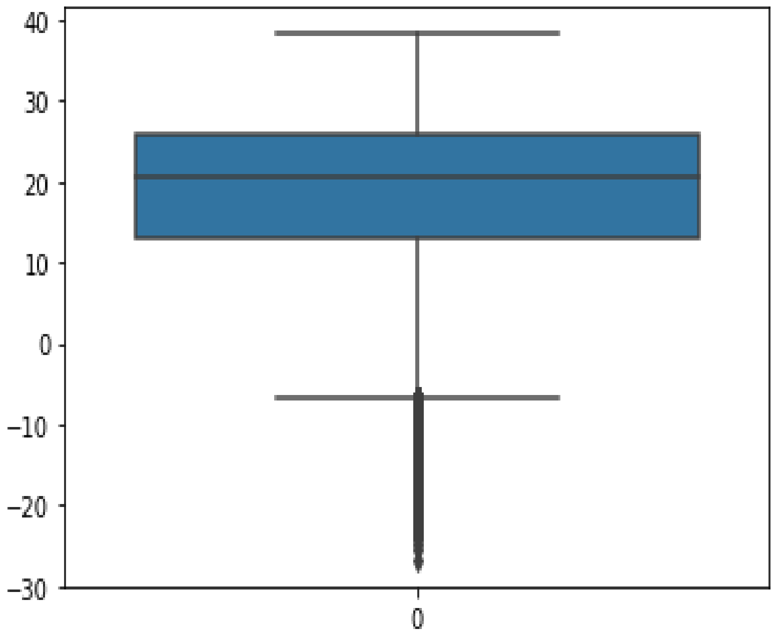
Code to remove the outliers:

```
In [23]: #Hence we need to remove the outliers and then use the data
         #for that we need to compute Inter-Quartile Range(IQR)
         print("For Average Temperature:")
         IQR=sc.stats.iqr(df['AverageTemperature'])
         print(IQR)
         Q1=np.quantile(df['AverageTemperature'],0.25)
         Q2=np.quantile(df['AverageTemperature'],0.5)
         Q3=np.quantile(df['AverageTemperature'],0.75)
         IQR1=Q3-Q1
         print(Q1, Q2, Q3)
         print(IQR1)
         print()
         print()
         #Same is done for Average Temperature Uncertainity
         print("For Average Temperature:")
         IQRU=sc.stats.iqr(df['AverageTemperatureUncertainty'])
         print(IQRU)
         QU1=np.quantile(df['AverageTemperatureUncertainty'],0.25)
         QU2=np.quantile(df['AverageTemperatureUncertainty'],0.5)
         QU3=np.quantile(df['AverageTemperatureUncertainty'],0.75)
         IQRU1=QU3-QU1
         print(QU1, QU2, QU3)
         print(IQRU1)
```

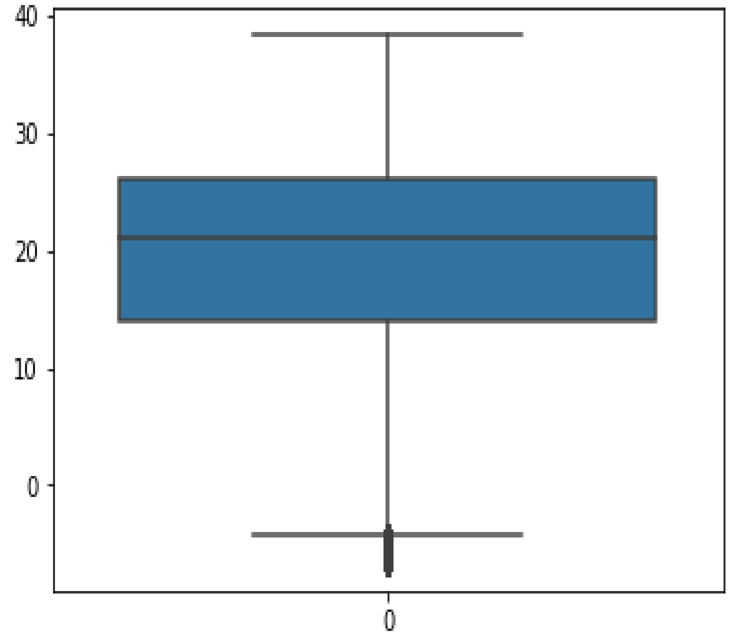
```
In [24]: #Eliminate all outliers based on formulae
         print("For Average Temperature:")
         O=Q3+1.5*IQR
         O1=Q1-1.5*IQR
         print(O,O1)
         print()
         print()
         print("For Average Temperature Uncertainity:")
         OU=QU3+1.5*IQRU
         OU1=QU1-1.5*IQRU
         print(OU,OU1)

For Average Temperature:
45.4735 -6.698499999999999


For Average Temperature Uncertainity:
3.0315 -1.2605
```

Average Temperature Outliers:

```
In [22]: #Box plot is to identify the outliers
         #hence we suspect the outliers in case of AverageTemperature
         sns.boxplot(data=df['AverageTemperature'])

Out[22]: <matplotlib.axes._subplots.AxesSubplot at 0x22b93438eb8>
```

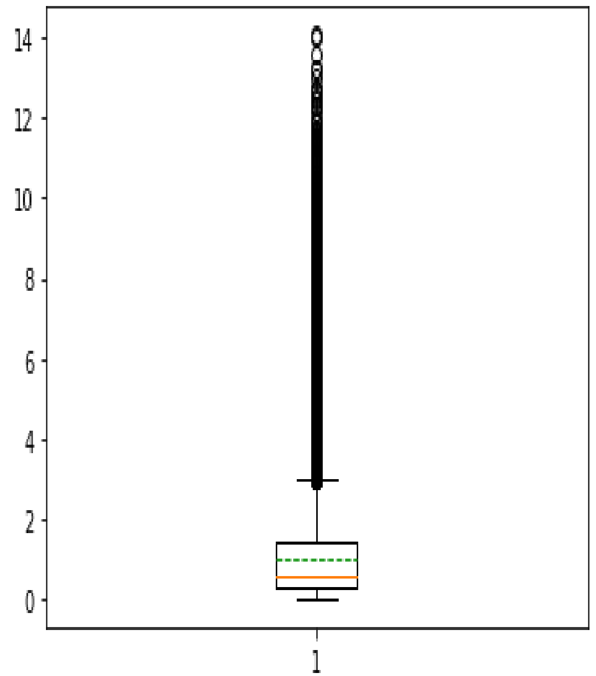



```
In [27]: #Now we plot the graph to check if outliers are removed(O
         sns.boxplot(data=df['AverageTemperature'])

Out[27]: <matplotlib.axes._subplots.AxesSubplot at 0x22b93112908>
```



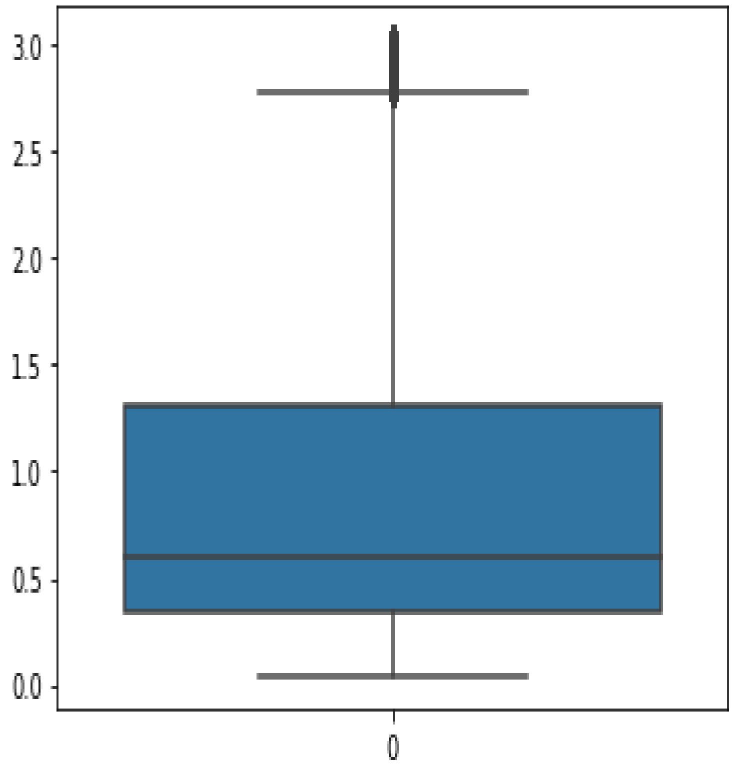
Average Temperature Uncertainty Outliers:

In [21]:
```
#Box plot is to identify the outliers
#hence we suspect the outliers in case of AverageTemperatureUncertainty
plt.boxplot(df['AverageTemperatureUncertainty'], meanline=True, showmeans=True, vert=True)
plt.show()
```



In [28]:
```
#Now we plot the graph to check if outliers are removed(
sns.boxplot(data=df['AverageTemperatureUncertainty'])
```

Out[28]: <matplotlib.axes._subplots.AxesSubplot at 0x22b93578cc0>



Probability Plots

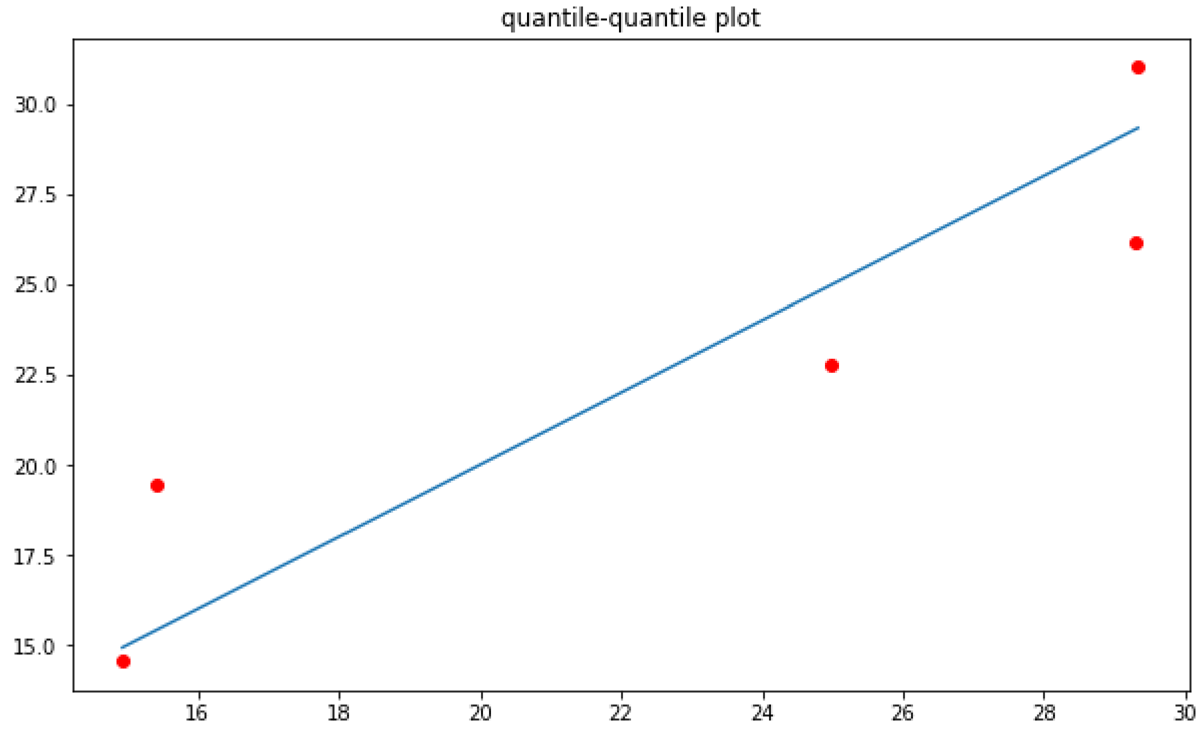
5/4/2020

- Probability Plots are used to detect if the population from which the samples have come are Normal or not.
- Most frequently used is the quantile-quantile plot
- Z-score vs theoretical values plot

Random sample of 5 points from population(not normal)

```
c=new[0].sample(5).tolist()
```

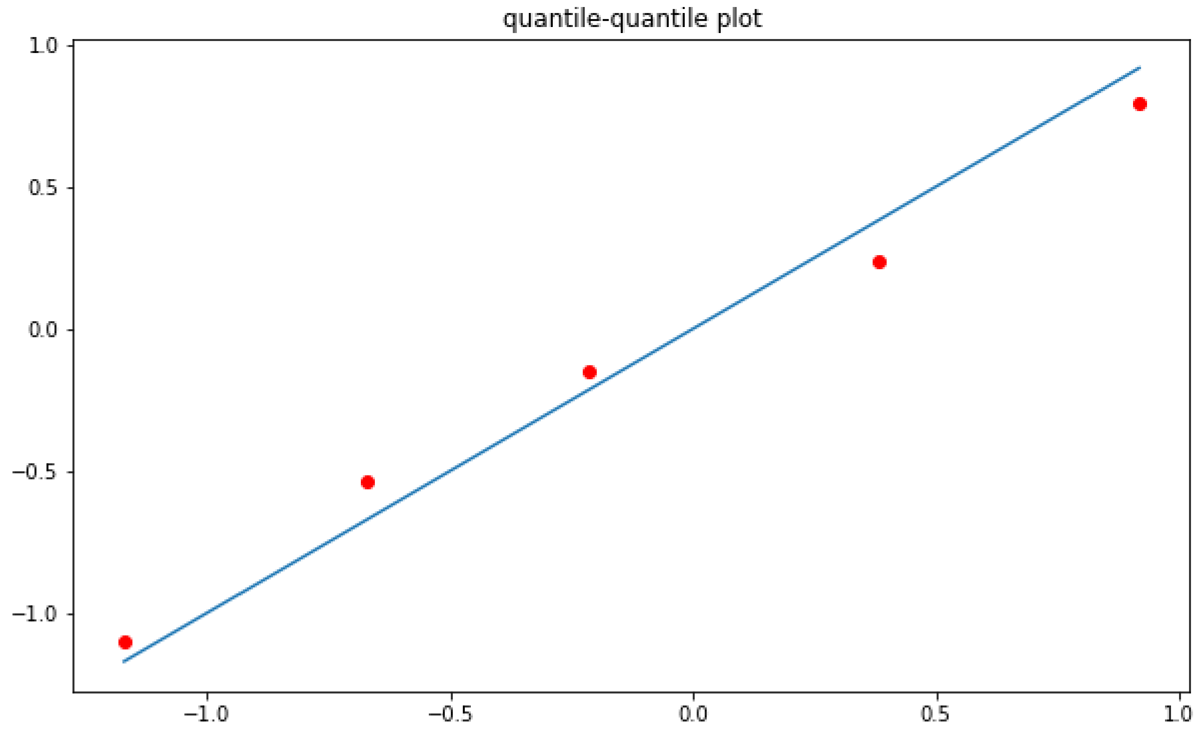
```
proplot(c)
```



5/4/2020

Sample of 5 points from same population after normalization


```
In [396]: afternorm=nnorm[0].sample(5).tolist()

In [397]: proplot(afternorm)
```



Normalizing the Numeric Columns

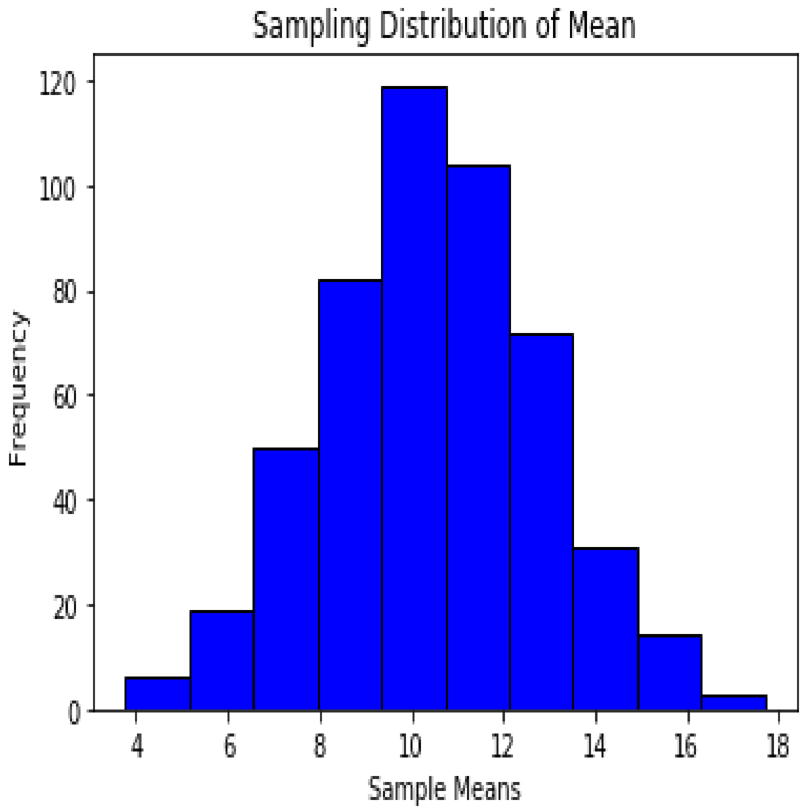
- Tools used:
  - Average Temperature and Uncertainty columns are to be normalized here.
  - mean() and std() attributes of np array have been used here
  - After normalizing mean =0 , stdev=1
- Why should we Normalize?
  - To ensure that all numeric columns are within similar RANGE and COMMON scale.
  - Suppose The great difference in the scale of the numbers could cause problems when you attempt to combine the values as features during modeling.

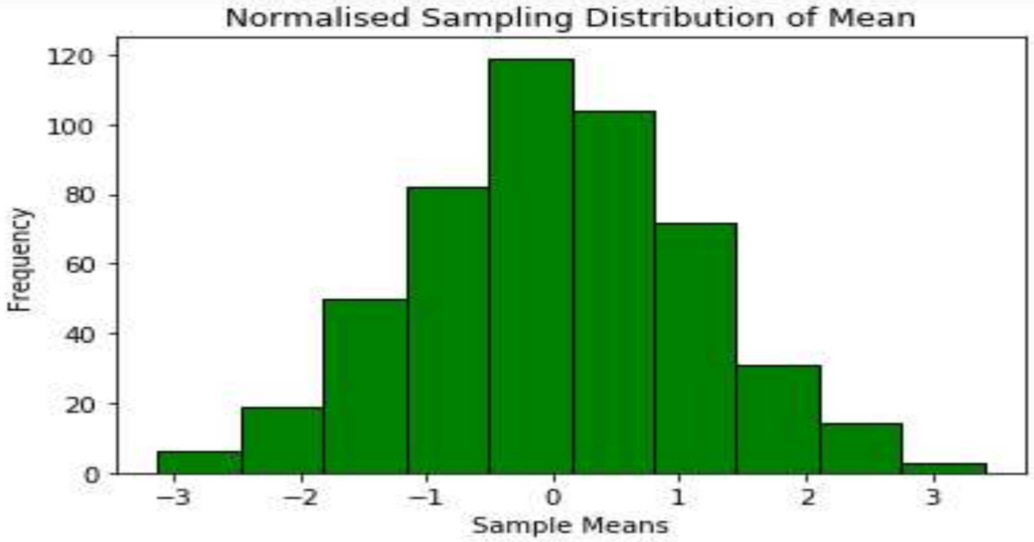
Normalization Code($\mu = 0 \; and \; \sigma = 1$)

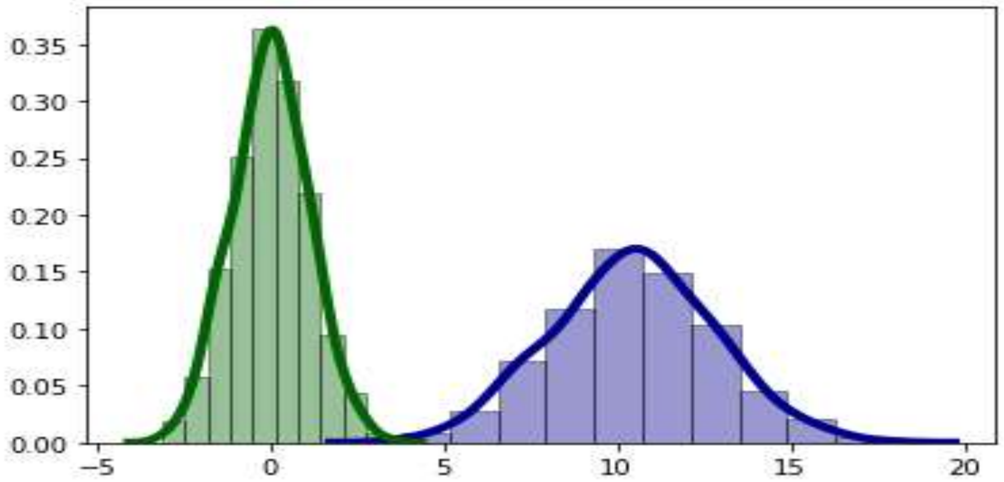
```python
array=[*Chic_df['AverageTemperature'],]
#Sampling distribution taking many sample means into considerations
mu=st.mean(array)
sigma=st.stdev(array)
def normalise(arr, n, number_of_samples):
    normal=[]
    x=[]
    sd=sigma/math.sqrt(n)
    for i in range(0,number_of_samples):
        s=sample(array,n)
        xbar=st.mean(s)
        x.append(xbar)
        l=(xbar-mu)/sd
        normal.append(l)
    plt.hist(x, color='blue', edgecolor='black')
    plt.xlabel("Sample Means")
    plt.ylabel("Frequency")
    plt.title("Sampling Distribution of Mean")
    plt.show()

    plt.hist(normal, color='green', edgecolor='black')
    plt.xlabel("Sample Means")
    plt.ylabel("Frequency")
    plt.title("Normalised Sampling Distribution of Mean")
    plt.show()

    sns.distplot(x, hist=True, kde=True, bins=10, color = 'darkblue',
                 hist_kws={'edgecolor':'black'}, kde_kws={'linewidth': 4})

    sns.distplot(normal, hist=True, kde=True, bins=10, color = 'darkgreen',
                 hist_kws={'edgecolor':'black'}, kde_kws={'linewidth': 4})
```

Sampling Distribution with n<30(sigma known):

In [40]:
```
#With n<30 (t distribution is seen)
normalise(array, 15, 500)
```

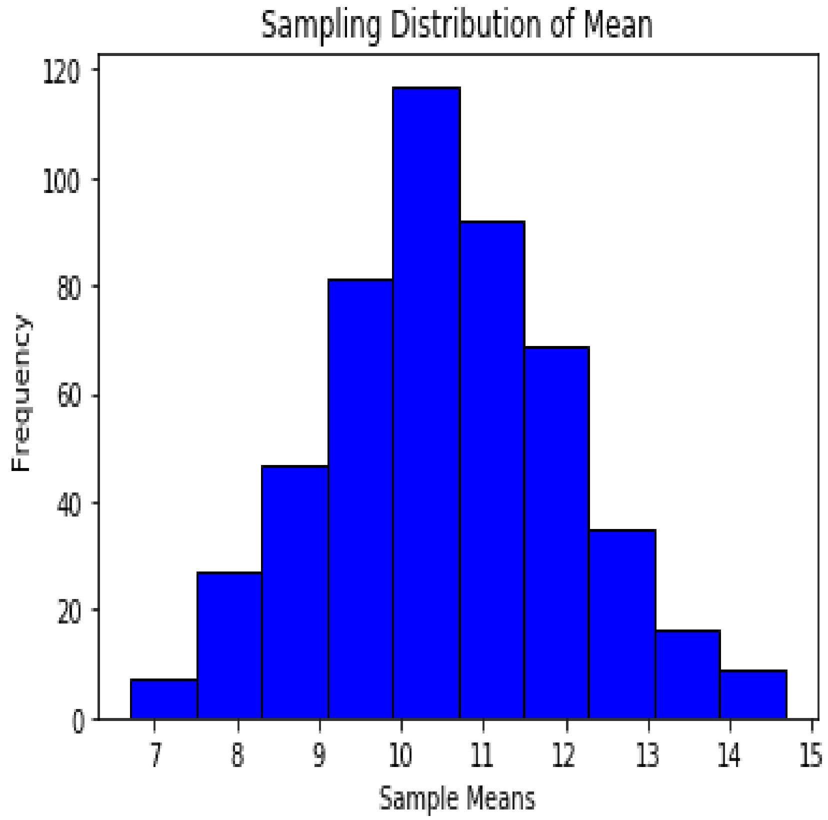


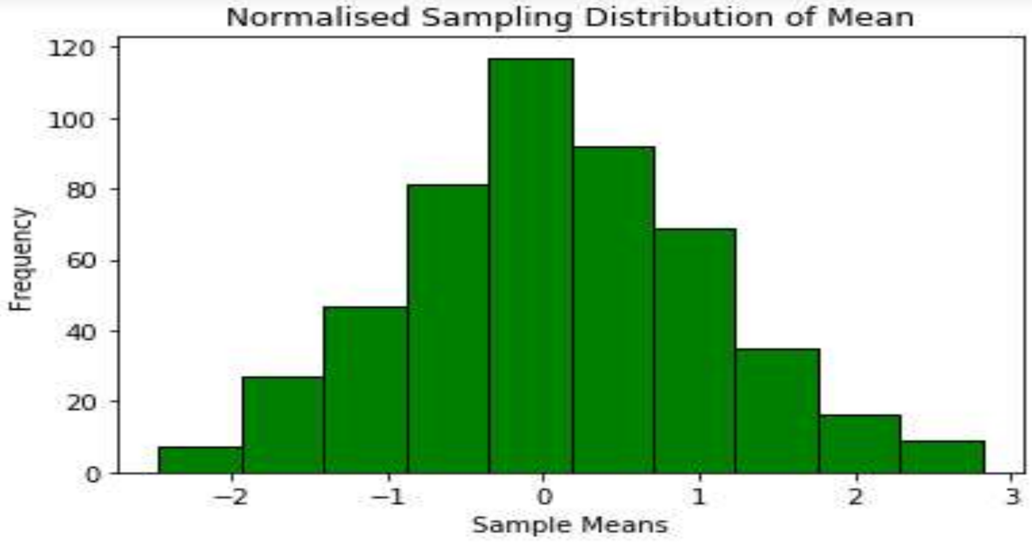


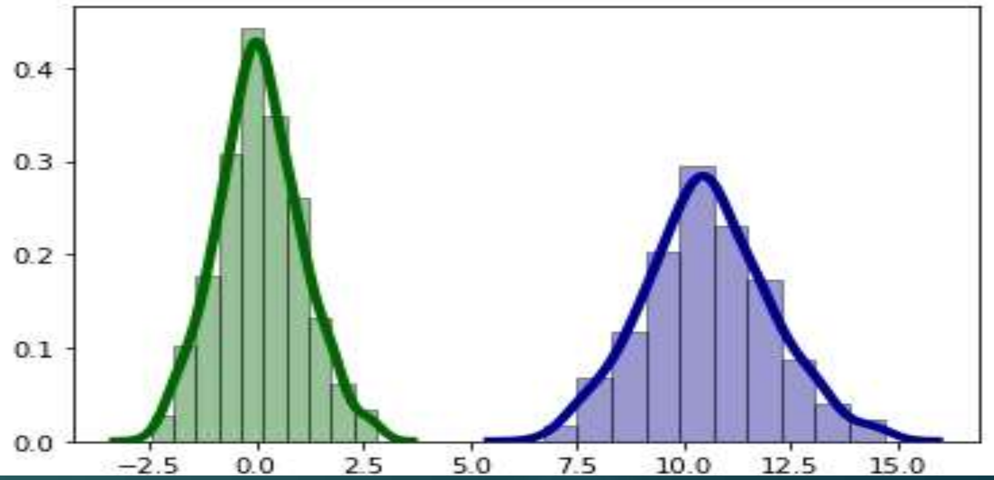


Sampling Distribution with n=30(Z-distribution)

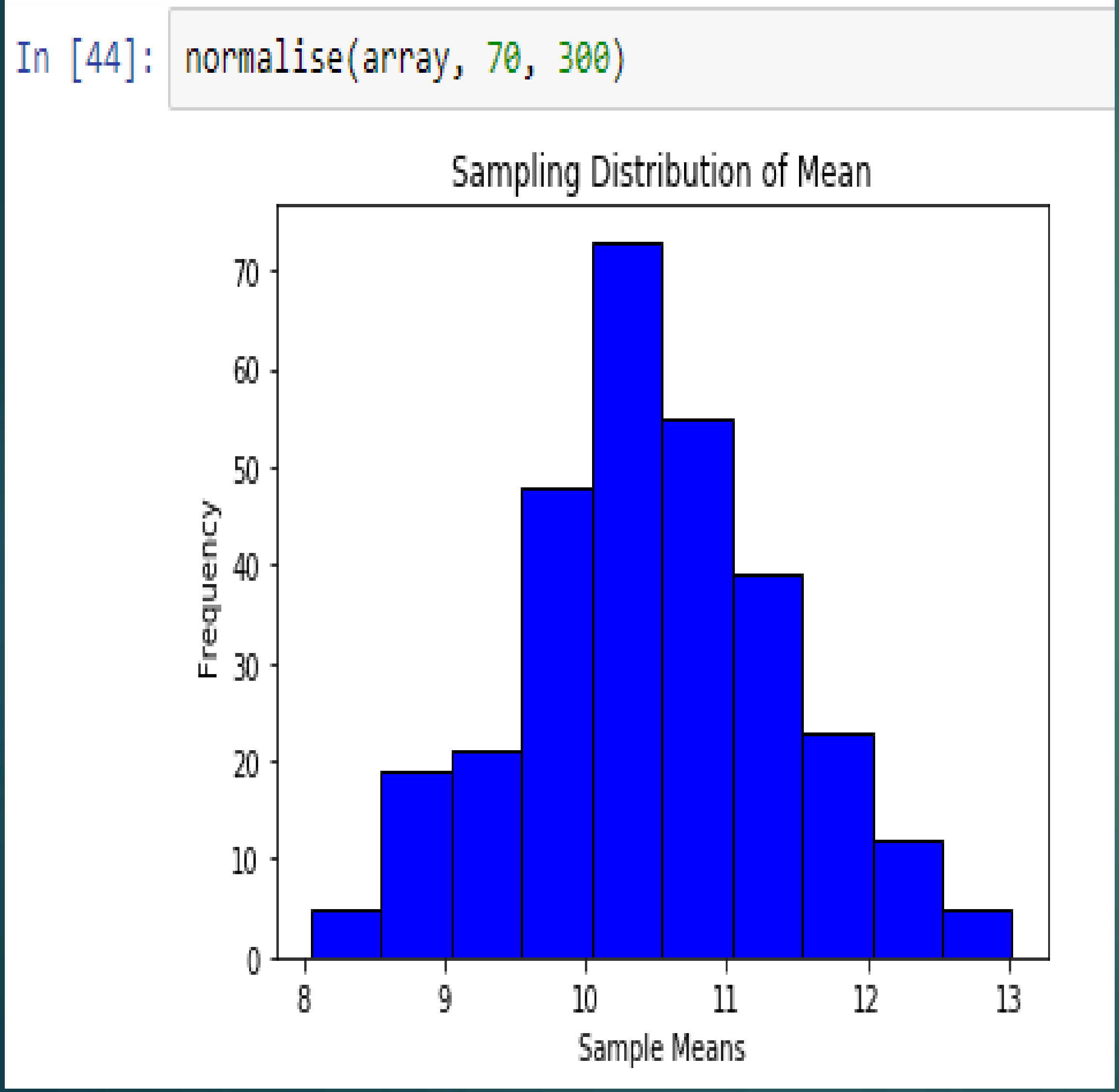
In [42]:
```
#with n=30 approximate z distribution can be obtained
normalise(array, 30, 500)
```

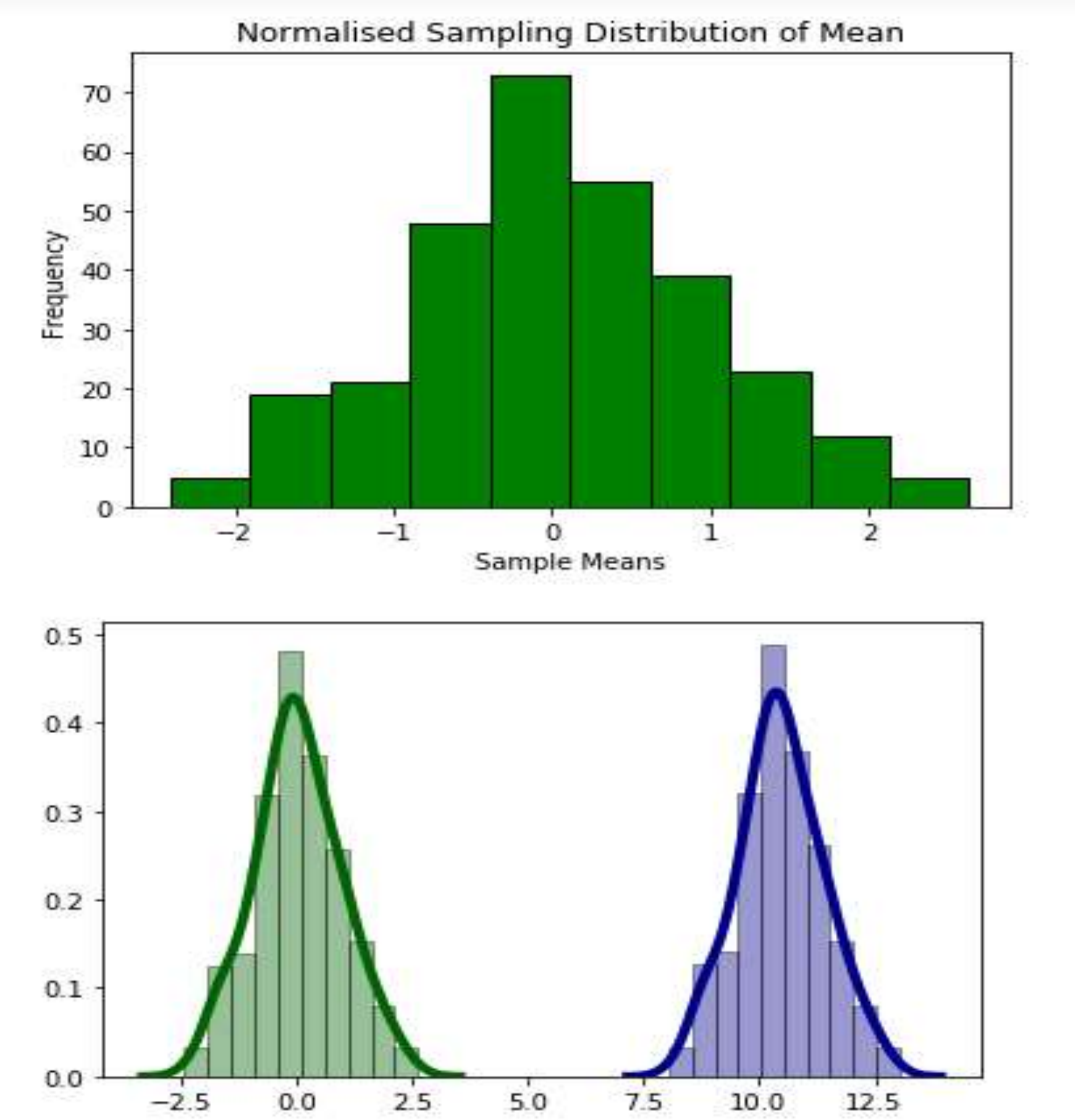






Sampling Distribution for n>30(Z-Distribution)





Normalization

- BEFORE NORMALIZATION

```
p=df['AverageTemperature']
p=pd.DataFrame(p)
p
```

	AverageTemperature
0	26.704
1	27.434
2	28.101
3	26.140
4	25.427
...	...
239172	18.979
239173	23.522
239174	25.251
239175	24.528
239176	24.528

239177 rows × 1 columns

```
x=df['AverageTemperatureUncertainty']
x=pd.DataFrame(x)
x
```

	AverageTemperatureUncertainty
0	1.435
1	1.362
2	1.612
3	1.387
4	1.200
...	...
239172	0.807
239173	0.647
239174	1.042
239175	0.840
239176	0.840

239177 rows × 1 columns

After Normalization


```
dfnorm=(df['AverageTemperature']-df['AverageTemperature'].mean())/df['AverageTemperature'].std()
dfnorm=pd.DataFrame(dfnorm)
len(dfnorm)
```

239177

```
dfnorm
```

	AverageTemperature
0	0.857283
1	0.930637
2	0.997659
3	0.800610
4	0.728965
...	...
239172	0.081045
239173	0.537544
239174	0.711280
239175	0.638630
239176	0.638630

239177 rows × 1 columns

```
dfnorm.describe()
```

	AverageTemperature
count	2.391770e+05
mean	8.185099e-16
std	1.000000e+00
min	-4.516195e+00
25%	-5.332133e-01
50%	2.317709e-01
75%	7.773987e-01
max	2.020787e+00

After Normalization

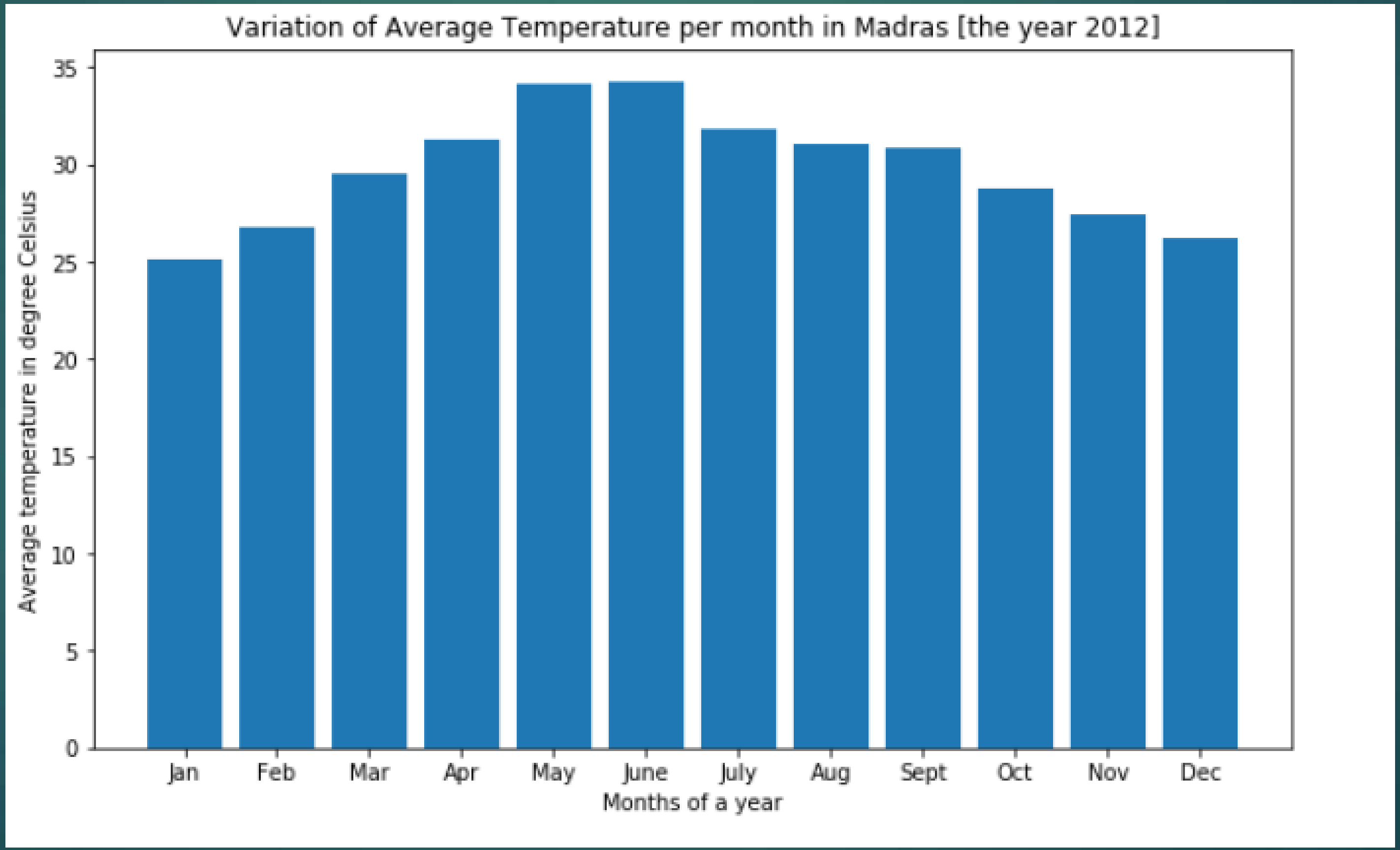
```
dfnorm2=(df['AverageTemperatureUncertainty']-df['AverageTemperatureUncertainty'].mean())/df['AverageTemperatureUncertainty'].s
dfnorm2=pd.DataFrame(dfnorm2)
dfnorm2
```

	AverageTemperatureUncertainty
0	0.433896
1	0.359566
2	0.614120
3	0.385021
4	0.194615
...	...
239172	-0.205543
239173	-0.368457
239174	0.033738
239175	-0.171942
239176	-0.171942

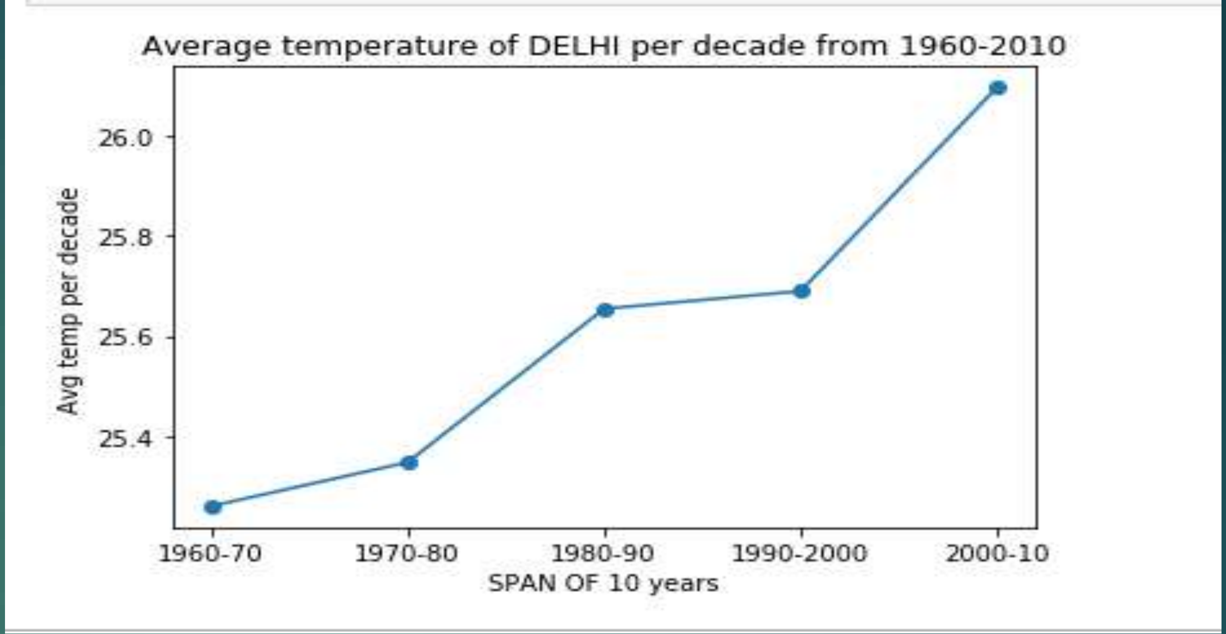
239177 rows × 1 columns

	AverageTemperatureUncertainty
count	2.391770e+05
mean	7.652735e-17
std	1.000000e+00
min	-9.865133e-01
25%	-6.718851e-01
50%	-3.847486e-01
75%	4.206590e-01
max	1.326543e+01

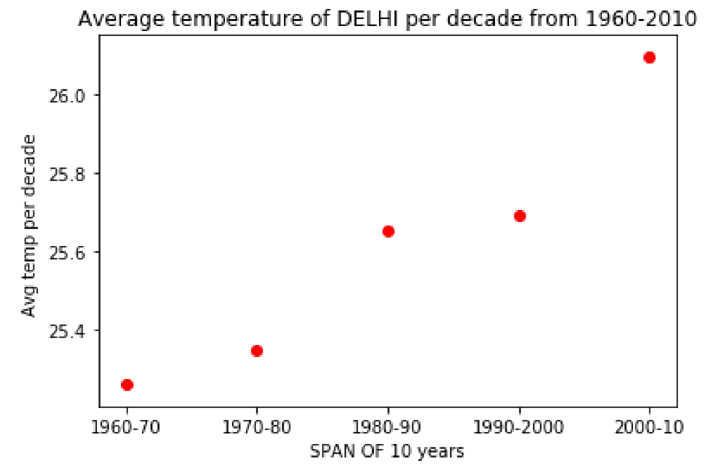
Some Meaningful insights

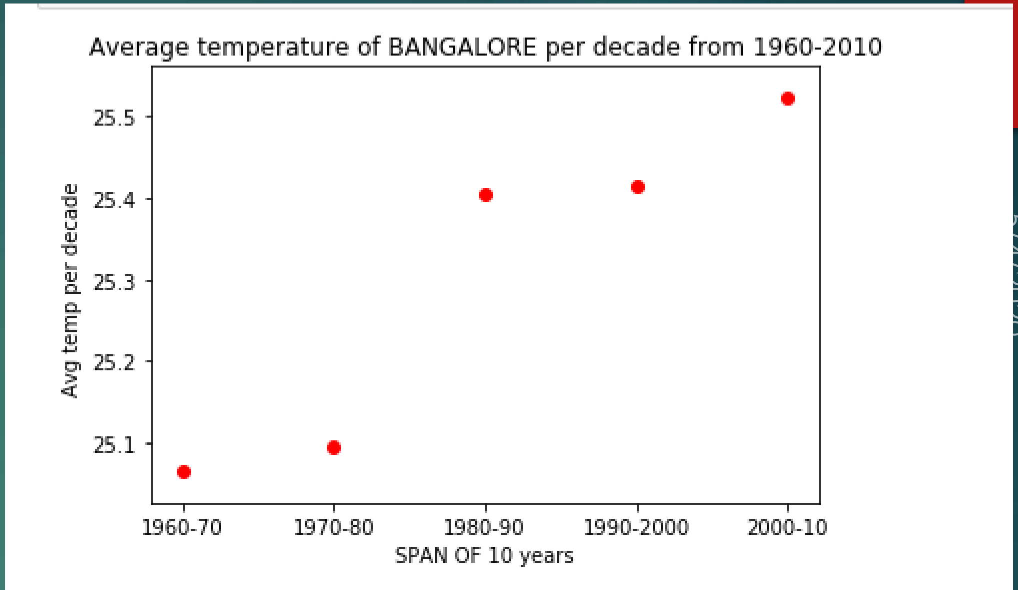


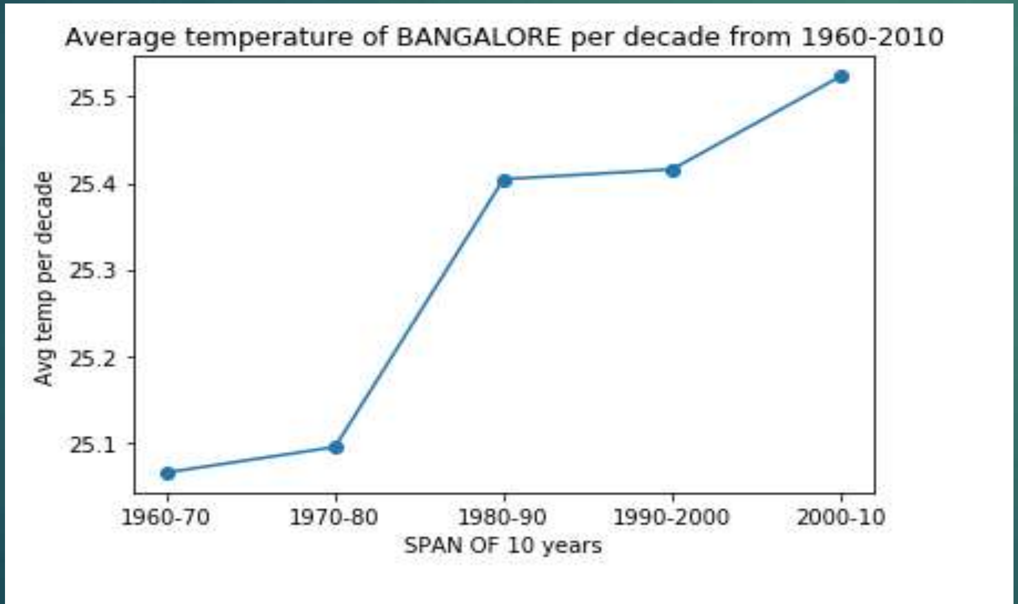
5/4/2020




```python
x=['1960-70','1970-80','1980-90','1990-2000','2000-10']
plt.scatter(x, array, color='r')    #single color for all points, points marker square
plt.xlabel('SPAN OF 10 years')
plt.ylabel('Avg temp per decade ')
plt.title('Average temperature of DELHI per decade from 1960-2010')
plt.show()
```

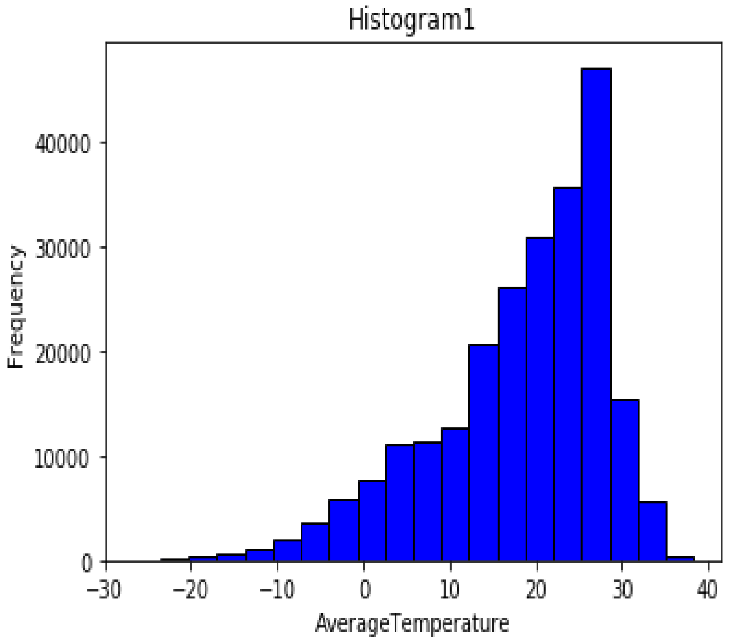



Average temperature of BANGALORE per decade from 1960-2010
Avg temp per decade
25.5
25.4
25.3
25.2
25.1
1960-70
1970-80
1980-90
1990-2000
2000-10
SPAN OF 10 years

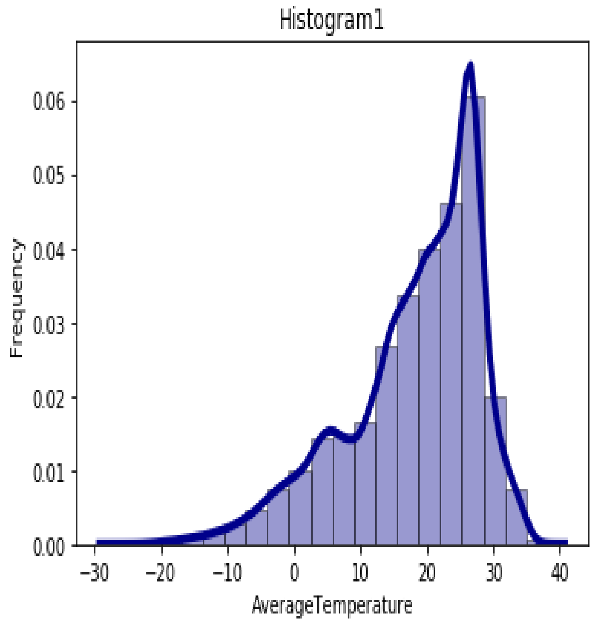

Average temperature of BANGALORE per decade from 1960-2010
Avg temp per decade
25.5
25.4
25.3
25.2
25.1
1960-70
1970-80
1980-90
1990-2000
2000-10
SPAN OF 10 years

Average Temperature Graphs:

In [12]:
```
# a negatively skewed histogram is obtained on Average Temperature data.
plt.hist(df['AverageTemperature'], color = 'blue', edgecolor = 'black',
         bins = 20)
plt.xlabel('AverageTemperature')
plt.ylabel('Frequency')
plt.title('Histogram1')
plt.show()
```



In [13]:
```
# a negatively skewed histogram is obtained on Average Temperature data with line fit.
sns.distplot(df['AverageTemperature'], hist=True, kde=True, bins=20, color = 'darkblue',
             hist_kws={'edgecolor':'black'}, kde_kws={'linewidth': 4})
plt.xlabel('AverageTemperature')
plt.ylabel('Frequency')
plt.title('Histogram1')
plt.show()
```



Inferences

In [14]:
```
#Population parameters and statistics using inbuilt libraries.
print("Mean:",st.mean(df['AverageTemperature']))
print("Median:",st.median(df['AverageTemperature']))
print("Mode:",st.mode(df['AverageTemperature']))
print("Standard deviation pop:",st.pstdev(df['AverageTemperature']))
print("S D:",st.stdev(df['AverageTemperature']))
```

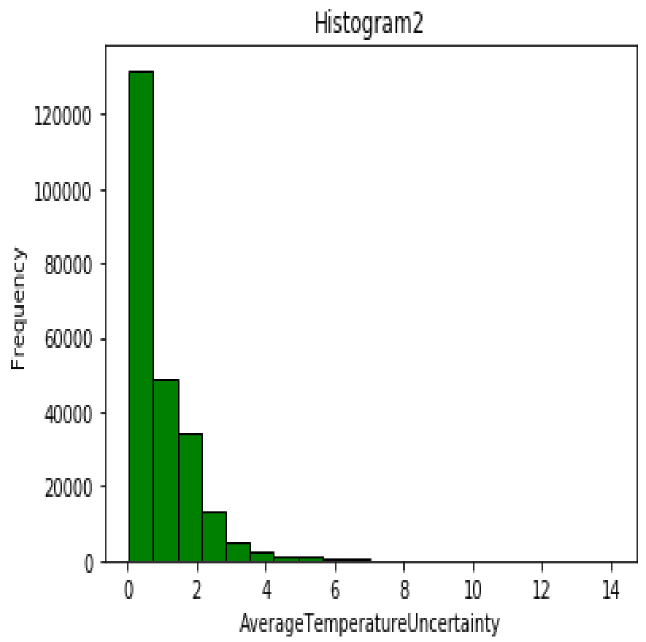
```
Mean: 18.172452967885707
Median: 20.479
Mode: 26.612
Standard deviation pop: 9.951819045569652
S D: 9.951839849932638
```

In [15]:
```
#Parameters using the histogram.
print("1.Based on Histogram we can analise that the parameters are matching with what is computed using inbuilt libr
print("2.As positively skewed mean < median < mode")
print("2.We can infer that there are few places with negative temperatures. We can find few such cities.")
print("3.But we can infer that majority of cities have positive temperatures from yhe graph.")
print("4.The graph is negatively skewed.")
```

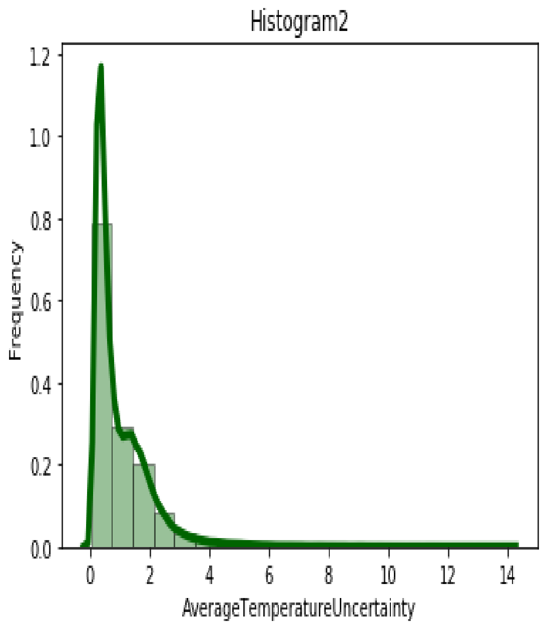
```
1.Based on Histogram we can analise that the parameters are matching with what is computed using inbuilt libraries.
2.As positively skewed mean < median < mode
2.We can infer that there are few places with negative temperatures. We can find few such cities.
3.But we can infer that majority of cities have positive temperatures from yhe graph.
4.The graph is negatively skewed.
```

Average Temperature Uncertainty Graphs:

```
In [17]: plt.hist(df['AverageTemperatureUncertainty'], color = 'green', edgecolor = 'black',
                  bins = 20)
         plt.xlabel('AverageTemperatureUncertainty')
         plt.ylabel('Frequency')
         plt.title('Histogram2')
         plt.show()
```



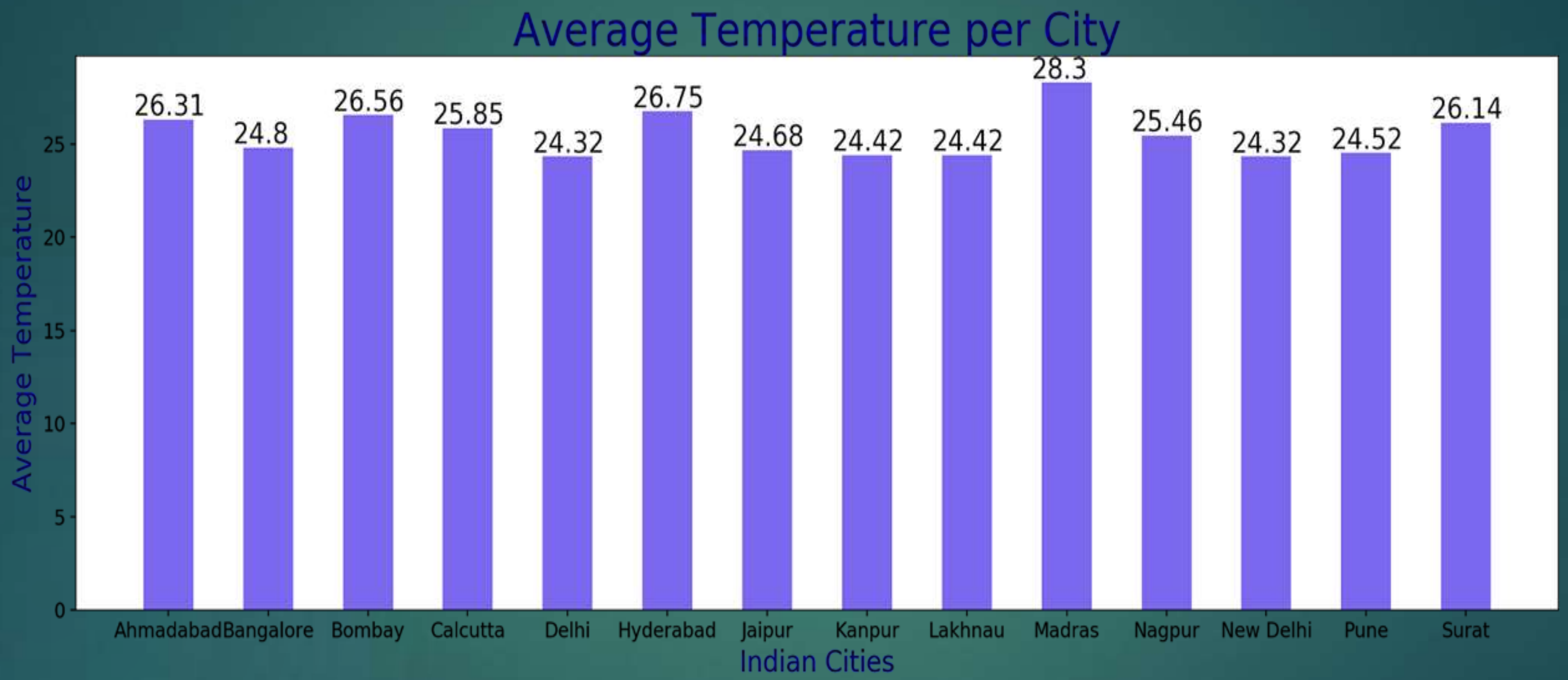

```
In [18]: #Positively skewed graph for the uncertainity
         sns.distplot(df['AverageTemperatureUncertainty'], hist=True, kde=True, bins=20, color = 'darkgreen',
                      hist_kws={'edgecolor':'black'}, kde_kws={'linewidth': 4})
         plt.xlabel('AverageTemperatureUncertainty')
         plt.ylabel('Frequency')
         plt.title('Histogram2')
         plt.show()
```



Inferences:

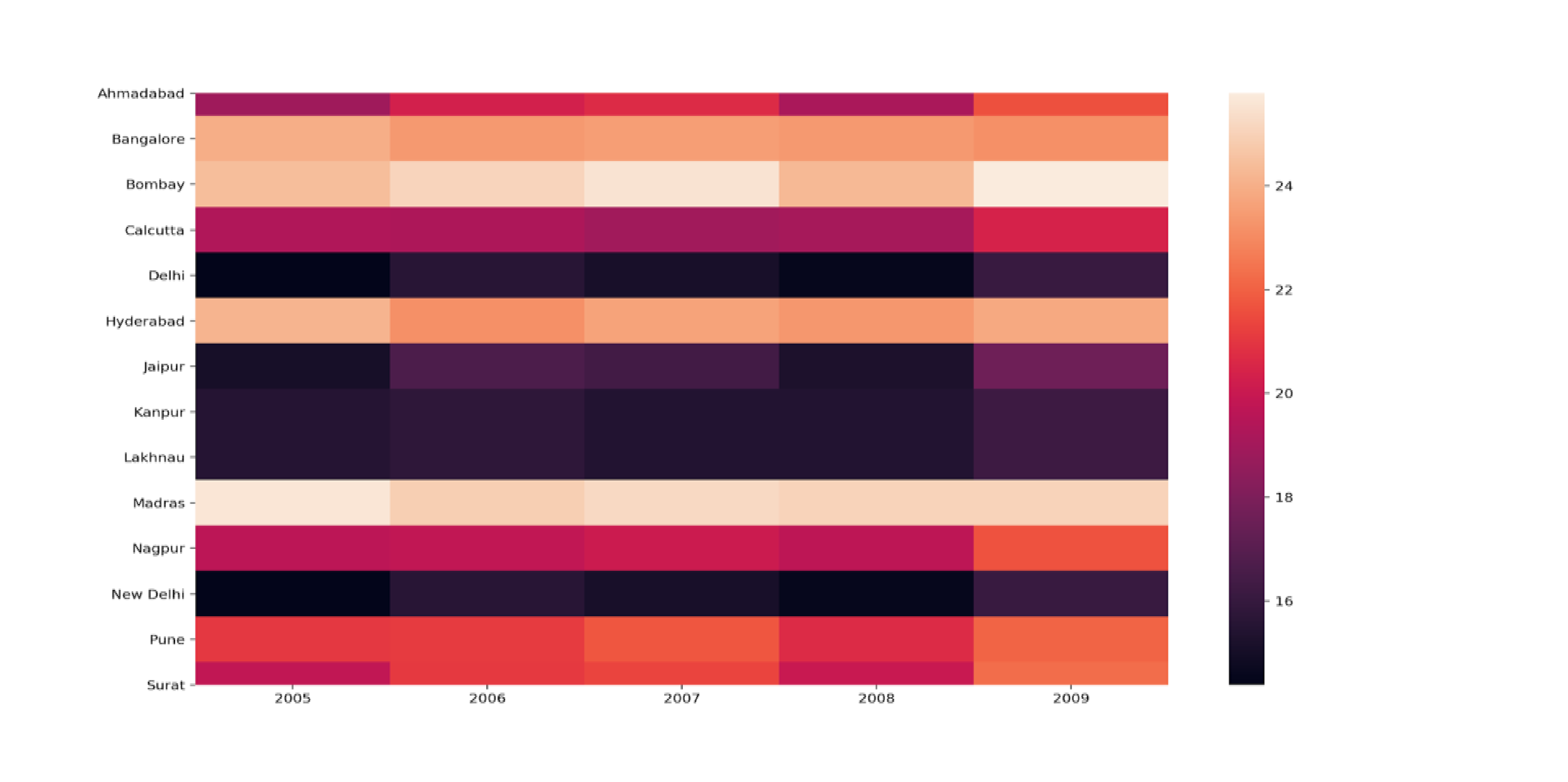
```
In [19]: #Population parameters and statistics using inbuilt libraries.
         print("Mean:",st.mean(df['AverageTemperatureUncertainty']))
         print("Median:",st.median(df['AverageTemperatureUncertainty']))
         print("Mode:",st.mode(df['AverageTemperatureUncertainty']))
         print("Standard deviation pop:",st.pstdev(df['AverageTemperatureUncertainty']))
         print("S D:",st.stdev(df['AverageTemperatureUncertainty']))

         Mean: 1.0088659925494508
         Median: 0.631
         Mode: 0.256
         Standard deviation pop: 0.9821093564750616
         S D: 0.9821114095830865

In [20]: #Inferences ,ade on the graph
         print("1.The graph is highly positively skewed.")
         print("2.As positively skewed mean > median > mode")
         print("3.The graph may contain few outliers.")
         print("4.This is unimodal with peak at the starting")

         1.The graph is highly positively skewed.
         2.As positively skewed mean > median > mode
         3.The graph may contain few outliers.
         4.This is unimodal with peak at the starting
```

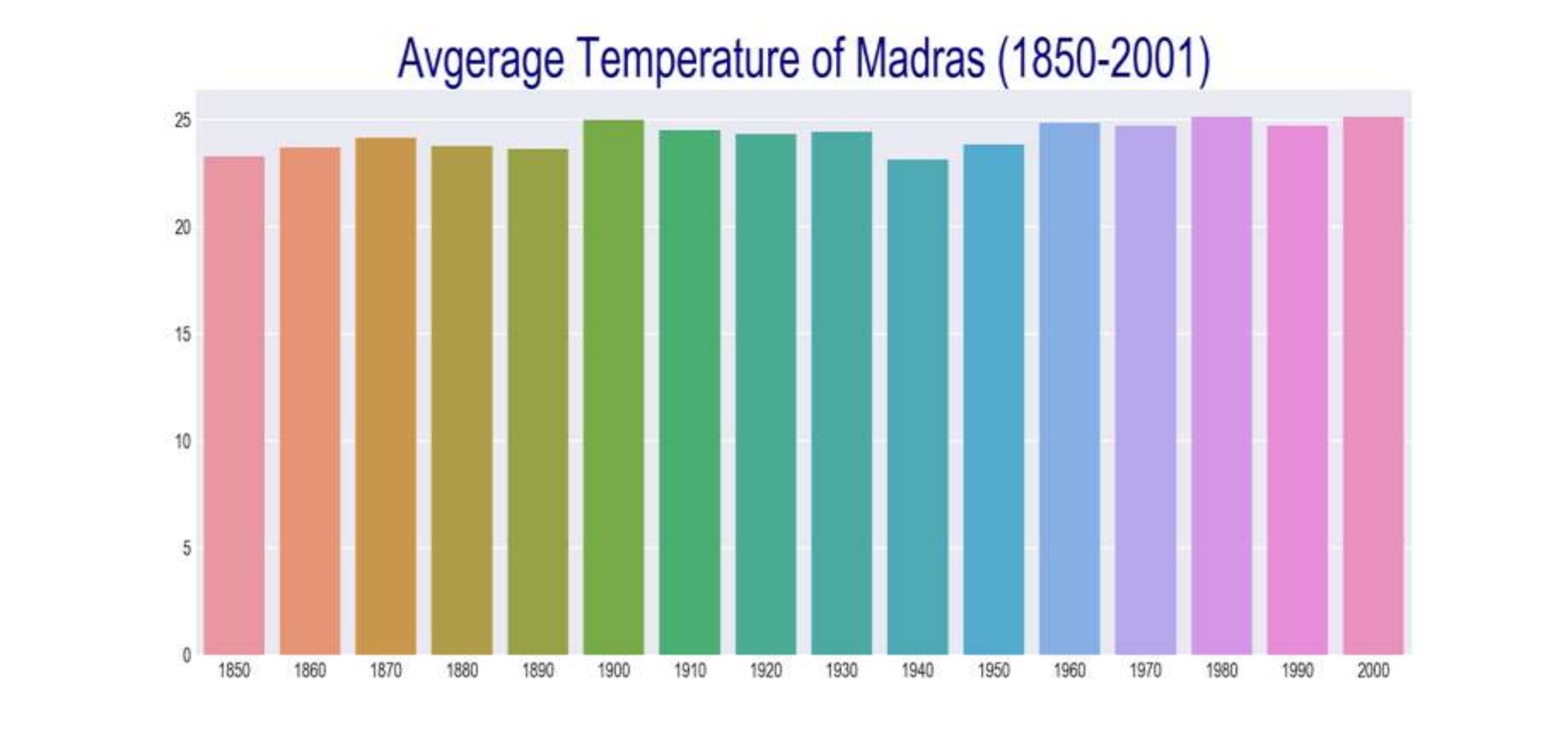
Average Temperature per City



Observe Madras has highest average temperature (1743-2013)

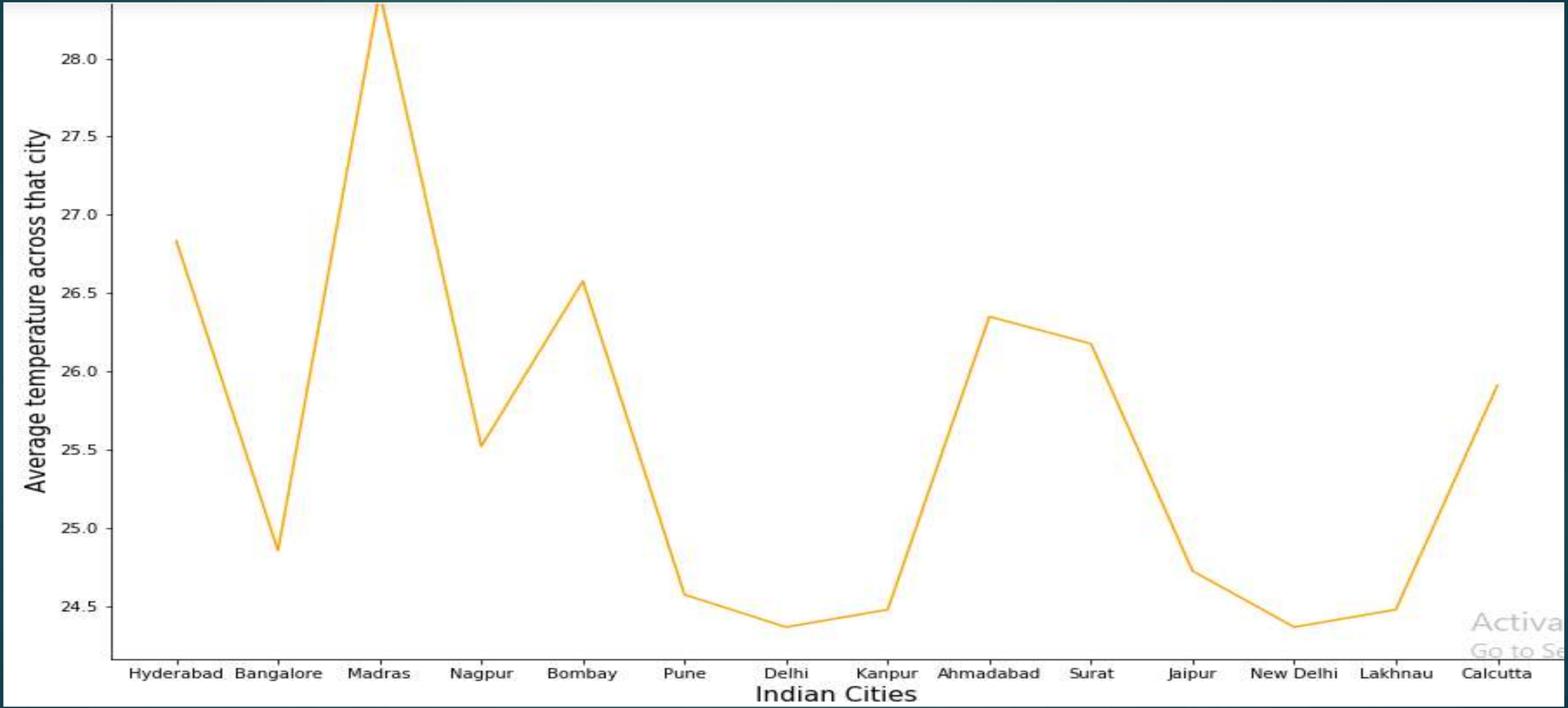
Heatmap of different regions in India





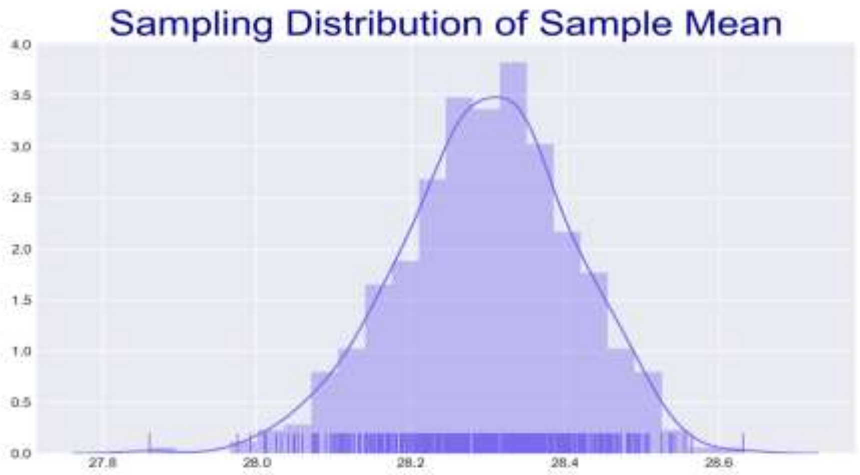
The trend of average temperature keeps increasing due to several factors.

Line Plot for Indian Cities



Central Limit Theorem

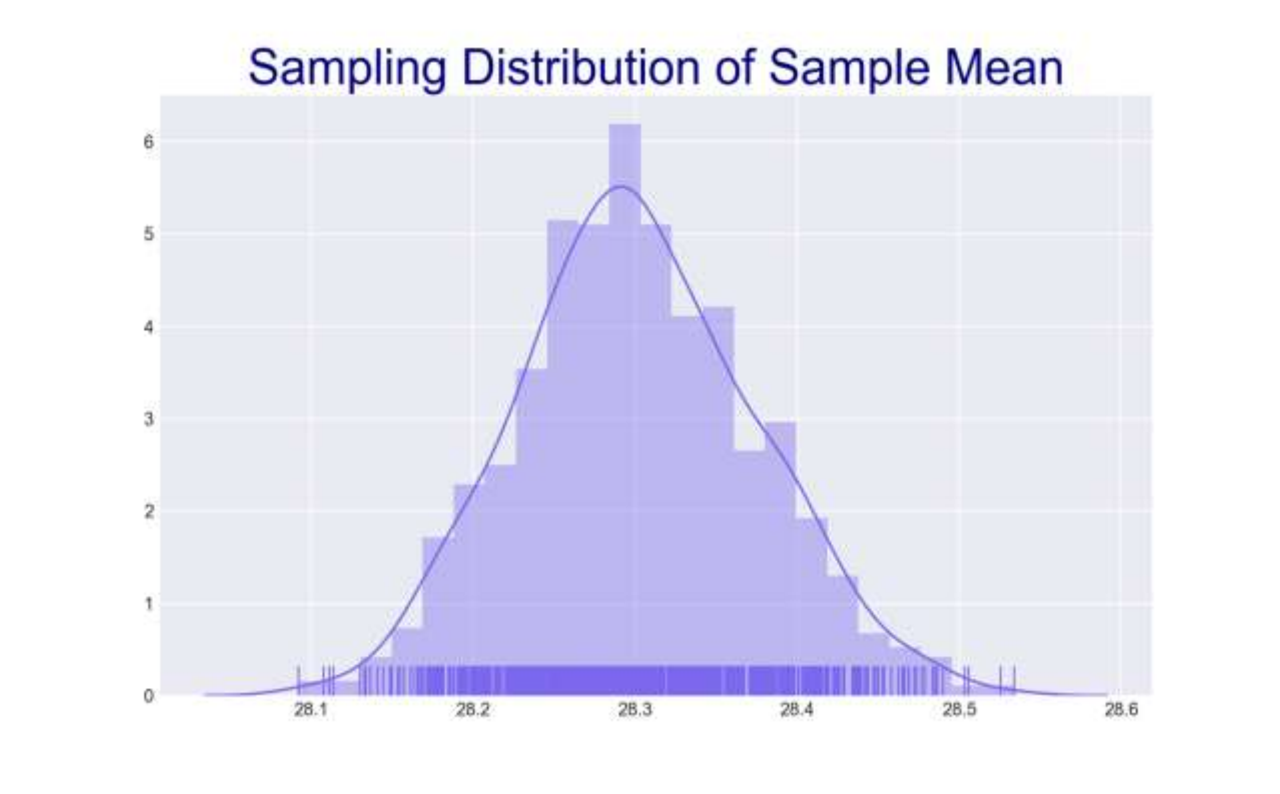
```
For,
         Number of Samples =  500
         Number of Times =  500


Madras Population Mean =  28.2975
Mean of sample mean =  28.2969

Hence, Mean of sample mean = Population Mean


Standard Error =  0.005


                                This proves Central Limit Theorem.
```

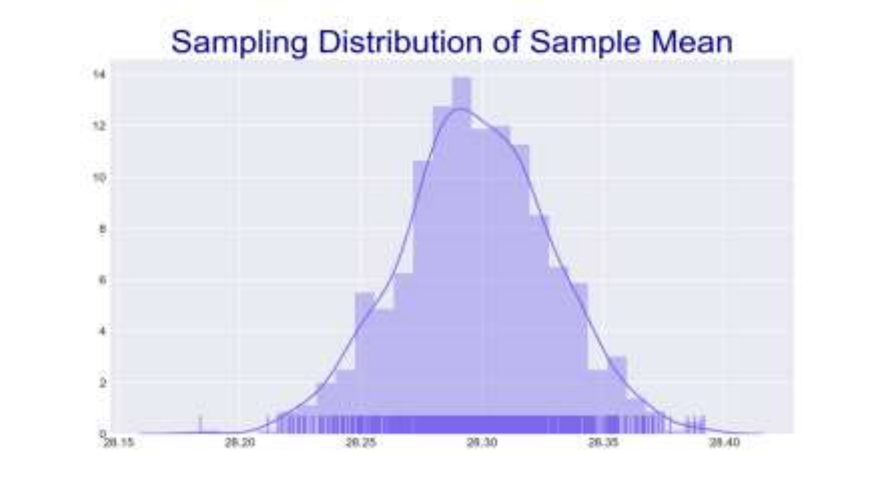


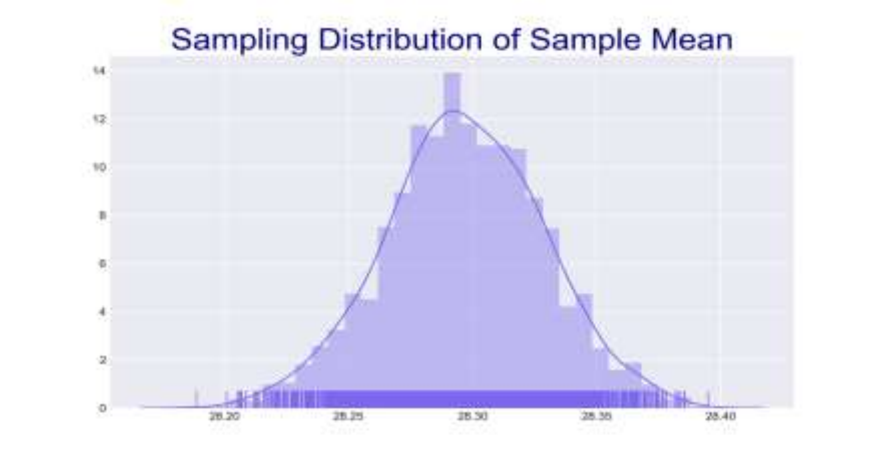

```python
seed(2)
n = df_madras.shape[0]  #Population Size
print("Population Size = ",n)

def sampling_distribution(s,t):
    global img
    madras_mean = []
    no_of_samples = s
    no_of_times = t

    for i in range(no_of_times):
        random_index = sample(range(df_madras.shape[0]), no_of_samples)
        madras_mean.append(df_madras.iloc[random_index]['AverageTemperature'].values.mean())

    plt.figure(figsize=(10,6))
    #plt.hist(madras_mean,bins=25,color = 'mediumslateblue')
    sns.distplot(madras_mean,color = 'mediumslateblue',rug=True)
    plt.title('Sampling Distribution of Sample Mean',fontsize=27,color='navy')
    plt.savefig(str(img)+'.png',dpi=200)
    img+=1;

    return mean(madras_mean),stdev(madras_mean)

def result(s,t):
    sample_mean,sample_std = sampling_distribution(s,t)
    print('For,\n\t Number of Samples = ',s,'\n\t Number of Times = ',t)

    print('\n\nMadras Population Mean = ',round(madras_population_mean,4))
    print('Mean of sample mean = ',round(sample_mean,4))
    print("\nHence, Mean of sample mean = Population Mean\n\n")

    print('Standard Error = ',round(sample_std / sqrt(s),4))

    print("\n\n\t\t\t\tThis proves Central Limit Theorem.\n\n\n")
```

```
Population Size =  2613
```

Different Sample Size



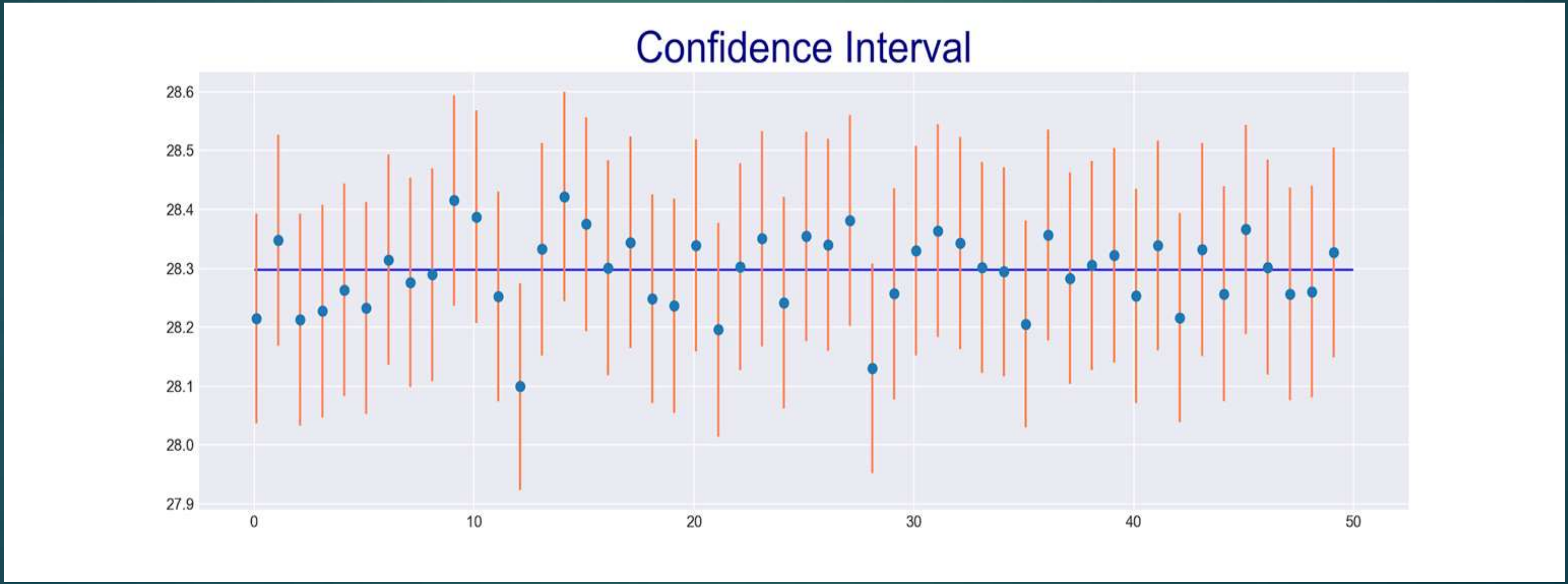


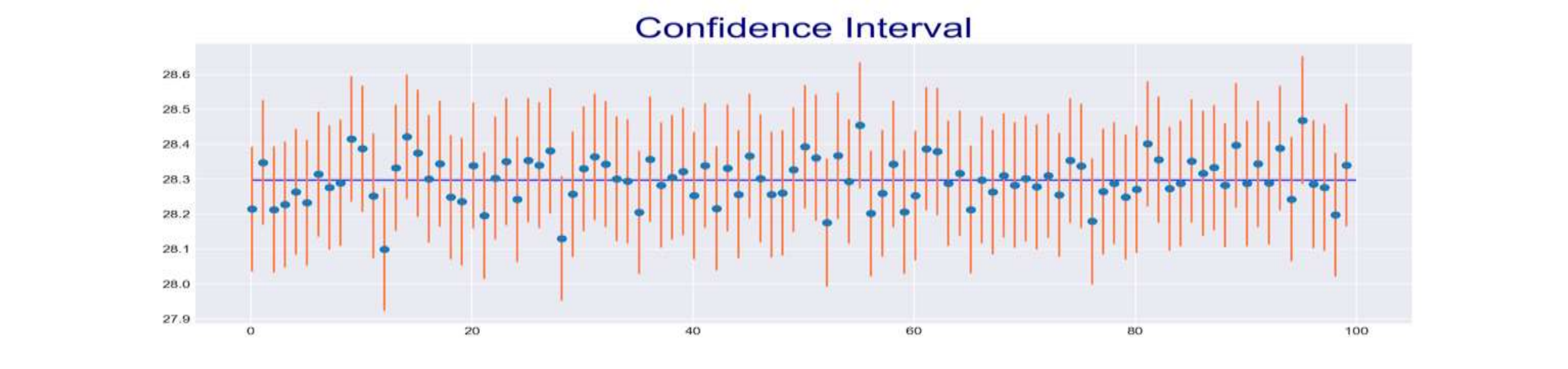


As the sample size increases, the sampling mean becomes equal to population mean.

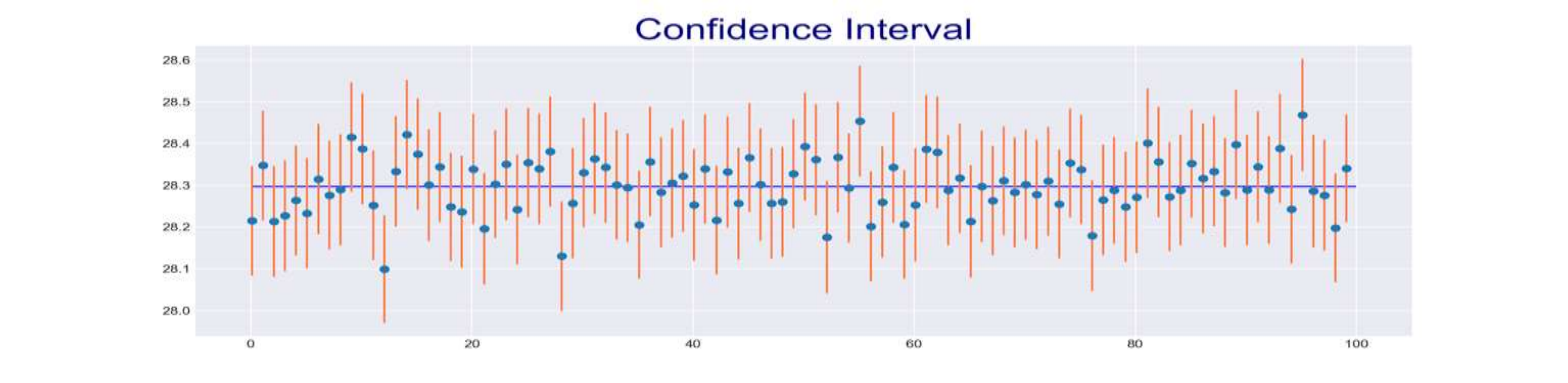
Confidence Interval

- Proportion of CIs covering Population mean for different sample size and different confidence coefficient



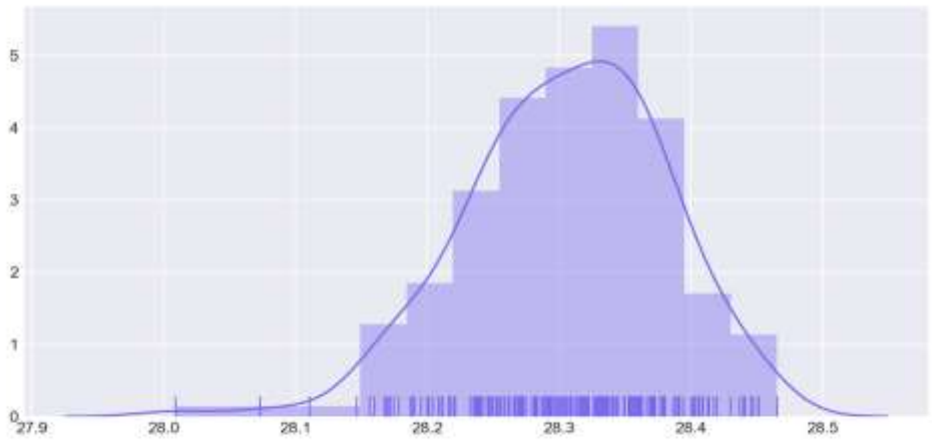


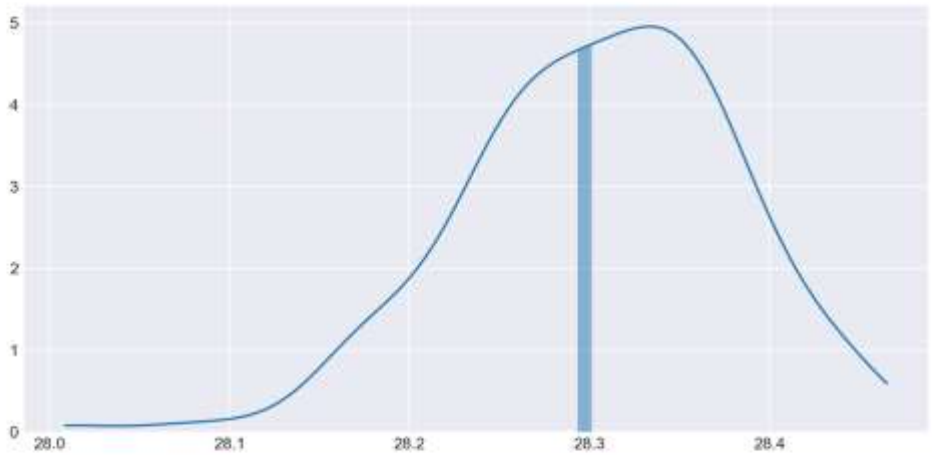
Confidence Interval
28.6
28.5
28.4
28.3
28.2
28.1
28.0
27.9
0
20
40
60
80
100



Confidence Interval
28.6
28.5
28.4
28.3
28.2
28.1
28.0
0
20
40
60
80
100

Estimation of Confidence Interval






```
#parameters....population, required_CI, sample_size, no_of_samples
def CI(pop, cc, s, t):
    global img
    plt.figure(figsize=(10,5))
    print("\nConfidence Interval :", cc, " and Sample Size :", s)
    pop_mean = round(np.mean(pop),4)
    print('\nActual Mean :',pop_mean)

    #calculation of same using CI
    samp_means = []       #mean of all the samples
    for i in range(t):
        samp_means.append(np.mean(sample(population, s)))
    #calculation of interval
    print('\nMean of Samples :', round(np.mean(samp_means),4))
    pop_stdev = round(np.std(samp_means) / math.sqrt(s),4)
    print('\nStandard Error = ',pop_stdev)
    z = st.norm.ppf(cc)
    print(z)
    print("\nConfidence Interval :", pop_mean, "+-", z*pop_stdev)
    #plt.hist(samp_means)
    ax = sns.distplot(samp_means,color = 'mediumslateblue',rug=True)

    plt.savefig(str(img)+'.png',dpi=200)
    img+=1;
    plt.show()


    plt.style.use("seaborn-darkgrid")

    plt.figure(figsize=(10,5))
    minimum = pop_mean - z*pop_stdev
    maximum = pop_mean + z*pop_stdev
    samp_means = np.asarray(samp_means)
    kde = st.gaussian_kde(samp_means)
    pos = np.linspace(samp_means.min(), samp_means.max(), 101)
    plt.plot(pos, kde(pos))
    #shade = np.linspace(pop_mean-z*pop_stdev,pop_mean+z*pop_stdev, 101)
    shade = np.linspace(minimum,maximum, 101)
    plt.fill_between(shade,kde(shade), alpha=0.5)
```

Population (Chicago):

5/4

```
In [34]: # we will use Chicago city as population for Hypothesis testing
         Chic_df=df.loc[df['City']=='Chicago']
         Chic_df
```

Out[34]:

	dt	AverageTemperature	AverageTemperatureUncertainty	City	Country	Latitude	Longitude
51674	1743-11-01	5.436000	2.205000	Chicago	United States	42.59N	87.27W
51675	1743-12-01	6.102000	2.235400	Chicago	United States	42.59N	87.27W
51676	1744-01-01	6.768000	2.265800	Chicago	United States	42.59N	87.27W
51677	1744-02-01	7.434000	2.296200	Chicago	United States	42.59N	87.27W
51678	1744-03-01	8.100000	2.326600	Chicago	United States	42.59N	87.27W

Population (New York):

In [131]:
```
#population 2 for hypothesis testing
NY_df=df.loc[df['City']=='New York']
arr=[*NY_df['AverageTemperature'],]
mu2=st.mean(arr)
sigma2=st.stdev(arr)
mu1=mu
sigma1=sigma
print(mu2,sigma2)
```

```
9.904255802823641 8.99116900108305
```

In [132]:
```
#Hypothesis testing for difference of mean between two population (two tail test)
#One population is Chicago
#Second population is Ney York
#Both population belong to same country but are independent of each other.
array1=array#Chicago population
array2=arr#New York population
```

Upper Tail test (n>30):


```
In [48]: #Generalise hypothesis testing for upper tailtest:(n>30)
         def upper_tail_HT(m, xbar, n, alpha):
             #Z-score of xbar
             sd=sigma/math.sqrt(n)
             z=(xbar-m)/sd
             #Area towards right of it
             area=1-norm.cdf(z)
             if(area > alpha):
                 print("Fail to reject the Null hypothesis or Reject the alternate hypothesis.")
             else:
                 print("Reject the NUll hypothesis or Fail to reject Alternate Hypothesis")
```

Upper Tail test (n>30):

```
In [82]: #we define aur query as:
         #Only if the mean is greater than 9 we consider it else we reject it.
         print("Null Hypothesis(Ho):m0 <= 9")
         print("Alternate Hypothesis(Ha):m0 > 9")
         n=70
         #we check for 95% and 90% condfidence intervals.
         s=sample(array, n)
         xbar=st.mean(s)
         print(xbar)
         upper_tail_HT(9, xbar, n, 0.05)
         upper_tail_HT(9 ,xbar, n, 0.1)
```

```
Null Hypothesis(Ho):m0 <= 9
Alternate Hypothesis(Ha):m0 > 9
10.603347368421053
Fail to reject the Null hypothesis or Reject the alternate hypothesis.
Reject the NUll hypothesis or Fail to reject Alternate Hypothesis
```

```
In [89]: #For n=50
         s=sample(array, 50)
         xbar=st.mean(s)
         print(xbar)
         upper_tail_HT(9, xbar , 50, 0.05)
         upper_tail_HT(9, xbar , 50, 0.1)
         #For n=100
         s=sample(array, 100)
         xbar=st.mean(s)
         print(xbar)
         upper_tail_HT(9, xbar , 100, 0.05)
         upper_tail_HT(9, xbar , 100, 0.1)
```

```
9.889598947368421
Fail to reject the Null hypothesis or Reject the alternate hypothesis.
Fail to reject the Null hypothesis or Reject the alternate hypothesis.
10.170432105263158
Fail to reject the Null hypothesis or Reject the alternate hypothesis.
Reject the NUll hypothesis or Fail to reject Alternate Hypothesis
```

Lower Tail Test with n<30 (σ $known$)

```
In [94]: #Hypothesis Testing for lower tail test with n<30 (as sigma is known we use z table instead of t table)
         def lower_tail_HT(m, xbar, n, alpha):
             #z-score of xbar
             sd=sigma/math.sqrt(n)
             z=(xbar-m)/sd
             #Area towards right of it
             area=norm.cdf(z)
             if(area > alpha):
                 print("Fail to reject the Null hypothesis or Reject the alternate hypothesis.")
             else:
                 print("Reject the NUll hypothesis or Fail to reject Alternate Hypothesis")
```

```
In [99]: #we define aur query as:
         #Only if the mean is less than 12 we consider it else we reject it.
         print("Null Hypothesis(Ho):m0 >= 12")
         print("Alternate Hypothesis(Ha):m0 < 12")
         n=20
         #we check for 95% and 90% condfidence intervals.
         s=sample(array, n)
         xbar=st.mean(s)
         print(xbar)
         lower_tail_HT(12, xbar, n, 0.05)
         lower_tail_HT(12, xbar, n, 0.1)
```

```
Null Hypothesis(Ho):m0 >= 12
Alternate Hypothesis(Ha):m0 < 12
9.544775
Fail to reject the Null hypothesis or Reject the alternate hypothesis.
Reject the NUll hypothesis or Fail to reject Alternate Hypothesis
```

Lower Tail Test with n<30(σ $known$)

In [116]:
```
#if n=25
s=sample(array, 25)
xbar=st.mean(s)
print(xbar)
lower_tail_HT(12, xbar, 25, 0.05)
lower_tail_HT(12, xbar, 25, 0.1)
#If n=15
s=sample(array, 15)
xbar=st.mean(s)
print(xbar)
lower_tail_HT(12, xbar, 15, 0.05)
lower_tail_HT(12, xbar, 15, 0.1)
```

```
10.05548
Fail to reject the Null hypothesis or Reject the alternate hypothesis.
Fail to reject the Null hypothesis or Reject the alternate hypothesis.
9.180133333333334
Fail to reject the Null hypothesis or Reject the alternate hypothesis.
Reject the NUll hypothesis or Fail to reject Alternate Hypothesis
```

Two Tail Test for difference of mean:


```
In [134]: #Hypothesis for difference of mean(Two tail test)(n>30)
          def two_tail_HT(diff, m1, m2, n1, n2, alpha):
               #z-score of xbar
              sd=math.sqrt((math.pow(sigma1,2)/n1)+(math.pow(sigma2,2)/n2))
              z=((m1-m2)-diff)/sd
              #Area towards right of it
              area=norm.cdf(z)
              if(area > alpha/2 and area < (1-(alpha/2))):
                  print("Fail to reject the Null hypothesis or Reject the alternate hypothesis.")
              else:
                  print("Reject the NUll hypothesis or Fail to reject Alternate Hypothesis")
```

Two Tail Test for difference of mean:

```
In [153]: #Test the two tail test with following hypothesis
          #we define aur query as:
          #Only if the mean difference is not equal to 1 we consider it else we reject it.
          print("Null Hypothesis(Ho):m1-m2 == 4")
          print("Alternate Hypothesis(Ha):m1-m2 != 4")#(Two tail test)
          n1 = 30
          n2 = 30
          #we check for 95% and 90% condfidence intervals.
          s1=sample(array1, n1)
          m1=st.mean(s1)
          s2=sample(array2, n2)
          m2=st.mean(s2)
          print(m1, m2)
          two_tail_HT(4, m1, m2, n1, n2, 0.05)
          two_tail_HT(4, m1, m2, n1, n2, 0.1)

          Null Hypothesis(Ho):m1-m2 == 4
          Alternate Hypothesis(Ha):m1-m2 != 4
          10.634666666666666 10.554083333333333
          Fail to reject the Null hypothesis or Reject the alternate hypothesis.
          Reject the NUll hypothesis or Fail to reject Alternate Hypothesis
```

```
In [170]: #For difference to be as 2
          print("Null Hypothesis(Ho):m1-m2 == 2")
          print("Alternate Hypothesis(Ha):m1-m2 != 2")#(Two tail test)
          #For variable n values
          n1 = 40
          n2 = 50
          #we check for 95% and 90% condfidence intervals.
          s1=sample(array1, n1)
          m1=st.mean(s1)
          s2=sample(array2, n2)
          m2=st.mean(s2)
          print(m1, m2)
          two_tail_HT(2, m1, m2, n1, n2, 0.05)
          two_tail_HT(2, m1, m2, n1, n2, 0.1)

          Null Hypothesis(Ho):m1-m2 == 2
          Alternate Hypothesis(Ha):m1-m2 != 2
          10.4588 11.68093
          Fail to reject the Null hypothesis or Reject the alternate hypothesis.
          Reject the NUll hypothesis or Fail to reject Alternate Hypothesis
```

Correlation Co-efficient

- The Pearson's Coefficient method helps us to find how strongly are 2 numerical parameters RELATED LINEARLY
- If Pearson's Co-efficient is
  - Close to -1 or 1 , then the parameters are strongly related in a (negative fashion) or (positive fashion) respectively
  - If Close to 0, then we can say that the both the parameters are weakly related linearly.

Pearson's Coefficient

```
In [281]: df.corr(method='pearson')
```

Out[281]:

	AverageTemperature	AverageTemperatureUncertainty
AverageTemperature	1.000000	-0.196338
AverageTemperatureUncertainty	-0.196338	1.000000

Pearson's Co-efficient for our dataset

- It can be seen that , correlation coefficient is close to zero and far from the numbers -1 or 1
- Hence we can conclude that there is no strong linear relation between Average Temperature and Average Temperature Uncertainty

Scatter Plot Representation

- Population of interest:

  numerical column1 :Average temperature per month of Bangalore from 1960-2010

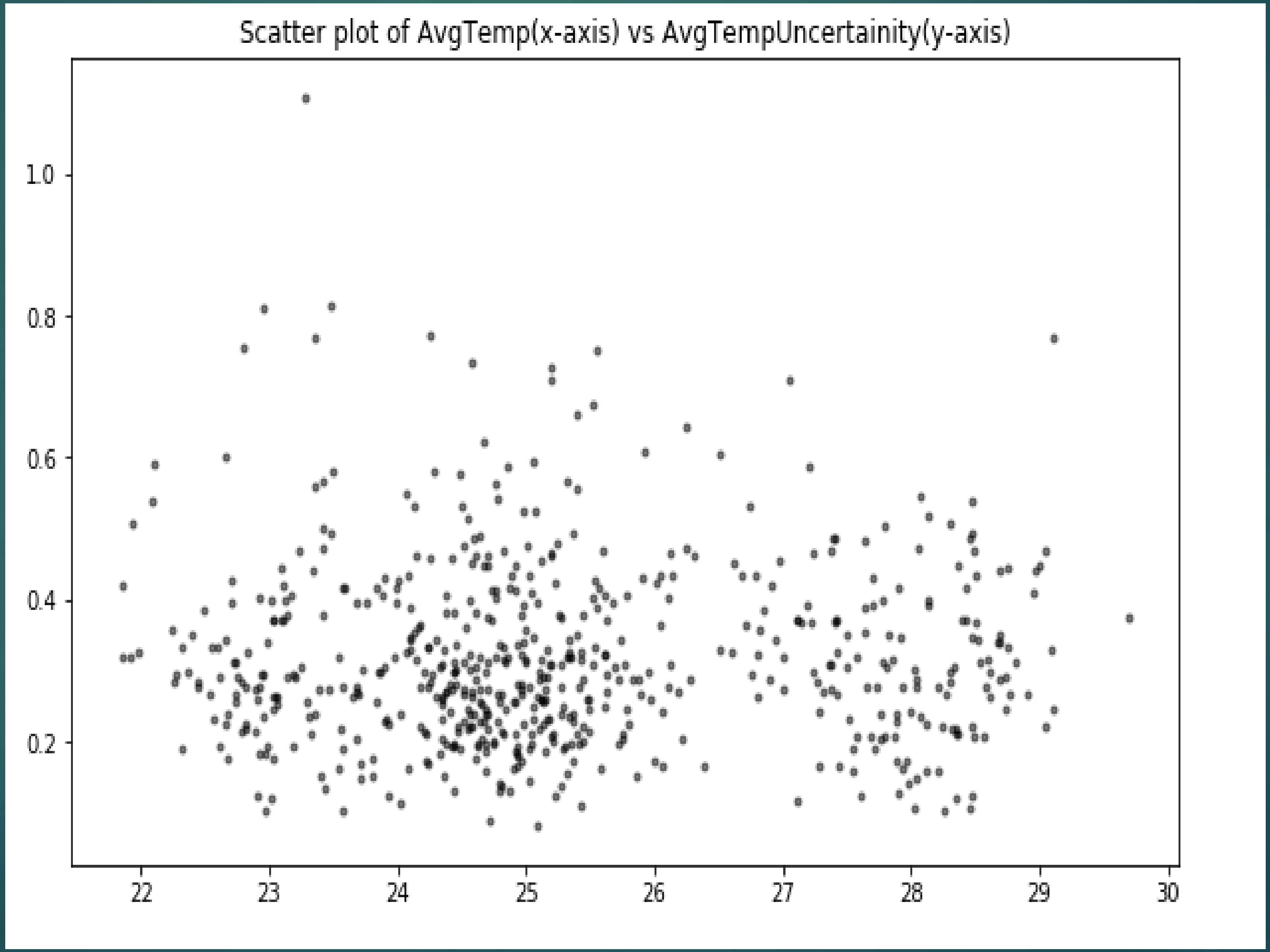
  numerical column 2: corresponding Average temperature

  uncertainty of Bangalore from 1960-2010

- It can be seen that the points are largely scattered in a random fashion.

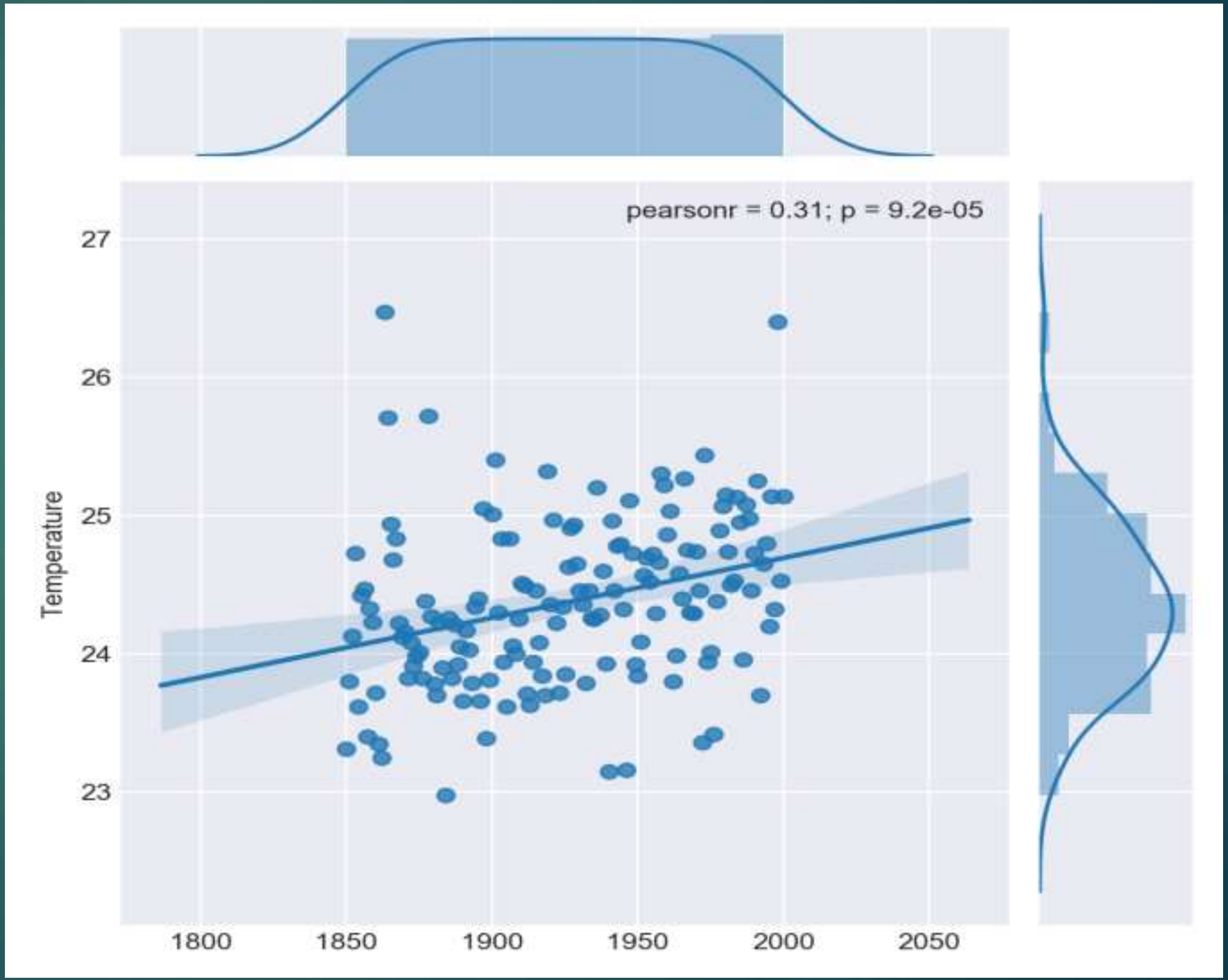
Scatter Plot Representation

5/4/2020



Scatter Plot with r = 0.31

Increasing Average Temperature (Compared to different cities)



Conclusion

- Average Temperature of all cities is changing rapidly and this climate change is a thread.
- We should find ways to stop or limit this climate change as soon as possible.

Thank You

5/4/2020