Python for Data Sciences – Numpy, Data Statistics, DataFrames

Numpy provides arithmetic operations on single and multi-dimensional arrays. It allows arithmetic, vector and matrix operations in a simple manner so that we do not have to write loops. To demonstrate the different capabilities of numpy, create a Python application project called NumpyTest. Then type the following code in the NumpyTest.py. Type up to a print statement and then examine and output to understand the construct.

```
import sys
import numpy as np
import math
import matplotlib.pyplot as plt

def main():
    # numpy is the math computing library
    a1 = np.zeros(4) # array of size 4 initialized to zeros
    print(a1)
    a1[2] = 7
    print(a1)

    a11 = np.ones(4)  # array of all 1's
    print(a11)

    a15 = np.full(4, 5)
    print(a15)

    # initialize with random values
    a1r = np.random.rand(5,2)  # 5 by 2 array
    print(a1r)
    a1r = np.random.rand(5000)
    plt.hist(a1r, bins = 25, density = True)
    plt.show()

    a2r = np.random.randint(5,10,size=(15,2)) # 5 to 9,  total 15 values
    print(a2r)

    a3r = np.random.uniform(-5, 5, size=(5000)) # 1/b-a  a=-5, b=5, 5000 values
    plt.hist(a3r, bins = 25, density = True)
    plt.show()

    a4r = np.random.normal(loc = 0, scale = 1, size = (5000))
    plt.hist(a4r, bins = 25, density = True)
    plt.show()

    # convert list to numpy array
    dlist = [7.5, 4.2, 6.7, 8]
    a2 = np.array(dlist)
    print(a2)

    m1 = [3, 1, 5, 2]
    m1np = np.array(m1)
    m2 = [3, 4, 2, 5]
    m2np = np.array(m2)
    # shape property tells us size of data
```

```
print(m1np.shape)
# use dot to compute dot product on two arrays
res = m1np.dot(m2np) # element by element multiplication
print('inner product=', res)

m3 = np.zeros((10,2,4))  # 10 rows and 2 cols
print(m3)
print(m3.shape)  # shape returns a tuple
print('m3: dimension 0 size =',m3.shape[0])

v1list = [1,2,4,3]
v2list = [5,2,3,6]

v1 = np.array(v1list)
v2 = np.array(v2list)
# we can use standard arithmetic operators to perform
# array addition, subtraction, multiplication and divide
vadd = v1 + v2
print(vadd)

vsub = v1 - v2
print(vsub)

vmul = v1 * v2
print(vmul)

vdiv = v1/v2
print(vdiv)

# we can do other arithmetic on entire array as well
# np.log, np.sin, np.exp  ...  many operations are available
vsqrt = np.sqrt(v1)
print(vsqrt)

vlog = np.log(v1)
print(vlog)

# compute softmax on an array
vsmax = np.exp(v1)/np.sum(np.exp(v1))
print(vsmax)


# -------Matrices----------
v1 = v1.reshape(len(v1),1)  # convert to 4x1
v2 = v2.reshape(len(v2),1) # convert to 4x1
# numpy does matrix multiplicatin if the objects are nxm
# If the objects are two arrays, then numpy does dot product
v3 = v1.dot(v2.T)
print(v3)
v4 = (v1.T).dot(v2)
print(v4)
m1 = np.array([[2,3,4],[5,3,2]]) # 2x3
m2 = np.array([[2,1],[3,4],[5,2]]) # 3x2
m3 = np.dot(m1,m2)
print(m3)
```

```
    # slicing operations (similar to lists) are available
    ma = np.array([[2,3,4],[5,3,2],[3,2,9],[6,4,7]]) # 3x4
    print(ma)
    mr2 = ma[2,:]  # row 2 of ma, can also use ma[2,]
    print(mr2)

    mc3 = ma[:,1]  # col 1 of ma, can also use ma[2,]
    print(mc3)

    mc4 = ma[1:3,0:2]
    print(mc4)
    print(ma[1,2])

    rowsum = np.sum(ma,axis=1) # axis=0 row,   axis=1  column
    print(rowsum)

    colsum = np.sum(ma,axis=0)
    print(colsum)

    # arange
    # Create an array of indices
    b = np.arange(3)
    print(b)

    # Select one element from each row of a using the indices in b
    print(ma[np.arange(3), b])  # prints ma[0,0], ma[1,1], ma[2,2]

    z = np.array([5,3,9,2,7])
    min_index = np.argmin(z)
    print('min value occurs at position=', min_index)

    z = np.array([5,3,9,2,7])
    max_index = np.argmax(z)
    print('max value occurs at position=', max_index)

    # we can use boolean conditions to filter the numpy array
    z2 = z[z>4]
    print(z2)

if __name__ == "__main__":
    sys.exit(int(main() or 0))
```

Data Statistics:

When we analyze data, we are often interested in the fundamental statistics in the data. If we analyze one particular aspect of data, we use the term **univariate** analysis. If we are examining how two features of data vary with respect to one another, this is referred to as **bivariate** analysis. Similarly, when many features are involved, the analysis becomes **multivariate**.

The basic statistics consist of the following:

- Central Tendency – Mean, Median and Mode

- Data Variability – Variance and Standard Deviation
- Joint Variability – Covariance and Correlation

Note that, in practice, the data is not always completely proper. There could be missing values, or due to experimental or measurement errors, there can be outliers. One of the goals of data sciences is to draw conclusions about the data without being too affected by the missing or outliers in the data,

When using Python for data sciences, we often use numpy arrays for storing data and performing appropriate mathematical calculations on it. Other Python libraries useful for data sciences related statistics and visualization include: sklearn, scikit, scipy, and scipy.stats, statistics, pandas and matplotlib.

Basic statistics on data include, mean, harmonic mean, geometric mean, median, mode, standard deviation, skewness, quantiles, covariance, correlation etc.. The mean is simple average of the data. Sometimes we need weights to be assigned to certain data, in that case, we compute the weighted mean.

$$= \frac{\sum_{i=1}^{n} w_i X_i}{\sum_{i=1}^{n} w_i}$$

Harmonic Mean is defined to be: $n \,/\, \Sigma_i(1/x_i)$
The weighted harmonic mean can be calculated using the following formula:

Weighted Harmonic Mean = (∑wi) / (∑wi/xi)

where:

wi – the weight of the data point
xi – the point in a dataset

Harmonic mean provides a better idea of the mean when the data involves ratios (e.g., P/E ratio in stocks)

Example of Harmonic Mean:
Determine the P/E ratio of the index of the stocks of Company A and Company B. Company A reports a market capitalization of $1 billion and earnings of $10 million, while Company B reports a market capitalization of $10 billion and earnings of $1 billion. The index consists of 40% of Company A and 60% of Company B.

First, we need to find the P/E ratios of each company. The P/E ratio is essentially the market capitalization divided by the earnings.

P/E (Company A) = ($1 billion) / ($10 million) = 100
P/E (Company B) = ($10 billion) / ($1 billion) = 10

We use the weighted harmonic mean to calculate the P/E ratio of the index. Using the formula for the weighted harmonic mean, the P/E ratio of the index can be found as:

P/E (Index) = (0.4+0.6) / (0.4/100 + 0.6/10) = 15.625

Note that if we calculate the P/E ratio of the index using the weighted arithmetic mean, it would be significantly overstated:

P/E (Index) = 0.4×100 + 0.6×10 = 46

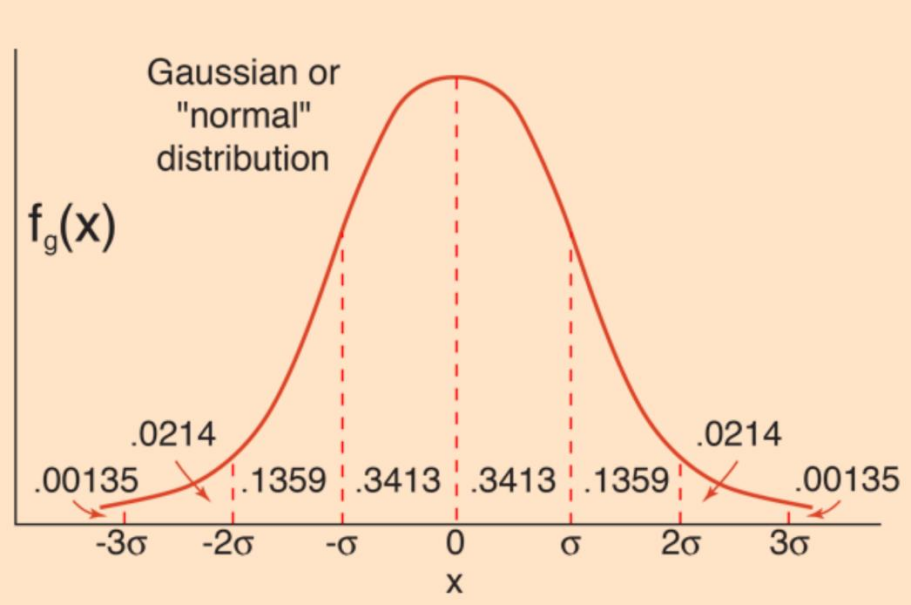
Geometric mean is more useful, when the data calculation involves powers such as interest rate calculations. The formula for the geometric mean is:

$$\left(\prod_{i=1}^{n} x_i\right)^{\frac{1}{n}} = \sqrt[n]{x_1 x_2 \cdots x_n}$$

Median is the middle element of sorted data. Mode is defined as the most commonly occurring element. The formula for standard deviation is:

$$\sigma = \sqrt{\frac{\sum (x_i - \mu)^2}{N}}$$

Note that square of the standard deviation is called the variance. For Gaussian distributed data, 68% of the data can be found with in one standard deviation of the mean, 95.5% of the data within two standard deviations of the mean, and 99.7% of the data within three standard deviations of the mean.



When multi-variate data is involved, we need concepts of covariance and correlation. The formula for covariance between two sets of data X and Y is:

$$\mathrm{Cov}(X, Y) = \frac{1}{n} \sum_{i=1}^{n} (x_i - \mu_x)(y_i - \mu_y)$$

We often use correlation coefficient to normalize the covariance. The correlation coefficient ranges from -1 to 1, with a value close to -1 or 1 indicating high correlation, while a value close to 0 indicates low correlation between the two data sets.

$$r = \frac{\mathrm{cov}(X, Y)}{S_X S_Y}$$

Where Sx, Sy are the standard deviations of X and Y.

Create a Python application project called DataStatistics. Add a file to the project called Utils.py with the following code in it. It has functions to visualize the

```
import matplotlib.pyplot as plt
import numpy as np
import scipy

def plot_data(data, mn):
    fig, ax = plt.subplots()
    y = [0 for x in range(0,len(data))]
    ax.plot(mn,0,'ro')
    ax.scatter(data,y)
    plt.show()

def plot_data3(data, mean, hmean,gmean):
    fig, ax = plt.subplots()
    y = [0 for x in range(0,len(data))]
    ax.plot(mean,0,'ro')
    ax.plot(hmean,0,'yo')
    ax.plot(gmean,0,'go')
    ax.scatter(data,y)
    plt.show()
```

Type the following code in DataStatistics.py. Type up to a print statement and examine the output to understand the code nicely.

```
import sys
import math
import statistics
import scipy.stats
import Utils
import numpy as np

def main():
    # scipy has many data science algorithms implemented in it
    dx = [6.5, 3.7, 2.8, 9.4, 8.3]
    # often our data has missing or invalid values
    dxn = [6.5, 3.7, 2.8, math.nan, 9.4, 8.3, 7.3]
    dxn[5] = math.nan
    print(dxn)
    dxn[2] = float('nan')  # same as mat.nan
    print(dxn)
```

```
# find the count of invalid i.e., nan data
count_nan = sum([1 for x in dxn if math.isnan(x)])
print('nan found in data =',count_nan)

# find mean of data
avg = sum(dxn)/len(dxn)
print('mean of data =',avg)

dx_clean = [x for x in dxn if not math.isnan(x)]
print(dx_clean)
avg = sum(dx_clean)/len(dx_clean)
print('mean of clean data=',avg)
Utils.plot_data(dx_clean,avg)

# weighted mean - each data item may have a different weight
wt = [0.05, 0.05, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.05, 0.05]
scores = [15, 25, 43, 65, 68, 72, 76, 78, 85, 92]
res = sum(list(map(lambda x,y:x*y, scores, wt)))
print('weighted avg=', res,' avg=',np.mean(np.array(scores)))

# harmonic mean - e.g., used in computing PE of index funds
dx = [2.4, 3.1, 2.7, 3.5, 2.0, 19]
mean = sum(dx)/len(dx)
hmean = len(dx)/sum(1/x for x in dx)
print('mean=', mean,'harmonic mean=', hmean)

# weighted harmonic mean
wt = [0.2, 0.1, 0.1, 0.3, 0.2,0.1]
hmean_weighted = sum(wt)/sum(w/x for (x,w) in zip(dx,wt))
print('weighted harmonic mean = ',hmean_weighted)

wt = [0.4, 0.6]
pe = [50,4]
hmean_weighted = sum(wt)/sum(w/x for (x,w) in zip(pe,wt))
print('index fund pe = ',hmean_weighted)
hmean_stats = statistics.harmonic_mean(dx)
print('harmonic mean from statistics =',hmean_stats)

# geometric mean - used in interest rates (powers are involved)
dx = [2.4, 3.1, 2.7, 3.5, 2.0, 19]
prod = 1
for x in dx:
    prod *= x
gmean = prod**(1/len(dx))
print('geometric mean =',gmean)

gmean_stats = statistics.geometric_mean(dx)
print(' geometric mean from statistics=',gmean_stats)
mean = sum(dx)/len(dx)
Utils.plot_data3(dx,mean,hmean,gmean)

# median - middle element of sorted data
dx = [4, 3, 6, 8, 7, 32, 5, 19, 8, 6]
dxs = dx.copy()  # copy the data
dxs.sort()
```

```
    if (len(dx)%2 == 1):
        median = dxs[int(len(dxs/2))]
    else:
        median = (dxs[int(len(dxs)/2)] + dxs[int(len(dxs)/2)-1])/2
    print(dxs)
    print('median =', median)
    # median is less affected by outliers (usually)
    # sometimes we need the median to be the actual data
    # median low and median high
    medlow = statistics.median_low(dx)
    medhigh = statistics.median_high(dx)
    print('median low=',medlow, ' median high =',medhigh)

    # mode - value that occurs most often
    dx = [4, 3, 6, 8, 7, 8, 15, 8, 6, 3, 8]
    posmax = np.argmax([dx.count(x) for x in dx])  # [1,2,2,4,1,4,---
    mode = dx[posmax]
    print('mode = ', mode)
    mode_stats = statistics.mode(dx)
    print('mode from statistics =',mode_stats)

    dx = [58, 65, 67, 68, 72, 77, 80, 87, 95 ]
    mean = sum(dx)/len(dx)
    var = sum([(x-mean)**2 for x in dx])/(len(dx)-1)
    std = math.sqrt(var)
    print('mean=', mean,' variance =',var, ' std=',std)

    var_stats = statistics.variance(dx)
    print('variance from stats=',var_stats)

    # percentiles, quantiles
    res = np.percentile(dx,[50,70])
    print(res)
    # quantile :  first quantile = first 25%, fourth quantile >75%
    dx.sort()
    #qt4 = statistics.quantiles(dx,n=1) # second quantile = mean
    #print(qt4)

    # correlation - between two sets of data
    x = [56, 59, 63, 64, 67, 68, 71, 85, 88, 91]
    y = [63, 67, 73, 77, 81, 83, 84, 87, 89, 94]

    mean_x, mean_y = sum(x)/len(x), sum(y)/len(y)
    cov_xy = sum((x[k]-mean_x)*(y[k]-mean_y) for k in range(len(x)))/(len(x)-1)
    print('covxy=',cov_xy)
    var_x, var_y = statistics.variance(x), statistics.variance(y)
    std_x, std_y = var_x**0.5, var_y**0.5
    corr_coeff = cov_xy/(std_x*std_y)
    print('corr_coeff=',corr_coeff)

    r, p = scipy.stats.pearsonr(x,y)
    print('r =',r, ' p=',p)

if __name__ == "__main__":
    sys.exit(int(main() or 0))
```

Data Visualization:

We often visualize the data before proceeding with in depth analytics on it. For data read from an Excel or a comma/tab delimited file, we often read into a Pandas dataframe. Pandas library provides a Series object for 1-D data, and a DataFrame for 2-D data. We can easily convert a series, or a dataframe to numpy array by its values property.

Create another project called TwoDData. Type the following code in the TwoData.py.

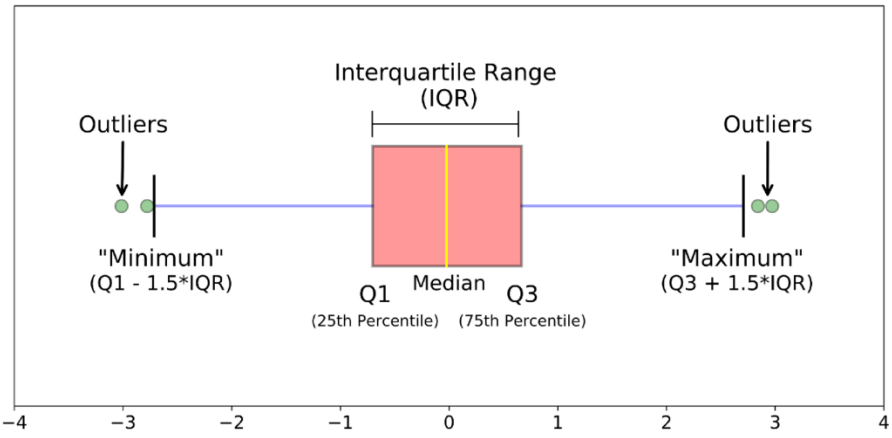
```
import sys
import numpy as np
import scipy.stats
import pandas as pd

def main():
    a = np.array([[27,50000,7],[32,36000,8],[43,90000,2],
                  [39,40000,5],[35,82000,8]])
    print(a)
    print(a.shape)
    mean = np.mean(a)
    print('mean =', mean)
    mn1 = np.mean(a, axis=0)  # axis=0  result row,  axis=1=> column
    print(mn1)
    desc = scipy.stats.describe(a, axis=0)
    print(desc)

    # data is usually arranged as rows and columns
    # ususally we read it into a dataframe (pandas library)
    # think of dataframe like an excel sheet
    col_names = ['age','salary','years_worked']
    df = pd.DataFrame(a,columns=col_names)
    print(df)
    print(df.mean(axis=0))
    print('avg salary=',df['salary'].mean())
    print(df.describe())
    sal = df.iloc[3,1]  # index via position
    print('salary=',sal)

if __name__ == "__main__":
    sys.exit(int(main() or 0))
```

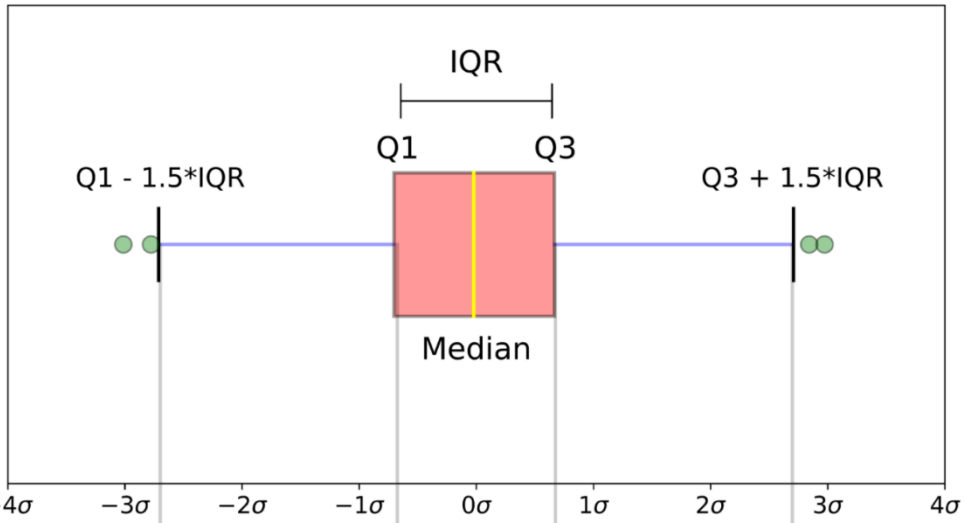
For visualization of data, we often use, boxplots, bar graphs, scatter plots, pie charts etc.. To demonstrate some of the basic plots, add a file to the project called Visualization.py with the following code in it. Before you run it, set it as the start file.

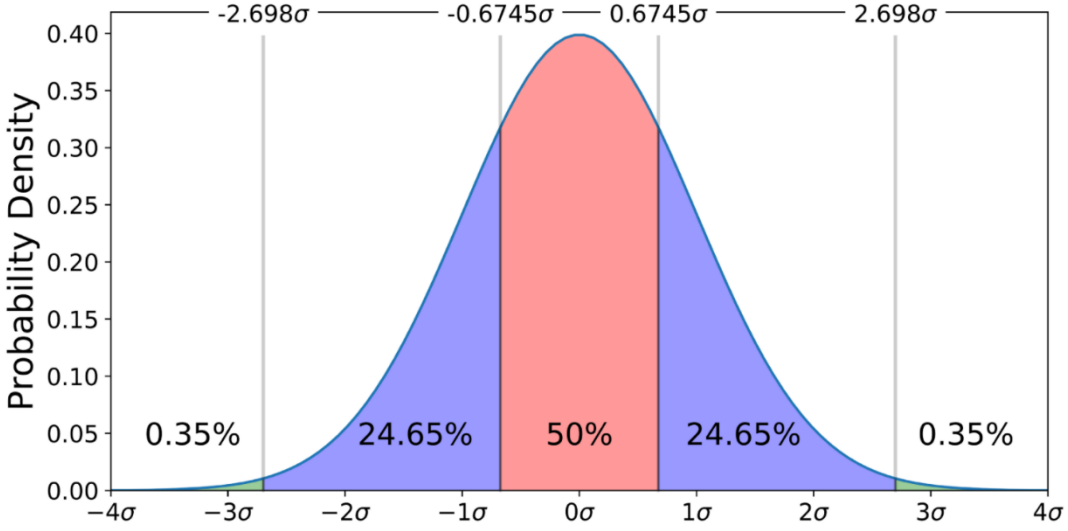
```
import sys
import numpy as np
import matplotlib.pyplot as plt

def main():
    # for random initialization of data
    # normally distributed (bell shaped) - np.random.randn()
    # uniformally distributed integers - np.random.randint()
    np.random.seed(seed=0) # for repeatable results
```

```
    x = np.random.randn(100000) # mean=0, stddev = 1
    y = np.random.randn(100)
    z = np.random.randn(10)
    print(z)
    fig,ax = plt.subplots()
    ax.boxplot((x,y,z),vert=False, showmeans=True, meanline=True,
               labels=('x','y','z'))
    plt.show()

    # histogram
    hist, bin_edges = np.histogram(x, bins=100)
    fig, ax = plt.subplots()
    ax.hist(x, bin_edges, cumulative=False)
    ax.set_xlabel('x data')
    ax.set_ylabel('count of x')
    plt.show()

    # histogram
    hist, bin_edges = np.histogram(x, bins=100)
    fig, ax = plt.subplots()
    ax.hist(x, bin_edges, cumulative=True)
    ax.set_xlabel('x data')
    ax.set_ylabel('count of x')
    plt.show()

    # pie chart
    #x, y, z,w = 128000, 125600, 212840, 252000
    sales = [128000, 125600, 212840, 252000]
    fig, ax = plt.subplots()
    #ax.pie((x,y,z,w),labels=('Q1','Q2','Q3','Q4'))
    ax.pie(sales,labels=('Q1','Q2','Q3','Q4'))
    plt.show()

    # bar graph
    person = [1, 2, 3, 4, 5]
    age = [17, 29, 37, 40, 26]
    fig,ax = plt.subplots()
    ax.bar(person,age)
    ax.set_xlabel('person')
    ax.set_ylabel('age')
    plt.show()

if __name__ == "__main__":
    sys.exit(int(main() or 0))
```

Box plot is clarified though the following diagrams.







Data Scaling:

Before we apply data analytics, or AI/machine learning to data, we often scale the data. There are three popular scaling techniques:

- Max scaling
- Min-Max Scaling
- Z-Score scaling

In max scaling, we divide each data item by the maximum of that particular feature. For example, if one of the features is height of a person, then we may divide each person's height data by 7 (assuming 7 ft is the maximum possible height in our dataset).

The min-max scaling adjusts each data item as:

xi = (xi-min)/(max-min)

The z score scaling subtracts the mean of the feature and divides it by the standard deviation of that feature.

xi = (xi – mean)/std dev

To visualize the effect of scaling on each feature, create another project called "DataScaling". Then type the following code in the datascaling.py file.

```
import sys
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.preprocessing import MinMaxScaler, StandardScaler

def main():
    df = pd.DataFrame([
        [2017,85000,150,30],
        [2012,175000,130,27],
        [2005,265000,170,18],
        [2019,25000,250,35],
        ],
          columns=['year','miles','hp','mpg'])
    print(df)
    df.plot(kind='bar')
    plt.show()

    # Max scaling
    dfcopy = df.copy()
    for column in df.columns:
        dfcopy[column] = dfcopy[column]/dfcopy[column].abs().max()
    print(dfcopy)
    dfcopy.plot(kind='bar')
    plt.show()

    # In previous scaling, the year was always close to 1
    # after scaling. Min Max scaling can solve this problem
    # xi = (xi - min)/(max - min)
    dfcopy2 = df.copy()
    for column in dfcopy2.columns:
```

```
        dfcopy2[column] = (dfcopy2[column] - dfcopy2[column].min())/ \
            (dfcopy2[column].max() - dfcopy2[column].min())
    print(dfcopy2)
    dfcopy2.plot(kind='bar')
    plt.show()

    # Z score scaling   xi = (xi - mean)/std dev
    dfcopy3 = df.copy()
    for column in dfcopy3.columns:
        dfcopy3[column] = (dfcopy3[column] - dfcopy3[column].mean())/ \
            dfcopy3[column].std()
    print(dfcopy3)
    dfcopy3.plot(kind='bar')
    plt.show()

    dcars = df.values  # converts dataframe to numpy array
    print(dcars)
    scalermm = MinMaxScaler()
    scaled_dcars = scalermm.fit_transform(dcars)
    print(scaled_dcars)

    # standard scaler  - zscore scaler
    scaler = StandardScaler()
    scaled_dcarsz = scaler.fit_transform(dcars)
    print(scaled_dcarsz)


if __name__ == "__main__":
    sys.exit(int(main() or 0))
```