TASK #1: TRANSFORMATION: PERFORM NORMALIZATION

FEATURE SCALING

- Feature Scaling is an important step to take prior to training of machine learning models to ensure that features are within the same scale.
- Example: interest rate and employment score are at a different scale. This will result in one feature dominating the other feature.
- Scikit Learn offers several tools to perform feature scaling.

RAW ORIGINAL DATASET

	Interest Rates	Employment	S&P 500 Price
0	1.943859	55.413571	2206.680582
1	2.258229	59.546305	2486.474488
2	2.215863	57.414687	2405.868337
3	1.977960	49.908353	2140.434475
4	2.437723	52.035492	2411.275663
5	2.143637	56.060598	2187.344909
6	2.148647	51.513208	2263.049249
7	2.176184	53.475909	2281.496374
8	2.125352	63.668422	2355.163011
9	2.225682	56.993396	2326.330337
10	1.814688	55.361780	2078.553895
11	2.281897	58.484752	2337.504507
12	2.426738	55.709328	2485.774097

QUICK STATS!

	Interest Rates	Employment	S&P 500 Price
count	1000.00	1000.00	1000.00
mean	2.20	56.25	2320.00
std	0.24	4.86	193.85
min	1.50	40.00	1800.00
25%	2.04	53.03	2190.45
50%	2.20	56.16	2312.44
75%	2.36	59.42	2455.76
max	3.00	70.00	3000.00

NORMALIZATION

- Normalization is conducted to make feature values range from 0 to 1.

$$\boxed{x' = \frac{x - \min(x)}{\max(x) - min(x)}}$$

```python
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()
stock_df = scaler.fit_transform(stock_df)
```

In [4]:

```

data = {'Age': [25, 30, 35, 40, 45],
'Income': [50000, 60000, 75000, 90000, 100000]}

df = pd.DataFrame(data)
df
```

Out[4]:

	Age	Income
0	25	50000
1	30	60000
2	35	75000
3	40	90000
4	45	100000

In [5]:

```
# Min-Max Scaling
min_age = df['Age'].min()
max_age = df['Age'].max()
min_income = df['Income'].min()
max_income = df['Income'].max()

df['Scaled_Age'] = (df['Age'] - min_age) / (max_age - min_age)
df['Scaled_Income'] = (df['Income'] - min_income) / (max_income - min_income)

df
```

Out[5]:

	Age	Income	Scaled_Age	Scaled_Income
0	25	50000	0.00	0.0
1	30	60000	0.25	0.2
2	35	75000	0.50	0.5
3	40	90000	0.75	0.8
4	45	100000	1.00	1.0

In [ ]:

```
# Let's read a CSV file using Pandas as follows
df = pd.read_csv('Life_Expectancy_Data.csv')
df
```

In [ ]:

```
df['Life expectancy '].values
```

In [ ]:

```
# Normalization is conducted to make feature values range from 0 to 1
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()
df['Life expectancy '] = scaler.fit_transform(df['Life expectancy '].values.reshape(-1,1))
```

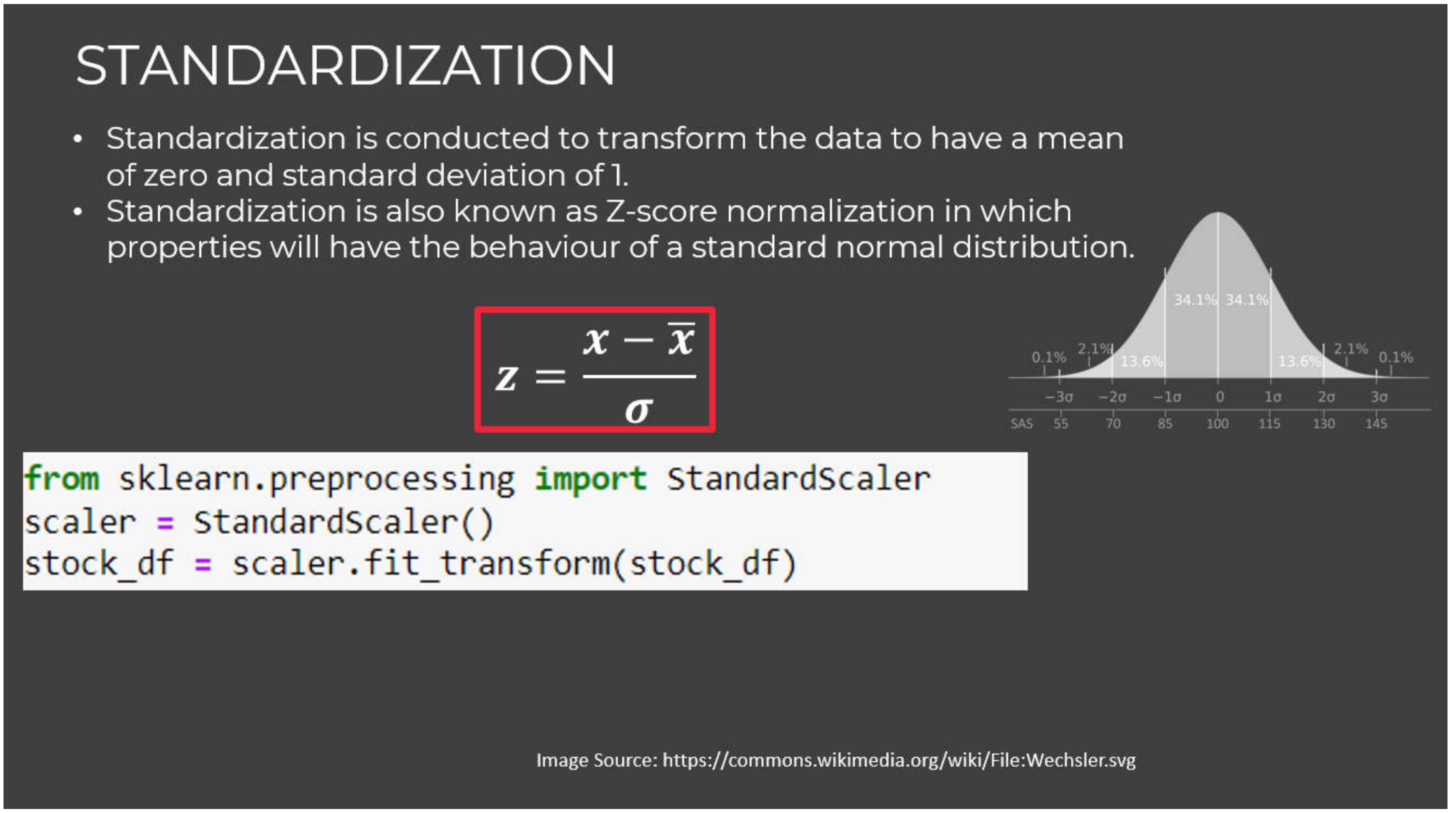
In [ ]:

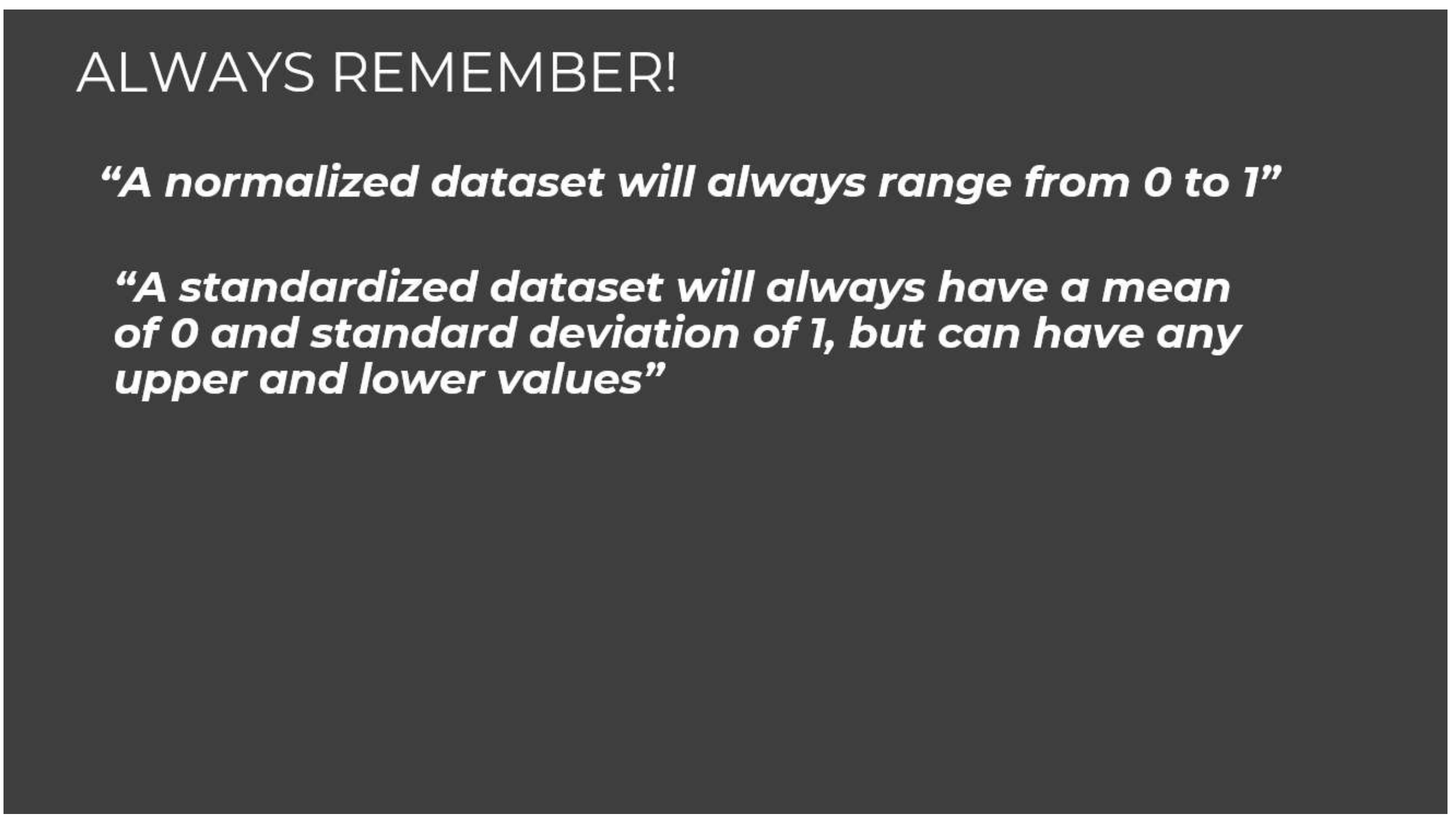
```
df['Life expectancy ']
```

In [ ]:

```
round(df['Life expectancy '].describe())
```

TASK #2: PERFORM STANDARDIZATION





In [6]:

```python

data = {'Age': [25, 30, 35, 40, 45],
'Income': [50000, 60000, 75000, 90000, 100000]}

df = pd.DataFrame(data)
df

```

Out[6]:

	Age	Income
0	25	50000
1	30	60000
2	35	75000
3	40	90000
4	45	100000

In [ ]:

```python

```

In [10]:

```python
# Standarization
mean_age = df['Age'].mean()
std_age = df['Age'].std()
mean_income = df['Income'].mean()
std_income = df['Income'].std()

df['Normalized_Age'] = (df['Age'] - mean_age) / std_age
df['Normalized_Income'] = (df['Income'] - mean_income) / std_income

df
```

Out[10]:

	Age	Income	Normalized_Age	Normalized_Income
0	25	50000	-1.264911	-1.212678
1	30	60000	-0.632456	-0.727607
2	35	75000	0.000000	0.000000
3	40	90000	0.632456	0.727607
4	45	100000	1.264911	1.212678

In [11]:

```python
df.describe()
```

Out[11]:

	Age	Income	Normalized_Age	Normalized_Income
count	5.000000	5.000000	5.000000e+00	5.000000
mean	35.000000	75000.000000	4.440892e-17	0.000000
std	7.905694	20615.528128	1.000000e+00	1.000000
min	25.000000	50000.000000	-1.264911e+00	-1.212678
25%	30.000000	60000.000000	-6.324555e-01	-0.727607
50%	35.000000	75000.000000	0.000000e+00	0.000000
75%	40.000000	90000.000000	6.324555e-01	0.727607
max	45.000000	100000.000000	1.264911e+00	1.212678

In [ ]:

```
# Let's read a CSV file using Pandas as follows
df = pd.read_csv('Life_Expectancy_Data.csv')
df
```

In [ ]:

```
df['Life expectancy '].values
```

In [ ]:

```
# Normalization is conducted to make feature values range from 0 to 1
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
df['Life expectancy '] = scaler.fit_transform(df['Life expectancy '].values.reshape(-1,1))
```

In [ ]:

```
df['Life expectancy ']
```

In [ ]:

```
round(df['Life expectancy '].describe())
```

In [ ]:

```

```

TASK #3. DIMENTIONALITY REDUCTION

In [15]:

```
import numpy as np
from scipy import linalg
```

In [16]:

```
X = np.array([[20,50,4],[25,60,5],[30,65,5.5],[40,75,6]])
```

In [17]:

```
data = pd.DataFrame(X,columns=["age","weight","height"])
data
```

Out[17]:

	age	weight	height
0	20.0	50.0	4.0
1	25.0	60.0	5.0
2	30.0	65.0	5.5
3	40.0	75.0	6.0

In [18]:

```
X = np.array(X)
#mean of each column
M=np.mean(X, axis=0)
print(M)
#center columns by subtracting mean
C= X-M
print(C)
# calculate covariance matrix
#V=np.cov(C,rowvar = False)
V=np.cov(C.T)
print(V)
```

```
[28.75  62.5    5.125]
[[ -8.75  -12.5    -1.125]
 [ -3.75   -2.5    -0.125]
 [  1.25    2.5     0.375]
 [ 11.25   12.5     0.875]]
[[ 72.91666667  87.5          6.875     ]
 [ 87.5        108.33333333   8.75      ]
 [  6.875        8.75         0.72916667]]
```

In [19]:

```
#calculate eigen vectors and values
eigen_values, eigen_vectors = linalg.eig(V)
print(eigen_values)
print(eigen_vectors)
```

```
[1.80587147e+02+0.j 1.38624052e+00+0.j 5.77924152e-03+0.j]
[[ 0.63183775  0.77042797  0.08498117]
 [ 0.77263783 -0.61729995 -0.14822808]
 [ 0.06174019 -0.15931576  0.9852952 ]]
```

In [20]:

```
#check explained variance ratio of each component
print(eigen_values[0]/np.sum(eigen_values))
print(eigen_values[1]/np.sum(eigen_values))
print(eigen_values[2]/np.sum(eigen_values))
```

```
(0.9923506641518309+0j)
(0.007617578138632018+0j)
(3.175770953700823e-05+0j)
```

In [21]:

```
#sort the eigenvalues in descending order
sorted_index = np.argsort(eigen_values)[::-1]
print(sorted_index)
sorted_eigenvalue = eigen_values[sorted_index]
print(sorted_eigenvalue)
#similarly sort the eigenvectors
sorted_eigenvectors = eigen_vectors[:,sorted_index]
print(sorted_eigenvectors)
```

```
[0 1 2]
[1.80587147e+02+0.j 1.38624052e+00+0.j 5.77924152e-03+0.j]
[[ 0.63183775  0.77042797  0.08498117]
 [ 0.77263783 -0.61729995 -0.14822808]
 [ 0.06174019 -0.15931576  0.9852952 ]]
```

In [22]:

```
# select the first n eigenvectors, n is desired dimension
# of our final reduced data.

n_components = 2 #you can select any number of components.
eigenvector_subset = sorted_eigenvectors[:,0:n_components]
```

In [24]:

```
X_reduced = np.dot(eigenvector_subset.transpose() , C.transpose() ).transpose()
X_reduced
```

Out[24]:

```
array([[-15.25601083,   1.15423481],
       [ -4.30870364,  -1.32594056],
       [  2.74454432,  -0.63995831],
       [ 16.82017015,   0.81166407]])
```

In [25]:

```
#new dataframe with reduced features
df = pd.DataFrame(X_reduced,columns=["PC1","PC2"])
df
```

Out[25]:

	PC1	PC2
0	-15.256011	1.154235
1	-4.308704	-1.325941
2	2.744544	-0.639958
3	16.820170	0.811664

In [12]:

```
# Let's read a CSV file using Pandas as follows
df = pd.read_csv('Life_Expectancy_Data.csv')
df
```

Out[12]:

	Year	Status	Life expectancy	Adult Mortality	infant deaths	Alcohol	percentage expenditure	Hepatitis B	Measles	BMI	...	Polio	Total expenditure	Diphtheria	HIV/
0	2015	Developing	65.0	263.0	62	0.01	71.279624	65.0	1154	19.1	...	6.0	8.16	65.0	
1	2014	Developing	59.9	271.0	64	0.01	73.523582	62.0	492	18.6	...	58.0	8.18	62.0	
2	2013	Developing	59.9	268.0	66	0.01	73.219243	64.0	430	18.1	...	62.0	8.13	64.0	
3	2012	Developing	59.5	272.0	69	0.01	78.184215	67.0	2787	17.6	...	67.0	8.52	67.0	
4	2011	Developing	59.2	275.0	71	0.01	7.097109	68.0	3013	17.2	...	68.0	7.87	68.0	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
2933	2004	Developing	44.3	723.0	27	4.36	0.000000	68.0	31	27.1	...	67.0	7.13	65.0	
2934	2003	Developing	44.5	715.0	26	4.06	0.000000	7.0	998	26.7	...	7.0	6.52	68.0	
2935	2002	Developing	44.8	73.0	25	4.43	0.000000	73.0	304	26.3	...	73.0	6.53	71.0	
2936	2001	Developing	45.3	686.0	25	1.72	0.000000	76.0	529	25.9	...	76.0	6.16	75.0	
2937	2000	Developing	46.0	665.0	24	1.68	0.000000	79.0	1483	25.5	...	78.0	7.10	78.0	

2938 rows × 21 columns

In [ ]:

```

```

TASK #4. PANDAS WITH FUNCTIONS

In [ ]:

```
# Let's read a CSV file using Pandas as follows
df = pd.read_csv('Life_Expectancy_Data.csv')
df
```

In [ ]:

```
# Let's assume the percentage expenditure has increased by 5
# Define a function that increases all elements by a fixed value of 5% (for simplicity sake)
def percentage_expenditure_update(balance):
    return balance + 5
```

In [ ]:

```
# You can apply a function to the DataFrame
df['percentage expenditure'] = df['percentage expenditure'].apply(percentage_expenditure_update)
df
```

In [ ]:

```

```

HOME TASK OVERVIEW

- In this project, we will perform basic Exploratory Data Analysis (EDA) on the Kyphosis disease Dataset.
- Kyphosis is an abnormally excessive convex curvature of the spine.
- Dataset contains 81 rows and 4 columns representing data on children who have had corrective spinal surgery.
  - INPUTS: 1. Age: in months, 2. Number: the number of vertebrae involved, 3. Start: the number of the first (topmost) vertebra operated on.
  - OUTPUTS: Kyphosis which represents a factor with levels absent present indicating if a kyphosis (a type of deformation) was present after the operation.
- Using the “kyphosis.csv" included in the course package, write a python script to perform the following tasks:
  1. Import the “kyphosis.csv" file using Pandas
  2. Perform basic Exploratory Data Analysis (EDA) on the data
  3. List the average, minimum and maximum age (in years) considered in this study using 2 methods
  4. Plot the correlation matrix
  5. Convert the age column datatype from int64 to float64
  6. Define a function that converts age from months to years
  7. Apply the function to the “Age” column and add the results into a new column entitled “Age in Years”
  8. What are the features of the oldest and youngest child in this study?
  9. Scale the raw Age column (in months) using both standardization and Normalization. Perform a sanity check.

Great Job!