ML Feature Engineering Assignment

1. Loading Dataset (Iris, Boston)

```
import pandas as pd
import numpy as np

from sklearn import datasets
```



```
df_iris.head()
```

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	target
0	5.1	3.5	1.4	0.2	0
1	4.9	3.0	1.4	0.2	0
2	4.7	3.2	1.3	0.2	0
3	4.6	3.1	1.5	0.2	0
4	5.0	3.6	1.4	0.2	0

```
boston_data = datasets.load_boston()
df_boston = pd.DataFrame(boston_data.data,columns=boston_data.feature_names)
df_boston['target'] = pd.Series(boston_data.target)
df_boston.head()
```

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTAT	target
0	0.00632	18.0	2.31	0.0	0.538	6.575	65.2	4.0900	1.0	296.0	15.3	396.90	4.98	24.0
1	0.02731	0.0	7.07	0.0	0.469	6.421	78.9	4.9671	2.0	242.0	17.8	396.90	9.14	21.6
2	0.02729	0.0	7.07	0.0	0.469	7.185	61.1	4.9671	2.0	242.0	17.8	392.83	4.03	34.7
3	0.03237	0.0	2.18	0.0	0.458	6.998	45.8	6.0622	3.0	222.0	18.7	394.63	2.94	33.4
4	0.06905	0.0	2.18	0.0	0.458	7.147	54.2	6.0622	3.0	222.0	18.7	396.90	5.33	36.2

2. Standardization, Normalization, Scaling (Standard scaler, min-max scaler)

In statistics and machine learning, data **standardization** is a process of converting data to z-score values based on the mean and standard deviation of the data. **Z_i = (x_i - mean(x))/SD(x)**

```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()

standardized_data = scaler.fit_transform(df_boston)

print(standardized_data)
```

```
[[-0.41978194  0.28482986 -1.2879095  ...  0.44105193 -1.0755623
   0.15968566]
 [-0.41733926 -0.48772236 -0.59338101 ...  0.44105193 -0.49243937
  -0.10152429]
 [-0.41734159 -0.48772236 -0.59338101 ...  0.39642699 -1.2087274
   1.32424667]
 ...
 [-0.41344658 -0.48772236  0.11573841 ...  0.44105193 -0.98304761
   0.14880191]
 [-0.40776407 -0.48772236  0.11573841 ...  0.4032249  -0.86530163
  -0.0579893 ]
 [-0.41500016 -0.48772236  0.11573841 ...  0.44105193 -0.66905833
  -1.15724782]]
```

```
standardized_df = pd.DataFrame(standardized_data, columns=df_boston.columns)

standardized_df.head()
```

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTAT	t
0	-0.419782	0.284830	-1.287909	-0.272599	-0.144217	0.413672	-0.120013	0.140214	-0.982843	-0.666608	-1.459000	0.441052	-1.075562	0.1
1	-0.417339	-0.487722	-0.593381	-0.272599	-0.740262	0.194274	0.367166	0.557160	-0.867883	-0.987329	-0.303094	0.441052	-0.492439	-0.1
2	-0.417342	-0.487722	-0.593381	-0.272599	-0.740262	1.282714	-0.265812	0.557160	-0.867883	-0.987329	-0.303094	0.396427	-1.208727	1.3
3	-0.416750	-0.487722	-1.306878	-0.272599	-0.835284	1.016303	-0.809889	1.077737	-0.752922	-1.106115	0.113032	0.416163	-1.361517	1.1
4	-0.412482	-0.487722	-1.306878	-0.272599	-0.835284	1.228577	-0.511180	1.077737	-0.752922	-1.106115	0.113032	0.441052	-1.026501	1.4

Double click (or enter) to edit

Machine learning algorithms tend to perform better or converge faster when the different features (variables) are on a smaller scale. Therefore it is common practice to normalize the data before training machine learning models on it. **Normalization** also makes the training process less sensitive to the scale of the features. This results in getting better coefficients after training. The formula for Normalization is **x_norm = (x-x_min)/(x_max-x_min)**

```
from sklearn import preprocessing
d = preprocessing.normalize(df_boston, axis=0)
scaled_df = pd.DataFrame(d, columns=df_boston.columns)
scaled_df.head()
```

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTAT	target
0	0.000030	0.030868	0.007853	0.0	0.042208	0.046222	0.039107	0.041904	0.003442	0.029798	0.036604	0.047927	0.015241	0.043845
1	0.000130	0.000000	0.024035	0.0	0.036795	0.045139	0.047324	0.050890	0.006883	0.024362	0.042585	0.047927	0.027973	0.039461
2	0.000130	0.000000	0.024035	0.0	0.036795	0.050510	0.036648	0.050890	0.006883	0.024362	0.042585	0.047436	0.012334	0.063393
3	0.000154	0.000000	0.007411	0.0	0.035932	0.049196	0.027471	0.062109	0.010325	0.022349	0.044738	0.047653	0.008998	0.061018
4	0.000329	0.000000	0.007411	0.0	0.035932	0.050243	0.032509	0.062109	0.010325	0.022349	0.044738	0.047927	0.016312	0.066133

Using **MinMaxScaler() to Normalize Data** in Python Sklearn provides another option when it comes to normalizing data: MinMaxScaler.

This is a more popular choice for normalizing datasets.

Here's the code for normalizing the housing dataset using MinMaxScaler :

```
from sklearn.preprocessing import MinMaxScaler

scaler = MinMaxScaler()

normalized_data = scaler.fit_transform(df_boston)

normalized_df = pd.DataFrame(normalized_data, columns=df_boston.columns)

normalized_df.head()
```

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTAT	target
0	0.000000	0.18	0.067815	0.0	0.314815	0.577505	0.641607	0.269203	0.000000	0.208015	0.287234	1.000000	0.089680	0.422222
1	0.000236	0.00	0.242302	0.0	0.172840	0.547998	0.782698	0.348962	0.043478	0.104962	0.553191	1.000000	0.204470	0.368889
2	0.000236	0.00	0.242302	0.0	0.172840	0.694386	0.599382	0.348962	0.043478	0.104962	0.553191	0.989737	0.063466	0.660000
3	0.000293	0.00	0.063050	0.0	0.150206	0.658555	0.441813	0.448545	0.086957	0.066794	0.648936	0.994276	0.033389	0.631111
4	0.000705	0.00	0.063050	0.0	0.150206	0.687105	0.528321	0.448545	0.086957	0.066794	0.648936	1.000000	0.099338	0.693333

3. Label Encoder, Label binarizer, oneHot Encoder

Label Encoding refers to converting the labels into a numeric form so as to convert them into the machine-readable form. Machine learning algorithms can then decide in a better way how those labels must be operated. It is an important pre-processing step for the structured dataset in supervised learning.

```
le = preprocessing.LabelEncoder()
le.fit(["paris", "paris", "tokyo", "amsterdam"])
```

```
LabelEncoder()
```

```
list(le.classes_)
```



```
le.transform(["tokyo", "tokyo", "paris"])
```

```
array([2, 2, 1])
```

```
list(le.inverse_transform([2, 2, 1]))
```

```
['tokyo', 'tokyo', 'paris']
```

Difference between Label Encoding and One-hot Encoding

1. **Label Encoder:** The categorical values are labeled into numeric values by assigning each category to a number.

   One-hot Encoding: A column with categorical values is split into multiple columns.

2. **Label Encoder:** Different columns are not added. Rather different categories are converted into numeric values. So fewer computations.

   One-hot Encoding: It will add more columns and will be computationally heavy

3. **Label Encoder:** Unique information is there

   One-hot Encoding: Redundant information is there

4. **Label Encoder:** Different integers are used to represent data

   One-hot Encoding: Only 0 and 1 are used to represent data

```
from numpy import array
from sklearn.preprocessing import LabelEncoder
from sklearn.preprocessing import OneHotEncoder
from sklearn.preprocessing import LabelBinarizer

# define example
data = ['cold', 'cold', 'warm', 'cold', 'hot', 'hot', 'warm', 'cold',
'warm', 'hot']
values = array(data)
print("Data:\n ", values)
# integer encode
label_encoder = LabelEncoder()
integer_encoded = label_encoder.fit_transform(values)
print("Label Encoder:\n" ,integer_encoded)

# onehot encode
onehot_encoder = OneHotEncoder(sparse=False)
integer_encoded = integer_encoded.reshape(len(integer_encoded), 1)
onehot_encoded = onehot_encoder.fit_transform(integer_encoded)
print("OneHot Encoder:\n", onehot_encoded)

#Binary encode
lb = LabelBinarizer()
print("Label Binarizer:\n", lb.fit_transform(values))
```

```
Data:
  ['cold' 'cold' 'warm' 'cold' 'hot' 'hot' 'warm' 'cold' 'warm' 'hot']
Label Encoder:
 [0 0 2 0 1 1 2 0 2 1]
OneHot Encoder:
 [[1. 0. 0.]
 [1. 0. 0.]
 [0. 0. 1.]
 [1. 0. 0.]
```

```
 [0. 1. 0.]
 [0. 1. 0.]
 [0. 0. 1.]
 [1. 0. 0.]
 [0. 0. 1.]
 [0. 1. 0.]]
Label Binarizer:
 [[1 0 0]
 [1 0 0]
 [0 0 1]
 [1 0 0]
 [0 1 0]
 [0 1 0]
 [0 0 1]
 [1 0 0]
```


4. Discretization

Data discretization is the process of converting continuous data into discrete buckets by grouping it. Discretization is also known for easy maintainability of the data. Training a model with discrete data becomes faster and more effective than when attempting the same with continuous data. Although continuous-valued data contains more information, huge amounts of data can slow the model down. Here, discretization can help us strike a balance between both. Some famous methods of data discretization are binning and using a histogram. Although data discretization is useful, we need to effectively pick the range of each bucket, which is a challenge.

The main challenge in discretization is to choose the number of intervals or bins and how to decide on their width.

Here we make use of a function called **pandas.cut()**. This function is useful to achieve the bucketing and sorting of segmented data.

```
df_boston['binned']=pd.cut(x=df_boston['AGE'], bins=[0,18,36,54,72,100],labels = [0, 1, 2,3,4])
df_boston.head()
```

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTAT	target	binned
0	0.00632	18.0	2.31	0.0	0.538	6.575	65.2	4.0900	1.0	296.0	15.3	396.90	4.98	24.0	3
1	0.02731	0.0	7.07	0.0	0.469	6.421	78.9	4.9671	2.0	242.0	17.8	396.90	9.14	21.6	4
2	0.02729	0.0	7.07	0.0	0.469	7.185	61.1	4.9671	2.0	242.0	17.8	392.83	4.03	34.7	3
3	0.03237	0.0	2.18	0.0	0.458	6.998	45.8	6.0622	3.0	222.0	18.7	394.63	2.94	33.4	2
4	0.06905	0.0	2.18	0.0	0.458	7.147	54.2	6.0622	3.0	222.0	18.7	396.90	5.33	36.2	3

5. Feature Selection

5.1 Removing features with low variance

VarianceThreshold is a simple baseline approach to feature selection. It removes all features whose variance doesn't meet some threshold. By default, it removes all zero-variance features, i.e. features that have the same value in all samples. threshold=(.8 * (1 - .8) means if variance is 0 for 80% of samples then those values will be removed.

```
from sklearn.feature_selection import VarianceThreshold
#X = [[0, 0, 1], [0, 1, 0], [1, 0, 0], [0, 1, 1], [0, 1, 0], [0, 1, 1]]
sel = VarianceThreshold(threshold=(.8 * (1 - .8)))
fsel_data = sel.fit_transform(df_boston)
fsel_df = pd.DataFrame(fsel_data)
fsel_df.head()
```

	0	1	2	3	4	5	6	7	8	9	10	11	12
0	0.00632	18.0	2.31	6.575	65.2	4.0900	1.0	296.0	15.3	396.90	4.98	24.0	3.0
1	0.02731	0.0	7.07	6.421	78.9	4.9671	2.0	242.0	17.8	396.90	9.14	21.6	4.0
2	0.02729	0.0	7.07	7.185	61.1	4.9671	2.0	242.0	17.8	392.83	4.03	34.7	3.0
3	0.03237	0.0	2.18	6.998	45.8	6.0622	3.0	222.0	18.7	394.63	2.94	33.4	2.0
4	0.06905	0.0	2.18	7.147	54.2	6.0622	3.0	222.0	18.7	396.90	5.33	36.2	3.0

```
df_boston.head()
```

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTAT	target	binned
0	0.00632	18.0	2.31	0.0	0.538	6.575	65.2	4.0900	1.0	296.0	15.3	396.90	4.98	24.0	3
1	0.02731	0.0	7.07	0.0	0.469	6.421	78.9	4.9671	2.0	242.0	17.8	396.90	9.14	21.6	4
2	0.02729	0.0	7.07	0.0	0.469	7.185	61.1	4.9671	2.0	242.0	17.8	392.83	4.03	34.7	3
3	0.03237	0.0	2.18	0.0	0.458	6.998	45.8	6.0622	3.0	222.0	18.7	394.63	2.94	33.4	2
4	0.06905	0.0	2.18	0.0	0.458	7.147	54.2	6.0622	3.0	222.0	18.7	396.90	5.33	36.2	3

s had variance 0.

6. Feature Extraction (LDA, PCA)

6.1 PCA

```
# Here we are using inbuilt dataset of scikit learn
from sklearn.datasets import load_breast_cancer

# instantiating
cancer = load_breast_cancer()

# creating dataframe
df = pd.DataFrame(cancer['data'], columns = cancer['feature_names'])

# checking head of dataframe
df.head()
```

	mean radius	mean texture	mean perimeter	mean area	mean smoothness	mean compactness	mean concavity	mean concave points	mean symmetry	mean fractal dimension	...	worst radius	worst texture	wo perime
0	17.99	10.38	122.80	1001.0	0.11840	0.27760	0.3001	0.14710	0.2419	0.07871	...	25.38	17.33	18
1	20.57	17.77	132.90	1326.0	0.08474	0.07864	0.0869	0.07017	0.1812	0.05667	...	24.99	23.41	15
2	19.69	21.25	130.00	1203.0	0.10960	0.15990	0.1974	0.12790	0.2069	0.05999	...	23.57	25.53	15
3	11.42	20.38	77.58	386.1	0.14250	0.28390	0.2414	0.10520	0.2597	0.09744	...	14.91	26.50	9
4	20.29	14.34	135.10	1297.0	0.10030	0.13280	0.1980	0.10430	0.1809	0.05883	...	22.54	16.67	15

5 rows × 30 columns

```
# Importing standardscalar module
from sklearn.preprocessing import StandardScaler

scalar = StandardScaler()

# fitting
scalar.fit(df)
scaled_data = scalar.transform(df)

# Importing PCA
from sklearn.decomposition import PCA

# Let's say, components = 2
pca = PCA(n_components = 2)
pca.fit(scaled_data)
x_pca = pca.transform(scaled_data)

x_pca.shape
```

```
(569, 2)
```

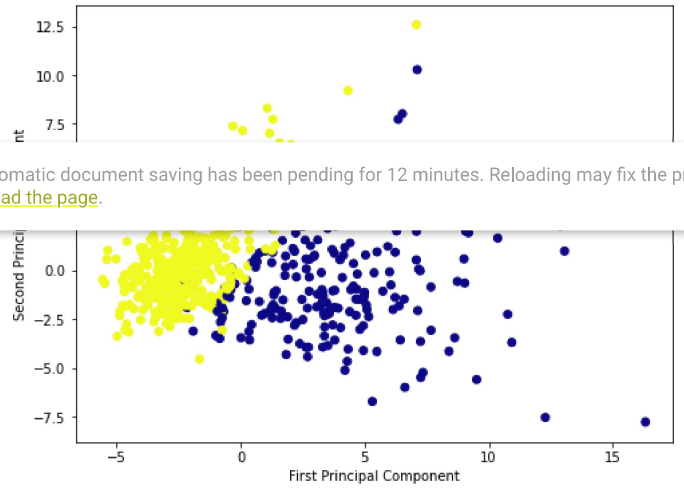
```
from matplotlib import pyplot as plt
# giving a larger plot
```

```
plt.figure(figsize =(8, 6))

plt.scatter(x_pca[:, 0], x_pca[:, 1], c = cancer['target'], cmap ='plasma')

# labeling x and y axes
plt.xlabel('First Principal Component')
plt.ylabel('Second Principal Component')
```

```
Text(0, 0.5, 'Second Principal Component')
```




```
pca.components_
```

```
array([[ 0.21890244,  0.10372458,  0.22753729,  0.22099499,  0.14258969,
         0.23928535,  0.25840048,  0.26085376,  0.13816696,  0.06436335,
         0.20597878,  0.01742803,  0.21132592,  0.20286964,  0.01453145,
         0.17039345,  0.15358979,  0.1834174 ,  0.04249842,  0.10256832,
         0.22799663,  0.10446933,  0.23663968,  0.22487053,  0.12795256,
         0.21009588,  0.22876753,  0.25088597,  0.12290456,  0.13178394],
       [-0.23385713, -0.05970609, -0.21518136, -0.23107671,  0.18611302,
         0.15189161,  0.06016536, -0.0347675 ,  0.19034877,  0.36657547,
        -0.10555215,  0.08997968, -0.08945723, -0.15229263,  0.20443045,
         0.2327159 ,  0.19720728,  0.13032156,  0.183848  ,  0.28009203,
        -0.21986638, -0.0454673 , -0.19987843, -0.21935186,  0.17230435,
         0.14359317,  0.09796411, -0.00825724,  0.14188335,  0.27533947]])
```

Q: Predict Sepal length in Iris dataset

```
#loading dataset
from sklearn.datasets import load_iris
data = load_iris()
x = data['data']
#separation of data into x and y
y = x[:,0]
x = x[:,1:4]
```

Prediction using SKLEARN

```
#implementation using sklearn
#splitting dataset into train set and test set
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(x,y,test_size = 0.25,random_state=0)
#creating model
from sklearn.linear_model import LinearRegression
regressor = LinearRegression()
regressor.fit(x_train, y_train)
y_pred = regressor.predict(x_test)


from sklearn.metrics import r2_score
r2score = r2_score(y_test,y_pred)
print("R2Score",r2score*100)
```

```
R2Score 72.60811953040603
```

Prediction using equation

$Y = \beta X$, where

$\beta = (X^T X)^{-1} X^T Y$

```
import numpy as np
x = x_test
x_T = x.transpose()
x_T_x = x_T.dot(x)
x_T_x_inv = np.linalg.inv(x_T_x)
beta = x_T_x_inv.dot(x_T).dot(y_test)
print(beta)
```



```
[ 1.11833096  0.87349176 -0.81477536]
[5.63067381 5.1395198  5.75692343 7.27956223 4.94960784 6.89450434
 4.80526505 6.35007422 6.18340164 5.56608577 6.65852885 6.28720896
 6.25900753 5.92722575 6.20788556 5.16740239 5.95170967 5.77329381
 4.33344874 5.1711199  5.7818856  6.06354277 5.29900454 4.30309318
 5.85749149 4.73652815 5.58338186 5.93996639 4.63990866 4.87400194
 6.80443502 6.06354277 5.06144094 6.16850686 6.23037476 5.28542596
 5.49016104 6.170661  ]
```

```
from sklearn.metrics import r2_score
r2score1 = r2_score(y_test,beta_x)
print("R2Score",r2score1*100)
```

```
R2Score 70.99401512456086
```

```
plt.plot(np.arange(len(beta_x)),beta_x)
plt.plot(np.arange(len(x_test)),y_pred)
plt.plot(np.arange(len(x_test)),y_test)
plt.legend(['Y Calculated', 'Y Predicted','Actual Y'])
plt.title('Y_pred vs Y_calculated vs Actual Y')
plt.show()
```