Machine Learning Lab - 2A
Feature scaling

Standardization is a data preprocessing technique that modifies the range of features in a dataset by transforming them to have a specified mean and standard deviation. This ensures that all features contribute equally to the analysis, which is crucial for algorithms that are sensitive to the scale of data. Apply various transformations and appropriate feature scaling techniques to the Titanic dataset, and preprocess the data based on the classification tasks outlined in the previous week's assignment.

Code -

```
# Sex Label Encoding
# male -> 0
# female -> 1

# Manual
# data["Sex"] = data["Sex"].map(lambda ele : 0 if ele == 'male' else 1)
# data.head(10)

from sklearn import preprocessing

# Using Label Encoder
# labelEncoder = preprocessing.LabelEncoder()
# data["Sex"] = labelEncoder.fit_transform(data["Sex"])
# data.head(15)

# Using One hot Encoder
oneHotEncoder = preprocessing.OneHotEncoder(sparse_output=False)
encodedData = oneHotEncoder.fit_transform(data[["Sex"]])
encodedData

encodedData = pd.DataFrame(encodedData,
columns=oneHotEncoder.get_feature_names_out(["Sex"]))
encodedData

data = pd.concat([data, encodedData], axis=1)
data = data.drop(['Sex'], axis=1)
data
```

Udit Jain
24CSM1R23

Output

	PassengerId	Survived	Pclass	Name	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked	HasCabin	Sex_female	Sex_male
0	892	0	3	Kelly, Mr. James	34.5	0	0	330911	7.8292	NaN	Q	0	0.0	1.0
1	893	1	3	Wilkes, Mrs. James (Ellen Needs)	47.0	1	0	363272	7.0000	NaN	S	0	1.0	0.0
2	894	0	2	Myles, Mr. Thomas Francis	62.0	0	0	240276	9.6875	NaN	Q	0	0.0	1.0
3	895	0	3	Wirz, Mr. Albert	27.0	0	0	315154	8.6625	NaN	S	0	0.0	1.0
4	896	1	3	Hirvonen, Mrs. Alexander (Helga E Lindqvist)	22.0	1	1	3101298	12.2875	NaN	S	0	1.0	0.0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
413	1305	0	3	Spector, Mr. Woolf	30.0	0	0	A.5. 3236	8.0500	NaN	S	0	0.0	1.0
414	1306	1	1	Oliva y Ocana, Dona. Fermina	39.0	0	0	PC 17758	108.9000	C105	C	1	1.0	0.0
415	1307	0	3	Saether, Mr. Simon Sivertsen	38.5	0	0	SOTON/O.Q. 3101262	7.2500	NaN	S	0	0.0	1.0
416	1308	0	3	Ware, Mr. Frederick	30.0	0	0	359309	8.0500	NaN	S	0	0.0	1.0
417	1309	0	3	Peter, Master. Michael J	30.0	1	1	2668	22.3583	NaN	C	0	0.0	1.0

```
# Embarked Label Encoding

# Manual
# C -> 0, Q -> 1, s -> 2
# data["Embarked"] = data["Embarked"].map({ 'C' : 0, 'Q' : 1, 'S' : 2 })
# data.head(10)

# Using Label Encoder
labelEncoder = preprocessing.LabelEncoder()
data["Embarked"] = labelEncoder.fit_transform(data["Embarked"])
data.head(15)
```

Output -

	PassengerId	Survived	Pclass	Name	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked	HasCabin	Sex_female	Sex_male
0	892	0	3	Kelly, Mr. James	34.5	0	0	330911	7.8292	NaN	1	0	0.0	1.0
1	893	1	3	Wilkes, Mrs. James (Ellen Needs)	47.0	1	0	363272	7.0000	NaN	2	0	1.0	0.0
2	894	0	2	Myles, Mr. Thomas Francis	62.0	0	0	240276	9.6875	NaN	1	0	0.0	1.0
3	895	0	3	Wirz, Mr. Albert	27.0	0	0	315154	8.6625	NaN	2	0	0.0	1.0
4	896	1	3	Hirvonen, Mrs. Alexander (Helga E Lindqvist)	22.0	1	1	3101298	12.2875	NaN	2	0	1.0	0.0
5	897	0	3	Svensson, Mr. Johan Cervin	14.0	0	0	7538	9.2250	NaN	2	0	0.0	1.0
6	898	1	3	Connolly, Miss. Kate	30.0	0	0	330972	7.6292	NaN	1	0	1.0	0.0
7	899	0	2	Caldwell, Mr. Albert Francis	26.0	1	1	248738	29.0000	NaN	2	0	0.0	1.0
8	900	1	3	Abrahim, Mrs. Joseph (Sophie Halaut Easu)	18.0	0	0	2657	7.2292	NaN	0	0	1.0	0.0
9	901	0	3	Davies, Mr. John Samuel	21.0	2	0	A/4 48871	24.1500	NaN	2	0	0.0	1.0
10	902	0	3	Ilieff, Mr. Ylio	30.0	0	0	349220	7.8958	NaN	2	0	0.0	1.0
11	903	0	1	Jones, Mr. Charles Cresson	46.0	0	0	694	26.0000	NaN	2	0	0.0	1.0
12	904	1	1	Snyder, Mrs. John Pillsbury (Nelle Stevenson)	23.0	1	0	21228	82.2667	B45	2	1	1.0	0.0

Udit Jain
24CSM1R23

```
data.describe()
```

Output -

	PassengerId	Survived	Pclass	Name	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked	HasCabin	Sex_female
count	418.000000	418.000000	418.000000	418.000000	418.000000	418.000000	418.000000	418.000000	418.000000	418.000000	418.000000	418.000000	418.000000
mean	1100.500000	0.363636	2.265550	208.500000	30.216507	0.447368	0.392344	173.497608	35.625688	7.827751	1.401914	0.217703	0.363636
std	120.810458	0.481622	0.841838	120.810458	12.635016	0.896760	0.981429	103.461117	55.840509	18.053850	0.854496	0.413179	0.481622
min	892.000000	0.000000	1.000000	0.000000	0.170000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
25%	996.250000	0.000000	1.000000	104.250000	23.000000	0.000000	0.000000	83.250000	7.895800	0.000000	1.000000	0.000000	0.000000
50%	1100.500000	0.000000	3.000000	208.500000	30.000000	0.000000	0.000000	170.500000	14.454200	0.000000	2.000000	0.000000	0.000000
75%	1204.750000	1.000000	3.000000	312.750000	35.750000	1.000000	0.000000	260.750000	31.500000	0.000000	2.000000	0.000000	1.000000
max	1309.000000	1.000000	3.000000	417.000000	76.000000	8.000000	9.000000	362.000000	512.329200	76.000000	2.000000	1.000000	1.000000

```
from scipy.stats import skew
skewness = data.apply(lambda x: skew(x))
skewness
```

Output -

PassengerId	0.000000
Survived	0.566947
Pclass	-0.532252
Name	0.000000
Age	0.524141
SibSp	4.153363
Parch	4.637742
Ticket	0.094916
Fare	3.678418
Cabin	2.316255
Embarked	-0.870686
HasCabin	1.368099
Sex_female	0.566947
Sex_male	-0.566947
FamilyStatus	0.430706
HighFare	1.269433
AgeGroup	-0.917140

Udit Jain
24CSM1R23

```
# Scaling
from sklearn import preprocessing
import numpy as np
columns = data.columns

# # 1. Z Score

# Manually
import statistics

meanPassengerId = data["PassengerId"].mean()
stDevPassengerId = statistics.stdev(data["PassengerId"])

def getNewPID(ele):
  return float(ele - meanPassengerId) / float(stDevPassengerId)

passengerIdNew = data["PassengerId"].apply(lambda ele : getNewPID(ele))

# Using Library
scaler = preprocessing.StandardScaler()
data = scaler.fit_transform(data)
data = pd.DataFrame(data, columns=columns)

for i in range(0, len(data["PassengerId"])):
  print(passengerIdNew[i], data["PassengerId"][i], sep=" - ")
```

Output -

Udit Jain
24CSM1R23

```
-1.7258439719031378 - -1.7279120900470566
-1.7175665427813 - -1.7196247418933537
-1.7092891136594626 - -1.7113373937396508
-1.7010116845376249 - -1.7030500455859479
-1.6927342554157874 - -1.694762697432245
-1.6844568262939497 - -1.686475349278542
-1.6761793971721122 - -1.6781880011248391
-1.6679019680502745 - -1.6699006529711362
-1.659624538928437 - -1.6616133048174333
-1.6513471098065993 - -1.6533259566637304
-1.6430696806847618 - -1.6450386085100275
-1.6347922515629243 - -1.6267512603563246
-1.6265148224410866 - -1.6284639122026217
-1.618237393319249 - -1.6201765640489187
-1.6099599641974114 - -1.6118892158952158
-1.601682535075574 - -1.603601867741513
-1.5934051059537362 - -1.59531451958781
-1.5851276768318987 - -1.587027171434107
-1.576850247710061 - -1.5787398232804042
-1.5685728185882235 - -1.5704524751267013
-1.5602953894663858 - -1.5621651269729984
-1.5520179603445483 - -1.5538777788192955
-1.5437405312227106 - -1.5455904306655925
-1.5354631021008731 - -1.5373030825118896
-1.5271856729790354 - -1.5290157343581867
-1.518908243857198 - -1.5207283862044838
-1.5106308147353604 - -1.512441038050781
-1.5023533856135227 - -1.504153689897078
-1.4940759564916852 - -1.495866341743375
```

```python
# 2. Min-Max Scaler

# Manually
minFare = data["Fare"].min()
maxFare = data["Fare"].max()

def getNewFare(ele):
  return float(ele - minFare) / float(maxFare - minFare)

fareNew = data["Fare"].apply(lambda ele : getNewFare(ele))
fareNew

# Using Library
scaler = preprocessing.MinMaxScaler(feature_range=(0, 1))
data = scaler.fit_transform(data)
data = pd.DataFrame(data, columns=columns)

for i in range(0, len(data["Fare"])):
  print(fareNew[i], data["Fare"][i], sep=" - ")
```

Udit Jain
24CSM1R23

Output -

```
0.015281580671177828 - 0.015281580671177828
0.013663090060062943 - 0.013663090060062943
0.018908740708122825 - 0.018908740708122825
0.016908073949327893 - 0.016908073949327893
0.023983602730431916 - 0.023983602730431916
0.01800600082915438 - 0.01800600082915438
0.014891206669461744 - 0.014891206669461744
0.056604230248832196 - 0.056604230248832196
0.014110458666029575 - 0.014110458666029575
0.04713766070721715 - 0.04713766070721715
0.015411575213749284 - 0.015411575213749286
0.050748620223090936 - 0.050748620223090936
0.16057390443488287 - 0.16057390443488287
0.050748620223090936 - 0.050748620223090936
0.11940564777490723 - 0.11940564777490723
0.05410739813385612 - 0.05410739813385612
0.024105594605968193 - 0.024105594605968193
0.014102260811993537 - 0.014102260811993537
0.015468569817999833 - 0.015468569817999833
0.014102260811993537 - 0.014102260811993537
0.11594107850967697 - 0.11594107850967697
0.006188989423206797 - 0.0061889894232067975
0.06184168304285603 - 0.06184168304285603
0.11980421963065935 - 0.11980421963065935
0.5121218935012878 - 0.5121218935012878
0.028302115124416098 - 0.028302115124416098
0.1209753416358076 - 0.1209753416358076
0.014102260811993537 - 0.014102260811993537
0.05953203526170282 - 0.05953203526170282
0.04231498029001666 - 0.04231498029001666
0.050748620223090936 - 0.050748620223090936
```

```python
# 3. Max Absolute Scaling

# Manually
maxFare = data["Fare"].max()

def getNewFare(ele):
  return float(ele) / float(maxFare)

fareNew = data["Fare"].apply(lambda ele : getNewFare(ele))
fareNew

# Using Library
# scaler = preprocessing.MaxAbsScaler()
data = scaler.fit_transform(data)
data = pd.DataFrame(data, columns=columns)
```

Udit Jain
24CSM1R23

```
for i in range(0, len(data["Fare"])):
  print(fareNew[i], data["Fare"][i], sep=" - ")
```

Output -

```
0.015281580671177828 - 0.015281580671177828
0.013663090060062943 - 0.013663090060062943
0.018908740708122825 - 0.018908740708122825
0.016908073949327893 - 0.016908073949327893
0.023983602730431916 - 0.023983602730431916
0.01800600082915438 - 0.01800600082915438
0.014891206669461744 - 0.014891206669461744
0.056604230248832196 - 0.056604230248832196
0.014110458666029575 - 0.014110458666029575
0.04713766070721715 - 0.04713766070721715
0.015411575213749284 - 0.015411575213749286
0.050748620223090936 - 0.050748620223090936
0.16057390443488287 - 0.16057390443488287
0.050748620223090936 - 0.050748620223090936
0.11940564777490723 - 0.11940564777490723
0.05410739813385612 - 0.05410739813385612
0.024105594605968193 - 0.024105594605968193
0.014102260811993537 - 0.014102260811993537
0.015468569817999833 - 0.015468569817999833
0.014102260811993537 - 0.014102260811993537
0.11594107850967697 - 0.11594107850967697
0.006188989423206797 - 0.0061889894232067975
0.06184168304285603 - 0.06184168304285603
0.11980421963065935 - 0.11980421963065935
0.5121218935012878 - 0.5121218935012878
0.028302115124416098 - 0.028302115124416098
0.1209753416358076 - 0.1209753416358076
0.014102260811993537 - 0.014102260811993537
0.05953203526170282 - 0.05953203526170282
0.04231498029001666 - 0.04231498029001666
0.050748620223090936 - 0.050748620223090936
0.061483905270283246 - 0.061483905270283246
0.04015972542654215 - 0.04015972542654215
0.04577135170121086 - 0.04577135170121086
0.11272049299551928 - 0.11272049299551928
0.014110458666029575 - 0.014110458666029575
0.015712553569072387 - 0.015712553569072387
0.016908073949327893 - 0.016908073949327893
```

```
# 4. Robust Scaling

# Manually
medianPassengerId = data["PassengerId"].median()

q1 = np.percentile(data["PassengerId"], 25)
q3 = np.percentile(data["PassengerId"], 75)
iqr = q3 - q1
```

Udit Jain
24CSM1R23

```
def getNewPID(ele):
    return float(ele - medianPassengerId) / float(iqr)

passengerIdNew = data["PassengerId"].apply(lambda ele : getNewPID(ele))

# Using Library
scaler = preprocessing.RobustScaler()
data = scaler.fit_transform(data)
data = pd.DataFrame(data, columns=columns)

for i in range(len(data["PassengerId"])):
  print(passengerIdNew[i], data["PassengerId"][i], sep=" - ")
```

Output -

```
-1.0 - -1.0
-0.9952038369304557 - -0.9952038369304557
-0.9904076738609112 - -0.9904076738609112
-0.9856115107913669 - -0.9856115107913669
-0.9808153477218226 - -0.9808153477218226
-0.9760191846522782 - -0.9760191846522782
-0.9712230215827338 - -0.9712230215827338
-0.9664268585131894 - -0.9664268585131894
-0.9616306954436451 - -0.9616306954436451
-0.9568345323741008 - -0.9568345323741008
-0.9520383693045563 - -0.9520383693045563
-0.947242206235012 - -0.947242206235012
-0.9424460431654677 - -0.9424460431654677
-0.9376498800959233 - -0.9376498800959233
-0.9328537170263789 - -0.9328537170263789
-0.9280575539568345 - -0.9280575539568345
-0.9232613908872902 - -0.9232613908872902
-0.9184652278177458 - -0.9184652278177458
-0.9136690647482014 - -0.9136690647482014
-0.9088729016786571 - -0.9088729016786571
-0.9040767386091128 - -0.9040767386091128
-0.8992805755395683 - -0.8992805755395683
-0.894484412470024 - -0.894484412470024
-0.8896882494004796 - -0.8896882494004796
-0.8848920863309353 - -0.8848920863309353
-0.8800959232613909 - -0.8800959232613909
-0.8752997601918465 - -0.8752997601918465
-0.8705035971223022 - -0.8705035971223022
-0.8657074340527577 - -0.8657074340527577
-0.8609112709832134 - -0.8609112709832134
-0.8561151079136691 - -0.8561151079136691
-0.8513189448441247 - -0.8513189448441247
-0.8465227817745803 - -0.8465227817745803
-0.841726618705036 - -0.841726618705036
```

Udit Jain
24CSM1R23

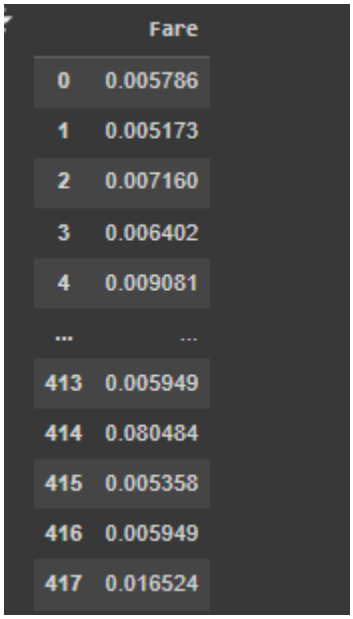
```python
# 5. Unit Vactor Scaling

# Manually
import math

dis = 0
for ele in data["Fare"]:
  dis += (ele * ele)
dis = float(math.sqrt(dis))

fareNew = data["Fare"].apply(lambda ele : float(ele) / float(dis))
fareNew
```

Output -



	Fare
0	0.005786
1	0.005173
2	0.007160
3	0.006402
4	0.009081
...	...
413	0.005949
414	0.080484
415	0.005358
416	0.005949
417	0.016524

Udit Jain
24CSM1R23