DATA MINING   
LAB Experiments

NAME: DEBATREYA DAS  
ROLL NO. 12212070  
CS A4

# Data Mining LAB : Experiment 2

Submitted By:

Name: Debatreya Das

Roll No. 12212070

CS A4

Data Mining LAB

Problem Statement

Download any two datasets of your choice comprising different types of attributes. Compute and analyze central tendency (mean, median and mode), dispersion (range, quartiles, interquartile range, variance and standard deviation) of different attributes and covariance and correlation matrix for the given datasets. Discuss your observations regarding which operation is logically apt for given attribute type and regarding characteristics of datasets which can be observed based on covariance and correlation matrix.

The objective is to understand the central tendencies and dispersion of the data attributes.

Data Set

**WINE**: (Link won’t work here)

**IRIS**: (Link wont work here)

Code and Output

Observations

No such observations (The objective is to understand the central tendencies and dispersions of the data attributes.)

Wine Notebook

Statistical Analysis: Wine Dataset

We will calculate the following statistics for each attribute

[] Central Tendancy

Mean

Median

Mode

[] Dispersion

Range

Quartiles

InterQuartile Range

Variance

Santard Deviation

[] Covariance Matrix

[] Correlation Matrix

Importing necessary Libraries

**import** pandas **as** pd  
**import** numpy **as** np  
**import** seaborn **as** sns  
**import** matplotlib.pyplot **as** plt

Load datasets

wine\_df = pd.read\_csv('../wine/wine.data', header=None)  
wine\_df.columns = [  
 "Class", "Alcohol", "Malic\_acid", "Ash", "Alcalinity\_of\_ash",   
 "Magnesium", "Total\_phenols", "Flavanoids", "Nonflavanoid\_phenols",  
 "Proanthocyanins", "Color\_intensity", "Hue", "OD280/OD315\_of\_diluted\_wines", "Proline"  
]  
wine\_df

Class Alcohol Malic\_acid Ash Alcalinity\_of\_ash Magnesium \  
0 1 14.23 1.71 2.43 15.6 127   
1 1 13.20 1.78 2.14 11.2 100   
2 1 13.16 2.36 2.67 18.6 101   
3 1 14.37 1.95 2.50 16.8 113   
4 1 13.24 2.59 2.87 21.0 118   
.. ... ... ... ... ... ...   
173 3 13.71 5.65 2.45 20.5 95   
174 3 13.40 3.91 2.48 23.0 102   
175 3 13.27 4.28 2.26 20.0 120   
176 3 13.17 2.59 2.37 20.0 120   
177 3 14.13 4.10 2.74 24.5 96   
  
 Total\_phenols Flavanoids Nonflavanoid\_phenols Proanthocyanins \  
0 2.80 3.06 0.28 2.29   
1 2.65 2.76 0.26 1.28   
2 2.80 3.24 0.30 2.81   
3 3.85 3.49 0.24 2.18   
4 2.80 2.69 0.39 1.82   
.. ... ... ... ...   
173 1.68 0.61 0.52 1.06   
174 1.80 0.75 0.43 1.41   
175 1.59 0.69 0.43 1.35   
176 1.65 0.68 0.53 1.46   
177 2.05 0.76 0.56 1.35   
  
 Color\_intensity Hue OD280/OD315\_of\_diluted\_wines Proline   
0 5.64 1.04 3.92 1065   
1 4.38 1.05 3.40 1050   
2 5.68 1.03 3.17 1185   
3 7.80 0.86 3.45 1480   
4 4.32 1.04 2.93 735   
.. ... ... ... ...   
173 7.70 0.64 1.74 740   
174 7.30 0.70 1.56 750   
175 10.20 0.59 1.56 835   
176 9.30 0.60 1.62 840   
177 9.20 0.61 1.60 560   
  
[178 rows x 14 columns]

Compute central tendency (mean, median, mode)

mean\_values = wine\_df.mean()  
median\_values = wine\_df.median()  
mode\_values = wine\_df.mode().iloc[0] *# mode() returns a dataframe, so we take the first row*  
  
*# Combine them into a single DataFrame*  
central\_tendency\_table = pd.DataFrame({  
 'Mean': mean\_values,  
 'Median': median\_values,  
 'Mode': mode\_values  
})  
  
*# Display the table*  
central\_tendency\_table

Mean Median Mode  
Class 1.938202 2.000 2.00  
Alcohol 13.000618 13.050 12.37  
Malic\_acid 2.336348 1.865 1.73  
Ash 2.366517 2.360 2.28  
Alcalinity\_of\_ash 19.494944 19.500 20.00  
Magnesium 99.741573 98.000 88.00  
Total\_phenols 2.295112 2.355 2.20  
Flavanoids 2.029270 2.135 2.65  
Nonflavanoid\_phenols 0.361854 0.340 0.26  
Proanthocyanins 1.590899 1.555 1.35  
Color\_intensity 5.058090 4.690 2.60  
Hue 0.957449 0.965 1.04  
OD280/OD315\_of\_diluted\_wines 2.611685 2.780 2.87  
Proline 746.893258 673.500 520.00

Measures of Dispersion (range, quartiles, InterQuartile Ranges, Variance, Standard Deviations)

wine\_dispersion = wine\_df.describe().T  
wine\_dispersion['IQR'] = wine\_dispersion['75%'] - wine\_dispersion['25%']  
wine\_dispersion['Variance'] = wine\_dispersion['std'] \*\* 2  
  
wine\_dispersion

count mean std min 25% \  
Class 178.0 1.938202 0.775035 1.00 1.0000   
Alcohol 178.0 13.000618 0.811827 11.03 12.3625   
Malic\_acid 178.0 2.336348 1.117146 0.74 1.6025   
Ash 178.0 2.366517 0.274344 1.36 2.2100   
Alcalinity\_of\_ash 178.0 19.494944 3.339564 10.60 17.2000   
Magnesium 178.0 99.741573 14.282484 70.00 88.0000   
Total\_phenols 178.0 2.295112 0.625851 0.98 1.7425   
Flavanoids 178.0 2.029270 0.998859 0.34 1.2050   
Nonflavanoid\_phenols 178.0 0.361854 0.124453 0.13 0.2700   
Proanthocyanins 178.0 1.590899 0.572359 0.41 1.2500   
Color\_intensity 178.0 5.058090 2.318286 1.28 3.2200   
Hue 178.0 0.957449 0.228572 0.48 0.7825   
OD280/OD315\_of\_diluted\_wines 178.0 2.611685 0.709990 1.27 1.9375   
Proline 178.0 746.893258 314.907474 278.00 500.5000   
  
 50% 75% max IQR \  
Class 2.000 3.0000 3.00 2.0000   
Alcohol 13.050 13.6775 14.83 1.3150   
Malic\_acid 1.865 3.0825 5.80 1.4800   
Ash 2.360 2.5575 3.23 0.3475   
Alcalinity\_of\_ash 19.500 21.5000 30.00 4.3000   
Magnesium 98.000 107.0000 162.00 19.0000   
Total\_phenols 2.355 2.8000 3.88 1.0575   
Flavanoids 2.135 2.8750 5.08 1.6700   
Nonflavanoid\_phenols 0.340 0.4375 0.66 0.1675   
Proanthocyanins 1.555 1.9500 3.58 0.7000   
Color\_intensity 4.690 6.2000 13.00 2.9800   
Hue 0.965 1.1200 1.71 0.3375   
OD280/OD315\_of\_diluted\_wines 2.780 3.1700 4.00 1.2325   
Proline 673.500 985.0000 1680.00 484.5000   
  
 Variance   
Class 0.600679   
Alcohol 0.659062   
Malic\_acid 1.248015   
Ash 0.075265   
Alcalinity\_of\_ash 11.152686   
Magnesium 203.989335   
Total\_phenols 0.391690   
Flavanoids 0.997719   
Nonflavanoid\_phenols 0.015489   
Proanthocyanins 0.327595   
Color\_intensity 5.374449   
Hue 0.052245   
OD280/OD315\_of\_diluted\_wines 0.504086   
Proline 99166.717355

Compute covariance and correlation matrices

wine\_covariance = wine\_df.cov()  
wine\_correlation = wine\_df.corr()

WINE Covariance Matrix

wine\_covariance

Class Alcohol Malic\_acid Ash \  
Class 0.600679 -0.206515 0.379039 -0.010555   
Alcohol -0.206515 0.659062 0.085611 0.047115   
Malic\_acid 0.379039 0.085611 1.248015 0.050277   
Ash -0.010555 0.047115 0.050277 0.075265   
Alcalinity\_of\_ash 1.340364 -0.841093 1.076332 0.406208   
Magnesium -2.315495 3.139878 -0.870780 1.122937   
Total\_phenols -0.348835 0.146887 -0.234338 0.022146   
Flavanoids -0.656091 0.192033 -0.458630 0.031535   
Nonflavanoid\_phenols 0.047177 -0.015754 0.040733 0.006358   
Proanthocyanins -0.221413 0.063518 -0.141147 0.001516   
Color\_intensity 0.477339 1.028283 0.644838 0.164654   
Hue -0.109368 -0.013313 -0.143326 -0.004682   
OD280/OD315\_of\_diluted\_wines -0.433737 0.041698 -0.292447 0.000762   
Proline -154.667651 164.567185 -67.548867 19.319739   
  
 Alcalinity\_of\_ash Magnesium Total\_phenols \  
Class 1.340364 -2.315495 -0.348835   
Alcohol -0.841093 3.139878 0.146887   
Malic\_acid 1.076332 -0.870780 -0.234338   
Ash 0.406208 1.122937 0.022146   
Alcalinity\_of\_ash 11.152686 -3.974760 -0.671149   
Magnesium -3.974760 203.989335 1.916470   
Total\_phenols -0.671149 1.916470 0.391690   
Flavanoids -1.172083 2.793087 0.540470   
Nonflavanoid\_phenols 0.150422 -0.455563 -0.035045   
Proanthocyanins -0.377176 1.932832 0.219373   
Color\_intensity 0.145024 6.620521 -0.079998   
Hue -0.209118 0.180851 0.062039   
OD280/OD315\_of\_diluted\_wines -0.656234 0.669308 0.311021   
Proline -463.355345 1769.158700 98.171057   
  
 Flavanoids Nonflavanoid\_phenols \  
Class -0.656091 0.047177   
Alcohol 0.192033 -0.015754   
Malic\_acid -0.458630 0.040733   
Ash 0.031535 0.006358   
Alcalinity\_of\_ash -1.172083 0.150422   
Magnesium 2.793087 -0.455563   
Total\_phenols 0.540470 -0.035045   
Flavanoids 0.997719 -0.066867   
Nonflavanoid\_phenols -0.066867 0.015489   
Proanthocyanins 0.373148 -0.026060   
Color\_intensity -0.399169 0.040121   
Hue 0.124082 -0.007471   
OD280/OD315\_of\_diluted\_wines 0.558262 -0.044469   
Proline 155.447492 -12.203586   
  
 Proanthocyanins Color\_intensity Hue \  
Class -0.221413 0.477339 -0.109368   
Alcohol 0.063518 1.028283 -0.013313   
Malic\_acid -0.141147 0.644838 -0.143326   
Ash 0.001516 0.164654 -0.004682   
Alcalinity\_of\_ash -0.377176 0.145024 -0.209118   
Magnesium 1.932832 6.620521 0.180851   
Total\_phenols 0.219373 -0.079998 0.062039   
Flavanoids 0.373148 -0.399169 0.124082   
Nonflavanoid\_phenols -0.026060 0.040121 -0.007471   
Proanthocyanins 0.327595 -0.033504 0.038665   
Color\_intensity -0.033504 5.374449 -0.276506   
Hue 0.038665 -0.276506 0.052245   
OD280/OD315\_of\_diluted\_wines 0.210933 -0.705813 0.091766   
Proline 59.554334 230.767480 17.000223   
  
 OD280/OD315\_of\_diluted\_wines Proline   
Class -0.433737 -154.667651   
Alcohol 0.041698 164.567185   
Malic\_acid -0.292447 -67.548867   
Ash 0.000762 19.319739   
Alcalinity\_of\_ash -0.656234 -463.355345   
Magnesium 0.669308 1769.158700   
Total\_phenols 0.311021 98.171057   
Flavanoids 0.558262 155.447492   
Nonflavanoid\_phenols -0.044469 -12.203586   
Proanthocyanins 0.210933 59.554334   
Color\_intensity -0.705813 230.767480   
Hue 0.091766 17.000223   
OD280/OD315\_of\_diluted\_wines 0.504086 69.927526   
Proline 69.927526 99166.717355

WINE Correlation Matrix

wine\_correlation

Class Alcohol Malic\_acid Ash \  
Class 1.000000 -0.328222 0.437776 -0.049643   
Alcohol -0.328222 1.000000 0.094397 0.211545   
Malic\_acid 0.437776 0.094397 1.000000 0.164045   
Ash -0.049643 0.211545 0.164045 1.000000   
Alcalinity\_of\_ash 0.517859 -0.310235 0.288500 0.443367   
Magnesium -0.209179 0.270798 -0.054575 0.286587   
Total\_phenols -0.719163 0.289101 -0.335167 0.128980   
Flavanoids -0.847498 0.236815 -0.411007 0.115077   
Nonflavanoid\_phenols 0.489109 -0.155929 0.292977 0.186230   
Proanthocyanins -0.499130 0.136698 -0.220746 0.009652   
Color\_intensity 0.265668 0.546364 0.248985 0.258887   
Hue -0.617369 -0.071747 -0.561296 -0.074667   
OD280/OD315\_of\_diluted\_wines -0.788230 0.072343 -0.368710 0.003911   
Proline -0.633717 0.643720 -0.192011 0.223626   
  
 Alcalinity\_of\_ash Magnesium Total\_phenols \  
Class 0.517859 -0.209179 -0.719163   
Alcohol -0.310235 0.270798 0.289101   
Malic\_acid 0.288500 -0.054575 -0.335167   
Ash 0.443367 0.286587 0.128980   
Alcalinity\_of\_ash 1.000000 -0.083333 -0.321113   
Magnesium -0.083333 1.000000 0.214401   
Total\_phenols -0.321113 0.214401 1.000000   
Flavanoids -0.351370 0.195784 0.864564   
Nonflavanoid\_phenols 0.361922 -0.256294 -0.449935   
Proanthocyanins -0.197327 0.236441 0.612413   
Color\_intensity 0.018732 0.199950 -0.055136   
Hue -0.273955 0.055398 0.433681   
OD280/OD315\_of\_diluted\_wines -0.276769 0.066004 0.699949   
Proline -0.440597 0.393351 0.498115   
  
 Flavanoids Nonflavanoid\_phenols \  
Class -0.847498 0.489109   
Alcohol 0.236815 -0.155929   
Malic\_acid -0.411007 0.292977   
Ash 0.115077 0.186230   
Alcalinity\_of\_ash -0.351370 0.361922   
Magnesium 0.195784 -0.256294   
Total\_phenols 0.864564 -0.449935   
Flavanoids 1.000000 -0.537900   
Nonflavanoid\_phenols -0.537900 1.000000   
Proanthocyanins 0.652692 -0.365845   
Color\_intensity -0.172379 0.139057   
Hue 0.543479 -0.262640   
OD280/OD315\_of\_diluted\_wines 0.787194 -0.503270   
Proline 0.494193 -0.311385   
  
 Proanthocyanins Color\_intensity Hue \  
Class -0.499130 0.265668 -0.617369   
Alcohol 0.136698 0.546364 -0.071747   
Malic\_acid -0.220746 0.248985 -0.561296   
Ash 0.009652 0.258887 -0.074667   
Alcalinity\_of\_ash -0.197327 0.018732 -0.273955   
Magnesium 0.236441 0.199950 0.055398   
Total\_phenols 0.612413 -0.055136 0.433681   
Flavanoids 0.652692 -0.172379 0.543479   
Nonflavanoid\_phenols -0.365845 0.139057 -0.262640   
Proanthocyanins 1.000000 -0.025250 0.295544   
Color\_intensity -0.025250 1.000000 -0.521813   
Hue 0.295544 -0.521813 1.000000   
OD280/OD315\_of\_diluted\_wines 0.519067 -0.428815 0.565468   
Proline 0.330417 0.316100 0.236183   
  
 OD280/OD315\_of\_diluted\_wines Proline   
Class -0.788230 -0.633717   
Alcohol 0.072343 0.643720   
Malic\_acid -0.368710 -0.192011   
Ash 0.003911 0.223626   
Alcalinity\_of\_ash -0.276769 -0.440597   
Magnesium 0.066004 0.393351   
Total\_phenols 0.699949 0.498115   
Flavanoids 0.787194 0.494193   
Nonflavanoid\_phenols -0.503270 -0.311385   
Proanthocyanins 0.519067 0.330417   
Color\_intensity -0.428815 0.316100   
Hue 0.565468 0.236183   
OD280/OD315\_of\_diluted\_wines 1.000000 0.312761   
Proline 0.312761 1.000000

Iris Notebook

Statistical Analysis: Iris Dataset

We will calculate the following statistics for each attribute

[] Central Tendancy

Mean

Median

Mode

[] Dispersion

Range

Quartiles

InterQuartile Range

Variance

Santard Deviation

[] Covariance Matrix

[] Correlation Matrix

Importing necessary Libraries

**import** pandas **as** pd

Load datasets

iris\_df = pd.read\_csv('../iris/iris.data', header=None)  
iris\_df.columns = [  
 "sepal\_length", "sepal\_width", "petal\_length", "petal\_width", "class"   
]  
iris\_df

sepal\_length sepal\_width petal\_length petal\_width class  
0 5.1 3.5 1.4 0.2 Iris-setosa  
1 4.9 3.0 1.4 0.2 Iris-setosa  
2 4.7 3.2 1.3 0.2 Iris-setosa  
3 4.6 3.1 1.5 0.2 Iris-setosa  
4 5.0 3.6 1.4 0.2 Iris-setosa  
.. ... ... ... ... ...  
145 6.7 3.0 5.2 2.3 Iris-virginica  
146 6.3 2.5 5.0 1.9 Iris-virginica  
147 6.5 3.0 5.2 2.0 Iris-virginica  
148 6.2 3.4 5.4 2.3 Iris-virginica  
149 5.9 3.0 5.1 1.8 Iris-virginica  
  
[150 rows x 5 columns]

Handle 'class' attribute which is a categorial data

*# Handling categorical data*  
  
*# Iris-setosa -> 0*  
iris\_df.loc[iris\_df['class'] == 'Iris-setosa', 'class'] = 0  
*# Iris-versicolor -> 1*  
iris\_df.loc[iris\_df['class'] == 'Iris-versicolor', 'class'] = 1  
*# Iris-virginica -> 2*  
iris\_df.loc[iris\_df['class'] == 'Iris-virginica', 'class'] = 2  
  
iris\_df

sepal\_length sepal\_width petal\_length petal\_width class  
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  
.. ... ... ... ... ...  
145 6.7 3.0 5.2 2.3 2  
146 6.3 2.5 5.0 1.9 2  
147 6.5 3.0 5.2 2.0 2  
148 6.2 3.4 5.4 2.3 2  
149 5.9 3.0 5.1 1.8 2  
  
[150 rows x 5 columns]

Compute central tendency (mean, median, mode)

mean\_values = iris\_df.mean()  
median\_values = iris\_df.median()  
mode\_values = iris\_df.mode().iloc[0] *# mode() returns a dataframe, so we take the first row*  
  
*# Combine them into a single DataFrame*  
central\_tendency\_table = pd.DataFrame({  
 'Mean': mean\_values,  
 'Median': median\_values,  
 'Mode': mode\_values  
})  
  
*# Display the table*  
central\_tendency\_table

Mean Median Mode  
sepal\_length 5.843333 5.8 5.0  
sepal\_width 3.054 3.0 3.0  
petal\_length 3.758667 4.35 1.5  
petal\_width 1.198667 1.3 0.2  
class 1.0 1.0 0

Measures of Dispersion (range, quartiles, InterQuartile Ranges, Variance, Standard Deviations)

iris\_dispersion = iris\_df.describe().T  
iris\_dispersion['IQR'] = iris\_dispersion['75%'] - iris\_dispersion['25%']  
iris\_dispersion['Variance'] = iris\_dispersion['std'] \*\* 2  
  
iris\_dispersion

count mean std min 25% 50% 75% max IQR \  
sepal\_length 150.0 5.843333 0.828066 4.3 5.1 5.80 6.4 7.9 1.3   
sepal\_width 150.0 3.054000 0.433594 2.0 2.8 3.00 3.3 4.4 0.5   
petal\_length 150.0 3.758667 1.764420 1.0 1.6 4.35 5.1 6.9 3.5   
petal\_width 150.0 1.198667 0.763161 0.1 0.3 1.30 1.8 2.5 1.5   
  
 Variance   
sepal\_length 0.685694   
sepal\_width 0.188004   
petal\_length 3.113179   
petal\_width 0.582414

Compute covariance and correlation matrices

iris\_covariance = iris\_df.cov()  
iris\_correlation = iris\_df.corr()

IRIS Covariance Matrix

iris\_covariance

sepal\_length sepal\_width petal\_length petal\_width class  
sepal\_length 0.685694 -0.039268 1.273682 0.516904 0.530872  
sepal\_width -0.039268 0.188004 -0.321713 -0.117981 -0.148993  
petal\_length 1.273682 -0.321713 3.113179 1.296387 1.371812  
petal\_width 0.516904 -0.117981 1.296387 0.582414 0.597987  
class 0.530872 -0.148993 1.371812 0.597987 0.671141

IRIS Correlation Matrix

iris\_correlation

sepal\_length sepal\_width petal\_length petal\_width class  
sepal\_length 1.000000 -0.109369 0.871754 0.817954 0.782561  
sepal\_width -0.109369 1.000000 -0.420516 -0.356544 -0.419446  
petal\_length 0.871754 -0.420516 1.000000 0.962757 0.949043  
petal\_width 0.817954 -0.356544 0.962757 1.000000 0.956464  
class 0.782561 -0.419446 0.949043 0.956464 1.000000

# Data Mining LAB : Experiment 3

Submitted By:

Name: Debatreya Das

Roll No. 12212070

CS A4

Data Mining LAB

Problem Statement

Select two publically available datasets comprising different types of attributes viz. nominal, ordinal, interval-scaled and ratio-scaled. **A dataset must comprise minimally 2 different types of attribute.**

Compute the

proximity (similarity and/or dissimilarity) between data points using following metrics:

Simple Matching Coefficient,

Jaccard Coefficient,

Cosine Similarity,

Euclidean Distance,

Manhattan Distance,

Supremum Distance, and

Correlation as similarity metric.

Initially consider each attribute individually for populating corresponding proximity matrix then consider each data object as represented by a vector of mixed attribute types and compute the proximity matrix for your dataset.

Discuss your observation regarding applicability of different metric and any pattern prevailing in your data.

Datasets used

**ADULT**: (Links Won't Work)

**TITANIC**: (Links Won’t Work)

Code and Output

**Jupyter Notebooks**

Observation and Discussion

After computing the proximity metrics:

Simple Matching & Jaccard Coefficients are particularly useful for nominal attributes. SMC includes all matches, while Jaccard focuses only on the presence/absence match.

Euclidean & Manhattan Distances are more sensitive to differences in scale, which is why standardization might be necessary before applying them to attributes with different scales.

Cosine Similarity is useful when the magnitude of the data points is irrelevant, focusing purely on the direction of the data points.

Supremum Distance is useful when interested in the maximum difference among dimensions, often highlighting outliers.

Correlation is apt for understanding linear relationships between attributes, especially in interval or ratio-scaled data.

Conclusion

Each proximity metric has its own strengths and is applicable in different contexts based on the type of attributes and the specific analysis goal. Mixed-type datasets, like those in the experiment, require careful selection of proximity metrics to ensure meaningful comparisons.

Simple Matching Coefficient

Simple Matching Coefficient (SMC)

It is typically used for comparing binary or nominal attributes.

Importing required Libraries

**import** pandas **as** pd  
**import** numpy **as** np  
**from** sklearn.preprocessing **import** LabelEncoder

Load Datasets

*# Load datasets*  
adult\_df = pd.read\_csv("../adult/adult\_trim.data", header=None) *# No header*  
titanic\_df = pd.read\_csv('../titanic/train.csv') *# Has header*  
  
*# Rename columns for clarity*  
adult\_df.columns = ["age", "workclass", "fnlwgt", "education", "education\_num",   
 "marital\_status", "occupation", "relationship", "race", "sex",   
 "capital\_gain", "capital\_loss", "hours\_per\_week", "native\_country", "income"]  
adult\_df.dropna(inplace=True)

adult\_df

age workclass fnlwgt education education\_num \  
0 39 State-gov 77516 Bachelors 13   
1 50 Self-emp-not-inc 83311 Bachelors 13   
2 38 Private 215646 HS-grad 9   
3 53 Private 234721 11th 7   
4 28 Private 338409 Bachelors 13   
.. ... ... ... ... ...   
95 29 Local-gov 115585 Some-college 10   
96 48 Self-emp-not-inc 191277 Doctorate 16   
97 37 Private 202683 Some-college 10   
98 48 Private 171095 Assoc-acdm 12   
99 32 Federal-gov 249409 HS-grad 9   
  
 marital\_status occupation relationship race sex \  
0 Never-married Adm-clerical Not-in-family White Male   
1 Married-civ-spouse Exec-managerial Husband White Male   
2 Divorced Handlers-cleaners Not-in-family White Male   
3 Married-civ-spouse Handlers-cleaners Husband Black Male   
4 Married-civ-spouse Prof-specialty Wife Black Female   
.. ... ... ... ... ...   
95 Never-married Handlers-cleaners Not-in-family White Male   
96 Married-civ-spouse Prof-specialty Husband White Male   
97 Married-civ-spouse Sales Husband White Male   
98 Divorced Exec-managerial Unmarried White Female   
99 Never-married Other-service Own-child Black Male   
  
 capital\_gain capital\_loss hours\_per\_week native\_country income   
0 2174 0 40 United-States <=50K   
1 0 0 13 United-States <=50K   
2 0 0 40 United-States <=50K   
3 0 0 40 United-States <=50K   
4 0 0 40 Cuba <=50K   
.. ... ... ... ... ...   
95 0 0 50 United-States <=50K   
96 0 1902 60 United-States >50K   
97 0 0 48 United-States >50K   
98 0 0 40 England <=50K   
99 0 0 40 United-States <=50K   
  
[100 rows x 15 columns]

titanic\_df

PassengerId Survived Pclass \  
0 1 0 3   
1 2 1 1   
2 3 1 3   
3 4 1 1   
4 5 0 3   
.. ... ... ...   
886 887 0 2   
887 888 1 1   
888 889 0 3   
889 890 1 1   
890 891 0 3   
  
 Name Sex Age SibSp \  
0 Braund, Mr. Owen Harris male 22.0 1   
1 Cumings, Mrs. John Bradley (Florence Briggs Th... female 38.0 1   
2 Heikkinen, Miss. Laina female 26.0 0   
3 Futrelle, Mrs. Jacques Heath (Lily May Peel) female 35.0 1   
4 Allen, Mr. William Henry male 35.0 0   
.. ... ... ... ...   
886 Montvila, Rev. Juozas male 27.0 0   
887 Graham, Miss. Margaret Edith female 19.0 0   
888 Johnston, Miss. Catherine Helen "Carrie" female NaN 1   
889 Behr, Mr. Karl Howell male 26.0 0   
890 Dooley, Mr. Patrick male 32.0 0   
  
 Parch Ticket Fare Cabin Embarked   
0 0 A/5 21171 7.2500 NaN S   
1 0 PC 17599 71.2833 C85 C   
2 0 STON/O2. 3101282 7.9250 NaN S   
3 0 113803 53.1000 C123 S   
4 0 373450 8.0500 NaN S   
.. ... ... ... ... ...   
886 0 211536 13.0000 NaN S   
887 0 112053 30.0000 B42 S   
888 2 W./C. 6607 23.4500 NaN S   
889 0 111369 30.0000 C148 C   
890 0 370376 7.7500 NaN Q   
  
[891 rows x 12 columns]

Select relevant columns from Adult dataset (mix of nominal and ratio-scaled)

adult\_df = adult\_df[["age", "workclass", "education", "education\_num", "sex"]]  
  
adult\_df

age workclass education education\_num sex  
0 39 State-gov Bachelors 13 Male  
1 50 Self-emp-not-inc Bachelors 13 Male  
2 38 Private HS-grad 9 Male  
3 53 Private 11th 7 Male  
4 28 Private Bachelors 13 Female  
.. ... ... ... ... ...  
95 29 Local-gov Some-college 10 Male  
96 48 Self-emp-not-inc Doctorate 16 Male  
97 37 Private Some-college 10 Male  
98 48 Private Assoc-acdm 12 Female  
99 32 Federal-gov HS-grad 9 Male  
  
[100 rows x 5 columns]

Encode nominal attributes as integers for processing

label\_encoders = {}  
**for** column **in** adult\_df.columns:  
 **if** adult\_df[column].dtype == object:  
 le = LabelEncoder()  
 adult\_df[column] = le.fit\_transform(adult\_df[column])  
 label\_encoders[column] = le  
  
adult\_df

C:\Users\debat\AppData\Local\Temp\ipykernel\_12436\183426126.py:5: SettingWithCopyWarning:   
A value is trying to be set on a copy of a slice from a DataFrame.  
Try using .loc[row\_indexer,col\_indexer] = value instead  
  
See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user\_guide/indexing.html#returning-a-view-versus-a-copy  
 adult\_df[column] = le.fit\_transform(adult\_df[column])  
C:\Users\debat\AppData\Local\Temp\ipykernel\_12436\183426126.py:5: SettingWithCopyWarning:   
A value is trying to be set on a copy of a slice from a DataFrame.  
Try using .loc[row\_indexer,col\_indexer] = value instead  
  
See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user\_guide/indexing.html#returning-a-view-versus-a-copy  
 adult\_df[column] = le.fit\_transform(adult\_df[column])  
C:\Users\debat\AppData\Local\Temp\ipykernel\_12436\183426126.py:5: SettingWithCopyWarning:   
A value is trying to be set on a copy of a slice from a DataFrame.  
Try using .loc[row\_indexer,col\_indexer] = value instead  
  
See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user\_guide/indexing.html#returning-a-view-versus-a-copy  
 adult\_df[column] = le.fit\_transform(adult\_df[column])

age workclass education education\_num sex  
0 39 6 7 13 1  
1 50 5 7 13 1  
2 38 3 9 9 1  
3 53 3 1 7 1  
4 28 3 7 13 0  
.. ... ... ... ... ...  
95 29 2 12 10 1  
96 48 5 8 16 1  
97 37 3 12 10 1  
98 48 3 5 12 0  
99 32 1 9 9 1  
  
[100 rows x 5 columns]

Clean and preprocess Titanic dataset

titanic\_df.dropna(inplace=True)  
titanic\_df

PassengerId Survived Pclass \  
1 2 1 1   
3 4 1 1   
6 7 0 1   
10 11 1 3   
11 12 1 1   
.. ... ... ...   
871 872 1 1   
872 873 0 1   
879 880 1 1   
887 888 1 1   
889 890 1 1   
  
 Name Sex Age SibSp \  
1 Cumings, Mrs. John Bradley (Florence Briggs Th... female 38.0 1   
3 Futrelle, Mrs. Jacques Heath (Lily May Peel) female 35.0 1   
6 McCarthy, Mr. Timothy J male 54.0 0   
10 Sandstrom, Miss. Marguerite Rut female 4.0 1   
11 Bonnell, Miss. Elizabeth female 58.0 0   
.. ... ... ... ...   
871 Beckwith, Mrs. Richard Leonard (Sallie Monypeny) female 47.0 1   
872 Carlsson, Mr. Frans Olof male 33.0 0   
879 Potter, Mrs. Thomas Jr (Lily Alexenia Wilson) female 56.0 0   
887 Graham, Miss. Margaret Edith female 19.0 0   
889 Behr, Mr. Karl Howell male 26.0 0   
  
 Parch Ticket Fare Cabin Embarked   
1 0 PC 17599 71.2833 C85 C   
3 0 113803 53.1000 C123 S   
6 0 17463 51.8625 E46 S   
10 1 PP 9549 16.7000 G6 S   
11 0 113783 26.5500 C103 S   
.. ... ... ... ... ...   
871 1 11751 52.5542 D35 S   
872 0 695 5.0000 B51 B53 B55 S   
879 1 11767 83.1583 C50 C   
887 0 112053 30.0000 B42 S   
889 0 111369 30.0000 C148 C   
  
[183 rows x 12 columns]

Select relevant columns from Titanic dataset (mix of nominal and ratio-scaled)

titanic\_df = titanic\_df[["Age", "Sex", "Pclass", "Fare", "Embarked"]]  
titanic\_df

Age Sex Pclass Fare Embarked  
1 38.0 female 1 71.2833 C  
3 35.0 female 1 53.1000 S  
6 54.0 male 1 51.8625 S  
10 4.0 female 3 16.7000 S  
11 58.0 female 1 26.5500 S  
.. ... ... ... ... ...  
871 47.0 female 1 52.5542 S  
872 33.0 male 1 5.0000 S  
879 56.0 female 1 83.1583 C  
887 19.0 female 1 30.0000 S  
889 26.0 male 1 30.0000 C  
  
[183 rows x 5 columns]

Encode Nominal as Integers for processing

label\_encoders\_titanic = {}  
**for** column **in** titanic\_df.columns:  
 **if** titanic\_df[column].dtype == object:  
 le = LabelEncoder()  
 titanic\_df[column] = le.fit\_transform(titanic\_df[column])  
 label\_encoders[column] = le  
  
titanic\_df

C:\Users\debat\AppData\Local\Temp\ipykernel\_12436\3305425594.py:5: SettingWithCopyWarning:   
A value is trying to be set on a copy of a slice from a DataFrame.  
Try using .loc[row\_indexer,col\_indexer] = value instead  
  
See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user\_guide/indexing.html#returning-a-view-versus-a-copy  
 titanic\_df[column] = le.fit\_transform(titanic\_df[column])  
C:\Users\debat\AppData\Local\Temp\ipykernel\_12436\3305425594.py:5: SettingWithCopyWarning:   
A value is trying to be set on a copy of a slice from a DataFrame.  
Try using .loc[row\_indexer,col\_indexer] = value instead  
  
See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user\_guide/indexing.html#returning-a-view-versus-a-copy  
 titanic\_df[column] = le.fit\_transform(titanic\_df[column])

Age Sex Pclass Fare Embarked  
1 38.0 0 1 71.2833 0  
3 35.0 0 1 53.1000 2  
6 54.0 1 1 51.8625 2  
10 4.0 0 3 16.7000 2  
11 58.0 0 1 26.5500 2  
.. ... ... ... ... ...  
871 47.0 0 1 52.5542 2  
872 33.0 1 1 5.0000 2  
879 56.0 0 1 83.1583 0  
887 19.0 0 1 30.0000 2  
889 26.0 1 1 30.0000 0  
  
[183 rows x 5 columns]

Combine the datasets into a list for further processing

*# Combine the datasets into a list for further processing*  
datasets = {  
 "Adult Dataset": adult\_df,  
 "Titanic Dataset": titanic\_df  
}

Code to Calculate SMC

**def** simple\_matching\_coefficient(a, b):  
 *"""Calculate the Simple Matching Coefficient between two vectors."""*  
 **try**:  
 **return** np.sum(a == b) / len(a)  
 **except** Exception **as** e:  
 **return** np.nan  
  
*# Function to create the proximity matrix*  
**def** calculate\_smc\_matrix(dataset):  
 n = len(dataset)  
 smc\_matrix = np.zeros((n, n))  
   
 **for** i **in** range(n):  
 *# print(f"{i}/{n}")*  
 **for** j **in** range(n):  
 smc\_matrix[i, j] = simple\_matching\_coefficient(dataset.iloc[i].values, dataset.iloc[j].values)  
   
 **return** pd.DataFrame(smc\_matrix)

Calculate SMC matrices for each dataset

smc\_matrix\_adult = calculate\_smc\_matrix(adult\_df)  
smc\_matrix\_titanic = calculate\_smc\_matrix(titanic\_df)

Print SMC matrices

*Adult Dataset SMC Matrix*

smc\_matrix\_adult

0 1 2 3 4 5 6 7 8 9 ... 90 91 92 93 \  
0 1.0 0.6 0.2 0.2 0.4 0.0 0.0 0.2 0.0 0.6 ... 0.2 0.0 0.0 0.0   
1 0.6 1.0 0.2 0.2 0.4 0.0 0.0 0.4 0.0 0.6 ... 0.2 0.0 0.0 0.0   
2 0.2 0.2 1.0 0.4 0.2 0.2 0.2 0.6 0.2 0.4 ... 0.4 0.2 0.2 0.6   
3 0.2 0.2 0.4 1.0 0.2 0.2 0.2 0.2 0.2 0.4 ... 0.4 0.2 0.2 0.2   
4 0.4 0.4 0.2 0.2 1.0 0.4 0.4 0.0 0.4 0.6 ... 0.2 0.4 0.6 0.4   
.. ... ... ... ... ... ... ... ... ... ... ... ... ... ... ...   
95 0.2 0.2 0.2 0.2 0.0 0.0 0.0 0.2 0.0 0.2 ... 0.2 0.4 0.4 0.0   
96 0.2 0.4 0.2 0.2 0.0 0.0 0.0 0.4 0.0 0.2 ... 0.2 0.0 0.0 0.0   
97 0.2 0.2 0.4 0.4 0.2 0.4 0.2 0.2 0.2 0.4 ... 0.4 0.8 0.6 0.2   
98 0.0 0.0 0.2 0.2 0.4 0.4 0.4 0.0 0.4 0.2 ... 0.2 0.4 0.4 0.4   
99 0.2 0.2 0.6 0.2 0.0 0.0 0.0 0.6 0.0 0.2 ... 0.2 0.0 0.0 0.4   
  
 94 95 96 97 98 99   
0 0.6 0.2 0.2 0.2 0.0 0.2   
1 0.6 0.2 0.4 0.2 0.0 0.2   
2 0.2 0.2 0.2 0.4 0.2 0.6   
3 0.2 0.2 0.2 0.4 0.2 0.2   
4 0.4 0.0 0.0 0.2 0.4 0.0   
.. ... ... ... ... ... ...   
95 0.4 1.0 0.2 0.6 0.0 0.2   
96 0.2 0.2 1.0 0.2 0.2 0.2   
97 0.2 0.6 0.2 1.0 0.2 0.2   
98 0.0 0.0 0.2 0.2 1.0 0.0   
99 0.2 0.2 0.2 0.2 0.0 1.0   
  
[100 rows x 100 columns]

*Titanic Dataset SMC Matrix*

smc\_matrix\_titanic

0 1 2 3 4 5 6 7 8 9 ... 173 174 175 \  
0 1.0 0.4 0.2 0.2 0.4 0.0 0.2 0.2 0.6 0.4 ... 0.6 0.4 0.2   
1 0.4 1.0 0.4 0.4 0.6 0.2 0.4 0.4 0.4 0.2 ... 0.4 0.6 0.4   
2 0.2 0.4 1.0 0.2 0.4 0.4 0.6 0.6 0.2 0.4 ... 0.2 0.4 0.6   
3 0.2 0.4 0.2 1.0 0.4 0.2 0.2 0.2 0.2 0.0 ... 0.2 0.4 0.2   
4 0.4 0.6 0.4 0.4 1.0 0.2 0.4 0.4 0.4 0.2 ... 0.4 0.6 0.6   
.. ... ... ... ... ... ... ... ... ... ... ... ... ... ...   
178 0.4 0.6 0.4 0.4 0.6 0.2 0.4 0.4 0.4 0.2 ... 0.4 0.6 0.4   
179 0.2 0.4 0.6 0.2 0.4 0.4 0.6 0.6 0.2 0.4 ... 0.2 0.4 0.6   
180 0.6 0.4 0.2 0.2 0.4 0.0 0.2 0.2 0.6 0.4 ... 0.8 0.4 0.2   
181 0.4 0.6 0.4 0.4 0.6 0.2 0.4 0.6 0.4 0.2 ... 0.4 0.6 0.4   
182 0.4 0.2 0.4 0.0 0.2 0.2 0.4 0.4 0.4 0.6 ... 0.4 0.2 0.4   
  
 176 177 178 179 180 181 182   
0 0.4 0.2 0.4 0.2 0.6 0.4 0.4   
1 0.6 0.4 0.6 0.4 0.4 0.6 0.2   
2 0.4 0.6 0.4 0.6 0.2 0.4 0.4   
3 0.4 0.2 0.4 0.2 0.2 0.4 0.0   
4 0.6 0.4 0.6 0.4 0.4 0.6 0.2   
.. ... ... ... ... ... ... ...   
178 0.6 0.4 1.0 0.4 0.4 0.6 0.2   
179 0.4 0.6 0.4 1.0 0.2 0.4 0.4   
180 0.4 0.2 0.4 0.2 1.0 0.4 0.4   
181 0.6 0.4 0.6 0.4 0.4 1.0 0.4   
182 0.2 0.4 0.2 0.4 0.4 0.4 1.0   
  
[183 rows x 183 columns]

Jaccard Coefficient

Jaccad Coefficient (SMC)

it's specifically useful for binary attributes where the focus is on the presence of attributes (ignoring the absence).

Importing required Libraries

**import** pandas **as** pd  
**import** numpy **as** np  
**from** sklearn.preprocessing **import** LabelEncoder  
**from** sklearn.metrics **import** jaccard\_score

Load Datasets

*# Load datasets*  
adult\_df = pd.read\_csv("../adult/adult\_trim.data", header=None) *# No header*  
titanic\_df = pd.read\_csv('../titanic/titanic\_trim.csv') *# Has header*  
  
*# Rename columns for clarity*  
adult\_df.columns = ["age", "workclass", "fnlwgt", "education", "education\_num",   
 "marital\_status", "occupation", "relationship", "race", "sex",   
 "capital\_gain", "capital\_loss", "hours\_per\_week", "native\_country", "income"]  
adult\_df.dropna(inplace=True)

adult\_df

age workclass fnlwgt education education\_num \  
0 39 State-gov 77516 Bachelors 13   
1 50 Self-emp-not-inc 83311 Bachelors 13   
2 38 Private 215646 HS-grad 9   
3 53 Private 234721 11th 7   
4 28 Private 338409 Bachelors 13   
.. ... ... ... ... ...   
95 29 Local-gov 115585 Some-college 10   
96 48 Self-emp-not-inc 191277 Doctorate 16   
97 37 Private 202683 Some-college 10   
98 48 Private 171095 Assoc-acdm 12   
99 32 Federal-gov 249409 HS-grad 9   
  
 marital\_status occupation relationship race sex \  
0 Never-married Adm-clerical Not-in-family White Male   
1 Married-civ-spouse Exec-managerial Husband White Male   
2 Divorced Handlers-cleaners Not-in-family White Male   
3 Married-civ-spouse Handlers-cleaners Husband Black Male   
4 Married-civ-spouse Prof-specialty Wife Black Female   
.. ... ... ... ... ...   
95 Never-married Handlers-cleaners Not-in-family White Male   
96 Married-civ-spouse Prof-specialty Husband White Male   
97 Married-civ-spouse Sales Husband White Male   
98 Divorced Exec-managerial Unmarried White Female   
99 Never-married Other-service Own-child Black Male   
  
 capital\_gain capital\_loss hours\_per\_week native\_country income   
0 2174 0 40 United-States <=50K   
1 0 0 13 United-States <=50K   
2 0 0 40 United-States <=50K   
3 0 0 40 United-States <=50K   
4 0 0 40 Cuba <=50K   
.. ... ... ... ... ...   
95 0 0 50 United-States <=50K   
96 0 1902 60 United-States >50K   
97 0 0 48 United-States >50K   
98 0 0 40 England <=50K   
99 0 0 40 United-States <=50K   
  
[100 rows x 15 columns]

titanic\_df

PassengerId Survived Pclass \  
0 1 0 3   
1 2 1 1   
2 3 1 3   
3 4 1 1   
4 5 0 3   
.. ... ... ...   
150 151 0 2   
151 152 1 1   
152 153 0 3   
153 154 0 3   
154 155 0 3   
  
 Name Sex Age SibSp \  
0 Braund, Mr. Owen Harris male 22.0 1   
1 Cumings, Mrs. John Bradley (Florence Briggs Th... female 38.0 1   
2 Heikkinen, Miss. Laina female 26.0 0   
3 Futrelle, Mrs. Jacques Heath (Lily May Peel) female 35.0 1   
4 Allen, Mr. William Henry male 35.0 0   
.. ... ... ... ...   
150 Bateman, Rev. Robert James male 51.0 0   
151 Pears, Mrs. Thomas (Edith Wearne) female 22.0 1   
152 Meo, Mr. Alfonzo male 55.5 0   
153 van Billiard, Mr. Austin Blyler male 40.5 0   
154 Olsen, Mr. Ole Martin male NaN 0   
  
 Parch Ticket Fare Cabin Embarked   
0 0 A/5 21171 7.2500 NaN S   
1 0 PC 17599 71.2833 C85 C   
2 0 STON/O2. 3101282 7.9250 NaN S   
3 0 113803 53.1000 C123 S   
4 0 373450 8.0500 NaN S   
.. ... ... ... ... ...   
150 0 S.O.P. 1166 12.5250 NaN S   
151 0 113776 66.6000 C2 S   
152 0 A.5. 11206 8.0500 NaN S   
153 2 A/5. 851 14.5000 NaN S   
154 0 Fa 265302 7.3125 NaN S   
  
[155 rows x 12 columns]

Select relevant columns from Adult dataset (mix of nominal and ratio-scaled)

adult\_df = adult\_df[["age", "workclass", "education", "education\_num", "sex"]]  
  
adult\_df

age workclass education education\_num sex  
0 39 State-gov Bachelors 13 Male  
1 50 Self-emp-not-inc Bachelors 13 Male  
2 38 Private HS-grad 9 Male  
3 53 Private 11th 7 Male  
4 28 Private Bachelors 13 Female  
.. ... ... ... ... ...  
95 29 Local-gov Some-college 10 Male  
96 48 Self-emp-not-inc Doctorate 16 Male  
97 37 Private Some-college 10 Male  
98 48 Private Assoc-acdm 12 Female  
99 32 Federal-gov HS-grad 9 Male  
  
[100 rows x 5 columns]

Encode nominal attributes as integers for processing

label\_encoders = {}  
**for** column **in** adult\_df.columns:  
 **if** adult\_df[column].dtype == object:  
 le = LabelEncoder()  
 adult\_df[column] = le.fit\_transform(adult\_df[column])  
 label\_encoders[column] = le  
  
adult\_df

Clean and preprocess Titanic dataset

titanic\_df.dropna(inplace=True)  
titanic\_df

PassengerId Survived Pclass \  
1 2 1 1   
3 4 1 1   
6 7 0 1   
10 11 1 3   
11 12 1 1   
21 22 1 2   
23 24 1 1   
27 28 0 1   
52 53 1 1   
54 55 0 1   
62 63 0 1   
66 67 1 2   
75 76 0 3   
88 89 1 1   
92 93 0 1   
96 97 0 1   
97 98 1 1   
102 103 0 1   
110 111 0 1   
118 119 0 1   
123 124 1 2   
124 125 0 1   
136 137 1 1   
137 138 0 1   
139 140 0 1   
148 149 0 2   
151 152 1 1   
  
 Name Sex Age SibSp \  
1 Cumings, Mrs. John Bradley (Florence Briggs Th... female 38.0 1   
3 Futrelle, Mrs. Jacques Heath (Lily May Peel) female 35.0 1   
6 McCarthy, Mr. Timothy J male 54.0 0   
10 Sandstrom, Miss. Marguerite Rut female 4.0 1   
11 Bonnell, Miss. Elizabeth female 58.0 0   
21 Beesley, Mr. Lawrence male 34.0 0   
23 Sloper, Mr. William Thompson male 28.0 0   
27 Fortune, Mr. Charles Alexander male 19.0 3   
52 Harper, Mrs. Henry Sleeper (Myna Haxtun) female 49.0 1   
54 Ostby, Mr. Engelhart Cornelius male 65.0 0   
62 Harris, Mr. Henry Birkhardt male 45.0 1   
66 Nye, Mrs. (Elizabeth Ramell) female 29.0 0   
75 Moen, Mr. Sigurd Hansen male 25.0 0   
88 Fortune, Miss. Mabel Helen female 23.0 3   
92 Chaffee, Mr. Herbert Fuller male 46.0 1   
96 Goldschmidt, Mr. George B male 71.0 0   
97 Greenfield, Mr. William Bertram male 23.0 0   
102 White, Mr. Richard Frasar male 21.0 0   
110 Porter, Mr. Walter Chamberlain male 47.0 0   
118 Baxter, Mr. Quigg Edmond male 24.0 0   
123 Webber, Miss. Susan female 32.5 0   
124 White, Mr. Percival Wayland male 54.0 0   
136 Newsom, Miss. Helen Monypeny female 19.0 0   
137 Futrelle, Mr. Jacques Heath male 37.0 1   
139 Giglio, Mr. Victor male 24.0 0   
148 Navratil, Mr. Michel ("Louis M Hoffman") male 36.5 0   
151 Pears, Mrs. Thomas (Edith Wearne) female 22.0 1   
  
 Parch Ticket Fare Cabin Embarked   
1 0 PC 17599 71.2833 C85 C   
3 0 113803 53.1000 C123 S   
6 0 17463 51.8625 E46 S   
10 1 PP 9549 16.7000 G6 S   
11 0 113783 26.5500 C103 S   
21 0 248698 13.0000 D56 S   
23 0 113788 35.5000 A6 S   
27 2 19950 263.0000 C23 C25 C27 S   
52 0 PC 17572 76.7292 D33 C   
54 1 113509 61.9792 B30 C   
62 0 36973 83.4750 C83 S   
66 0 C.A. 29395 10.5000 F33 S   
75 0 348123 7.6500 F G73 S   
88 2 19950 263.0000 C23 C25 C27 S   
92 0 W.E.P. 5734 61.1750 E31 S   
96 0 PC 17754 34.6542 A5 C   
97 1 PC 17759 63.3583 D10 D12 C   
102 1 35281 77.2875 D26 S   
110 0 110465 52.0000 C110 S   
118 1 PC 17558 247.5208 B58 B60 C   
123 0 27267 13.0000 E101 S   
124 1 35281 77.2875 D26 S   
136 2 11752 26.2833 D47 S   
137 0 113803 53.1000 C123 S   
139 0 PC 17593 79.2000 B86 C   
148 2 230080 26.0000 F2 S   
151 0 113776 66.6000 C2 S

Select relevant columns from Titanic dataset (mix of nominal and ratio-scaled)

titanic\_df = titanic\_df[["Age", "Sex", "Pclass", "Fare", "Embarked"]]  
titanic\_df

Age Sex Pclass Fare Embarked  
1 38.0 female 1 71.2833 C  
3 35.0 female 1 53.1000 S  
6 54.0 male 1 51.8625 S  
10 4.0 female 3 16.7000 S  
11 58.0 female 1 26.5500 S  
21 34.0 male 2 13.0000 S  
23 28.0 male 1 35.5000 S  
27 19.0 male 1 263.0000 S  
52 49.0 female 1 76.7292 C  
54 65.0 male 1 61.9792 C  
62 45.0 male 1 83.4750 S  
66 29.0 female 2 10.5000 S  
75 25.0 male 3 7.6500 S  
88 23.0 female 1 263.0000 S  
92 46.0 male 1 61.1750 S  
96 71.0 male 1 34.6542 C  
97 23.0 male 1 63.3583 C  
102 21.0 male 1 77.2875 S  
110 47.0 male 1 52.0000 S  
118 24.0 male 1 247.5208 C  
123 32.5 female 2 13.0000 S  
124 54.0 male 1 77.2875 S  
136 19.0 female 1 26.2833 S  
137 37.0 male 1 53.1000 S  
139 24.0 male 1 79.2000 C  
148 36.5 male 2 26.0000 S  
151 22.0 female 1 66.6000 S

Encode Nominal as Integers for processing

label\_encoders\_titanic = {}  
**for** column **in** titanic\_df.columns:  
 **if** titanic\_df[column].dtype == object:  
 le = LabelEncoder()  
 titanic\_df[column] = le.fit\_transform(titanic\_df[column])  
 label\_encoders[column] = le  
  
titanic\_df

C:\Users\debat\AppData\Local\Temp\ipykernel\_16092\3305425594.py:5: SettingWithCopyWarning:   
A value is trying to be set on a copy of a slice from a DataFrame.  
Try using .loc[row\_indexer,col\_indexer] = value instead  
  
See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user\_guide/indexing.html#returning-a-view-versus-a-copy  
 titanic\_df[column] = le.fit\_transform(titanic\_df[column])  
C:\Users\debat\AppData\Local\Temp\ipykernel\_16092\3305425594.py:5: SettingWithCopyWarning:   
A value is trying to be set on a copy of a slice from a DataFrame.  
Try using .loc[row\_indexer,col\_indexer] = value instead  
  
See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user\_guide/indexing.html#returning-a-view-versus-a-copy  
 titanic\_df[column] = le.fit\_transform(titanic\_df[column])

Age Sex Pclass Fare Embarked  
1 38.0 0 1 71.2833 0  
3 35.0 0 1 53.1000 1  
6 54.0 1 1 51.8625 1  
10 4.0 0 3 16.7000 1  
11 58.0 0 1 26.5500 1  
21 34.0 1 2 13.0000 1  
23 28.0 1 1 35.5000 1  
27 19.0 1 1 263.0000 1  
52 49.0 0 1 76.7292 0  
54 65.0 1 1 61.9792 0  
62 45.0 1 1 83.4750 1  
66 29.0 0 2 10.5000 1  
75 25.0 1 3 7.6500 1  
88 23.0 0 1 263.0000 1  
92 46.0 1 1 61.1750 1  
96 71.0 1 1 34.6542 0  
97 23.0 1 1 63.3583 0  
102 21.0 1 1 77.2875 1  
110 47.0 1 1 52.0000 1  
118 24.0 1 1 247.5208 0  
123 32.5 0 2 13.0000 1  
124 54.0 1 1 77.2875 1  
136 19.0 0 1 26.2833 1  
137 37.0 1 1 53.1000 1  
139 24.0 1 1 79.2000 0  
148 36.5 1 2 26.0000 1  
151 22.0 0 1 66.6000 1

Combine the datasets into a list for further processing

*# Combine the datasets into a list for further processing*  
datasets = {  
 "Adult Dataset": adult\_df,  
 "Titanic Dataset": titanic\_df  
}

Computing Jaccard Coefficient

*The Jaccard Coefficient is calculated as the size of the intersection divided by the size of the union of the attribute sets. It's typically used for binary or nominal data.*

**def** jaccard\_coefficient(a, b):  
 *"""Calculate the Jaccard Coefficient between two vectors."""*  
 **try**:  
 *# Convert to binary format if necessary (for categorical data)*  
 **return** jaccard\_score(a, b, average='macro')  
 **except** Exception **as** e:  
 **return** np.nan  
  
*# Function to create the Jaccard proximity matrix*  
**def** calculate\_jaccard\_matrix(dataset):  
 n = len(dataset)  
 jaccard\_matrix = np.zeros((n, n))  
   
 **for** i **in** range(n):  
 *# print(f"{i}/{n}")*  
 **for** j **in** range(n):  
 jaccard\_matrix[i, j] = jaccard\_coefficient(dataset.iloc[i].values, dataset.iloc[j].values)  
   
 **return** pd.DataFrame(jaccard\_matrix)

Calculating Matrices for the data sets

jaccard\_matrix\_adult = calculate\_jaccard\_matrix(adult\_df)  
jaccard\_matrix\_titanic = calculate\_jaccard\_matrix(titanic\_df)

Displaying the Matrices

*Adult Dataset: Jaccard Matrix*

jaccard\_matrix\_adult

0 1 2 3 4 5 6 \  
0 1.000000 0.428571 0.125000 0.071429 0.250000 0.000000 0.000000   
1 0.428571 1.000000 0.125000 0.071429 0.250000 0.000000 0.000000   
2 0.125000 0.125000 1.000000 0.250000 0.125000 0.125000 0.125000   
3 0.071429 0.071429 0.250000 1.000000 0.142857 0.125000 0.125000   
4 0.250000 0.250000 0.125000 0.142857 1.000000 0.250000 0.250000   
.. ... ... ... ... ... ... ...   
95 0.111111 0.111111 0.125000 0.062500 0.000000 0.000000 0.000000   
96 0.111111 0.250000 0.125000 0.062500 0.000000 0.000000 0.000000   
97 0.111111 0.111111 0.285714 0.214286 0.111111 0.285714 0.111111   
98 0.000000 0.000000 0.125000 0.125000 0.250000 0.250000 0.285714   
99 0.071429 0.071429 0.300000 0.055556 0.000000 0.000000 0.000000   
  
 7 8 9 ... 90 91 92 93 \  
0 0.125000 0.000 0.428571 ... 0.125000 0.000000 0.000000 0.000000   
1 0.285714 0.000 0.428571 ... 0.111111 0.000000 0.000000 0.000000   
2 0.333333 0.125 0.285714 ... 0.285714 0.125000 0.125000 0.333333   
3 0.071429 0.125 0.250000 ... 0.214286 0.125000 0.125000 0.142857   
4 0.000000 0.250 0.428571 ... 0.111111 0.250000 0.428571 0.285714   
.. ... ... ... ... ... ... ... ...   
95 0.125000 0.000 0.111111 ... 0.111111 0.250000 0.250000 0.000000   
96 0.285714 0.000 0.111111 ... 0.111111 0.000000 0.000000 0.000000   
97 0.125000 0.125 0.250000 ... 0.250000 0.666667 0.428571 0.125000   
98 0.000000 0.250 0.111111 ... 0.111111 0.285714 0.285714 0.285714   
99 0.300000 0.000 0.071429 ... 0.071429 0.000000 0.000000 0.166667   
  
 94 95 96 97 98 99   
0 0.428571 0.111111 0.111111 0.111111 0.000 0.071429   
1 0.428571 0.111111 0.250000 0.111111 0.000 0.071429   
2 0.125000 0.125000 0.125000 0.285714 0.125 0.300000   
3 0.071429 0.062500 0.062500 0.214286 0.125 0.055556   
4 0.250000 0.000000 0.000000 0.111111 0.250 0.000000   
.. ... ... ... ... ... ...   
95 0.250000 1.000000 0.111111 0.428571 0.000 0.071429   
96 0.111111 0.111111 1.000000 0.111111 0.125 0.071429   
97 0.111111 0.428571 0.111111 1.000000 0.125 0.071429   
98 0.000000 0.000000 0.125000 0.125000 1.000 0.000000   
99 0.071429 0.071429 0.071429 0.071429 0.000 1.000000   
  
[100 rows x 100 columns]

*Titanic Dataset: Jaccard Matrix*

jaccard\_matrix\_titanic

0 1 2 3 4 5 6 7 8 9 ... 17 18 19 \  
0 NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN ... NaN NaN NaN   
1 NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN ... NaN NaN NaN   
2 NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN ... NaN NaN NaN   
3 NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN ... NaN NaN NaN   
4 NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN ... NaN NaN NaN   
5 NaN NaN NaN NaN NaN 1.000000 NaN 0.111111 NaN NaN ... NaN 0.111111 NaN   
6 NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN ... NaN NaN NaN   
7 NaN NaN NaN NaN NaN 0.111111 NaN 1.000000 NaN NaN ... NaN 0.200000 NaN   
8 NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN ... NaN NaN NaN   
9 NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN ... NaN NaN NaN   
10 NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN ... NaN NaN NaN   
11 NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN ... NaN NaN NaN   
12 NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN ... NaN NaN NaN   
13 NaN NaN NaN NaN NaN 0.047619 NaN 0.333333 NaN NaN ... NaN 0.111111 NaN   
14 NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN ... NaN NaN NaN   
15 NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN ... NaN NaN NaN   
16 NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN ... NaN NaN NaN   
17 NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN ... NaN NaN NaN   
18 NaN NaN NaN NaN NaN 0.111111 NaN 0.200000 NaN NaN ... NaN 1.000000 NaN   
19 NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN ... NaN NaN NaN   
20 NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN ... NaN NaN NaN   
21 NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN ... NaN NaN NaN   
22 NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN ... NaN NaN NaN   
23 NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN ... NaN NaN NaN   
24 NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN ... NaN NaN NaN   
25 NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN ... NaN NaN NaN   
26 NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN ... NaN NaN NaN   
  
 20 21 22 23 24 25 26   
0 NaN NaN NaN NaN NaN NaN NaN   
1 NaN NaN NaN NaN NaN NaN NaN   
2 NaN NaN NaN NaN NaN NaN NaN   
3 NaN NaN NaN NaN NaN NaN NaN   
4 NaN NaN NaN NaN NaN NaN NaN   
5 NaN NaN NaN NaN NaN NaN NaN   
6 NaN NaN NaN NaN NaN NaN NaN   
7 NaN NaN NaN NaN NaN NaN NaN   
8 NaN NaN NaN NaN NaN NaN NaN   
9 NaN NaN NaN NaN NaN NaN NaN   
10 NaN NaN NaN NaN NaN NaN NaN   
11 NaN NaN NaN NaN NaN NaN NaN   
12 NaN NaN NaN NaN NaN NaN NaN   
13 NaN NaN NaN NaN NaN NaN NaN   
14 NaN NaN NaN NaN NaN NaN NaN   
15 NaN NaN NaN NaN NaN NaN NaN   
16 NaN NaN NaN NaN NaN NaN NaN   
17 NaN NaN NaN NaN NaN NaN NaN   
18 NaN NaN NaN NaN NaN NaN NaN   
19 NaN NaN NaN NaN NaN NaN NaN   
20 NaN NaN NaN NaN NaN NaN NaN   
21 NaN NaN NaN NaN NaN NaN NaN   
22 NaN NaN NaN NaN NaN NaN NaN   
23 NaN NaN NaN NaN NaN NaN NaN   
24 NaN NaN NaN NaN NaN NaN NaN   
25 NaN NaN NaN NaN NaN NaN NaN   
26 NaN NaN NaN NaN NaN NaN NaN   
  
[27 rows x 27 columns]

Explanation

Jaccard Coefficient Calculation: This metric focuses on the presence of attributes, making it useful for binary data. It calculates the ratio of the intersection of attributes to the union of attributes between two data points.

Handling Nominal Data: The Jaccard Coefficient is generally used for binary data, but in this case, we can apply it to the encoded nominal attributes as they are binary representations of categories.

Observation and Analysis

The resulting matrices will have values between 0 and 1, where 1 indicates identical presence/absence of attributes (perfect similarity) and 0 indicates no similarity.

Jaccard Coefficient is particularly sensitive to the presence of attributes (1s) and ignores cases where both attributes are absent (0s), making it a useful measure for sparse datasets.

Cosine Similarity

Cosine Similarity

This metric measures the cosine of the angle between two non-zero vectors in a multi-dimensional space. It is particularly useful for interval and ratio-scaled attributes, where the magnitude of the vectors is not as important as their direction.

Importing required Libraries

**import** pandas **as** pd  
**import** numpy **as** np  
**from** sklearn.preprocessing **import** LabelEncoder  
**from** scipy.spatial.distance **import** cosine

Load Datasets

*# Load datasets*  
adult\_df = pd.read\_csv("../adult/adult\_trim.data", header=None) *# No header*  
titanic\_df = pd.read\_csv('../titanic/titanic\_trim.csv') *# Has header*  
  
*# Rename columns for clarity*  
adult\_df.columns = ["age", "workclass", "fnlwgt", "education", "education\_num",   
 "marital\_status", "occupation", "relationship", "race", "sex",   
 "capital\_gain", "capital\_loss", "hours\_per\_week", "native\_country", "income"]  
adult\_df.dropna(inplace=True)

adult\_df

age workclass fnlwgt education education\_num \  
0 39 State-gov 77516 Bachelors 13   
1 50 Self-emp-not-inc 83311 Bachelors 13   
2 38 Private 215646 HS-grad 9   
3 53 Private 234721 11th 7   
4 28 Private 338409 Bachelors 13   
.. ... ... ... ... ...   
95 29 Local-gov 115585 Some-college 10   
96 48 Self-emp-not-inc 191277 Doctorate 16   
97 37 Private 202683 Some-college 10   
98 48 Private 171095 Assoc-acdm 12   
99 32 Federal-gov 249409 HS-grad 9   
  
 marital\_status occupation relationship race sex \  
0 Never-married Adm-clerical Not-in-family White Male   
1 Married-civ-spouse Exec-managerial Husband White Male   
2 Divorced Handlers-cleaners Not-in-family White Male   
3 Married-civ-spouse Handlers-cleaners Husband Black Male   
4 Married-civ-spouse Prof-specialty Wife Black Female   
.. ... ... ... ... ...   
95 Never-married Handlers-cleaners Not-in-family White Male   
96 Married-civ-spouse Prof-specialty Husband White Male   
97 Married-civ-spouse Sales Husband White Male   
98 Divorced Exec-managerial Unmarried White Female   
99 Never-married Other-service Own-child Black Male   
  
 capital\_gain capital\_loss hours\_per\_week native\_country income   
0 2174 0 40 United-States <=50K   
1 0 0 13 United-States <=50K   
2 0 0 40 United-States <=50K   
3 0 0 40 United-States <=50K   
4 0 0 40 Cuba <=50K   
.. ... ... ... ... ...   
95 0 0 50 United-States <=50K   
96 0 1902 60 United-States >50K   
97 0 0 48 United-States >50K   
98 0 0 40 England <=50K   
99 0 0 40 United-States <=50K   
  
[100 rows x 15 columns]

titanic\_df

PassengerId Survived Pclass \  
0 1 0 3   
1 2 1 1   
2 3 1 3   
3 4 1 1   
4 5 0 3   
.. ... ... ...   
150 151 0 2   
151 152 1 1   
152 153 0 3   
153 154 0 3   
154 155 0 3   
  
 Name Sex Age SibSp \  
0 Braund, Mr. Owen Harris male 22.0 1   
1 Cumings, Mrs. John Bradley (Florence Briggs Th... female 38.0 1   
2 Heikkinen, Miss. Laina female 26.0 0   
3 Futrelle, Mrs. Jacques Heath (Lily May Peel) female 35.0 1   
4 Allen, Mr. William Henry male 35.0 0   
.. ... ... ... ...   
150 Bateman, Rev. Robert James male 51.0 0   
151 Pears, Mrs. Thomas (Edith Wearne) female 22.0 1   
152 Meo, Mr. Alfonzo male 55.5 0   
153 van Billiard, Mr. Austin Blyler male 40.5 0   
154 Olsen, Mr. Ole Martin male NaN 0   
  
 Parch Ticket Fare Cabin Embarked   
0 0 A/5 21171 7.2500 NaN S   
1 0 PC 17599 71.2833 C85 C   
2 0 STON/O2. 3101282 7.9250 NaN S   
3 0 113803 53.1000 C123 S   
4 0 373450 8.0500 NaN S   
.. ... ... ... ... ...   
150 0 S.O.P. 1166 12.5250 NaN S   
151 0 113776 66.6000 C2 S   
152 0 A.5. 11206 8.0500 NaN S   
153 2 A/5. 851 14.5000 NaN S   
154 0 Fa 265302 7.3125 NaN S   
  
[155 rows x 12 columns]

Select relevant columns from Adult dataset (mix of nominal and ratio-scaled)

adult\_df = adult\_df[["age", "workclass", "education", "education\_num", "sex"]]  
  
adult\_df

age workclass education education\_num sex  
0 39 State-gov Bachelors 13 Male  
1 50 Self-emp-not-inc Bachelors 13 Male  
2 38 Private HS-grad 9 Male  
3 53 Private 11th 7 Male  
4 28 Private Bachelors 13 Female  
.. ... ... ... ... ...  
95 29 Local-gov Some-college 10 Male  
96 48 Self-emp-not-inc Doctorate 16 Male  
97 37 Private Some-college 10 Male  
98 48 Private Assoc-acdm 12 Female  
99 32 Federal-gov HS-grad 9 Male  
  
[100 rows x 5 columns]

Encode nominal attributes as integers for processing

label\_encoders = {}  
**for** column **in** adult\_df.columns:  
 **if** adult\_df[column].dtype == object:  
 le = LabelEncoder()  
 adult\_df[column] = le.fit\_transform(adult\_df[column])  
 label\_encoders[column] = le  
  
adult\_df

C:\Users\debat\AppData\Local\Temp\ipykernel\_15644\183426126.py:5: SettingWithCopyWarning:   
A value is trying to be set on a copy of a slice from a DataFrame.  
Try using .loc[row\_indexer,col\_indexer] = value instead  
  
See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user\_guide/indexing.html#returning-a-view-versus-a-copy  
 adult\_df[column] = le.fit\_transform(adult\_df[column])  
C:\Users\debat\AppData\Local\Temp\ipykernel\_15644\183426126.py:5: SettingWithCopyWarning:   
A value is trying to be set on a copy of a slice from a DataFrame.  
Try using .loc[row\_indexer,col\_indexer] = value instead  
  
See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user\_guide/indexing.html#returning-a-view-versus-a-copy  
 adult\_df[column] = le.fit\_transform(adult\_df[column])  
C:\Users\debat\AppData\Local\Temp\ipykernel\_15644\183426126.py:5: SettingWithCopyWarning:   
A value is trying to be set on a copy of a slice from a DataFrame.  
Try using .loc[row\_indexer,col\_indexer] = value instead  
  
See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user\_guide/indexing.html#returning-a-view-versus-a-copy  
 adult\_df[column] = le.fit\_transform(adult\_df[column])

age workclass education education\_num sex  
0 39 6 7 13 1  
1 50 5 7 13 1  
2 38 3 9 9 1  
3 53 3 1 7 1  
4 28 3 7 13 0  
.. ... ... ... ... ...  
95 29 2 12 10 1  
96 48 5 8 16 1  
97 37 3 12 10 1  
98 48 3 5 12 0  
99 32 1 9 9 1  
  
[100 rows x 5 columns]

Clean and preprocess Titanic dataset

titanic\_df.dropna(inplace=True)  
titanic\_df

PassengerId Survived Pclass \  
1 2 1 1   
3 4 1 1   
6 7 0 1   
10 11 1 3   
11 12 1 1   
21 22 1 2   
23 24 1 1   
27 28 0 1   
52 53 1 1   
54 55 0 1   
62 63 0 1   
66 67 1 2   
75 76 0 3   
88 89 1 1   
92 93 0 1   
96 97 0 1   
97 98 1 1   
102 103 0 1   
110 111 0 1   
118 119 0 1   
123 124 1 2   
124 125 0 1   
136 137 1 1   
137 138 0 1   
139 140 0 1   
148 149 0 2   
151 152 1 1   
  
 Name Sex Age SibSp \  
1 Cumings, Mrs. John Bradley (Florence Briggs Th... female 38.0 1   
3 Futrelle, Mrs. Jacques Heath (Lily May Peel) female 35.0 1   
6 McCarthy, Mr. Timothy J male 54.0 0   
10 Sandstrom, Miss. Marguerite Rut female 4.0 1   
11 Bonnell, Miss. Elizabeth female 58.0 0   
21 Beesley, Mr. Lawrence male 34.0 0   
23 Sloper, Mr. William Thompson male 28.0 0   
27 Fortune, Mr. Charles Alexander male 19.0 3   
52 Harper, Mrs. Henry Sleeper (Myna Haxtun) female 49.0 1   
54 Ostby, Mr. Engelhart Cornelius male 65.0 0   
62 Harris, Mr. Henry Birkhardt male 45.0 1   
66 Nye, Mrs. (Elizabeth Ramell) female 29.0 0   
75 Moen, Mr. Sigurd Hansen male 25.0 0   
88 Fortune, Miss. Mabel Helen female 23.0 3   
92 Chaffee, Mr. Herbert Fuller male 46.0 1   
96 Goldschmidt, Mr. George B male 71.0 0   
97 Greenfield, Mr. William Bertram male 23.0 0   
102 White, Mr. Richard Frasar male 21.0 0   
110 Porter, Mr. Walter Chamberlain male 47.0 0   
118 Baxter, Mr. Quigg Edmond male 24.0 0   
123 Webber, Miss. Susan female 32.5 0   
124 White, Mr. Percival Wayland male 54.0 0   
136 Newsom, Miss. Helen Monypeny female 19.0 0   
137 Futrelle, Mr. Jacques Heath male 37.0 1   
139 Giglio, Mr. Victor male 24.0 0   
148 Navratil, Mr. Michel ("Louis M Hoffman") male 36.5 0   
151 Pears, Mrs. Thomas (Edith Wearne) female 22.0 1   
  
 Parch Ticket Fare Cabin Embarked   
1 0 PC 17599 71.2833 C85 C   
3 0 113803 53.1000 C123 S   
6 0 17463 51.8625 E46 S   
10 1 PP 9549 16.7000 G6 S   
11 0 113783 26.5500 C103 S   
21 0 248698 13.0000 D56 S   
23 0 113788 35.5000 A6 S   
27 2 19950 263.0000 C23 C25 C27 S   
52 0 PC 17572 76.7292 D33 C   
54 1 113509 61.9792 B30 C   
62 0 36973 83.4750 C83 S   
66 0 C.A. 29395 10.5000 F33 S   
75 0 348123 7.6500 F G73 S   
88 2 19950 263.0000 C23 C25 C27 S   
92 0 W.E.P. 5734 61.1750 E31 S   
96 0 PC 17754 34.6542 A5 C   
97 1 PC 17759 63.3583 D10 D12 C   
102 1 35281 77.2875 D26 S   
110 0 110465 52.0000 C110 S   
118 1 PC 17558 247.5208 B58 B60 C   
123 0 27267 13.0000 E101 S   
124 1 35281 77.2875 D26 S   
136 2 11752 26.2833 D47 S   
137 0 113803 53.1000 C123 S   
139 0 PC 17593 79.2000 B86 C   
148 2 230080 26.0000 F2 S   
151 0 113776 66.6000 C2 S

Select relevant columns from Titanic dataset (mix of nominal and ratio-scaled)

titanic\_df = titanic\_df[["Age", "Sex", "Pclass", "Fare", "Embarked"]]  
titanic\_df

Age Sex Pclass Fare Embarked  
1 38.0 female 1 71.2833 C  
3 35.0 female 1 53.1000 S  
6 54.0 male 1 51.8625 S  
10 4.0 female 3 16.7000 S  
11 58.0 female 1 26.5500 S  
21 34.0 male 2 13.0000 S  
23 28.0 male 1 35.5000 S  
27 19.0 male 1 263.0000 S  
52 49.0 female 1 76.7292 C  
54 65.0 male 1 61.9792 C  
62 45.0 male 1 83.4750 S  
66 29.0 female 2 10.5000 S  
75 25.0 male 3 7.6500 S  
88 23.0 female 1 263.0000 S  
92 46.0 male 1 61.1750 S  
96 71.0 male 1 34.6542 C  
97 23.0 male 1 63.3583 C  
102 21.0 male 1 77.2875 S  
110 47.0 male 1 52.0000 S  
118 24.0 male 1 247.5208 C  
123 32.5 female 2 13.0000 S  
124 54.0 male 1 77.2875 S  
136 19.0 female 1 26.2833 S  
137 37.0 male 1 53.1000 S  
139 24.0 male 1 79.2000 C  
148 36.5 male 2 26.0000 S  
151 22.0 female 1 66.6000 S

Encode Nominal as Integers for processing

label\_encoders\_titanic = {}  
**for** column **in** titanic\_df.columns:  
 **if** titanic\_df[column].dtype == object:  
 le = LabelEncoder()  
 titanic\_df[column] = le.fit\_transform(titanic\_df[column])  
 label\_encoders[column] = le  
  
titanic\_df

C:\Users\debat\AppData\Local\Temp\ipykernel\_15644\3305425594.py:5: SettingWithCopyWarning:   
A value is trying to be set on a copy of a slice from a DataFrame.  
Try using .loc[row\_indexer,col\_indexer] = value instead  
  
See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user\_guide/indexing.html#returning-a-view-versus-a-copy  
 titanic\_df[column] = le.fit\_transform(titanic\_df[column])  
C:\Users\debat\AppData\Local\Temp\ipykernel\_15644\3305425594.py:5: SettingWithCopyWarning:   
A value is trying to be set on a copy of a slice from a DataFrame.  
Try using .loc[row\_indexer,col\_indexer] = value instead  
  
See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user\_guide/indexing.html#returning-a-view-versus-a-copy  
 titanic\_df[column] = le.fit\_transform(titanic\_df[column])

Age Sex Pclass Fare Embarked  
1 38.0 0 1 71.2833 0  
3 35.0 0 1 53.1000 1  
6 54.0 1 1 51.8625 1  
10 4.0 0 3 16.7000 1  
11 58.0 0 1 26.5500 1  
21 34.0 1 2 13.0000 1  
23 28.0 1 1 35.5000 1  
27 19.0 1 1 263.0000 1  
52 49.0 0 1 76.7292 0  
54 65.0 1 1 61.9792 0  
62 45.0 1 1 83.4750 1  
66 29.0 0 2 10.5000 1  
75 25.0 1 3 7.6500 1  
88 23.0 0 1 263.0000 1  
92 46.0 1 1 61.1750 1  
96 71.0 1 1 34.6542 0  
97 23.0 1 1 63.3583 0  
102 21.0 1 1 77.2875 1  
110 47.0 1 1 52.0000 1  
118 24.0 1 1 247.5208 0  
123 32.5 0 2 13.0000 1  
124 54.0 1 1 77.2875 1  
136 19.0 0 1 26.2833 1  
137 37.0 1 1 53.1000 1  
139 24.0 1 1 79.2000 0  
148 36.5 1 2 26.0000 1  
151 22.0 0 1 66.6000 1

Combine the datasets into a list for further processing

*# Combine the datasets into a list for further processing*  
datasets = {  
 "Adult Dataset": adult\_df,  
 "Titanic Dataset": titanic\_df  
}

Compute Cosine Similarity

**def** cosine\_similarity(a, b):  
 *"""Calculate the Cosine Similarity between two vectors."""*  
 **try**:  
 **return** 1 - cosine(a, b) *# scipy returns distance, so we subtract from 1 to get similarity*  
 **except** Exception **as** e:  
 **return** np.nan  
  
*# Function to create the Cosine Similarity matrix*  
**def** calculate\_cosine\_similarity\_matrix(dataset):  
 n = len(dataset)  
 cosine\_matrix = np.zeros((n, n))  
   
 **for** i **in** range(n):  
 **for** j **in** range(n):  
 cosine\_matrix[i, j] = cosine\_similarity(dataset.iloc[i].values, dataset.iloc[j].values)  
   
 **return** pd.DataFrame(cosine\_matrix)

Calculate Cosine SImilarity

*For Adult Dataset*

cosine\_matrix\_adult = calculate\_cosine\_similarity\_matrix(adult\_df)  
cosine\_matrix\_adult

0 1 2 3 4 5 6 \  
0 1.000000 0.996120 0.992260 0.967533 0.991062 0.993478 0.969143   
1 0.996120 1.000000 0.995356 0.985101 0.980434 0.987759 0.986363   
2 0.992260 0.995356 1.000000 0.973431 0.980291 0.991643 0.980816   
3 0.967533 0.985101 0.973431 1.000000 0.934177 0.948206 0.997445   
4 0.991062 0.980434 0.980291 0.934177 1.000000 0.996981 0.936376   
.. ... ... ... ... ... ... ...   
95 0.976034 0.968522 0.984786 0.919479 0.983085 0.991265 0.933007   
96 0.998883 0.997556 0.993527 0.971956 0.991562 0.994680 0.972707   
97 0.987173 0.985626 0.996606 0.951403 0.982858 0.993749 0.962654   
98 0.991174 0.998577 0.991648 0.990072 0.974700 0.982658 0.989681   
99 0.987921 0.989193 0.997422 0.959821 0.984550 0.994882 0.968122   
  
 7 8 9 ... 90 91 92 93 \  
0 0.987947 0.986347 0.996862 ... 0.985347 0.986888 0.975612 0.991905   
1 0.996085 0.974868 0.998307 ... 0.996532 0.985455 0.965694 0.988742   
2 0.996382 0.980890 0.995442 ... 0.991265 0.996296 0.981828 0.996523   
3 0.987336 0.923486 0.976317 ... 0.994705 0.951232 0.913116 0.953271   
4 0.965545 0.997933 0.988638 ... 0.963205 0.983161 0.983478 0.988883   
.. ... ... ... ... ... ... ... ...   
95 0.967070 0.991735 0.976569 ... 0.954591 0.994996 0.998900 0.994023   
96 0.989034 0.986548 0.999319 ... 0.989125 0.987346 0.974227 0.991879   
97 0.986846 0.987841 0.988844 ... 0.977234 0.999692 0.993583 0.998922   
98 0.994338 0.967490 0.996608 ... 0.998355 0.979346 0.955547 0.982578   
99 0.988367 0.987459 0.993277 ... 0.983371 0.997737 0.988346 0.997355   
  
 94 95 96 97 98 99   
0 0.994913 0.976034 0.998883 0.987173 0.991174 0.987921   
1 0.991649 0.968522 0.997556 0.985626 0.998577 0.989193   
2 0.990753 0.984786 0.993527 0.996606 0.991648 0.997422   
3 0.958745 0.919479 0.971956 0.951403 0.990072 0.959821   
4 0.995978 0.983085 0.991562 0.982858 0.974700 0.984550   
.. ... ... ... ... ... ...   
95 0.982860 1.000000 0.976136 0.995441 0.959385 0.992139   
96 0.997575 0.976136 1.000000 0.987524 0.994687 0.990804   
97 0.988614 0.995441 0.987524 1.000000 0.979044 0.998150   
98 0.988623 0.959385 0.994687 0.979044 1.000000 0.985352   
99 0.993014 0.992139 0.990804 0.998150 0.985352 1.000000   
  
[100 rows x 100 columns]

*For TItanic Dataset*

cosine\_matrix\_titanic = calculate\_cosine\_similarity\_matrix(titanic\_df)  
cosine\_matrix\_titanic

0 1 2 3 4 5 6 \  
0 1.000000 0.995550 0.950379 0.953797 0.794964 0.753466 0.983664   
1 0.995550 1.000000 0.975197 0.927993 0.847999 0.811493 0.996103   
2 0.950379 0.975197 1.000000 0.830554 0.944041 0.920198 0.990421   
3 0.953797 0.927993 0.830554 1.000000 0.609816 0.565279 0.897217   
4 0.794964 0.847999 0.944041 0.609816 1.000000 0.996740 0.889719   
5 0.753466 0.811493 0.920198 0.565279 0.996740 1.000000 0.858714   
6 0.983664 0.996103 0.990421 0.897217 0.889719 0.858714 1.000000   
7 0.914004 0.872311 0.742794 0.971280 0.480649 0.422964 0.827375   
8 0.996915 0.999761 0.971814 0.931194 0.840094 0.802464 0.994507   
9 0.949365 0.974291 0.999899 0.827611 0.945111 0.921172 0.989670   
10 0.999876 0.996019 0.951987 0.952835 0.797846 0.756918 0.984815   
11 0.741537 0.800941 0.912633 0.552546 0.995315 0.999406 0.848895   
12 0.703764 0.766632 0.887558 0.516672 0.985513 0.995827 0.818987   
13 0.920035 0.879602 0.752778 0.973587 0.493846 0.436473 0.835675   
14 0.987848 0.997989 0.987099 0.904854 0.879097 0.846107 0.999611   
15 0.809779 0.860649 0.951917 0.627451 0.999486 0.994687 0.900684   
16 0.989890 0.972398 0.897178 0.979511 0.701412 0.654025 0.949109   
17 0.974767 0.949966 0.857635 0.985645 0.640159 0.589457 0.920013   
18 0.969902 0.988348 0.997505 0.866148 0.918423 0.890551 0.997659   
19 0.923689 0.883987 0.758953 0.974731 0.501963 0.444933 0.840815   
20 0.763595 0.820538 0.925919 0.578173 0.997874 0.999505 0.866089   
21 0.992686 0.999551 0.980901 0.917484 0.861941 0.826884 0.998122   
22 0.990109 0.998842 0.983530 0.915496 0.869931 0.836496 0.998803   
23 0.992710 0.999548 0.980661 0.918877 0.861284 0.826546 0.998180   
24 0.980876 0.958475 0.872046 0.983677 0.661958 0.612193 0.930736   
25 0.894276 0.932092 0.988775 0.749304 0.981410 0.968018 0.959776   
26 0.985364 0.965439 0.883920 0.983066 0.680559 0.631707 0.939538   
  
 7 8 9 ... 17 18 19 20 \  
0 0.914004 0.996915 0.949365 ... 0.974767 0.969902 0.923689 0.763595   
1 0.872311 0.999761 0.974291 ... 0.949966 0.988348 0.883987 0.820538   
2 0.742794 0.971814 0.999899 ... 0.857635 0.997505 0.758953 0.925919   
3 0.971280 0.931194 0.827611 ... 0.985645 0.866148 0.974731 0.578173   
4 0.480649 0.840094 0.945111 ... 0.640159 0.918423 0.501963 0.997874   
5 0.422964 0.802464 0.921172 ... 0.589457 0.890551 0.444933 0.999505   
6 0.827375 0.994507 0.989670 ... 0.920013 0.997659 0.840815 0.866089   
7 1.000000 0.879364 0.740421 ... 0.981288 0.788166 0.999692 0.436753   
8 0.879364 1.000000 0.971077 ... 0.954297 0.985949 0.890788 0.811660   
9 0.740421 0.971077 1.000000 ... 0.855681 0.997135 0.756698 0.926885   
10 0.912095 0.997160 0.950829 ... 0.973872 0.971209 0.921846 0.766711   
11 0.406600 0.791583 0.913638 ... 0.574645 0.881853 0.428725 0.999417   
12 0.358592 0.756428 0.888446 ... 0.531106 0.853892 0.381081 0.994150   
13 0.999879 0.886464 0.750455 ... 0.984037 0.797324 0.999940 0.450268   
14 0.840397 0.996910 0.986426 ... 0.928908 0.995935 0.853398 0.854003   
15 0.502247 0.853312 0.953092 ... 0.658975 0.927929 0.523320 0.995829   
16 0.962005 0.975764 0.895647 ... 0.996483 0.926090 0.968415 0.665266   
17 0.981288 0.954297 0.855681 ... 1.000000 0.891790 0.985667 0.601552   
18 0.788166 0.985949 0.997135 ... 0.891790 1.000000 0.802980 0.897225   
19 0.999692 0.890788 0.756698 ... 0.985667 0.802980 1.000000 0.458551   
20 0.436753 0.811660 0.926885 ... 0.601552 0.897225 0.458551 1.000000   
21 0.858832 0.999026 0.980147 ... 0.941250 0.992170 0.871109 0.835285   
22 0.849936 0.997672 0.982543 ... 0.935341 0.993696 0.862468 0.845044   
23 0.859375 0.998942 0.979817 ... 0.941682 0.992030 0.871606 0.834844   
24 0.975362 0.962606 0.870352 ... 0.999504 0.904389 0.980484 0.624022   
25 0.636737 0.926405 0.988988 ... 0.773225 0.976086 0.655360 0.971460   
26 0.969558 0.968991 0.882138 ... 0.998476 0.914699 0.975217 0.643654   
  
 21 22 23 24 25 26   
0 0.992686 0.990109 0.992710 0.980876 0.894276 0.985364   
1 0.999551 0.998842 0.999548 0.958475 0.932092 0.965439   
2 0.980901 0.983530 0.980661 0.872046 0.988775 0.883920   
3 0.917484 0.915496 0.918877 0.983677 0.749304 0.983066   
4 0.861941 0.869931 0.861284 0.661958 0.981410 0.680559   
5 0.826884 0.836496 0.826546 0.612193 0.968018 0.631707   
6 0.998122 0.998803 0.998180 0.930736 0.959776 0.939538   
7 0.858832 0.849936 0.859375 0.975362 0.636737 0.969558   
8 0.999026 0.997672 0.998942 0.962606 0.926405 0.968991   
9 0.980147 0.982543 0.979817 0.870352 0.988988 0.882138   
10 0.993348 0.990909 0.993459 0.979977 0.896648 0.984597   
11 0.816352 0.826794 0.815936 0.597622 0.963001 0.617824   
12 0.783153 0.794927 0.783146 0.554669 0.946423 0.575226   
13 0.866438 0.857785 0.866945 0.978540 0.648211 0.973147   
14 0.999382 0.999423 0.999338 0.939137 0.952714 0.947365   
15 0.874245 0.881368 0.873584 0.680496 0.985656 0.698297   
16 0.965910 0.961020 0.966199 0.998536 0.822869 0.999363   
17 0.941250 0.935341 0.941682 0.999504 0.773225 0.998476   
18 0.992170 0.993696 0.992030 0.904389 0.976086 0.914699   
19 0.871109 0.862468 0.871606 0.980484 0.655360 0.975217   
20 0.835285 0.845044 0.834844 0.624022 0.971460 0.643654   
21 1.000000 0.999406 0.999964 0.950564 0.941469 0.957961   
22 0.999406 1.000000 0.999494 0.944830 0.947106 0.953060   
23 0.999964 0.999494 1.000000 0.950895 0.941296 0.958281   
24 0.950564 0.944830 0.950895 1.000000 0.790968 0.999515   
25 0.941469 0.947106 0.941296 0.790968 1.000000 0.805963   
26 0.957961 0.953060 0.958281 0.999515 0.805963 1.000000   
  
[27 rows x 27 columns]

Explanation

Cosine Similarity Calculation: This metric focuses on the orientation of the data points in a multi-dimensional space. A value of 1 means that the vectors are identical, while 0 indicates orthogonal vectors (no similarity), and -1 indicates opposite vectors.

Handling Different Data Types: Cosine similarity works best with interval and ratio-scaled data, as it considers the direction (or relative distribution) rather than the magnitude of the data points.

Observation and Analysis

The resulting matrices will show the similarity between data points based on the angle between their corresponding vectors. This is especially useful in text mining and other applications where the magnitude of features is less important than their relative importance.

Cosine Similarity is not affected by the scale of the data, making it useful when the attributes have been normalized or when dealing with sparse data.

Euclidean Distance

Eucledian Distance

This metric is the most common distance measure and is defined as the straight-line distance between two points in a multi-dimensional space. It works well with interval and ratio-scaled data.

Importing required Libraries

**import** pandas **as** pd  
**import** numpy **as** np  
**from** sklearn.preprocessing **import** LabelEncoder  
**from** scipy.spatial.distance **import** euclidean

Load Datasets

*# Load datasets*  
adult\_df = pd.read\_csv("../adult/adult\_trim.data", header=None) *# No header*  
titanic\_df = pd.read\_csv('../titanic/titanic\_trim.csv') *# Has header*  
  
*# Rename columns for clarity*  
adult\_df.columns = ["age", "workclass", "fnlwgt", "education", "education\_num",   
 "marital\_status", "occupation", "relationship", "race", "sex",   
 "capital\_gain", "capital\_loss", "hours\_per\_week", "native\_country", "income"]  
adult\_df.dropna(inplace=True)

adult\_df

age workclass fnlwgt education education\_num \  
0 39 State-gov 77516 Bachelors 13   
1 50 Self-emp-not-inc 83311 Bachelors 13   
2 38 Private 215646 HS-grad 9   
3 53 Private 234721 11th 7   
4 28 Private 338409 Bachelors 13   
.. ... ... ... ... ...   
95 29 Local-gov 115585 Some-college 10   
96 48 Self-emp-not-inc 191277 Doctorate 16   
97 37 Private 202683 Some-college 10   
98 48 Private 171095 Assoc-acdm 12   
99 32 Federal-gov 249409 HS-grad 9   
  
 marital\_status occupation relationship race sex \  
0 Never-married Adm-clerical Not-in-family White Male   
1 Married-civ-spouse Exec-managerial Husband White Male   
2 Divorced Handlers-cleaners Not-in-family White Male   
3 Married-civ-spouse Handlers-cleaners Husband Black Male   
4 Married-civ-spouse Prof-specialty Wife Black Female   
.. ... ... ... ... ...   
95 Never-married Handlers-cleaners Not-in-family White Male   
96 Married-civ-spouse Prof-specialty Husband White Male   
97 Married-civ-spouse Sales Husband White Male   
98 Divorced Exec-managerial Unmarried White Female   
99 Never-married Other-service Own-child Black Male   
  
 capital\_gain capital\_loss hours\_per\_week native\_country income   
0 2174 0 40 United-States <=50K   
1 0 0 13 United-States <=50K   
2 0 0 40 United-States <=50K   
3 0 0 40 United-States <=50K   
4 0 0 40 Cuba <=50K   
.. ... ... ... ... ...   
95 0 0 50 United-States <=50K   
96 0 1902 60 United-States >50K   
97 0 0 48 United-States >50K   
98 0 0 40 England <=50K   
99 0 0 40 United-States <=50K   
  
[100 rows x 15 columns]

titanic\_df

PassengerId Survived Pclass \  
0 1 0 3   
1 2 1 1   
2 3 1 3   
3 4 1 1   
4 5 0 3   
.. ... ... ...   
150 151 0 2   
151 152 1 1   
152 153 0 3   
153 154 0 3   
154 155 0 3   
  
 Name Sex Age SibSp \  
0 Braund, Mr. Owen Harris male 22.0 1   
1 Cumings, Mrs. John Bradley (Florence Briggs Th... female 38.0 1   
2 Heikkinen, Miss. Laina female 26.0 0   
3 Futrelle, Mrs. Jacques Heath (Lily May Peel) female 35.0 1   
4 Allen, Mr. William Henry male 35.0 0   
.. ... ... ... ...   
150 Bateman, Rev. Robert James male 51.0 0   
151 Pears, Mrs. Thomas (Edith Wearne) female 22.0 1   
152 Meo, Mr. Alfonzo male 55.5 0   
153 van Billiard, Mr. Austin Blyler male 40.5 0   
154 Olsen, Mr. Ole Martin male NaN 0   
  
 Parch Ticket Fare Cabin Embarked   
0 0 A/5 21171 7.2500 NaN S   
1 0 PC 17599 71.2833 C85 C   
2 0 STON/O2. 3101282 7.9250 NaN S   
3 0 113803 53.1000 C123 S   
4 0 373450 8.0500 NaN S   
.. ... ... ... ... ...   
150 0 S.O.P. 1166 12.5250 NaN S   
151 0 113776 66.6000 C2 S   
152 0 A.5. 11206 8.0500 NaN S   
153 2 A/5. 851 14.5000 NaN S   
154 0 Fa 265302 7.3125 NaN S   
  
[155 rows x 12 columns]

Select relevant columns from Adult dataset (mix of nominal and ratio-scaled)

adult\_df = adult\_df[["age", "workclass", "education", "education\_num", "sex"]]  
  
adult\_df

age workclass education education\_num sex  
0 39 State-gov Bachelors 13 Male  
1 50 Self-emp-not-inc Bachelors 13 Male  
2 38 Private HS-grad 9 Male  
3 53 Private 11th 7 Male  
4 28 Private Bachelors 13 Female  
.. ... ... ... ... ...  
95 29 Local-gov Some-college 10 Male  
96 48 Self-emp-not-inc Doctorate 16 Male  
97 37 Private Some-college 10 Male  
98 48 Private Assoc-acdm 12 Female  
99 32 Federal-gov HS-grad 9 Male  
  
[100 rows x 5 columns]

Encode nominal attributes as integers for processing

label\_encoders = {}  
**for** column **in** adult\_df.columns:  
 **if** adult\_df[column].dtype == object:  
 le = LabelEncoder()  
 adult\_df[column] = le.fit\_transform(adult\_df[column])  
 label\_encoders[column] = le  
  
adult\_df

C:\Users\debat\AppData\Local\Temp\ipykernel\_11476\183426126.py:5: SettingWithCopyWarning:   
A value is trying to be set on a copy of a slice from a DataFrame.  
Try using .loc[row\_indexer,col\_indexer] = value instead  
  
See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user\_guide/indexing.html#returning-a-view-versus-a-copy  
 adult\_df[column] = le.fit\_transform(adult\_df[column])  
C:\Users\debat\AppData\Local\Temp\ipykernel\_11476\183426126.py:5: SettingWithCopyWarning:   
A value is trying to be set on a copy of a slice from a DataFrame.  
Try using .loc[row\_indexer,col\_indexer] = value instead  
  
See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user\_guide/indexing.html#returning-a-view-versus-a-copy  
 adult\_df[column] = le.fit\_transform(adult\_df[column])  
C:\Users\debat\AppData\Local\Temp\ipykernel\_11476\183426126.py:5: SettingWithCopyWarning:   
A value is trying to be set on a copy of a slice from a DataFrame.  
Try using .loc[row\_indexer,col\_indexer] = value instead  
  
See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user\_guide/indexing.html#returning-a-view-versus-a-copy  
 adult\_df[column] = le.fit\_transform(adult\_df[column])

age workclass education education\_num sex  
0 39 6 7 13 1  
1 50 5 7 13 1  
2 38 3 9 9 1  
3 53 3 1 7 1  
4 28 3 7 13 0  
.. ... ... ... ... ...  
95 29 2 12 10 1  
96 48 5 8 16 1  
97 37 3 12 10 1  
98 48 3 5 12 0  
99 32 1 9 9 1  
  
[100 rows x 5 columns]

Clean and preprocess Titanic dataset

titanic\_df.dropna(inplace=True)  
titanic\_df

PassengerId Survived Pclass \  
1 2 1 1   
3 4 1 1   
6 7 0 1   
10 11 1 3   
11 12 1 1   
21 22 1 2   
23 24 1 1   
27 28 0 1   
52 53 1 1   
54 55 0 1   
62 63 0 1   
66 67 1 2   
75 76 0 3   
88 89 1 1   
92 93 0 1   
96 97 0 1   
97 98 1 1   
102 103 0 1   
110 111 0 1   
118 119 0 1   
123 124 1 2   
124 125 0 1   
136 137 1 1   
137 138 0 1   
139 140 0 1   
148 149 0 2   
151 152 1 1   
  
 Name Sex Age SibSp \  
1 Cumings, Mrs. John Bradley (Florence Briggs Th... female 38.0 1   
3 Futrelle, Mrs. Jacques Heath (Lily May Peel) female 35.0 1   
6 McCarthy, Mr. Timothy J male 54.0 0   
10 Sandstrom, Miss. Marguerite Rut female 4.0 1   
11 Bonnell, Miss. Elizabeth female 58.0 0   
21 Beesley, Mr. Lawrence male 34.0 0   
23 Sloper, Mr. William Thompson male 28.0 0   
27 Fortune, Mr. Charles Alexander male 19.0 3   
52 Harper, Mrs. Henry Sleeper (Myna Haxtun) female 49.0 1   
54 Ostby, Mr. Engelhart Cornelius male 65.0 0   
62 Harris, Mr. Henry Birkhardt male 45.0 1   
66 Nye, Mrs. (Elizabeth Ramell) female 29.0 0   
75 Moen, Mr. Sigurd Hansen male 25.0 0   
88 Fortune, Miss. Mabel Helen female 23.0 3   
92 Chaffee, Mr. Herbert Fuller male 46.0 1   
96 Goldschmidt, Mr. George B male 71.0 0   
97 Greenfield, Mr. William Bertram male 23.0 0   
102 White, Mr. Richard Frasar male 21.0 0   
110 Porter, Mr. Walter Chamberlain male 47.0 0   
118 Baxter, Mr. Quigg Edmond male 24.0 0   
123 Webber, Miss. Susan female 32.5 0   
124 White, Mr. Percival Wayland male 54.0 0   
136 Newsom, Miss. Helen Monypeny female 19.0 0   
137 Futrelle, Mr. Jacques Heath male 37.0 1   
139 Giglio, Mr. Victor male 24.0 0   
148 Navratil, Mr. Michel ("Louis M Hoffman") male 36.5 0   
151 Pears, Mrs. Thomas (Edith Wearne) female 22.0 1   
  
 Parch Ticket Fare Cabin Embarked   
1 0 PC 17599 71.2833 C85 C   
3 0 113803 53.1000 C123 S   
6 0 17463 51.8625 E46 S   
10 1 PP 9549 16.7000 G6 S   
11 0 113783 26.5500 C103 S   
21 0 248698 13.0000 D56 S   
23 0 113788 35.5000 A6 S   
27 2 19950 263.0000 C23 C25 C27 S   
52 0 PC 17572 76.7292 D33 C   
54 1 113509 61.9792 B30 C   
62 0 36973 83.4750 C83 S   
66 0 C.A. 29395 10.5000 F33 S   
75 0 348123 7.6500 F G73 S   
88 2 19950 263.0000 C23 C25 C27 S   
92 0 W.E.P. 5734 61.1750 E31 S   
96 0 PC 17754 34.6542 A5 C   
97 1 PC 17759 63.3583 D10 D12 C   
102 1 35281 77.2875 D26 S   
110 0 110465 52.0000 C110 S   
118 1 PC 17558 247.5208 B58 B60 C   
123 0 27267 13.0000 E101 S   
124 1 35281 77.2875 D26 S   
136 2 11752 26.2833 D47 S   
137 0 113803 53.1000 C123 S   
139 0 PC 17593 79.2000 B86 C   
148 2 230080 26.0000 F2 S   
151 0 113776 66.6000 C2 S

Select relevant columns from Titanic dataset (mix of nominal and ratio-scaled)

titanic\_df = titanic\_df[["Age", "Sex", "Pclass", "Fare", "Embarked"]]  
titanic\_df

Age Sex Pclass Fare Embarked  
1 38.0 female 1 71.2833 C  
3 35.0 female 1 53.1000 S  
6 54.0 male 1 51.8625 S  
10 4.0 female 3 16.7000 S  
11 58.0 female 1 26.5500 S  
21 34.0 male 2 13.0000 S  
23 28.0 male 1 35.5000 S  
27 19.0 male 1 263.0000 S  
52 49.0 female 1 76.7292 C  
54 65.0 male 1 61.9792 C  
62 45.0 male 1 83.4750 S  
66 29.0 female 2 10.5000 S  
75 25.0 male 3 7.6500 S  
88 23.0 female 1 263.0000 S  
92 46.0 male 1 61.1750 S  
96 71.0 male 1 34.6542 C  
97 23.0 male 1 63.3583 C  
102 21.0 male 1 77.2875 S  
110 47.0 male 1 52.0000 S  
118 24.0 male 1 247.5208 C  
123 32.5 female 2 13.0000 S  
124 54.0 male 1 77.2875 S  
136 19.0 female 1 26.2833 S  
137 37.0 male 1 53.1000 S  
139 24.0 male 1 79.2000 C  
148 36.5 male 2 26.0000 S  
151 22.0 female 1 66.6000 S

Encode Nominal as Integers for processing

label\_encoders\_titanic = {}  
**for** column **in** titanic\_df.columns:  
 **if** titanic\_df[column].dtype == object:  
 le = LabelEncoder()  
 titanic\_df[column] = le.fit\_transform(titanic\_df[column])  
 label\_encoders[column] = le  
  
titanic\_df

C:\Users\debat\AppData\Local\Temp\ipykernel\_11476\3305425594.py:5: SettingWithCopyWarning:   
A value is trying to be set on a copy of a slice from a DataFrame.  
Try using .loc[row\_indexer,col\_indexer] = value instead  
  
See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user\_guide/indexing.html#returning-a-view-versus-a-copy  
 titanic\_df[column] = le.fit\_transform(titanic\_df[column])  
C:\Users\debat\AppData\Local\Temp\ipykernel\_11476\3305425594.py:5: SettingWithCopyWarning:   
A value is trying to be set on a copy of a slice from a DataFrame.  
Try using .loc[row\_indexer,col\_indexer] = value instead  
  
See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user\_guide/indexing.html#returning-a-view-versus-a-copy  
 titanic\_df[column] = le.fit\_transform(titanic\_df[column])

Age Sex Pclass Fare Embarked  
1 38.0 0 1 71.2833 0  
3 35.0 0 1 53.1000 1  
6 54.0 1 1 51.8625 1  
10 4.0 0 3 16.7000 1  
11 58.0 0 1 26.5500 1  
21 34.0 1 2 13.0000 1  
23 28.0 1 1 35.5000 1  
27 19.0 1 1 263.0000 1  
52 49.0 0 1 76.7292 0  
54 65.0 1 1 61.9792 0  
62 45.0 1 1 83.4750 1  
66 29.0 0 2 10.5000 1  
75 25.0 1 3 7.6500 1  
88 23.0 0 1 263.0000 1  
92 46.0 1 1 61.1750 1  
96 71.0 1 1 34.6542 0  
97 23.0 1 1 63.3583 0  
102 21.0 1 1 77.2875 1  
110 47.0 1 1 52.0000 1  
118 24.0 1 1 247.5208 0  
123 32.5 0 2 13.0000 1  
124 54.0 1 1 77.2875 1  
136 19.0 0 1 26.2833 1  
137 37.0 1 1 53.1000 1  
139 24.0 1 1 79.2000 0  
148 36.5 1 2 26.0000 1  
151 22.0 0 1 66.6000 1

Combine the datasets into a list for further processing

*# Combine the datasets into a list for further processing*  
datasets = {  
 "Adult Dataset": adult\_df,  
 "Titanic Dataset": titanic\_df  
}

Compute Eucledian Distance

**def** euclidean\_distance(a, b):  
 *"""Calculate the Euclidean Distance between two vectors."""*  
 **try**:  
 **return** euclidean(a, b)  
 **except** Exception **as** e:  
 **return** np.nan  
  
*# Function to create the Euclidean Distance matrix*  
**def** calculate\_euclidean\_matrix(dataset):  
 n = len(dataset)  
 euclidean\_matrix = np.zeros((n, n))  
   
 **for** i **in** range(n):  
 **for** j **in** range(n):  
 euclidean\_matrix[i, j] = euclidean\_distance(dataset.iloc[i].values, dataset.iloc[j].values)  
   
 **return** pd.DataFrame(euclidean\_matrix)

Calculate Eucledian Distance

*For Adult Dataset*

euclidean\_matrix\_adult = calculate\_euclidean\_matrix(adult\_df)  
euclidean\_matrix\_adult

0 1 2 3 4 5 \  
0 0.000000 11.045361 5.477226 16.643317 11.445523 4.898979   
1 11.045361 0.000000 12.961481 9.219544 22.113344 13.564660   
2 5.477226 12.961481 0.000000 17.117243 11.000000 5.291503   
3 16.643317 9.219544 17.117243 0.000000 26.419690 19.672316   
4 11.445523 22.113344 11.000000 26.419690 0.000000 9.539392   
.. ... ... ... ... ... ...   
95 12.247449 22.000000 9.591663 26.589472 6.082763 9.273618   
96 9.591663 3.741657 12.409674 12.609520 20.371549 11.575837   
97 6.855655 14.387495 3.316625 19.646883 10.770330 4.582576   
98 9.797959 3.741657 11.224972 8.185353 20.124612 12.247449   
99 9.695360 18.973666 6.324555 22.649503 6.403124 7.483315   
  
 6 7 8 9 ... 90 91 \  
0 13.527749 13.784049 9.165151 4.242641 ... 18.384776 6.928203   
1 8.888194 4.898979 19.390719 8.246211 ... 7.615773 14.422205   
2 12.767145 14.142136 8.717798 6.000000 ... 19.339080 3.464102   
3 5.477226 8.544004 24.799194 13.892444 ... 7.549834 19.672316   
4 22.671568 24.515301 4.358899 14.035669 ... 29.103264 10.723805   
.. ... ... ... ... ... ... ...   
95 22.158520 23.409400 5.099020 14.282857 ... 28.670542 8.124038   
96 11.958261 8.124038 17.378147 7.071068 ... 10.677078 13.341664   
97 15.297059 15.459625 7.549834 7.681146 ... 20.904545 1.000000   
98 7.141428 6.782330 17.832555 6.480741 ... 9.165151 13.190906   
99 18.303005 20.396078 5.656854 11.135529 ... 25.337719 6.324555   
  
 92 93 94 95 96 97 \  
0 12.845233 10.535654 6.403124 12.247449 9.591663 6.855655   
1 22.869193 20.615528 16.278821 22.000000 3.741657 14.387495   
2 10.535654 8.062258 6.082763 9.591663 12.409674 3.316625   
3 27.495454 24.454039 20.832667 26.589472 12.609520 19.646883   
4 5.830952 4.898979 6.164414 6.082763 20.371549 10.770330   
.. ... ... ... ... ... ...   
95 1.732051 3.605551 7.681146 0.000000 20.542639 8.062258   
96 21.377558 19.467922 14.662878 20.542639 0.000000 13.304135   
97 9.055385 7.745967 6.633250 8.062258 13.304135 0.000000   
98 21.283797 18.681542 14.247807 20.396078 5.477226 13.228757   
99 5.567764 3.000000 5.000000 4.472136 17.944358 6.244998   
  
 98 99   
0 9.797959 9.695360   
1 3.741657 18.973666   
2 11.224972 6.324555   
3 8.185353 22.649503   
4 20.124612 6.403124   
.. ... ...   
95 20.396078 4.472136   
96 5.477226 17.944358   
97 13.228757 6.244998   
98 0.000000 16.911535   
99 16.911535 0.000000   
  
[100 rows x 100 columns]

*For Titanic Dataset*

euclidean\_matrix\_titanic = calculate\_euclidean\_matrix(titanic\_df)  
euclidean\_matrix\_titanic

0 1 2 3 4 5 \  
0 0.000000 18.456229 25.202529 64.345448 49.010898 58.446070   
1 18.456229 0.000000 19.066500 47.853527 35.126948 40.137389   
2 25.202529 19.066500 0.000000 61.166996 25.646104 43.718347   
3 64.345448 47.853527 61.166996 0.000000 54.927429 30.260370   
4 49.010898 35.126948 25.646104 54.927429 0.000000 27.597147   
5 58.446070 40.137389 43.718347 30.260370 27.597147 0.000000   
6 37.181239 18.967340 30.720212 30.568611 31.322556 23.307724   
7 192.661083 210.511306 214.018793 246.766469 239.646829 250.451592   
8 12.274275 27.483433 25.403794 75.056678 50.989726 65.493595   
9 28.575624 31.318368 14.978238 76.007933 36.141779 57.982429   
10 14.129315 31.994384 32.868680 78.389417 58.399106 71.335304   
11 61.462261 43.032081 48.351385 25.776734 33.160255 5.678908   
12 64.993822 46.590798 52.912618 22.888917 38.094750 10.517723   
13 192.305208 210.242741 213.403477 247.039855 239.026364 250.245879   
14 12.968336 13.682311 12.276916 61.212953 36.659114 49.657131   
15 49.312179 40.475271 24.210031 69.407156 15.384345 42.894107   
16 16.994282 15.850322 33.077990 50.438051 50.811917 51.565089   
17 18.084535 27.964891 41.658500 62.967016 62.803614 65.596362   
18 21.327111 12.091733 7.001350 55.678452 27.743513 41.121770   
19 176.795521 194.736867 197.947393 231.698601 223.575702 234.738164   
20 58.559312 40.190297 44.435840 28.756564 28.893814 1.802776   
21 17.147898 30.773936 25.425000 78.586546 50.904753 67.334112   
22 48.856934 31.227158 43.362374 17.911997 39.000912 20.085967   
23 18.265607 2.236068 17.044982 49.182924 33.865949 40.224495   
24 16.114408 28.358597 40.599740 65.667724 62.689892 66.965962   
25 45.341231 27.178300 31.242902 33.834007 21.553480 13.238202   
26 16.701296 18.741665 35.244771 53.084932 53.851671 54.945063   
  
 6 7 8 9 ... 17 \  
0 37.181239 192.661083 12.274275 28.575624 ... 18.084535   
1 18.967340 210.511306 27.483433 31.318368 ... 27.964891   
2 30.720212 214.018793 25.403794 14.978238 ... 41.658500   
3 30.568611 246.766469 75.056678 76.007933 ... 62.967016   
4 31.322556 239.646829 50.989726 36.141779 ... 62.803614   
5 23.307724 250.451592 65.493595 57.982429 ... 65.596362   
6 0.000000 227.677952 46.290895 45.509867 ... 42.369743   
7 227.677952 0.000000 188.676472 206.219209 ... 185.723269   
8 46.290895 188.676472 0.000000 21.784455 ... 28.041250   
9 45.509867 206.219209 21.784455 0.000000 ... 46.597683   
10 50.897943 181.397976 7.969054 29.378043 ... 24.784777   
11 25.059928 252.701899 69.197593 62.841929 ... 67.279790   
12 28.082423 255.428312 73.170594 67.503052 ... 69.780953   
13 227.557136 4.123106 188.079268 205.366409 ... 185.725961   
14 31.356110 203.623011 15.903872 19.043286 ... 29.742439   
15 43.019942 234.193946 47.490058 27.975983 ... 65.716043   
16 28.321103 199.684272 29.253734 42.022636 ... 14.107537   
17 42.369743 185.723269 28.041250 46.597683 ... 0.000000   
18 25.164459 212.849712 24.850218 20.605447 ... 36.269238   
19 212.060887 16.297412 172.614515 190.017592 ... 170.262669   
20 22.989128 250.368229 65.845736 58.806565 ... 65.323293   
21 49.215802 188.981832 5.226060 18.877077 ... 33.000000   
22 12.920819 236.718812 58.700842 58.242573 ... 51.053192   
23 19.767650 210.670382 26.539388 29.391158 ... 29.000606   
24 43.894077 183.870715 25.141696 44.469720 ... 3.695627   
25 12.786712 237.647323 52.275250 45.921159 ... 53.587850   
26 31.689273 196.425457 28.854821 43.270681 ... 10.780661   
  
 18 19 20 21 22 23 \  
0 21.327111 176.795521 58.559312 17.147898 48.856934 18.265607   
1 12.091733 194.736867 40.190297 30.773936 31.227158 2.236068   
2 7.001350 197.947393 44.435840 25.425000 43.362374 17.044982   
3 55.678452 231.698601 28.756564 78.586546 17.911997 49.182924   
4 27.743513 223.575702 28.893814 50.904753 39.000912 33.865949   
5 41.121770 234.738164 1.802776 67.334112 20.085967 40.224495   
6 25.164459 212.060887 22.989128 49.215802 12.920819 19.767650   
7 212.849712 16.297412 250.368229 188.981832 236.718812 210.670382   
8 24.850218 172.614515 65.845736 5.226060 58.700842 26.539388   
9 20.605447 190.017592 58.806565 18.877077 58.242573 29.391158   
10 31.538478 165.387498 71.588935 10.921774 62.832241 31.410836   
11 45.257596 237.079859 4.301163 71.327205 18.711295 43.367730   
12 49.547174 239.883306 9.320542 75.461125 19.702788 47.050000   
13 212.362897 15.575803 250.182433 188.284712 236.750493 210.368748   
14 9.229335 187.642632 50.050780 17.989237 44.129703 12.091552   
15 29.628985 217.993554 44.205818 45.908586 52.688442 38.694283   
16 26.570867 184.165215 51.275807 34.000333 37.316962 17.384842   
17 36.269238 170.262669 65.323293 33.000000 51.053192 29.000606   
18 0.000000 196.871489 41.632319 26.238477 38.030891 10.060318   
19 196.871489 0.000000 234.681179 172.859412 221.298512 194.857506   
20 41.632319 234.681179 0.000000 67.802158 18.965655 40.376478   
21 26.238477 172.859412 67.802158 0.000000 61.866214 29.564086   
22 38.030891 221.298512 18.965655 61.866214 0.000000 32.313084   
23 10.060318 194.857506 40.376478 29.564086 32.313084 0.000000   
24 35.634814 168.320800 66.765934 30.077527 53.171206 29.175503   
25 28.057976 221.877702 13.638182 54.200163 17.559335 27.123053   
26 28.968258 180.937381 54.627923 33.752373 40.428162 20.205197   
  
 24 25 26   
0 16.114408 45.341231 16.701296   
1 28.358597 27.178300 18.741665   
2 40.599740 31.242902 35.244771   
3 65.667724 33.834007 53.084932   
4 62.689892 21.553480 53.851671   
5 66.965962 13.238202 54.945063   
6 43.894077 12.786712 31.689273   
7 183.870715 237.647323 196.425457   
8 25.141696 52.275250 28.854821   
9 44.469720 45.921159 43.270681   
10 21.454035 58.108740 28.544100   
11 68.903483 17.248188 56.543877   
12 71.591916 21.678849 59.068625   
13 183.808161 237.388395 196.402546   
14 28.458753 36.449014 24.625812   
15 64.755913 35.596983 58.510975   
16 15.873231 39.747863 3.675407   
17 3.695627 53.587850 10.780661   
18 35.634814 28.057976 28.968258   
19 168.320800 221.877702 180.937381   
20 66.765934 13.638182 54.627923   
21 30.077527 54.200163 33.752373   
22 53.171206 17.559335 40.428162   
23 29.175503 27.123053 20.205197   
24 0.000000 54.667083 12.835887   
25 54.667083 0.000000 43.134789   
26 12.835887 43.134789 0.000000   
  
[27 rows x 27 columns]

Explanation

Euclidean Distance Calculation: This metric is a straightforward calculation of the straight-line distance between two points in multi-dimensional space. It's suitable for interval and ratio-scaled data.

Handling Different Data Types: While Euclidean Distance works best with interval and ratio-scaled data, it may not be meaningful for nominal or ordinal data without preprocessing or encoding.

Observation and Analysis

The resulting matrices will represent the pairwise Euclidean distances between data points. A smaller value indicates that the data points are closer to each other, while a larger value indicates they are further apart.

Euclidean Distance is sensitive to the scale of the data, so if attributes have different units or scales, standardization or normalization is often necessary before applying this metric.

Manhattan Distance

Manhattan Distance

Also known as the L1 norm or taxicab distance, this metric measures the distance between two points by summing the absolute differences of their corresponding coordinates. It’s useful for measuring distance in grid-like paths (e.g., city blocks).

Importing required Libraries

**import** pandas **as** pd  
**import** numpy **as** np  
**from** sklearn.preprocessing **import** LabelEncoder  
**from** scipy.spatial.distance **import** cityblock

Load Datasets

*# Load datasets*  
adult\_df = pd.read\_csv("../adult/adult\_trim.data", header=None) *# No header*  
titanic\_df = pd.read\_csv('../titanic/titanic\_trim.csv') *# Has header*  
  
*# Rename columns for clarity*  
adult\_df.columns = ["age", "workclass", "fnlwgt", "education", "education\_num",   
 "marital\_status", "occupation", "relationship", "race", "sex",   
 "capital\_gain", "capital\_loss", "hours\_per\_week", "native\_country", "income"]  
adult\_df.dropna(inplace=True)

adult\_df

age workclass fnlwgt education education\_num \  
0 39 State-gov 77516 Bachelors 13   
1 50 Self-emp-not-inc 83311 Bachelors 13   
2 38 Private 215646 HS-grad 9   
3 53 Private 234721 11th 7   
4 28 Private 338409 Bachelors 13   
.. ... ... ... ... ...   
95 29 Local-gov 115585 Some-college 10   
96 48 Self-emp-not-inc 191277 Doctorate 16   
97 37 Private 202683 Some-college 10   
98 48 Private 171095 Assoc-acdm 12   
99 32 Federal-gov 249409 HS-grad 9   
  
 marital\_status occupation relationship race sex \  
0 Never-married Adm-clerical Not-in-family White Male   
1 Married-civ-spouse Exec-managerial Husband White Male   
2 Divorced Handlers-cleaners Not-in-family White Male   
3 Married-civ-spouse Handlers-cleaners Husband Black Male   
4 Married-civ-spouse Prof-specialty Wife Black Female   
.. ... ... ... ... ...   
95 Never-married Handlers-cleaners Not-in-family White Male   
96 Married-civ-spouse Prof-specialty Husband White Male   
97 Married-civ-spouse Sales Husband White Male   
98 Divorced Exec-managerial Unmarried White Female   
99 Never-married Other-service Own-child Black Male   
  
 capital\_gain capital\_loss hours\_per\_week native\_country income   
0 2174 0 40 United-States <=50K   
1 0 0 13 United-States <=50K   
2 0 0 40 United-States <=50K   
3 0 0 40 United-States <=50K   
4 0 0 40 Cuba <=50K   
.. ... ... ... ... ...   
95 0 0 50 United-States <=50K   
96 0 1902 60 United-States >50K   
97 0 0 48 United-States >50K   
98 0 0 40 England <=50K   
99 0 0 40 United-States <=50K   
  
[100 rows x 15 columns]

titanic\_df

PassengerId Survived Pclass \  
0 1 0 3   
1 2 1 1   
2 3 1 3   
3 4 1 1   
4 5 0 3   
.. ... ... ...   
150 151 0 2   
151 152 1 1   
152 153 0 3   
153 154 0 3   
154 155 0 3   
  
 Name Sex Age SibSp \  
0 Braund, Mr. Owen Harris male 22.0 1   
1 Cumings, Mrs. John Bradley (Florence Briggs Th... female 38.0 1   
2 Heikkinen, Miss. Laina female 26.0 0   
3 Futrelle, Mrs. Jacques Heath (Lily May Peel) female 35.0 1   
4 Allen, Mr. William Henry male 35.0 0   
.. ... ... ... ...   
150 Bateman, Rev. Robert James male 51.0 0   
151 Pears, Mrs. Thomas (Edith Wearne) female 22.0 1   
152 Meo, Mr. Alfonzo male 55.5 0   
153 van Billiard, Mr. Austin Blyler male 40.5 0   
154 Olsen, Mr. Ole Martin male NaN 0   
  
 Parch Ticket Fare Cabin Embarked   
0 0 A/5 21171 7.2500 NaN S   
1 0 PC 17599 71.2833 C85 C   
2 0 STON/O2. 3101282 7.9250 NaN S   
3 0 113803 53.1000 C123 S   
4 0 373450 8.0500 NaN S   
.. ... ... ... ... ...   
150 0 S.O.P. 1166 12.5250 NaN S   
151 0 113776 66.6000 C2 S   
152 0 A.5. 11206 8.0500 NaN S   
153 2 A/5. 851 14.5000 NaN S   
154 0 Fa 265302 7.3125 NaN S   
  
[155 rows x 12 columns]

Select relevant columns from Adult dataset (mix of nominal and ratio-scaled)

adult\_df = adult\_df[["age", "workclass", "education", "education\_num", "sex"]]  
  
adult\_df

age workclass education education\_num sex  
0 39 State-gov Bachelors 13 Male  
1 50 Self-emp-not-inc Bachelors 13 Male  
2 38 Private HS-grad 9 Male  
3 53 Private 11th 7 Male  
4 28 Private Bachelors 13 Female  
.. ... ... ... ... ...  
95 29 Local-gov Some-college 10 Male  
96 48 Self-emp-not-inc Doctorate 16 Male  
97 37 Private Some-college 10 Male  
98 48 Private Assoc-acdm 12 Female  
99 32 Federal-gov HS-grad 9 Male  
  
[100 rows x 5 columns]

Encode nominal attributes as integers for processing

label\_encoders = {}  
**for** column **in** adult\_df.columns:  
 **if** adult\_df[column].dtype == object:  
 le = LabelEncoder()  
 adult\_df[column] = le.fit\_transform(adult\_df[column])  
 label\_encoders[column] = le  
  
adult\_df

C:\Users\debat\AppData\Local\Temp\ipykernel\_8272\183426126.py:5: SettingWithCopyWarning:   
A value is trying to be set on a copy of a slice from a DataFrame.  
Try using .loc[row\_indexer,col\_indexer] = value instead  
  
See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user\_guide/indexing.html#returning-a-view-versus-a-copy  
 adult\_df[column] = le.fit\_transform(adult\_df[column])  
C:\Users\debat\AppData\Local\Temp\ipykernel\_8272\183426126.py:5: SettingWithCopyWarning:   
A value is trying to be set on a copy of a slice from a DataFrame.  
Try using .loc[row\_indexer,col\_indexer] = value instead  
  
See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user\_guide/indexing.html#returning-a-view-versus-a-copy  
 adult\_df[column] = le.fit\_transform(adult\_df[column])  
C:\Users\debat\AppData\Local\Temp\ipykernel\_8272\183426126.py:5: SettingWithCopyWarning:   
A value is trying to be set on a copy of a slice from a DataFrame.  
Try using .loc[row\_indexer,col\_indexer] = value instead  
  
See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user\_guide/indexing.html#returning-a-view-versus-a-copy  
 adult\_df[column] = le.fit\_transform(adult\_df[column])

age workclass education education\_num sex  
0 39 6 7 13 1  
1 50 5 7 13 1  
2 38 3 9 9 1  
3 53 3 1 7 1  
4 28 3 7 13 0  
.. ... ... ... ... ...  
95 29 2 12 10 1  
96 48 5 8 16 1  
97 37 3 12 10 1  
98 48 3 5 12 0  
99 32 1 9 9 1  
  
[100 rows x 5 columns]

Clean and preprocess Titanic dataset

titanic\_df.dropna(inplace=True)  
titanic\_df

PassengerId Survived Pclass \  
1 2 1 1   
3 4 1 1   
6 7 0 1   
10 11 1 3   
11 12 1 1   
21 22 1 2   
23 24 1 1   
27 28 0 1   
52 53 1 1   
54 55 0 1   
62 63 0 1   
66 67 1 2   
75 76 0 3   
88 89 1 1   
92 93 0 1   
96 97 0 1   
97 98 1 1   
102 103 0 1   
110 111 0 1   
118 119 0 1   
123 124 1 2   
124 125 0 1   
136 137 1 1   
137 138 0 1   
139 140 0 1   
148 149 0 2   
151 152 1 1   
  
 Name Sex Age SibSp \  
1 Cumings, Mrs. John Bradley (Florence Briggs Th... female 38.0 1   
3 Futrelle, Mrs. Jacques Heath (Lily May Peel) female 35.0 1   
6 McCarthy, Mr. Timothy J male 54.0 0   
10 Sandstrom, Miss. Marguerite Rut female 4.0 1   
11 Bonnell, Miss. Elizabeth female 58.0 0   
21 Beesley, Mr. Lawrence male 34.0 0   
23 Sloper, Mr. William Thompson male 28.0 0   
27 Fortune, Mr. Charles Alexander male 19.0 3   
52 Harper, Mrs. Henry Sleeper (Myna Haxtun) female 49.0 1   
54 Ostby, Mr. Engelhart Cornelius male 65.0 0   
62 Harris, Mr. Henry Birkhardt male 45.0 1   
66 Nye, Mrs. (Elizabeth Ramell) female 29.0 0   
75 Moen, Mr. Sigurd Hansen male 25.0 0   
88 Fortune, Miss. Mabel Helen female 23.0 3   
92 Chaffee, Mr. Herbert Fuller male 46.0 1   
96 Goldschmidt, Mr. George B male 71.0 0   
97 Greenfield, Mr. William Bertram male 23.0 0   
102 White, Mr. Richard Frasar male 21.0 0   
110 Porter, Mr. Walter Chamberlain male 47.0 0   
118 Baxter, Mr. Quigg Edmond male 24.0 0   
123 Webber, Miss. Susan female 32.5 0   
124 White, Mr. Percival Wayland male 54.0 0   
136 Newsom, Miss. Helen Monypeny female 19.0 0   
137 Futrelle, Mr. Jacques Heath male 37.0 1   
139 Giglio, Mr. Victor male 24.0 0   
148 Navratil, Mr. Michel ("Louis M Hoffman") male 36.5 0   
151 Pears, Mrs. Thomas (Edith Wearne) female 22.0 1   
  
 Parch Ticket Fare Cabin Embarked   
1 0 PC 17599 71.2833 C85 C   
3 0 113803 53.1000 C123 S   
6 0 17463 51.8625 E46 S   
10 1 PP 9549 16.7000 G6 S   
11 0 113783 26.5500 C103 S   
21 0 248698 13.0000 D56 S   
23 0 113788 35.5000 A6 S   
27 2 19950 263.0000 C23 C25 C27 S   
52 0 PC 17572 76.7292 D33 C   
54 1 113509 61.9792 B30 C   
62 0 36973 83.4750 C83 S   
66 0 C.A. 29395 10.5000 F33 S   
75 0 348123 7.6500 F G73 S   
88 2 19950 263.0000 C23 C25 C27 S   
92 0 W.E.P. 5734 61.1750 E31 S   
96 0 PC 17754 34.6542 A5 C   
97 1 PC 17759 63.3583 D10 D12 C   
102 1 35281 77.2875 D26 S   
110 0 110465 52.0000 C110 S   
118 1 PC 17558 247.5208 B58 B60 C   
123 0 27267 13.0000 E101 S   
124 1 35281 77.2875 D26 S   
136 2 11752 26.2833 D47 S   
137 0 113803 53.1000 C123 S   
139 0 PC 17593 79.2000 B86 C   
148 2 230080 26.0000 F2 S   
151 0 113776 66.6000 C2 S

Select relevant columns from Titanic dataset (mix of nominal and ratio-scaled)

titanic\_df = titanic\_df[["Age", "Sex", "Pclass", "Fare", "Embarked"]]  
titanic\_df

Age Sex Pclass Fare Embarked  
1 38.0 female 1 71.2833 C  
3 35.0 female 1 53.1000 S  
6 54.0 male 1 51.8625 S  
10 4.0 female 3 16.7000 S  
11 58.0 female 1 26.5500 S  
21 34.0 male 2 13.0000 S  
23 28.0 male 1 35.5000 S  
27 19.0 male 1 263.0000 S  
52 49.0 female 1 76.7292 C  
54 65.0 male 1 61.9792 C  
62 45.0 male 1 83.4750 S  
66 29.0 female 2 10.5000 S  
75 25.0 male 3 7.6500 S  
88 23.0 female 1 263.0000 S  
92 46.0 male 1 61.1750 S  
96 71.0 male 1 34.6542 C  
97 23.0 male 1 63.3583 C  
102 21.0 male 1 77.2875 S  
110 47.0 male 1 52.0000 S  
118 24.0 male 1 247.5208 C  
123 32.5 female 2 13.0000 S  
124 54.0 male 1 77.2875 S  
136 19.0 female 1 26.2833 S  
137 37.0 male 1 53.1000 S  
139 24.0 male 1 79.2000 C  
148 36.5 male 2 26.0000 S  
151 22.0 female 1 66.6000 S

Encode Nominal as Integers for processing

label\_encoders\_titanic = {}  
**for** column **in** titanic\_df.columns:  
 **if** titanic\_df[column].dtype == object:  
 le = LabelEncoder()  
 titanic\_df[column] = le.fit\_transform(titanic\_df[column])  
 label\_encoders[column] = le  
  
titanic\_df

C:\Users\debat\AppData\Local\Temp\ipykernel\_8272\3305425594.py:5: SettingWithCopyWarning:   
A value is trying to be set on a copy of a slice from a DataFrame.  
Try using .loc[row\_indexer,col\_indexer] = value instead  
  
See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user\_guide/indexing.html#returning-a-view-versus-a-copy  
 titanic\_df[column] = le.fit\_transform(titanic\_df[column])  
C:\Users\debat\AppData\Local\Temp\ipykernel\_8272\3305425594.py:5: SettingWithCopyWarning:   
A value is trying to be set on a copy of a slice from a DataFrame.  
Try using .loc[row\_indexer,col\_indexer] = value instead  
  
See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user\_guide/indexing.html#returning-a-view-versus-a-copy  
 titanic\_df[column] = le.fit\_transform(titanic\_df[column])

Age Sex Pclass Fare Embarked  
1 38.0 0 1 71.2833 0  
3 35.0 0 1 53.1000 1  
6 54.0 1 1 51.8625 1  
10 4.0 0 3 16.7000 1  
11 58.0 0 1 26.5500 1  
21 34.0 1 2 13.0000 1  
23 28.0 1 1 35.5000 1  
27 19.0 1 1 263.0000 1  
52 49.0 0 1 76.7292 0  
54 65.0 1 1 61.9792 0  
62 45.0 1 1 83.4750 1  
66 29.0 0 2 10.5000 1  
75 25.0 1 3 7.6500 1  
88 23.0 0 1 263.0000 1  
92 46.0 1 1 61.1750 1  
96 71.0 1 1 34.6542 0  
97 23.0 1 1 63.3583 0  
102 21.0 1 1 77.2875 1  
110 47.0 1 1 52.0000 1  
118 24.0 1 1 247.5208 0  
123 32.5 0 2 13.0000 1  
124 54.0 1 1 77.2875 1  
136 19.0 0 1 26.2833 1  
137 37.0 1 1 53.1000 1  
139 24.0 1 1 79.2000 0  
148 36.5 1 2 26.0000 1  
151 22.0 0 1 66.6000 1

Combine the datasets into a list for further processing

*# Combine the datasets into a list for further processing*  
datasets = {  
 "Adult Dataset": adult\_df,  
 "Titanic Dataset": titanic\_df  
}

Compute Manhattan Distance

**def** manhattan\_distance(a, b):  
 *"""Calculate the Manhattan Distance between two vectors."""*  
 **try**:  
 **return** cityblock(a, b)  
 **except** Exception **as** e:  
 **return** np.nan  
  
*# Function to create the Manhattan Distance matrix*  
**def** calculate\_manhattan\_matrix(dataset):  
 n = len(dataset)  
 manhattan\_matrix = np.zeros((n, n))  
   
 **for** i **in** range(n):  
 **for** j **in** range(n):  
 manhattan\_matrix[i, j] = manhattan\_distance(dataset.iloc[i].values, dataset.iloc[j].values)  
   
 **return** pd.DataFrame(manhattan\_matrix)

Calculate Manhattan Distance

*For Adult Dataset*

manhattan\_matrix\_adult = calculate\_manhattan\_matrix(adult\_df)  
manhattan\_matrix\_adult

0 1 2 3 4 5 6 7 8 9 ... 90 \  
0 0.0 12.0 10.0 29.0 15.0 10.0 25.0 20.0 16.0 6.0 ... 24.0   
1 12.0 0.0 20.0 17.0 25.0 20.0 15.0 8.0 26.0 10.0 ... 12.0   
2 10.0 20.0 0.0 25.0 17.0 8.0 21.0 16.0 14.0 10.0 ... 24.0   
3 29.0 17.0 25.0 0.0 38.0 33.0 10.0 13.0 39.0 23.0 ... 13.0   
4 15.0 25.0 17.0 38.0 0.0 13.0 32.0 33.0 7.0 15.0 ... 33.0   
.. ... ... ... ... ... ... ... ... ... ... ... ...   
95 22.0 32.0 14.0 39.0 11.0 16.0 35.0 30.0 10.0 22.0 ... 36.0   
96 14.0 6.0 20.0 23.0 27.0 18.0 19.0 12.0 24.0 12.0 ... 18.0   
97 13.0 23.0 5.0 30.0 18.0 7.0 26.0 21.0 13.0 13.0 ... 27.0   
98 16.0 8.0 18.0 15.0 23.0 18.0 9.0 14.0 24.0 10.0 ... 12.0   
99 18.0 28.0 8.0 33.0 13.0 14.0 29.0 24.0 10.0 18.0 ... 32.0   
  
 91 92 93 94 95 96 97 98 99   
0 14.0 23.0 19.0 9.0 22.0 14.0 13.0 16.0 18.0   
1 24.0 33.0 29.0 19.0 32.0 6.0 23.0 8.0 28.0   
2 6.0 15.0 9.0 11.0 14.0 20.0 5.0 18.0 8.0   
3 31.0 40.0 34.0 32.0 39.0 23.0 30.0 15.0 33.0   
4 17.0 8.0 8.0 8.0 11.0 27.0 18.0 23.0 13.0   
.. ... ... ... ... ... ... ... ... ...   
95 10.0 3.0 7.0 13.0 0.0 32.0 9.0 30.0 8.0   
96 24.0 33.0 29.0 21.0 32.0 0.0 23.0 10.0 28.0   
97 1.0 10.0 12.0 12.0 9.0 23.0 0.0 21.0 11.0   
98 20.0 29.0 25.0 19.0 30.0 10.0 21.0 0.0 26.0   
99 12.0 11.0 5.0 9.0 8.0 28.0 11.0 26.0 0.0   
  
[100 rows x 100 columns]

*For Titanic Dataset*

manhattan\_matrix\_titanic = calculate\_manhattan\_matrix(titanic\_df)  
manhattan\_matrix\_titanic

0 1 2 3 4 5 6 \  
0 0.0000 22.1833 37.4208 91.5833 65.7333 65.2833 47.7833   
1 22.1833 0.0000 21.2375 69.4000 49.5500 43.1000 25.6000   
2 37.4208 21.2375 0.0000 88.1625 30.3125 59.8625 42.3625   
3 91.5833 69.4000 88.1625 0.0000 65.8500 35.7000 45.8000   
4 65.7333 49.5500 30.3125 65.8500 0.0000 39.5500 39.9500   
5 65.2833 43.1000 59.8625 35.7000 39.5500 0.0000 29.5000   
6 47.7833 25.6000 42.3625 45.8000 39.9500 29.5000 0.0000   
7 212.7167 226.9000 246.1375 264.3000 276.4500 266.0000 236.5000   
8 16.4459 38.6292 31.8667 108.0292 60.1792 81.7292 64.2292   
9 37.3041 40.8792 22.1167 110.2792 44.4292 81.9792 64.4792   
10 21.1917 41.3750 40.6125 110.7750 70.9250 82.4750 64.9750   
11 71.7833 49.6000 68.3625 32.2000 46.0500 8.5000 28.0000   
12 80.6333 58.4500 75.2125 31.0500 54.9000 15.3500 32.8500   
13 207.7167 221.9000 243.1375 267.3000 271.4500 263.0000 233.5000   
14 20.1083 20.0750 17.3125 89.4750 47.6250 61.1750 43.6750   
15 70.6291 56.4458 35.2083 88.9542 23.1042 60.6542 44.8458   
16 23.9250 24.2583 43.4958 69.6583 73.8083 63.3583 33.8583   
17 25.0042 39.1875 58.4250 80.5875 88.7375 78.2875 48.7875   
18 30.2833 14.1000 7.1375 81.3000 37.4500 53.0000 35.5000   
19 191.2375 207.4208 226.6583 254.8208 256.9708 246.5208 217.0208   
20 65.7833 43.6000 62.3625 33.2000 40.0500 2.5000 29.0000   
21 24.0042 44.1875 25.4250 113.5875 55.7375 85.2875 67.7875   
22 65.0000 42.8167 61.5792 26.5833 39.2667 30.2833 19.2167   
23 21.1833 3.0000 18.2375 72.4000 48.5500 44.1000 26.6000   
24 22.9167 39.1000 58.3375 86.5000 88.6500 78.2000 48.7000   
25 49.7833 30.6000 44.3625 43.8000 24.0500 15.5000 19.0000   
26 21.6833 26.5000 47.7375 69.9000 76.0500 67.6000 38.1000   
  
 7 8 9 ... 17 18 19 20 \  
0 212.7167 16.4459 37.3041 ... 25.0042 30.2833 191.2375 65.7833   
1 226.9000 38.6292 40.8792 ... 39.1875 14.1000 207.4208 43.6000   
2 246.1375 31.8667 22.1167 ... 58.4250 7.1375 226.6583 62.3625   
3 264.3000 108.0292 110.2792 ... 80.5875 81.3000 254.8208 33.2000   
4 276.4500 60.1792 44.4292 ... 88.7375 37.4500 256.9708 40.0500   
5 266.0000 81.7292 81.9792 ... 78.2875 53.0000 246.5208 2.5000   
6 236.5000 64.2292 64.4792 ... 48.7875 35.5000 217.0208 29.0000   
7 0.0000 218.2708 248.0208 ... 187.7125 239.0000 21.4792 265.5000   
8 218.2708 0.0000 31.7500 ... 30.5583 28.7292 196.7916 82.2292   
9 248.0208 31.7500 0.0000 ... 60.3083 28.9792 226.5416 84.4792   
10 205.5250 12.7458 42.4958 ... 30.1875 33.4750 186.0458 84.9750   
11 264.5000 88.2292 90.4792 ... 76.7875 61.5000 245.0208 6.0000   
12 263.3500 97.0792 97.3292 ... 75.6375 68.3500 243.8708 14.8500   
13 5.0000 213.2708 245.0208 ... 188.7125 236.0000 18.4792 260.5000   
14 228.8250 20.5542 20.8042 ... 41.1125 10.1750 209.3458 63.6750   
15 281.3458 65.0750 33.3250 ... 93.6333 42.3458 259.8666 63.1542   
16 204.6417 40.3709 43.3791 ... 16.9292 36.3583 185.1625 62.8583   
17 187.7125 30.5583 60.3083 ... 0.0000 51.2875 174.2333 77.7875   
18 239.0000 28.7292 28.9792 ... 51.2875 0.0000 219.5208 55.5000   
19 21.4792 196.7916 226.5416 ... 174.2333 219.5208 0.0000 246.0208   
20 265.5000 82.2292 84.4792 ... 77.7875 55.5000 246.0208 0.0000   
21 220.7125 7.5583 27.3083 ... 33.0000 32.2875 201.2333 87.7875   
22 237.7167 81.4459 83.6959 ... 54.0042 54.7167 228.2375 27.7833   
23 227.9000 37.6292 37.8792 ... 40.1875 11.1000 208.4208 46.6000   
24 189.8000 28.4708 58.2208 ... 5.9125 51.2000 168.3208 77.7000   
25 255.5000 66.2292 66.4792 ... 67.7875 37.5000 236.0208 18.0000   
26 200.4000 38.1292 49.6208 ... 12.6875 40.6000 184.9208 65.1000   
  
 21 22 23 24 25 26   
0 24.0042 65.0000 21.1833 22.9167 49.7833 21.6833   
1 44.1875 42.8167 3.0000 39.1000 30.6000 26.5000   
2 25.4250 61.5792 18.2375 58.3375 44.3625 47.7375   
3 113.5875 26.5833 72.4000 86.5000 43.8000 69.9000   
4 55.7375 39.2667 48.5500 88.6500 24.0500 76.0500   
5 85.2875 30.2833 44.1000 78.2000 15.5000 67.6000   
6 67.7875 19.2167 26.6000 48.7000 19.0000 38.1000   
7 220.7125 237.7167 227.9000 189.8000 255.5000 200.4000   
8 7.5583 81.4459 37.6292 28.4708 66.2292 38.1292   
9 27.3083 83.6959 37.8792 58.2208 66.4792 49.6208   
10 15.1875 84.1917 38.3750 26.2750 66.9750 40.8750   
11 93.7875 26.7833 52.6000 76.7000 24.0000 64.1000   
12 100.6375 27.6333 59.4500 75.5500 30.8500 64.9500   
13 217.7125 240.7167 224.9000 186.8000 252.5000 197.4000   
14 24.1125 62.8917 17.0750 41.0250 45.6750 30.4250   
15 60.6333 62.3709 53.4458 91.5458 45.1542 82.9458   
16 45.9292 43.0750 25.2583 16.8417 52.8583 6.2417   
17 33.0000 54.0042 40.1875 5.9125 67.7875 12.6875   
18 32.2875 54.7167 11.1000 51.2000 37.5000 40.6000   
19 201.2333 228.2375 208.4208 168.3208 236.0208 184.9208   
20 87.7875 27.7833 46.6000 77.7000 18.0000 65.1000   
21 0.0000 87.0042 41.1875 32.9125 69.7875 43.6875   
22 87.0042 0.0000 45.8167 59.9167 19.7833 43.3167   
23 41.1875 45.8167 0.0000 40.1000 28.6000 29.5000   
24 32.9125 59.9167 40.1000 0.0000 67.7000 16.6000   
25 69.7875 19.7833 28.6000 67.7000 0.0000 57.1000   
26 43.6875 43.3167 29.5000 16.6000 57.1000 0.0000   
  
[27 rows x 27 columns]

Explanation

Manhattan Distance Calculation: This metric is the sum of the absolute differences between corresponding coordinates of two points. It’s especially useful in grid-like environments where movement is restricted to horizontal and vertical paths.

Handling Different Data Types: Like Euclidean Distance, Manhattan Distance is more meaningful for interval and ratio-scaled data, but it can also be applied to ordinal data with caution. It’s less sensitive to outliers compared to Euclidean Distance.

Observation and Analysis

The resulting matrices will represent the pairwise Manhattan distances between data points. A smaller value indicates that the data points are closer to each other in terms of their grid-like path, while a larger value indicates they are further apart.

Manhattan Distance is particularly useful in scenarios where the difference in individual dimensions is more meaningful than their squared differences (as in Euclidean Distance).

Supremum Distance

Supremum Distance

also known as the Chebyshev Distance. This metric measures the maximum absolute difference between the coordinates of two points. It is particularly useful in scenarios where the most significant difference between any single dimension matters the most.

Importing required Libraries

**import** pandas **as** pd  
**import** numpy **as** np  
**from** sklearn.preprocessing **import** LabelEncoder  
**from** scipy.spatial.distance **import** chebyshev

Load Datasets

*# Load datasets*  
adult\_df = pd.read\_csv("../adult/adult\_trim.data", header=None) *# No header*  
titanic\_df = pd.read\_csv('../titanic/titanic\_trim.csv') *# Has header*  
  
*# Rename columns for clarity*  
adult\_df.columns = ["age", "workclass", "fnlwgt", "education", "education\_num",   
 "marital\_status", "occupation", "relationship", "race", "sex",   
 "capital\_gain", "capital\_loss", "hours\_per\_week", "native\_country", "income"]  
adult\_df.dropna(inplace=True)

adult\_df

age workclass fnlwgt education education\_num \  
0 39 State-gov 77516 Bachelors 13   
1 50 Self-emp-not-inc 83311 Bachelors 13   
2 38 Private 215646 HS-grad 9   
3 53 Private 234721 11th 7   
4 28 Private 338409 Bachelors 13   
.. ... ... ... ... ...   
95 29 Local-gov 115585 Some-college 10   
96 48 Self-emp-not-inc 191277 Doctorate 16   
97 37 Private 202683 Some-college 10   
98 48 Private 171095 Assoc-acdm 12   
99 32 Federal-gov 249409 HS-grad 9   
  
 marital\_status occupation relationship race sex \  
0 Never-married Adm-clerical Not-in-family White Male   
1 Married-civ-spouse Exec-managerial Husband White Male   
2 Divorced Handlers-cleaners Not-in-family White Male   
3 Married-civ-spouse Handlers-cleaners Husband Black Male   
4 Married-civ-spouse Prof-specialty Wife Black Female   
.. ... ... ... ... ...   
95 Never-married Handlers-cleaners Not-in-family White Male   
96 Married-civ-spouse Prof-specialty Husband White Male   
97 Married-civ-spouse Sales Husband White Male   
98 Divorced Exec-managerial Unmarried White Female   
99 Never-married Other-service Own-child Black Male   
  
 capital\_gain capital\_loss hours\_per\_week native\_country income   
0 2174 0 40 United-States <=50K   
1 0 0 13 United-States <=50K   
2 0 0 40 United-States <=50K   
3 0 0 40 United-States <=50K   
4 0 0 40 Cuba <=50K   
.. ... ... ... ... ...   
95 0 0 50 United-States <=50K   
96 0 1902 60 United-States >50K   
97 0 0 48 United-States >50K   
98 0 0 40 England <=50K   
99 0 0 40 United-States <=50K   
  
[100 rows x 15 columns]

titanic\_df

PassengerId Survived Pclass \  
0 1 0 3   
1 2 1 1   
2 3 1 3   
3 4 1 1   
4 5 0 3   
.. ... ... ...   
150 151 0 2   
151 152 1 1   
152 153 0 3   
153 154 0 3   
154 155 0 3   
  
 Name Sex Age SibSp \  
0 Braund, Mr. Owen Harris male 22.0 1   
1 Cumings, Mrs. John Bradley (Florence Briggs Th... female 38.0 1   
2 Heikkinen, Miss. Laina female 26.0 0   
3 Futrelle, Mrs. Jacques Heath (Lily May Peel) female 35.0 1   
4 Allen, Mr. William Henry male 35.0 0   
.. ... ... ... ...   
150 Bateman, Rev. Robert James male 51.0 0   
151 Pears, Mrs. Thomas (Edith Wearne) female 22.0 1   
152 Meo, Mr. Alfonzo male 55.5 0   
153 van Billiard, Mr. Austin Blyler male 40.5 0   
154 Olsen, Mr. Ole Martin male NaN 0   
  
 Parch Ticket Fare Cabin Embarked   
0 0 A/5 21171 7.2500 NaN S   
1 0 PC 17599 71.2833 C85 C   
2 0 STON/O2. 3101282 7.9250 NaN S   
3 0 113803 53.1000 C123 S   
4 0 373450 8.0500 NaN S   
.. ... ... ... ... ...   
150 0 S.O.P. 1166 12.5250 NaN S   
151 0 113776 66.6000 C2 S   
152 0 A.5. 11206 8.0500 NaN S   
153 2 A/5. 851 14.5000 NaN S   
154 0 Fa 265302 7.3125 NaN S   
  
[155 rows x 12 columns]

Select relevant columns from Adult dataset (mix of nominal and ratio-scaled)

adult\_df = adult\_df[["age", "workclass", "education", "education\_num", "sex"]]  
  
adult\_df

age workclass education education\_num sex  
0 39 State-gov Bachelors 13 Male  
1 50 Self-emp-not-inc Bachelors 13 Male  
2 38 Private HS-grad 9 Male  
3 53 Private 11th 7 Male  
4 28 Private Bachelors 13 Female  
.. ... ... ... ... ...  
95 29 Local-gov Some-college 10 Male  
96 48 Self-emp-not-inc Doctorate 16 Male  
97 37 Private Some-college 10 Male  
98 48 Private Assoc-acdm 12 Female  
99 32 Federal-gov HS-grad 9 Male  
  
[100 rows x 5 columns]

Encode nominal attributes as integers for processing

label\_encoders = {}  
**for** column **in** adult\_df.columns:  
 **if** adult\_df[column].dtype == object:  
 le = LabelEncoder()  
 adult\_df[column] = le.fit\_transform(adult\_df[column])  
 label\_encoders[column] = le  
  
adult\_df

C:\Users\debat\AppData\Local\Temp\ipykernel\_5048\183426126.py:5: SettingWithCopyWarning:   
A value is trying to be set on a copy of a slice from a DataFrame.  
Try using .loc[row\_indexer,col\_indexer] = value instead  
  
See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user\_guide/indexing.html#returning-a-view-versus-a-copy  
 adult\_df[column] = le.fit\_transform(adult\_df[column])  
C:\Users\debat\AppData\Local\Temp\ipykernel\_5048\183426126.py:5: SettingWithCopyWarning:   
A value is trying to be set on a copy of a slice from a DataFrame.  
Try using .loc[row\_indexer,col\_indexer] = value instead  
  
See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user\_guide/indexing.html#returning-a-view-versus-a-copy  
 adult\_df[column] = le.fit\_transform(adult\_df[column])  
C:\Users\debat\AppData\Local\Temp\ipykernel\_5048\183426126.py:5: SettingWithCopyWarning:   
A value is trying to be set on a copy of a slice from a DataFrame.  
Try using .loc[row\_indexer,col\_indexer] = value instead  
  
See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user\_guide/indexing.html#returning-a-view-versus-a-copy  
 adult\_df[column] = le.fit\_transform(adult\_df[column])

age workclass education education\_num sex  
0 39 6 7 13 1  
1 50 5 7 13 1  
2 38 3 9 9 1  
3 53 3 1 7 1  
4 28 3 7 13 0  
.. ... ... ... ... ...  
95 29 2 12 10 1  
96 48 5 8 16 1  
97 37 3 12 10 1  
98 48 3 5 12 0  
99 32 1 9 9 1  
  
[100 rows x 5 columns]

Clean and preprocess Titanic dataset

titanic\_df.dropna(inplace=True)  
titanic\_df

PassengerId Survived Pclass \  
1 2 1 1   
3 4 1 1   
6 7 0 1   
10 11 1 3   
11 12 1 1   
21 22 1 2   
23 24 1 1   
27 28 0 1   
52 53 1 1   
54 55 0 1   
62 63 0 1   
66 67 1 2   
75 76 0 3   
88 89 1 1   
92 93 0 1   
96 97 0 1   
97 98 1 1   
102 103 0 1   
110 111 0 1   
118 119 0 1   
123 124 1 2   
124 125 0 1   
136 137 1 1   
137 138 0 1   
139 140 0 1   
148 149 0 2   
151 152 1 1   
  
 Name Sex Age SibSp \  
1 Cumings, Mrs. John Bradley (Florence Briggs Th... female 38.0 1   
3 Futrelle, Mrs. Jacques Heath (Lily May Peel) female 35.0 1   
6 McCarthy, Mr. Timothy J male 54.0 0   
10 Sandstrom, Miss. Marguerite Rut female 4.0 1   
11 Bonnell, Miss. Elizabeth female 58.0 0   
21 Beesley, Mr. Lawrence male 34.0 0   
23 Sloper, Mr. William Thompson male 28.0 0   
27 Fortune, Mr. Charles Alexander male 19.0 3   
52 Harper, Mrs. Henry Sleeper (Myna Haxtun) female 49.0 1   
54 Ostby, Mr. Engelhart Cornelius male 65.0 0   
62 Harris, Mr. Henry Birkhardt male 45.0 1   
66 Nye, Mrs. (Elizabeth Ramell) female 29.0 0   
75 Moen, Mr. Sigurd Hansen male 25.0 0   
88 Fortune, Miss. Mabel Helen female 23.0 3   
92 Chaffee, Mr. Herbert Fuller male 46.0 1   
96 Goldschmidt, Mr. George B male 71.0 0   
97 Greenfield, Mr. William Bertram male 23.0 0   
102 White, Mr. Richard Frasar male 21.0 0   
110 Porter, Mr. Walter Chamberlain male 47.0 0   
118 Baxter, Mr. Quigg Edmond male 24.0 0   
123 Webber, Miss. Susan female 32.5 0   
124 White, Mr. Percival Wayland male 54.0 0   
136 Newsom, Miss. Helen Monypeny female 19.0 0   
137 Futrelle, Mr. Jacques Heath male 37.0 1   
139 Giglio, Mr. Victor male 24.0 0   
148 Navratil, Mr. Michel ("Louis M Hoffman") male 36.5 0   
151 Pears, Mrs. Thomas (Edith Wearne) female 22.0 1   
  
 Parch Ticket Fare Cabin Embarked   
1 0 PC 17599 71.2833 C85 C   
3 0 113803 53.1000 C123 S   
6 0 17463 51.8625 E46 S   
10 1 PP 9549 16.7000 G6 S   
11 0 113783 26.5500 C103 S   
21 0 248698 13.0000 D56 S   
23 0 113788 35.5000 A6 S   
27 2 19950 263.0000 C23 C25 C27 S   
52 0 PC 17572 76.7292 D33 C   
54 1 113509 61.9792 B30 C   
62 0 36973 83.4750 C83 S   
66 0 C.A. 29395 10.5000 F33 S   
75 0 348123 7.6500 F G73 S   
88 2 19950 263.0000 C23 C25 C27 S   
92 0 W.E.P. 5734 61.1750 E31 S   
96 0 PC 17754 34.6542 A5 C   
97 1 PC 17759 63.3583 D10 D12 C   
102 1 35281 77.2875 D26 S   
110 0 110465 52.0000 C110 S   
118 1 PC 17558 247.5208 B58 B60 C   
123 0 27267 13.0000 E101 S   
124 1 35281 77.2875 D26 S   
136 2 11752 26.2833 D47 S   
137 0 113803 53.1000 C123 S   
139 0 PC 17593 79.2000 B86 C   
148 2 230080 26.0000 F2 S   
151 0 113776 66.6000 C2 S

Select relevant columns from Titanic dataset (mix of nominal and ratio-scaled)

titanic\_df = titanic\_df[["Age", "Sex", "Pclass", "Fare", "Embarked"]]  
titanic\_df

Age Sex Pclass Fare Embarked  
1 38.0 female 1 71.2833 C  
3 35.0 female 1 53.1000 S  
6 54.0 male 1 51.8625 S  
10 4.0 female 3 16.7000 S  
11 58.0 female 1 26.5500 S  
21 34.0 male 2 13.0000 S  
23 28.0 male 1 35.5000 S  
27 19.0 male 1 263.0000 S  
52 49.0 female 1 76.7292 C  
54 65.0 male 1 61.9792 C  
62 45.0 male 1 83.4750 S  
66 29.0 female 2 10.5000 S  
75 25.0 male 3 7.6500 S  
88 23.0 female 1 263.0000 S  
92 46.0 male 1 61.1750 S  
96 71.0 male 1 34.6542 C  
97 23.0 male 1 63.3583 C  
102 21.0 male 1 77.2875 S  
110 47.0 male 1 52.0000 S  
118 24.0 male 1 247.5208 C  
123 32.5 female 2 13.0000 S  
124 54.0 male 1 77.2875 S  
136 19.0 female 1 26.2833 S  
137 37.0 male 1 53.1000 S  
139 24.0 male 1 79.2000 C  
148 36.5 male 2 26.0000 S  
151 22.0 female 1 66.6000 S

Encode Nominal as Integers for processing

label\_encoders\_titanic = {}  
**for** column **in** titanic\_df.columns:  
 **if** titanic\_df[column].dtype == object:  
 le = LabelEncoder()  
 titanic\_df[column] = le.fit\_transform(titanic\_df[column])  
 label\_encoders[column] = le  
  
titanic\_df

C:\Users\debat\AppData\Local\Temp\ipykernel\_5048\3305425594.py:5: SettingWithCopyWarning:   
A value is trying to be set on a copy of a slice from a DataFrame.  
Try using .loc[row\_indexer,col\_indexer] = value instead  
  
See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user\_guide/indexing.html#returning-a-view-versus-a-copy  
 titanic\_df[column] = le.fit\_transform(titanic\_df[column])  
C:\Users\debat\AppData\Local\Temp\ipykernel\_5048\3305425594.py:5: SettingWithCopyWarning:   
A value is trying to be set on a copy of a slice from a DataFrame.  
Try using .loc[row\_indexer,col\_indexer] = value instead  
  
See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user\_guide/indexing.html#returning-a-view-versus-a-copy  
 titanic\_df[column] = le.fit\_transform(titanic\_df[column])

Age Sex Pclass Fare Embarked  
1 38.0 0 1 71.2833 0  
3 35.0 0 1 53.1000 1  
6 54.0 1 1 51.8625 1  
10 4.0 0 3 16.7000 1  
11 58.0 0 1 26.5500 1  
21 34.0 1 2 13.0000 1  
23 28.0 1 1 35.5000 1  
27 19.0 1 1 263.0000 1  
52 49.0 0 1 76.7292 0  
54 65.0 1 1 61.9792 0  
62 45.0 1 1 83.4750 1  
66 29.0 0 2 10.5000 1  
75 25.0 1 3 7.6500 1  
88 23.0 0 1 263.0000 1  
92 46.0 1 1 61.1750 1  
96 71.0 1 1 34.6542 0  
97 23.0 1 1 63.3583 0  
102 21.0 1 1 77.2875 1  
110 47.0 1 1 52.0000 1  
118 24.0 1 1 247.5208 0  
123 32.5 0 2 13.0000 1  
124 54.0 1 1 77.2875 1  
136 19.0 0 1 26.2833 1  
137 37.0 1 1 53.1000 1  
139 24.0 1 1 79.2000 0  
148 36.5 1 2 26.0000 1  
151 22.0 0 1 66.6000 1

Combine the datasets into a list for further processing

*# Combine the datasets into a list for further processing*  
datasets = {  
 "Adult Dataset": adult\_df,  
 "Titanic Dataset": titanic\_df  
}

Compute Supremum Distance

**def** supremum\_distance(a, b):  
 *"""Calculate the Supremum (Chebyshev) Distance between two vectors."""*  
 **try**:  
 **return** chebyshev(a, b)  
 **except** Exception **as** e:  
 **return** np.nan  
  
*# Function to create the Supremum Distance matrix*  
**def** calculate\_supremum\_matrix(dataset):  
 n = len(dataset)  
 supremum\_matrix = np.zeros((n, n))  
   
 **for** i **in** range(n):  
 **for** j **in** range(n):  
 supremum\_matrix[i, j] = supremum\_distance(dataset.iloc[i].values, dataset.iloc[j].values)  
   
 **return** pd.DataFrame(supremum\_matrix)

Calculate Supremum Distance

*For Adult Dataset*

supremum\_matrix\_adult = calculate\_supremum\_matrix(adult\_df)  
supremum\_matrix\_adult

0 1 2 3 4 5 6 7 8 9 ... 90 \  
0 0.0 11.0 4.0 14.0 11.0 3.0 10.0 13.0 8.0 3.0 ... 18.0   
1 11.0 0.0 12.0 6.0 22.0 13.0 8.0 4.0 19.0 8.0 ... 7.0   
2 4.0 12.0 0.0 15.0 10.0 5.0 11.0 14.0 7.0 4.0 ... 19.0   
3 14.0 6.0 15.0 0.0 25.0 16.0 4.0 8.0 22.0 11.0 ... 5.0   
4 11.0 22.0 10.0 25.0 0.0 9.0 21.0 24.0 3.0 14.0 ... 29.0   
.. ... ... ... ... ... ... ... ... ... ... ... ...   
95 10.0 21.0 9.0 24.0 5.0 8.0 20.0 23.0 4.0 13.0 ... 28.0   
96 9.0 3.0 10.0 9.0 20.0 11.0 11.0 7.0 17.0 6.0 ... 9.0   
97 5.0 13.0 3.0 16.0 9.0 4.0 12.0 15.0 6.0 5.0 ... 20.0   
98 9.0 2.0 10.0 5.0 20.0 11.0 7.0 4.0 17.0 6.0 ... 9.0   
99 7.0 18.0 6.0 21.0 4.0 5.0 17.0 20.0 5.0 10.0 ... 25.0   
  
 91 92 93 94 95 96 97 98 99   
0 5.0 11.0 9.0 5.0 10.0 9.0 5.0 9.0 7.0   
1 13.0 22.0 20.0 16.0 21.0 3.0 13.0 2.0 18.0   
2 3.0 10.0 8.0 4.0 9.0 10.0 3.0 10.0 6.0   
3 16.0 25.0 23.0 19.0 24.0 9.0 16.0 5.0 21.0   
4 9.0 5.0 4.0 6.0 5.0 20.0 9.0 20.0 4.0   
.. ... ... ... ... ... ... ... ... ...   
95 8.0 1.0 3.0 5.0 0.0 19.0 8.0 19.0 3.0   
96 11.0 20.0 18.0 14.0 19.0 0.0 11.0 4.0 16.0   
97 1.0 9.0 7.0 5.0 8.0 11.0 0.0 11.0 5.0   
98 11.0 20.0 18.0 14.0 19.0 4.0 11.0 0.0 16.0   
99 5.0 4.0 2.0 4.0 3.0 16.0 5.0 16.0 0.0   
  
[100 rows x 100 columns]

*For Titanic Dataset*

supremum\_matrix\_titanic = calculate\_supremum\_matrix(titanic\_df)  
supremum\_matrix\_titanic

0 1 2 3 4 5 6 \  
0 0.0000 18.1833 19.4208 54.5833 44.7333 58.2833 35.7833   
1 18.1833 0.0000 19.0000 36.4000 26.5500 40.1000 17.6000   
2 19.4208 19.0000 0.0000 50.0000 25.3125 38.8625 26.0000   
3 54.5833 36.4000 50.0000 0.0000 54.0000 30.0000 24.0000   
4 44.7333 26.5500 25.3125 54.0000 0.0000 24.0000 30.0000   
5 58.2833 40.1000 38.8625 30.0000 24.0000 0.0000 22.5000   
6 35.7833 17.6000 26.0000 24.0000 30.0000 22.5000 0.0000   
7 191.7167 209.9000 211.1375 246.3000 236.4500 250.0000 227.5000   
8 11.0000 23.6292 24.8667 60.0292 50.1792 63.7292 41.2292   
9 27.0000 30.0000 11.0000 61.0000 35.4292 48.9792 37.0000   
10 12.1917 30.3750 31.6125 66.7750 56.9250 70.4750 47.9750   
11 60.7833 42.6000 41.3625 25.0000 29.0000 5.0000 25.0000   
12 63.6333 45.4500 44.2125 21.0000 33.0000 9.0000 27.8500   
13 191.7167 209.9000 211.1375 246.3000 236.4500 250.0000 227.5000   
14 10.1083 11.0000 9.3125 44.4750 34.6250 48.1750 25.6750   
15 36.6291 36.0000 17.2083 67.0000 13.0000 37.0000 43.0000   
16 15.0000 12.0000 31.0000 46.6583 36.8083 50.3583 27.8583   
17 17.0000 24.1875 33.0000 60.5875 50.7375 64.2875 41.7875   
18 19.2833 12.0000 7.0000 43.0000 25.4500 39.0000 19.0000   
19 176.2375 194.4208 195.6583 230.8208 220.9708 234.5208 212.0208   
20 58.2833 40.1000 38.8625 28.5000 25.5000 1.5000 22.5000   
21 16.0000 24.1875 25.4250 60.5875 50.7375 64.2875 41.7875   
22 45.0000 26.8167 35.0000 15.0000 39.0000 15.0000 9.2167   
23 18.1833 2.0000 17.0000 36.4000 26.5500 40.1000 17.6000   
24 14.0000 26.1000 30.0000 62.5000 52.6500 66.2000 43.7000   
25 45.2833 27.1000 25.8625 32.5000 21.5000 13.0000 9.5000   
26 16.0000 13.5000 32.0000 49.9000 40.0500 53.6000 31.1000   
  
 7 8 9 ... 17 18 19 20 \  
0 191.7167 11.0000 27.0000 ... 17.0000 19.2833 176.2375 58.2833   
1 209.9000 23.6292 30.0000 ... 24.1875 12.0000 194.4208 40.1000   
2 211.1375 24.8667 11.0000 ... 33.0000 7.0000 195.6583 38.8625   
3 246.3000 60.0292 61.0000 ... 60.5875 43.0000 230.8208 28.5000   
4 236.4500 50.1792 35.4292 ... 50.7375 25.4500 220.9708 25.5000   
5 250.0000 63.7292 48.9792 ... 64.2875 39.0000 234.5208 1.5000   
6 227.5000 41.2292 37.0000 ... 41.7875 19.0000 212.0208 22.5000   
7 0.0000 186.2708 201.0208 ... 185.7125 211.0000 15.4792 250.0000   
8 186.2708 0.0000 16.0000 ... 28.0000 24.7292 170.7916 63.7292   
9 201.0208 16.0000 0.0000 ... 44.0000 18.0000 185.5416 48.9792   
10 179.5250 6.7458 21.4958 ... 24.0000 31.4750 164.0458 70.4750   
11 252.5000 66.2292 51.4792 ... 66.7875 41.5000 237.0208 3.5000   
12 255.3500 69.0792 54.3292 ... 69.6375 44.3500 239.8708 7.5000   
13 4.0000 186.2708 201.0208 ... 185.7125 211.0000 15.4792 250.0000   
14 201.8250 15.5542 19.0000 ... 25.0000 9.1750 186.3458 48.1750   
15 228.3458 42.0750 27.3250 ... 50.0000 24.0000 212.8666 38.5000   
16 199.6417 26.0000 42.0000 ... 13.9292 24.0000 184.1625 50.3583   
17 185.7125 28.0000 44.0000 ... 0.0000 26.0000 170.2333 64.2875   
18 211.0000 24.7292 18.0000 ... 26.0000 0.0000 195.5208 39.0000   
19 15.4792 170.7916 185.5416 ... 170.2333 195.5208 0.0000 234.5208   
20 250.0000 63.7292 48.9792 ... 64.2875 39.0000 234.5208 0.0000   
21 185.7125 5.0000 15.3083 ... 33.0000 25.2875 170.2333 64.2875   
22 236.7167 50.4459 46.0000 ... 51.0042 28.0000 221.2375 13.5000   
23 209.9000 23.6292 28.0000 ... 24.1875 10.0000 194.4208 40.1000   
24 183.8000 25.0000 41.0000 ... 3.0000 27.2000 168.3208 66.2000   
25 237.0000 50.7292 35.9792 ... 51.2875 26.0000 221.5208 13.0000   
26 196.4000 27.0000 43.0000 ... 10.6875 25.0000 180.9208 53.6000   
  
 21 22 23 24 25 26   
0 16.0000 45.0000 18.1833 14.0000 45.2833 16.0000   
1 24.1875 26.8167 2.0000 26.1000 27.1000 13.5000   
2 25.4250 35.0000 17.0000 30.0000 25.8625 32.0000   
3 60.5875 15.0000 36.4000 62.5000 32.5000 49.9000   
4 50.7375 39.0000 26.5500 52.6500 21.5000 40.0500   
5 64.2875 15.0000 40.1000 66.2000 13.0000 53.6000   
6 41.7875 9.2167 17.6000 43.7000 9.5000 31.1000   
7 185.7125 236.7167 209.9000 183.8000 237.0000 196.4000   
8 5.0000 50.4459 23.6292 25.0000 50.7292 27.0000   
9 15.3083 46.0000 28.0000 41.0000 35.9792 43.0000   
10 9.0000 57.1917 30.3750 21.0000 57.4750 23.0000   
11 66.7875 15.7833 42.6000 68.7000 15.5000 56.1000   
12 69.6375 18.6333 45.4500 71.5500 18.3500 58.9500   
13 185.7125 236.7167 209.9000 183.8000 237.0000 196.4000   
14 16.1125 34.8917 9.0000 22.0000 35.1750 24.0000   
15 42.6333 52.0000 34.0000 47.0000 34.5000 49.0000   
16 31.0000 37.0750 14.0000 15.8417 37.3583 3.2417   
17 33.0000 51.0042 24.1875 3.0000 51.2875 10.6875   
18 25.2875 28.0000 10.0000 27.2000 26.0000 25.0000   
19 170.2333 221.2375 194.4208 168.3208 221.5208 180.9208   
20 64.2875 13.5000 40.1000 66.2000 13.0000 53.6000   
21 0.0000 51.0042 24.1875 30.0000 51.2875 32.0000   
22 51.0042 0.0000 26.8167 52.9167 17.5000 40.3167   
23 24.1875 26.8167 0.0000 26.1000 27.1000 15.0000   
24 30.0000 52.9167 26.1000 0.0000 53.2000 12.6000   
25 51.2875 17.5000 27.1000 53.2000 0.0000 40.6000   
26 32.0000 40.3167 15.0000 12.6000 40.6000 0.0000   
  
[27 rows x 27 columns]

Explanation

Supremum Distance Calculation: This metric considers the largest difference between any single dimension of two data points. It can be thought of as the distance you would move in a chessboard when the king moves, as it captures the maximum deviation across all dimensions.

Handling Different Data Types: Supremum Distance works well with interval and ratio-scaled data, and can be applied to ordinal data as well. It's particularly useful when the most significant difference in any single attribute is of primary concern.

Observation and Analysis

The resulting matrices will show the pairwise Supremum distances between data points. A smaller value indicates that the largest difference between any dimension of the data points is small, while a larger value indicates a significant difference in at least one dimension.

Supremum Distance is useful in scenarios where outliers or extreme differences in one dimension are more significant than cumulative differences across all dimensions.

Correlation Matrix

Correlation Matrix

Correlation measures the linear relationship between two variables and is particularly useful when dealing with interval and ratio-scaled data. It can indicate how similar the variations in two data points are.

Importing required Libraries

**import** pandas **as** pd  
**import** numpy **as** np  
**from** sklearn.preprocessing **import** LabelEncoder

Load Datasets

*# Load datasets*  
adult\_df = pd.read\_csv("../adult/adult\_trim.data", header=None) *# No header*  
titanic\_df = pd.read\_csv('../titanic/titanic\_trim.csv') *# Has header*  
  
*# Rename columns for clarity*  
adult\_df.columns = ["age", "workclass", "fnlwgt", "education", "education\_num",   
 "marital\_status", "occupation", "relationship", "race", "sex",   
 "capital\_gain", "capital\_loss", "hours\_per\_week", "native\_country", "income"]  
adult\_df.dropna(inplace=True)

adult\_df

age workclass fnlwgt education education\_num \  
0 39 State-gov 77516 Bachelors 13   
1 50 Self-emp-not-inc 83311 Bachelors 13   
2 38 Private 215646 HS-grad 9   
3 53 Private 234721 11th 7   
4 28 Private 338409 Bachelors 13   
.. ... ... ... ... ...   
95 29 Local-gov 115585 Some-college 10   
96 48 Self-emp-not-inc 191277 Doctorate 16   
97 37 Private 202683 Some-college 10   
98 48 Private 171095 Assoc-acdm 12   
99 32 Federal-gov 249409 HS-grad 9   
  
 marital\_status occupation relationship race sex \  
0 Never-married Adm-clerical Not-in-family White Male   
1 Married-civ-spouse Exec-managerial Husband White Male   
2 Divorced Handlers-cleaners Not-in-family White Male   
3 Married-civ-spouse Handlers-cleaners Husband Black Male   
4 Married-civ-spouse Prof-specialty Wife Black Female   
.. ... ... ... ... ...   
95 Never-married Handlers-cleaners Not-in-family White Male   
96 Married-civ-spouse Prof-specialty Husband White Male   
97 Married-civ-spouse Sales Husband White Male   
98 Divorced Exec-managerial Unmarried White Female   
99 Never-married Other-service Own-child Black Male   
  
 capital\_gain capital\_loss hours\_per\_week native\_country income   
0 2174 0 40 United-States <=50K   
1 0 0 13 United-States <=50K   
2 0 0 40 United-States <=50K   
3 0 0 40 United-States <=50K   
4 0 0 40 Cuba <=50K   
.. ... ... ... ... ...   
95 0 0 50 United-States <=50K   
96 0 1902 60 United-States >50K   
97 0 0 48 United-States >50K   
98 0 0 40 England <=50K   
99 0 0 40 United-States <=50K   
  
[100 rows x 15 columns]

titanic\_df

PassengerId Survived Pclass \  
0 1 0 3   
1 2 1 1   
2 3 1 3   
3 4 1 1   
4 5 0 3   
.. ... ... ...   
150 151 0 2   
151 152 1 1   
152 153 0 3   
153 154 0 3   
154 155 0 3   
  
 Name Sex Age SibSp \  
0 Braund, Mr. Owen Harris male 22.0 1   
1 Cumings, Mrs. John Bradley (Florence Briggs Th... female 38.0 1   
2 Heikkinen, Miss. Laina female 26.0 0   
3 Futrelle, Mrs. Jacques Heath (Lily May Peel) female 35.0 1   
4 Allen, Mr. William Henry male 35.0 0   
.. ... ... ... ...   
150 Bateman, Rev. Robert James male 51.0 0   
151 Pears, Mrs. Thomas (Edith Wearne) female 22.0 1   
152 Meo, Mr. Alfonzo male 55.5 0   
153 van Billiard, Mr. Austin Blyler male 40.5 0   
154 Olsen, Mr. Ole Martin male NaN 0   
  
 Parch Ticket Fare Cabin Embarked   
0 0 A/5 21171 7.2500 NaN S   
1 0 PC 17599 71.2833 C85 C   
2 0 STON/O2. 3101282 7.9250 NaN S   
3 0 113803 53.1000 C123 S   
4 0 373450 8.0500 NaN S   
.. ... ... ... ... ...   
150 0 S.O.P. 1166 12.5250 NaN S   
151 0 113776 66.6000 C2 S   
152 0 A.5. 11206 8.0500 NaN S   
153 2 A/5. 851 14.5000 NaN S   
154 0 Fa 265302 7.3125 NaN S   
  
[155 rows x 12 columns]

Select relevant columns from Adult dataset (mix of nominal and ratio-scaled)

adult\_df = adult\_df[["age", "workclass", "education", "education\_num", "sex"]]  
  
adult\_df

age workclass education education\_num sex  
0 39 State-gov Bachelors 13 Male  
1 50 Self-emp-not-inc Bachelors 13 Male  
2 38 Private HS-grad 9 Male  
3 53 Private 11th 7 Male  
4 28 Private Bachelors 13 Female  
.. ... ... ... ... ...  
95 29 Local-gov Some-college 10 Male  
96 48 Self-emp-not-inc Doctorate 16 Male  
97 37 Private Some-college 10 Male  
98 48 Private Assoc-acdm 12 Female  
99 32 Federal-gov HS-grad 9 Male  
  
[100 rows x 5 columns]

Encode nominal attributes as integers for processing

label\_encoders = {}  
**for** column **in** adult\_df.columns:  
 **if** adult\_df[column].dtype == object:  
 le = LabelEncoder()  
 adult\_df[column] = le.fit\_transform(adult\_df[column])  
 label\_encoders[column] = le  
  
adult\_df

C:\Users\debat\AppData\Local\Temp\ipykernel\_8904\183426126.py:5: SettingWithCopyWarning:   
A value is trying to be set on a copy of a slice from a DataFrame.  
Try using .loc[row\_indexer,col\_indexer] = value instead  
  
See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user\_guide/indexing.html#returning-a-view-versus-a-copy  
 adult\_df[column] = le.fit\_transform(adult\_df[column])  
C:\Users\debat\AppData\Local\Temp\ipykernel\_8904\183426126.py:5: SettingWithCopyWarning:   
A value is trying to be set on a copy of a slice from a DataFrame.  
Try using .loc[row\_indexer,col\_indexer] = value instead  
  
See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user\_guide/indexing.html#returning-a-view-versus-a-copy  
 adult\_df[column] = le.fit\_transform(adult\_df[column])  
C:\Users\debat\AppData\Local\Temp\ipykernel\_8904\183426126.py:5: SettingWithCopyWarning:   
A value is trying to be set on a copy of a slice from a DataFrame.  
Try using .loc[row\_indexer,col\_indexer] = value instead  
  
See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user\_guide/indexing.html#returning-a-view-versus-a-copy  
 adult\_df[column] = le.fit\_transform(adult\_df[column])

age workclass education education\_num sex  
0 39 6 7 13 1  
1 50 5 7 13 1  
2 38 3 9 9 1  
3 53 3 1 7 1  
4 28 3 7 13 0  
.. ... ... ... ... ...  
95 29 2 12 10 1  
96 48 5 8 16 1  
97 37 3 12 10 1  
98 48 3 5 12 0  
99 32 1 9 9 1  
  
[100 rows x 5 columns]

Clean and preprocess Titanic dataset

titanic\_df.dropna(inplace=True)  
titanic\_df

PassengerId Survived Pclass \  
1 2 1 1   
3 4 1 1   
6 7 0 1   
10 11 1 3   
11 12 1 1   
21 22 1 2   
23 24 1 1   
27 28 0 1   
52 53 1 1   
54 55 0 1   
62 63 0 1   
66 67 1 2   
75 76 0 3   
88 89 1 1   
92 93 0 1   
96 97 0 1   
97 98 1 1   
102 103 0 1   
110 111 0 1   
118 119 0 1   
123 124 1 2   
124 125 0 1   
136 137 1 1   
137 138 0 1   
139 140 0 1   
148 149 0 2   
151 152 1 1   
  
 Name Sex Age SibSp \  
1 Cumings, Mrs. John Bradley (Florence Briggs Th... female 38.0 1   
3 Futrelle, Mrs. Jacques Heath (Lily May Peel) female 35.0 1   
6 McCarthy, Mr. Timothy J male 54.0 0   
10 Sandstrom, Miss. Marguerite Rut female 4.0 1   
11 Bonnell, Miss. Elizabeth female 58.0 0   
21 Beesley, Mr. Lawrence male 34.0 0   
23 Sloper, Mr. William Thompson male 28.0 0   
27 Fortune, Mr. Charles Alexander male 19.0 3   
52 Harper, Mrs. Henry Sleeper (Myna Haxtun) female 49.0 1   
54 Ostby, Mr. Engelhart Cornelius male 65.0 0   
62 Harris, Mr. Henry Birkhardt male 45.0 1   
66 Nye, Mrs. (Elizabeth Ramell) female 29.0 0   
75 Moen, Mr. Sigurd Hansen male 25.0 0   
88 Fortune, Miss. Mabel Helen female 23.0 3   
92 Chaffee, Mr. Herbert Fuller male 46.0 1   
96 Goldschmidt, Mr. George B male 71.0 0   
97 Greenfield, Mr. William Bertram male 23.0 0   
102 White, Mr. Richard Frasar male 21.0 0   
110 Porter, Mr. Walter Chamberlain male 47.0 0   
118 Baxter, Mr. Quigg Edmond male 24.0 0   
123 Webber, Miss. Susan female 32.5 0   
124 White, Mr. Percival Wayland male 54.0 0   
136 Newsom, Miss. Helen Monypeny female 19.0 0   
137 Futrelle, Mr. Jacques Heath male 37.0 1   
139 Giglio, Mr. Victor male 24.0 0   
148 Navratil, Mr. Michel ("Louis M Hoffman") male 36.5 0   
151 Pears, Mrs. Thomas (Edith Wearne) female 22.0 1   
  
 Parch Ticket Fare Cabin Embarked   
1 0 PC 17599 71.2833 C85 C   
3 0 113803 53.1000 C123 S   
6 0 17463 51.8625 E46 S   
10 1 PP 9549 16.7000 G6 S   
11 0 113783 26.5500 C103 S   
21 0 248698 13.0000 D56 S   
23 0 113788 35.5000 A6 S   
27 2 19950 263.0000 C23 C25 C27 S   
52 0 PC 17572 76.7292 D33 C   
54 1 113509 61.9792 B30 C   
62 0 36973 83.4750 C83 S   
66 0 C.A. 29395 10.5000 F33 S   
75 0 348123 7.6500 F G73 S   
88 2 19950 263.0000 C23 C25 C27 S   
92 0 W.E.P. 5734 61.1750 E31 S   
96 0 PC 17754 34.6542 A5 C   
97 1 PC 17759 63.3583 D10 D12 C   
102 1 35281 77.2875 D26 S   
110 0 110465 52.0000 C110 S   
118 1 PC 17558 247.5208 B58 B60 C   
123 0 27267 13.0000 E101 S   
124 1 35281 77.2875 D26 S   
136 2 11752 26.2833 D47 S   
137 0 113803 53.1000 C123 S   
139 0 PC 17593 79.2000 B86 C   
148 2 230080 26.0000 F2 S   
151 0 113776 66.6000 C2 S

Select relevant columns from Titanic dataset (mix of nominal and ratio-scaled)

titanic\_df = titanic\_df[["Age", "Sex", "Pclass", "Fare", "Embarked"]]  
titanic\_df

Age Sex Pclass Fare Embarked  
1 38.0 female 1 71.2833 C  
3 35.0 female 1 53.1000 S  
6 54.0 male 1 51.8625 S  
10 4.0 female 3 16.7000 S  
11 58.0 female 1 26.5500 S  
21 34.0 male 2 13.0000 S  
23 28.0 male 1 35.5000 S  
27 19.0 male 1 263.0000 S  
52 49.0 female 1 76.7292 C  
54 65.0 male 1 61.9792 C  
62 45.0 male 1 83.4750 S  
66 29.0 female 2 10.5000 S  
75 25.0 male 3 7.6500 S  
88 23.0 female 1 263.0000 S  
92 46.0 male 1 61.1750 S  
96 71.0 male 1 34.6542 C  
97 23.0 male 1 63.3583 C  
102 21.0 male 1 77.2875 S  
110 47.0 male 1 52.0000 S  
118 24.0 male 1 247.5208 C  
123 32.5 female 2 13.0000 S  
124 54.0 male 1 77.2875 S  
136 19.0 female 1 26.2833 S  
137 37.0 male 1 53.1000 S  
139 24.0 male 1 79.2000 C  
148 36.5 male 2 26.0000 S  
151 22.0 female 1 66.6000 S

Encode Nominal as Integers for processing

label\_encoders\_titanic = {}  
**for** column **in** titanic\_df.columns:  
 **if** titanic\_df[column].dtype == object:  
 le = LabelEncoder()  
 titanic\_df[column] = le.fit\_transform(titanic\_df[column])  
 label\_encoders[column] = le  
  
titanic\_df

C:\Users\debat\AppData\Local\Temp\ipykernel\_8904\3305425594.py:5: SettingWithCopyWarning:   
A value is trying to be set on a copy of a slice from a DataFrame.  
Try using .loc[row\_indexer,col\_indexer] = value instead  
  
See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user\_guide/indexing.html#returning-a-view-versus-a-copy  
 titanic\_df[column] = le.fit\_transform(titanic\_df[column])  
C:\Users\debat\AppData\Local\Temp\ipykernel\_8904\3305425594.py:5: SettingWithCopyWarning:   
A value is trying to be set on a copy of a slice from a DataFrame.  
Try using .loc[row\_indexer,col\_indexer] = value instead  
  
See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user\_guide/indexing.html#returning-a-view-versus-a-copy  
 titanic\_df[column] = le.fit\_transform(titanic\_df[column])

Age Sex Pclass Fare Embarked  
1 38.0 0 1 71.2833 0  
3 35.0 0 1 53.1000 1  
6 54.0 1 1 51.8625 1  
10 4.0 0 3 16.7000 1  
11 58.0 0 1 26.5500 1  
21 34.0 1 2 13.0000 1  
23 28.0 1 1 35.5000 1  
27 19.0 1 1 263.0000 1  
52 49.0 0 1 76.7292 0  
54 65.0 1 1 61.9792 0  
62 45.0 1 1 83.4750 1  
66 29.0 0 2 10.5000 1  
75 25.0 1 3 7.6500 1  
88 23.0 0 1 263.0000 1  
92 46.0 1 1 61.1750 1  
96 71.0 1 1 34.6542 0  
97 23.0 1 1 63.3583 0  
102 21.0 1 1 77.2875 1  
110 47.0 1 1 52.0000 1  
118 24.0 1 1 247.5208 0  
123 32.5 0 2 13.0000 1  
124 54.0 1 1 77.2875 1  
136 19.0 0 1 26.2833 1  
137 37.0 1 1 53.1000 1  
139 24.0 1 1 79.2000 0  
148 36.5 1 2 26.0000 1  
151 22.0 0 1 66.6000 1

Combine the datasets into a list for further processing

*# Combine the datasets into a list for further processing*  
datasets = {  
 "Adult Dataset": adult\_df,  
 "Titanic Dataset": titanic\_df  
}

Compute Correlation Matrix

**def** correlation\_coefficient(a, b):  
 *"""Calculate the Pearson Correlation Coefficient between two vectors."""*  
 **try**:  
 **return** np.corrcoef(a, b)[0, 1]  
 **except** Exception **as** e:  
 **return** np.nan  
  
*# Function to create the Correlation matrix*  
**def** calculate\_correlation\_matrix(dataset):  
 n = len(dataset)  
 correlation\_matrix = np.zeros((n, n))  
   
 **for** i **in** range(n):  
 **for** j **in** range(n):  
 correlation\_matrix[i, j] = correlation\_coefficient(dataset.iloc[i].values, dataset.iloc[j].values)  
   
 **return** pd.DataFrame(correlation\_matrix)

Calculate Correlation Matrix

*For Adult Dataset*

correlation\_matrix\_adult = calculate\_correlation\_matrix(adult\_df)  
correlation\_matrix\_adult

0 1 2 3 4 5 6 \  
0 1.000000 0.997350 0.987351 0.979855 0.982608 0.987214 0.979155   
1 0.997350 1.000000 0.992346 0.990383 0.971402 0.981827 0.990318   
2 0.987351 0.992346 1.000000 0.977230 0.966611 0.986108 0.986540   
3 0.979855 0.990383 0.977230 1.000000 0.931087 0.947285 0.996425   
4 0.982608 0.971402 0.966611 0.931087 1.000000 0.994572 0.930666   
.. ... ... ... ... ... ... ...   
95 0.952796 0.951934 0.977942 0.912323 0.964839 0.984216 0.931341   
96 0.998811 0.996954 0.988350 0.978175 0.986529 0.990747 0.976581   
97 0.974860 0.977875 0.995414 0.952534 0.966230 0.987763 0.967844   
98 0.996972 0.999669 0.990512 0.990242 0.972744 0.981685 0.988416   
99 0.978260 0.981818 0.995396 0.957604 0.973384 0.991406 0.968203   
  
 7 8 9 ... 90 91 92 93 \  
0 0.985819 0.973770 0.996797 ... 0.990631 0.974937 0.952244 0.984207   
1 0.993768 0.963523 0.997273 ... 0.997941 0.976220 0.947628 0.983245   
2 0.995618 0.969761 0.991901 ... 0.992350 0.993895 0.973154 0.995007   
3 0.989915 0.918105 0.979980 ... 0.995363 0.948173 0.903421 0.954681   
4 0.948588 0.995967 0.983983 ... 0.957142 0.968306 0.965757 0.978583   
.. ... ... ... ... ... ... ... ...   
95 0.955238 0.982308 0.963549 ... 0.946838 0.993654 0.997665 0.990267   
96 0.983955 0.978473 0.999202 ... 0.990899 0.976483 0.955091 0.985272   
97 0.983614 0.976983 0.981705 ... 0.976156 0.999626 0.989609 0.997909   
98 0.991396 0.963678 0.997955 ... 0.997641 0.973235 0.944175 0.980862   
99 0.982442 0.980936 0.988149 ... 0.980528 0.996103 0.984183 0.995884   
  
 94 95 96 97 98 99   
0 0.990559 0.952796 0.998811 0.974860 0.996972 0.978260   
1 0.987215 0.951934 0.996954 0.977875 0.999669 0.981818   
2 0.983535 0.977942 0.988350 0.995414 0.990512 0.995396   
3 0.959776 0.912323 0.978175 0.952534 0.990242 0.957604   
4 0.993915 0.964839 0.986529 0.966230 0.972744 0.973384   
.. ... ... ... ... ... ...   
95 0.969546 1.000000 0.958159 0.992814 0.949612 0.990722   
96 0.995602 0.958159 1.000000 0.977120 0.997514 0.983073   
97 0.978402 0.992814 0.977120 1.000000 0.975251 0.997584   
98 0.988923 0.949612 0.997514 0.975251 1.000000 0.980972   
99 0.987223 0.990722 0.983073 0.997584 0.980972 1.000000   
  
[100 rows x 100 columns]

*For Titanic Dataset*

correlation\_matrix\_titanic = calculate\_correlation\_matrix(titanic\_df)  
correlation\_matrix\_titanic

0 1 2 3 4 5 6 \  
0 1.000000 0.993438 0.919708 0.925325 0.674371 0.601460 0.975698   
1 0.993438 1.000000 0.958218 0.879845 0.753852 0.687969 0.994154   
2 0.919708 0.958218 1.000000 0.712509 0.909944 0.866324 0.983353   
3 0.925325 0.879845 0.712509 1.000000 0.367142 0.280890 0.824795   
4 0.674371 0.753852 0.909944 0.367142 1.000000 0.995190 0.819495   
5 0.601460 0.687969 0.866324 0.280890 0.995190 1.000000 0.761294   
6 0.975698 0.994154 0.983353 0.824795 0.819495 0.761294 1.000000   
7 0.892647 0.835749 0.644276 0.981587 0.269311 0.176781 0.772517   
8 0.995210 0.999768 0.953679 0.886621 0.743310 0.676623 0.992425   
9 0.917958 0.956767 0.999920 0.709759 0.911627 0.868651 0.982463   
10 0.999916 0.993691 0.920753 0.922606 0.676151 0.603121 0.976317   
11 0.586313 0.674354 0.855988 0.267991 0.992610 0.999181 0.748391   
12 0.514880 0.608010 0.807828 0.188809 0.977701 0.993312 0.688286   
13 0.899952 0.844715 0.656760 0.983598 0.285164 0.192984 0.782849   
14 0.981440 0.996751 0.977880 0.839658 0.803135 0.742859 0.999608   
15 0.697116 0.773668 0.922489 0.394079 0.999193 0.991746 0.837030   
16 0.984967 0.958818 0.838373 0.970090 0.536940 0.454802 0.923426   
17 0.963749 0.927261 0.781920 0.983040 0.453165 0.366543 0.882174   
18 0.952177 0.980598 0.995678 0.772886 0.867522 0.816266 0.995979   
19 0.904026 0.849637 0.663848 0.984075 0.294109 0.202236 0.788686   
20 0.620265 0.705193 0.877352 0.307199 0.996774 0.999254 0.776057   
21 0.988953 0.999264 0.967654 0.861830 0.775961 0.712499 0.997386   
22 0.985852 0.998496 0.971436 0.856703 0.787108 0.724881 0.997961   
23 0.989423 0.999370 0.966838 0.863373 0.773931 0.710243 0.997149   
24 0.972736 0.939975 0.803826 0.979446 0.484868 0.399972 0.898512   
25 0.825618 0.884093 0.980519 0.569122 0.973190 0.947340 0.928780   
26 0.978661 0.949193 0.819682 0.977002 0.508782 0.424763 0.910088   
  
 7 8 9 ... 17 18 19 20 \  
0 0.892647 0.995210 0.917958 ... 0.963749 0.952177 0.904026 0.620265   
1 0.835749 0.999768 0.956767 ... 0.927261 0.980598 0.849637 0.705193   
2 0.644276 0.953679 0.999920 ... 0.781920 0.995678 0.663848 0.877352   
3 0.981587 0.886621 0.709759 ... 0.983040 0.772886 0.984075 0.307199   
4 0.269311 0.743310 0.911627 ... 0.453165 0.867522 0.294109 0.996774   
5 0.176781 0.676623 0.868651 ... 0.366543 0.816266 0.202236 0.999254   
6 0.772517 0.992425 0.982463 ... 0.882174 0.995979 0.788686 0.776057   
7 1.000000 0.844335 0.640809 ... 0.980528 0.712522 0.999658 0.200376   
8 0.844335 1.000000 0.952328 ... 0.933093 0.977476 0.857932 0.693802   
9 0.640809 0.952328 1.000000 ... 0.779074 0.995183 0.660500 0.879412   
10 0.891599 0.995384 0.918937 ... 0.963164 0.953009 0.903014 0.621688   
11 0.158591 0.662480 0.858213 ... 0.349063 0.804574 0.184045 0.999078   
12 0.073910 0.595811 0.810849 ... 0.267674 0.750107 0.099762 0.990900   
13 0.999859 0.853044 0.653319 ... 0.983618 0.723959 0.999935 0.216593   
14 0.789992 0.995461 0.976868 ... 0.895013 0.993080 0.805590 0.758153   
15 0.299276 0.763857 0.924292 ... 0.480811 0.882669 0.323915 0.993413   
16 0.956873 0.963423 0.836050 ... 0.995215 0.885357 0.964110 0.475582   
17 0.980528 0.933093 0.779074 ... 1.000000 0.836438 0.985275 0.388607   
18 0.712522 0.977476 0.995183 ... 0.836438 1.000000 0.730437 0.829249   
19 0.999658 0.857932 0.660500 ... 0.985275 0.730437 1.000000 0.225628   
20 0.200376 0.693802 0.879412 ... 0.388607 0.829249 0.225628 1.000000   
21 0.816380 0.998669 0.966449 ... 0.913895 0.986902 0.831046 0.728610   
22 0.805580 0.997263 0.970112 ... 0.906064 0.989061 0.820552 0.741658   
23 0.818231 0.998825 0.965620 ... 0.915195 0.986379 0.832829 0.726412   
24 0.972742 0.945445 0.801248 ... 0.999278 0.855589 0.978447 0.421488   
25 0.482654 0.876786 0.981455 ... 0.645182 0.958178 0.505192 0.954333   
26 0.966053 0.953936 0.816975 ... 0.997898 0.869324 0.972372 0.446578   
  
 21 22 23 24 25 26   
0 0.988953 0.985852 0.989423 0.972736 0.825618 0.978661   
1 0.999264 0.998496 0.999370 0.939975 0.884093 0.949193   
2 0.967654 0.971436 0.966838 0.803826 0.980519 0.819682   
3 0.861830 0.856703 0.863373 0.979446 0.569122 0.977002   
4 0.775961 0.787108 0.773931 0.484868 0.973190 0.508782   
5 0.712499 0.724881 0.710243 0.399972 0.947340 0.424763   
6 0.997386 0.997961 0.997149 0.898512 0.928780 0.910088   
7 0.816380 0.805580 0.818231 0.972742 0.482654 0.966053   
8 0.998669 0.997263 0.998825 0.945445 0.876786 0.953936   
9 0.966449 0.970112 0.965620 0.801248 0.981455 0.816975   
10 0.989397 0.986127 0.989859 0.972162 0.826771 0.978138   
11 0.698693 0.712487 0.696399 0.382525 0.940667 0.408168   
12 0.634393 0.648889 0.631920 0.302573 0.907857 0.328379   
13 0.825760 0.815311 0.827568 0.976391 0.497025 0.970209   
14 0.999018 0.999097 0.998871 0.910445 0.918061 0.921328   
15 0.795318 0.805268 0.793367 0.512222 0.979875 0.535128   
16 0.948722 0.942214 0.949731 0.998156 0.716015 0.999120   
17 0.913895 0.906064 0.915195 0.999278 0.645182 0.997898   
18 0.986902 0.989061 0.986379 0.855589 0.958178 0.869324   
19 0.831046 0.820552 0.832829 0.978447 0.505192 0.972372   
20 0.728610 0.741658 0.726412 0.421488 0.954333 0.446578   
21 1.000000 0.999294 0.999995 0.927869 0.899629 0.937636   
22 0.999294 1.000000 0.999232 0.920417 0.907114 0.931298   
23 0.999995 0.999232 1.000000 0.929061 0.898224 0.938747   
24 0.927869 0.920417 0.929061 1.000000 0.672388 0.999371   
25 0.899629 0.907114 0.898224 0.672388 1.000000 0.692271   
26 0.937636 0.931298 0.938747 0.999371 0.692271 1.000000   
  
[27 rows x 27 columns]

Explanation

Correlation Calculation: This metric measures the degree of linear relationship between two variables. A high positive value indicates that as one variable increases, the other also tends to increase, while a high negative value indicates an inverse relationship.

Handling Different Data Types: Correlation is best suited for interval and ratio-scaled data. It may not be meaningful for nominal or ordinal data unless they are encoded or transformed appropriately.

Observation and Analysis

The resulting matrices will show the correlation coefficients between each pair of data points. A value close to 1 or −1 indicates a strong relationship, while a value close to 0 indicates little to no linear relationship.

Correlation as a Similarity Metric is useful for understanding the degree to which data points share a linear relationship, which can be especially important in fields like finance and economics.

# Data Mining LAB : Experiment 4

Submitted By:

Name: Debatreya Das

Roll No. 12212070

CS A4

Data Mining LAB

Problem Statement

Select a dataset which have issues of missing values and noisy data points. This information can be checked from metadata or documentation provided with the dataset. Apply different missing values handing methods namely

Ignore the tuple,

Use a global constant to fill in the missing value,

Use a measure of central tendency for the attribute (e.g., the mean or median) to fill in the missing value,

Use the attribute mean or median for all samples belonging to the same class as the given tuple,

Use the most probable value to fill in the missing value on your datasets.

Further, address the issue of noisy data points still pertaining in the datasets even after handling the missing values using

Binning and

Regression methods.

Analyze the effect of different techniques on dataset in terms of statistical parameters such as central tendency and dispersion.

Dataset

(Link won’ work)

Code and Output

Observations

After applying various techniques for handling missing values and noisy data points, we can observe the following effects on the central tendency and dispersion of the selected attributes (chol for cholesterol and trestbps for resting blood pressure):

Ignoring Tuples with Missing Values: Central Tendency: The mean, median, and mode values are calculated based on the remaining data, potentially leading to biased estimates if the missing data is not randomly distributed. Dispersion: Ignoring tuples typically reduces the dataset size, which can affect the variance and standard deviation. If the ignored data had extreme values, the range might decrease, and variance and standard deviation might be underestimated.

Using a Global Constant: Central Tendency: Replacing missing values with a constant (e.g., -1) significantly alters the mean, median, and mode, especially if the constant is far from the actual data distribution. Dispersion: The introduction of a constant leads to a distortion in variance and standard deviation, often inflating these measures since the constant does not reflect the natural variability of the data.

Filling with Mean/Median: Central Tendency: Filling missing values with the mean or median helps maintain the dataset's central tendency. However, it might reduce variability and lead to a slightly biased estimate if the missing data is not random. Dispersion: The variance and standard deviation might be slightly reduced as filling with the mean or median introduces less variability compared to actual observations.

Using Class-Based Mean/Median: Central Tendency: This approach maintains the central tendency more accurately within specific classes, especially in datasets with distinct subgroups. It reduces the bias introduced by filling with a global mean or median. Dispersion: Variance and standard deviation are better preserved within each class, maintaining the natural variability of the data.

Using the Most Probable Value (Mode): Central Tendency: Filling missing values with the mode maintains the most frequent value in the dataset, which may lead to a higher peak in the distribution. Dispersion: This method might reduce the range and variability, especially if the mode is a frequent and non-extreme value, leading to an underestimation of variance and standard deviation.

Binning: Central Tendency: Binning smooths the data by replacing values with the bin’s representative (e.g., bin mean), which can help reduce the impact of noise on central tendency measures. Dispersion: Binning typically reduces the variance and standard deviation by grouping data into larger intervals, leading to less variation within each bin.

Regression: Central Tendency: Regression replaces noisy values with predicted values based on other attributes, preserving the central tendency while accounting for relationships between variables. Dispersion: This method reduces noise-induced variability, potentially lowering variance and standard deviation if the original noisy data had outliers.

Conclusion

From the analysis, we can draw the following conclusions:

Suitability of Techniques:

Ignoring Tuples: Best used when the proportion of missing data is low, as it avoids introducing bias but risks losing valuable information.

Global Constant: Should be used cautiously as it can distort the data's central tendency and dispersion, especially if the constant is arbitrary.

Filling with Mean/Median: Provides a simple and effective method to handle missing values while preserving central tendency, though it may reduce variability.

Class-Based Mean/Median: More accurate for datasets with distinct subgroups, preserving the natural characteristics within each class.

Most Probable Value: Effective when the mode is representative, though it may reduce variability.

Binning: Useful for reducing the impact of noise, especially when the data has outliers, but may oversimplify the data distribution.

Regression: Provides a sophisticated method to smooth noisy data by leveraging relationships between variables, preserving central tendency while reducing noise-induced variability.

Impact on Statistical Parameters:

Techniques that introduce less arbitrary changes (like class-based mean/median and regression) tend to preserve the dataset's natural characteristics better than those that impose constant or mode values.

Techniques that smooth the data (like binning and regression) effectively reduce noise but may also reduce the dataset's variability.

Jupyter Notebook

Lab Experiment 4 : Data Mining

Apply different missing values handing methods namely

Ignore the tuple,

Use a global constant to fill in the missing value,

Use a measure of central tendency for the attribute (e.g., the mean or median) to fill in the missing value,

Use the attribute mean or median for all samples belonging to the same class as the given tuple,

Use the most probable value to fill in the missing value on your datasets.

Further, address the issue of noisy data points still pertaining in the datasets even after handling the missing values using

Binning and

Regression methods.

Analyze the effect of different techniques on dataset in terms of statistical parameters such as central tendency and dispersion.

Importing Libraries

**import** pandas **as** pd  
**import** numpy **as** np

Load the dataset

*# url = "https://archive.ics.uci.edu/ml/machine-learning-databases/heart-disease/processed.cleveland.data"*  
*# column\_names = ['age', 'sex', 'cp', 'trestbps', 'chol', 'fbs', 'restecg', 'thalach', 'exang', 'oldpeak', 'slope', 'ca', 'thal', 'target']*  
*# heart\_df = pd.read\_csv(url, header=None, names=column\_names, na\_values='?')*  
  
url = "../heart/heart.csv"  
heart\_df = pd.read\_csv(url)  
  
*# Display the first few rows and summary statistics*  
heart\_df

age sex cp trestbps chol fbs restecg thalach exang oldpeak \  
0 52 1 0 125 212 0 1 168 0 1.0   
1 53 1 0 140 203 1 0 155 1 3.1   
2 70 1 0 145 174 0 1 125 1 2.6   
3 61 1 0 148 203 0 1 161 0 0.0   
4 62 0 0 138 294 1 1 106 0 1.9   
... ... ... .. ... ... ... ... ... ... ...   
1020 59 1 1 140 221 0 1 164 1 0.0   
1021 60 1 0 125 258 0 0 141 1 2.8   
1022 47 1 0 110 275 0 0 118 1 1.0   
1023 50 0 0 110 254 0 0 159 0 0.0   
1024 54 1 0 120 188 0 1 113 0 1.4   
  
 slope ca thal target   
0 2 2 3 0   
1 0 0 3 0   
2 0 0 3 0   
3 2 1 3 0   
4 1 3 2 0   
... ... .. ... ...   
1020 2 0 2 1   
1021 1 1 3 0   
1022 1 1 2 0   
1023 2 0 2 1   
1024 1 1 3 0   
  
[1025 rows x 14 columns]

Remove rows having null

*# Ignore the tuples with missing values*  
heart\_df\_ignored = heart\_df.dropna()  
print("Dataset after ignoring tuples with missing values:")  
print(heart\_df\_ignored.info())

Dataset after ignoring tuples with missing values:  
<class 'pandas.core.frame.DataFrame'>  
RangeIndex: 1025 entries, 0 to 1024  
Data columns (total 14 columns):  
 # Column Non-Null Count Dtype   
--- ------ -------------- -----   
 0 age 1025 non-null int64   
 1 sex 1025 non-null int64   
 2 cp 1025 non-null int64   
 3 trestbps 1025 non-null int64   
 4 chol 1025 non-null int64   
 5 fbs 1025 non-null int64   
 6 restecg 1025 non-null int64   
 7 thalach 1025 non-null int64   
 8 exang 1025 non-null int64   
 9 oldpeak 1025 non-null float64  
 10 slope 1025 non-null int64   
 11 ca 1025 non-null int64   
 12 thal 1025 non-null int64   
 13 target 1025 non-null int64   
dtypes: float64(1), int64(13)  
memory usage: 112.2 KB  
None

Fill Missing with -1

heart\_df\_global\_constant = heart\_df.fillna(-1)  
  
print("Dataset after using global constant to fill missing values:")  
print(heart\_df\_global\_constant.info())

Dataset after using global constant to fill missing values:  
<class 'pandas.core.frame.DataFrame'>  
RangeIndex: 1025 entries, 0 to 1024  
Data columns (total 14 columns):  
 # Column Non-Null Count Dtype   
--- ------ -------------- -----   
 0 age 1025 non-null int64   
 1 sex 1025 non-null int64   
 2 cp 1025 non-null int64   
 3 trestbps 1025 non-null int64   
 4 chol 1025 non-null int64   
 5 fbs 1025 non-null int64   
 6 restecg 1025 non-null int64   
 7 thalach 1025 non-null int64   
 8 exang 1025 non-null int64   
 9 oldpeak 1025 non-null float64  
 10 slope 1025 non-null int64   
 11 ca 1025 non-null int64   
 12 thal 1025 non-null int64   
 13 target 1025 non-null int64   
dtypes: float64(1), int64(13)  
memory usage: 112.2 KB  
None

Use a Measure of Central Tendency (Mean/Median) to fill missing values

*# Fill missing values with the mean*  
heart\_df\_mean = heart\_df.fillna(heart\_df.mean())  
  
*# Fill missing values with the median*  
heart\_df\_median = heart\_df.fillna(heart\_df.median())  
  
print("Dataset after using mean to fill missing values:")  
print(heart\_df\_mean.info())  
  
print("Dataset after using median to fill missing values:")  
print(heart\_df\_median.info())

Dataset after using mean to fill missing values:  
<class 'pandas.core.frame.DataFrame'>  
RangeIndex: 1025 entries, 0 to 1024  
Data columns (total 14 columns):  
 # Column Non-Null Count Dtype   
--- ------ -------------- -----   
 0 age 1025 non-null int64   
 1 sex 1025 non-null int64   
 2 cp 1025 non-null int64   
 3 trestbps 1025 non-null int64   
 4 chol 1025 non-null int64   
 5 fbs 1025 non-null int64   
 6 restecg 1025 non-null int64   
 7 thalach 1025 non-null int64   
 8 exang 1025 non-null int64   
 9 oldpeak 1025 non-null float64  
 10 slope 1025 non-null int64   
 11 ca 1025 non-null int64   
 12 thal 1025 non-null int64   
 13 target 1025 non-null int64   
dtypes: float64(1), int64(13)  
memory usage: 112.2 KB  
None  
Dataset after using median to fill missing values:  
<class 'pandas.core.frame.DataFrame'>  
RangeIndex: 1025 entries, 0 to 1024  
Data columns (total 14 columns):  
 # Column Non-Null Count Dtype   
--- ------ -------------- -----   
 0 age 1025 non-null int64   
 1 sex 1025 non-null int64   
 2 cp 1025 non-null int64   
 3 trestbps 1025 non-null int64   
 4 chol 1025 non-null int64   
 5 fbs 1025 non-null int64   
 6 restecg 1025 non-null int64   
 7 thalach 1025 non-null int64   
 8 exang 1025 non-null int64   
 9 oldpeak 1025 non-null float64  
 10 slope 1025 non-null int64   
 11 ca 1025 non-null int64   
 12 thal 1025 non-null int64   
 13 target 1025 non-null int64   
dtypes: float64(1), int64(13)  
memory usage: 112.2 KB  
None

Fill missing values using the mean or median for samples belonging to the same class (e.g., the same target class).

*# Fill missing values using the mean of the same class*  
heart\_df\_class\_mean = heart\_df.copy()  
**for** column **in** heart\_df.columns:  
 heart\_df\_class\_mean[column].fillna(heart\_df.groupby('target')[column].transform('mean'), inplace=True)  
  
*# Fill missing values using the median of the same class*  
heart\_df\_class\_median = heart\_df.copy()  
**for** column **in** heart\_df.columns:  
 heart\_df\_class\_median[column].fillna(heart\_df.groupby('target')[column].transform('median'), inplace=True)  
  
print("Dataset after using class mean to fill missing values:")  
print(heart\_df\_class\_mean.info())  
  
print("Dataset after using class median to fill missing values:")  
print(heart\_df\_class\_median.info())

Dataset after using class mean to fill missing values:  
<class 'pandas.core.frame.DataFrame'>  
RangeIndex: 1025 entries, 0 to 1024  
Data columns (total 14 columns):  
 # Column Non-Null Count Dtype   
--- ------ -------------- -----   
 0 age 1025 non-null int64   
 1 sex 1025 non-null int64   
 2 cp 1025 non-null int64   
 3 trestbps 1025 non-null int64   
 4 chol 1025 non-null int64   
 5 fbs 1025 non-null int64   
 6 restecg 1025 non-null int64   
 7 thalach 1025 non-null int64   
 8 exang 1025 non-null int64   
 9 oldpeak 1025 non-null float64  
 10 slope 1025 non-null int64   
 11 ca 1025 non-null int64   
 12 thal 1025 non-null int64   
 13 target 1025 non-null int64   
dtypes: float64(1), int64(13)  
memory usage: 112.2 KB  
None  
Dataset after using class median to fill missing values:  
<class 'pandas.core.frame.DataFrame'>  
RangeIndex: 1025 entries, 0 to 1024  
Data columns (total 14 columns):  
 # Column Non-Null Count Dtype   
--- ------ -------------- -----   
 0 age 1025 non-null int64   
 1 sex 1025 non-null int64   
 2 cp 1025 non-null int64   
 3 trestbps 1025 non-null int64   
 4 chol 1025 non-null int64   
 5 fbs 1025 non-null int64   
 6 restecg 1025 non-null int64   
 7 thalach 1025 non-null int64   
 8 exang 1025 non-null int64   
 9 oldpeak 1025 non-null float64  
 10 slope 1025 non-null int64   
 11 ca 1025 non-null int64   
 12 thal 1025 non-null int64   
 13 target 1025 non-null int64   
dtypes: float64(1), int64(13)  
memory usage: 112.2 KB  
None

C:\Users\debat\AppData\Local\Temp\ipykernel\_15200\535031063.py:4: FutureWarning: A value is trying to be set on a copy of a DataFrame or Series through chained assignment using an inplace method.  
The behavior will change in pandas 3.0. This inplace method will never work because the intermediate object on which we are setting values always behaves as a copy.  
  
For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col: value}, inplace=True)' or df[col] = df[col].method(value) instead, to perform the operation inplace on the original object.  
  
  
 heart\_df\_class\_mean[column].fillna(heart\_df.groupby('target')[column].transform('mean'), inplace=True)  
C:\Users\debat\AppData\Local\Temp\ipykernel\_15200\535031063.py:9: FutureWarning: A value is trying to be set on a copy of a DataFrame or Series through chained assignment using an inplace method.  
The behavior will change in pandas 3.0. This inplace method will never work because the intermediate object on which we are setting values always behaves as a copy.  
  
For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col: value}, inplace=True)' or df[col] = df[col].method(value) instead, to perform the operation inplace on the original object.  
  
  
 heart\_df\_class\_median[column].fillna(heart\_df.groupby('target')[column].transform('median'), inplace=True)

Fill missing values with the most probable value, which could be inferred through methods like regression, k-NN, or similar techniques.

*# Fill missing values with the most probable value (mode)*  
heart\_df\_mode = heart\_df.apply(**lambda** x: x.fillna(x.mode()[0]))  
  
print("Dataset after using mode to fill missing values:")  
print(heart\_df\_mode.info())

Dataset after using mode to fill missing values:  
<class 'pandas.core.frame.DataFrame'>  
RangeIndex: 1025 entries, 0 to 1024  
Data columns (total 14 columns):  
 # Column Non-Null Count Dtype   
--- ------ -------------- -----   
 0 age 1025 non-null int64   
 1 sex 1025 non-null int64   
 2 cp 1025 non-null int64   
 3 trestbps 1025 non-null int64   
 4 chol 1025 non-null int64   
 5 fbs 1025 non-null int64   
 6 restecg 1025 non-null int64   
 7 thalach 1025 non-null int64   
 8 exang 1025 non-null int64   
 9 oldpeak 1025 non-null float64  
 10 slope 1025 non-null int64   
 11 ca 1025 non-null int64   
 12 thal 1025 non-null int64   
 13 target 1025 non-null int64   
dtypes: float64(1), int64(13)  
memory usage: 112.2 KB  
None

Handling Noisy Data Points

Binning Method

Binning is a simple technique that smooths noisy data by grouping it into bins and then replacing the values within each bin with a representative value (such as the mean, median, or boundaries).

Steps for Binning:

Equal-width Binning: Divides the range of the data into equal-sized intervals.

Equal-frequency Binning: Divides the data into intervals that each contain approximately the same number of data points.

We'll apply equal-width binning to smooth the chol (cholesterol) attribute, which might have noisy data.

*# Apply equal-width binning on 'chol' attribute*  
heart\_df\_binned = heart\_df\_mean.copy()  
  
*# Define the number of bins*  
num\_bins = 4  
  
*# Binning using pandas cut function*  
heart\_df\_binned['chol\_binned'] = pd.cut(heart\_df\_binned['chol'], bins=num\_bins, labels=False)  
  
*# Replace original 'chol' values with bin means*  
bin\_means = heart\_df\_binned.groupby('chol\_binned')['chol'].mean()  
heart\_df\_binned['chol'] = heart\_df\_binned['chol\_binned'].map(bin\_means)  
  
print("Dataset after applying equal-width binning on 'chol' attribute:")  
print(heart\_df\_binned[['chol', 'chol\_binned']].head())

Dataset after applying equal-width binning on 'chol' attribute:  
 chol chol\_binned  
0 204.80083 0  
1 204.80083 0  
2 204.80083 0  
3 204.80083 0  
4 276.40619 1

Regression Method

Regression can be used to predict and smooth out noisy data by fitting a regression model to the data. We'll use linear regression to predict the trestbps (resting blood pressure) attribute based on other attributes and replace its values with the predicted ones to smooth the data.

**from** sklearn.linear\_model **import** LinearRegression  
  
*# Prepare data for regression*  
regression\_df = heart\_df\_mean.dropna(subset=['trestbps'])  
X = regression\_df.drop(['trestbps', 'target'], axis=1)  
y = regression\_df['trestbps']  
  
*# Fit a linear regression model*  
regressor = LinearRegression()  
regressor.fit(X, y)  
  
*# Predict 'trestbps' values*  
heart\_df\_regression = heart\_df\_mean.copy()  
predicted\_trestbps = regressor.predict(heart\_df\_regression.drop(['trestbps', 'target'], axis=1))  
heart\_df\_regression['trestbps'] = predicted\_trestbps  
  
print("Dataset after applying regression on 'trestbps' attribute:")  
print(heart\_df\_regression[['trestbps']].head())

Dataset after applying regression on 'trestbps' attribute:  
 trestbps  
0 127.606252  
1 143.082032  
2 137.413752  
3 128.398309  
4 138.927470

Analyze the Effect of Different Techniques

**def** analyze\_statistics(df, attribute):  
 *"""Calculate and display central tendency and dispersion statistics for a given attribute."""*  
 mean = df[attribute].mean()  
 median = df[attribute].median()  
 mode = df[attribute].mode()[0]  
 range\_val = df[attribute].max() - df[attribute].min()  
 variance = df[attribute].var()  
 std\_dev = df[attribute].std()  
   
 print(f"Statistics for {attribute}:")  
 print(f"Mean: {mean}, Median: {median}, Mode: {mode}")  
 print(f"Range: {range\_val}, Variance: {variance}, Standard Deviation: {std\_dev}\n")  
  
*# Analyze the 'chol' and 'trestbps' attributes across different techniques*  
print("After Ignoring Tuples:")  
analyze\_statistics(heart\_df\_ignored, 'chol')  
analyze\_statistics(heart\_df\_ignored, 'trestbps')  
  
print("After Filling with Global Constant:")  
analyze\_statistics(heart\_df\_global\_constant, 'chol')  
analyze\_statistics(heart\_df\_global\_constant, 'trestbps')  
  
print("After Filling with Mean:")  
analyze\_statistics(heart\_df\_mean, 'chol')  
analyze\_statistics(heart\_df\_mean, 'trestbps')  
  
print("After Binning (chol):")  
analyze\_statistics(heart\_df\_binned, 'chol')  
  
print("After Regression (trestbps):")  
analyze\_statistics(heart\_df\_regression, 'trestbps')

After Ignoring Tuples:  
Statistics for chol:  
Mean: 246.0, Median: 240.0, Mode: 204  
Range: 438, Variance: 2661.787109375, Standard Deviation: 51.59251020618206  
  
Statistics for trestbps:  
Mean: 131.61170731707318, Median: 130.0, Mode: 120  
Range: 106, Variance: 306.835409679878, Standard Deviation: 17.516718005376408  
  
After Filling with Global Constant:  
Statistics for chol:  
Mean: 246.0, Median: 240.0, Mode: 204  
Range: 438, Variance: 2661.787109375, Standard Deviation: 51.59251020618206  
  
Statistics for trestbps:  
Mean: 131.61170731707318, Median: 130.0, Mode: 120  
Range: 106, Variance: 306.835409679878, Standard Deviation: 17.516718005376408  
  
After Filling with Mean:  
Statistics for chol:  
Mean: 246.0, Median: 240.0, Mode: 204  
Range: 438, Variance: 2661.787109375, Standard Deviation: 51.59251020618206  
  
Statistics for trestbps:  
Mean: 131.61170731707318, Median: 130.0, Mode: 120  
Range: 106, Variance: 306.835409679878, Standard Deviation: 17.516718005376408  
  
After Binning (chol):  
Statistics for chol:  
Mean: 246.0, Median: 276.4061895551257, Mode: 276.4061895551257  
Range: 359.1991701244813, Variance: 1992.448015590442, Standard Deviation: 44.636845941334634  
  
After Regression (trestbps):  
Statistics for trestbps:  
Mean: 131.61170731707318, Median: 131.40491906597907, Mode: 120.60533396201558  
Range: 32.82357606343098, Variance: 44.10907254282059, Standard Deviation: 6.641466144069439

Observation and Conclusion

Based on the results, we can draw conclusions about how different missing value handling and noise reduction techniques affect the dataset in terms of central tendency (mean, median, mode) and dispersion (range, variance, standard deviation).

# Data Mining LAB : Experiment 5

Submitted By:

Name: Debatreya Das

Roll No. 12212070

CS A4

Data Mining LAB

Problem Statement

Select a dataset which comprises numeric attributes of varying range. Apply different normalization techniques viz. Min-max normalization, z-score normalization, Decimal scaling on your datasets. Further, discretize the numeric attributes using Binning and Histogram analysis method. Analyze the effect of different techniques on dataset in terms of type of attributes, statistical parameters such as central tendency and dispersion and change in aptness of proximity metrics

Data Set

**WINE**:

For this experiment, I'll use the **Wine Quality** dataset from the . This dataset is ideal because it comprises several numeric attributes with varying ranges, making it suitable for applying different normalization and discretization techniques.

**Dataset Overview**

**Features (Input Variables):**

Fixed acidity

Volatile acidity

Citric acid

Residual sugar

Chlorides

Free sulfur dioxide

Total sulfur dioxide

Density

pH

Sulphates

Alcohol

**Target Variable:**

Quality (score between 0 and 10)

Code and Output

Observations

**Min-Max Normalization**: All attributes are scaled between 0 and 1, making them directly comparable, but it is sensitive to outliers.

**Z-Score Normalization**: The transformed data has a mean of 0 and unit variance, making it suitable for techniques that assume a normal distribution.

**Decimal Scaling**: The attributes are scaled by a power of 10. This technique may be more suited for simpler datasets with fewer variations.

**Discretization**: Both binning techniques reduce the granularity of numeric attributes, but equal-frequency binning captures distributional characteristics better than equal-width binning.

**Proximity Metrics**: The Euclidean distance changes significantly after normalization. The distances become more meaningful as attributes become comparable in magnitude.

Conclusion

**Normalization** techniques significantly impact both the distribution of attributes and their proximity metrics. Min-max normalization makes attributes comparable, whereas Z-score normalization ensures that attributes have a consistent scale but doesn’t retain the original range.

**Binning** techniques simplify the data by grouping similar values together, but different binning strategies (equal-width vs. equal-frequency) lead to different groupings.

**Proximity metrics** like Euclidean distance become more meaningful after normalization, as attributes are on a comparable scale.

Jupyter Notebook

Data Mining Lab : Experiment 5

Importing Libraries

**import** pandas **as** pd  
**import** numpy **as** np  
**from** sklearn.preprocessing **import** MinMaxScaler, StandardScaler  
**import** matplotlib.pyplot **as** plt  
**import** seaborn **as** sns

Loading Dataset :

datasetUrl = 'https://archive.ics.uci.edu/ml/machine-learning-databases/wine-quality/winequality-red.csv'  
df = pd.read\_csv(datasetUrl, sep=';')  
  
*# Display the first five rows*  
print(df.head())  
  
*# Get information about data types and missing values*  
print(df.info())  
  
*# Summary statistics*  
print(df.describe())

fixed acidity volatile acidity citric acid residual sugar chlorides \  
0 7.4 0.70 0.00 1.9 0.076   
1 7.8 0.88 0.00 2.6 0.098   
2 7.8 0.76 0.04 2.3 0.092   
3 11.2 0.28 0.56 1.9 0.075   
4 7.4 0.70 0.00 1.9 0.076   
  
 free sulfur dioxide total sulfur dioxide density pH sulphates \  
0 11.0 34.0 0.9978 3.51 0.56   
1 25.0 67.0 0.9968 3.20 0.68   
2 15.0 54.0 0.9970 3.26 0.65   
3 17.0 60.0 0.9980 3.16 0.58   
4 11.0 34.0 0.9978 3.51 0.56   
  
 alcohol quality   
0 9.4 5   
1 9.8 5   
2 9.8 5   
3 9.8 6   
4 9.4 5   
<class 'pandas.core.frame.DataFrame'>  
RangeIndex: 1599 entries, 0 to 1598  
Data columns (total 12 columns):  
 # Column Non-Null Count Dtype   
--- ------ -------------- -----   
 0 fixed acidity 1599 non-null float64  
 1 volatile acidity 1599 non-null float64  
 2 citric acid 1599 non-null float64  
 3 residual sugar 1599 non-null float64  
 4 chlorides 1599 non-null float64  
 5 free sulfur dioxide 1599 non-null float64  
 6 total sulfur dioxide 1599 non-null float64  
 7 density 1599 non-null float64  
 8 pH 1599 non-null float64  
 9 sulphates 1599 non-null float64  
 10 alcohol 1599 non-null float64  
 11 quality 1599 non-null int64   
dtypes: float64(11), int64(1)  
memory usage: 150.0 KB  
None  
 fixed acidity volatile acidity citric acid residual sugar \  
count 1599.000000 1599.000000 1599.000000 1599.000000   
mean 8.319637 0.527821 0.270976 2.538806   
std 1.741096 0.179060 0.194801 1.409928   
min 4.600000 0.120000 0.000000 0.900000   
25% 7.100000 0.390000 0.090000 1.900000   
50% 7.900000 0.520000 0.260000 2.200000   
75% 9.200000 0.640000 0.420000 2.600000   
max 15.900000 1.580000 1.000000 15.500000   
  
 chlorides free sulfur dioxide total sulfur dioxide density \  
count 1599.000000 1599.000000 1599.000000 1599.000000   
mean 0.087467 15.874922 46.467792 0.996747   
std 0.047065 10.460157 32.895324 0.001887   
min 0.012000 1.000000 6.000000 0.990070   
25% 0.070000 7.000000 22.000000 0.995600   
50% 0.079000 14.000000 38.000000 0.996750   
75% 0.090000 21.000000 62.000000 0.997835   
max 0.611000 72.000000 289.000000 1.003690   
  
 pH sulphates alcohol quality   
count 1599.000000 1599.000000 1599.000000 1599.000000   
mean 3.311113 0.658149 10.422983 5.636023   
std 0.154386 0.169507 1.065668 0.807569   
min 2.740000 0.330000 8.400000 3.000000   
25% 3.210000 0.550000 9.500000 5.000000   
50% 3.310000 0.620000 10.200000 6.000000   
75% 3.400000 0.730000 11.100000 6.000000   
max 4.010000 2.000000 14.900000 8.000000

*Plot Original Data*

df.hist(bins=10, figsize=(10, 6))  
plt.suptitle("Original Data", fontsize=16)  
plt.show()

Normalization Techniques

*Min-Max Normalization*

Scales the data to a fixed range, typically [0, 1].

*# Initialize the MinMaxScaler*  
min\_max\_scaler = MinMaxScaler()  
  
*# Apply Min-Max normalization to all features except the target*  
df\_min\_max = df.copy()  
df\_min\_max.iloc[:, :-1] = min\_max\_scaler.fit\_transform(df\_min\_max.iloc[:, :-1])  
  
df\_min\_max

fixed acidity volatile acidity citric acid residual sugar chlorides \  
0 0.247788 0.397260 0.00 0.068493 0.106845   
1 0.283186 0.520548 0.00 0.116438 0.143573   
2 0.283186 0.438356 0.04 0.095890 0.133556   
3 0.584071 0.109589 0.56 0.068493 0.105175   
4 0.247788 0.397260 0.00 0.068493 0.106845   
... ... ... ... ... ...   
1594 0.141593 0.328767 0.08 0.075342 0.130217   
1595 0.115044 0.294521 0.10 0.089041 0.083472   
1596 0.150442 0.267123 0.13 0.095890 0.106845   
1597 0.115044 0.359589 0.12 0.075342 0.105175   
1598 0.123894 0.130137 0.47 0.184932 0.091820   
  
 free sulfur dioxide total sulfur dioxide density pH \  
0 0.140845 0.098940 0.567548 0.606299   
1 0.338028 0.215548 0.494126 0.362205   
2 0.197183 0.169611 0.508811 0.409449   
3 0.225352 0.190813 0.582232 0.330709   
4 0.140845 0.098940 0.567548 0.606299   
... ... ... ... ...   
1594 0.436620 0.134276 0.354626 0.559055   
1595 0.535211 0.159011 0.370778 0.614173   
1596 0.394366 0.120141 0.416300 0.535433   
1597 0.436620 0.134276 0.396476 0.653543   
1598 0.239437 0.127208 0.397944 0.511811   
  
 sulphates alcohol quality   
0 0.137725 0.153846 5   
1 0.209581 0.215385 5   
2 0.191617 0.215385 5   
3 0.149701 0.215385 6   
4 0.137725 0.153846 5   
... ... ... ...   
1594 0.149701 0.323077 5   
1595 0.257485 0.430769 6   
1596 0.251497 0.400000 6   
1597 0.227545 0.276923 5   
1598 0.197605 0.400000 6   
  
[1599 rows x 12 columns]

*Plot the Min Max Normalization*

df\_min\_max.hist(bins=10, figsize=(10, 6))  
plt.suptitle("Min-Max Normalized Data", fontsize=16)  
plt.show()

*Z-Score Normalization*

Transforms the data to have a mean of 0 and a standard deviation of 1.

*# Initialize the StandardScaler*  
z\_score\_scaler = StandardScaler()  
  
*# Apply Z-score normalization to all features except the target*  
df\_z\_score = df.copy()  
df\_z\_score.iloc[:, :-1] = z\_score\_scaler.fit\_transform(df\_z\_score.iloc[:, :-1])  
  
df\_z\_score

fixed acidity volatile acidity citric acid residual sugar chlorides \  
0 -0.528360 0.961877 -1.391472 -0.453218 -0.243707   
1 -0.298547 1.967442 -1.391472 0.043416 0.223875   
2 -0.298547 1.297065 -1.186070 -0.169427 0.096353   
3 1.654856 -1.384443 1.484154 -0.453218 -0.264960   
4 -0.528360 0.961877 -1.391472 -0.453218 -0.243707   
... ... ... ... ... ...   
1594 -1.217796 0.403229 -0.980669 -0.382271 0.053845   
1595 -1.390155 0.123905 -0.877968 -0.240375 -0.541259   
1596 -1.160343 -0.099554 -0.723916 -0.169427 -0.243707   
1597 -1.390155 0.654620 -0.775267 -0.382271 -0.264960   
1598 -1.332702 -1.216849 1.021999 0.752894 -0.434990   
  
 free sulfur dioxide total sulfur dioxide density pH \  
0 -0.466193 -0.379133 0.558274 1.288643   
1 0.872638 0.624363 0.028261 -0.719933   
2 -0.083669 0.229047 0.134264 -0.331177   
3 0.107592 0.411500 0.664277 -0.979104   
4 -0.466193 -0.379133 0.558274 1.288643   
... ... ... ... ...   
1594 1.542054 -0.075043 -0.978765 0.899886   
1595 2.211469 0.137820 -0.862162 1.353436   
1596 1.255161 -0.196679 -0.533554 0.705508   
1597 1.542054 -0.075043 -0.676657 1.677400   
1598 0.203223 -0.135861 -0.666057 0.511130   
  
 sulphates alcohol quality   
0 -0.579207 -0.960246 5   
1 0.128950 -0.584777 5   
2 -0.048089 -0.584777 5   
3 -0.461180 -0.584777 6   
4 -0.579207 -0.960246 5   
... ... ... ...   
1594 -0.461180 0.072294 5   
1595 0.601055 0.729364 6   
1596 0.542042 0.541630 6   
1597 0.305990 -0.209308 5   
1598 0.010924 0.541630 6   
  
[1599 rows x 12 columns]

*Plot Z-Score Normalization Data*

df\_z\_score.hist(bins=10, figsize=(10, 6))  
plt.suptitle("Z-score Normalized Data", fontsize=16)  
plt.show()

*Decimal Scaling*

Moves the decimal point of values to bring them within a certain range, typically [-1, 1].

*# Function for Decimal Scaling*  
**def** decimal\_scaling(df):  
 df\_decimal = df.copy()  
 **for** column **in** df\_decimal.columns[:-1]:  
 max\_abs = df\_decimal[column].abs().max()  
 j = np.ceil(np.log10(max\_abs + 1))  
 df\_decimal[column] = df\_decimal[column] / (10 \*\* j)  
 **return** df\_decimal  
  
*# Apply Decimal Scaling*  
df\_decimal = decimal\_scaling(df)  
  
df\_decimal

fixed acidity volatile acidity citric acid residual sugar chlorides \  
0 0.074 0.0700 0.000 0.019 0.0076   
1 0.078 0.0880 0.000 0.026 0.0098   
2 0.078 0.0760 0.004 0.023 0.0092   
3 0.112 0.0280 0.056 0.019 0.0075   
4 0.074 0.0700 0.000 0.019 0.0076   
... ... ... ... ... ...   
1594 0.062 0.0600 0.008 0.020 0.0090   
1595 0.059 0.0550 0.010 0.022 0.0062   
1596 0.063 0.0510 0.013 0.023 0.0076   
1597 0.059 0.0645 0.012 0.020 0.0075   
1598 0.060 0.0310 0.047 0.036 0.0067   
  
 free sulfur dioxide total sulfur dioxide density pH sulphates \  
0 0.11 0.034 0.099780 0.351 0.056   
1 0.25 0.067 0.099680 0.320 0.068   
2 0.15 0.054 0.099700 0.326 0.065   
3 0.17 0.060 0.099800 0.316 0.058   
4 0.11 0.034 0.099780 0.351 0.056   
... ... ... ... ... ...   
1594 0.32 0.044 0.099490 0.345 0.058   
1595 0.39 0.051 0.099512 0.352 0.076   
1596 0.29 0.040 0.099574 0.342 0.075   
1597 0.32 0.044 0.099547 0.357 0.071   
1598 0.18 0.042 0.099549 0.339 0.066   
  
 alcohol quality   
0 0.094 5   
1 0.098 5   
2 0.098 5   
3 0.098 6   
4 0.094 5   
... ... ...   
1594 0.105 5   
1595 0.112 6   
1596 0.110 6   
1597 0.102 5   
1598 0.110 6   
  
[1599 rows x 12 columns]

Discretization Techniques

*Binning (Equal Width)*

Divides the range of the data into intervals of equal size.

*# Define the number of bins*  
num\_bins = 10  
  
*# Apply Binning to all features except the target*  
df\_binned = df.copy()  
**for** column **in** df\_binned.columns[:-1]:  
 df\_binned[column] = pd.cut(df\_binned[column], bins=num\_bins, labels=False)  
  
df\_binned

fixed acidity volatile acidity citric acid residual sugar chlorides \  
0 2 3 0 0 1   
1 2 5 0 1 1   
2 2 4 0 0 1   
3 5 1 5 0 1   
4 2 3 0 0 1   
... ... ... ... ... ...   
1594 1 3 0 0 1   
1595 1 2 0 0 0   
1596 1 2 1 0 1   
1597 1 3 1 0 1   
1598 1 1 4 1 0   
  
 free sulfur dioxide total sulfur dioxide density pH sulphates \  
0 1 0 5 6 1   
1 3 2 4 3 2   
2 1 1 5 4 1   
3 2 1 5 3 1   
4 1 0 5 6 1   
... ... ... ... .. ...   
1594 4 1 3 5 1   
1595 5 1 3 6 2   
1596 3 1 4 5 2   
1597 4 1 3 6 2   
1598 2 1 3 5 1   
  
 alcohol quality   
0 1 5   
1 2 5   
2 2 5   
3 2 6   
4 1 5   
... ... ...   
1594 3 5   
1595 4 6   
1596 3 6   
1597 2 5   
1598 3 6   
  
[1599 rows x 12 columns]

Histogram-Based Binning

Uses the histogram of the data to determine bin edges, ensuring each bin has roughly the same number of samples.

*# Equal Frequency Binning*  
df\_hist\_binned = df.apply(**lambda** x: pd.qcut(x, q=5, labels=False, duplicates='drop'))  
  
df\_hist\_binned

fixed acidity volatile acidity citric acid residual sugar chlorides \  
0 1 4 0 1 1   
1 2 4 0 3 4   
2 2 4 0 2 3   
3 4 0 4 1 1   
4 1 4 0 1 1   
... ... ... ... ... ...   
1594 0 3 1 1 3   
1595 0 2 1 2 0   
1596 0 2 1 2 1   
1597 0 3 1 1 1   
1598 0 0 4 4 0   
  
 free sulfur dioxide total sulfur dioxide density pH sulphates \  
0 1 2 3 4 1   
1 4 3 2 1 3   
2 2 3 2 1 2   
3 3 3 3 0 1   
4 1 2 3 4 1   
... ... ... ... .. ...   
1594 4 2 0 4 1   
1595 4 3 0 4 3   
1596 4 2 1 3 3   
1597 4 2 1 4 3   
1598 3 2 1 3 3   
  
 alcohol quality   
0 0 0   
1 1 0   
2 1 0   
3 1 1   
4 0 0   
... ... ...   
1594 2 0   
1595 3 1   
1596 3 1   
1597 2 0   
1598 3 1   
  
[1599 rows x 12 columns]

Analyzing the Effects of Different Techniques

We'll analyze the impact of normalization and discretization on:

**Type of Attributes**

**Statistical Parameters (Central Tendency and Dispersion)**

**Aptness of Proximity Metrics**

*Type of Attributes*

**Normalization:** Does not change the type of attributes; they remain continuous.

**Discretization:** Converts continuous attributes into categorical (ordinal) attributes.

*Statistical Parameters*

*# Original Data Statistics*  
print("Original Data Statistics:")  
print(df.describe())  
  
*# Min-Max Normalized Data Statistics*  
print("\nMin-Max Normalized Data Statistics:")  
print(df\_min\_max.describe())  
  
*# Z-Score Normalized Data Statistics*  
print("\nZ-Score Normalized Data Statistics:")  
print(df\_z\_score.describe())  
  
*# Decimal Scaled Data Statistics*  
print("\nDecimal Scaled Data Statistics:")  
print(df\_decimal.describe())

Original Data Statistics:  
 fixed acidity volatile acidity citric acid residual sugar \  
count 1599.000000 1599.000000 1599.000000 1599.000000   
mean 8.319637 0.527821 0.270976 2.538806   
std 1.741096 0.179060 0.194801 1.409928   
min 4.600000 0.120000 0.000000 0.900000   
25% 7.100000 0.390000 0.090000 1.900000   
50% 7.900000 0.520000 0.260000 2.200000   
75% 9.200000 0.640000 0.420000 2.600000   
max 15.900000 1.580000 1.000000 15.500000   
  
 chlorides free sulfur dioxide total sulfur dioxide density \  
count 1599.000000 1599.000000 1599.000000 1599.000000   
mean 0.087467 15.874922 46.467792 0.996747   
std 0.047065 10.460157 32.895324 0.001887   
min 0.012000 1.000000 6.000000 0.990070   
25% 0.070000 7.000000 22.000000 0.995600   
50% 0.079000 14.000000 38.000000 0.996750   
75% 0.090000 21.000000 62.000000 0.997835   
max 0.611000 72.000000 289.000000 1.003690   
  
 pH sulphates alcohol quality   
count 1599.000000 1599.000000 1599.000000 1599.000000   
mean 3.311113 0.658149 10.422983 5.636023   
std 0.154386 0.169507 1.065668 0.807569   
min 2.740000 0.330000 8.400000 3.000000   
25% 3.210000 0.550000 9.500000 5.000000   
50% 3.310000 0.620000 10.200000 6.000000   
75% 3.400000 0.730000 11.100000 6.000000   
max 4.010000 2.000000 14.900000 8.000000   
  
Min-Max Normalized Data Statistics:  
 fixed acidity volatile acidity citric acid residual sugar \  
count 1599.000000 1599.000000 1599.000000 1599.000000   
mean 0.329171 0.279329 0.270976 0.112247   
std 0.154079 0.122644 0.194801 0.096570   
min 0.000000 0.000000 0.000000 0.000000   
25% 0.221239 0.184932 0.090000 0.068493   
50% 0.292035 0.273973 0.260000 0.089041   
75% 0.407080 0.356164 0.420000 0.116438   
max 1.000000 1.000000 1.000000 1.000000   
  
 chlorides free sulfur dioxide total sulfur dioxide density \  
count 1599.000000 1599.000000 1599.000000 1599.000000   
mean 0.125988 0.209506 0.142996 0.490211   
std 0.078573 0.147326 0.116238 0.138571   
min 0.000000 0.000000 0.000000 0.000000   
25% 0.096828 0.084507 0.056537 0.406021   
50% 0.111853 0.183099 0.113074 0.490455   
75% 0.130217 0.281690 0.197880 0.570117   
max 1.000000 1.000000 1.000000 1.000000   
  
 pH sulphates alcohol quality   
count 1599.000000 1599.000000 1599.000000 1599.000000   
mean 0.449695 0.196496 0.311228 5.636023   
std 0.121564 0.101501 0.163949 0.807569   
min 0.000000 0.000000 0.000000 3.000000   
25% 0.370079 0.131737 0.169231 5.000000   
50% 0.448819 0.173653 0.276923 6.000000   
75% 0.519685 0.239521 0.415385 6.000000   
max 1.000000 1.000000 1.000000 8.000000   
  
Z-Score Normalized Data Statistics:  
 fixed acidity volatile acidity citric acid residual sugar \  
count 1.599000e+03 1.599000e+03 1.599000e+03 1.599000e+03   
mean 3.554936e-16 1.733031e-16 -8.887339e-17 -1.244227e-16   
std 1.000313e+00 1.000313e+00 1.000313e+00 1.000313e+00   
min -2.137045e+00 -2.278280e+00 -1.391472e+00 -1.162696e+00   
25% -7.007187e-01 -7.699311e-01 -9.293181e-01 -4.532184e-01   
50% -2.410944e-01 -4.368911e-02 -5.636026e-02 -2.403750e-01   
75% 5.057952e-01 6.266881e-01 7.652471e-01 4.341614e-02   
max 4.355149e+00 5.877976e+00 3.743574e+00 9.195681e+00   
  
 chlorides free sulfur dioxide total sulfur dioxide density \  
count 1.599000e+03 1.599000e+03 1.599000e+03 1.599000e+03   
mean 3.732682e-16 -6.221137e-17 4.443669e-17 -3.473172e-14   
std 1.000313e+00 1.000313e+00 1.000313e+00 1.000313e+00   
min -1.603945e+00 -1.422500e+00 -1.230584e+00 -3.538731e+00   
25% -3.712290e-01 -8.487156e-01 -7.440403e-01 -6.077557e-01   
50% -1.799455e-01 -1.793002e-01 -2.574968e-01 1.760083e-03   
75% 5.384542e-02 4.901152e-01 4.723184e-01 5.768249e-01   
max 1.112703e+01 5.367284e+00 7.375154e+00 3.680055e+00   
  
 pH sulphates alcohol quality   
count 1.599000e+03 1.599000e+03 1.599000e+03 1599.000000   
mean 2.861723e-15 6.754377e-16 1.066481e-16 5.636023   
std 1.000313e+00 1.000313e+00 1.000313e+00 0.807569   
min -3.700401e+00 -1.936507e+00 -1.898919e+00 3.000000   
25% -6.551405e-01 -6.382196e-01 -8.663789e-01 5.000000   
50% -7.212705e-03 -2.251281e-01 -2.093081e-01 6.000000   
75% 5.759223e-01 4.240158e-01 6.354971e-01 6.000000   
max 4.528282e+00 7.918677e+00 4.202453e+00 8.000000   
  
Decimal Scaled Data Statistics:  
 fixed acidity volatile acidity citric acid residual sugar \  
count 1599.000000 1599.000000 1599.000000 1599.000000   
mean 0.083196 0.052782 0.027098 0.025388   
std 0.017411 0.017906 0.019480 0.014099   
min 0.046000 0.012000 0.000000 0.009000   
25% 0.071000 0.039000 0.009000 0.019000   
50% 0.079000 0.052000 0.026000 0.022000   
75% 0.092000 0.064000 0.042000 0.026000   
max 0.159000 0.158000 0.100000 0.155000   
  
 chlorides free sulfur dioxide total sulfur dioxide density \  
count 1599.000000 1599.000000 1599.000000 1599.000000   
mean 0.008747 0.158749 0.046468 0.099675   
std 0.004707 0.104602 0.032895 0.000189   
min 0.001200 0.010000 0.006000 0.099007   
25% 0.007000 0.070000 0.022000 0.099560   
50% 0.007900 0.140000 0.038000 0.099675   
75% 0.009000 0.210000 0.062000 0.099783   
max 0.061100 0.720000 0.289000 0.100369   
  
 pH sulphates alcohol quality   
count 1599.000000 1599.000000 1599.000000 1599.000000   
mean 0.331111 0.065815 0.104230 5.636023   
std 0.015439 0.016951 0.010657 0.807569   
min 0.274000 0.033000 0.084000 3.000000   
25% 0.321000 0.055000 0.095000 5.000000   
50% 0.331000 0.062000 0.102000 6.000000   
75% 0.340000 0.073000 0.111000 6.000000   
max 0.401000 0.200000 0.149000 8.000000

**Aptness of Proximity Metrics**

**Without Normalization:**

Features with larger ranges dominate distance calculations, potentially biasing models.

**With Normalization:**

Ensures each feature contributes equally, leading to more balanced and meaningful distance metrics.

**Implications:**

**K-Nearest Neighbors (KNN):** Performance significantly improves with normalization.

**Clustering Algorithms (e.g., K-Means):** Better cluster formation due to balanced feature contributions.

**Discretization Effects**

**Binning (Equal Width):**

Simple to implement but may not account for data distribution.

**Histogram-Based Binning:**

Accounts for data distribution, providing more balanced bins.

**Implications:**

**Decision Trees:** Discretization can lead to simpler tree structures.

**Reduced Information:** May lose some information due to grouping, potentially impacting model performance.

**Conclusion**

Normalization and discretization are crucial preprocessing steps in machine learning workflows. They address issues related to varying feature scales and continuous data representation, respectively. Here's a summary of the key takeaways from the experiment:

**Normalization:**

Essential for algorithms sensitive to feature scales.

Techniques like Min-Max and Z-score normalization adjust data scales effectively.

Z-score normalization is particularly useful for standardizing data distributions.

**Discretization:**

Transforms continuous data into categorical bins, suitable for certain algorithms.

Binning methods can simplify data but may lead to information loss.

Histogram-based binning provides a more balanced approach by considering data distribution.

**Impact on Proximity Metrics:**

Normalization ensures equitable feature contribution in distance calculations.

Enhances the performance of distance-based algorithms like KNN and clustering.

**Statistical Parameters:**

Normalization alters central tendency and dispersion, making data suitable for various algorithms.

Discretization affects the representation of data, converting continuous attributes into categorical ones.

**Final Recommendation:**

**Before Applying Normalization:**

Assess the algorithms to be used and their sensitivity to feature scales.

Choose normalization techniques that align with the model requirements.

**Before Applying Discretization:**

Determine if the model benefits from categorical representation.

Select appropriate binning methods based on data distribution and model needs.

In summary, thoughtful application of normalization and discretization enhances model performance, ensures balanced feature contributions, and aligns data representation with algorithmic requirements.

# Data Mining LAB : Experiment 6

Submitted By:

Name: Debatreya Das  
Roll No. 12212070  
CS A4  
Data Mining LAB

Part A

Objective: To compute maximal frequent itemset.

Compute candidate 3-itemsets from frequent 2-itemsets using join C3 = L2 x L2. (Han’s book example)

Generalize the algorithm for generating candidate Ci+1 itemsets from frequent Li itemsets Ci+1 = Li x Li

Generating Candidate 3-itemsets (C3) from Frequent 2-itemsets (L2)

from itertools import combinations  
  
# Function to generate candidate 3-itemsets from frequent 2-itemsets  
def generate\_candidate\_3\_itemsets(L2):  
 C3 = set() # Store candidates in a set to avoid duplicates  
   
 # Join step: combine two frequent 2-itemsets to form a candidate 3-itemset  
 for itemset1 in L2:  
 for itemset2 in L2:  
 # Join if first two items match (i.e., {a, b} U {a, c} -> {a, b, c})  
 if len(itemset1.intersection(itemset2)) == 1:  
 candidate = itemset1.union(itemset2)  
 if len(candidate) == 3:  
 C3.add(frozenset(candidate)) # Frozenset to make itemsets hashable  
   
 return C3

Generalizing for 𝐶𝑖+1 from 𝐿𝑖

def generate\_candidate\_itemsets(Li, k):  
 Ci\_plus\_1 = set()  
   
 # Join step: combine k-itemsets that differ by only one item  
 for itemset1 in Li:  
 for itemset2 in Li:  
 # Join if the first (k-1) items are the same  
 if len(itemset1.intersection(itemset2)) == k-1:  
 candidate = itemset1.union(itemset2)  
 if len(candidate) == k + 1:  
 Ci\_plus\_1.add(frozenset(candidate))  
   
 return Ci\_plus\_1

Part B

Objective: To develop prune operation using apriory property.

Prune unnecessary 3-itemsets from the set of generated 3-itemsets C3 to make C3 to set of frequent 3-itemsets L3. (Han book example)

Generalize the algorithm for pruning unnecessary i-itemsets from the set of generated i-itemsets Ci to make Ci to set of frequent i-itemsets Li.

3 Itemset Prunning

def prune\_3\_itemsets(C3, L2):  
 pruned\_C3 = set()  
   
 # For each candidate 3-itemset  
 for candidate in C3:  
 valid = True  
 # Generate all 2-itemset subsets (since we're pruning 3-itemsets)  
 for subset in combinations(candidate, 2):  
 # If any 2-itemset subset is not in L2, prune the candidate  
 if frozenset(subset) not in L2:  
 valid = False  
 break  
 # If all 2-itemset subsets are frequent, keep the 3-itemset  
 if valid:  
 pruned\_C3.add(candidate)  
   
 return pruned\_C3

Prunning infrequent itemset for Ci+1

def prune\_candidates(Ci\_plus\_1, Li):  
 pruned\_Ci\_plus\_1 = set()  
   
 for candidate in Ci\_plus\_1:  
 # Generate all k-sized subsets of the candidate  
 valid = True  
 for subset in combinations(candidate, len(candidate)-1):  
 if frozenset(subset) not in Li:  
 valid = False  
 break  
 if valid:  
 pruned\_Ci\_plus\_1.add(candidate)  
   
 return pruned\_Ci\_plus\_1

Part C

Write Apriori algorithm using the above join and prune procedures.

from itertools import chain, combinations  
  
# Helper function to generate all candidate itemsets from a dataset  
def get\_itemsets\_from\_transactions(transactions, k):  
 itemsets = set()  
 for transaction in transactions:  
 for itemset in combinations(transaction, k):  
 itemsets.add(frozenset(itemset))  
 return itemsets  
  
# Helper function to calculate support of itemsets  
def calculate\_support(transactions, candidates):  
 support\_count = {itemset: 0 for itemset in candidates}  
 for transaction in transactions:  
 for candidate in candidates:  
 if candidate.issubset(transaction):  
 support\_count[candidate] += 1  
 return support\_count  
  
# Apriori algorithm  
def apriori(transactions, min\_support):  
 # Step 1: Generate frequent 1-itemsets (L1)  
 single\_items = chain.from\_iterable(transactions)  
 item\_count = {}  
 for item in single\_items:  
 item\_count[frozenset([item])] = item\_count.get(frozenset([item]), 0) + 1  
   
 # Filter 1-itemsets by min support  
 L1 = {itemset for itemset, count in item\_count.items() if count >= min\_support}  
 frequent\_itemsets = {1: L1}  
   
 k = 2  
 Li = L1  
 while Li:  
 # Step 2: Generate candidates Ci+1 from frequent Li itemsets  
 candidates = generate\_candidate\_itemsets(Li, k-1)  
   
 # Step 3: Calculate support for candidates  
 support\_count = calculate\_support(transactions, candidates)  
   
 # Step 4: Prune candidates whose support is less than min\_support  
 Li = {itemset for itemset, count in support\_count.items() if count >= min\_support}  
   
 if Li:  
 frequent\_itemsets[k] = Li  
 k += 1  
   
 return frequent\_itemsets  
  
# Example transactions (dataset)  
transactions = [  
 {1, 2, 3},  
 {1, 2, 4},  
 {2, 3, 4},  
 {1, 3, 4},  
 {1, 2, 3, 4}  
]  
  
# Minimum support threshold  
min\_support = 2  
  
# Run the Apriori algorithm  
frequent\_itemsets = apriori(transactions, min\_support)  
  
# Output the result  
for k, itemsets in frequent\_itemsets.items():  
 print(f"Frequent {k}-itemsets: {itemsets}")

Frequent 1-itemsets: {frozenset({3}), frozenset({2}), frozenset({1}), frozenset({4})}  
Frequent 2-itemsets: {frozenset({3, 4}), frozenset({1, 4}), frozenset({2, 3}), frozenset({1, 2}), frozenset({2, 4}), frozenset({1, 3})}  
Frequent 3-itemsets: {frozenset({1, 2, 3}), frozenset({2, 3, 4}), frozenset({1, 3, 4}), frozenset({1, 2, 4})}

Tests

Example 2 Itemset

L2 = [frozenset([1, 2]), frozenset([1, 3]), frozenset([2, 3]), frozenset([2, 4])]

Generate candidate 3-itemsets (C3)

C3 = generate\_candidate\_3\_itemsets(L2)  
print("Candidate 3-itemsets:", C3)

Candidate 3-itemsets: {frozenset({1, 2, 3}), frozenset({2, 3, 4}), frozenset({1, 2, 4})}

Generalized candidate generation for k+1 from k

L3 = generate\_candidate\_itemsets(L2, 2)  
print("Generalized candidate 3-itemsets:", L3)

Generalized candidate 3-itemsets: {frozenset({1, 2, 3}), frozenset({2, 3, 4}), frozenset({1, 2, 4})}

Itemset Prunning

pruned\_C3 = prune\_3\_itemsets(C3, L2)  
print("Pruned 3-itemsets:", pruned\_C3)

Pruned 3-itemsets: {frozenset({1, 2, 3})}

Generalized Pruning the candidates

pruned\_L3 = prune\_candidates(L3, L2)  
print("Pruned 3-itemsets:", pruned\_L3)

Pruned 3-itemsets: {frozenset({1, 2, 3})}

Data Mining Lab : Experiment 7

Submitted By

NAME: Debatreya Das

Roll-No.: 12212070

CS-A4

Data Mining Lab 7

Objective:

To compute maximal frequent itemset.

Part A

Compute item set of frequent items (1-itemsets) and their support counts from a given transactional dataset. Sort frequent itemsets and generate them in L order (descending order of support counts).

Importing Libraries

## IMPORTING LIBRARIES  
import pandas as pd  
from collections import defaultdict

Loading Transaction Data

# Load the CSV file containing transactions  
file\_path = 'fp\_growth\_transactions.csv' # Update with the correct path  
df\_transactions = pd.read\_csv(file\_path)  
df\_transactions

TID Items  
0 T1 bread,milk  
1 T2 bread,diaper,beer,eggs  
2 T3 milk,diaper,beer,cola  
3 T4 bread,milk,diaper,beer  
4 T5 bread,milk,cola

Function to compute support of 1-itemsets compute\_1\_itemsets

# Function to compute support counts of 1-itemsets  
def compute\_1\_itemsets(transactions):  
 item\_support\_count = defaultdict(int)  
 for transaction in transactions:  
 items = transaction.split(',')  
 for item in items:  
 item\_support\_count[item] += 1  
 return dict(item\_support\_count)

Calculating frequent 1-itemsets

# Extract transactions from the CSV data  
transaction\_list = df\_transactions['Items'].tolist()  
  
# Compute 1-itemsets with support counts  
item\_support\_counts = compute\_1\_itemsets(transaction\_list)  
  
# Sorting the 1-itemsets by support counts in descending order  
sorted\_item\_support\_counts = sorted(item\_support\_counts.items(), key=lambda x: x[1], reverse=True)  
  
# Display the sorted frequent 1-itemsets  
sorted\_item\_support\_counts

[('bread', 4),  
 ('milk', 4),  
 ('diaper', 3),  
 ('beer', 3),  
 ('cola', 2),  
 ('eggs', 1)]

Part B

Sort items in the transactions of the dataset in L-order (descending order of support counts).

Function to sort items in each transaction by L-order

# Helper function to sort items in each transaction by L-order (support counts)  
def sort\_transactions\_by\_l\_order(transactions, support\_counts):  
 sorted\_transactions = []  
 for transaction in transactions:  
 items = transaction.split(',')  
 # Sort items in transaction based on the L-order (support counts)  
 sorted\_items = sorted(items, key=lambda item: support\_counts[item], reverse=True)  
 sorted\_transactions.append(sorted\_items)  
 return sorted\_transactions

Calculating sorted transaction by L order

# Sorting transactions by L-order  
sorted\_transactions = sort\_transactions\_by\_l\_order(transaction\_list, dict(sorted\_item\_support\_counts))  
  
# Display the sorted transactions  
sorted\_transactions

[['bread', 'milk'],  
 ['bread', 'diaper', 'beer', 'eggs'],  
 ['milk', 'diaper', 'beer', 'cola'],  
 ['bread', 'milk', 'diaper', 'beer'],  
 ['bread', 'milk', 'cola']]

Part C

Construct FP tree using the Han’s book example. Display FP tree using appropriate notation/representation.

Class: FPTreeNode

# FP-Tree Node structure  
class FPTreeNode:  
 def \_\_init\_\_(self, item\_name, count, parent):  
 self.item\_name = item\_name  
 self.count = count  
 self.parent = parent  
 self.children = {}  
 self.link = None # Link to next node of the same item  
  
 def increment(self, count):  
 """Increment the count of the node."""  
 self.count += count  
  
# FP-Tree structure  
class FPTree:  
 def \_\_init\_\_(self):  
 self.root = FPTreeNode(None, 1, None) # Root node with no item  
 self.header\_table = {}  
  
 def update\_header\_table(self, node, item):  
 """Update header table to point to nodes of the same item."""  
 if item in self.header\_table:  
 current\_node = self.header\_table[item]  
 while current\_node.link is not None:  
 current\_node = current\_node.link  
 current\_node.link = node  
 else:  
 self.header\_table[item] = node  
  
 def insert\_transaction(self, transaction):  
 """Insert a sorted transaction into the FP-Tree."""  
 current\_node = self.root  
 for item in transaction:  
 if item in current\_node.children:  
 current\_node.children[item].increment(1)  
 else:  
 new\_node = FPTreeNode(item, 1, current\_node)  
 current\_node.children[item] = new\_node  
 self.update\_header\_table(new\_node, item)  
 current\_node = current\_node.children[item]

Function to construct the FP-Tree from sorted transactions

# Construct the FP-Tree from sorted transactions  
def construct\_fp\_tree(transactions):  
 tree = FPTree()  
 for transaction in transactions:  
 tree.insert\_transaction(transaction)  
 return tree

Build the FP Growth Tree

# Build the FP-Tree from sorted transactions  
fp\_tree = construct\_fp\_tree(sorted\_transactions)

Printing the tree

def print\_fp\_tree(node, indent=0):  
 print(' ' \* indent + f'{node.item\_name}: {node.count}')  
 for child in node.children.values():  
 print\_fp\_tree(child, indent + 1)  
  
# Print the FP-Tree structure  
print\_fp\_tree(fp\_tree.root)

None: 1  
 bread: 4  
 milk: 3  
 diaper: 1  
 beer: 1  
 cola: 1  
 diaper: 1  
 beer: 1  
 eggs: 1  
 milk: 1  
 diaper: 1  
 beer: 1  
 cola: 1

Part D

Using FP tree, construct pattern bases and conditional FP trees.

# Function to extract the conditional pattern base for an item  
def find\_prefix\_paths(base\_item, header\_table):  
 cond\_pattern\_base = []  
 node = header\_table[base\_item]  
 while node is not None:  
 prefix\_path = []  
 current\_node = node  
 while current\_node.parent.item\_name is not None:  
 prefix\_path.append(current\_node.parent.item\_name)  
 current\_node = current\_node.parent  
 if len(prefix\_path) > 0:  
 cond\_pattern\_base.append((prefix\_path, node.count))  
 node = node.link  
 return cond\_pattern\_base

Finding Pattern Bases for all Items

Items = ["bread", "milk", "diaper", "beer", "eggs", "cola"]  
  
PatternBases = {  
 "bread": find\_prefix\_paths("bread", fp\_tree.header\_table),  
 "milk": find\_prefix\_paths("milk", fp\_tree.header\_table),  
 "diaper": find\_prefix\_paths("diaper", fp\_tree.header\_table),  
 "beer": find\_prefix\_paths("beer", fp\_tree.header\_table),  
 "eggs": find\_prefix\_paths("eggs", fp\_tree.header\_table),  
 "cola": find\_prefix\_paths("cola", fp\_tree.header\_table)  
}  
  
for i in Items:  
 print(f"PatternBases for {i}: ", PatternBases[i])

PatternBases for bread: []  
PatternBases for milk: [(['bread'], 3)]  
PatternBases for diaper: [(['bread'], 1), (['milk'], 1), (['milk', 'bread'], 1)]  
PatternBases for beer: [(['diaper', 'bread'], 1), (['diaper', 'milk'], 1), (['diaper', 'milk', 'bread'], 1)]  
PatternBases for eggs: [(['beer', 'diaper', 'bread'], 1)]  
PatternBases for cola: [(['beer', 'diaper', 'milk'], 1), (['milk', 'bread'], 1)]

Part E

Generate frequent patterns.

Helper Functions

from collections import defaultdict  
  
# Function to filter items by support threshold  
def filter\_items\_by\_support(transactions, min\_support):  
 item\_count = defaultdict(int)  
 for transaction in transactions:  
 for item in transaction:  
 item\_count[item] += 1  
  
 # Filter out items that don't meet the minimum support  
 filtered\_items = {item for item, count in item\_count.items() if count >= min\_support}  
 return filtered\_items, item\_count

Function to generate the conditional fp-trees

# Function to create a conditional FP-Tree  
def construct\_conditional\_fp\_tree(base\_item, cond\_pattern\_base, min\_support):  
 tree = FPTree()  
 for prefix\_path, count in cond\_pattern\_base:  
 # Filter out items that don't meet min support in prefix path  
 filtered\_prefix\_path = [item for item in prefix\_path if item in tree.header\_table]  
 filtered\_prefix\_path.sort(key=lambda item: tree.header\_table[item].count, reverse=True)  
 for \_ in range(count):  
 tree.insert\_transaction(filtered\_prefix\_path)  
 return tree  
  
# Function to mine frequent patterns using conditional FP-trees  
def mine\_fp\_tree(tree, min\_support, prefix, frequent\_patterns):  
 # Sort items in the header table by frequency  
 sorted\_items = sorted(tree.header\_table.items(), key=lambda x: x[1].count)  
  
 for base\_item, node in sorted\_items:  
 new\_prefix = prefix.copy()  
   
 # Ensure no duplicate items in the new prefix  
 if base\_item not in new\_prefix:  
 new\_prefix.append(base\_item)  
  
 # Calculate total support for the current pattern  
 total\_support = 0  
 current\_node = node  
 while current\_node is not None:  
 total\_support += current\_node.count  
 current\_node = current\_node.link  
  
 # Add the frequent pattern (prefix + item) to the result  
 if total\_support >= min\_support:  
 frequent\_patterns.append((new\_prefix, total\_support))  
  
 # Find conditional pattern base  
 cond\_pattern\_base = find\_prefix\_paths(base\_item, tree.header\_table)  
  
 # Construct conditional FP-Tree for current item  
 cond\_tree = FPTree()  
 for path, count in cond\_pattern\_base:  
 cond\_tree.insert\_transaction(path \* count) # Insert with counts  
  
 # Recursively mine the conditional FP-tree  
 if cond\_tree.root.children:  
 mine\_fp\_tree(cond\_tree, min\_support, new\_prefix, frequent\_patterns)

Generate Frequent Patterns

# Main driver function  
def generate\_frequent\_patterns(transactions, min\_support):  
 # Step 1: Filter items by min\_support  
 filtered\_items, item\_count = filter\_items\_by\_support(transactions, min\_support)  
  
 # Step 2: Filtered transactions  
 filtered\_transactions = [[item for item in transaction if item in filtered\_items]  
 for transaction in transactions]  
  
 # Step 3: Construct the initial FP-Tree  
 tree = construct\_fp\_tree(filtered\_transactions)  
  
 # Step 4: Recursively mine the FP-Tree  
 frequent\_patterns = []  
 mine\_fp\_tree(tree, min\_support, [], frequent\_patterns)  
  
 return frequent\_patterns

# DRIVER CODE  
transactions = [items.split(',') for items in df\_transactions['Items']]  
min\_support = 2  
frequent\_patterns = generate\_frequent\_patterns(transactions, min\_support)  
  
  
# Display all frequent patterns  
for pattern, count in frequent\_patterns:  
 print(f"Pattern: {pattern}, Support: {count}")

Pattern: ['diaper'], Support: 3  
Pattern: ['diaper', 'bread'], Support: 2  
Pattern: ['diaper', 'milk'], Support: 2  
Pattern: ['beer'], Support: 3  
Pattern: ['beer', 'bread'], Support: 2  
Pattern: ['beer', 'bread', 'diaper'], Support: 2  
Pattern: ['beer', 'milk'], Support: 2  
Pattern: ['beer', 'milk', 'diaper'], Support: 2  
Pattern: ['beer', 'diaper'], Support: 3  
Pattern: ['cola'], Support: 2  
Pattern: ['cola', 'milk'], Support: 2  
Pattern: ['milk'], Support: 4  
Pattern: ['milk', 'bread'], Support: 3  
Pattern: ['bread'], Support: 4

# (LAB: 8) Hierarchical Clustering using Agglomerative Nesting (AGNES)

Name: Debatreya Das

Roll No. 12212070

CS A4

Data Mining Lab 8

Algorithm

Initialize the Proximity Matrix

Make each point a cluster

Loop

a. Merge the 2 closest cluster

b. Update the Proximity Matrix

Until only one cluster is left

Types of Agglomerative Clustering

(Based on How proximity between two clusters is calculated)

Min (Single-Link)

Max (Complete Link)

Average

Ward

CODE: Hierarchical Clustering

Importing Required Libraries

import matplotlib.pyplot as plt  
import pandas as pd  
%matplotlib inline  
import numpy as np

Loading Data

customerData = pd.read\_csv('./shopping.csv')  
print("SHAPE: ", customerData.shape)  
print("HEAD: \n", customerData.head())

SHAPE: (200, 5)  
HEAD:   
 CustomerID Gender Age Annual Income (k$) Spending Score (1-100)  
0 1 Male 19 15 39  
1 2 Male 21 15 81  
2 3 Female 20 16 6  
3 4 Female 23 16 77  
4 5 Female 31 17 40

Selecting 2 Attributes from the data for CLustering

data = customerData.iloc[:, 3:5].values  
data

array([[ 15, 39],  
 [ 15, 81],  
 [ 16, 6],  
 [ 16, 77],  
 [ 17, 40],  
 [ 17, 76],  
 [ 18, 6],  
 [ 18, 94],  
 [ 19, 3],  
 [ 19, 72],  
 [ 19, 14],  
 [ 19, 99],  
 [ 20, 15],  
 [ 20, 77],  
 [ 20, 13],  
 [ 20, 79],  
 [ 21, 35],  
 [ 21, 66],  
 [ 23, 29],  
 [ 23, 98],  
 [ 24, 35],  
 [ 24, 73],  
 [ 25, 5],  
 [ 25, 73],  
 [ 28, 14],  
 [ 28, 82],  
 [ 28, 32],  
 [ 28, 61],  
 [ 29, 31],  
 [ 29, 87],  
 [ 30, 4],  
 [ 30, 73],  
 [ 33, 4],  
 [ 33, 92],  
 [ 33, 14],  
 [ 33, 81],  
 [ 34, 17],  
 [ 34, 73],  
 [ 37, 26],  
 [ 37, 75],  
 [ 38, 35],  
 [ 38, 92],  
 [ 39, 36],  
 [ 39, 61],  
 [ 39, 28],  
 [ 39, 65],  
 [ 40, 55],  
 [ 40, 47],  
 [ 40, 42],  
 [ 40, 42],  
 [ 42, 52],  
 [ 42, 60],  
 [ 43, 54],  
 [ 43, 60],  
 [ 43, 45],  
 [ 43, 41],  
 [ 44, 50],  
 [ 44, 46],  
 [ 46, 51],  
 [ 46, 46],  
 [ 46, 56],  
 [ 46, 55],  
 [ 47, 52],  
 [ 47, 59],  
 [ 48, 51],  
 [ 48, 59],  
 [ 48, 50],  
 [ 48, 48],  
 [ 48, 59],  
 [ 48, 47],  
 [ 49, 55],  
 [ 49, 42],  
 [ 50, 49],  
 [ 50, 56],  
 [ 54, 47],  
 [ 54, 54],  
 [ 54, 53],  
 [ 54, 48],  
 [ 54, 52],  
 [ 54, 42],  
 [ 54, 51],  
 [ 54, 55],  
 [ 54, 41],  
 [ 54, 44],  
 [ 54, 57],  
 [ 54, 46],  
 [ 57, 58],  
 [ 57, 55],  
 [ 58, 60],  
 [ 58, 46],  
 [ 59, 55],  
 [ 59, 41],  
 [ 60, 49],  
 [ 60, 40],  
 [ 60, 42],  
 [ 60, 52],  
 [ 60, 47],  
 [ 60, 50],  
 [ 61, 42],  
 [ 61, 49],  
 [ 62, 41],  
 [ 62, 48],  
 [ 62, 59],  
 [ 62, 55],  
 [ 62, 56],  
 [ 62, 42],  
 [ 63, 50],  
 [ 63, 46],  
 [ 63, 43],  
 [ 63, 48],  
 [ 63, 52],  
 [ 63, 54],  
 [ 64, 42],  
 [ 64, 46],  
 [ 65, 48],  
 [ 65, 50],  
 [ 65, 43],  
 [ 65, 59],  
 [ 67, 43],  
 [ 67, 57],  
 [ 67, 56],  
 [ 67, 40],  
 [ 69, 58],  
 [ 69, 91],  
 [ 70, 29],  
 [ 70, 77],  
 [ 71, 35],  
 [ 71, 95],  
 [ 71, 11],  
 [ 71, 75],  
 [ 71, 9],  
 [ 71, 75],  
 [ 72, 34],  
 [ 72, 71],  
 [ 73, 5],  
 [ 73, 88],  
 [ 73, 7],  
 [ 73, 73],  
 [ 74, 10],  
 [ 74, 72],  
 [ 75, 5],  
 [ 75, 93],  
 [ 76, 40],  
 [ 76, 87],  
 [ 77, 12],  
 [ 77, 97],  
 [ 77, 36],  
 [ 77, 74],  
 [ 78, 22],  
 [ 78, 90],  
 [ 78, 17],  
 [ 78, 88],  
 [ 78, 20],  
 [ 78, 76],  
 [ 78, 16],  
 [ 78, 89],  
 [ 78, 1],  
 [ 78, 78],  
 [ 78, 1],  
 [ 78, 73],  
 [ 79, 35],  
 [ 79, 83],  
 [ 81, 5],  
 [ 81, 93],  
 [ 85, 26],  
 [ 85, 75],  
 [ 86, 20],  
 [ 86, 95],  
 [ 87, 27],  
 [ 87, 63],  
 [ 87, 13],  
 [ 87, 75],  
 [ 87, 10],  
 [ 87, 92],  
 [ 88, 13],  
 [ 88, 86],  
 [ 88, 15],  
 [ 88, 69],  
 [ 93, 14],  
 [ 93, 90],  
 [ 97, 32],  
 [ 97, 86],  
 [ 98, 15],  
 [ 98, 88],  
 [ 99, 39],  
 [ 99, 97],  
 [101, 24],  
 [101, 68],  
 [103, 17],  
 [103, 85],  
 [103, 23],  
 [103, 69],  
 [113, 8],  
 [113, 91],  
 [120, 16],  
 [120, 79],  
 [126, 28],  
 [126, 74],  
 [137, 18],  
 [137, 83]], dtype=int64)

Ward: AGNES Hierarchical Clustering (using sklearn)

import scipy.cluster.hierarchy as shc  
  
plt.figure(figsize=(10, 7))  
plt.title("Customer Dendograms")  
dend = shc.dendrogram(shc.linkage(data, method='ward'))

We get the number of clusters = 5 from the above dendogram, now apply AGNES to create 5 clusters.

from sklearn.cluster import AgglomerativeClustering  
  
cluster = AgglomerativeClustering(n\_clusters=5, metric='euclidean', linkage='ward')  
labels\_ = cluster.fit\_predict(data)

labels\_

array([4, 3, 4, 3, 4, 3, 4, 3, 4, 3, 4, 3, 4, 3, 4, 3, 4, 3, 4, 3, 4, 3,  
 4, 3, 4, 3, 4, 3, 4, 3, 4, 3, 4, 3, 4, 3, 4, 3, 4, 3, 4, 3, 4, 1,  
 4, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,  
 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,  
 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,  
 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 1, 2, 1, 2, 0, 2, 0, 2,  
 1, 2, 0, 2, 0, 2, 0, 2, 0, 2, 1, 2, 0, 2, 1, 2, 0, 2, 0, 2, 0, 2,  
 0, 2, 0, 2, 0, 2, 1, 2, 0, 2, 0, 2, 0, 2, 0, 2, 0, 2, 0, 2, 0, 2,  
 0, 2, 0, 2, 0, 2, 0, 2, 0, 2, 0, 2, 0, 2, 0, 2, 0, 2, 0, 2, 0, 2,  
 0, 2], dtype=int64)

Plotting the Data

plt.figure(figsize=(10, 7))  
plt.scatter(data[:,0], data[:,1], c=cluster.labels\_, cmap='rainbow')

<matplotlib.collections.PathCollection at 0x1b636125290>

# (LAB: 9) K-Means Clustering

Name: Debatreya Das

Roll No. 12212070

CS A4

Data Mining Lab 9

Dataset used

K-Means clustering is an unsupervised machine learning algorithm used to partition data into distinct groups or clusters based on similarity. The algorithm aims to minimize the variance within each cluster, creating groups where data points are more similar to each other than to those in other clusters.

The K-Means algorithm involves these main steps:

**Initialization**: Select k initial centroids randomly, where k is the number of clusters chosen in advance.

**Assignment**: Each data point is assigned to the nearest centroid, forming k clusters.

**Update**: Calculate the mean of all points within each cluster to update the centroids.

**Repeat**: Steps 2 and 3 are repeated until the centroids stabilize or change only minimally (convergence).

K-Means is popular for its simplicity and effectiveness in tasks like market segmentation, image compression, and anomaly detection. However, it has some limitations, such as sensitivity to the initial placement of centroids and difficulties with clusters of non-spherical shapes or varying densities.

CODE

Importing Necessary Libraries

import pandas as pd  
import numpy as np  
import matplotlib.pyplot as plt

Loading Dataset

# Load the dataset  
data = pd.read\_csv("./assets/mall\_customers.csv")  
data.head()

CustomerID Gender Age Annual Income (k$) Spending Score (1-100)  
0 1 Male 19 15 39  
1 2 Male 21 15 81  
2 3 Female 20 16 6  
3 4 Female 23 16 77  
4 5 Female 31 17 40

Data Preprocessing

# Check for missing values  
print("Missing values:\n", data.isnull().sum())  
  
# Select the 'Annual Income (k$)' and 'Spending Score (1-100)' columns  
X = data[['Annual Income (k$)', 'Spending Score (1-100)']].values

Missing values:  
 CustomerID 0  
Gender 0  
Age 0  
Annual Income (k$) 0  
Spending Score (1-100) 0  
dtype: int64

KMeans

Define Helper Functions for K-Means

# Function to calculate the Euclidean distance between two points  
def euclidean\_distance(point1, point2):  
 return np.sqrt(np.sum((point1 - point2) \*\* 2))

Implement the K-Means Algorithm

# Initialize centroids randomly  
def initialize\_centroids(X, k):  
 np.random.seed(0)  
 random\_indices = np.random.permutation(X.shape[0])  
 centroids = X[random\_indices[:k]]  
 return centroids  
  
# Assign each data point to the nearest centroid  
def assign\_clusters(X, centroids):  
 clusters = []  
 for point in X:  
 distances = [euclidean\_distance(point, centroid) for centroid in centroids]  
 closest\_centroid = np.argmin(distances)  
 clusters.append(closest\_centroid)  
 return np.array(clusters)  
  
# Update the centroids by calculating the mean of all points in each cluster  
def update\_centroids(X, clusters, k):  
 new\_centroids = []  
 for i in range(k):  
 cluster\_points = X[clusters == i]  
 if len(cluster\_points) > 0:  
 new\_centroid = cluster\_points.mean(axis=0)  
 else:  
 new\_centroid = X[np.random.choice(X.shape[0])]  
 new\_centroids.append(new\_centroid)  
 return np.array(new\_centroids)  
  
# Full K-means algorithm implementation  
def k\_means(X, k, max\_iterations=100, tolerance=1e-4):  
 centroids = initialize\_centroids(X, k)  
 for \_ in range(max\_iterations):  
 clusters = assign\_clusters(X, centroids)  
 new\_centroids = update\_centroids(X, clusters, k)  
 diff = np.linalg.norm(new\_centroids - centroids)  
 if diff < tolerance:  
 break  
 centroids = new\_centroids  
 return clusters, centroids

Run K-Means and Visualize the Clusters

# Set number of clusters and run K-means  
k = 5  
clusters, centroids = k\_means(X, k)  
  
# Plotting the clusters  
plt.figure(figsize=(10, 6))  
colors = ['r', 'g', 'b', 'c', 'm']  
for i in range(k):  
 cluster\_points = X[clusters == i]  
 plt.scatter(cluster\_points[:, 0], cluster\_points[:, 1], c=colors[i], label=f'Cluster {i+1}')  
# Plot centroids  
plt.scatter(centroids[:, 0], centroids[:, 1], s=300, c='yellow', marker='\*', label='Centroids')  
plt.xlabel('Annual Income (k$)')  
plt.ylabel('Spending Score (1-100)')  
plt.legend()  
plt.title("K-Means Clustering of Mall Customers")  
plt.show()

Calculate WCSS for Model Evaluation

To evaluate the clustering performance, we'll calculate the Within-Cluster Sum of Squares (WCSS). This helps assess the compactness of clusters, where lower values indicate better clustering.

# Function to calculate WCSS (inertia)  
def calculate\_wcss(X, clusters, centroids):  
 wcss = 0  
 for i, centroid in enumerate(centroids):  
 cluster\_points = X[clusters == i]  
 wcss += np.sum((cluster\_points - centroid) \*\* 2)  
 return wcss  
  
# Calculate WCSS for the trained model  
wcss = calculate\_wcss(X, clusters, centroids)  
print(f"Within-Cluster Sum of Squares (WCSS): {wcss}")

Within-Cluster Sum of Squares (WCSS): 44448.45544793371

Elbow Method for Optimal k

The elbow method helps find the ideal number of clusters by plotting WCSS across different k values and identifying the "elbow" point.

# Testing WCSS for different values of k  
wcss\_values = []  
k\_values = range(1, 11)  
  
for k in k\_values:  
 clusters, centroids = k\_means(X, k)  
 wcss = calculate\_wcss(X, clusters, centroids)  
 wcss\_values.append(wcss)  
  
# Plotting the WCSS values  
plt.figure(figsize=(8, 5))  
plt.plot(k\_values, wcss\_values, marker='o', linestyle='--', color='b')  
plt.xlabel('Number of Clusters (k)')  
plt.ylabel('Within-Cluster Sum of Squares (WCSS)')  
plt.title('Elbow Method for Optimal k')  
plt.show()

Test on Unseen Data

# Simulate unseen data (you can also load real unseen data if available)  
unseen\_data = np.array([  
 [40, 60], # Point with moderate income and spending score  
 [70, 90], # Point with high income and high spending score  
 [20, 30], # Point with low income and low spending score  
])  
  
# Function to predict cluster for each new point based on trained centroids  
def predict\_clusters(unseen\_data, centroids):  
 predictions = []  
 for point in unseen\_data:  
 distances = [euclidean\_distance(point, centroid) for centroid in centroids]  
 closest\_centroid = np.argmin(distances)  
 predictions.append(closest\_centroid)  
 return np.array(predictions)  
  
# Use centroids from the trained model  
predicted\_clusters = predict\_clusters(unseen\_data, centroids)  
print("Unseen Data Points:\n", unseen\_data)  
print("Predicted Clusters for Unseen Data:", predicted\_clusters)

Unseen Data Points:  
 [[40 60]  
 [70 90]  
 [20 30]]  
Predicted Clusters for Unseen Data: [3 4 0]

Visualize Unseen Data with Original Clusters

# Ensure that the necessary variables are defined  
k = 5 # Number of clusters  
# Plot the original clusters and centroids  
plt.figure(figsize=(10, 6))  
colors = ['r', 'g', 'b', 'c', 'm']  
for i in range(k):  
 cluster\_points = X[clusters == i]  
 plt.scatter(cluster\_points[:, 0], cluster\_points[:, 1], c=colors[i], label=f'Cluster {i+1}')  
# Plot centroids  
plt.scatter(centroids[:, 0], centroids[:, 1], s=300, c='yellow', marker='\*', label='Centroids')  
  
# Plot unseen data points with a different marker  
plt.scatter(unseen\_data[:, 0], unseen\_data[:, 1], s=150, c='black', marker='x', label='Unseen Data')  
plt.xlabel('Annual Income (k$)')  
plt.ylabel('Spending Score (1-100)')  
plt.legend()  
plt.title("K-Means Clustering with Unseen Data")  
plt.show()

# Lab 10 > Part A

Name: Debatreya Das

Roll No. 12212070

CS A4

Data Mining Lab 10

Create A data warehouse from different .csv files using PostgreSQL tool

Dataset

I have gathered 3 CSV files.

Now I will use Python to Put these Tables in Postgres and then Show Applying a Query

Import Libraries and Connect to PostgreSQL

import psycopg2  
import pandas as pd

# Connect with local Postgres  
conn = psycopg2.connect(  
 dbname="warehouse",   
 user="postgres",  
 password="postgres",   
 host="localhost"  
)  
cursor = conn.cursor()

Create Tables

# Drop tables if they exist  
cursor.execute("DROP TABLE IF EXISTS sales;")  
cursor.execute("DROP TABLE IF EXISTS customers;")  
cursor.execute("DROP TABLE IF EXISTS products;")  
  
# Create 'products' table  
cursor.execute("""  
CREATE TABLE products (  
 product\_id SERIAL PRIMARY KEY,  
 product\_name VARCHAR(255),  
 category VARCHAR(255),  
 price NUMERIC  
);  
""")  
  
# Create 'customers' table  
cursor.execute("""  
CREATE TABLE customers (  
 customer\_id SERIAL PRIMARY KEY,  
 first\_name VARCHAR(255),  
 last\_name VARCHAR(255),  
 city VARCHAR(255)  
);  
""")  
  
# Create 'sales' table  
cursor.execute("""  
CREATE TABLE sales (  
 sale\_id SERIAL PRIMARY KEY,  
 product\_id INT REFERENCES products(product\_id),  
 customer\_id INT REFERENCES customers(customer\_id),  
 sale\_date DATE,  
 quantity INT  
);  
""")  
  
conn.commit()

Load Data from CSV into Respective Files

# Load each .csv file into the respective table  
with open('./Data/products.csv', 'r') as f:  
 next(f) # Skip the header row  
 cursor.copy\_from(f, 'products', sep=',')  
  
with open('./Data/customer.csv', 'r') as f:  
 next(f)  
 cursor.copy\_from(f, 'customers', sep=',')  
  
with open('./Data/sales.csv', 'r') as f:  
 next(f)  
 cursor.copy\_from(f, 'sales', sep=',')  
  
  
conn.commit()

Now the Tables are ready.

Quering the Data Warehouse

# Example query to retrieve sales with customer details  
query = """  
SELECT s.sale\_id, s.sale\_date, p.product\_name, c.first\_name, c.last\_name, s.quantity, p.price  
FROM sales s  
JOIN products p ON s.product\_id = p.product\_id  
JOIN customers c ON s.customer\_id = c.customer\_id;  
"""  
  
# Run the query and display the results  
df = pd.read\_sql(query, conn)  
df.head()

C:\Users\debat\AppData\Local\Temp\ipykernel\_24728\2773805579.py:10: UserWarning: pandas only supports SQLAlchemy connectable (engine/connection) or database string URI or sqlite3 DBAPI2 connection. Other DBAPI2 objects are not tested. Please consider using SQLAlchemy.  
 df = pd.read\_sql(query, conn)

sale\_id sale\_date product\_name first\_name last\_name quantity price  
0 1 2024-01-01 Smartphone John Doe 2 500.0  
1 2 2024-01-03 Blender Jane Smith 1 80.0  
2 3 2024-01-05 Laptop Mike Brown 1 1200.0  
3 4 2024-01-08 Watch Sara Johnson 3 150.0  
4 5 2024-01-09 Shoes John Doe 1 60.0

Close connection

# Close cursor and connection  
cursor.close()  
conn.close()

# Lab 10 > Part B

Name: Debatreya Das

Roll No. 12212070

CS A4

Data Mining Lab 10

Create A data warehouse with given data using PostgreSQL tool and perform data extraction using SQL and OLAP.

Write the following queries:

Find the total sales.

Find total sales for each city.

Find total sales for each state.

Find total sales for each country.

Find sales of all cities of a specific state in a specific year.

Find year-wise total sales for each state.

Find year-wise total sales for each country.

Importing Required Libraries for Postgres

import psycopg2  
from psycopg2 import sql  
import pandas as pd

Create Connection

# Establish connection to PostgreSQL  
conn = psycopg2.connect(  
 dbname="datacube",  
 user="postgres",  
 password="postgres",  
 host="localhost",  
 port="5432"  
)  
cursor = conn.cursor()

Create the tables

create\_tables = [  
 """  
 CREATE TABLE IF NOT EXISTS Products (  
 P\_Id INTEGER PRIMARY KEY,  
 P\_name VARCHAR(100),  
 Category VARCHAR(50),  
 Price REAL  
 );  
 """,  
 """  
 CREATE TABLE IF NOT EXISTS Locations (  
 Loc\_Id INTEGER PRIMARY KEY,  
 City VARCHAR(50),  
 State VARCHAR(50),  
 Country VARCHAR(50)  
 );  
 """,  
 """  
 CREATE TABLE IF NOT EXISTS Times (  
 Time\_Id INTEGER PRIMARY KEY,  
 Date VARCHAR(10),  
 Week INTEGER,  
 Month INTEGER,  
 Quarter INTEGER,  
 Year INTEGER  
 );  
 """,  
 """  
 CREATE TABLE IF NOT EXISTS Sales (  
 Loc\_Id INTEGER REFERENCES Locations(Loc\_Id),  
 P\_Id INTEGER REFERENCES Products(P\_Id),  
 Time\_Id INTEGER REFERENCES Times(Time\_Id),  
 Sale REAL,  
 PRIMARY KEY (Loc\_Id, P\_Id, Time\_Id)  
 );  
 """  
]  
  
# Drop tables if they already exist   
cursor.execute("DROP TABLE IF EXISTS Sales;")  
cursor.execute("DROP TABLE IF EXISTS Products;")  
cursor.execute("DROP TABLE IF EXISTS Locations;")  
cursor.execute("DROP TABLE IF EXISTS Times;")  
  
# Execute each CREATE TABLE statement  
for command in create\_tables:  
 cursor.execute(command)  
  
# Commit changes and close connection  
conn.commit()  
  
print("Tables created successfully!")

Tables created successfully!

Inserting values in the Tables

# Insert data into Products table  
products\_data = [  
 (11, 'Lee Jeans', 'Apparel', 25),  
 (12, 'Zord', 'Toys', 18),  
 (13, 'Biro Pen', 'Stationery', 2)  
]  
cursor.executemany("INSERT INTO Products (P\_Id, P\_name, Category, Price) VALUES (%s, %s, %s, %s);", products\_data)  
  
# Insert data into Locations table  
locations\_data = [  
 (1, 'Madison', 'WI', 'USA'),  
 (2, 'Fresno', 'CA', 'USA'),  
 (5, 'Chennai', 'TN', 'India')  
]  
cursor.executemany("INSERT INTO Locations (Loc\_Id, City, State, Country) VALUES (%s, %s, %s, %s);", locations\_data)  
  
# Insert data into Times table  
times\_data = [  
 (1, '2023-01-01', 1, 1, 1, 2023),  
 (2, '2023-04-01', 14, 4, 2, 2023),  
 (3, '2023-07-01', 27, 7, 3, 2023)  
]  
cursor.executemany("INSERT INTO Times (Time\_Id, Date, Week, Month, Quarter, Year) VALUES (%s, %s, %s, %s, %s, %s);", times\_data)  
  
# Insert data into Sales table  
sales\_data = [  
 (1, 11, 1, 25), (1, 11, 2, 8), (1, 11, 3, 15),  
 (1, 12, 1, 30), (1, 12, 2, 20), (1, 12, 3, 50),  
 (1, 13, 1, 8), (1, 13, 2, 10), (1, 13, 3, 10),  
 (2, 11, 1, 35), (2, 11, 2, 22), (2, 11, 3, 10),  
 (2, 12, 1, 26), (2, 12, 2, 45), (2, 12, 3, 40),  
 (2, 13, 1, 20), (2, 13, 2, 20), (2, 13, 3, 40)  
]  
cursor.executemany("INSERT INTO Sales (Loc\_Id, P\_Id, Time\_Id, Sale) VALUES (%s, %s, %s, %s);", sales\_data)  
  
# Commit the transaction  
conn.commit()  
  
  
print("Data inserted successfully!")

Data inserted successfully!

SELECT \* FROM < TABLE > to check if its inserted properly

import pandas as pd  
  
# Select all rows from the Products table  
cursor.execute("SELECT \* FROM Products;")  
products = cursor.fetchall()  
  
print("Products Table")  
print(pd.DataFrame(products, columns=['P\_Id', 'P\_name', 'Category', 'Price']))  
  
# Select all rows from the Locations table  
cursor.execute("SELECT \* FROM Locations;")  
locations = cursor.fetchall()  
  
print("Locations Table")  
print(pd.DataFrame(locations, columns=['Loc\_Id', 'City', 'State', 'Country']))  
  
# Select all rows from the Times table  
cursor.execute("SELECT \* FROM Times;")  
times = cursor.fetchall()  
  
print("Times Table")  
print(pd.DataFrame(times, columns=['Time\_Id', 'Date', 'Week', 'Month', 'Quarter', 'Year']))  
  
# Select all rows from the Sales table  
cursor.execute("SELECT \* FROM Sales;")  
sales = cursor.fetchall()  
  
print("Sales Table")  
print(pd.DataFrame(sales, columns=['Loc\_Id', 'P\_Id', 'Time\_Id', 'Sale']))

Products Table  
 P\_Id P\_name Category Price  
0 11 Lee Jeans Apparel 25.0  
1 12 Zord Toys 18.0  
2 13 Biro Pen Stationery 2.0  
Locations Table  
 Loc\_Id City State Country  
0 1 Madison WI USA  
1 2 Fresno CA USA  
2 5 Chennai TN India  
Times Table  
 Time\_Id Date Week Month Quarter Year  
0 1 2023-01-01 1 1 1 2023  
1 2 2023-04-01 14 4 2 2023  
2 3 2023-07-01 27 7 3 2023  
Sales Table  
 Loc\_Id P\_Id Time\_Id Sale  
0 1 11 1 25.0  
1 1 11 2 8.0  
2 1 11 3 15.0  
3 1 12 1 30.0  
4 1 12 2 20.0  
5 1 12 3 50.0  
6 1 13 1 8.0  
7 1 13 2 10.0  
8 1 13 3 10.0  
9 2 11 1 35.0  
10 2 11 2 22.0  
11 2 11 3 10.0  
12 2 12 1 26.0  
13 2 12 2 45.0  
14 2 12 3 40.0  
15 2 13 1 20.0  
16 2 13 2 20.0  
17 2 13 3 40.0

Writing the SQL (without OLAP)

Find the total sales

query = """  
SELECT SUM(Sale) AS Total\_Sales  
FROM Sales;  
"""  
  
cursor.execute(query)  
total\_sales = cursor.fetchone()  
  
print("Total Sales: $", total\_sales[0])

Total Sales: $ 434.0

Find total sales for each city

query = """  
SELECT L.City, SUM(S.Sale) AS Total\_Sales  
FROM Sales S  
JOIN Locations L ON S.Loc\_Id = L.Loc\_Id  
GROUP BY L.City;  
"""  
  
cursor.execute(query)  
city\_sales = cursor.fetchall()  
  
print("City-wise Sales")  
print(pd.DataFrame(city\_sales, columns=['City', 'Total\_Sales']))

City-wise Sales  
 City Total\_Sales  
0 Fresno 258.0  
1 Madison 176.0

Find total sales for each state.

query = """  
SELECT L.State, SUM(S.Sale) AS Total\_Sales  
FROM Sales S  
JOIN Locations L ON S.Loc\_Id = L.Loc\_Id  
GROUP BY L.State;  
"""  
  
cursor.execute(query)  
state\_sales = cursor.fetchall()  
  
print("State-wise Sales")  
print(pd.DataFrame(state\_sales, columns=['State', 'Total\_Sales']))

State-wise Sales  
 State Total\_Sales  
0 WI 176.0  
1 CA 258.0

Find total sales for each country.

query = """  
SELECT L.Country, SUM(S.Sale) AS Total\_Sales  
FROM Sales S  
JOIN Locations L ON S.Loc\_Id = L.Loc\_Id  
GROUP BY L.Country;  
"""  
  
cursor.execute(query)  
country\_sales = cursor.fetchall()  
  
print("Country-wise Sales")  
print(pd.DataFrame(country\_sales, columns=['Country', 'Total\_Sales']))

Country-wise Sales  
 Country Total\_Sales  
0 USA 434.0

Find sales of all cities of a specific state (CA) in a specific year (2023).

query = """  
SELECT T.Year, L.City, SUM(S.Sale) AS Total\_Sales  
FROM Sales S  
JOIN Locations L ON S.Loc\_Id = L.Loc\_Id  
JOIN Times T ON S.Time\_Id = T.Time\_Id  
WHERE T.Year = 2023 AND L.State = 'CA'  
GROUP BY T.Year, L.City  
ORDER BY T.Year, L.City;  
"""  
  
cursor.execute(query)  
city\_sales\_2023 = cursor.fetchall()  
  
print("2023 City Sales in CA")  
print(pd.DataFrame(city\_sales\_2023, columns=['Year', 'City', 'Total\_Sales']))

2023 City Sales in CA  
 Year City Total\_Sales  
0 2023 Fresno 258.0

Find year-wise total sales for each state.

query = """  
SELECT T.Year, L.State, SUM(S.Sale) AS Total\_Sales  
FROM Sales S  
JOIN Locations L ON S.Loc\_Id = L.Loc\_Id  
JOIN Times T ON S.Time\_Id = T.Time\_Id  
GROUP BY T.Year, L.State  
ORDER BY T.Year, L.State;  
"""  
  
cursor.execute(query)  
year\_state\_sales = cursor.fetchall()  
  
print("Year and State-wise Sales")  
print(pd.DataFrame(year\_state\_sales, columns=['Year', 'State', 'Total\_Sales']))

Year and State-wise Sales  
 Year State Total\_Sales  
0 2023 CA 258.0  
1 2023 WI 176.0

Find year-wise total sales for each country.

query = """  
SELECT T.Year, L.Country, SUM(S.Sale) AS Total\_Sales  
FROM Sales S  
JOIN Locations L ON S.Loc\_Id = L.Loc\_Id  
JOIN Times T ON S.Time\_Id = T.Time\_Id  
GROUP BY T.Year, L.Country  
ORDER BY T.Year, L.Country;  
"""  
  
cursor.execute(query)  
year\_country\_sales = cursor.fetchall()  
  
print("Year and Country-wise Sales")  
print(pd.DataFrame(year\_country\_sales, columns=['Year', 'Country', 'Total\_Sales']))

Year and Country-wise Sales  
 Year Country Total\_Sales  
0 2023 USA 434.0

OLAP Queries

Find the total sales

query = """  
SELECT SUM(Sale) AS Total\_Sales  
FROM Sales  
GROUP BY CUBE(Loc\_Id, P\_Id, Time\_Id);  
"""  
  
cursor.execute(query)  
total\_sales = cursor.fetchone()  
  
print("Total Sales: $", total\_sales[0])

Total Sales: $ 434.0

Find total sales for each city.

query = """  
SELECT L.City, SUM(S.Sale) AS Total\_Sales  
FROM Sales S  
JOIN Locations L ON S.Loc\_Id = L.Loc\_Id  
GROUP BY ROLLUP (L.City);  
"""  
  
cursor.execute(query)  
city\_sales = cursor.fetchall()  
  
print("City-wise Sales")  
print(pd.DataFrame(city\_sales, columns=['City', 'Total\_Sales']))

City-wise Sales  
 City Total\_Sales  
0 None 434.0  
1 Fresno 258.0  
2 Madison 176.0

Find total sales for each state.

query = """  
SELECT L.State, SUM(S.Sale) AS Total\_Sales  
FROM Sales S  
JOIN Locations L ON S.Loc\_Id = L.Loc\_Id  
GROUP BY ROLLUP (L.State);  
"""  
  
cursor.execute(query)  
state\_sales = cursor.fetchall()  
  
print("State-wise Sales")  
print(pd.DataFrame(state\_sales, columns=['State', 'Total\_Sales']))

State-wise Sales  
 State Total\_Sales  
0 None 434.0  
1 WI 176.0  
2 CA 258.0

Find total sales for each country.

query = """  
SELECT L.Country, SUM(S.Sale) AS Total\_Sales  
FROM Sales S  
JOIN Locations L ON S.Loc\_Id = L.Loc\_Id  
GROUP BY ROLLUP (L.Country);  
"""  
  
cursor.execute(query)  
country\_sales = cursor.fetchall()  
  
print("Country-wise Sales")  
print(pd.DataFrame(country\_sales, columns=['Country', 'Total\_Sales']))

Country-wise Sales  
 Country Total\_Sales  
0 None 434.0  
1 USA 434.0

Find sales of all cities of a specific state (WI) in a specific year (2023).

query = """  
SELECT L.State, L.City, T.Year, SUM(S.Sale) AS Total\_Sales  
FROM Sales S  
JOIN Locations L ON S.Loc\_Id = L.Loc\_Id  
JOIN Times T ON S.Time\_Id = T.Time\_Id  
WHERE L.State = 'WI' AND T.Year = '2023'  
GROUP BY ROLLUP (L.City, L.State, T.Year);  
"""  
  
cursor.execute(query)  
city\_sales\_2023 = cursor.fetchall()  
  
print("2023 City Sales in WI")  
print(pd.DataFrame(city\_sales\_2023, columns=['State', 'City', 'Year', 'Total\_Sales']))

2023 City Sales in WI  
 State City Year Total\_Sales  
0 None None NaN 176.0  
1 WI Madison 2023.0 176.0  
2 WI Madison NaN 176.0  
3 None Madison NaN 176.0

Find year-wise total sales for each state.

query = """  
SELECT T.Year, L.State, SUM(S.Sale) AS Total\_Sales  
FROM Sales S  
JOIN Locations L ON S.Loc\_Id = L.Loc\_Id  
JOIN Times T ON S.Time\_Id = T.Time\_Id  
GROUP BY ROLLUP (T.Year, L.State)  
ORDER BY T.Year, L.State;  
"""  
  
cursor.execute(query)  
year\_state\_sales = cursor.fetchall()  
  
print("Year and State-wise Sales")  
print(pd.DataFrame(year\_state\_sales, columns=['Year', 'State', 'Total\_Sales']))

Year and State-wise Sales  
 Year State Total\_Sales  
0 2023.0 CA 258.0  
1 2023.0 WI 176.0  
2 2023.0 None 434.0  
3 NaN None 434.0

Find year-wise total sales for each country.

query = """  
SELECT T.Year, L.Country, SUM(S.Sale) AS Total\_Sales  
FROM Sales S  
JOIN Locations L ON S.Loc\_Id = L.Loc\_Id  
JOIN Times T ON S.Time\_Id = T.Time\_Id  
GROUP BY ROLLUP (T.Year, L.Country)  
ORDER BY T.Year, L.Country;  
"""  
  
cursor.execute(query)  
year\_country\_sales = cursor.fetchall()  
  
print("Year and Country-wise Sales")  
print(pd.DataFrame(year\_country\_sales, columns=['Year', 'Country', 'Total\_Sales']))

Year and Country-wise Sales  
 Year Country Total\_Sales  
0 2023.0 USA 434.0  
1 2023.0 None 434.0  
2 NaN None 434.0