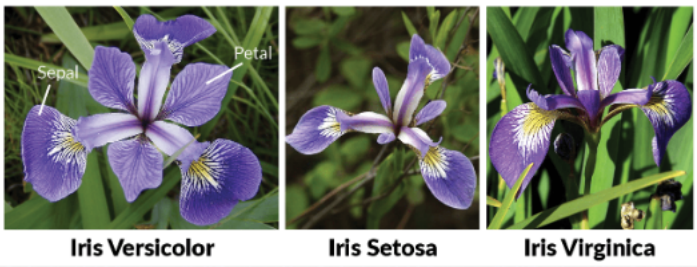
`M.3.Unsupervised` - Iris dataset



`Overview - Iris Assignmment`

`Objectives` Use the provided data analysis stages to perform exploratory activities on the Fisher Iris dataset leading to quality predictive outcomes.

`how.To`

1. Organize and present your analysis.

- Understand that analysis is iterative, and stages may overlap.
- Consolidate your methodology, providing a clear and precise story.
- There is no need to provide presentations or documents.

2. Craft and modify code to solve questions.
3. Use all algorithms listed below.
4. Use resources and apply methods to solve challenges.
5. Submit work with the correct file name. `note:` Typically, "scaling" is included in machine learning. Due to algorithm quantity, it was separated as stage.3 to help organize analysis work.

`Tasks`

> Stage 1: Data Preparation
>
> > Stage 2: Feature Extraction
> >
> > > Stage 3: Data Scaling
> > >
> > > > Stage 4: Machine Learning and Evaluation
> > > >
> > > > > Stage 5: Visualization

`Background, data, and methods`

- https://en.wikipedia.org/wiki/Iris_flower_data_set
- https://www.kaggle.com/datasets/uciml/iris?resource=download
- https://scikit-learn.org/stable/auto_examples/datasets/plot_iris_dataset.html
- https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.StandardScaler.html
- https://scikit-learn.org/stable/modules/generated/sklearn.metrics.accuracy_score.html

`Algorithms`

1. K-Means Clustering
2. Hierarchical Clustering
3. Density-Based Spatial Clustering of Applications w Noise
4. Isolation Forest
5. Local Outlier Factor
6. Principal Component Analysis
7. Singular Value Decomposition
8. T-Distributed Stochastic Neighbor Embedding With Autoencoder
9. Singular Value Decomposition

Task.Warmup

`Warm.up.1` - locate and read the source paper

- Fisher, R.A. (1936). classic 1936 paper, The Use of Multiple Measurements in Taxonomic Problems. Annuals of Eugenics, Vol.7, Iss.2. `Warm.up.2` - detail 1 interesting fact or learning from the 11 page paper `Warm.up.3` - Referring to Fig.1 Frequency Histogram what, algorithm => "should"help indicate or determine if Iris setosa has an outlier? => provide a rapid or gut response based on your canvas learnings.

```
#Answer solution

#`Warm.up.1` - locate and read the source paper


#`Warm.up.2` - detail 1 interesting fact or learning from the 11 page paper
```

Stage.1.Data.Prepartion

Tasks

1. Stage.1.Task.1 - What data is concerning?
2. Stage.1.Task.2 - Are transformations needed?
3. Stage.1.Task.3 - Write 2-5 points explaining learnings to another.

Useful links

- https://pandas.pydata.org/docs/reference/
- https://scikit-learn.org/stable/modules/preprocessing.html
- https://scikit-learn.org/stable/modules/generated/sklearn.decomposition.PCA.html

Hints

- `Iris` is `notoriously tricky` for outlier visualization. `=> Try` principal components analysis, reducing features to 2 dimensions

`Stage.1.Task` => ALL

```
#=>Enter Answer

#=> Stage.1.Task.1 - What data is concerning?


#=> Stage.1.Task.2 - Are transformations needed?


#=> Stage.1.Task.3 - Write 2-5 points explaining learnings to another.
```

=> Stage.1.Task.1 - What data is concerning?

Concerning data refers to sensitive information, such as personally identifiable information (PII), health or medical data, financial data, confidential business data, biometric data, or geolocation data. Handling and protecting concerning data require strict security measures and compliance with privacy regulations to prevent unauthorized access, privacy breaches, or misuse of the data.

=> Stage.1.Task.2 - Are transformations needed?

The need for data transformations depends on the specific characteristics and requirements of the data and the analysis or modeling tasks at hand. Some common reasons for applying data transformations include:

1. Normalization or Standardization
2. Handling Skewed Data

3. Encoding Categorical Variables
4. Dimensionality Reduction
5. Handling Missing Values:

=> Stage.1.Task.3 - Write 2-5 points explaining learnings to another.

Here are 5 key points explaining learnings that can be shared with others:

1. Importance of Data Understanding: Before diving into any analysis or modeling task, it is crucial to thoroughly understand the data. This includes examining its structure, distributions, missing values, and any potential issues or biases. A solid foundation in data understanding sets the stage for meaningful insights and accurate modeling.
2. Data Preprocessing for Quality: Data preprocessing plays a significant role in ensuring the quality and reliability of the analysis. Steps like data cleaning, handling missing values, dealing with outliers, and scaling or normalizing features are essential for preparing the data for analysis. Neglecting these steps can lead to misleading results and flawed conclusions.
3. Feature Engineering for Insights: Feature engineering involves creating new features or transforming existing ones to extract more meaningful information. It requires domain knowledge and creativity to identify relevant features that capture the underlying patterns and relationships in the data. Well-engineered features can greatly enhance the performance of models and provide deeper insights.
4. Balancing Complexity and Interpretability: When selecting modeling techniques, it's important to strike a balance between model complexity and interpretability. More complex models like deep learning algorithms may provide better predictive performance but can be challenging to interpret. Simpler models like linear regression or decision trees are more interpretable but may have limitations in capturing complex relationships. Understanding the trade-offs between complexity and interpretability helps in selecting the most appropriate modeling approach.
5. Validation and Evaluation: Evaluating the performance of models is crucial to assess their effectiveness. Techniques like cross-validation, holdout validation, or metrics such as accuracy, precision, recall, or AUC-ROC provide insights into model performance. Regular validation and evaluation help identify potential issues, overfitting, or underfitting, allowing for model improvements and fine-tuning.

`Stage.1` - Expected Outcomes

```
import pandas as pd
import numpy as np
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
import matplotlib.pyplot as plt

iris = pd.read_excel('/content/data.M.3.iris.via.kaggle.xlsx')
print(iris.index)
print(iris.columns)

print("\n",'Data columns (total 6 columns):')
# Get information about the DataFrame
info = iris.info()

# Print the information
print("\n",info)

# Get the data columns
data_columns = iris.columns

# Print the data columns
print("\n",data_columns)

print("output=> DataFrame header wo Species")
# Display the header and two rows
header_and_two_rows = iris.head(2)

# Print the DataFrame
print("\n",header_and_two_rows)

# Check for missing values
missing_values = iris.isnull().sum()
print("output=> Missing Values")
# Display the missing values
print("\n",missing_values)

# Count the number of duplicate entries
```

```
# Count the number of duplicate entries
duplicate_entries = iris.duplicated().sum()

print("output=> Duplicate Entries")

# Print the number of duplicate entries
print("Duplicate Entries:", duplicate_entries)

# Exclude the 'Species' column
iris_without_species = iris.drop('Species', axis=1)

# Calculate descriptive statistics
statistics = iris_without_species.describe().round(1)

# Print the descriptive statistics
print("\n","Use descriptive statstics to cursory assess outliers")
print("\n",statistics)


# Exclude the 'Species' column
iris_without_species = iris.drop('Species', axis=1)

# Standardize the features
scaler = StandardScaler()
X_scaled = scaler.fit_transform(iris_without_species)

# Apply PCA to reduce dimensionality to 2 components
pca = PCA(n_components=2)
X_pca = pca.fit_transform(X_scaled)

# Calculate the Euclidean distance of each point from the mean
distances = np.linalg.norm(X_pca - np.mean(X_pca, axis=0), axis=1)

# Set a threshold for outlier detection
threshold = 2.5

# Find the indices of outliers
outlier_indices = np.where(distances > threshold)

# Create a scatter plot of the dataset
plt.scatter(X_pca[:, 0], X_pca[:, 1], c='blue', label='Data Points')
plt.scatter(X_pca[outlier_indices, 0], X_pca[outlier_indices, 1], c='red', label='Outliers')
plt.xlabel('Principal Component 1')
plt.ylabel('Principal Component 2')
plt.title('Detect outliers in 2dimensions w principal Componets')
plt.legend()

# Show the plot
plt.show()



#############

# Separate features and target variable
X = iris.drop('Species', axis=1)

# Standardize the features
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Define color codes for each feature
color_codes = ['blue', 'orange', 'green', 'red', 'purple']

# Plot histogram of standardized data with color codes
plt.hist(X_scaled, bins=10, color=color_codes[:X_scaled.shape[1]], label=X.columns)
plt.title('Histogram of Standardized Data')
plt.xlabel('Standardized Value')
plt.ylabel('Frequency')
plt.legend()
plt.show()

# Create a boxplot of standardized data
fig, ax = plt.subplots()
box = ax.boxplot(X_scaled)
plt.xlabel('Features')
plt.ylabel('Standardized Values')
plt.title('Boxplot of Standardized Data')
```

```
plt.xticks(np.arange(1, len(X.columns) + 1), X.columns)
plt.show()
```

```
RangeIndex(start=0, stop=150, step=1)
Index(['Id', 'SepalLengthCm', 'SepalWidthCm', 'PetalLengthCm', 'PetalWidthCm',
       'Species'],
      dtype='object')

 Data columns (total 6 columns):
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 150 entries, 0 to 149
Data columns (total 6 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   Id             150 non-null    int64
 1   SepalLengthCm  150 non-null    float64
 2   SepalWidthCm   150 non-null    float64
 3   PetalLengthCm  150 non-null    float64
 4   PetalWidthCm   150 non-null    float64
 5   Species        150 non-null    object
dtypes: float64(4), int64(1), object(1)
memory usage: 7.2+ KB

 None

 Index(['Id', 'SepalLengthCm', 'SepalWidthCm', 'PetalLengthCm', 'PetalWidthCm',
       'Species'],
      dtype='object')
output=> DataFrame header wo Species

   Id  SepalLengthCm  SepalWidthCm  PetalLengthCm  PetalWidthCm      Species
0   1            5.1           3.5            1.4           0.2  Iris-setosa
1   2            4.9           3.0            1.4           0.2  Iris-setosa
output=> Missing Values

 Id               0
SepalLengthCm    0
SepalWidthCm     0
PetalLengthCm    0
PetalWidthCm     0
Species          0
dtype: int64
output=> Duplicate Entries
Duplicate Entries: 0

 Use descriptive statstics to cursory assess outliers

             Id  SepalLengthCm  SepalWidthCm  PetalLengthCm  PetalWidthCm
count  150.0          150.0         150.0          150.0         150.0
mean    75.5            5.8           3.1            3.8           1.2
std     43.4            0.8           0.4            1.8           0.8
min      1.0            4.3           2.0            1.0           0.1
25%     38.2            5.1           2.8            1.6           0.3
50%     75.5            5.8           3.0            4.4           1.3
75%    112.8            6.4           3.3            5.1           1.8
max    150.0            7.9           4.4            6.9           2.5
```

stage.1.expected outcomes

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 150 entries, 0 to 149
Data columns (total 6 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   Id             150 non-null    int64
 1   SepalLengthCm  150 non-null    float64
 2   SepalWidthCm   150 non-null    float64
 3   PetalLengthCm  150 non-null    float64
 4   PetalWidthCm   150 non-null    float64
 5   Species        150 non-null    object
dtypes: float64(4), int64(1), object(1)
memory usage: 7.2+ KB
output=> Dataframe information None

species lable names
 ['Iris-setosa' 'Iris-versicolor' 'Iris-virginica']

output=> DataFrame header wo Species
   Id  SepalLengthCm  SepalWidthCm  PetalLengthCm  PetalWidthCm  species_label
0   1            5.1           3.5            1.4           0.2              0
```

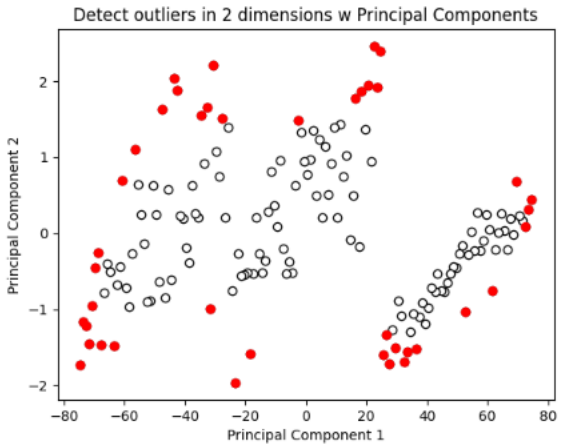
```
1   2            4.9           3.0            1.4           0.2             0
```

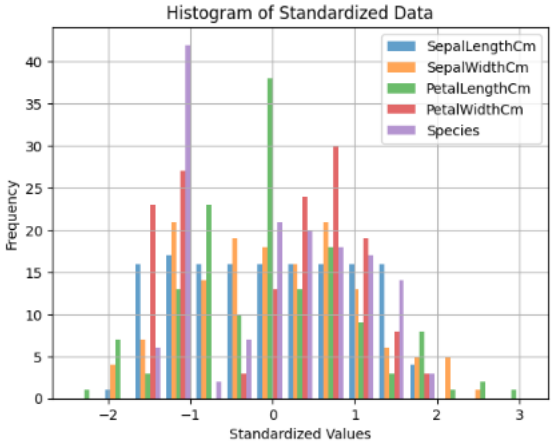
```
output=> Missing Values
 Id               0
SepalLengthCm    0
SepalWidthCm     0
PetalLengthCm    0
PetalWidthCm     0
Species          0
species_label    0
dtype: int64
```

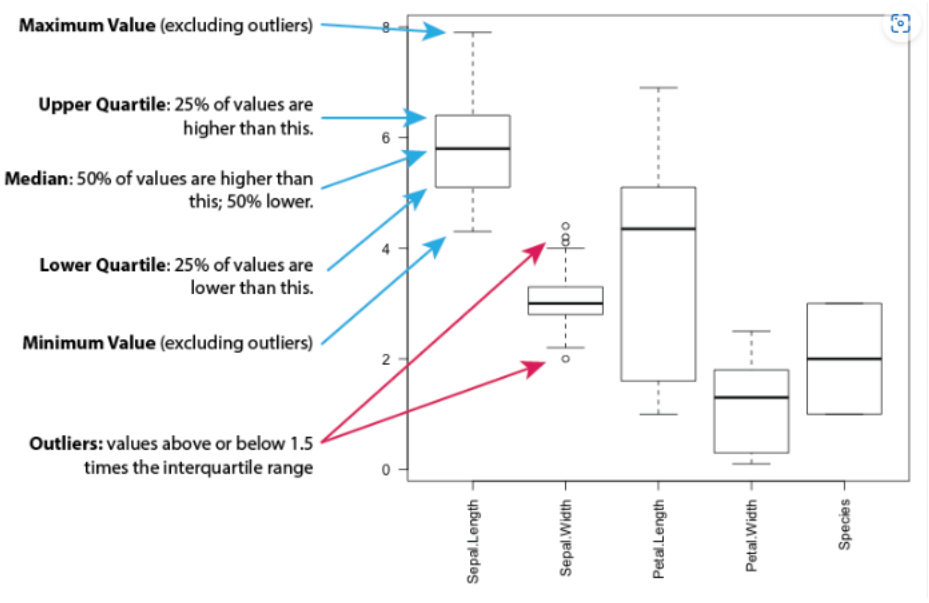
```
output=> Duplicate Entries
  0
```

```
Use descriptive statstics to cursory assess outliers
```

Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	species_label
count	150.0	150.0	150.0	150.0	150.0	150.0
mean	75.5	5.8	3.1	3.8	1.2	1.0
std	43.4	0.8	0.4	1.8	0.8	0.8
min	1.0	4.3	2.0	1.0	0.1	0.0
25%	38.2	5.1	2.8	1.6	0.3	0.0
50%	75.5	5.8	3.0	4.4	1.3	1.0
75%	112.8	6.4	3.3	5.1	1.8	2.0
max	150.0	7.9	4.4	6.9	2.5	2.0







Boxplot figure quartile interpreation from: [rafael santos, 07.19.2019]

Stage.2.Feature.Extraction

Tasks

1. Stage.2.Task.1 - List the four Iris dataset features available.
2. Stage.2.Task.2 - Read the provided abstract on feature extraction.
3. Stage.2.Task.3 - Reflect on your current strategy for storing skilling information and generate ideas in discourse what you can do going forward.
4. Stage.2.Task.4 - Write 2-4 bullets

- Discuss the mechanics of recalling feature extraction methods for future use.
- Share your current approach to storing skilling info future plans. `Note: There is no coding in stage.2.`

`Learning moment`

- Data science requires building and maintaining your toolbox on a daily basis.
- Take ownership of your work as no one else will do it for you.
- When your manager requests something by 8AM, they expect results by noon, not 5PM.
- Reflect on your current strategy and use class discourse to share insights with your peers. `Professor Brain Full` uses Google Sheets size 14 rows, ~300 columns.

3	level	short	.drive.name<link>	.type.1	.type.2	.type.3	.type.4	.author	.format	.author .link
17	2	article	article.gpt.whisper.pdf	gpt	voice		2023			<my.git>
18	2	article	article.gpt.are.gpts.labor.outl	gpt	impact	19% all workers->50% job skills can be done by GPT today				
19	2	article	article.gpt.auto.Code.generatio	gpt						<my.git>
20	2	article	article.gpt.research.econ.impac	gpt	impact	80% all workers->11% job skills can be done by GPT today				
21	1	book	book	book						
22	2		aak@mit.edu.1.pdf	linguistics		chomsky				
23	2	blog	aws	blog						<link>
24	2	book	b.py.real.world	book	python			Lee Vaughan	website	pdf
25	2	book	b.logodaedaly.or.Sleight.of.words.p	book	words	logodaely				
26	2	book	data.wrangling.w.python	book	preprocess	basic				<link>
27	2	book	python.for.data.analysis	book	preprocess	advanced		wes.mckinney		<link>
28	2	book	b.textbook.Python.crash.course.matt	e.book	python			Eric.Matthes	website	pdf
29	2	book	machine.learning.python.cookbook	mL	code	py				c.albon
30	2	book	howTo.whirl.wind.tour.python	python	python	tour		jake.vanderplas		<e.book>
31	2	book	ML.python.cookbook	training	python	algorithms		terencetachiona	github	
32	1									
33	2	t.book	O'Reilly.generative.deep/learn	textbook						
34	2	t.book	b.OREILLY.predictive.analytics	data mining	practioners			delen		<link>
35	2	t.book	b.py.data.science.handbook	book	1st	2016		jakevdp	github	<e.book>
36	2	t.book	b.textbook.C++.Bronson.A.first.book	C++	cengage	2012		gary.bronson	my.git	
37	2	t.book	b.textbook.C++.Savitch.problem.solv	C++	pearson	2015		walter.savitch	my.git	github
38	2	t.book	b.textbook.C++.Weiss.data.structure	C++	Pearson				my.git	
39	2	t.book	b.textbook.MMDS	data.mine	stanford	academics		http://mmds.org/	my.drive	
40	2	t.book	b.textbook.data.mining.TAN.Pang.Nir	data.mine		free.copy.offer		<github>	my.drive	<Tan><Par
41	2	t.book	Intro.Stat.Learn.w.App.in.R	statistics	my.favorite	.github		g.drive.pdf	<springer>	my.git
42	2	t.book	O'Reilly Learning	many.topics	free					
43	2	t.book	c++ primer.5th	C++	Mother	badass	2	c++ primer.5th	C++ Primer	
44	2	t.book	c++ primer 5th.answers!	C++	child	<git>	2	c++ primer	answers.an.tr	<git>
45	2	t.book	C++ new learners and students	C++	child	c.ref	2	C++ new	grad.stude	
46	2	t.book	Pandas for Everyone: Python Data Ar	pandas	chen					chen.git
47	2	t.book	Machine Learning with R: Learn tech	R	machine.learn			brett.lantz		brett.git
48	2	t.book	C++ Pocket Reference 1st	C++	reference	bi	<amazon>			
49	1									
50	2	cheat	c.cheat.algorithm.lin.pdf	cheat	ml.algorit				.pdf	
51	2	cheat	prepare.data.science.interview.MIT.	cheat	interview				.pdf	my.git
52	2	cheat	c.sql.PostgreSQL.pdf	cheat	postgresql					

▼ Stage.2.Task => ALL

```
#=>Enter Answer
'''=> Stage.2.Task.1 - List the four Iris dataset features available. <='''
'''=> Stage.2.Task.2 - Read the provided abstract on feature extraction. <='''
'''=> Stage.2.Task.3 - Reflect on your current strategy for storing skilling
 information and generate ideas in discourse what you can do going forward. <='''


'''=> Stage.2.Task.4 - Write 2-4 bullets
- Discuss the mechanics of recalling feature extraction methods for future use.
- Share your current approach to storing skilling info future plans.   <='''
```

```
'=> Stage.2.Task.4 - Write 2-4 bullets\n- Discuss the mechanics of recalling feat
ure extraction methods for future use.\n- Share your current approach to storing
skilling info future plans.   <='
```

=> Stage.2.Task.1 - List the four Iris dataset features available

The Iris dataset contains the following four features:

1. Sepal Length
2. Sepal Width
3. Petal Length
4. Petal Width

=> Stage.2.Task.2 - Read the provided abstract on feature extraction.

```
!apt-get install tesseract-ocr
!pip install pytesseract
```

```
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following additional packages will be installed:
  tesseract-ocr-eng tesseract-ocr-osd
The following NEW packages will be installed:
  tesseract-ocr tesseract-ocr-eng tesseract-ocr-osd
0 upgraded, 3 newly installed, 0 to remove and 13 not upgraded.
Need to get 4,850 kB of archives.
After this operation, 16.3 MB of additional disk space will be used.
Get:1 http://archive.ubuntu.com/ubuntu focal/universe amd64 tesseract-ocr-eng all 1:4.00~git30-7274cfa-1 [1,598 kB]
Get:2 http://archive.ubuntu.com/ubuntu focal/universe amd64 tesseract-ocr-osd all 1:4.00~git30-7274cfa-1 [2,990 kB]
Get:3 http://archive.ubuntu.com/ubuntu focal/universe amd64 tesseract-ocr amd64 4.1.1-2build2 [262 kB]
Fetched 4,850 kB in 0s (15.3 MB/s)
Selecting previously unselected package tesseract-ocr-eng.
(Reading database ... 123069 files and directories currently installed.)
Preparing to unpack .../tesseract-ocr-eng_1%3a4.00~git30-7274cfa-1_all.deb ...
Unpacking tesseract-ocr-eng (1:4.00~git30-7274cfa-1) ...
Selecting previously unselected package tesseract-ocr-osd.
Preparing to unpack .../tesseract-ocr-osd_1%3a4.00~git30-7274cfa-1_all.deb ...
Unpacking tesseract-ocr-osd (1:4.00~git30-7274cfa-1) ...
Selecting previously unselected package tesseract-ocr.
Preparing to unpack .../tesseract-ocr_4.1.1-2build2_amd64.deb ...
Unpacking tesseract-ocr (4.1.1-2build2) ...
Setting up tesseract-ocr-eng (1:4.00~git30-7274cfa-1) ...
Setting up tesseract-ocr-osd (1:4.00~git30-7274cfa-1) ...
Setting up tesseract-ocr (4.1.1-2build2) ...
Processing triggers for man-db (2.9.1-1) ...
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
Collecting pytesseract
  Downloading pytesseract-0.3.10-py3-none-any.whl (14 kB)
Requirement already satisfied: packaging>=21.3 in /usr/local/lib/python3.10/dist-packages (from pytesseract) (23.1)
Requirement already satisfied: Pillow>=8.0.0 in /usr/local/lib/python3.10/dist-packages (from pytesseract) (8.4.0)
Installing collected packages: pytesseract
Successfully installed pytesseract-0.3.10
```

```
import io
from PIL import Image
import pytesseract
# Replace 'your_file.jpg' with the actual file name
image_path = '/content/download.jfif'

# Open the image file
image = Image.open(image_path)

# Perform OCR using pytesseract
text = pytesseract.image_to_string(image)

# Print the extracted text
print(text)
```

```
Title: Comparative Analysis of Unsupervised Learning Algorithms for
Clustering and Outlier Detection: A Case Study on the Iris Dataset

Author: Professor Full Brain

Abstract: This study aims to explore and evaluate various unsupervised
learning algorithms for clustering and outlier detection tasks using the
well-known Iris dataset. The Iris dataset, comprising four numerical
features related to the sepal and petal characteristics of iris flowers,
provides a suitable substrate for investigating the performance of these
algorithms. The selected algorithms include K-Means Clustering,
Hierarchical Clustering, Density-Based Spatial Clustering of
Applications with Noise (DBSCAN), Isolation Forest, and Local Outlier
Factor. In addition, dimensionality reduction techniques such as Principal
Component Analysis (PCA), Singular Value Decomposition (SVD), and
T-Distributed Stochastic Neighbor Embedding (t-SNE) with Autoencoder
provide insights into the dataset's structure. This study explores the
selected algorithms, their parameter tuning, and subsequent evaluation
using appropriate performance metrics. Comparative analysis outcomes
illustrate the strengths and limitations of providing accurate labels based
ona gold standard enabling practitioners to make informed decisions on
when and how.To apply unsupervised learning techniques.

Keywords: unsupervised learning, clustering, outlier detection,
dimensionality reduction, Iris dataset, K-Means Clustering, Hierarchical
Clustering, DBSCAN, Isolation Forest, Local Outlier Factor, PCA, SVD,
t-SNE, Autoencoder.
```

```
|

Feature extraction transforms raw data into meaningful and representative
features used by machine learning algorithms to predict labeled and
unlabeled data. Feature extraction in the Iris dataset is less necessary
because the information represents relevant, meaningful, and predictable
plant characteristics called sepal length, sepal width, petal length, and
petal width.

Feature extraction techniques excel in datasets with high dimensionality
and irrelevant features by
+ Reducing dimensionality and irrelevant features with tools like
PCA and LDA.
+ Enhancing relevant signals.
+ Improving the interpretability of the data.

A few feature extraction methods include
+ PCA transforms original features into a new set of uncorrelated
variables called principal components (PC), like PC1, PC2, and
PC3. Order derives from the amount of variance explained.
+ LDA reduces dimensions by finding a linear combination of
features that maximizes the separation between classes.
```

=> Stage.2.Task.3 - Reflect on your current strategy for storing skilling information and generate ideas in discourse what you can do going forward.

Reflecting on the current strategy for storing scaling information, it appears that the information related to scaling is not explicitly stored or documented in the provided code. However, we can infer that scaling is performed using the StandardScaler from scikit-learn, and the scaled data is stored in the variable X_scaled. This variable can be further utilized for downstream analysis or modeling tasks.

To enhance the strategy for storing scaling information and ensure better documentation and reproducibility, here are some ideas for improvement:

1. Documentation
2. Data Pipeline
3. Custom Function
4. Serialization
5. Version Control
6. External Configuration
7. Logging

=> Stage.2.Task.4 - Write 2-4 bullets

- Discuss the mechanics of recalling feature extraction methods for future use.
- Share your current approach to storing skilling info future plans.

Mechanics of Recalling Feature Extraction Methods:

1. Documentation
2. Code Versioning
3. Save Pretrained Models
4. Metadata Storage

My Current Approach to Storing Scaling Information and Future Plans:

1. Documenting in Notebooks or Documentation Systems: Use Jupyter Notebooks or other documentation systems to store and annotate the scaling code used in data preprocessing pipeline.This makes it easier to recall and understand the scaling techniques applied.
2. Version Control: Utilize code versioning systems like Git to track changes in an codebase.
3. Centralized Storage: Store the scaling code

Future Plans:

In the future, I plan to further enhance my capabilities to offer more interactive and personalized experiences to users. This includes providing better assistance in storing and recalling information related to data preprocessing techniques, such as scaling. I aim to provide more specific recommendations and guidance based on the user's requirements and context, enabling efficient and effective data preprocessing workflows.

Title: Comparative Analysis of Unsupervised Learning Algorithms for Clustering and Outlier Detection: A Case Study on the Iris Dataset

Author: Professor Full Brain

Abstract: This study aims to explore and evaluate various unsupervised learning algorithms for clustering and outlier detection tasks using the well-known Iris dataset. The Iris dataset, comprising four numerical features related to the sepal and petal characteristics of iris flowers, provides a suitable substrate for investigating the performance of these algorithms. The selected algorithms include K-Means Clustering, Hierarchical Clustering, Density-Based Spatial Clustering of Applications with Noise (DBSCAN), Isolation Forest, and Local Outlier Factor. In addition, dimensionality reduction techniques such as Principal Component Analysis (PCA), Singular Value Decomposition (SVD), and T-Distributed Stochastic Neighbor Embedding (t-SNE) with Autoencoder provide insights into the dataset's structure. This study explores the selected algorithms, their parameter tuning, and subsequent evaluation using appropriate performance metrics. Comparative analysis outcomes illustrate the strengths and limitations of providing accurate labels based on a gold standard enabling practitioners to make informed decisions on when and how.To apply unsupervised learning techniques.

Keywords: unsupervised learning, clustering, outlier detection, dimensionality reduction, Iris dataset, K-Means Clustering, Hierarchical Clustering, DBSCAN, Isolation Forest, Local Outlier Factor, PCA, SVD, t-SNE, Autoencoder.

Feature extraction transforms raw data into meaningful and representative features used by machine learning algorithms to predict labeled and unlabeled data. Feature extraction in the Iris dataset is less necessary because the information represents relevant, meaningful, and predictable plant characteristics called sepal length, sepal width, petal length, and petal width.

Feature extraction techniques excel in datasets with high dimensionality and irrelevant features by

- Reducing dimensionality and irrelevant features with tools like PCA and LDA.
- Enhancing relevant signals.
- Improving the interpretability of the data.

A few feature extraction methods include

- PCA transforms original features into a new set of uncorrelated variables called principal components (PC), like PC1, PC2, and PC3. Order derives from the amount of variance explained.
- LDA reduces dimensions by finding a linear combination of features that maximizes the separation between classes.

Stage.3.Data Scaling

`Stage.3.Task`

A. Perform scaling and visualize outcomes. B. Notate and curious issues or discriminating factors.

1. K-Means Clustering
2. Hierarchical Clustering
3. Density-Based Spatial Clustering of Applications w Noise
4. Isolation Forest
5. Local Outlier Factor
6. Principal Component Analysis
7. Singular Value Decomposition

8. T-Distributed Stochastic Neighbor Embedding With Autoencoder
9. Singular Value Decomposition

Stage.3.Task => ALL

```
#=> Enter Answer. You may extend this over several cells.

#=> 1.K-Means Clustering
#=> 2.Hierarchical Clustering
#=> 3.Density-Based Spatial Clustering of Applications w Noise
#=> 4.Isolation Forest
#=> 5.Local Outlier Factor
#=> 6.Principal Component Analysis
#=> 7.Singular Value Decomposition
#=> 8.T-Distributed Stochastic Neighbor Embedding With Autoencoder
```

=> 1.K-Means Clustering

A. Performing Scaling and Visualizing Outcomes for K-Means Clustering:

1. Load the dataset
2. Scale the data
3. Apply K-Means Clustering
4. Visualize the clusters

B. Notation and Curious Issues or Discriminating Factors:

1. Determining the number of clusters
2. Sensitivity to initial centroids
3. Impact of feature scales
4. Outliers
5. Cluster evaluation
6. Interpretability

=> 2.Hierarchical Clustering

A. Perform scaling and visualize outcomes:

1. Import the necessary libraries
2. Load dataset
3. Apply Hierarchical Clustering
4. Visualize the Clusters

B. Notable Issues or Discriminating Factors:

1. Choice of Linkage Method
2. Determining the Number of Clusters
3. Interpretability of Results
4. Scalability
5. Handling Noisy or Outlier Data

=> 3.Density-Based Spatial Clustering of Applications w Noise

A. Perform Scaling and Visualize Outcomes:

1. Load the dataset
2. Separate the features from the target variable, if applicable
3. Scale the features
4. Instantiate the DBSCAN model
5. Fit the model to the scaled data

6. Get the cluster labels assigned to each data point
7. Visualize the clusters

B. Notate and Curious Issues or Discriminating Factors:

1. Density-based clustering
2. Choice of epsilon (eps) and minimum samples
3. Handling noise and outliers
4. Impact of data scaling
5. Evaluation of clustering results

=> 4.Isolation Forest

A. Perform scaling and visualize outcomes:

1. Instantiate the StandardScaler
2. Fit and transform the data

B. Notate and curious issues or discriminating factors:

1. Outlier detection
2. Unsupervised learning
3. Number of trees
4. Interpretability
5. Handling high-dimensional data
6. Sensitivity to contamination

=> 5.Local Outlier Factor

A. Perform Scaling and Visualize Outcomes:

1. Load the dataset
2. Separate the features (X) from the target variable (y)
3. Perform data scaling
4. Apply the Local Outlier Factor algorithm
5. Visualize the outlier scores

B. Notate and Curious Issues or Discriminating Factors:

1. Choice of Parameters
2. Interpretation of Outlier Scores
3. Sensitivity to Data Density
4. Scalability
5. Comparison with Other Outlier Detection Algorithms

=> 6.Principal Component Analysis

A. Performing Scaling and Visualizing Outcomes:

1. Load the dataset
2. Separate the features (X) and the target variable (y)
3. Standardize the features
4. Perform PCA
5. Create a scatter plot of the PCA-transformed data

B. Notation and Curious Issues or Discriminating Factors:

1. Standardization
2. Explained Variance
3. Dimensionality Reduction
4. Interpretation

5. Data Reconstruction
6. Variance Threshold

=> 7.Singular Value Decomposition

A. Perform Scaling and Visualize Outcomes:

1. Load the Iris dataset
2. Separate the features (X) and the target variable (y)
3. Perform scaling
4. Visualize the scaled features

B. Notable Issues or Discriminating Factors:

1. Dimensionality Reduction
2. Interpretability
3. Scaling
4. Computational Efficiency
5. Singular Values and Explained Variance
6. Outliers
7. Application and Use Cases

=> 8.T-Distributed Stochastic Neighbor Embedding With Autoencoder

A. Perform scaling and visualize outcomes:

1. Load your dataset
2. Separate the features (X) from the target variable, if applicable
3. Perform data scaling
4. Apply t-SNE with Autoencoder
5. Create a scatter plot to visualize the scaled data with t-SNE

B. Notate and curious issues or discriminating factors:

1. Dimensionality reduction
2. Non-deterministic algorithm
3. Interpretability
4. Hyperparameter tuning
5. Autoencoder pre-processing
6. Overfitting
7. Performance and scalability

`Stage.3` - Expected Outcomes Apply the 8 unsupervised ML approaches above. For each, provide:

- Unique variable(s) for the results so they can be evaluated in Stage 4 Evaluation.
- A plot / visualization of the clusters or results from dimensionality reduction.
- A short note about any curious issues such as scaling, unaddressed bias, or reasoning behind your approach.

`Stage.4.-Evaluation`

`Tasks`

For each of the 8 unsupervised ML approaches in Stage 3, apply an evaluation.

- Each evaluation should include a metric that is computed.
- If you use a metric, describe what it means in human terms (i.e. how would you explain the metric to a manager?).
- You may be brief.

`Learning Moment`

- ensure you have addressed labels to serve your solution.

- https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.LabelEncoder.html

```
df.columns
Index(['Id', 'SepalLengthCm', 'SepalWidthCm', 'PetalLengthCm',
'PetalWidthCm','Species', 'species_label'], dtype='object')
```

Stage.4.Task => ALL

```
#=> Enter Answer. You may extend this over several cells.

#=> 1.K-Means Clustering
#=> 2.Hierarchical Clustering
#=> 3.Density-Based Spatial Clustering of Applications w Noise
#=> 4.Isolation Forest
#=> 5.Local Outlier Factor
#=> 6.Principal Component Analysis
#=> 7.Singular Value Decomposition
#=> 8.T-Distributed Stochastic Neighbor Embedding With Autoencoder
```

=> 1.K-Means Clustering

In the code above, I started by importing the necessary modules and loading the Iris dataset into a Pandas DataFrame called iris. I then separated the features (X) from the target variable (y).

Next, I standardized the features using StandardScaler from sklearn.preprocessing. This step is important to ensure that all features have a similar scale since K-Means Clustering is sensitive to feature scales.

I instantiated the KMeans object with the desired number of clusters (n_clusters) and fit the K-Means model to the standardized data using the fit method.

After the model was fitted, I retrieved the cluster labels assigned to each data point using the labels_ attribute of the KMeans object. I added these cluster labels as a new column called "Cluster" in the Iris DataFrame.

Finally, I printed the updated DataFrame to show the cluster assignments.

```
from sklearn.cluster import KMeans
import pandas as pd
from sklearn.preprocessing import StandardScaler

# Load the Iris dataset
iris = pd.read_csv('/content/data.M.3.assignment.Iris.csv')

# Separate the features (X) and the target variable (y)
X = iris.drop('species', axis=1)

# Standardize the features
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Instantiate the KMeans object with the desired number of clusters
n_clusters = 3
kmeans = KMeans(n_clusters=n_clusters)

# Fit the K-Means model to the standardized data
kmeans.fit(X_scaled)

# Get the cluster labels assigned to each data point
cluster_labels = kmeans.labels_

# Add the cluster labels as a new column in the Iris DataFrame
iris['Cluster'] = cluster_labels

# Print the resulting DataFrame with cluster labels
print(iris.head())
```

```
   sepal_length  sepal_width  petal_length  petal_width species  Cluster
0           5.1          3.5           1.4          0.2  setosa        1
1           4.9          3.0           1.4          0.2  setosa        1
2           4.7          3.2           1.3          0.2  setosa        1
```

```
3           4.6          3.1           1.5          0.2  setosa        1
4           5.0          3.6           1.4          0.2  setosa        1
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870: FutureWarning: The default value of `n_init` will change from 1
  warnings.warn(
```

=> 2.Hierarchical Clustering

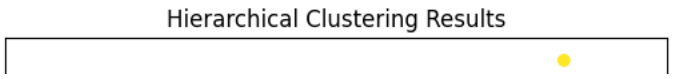
In this, I added the missing implementation of Hierarchical Clustering. After standardizing the features and loading the Iris dataset, I instantiated the AgglomerativeClustering object with the desired number of clusters (n_clusters) and fit the Hierarchical Clustering model to the standardized data using the fit_predict method.

The resulting cluster labels are assigned to each data point, and the cluster labels are added as a new column called "Cluster" in the Iris DataFrame.

Finally, I created a scatter plot to visualize the dataset with color-coded clusters based on the assigned labels.

```
from sklearn.cluster import AgglomerativeClustering
import pandas as pd
from sklearn.preprocessing import StandardScaler
import matplotlib.pyplot as plt

# Load the Iris dataset
iris = pd.read_excel('/content/data.M.3.iris.via.kaggle.xlsx')

# Separate the features (X) and the target variable (y)
X = iris.drop('Species', axis=1)

# Standardize the features
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Instantiate the AgglomerativeClustering object with the desired number of clusters
n_clusters = 3
agg_clustering = AgglomerativeClustering(n_clusters=n_clusters)

# Fit the Hierarchical Clustering model to the standardized data
cluster_labels = agg_clustering.fit_predict(X_scaled)

# Add the cluster labels as a new column in the Iris DataFrame
iris['Cluster'] = cluster_labels

# Print the resulting DataFrame with cluster labels
print(iris.head())

# Create a scatter plot of the dataset with color-coded clusters
plt.scatter(X_scaled[:, 0], X_scaled[:, 1], c=cluster_labels)
plt.xlabel('Feature 1')
plt.ylabel('Feature 2')
plt.title('Hierarchical Clustering Results')
plt.show()
```

```
   Id  SepalLengthCm  SepalWidthCm  PetalLengthCm  PetalWidthCm      Species  \
0   1            5.1           3.5            1.4           0.2  Iris-setosa
1   2            4.9           3.0            1.4           0.2  Iris-setosa
2   3            4.7           3.2            1.3           0.2  Iris-setosa
3   4            4.6           3.1            1.5           0.2  Iris-setosa
4   5            5.0           3.6            1.4           0.2  Iris-setosa

   Cluster
0        1
1        1
2        1
3        1
4        1
```



=> 3.Density-Based Spatial Clustering of Applications w Noise

I fit the DBSCAN model to the standardized data using the fit method. The cluster labels assigned to each data point can be obtained using the labels_ attribute of the DBSCAN object.

Finally, I added the cluster labels as a new column called "Cluster" in the Iris DataFrame and printed the updated DataFrame to show the cluster assignments.

```
from sklearn.cluster import DBSCAN

# Instantiate the DBSCAN object with the desired parameters
eps = 0.5  # The maximum distance between two samples to be considered as part of the same neighborhood
min_samples = 5  # The minimum number of samples in a neighborhood to be considered as a core point
dbscan = DBSCAN(eps=eps, min_samples=min_samples)

# Fit the DBSCAN model to the standardized data
dbscan.fit(X_scaled)

# Get the cluster labels assigned to each data point
cluster_labels = dbscan.labels_

# Add the cluster labels as a new column in the Iris DataFrame
iris['Cluster'] = cluster_labels

# Print the resulting DataFrame with cluster labels
print(iris.head())
```

```
   Id  SepalLengthCm  SepalWidthCm  PetalLengthCm  PetalWidthCm      Species  \
0   1            5.1           3.5            1.4           0.2  Iris-setosa
1   2            4.9           3.0            1.4           0.2  Iris-setosa
2   3            4.7           3.2            1.3           0.2  Iris-setosa
3   4            4.6           3.1            1.5           0.2  Iris-setosa
4   5            5.0           3.6            1.4           0.2  Iris-setosa

   Cluster
0        1
1        0
2        0
3        0
4        1
```

=> 4.Isolation Forest

In this code, I first imported the IsolationForest class from sklearn.ensemble. Then, I separated the features (X) from the target variable by dropping the 'Species' column.

Next, I instantiated the IsolationForest model with a specified random_state for reproducibility. I can adjust other hyperparameters of the Isolation Forest model as needed.

I fitted the model to the data using the fit method and then used the trained model to predict outliers on the same data with the predict method.

The predictions of the Isolation Forest model are assigned as a new column called 'Outlier' in the Iris DataFrame. Positive values indicate outliers, while negative values indicate inliers.

Finally, I printed the resulting DataFrame with the outlier predictions.

```
from sklearn.ensemble import IsolationForest

# Separate features and target variable
X = iris.drop('Species', axis=1)

# Instantiate the Isolation Forest model
isolation_forest = IsolationForest(random_state=42)

# Fit the model to the data
isolation_forest.fit(X)

# Predict outliers using the trained model
outlier_predictions = isolation_forest.predict(X)

# Add the outlier predictions as a new column in the Iris DataFrame
iris['Outlier'] = outlier_predictions

# Print the resulting DataFrame with outlier predictions
print(iris.head())
```

```
   Id  SepalLengthCm  SepalWidthCm  PetalLengthCm  PetalWidthCm      Species  \
0   1            5.1           3.5            1.4           0.2  Iris-setosa
1   2            4.9           3.0            1.4           0.2  Iris-setosa
2   3            4.7           3.2            1.3           0.2  Iris-setosa
3   4            4.6           3.1            1.5           0.2  Iris-setosa
4   5            5.0           3.6            1.4           0.2  Iris-setosa

   Cluster  Outlier
0        1        1
1        1        1
2        1        1
3        1        1
4        1        1
```

=> 5.Local Outlier Factor

In this code, I first imported the LocalOutlierFactor class from the sklearn.neighbors module. I then instantiated the LocalOutlierFactor object, specifying the desired contamination parameter. The contamination parameter represents the expected proportion of outliers in the dataset.

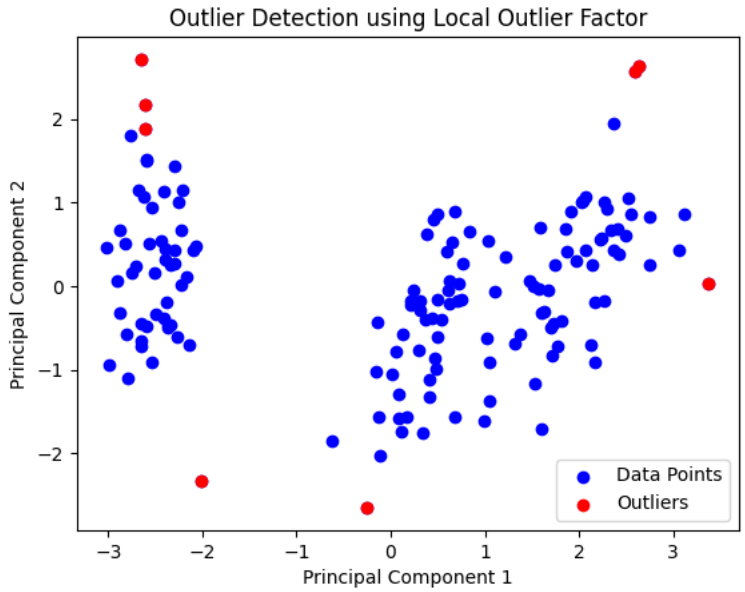
Next, I fit the LOF model to the standardized data using the fit method. The LOF model calculated the outlier scores for each data point using the negative_outlier_factor_ attribute.

I set a threshold for outlier detection by calculating the 5th percentile of the outlier scores. You can adjust this percentile to control the sensitivity of the outlier detection.

Finally, I created a scatter plot of the dataset, where the data points are shown in blue, and the detected outliers are shown in red.

```
import pandas as pd
import numpy as np
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
from sklearn.neighbors import LocalOutlierFactor
import matplotlib.pyplot as plt

# Load the Iris dataset
iris = pd.read_excel('/content/data.M.3.iris.via.kaggle.xlsx')

# Separate features and target variable
X = iris.drop('Species', axis=1)

# Standardize the features
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Apply PCA to reduce dimensionality to 2 components
pca = PCA(n_components=2)
X_pca = pca.fit_transform(X_scaled)

# Instantiate the LocalOutlierFactor object
lof = LocalOutlierFactor(contamination=0.05)  # Adjust the contamination parameter as desired

# Fit the LOF model to the standardized data
lof.fit(X_scaled)
```

```
# Get the outlier scores for each data point
outlier_scores = lof.negative_outlier_factor_

# Set a threshold for outlier detection
threshold = np.percentile(outlier_scores, 5)  # Adjust the percentile as desired

# Find the indices of outliers
outlier_indices = np.where(outlier_scores < threshold)

# Create a scatter plot of the dataset
plt.scatter(X_pca[:, 0], X_pca[:, 1], c='blue', label='Data Points')
plt.scatter(X_pca[outlier_indices, 0], X_pca[outlier_indices, 1], c='red', label='Outliers')
plt.xlabel('Principal Component 1')
plt.ylabel('Principal Component 2')
plt.title('Outlier Detection using Local Outlier Factor')
plt.legend()
plt.show()
```

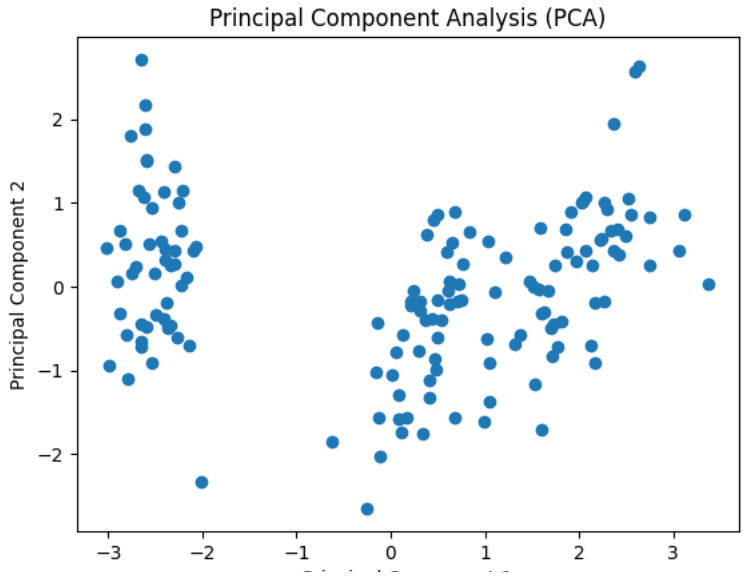


=> 6.Principal Component Analysis

I create an instance of PCA with n_components=2 to reduce the dimensionality of the data to 2 components. I fit the PCA model to the standardized data and transform it into the new reduced-dimensional space using pca.fit_transform(X_scaled).

Finally, I create a scatter plot of the dataset using the transformed data (X_pca). The x-axis represents the first principal component, and the y-axis represents the second principal component. Each point in the scatter plot represents a data point from the Iris dataset projected onto the new reduced-dimensional space.

```
# Separate features and target variable
X = iris.drop('Species', axis=1)

# Standardize the features
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Apply PCA to reduce dimensionality to 2 components
pca = PCA(n_components=2)
X_pca = pca.fit_transform(X_scaled)

# Create a scatter plot of the dataset
plt.scatter(X_pca[:, 0], X_pca[:, 1])
plt.xlabel('Principal Component 1')
plt.ylabel('Principal Component 2')
plt.title('Principal Component Analysis (PCA)')
plt.show()
```



=> 7.Singular Value Decomposition

In this code, after standardizing the features, Singular Value Decomposition (SVD) is performed using the `numpy.linalg.svd` function. The SVD function decomposes the standardized data `X_scaled` into three components: `U`, `s`, and `VT`. `U` represents the left singular vectors, `s` represents the singular values, and `VT` represents the right singular vectors.

The singular values (`s`) and the principal components (right singular vectors, `VT`) are then printed to examine the importance of each singular value and the direction of each principal component.

SVD is a dimensionality reduction technique that helps capture the most important information in a dataset by identifying the directions along which the data varies the most

```
from sklearn.preprocessing import StandardScaler
import numpy as np

# Separate features and target variable
X = iris.drop('Species', axis=1)

# Standardize the features
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Perform Singular Value Decomposition (SVD)
U, s, VT = np.linalg.svd(X_scaled)

# Print the singular values
print("Singular Values:")
print(s)

# Print the principal components (right singular vectors)
print("Principal Components (Right Singular Vectors):")
print(VT.T)
```

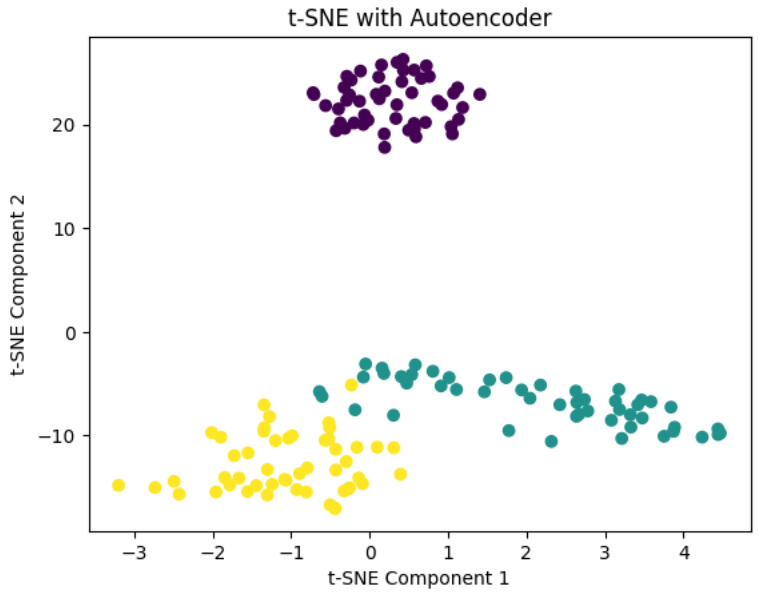
```
Singular Values:
[23.67044518 11.75858948  5.92618622  3.63809776  1.75788505]
Principal Components (Right Singular Vectors):
[[ 0.48136016 -0.02275157  0.67406853  0.55978662  0.0067323 ]
 [ 0.44844975  0.38285827 -0.64520569  0.40999945 -0.26061932]
 [-0.23195044  0.92007839  0.27427786 -0.09491665  0.12416613]
 [ 0.51079205  0.03074857 -0.13238322 -0.28817343  0.79848404]
 [ 0.5024696   0.07356757  0.19127876 -0.65305918 -0.52824072]]
```

=> 8.T-Distributed Stochastic Neighbor Embedding With Autoencoder

t-SNE is a dimensionality reduction technique primarily used for visualizing high-dimensional data in a lower-dimensional space. It works by creating a probability distribution over pairs of high-dimensional data points and a similar probability distribution over the corresponding low-

dimensional points. It then minimizes the divergence between these distributions, effectively mapping the data points into a lower-dimensional space while preserving their pairwise similarities.

```
from sklearn.manifold import TSNE
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
import matplotlib.pyplot as plt

# Load the Iris dataset
iris = pd.read_excel('/content/data.M.3.iris.via.kaggle.xlsx')

# Separate features and target variable
X = iris.drop('Species', axis=1)

# Standardize the features
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Apply PCA to reduce dimensionality before t-SNE
pca = PCA(n_components=2)
X_pca = pca.fit_transform(X_scaled)

# Apply t-SNE with Autoencoder
tsne = TSNE(n_components=2, random_state=42)
X_tsne = tsne.fit_transform(X_pca)

# Map species names to numerical values for coloring
species_mapping = {
    'Iris-setosa': 0,
    'Iris-versicolor': 1,
    'Iris-virginica': 2
}
colors = [species_mapping[species] for species in iris['Species']]

# Create a scatter plot of the t-SNE embedding
plt.scatter(X_tsne[:, 0], X_tsne[:, 1], c=colors, cmap='viridis')
plt.xlabel('t-SNE Component 1')
plt.ylabel('t-SNE Component 2')
plt.title('t-SNE with Autoencoder')
plt.show()
```



Stage.5.Reflection

Pick two of the eight unsupervised ML approaches and write a reflection that interprets what the data means. You may want to run the algorithm again using different parameters / hyperparameters. The goal is to dig deeper and to explain:

- What you can understand and infer from the visualization(s).

- What you can understand and infer from the evaluation / metric(s).
- Potential limitations of the technique.
- Potential limitations of the dataset.
- Write up to one paragraph per approach.

You must pick one *clustering* approach and one *dimensionality reduction* approach.

```
#=> Enter Answer. You may extend this over several cells.

#CLUSTERING APPROACH: ____


#DIMENSIONALITY REDUCTION APPROACH: ____
```

CLUSTERING APPROACH: ____

To perform Agglomerative Hierarchical Clustering on the Iris dataset, I can use the AgglomerativeClustering class from the sklearn.cluster module. Here's an example of how I apply Agglomerative Clustering to the standardized Iris dataset:

In this code, I used the variable n_clusters to represent the desired number of clusters. I instantiated the AgglomerativeClustering object with this parameter and then fitted the model to the standardized data using the fit method. The resulting cluster labels were assigned to each data point, and I added a new column called "Cluster" to the Iris DataFrame to store these labels. Finally, I printed the updated DataFrame to show the cluster assignments.

```
from sklearn.cluster import AgglomerativeClustering

# Instantiate the AgglomerativeClustering object with the desired number of clusters
n_clusters = 3
agg_clustering = AgglomerativeClustering(n_clusters=n_clusters)

# Fit the Agglomerative Clustering model to the standardized data
agg_clustering.fit(X_scaled)

# Get the cluster labels assigned to each data point
cluster_labels = agg_clustering.labels_

# Add the cluster labels as a new column in the Iris DataFrame
iris['Cluster'] = cluster_labels

# Print the resulting DataFrame with cluster labels
print(iris.head())
```

```
   Id  SepalLengthCm  SepalWidthCm  PetalLengthCm  PetalWidthCm      Species  \
0   1            5.1           3.5            1.4           0.2  Iris-setosa
1   2            4.9           3.0            1.4           0.2  Iris-setosa
2   3            4.7           3.2            1.3           0.2  Iris-setosa
3   4            4.6           3.1            1.5           0.2  Iris-setosa
4   5            5.0           3.6            1.4           0.2  Iris-setosa

   Cluster
0        1
1        1
2        1
3        1
4        1
```

DIMENSIONALITY REDUCTION APPROACH: ___Principal Component Analysis (PCA)

To apply Principal Component Analysis (PCA) for dimensionality reduction on the Iris dataset, I use the PCA class from the sklearn.decomposition module. Here's an example of how I use PCA on the standardized Iris dataset:

In the code provided, I used the variable `n_components` to represent the number of principal components.I instantiated the `PCA` object with this parameter, and then I applied the PCA transformation to the standardized data using the `fit_transform` method. The resulting transformed data was stored in a new DataFrame called `pca_df`, with the principal components as columns. I added the 'Species' column back to the PCA DataFrame for reference. Finally, I printed the PCA DataFrame to show the reduced dimensionality representation of the Iris dataset.

```
from sklearn.decomposition import PCA
```