Import libraries

```
import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
import matplotlib.pyplot as plt
import seaborn as sns
```

Load Dataset

```
# Load data
data = pd.read_csv('/content/column_2C_weka.csv')
# get_dummies
df = pd.get_dummies(data)
df.head(10)
```

	pelvic_incidence	pelvic_tilt numeric	lumbar_lordosis_angle	sacral_slope	pelvic_radius
0	63.027817	22.552586	39.609117	40.475232	98.672917
1	39.056951	10.060991	25.015378	28.995960	114.405425
2	68.832021	22.218482	50.092194	46.613539	105.985135
3	69.297008	24.652878	44.311238	44.644130	101.868495
4	49.712859	9.652075	28.317406	40.060784	108.168725
5	40.250200	13.921907	25.124950	26.328293	130.327871
6	53.432928	15.864336	37.165934	37.568592	120.567523
7	45.366754	10.755611	29.038349	34.611142	117.270067
8	43.790190	13.533753	42.690814	30.256437	125.002893

Next steps: Generate code with df View recommended plots

```
# drop one of the feature
df.drop("class_Normal",axis = 1, inplace = True)
df.head(10)
```

	pelvic_incidence	pelvic_tilt numeric	lumbar_lordosis_angle	sacral_slope	pelvic_radius
0	63.027817	22.552586	39.609117	40.475232	98.672917
1	39.056951	10.060991	25.015378	28.995960	114.405425
2	68.832021	22.218482	50.092194	46.613539	105.985135
3	69.297008	24.652878	44.311238	44.644130	101.868495
4	49.712859	9.652075	28.317406	40.060784	108.168725
5	40.250200	13.921907	25.124950	26.328293	130.327871
6	53.432928	15.864336	37.165934	37.568592	120.567523
7	45.366754	10.755611	29.038349	34.611142	117.270067
8	43.790190	13.533753	42.690814	30.256437	125.002893

Next steps: Generate code with df View recommended plots

```
from sklearn.svm import SVC
from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import Pipeline
from sklearn.model_selection import train_test_split, GridSearchCV

# Define the steps of the pipeline
steps = [('scalar', StandardScaler()),
         ('SVM', SVC())]

# Create the pipeline
pipeline = Pipeline(steps)

# Define the parameter grid for GridSearchCV
parameters = {'SVM__C': [1, 10, 100],
              'SVM__gamma': [0.1, 0.01]}

# Split the data into training and testing sets
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2, random_state=1)

# Create and fit the GridSearchCV
cv = GridSearchCV(pipeline, param_grid=parameters, cv=3)
cv.fit(x_train, y_train)

# Make predictions
y_pred = cv.predict(x_test)

# Print the accuracy score
print("Accuracy: {}".format(cv.score(x_test, y_test)))
```

```
Accuracy: 0.8548387096774194
```

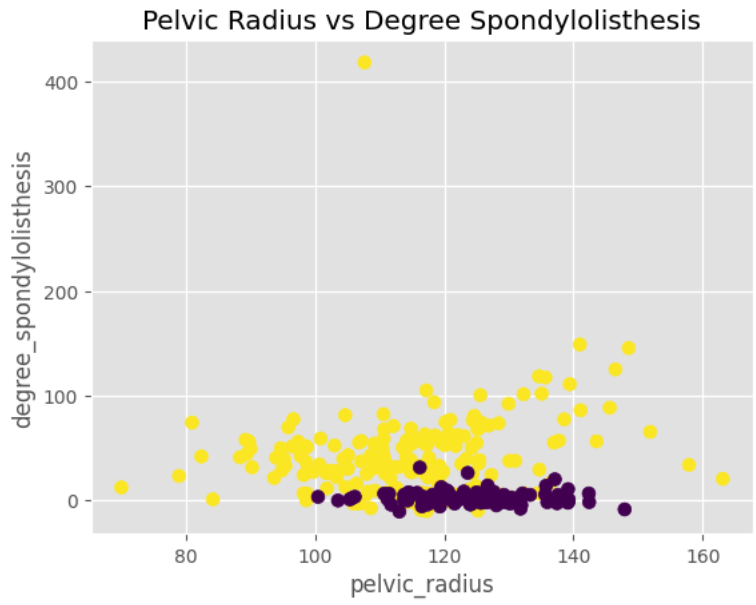
UNSUPERVISED LEARNING

Unsupervised learning: It uses data that has unlabeled and uncover hidden patterns from unlabeled data. Example, there are orthopedic patients data that do not have labels. You do not know which orthopedic patient is normal or abnormal. As you know orthopedic patients data is labeled (supervised) data. It has target variables. In order to work on unsupervised learning, lets drop target variables and to visualize just consider pelvic_radius and degree_spondylolisthesis

KMEANS Lets try our first unsupervised method that is KMeans Cluster KMeans Cluster: The algorithm works iteratively to assign each data point to one of K groups based on the features that are provided. Data points are clustered based on feature similarity KMeans(n_clusters = 2): n_clusters = 2 means that create 2 cluster

```
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.svm import SVC
from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import Pipeline
from sklearn.model_selection import train_test_split, GridSearchCV

# Load the dataset
data = pd.read_csv('/content/column_2C_weka.csv')

# Extract features and labels
X = data[['pelvic_radius', 'degree_spondylolisthesis']]
y = data['class']  # Assuming 'class' is the name of the target variable

# Split the data into training and testing sets
x_train, x_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=1)

# Define the steps of the pipeline
steps = [('scaler', StandardScaler()), ('SVM', SVC())]

# Create the pipeline
pipeline = Pipeline(steps)

# Define the parameter grid for GridSearchCV
parameters = {'SVM__C': [1, 10, 100],
              'SVM__gamma': [0.1, 0.01]}

# Create and fit the GridSearchCV
cv = GridSearchCV(pipeline, param_grid=parameters, cv=3)
cv.fit(x_train, y_train)

# Print the accuracy score
print("Accuracy: {}".format(cv.score(x_test, y_test)))

# Plot the dataset
plt.scatter(data['pelvic_radius'], data['degree_spondylolisthesis'], c=y.map({'Abnormal': 1, 'Normal': 0}), s=50, cmap='viridis')
plt.xlabel('pelvic_radius')
plt.ylabel('degree_spondylolisthesis')
plt.title("Pelvic Radius vs Degree Spondylolisthesis")
plt.show()
```

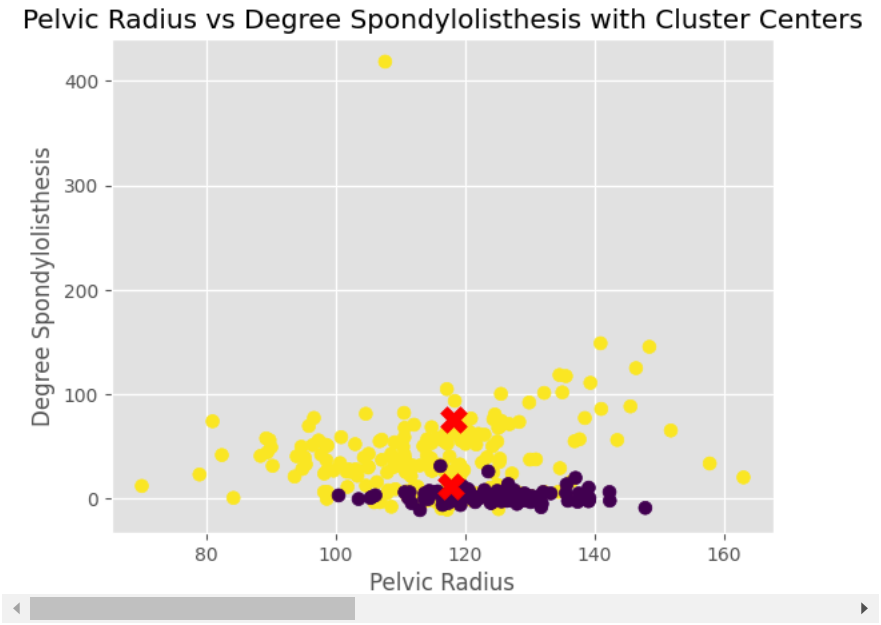
```
Accuracy: 0.8064516129032258
```




```
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.svm import SVC
from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import Pipeline
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.cluster import KMeans

# Load the dataset
data = pd.read_csv('/content/column_2C_weka.csv')

# Extract features and labels
X = data[['pelvic_radius', 'degree_spondylolisthesis']]
y = data['class']  # Assuming 'class' is the name of the target variable
```

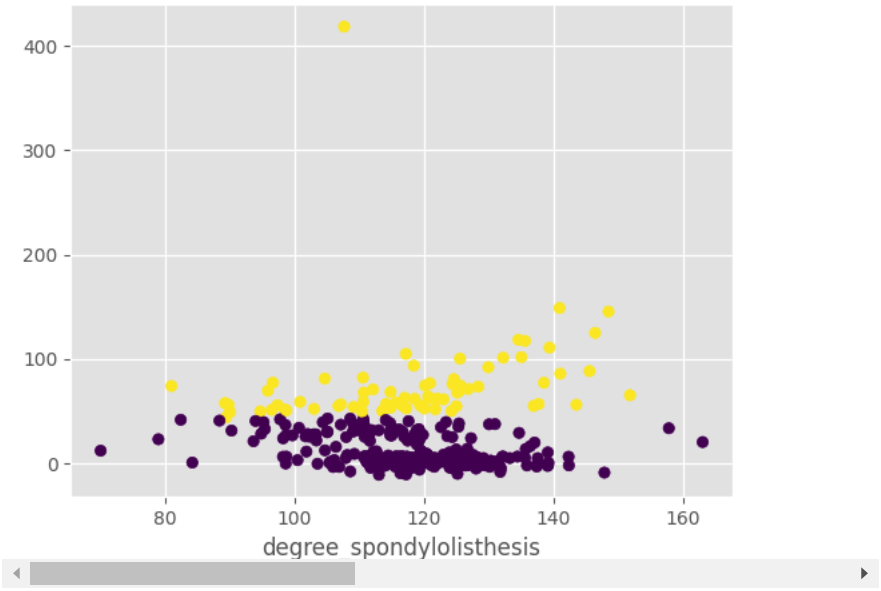
```
# Split the data into training and testing sets
x_train, x_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=1)

# Define the steps of the pipeline
steps = [('scaler', StandardScaler()), ('SVM', SVC())]

# Create the pipeline
pipeline = Pipeline(steps)

# Define the parameter grid for GridSearchCV
parameters = {'SVM__C': [1, 10, 100],
              'SVM__gamma': [0.1, 0.01]}

# Create and fit the GridSearchCV
cv = GridSearchCV(pipeline, param_grid=parameters, cv=3)
cv.fit(x_train, y_train)

# Print the accuracy score
print("Accuracy: {}".format(cv.score(x_test, y_test)))

# Apply k-means clustering to find cluster centers
kmeans = KMeans(n_clusters=2, random_state=0)
kmeans.fit(X)
cluster_centers = kmeans.cluster_centers_

# Plot the dataset colored by labels
plt.scatter(data['pelvic_radius'], data['degree_spondylolisthesis'], c=y.map({'Abnormal': 1, 'Normal': 0}), s=50, cmap='viridis')
plt.scatter(cluster_centers[:, 0], cluster_centers[:, 1], s=200, c='red', marker='X')  # Plot cluster centers
plt.xlabel('Pelvic Radius')
plt.ylabel('Degree Spondylolisthesis')
plt.title("Pelvic Radius vs Degree Spondylolisthesis with Cluster Centers")
plt.show()
```

```
Accuracy: 0.8064516129032258
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870: FutureWarning
  warnings.warn(
```

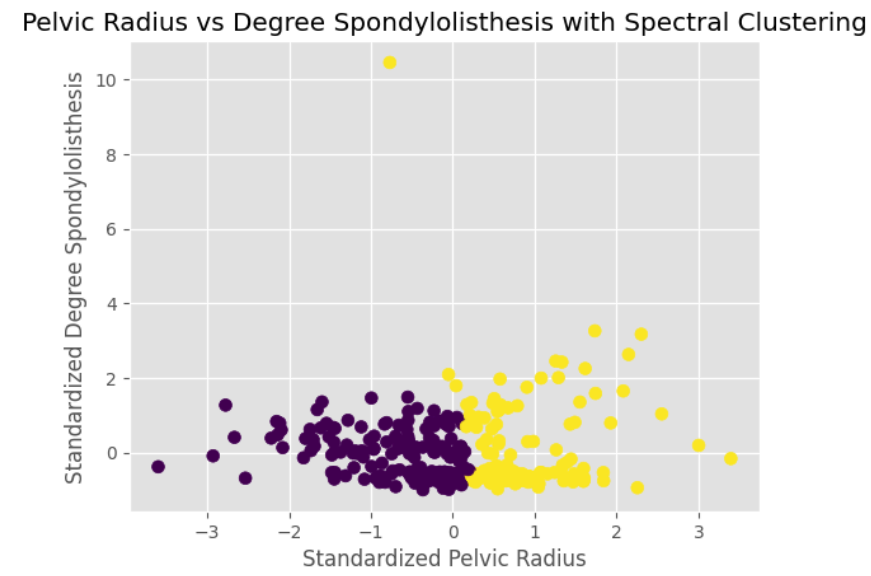



```
# KMeans Clustering
data2 = data.loc[:,['degree_spondylolisthesis','pelvic_radius']]
from sklearn.cluster import KMeans
kmeans = KMeans(n_clusters = 2)
kmeans.fit(data2)
labels = kmeans.predict(data2)
plt.scatter(data['pelvic_radius'],data['degree_spondylolisthesis'],c = labels)
plt.xlabel('pelvic_radius')
plt.xlabel('degree_spondylolisthesis')
plt.show()
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870: FutureWarning
  warnings.warn(
```




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

# Load the dataset
data = pd.read_csv('/content/column_2C_weka.csv')

# Extract features
X = data[['pelvic_radius', 'degree_spondylolisthesis']]

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

# Apply Spectral Clustering
spectral = SpectralClustering(n_clusters=2, affinity='nearest_neighbors', assign_labels='kmeans', random_state=0)
labels = spectral.fit_predict(X_scaled)

# Plot the dataset colored by cluster labels
plt.scatter(X_scaled[:, 0], X_scaled[:, 1], c=labels, s=50, cmap='viridis')
plt.xlabel('Standardized Pelvic Radius')
plt.ylabel('Standardized Degree Spondylolisthesis')
plt.title("Pelvic Radius vs Degree Spondylolisthesis with Spectral Clustering")
plt.show()
```

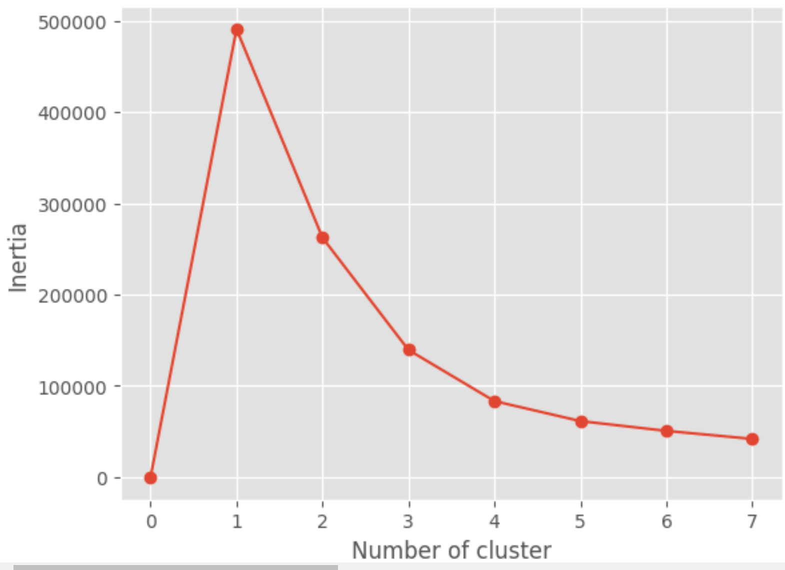


```
# cross tabulation table
df = pd.DataFrame({'labels':labels,"class":data['class']})
ct = pd.crosstab(df['labels'],df['class'])
print(ct)
```

```
class   Abnormal  Normal
labels
0            138     100
1             72       0
```

```
# inertia
inertia_list = np.empty(8)
for i in range(1,8):
    kmeans = KMeans(n_clusters=i)
    kmeans.fit(data2)
    inertia_list[i] = kmeans.inertia_
plt.plot(range(0,8),inertia_list,'-o')
plt.xlabel('Number of cluster')
plt.ylabel('Inertia')
plt.show()
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870: FutureWarning
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870: FutureWarning
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870: FutureWarning
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870: FutureWarning
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870: FutureWarning
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870: FutureWarning
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870: FutureWarning
  warnings.warn(
```



k-means can be slow for large numbers of samples Because each iteration of k-means must access every point in the dataset, the algorithm can be relatively slow as the number of samples grows. You might wonder if this requirement to use all data at each iteration can be relaxed; for example, you might just use a subset of the data to update the cluster centers at each step.

Example 1: k-Means on Digits

```
%matplotlib inline
import matplotlib.pyplot as plt
import seaborn as sns; sns.set()
import numpy as np
from sklearn.cluster import KMeans
```

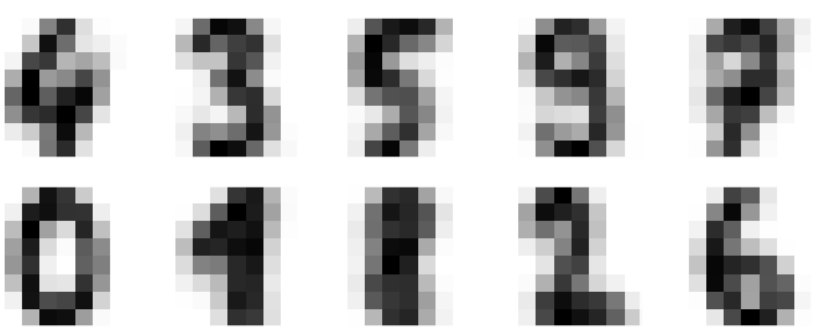
```
from sklearn.datasets import load_digits
digits = load_digits()
digits.data.shape
```

```
(1797, 64)
```

```
kmeans = KMeans(n_clusters=10, random_state=0)
clusters = kmeans.fit_predict(digits.data)
kmeans.cluster_centers_.shape
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870: FutureWarning: The default value of `n_init` will change fro
  warnings.warn(
(10, 64)
```

```
fig, ax = plt.subplots(2, 5, figsize=(8, 3))
centers = kmeans.cluster_centers_.reshape(10, 8, 8)
for axi, center in zip(ax.flat, centers):
   axi.set(xticks=[], yticks=[])
   axi.imshow(center, interpolation='nearest', cmap=plt.cm.binary)
```



The code below will match the learned cluster labels with the actual labels found in them

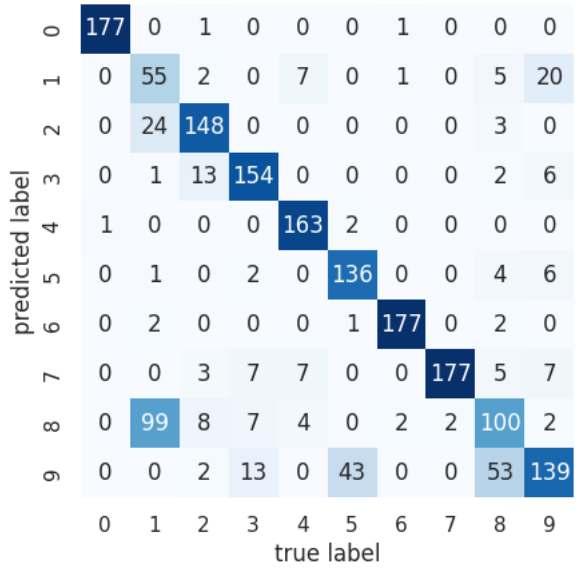
```
from scipy.stats import mode
labels = np.zeros_like(clusters)
for i in range(10):
   mask = (clusters == i)
   labels[mask] = mode(digits.target[mask])[0]
```

```
from sklearn.metrics import accuracy_score
accuracy_score(digits.target, labels)
```

```
0.7935447968836951
```

The above output indicates that the accuracy is roughly 80%

```
from sklearn.metrics import confusion_matrix
import seaborn as sns
mat = confusion_matrix(digits.target, labels)
sns.heatmap(mat.T, square=True, annot=True, fmt='d',
            cbar=False, cmap='Blues',
            xticklabels=digits.target_names,
            yticklabels=digits.target_names)
plt.xlabel('true label')
plt.ylabel('predicted label');
```



t-SNE. t-distributed stochastic neighbor

We can use the t-distributed stochastic neighbor embedding algorithm to preprocess the data before performing k-means. t-SNE is a nonlinear embedding algorithm that is particularly adept at preserving points within clusters

```
from sklearn.manifold import TSNE
# Project the data: this step will take several seconds
tsne = TSNE(n_components=2, init='random',
            learning_rate='auto',random_state=0)
digits_proj = tsne.fit_transform(digits.data)
# Compute the clusters
kmeans = KMeans(n_clusters=10, random_state=0)
clusters = kmeans.fit_predict(digits_proj)
# Permute the labels
labels = np.zeros_like(clusters)
for i in range(10):
    mask = (clusters == i)
    labels[mask] = mode(digits.target[mask])[0]
# Compute the accuracy
accuracy_score(digits.target, labels)
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870: FutureWarning: The default value of `n_init` will change fro
  warnings.warn(
0.9415692821368948
```

That's a 94% classification accuracy without using the labels.

```
import numpy as np
from skimage import io
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans


plt.rcParams['figure.figsize'] = (20, 12)
image = io.imread('/content/penguin.jpg')
labels = plt.axes(xticks=[], yticks=[])
labels.imshow(image);
```

```
rows = image.shape[0]
cols = image.shape[1]

image = image.reshape(rows*cols, 3)
```

```
kmeans = KMeans(n_clusters=64)
kmeans.fit(image)
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870: FutureWarning: The default value of `n_init` will change fro
  warnings.warn(
```

```
KMeans(n_clusters=64)
```

```
compressed_image = kmeans.cluster_centers_[kmeans.labels_]
compressed_image = np.clip(compressed_image.astype('uint8'), 0, 255)


compressed_image = compressed_image.reshape(rows, cols, 3)


io.imsave('compressed_image_4.png', compressed_image)
io.imshow(compressed_image)
io.show()
```

0