```
from google.colab import drive
drive.mount('/content/drive')
```

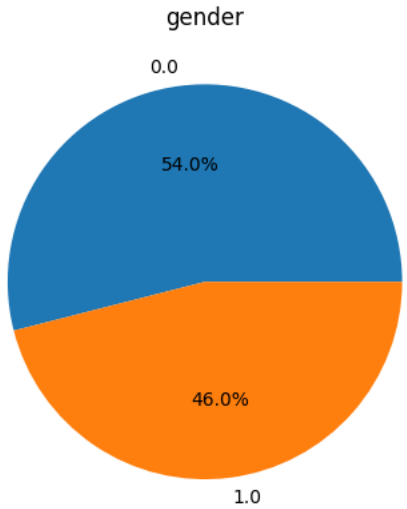
```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True).
```

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

train_data_path = "/content/drive/MyDrive/Data mining/Mini project/train_v1.csv"
test_data_path = "/content/drive/MyDrive/Data mining/Mini project/test_v1.csv"

map = {
    'smoking_status': {'Non-Smoker': 0, 'Smoker': 1, 'Unknown' : 0.25},
    'residence_type': {'Rural': 0, 'Urban': 1}
}

numerical_columns = ['age', 'blood_pressure', 'cholesterol', 'max_heart_rate', 'plasma_glucose', 'skin_thickness', 'insulin', 'bmi', 'diabete
categorical_columns = ['gender', 'chest_pain_type', 'exercise_angina', 'hypertension',  'heart_disease' , 'residence_type', 'smoking_status'


train_df = pd.read_csv(train_data_path, index_col= False)
test_df = pd.read_csv(test_data_path, index_col= False)
```

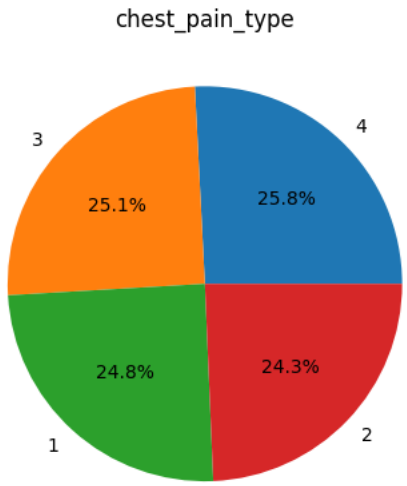
+ Code + Text

```
train_df_original = train_df.copy()
test_df_original = test_df.copy()


train_df = train_df.drop(columns= ['patient_id'])
test_df = test_df.drop(columns= ['patient_id'])


for col in categorical_columns:
        print(f"Column = {col}")
        catcol_counts = train_df[col].value_counts()

        # Plotting the distribution of column as a pie chart
        plt.pie(catcol_counts.values, labels=catcol_counts.index, autopct='%1.1f%%')

        # Adding title to the plot
        plt.title(col)

        # Display the plot
        plt.show()

for col in numerical_columns:
        print(f"Column = {col}")
        plt.hist(train_df[col], bins='auto', edgecolor='black')
        plt.title(col)
        plt.show()
```

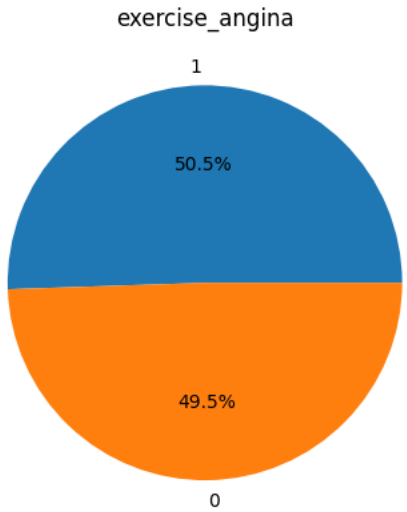
```
Column = gender
```



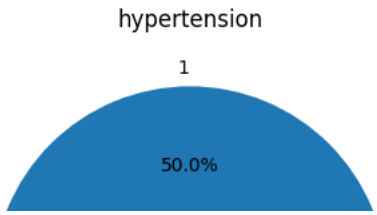

```
Column = chest_pain_type
```

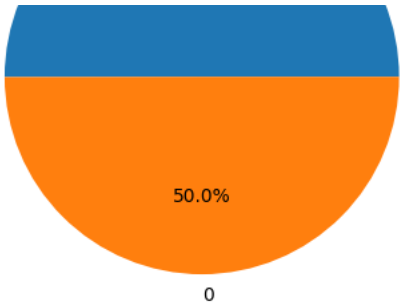



```
Column = exercise_angina
```

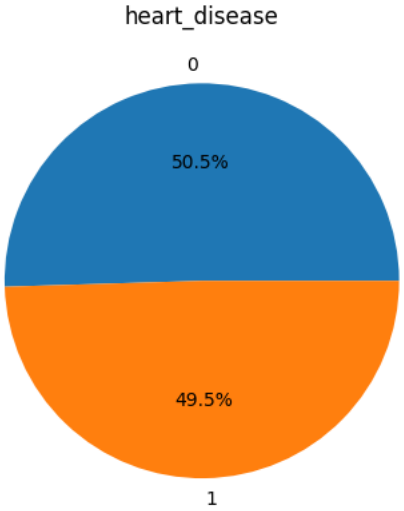



```
Column = hypertension
```

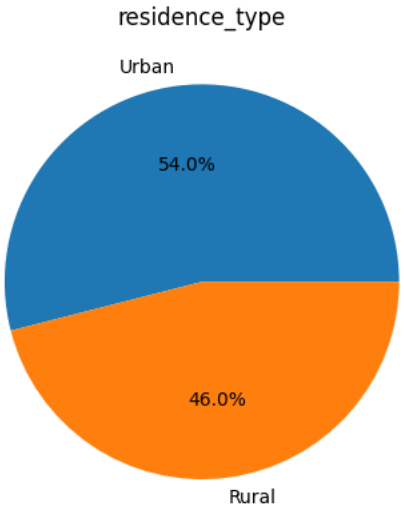




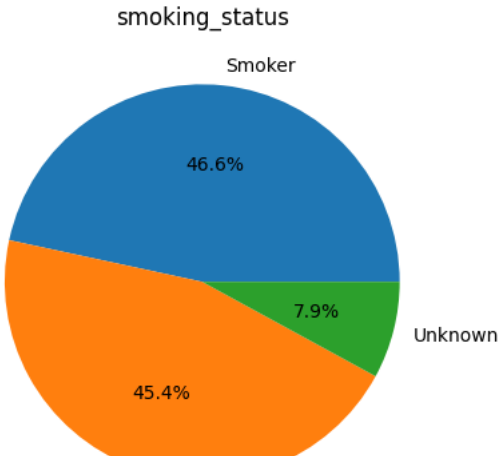

```
Column = heart_disease
```



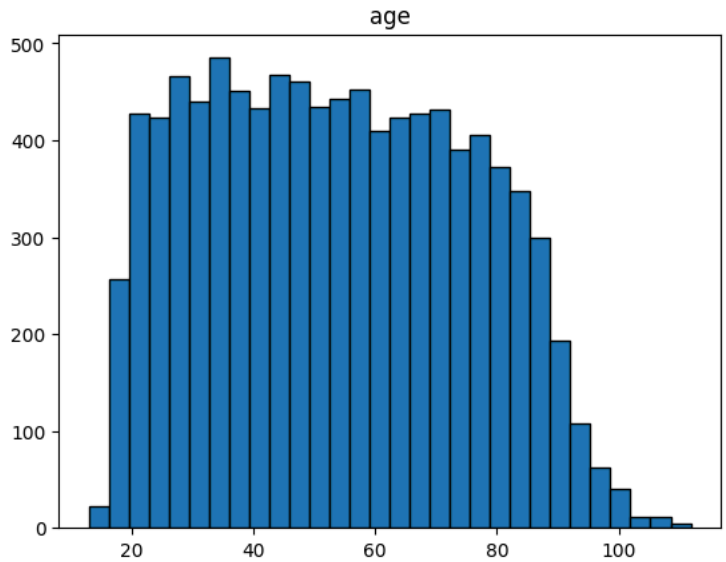

```
Column = residence_type
```



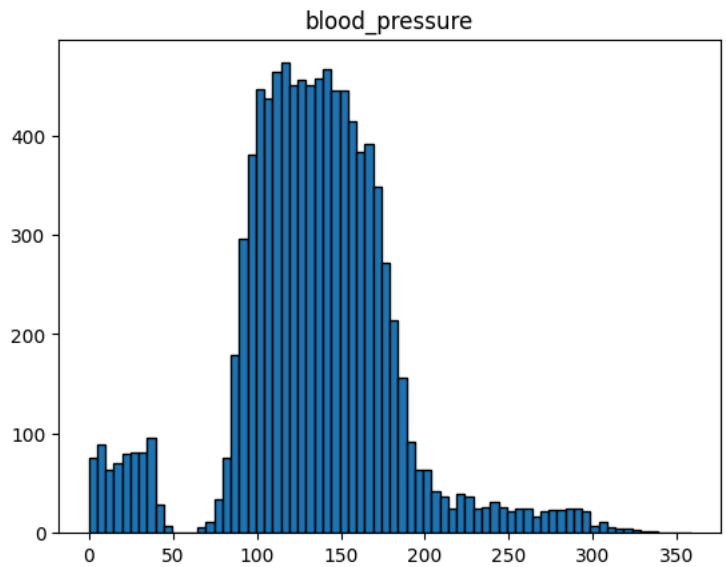

```
Column = smoking_status
```



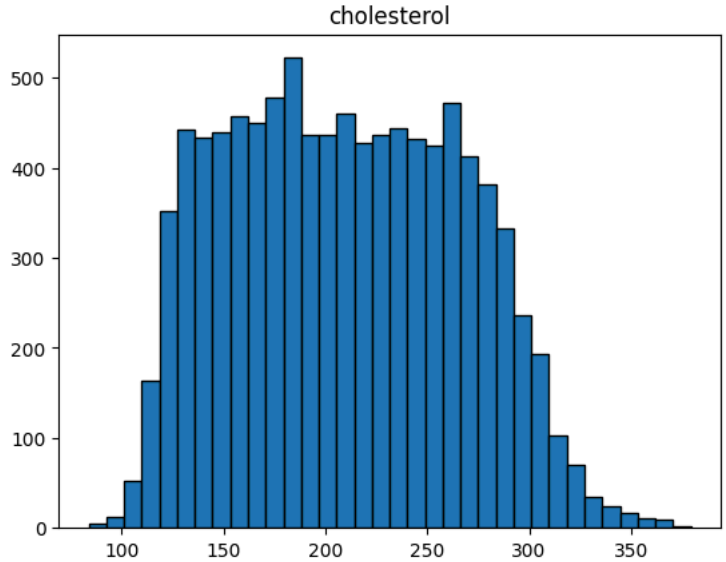

```
Column = age
```



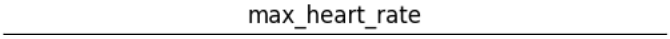

```
Column = blood_pressure
```

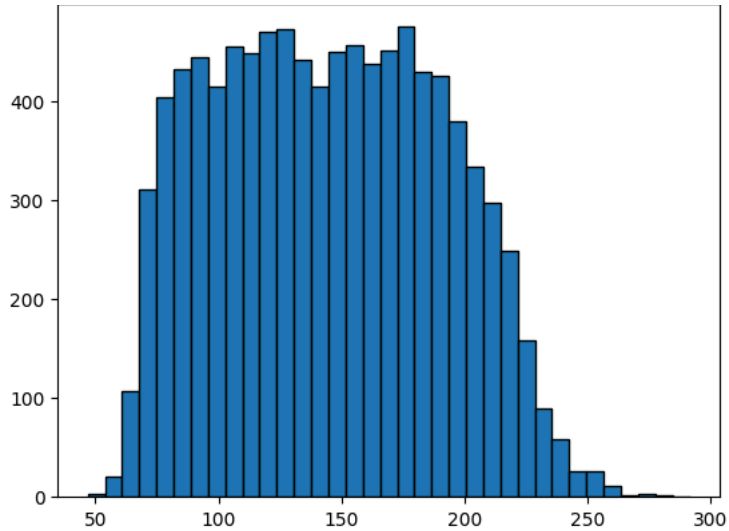



```
Column = cholesterol
```

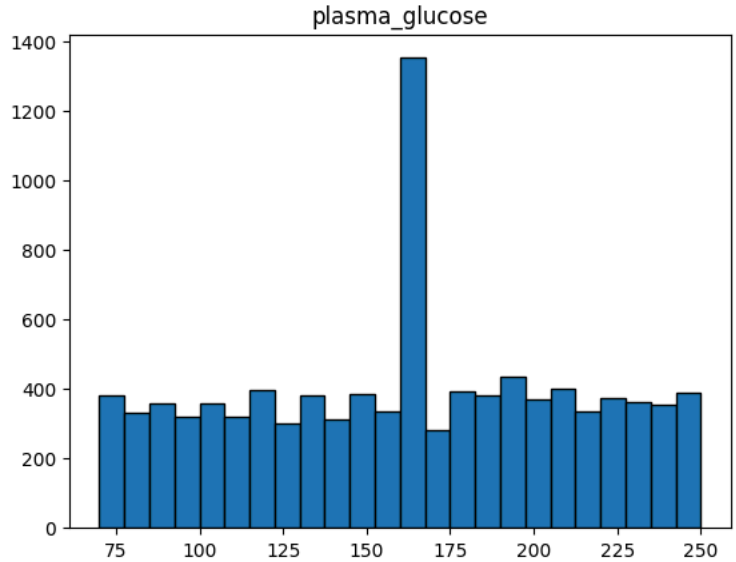



```
Column = max_heart_rate
```

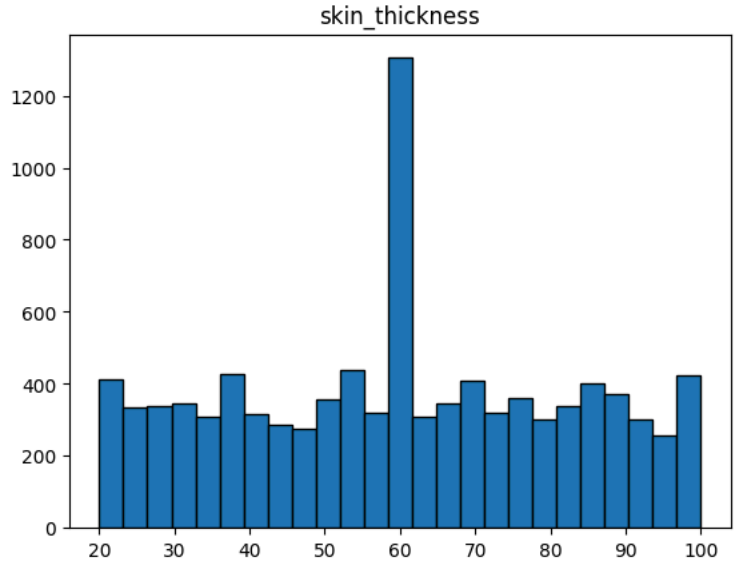




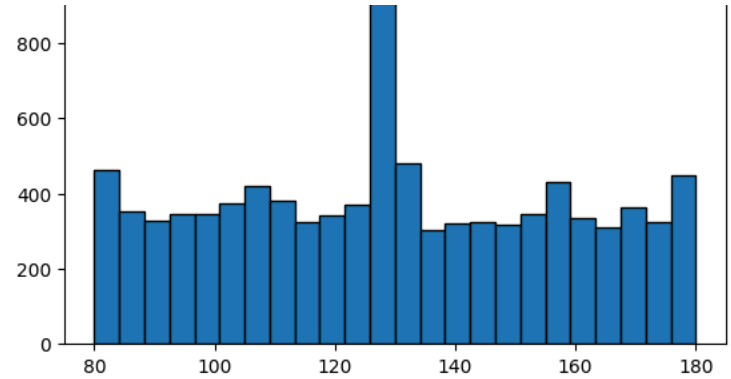

```
Column = plasma_glucose
```



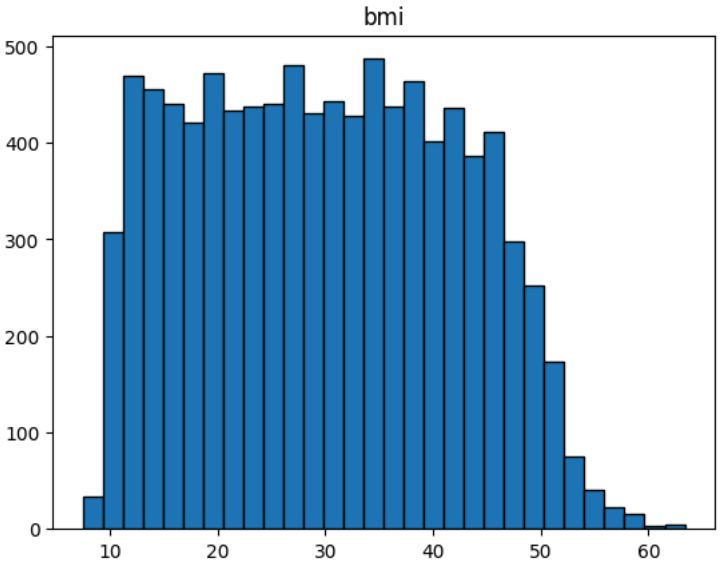

```
Column = skin_thickness
```



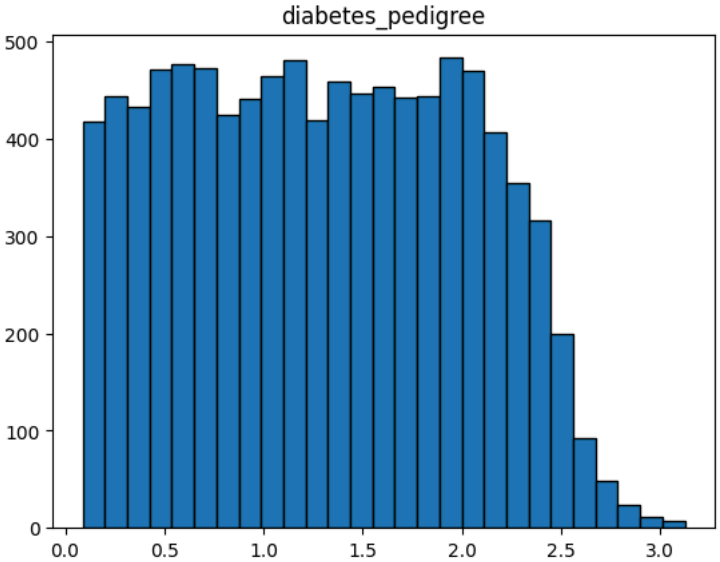

```
Column = insulin
```




```
Column = bmi
```




```
Column = diabetes_pedigree
```



```
# train_df = train_df.drop(train_df[train_df['smoking_status'] == 'Unknown'].index)


train_df = train_df.replace(map)
test_df = test_df.replace(map)
```

```
<ipython-input-36-0d416d399f1f>:1: FutureWarning:

Downcasting behavior in `replace` is deprecated and will be removed in a future version. To retain the old behavior, explicitly call `re

<ipython-input-36-0d416d399f1f>:2: FutureWarning:

Downcasting behavior in `replace` is deprecated and will be removed in a future version. To retain the old behavior, explicitly call `re
```

```
train_df.head()
test_df.head()
```

	age	gender	chest_pain_type	blood_pressure	cholesterol	max_heart_rate	exercise_angina	plasma_glucose	skin_thickness	insu
0	27.874177	1.0	2	96.527038	189.352193	170.874313	0	182.0	57.0	8
1	49.094664	0.0	4	12.474058	138.573618	139.785337	1	103.0	21.0	1
2	36.515462	1.0	3	169.357937	177.247023	108.974898	0	71.0	46.0	1
3	55.152150	0.0	1	106.749584	276.231050	72.084170	1	141.0	33.0	1
4	46.860684	1.0	4	148.547370	204.798557	182.933404	0	75.0	66.0	1

Next steps: Generate code with test_df | View recommended plots | New interactive sheet

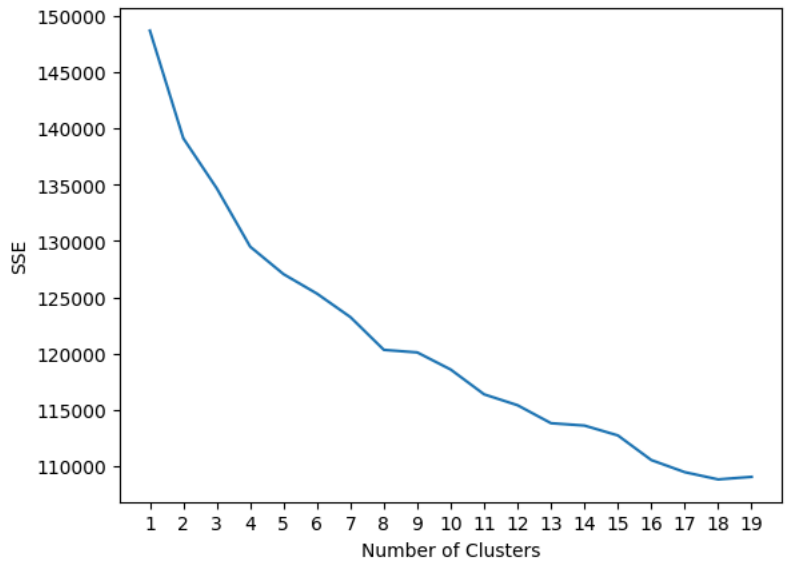
```
train_df['smoking_status'].value_counts()
```

smoking_status	count
1.00	4475
0.00	4363
0.25	762

dtype: int64

```
from sklearn.preprocessing import StandardScaler, MinMaxScaler


standard_scaler = StandardScaler()
minmax_scaler = MinMaxScaler(feature_range=(-1, 1))


train_df[numerical_columns] = standard_scaler.fit_transform(train_df[numerical_columns])
test_df[numerical_columns] = standard_scaler.transform(test_df[numerical_columns])

train_df[categorical_columns] = minmax_scaler.fit_transform(train_df[categorical_columns])
test_df[categorical_columns] = minmax_scaler.transform(test_df[categorical_columns])


from sklearn.cluster import KMeans

#initialize kmeans parameters
kmeans_kwargs = {
"init": "k-means++",
"n_init": 'auto',
"random_state": 42,
}

#create list to hold SSE values for each k
sse = []
for k in range(1, 20):
    kmeans = KMeans(n_clusters=k, **kmeans_kwargs)
    kmeans.fit(train_df)
    sse.append(kmeans.inertia_)

#visualize results
```

```
plt.plot(range(1, 20), sse)
plt.xticks(range(1, 20))
plt.xlabel("Number of Clusters")
plt.ylabel("SSE")
plt.show()
```




```
#instantiate the k-means class, using optimal number of clusters
kmeans = KMeans(init="k-means++", n_clusters=4, n_init='auto', random_state=42)

#fit k-means algorithm to data
kmeans.fit(train_df)

#view cluster assignments for each observation
predictions = kmeans.labels_
```

Generate | compute Silhouette Score Davies-Bouldin Index: | Close

1 of 1 | Undo Changes | Use code with caution

```
from sklearn.metrics import silhouette_score, davies_bouldin_score

# Assuming 'predictions' contains the cluster labels assigned by KMeans
# and 'train_df' is your dataframe used for clustering

silhouette_avg = silhouette_score(train_df, predictions)
davies_bouldin_index = davies_bouldin_score(train_df, predictions)

print(f"Silhouette Score: {silhouette_avg}")
print(f"Davies-Bouldin Index: {davies_bouldin_index}")
```

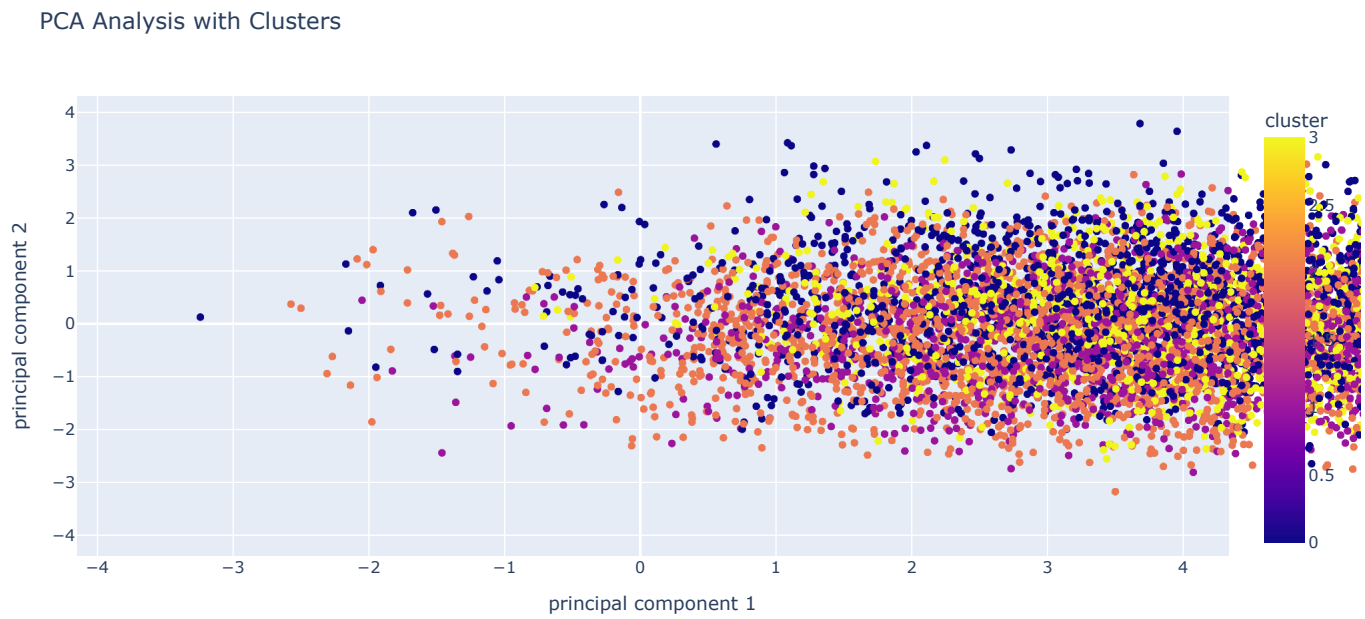
```
Silhouette Score: 0.06853460769906784
Davies-Bouldin Index: 3.635546667878676
```

Generate | apply PCA analysis with plots | Close

1 of 1 | Use code with caution

```
# prompt: apply PCA analysis with plots

import pandas as pd
import plotly.express as px
from sklearn.decomposition import PCA

# Apply PCA with 2 components
pca = PCA(n_components=2)
principalComponents = pca.fit_transform(train_df)
principalDf = pd.DataFrame(data = principalComponents, columns = ['principal component 1', 'principal component 2'])
```

```
# Add cluster labels to the principal component dataframe
principalDf['cluster'] = predictions

# Create a scatter plot using plotly express
fig = px.scatter(principalDf, x="principal component 1", y="principal component 2", color="cluster", title="PCA Analysis with Clusters")
fig.show()

# Explained variance ratio
explained_variance = pca.explained_variance_ratio_
print(f"Explained variance ratio: {explained_variance}")
```




```
Explained variance ratio: [0.07004619 0.0692001 ]
```

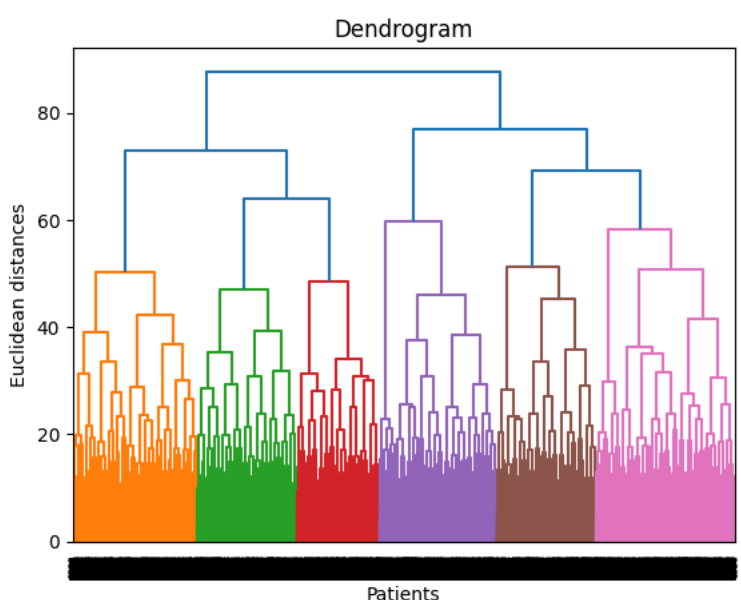
```
result = kmeans.predict(test_df)
```

```
df_final = test_df_original.copy()
df_final['cluster_label'] = result
df_final = df_final[['patient_id' , 'cluster_label']]
df_final.to_csv('cluster_out.csv',index= False)
```

Generate | use hierarchical clustering | Close

1 of 1 | Undo Changes | Use code with caution

```
# prompt: use hierarchical clustering

import matplotlib.pyplot as plt
from sklearn.cluster import AgglomerativeClustering
import scipy.cluster.hierarchy as sch

# Create dendrogram
dendrogram = sch.dendrogram(sch.linkage(train_df, method='ward'))
plt.title('Dendrogram')
plt.xlabel('Patients')
plt.ylabel('Euclidean distances')
plt.show()

# Fit hierarchical clustering model
# Removed the 'affinity' argument as it is not needed for 'ward' linkage
agg_clustering = AgglomerativeClustering(n_clusters=4, linkage='ward')
agg_clustering.fit(train_df)

# View cluster assignments for each observation
hierarchical_predictions = agg_clustering.labels_
```




```
from sklearn.metrics import silhouette_score, davies_bouldin_score

# Assuming 'predictions' contains the cluster labels assigned by KMeans
# and 'train_df' is your dataframe used for clustering

silhouette_avg = silhouette_score(train_df, hierarchical_predictions)
davies_bouldin_index = davies_bouldin_score(train_df, hierarchical_predictions)

print(f"Silhouette Score: {silhouette_avg}")
print(f"Davies-Bouldin Index: {davies_bouldin_index}")
```

```
Silhouette Score: 0.017609446291135016
Davies-Bouldin Index: 4.948187122843915
```

```
# prompt: predict using test_df with agg_clustering

# Predict using test_df with agg_clustering
hierarchical_result = agg_clustering.fit_predict(test_df)

df_final_hierarchical = test_df_original.copy()
df_final_hierarchical['cluster_label'] = hierarchical_result
df_final_hierarchical = df_final_hierarchical[['patient_id', 'cluster_label']]
df_final_hierarchical.to_csv('hierarchical_cluster_out.csv', index=False)
```

Generate | use autoencoder clustering and use that model to testing | Close

1 of 1 | Undo Changes | Use code with caution

```
# prompt: use autoencoder clustering and use that model to testing

import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers
from sklearn.model_selection import train_test_split
from sklearn.metrics import silhouette_score

# Assuming 'train_df' is your preprocessed dataframe
X_train, X_test = train_test_split(train_df, test_size=0.2, random_state=42)

# Define the autoencoder model
latent_dim = 2  # Dimension of the latent space

input_layer = keras.Input(shape=(X_train.shape[1],))
encoded = layers.Dense(128, activation='relu')(input_layer)
encoded = layers.Dense(64, activation='relu')(encoded)
encoded = layers.Dense(latent_dim, activation='relu')(encoded)
decoded = layers.Dense(64, activation='relu')(encoded)
decoded = layers.Dense(128, activation='relu')(decoded)
decoded = layers.Dense(X_train.shape[1], activation='sigmoid')(decoded)
```

```
autoencoder = keras.Model(input_layer, decoded)
encoder = keras.Model(input_layer, encoded)

# Compile the autoencoder
autoencoder.compile(optimizer='adam', loss='mse')

# Train the autoencoder
autoencoder.fit(X_train, X_train, epochs=50, batch_size=32, shuffle=True, validation_data=(X_test, X_test))

# Get the encoded representations (latent space)
encoded_data = encoder.predict(train_df)
```

```
Epoch 18/50
240/240 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - loss: 0.8261 - val_loss: 0.8404
Epoch 19/50
240/240 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - loss: 0.8273 - val_loss: 0.8414
Epoch 20/50
240/240 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - loss: 0.8284 - val_loss: 0.8414
Epoch 21/50
240/240 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - loss: 0.8259 - val_loss: 0.8413
Epoch 22/50
240/240 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.8294 - val_loss: 0.8404
Epoch 23/50
240/240 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - loss: 0.8264 - val_loss: 0.8402
Epoch 24/50
240/240 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.8275 - val_loss: 0.8392
Epoch 25/50
240/240 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - loss: 0.8284 - val_loss: 0.8414
Epoch 26/50
240/240 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - loss: 0.8245 - val_loss: 0.8394
Epoch 27/50
240/240 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - loss: 0.8275 - val_loss: 0.8402
Epoch 28/50
240/240 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - loss: 0.8263 - val_loss: 0.8415
Epoch 29/50
240/240 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - loss: 0.8224 - val_loss: 0.8376
Epoch 30/50
240/240 ━━━━━━━━━━━━━━━━━━━━ 2s 5ms/step - loss: 0.8228 - val_loss: 0.8386
Epoch 31/50
240/240 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - loss: 0.8211 - val_loss: 0.8388
Epoch 32/50
240/240 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.8281 - val_loss: 0.8374
Epoch 33/50
240/240 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - loss: 0.8224 - val_loss: 0.8378
Epoch 34/50
240/240 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - loss: 0.8302 - val_loss: 0.8385
Epoch 35/50
240/240 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - loss: 0.8259 - val_loss: 0.8370
Epoch 36/50
240/240 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.8240 - val_loss: 0.8363
Epoch 37/50
240/240 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - loss: 0.8265 - val_loss: 0.8366
Epoch 38/50
240/240 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.8223 - val_loss: 0.8385
Epoch 39/50
240/240 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.8187 - val_loss: 0.8370
Epoch 40/50
240/240 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.8214 - val_loss: 0.8364
Epoch 41/50
240/240 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - loss: 0.8183 - val_loss: 0.8364
Epoch 42/50
240/240 ━━━━━━━━━━━━━━━━━━━━ 2s 4ms/step - loss: 0.8208 - val_loss: 0.8355
Epoch 43/50
240/240 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - loss: 0.8231 - val_loss: 0.8372
Epoch 44/50
240/240 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - loss: 0.8186 - val_loss: 0.8360
Epoch 45/50
240/240 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - loss: 0.8210 - val_loss: 0.8359
Epoch 46/50
240/240 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - loss: 0.8172 - val_loss: 0.8358
Epoch 47/50
```

```
# Perform clustering on the encoded data (e.g., using KMeans)
kmeans = KMeans(n_clusters=4, random_state=42)
kmeans.fit(encoded_data)
cluster_labels = kmeans.labels_

# Evaluate the clustering using silhouette score
silhouette_avg = silhouette_score(encoded_data, cluster_labels)
```

```
davies_bouldin_index = davies_bouldin_score(encoded_data, cluster_labels)
print(f"Silhouette Score: {silhouette_avg}")
print(f"Davies-Bouldin Index: {davies_bouldin_index}")

# Predict cluster labels for the test data
encoded_test_data = encoder.predict(test_df)
test_cluster_labels = kmeans.predict(encoded_test_data)

# Add cluster labels to your test dataframe
test_df_original['cluster_label'] = test_cluster_labels

# Save the results
test_df_original[['patient_id', 'cluster_label']].to_csv('autoencoder_cluster_out.csv', index=False)
```

```
Silhouette Score: 0.6165102124214172
Davies-Bouldin Index: 0.6983583679075803
75/75 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step
```

Generate | plot the PCA analysis for the encoded clustering | Close

1 of 1 Use code with caution

```
# prompt: plot the PCA analysis for the encoded clustering

import pandas as pd
# Assuming 'encoded_data' contains the encoded data from the autoencoder
# and 'cluster_labels' contains the cluster assignments

# Apply PCA with 2 components
pca = PCA(n_components=2)
principalComponents = pca.fit_transform(encoded_data)
principalDf = pd.DataFrame(data = principalComponents, columns = ['principal component 1', 'principal component 2'])

# Add cluster labels to the principal component dataframe
principalDf['cluster'] = cluster_labels

# Create a scatter plot using plotly express
fig = px.scatter(principalDf, x="principal component 1", y="principal component 2", color="cluster", title="PCA Analysis of Autoencoder Enco
fig.show()

# Explained variance ratio
explained_variance = pca.explained_variance_ratio_
print(f"Explained variance ratio: {explained_variance}")
```


```
Explained variance ratio: [0.7733834  0.22661667]
```

```
from sklearn.cluster import DBSCAN
from sklearn.metrics import silhouette_score

eps_values = np.arange(0.1, 2.0, 0.1)
best_score = -1
best_eps = None

for eps in eps_values:
    dbscan = DBSCAN(eps=eps, min_samples=4)
    labels = dbscan.fit_predict(train_df)
    # Ignore clustering where all points are noise or 1 cluster
    if len(set(labels)) > 1 and -1 in labels:
        score = silhouette_score(train_df, labels)
        if score > best_score:
            best_score = score
            best_eps = eps

print(f"Best eps: {best_eps} with Silhouette Score: {best_score:.3f}")
```

```
Best eps: 1.0 with Silhouette Score: -0.026
```

```
for min_samples in [3, 5, 10, 15]:
    dbscan = DBSCAN(eps=best_eps, min_samples=min_samples)
    labels = dbscan.fit_predict(train_df)
    if len(set(labels)) > 1 and -1 in labels:
        score = silhouette_score(train_df, labels)
        print(f"min_samples = {min_samples}, Silhouette Score = {score:.3f}")
```

```
min_samples = 3, Silhouette Score = -0.111
```

Generate | use DBSCAN | Close

1 of 1 | Undo Changes | Use code with caution

```
# prompt: use DBSCAN

from sklearn.cluster import DBSCAN

dbscan = DBSCAN(eps=1.0, min_samples=3)

# Fit DBSCAN to your data
dbscan.fit(train_df)

# Get the cluster labels assigned by DBSCAN
dbscan_labels = dbscan.labels_

# Evaluate the clustering using silhouette score (if applicable)
# Note that silhouette score may not be ideal for DBSCAN as it assumes clusters are well-separated
if len(set(dbscan_labels)) > 1:  # Ensure there are multiple clusters
  silhouette_avg = silhouette_score(train_df, dbscan_labels)
  print(f"Silhouette Score: {silhouette_avg}")

# Predict cluster labels for the test data
dbscan_test_labels = dbscan.fit_predict(test_df)

# Add cluster labels to your test dataframe
test_df_original['cluster_label'] = dbscan_test_labels

# Save the results
test_df_original[['patient_id', 'cluster_label']].to_csv('dbscan_cluster_out.csv', index=False)
```

```
Silhouette Score: -0.11065802697724521
```

Generate | plot the PCA analysis for the DBSCAN clustering | Close

1 of 1 | Use code with caution

Generated code may be subject to a license | oskwys/scGeneRAI

```
# prompt: plot the PCA analysis for the DBSCAN clustering
```

```
import pandas as pd
# Apply PCA with 2 components
pca = PCA(n_components=2)
principalComponents = pca.fit_transform(train_df)
principalDf = pd.DataFrame(data = principalComponents, columns = ['principal component 1', 'principal component 2'])

# Add cluster labels to the principal component dataframe
principalDf['cluster'] = dbscan_labels

# Create a scatter plot using plotly express
fig = px.scatter(principalDf, x="principal component 1", y="principal component 2", color="cluster", title="PCA Analysis of DBSCAN Clusters")
fig.show()

# Explained variance ratio
explained_variance = pca.explained_variance_ratio_
print(f"Explained variance ratio: {explained_variance}")
```