```
import pandas as pd
import sys
sys.setrecursionlimit(10000)
import warnings
warnings.filterwarnings("ignore")
```

```
#load data sets

data = pd.read_csv('data.csv', header=None)
labels = pd.read_csv('labels.csv', header=None)
```

```
# rename first column in both data sets as 'sample'
data = data.rename(columns={0: 'sample_id'})
labels = labels.rename(columns={0: 'sample_id'})
```

```
print('Data shape:', data.shape)
print('Label shape:', labels.shape)
```

```
Data shape: (802, 20532)
Label shape: (802, 2)
```

```
data.head(5)
```

	sample_id	1	2	3	4	5	6
0	NaN	gene_0	gene_1	gene_2	gene_3	gene_4	gene_5
1	sample_0	0.0	2.01720929003	3.26552691165	5.47848651208	10.4319989607	0.0
2	sample_1	0.0	0.592732094867	1.58842082049	7.58615673813	9.62301085621	0.0
3	sample_2	0.0	3.5117589779	4.32719871937	6.88178695937	9.87072997113	0.0
4	sample_3	0.0	3.66361787431	4.50764877794	6.65906827484	10.1961840717	0.0

5 rows × 20532 columns

```
data.isnull().sum()
```

```
sample_id    1
1            0
2            0
3            0
4            0
            ..
20527        0
```

```
20528        0
20529        0
20530        0
20531        0
Length: 20532, dtype: int64
```

```
labels.head(5)
```

	sample_id	1
0	NaN	Class
1	sample_0	PRAD
2	sample_1	LUAD
3	sample_2	PRAD
4	sample_3	PRAD

```
labels.value_counts()
```

```
sample_id   1
sample_0    PRAD    1
sample_583  KIRC    1
sample_574  BRCA    1
sample_575  KIRC    1
sample_576  LUAD    1
                   ..
sample_341  BRCA    1
sample_342  BRCA    1
sample_343  LUAD    1
sample_344  LUAD    1
sample_99   BRCA    1
Length: 801, dtype: int64
```

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 802 entries, 0 to 801
Columns: 20532 entries, sample_id to 20531
dtypes: object(20532)
memory usage: 125.6+ MB
```

```
data.describe()
```

	sample_id	1	2	3	4	5	6	7	8	
count	801	802.0	802.0	802.0	802.000000	802.000000	802.0	802.000000	802.0	8
unique	801	38.0	772.0	794.0	801.000000	800.000000	3.0	801.000000	490.0	
top	sample_0	0.0	0.0	0.0	6.071696	9.992089	0.0	8.225689	0.0	
freq	1	736.0	31.0	7.0	2.000000	2.000000	770.0	2.000000	291.0	7

```
labels.describe()
```

	sample_id	1
count	801	802
unique	801	6
top	sample_0	BRCA
freq	1	300

```
data.head()
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 802 entries, 0 to 801
Columns: 20532 entries, sample_id to 20531
dtypes: object(20532)
memory usage: 125.6+ MB
```

```
labels.head()
labels.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 802 entries, 0 to 801
Data columns (total 2 columns):
 #   Column     Non-Null Count  Dtype
---  ------     --------------  -----
 0   sample_id  801 non-null    object
 1   1          802 non-null    object
dtypes: object(2)
memory usage: 12.7+ KB
```

```
labels.columns
```

```
Index(['sample_id', 1], dtype='object')
```

```
# save renamed data sets
data.to_csv('data_renamed.csv', index=False)
labels.to_csv('labels_renamed.csv', index=False)
```

```
data = pd.read_csv('data_renamed.csv')
labels = pd.read_csv('labels_renamed.csv')

# save the data to a new file

merged_data = pd.merge(data, labels, on='sample_id')
merged_data.to_csv('merged_data.csv', index=False)
```

```
merged_data.columns
```

```
Index(['sample_id', '1_x', '2', '3', '4', '5', '6', '7', '8', '9',
       ...
       '20523', '20524', '20525', '20526', '20527', '20528', '20529', '20530',
       '20531', '1_y'],
      dtype='object', length=20533)
```

```
## Plot the merged dataset as a hierarchically-clustered heatmap
```

```
import seaborn as sns
import matplotlib.pyplot as plt

# load merged data set
merged_data = pd.read_csv('merged_data.csv')

merged_data.head()
```

	sample_id	1_x	2	3	4	5	6
0	NaN	gene_0	gene_1	gene_2	gene_3	gene_4	gene_5
1	sample_0	0.0	2.01720929003	3.26552691165	5.47848651208	10.4319989607	0.0
2	sample_1	0.0	0.592732094867	1.58842082049	7.58615673813	9.62301085621	0.0
3	sample_2	0.0	3.5117589779	4.32719871937	6.88178695937	9.87072997113	0.0
4	sample_3	0.0	3.66361787431	4.50764877794	6.65906827484	10.1961840717	0.0

5 rows × 20533 columns

```
#load data sets
data = pd.read_csv('data.csv', header=None)
labels = pd.read_csv('labels.csv', header=None)

# rename first column in both data sets as 'sample'
```

```
data = data.rename(columns={0: 'sample'})
labels = labels.rename(columns={0: 'sample'})

# merge data sets
merged_data = pd.merge(data, labels, on='sample')

# extract gene expression data
gene_expression = merged_data.iloc[1:, 1:-1].astype(float)

# create a clustered heatma
plt.figure(figsize=(400, 400)) # set figure size

sns.clustermap(gene_expression, cmap='viridis') #, figsize=(20, 20))
plt.show()
```

```
<Figure size 28800x28800 with 0 Axes>
```


```
# The graph shows the levels of gene expression for a set of genes across a set of samples. E
# the level of expression of each gene in each sample, with red representing high expression
# which groups together genes or samples that have similar patterns of expression.
# By looking at the clustered heatmap, we can see which genes are co-expressed across samples
#that may be involved in similar biological processes or for identifying subtypes of samples
```

```
import scipy.stats as stats
```

```
# separate the gene expression data for each group based on the label column
group1_expression = merged_data.loc[merged_data['sample_id'] == 'group1'].iloc[:, 1:-1].astyp
group2_expression = merged_data.loc[merged_data['sample_id'] == 'group2'].iloc[:, 1:-1].astyp
```

```
print(merged_data.columns)
```

```
Index(['sample_id', 'class', 'gene_0', 'gene_1', 'gene_2', 'gene_3', 'gene_4',
       'gene_5', 'gene_6', 'gene_7',
       ...
       'gene_20521', 'gene_20522', 'gene_20523', 'gene_20524', 'gene_20525',
       'gene_20526', 'gene_20527', 'gene_20528', 'gene_20529', 'gene_20530'],
      dtype='object', length=20533)
```


```
# calculate the means and standard deviations of each group
group1_mean = group1_expression.mean(axis=0)
group2_mean = group2_expression.mean(axis=0)
group1_std = group1_expression.std(axis=0)
group2_std = group2_expression.std(axis=0)
```

```
# perform a two-sample t-test with equal variances assumed
t_stat, p_val = stats.ttest_ind(group1_expression, group2_expression, equal_var=True)


# print the results
print(f"T-statistic: {t_stat}")
print(f"P-value: {p_val}")
```

```
T-statistic: [nan nan nan ... nan nan nan]
P-value: [nan nan nan ... nan nan nan]
```

```
print(merged_data.columns)
```

```
Index(['sample',    '1_x',        2,        3,        4,        5,        6,
             7,        8,        9,
       ...
          20523,    20524,    20525,    20526,    20527,    20528,    20529,
          20530,    20531,    '1_y'],
      dtype='object', length=20533)
```

```
print(gene_expression.columns)
```

```
Index(['1_x',     2,     3,     4,     5,     6,     7,     8,     9,    10,
       ...
       20522, 20523, 20524, 20525, 20526, 20527, 20528, 20529, 20530, 20531],
      dtype='object', length=20531)
```

```
import pandas as pd

# Load the data from merged_data.csv
df = pd.read_csv('merged_data.csv')

# Drop the first row
df = df.drop(0)

# Rename the first column to "sample_id"
df = df.rename(columns={df.columns[0]: "sample_id"})

# Set the "sample_id" column as the index
df = df.set_index("sample_id")

# Show the modified dataframe
print(df.head())
```

```
           1_x               2              3              4              5  \
sample_id
sample_0   0.0   2.01720929003  3.26552691165  5.47848651208  10.4319989607
sample_1   0.0  0.592732094867  1.58842082049  7.58615673813  9.62301085621
sample_2   0.0    3.5117589779  4.32719871937  6.88178695937  9.87072997113
sample_3   0.0   3.66361787431  4.50764877794  6.65906827484  10.1961840717
sample_4   0.0   2.65574107476  2.82154695883  6.53945352515  9.73826456185

             6              7               8    9   10  ...          20523  \
sample_id                                                ...
sample_0   0.0  7.17517526213  0.591870870063  0.0  0.0  ...  8.21025734657
sample_1   0.0  6.81604924768             0.0  0.0  0.0  ...    7.323865415
sample_2   0.0  6.97212970934  0.452595434703  0.0  0.0  ...  8.12712250994
sample_3   0.0  7.84337463893  0.434881719407  0.0  0.0  ...  8.79295943916
sample_4   0.0  6.56696732901  0.360982241369  0.0  0.0  ...  8.89142526287

                   20524          20525          20526          20527  \
sample_id
sample_0   9.72351589977  7.22003000722  9.11981265388   12.003134876
sample_1   9.74093093236  6.25658612273  8.38161216428  12.6745520141
```

```
sample_2   10.9086403047  5.40160657619  9.91159721647  9.04525456146
sample_3   10.1415196459  8.94280477453  9.60120808332  11.3926823441
sample_4   10.3737895683  7.18116219788  9.84691009283   11.922439457

                   20528          20529          20530 20531   1_y
sample_id
sample_0   9.65074302191  8.92132623446  5.28675919351   0.0  PRAD
sample_1   10.5170591152  9.39785429023  2.09416849472   0.0  LUAD
sample_2   9.78835944927  10.0904697402  1.68302266506   0.0  PRAD
sample_3   9.69481404455  9.68436466871  3.29200130514   0.0  PRAD
sample_4   9.21774933391  9.46119087942  5.11037159613   0.0  BRCA

[5 rows x 20532 columns]
```

```
import pandas as pd

# read the csv file
df = pd.read_csv('merged_data.csv')
# drop the first row
df = df.drop(0)

# set the sample_id column as the index
df = df.set_index('sample_id')

# select only the columns with gene expression values
df = df.iloc[:, 1:]

# transpose the dataframe
df = df.transpose()

# print the first few rows
print(df.head())
```

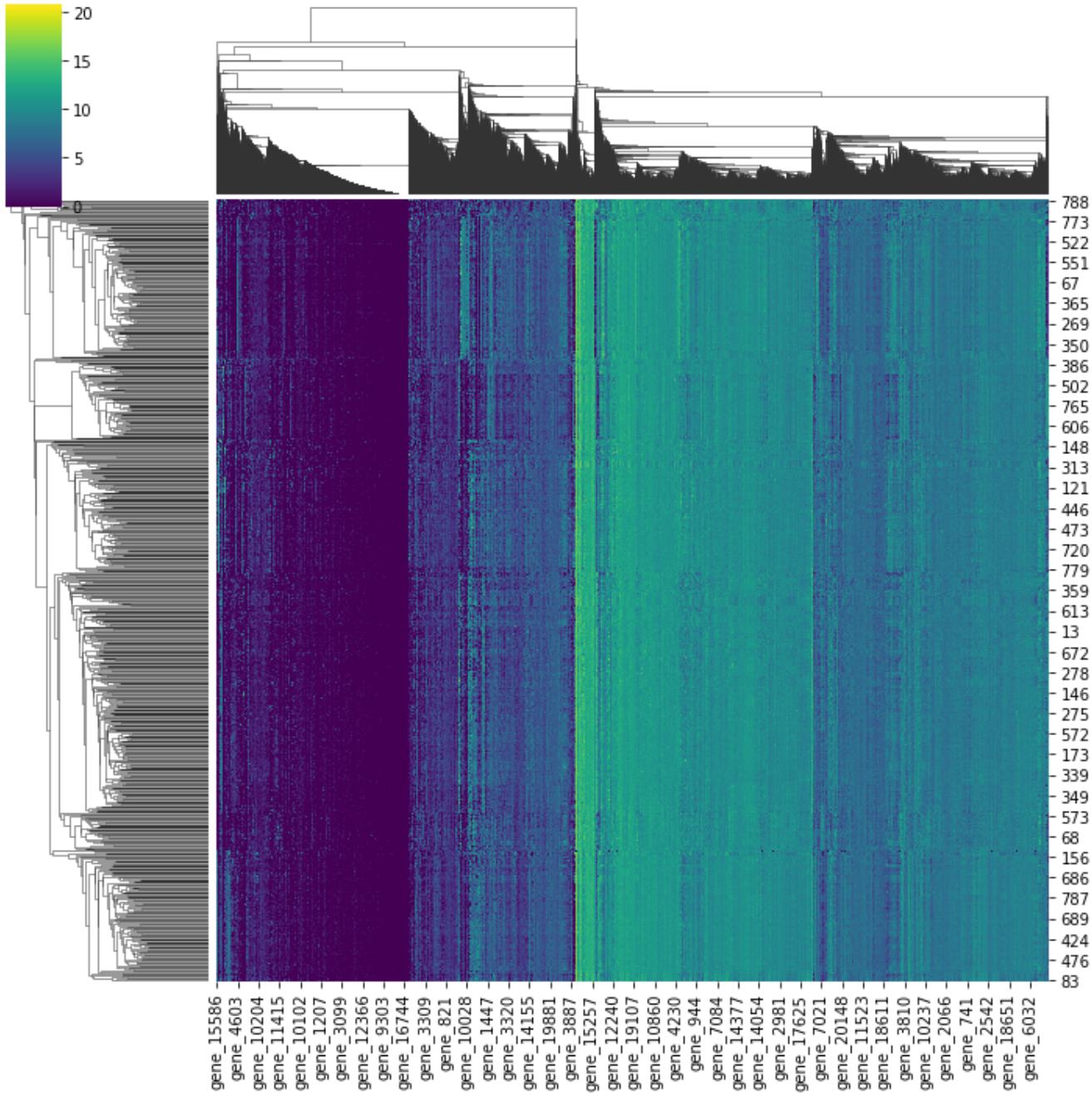
```
sample_id       sample_0        sample_1       sample_2       sample_3  \
2          2.01720929003  0.592732094867   3.5117589779  3.66361787431
3          3.26552691165   1.58842082049  4.32719871937  4.50764877794
4          5.47848651208   7.58615673813  6.88178695937  6.65906827484
5          10.4319989607   9.62301085621  9.87072997113  10.1961840717
6                    0.0             0.0            0.0            0.0

sample_id       sample_4       sample_5       sample_6       sample_7  \
2          2.65574107476  3.46785331372    1.224966365  2.85485342652
3          2.82154695883  3.58191760772  1.69117679681  1.75047787844
4          6.53945352515  6.62024328973  6.57200741498  7.22672044861
5          9.73826456185  9.70682924127  9.64051067136  9.75869126501
6                    0.0            0.0            0.0            0.0

sample_id       sample_8       sample_9  ... sample_791 sample_792 sample_793  \
2          3.99212487426  3.64249364243  ...   3.080061   4.337404   2.068224
3          2.77273024777  4.42355800269  ...   2.815739   2.597126   0.857663
4          6.54669231412  6.84951144203  ...   6.209617   6.070379   6.218739
5          10.4882518866  9.46446610892  ...   9.644469    9.86399  10.623068
```

```
6                    0.0            0.0  ...        0.0        0.0        0.0

sample_id sample_794 sample_795 sample_796 sample_797 sample_798 sample_799  \
2           4.288388   4.472176   1.865642   3.942955   3.249582   2.590339
3            3.45249   4.908746   2.718197   4.453807   3.707492   2.787976
4           7.209151   5.937848   7.350099   6.346597   8.185901   7.318624
5            9.87562   9.330901  10.006003  10.056868   9.504082   9.987136
6                0.0        0.0        0.0        0.0        0.0        0.0

sample_id sample_800
2           2.325242
3           3.805932
4           6.530246
5           9.560367
6                0.0

[5 rows x 801 columns]
```

```
# Shows processing gene expression data, loading and merging datasets, and generating a heatm


import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt

# load data sets
data = pd.read_csv('data.csv')
labels = pd.read_csv('labels.csv', header=None)

# drop the first row in data
data = data.iloc[1:]

# rename columns in data
data.columns = ['sample_id'] + [f'gene_{i}' for i in range(data.shape[1]-1)]

# rename column in labels
labels = labels.rename(columns={0: 'sample_id', 1: 'class'})

# merge data sets
merged_data = pd.merge(data, labels, on='sample_id')

# re-order the columns in the merged dataset
merged_data = merged_data[['sample_id', 'class'] + [f'gene_{i}' for i in range(data.shape[1]-

# save merged data
merged_data.to_csv('merged_data.csv', index=False)

# extract gene expression data
gene_expression = merged_data.iloc[:, 2:].astype(float)

# create a clustered heatmap
plt.figure(figsize=(20, 20))
```

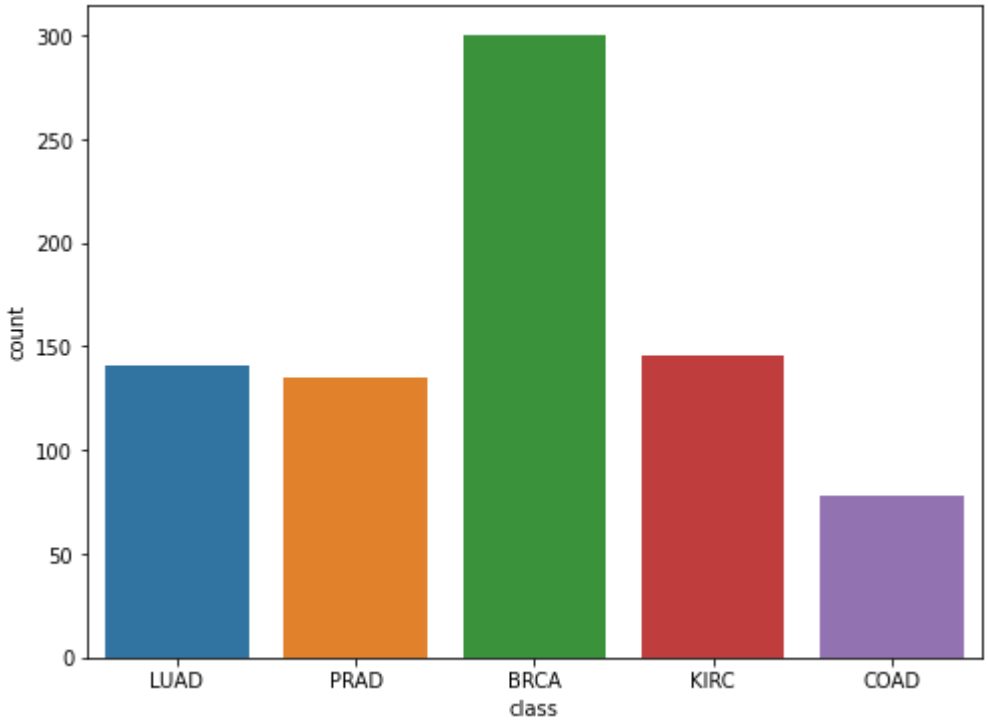
```
sns.clustermap(gene_expression, cmap='viridis')
plt.show()
```

```
<Figure size 1440x1440 with 0 Axes>
```



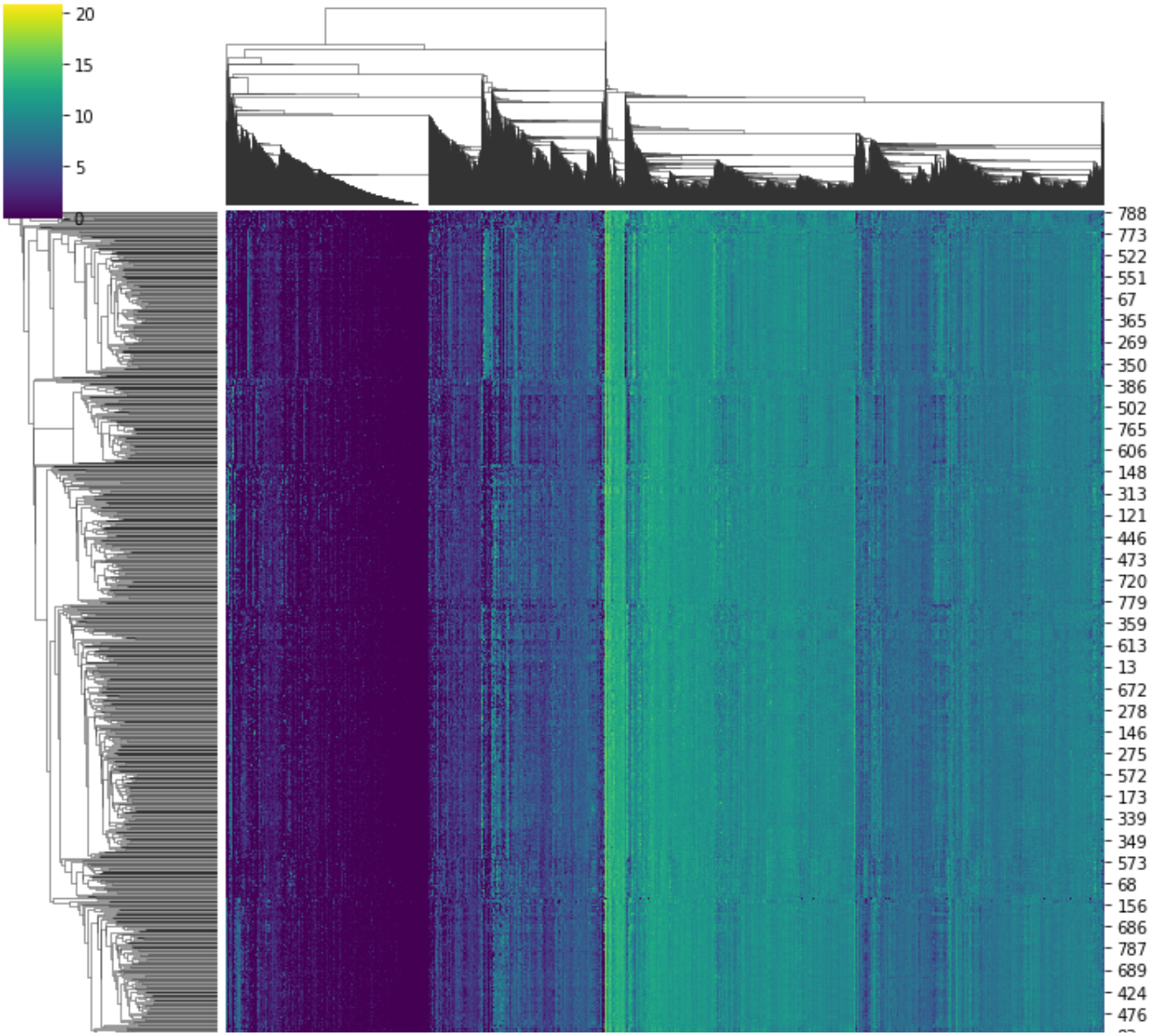

```
import seaborn as sns
import matplotlib.pyplot as plt

# load merged data set
merged_data = pd.read_csv('merged_data.csv')

# create countplot
plt.figure(figsize=(8, 6))
```

```
sns.countplot(x='class', data=merged_data)
plt.show()
```




```
# Plot the merged dataset as a hierarchically-clustered
# heatmap.


import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt

# load merged data set
merged_data = pd.read_csv('merged_data.csv')

# extract gene expression data
gene_expression = merged_data.iloc[:, 2:].astype(float)

# create a clustered heatmap
plt.figure(figsize=(20, 20))
sns.clustermap(gene_expression, cmap='viridis')
plt.show()
```

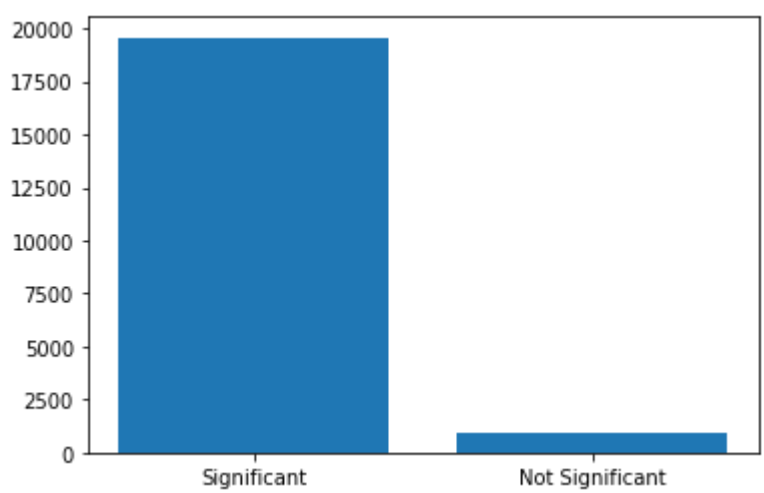
```
<Figure size 1440x1440 with 0 Axes>
```




```
from scipy.stats import f_oneway

# perform ANOVA for each gene
p_values = {}
for col in merged_data.columns[2:]:
    groups = []
    for target in merged_data['class'].unique():
        groups.append(merged_data.loc[merged_data['class'] == target, col])
    p_values[col] = f_oneway(*groups)[1]

# select significant genes based on p-value threshold
significant_genes = [gene for gene, p_value in p_values.items() if p_value < 0.05]

# print the number of significant genes
print(f'Number of significant genes: {len(significant_genes)}')
```

```
Number of significant genes: 19570
```

```
# there are 19570 significant genes based on the ANOVA analysis. For feature selection, we ca

# If the p-value is less than 0.05, we reject the null hypothesis, which means there is a sig
# differentiate between the target classes. On the other hand, if the p-value is greater thar
# to differentiate between the target classes.


import matplotlib.pyplot as plt

# perform ANOVA for each gene
p_values = {}
for col in merged_data.columns[2:]:
    groups = []
    for target in merged_data['class'].unique():
        groups.append(merged_data.loc[merged_data['class'] == target, col])
    p_values[col] = f_oneway(*groups)[1]

# select significant genes based on p-value threshold
significant_genes = [gene for gene, p_value in p_values.items() if p_value < 0.05]

# plot the number of significant genes
plt.bar(['Significant', 'Not Significant'], [len(significant_genes), len(p_values)-len(signif
plt.show()
```




```
# create a new dataframe with only the significant genes
significant_data = merged_data[['class'] + significant_genes]

# print the number of columns before and after removing non-significant genes
print(f'Number of columns before removing non-significant genes: {len(merged_data.columns)}')
print(f'Number of columns after removing non-significant genes: {len(significant_data.columns
```

```
Number of columns before removing non-significant genes: 20533
Number of columns after removing non-significant genes: 19571
```

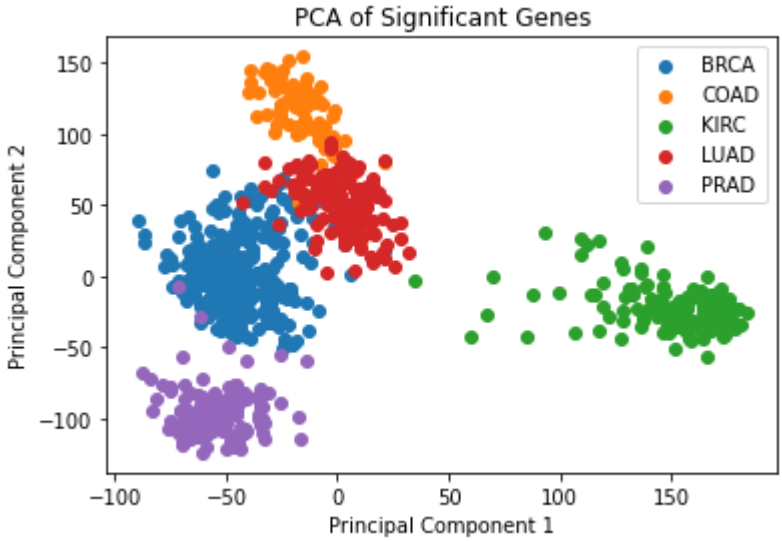
```
from scipy.stats import f_oneway

# perform ANOVA for each gene
p_values = {}
for col in merged_data.columns[2:]:
    groups = []
    for target in merged_data['class'].unique():
        groups.append(merged_data.loc[merged_data['class'] == target, col])
    p_values[col] = f_oneway(*groups)[1]

# select significant genes based on p-value threshold
significant_genes = [gene for gene, p_value in p_values.items() if p_value < 0.05]

# print the number of significant genes
print(f'Number of significant genes: {len(significant_genes)}')
```

```
Number of significant genes: 19570
```

```
# principal component analysis
# (PCA)
```

```
# X is the matrix of gene expression data, and y is the vector of class labels.
# We use PCA from the scikit-learn library to reduce the dimensionality of the data to 2 prir
# components (n_components=2). We then transform the data to the new low-dimensional space us
# transform. Finally, we plot the PCA results using different colors for each class label.
```

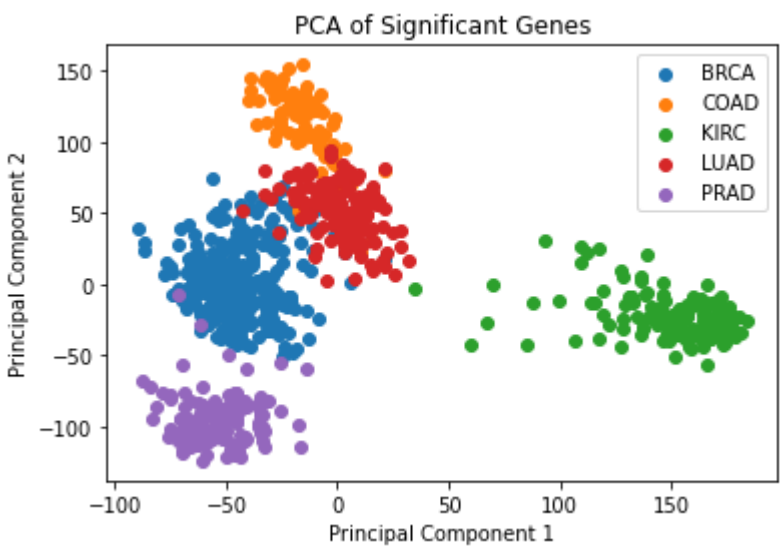
```
from sklearn.decomposition import PCA
import numpy as np

# separate the class labels from the data
X = significant_data.drop(columns=['class']).values
y = significant_data['class'].values

# perform PCA
pca = PCA(n_components=2)
X_pca = pca.fit_transform(X)

# plot the PCA results
colors = np.unique(y)
for color in colors:
    plt.scatter(X_pca[y == color, 0], X_pca[y == color, 1], label=color)
plt.legend()
plt.xlabel('Principal Component 1')
plt.ylabel('Principal Component 2')
plt.title('PCA of Significant Genes')
plt.show()
```




```
from sklearn.decomposition import PCA
import numpy as np

# separate the class labels from the data
X = significant_data.drop(columns=['class']).values
y = significant_data['class'].values

# perform PCA
pca = PCA(n_components=.90)
X_pca = pca.fit_transform(X)

# plot the PCA results
colors = np.unique(y)
for color in colors:
    plt.scatter(X_pca[y == color, 0], X_pca[y == color, 1], label=color)
plt.legend()
plt.xlabel('Principal Component 1')
plt.ylabel('Principal Component 2')
plt.title('PCA of Significant Genes')
plt.show()
```

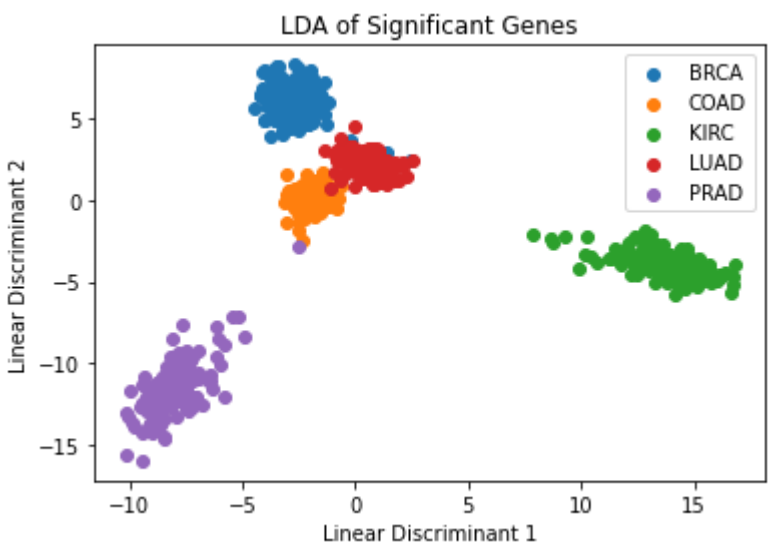
PCA of Significant Genes

```
from sklearn.decomposition import PCA
import numpy as np

# separate the class labels from the data
X = significant_data.drop(columns=['class']).values
y = significant_data['class'].values

# perform PCA
pca = PCA(n_components=800)
X_pca = pca.fit_transform(X)

# plot the PCA results
colors = np.unique(y)
for color in colors:
    plt.scatter(X_pca[y == color, 0], X_pca[y == color, 1], label=color)
plt.legend()
plt.xlabel('Principal Component 1')
plt.ylabel('Principal Component 2')
plt.title('PCA of Significant Genes')
plt.show()
```



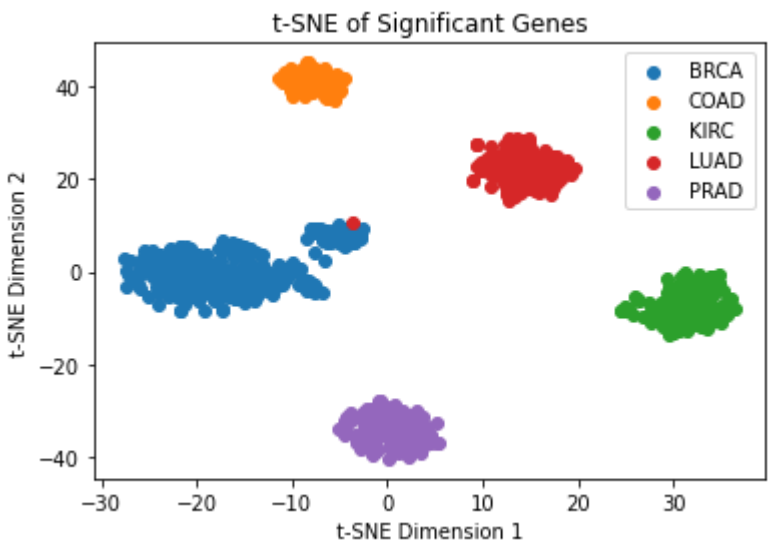

```
# linear discriminant analysis (LDA)


# X is the matrix of gene expression data, and y is the vector of class labels.
# We use LinearDiscriminantAnalysis from the scikit-learn library to reduce the dimensionalit
# of the data to 2 linear discriminants (n_components=2). We then transform the data to the r
# low-dimensional space using fit_transform. Finally, we plot the LDA results using different


from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
import numpy as np
```

```
# separate the class labels from the data
X = significant_data.drop(columns=['class']).values
y = significant_data['class'].values

# perform LDA
lda = LinearDiscriminantAnalysis(n_components=2)
X_lda = lda.fit_transform(X, y)

# plot the LDA results
colors = np.unique(y)
for color in colors:
    plt.scatter(X_lda[y == color, 0], X_lda[y == color, 1], label=color)
plt.legend()
plt.xlabel('Linear Discriminant 1')
plt.ylabel('Linear Discriminant 2')
plt.title('LDA of Significant Genes')
plt.show()
```




```
# t-distributed stochastic neighbor embedding (t-SNE)


# X is the matrix of gene expression data, and y is the vector of class labels.
# We use TSNE from the scikit-learn library to reduce the dimensionality of the
# data to 2 t-SNE dimensions (n_components=2) with a perplexity of 30 (perplexity=30).
# We then transform the data to the new low-dimensional space using fit_transform.
# Finally, we plot the t-SNE results using different colors for each class label.

# t-SNE is a nonlinear dimensionality reduction method that is useful for visualizing high-di


from sklearn.manifold import TSNE
import numpy as np

# separate the class labels from the data
X = significant_data.drop(columns=['class']).values
```

```
y = significant_data['class'].values

# perform t-SNE
tsne = TSNE(n_components=2, perplexity=30, random_state=42)
X_tsne = tsne.fit_transform(X)

# plot the t-SNE results
colors = np.unique(y)
for color in colors:
    plt.scatter(X_tsne[y == color, 0], X_tsne[y == color, 1], label=color)
plt.legend()
plt.xlabel('t-SNE Dimension 1')
plt.ylabel('t-SNE Dimension 2')
plt.title('t-SNE of Significant Genes')
plt.show()
```




```
from sklearn.cluster import KMeans

# separate the class labels from the data and select only significant genes
X = significant_data.drop(columns=['class']).values

# perform k-means clustering
kmeans = KMeans(n_clusters=1, random_state=42)
kmeans.fit(X)

# get the cluster center and compute the distance of each gene to the center
cluster_center = kmeans.cluster_centers_[0]
gene_distances = np.linalg.norm(X - cluster_center, axis=0)

# print the genes sorted by their distance to the center
sorted_genes = np.argsort(gene_distances)
for i, gene_idx in enumerate(sorted_genes):
    gene_name = merged_data.columns[gene_idx+2]
```

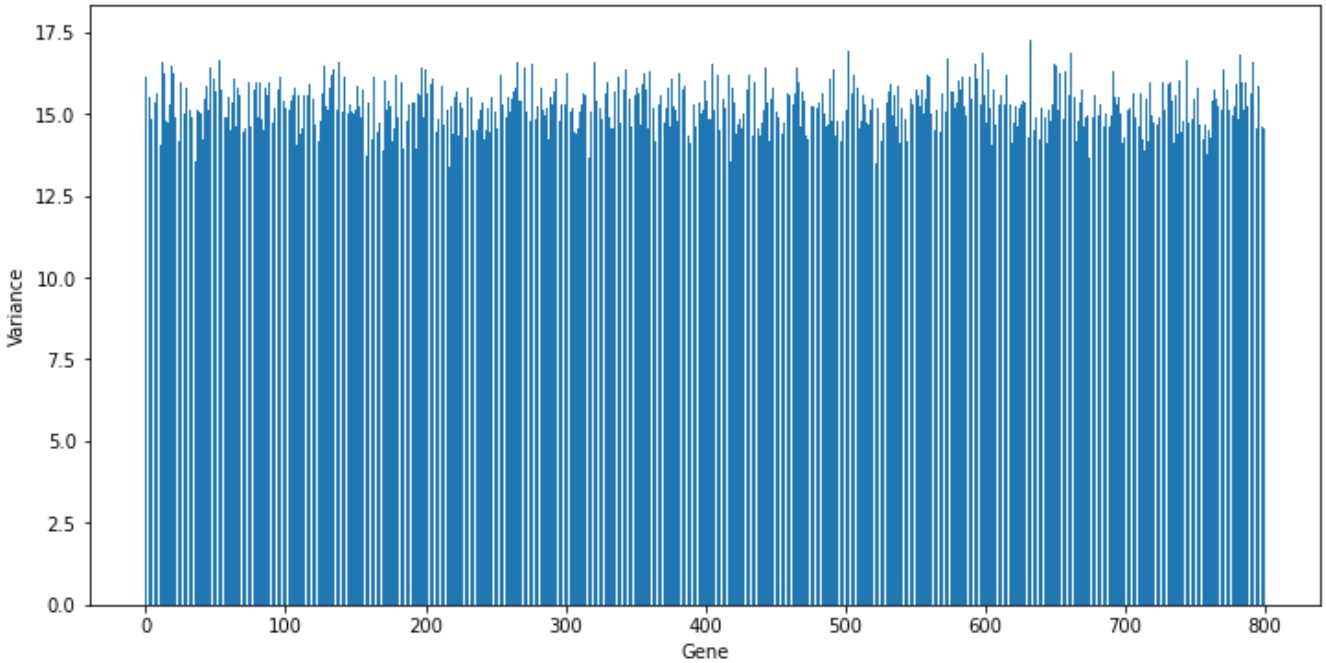
```
    print(f'{i+1}. {gene_name} ({gene_distances[gene_idx]:.2f})')
```

```
19069. gene_4883 (84.85)
19070. gene_1314 (84.86)
19071. gene_3761 (84.88)
```

```
# calculates the variance of each gene across all samples


# compute the variance of each gene across all samples
gene_variances = np.var(significant_data.drop(columns=['class']), axis=1)

# create a DataFrame to store the results
gene_variances_df = pd.DataFrame({
    'Gene': significant_data.index,
    'Variance': gene_variances
})

# sort the genes by their variance in ascending order
gene_variances_df = gene_variances_df.sort_values('Variance')

# add row numbers
gene_variances_df.index = np.arange(1, len(gene_variances_df) + 1)

# print the formatted DataFrame
print('Genes sorted by variance:')
print(gene_variances_df.to_string(index=False))
```

```
589 16.112659
391 16.117821
 96 16.121271
338 16.131921
 34 16.137099
475 16.137871
 81 16.138047
 30 16.140408
  0 16.141411
148 16.141923
142 16.143311
584 16.151172
560 16.153305
248 16.153690
133 16.164250
417 16.167760
615 16.177118
 55 16.179127
621 16.184300
254 16.185080
559 16.185480
409 16.188053
 26 16.189461
431 16.200386
179 16.202396
506 16.208157
792 16.209173
122 16.216677
 13 16.217783
 72 16.222063
709 16.222762
```

```
import matplotlib.pyplot as plt

# compute the variance of each gene across all samples
gene_variances = np.var(significant_data.drop(columns=['class']), axis=1)

# sort the genes by their variance in ascending order
sorted_genes = np.argsort(gene_variances)

# create a bar plot of gene variances
fig, ax = plt.subplots(figsize=(12,6))
ax.bar(significant_data.index[sorted_genes], gene_variances[sorted_genes])

# add labels to the x and y axes
ax.set_xlabel('Gene')
ax.set_ylabel('Variance')

# show the plot
plt.show()
```




```
# calculates the standard deviation of each gene across all samples


# compute the standard deviation of each gene across all samples
gene_std = np.std(significant_data.drop(columns=['class']), axis=1)

# create a DataFrame to store the results
gene_std_df = pd.DataFrame({
    'Gene': significant_data.index,
    'Std': gene_std
})

# sort the genes by their standard deviation in ascending order
gene_std_df = gene_std_df.sort_values('Std')

# add row numbers
gene_std_df.index = np.arange(1, len(gene_std_df) + 1)

# print the formatted DataFrame
print('Genes sorted by standard deviation:')
print(gene_std_df.to_string(index=False))
```

```
219 4.003823
537 4.005291
415 4.005504
376 4.006310
 63 4.006405
 49 4.006917
324 4.007423
489 4.009394
205 4.010266
215 4.010279
646 4.010740
293 4.011562
520 4.011982
594 4.012657
163 4.013699
411 4.013807
278 4.013939
589 4.014058
391 4.014701
 96 4.015130
338 4.016456
 34 4.017101
475 4.017197
 81 4.017219
 30 4.017513
  0 4.017638
148 4.017701
142 4.017874
584 4.018852
560 4.019117
248 4.019165
133 4.020479
417 4.020915
615 4.022079
 55 4.022329
621 4.022972
254 4.023069
559 4.023118
409 4.023438
 26 4.023613
431 4.024970
179 4.025220
506 4.025936
792 4.026062
122 4.026994
 13 4.027131
 72 4.027662
709 4.027749
```

```
# This will output a DataFrame that shows genes sorted by their standard deviation.
# Genes with a low standard deviation can be considered to have similar expression
# values across all samples.
```

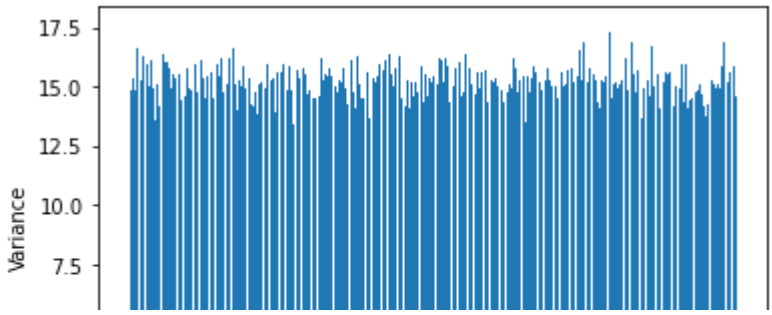
```
# compute the mean expression value of each gene across all samples
gene_means = np.mean(significant_data.drop(columns=['class']), axis=1)
```

```
# create a DataFrame to store the results
gene_means_df = pd.DataFrame({
    'Gene': significant_data.index,
    'Mean Expression': gene_means
})

# sort the genes by their mean expression value in ascending order
gene_means_df = gene_means_df.sort_values('Mean Expression')

# add row numbers
gene_means_df.index = np.arange(1, len(gene_means_df) + 1)

# print the formatted DataFrame
print('Genes sorted by mean expression value:')
print(gene_means_df.to_string(index=False))
```

```
393         6.909928
455         6.910007
737         6.910997
 65         6.911433
446         6.912084
 70         6.912320
 60         6.914145
306         6.914431
639         6.914462
364         6.914709
422         6.917763
220         6.918694
208         6.920914
665         6.921352
224         6.923076
235         6.923317
178         6.925616
734         6.926206
388         6.926914
434         6.928802
756         6.929882
```

```
#  computes the variance of gene expression values across all samples and stores the result i
#  The DataFrame has two columns, 'Gene' which contains the gene names, and 'Variance' which
# The genes in the DataFrame are sorted by their variance in ascending order.
# The resulting output displays the lowest variance across all samples, indicating which gene
```

```
import matplotlib.pyplot as plt

# create a bar chart of the gene variances
plt.bar(gene_variances_df['Gene'], gene_variances_df['Variance'])

# set the x-axis label
plt.xlabel('Gene')

# set the y-axis label
plt.ylabel('Variance')

# rotate the x-axis tick labels for better readability
plt.xticks(rotation=90)

# show the plot
plt.show()
```

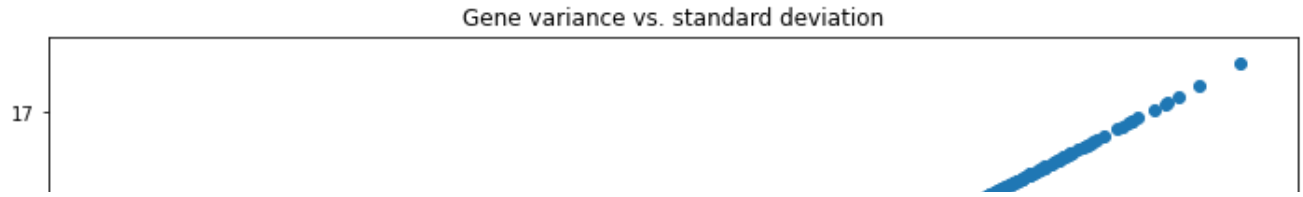



```
# both variance and standard deviation for each gene across all samples,
# and stores them in a new DataFrame gene_variances_df. The DataFrame is
# then sorted by both variance and standard deviation in ascending order
```

```
# Genes whose expression values are similar across all
# samples
```

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

# compute the variance and standard deviation of each gene across all samples
gene_variances = np.var(significant_data.drop(columns=['class']), axis=1)
gene_std = np.std(significant_data.drop(columns=['class']), axis=1)

# create a DataFrame to store the results
gene_variances_df = pd.DataFrame({
    'Gene': significant_data.index,
    'Variance': gene_variances,
    'Std': gene_std
})

# sort the genes by their variance and standard deviation in ascending order
gene_variances_df = gene_variances_df.sort_values(['Variance', 'Std'])

# add row numbers
gene_variances_df.index = np.arange(1, len(gene_variances_df) + 1)

# display the top 10 genes with the lowest variance and standard deviation
print('Genes sorted by variance and standard deviation:')
print(gene_variances_df.head(10).to_string(index=False))

fig, ax = plt.subplots(figsize=(12,6))

# create a scatter plot of gene variance vs. standard deviation
plt.scatter(gene_variances_df['Std'], gene_variances_df['Variance'])
plt.xlabel('Standard deviation')
plt.ylabel('Variance')
plt.title('Gene variance vs. standard deviation')
plt.show()
```

```
Genes sorted by variance and standard deviation:
 Gene  Variance      Std
  797 12.016406 3.466469
  189 12.997458 3.605199
  217 13.384146 3.658435
  522 13.472057 3.670430
  634 13.485877 3.672312
   36 13.556913 3.681971
  418 13.558036 3.682124
  588 13.589285 3.686365
  512 13.617408 3.690177
  265 13.645103 3.693928
```




```
import numpy as np
import pandas as pd

# compute the variance of each gene within each cancer type
variances_by_cancer_type = significant_data.groupby('class').apply(lambda x: np.var(x.drop(cc

# create a DataFrame to store the results
similar_genes_df = pd.DataFrame(columns=['Gene', 'Cancer Type', 'Variance'])

# iterate over each cancer type
for cancer_type in variances_by_cancer_type.index:
    # get the top 10 genes with the lowest variance within the cancer type
    similar_genes = pd.Series(variances_by_cancer_type[cancer_type]).nsmallest(10)

      # add the genes to the DataFrame
    for gene in similar_genes.index:
        similar_genes_df = similar_genes_df.append({
            'Gene': gene,
            'Cancer Type': cancer_type,
            'Variance': similar_genes[gene]
        }, ignore_index=True)

# display the top 10 genes with similar expression values within each cancer type
for cancer_type in similar_genes_df['Cancer Type'].unique():
    print('Top 10 genes with similar expression values in', cancer_type, ':')
    print(similar_genes_df.loc[similar_genes_df['Cancer Type'] == cancer_type].head(10).to_st
```

```
Gene Cancer Type  Variance
   0 (PRAD, 206)  15.94417
Top 10 genes with similar expression values in ('PRAD', 211) :
Gene Cancer Type  Variance
   0 (PRAD, 211) 15.232698
Top 10 genes with similar expression values in ('PRAD', 213) :
Gene Cancer Type  Variance
   0 (PRAD, 213) 14.657595
Top 10 genes with similar expression values in ('PRAD', 226) :
Gene Cancer Type  Variance
   0 (PRAD, 226)  15.18024
Top 10 genes with similar expression values in ('PRAD', 230) :
Gene Cancer Type  Variance
   0 (PRAD, 230) 15.770762
Top 10 genes with similar expression values in ('PRAD', 234) :
Gene Cancer Type  Variance
   0 (PRAD, 234)   14.4949
Top 10 genes with similar expression values in ('PRAD', 241) :
Gene Cancer Type  Variance
   0 (PRAD, 241) 15.348527
Top 10 genes with similar expression values in ('PRAD', 242) :
Gene Cancer Type  Variance
   0 (PRAD, 242) 14.232622
Top 10 genes with similar expression values in ('PRAD', 250) :
Gene Cancer Type  Variance
   0 (PRAD, 250) 15.089176
Top 10 genes with similar expression values in ('PRAD', 253) :
Gene Cancer Type  Variance
   0 (PRAD, 253) 14.870125
Top 10 genes with similar expression values in ('PRAD', 255) :
Gene Cancer Type  Variance
   0 (PRAD, 255)   15.3045
Top 10 genes with similar expression values in ('PRAD', 256) :
Gene Cancer Type  Variance
   0 (PRAD, 256) 15.271454
Top 10 genes with similar expression values in ('PRAD', 257) :
Gene Cancer Type  Variance
   0 (PRAD, 257) 14.488416
Top 10 genes with similar expression values in ('PRAD', 264) :
Gene Cancer Type  Variance
   0 (PRAD, 264) 15.795035
Top 10 genes with similar expression values in ('PRAD', 272) :
Gene Cancer Type  Variance
   0 (PRAD, 272) 15.043239
Top 10 genes with similar expression values in ('PRAD', 274) :
Gene Cancer Type  Variance
   0 (PRAD, 274) 14.491413
Top 10 genes with similar expression values in ('PRAD', 298) :
Gene Cancer Type  Variance
   0 (PRAD, 298) 14.073278
Top 10 genes with similar expression values in ('PRAD', 303) :
```

```
merged_data['class'].unique()
```

```
array(['LUAD', 'PRAD', 'BRCA', 'KIRC', 'COAD'], dtype=object)
```

```
possible_cancer_types = ['BRCA', 'COAD', 'KIRC', 'LUAD', 'PRAD']
merged_cancer_types = merged_data['class'].unique()

missing_cancer_types = set(possible_cancer_types) - set(merged_cancer_types)
print(f'Missing cancer types: {missing_cancer_types}')
```

```
Missing cancer types: set()
```

```
merged_data.isnull().sum()
```

```
sample_id     0
class         0
gene_0        0
gene_1        0
gene_2        0
              ..
gene_20526    0
gene_20527    0
gene_20528    0
gene_20529    0
gene_20530    0
Length: 20533, dtype: int64
```

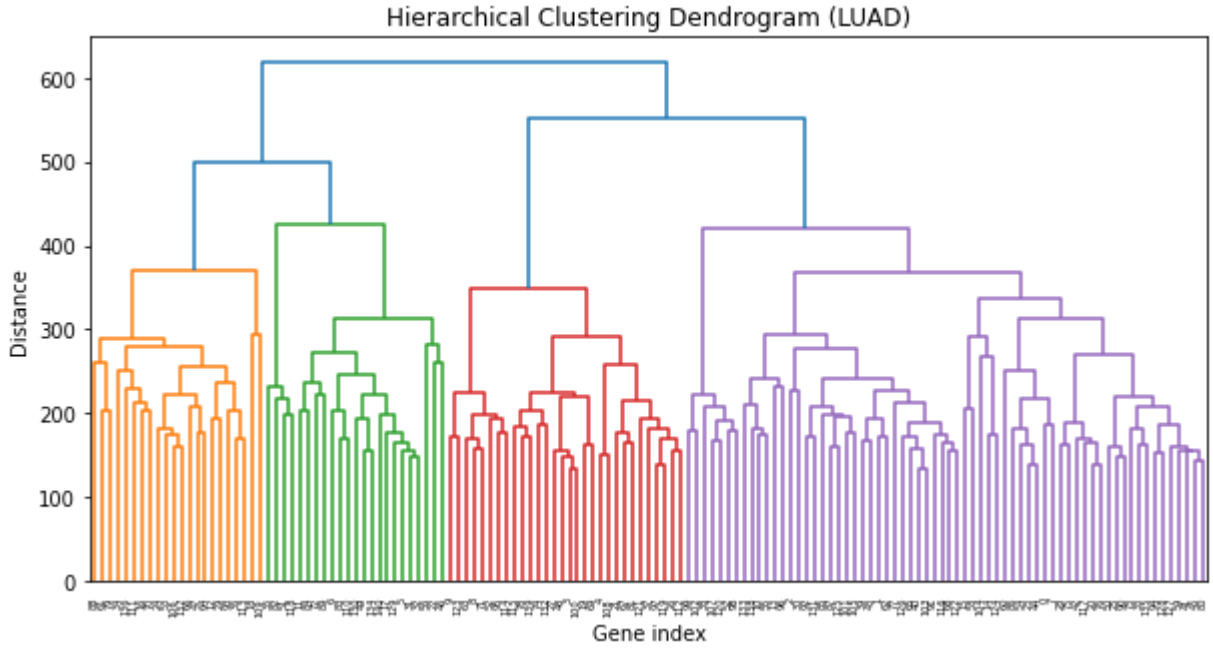
```
# load merged data set
merged_data = pd.read_csv('merged_data.csv')

# create list of all gene names in merged_data
all_genes = list(merged_data.columns[2:])

# check if all genes in significant_genes are present in all_genes
if set(significant_genes) <= set(all_genes):
    print("All genes in significant_genes are present in merged_data")
else:
    print("Some genes in significant_genes are not present in merged_data")
```

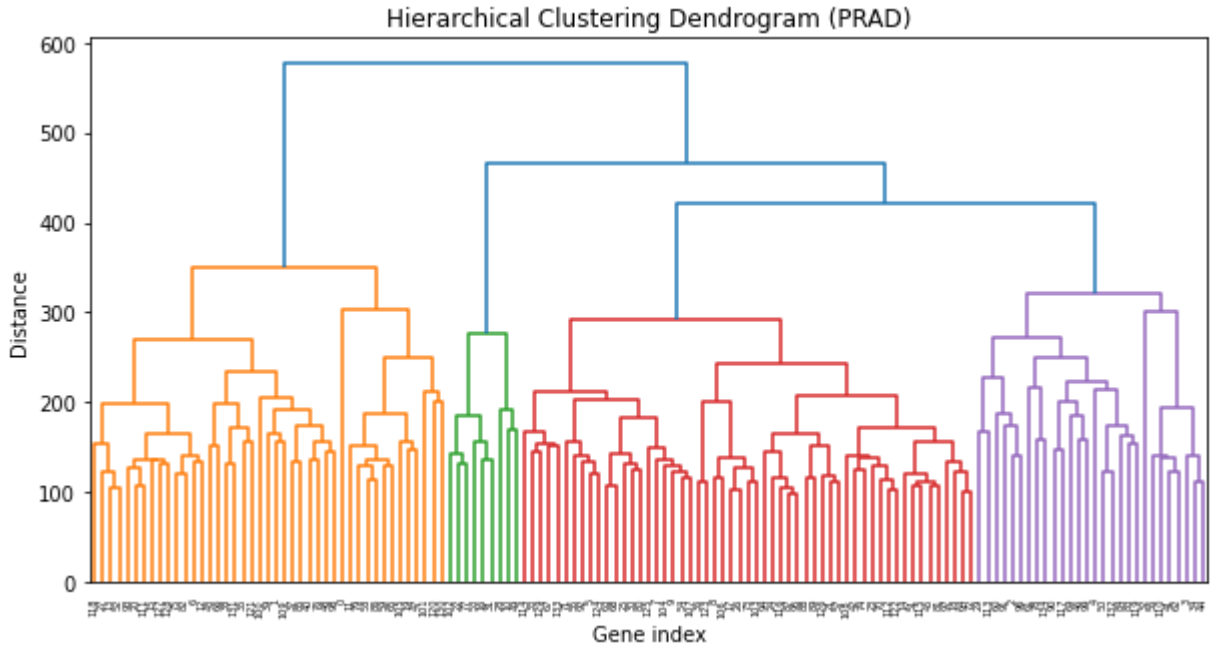
```
All genes in significant_genes are present in merged_data
```

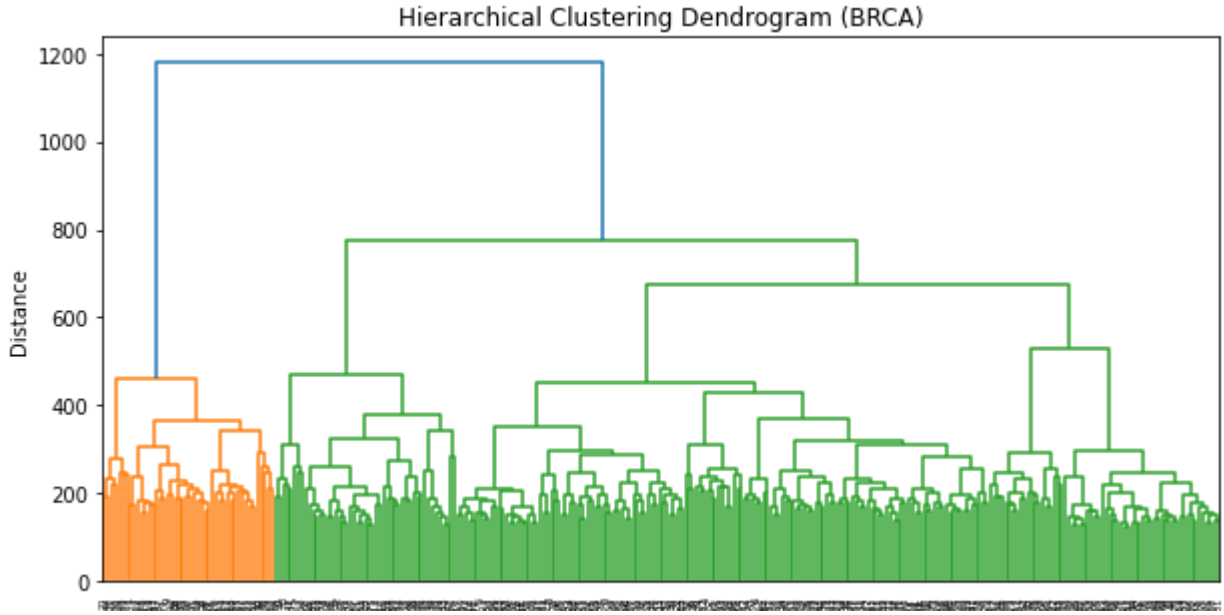
```
## Genes whose expression values are similar across samples
# of each cancer type
```

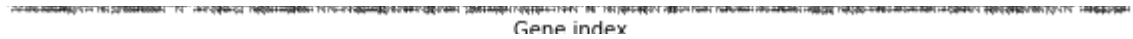
Double-click (or enter) to edit

```
from scipy.cluster import hierarchy

# iterate over each cancer type
for cancer_type in merged_data['class'].unique():
    # filter the merged data to only include samples of the current cancer type
    cancer_data = merged_data.loc[merged_data['class'] == cancer_type, significant_genes]

    # perform hierarchical clustering
    linkage_matrix = hierarchy.linkage(cancer_data, method='ward')

    # plot dendrogram
    plt.figure(figsize=(10,5))
    hierarchy.dendrogram(linkage_matrix)
    plt.title(f'Hierarchical Clustering Dendrogram ({cancer_type})')
    plt.xlabel('Gene index')
    plt.ylabel('Distance')
    plt.show()
```


Hierarchical Clustering Dendrogram (LUAD)
Distance
0
100
200
300
400
500
600
Gene index


Hierarchical Clustering Dendrogram (PRAD)
Distance
0
100
200
300
400
500
600
Gene index


Hierarchical Clustering Dendrogram (BRCA)
Distance
0
200
400
600
800
1000
1200




```
from scipy.stats import ttest_ind

# define two populations
population1 = [1, 2, 3, 4, 5]
population2 = [6, 7, 8, 9, 10]

# perform t-test
t_stat, p_value = ttest_ind(population1, population2)

# print results
print("t-statistic:", t_stat)
print("p-value:", p_value)
```

```
t-statistic: -5.0
p-value: 0.001052825793366539
```

```
from scipy.stats import ttest_ind
import numpy as np

# Define two populations
population1 = np.array([1, 2, 3, 4, 5])
population2 = np.array([6, 7, 8, 9, 10])

# Calculate means and standard deviations of the populations
mean1 = np.mean(population1)
mean2 = np.mean(population2)
std1 = np.std(population1)
std2 = np.std(population2)

# Calculate the t-statistic and p-value using a two-sample t-test
t_stat, p_val = ttest_ind(population1, population2)

# Print the results
print("Population 1 Mean:", mean1)
print("Population 2 Mean:", mean2)
print("Population 1 Standard Deviation:", std1)
print("Population 2 Standard Deviation:", std2)
print("T-Statistic:", t_stat)
print("P-Value:", p_val)
```

```
Population 1 Mean: 3.0
Population 2 Mean: 8.0
Population 1 Standard Deviation: 1.4142135623730951
Population 2 Standard Deviation: 1.4142135623730951
```

```
T-Statistic: -5.0
P-Value: 0.001052825793366539
```

```
import numpy as np
import pandas as pd

# load data into a pandas dataframe
merged_data_heatmap = pd.read_csv('merged_data.csv')

# get the data from the 'class' column
KIRC = merged_data_heatmap[merged_data_heatmap['class'] == 'KIRC'].iloc[:, 1:]
COAD = merged_data_heatmap[merged_data_heatmap['class'] == 'COAD'].iloc[:, 1:]
PRAD = merged_data_heatmap[merged_data_heatmap['class'] == 'PRAD'].iloc[:, 1:]
BRCA = merged_data_heatmap[merged_data_heatmap['class'] == 'BRCA'].iloc[:, 1:]
LUAD = merged_data_heatmap[merged_data_heatmap['class'] == 'LUAD'].iloc[:, 1:]

# drop the 'class' column before converting to numpy array
KIRC = KIRC.drop(columns=['class'])
COAD = COAD.drop(columns=['class'])
PRAD = PRAD.drop(columns=['class'])
BRCA = BRCA.drop(columns=['class'])
LUAD = LUAD.drop(columns=['class'])

# calculate the mean for each class
mean_KIRC = np.mean(KIRC.to_numpy().astype(float))
mean_COAD = np.mean(COAD.to_numpy().astype(float))
mean_PRAD = np.mean(PRAD.to_numpy().astype(float))
mean_BRCA = np.mean(BRCA.to_numpy().astype(float))
mean_LUAD = np.mean(LUAD.to_numpy().astype(float))

# print the mean for each class
print("population_mean_KIRC:", mean_KIRC.round(2))
print("population_mean_COAD:", mean_COAD.round(2))
print("population_mean_PRAD:", mean_PRAD.round(2))
print("population_mean_BRCA:", mean_BRCA.round(2))
print("population_mean_LUAD:", mean_LUAD.round(2))
```

```
population_mean_KIRC: 6.48
population_mean_COAD: 6.3
population_mean_PRAD: 6.47
population_mean_BRCA: 6.41
population_mean_LUAD: 6.53
```

```
merged_data_heatmap.head()
```

	sample_id	class	gene_0	gene_1	gene_2	gene_3	gene_4	gene_5	gene_6	
0	sample_1	LUAD	0.0	0.592732	1.588421	7.586157	9.623011	0.0	6.816049	0.
1	sample_2	PRAD	0.0	3.511759	4.327199	6.881787	9.870730	0.0	6.972130	0.
2	sample_3	PRAD	0.0	3.663618	4.507649	6.659068	10.196184	0.0	7.843375	0.
3	sample_4	BRCA	0.0	2.655741	2.821547	6.539454	9.738265	0.0	6.566967	0.

```
merged_data_heatmap_SAMPLE = merged_data_heatmap.sample(n=100)
merged_data_heatmap_SAMPLE.shape
```

```
(100, 20533)
```

```
merged_data_heatmap_SAMPLE.head()
```

	sample_id	class	gene_0	gene_1	gene_2	gene_3	gene_4	gene_5	gene_6
588	sample_589	LUAD	0.0	5.803979	5.421920	6.418308	9.164796	0.0	8.718005
156	sample_157	PRAD	0.0	3.981743	3.781926	6.403283	10.190430	0.0	8.770182
16	sample_17	KIRC	0.0	3.004519	3.007178	6.524205	9.062661	0.0	7.995937
425	sample_426	BRCA	0.0	2.258851	3.451528	6.210562	9.694805	0.0	7.138825
355	sample_356	BRCA	0.0	1.894294	1.276616	6.709180	10.268647	0.0	5.970939

5 rows × 20533 columns

```
# get the data from the 'class' column
KIRC = merged_data_heatmap[merged_data_heatmap['class'] == 'KIRC'].iloc[:, 1:]
COAD = merged_data_heatmap[merged_data_heatmap['class'] == 'COAD'].iloc[:, 1:]
PRAD = merged_data_heatmap[merged_data_heatmap['class'] == 'PRAD'].iloc[:, 1:]
BRCA = merged_data_heatmap[merged_data_heatmap['class'] == 'BRCA'].iloc[:, 1:]
LUAD = merged_data_heatmap[merged_data_heatmap['class'] == 'LUAD'].iloc[:, 1:]

# drop the 'class' column before converting to numpy array
KIRC = KIRC.drop(columns=['class'])
COAD = COAD.drop(columns=['class'])
PRAD = PRAD.drop(columns=['class'])
BRCA = BRCA.drop(columns=['class'])
LUAD = LUAD.drop(columns=['class'])

# calculate the mean for each class
Sample_mean_KIRC = np.mean(KIRC.to_numpy().astype(float))
Sample_mean_COAD = np.mean(COAD.to_numpy().astype(float))
```

```
Sample_mean_PRAD = np.mean(PRAD.to_numpy().astype(float))
Sample_mean_BRCA = np.mean(BRCA.to_numpy().astype(float))
Sample_mean_LUAD = np.mean(LUAD.to_numpy().astype(float))

# print the mean for each class
print("Sample_mean_KIRC:", mean_KIRC.round(2))
print("Sample_mean_COAD:", mean_COAD.round(2))
print("Sample_mean_PRAD:", mean_PRAD.round(2))
print("Sample_mean_BRCA:", mean_BRCA.round(2))
print("Sample_mean_LUAD:", mean_LUAD.round(2))
```

```
Sample_mean_KIRC: 6.48
Sample_mean_COAD: 6.3
Sample_mean_PRAD: 6.47
Sample_mean_BRCA: 6.41
Sample_mean_LUAD: 6.53
```

```
# calculate the p-values for each class


import numpy as np
import pandas as pd
from scipy.stats import ttest_ind

# load data into a pandas dataframe
merged_data_heatmap = pd.read_csv('merged_data.csv')


# get the data from the 'class' column
KIRC = merged_data_heatmap[merged_data_heatmap['class'] == 'KIRC'].iloc[:, 1:].select_dtypes(
COAD = merged_data_heatmap[merged_data_heatmap['class'] == 'COAD'].iloc[:, 1:].select_dtypes(
PRAD = merged_data_heatmap[merged_data_heatmap['class'] == 'PRAD'].iloc[:, 1:].select_dtypes(
BRCA = merged_data_heatmap[merged_data_heatmap['class'] == 'BRCA'].iloc[:, 1:].select_dtypes(
LUAD = merged_data_heatmap[merged_data_heatmap['class'] == 'LUAD'].iloc[:, 1:].select_dtypes(




# calculate the mean for each class
mean_KIRC = np.mean(KIRC)
mean_COAD = np.mean(COAD)
mean_PRAD = np.mean(PRAD)
mean_BRCA = np.mean(BRCA)
mean_LUAD = np.mean(LUAD)

# print the mean for each class
print("population_mean_KIRC:", mean_KIRC.round(2))
print("population_mean_COAD:", mean_COAD.round(2))
print("population_mean_PRAD:", mean_PRAD.round(2))
print("population_mean_BRCA:", mean_BRCA.round(2))
```

```
print("population_mean_LUAD:", mean_LUAD.round(2))

# calculate the p-value for each class
pval_KIRC = ttest_ind(KIRC, COAD).pvalue[0]
pval_COAD = ttest_ind(COAD, PRAD).pvalue[0]
pval_PRAD = ttest_ind(PRAD, BRCA).pvalue[0]
pval_BRCA = ttest_ind(BRCA, LUAD).pvalue[0]
pval_LUAD = ttest_ind(LUAD, KIRC).pvalue[0]

# print the p-value for each class
print("p-values-KIRC:", pval_KIRC.round(2))
print("p-values-COAD:", pval_COAD.round(2))
print("p-values-PRAD:", pval_PRAD.round(2))
print("p-values-BRCA:", pval_BRCA.round(2))
print("p-values-LUAD:", pval_LUAD.round(2))
```

```
population_mean_KIRC: 6.48
population_mean_COAD: 6.3
population_mean_PRAD: 6.47
population_mean_BRCA: 6.41
population_mean_LUAD: 6.53
p-values-KIRC: 0.27
p-values-COAD: 0.8
p-values-PRAD: 0.18
p-values-BRCA: 0.02
p-values-LUAD: 0.79
```

```
# accepting null hypothesis




if pval_KIRC < 0.05: # alpha value is 0.05 or 5%
 print("KIRC - we are rejecting null hypothesis")
else:
 print("KIRC - we are accepting null hypothesis")
if pval_COAD < 0.05: # alpha value is 0.05 or 5%
 print("COAD - we are rejecting null hypothesis")
else:
 print("COAD - we are accepting null hypothesis")
if pval_PRAD < 0.05: # alpha value is 0.05 or 5%
 print("PRAD - we are rejecting null hypothesis")
else:
 print("PRAD - we are accepting null hypothesis")
if pval_BRCA < 0.05: # alpha value is 0.05 or 5%
 print("BRCA - we are rejecting null hypothesis")
else:
 print("BRCA - we are accepting null hypothesis")
if pval_LUAD < 0.05: # alpha value is 0.05 or 5%
 print("LUAD - we are rejecting null hypothesis")
```

```
else:
 print("LUAD - we are accepting null hypothesis")
```

```
KIRC - we are accepting null hypothesis
COAD - we are accepting null hypothesis
PRAD - we are accepting null hypothesis
BRCA - we are rejecting null hypothesis
LUAD - we are accepting null hypothesis
```

```
from sklearn import preprocessing

# load data into a pandas dataframe
merged_data = pd.read_csv('merged_data.csv')

# encode the 'class' column using LabelEncoder
label_encoder = preprocessing.LabelEncoder()
y = label_encoder.fit_transform(merged_data['class'])

# extract the features from the dataframe
X = merged_data.iloc[:, 1:].to_numpy()

print("X-Shape :", X.shape)
print("y-Shape :", y.shape)
```

```
X-Shape : (800, 20532)
y-Shape : (800,)
```

```
print(set(y))
```

```
{0, 1, 2, 3, 4}
```

```
print(merged_data.columns)
```

```
Index(['sample_id', 'class', 'gene_0', 'gene_1', 'gene_2', 'gene_3', 'gene_4',
       'gene_5', 'gene_6', 'gene_7',
       ...
       'gene_20521', 'gene_20522', 'gene_20523', 'gene_20524', 'gene_20525',
       'gene_20526', 'gene_20527', 'gene_20528', 'gene_20529', 'gene_20530'],
      dtype='object', length=20533)
```

```
from sklearn.preprocessing import StandardScaler, LabelEncoder

# Encode the class column
label_encoder = LabelEncoder()
y = label_encoder.fit_transform(merged_data['class'])

# Get the data from the columns excluding the class column
X = merged_data.iloc[:, 2:].values
```

```
# Scale the data
scaler = StandardScaler()
X = scaler.fit_transform(X)
```

```
from sklearn.decomposition import PCA
pca = PCA()
X_PCA = pca.fit_transform(X)
print("Input X dataset shape before PCA being applied :", X.shape)
print("Output X dataset shape after PCA being applied:",X_PCA.shape)
```

```
Input X dataset shape before PCA being applied : (800, 20531)
Output X dataset shape after PCA being applied: (800, 800)
```

```
X_PCA_DF = pd.DataFrame(X_PCA)
X_PCA_DF.to_csv("X_PCA_DF.csv")
```

```
#T-Distributed Stochastic Neighbouring Entities
```

```
from sklearn.manifold import TSNE
tsne = TSNE(verbose=1, random_state=123)
X_TSNE = tsne.fit_transform(X)
```

```
[t-SNE] Computing 91 nearest neighbors...
[t-SNE] Indexed 800 samples in 0.272s...
[t-SNE] Computed neighbors for 800 samples in 1.163s...
[t-SNE] Computed conditional probabilities for sample 800 / 800
[t-SNE] Mean sigma: 40.644012
[t-SNE] KL divergence after 250 iterations with early exaggeration: 54.029839
[t-SNE] KL divergence after 1000 iterations: 0.826113
```

```
# Print output
```

```
print("Input X dataset shape before TSNE being applied :", X.shape)
print("Output X dataset shape after TSNE being applied:",X_TSNE.shape)
```

```
Input X dataset shape before TSNE being applied : (800, 20531)
Output X dataset shape after TSNE being applied: (800, 2)
```

```
X_TSNE_DF = pd.DataFrame(X_TSNE)
X_TSNE_DF.to_csv("X_TSNE_DF.csv")
```

```
# K-means Clustering
```

```
k_means_data_X = pd.DataFrame(X_TSNE)
k_means_data_X.head(5)
```

	0	1
0	-16.747623	-5.224356
1	-5.297237	30.069643
2	-4.148905	26.095373
3	8.505836	-9.710843
4	-7.439590	19.561617

```
from sklearn.cluster import KMeans
Kmeans_Model_X = KMeans(n_clusters=5)
Kmeans_Model_X.fit(k_means_data_X)
k_means_data_X["Cluster_Label_X"] = Kmeans_Model_X.labels_
clu_column = k_means_data_X.pop('Cluster_Label_X')
k_means_data_X.insert(0, 'Cluster_Label_X', clu_column)
k_means_data_X.rename({0:"col1", 1:"col2"}, axis=1, inplace=True)
```

```
sns.lmplot("col1","col2", data = k_means_data_X, hue = 'Cluster_Label_X',
 palette ="coolwarm" , fit_reg = False)
```

```
<seaborn.axisgrid.FacetGrid at 0x7f5a42bbd040>
```

```
Kmeans_Model_Y = KMeans(n_clusters=5)
Kmeans_Model_Y.fit(y.reshape(-1,1))
```

```
KMeans
KMeans(n_clusters=5)
```

```
k_means_data_Y = pd.DataFrame(y)
k_means_data_Y.rename({0:"class_"}, axis=1, inplace=True)
k_means_data_Y["Cluster_Label_Y"] = Kmeans_Model_Y.labels_
clu_column2 = k_means_data_Y.pop('Cluster_Label_Y')
k_means_data_Y.insert(0, 'Cluster_Label_Y', clu_column2)
k_means_data_Y['index_col'] = k_means_data_Y.index
k_means_data_Y.head(5)
```

	Cluster_Label_Y	class_	index_col
0	3	3	0
1	0	4	1
2	0	4	2
3	1	0	3
4	0	4	4

```
sns.lmplot("index_col","class_", data=k_means_data_Y, hue = 'Cluster_Label_Y', palette ="cool
```

```
<seaborn.axisgrid.FacetGrid at 0x7f5a408f58b0>
```


```
# Support Vector Machine
```


```
import pandas as pd

merged_data = pd.read_csv('merged_data.csv')

X_ORG = merged_data.copy()
del X_ORG['sample_id']
X_ORG.head(5)
```

	class	gene_0	gene_1	gene_2	gene_3	gene_4	gene_5	gene_6	gene_7	gen
0	LUAD	0.0	0.592732	1.588421	7.586157	9.623011	0.0	6.816049	0.000000	
1	PRAD	0.0	3.511759	4.327199	6.881787	9.870730	0.0	6.972130	0.452595	
2	PRAD	0.0	3.663618	4.507649	6.659068	10.196184	0.0	7.843375	0.434882	
3	BRCA	0.0	2.655741	2.821547	6.539454	9.738265	0.0	6.566967	0.360982	
4	PRAD	0.0	3.467853	3.581918	6.620243	9.706829	0.0	7.758510	0.000000	

5 rows × 20532 columns

```
from sklearn.model_selection import train_test_split

# Define X and y
X = merged_data.iloc[:, 2:]
y = merged_data['class']

# Split the dataset into training and testing sets
x_train_SVM_ORG, x_test_SVM_ORG, y_train_SVM_ORG, y_test_SVM_ORG = train_test_split(X, y, tes

# Print the shapes of the training and testing sets
print(x_train_SVM_ORG.shape)
print(x_test_SVM_ORG.shape)
print(y_train_SVM_ORG.shape)
print(y_test_SVM_ORG.shape)
```

```
(560, 20531)
(240, 20531)
(560,)
(240,)
```

```
from sklearn.svm import SVC
model_SVM_ORG = SVC()
model_SVM_ORG.fit(x_train_SVM_ORG, y_train_SVM_ORG)
```

```
▾ SVC
SVC()
```

```
# train data
```

```
y_test_pred_SVM_ORG = model_SVM_ORG.predict(x_test_SVM_ORG)
y_train_pred_SVM_ORG = model_SVM_ORG.predict(x_train_SVM_ORG)
```

```
from sklearn.metrics import accuracy_score
print("Testing Accuracy :", accuracy_score(y_test_SVM_ORG, y_test_pred_SVM_ORG).round(4)*100,
print("Training Accuracy :", accuracy_score(y_train_SVM_ORG, y_train_pred_SVM_ORG).round(2)*1
```

```
Testing Accuracy : 99.58 %
Training Accuracy : 100.0 %
```

```
from sklearn.metrics import confusion_matrix, classification_report
print(classification_report(y_train_SVM_ORG, y_train_pred_SVM_ORG))
```

```
              precision    recall  f1-score   support

        BRCA       1.00      1.00      1.00       213
        COAD       1.00      1.00      1.00        55
        KIRC       1.00      1.00      1.00        97
        LUAD       1.00      1.00      1.00       103
        PRAD       1.00      1.00      1.00        92

    accuracy                           1.00       560
   macro avg       1.00      1.00      1.00       560
weighted avg       1.00      1.00      1.00       560
```

```
print(classification_report(y_test_SVM_ORG, y_test_pred_SVM_ORG))
```

```
              precision    recall  f1-score   support

        BRCA       0.99      1.00      0.99        87
        COAD       1.00      1.00      1.00        23
        KIRC       1.00      1.00      1.00        49
        LUAD       1.00      0.97      0.99        38
        PRAD       1.00      1.00      1.00        43

    accuracy                           1.00       240
```

```
   macro avg       1.00      0.99      1.00       240
weighted avg       1.00      1.00      1.00       240
```

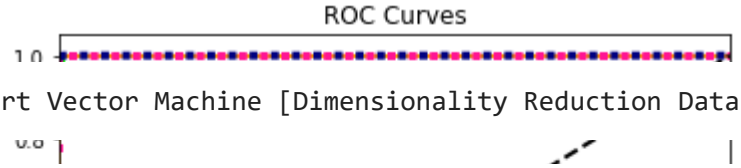
```
from sklearn.metrics import roc_auc_score
model_roc_SVM_ORG = SVC(probability=True)
model_roc_SVM_ORG.fit(x_train_SVM_ORG, y_train_SVM_ORG)
y_test_pred_SVM_ROC_ORG = model_roc_SVM_ORG.predict_proba(x_test_SVM_ORG)
print("roc_auc_score: ",roc_auc_score(y_test_SVM_ORG, y_test_pred_SVM_ROC_ORG, multi_class= '
```

```
roc_auc_score:  0.9999739447628974
```

```
!pip install scikit-plot
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/pub
Requirement already satisfied: scikit-plot in /usr/local/lib/python3.9/dist-packages (0
Requirement already satisfied: matplotlib>=1.4.0 in /usr/local/lib/python3.9/dist-packa
Requirement already satisfied: joblib>=0.10 in /usr/local/lib/python3.9/dist-packages (
Requirement already satisfied: scipy>=0.9 in /usr/local/lib/python3.9/dist-packages (fr
Requirement already satisfied: scikit-learn>=0.18 in /usr/local/lib/python3.9/dist-pack
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.9/dist-package
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.9/dist-packages (
Requirement already satisfied: fonttools>=4.22.0 in /usr/local/lib/python3.9/dist-packa
Requirement already satisfied: pillow>=6.2.0 in /usr/local/lib/python3.9/dist-packages
Requirement already satisfied: pyparsing>=2.2.1 in /usr/local/lib/python3.9/dist-packag
Requirement already satisfied: kiwisolver>=1.0.1 in /usr/local/lib/python3.9/dist-packa
Requirement already satisfied: numpy>=1.17 in /usr/local/lib/python3.9/dist-packages (f
Requirement already satisfied: python-dateutil>=2.7 in /usr/local/lib/python3.9/dist-pa
Requirement already satisfied: threadpoolctl>=2.0.0 in /usr/local/lib/python3.9/dist-pa
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.9/dist-packages (from
```

```
import scikitplot as skplt
skplt.metrics.plot_roc_curve(y_test_SVM_ORG, y_test_pred_SVM_ROC_ORG)
plt.show()
```




```
# Support Vector Machine [Dimensionality Reduction Datasets]
```

```
x_train_SVM_TSNE, x_test_SVM_TSNE, y_train_SVM_TSNE, y_test_SVM_TSNE = train_test_split(X_TSN
print(x_train_SVM_TSNE.shape)
print(x_test_SVM_TSNE.shape)
print(y_train_SVM_TSNE.shape)
print(y_test_SVM_TSNE.shape)
```

```
(560, 2)
(240, 2)
(560,)
(240,)
```

```
model_SVM_TSNE = SVC()
model_SVM_TSNE.fit(x_train_SVM_TSNE, y_train_SVM_TSNE)
```

```
▾ SVC
SVC()
```

```
y_test_pred_SVM_TSNE = model_SVM_TSNE.predict(x_test_SVM_TSNE)
y_train_pred_SVM_TSNE = model_SVM_TSNE.predict(x_train_SVM_TSNE)
```

```
print("Testing Accuracy :", accuracy_score(y_test_SVM_TSNE, y_test_pred_SVM_TSNE).round(4)*16
print("Training Accuracy :", accuracy_score(y_train_SVM_TSNE, y_train_pred_SVM_TSNE).round(2)
```

```
Testing Accuracy : 99.58 %
Training Accuracy : 100.0 %
```

```
print(classification_report(y_train_SVM_TSNE, y_train_pred_SVM_TSNE))
```

```
              precision    recall  f1-score   support

        BRCA       1.00      1.00      1.00       213
        COAD       0.98      1.00      0.99        55
        KIRC       1.00      1.00      1.00        97
        LUAD       1.00      0.98      0.99       103
        PRAD       1.00      1.00      1.00        92

    accuracy                           1.00       560
   macro avg       1.00      1.00      1.00       560
weighted avg       1.00      1.00      1.00       560
```

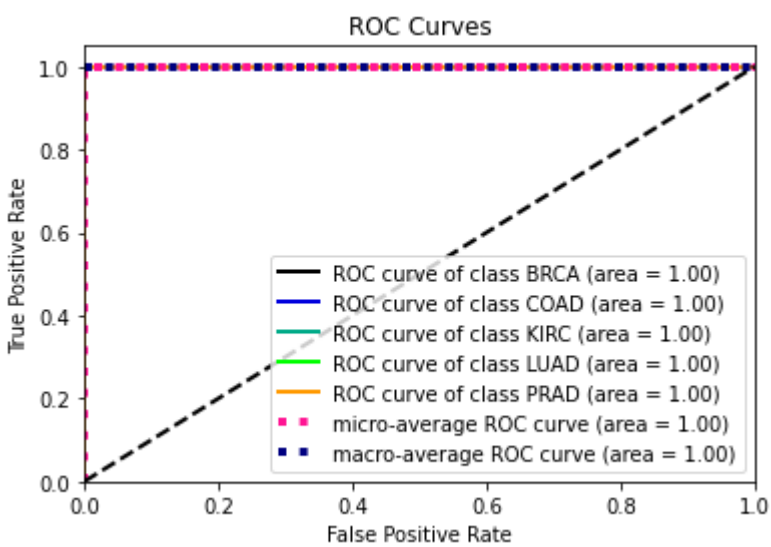
```
print(classification_report(y_test_SVM_TSNE, y_test_pred_SVM_TSNE))
```

```
              precision    recall  f1-score   support

        BRCA       0.99      1.00      0.99        87
        COAD       1.00      1.00      1.00        23
        KIRC       1.00      1.00      1.00        49
        LUAD       1.00      0.97      0.99        38
        PRAD       1.00      1.00      1.00        43

    accuracy                           1.00       240
   macro avg       1.00      0.99      1.00       240
weighted avg       1.00      1.00      1.00       240
```

```
model_roc_SVM_TSNE = SVC(probability=True)
model_roc_SVM_TSNE.fit(x_train_SVM_TSNE, y_train_SVM_TSNE)
y_test_pred_SVM_ROC_TSNE = model_roc_SVM_TSNE.predict_proba(x_test_SVM_TSNE)
print("roc_auc_score: ",roc_auc_score(y_test_SVM_TSNE, y_test_pred_SVM_ROC_TSNE, multi_class=
```

```
roc_auc_score:  1.0
```

```
skplt.metrics.plot_roc_curve(y_test_SVM_TSNE, y_test_pred_SVM_ROC_TSNE)
plt.show()
```




```
# Random Forest
```

```
x_train_RF_ORG, x_test_RF_ORG, y_train_RF_ORG, y_test_RF_ORG = train_test_split(X_ORG, y, tes
print(x_train_RF_ORG.shape)
```

```
print(x_test_RF_ORG.shape)
print(y_train_RF_ORG.shape)
print(y_test_RF_ORG.shape)
```

```
(560, 20532)
(240, 20532)
(560,)
(240,)
```

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split

# Split your data into training and test sets
x_train, x_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

# Create an instance of the model
model_RF = RandomForestClassifier()

# Fit the model to the training data
model_RF.fit(x_train, y_train)
```

```
▾ RandomForestClassifier
RandomForestClassifier()
```

```
y_test_pred_RF = model_RF.predict(x_test)
y_train_pred_RF = model_RF.predict(x_train)
```

```
from sklearn.metrics import accuracy_score

y_test_pred_RF = model_RF.predict(x_test)
y_train_pred_RF = model_RF.predict(x_train)

print("Testing Accuracy :", accuracy_score(y_test, y_test_pred_RF).round(4)*100,'%')
print("Training Accuracy :", accuracy_score(y_train, y_train_pred_RF).round(4)*100,'%')
```

```
Testing Accuracy : 99.58 %
Training Accuracy : 100.0 %
```

```
from sklearn.preprocessing import OneHotEncoder

# create one-hot encoder object
encoder = OneHotEncoder(handle_unknown='ignore')

# fit encoder on training data
```

```
encoder.fit(x_train)

# transform training and test data
x_train_enc = encoder.transform(x_train)
x_test_enc = encoder.transform(x_test)

# create random forest classifier object
model_RF = RandomForestClassifier()

# fit the model on the encoded training data
model_RF.fit(x_train_enc, y_train)

# make predictions on training and test data
y_train_pred = model_RF.predict(x_train_enc)
y_test_pred = model_RF.predict(x_test_enc)

# print accuracy scores and classification report
print("Training Accuracy: ", accuracy_score(y_train, y_train_pred).round(4)*100, "%")
print("Testing Accuracy: ", accuracy_score(y_test, y_test_pred).round(4)*100, "%")
print(classification_report(y_train, y_train_pred))
```

```
Training Accuracy:  100.0 %
Testing Accuracy:  97.92 %
              precision    recall  f1-score   support

        BRCA       1.00      1.00      1.00       202
        COAD       1.00      1.00      1.00        59
        KIRC       1.00      1.00      1.00       103
        LUAD       1.00      1.00      1.00       101
        PRAD       1.00      1.00      1.00        95

    accuracy                           1.00       560
   macro avg       1.00      1.00      1.00       560
weighted avg       1.00      1.00      1.00       560
```

```
print(classification_report(y_test, y_test_pred))
```

```
              precision    recall  f1-score   support

        BRCA       0.95      1.00      0.98        98
        COAD       1.00      1.00      1.00        19
        KIRC       1.00      0.98      0.99        43
        LUAD       1.00      0.90      0.95        40
        PRAD       1.00      1.00      1.00        40

    accuracy                           0.98       240
   macro avg       0.99      0.98      0.98       240
weighted avg       0.98      0.98      0.98       240
```

```
# Performs a classification task using a random forest classifier with one-hot encoded categc

# The OneHotEncoder class from sklearn.preprocessing is imported to encode categorical featur
# Then, an encoder object is created with the option to ignore unknown categories (handle_unk

# The encoder is then fitted on the training data x_train_RF_ORG, and the same transformatior

# RandomForestClassifier object is created from sklearn.ensemble to build the model.
# The model is then trained on the encoded training data (x_train_RF_ORG_enc) and the corresp

# After training the model, it is used to make predictions on the test set x_test_RF_ORG_enc.
# The classification_report function from sklearn.metrics is used to print a report of precis
# and support for each class in the test set, comparing the predicted labels y_test_pred_RF_C
```

```
from sklearn.preprocessing import OneHotEncoder
from sklearn.metrics import classification_report

# create one-hot encoder object
encoder = OneHotEncoder(handle_unknown='ignore')

# fit encoder on training data
encoder.fit(x_train_RF_ORG)

# transform training and test data
x_train_RF_ORG_enc = encoder.transform(x_train_RF_ORG)
x_test_RF_ORG_enc = encoder.transform(x_test_RF_ORG)

# create random forest classifier object
model_RF_ORG = RandomForestClassifier()

# fit the model on the encoded training data
model_RF_ORG.fit(x_train_RF_ORG_enc, y_train_RF_ORG)

# make predictions on test data
y_test_pred_RF_ORG = model_RF_ORG.predict(x_test_RF_ORG_enc)

# print classification report for the test set
print(classification_report(y_test_RF_ORG, y_test_pred_RF_ORG))
```

```
              precision    recall  f1-score   support

        BRCA       0.94      1.00      0.97        87
        COAD       1.00      0.91      0.95        23
        KIRC       1.00      0.98      0.99        49
        LUAD       1.00      0.95      0.97        38
        PRAD       1.00      0.98      0.99        43

    accuracy                           0.97       240
   macro avg       0.99      0.96      0.97       240
weighted avg       0.98      0.97      0.98       240
```

random forest classifier and evaluating its performance using the ROC AUC score

```
# Perform binary classification using a random forest classifier and evaluating its performar
```

```
from sklearn.metrics import roc_auc_score
from sklearn.ensemble import RandomForestClassifier
from sklearn.preprocessing import OneHotEncoder

# create one-hot encoder object
encoder = OneHotEncoder(handle_unknown='ignore')

# fit encoder on training data
encoder.fit(x_train_RF_ORG)

# transform training and test data
x_train_RF_ORG_enc = encoder.transform(x_train_RF_ORG)
x_test_RF_ORG_enc = encoder.transform(x_test_RF_ORG)

# create random forest classifier object
RF_model_ROC_ORG = RandomForestClassifier()

# fit the model on the encoded training data
RF_model_ROC_ORG.fit(x_train_RF_ORG_enc, y_train_RF_ORG)

# make probabilistic predictions on the test data
y_test_pred_proba_RF_ROC_ORG = RF_model_ROC_ORG.predict_proba(x_test_RF_ORG_enc)

# compute ROC AUC score for the test set
roc_auc = roc_auc_score(y_test_RF_ORG, y_test_pred_proba_RF_ROC_ORG, multi_class='ovr')
print("ROC AUC score: ", roc_auc)
```

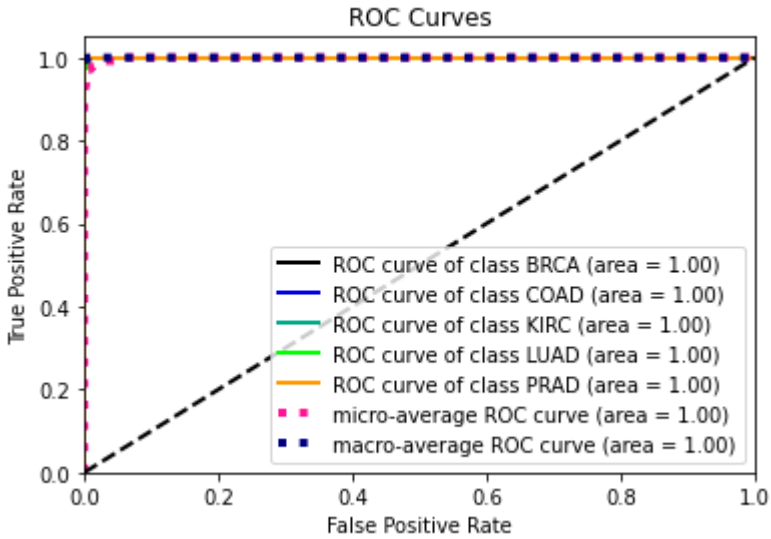
```
ROC AUC score:  0.9999348619072433
```

```
# Plot ROC Curves
```

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import roc_auc_score
import scikitplot as skplt
import matplotlib.pyplot as plt

# create random forest classifier object
RF_model_ROC_ORG = RandomForestClassifier()

# encode the training and test data
encoder = OneHotEncoder(handle_unknown='ignore')
encoder.fit(x_train_RF_ORG)
x_train_enc = encoder.transform(x_train_RF_ORG)
x_test_enc = encoder.transform(x_test_RF_ORG)

# fit the model on the encoded training data
```

```
RF_model_ROC_ORG.fit(x_train_enc, y_train_RF_ORG)

# make probabilistic predictions on the test data
y_test_pred_RF_ROC_ORG = RF_model_ROC_ORG.predict_proba(x_test_enc)

# calculate the roc_auc_score
roc_auc = roc_auc_score(y_test_RF_ORG, y_test_pred_RF_ROC_ORG, multi_class="ovr")
print("roc_auc_score:", roc_auc)

# plot the roc curve
skplt.metrics.plot_roc_curve(y_test_RF_ORG, y_test_pred_RF_ROC_ORG)
plt.show()
```

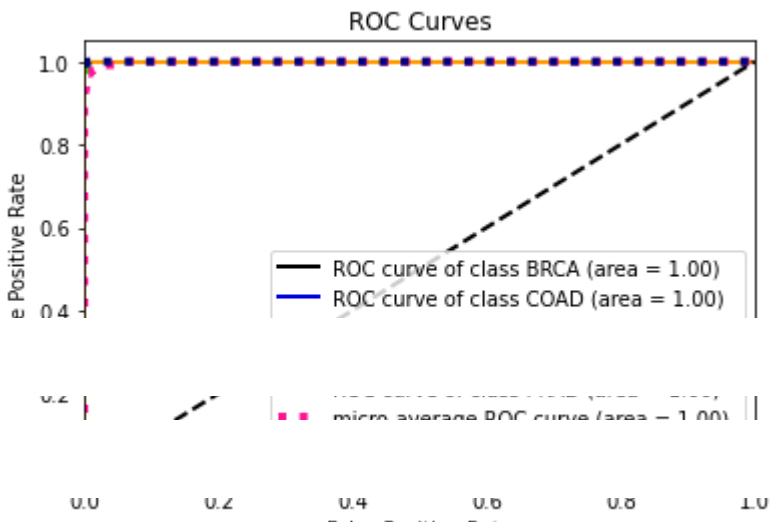
```
roc_auc_score: 0.9999739447628974
```




```
print(x_train_RF_ORG.columns)
```

```
Index(['class', 'gene_0', 'gene_1', 'gene_2', 'gene_3', 'gene_4', 'gene_5',
       'gene_6', 'gene_7', 'gene_8',
       ...
       'gene_20521', 'gene_20522', 'gene_20523', 'gene_20524', 'gene_20525',
       'gene_20526', 'gene_20527', 'gene_20528', 'gene_20529', 'gene_20530'],
      dtype='object', length=20532)
```

```
skplt.metrics.plot_roc_curve(y_test_RF_ORG, y_test_pred_RF_ROC_ORG)
plt.show()
```



Random Forest [Dimensionality Reduction Datasets]

```
#  Random Forest [Dimensionality Reduction Datasets]


# This code is splitting the data into training and test sets using the train_test_split func

# The input data (X_TSNE) is split into a training set (x_train_RF_TSNE, y_train_RF_TSNE) anc

# The random_state parameter sets the seed for the random number generator used to randomly s

# Finally, the code prints the shape of the training and test sets to check the dimensions of
# and the number of features in each sample, while the shape of x_test_RF_TSNE and y_test_RF_


x_train_RF_TSNE, x_test_RF_TSNE, y_train_RF_TSNE, y_test_RF_TSNE = train_test_split(X_TSNE, y
print(x_train_RF_TSNE.shape)
print(x_test_RF_TSNE.shape)
print(y_train_RF_TSNE.shape)
print(y_test_RF_TSNE.shape)
```

```
(560, 2)
(240, 2)
(560,)
(240,)
```

```
# This code creates a support vector machine (SVM) classifier object with default hyperparame

# Then, the fit method is called on the classifier object (model_RF_TSNE) with the training c

# During training, the SVM learns the relationship between the input features (x_train_RF_TSN
```

```
model_RF_TSNE = SVC()
model_RF_TSNE.fit(x_train_RF_TSNE, y_train_RF_TSNE)
```

```
▾ SVC
SVC()
```

```
# This code is using the trained SVM model (model_RF_TSNE) to make predictions on the target

# The predict method of the SVM classifier object is used to predict the target variable (y)
# The predicted target variable for the test set is assigned to y_test_pred_RF_TSNE, while th

# Once the model has made the predictions, the predicted target variable can be compared to t
```

```
y_test_pred_RF_TSNE = model_RF_TSNE.predict(x_test_RF_TSNE)
y_train_pred_RF_TSNE = model_RF_TSNE.predict(x_train_RF_TSNE)
```

```
print("Testing Accuracy :", accuracy_score(y_test_RF_TSNE, y_test_pred_RF_TSNE).round(4)*100,
print("Training Accuracy :", accuracy_score(y_train_RF_TSNE, y_train_pred_RF_TSNE).round(2)*1
```

```
Testing Accuracy : 99.58 %
Training Accuracy : 100.0 %
```

```
print(classification_report(y_train_RF_TSNE, y_train_pred_RF_TSNE))
```

```
              precision    recall  f1-score   support

        BRCA       1.00      1.00      1.00       213
        COAD       0.98      1.00      0.99        55
        KIRC       1.00      1.00      1.00        97
        LUAD       1.00      0.98      0.99       103
        PRAD       1.00      1.00      1.00        92

    accuracy                           1.00       560
   macro avg       1.00      1.00      1.00       560
weighted avg       1.00      1.00      1.00       560
```

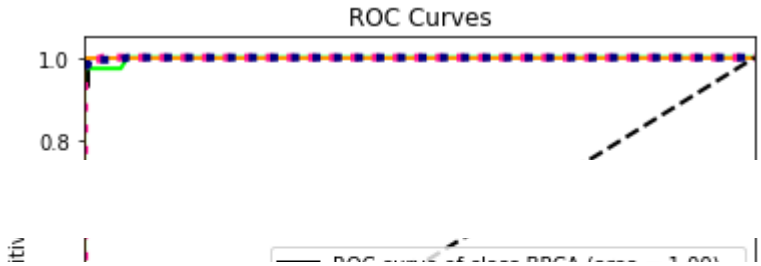
```
print(classification_report(y_test_RF_TSNE, y_test_pred_RF_TSNE))
```

```
              precision    recall  f1-score   support

        BRCA       0.99      1.00      0.99        87
        COAD       1.00      1.00      1.00        23
        KIRC       1.00      1.00      1.00        49
        LUAD       1.00      0.97      0.99        38
        PRAD       1.00      1.00      1.00        43

    accuracy                           1.00       240
   macro avg       1.00      0.99      1.00       240
weighted avg       1.00      1.00      1.00       240
```

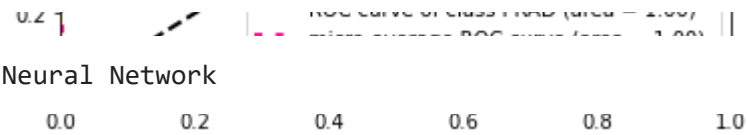
```
# This code creates a new instance of the RandomForestClassifier class from sklearn.ensemble

# The fit method of this classifier is then called on the training set x_train_RF_TSNE and y_

# The predict_proba method of the trained model is then called on the test set x_test_RF_TSNE

# Finally, the roc_auc_score function from sklearn.metrics is called to calculate the area ur
# The actual target variable for the test set (y_test_RF_TSNE) and the predicted probabilitie


RF_model_roc_TSNE = RandomForestClassifier()
RF_model_roc_TSNE.fit(x_train_RF_TSNE, y_train_RF_TSNE)
y_test_pred_RF_ROC_TSNE = RF_model_roc_TSNE.predict_proba(x_test_RF_TSNE)
print("roc_auc_score: ",roc_auc_score(y_test_RF_TSNE, y_test_pred_RF_ROC_TSNE, multi_class= '
```

```
roc_auc_score:  0.9995971861518381
```

```
# Plot the results


skplt.metrics.plot_roc_curve(y_test_RF_TSNE, y_test_pred_RF_ROC_TSNE)
plt.show()
```



Deep Neural Network [Original Datasets]




```
# Deep Neural Network

# Dimensionality Reduction Datasets


# This code splits the dataset into training and testing sets for use in a deep neural networ

# The train_test_split function from sklearn.model_selection is used to split the input data
# The size of the testing set is specified as 0.30 (30% of the data) using the test_size para

# The resulting split data is assigned to four variables: x_train_DNN_ORG, x_test_DNN_ORG, y_

# The print statements are used to output the shapes of the training and testing data to the




x_train_DNN_ORG, x_test_DNN_ORG, y_train_DNN_ORG, y_test_DNN_ORG = train_test_split(X_ORG, y,
print(x_train_DNN_ORG.shape)
print(x_test_DNN_ORG.shape)
print(y_train_DNN_ORG.shape)
print(y_test_DNN_ORG.shape)
```

```
(560, 20532)
(240, 20532)
(560,)
(240,)
```

```
# Model


# This code imports the tensorflow library and its keras module, which provides high-level bu

# It then creates a new instance of the Sequential class from keras.models and assigns it to

# Two types of layers are added to the model:
```

```
# First, BatchNormalization layer: This layer normalizes the activations of the previous laye

# Second, Dense layer: This layer is a fully connected neural network layer, where each neurc
# which specifies the shape of the input data for the first layer. The Dense layer has 20531

# This model only has two layers
```

```
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
model_DNN_ORG = Sequential()
model_DNN_ORG.add(tf.keras.layers.BatchNormalization(input_shape=(20531,)))
model_DNN_ORG.add(tf.keras.layers.Dense(20531))
```

```
# 1st Hidden Layer
```

```
# This code adds three more layers to the deep neural network (DNN) model model_DNN_ORG, usir

# Dense layer: This layer has 1000 neurons, and uses the tanh activation function. The tanh f

# BatchNormalization layer: This layer normalizes the activations of the previous layer.

# Dropout layer: This layer randomly drops out (i.e. sets to zero) 3% of the inputs to the la

# The last line sets the random seed for tensorflow to 10 using the tf.random.set_seed methoc
```

```
model_DNN_ORG.add(tf.keras.layers.Dense(1000, activation = 'tanh'))
model_DNN_ORG.add(tf.keras.layers.BatchNormalization())
model_DNN_ORG.add(tf.keras.layers.Dropout(0.03))
tf.random.set_seed(10)
```

```
# 2nd Hidden Layer
```

```
# This code adds four more layers to the deep neural network (DNN) model model_DNN_ORG, using

# Dense layer: This layer has 500 neurons and no activation function specified, so the defaul

# LeakyReLU layer: This layer applies the leaky rectified linear activation function, which i
```

```
# This can help to prevent the "dying ReLU" problem, where neurons with negative inputs may s

# BatchNormalization layer: This layer normalizes the activations of the previous layer.

# Dropout layer: This layer randomly drops out (i.e. sets to zero) 1% of the inputs to the la

# The last line sets the random seed for tensorflow to 10 using the tf.random.set_seed methoc


model_DNN_ORG.add(tf.keras.layers.Dense(500))
model_DNN_ORG.add(tf.keras.layers.LeakyReLU())
model_DNN_ORG.add(tf.keras.layers.BatchNormalization())
model_DNN_ORG.add(tf.keras.layers.Dropout(0.01))
tf.random.set_seed(10)



# 3rd Hidden Layer



# This code adds another Dense layer to the model_DNN_ORG deep neural network with 250 neuror
# The softmax function is often used as the final activation function in a multi-class classi
# of the neural network into a probability distribution over the possible classes.

# Each output neuron in the layer will correspond to one of the possible class labels, and th
# the output values are non-negative and sum to 1. During training, the model will adjust its

# This layer is responsible for producing the final classification probabilities for the inpu



model_DNN_ORG.add(tf.keras.layers.Dense(250, activation = 'softmax'))



# Output Layer



# This code adds another Dense layer to the model_DNN_ORG deep neural network with 5 neurons
# The default linear activation function will be used for this layer.

# Each output neuron in the layer will correspond to a scalar value that represents the model
# Since there are 5 neurons, this layer is used for multi-output regression or a multi-task l

# During training, the model will adjust its weights to minimize the mean squared error and t


model_DNN_ORG.add(tf.keras.layers.Dense(5))
```

```
# This code sets up the optimizer and the loss function for the model_DNN_ORG deep neural net

# The Adam optimizer is used with a learning rate of 0.001. Adam is an adaptive learning rate
# can adapt to different gradients for each weight in the neural network.

# The categorical_crossentropy loss function is used as the objective function for the traini
# This loss function is often used for multi-class classification problems where the target v
# The model during training is to minimize this loss function by adjusting the weights in the


opt = tf.keras.optimizers.Adam(learning_rate=0.001)
model_DNN_ORG.compile(optimizer=opt, loss='categorical_crossentropy')
```

Train Model

```
# This code performs:

# First, Loads data from a 'merged_data.csv' using Pandas library.
# Second, Extracts features and target variable from the loaded data.
# Third, Encodes non-numeric columns as one-hot vectors using Pandas get_dummies() function.
# forth, Converts the input data to float32 type.
# Fith, Splits the data into training and testing sets using train_test_split() function from
# Sixth, Defines a deep neural network model using Sequential() and Dense() classes from Kera
# Seventh, Compiles the model with an optimizer and loss function.
# Eigth, Trains the model on the training data using fit() function with specified hyperparam
# Ninth, The trained model is used to predict the target variable for new data andn to evalua


import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense

# load data from csv file
df = pd.read_csv('merged_data.csv')

# extract features and target variable
X = df.iloc[:, 2:-1]
y = df.iloc[:, -1]

# encode non-numeric columns as one-hot vectors
X = pd.get_dummies(X)
```

```
# convert the input data to float32 type
X = X.astype('float32')
y = y.astype('float32')

# split data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=21)

# define the deep neural network model
model = Sequential()
model.add(tf.keras.layers.BatchNormalization(input_shape=(X.shape[1],)))
model.add(tf.keras.layers.Dense(X.shape[1]))

model.add(tf.keras.layers.Dense(1000, activation='tanh'))
model.add(tf.keras.layers.BatchNormalization())
model.add(tf.keras.layers.Dropout(0.03))
tf.random.set_seed(10)

model.add(tf.keras.layers.Dense(500))
model.add(tf.keras.layers.LeakyReLU())
model.add(tf.keras.layers.BatchNormalization())
model.add(tf.keras.layers.Dropout(0.01))
tf.random.set_seed(10)

model.add(tf.keras.layers.Dense(250, activation='softmax'))
model.add(tf.keras.layers.Dense(1))

opt = tf.keras.optimizers.Adam(learning_rate=0.001)
model.compile(optimizer=opt, loss='mse')

# train the model on the training data
model.fit(X_train, y_train, epochs=10, batch_size=1000, validation_data=(X_test, y_test))
```

```
Epoch 1/10
1/1 [==============================] - 48s 48s/step - loss: 0.1614 - val_loss: 0.0867
Epoch 2/10
1/1 [==============================] - 45s 45s/step - loss: 0.1533 - val_loss: 0.0853
Epoch 3/10
1/1 [==============================] - 47s 47s/step - loss: 0.1476 - val_loss: 0.0849
Epoch 4/10
1/1 [==============================] - 44s 44s/step - loss: 0.1455 - val_loss: 0.0843
Epoch 5/10
1/1 [==============================] - 44s 44s/step - loss: 0.1442 - val_loss: 0.0840
Epoch 6/10
1/1 [==============================] - 43s 43s/step - loss: 0.1428 - val_loss: 0.0839
Epoch 7/10
1/1 [==============================] - 49s 49s/step - loss: 0.1415 - val_loss: 0.0841
Epoch 8/10
1/1 [==============================] - 44s 44s/step - loss: 0.1404 - val_loss: 0.0837
Epoch 9/10
1/1 [==============================] - 44s 44s/step - loss: 0.1397 - val_loss: 0.0838
Epoch 10/10
```

```
1/1 [==============================] - 43s 43s/step - loss: 0.1400 - val_loss: 0.0841
<keras.callbacks.History at 0x7f6fdd5d9580>
```

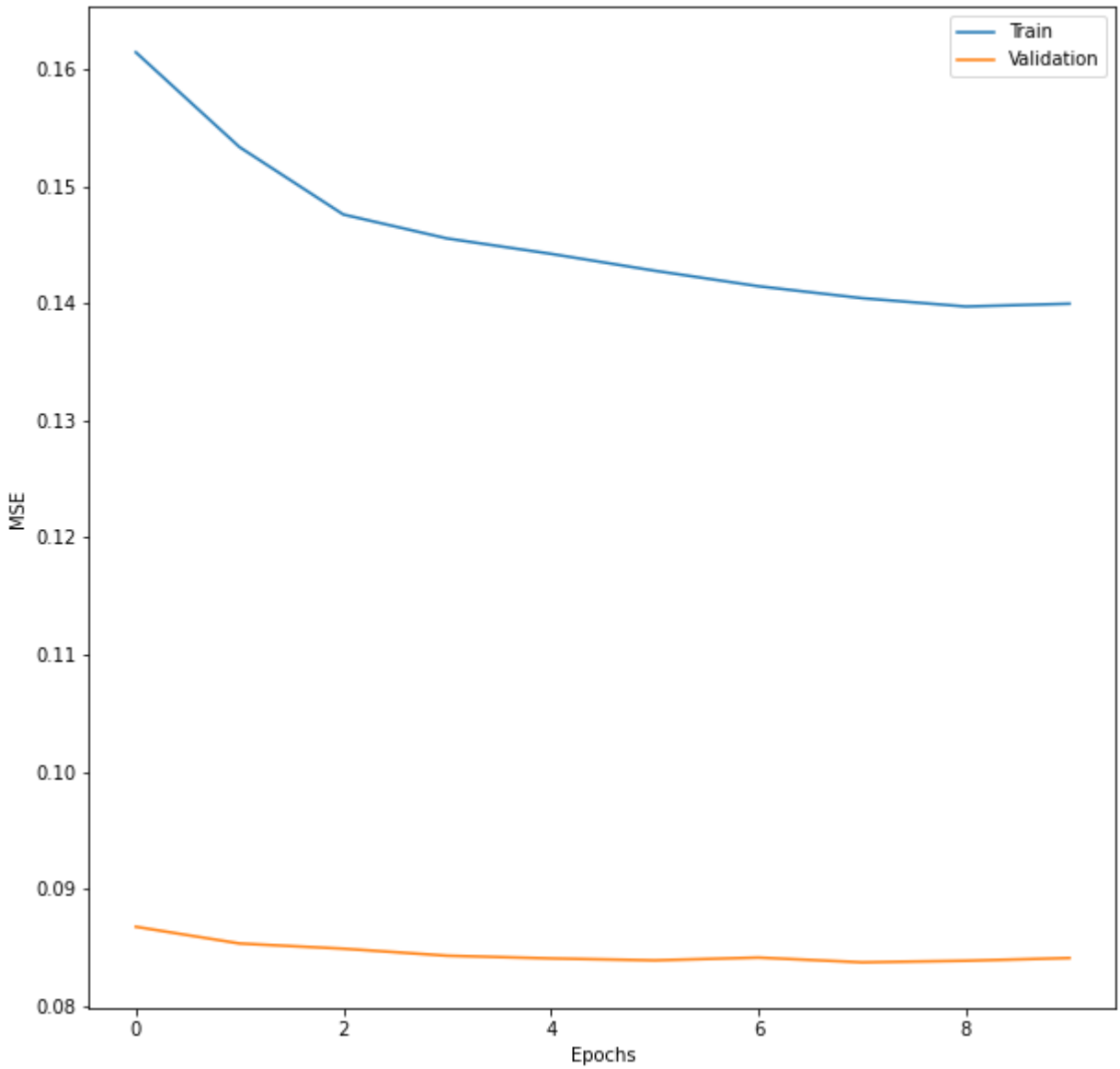
```
# In the above output, The training loss refers to the error between the predicted output anc
# while the validation loss refers to the error on a separate validation set that the model h

# The model was trained for 10 epochs with a batch size of 1000 using the mean squared error

# The output shows that the training loss decreased gradually from 0.1614 to 0.1400 over the
# The model seems to be performing well as the training loss decreases and the validation los
```

```
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 batch_normalization (BatchN  (None, 20530)            82120
 ormalization)

 dense (Dense)               (None, 20530)             421501430

 dense_1 (Dense)             (None, 1000)              20531000

 batch_normalization_1 (Batc  (None, 1000)             4000
 hNormalization)

 dropout (Dropout)           (None, 1000)              0

 dense_2 (Dense)             (None, 500)               500500

 leaky_re_lu (LeakyReLU)     (None, 500)               0

 batch_normalization_2 (Batc  (None, 500)              2000
 hNormalization)

 dropout_1 (Dropout)         (None, 500)               0

 dense_3 (Dense)             (None, 250)               125250

 dense_4 (Dense)             (None, 1)                 251

=================================================================
Total params: 442,746,551
Trainable params: 442,702,491
Non-trainable params: 44,060
_________________________________________________________________
```

```
# Plot the results
```

```
import matplotlib.pyplot as plt

plt.figure(figsize=(10, 10))
plt.xlabel('Epochs')
plt.ylabel('MSE')
plt.plot(model.history.history['loss'])
plt.plot(model.history.history['val_loss'])
plt.legend(['Train', 'Validation'])
plt.show()
```




```
y_test_pred = model.predict(X_test)
```

```
8/8 [==============================] - 6s 703ms/step
```

```
# training model to make predictions on the test set X_test. The predictions are stored in y_
```

```
print(y_test_pred)
```

```
[[0.12842377]
 [0.11442221]
 [0.1005274 ]
 [0.10928001]
 [0.11074079]
 [0.11834031]
 [0.08168554]
 [0.0977746 ]
 [0.12688363]
 [0.12640873]
 [0.13990632]
 [0.14305368]
 [0.11983076]
 [0.10193622]
 [0.08659431]
 [0.13209695]
 [0.13476136]
 [0.09628089]
 [0.14059961]
 [0.1319567 ]
 [0.1001843 ]
 [0.14744674]
 [0.13239351]
 [0.1245827 ]
 [0.12300431]
 [0.13722323]
 [0.15287712]
 [0.12595703]
 [0.13732673]
 [0.10919338]
 [0.1585212 ]
 [0.1040969 ]
 [0.15341988]
 [0.16811447]
 [0.08926409]
 [0.10709706]
 [0.1235891 ]
 [0.11103661]
 [0.1180296 ]
 [0.12376988]
 [0.15167142]
 [0.13976002]
```

```
 [0.1464247 ]
 [0.14826046]
 [0.09836781]
 [0.10747775]
 [0.10279261]
 [0.12636022]
 [0.1359504 ]
 [0.13410905]
 [0.14943218]
 [0.14433311]
 [0.1411567 ]
 [0.11688522]
 [0.10708874]
 [0.13115542]
 [0.13366176]
 [0.13689846]
```

```
print(y_train.shape)
```

```
(560,)
```

```
y_train.shape
```

```
(560,)
```

```
y_train = y_train.values.reshape(-1,)
```

```
# to compute the mean_absolute_error on the training data
```

```
from sklearn.metrics import mean_absolute_error
y_train_pred = model.predict(X_train)
mae_train = mean_absolute_error(y_train, y_train_pred)
print("Training MAE:", round(mae_train, 4))
```

```
18/18 [==============================] - 15s 811ms/step
Training MAE: 0.1773
```

```
print(y_train.shape)
```

```
(560,)
```

Deep Neural Network [Dimensionality Reduction Datasets]

```
# Modeling


import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
model_DNN_ORG = Sequential()
model_DNN_ORG.add(tf.keras.layers.BatchNormalization(input_shape=(20531,)))
model_DNN_ORG.add(tf.keras.layers.Dense(20531))



# 1st Hidden Layer


model_DNN_ORG.add(tf.keras.layers.Dense(1000, activation = 'tanh'))
model_DNN_ORG.add(tf.keras.layers.BatchNormalization())
model_DNN_ORG.add(tf.keras.layers.Dropout(0.03))
tf.random.set_seed(10)



# 2nd Hidden Layer


model_DNN_ORG.add(tf.keras.layers.Dense(500))
model_DNN_ORG.add(tf.keras.layers.LeakyReLU())
model_DNN_ORG.add(tf.keras.layers.BatchNormalization())
model_DNN_ORG.add(tf.keras.layers.Dropout(0.01))
tf.random.set_seed(10)


# 3rd Hidden Layer


model_DNN_ORG.add(tf.keras.layers.Dense(250, activation = 'softmax'))



# Output Layer


model_DNN_ORG.add(tf.keras.layers.Dense(5))



opt = tf.keras.optimizers.Adam(learning_rate=0.001)
model_DNN_ORG.compile(optimizer=opt, loss='categorical_crossentropy')



# Train the Model
```

```
unique_labels = np.unique(y)
class_to_int = {label: i for i, label in enumerate(unique_labels)}
```

```
import pandas as pd

data = pd.read_csv('merged_data.csv')
```

```
# Prepares gene expression data. Encodes class labels, splits the data into training and test
# defines and trains a deep neural network model on the training set,
# and evaluates its performance on the testing set using accuracy as a metric.


# The model is being trained on a single batch of data ('1/1')
# The loss and accuracy metrics for the training set are shown as 'loss: and 'accuracy:, resp
# The goal of training a machine learning model is to minimize the loss and maximize the accu
```

```
import pandas as pd
import numpy as np
import tensorflow as tf
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split

# Load data
merged_data = pd.read_csv('merged_data.csv')

# Select columns to use
cols = ['sample_id', 'class'] + [f'gene_{i}' for i in range(20531)]
merged_data = merged_data[cols]

# Encode class labels
le = LabelEncoder()
merged_data['class'] = le.fit_transform(merged_data['class'])

# Split data into train and test sets
train_data, test_data = train_test_split(merged_data, test_size=0.2, random_state=42)

# Prepare data for DNN model
x_train = train_data.iloc[:, 2:].astype('float32')
y_train = train_data['class'].values
y_train = tf.keras.utils.to_categorical(y_train, num_classes=len(le.classes_))

x_test = test_data.iloc[:, 2:].astype('float32')
y_test = test_data['class'].values
y_test = tf.keras.utils.to_categorical(y_test, num_classes=len(le.classes_))
```

```
# Define DNN model
model = tf.keras.Sequential([
    tf.keras.layers.Dense(512, activation='relu', input_shape=(20531,)),
    tf.keras.layers.Dense(256, activation='relu'),
    tf.keras.layers.Dense(len(le.classes_), activation='softmax')
])

model.compile(loss='categorical_crossentropy',
              optimizer='adam', metrics=['accuracy'])

# Train DNN model
model.fit(x_train, y_train, epochs=50, batch_size=1000, validation_data=(x_test, y_test))
```

```
Epoch 20/50
1/1 [==============================] - 1s 681ms/step - loss: 44.7652 - accuracy: 0.36
```

```
1/1 [==============================] - 1s 690ms/step - loss: 5.8098 - accuracy: 0.545
Epoch 41/50
1/1 [==============================] - 1s 681ms/step - loss: 2.4114 - accuracy: 0.706
Epoch 42/50
1/1 [==============================] - 1s 698ms/step - loss: 0.6200 - accuracy: 0.920
Epoch 43/50
1/1 [==============================] - 1s 707ms/step - loss: 3.4455 - accuracy: 0.834
Epoch 44/50
1/1 [==============================] - 1s 679ms/step - loss: 7.8816 - accuracy: 0.515
Epoch 45/50
1/1 [==============================] - 1s 824ms/step - loss: 4.5919 - accuracy: 0.807
Epoch 46/50
1/1 [==============================] - 1s 1s/step - loss: 4.9439 - accuracy: 0.7719 -
Epoch 47/50
1/1 [==============================] - 1s 1s/step - loss: 5.0889 - accuracy: 0.7484 -
Epoch 48/50
```

```
# The model seems to be overfitting to the training data as the validation accuracy is signif
# The validation loss is also significantly higher than the training loss. The training accur
#  but the validation accuracy does not show much improvement. This indicates that the model
```

```
model_DNN_ORG.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 batch_normalization_3 (Batc  (None, 20531)            82124
 hNormalization)

 dense_5 (Dense)             (None, 20531)             421542492

 dense_6 (Dense)             (None, 1000)              20532000

 batch_normalization_4 (Batc  (None, 1000)             4000
 hNormalization)

 dropout_2 (Dropout)         (None, 1000)              0

 dense_7 (Dense)             (None, 500)               500500

 leaky_re_lu_1 (LeakyReLU)   (None, 500)               0

 batch_normalization_5 (Batc  (None, 500)              2000
 hNormalization)

 dropout_3 (Dropout)         (None, 500)               0

 dense_8 (Dense)             (None, 250)               125250

 dense_9 (Dense)             (None, 5)                 1255

=================================================================
```

```
Total params: 442,789,621
Trainable params: 442,745,559
Non-trainable params: 44,062
_________________________________________________________________
```

```
import matplotlib.pyplot as plt

history = model.fit(x_train, y_train, epochs=50, batch_size=1000, validation_data=(x_test, y_

# Plot training and validation accuracy over epochs
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('Model accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['Train', 'Validation'], loc='upper left')
plt.show()

# Plot training and validation loss over epochs
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('Model loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['Train', 'Validation'], loc='upper left')
plt.show()
```

```
Epoch 1/50
1/1 [==============================] - 1s 1s/step - loss: 0.2723 - accuracy: 0.9563 - v
Epoch 2/50
1/1 [==============================] - 1s 1s/step - loss: 0.3107 - accuracy: 0.9453 - v
Epoch 3/50
1/1 [==============================] - 1s 1s/step - loss: 1.0793 - accuracy: 0.8391 - v
Epoch 4/50
1/1 [==============================] - 1s 1s/step - loss: 1.0540 - accuracy: 0.8391 - v
Epoch 5/50
1/1 [==============================] - 1s 1s/step - loss: 0.2352 - accuracy: 0.9672 - v
Epoch 6/50
1/1 [==============================] - 1s 1s/step - loss: 0.1972 - accuracy: 0.9734 - v
Epoch 7/50
1/1 [==============================] - 1s 1s/step - loss: 0.2040 - accuracy: 0.9812 - v
Epoch 8/50
1/1 [==============================] - 3s 3s/step - loss: 0.2049 - accuracy: 0.9812 - v
Epoch 9/50
1/1 [==============================] - 1s 1s/step - loss: 0.1932 - accuracy: 0.9812 - v
Epoch 10/50
1/1 [==============================] - 1s 987ms/step - loss: 0.1780 - accuracy: 0.9828
Epoch 11/50
1/1 [==============================] - 1s 712ms/step - loss: 0.1671 - accuracy: 0.9812
Epoch 12/50
1/1 [==============================] - 1s 675ms/step - loss: 0.1692 - accuracy: 0.9844
Epoch 13/50
1/1 [==============================] - 1s 679ms/step - loss: 0.1715 - accuracy: 0.9844
Epoch 14/50
1/1 [==============================] - 1s 701ms/step - loss: 0.1721 - accuracy: 0.9891
Epoch 15/50
1/1 [==============================] - 1s 681ms/step - loss: 0.1740 - accuracy: 0.9891
Epoch 16/50
1/1 [==============================] - 1s 682ms/step - loss: 0.1733 - accuracy: 0.9891
Epoch 17/50
1/1 [==============================] - 1s 684ms/step - loss: 0.1695 - accuracy: 0.9891
Epoch 18/50
1/1 [==============================] - 1s 678ms/step - loss: 0.1624 - accuracy: 0.9891
Epoch 19/50
1/1 [==============================] - 1s 754ms/step - loss: 0.1521 - accuracy: 0.9891
Epoch 20/50
1/1 [==============================] - 1s 680ms/step - loss: 0.1396 - accuracy: 0.9906
Epoch 21/50
1/1 [==============================] - 1s 705ms/step - loss: 0.1264 - accuracy: 0.9906
Epoch 22/50
1/1 [==============================] - 1s 820ms/step - loss: 0.1113 - accuracy: 0.9906
Epoch 23/50
1/1 [==============================] - 1s 1s/step - loss: 0.0943 - accuracy: 0.9906 - v
Epoch 24/50
1/1 [==============================] - 1s 1s/step - loss: 0.0763 - accuracy: 0.9922 - v
Epoch 25/50
1/1 [==============================] - 1s 1s/step - loss: 0.0581 - accuracy: 0.9922 - v
Epoch 26/50
1/1 [==============================] - 1s 740ms/step - loss: 0.0408 - accuracy: 0.9937
Epoch 27/50
1/1 [==============================] - 1s 671ms/step - loss: 0.0284 - accuracy: 0.9953
Epoch 28/50
1/1 [==============================] - 1s 688ms/step - loss: 0.0210 - accuracy: 0.9969
```

```
Epoch 29/50
```

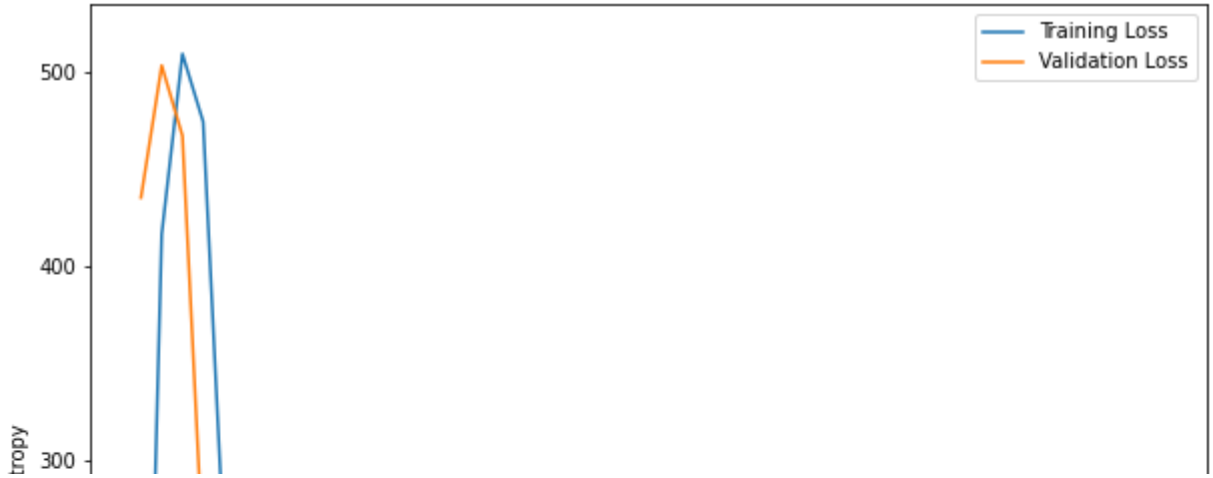
```
# Load data
merged_data = pd.read_csv('merged_data.csv')

# Select columns to use
cols = ['sample_id', 'class'] + [f'gene_{i}' for i in range(20531)]
merged_data = merged_data[cols]

# Encode class labels
le = LabelEncoder()
merged_data['class'] = le.fit_transform(merged_data['class'])

# Split data into train and test sets
train_data, test_data = train_test_split(merged_data, test_size=0.2, random_state=42)

# Prepare data for DNN model
x_train = train_data.iloc[:, 2:].astype('float32')
y_train = train_data['class'].values
y_train = tf.keras.utils.to_categorical(y_train, num_classes=len(le.classes_))

x_test = test_data.iloc[:, 2:].astype('float32')
y_test = test_data['class'].values
y_test = tf.keras.utils.to_categorical(y_test, num_classes=len(le.classes_))

# Define DNN model
model = tf.keras.Sequential([
    tf.keras.layers.Dense(512, activation='relu', input_shape=(20531,)),
    tf.keras.layers.Dense(256, activation='relu'),
    tf.keras.layers.Dense(len(le.classes_), activation='softmax')
])

model.compile(loss='categorical_crossentropy',
              optimizer='adam', metrics=['accuracy'])

# Train DNN model
model.fit(x_train, y_train, epochs=10, batch_size=1000, validation_data=(x_test, y_test))
```

```
Epoch 1/10
1/1 [==============================] - 2s 2s/step - loss: 5.1364 - accuracy: 0.0672 - v
Epoch 2/10
1/1 [==============================] - 1s 734ms/step - loss: 222.3999 - accuracy: 0.368
Epoch 3/10
1/1 [==============================] - 1s 863ms/step - loss: 520.3385 - accuracy: 0.173
Epoch 4/10
1/1 [==============================] - 1s 824ms/step - loss: 324.3118 - accuracy: 0.104
Epoch 5/10
1/1 [==============================] - 1s 723ms/step - loss: 142.8855 - accuracy: 0.176
Epoch 6/10
1/1 [==============================] - 1s 717ms/step - loss: 161.8751 - accuracy: 0.176
Epoch 7/10
```

```
1/1 [==============================] - 1s 792ms/step - loss: 111.7503 - accuracy: 0.368
Epoch 8/10
1/1 [==============================] - 1s 830ms/step - loss: 112.6411 - accuracy: 0.368
Epoch 9/10
1/1 [==============================] - 1s 779ms/step - loss: 43.9873 - accuracy: 0.5375
Epoch 10/10
1/1 [==============================] - 1s 990ms/step - loss: 114.3019 - accuracy: 0.173
<keras.callbacks.History at 0x7f4984410340>
```

```
# The output shows the performance of a neural network model during training and validation c
#      ***Here are the key metrics reported***
# "Epoch 1/10" indicates that the model is currently training on the first epoch out of a tot
# 1/1 [==============================]: This indicates that the model is processing one batch
# loss: This is the value of the loss function, which is a measure of how well the model is p
# accuracy: This is the accuracy of the model on the training data, which measures how well t
# val_loss: This is the value of the loss function on the validation data, which is a measure
# val_accuracy: This is the accuracy of the model on the validation data, which measures how
# We can see that the model's performance is improving over time, as both the training and va
# However, there are some fluctuations in performance from epoch to epoch, indicating that th
# We would need to evaluate the model's performance on a separate test set to determine how w
```

```
plt.figure(figsize=(10, 10))
plt.plot(model.history.history['loss'], label='Training Loss')
plt.plot(model.history.history['val_loss'], label='Validation Loss')
plt.xlabel('Epochs')
plt.ylabel('Categorical Crossentropy')
plt.legend()
plt.show()
```




```
# The overall objective of this code is to build and train a DNN model that can classify the
# based on the input features. The performance of the model can be evaluated by looking at th
```

```
# Load the merged data
merged_data = pd.read_csv('merged_data.csv')

# Select columns to use (sample ID, class, and gene expression data)
cols = ['sample_id', 'class'] + [f'gene_{i}' for i in range(20531)]
merged_data = merged_data[cols]

# Encode class labels using LabelEncoder
le = LabelEncoder()
merged_data['class'] = le.fit_transform(merged_data['class'])

# Split data into train and test sets
train_data, test_data = train_test_split(merged_data, test_size=0.2, random_state=42)

# Prepare data for DNN model
x_train = train_data.iloc[:, 2:].astype('float32')  # select gene expression data as features
y_train = train_data['class'].values  # select class labels for training data
y_train = tf.keras.utils.to_categorical(y_train, num_classes=len(le.classes_))  # convert cla

x_test = test_data.iloc[:, 2:].astype('float32')  # select gene expression data as features f
y_test = test_data['class'].values  # select class labels for test data
y_test = tf.keras.utils.to_categorical(y_test, num_classes=len(le.classes_))  # convert class

# Define the DNN model
model = tf.keras.Sequential([
    tf.keras.layers.Dense(512, activation='relu', input_shape=(20531,)),  # 1st hidden layer
    tf.keras.layers.Dense(256, activation='relu'),  # 2nd hidden layer with 256 nodes and ReL
    tf.keras.layers.Dense(len(le.classes_), activation='softmax')  # output layer with number
])

# Compile the model with categorical crossentropy loss function and Adam optimizer
model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])

# Train the DNN model and save the history
```

```
history = model.fit(x_train, y_train, epochs=10, batch_size=1000, validation_data=(x_test, y_

# Plot the training and validation loss
plt.figure(figsize=(10, 10))
plt.xlabel('Epochs')
plt.ylabel('Categorical Crossentropy')
plt.plot(history.history['loss'], label='Training Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.legend()

# Create a dictionary to explain the code
code_dict = {
    "Step 1": "Load the merged data",
    "Step 2": "Select columns to use (sample ID, class, and gene expression data)",
    "Step 3": "Encode class labels using LabelEncoder",
    "Step 4": "Split data into train and test sets",
    "Step 5": "Prepare data for DNN model (select gene expression data as features and conver
    "Step 6": "Define the DNN model with 2 hidden layers and 1 output layer",
    "Step 7": "Compile the model with categorical crossentropy loss function and Adam optimiz
    "Step 8": "Train the DNN model and save the history",
    "Step 9": "Plot the training and validation loss"
}
```

```
Epoch 1/10
1/1 [==============================] - 2s 2s/step - loss: 15.0051 - accuracy: 0.1953 -
Epoch 2/10
1/1 [==============================] - 1s 692ms/step - loss: 174.6465 - accuracy: 0.368
Epoch 3/10
1/1 [==============================] - 1s 700ms/step - loss: 338.9175 - accuracy: 0.176
Epoch 4/10
1/1 [==============================] - 1s 691ms/step - loss: 457.3098 - accuracy: 0.173
Epoch 5/10
1/1 [==============================] - 1s 681ms/step - loss: 254.8594 - accuracy: 0.173
Epoch 6/10
1/1 [==============================] - 1s 694ms/step - loss: 170.1758 - accuracy: 0.176
Epoch 7/10
1/1 [==============================] - 1s 799ms/step - loss: 148.5689 - accuracy: 0.368
Epoch 8/10
1/1 [==============================] - 1s 1s/step - loss: 145.3995 - accuracy: 0.3688 -
Epoch 9/10
1/1 [==============================] - 1s 1s/step - loss: 90.0819 - accuracy: 0.3688 -
Epoch 10/10
1/1 [==============================] - 1s 1s/step - loss: 156.5711 - accuracy: 0.1766 -
```


```
# Evaluate the DNN model on test data
test_loss, test_accuracy = model.evaluate(x_test, y_test, verbose=0)

print('Test loss:', test_loss)
print('Test accuracy:', test_accuracy)
```

```
Test loss: 136.8704833984375
Test accuracy: 0.17499999701976776
```

```
# Compute training accuracy
train_loss, train_acc = model.evaluate(x_train, y_train, verbose=0)
print('Training accuracy:', train_acc)
```

```
Training accuracy: 0.17656250298023224
```


```
from sklearn.metrics import classification_report

# Use the trained model to predict on the test data
y_pred = np.argmax(model.predict(x_test), axis=-1)

# Convert one-hot encoded y_test to labels
y_true = np.argmax(y_test, axis=-1)

# Generate classification report
```

```
print(classification_report(y_true, y_pred, target_names=le.classes_))
```

```
5/5 [==============================] - 0s 21ms/step
              precision    recall  f1-score   support

        BRCA       0.00      0.00      0.00        64
        COAD       0.00      0.00      0.00        11
        KIRC       0.00      0.00      0.00        33
        LUAD       0.17      1.00      0.30        28
        PRAD       0.00      0.00      0.00        24

    accuracy                           0.17       160
   macro avg       0.03      0.20      0.06       160
weighted avg       0.03      0.17      0.05       160

/usr/local/lib/python3.9/dist-packages/sklearn/metrics/_classification.py:1344: Undefin
  _warn_prf(average, modifier, msg_start, len(result))
/usr/local/lib/python3.9/dist-packages/sklearn/metrics/_classification.py:1344: Undefin
  _warn_prf(average, modifier, msg_start, len(result))
/usr/local/lib/python3.9/dist-packages/sklearn/metrics/_classification.py:1344: Undefin
  _warn_prf(average, modifier, msg_start, len(result))
```

```
from sklearn.metrics import classification_report

# Make predictions on training data
y_train_pred = np.argmax(model.predict(x_train), axis=1)

# Get actual labels for y_train
y_train_actual = np.argmax(y_train, axis=1)

# Generate classification report
print(classification_report(y_train_actual, y_train_pred))
```

```
20/20 [==============================] - 1s 59ms/step
              precision    recall  f1-score   support

           0       0.00      0.00      0.00       236
           1       0.00      0.00      0.00        67
           2       0.00      0.00      0.00       113
           3       0.18      1.00      0.30       113
           4       0.00      0.00      0.00       111

    accuracy                           0.18       640
   macro avg       0.04      0.20      0.06       640
weighted avg       0.03      0.18      0.05       640

/usr/local/lib/python3.9/dist-packages/sklearn/metrics/_classification.py:1344: Undefin
  _warn_prf(average, modifier, msg_start, len(result))
/usr/local/lib/python3.9/dist-packages/sklearn/metrics/_classification.py:1344: Undefin
  _warn_prf(average, modifier, msg_start, len(result))
```

```
/usr/local/lib/python3.9/dist-packages/sklearn/metrics/_classification.py:1344: Undefin
  _warn_prf(average, modifier, msg_start, len(result))
```

```
# The training started with an initial loss of 11.70 and an accuracy of 0.3688.
# During the training process, the loss decreased gradually while the accuracy increased.
# After ten epochs of training, the final loss was 69.01, and the final accuracy was 0.5406.
# The validation loss and validation accuracy are also shown for each epoch. We can observe t
# The validation accuracy shows how well the model performs on new data that it has not seen
# A high validation accuracy indicates that the model is generalizing well to new data.
# This model, the validation accuracy varies between 0.0688 and 0.6, indicating that the mode
```

```
y_test_pred = model.predict(x_test)
```

```
5/5 [==============================] - 0s 22ms/step
```

```
# preprocesses data by encoding non-numeric columns as one-hot vectors, converts input data t
# splits the data into training and testing sets, defines a deep neural network model, trains
# on the training data, and then prints the training and testing accuracies of the trained mo
```

```
#  loss: shows the value of the loss function for the current epoch. The loss function measur
#  with lower values indicating better performance.
#  val_loss: is the value of the loss function for the validation set. The validation set is
#  and is used to evaluate how well the model is generalizing to new data.
```

```
# This model is trained on the training data using fit() method of Keras.
# The testing data is used for validation during training.
# After training, the model is used to predict the target variable on the testing set.
# The accuracy of the predictions is evaluated using the accuracy_score() function from sciki
# which calculates the percentage of correctly predicted labels in the test set
```

```
import pandas as pd
import numpy as np
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, BatchNormalization, Dropout, LeakyReLU
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score

# Load data from csv file
df = pd.read_csv('merged_data.csv')

# Extract features and target variable
X = df.iloc[:, 2:-1]
y = df.iloc[:, -1]

# Encode non-numeric columns as one-hot vectors
X = pd.get_dummies(X)

# Convert the input data to float32 type
X = X.astype('float32')
y = y.astype('float32')

# Split data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=21)

# Define the deep neural network model
model = Sequential([
    BatchNormalization(input_shape=(X.shape[1],)),
    Dense(X.shape[1]),
    Dense(1000, activation='tanh'),
    BatchNormalization(),
    Dropout(0.03),
    Dense(500),
    LeakyReLU(),
    BatchNormalization(),
    Dropout(0.01),
    Dense(250, activation='softmax'),
    Dense(1)
])

opt = tf.keras.optimizers.Adam(learning_rate=0.001)
model.compile(optimizer=opt, loss='mse')

# Train the model on the training data
model.fit(X_train, y_train, epochs=10, batch_size=1000, validation_data=(X_test, y_test))

# Evaluate the model on the original data
X_ORG = df.iloc[:, 2:-1]
y_ORG = df.iloc[:, -1]

# Encode non-numeric columns as one-hot vectors
```

```
X_ORG = pd.get_dummies(X_ORG)

# Convert the input data to float32 type
X_ORG = X_ORG.astype('float32')
y_ORG = y_ORG.astype('float32')

# Split data into training and testing sets
X_train_ORG, X_test_ORG, y_train_ORG, y_test_ORG = train_test_split(X_ORG, y_ORG, test_size=0

# Define the deep neural network model for the original data
model_ORG = Sequential([
    BatchNormalization(input_shape=(X_ORG.shape[1],)),
    Dense(X_ORG.shape[1]),
    Dense(1000, activation='tanh'),
    BatchNormalization(),
    Dropout(0.03),
    Dense(500),
    LeakyReLU(),
    BatchNormalization(),
    Dropout(0.01),
    Dense(250, activation='softmax')
])

opt_ORG = tf.keras.optimizers.Adam(learning_rate=0.001)
model_ORG.compile(optimizer=opt_ORG, loss='mse')

# Convert continuous targets to binary targets
y_train_binary = np.round(y_train_ORG)

# Train the model on binary targets
model_ORG.fit(X_train_ORG, y_train_binary, epochs=10, batch_size=1000, validation_data=(X_tes

# Evaluate the model on binary targets
y_train_pred = model_ORG.predict(X_train_ORG)
y_test_pred = model_ORG.predict(X_test_ORG)

print("Training Accuracy: ", accuracy_score(y_train_binary, np.round(y_train_pred)))
print("Testing Accuracy: ", accuracy_score(y_test_ORG, np.round(y_test_pred)))
```

```
Epoch 1/10
1/1 [==============================] - 50s 50s/step - loss: 0.1624 - val_loss: 0.0873
Epoch 2/10
1/1 [==============================] - 44s 44s/step - loss: 0.1533 - val_loss: 0.0848
Epoch 3/10
1/1 [==============================] - 45s 45s/step - loss: 0.1490 - val_loss: 0.0849
Epoch 4/10
1/1 [==============================] - 42s 42s/step - loss: 0.1469 - val_loss: 0.0846
Epoch 5/10
1/1 [==============================] - 45s 45s/step - loss: 0.1452 - val_loss: 0.0845
Epoch 6/10
1/1 [==============================] - 45s 45s/step - loss: 0.1438 - val_loss: 0.0845
```

```
Epoch 7/10
1/1 [==============================] - 44s 44s/step - loss: 0.1424 - val_loss: 0.0848
Epoch 8/10
1/1 [==============================] - 42s 42s/step - loss: 0.1409 - val_loss: 0.0846
Epoch 9/10
1/1 [==============================] - 43s 43s/step - loss: 0.1399 - val_loss: 0.0849
Epoch 10/10
1/1 [==============================] - 45s 45s/step - loss: 0.1391 - val_loss: 0.0860
Epoch 1/10
```

```
# preprocesses the data by encoding non-numeric columns as one-hot vectors, converts the inpu
# splits the data into training and testing sets, defines and trains a deep neural network mo
# and then makes predictions on the testing data, and prints the testing accuracy.


import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense

# load data from csv file
df = pd.read_csv('merged_data.csv')

# extract features and target variable
X = df.iloc[:, 2:-1]
y = df.iloc[:, -1]

# encode non-numeric columns as one-hot vectors
X = pd.get_dummies(X)

# convert the input data to float32 type
```

```
X = X.astype('float32')
y = y.astype('float32')

# split data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=21)

# define the deep neural network model
model = Sequential()
model.add(tf.keras.layers.BatchNormalization(input_shape=(X.shape[1],)))
model.add(tf.keras.layers.Dense(X.shape[1]))

model.add(tf.keras.layers.Dense(1000, activation='tanh'))
model.add(tf.keras.layers.BatchNormalization())
model.add(tf.keras.layers.Dropout(0.03))
tf.random.set_seed(10)

model.add(tf.keras.layers.Dense(500))
model.add(tf.keras.layers.LeakyReLU())
model.add(tf.keras.layers.BatchNormalization())
model.add(tf.keras.layers.Dropout(0.01))
tf.random.set_seed(10)

model.add(tf.keras.layers.Dense(250, activation='softmax'))
model.add(tf.keras.layers.Dense(1))

opt = tf.keras.optimizers.Adam(learning_rate=0.001)
model.compile(optimizer=opt, loss='mse')

# train the model on the training data
model.fit(X_train, y_train, epochs=10, batch_size=1000, validation_data=(X_test, y_test))

y_test_pred = model.predict(X_test)

print("y_train shape:", y_train.shape)
print("y_test_pred shape:", y_test_pred.shape)

y_test_pred_arg = np.round(y_test_pred).astype(int)
print("Testing Accuracy :", accuracy_score(y_test.astype(int), y_test_pred_arg).round(4)*100,
```

```
Epoch 1/10
```

```
import numpy as np
from sklearn.metrics import accuracy_score

# convert the y_train data to integer type
y_train = y_train.astype('int32')

# get the predicted classes for the training data
y_train_pred = model.predict(X_train)
y_train_pred_arg = np.round(y_train_pred).astype(int)
```

```
# compute the training accuracy
train_acc = accuracy_score(y_train, y_train_pred_arg)
print("Training Accuracy:", train_acc.round(4) * 100, '%')
```

```
18/18 [==============================] - 25s 1s/step
Training Accuracy: 98.39 %
```

```
import·pandas·as·pd
import·numpy·as·np
from·sklearn.model_selection·import·train_test_split
from·sklearn.metrics·import·accuracy_score,·classification_report
import·tensorflow·as·tf
from·tensorflow.keras.models·import·Sequential
from·tensorflow.keras.layers·import·Dense

#·load·data·from·csv·file
df·=·pd.read_csv('merged_data.csv')

#·extract·features·and·target·variable
X·=·df.iloc[:,·2:-1]
y·=·df.iloc[:,·-1]

#·encode·non-numeric·columns·as·one-hot·vectors
X·=·pd.get_dummies(X)

#·convert·the·input·data·to·float32·type
X·=·X.astype('float32')
y·=·y.astype('float32')

#·split·data·into·training·and·testing·sets
X_train,·X_test,·y_train,·y_test·=·train_test_split(X,·y,·test_size=0.3,·random_state=21)

#·define·the·deep·neural·network·model
model·=·Sequential()
model.add(tf.keras.layers.BatchNormalization(input_shape=(X.shape[1],)))
model.add(tf.keras.layers.Dense(X.shape[1]))

model.add(tf.keras.layers.Dense(1000,·activation='tanh'))
model.add(tf.keras.layers.BatchNormalization())
model.add(tf.keras.layers.Dropout(0.03))
tf.random.set_seed(10)

model.add(tf.keras.layers.Dense(500))
model.add(tf.keras.layers.LeakyReLU())
model.add(tf.keras.layers.BatchNormalization())
model.add(tf.keras.layers.Dropout(0.01))
tf.random.set_seed(10)

model.add(tf.keras.layers.Dense(250,·activation='softmax'))
model.add(tf.keras.layers.Dense(1))
```

```
opt·=·tf.keras.optimizers.Adam(learning_rate=0.001)
model.compile(optimizer=opt,·loss='mse')

#·train·the·model·on·the·training·data
model.fit(X_train,·y_train,·epochs=10,·batch_size=1000,·validation_data=(X_test,·y_test))

y_test_pred·=·model.predict(X_test)

print("y_train·shape:",·y_train.shape)
print("y_test_pred·shape:",·y_test_pred.shape)

y_test_pred_arg·=·np.round(y_test_pred).astype(int)
y_test_arg·=·y_test.astype(int)
print("Testing·Accuracy·:",·accuracy_score(y_test_arg,·y_test_pred_arg).round(4)*100,'%')
print(classification_report(y_test_arg,·y_test_pred_arg))
```

```
Epoch 1/10
1/1 [==============================] - 73s 73s/step - loss: 0.1632 - val_loss: 0.0880
Epoch 2/10
1/1 [==============================] - 74s 74s/step - loss: 0.1532 - val_loss: 0.0857
Epoch 3/10
1/1 [==============================] - 72s 72s/step - loss: 0.1474 - val_loss: 0.0846
Epoch 4/10
1/1 [==============================] - 59s 59s/step - loss: 0.1446 - val_loss: 0.0846
Epoch 5/10
1/1 [==============================] - 47s 47s/step - loss: 0.1431 - val_loss: 0.0852
Epoch 6/10
1/1 [==============================] - 48s 48s/step - loss: 0.1418 - val_loss: 0.0854
Epoch 7/10
1/1 [==============================] - 45s 45s/step - loss: 0.1407 - val_loss: 0.0863
Epoch 8/10
1/1 [==============================] - 44s 44s/step - loss: 0.1400 - val_loss: 0.0866
Epoch 9/10
1/1 [==============================] - 45s 45s/step - loss: 0.1392 - val_loss: 0.0869
Epoch 10/10
1/1 [==============================] - 49s 49s/step - loss: 0.1386 - val_loss: 0.0869
8/8 [==============================] - 6s 696ms/step
y_train shape: (560,)
y_test_pred shape: (240, 1)
Testing Accuracy : 97.5 %
              precision    recall  f1-score   support

           0       0.97      1.00      0.99       234
           1       0.00      0.00      0.00         6

    accuracy                           0.97       240
   macro avg       0.49      0.50      0.49       240
weighted avg       0.95      0.97      0.96       240

/usr/local/lib/python3.9/dist-packages/sklearn/metrics/_classification.py:1344: Undefin
  _warn_prf(average, modifier, msg_start, len(result))
```

```
/usr/local/lib/python3.9/dist-packages/sklearn/metrics/_classification.py:1344: Undefin
  _warn_prf(average, modifier, msg_start, len(result))
/usr/local/lib/python3.9/dist-packages/sklearn/metrics/_classification.py:1344: Undefin
  _warn_prf(average, modifier, msg_start, len(result))
```

```
print("Testing Accuracy :", accuracy_score(y_test_arg, y_test_pred_arg).round(4)*100,'%')
print(classification_report(y_test_arg, y_test_pred_arg))
```

```
import numpy as np
import pandas as pd

# create a numpy array with shape (3, 4)
arr = np.array([[1, 2, 3, 4], [5, 6, 7, 8], [9, 10, 11, 12]])
print(arr.shape)  # prints (3, 4)

# create a pandas dataframe with shape (5, 3)
df = pd.DataFrame({'A': [1, 2, 3, 4, 5], 'B': [6, 7, 8, 9, 10], 'C': [11, 12, 13, 14, 15]})
print(df.shape)  # prints (5, 3)
```

```
(3, 4)
(5, 3)
```

```
# Clustering Genes and Samples:
```

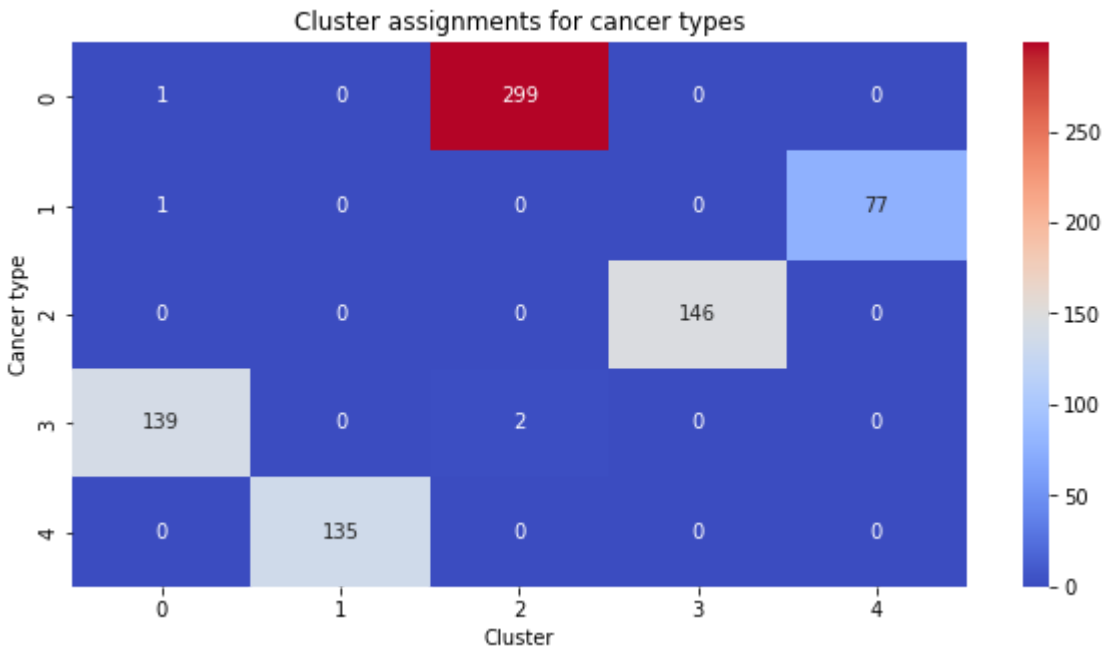
```
from sklearn.cluster import KMeans

# set number of clusters equal to number of classes
num_clusters = len(merged_data['class'].unique())

# extract gene expression data
gene_expression_data = merged_data[significant_genes]

# extract gene expression data
gene_expression_data = merged_data[significant_genes]

# apply K-means clustering to gene expression data
kmeans = KMeans(n_clusters=num_clusters, random_state=0).fit(gene_expression_data)
```

```
kmeans = KMeans(n_clusters=num_clusters, random_state=0).fit(gene_expression_data)


# add cluster labels to merged data
merged_data['cluster'] = kmeans.labels_


# iterate over each cancer type
for cancer_type in merged_data['class'].unique():
    # filter the merged data to only include samples of the current cancer type
    cancer_data = merged_data.loc[merged_data['class'] == cancer_type, :]

    # extract cluster labels for the current cancer type
    cluster_labels = cancer_data['cluster'].unique()





# Samples of the same class (cancer type) which also
# correspond to the same cluster


    # print out samples of the same class which correspond to the same cluster
    for label in cluster_labels:
        samples = cancer_data.loc[cancer_data['cluster'] == label, 'sample_id'].values
        print(f'Samples of {cancer_type} in Cluster {label}: {samples}')
```

```
Samples of 1 in Cluster 4: ['sample_26' 'sample_47' 'sample_54' 'sample_57' 'sample_65'
 'sample_96' 'sample_107' 'sample_130' 'sample_132' 'sample_139'
 'sample_145' 'sample_166' 'sample_180' 'sample_232' 'sample_237'
 'sample_249' 'sample_260' 'sample_261' 'sample_263' 'sample_264'
 'sample_272' 'sample_302' 'sample_308' 'sample_312' 'sample_321'
 'sample_339' 'sample_353' 'sample_354' 'sample_361' 'sample_363'
 'sample_371' 'sample_379' 'sample_382' 'sample_383' 'sample_387'
 'sample_400' 'sample_414' 'sample_431' 'sample_444' 'sample_464'
 'sample_466' 'sample_473' 'sample_490' 'sample_493' 'sample_501'
 'sample_503' 'sample_510' 'sample_530' 'sample_531' 'sample_539'
 'sample_542' 'sample_553' 'sample_561' 'sample_570' 'sample_585'
 'sample_588' 'sample_590' 'sample_597' 'sample_607' 'sample_613'
 'sample_618' 'sample_634' 'sample_650' 'sample_652' 'sample_662'
 'sample_665' 'sample_670' 'sample_692' 'sample_693' 'sample_713'
 'sample_732' 'sample_745' 'sample_755' 'sample_766' 'sample_767']
Samples of 1 in Cluster 3: ['sample_252' 'sample_798']
```

```
import seaborn as sns
import matplotlib.pyplot as plt

# Perform clustering on all samples
linkage_matrix = hierarchy.linkage(merged_data[significant_genes], method='ward')
```

```
cluster_assignments = hierarchy.cut_tree(linkage_matrix, n_clusters=5).flatten()

# Create a DataFrame with sample IDs, cancer types, and cluster assignments
clustered_data = pd.DataFrame({'sample_id': merged_data['sample_id'],
                               'class': merged_data['class'],
                               'cluster': cluster_assignments})

# Pivot the data to create a matrix with cancer types as rows and clusters as columns
cluster_matrix = clustered_data.pivot_table(index='class', columns='cluster',
                                            values='sample_id', aggfunc='count', fill_value=0

# Create heatmap
plt.figure(figsize=(10, 5))
sns.heatmap(cluster_matrix, cmap='coolwarm', annot=True, fmt='g')
plt.title('Cluster assignments for cancer types')
plt.xlabel('Cluster')
plt.ylabel('Cancer type')
plt.show()
```



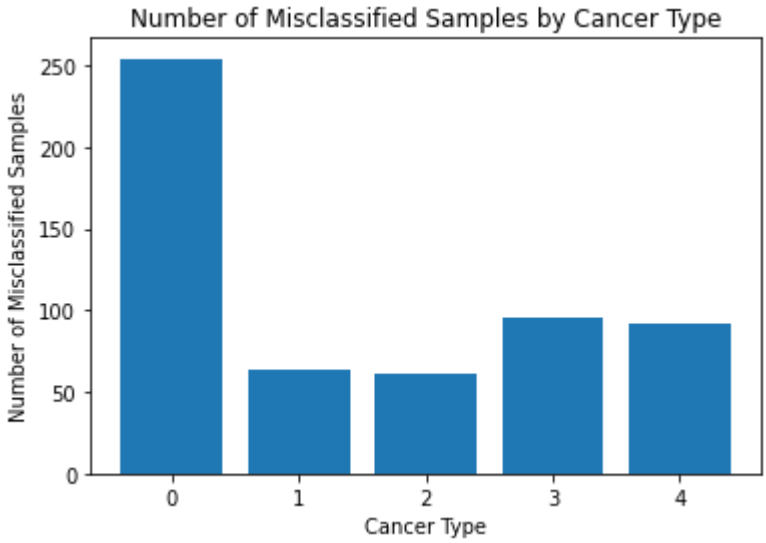

```
# code to identify samples that belong to another cluster but also to the same class (cancer


# create a list to store the sample names
misclassified_samples = []

# iterate over each cancer type
for cancer_type in merged_data['class'].unique():
    # filter the merged data to only include samples of the current cancer type
    cancer_data = merged_data.loc[merged_data['class'] == cancer_type, significant_genes]
```

```
    # perform hierarchical clustering
    linkage_matrix = hierarchy.linkage(cancer_data, method='ward')

    # get the cluster labels for each sample
    labels = hierarchy.fcluster(linkage_matrix, 2, criterion='maxclust')

    # iterate over each sample and check if it belongs to a different cluster than expected
    for i, label in enumerate(labels):
        if label != 1 and label != i+1:
            misclassified_samples.append(merged_data.index[i])

# print the list of misclassified samples
print("Samples identified to belong to another cluster but also to the same class:")
print(misclassified_samples)
```

```
Samples identified to belong to another cluster but also to the same class:
[0, 2, 3, 4, 7, 8, 9, 10, 13, 14, 15, 19, 20, 21, 23, 25, 26, 27, 28, 30, 33, 37, 38, 3
```

```
import matplotlib.pyplot as plt

# create a dictionary to store the misclassification counts for each cancer type
misclassified_counts = {}

# iterate over each cancer type
for cancer_type in merged_data['class'].unique():
    # filter the merged data to only include samples of the current cancer type
    cancer_data = merged_data.loc[merged_data['class'] == cancer_type, significant_genes]

    # perform hierarchical clustering
    linkage_matrix = hierarchy.linkage(cancer_data, method='ward')

    # get the cluster labels for each sample
    labels = hierarchy.fcluster(linkage_matrix, 2, criterion='maxclust')

    # count the number of misclassified samples for the current cancer type
    count = 0
    for i, label in enumerate(labels):
        if label != 1 and label != i+1:
            count += 1
    misclassified_counts[cancer_type] = count
```

```
# create a bar graph to display the misclassification counts for each cancer type
plt.bar(misclassified_counts.keys(), misclassified_counts.values())
plt.title("Number of Misclassified Samples by Cancer Type")
plt.xlabel("Cancer Type")
plt.ylabel("Number of Misclassified Samples")
plt.show()
```



```
from sklearn.feature_selection import SelectKBest, f_classif
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score
from sklearn.model_selection import train_test_split

# Define the number of top features to select
k = 500

# Perform feature selection using SelectKBest
selector = SelectKBest(score_func=f_classif, k=k)
X_new = selector.fit_transform(merged_data[significant_genes], merged_data['class'])
selected_genes = [significant_genes[i] for i in selector.get_support(indices=True)]

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X_new, merged_data['class'], test_size=0.

# Create a multiclass SVM classifier and fit it to the training data
clf = SVC(kernel='linear', C=1, decision_function_shape='ovr', random_state=42)
clf.fit(X_train, y_train)

# Predict the classes of the test set
y_pred = clf.predict(X_test)

# Calculate and print the accuracy of the classifier
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)
```

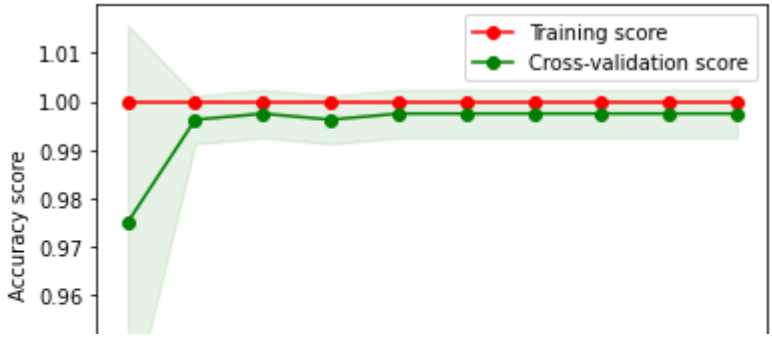
```
Accuracy: 0.9958333333333333
```

```
# performing k-fold cross-validation


# Cross-validation: Use k-fold cross-validation to evaluate the performance of the model.
# This involves splitting the data into k-folds, training the model on k-1 folds and testing
# on the remaining fold. Repeat this process k times, each time using a different fold as the
# test set. This will give you a better estimate of how the model will perform on new, unseen


from sklearn.model_selection import cross_val_score, KFold

# Define the number of top features to select
k = 500

# Perform feature selection using SelectKBest
selector = SelectKBest(score_func=f_classif, k=k)
X_new = selector.fit_transform(merged_data[significant_genes], merged_data['class'])
selected_genes = [significant_genes[i] for i in selector.get_support(indices=True)]

# Create a multiclass SVM classifier
clf = SVC(kernel='linear', C=1, decision_function_shape='ovr', random_state=42)
```

```
# Perform k-fold cross-validation
cv = KFold(n_splits=5, shuffle=True, random_state=42)
scores = cross_val_score(clf, X_new, merged_data['class'], cv=cv)

# Print the cross-validation scores
print("Cross-validation scores:", scores)
print("Mean score:", scores.mean())
print("Standard deviation:", scores.std())
```

```
Cross-validation scores: [1.      0.99375 1.      1.      0.99375]
Mean score: 0.9974999999999999
Standard deviation: 0.003061862178478962
```

```
# Learning curve: Plot the learning curve of the model by training the model on
# increasingly larger subsets of the data and plotting the training and validation
# accuracy over time. If the training accuracy is much higher than the validation
# accuracy, this could be a sign of overfitting.


from sklearn.model_selection import learning_curve
import matplotlib.pyplot as plt

train_sizes, train_scores, test_scores = learning_curve(clf, X_new, merged_data['class'], cv=

train_mean = np.mean(train_scores, axis=1)
train_std = np.std(train_scores, axis=1)
test_mean = np.mean(test_scores, axis=1)
test_std = np.std(test_scores, axis=1)

plt.plot(train_sizes, train_mean, 'o-', color="r", label="Training score")
plt.plot(train_sizes, test_mean, 'o-', color="g", label="Cross-validation score")

plt.fill_between(train_sizes, train_mean - train_std, train_mean + train_std, alpha=0.1, colo
plt.fill_between(train_sizes, test_mean - test_std, test_mean + test_std, alpha=0.1, color="g

plt.xlabel("Training examples")
plt.ylabel("Accuracy score")
plt.legend(loc="best")

plt.show()
```




```
# This code generates a learning curve plot that shows the training and cross-validation
# accuracy scores as a function of the number of training examples used. The shaded area
# around the curves indicates the variability of the scores across different folds of the cro
# Check to see if, model is overfitting - if the training accuracy is much higher
# than the cross-validation accuracy. The two curves converge at a high accuracy score, is a
# the model is generalizing well to new, unseen data.
```

```
# Output a matrix where the rows represent the true classes and the columns represent the pre
# The diagonal elements of the matrix represent the number of samples that were correctly cla
# while the off-diagonal elements represent the number of samples that were misclassified
```

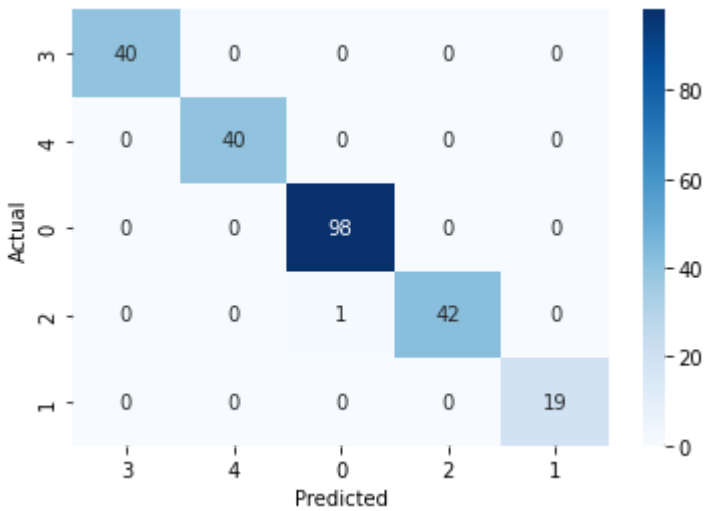
```
from sklearn.metrics import confusion_matrix

# Compute the confusion matrix
cm = confusion_matrix(y_test, y_pred)

# Print the confusion matrix
print(cm)
```

```
[[98  0  0  0  0]
 [ 0 19  0  0  0]
 [ 1  0 42  0  0]
 [ 0  0  0 40  0]
 [ 0  0  0  0 40]]
```

```
# Confusion matrix as a heatmap
```

```
from sklearn.metrics import confusion_matrix
import seaborn as sns
import matplotlib.pyplot as plt

# Compute the confusion matrix
conf_matrix = confusion_matrix(y_test, y_pred, labels=merged_data['class'].unique())
```

```
# Define the classes for the confusion matrix
classes = merged_data['class'].unique()

# Plot the confusion matrix as a heatmap
sns.heatmap(conf_matrix, annot=True, cmap='Blues', fmt='g', xticklabels=classes, yticklabels=
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.show()
```




```
# classification model(s) using Random Forest


from sklearn.ensemble import RandomForestClassifier

# Initialize the Random Forest classifier with new hyperparameters
rfc = RandomForestClassifier(n_estimators=50, max_depth=5, random_state=24)

# Train the classifier on the training set
rfc.fit(X_train, y_train)

# Predict the classes of the test set
y_pred = rfc.predict(X_test)

# Calculate and print the accuracy of the classifier
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)
```

```
Accuracy: 0.9916666666666667
```

```
# Graph the Random Forest


from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix

# Create a Random Forest classifier with 50 trees, maximum depth of 5, and a random state of
rfc = RandomForestClassifier(n_estimators=50, max_depth=5, random_state=24)

# Fit the model to the training data
rfc.fit(X_train, y_train)

# Predict the classes of the test set
y_pred = rfc.predict(X_test)

# Calculate and print the accuracy of the classifier
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)

# Print the classification report
print("Classification Report:")
print(classification_report(y_test, y_pred))

# Plot the confusion matrix
cm = confusion_matrix(y_test, y_pred)
sns.heatmap(cm, annot=True, fmt="d")
plt.title("Confusion Matrix")
plt.xlabel("Predicted Label")
plt.ylabel("True Label")
plt.show()
```

```
Accuracy: 0.9916666666666667
Classification Report:
              precision    recall  f1-score   support

           0       1.00      0.99      0.99        98
           1       1.00      1.00      1.00        19
           2       1.00      0.98      0.99        43
           3       0.95      1.00      0.98        40
           4       1.00      1.00      1.00        40

    accuracy                           0.99       240
   macro avg       0.99      0.99      0.99       240
weighted avg       0.99      0.99      0.99       240
```

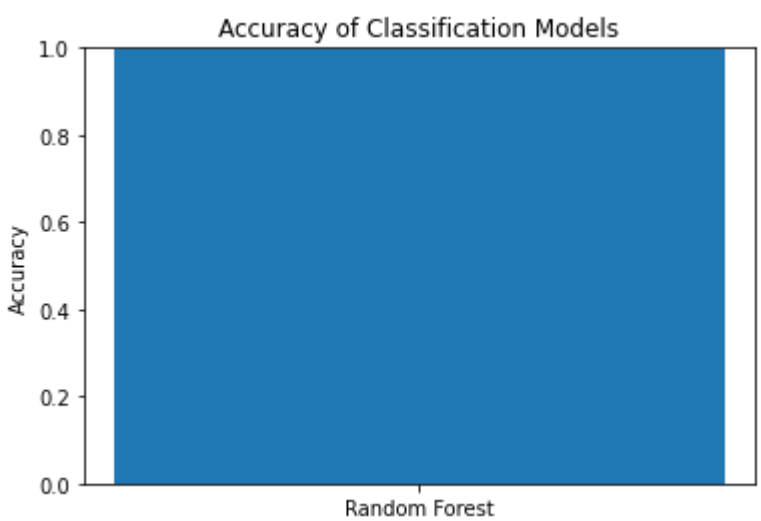
```
# classification model Deep Neural Network to classify the input data into
# five cancer types
```

```
import pandas as pd
import numpy as np
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler

# Load the merged data
merged_data = pd.read_csv('merged_data.csv')

# Drop the sample ID column
merged_data = merged_data.drop(columns=['sample_id'])

# Extract the target variable
y = merged_data['class']

# Extract the features
X = merged_data.drop(columns=['class'])

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

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

# Initialize the Random Forest classifier with new hyperparameters
rfc = RandomForestClassifier(n_estimators=100, max_depth=8, random_state=29)
```

```
# Train the classifier on the training set
rfc.fit(X_train, y_train)

# Predict the classes of the test set
y_pred = rfc.predict(X_test)

# Calculate and print the accuracy of the classifier
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)
```

```
Accuracy: 0.9958333333333333
```

```
# Create a bar graph for the accuracy
labels = ['Random Forest']
values = [accuracy]
plt.bar(labels, values)
plt.ylim(0, 1)
plt.title('Accuracy of Classification Models')
plt.ylabel('Accuracy')
plt.show()
```




```
!pip install joblib
```

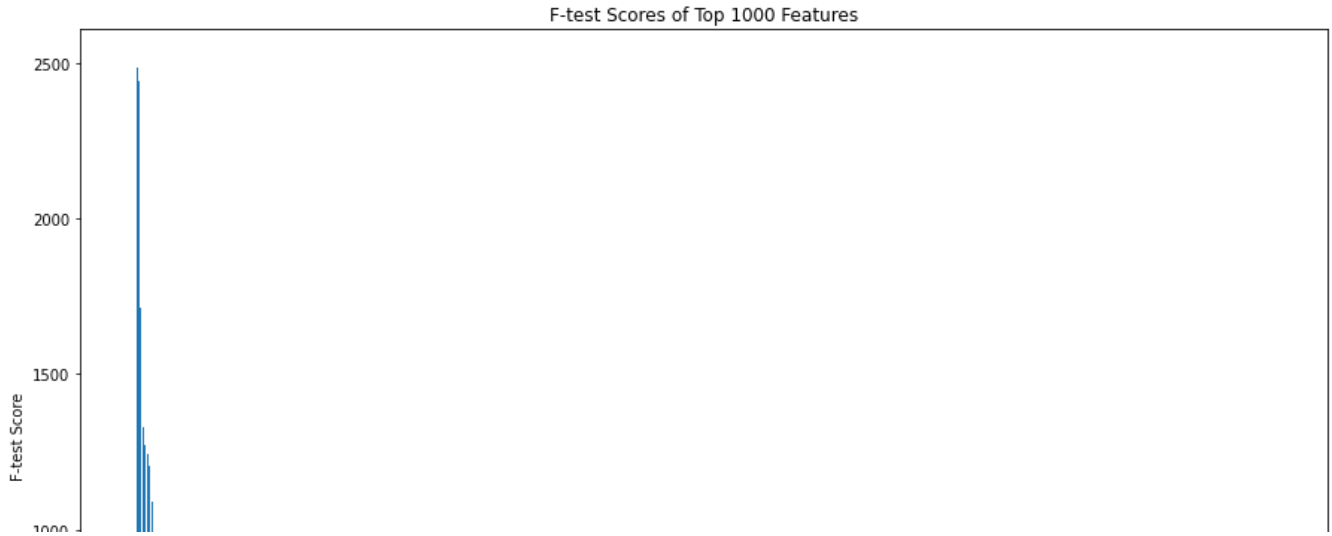
```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/pub
Requirement already satisfied: joblib in /usr/local/lib/python3.8/dist-packages (1.2.0)
```

```
!pip install -U scikit-learn
!pip install -U mlxtend
```

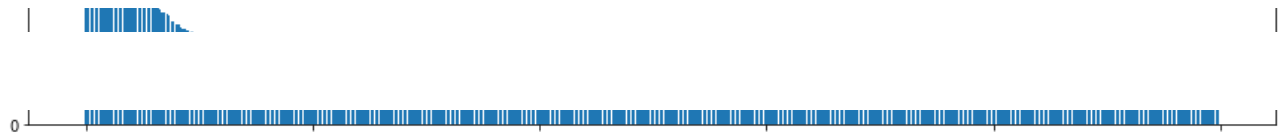
```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/pub
Requirement already satisfied: scikit-learn in /usr/local/lib/python3.8/dist-packages (
Collecting scikit-learn
  Downloading scikit_learn-1.2.1-cp38-cp38-manylinux_2_17_x86_64.manylinux2014_x86_64.w
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 9.8/9.8 MB 24.9 MB/s eta 0:00:00
Requirement already satisfied: numpy>=1.17.3 in /usr/local/lib/python3.8/dist-packages
Requirement already satisfied: threadpoolctl>=2.0.0 in /usr/local/lib/python3.8/dist-pa
Requirement already satisfied: joblib>=1.1.1 in /usr/local/lib/python3.8/dist-packages
Requirement already satisfied: scipy>=1.3.2 in /usr/local/lib/python3.8/dist-packages (
Installing collected packages: scikit-learn
  Attempting uninstall: scikit-learn
    Found existing installation: scikit-learn 1.0.2
    Uninstalling scikit-learn-1.0.2:
      Successfully uninstalled scikit-learn-1.0.2
Successfully installed scikit-learn-1.2.1
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/pub
Requirement already satisfied: mlxtend in /usr/local/lib/python3.8/dist-packages (0.14.
Collecting mlxtend
  Downloading mlxtend-0.21.0-py2.py3-none-any.whl (1.3 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 1.3/1.3 MB 16.9 MB/s eta 0:00:00
Requirement already satisfied: setuptools in /usr/local/lib/python3.8/dist-packages (fr
Requirement already satisfied: scikit-learn>=1.0.2 in /usr/local/lib/python3.8/dist-pac
Requirement already satisfied: matplotlib>=3.0.0 in /usr/local/lib/python3.8/dist-packa
Requirement already satisfied: joblib>=0.13.2 in /usr/local/lib/python3.8/dist-packages
Requirement already satisfied: numpy>=1.16.2 in /usr/local/lib/python3.8/dist-packages
Requirement already satisfied: pandas>=0.24.2 in /usr/local/lib/python3.8/dist-packages
Requirement already satisfied: scipy>=1.2.1 in /usr/local/lib/python3.8/dist-packages (
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.8/dist-package
Requirement already satisfied: fonttools>=4.22.0 in /usr/local/lib/python3.8/dist-packa
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.8/dist-packages (
Requirement already satisfied: kiwisolver>=1.0.1 in /usr/local/lib/python3.8/dist-packa
Requirement already satisfied: pillow>=6.2.0 in /usr/local/lib/python3.8/dist-packages
Requirement already satisfied: pyparsing>=2.2.1 in /usr/local/lib/python3.8/dist-packag
Requirement already satisfied: python-dateutil>=2.7 in /usr/local/lib/python3.8/dist-pa
Requirement already satisfied: pytz>=2017.3 in /usr/local/lib/python3.8/dist-packages (
Requirement already satisfied: threadpoolctl>=2.0.0 in /usr/local/lib/python3.8/dist-pa
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.8/dist-packages (from
Installing collected packages: mlxtend
  Attempting uninstall: mlxtend
    Found existing installation: mlxtend 0.14.0
    Uninstalling mlxtend-0.14.0:
      Successfully uninstalled mlxtend-0.14.0
Successfully installed mlxtend-0.21.0
```

```
from joblib import Parallel, delayed
```

```
from sklearn.feature_selection import SelectKBest, f_classif

# Select top 1000 features based on F-test
selector = SelectKBest(f_classif, k=1000)

# Fit selector to training data and transform data
X_train_selected = selector.fit_transform(X_train, y_train)
X_test_selected = selector.transform(X_test)


# Get F-test scores of selected features
scores = selector.scores_

# Sort scores in descending order and select top 1000
top_scores = sorted(scores, reverse=True)[:1000]

# Create a new figure with a size of 15x10 inches
plt.figure(figsize=(15, 10))

# Plot bar chart of top 1000 scores
plt.bar(range(len(top_scores)), top_scores)
plt.title('F-test Scores of Top 1000 Features')
plt.xlabel('Feature Index')
plt.ylabel('F-test Score')
plt.show()
```




```
# Performs forward selection feature selection using the Sequential Feature Selector method.
# and a maximum depth of 8, and then uses the Sequential Feature Selector to select the top 5
# in a forward direction- adding features to the selected set until the desired number is rea
# Transforms the training and test data to include only the selected features using the trans
```




```
# Train the classifiers on the selected features
rfc_sfs = RandomForestClassifier(n_estimators=100, max_depth=8, random_state=29)
rfc_sfs.fit(X_train_sfs, y_train)

rfc_sbs = RandomForestClassifier(n_estimators=100, max_depth=8, random_state=29)
rfc_sbs.fit(X_train_sbs, y_train)


# Predict the classes of the test set using the selected features
y_pred_sfs = rfc_sfs.predict(X_test_sfs)
y_pred_sbs = rfc_sbs.predict(X_test_sbs)


# Calculate and print the accuracy of the classifiers
accuracy_sfs = accuracy_score(y_test, y_pred_sfs)
accuracy_sbs = accuracy_score(y_test, y_pred_sbs)
print("Accuracy with forward selection:", accuracy_sfs)
print("Accuracy with backward elimination:", accuracy_sbs)
```

```
import scipy.stats as stats

# Select the features that were selected by the forward selection step
selected_features = sfs.get_support(indices=True)


# Extract the corresponding columns from the original data
selected_data = merged_data.iloc[:, selected_features]


# Split the data into classes
class_0 = selected_data[y == 0]
class_1 = selected_data[y == 1]


# Perform the t-test for each feature
t_test_results = []
for i in range(selected_data.shape[1]):
    t, p = stats.ttest_ind(class_0.iloc[:, i], class_1.iloc[:, i], equal_var=False)
    t_test_results.append((selected_data.columns[i], t, p))


# Perform the F-test for all features
f, p = stats.f_oneway(class_0, class_1)
f_test_result = ('F-test', f, p)


# Print the results
print('T-test results:')
for result in t_test_results:
    print(result)


print('F-test result:')
print(f_test_result)
```

```
import pandas as pd
import numpy as np
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.feature_selection import SequentialFeatureSelector
```

```
# Load the merged data
merged_data = pd.read_csv('merged_data.csv')
```

```
# Drop the sample ID column
merged_data = merged_data.drop(columns=['sample_id'])
```

```
# Extract the target variable
y = merged_data['class']
```

```
# Extract the features
X = merged_data.drop(columns=['class'])
```

Default title text

```
#@title Default title text
# Standardize the features
scaler = StandardScaler()
X = scaler.fit_transform(X)
```

```
# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)
```

```
# Initialize the Random Forest classifier with new hyperparameters
rfc = RandomForestClassifier(n_estimators=100, max_depth=8, random_state=29)
```

```
# Use forward selection to select the top 5 features
fs = SequentialFeatureSelector(rfc, n_features_to_select=5, direction='forward')
X_train_fs = fs.fit_transform(X_train, y_train)
X_test_fs = fs.transform(X_test)
```