# khan_code

December 16, 2023

[ ]:
```
# Question 1
# The code for question 1 must be inputted into R
# This portion of code is for the four histograms displaying the
# distribution of gene  expression in the four disease states: Healthy Control,␣
 ↪Convalescent,
# Dengue Fever and Dengue Haemorrhagic fever.
# The code for the histograms for the first question.
# Again this bit of code must be implemented in R.
#
#
> # Read the dengue data and metadata
> dengue_data <- read.csv("dengue_data.csv", stringsAsFactors = FALSE)
> metadata <- read.csv("dengue_metadata.csv", stringsAsFactors = FALSE)
>
>
> install.packages("tidyr")
> library(tidyr)
>
> # Reshape the data from wide to long format
> dengue_data_long <- gather(dengue_data, key = "sample", value = "expression",␣
 ↪-Gene, factor_key=TRUE)
> # Rename the first column to 'Gene'
> colnames(dengue_data)[1] <- 'Gene'
>
>
> library(tidyr)
> dengue_data_long <- gather(dengue_data, key = "sample", value = "expression",␣
 ↪-Gene, factor_key=TRUE)
>
> # Merging the reshaped dengue data with metadata
> merged_data <- merge(metadata, dengue_data_long, by = 'sample')
>
>
> library(ggplot2)
>
> # Filter and plot histogram for each disease state
> # Dengue Fever
```

```
> dengue_fever_data <- subset(merged_data, disease.state == "Dengue Fever")
> ggplot(dengue_fever_data, aes(x = expression)) +
+     geom_histogram(binwidth = 0.2, fill = "red", color = "black") +
+     theme_minimal() +
+     labs(title = "Gene Expression Distribution - Dengue Fever",
+          x = "Expression Level", y = "Frequency")
>
> # Dengue Hemorrhagic Fever
> dengue_hf_data <- subset(merged_data, disease.state == "Dengue Hemorrhagic␣
 ↪Fever")
> ggplot(dengue_hf_data, aes(x = expression)) +
+     geom_histogram(binwidth = 0.2, fill = "green", color = "black") +
+     theme_minimal() +
+     labs(title = "Gene Expression Distribution - Dengue Hemorrhagic Fever",
+          x = "Expression Level", y = "Frequency")
>
> # Convalescent
> convalescent_data <- subset(merged_data, disease.state == "Convalescent")
> ggplot(convalescent_data, aes(x = expression)) +
+     geom_histogram(binwidth = 0.2, fill = "blue", color = "black") +
+     theme_minimal() +
+     labs(title = "Gene Expression Distribution - Convalescent",
+          x = "Expression Level", y = "Frequency")
>
> # Filter for Healthy Control data
> healthy_control_data <- subset(merged_data, disease.state == "Healthy␣
 ↪Control")
>
> # Plot histogram for Healthy Control
> ggplot(healthy_control_data, aes(x = expression)) +
+     geom_histogram(binwidth = 0.2, fill = "gray", color = "black") +
+     theme_minimal() +
+     labs(title = "Gene Expression Distribution - Healthy Control",
+          x = "Expression Level", y = "Frequency")
>
> table(merged_data$disease.state)

           Convalescent             Dengue Fever
                 565763                   535986
Dengue Hemorrhagic Fever          healthy control
                 297770                   267993
>
> head(healthy_control_data)
[1] sample            X                 infection
[4] disease.state     disease.state.abbr individual
[7] description       Gene              expression
<0 rows> (or 0-length row.names)
```

```
>
> # Filter for Healthy Control data
> healthy_control_data <- merged_data[merged_data$disease.state == "healthy␣
 ↪control", ]
>
> # Check the first few rows of the Healthy Control data
> head(healthy_control_data)
           sample          X infection   disease.state
1399520 GSM1253075 GSM1253075   control healthy control
1399521 GSM1253075 GSM1253075   control healthy control
1399522 GSM1253075 GSM1253075   control healthy control
1399523 GSM1253075 GSM1253075   control healthy control
1399524 GSM1253075 GSM1253075   control healthy control
1399525 GSM1253075 GSM1253075   control healthy control
        disease.state.abbr individual
1399520               CTRL         c3
1399521               CTRL         c3
1399522               CTRL         c3
1399523               CTRL         c3
1399524               CTRL         c3
1399525               CTRL         c3
                                                                          ␣
 ↪description
1399520 Value for GSM1253075: Healthy control c3; src: Whole Blood of healthy␣
 ↪control
1399521 Value for GSM1253075: Healthy control c3; src: Whole Blood of healthy␣
 ↪control
1399522 Value for GSM1253075: Healthy control c3; src: Whole Blood of healthy␣
 ↪control
1399523 Value for GSM1253075: Healthy control c3; src: Whole Blood of healthy␣
 ↪control
1399524 Value for GSM1253075: Healthy control c3; src: Whole Blood of healthy␣
 ↪control
1399525 Value for GSM1253075: Healthy control c3; src: Whole Blood of healthy␣
 ↪control
         Gene expression
1399520   DDR1   2.566121
1399521   RFC2   2.835956
1399522  HSPA6   3.397800
1399523   PAX8   1.910920
1399524 GUCA1A   1.707218
1399525   UBA7   3.180164
>
>
>
> # Check unique values in disease.state column
```

```
> unique(merged_data$disease.state)
[1] "Dengue Fever"             "Dengue Hemorrhagic Fever"
[3] "Convalescent"             "healthy control"
>
> # Create a histogram for the healthy control group
> ggplot(healthy_control_data, aes(x = expression)) +
+     geom_histogram(binwidth = 0.5, fill = "blue", color = "black") +
+     theme_minimal() +
+     labs(title = "Histogram of Gene Expression in Healthy Control", x =␣
 ↪"Expression Level", y = "Frequency")
>
```

[ ]:
```
#Please execute this bit in Jupyter. Q1 continued, PCA.
# After PCA done a scree plot and biplot are also generated.
#Please execute this in Jupyter
import pandas as pd
from sklearn.decomposition import PCA


data = pd.read_csv('dengue_data.csv', index_col='Unnamed: 0')
metadata = pd.read_csv('dengue_metadata.csv', index_col='Unnamed: 0')

# Performing PCA
n_components = 10
pca = PCA(n_components)
pca_transformed = pca.fit_transform(data.T)


pca_df = pd.DataFrame(pca_transformed,
                      index=data.columns,
                      columns=[f'PCA {i}' for i in range(1, n_components + 1)])

# Display the PCA DataFrame
print(pca_df)
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np

# Calculate the percentage of variance explained by each of the selected␣
 ↪components
explained_variance_ratio = pca.explained_variance_ratio_
explained_variance_percentage = explained_variance_ratio * 100

# Scree Plot
plt.figure(figsize=(10, 5))
sns.barplot(x=[f'PCA {i}' for i in range(1, n_components + 1)],
            y=explained_variance_percentage)
```

```
plt.title('Scree Plot of PCA')
plt.xlabel('Principal Components')
plt.ylabel('Percentage of Variance Explained')
plt.show()


from scipy.cluster import hierarchy
import matplotlib.pyplot as plt
import seaborn as sns

# Transpose the dataframe to have samples as rows and features as columns
data_transposed = data.transpose()

# Ensure the order of the rows in the transposed data matches the order in the␣
 ↪metadata
data_transposed = data_transposed.loc[metadata.index]


linkage_of_data = hierarchy.linkage(data_transposed, method='ward')


unique_states = metadata['disease.state'].unique()
palette = sns.color_palette("hsv", len(unique_states))
color_dict = dict(zip(unique_states, palette))


leaf_colors = metadata['disease.state'].map(color_dict)


plt.figure(figsize=(8, 6))

# Generating a dendrogram
dn_of_data = hierarchy.dendrogram(linkage_of_data, color_threshold=50,␣
 ↪above_threshold_color='grey',
                                  labels=metadata.index, leaf_font_size=10)


ax = plt.gca()
xlbls = ax.get_xmajorticklabels()
for lbl in xlbls:
    lbl.set_color(leaf_colors[lbl.get_text()])


plt.title('Hierarchical Cluster Analysis')
plt.xlabel('Samples')
plt.ylabel('Cluster Distance')
```

```
plt.show()
```

[ ]:
```
# Please execute this in Jupyter. Q1 HCA.

from scipy.cluster import hierarchy
import matplotlib.pyplot as plt


data_transposed = data.transpose()


linkage_of_data = hierarchy.linkage(data_transposed, method='ward')


plt.figure(figsize=(8, 6))


dn_of_data = hierarchy.dendrogram(linkage_of_data, color_threshold=50)


plt.title('Hierarchical Cluster Analysis')
plt.xlabel('Samples')


plt.show()


from scipy.cluster import hierarchy
import matplotlib.pyplot as plt
import seaborn as sns
import matplotlib.patches as mpatches

# Transpose the data to have samples as rows and features as columns for␣
 ↪clustering
data_transposed = data.transpose()


linkage_of_data = hierarchy.linkage(data_transposed, method='ward')


unique_states = metadata['disease.state'].unique()
palette = sns.color_palette("hsv", len(unique_states))
color_dict = dict(zip(unique_states, palette))


leaf_colors = metadata['disease.state'].map(color_dict)
```

```
# Create a figure object and set the size of the plot
plt.figure(figsize=(10, 8))

# Generate a dendrogram
dn_of_data = hierarchy.dendrogram(linkage_of_data, color_threshold=50,␣
 ↪above_threshold_color='grey',
                                  labels=metadata.index, leaf_font_size=10)


ax = plt.gca()
xlbls = ax.get_xmajorticklabels()
for lbl in xlbls:
    lbl.set_color(leaf_colors[lbl.get_text()])

# Add title and labels to the plot
plt.title('Hierarchical Cluster Analysis')
plt.xlabel('Samples')
plt.ylabel('Cluster Distance')


legend_patches = [mpatches.Patch(color=color, label=state) for state, color in␣
 ↪color_dict.items()]
plt.legend(handles=legend_patches, title='Disease State', bbox_to_anchor=(1.05,␣
 ↪1), loc='upper left')


plt.tight_layout()


plt.show()

# Import necessary libraries
import seaborn as sns


state_colors = {
    "Convalescent": "blue",
    "Dengue Fever": "red",
    "Dengue Hemorrhagic Fever": "green",
    "Healthy Control": "gray"
    # Add other disease states and their corresponding colors as needed
}


# Ensure that the index of metadata matches the index of pca_df
row_colors = metadata.loc[pca_df.index, 'disease.state'].map(state_colors)
```

```
# Create a clustermap with dendrogram and heatmap using PCA data
sns.clustermap(pca_df, method='ward', cmap='viridis', row_colors=row_colors,␣
 ↪figsize=(12, 10))


plt.title('Hierarchical Clustering with Heatmap of PCA Data')


plt.show()
```

[ ]:
```
# Now lets move onto Question 2
# The code for question 2 should be executed in a Jupyter notebook
#
# We are aiming to do volcano plots
# Initial bit of code to ensure relevant packages are installed
#
import numpy as np
import matplotlib.pyplot as plt
import statsmodels.api as sm
from statsmodels.formula.api import ols
from statsmodels.stats.multitest import multipletests
```

[ ]:
```
# Question 2 continued
# This section is just to preprocess the data
# # Load your data
dengue_data = pd.read_csv('dengue_data.csv')
dengue_metadata = pd.read_csv('dengue_metadata.csv')

# Rename the first column in dengue_data to 'Gene'
dengue_data.rename(columns={dengue_data.columns[0]: 'Gene'}, inplace=True)

# Transpose dengue_data to get samples as rows and genes as columns
dengue_data_transposed = dengue_data.set_index('Gene').transpose()

# Merge the transposed data with the metadata
merged_data = pd.merge(dengue_metadata, dengue_data_transposed,␣
 ↪left_on='sample', right_index=True)

# Check the merged data
# Marker- you can delete this print statement to simplify things. Here for␣
 ↪testing benefit
print(merged_data.head())
```

[ ]:
```
#Question 2 continued. Anova testing may take a VERY long time. Apologies
def prepare_anova_data(data):
    all_data = []
```

```
    #
    gene_expression_start_index = data.columns.get_loc("description") + 1
    gene_expression_columns = data.columns[gene_expression_start_index:]

    for gene in gene_expression_columns:
        gene_data = data[['disease.state', gene]].dropna()
        gene_data = gene_data.rename(columns={gene: 'expression', 'disease.
 ↪state': 'DiseaseState'})
        gene_data['gene'] = gene
        all_data.append(gene_data)
    return pd.concat(all_data)

anova_ready_data = prepare_anova_data(merged_data)

# Check the ANOVA ready data. Again delete if needed.
print(anova_ready_data.head())
```

[ ]:
```
def statsmodels_anova(data, num_genes=None):
    results = []
    subset_genes = data['gene'].unique()[:num_genes] if num_genes is not None␣
 ↪else data['gene'].unique()
    for gene in subset_genes:
        gene_data = data[data['gene'] == gene]
        model = ols('expression ~ C(DiseaseState)', data=gene_data).fit()
        anova_results = sm.stats.anova_lm(model, typ=2)
        p_value = anova_results['PR(>F)'].iloc[0]
        results.append((gene, p_value))
    return pd.DataFrame(results, columns=['Gene', 'P-Value'])

anova_test_results = statsmodels_anova(anova_ready_data)


print(anova_test_results.head())
```

[ ]:
```
p_values = anova_test_results['P-Value'].values
adjusted_p = multipletests(p_values, method='fdr_bh')[1]
anova_test_results['Adjusted P-Value'] = adjusted_p

# Adjusted p-values
print(anova_test_results.head())
```

[ ]:
```
unique_groups = merged_data['disease.state'].unique()
gene_expression_columns = merged_data.select_dtypes(include=[np.number]).columns
fold_change_results = {}

for i, group1 in enumerate(unique_groups):
    for group2 in unique_groups[i+1:]:
```

```
        data_group1 = merged_data[merged_data['disease.state'] ==␣
 ↪group1][gene_expression_columns]
        data_group2 = merged_data[merged_data['disease.state'] ==␣
 ↪group2][gene_expression_columns]
        mean_group1 = data_group1.mean()
        mean_group2 = data_group2.mean()
        fold_change = mean_group1 / mean_group2
        fold_change_results[(group1, group2)] = fold_change

fold_change_df = pd.DataFrame(fold_change_results)

# marker-can be deleted
print(fold_change_df.head())
```

[ ]:
```
print(fold_change_df.columns)
!pip install --upgrade plotly
```

[ ]:
```
# Prepare data for the volcano plot
# Choose a specific comparison for fold change
comparison = ('Convalescent', 'Dengue Hemorrhagic Fever')
log2_fold_changes = np.log2(fold_change_df[comparison])
anova_test_results['log2FoldChange'] = log2_fold_changes.
 ↪loc[anova_test_results['Gene']].values
anova_test_results['-log10(Adjusted P-Value)'] = -np.
 ↪log10(anova_test_results['Adjusted P-Value'])

# Volcano plot
plt.figure(figsize=(10, 6))
plt.scatter(anova_test_results['log2FoldChange'],␣
 ↪anova_test_results['-log10(Adjusted P-Value)'], color='grey')

# Highlight significant points
significance_threshold = -np.log10(0.05)
fold_change_threshold = 1

significant = anova_test_results[(anova_test_results['-log10(Adjusted␣
 ↪P-Value)'] >= significance_threshold) &
                                 (abs(anova_test_results['log2FoldChange']) >=␣
 ↪fold_change_threshold)]
plt.scatter(significant['log2FoldChange'], significant['-log10(Adjusted␣
 ↪P-Value)'], color='red')

plt.xlabel('Log2 Fold Change (Convalescent vs Dengue Hemorrhagic Fever)')
plt.ylabel('-Log10 Adjusted P-Value')
plt.title('Volcano Plot')
plt.axhline(y=significance_threshold, color='blue', linestyle='--')
```

```
plt.axvline(x=fold_change_threshold, color='green', linestyle='--')
plt.axvline(x=-fold_change_threshold, color='green', linestyle='--')
plt.show()
```

[ ]:
```
import numpy as np
import matplotlib.pyplot as plt

def create_volcano_plot(anova_data, fold_change_data, comparison, title,␣
↪num_labels=10):

    log2_fold_changes = np.log2(fold_change_data[comparison])


    merged_data = anova_data.copy()
    merged_data['log2FoldChange'] = log2_fold_changes.loc[anova_data['Gene']].
↪values
    merged_data['-log10(Adjusted P-Value)'] = -np.log10(merged_data['Adjusted␣
↪P-Value'])


    plt.figure(figsize=(10, 6))


    significance_threshold = -np.log10(0.05)  # -log10(0.05) for p-value of 0.05
    fold_change_threshold = 0.585  # Log2 fold change threshold, corresponding␣
↪to a 1.5-fold change
    significant = merged_data[(merged_data['-log10(Adjusted P-Value)'] >=␣
↪significance_threshold) &
                              (abs(merged_data['log2FoldChange']) >=␣
↪fold_change_threshold)]

    plt.scatter(significant['log2FoldChange'], significant['-log10(Adjusted␣
↪P-Value)'], color='red')


    non_significant = merged_data[~((merged_data['-log10(Adjusted P-Value)'] >=␣
↪significance_threshold) &
                                    (abs(merged_data['log2FoldChange']) >=␣
↪fold_change_threshold))]

    plt.scatter(non_significant['log2FoldChange'],␣
↪non_significant['-log10(Adjusted P-Value)'], color='grey')


    plt.axhline(y=significance_threshold, color='blue', linestyle='--')
    plt.axvline(x=-fold_change_threshold, color='green', linestyle='--')
```

```
    plt.axvline(x=fold_change_threshold, color='green', linestyle='--')


    significant_sorted = significant.sort_values('Adjusted P-Value').
 ↪head(num_labels)
    for i, row in significant_sorted.iterrows():
        plt.text(row['log2FoldChange'], row['-log10(Adjusted P-Value)'],␣
 ↪row['Gene'], fontsize=8)


    plt.title(title)
    plt.xlabel(f'Log2 Fold Change ({comparison[0]} vs {comparison[1]})')
    plt.ylabel('-Log10 Adjusted P-Value')
    plt.show()


create_volcano_plot(anova_test_results, fold_change_df, ('Dengue Fever',␣
 ↪'healthy control'), 'Volcano Plot: Healthy Control vs Dengue Fever')
create_volcano_plot(anova_test_results, fold_change_df, ('Dengue Hemorrhagic␣
 ↪Fever', 'healthy control'), 'Volcano Plot: Healthy Control vs Dengue␣
 ↪Hemorrhagic Fever')
create_volcano_plot(anova_test_results, fold_change_df, ('Convalescent',␣
 ↪'healthy control'), 'Volcano Plot: Healthy Control vs Convalescent')
create_volcano_plot(anova_test_results, fold_change_df, ('Dengue Hemorrhagic␣
 ↪Fever', 'Dengue Fever'), 'Volcano Plot: Dengue Hemorrhagic Fever vs Dengue␣
 ↪Fever')
```

[ ]:
```
def print_significant_genes_for_comparison(anova_data, fold_change_data,␣
 ↪comparison):

    significance_threshold = 0.05
    fold_change_threshold = 0.5


    log2_fold_changes = np.log2(fold_change_data[comparison])


    merged_data = anova_data.copy()
    merged_data['log2FoldChange'] = log2_fold_changes.loc[anova_data['Gene']].
 ↪values


    significant_genes = merged_data[
        (merged_data['Adjusted P-Value'] <= significance_threshold) &
        (abs(merged_data['log2FoldChange']) >= fold_change_threshold)
    ]['Gene']
```

```
    print(f"Significant genes for {comparison[0]} vs {comparison[1]}:")
    for gene in significant_genes:
        print(gene)

# Print significant genes
print_significant_genes_for_comparison(anova_test_results, fold_change_df,␣
 ↪('Dengue Fever', 'healthy control'))

print_significant_genes_for_comparison(anova_test_results, fold_change_df,␣
 ↪('Dengue Hemorrhagic Fever', 'healthy control'))

print_significant_genes_for_comparison(anova_test_results, fold_change_df,␣
 ↪('Convalescent', 'healthy control'))

print_significant_genes_for_comparison(anova_test_results, fold_change_df,␣
 ↪('Dengue Hemorrhagic Fever', 'Dengue Fever'))
```

[ ]:
```
#Now lets move onto question 3
# To be executed in a jupyter notebook
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from sklearn.metrics import classification_report, confusion_matrix
from sklearn.preprocessing import StandardScaler
import numpy as np
from sklearn.utils import shuffle

# Load your data
dengue_data = pd.read_csv('dengue_data.csv')
dengue_metadata = pd.read_csv('dengue_metadata.csv')

# Transpose dengue_data to get samples as rows and genes as columns
dengue_data_transposed = dengue_data.set_index(dengue_data.columns[0]).
 ↪transpose()

# Merge the transposed data with the metadata
merged_data = pd.merge(dengue_metadata, dengue_data_transposed,␣
 ↪left_on='sample', right_index=True)

# Filter for 'Dengue Fever' and 'Dengue Hemorrhagic Fever' only
filtered_data = merged_data[merged_data['disease.state'].isin(['Dengue Fever',␣
 ↪'Dengue Hemorrhagic Fever'])]

# Isolate features and target variable
```

```
features = filtered_data.drop(['Unnamed: 0', 'sample', 'infection', 'disease.
 ↪state', 'disease.state.abbr', 'individual', 'description'], axis=1)
target = filtered_data['disease.state']


bootstrap_accuracies = []
permutation_accuracies = []

# Performing bootstrapping and permutation testing
for _ in range(100):
    # Bootstrapping
    X_train, X_test, y_train, y_test = train_test_split(features, target,␣
 ↪test_size=0.3)
    scaler = StandardScaler()
    X_train_scaled = scaler.fit_transform(X_train)
    X_test_scaled = scaler.transform(X_test)
    model = SVC()
    model.fit(X_train_scaled, y_train)
    y_pred = model.predict(X_test_scaled)
    bootstrap_accuracies.append(np.mean(y_pred == y_test))

    # Permutation Testing
    shuffled_target = shuffle(target)
    X_train, X_test, y_train, y_test = train_test_split(features,␣
 ↪shuffled_target, test_size=0.3)
    X_train_scaled = scaler.fit_transform(X_train)
    X_test_scaled = scaler.transform(X_test)
    model.fit(X_train_scaled, y_train)
    y_pred = model.predict(X_test_scaled)
    permutation_accuracies.append(np.mean(y_pred == y_test))

# Print results
average_bootstrap_accuracy = np.mean(bootstrap_accuracies)
average_permutation_accuracy = np.mean(permutation_accuracies)
print(f"Average bootstrap accuracy: {average_bootstrap_accuracy:.2f}")
print(f"Average permutation accuracy: {average_permutation_accuracy:.2f}")
```