Question 1

```
import pandas as pd
import random


# Load the data from the CSV file
file_path = '/heart.csv'
data = pd.read_csv(file_path)

# Randomly select 100 samples from the RestingBP variable
sampled_restingBP = data['RestingBP'].sample(n=100, random_state=random.seed())



sampled_restingBP.head()
```

```
727    158
711    120
113    140
592    190
196    120
Name: RestingBP, dtype: int64
```

```
# Manually calculating the z-statistic and p-value for the proportion test
proportion_hypothesis = 0.5

# Proportion in the sample and its standard error
sample_proportion = n_greater_equal_120 / len(sampled_restingBP)
standard_error = np.sqrt(proportion_hypothesis * (1 - proportion_hypothesis) / len(sampled_restingBP))

# Z-statistic for proportion
z_statistic_prop = (sample_proportion - proportion_hypothesis) / standard_error

# P-value for proportion
p_value_proportion = 2 * (1 - stats.norm.cdf(np.abs(z_statistic_prop)))

# calculating the confidence intervals for the proportion
proportion_confidence_interval = (
    sample_proportion - 1.96 * standard_error,
    sample_proportion + 1.96 * standard_error
)

(t_statistic, p_value_mean, mean_confidence_interval), (z_statistic_prop, p_value_proportion, proportion_confi
```

```
((41.795603748992264,
  0.2509095200870061,
  (128.65719277030516, 135.08280722969485)),
 (6.799999999999999, 1.0461853605647775e-11, (0.742, 0.938)))
```

Double-click (or enter) to edit

```
from scipy import stats
import numpy as np

# Hypothesis Test for Mean
mean_hypothesis = 130
t_statistic_mean, p_value_mean = stats.ttest_1samp(sampled_restingBP, mean_hypothesis)

# Confidence Interval for Mean
mean_confidence_interval = stats.t.interval(
    confidence_level,
```

```
    len(sampled_restingBP) - 1,
    loc=np.mean(sampled_restingBP),
    scale=stats.sem(sampled_restingBP)
)

# Hypothesis Test for Proportion
# Count the number of samples ≥ 120
n_greater_equal_120 = np.sum(sampled_restingBP >= 120)
sample_proportion = n_greater_equal_120 / len(sampled_restingBP)
standard_error_proportion = np.sqrt(0.5 * (1 - 0.5) / len(sampled_restingBP))
z_statistic_prop = (sample_proportion - 0.5) / standard_error_proportion
p_value_proportion = 2 * (1 - stats.norm.cdf(np.abs(z_statistic_prop)))

# Confidence Interval for Proportion
proportion_confidence_interval = (
    sample_proportion - 1.96 * standard_error_proportion,
    sample_proportion + 1.96 * standard_error_proportion
)

(t_statistic_mean, p_value_mean, mean_confidence_interval), (z_statistic_prop, p_value_proportion, proportion_

results = {
    "Mean Analysis": {
        "T-statistic": t_statistic_mean,
        "P-value": p_value_mean,
        "95% Confidence Interval": mean_confidence_interval
    },
    "Proportion Analysis": {
        "Z-statistic": z_statistic_prop,
        "P-value": p_value_proportion,
        "95% Confidence Interval": proportion_confidence_interval
    }
}

results
```

```
{'Mean Analysis': {'T-statistic': 1.1549045535712565,
  'P-value': 0.2509095200870061,
  '95% Confidence Interval': (128.65719277030516, 135.08280722969485)},
 'Proportion Analysis': {'Z-statistic': 6.799999999999999,
  'P-value': 1.0461853605647775e-11,
  '95% Confidence Interval': (0.742, 0.938)}}
```

Question 2

```
# Randomly select 100 samples from the Age and Cholesterol variables
sampled_age = data['Age'].sample(n=100, random_state=random.seed())
sampled_cholesterol = data['Cholesterol'].sample(n=100, random_state=random.seed())

# Calculate means and standard deviations for the Age and Cholesterol samples
mean_age = sampled_age.mean()
sd_age = sampled_age.std()
mean_cholesterol = sampled_cholesterol.mean()
sd_cholesterol = sampled_cholesterol.std()



# Output for means and standard deviations
age_cholesterol_stats = {
    "Age": {"Mean": mean_age, "Standard Deviation": sd_age},
    "Cholesterol": {"Mean": mean_cholesterol, "Standard Deviation": sd_cholesterol}
}

# Performing a two-sample t-test between Age and Cholesterol (assuming unequal variances)
t_statistic, p_value = stats.ttest_ind(sampled_age, sampled_cholesterol, equal_var=False)

# Calculating confidence interval for RestingBP variable
# Assuming the user wants a 95% confidence interval for the mean of the RestingBP variable
mean_restingBP = sampled_restingBP.mean()
sem_restingBP = stats.sem(sampled_restingBP)
confidence_interval_restingBP = stats.t.interval(0.95, len(sampled_restingBP)-1, loc=mean_restingBP, scale=se

age_cholesterol_stats, (t_statistic, p_value), mean_restingBP, sd_restingBP
```

```
({'Age': {'Mean': 53.81, 'Standard Deviation': 9.646996706464943},
  'Cholesterol': {'Mean': 195.63, 'Standard Deviation': 101.94855654602561}},
 (-13.849072680426575, 4.629569819330056e-25),
 131.87,
 16.191814243155353)
```

```
mean_restingBP
```

```
131.87
```

```
sd_restingBP = sampled_restingBP.std()

sd_restingBP
```

```
16.191814243155353
```

```
from scipy import stats
n1 = n2 = 100     # sample sizes

# Two-sample t-test (assuming unequal variances, i.e., Welch's t-test)
t_statistic, p_value = stats.ttest_ind_from_stats(mean1=mean_restingBP, std1=sd_restingBP, nobs1=n1,
                                                  mean2=mean_age, std2=sd_age, nobs2=n2,
                                                  equal_var=False)

# Confidence Interval
# Degrees of freedom for Welch's t-test
df = ((sd_restingBP**2/n1 + sd_age**2/n2)**2) / (((sd_restingBP**2/n1)**2)/(n1-1) + ((sd_age**2/n2)**2)/(n2-1

# Critical t-value for 95% confidence
alpha = 0.05
t_critical = stats.t.ppf(1 - alpha/2, df)

# Margin of error
margin_error = t_critical * ((sd_restingBP**2/n1 + sd_age**2/n2)**0.5)

# Confidence interval
ci_lower = (mean_restingBP - mean_age) - margin_error
ci_upper = (mean_restingBP - mean_age) + margin_error

(t_statistic, p_value, df, ci_lower, ci_upper)
```

```
(41.415984240481166,
 6.688105365804174e-88,
 161.4192078147625,
 74.33799529666672,
 81.78200470333329)
```

```
n3 = 100                   #sample size for cholesterol

# Two-sample t-test (assuming unequal variances, i.e., Welch's t-test) between Resting BP and Cholesterol
t_statistic_cholesterol, p_value_cholesterol = stats.ttest_ind_from_stats(
    mean1=mean_BP,
    std1=std_BP,
    nobs1=n1,
    mean2=mean_cholesterol,
    std2=sd_cholesterol,
    nobs2=n3,
    equal_var=False)

# Degrees of freedom for Welch's t-test between Resting BP and Cholesterol
df_cholesterol = ((std_BP**2/n1 + sd_cholesterol**2/n3)**2) / (((std_BP**2/n1)**2)/(n1-1) + ((sd_cholesterol*

# Margin of error for Resting BP and Cholesterol
margin_error_cholesterol = t_critical * ((std_BP**2/n1 + sd_cholesterol**2/n3)**0.5)

# Confidence interval for Resting BP and Cholesterol
ci_lower_cholesterol = (mean_BP - mean_cholesterol) - margin_error_cholesterol
ci_upper_cholesterol = (mean_BP - mean_cholesterol) + margin_error_cholesterol

(t_statistic_cholesterol, p_value_cholesterol, df_cholesterol, ci_lower_cholesterol, ci_upper_cholesterol)
```

```
(-5.565679724353898,
 2.0519708772600851e-07,
 103.89759311456383,
 -81.82067743274258,
 -38.939322567257406)
```

Question 3

```
# correlation between Age and Cholesterol
correlation_age_restingBP = sampled_restingBP.corr(sampled_age)
correlation_cholesterol_restingBP = sampled_cholesterol.corr(sampled_restingBP)

# Output correlation
correlation_age_restingBP, correlation_cholesterol_restingBP
```

```
(-0.08300976303244427, 0.2352572641098867)
```

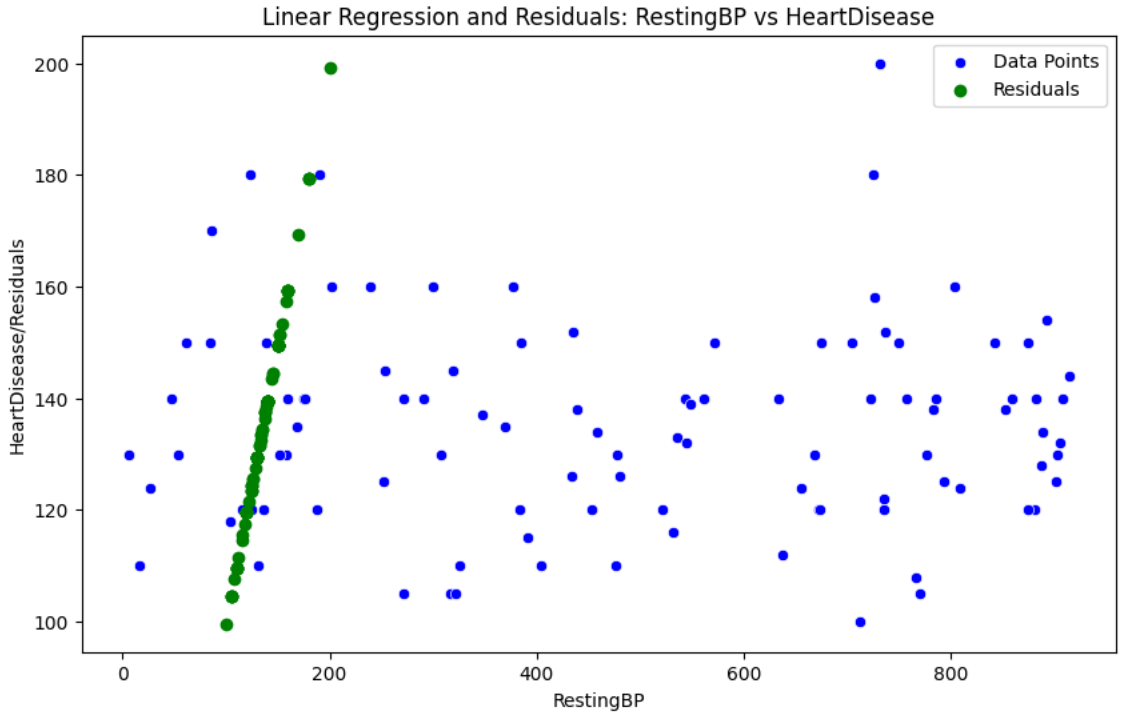
```
r1 = correlation_age_restingBP
r2 = correlation_cholesterol_restingBP

# Fisher's z-transformation
z1 = np.arctanh(r1)
z2 = np.arctanh(r2)

# Standard errors
se1 = 1 / np.sqrt(n1 - 3)
se2 = 1 / np.sqrt(n2 - 3)

# Test Z statistic
z = (z1 - z2) / np.sqrt(se1**2 + se2**2)

# P-value
p_value = stats.norm.sf(abs(z)) * 2  # Two-tailed test

# Confidence interval
alpha = 0.05  # 95% confidence
z_critical = stats.norm.ppf(1 - alpha/2)
ci_lower = z - z_critical * np.sqrt(se1**2 + se2**2)
ci_upper = z + z_critical * np.sqrt(se1**2 + se2**2)

# Convert CI back to r scale
ci_lower_r = np.tanh(ci_lower)
ci_upper_r = np.tanh(ci_upper)

z, p_value, ci_lower_r, ci_upper_r
```

```
(-2.2490777582093653,
 0.024507549350426226,
 -0.987401735901416,
 -0.96166880114805)
```

Question 4

```
import matplotlib.pyplot as plt
import seaborn as sns
from scipy.stats import linregress


# Aligning the samples for RestingBP and HeartDisease
aligned_samples = data[['RestingBP', 'HeartDisease']].sample(n=100, random_state=random.seed())

# Performing linear regression between RestingBP and HeartDisease
slope, intercept, r_value, p_value, std_err = linregress(aligned_samples['RestingBP'], aligned_samples['HeartD

# Linear regression equation: y = slope * x + intercept
def linear_regression_equation(x):
    return slope * x + intercept


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

```
# Scatter plot of the data
sns.scatterplot(aligned_samples['RestingBP'], color='blue', label='Data Points')


# Calculating and plotting residuals
residuals = aligned_samples['RestingBP'] - linear_regression_equation(aligned_samples['RestingBP'])
plt.scatter(aligned_samples['RestingBP'], residuals, color='green', label='Residuals')

plt.xlabel('RestingBP')
plt.ylabel('HeartDisease/Residuals')
plt.title('Linear Regression and Residuals: RestingBP vs HeartDisease')
plt.legend()
plt.show()

slope, intercept, r_value, p_value, std_err
```




```
(0.003752013544585813,
 0.024116013783494816,
 0.1372200940222656,
 0.17338782389379356,
 0.0027359359145781074)
```

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

# Scatter plot
sns.scatterplot(x=aligned_samples['RestingBP'], y=aligned_samples['HeartDisease'], color='blue', label='Data P

# Regression line
plt.plot(x_values, [linear_regression_equation(x) for x in x_values], color='red', label='Regression Line')

plt.xlabel('RestingBP')
plt.ylabel('HeartDisease')
plt.title('Linear Regression: RestingBP vs HeartDisease')
plt.legend()
plt.show()

print(linear_regression_equation(aligned_samples))
```


```
     RestingBP  HeartDisease
307   0.511878      0.024116
544   0.549398      0.024116
202   0.624438      0.024116
478   0.511878      0.027868
893   0.601926      0.027868
..         ...           ...
152   0.511878      0.024116
727   0.616934      0.027868
674   0.474358      0.027868
16    0.436838      0.027868
843   0.586918      0.024116

[100 rows x 2 columns]
```

Question 5

```
from sklearn.linear_model import LinearRegression
import numpy as np

# variables for multiple regression
selected_variables = data[['RestingBP', 'Age', 'Cholesterol', 'HeartDisease']].dropna()
```