



MA660E, Lab Report

Sirajulhaq Wahaj

Heart Disease Dataset

Overview

This dataset contains information about patients and various attributes related to heart disease, collected from Cleveland Clinic and made available on Kaggle. It includes both qualitative and quantitative variables, which are ideal for performing analyses such as descriptive statistics, confidence intervals, hypothesis testing, correlation analysis, and multiple linear regression.

Source: [Kaggle - Heart Disease Data](#)

Variables

Quantitative Variables

- **id**: Unique identifier for each patient
- **age**: Age of the patient in years
- **trestbps**: Resting blood pressure in mm Hg
- **chol**: Serum cholesterol level in mg/dl
- **thalch**: Maximum heart rate achieved
- **oldpeak**: ST depression induced by exercise relative to rest
- **ca**: Number of major vessels (0-3) colored by fluoroscopy
- **num**: Diagnosis of heart disease (angiographic disease status), where 0 indicates no disease and 1-4 indicates presence of disease

Qualitative Variables

- **sex**: Sex of the patient, either `Male` or `Female`
 - **dataset**: Source of the data, e.g., Cleveland
 - **cp**: Chest pain type, with categories `typical angina`, `asymptomatic`, `non-anginal`, or `atypical angina`
 - **fbs**: Fasting blood sugar > 120 mg/dl, represented as `TRUE` if true and `FALSE` otherwise
 - **restecg**: Resting electrocardiographic results, either `normal` or `lv hypertrophy` (left ventricular hypertrophy)
 - **exang**: Exercise-induced angina, with `TRUE` if present and `FALSE` otherwise
 - **slope**: Slope of the peak exercise ST segment, categorized as `upsloping`, `flat`, or `downsloping`
 - **thal**: Type of thalassemia, with values `normal`, `fixed defect`, or `reversible defect`
-

Part Two: 1. Descriptive Statistics

Task

Perform descriptive statistics analysis for at least two qualitative and two quantitative variables.

Solution

- **Confidence Interval for Job Satisfaction (Satis):** (10.06, 11.61)
- **Confidence Interval for the Difference in Job Satisfaction between Men and Women:** (-1.38, 1.85)

```
In [112...]:  
import pandas as pd  
import numpy as np  
from scipy import stats  
from scipy.stats import kruskal  
from scipy.stats import pearsonr  
import statsmodels.api as sm  
from sklearn.linear_model import LinearRegression  
from sklearn.model_selection import train_test_split  
from sklearn.metrics import r2_score, mean_squared_error
```

```
In [118...]:  
# silent downcasting and warnings  
pd.set_option('future.no_silent_downcasting', True)  
  
data_set = pd.read_csv('heart_disease_uci.csv')  
null_values = data_set.isnull().sum()
```

```
columns_with_null = null_values[null_values > 0]

for column, null_count in columns_with_null.items():
    print(f"{column}: {null_count} null values")

print(data_set.info())
print(data_set.describe())

# Columns with null values
quantitative_columns = ['trestbps', 'chol', 'thalch', 'oldpeak', 'ca']
qualitative_columns = ['fbs', 'restecg', 'exang', 'slope', 'thal', 'cp']

# 1. Quantitative Columns: Fill missing values with the median
data_cleaned = data_set.copy()
for col in quantitative_columns:
    if data_set[col].isnull().sum() > 0:
        median_value = data_set[col].median()
        data_cleaned[col] = data_set[col].fillna(median_value)

# 2. Qualitative Columns: Fill missing values with the mode
for col in qualitative_columns:
    if data_set[col].isnull().sum() > 0:
        mode_value = data_set[col].mode()[0]
        data_cleaned[col] = data_set[col].fillna(mode_value).infer_objects()
```

```

trestbps: 59 null values
chol: 30 null values
fbs: 90 null values
restecg: 2 null values
thalch: 55 null values
exang: 55 null values
oldpeak: 62 null values
slope: 309 null values
ca: 611 null values
thal: 486 null values
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 920 entries, 0 to 919
Data columns (total 16 columns):
 #   Column      Non-Null Count  Dtype  
--- 
 0   id          920 non-null    int64  
 1   age         920 non-null    int64  
 2   sex         920 non-null    object  
 3   dataset     920 non-null    object  
 4   cp          920 non-null    object  
 5   trestbps    861 non-null    float64 
 6   chol        890 non-null    float64 
 7   fbs         830 non-null    object  
 8   restecg     918 non-null    object  
 9   thalch      865 non-null    float64 
 10  exang       865 non-null    object  
 11  oldpeak     858 non-null    float64 
 12  slope        611 non-null    object  
 13  ca           309 non-null    float64 
 14  thal         434 non-null    object  
 15  num          920 non-null    int64  
dtypes: float64(5), int64(3), object(8)
memory usage: 115.1+ KB
None
      id      age  trestbps      chol      thalch  oldpeak \
count  920.000000  920.000000  861.000000  890.000000  865.000000  858.000000
mean   460.500000  53.510870  132.132404  199.130337  137.545665  0.878788
std    265.725422  9.424685  19.066070  110.780810  25.926276  1.091226
min    1.000000  28.000000  0.000000  0.000000  60.000000  -2.600000
25%   230.750000  47.000000  120.000000  175.000000  120.000000  0.000000
50%   460.500000  54.000000  130.000000  223.000000  140.000000  0.500000
75%   690.250000  60.000000  140.000000  268.000000  157.000000  1.500000
max   920.000000  77.000000  200.000000  603.000000  202.000000  6.200000

      ca      num
count  309.000000  920.000000
mean    0.676375  0.995652
std    0.935653  1.142693
min    0.000000  0.000000
25%   0.000000  0.000000
50%   0.000000  1.000000
75%   1.000000  2.000000
max   3.000000  4.000000

```

In [114...]: `null_values = data_cleaned.isnull().sum()`

```

columns_with_null = null_values[null_values > 0]
if len(columns_with_null) > 0:
    print("Columns with null values:")
else:
    print("No columns with null values.")

for column, null_count in columns_with_null.items():
    print(f"{column}: {null_count} null values")

```

No columns with null values.

Part Two: 1. Descriptive Statistics

Perform descriptive statistics analysis for at least two qualitative and two quantitative variables.

```

In [115...]: qualitative_vars = ['sex', 'cp']
print("\n--- Qualitative Variables Analysis ---\n")

for var in qualitative_vars:
    print(f"Descriptive Statistics for {var}:")
    print(data_set[var].value_counts())
    print(f"Number of unique values: {data_set[var].nunique()}")
    print(f"Mode: {data_set[var].mode()[0]}")
    print(f"Missing values: {data_set[var].isnull().sum()}")
    print("\n")

# Quantitative Variables Analysis
quantitative_vars = ['age', 'chol']
print("--- Quantitative Variables Analysis ---\n")

for var in quantitative_vars:
    print(f"Descriptive Statistics for {var}:")
    print(f"Mean: {data_set[var].mean():.2f}")
    print(f"Median: {data_set[var].median():.2f}")
    print(f"Standard Deviation: {data_set[var].std():.2f}")
    print(f"Minimum: {data_set[var].min()}")
    print(f"Maximum: {data_set[var].max()}")
    print(f"Missing values: {data_set[var].isnull().sum()}")
    print("\n")

```

```
--- Qualitative Variables Analysis ---
```

```
Descriptive Statistics for sex:
```

```
sex
Male      726
Female    194
Name: count, dtype: int64
Number of unique values: 2
Mode: Male
Missing values: 0
```

```
Descriptive Statistics for cp:
```

```
cp
asymptomatic     496
non-anginal       204
atypical angina   174
typical angina    46
Name: count, dtype: int64
Number of unique values: 4
Mode: asymptomatic
Missing values: 0
```

```
--- Quantitative Variables Analysis ---
```

```
Descriptive Statistics for age:
```

```
Mean: 53.51
Median: 54.00
Standard Deviation: 9.42
Minimum: 28
Maximum: 77
Missing values: 0
```

```
Descriptive Statistics for chol:
```

```
Mean: 199.13
Median: 223.00
Standard Deviation: 110.78
Minimum: 0.0
Maximum: 603.0
Missing values: 30
```

Part Two: 2. Confidence Intervals

Calculate the confidence interval for one quantitative variable and the confidence interval for the difference between two groups.

```
In [116...]
```

```
confidence_level = 0.95
```

```
# Calculate the mean, standard deviation, and standard error
age_mean = data_set['age'].mean()
```

```

age_std = data_set['age'].std()
age_n = data_set['age'].count()
age_se = age_std / np.sqrt(age_n)

# Calculate the confidence interval
age_ci = stats.t.interval(confidence_level, df=age_n-1, loc=age_mean, scale=age_se)
print(f"Confidence Interval for Mean Age ({confidence_level*100}%) : {age_ci[0]:.3f}")

# Confidence Interval for the Difference in Cholesterol Levels between Males and Females
chol_male = data_cleaned[data_cleaned['sex'] == 'Male']['chol']
chol_female = data_cleaned[data_cleaned['sex'] == 'Female']['chol']

# Calculate the means and standard deviations for each group
chol_male_mean = chol_male.mean()
chol_female_mean = chol_female.mean()
chol_male_std = chol_male.std()
chol_female_std = chol_female.std()

# sample sizes
n_male = chol_male.count()
n_female = chol_female.count()

# Calculate the standard error for the difference between means
se_diff = np.sqrt((chol_male_std**2 / n_male) + (chol_female_std**2 / n_female))

# Calculate the confidence interval for the difference in means
mean_diff = chol_male_mean - chol_female_mean
data_cleaned = min(n_male, n_female) - 1
ci_diff = stats.t.interval(confidence_level, df=data_cleaned, loc=mean_diff, scale=se_diff)

print(f"Confidence Interval for Difference in Cholesterol Levels (Male - Female) ({confidence_level*100}%) : {ci_diff[0]:.3f} to {ci_diff[1]:.3f}")

```

Confidence Interval for Mean Age (95.0%): 52.901 to 54.121
 Confidence Interval for Difference in Cholesterol Levels (Male - Female) (95.0%): -6.383 to -37.290

Part Two: 3. T-test or ANOVA

Conduct a T-test to check if there is a significant difference between two groups, or Perform an ANOVA to see if all groups have the same mean for a characteristic.

```

In [119...]: # Separate cholesterol levels by chest pain type (cp)
cp_groups = []
for cp in data_cleaned['cp'].unique():
    # Filter cholesterol values for each unique chest pain type without dropping null values
    chol_values = data_cleaned[data_cleaned['cp'] == cp]['chol']
    cp_groups.append(chol_values)

#cp_groups = [data_cleaned[data_cleaned['cp'] == cp]['chol'] for cp in data_cleaned['cp'].unique()]

# Perform one-way ANOVA
f_stat, p_value = stats.f_oneway(*cp_groups)

# Output the result

```

```

print("ANOVA Results for Cholesterol Levels across Chest Pain Types:")
print(f"F-statistic: {f_stat:.4f}")
print(f"P-value: {p_value:.4f}")

# Interpretation
if p_value < 0.05:
    print("Result: Significant differences in cholesterol levels across chest pain")
else:
    print("Result: No significant differences in cholesterol levels across chest pa

```

ANOVA Results for Cholesterol Levels across Chest Pain Types:
F-statistic: 7.5912
P-value: 0.0001
Result: Significant differences in cholesterol levels across chest pain types (p < 0.05).

Part Two: 4. Non-Parametric Test

Conduct a non-parametric test for the same variable as in Exercise 3 and compare the conclusions with ANOVA results.

```

In [73]: # Conduct the Kruskal-Wallis test
kruskal_stat, kruskal_p_value = kruskal(*cp_groups)

# Output the result
print("Kruskal-Wallis Test:")
print(f"Statistic: {kruskal_stat}, p-value: {kruskal_p_value}")

# Interpretation based on p-value
if kruskal_p_value < 0.05:
    print("Conclusion: There is a statistically significant difference in cholesterol levels among the chest pain types")
else:
    print("Conclusion: No statistically significant difference in cholesterol levels among the chest pain types")

```

Kruskal-Wallis Test:
Statistic: 12.772943982536457, p-value: 0.005154264553910447
Conclusion: There is a statistically significant difference in cholesterol levels among the chest pain types.

Part Two: 5. Correlation Analysis

Identify the strongest correlations and any statistically insignificant relationships.

```

In [75]: quantitative_columns = ['age', 'trestbps', 'chol', 'thalch', 'oldpeak']
correlation_matrix = data_cleaned[quantitative_columns].corr()

print("Correlation Matrix:")
print(correlation_matrix)

#find the strongest correlations like |correlation| > 0.5

strong_correlations = []

```

```

for col1 in quantitative_columns:
    for col2 in quantitative_columns:
        if col1 != col2:
            correlation = correlation_matrix.loc[col1, col2]
            if abs(correlation) > 0.5:
                strong_correlations.append((col1, col2, correlation))

print("\nStrongest Correlations (|correlation| > 0.5):")
for col1, col2, corr in strong_correlations:
    print(f"{col1} and {col2}: correlation = {corr:.2f}")

#Check statistically insignificant relationships (p > 0.05)
insignificant_correlations = []
for col1 in quantitative_columns:
    for col2 in quantitative_columns:
        if col1 != col2:
            corr, p_value = pearsonr(data_cleaned[col1].dropna(), data_cleaned[col2])
            if p_value > 0.05:
                insignificant_correlations.append((col1, col2, p_value))

print("\nStatistically Insignificant Relationships (p > 0.05):")
for col1, col2, p_value in insignificant_correlations:
    print(f"{col1} and {col2}: p-value = {p_value:.4f}")

```

Correlation Matrix:

	age	trestbps	chol	thalch	oldpeak
age	1.000000	0.230784	-0.086010	-0.349715	0.233550
trestbps	0.230784	1.000000	0.089484	-0.104747	0.161217
chol	-0.086010	0.089484	1.000000	0.226047	0.047454
thalch	-0.349715	-0.104747	0.226047	1.000000	-0.149401
oldpeak	0.233550	0.161217	0.047454	-0.149401	1.000000

Strongest Correlations (|correlation| > 0.5):

Statistically Insignificant Relationships (p > 0.05):
chol and oldpeak: p-value = 0.1504
oldpeak and chol: p-value = 0.1504

Part Two: 6. Multiple Linear Regression

Perform a multiple linear regression analysis.

In [77]:

```

# Define the target and predictor variables
X = data_cleaned[['age', 'trestbps', 'thalch', 'oldpeak']] # Predictor variables
y = data_cleaned['chol'] # Target variable

# Drop any rows with missing values in X or y
X = X.dropna()
y = y.loc[X.index] # Keep y aligned with the non-null X

# Add a constant to X to account for the intercept
X = sm.add_constant(X)

```

```

# Fit the model
model = sm.OLS(y, X).fit()

# Output the summary
print(model.summary())

```

OLS Regression Results

```

=====
Dep. Variable:                      chol   R-squared:                 0.070
Model:                             OLS    Adj. R-squared:            0.066
Method:                            Least Squares   F-statistic:             17.30
Date:                            Sat, 26 Oct 2024   Prob (F-statistic):        1.08e-13
Time:                            19:37:33         Log-Likelihood:           -5587.8
No. Observations:                  920    AIC:                     1.119e+04
Df Residuals:                      915    BIC:                     1.121e+04
Df Model:                           4
Covariance Type:                nonrobust
=====

      coef    std err          t      P>|t|      [0.025      0.975]
-----
const     -3.3781    40.494     -0.083     0.934     -82.851     76.094
age       -0.5604     0.408     -1.372     0.170     -1.362     0.241
trestbps   0.6669     0.195      3.422     0.001      0.284     1.049
thalch     1.0068     0.148      6.804     0.000      0.716     1.297
oldpeak    7.7561     3.409      2.275     0.023      1.065     14.447
=====

Omnibus:                   31.741   Durbin-Watson:            0.865
Prob(Omnibus):              0.000   Jarque-Bera (JB):        34.196
Skew:                      -0.451   Prob(JB):                3.75e-08
Kurtosis:                   3.278   Cond. No.                 2.32e+03
=====


```

Notes:

- [1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
- [2] The condition number is large, 2.32e+03. This might indicate that there are strong multicollinearity or other numerical problems.

```

In [79]: X = data_cleaned[['age', 'trestbps', 'thalch', 'oldpeak']].dropna()
y = data_cleaned['chol'].loc[X.index]

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

# Initialize and train the model
model = LinearRegression()
model.fit(X_train, y_train)

# Make predictions
y_pred = model.predict(X_test)

# Evaluate the model
print("R-squared:", r2_score(y_test, y_pred))
print("Mean Squared Error:", mean_squared_error(y_test, y_pred))
print("Coefficients:", model.coef_)
print("Intercept:", model.intercept_)

```

R-squared: 0.04126403991345806
Mean Squared Error: 10766.696003312703
Coefficients: [-0.50727281 0.5572145 1.08752555 10.46541804]
Intercept: -7.477523001130095