statistical analysis information

In [69]:
```
import numpy as np
import pandas as pd
```

In [2]:
```
data=pd.read_csv("C:/Users/chakr/OneDrive/heart-disease.csv")
df=data.head(8)
df
```

Out[2]:

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal	target
0	63	1	3	145	233	1	0	150	0	2.3	0	0	1	1
1	37	1	2	130	250	0	1	187	0	3.5	0	0	2	1
2	41	0	1	130	204	0	0	172	0	1.4	2	0	2	1
3	56	1	1	120	236	0	1	178	0	0.8	2	0	2	1
4	57	0	0	120	354	0	1	163	1	0.6	2	0	2	1
5	57	1	0	140	192	0	1	148	0	0.4	1	0	1	1
6	56	0	1	140	294	0	0	153	0	1.3	1	0	2	1
7	44	1	1	120	263	0	1	173	0	0.0	2	0	3	1

HERE THE SOME IMP.. INFOMATION INDICATES: **Sex of the patient (1 = male, 0 = female) **cp:Chest pain type (0 = typical angina, 1 = atypical angina, 2 = non-anginal pain, 3 = asymptomatic) **fbs:Fasting blood sugar > 120 mg/dl (1 = true; 0 = false) **exang:Exercise-induced angina (1 = yes; 0 = no)

Descreptive analysis

This provide simple summaries about the sample and the measures, forming the basis of virtually every quantitative analysis of data. Descriptive statistics help to simplify large amounts of data in a sensible way.

MEAN

In [3]:
```
# mean for particular attribute
print("Mean:\n", df['age'].mean())
```

```
Mean:
 51.375
```

In [4]:
```
# for all attributes in a dataset
df.mean()
```

Out[4]:
```
age          51.3750
sex           0.6250
cp            1.1250
trestbps    130.6250
chol        253.2500
fbs           0.1250
restecg       0.6250
thalach     165.5000
exang         0.1250
oldpeak       1.2875
slope         1.2500
ca            0.0000
thal          1.8750
target        1.0000
dtype: float64
```

MEDIAN

In [6]:
```
print( df['age'].median())
# for particular column
```

```
56.0
```

In [7]:
```
# for all attributes
df.median()
```

Out[7]:
```
age          56.00
sex           1.00
cp            1.00
trestbps    130.00
chol        243.00
fbs           0.00
restecg       1.00
thalach     167.50
exang         0.00
oldpeak       1.05
slope         1.50
ca            0.00
thal          2.00
target        1.00
dtype: float64
```

MODE

In [15]:
```
# mode for particular attribute
print("Mode:\n", df['age'].mode())
```

```
Mode:
 0    56
1    57
Name: age, dtype: int64
```

In [16]:
```
print("Mode:\n", df.mode().iloc[0])
# for all attributes
```

```
Mode:
 age          56.0
sex          1.0
cp           1.0
trestbps   120.0
chol       192.0
fbs          0.0
restecg      1.0
thalach    148.0
exang        0.0
oldpeak      0.0
slope        2.0
ca           0.0
thal         2.0
target       1.0
Name: 0, dtype: float64
```

standard devation and variance

In [18]:
```
# for all attributes
print("Standard Deviation:\n", df.std())
#VARIANCE
print("Variance:\n", df.var())
```

```
Standard Deviation:
 age          9.334077
sex         0.517549
cp          0.991031
trestbps   10.155048
chol       51.825945
fbs         0.353553
restecg     0.517549
thalach    14.272851
exang       0.353553
oldpeak     1.139470
slope       0.886405
ca          0.000000
thal        0.640870
target      0.000000
dtype: float64
Variance:
 age           87.125000
sex            0.267857
cp             0.982143
trestbps     103.125000
chol        2685.928571
fbs            0.125000
restecg        0.267857
thalach      203.714286
exang          0.125000
oldpeak        1.298393
slope          0.785714
ca             0.000000
thal           0.410714
target         0.000000
dtype: float64
```

RANGE

The range is the difference between the maximum and minimum values in a dataset.

In [19]:
```
print("Range:\n", df.max() - df.min())
#FOR all attributes
```

```
Range:
 age          26.0
sex          1.0
cp           3.0
trestbps    25.0
chol       162.0
fbs          1.0
restecg      1.0
thalach     39.0
exang        1.0
oldpeak      3.5
slope        2.0
ca           0.0
thal         2.0
target       0.0
dtype: float64
```

skewness

Skewness measures the asymmetry of the data distribution. A positive skew indicates that the right tail of the distribution is longer or fatter than the left tail (right-skewed). A negative skew indicates that the left tail is longer or fatter than the right tail (left-skewed). Skewness provides insights into the direction and degree of asymmetry in the data.

In [20]:
```
print("Skewness:\n", df.skew())
```

```
Skewness:
 age        -0.553590
sex        -0.644061
cp          0.862279
trestbps    0.185437
chol        0.996917
fbs         2.828427
restecg    -0.644061
thalach     0.100624
exang       2.828427
oldpeak     1.106675
slope      -0.615356
ca          0.000000
thal        0.067843
target      0.000000
dtype: float64
```

kurtosis

Kurtosis measures the “tailedness” or the peak of the data distribution. High kurtosis means that the data have heavy tails or outliers, whereas low kurtosis indicates light tails or fewer outliers.

In [22]:
```
print("Kurtosis:\n", df.kurt())
```

```
Kurtosis:
 age        -1.357701
sex        -2.240000
cp          0.840463
trestbps   -1.792680
chol        1.088883
fbs         8.000000
restecg    -2.240000
thalach    -1.438337
exang       8.000000
oldpeak     0.929133
slope      -1.480992
ca          0.000000
thal        0.741021
target      0.000000
dtype: float64
```

Inferential Statistics

Inferential statistics involve drawing conclusions about a population based on a sample. Here are some key techniques: .Hypothesis Testing: Used to determine whether there is enough evidence to reject a null hypothesis. .Confidence Intervals: Provide a range of values that likely contain the population parameter. .Regression Analysis: Examines relationships between variables.

In [25]:
```
import numpy as np
from scipy import stats
```

In [46]:
```
# Example data:
cp_values = df['cp']

# Hypothetical population mean
population_mean = 500.00

# Perform one-sample t-test
t_stat, p_value = stats.ttest_1samp(cp_values, population_mean)

print(f"T-Statistic: {t_stat}")
print(f"P-Value: {p_value}")
```

```
T-Statistic: -1423.8013587192945
P-Value: 2.2265103305843713e-20
```

** CONFIDENCE INTERVALS

Provide a range of values that likely contain the population parameter.

In [50]:
```
import numpy as np
from scipy import stats

# Sample mean and standard error
sample_mean = np.mean(cp_values)
standard_error = stats.sem(cp_values)

# Compute 95% confidence interval for BMI
confidence_interval = stats.norm.interval(0.95, loc=sample_mean, scale=standard_error)
print(f" mean of cp for s_data is:{sample_mean}")
print(f"95% Confidence Interval is: {confidence_interval}")
```

```
 mean of cp for s_data is:1.125
95% Confidence Interval is: (np.float64(0.4382630287226813), np.float64(1.8117369712773188))
```

*** REGRESSION ANALYSIS :Examines relationships between variables.

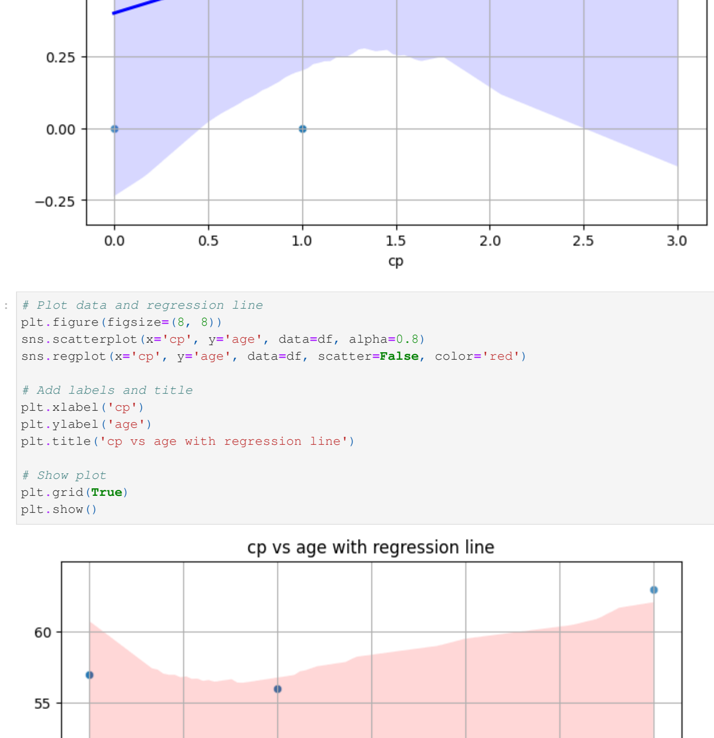
In [57]:
```
import statsmodels.api as sm

# Define independent variable (add constant for intercept)
A = df[['cp']]
A = sm.add_constant(A)

# Define dependent variable
B = df['age']

# Fit linear regression model
model = sm.OLS(B, A).fit()

# Print model summary
print(model.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                    age   R-squared:                       0.001
Model:                            OLS   Adj. R-squared:                 -0.165
Method:                 Least Squares   F-statistic:                  0.008083
Date:                Sun, 08 Sep 2024   Prob (F-statistic):              0.931
Time:                        21:57:59   Log-Likelihood:                -28.681
No. Observations:                   8   AIC:                             61.36
Df Residuals:                       6   BIC:                             61.52
Df Model:                           1
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const         51.7636      5.601      9.241      0.000      38.058      65.470
cp            -0.3455      3.843     -0.090      0.931      -9.748       9.057
==============================================================================
Omnibus:                        1.286   Durbin-Watson:                   1.769
Prob(Omnibus):                  0.526   Jarque-Bera (JB):                0.747
Skew:                          -0.376   Prob(JB):                        0.688
Kurtosis:                       1.706   Cond. No.                         3.04
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
C:\Users\chakr\AppData\Local\Programs\Python\Python312\Lib\site-packages\scipy\stats\_axis_nan_policy.py:418: UserWarning: `kurtosistest` p-value may be inaccurate with f
ewer than 20 observations; only n=8 observations were given.
  return hypotest_fun_in(*args, **kwds)
```

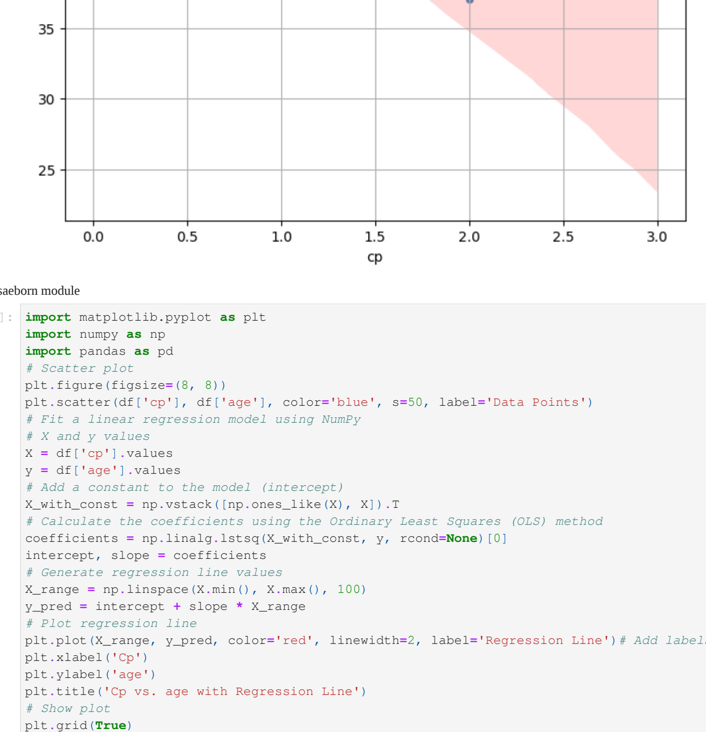
graphic repsentation

In [59]:
```
import matplotlib.pyplot as plt
import seaborn as sns
```

In [63]:
```
# Plot data and regression line
plt.figure(figsize=(8, 8))
sns.scatterplot(x='cp', y='sex', data=df, alpha=0.8)
sns.regplot(x='cp', y='sex', data=df, scatter=False, color='blue')

# Add labels and title
plt.xlabel('cp')
plt.ylabel('sex')
plt.title('cp vs sex with regression line')

# Show plot
plt.grid(True)
plt.show()
```

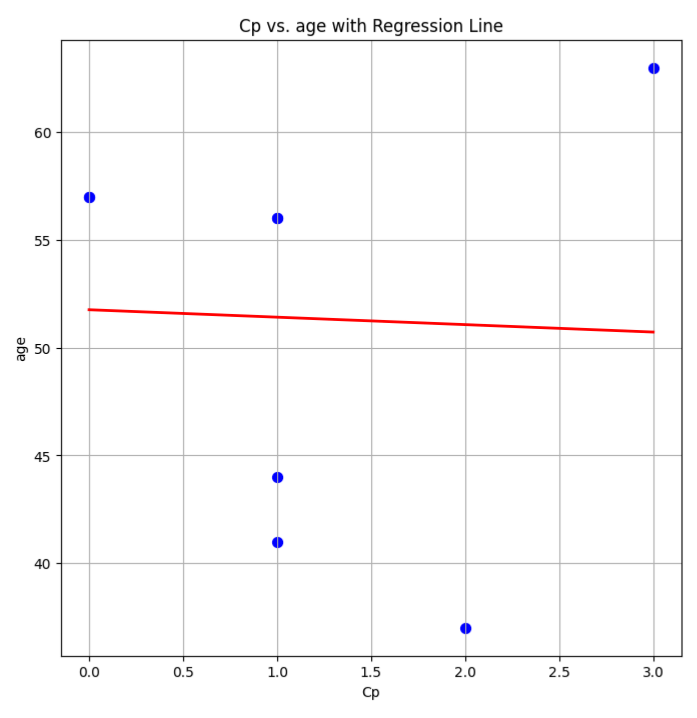


In [67]:
```
# Plot data and regression line
plt.figure(figsize=(8, 8))
sns.scatterplot(x='cp', y='age', data=df, alpha=0.8)
sns.regplot(x='cp', y='age', data=df, scatter=False, color='red')

# Add labels and title
plt.xlabel('cp')
plt.ylabel('age')
plt.title('cp vs age with regression line')

# Show plot
plt.grid(True)
plt.show()
```



without an saeborn module

In [68]:
```
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
# Scatter plot
plt.figure(figsize=(8, 8))
plt.scatter(df['cp'], df['age'], color='blue', s=50, label='Data Points')
# Fit a linear regression model using NumPy
# X and y values
X = df['cp'].values
y = df['age'].values
# Add a constant to the model (intercept)
X_with_const = np.vstack([np.ones_like(X), X]).T
# Calculate the coefficients using the Ordinary Least Squares (OLS) method
coefficients = np.linalg.lstsq(X_with_const, y, rcond=None)[0]
intercept, slope = coefficients
# Generate regression line values
X_range = np.linspace(X.min(), X.max(), 100)
y_pred = intercept + slope * X_range
# Plot regression line
plt.plot(X_range, y_pred, color='red', linewidth=2, label='Regression Line')# Add labels and title
plt.xlabel('Cp')
plt.ylabel('age')
plt.title('Cp vs. age with Regression Line')
# Show plot
plt.grid(True)
plt.show()
```


Cp vs. age with Regression Line
Cp
age
0.0
0.5
1.0
1.5
2.0
2.5
3.0
40
45
50
55
60