Exp 1 or Linear regression

Apply least squares model for the regression of number of units sold on TV advertising budget for the Advertising data, for least squares coefficient estimates for simple linear regression

In [1]:
```
import pandas as pd
from sklearn.linear_model import LinearRegression
from sklearn.metrics import r2_score, mean_squared_error

df = pd.read_csv('Advertising.csv')

X = df['TV'].values.reshape(-1, 1)
y = df['Sales']

model = LinearRegression()
model.fit(X, y)
y_pred = model.predict(X)
r_squared = r2_score(y, y_pred)
r_squared
```

Out[1]: 0.8121757029987414

Multiple linear regression

In [2]:
```
import pandas as pd
from sklearn.linear_model import LinearRegression
from sklearn.metrics import r2_score, mean_squared_error
df = pd.read_csv('Advertising.csv')
X = df[['TV', 'Radio', 'Newspaper']]
y = df['Sales']
model = LinearRegression()
model.fit(X, y)
y_pred = model.predict(X)
r_squared = r2_score(y, y_pred)
r_squared
```

Out[2]: 0.9025912899684558

KNN

In [3]:

```
import pandas as pd
from sklearn.neighbors import KNeighborsRegressor
from sklearn.model_selection import train_test_split
df = pd.read_csv('Advertising.csv')

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2
model = KNeighborsRegressor(n_neighbors=5)
model.fit(X_train, y_train)
y_pred = model.predict(X_test)
r_squared = r2_score(y_test, y_pred)
r_squared
```

Out[3]: 0.8991773755626823

Navi Bayesian

In [4]:

```
import pandas as pd
from sklearn.neighbors import KNeighborsRegressor
from sklearn.model_selection import train_test_split
from sklearn.metrics import r2_score

df = pd.read_csv('Advertising.csv')

X = df[['TV', 'Radio', 'Newspaper']]
y = df['Sales']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2

model = KNeighborsRegressor(n_neighbors=5)
model.fit(X_train, y_train)

y_pred = model.predict(X_test)

r_squared = r2_score(y_test, y_pred)
print("R-squared score:", r_squared)

```

```
R-squared score: 0.8991773755626823
```

Gradient Descent

In [5]:
```
import pandas as pd
from sklearn.linear_model import SGDRegressor
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split

df = pd.read_csv('Advertising.csv')

X = df[['TV', 'Radio', 'Newspaper']]
y = df['Sales']
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
model = SGDRegressor(loss='squared_error')
model.fit(X_train_scaled, y_train)

y_test_pred = model.predict(X_test_scaled)
test_r_squared = r2_score(y_test, y_test_pred)

print("Testing R-squared:", test_r_squared)
```

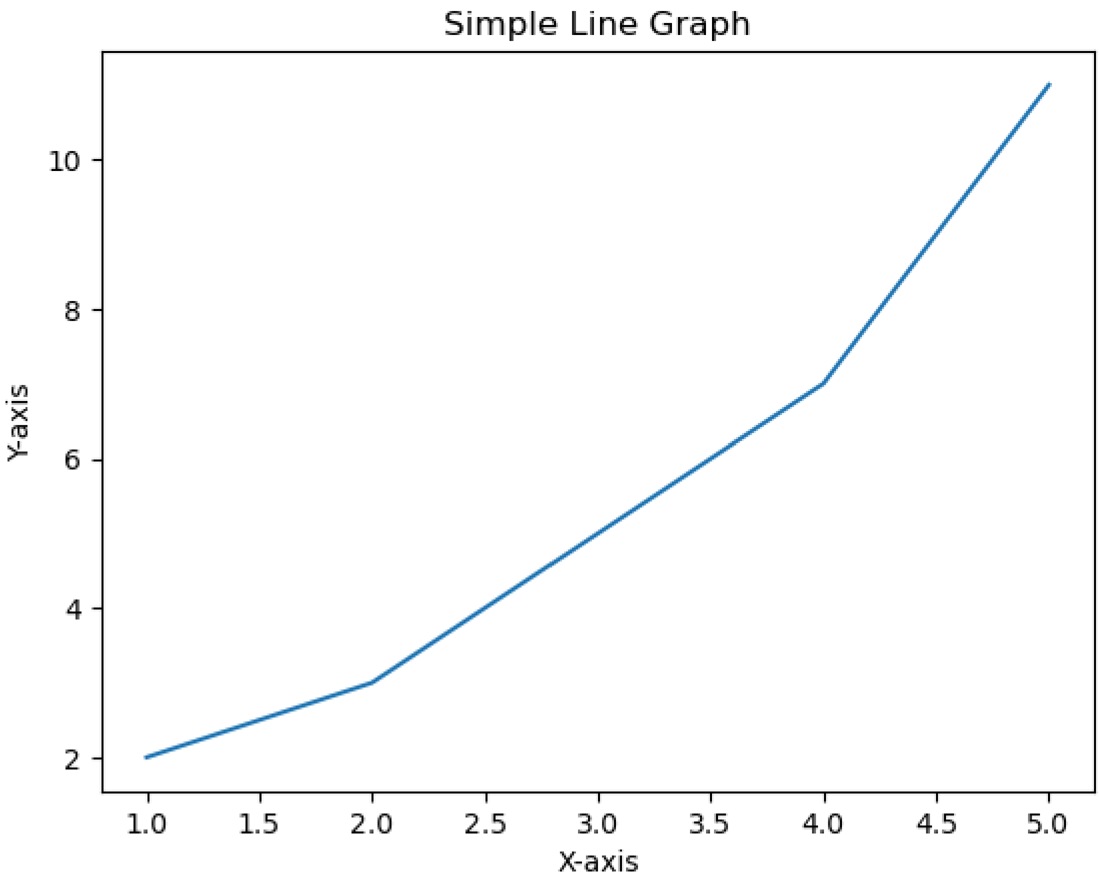
```
Testing R-squared: 0.9061334961284013
```

Matplotlib

Simple line graph

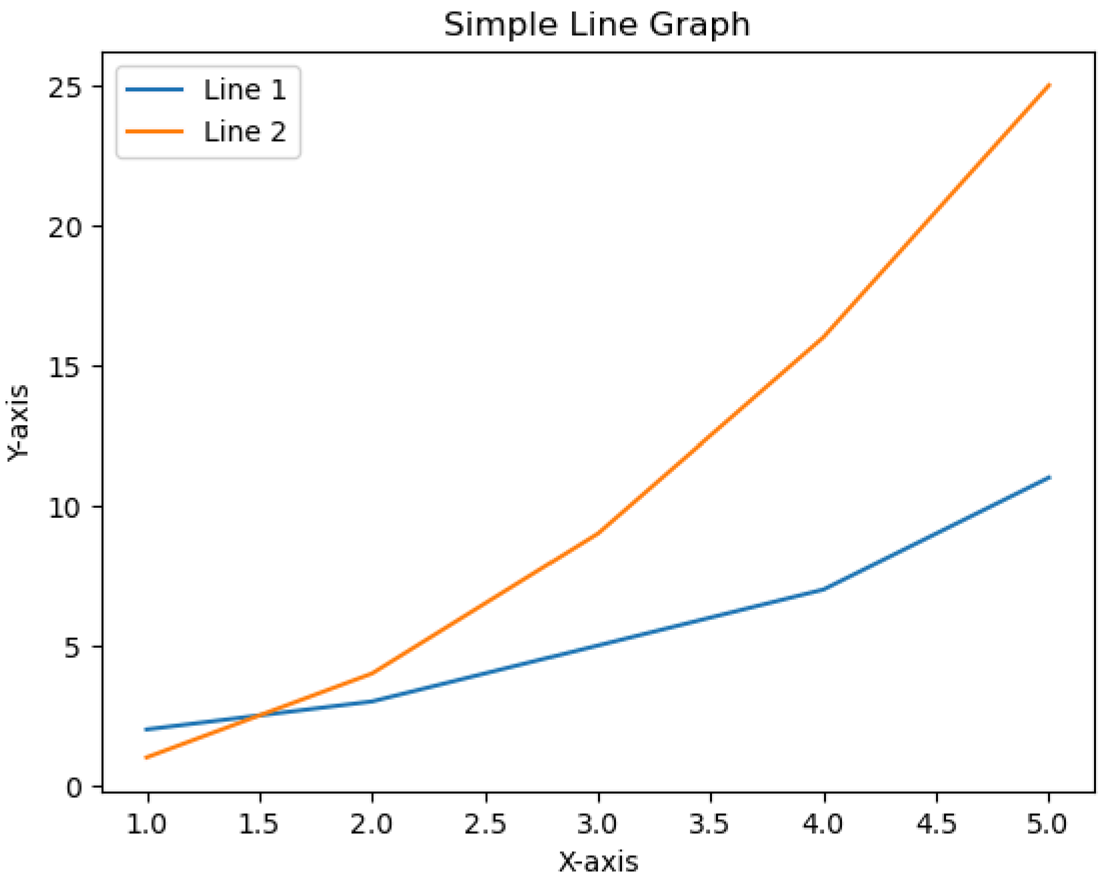
In [6]:

```
import matplotlib.pyplot as plt


x = [1, 2, 3, 4, 5]
y = [2, 3, 5, 7, 11]

plt.plot(x, y)

plt.xlabel('X-axis')
plt.ylabel('Y-axis')
plt.title('Simple Line Graph')


plt.show()

```



Multiple line graph

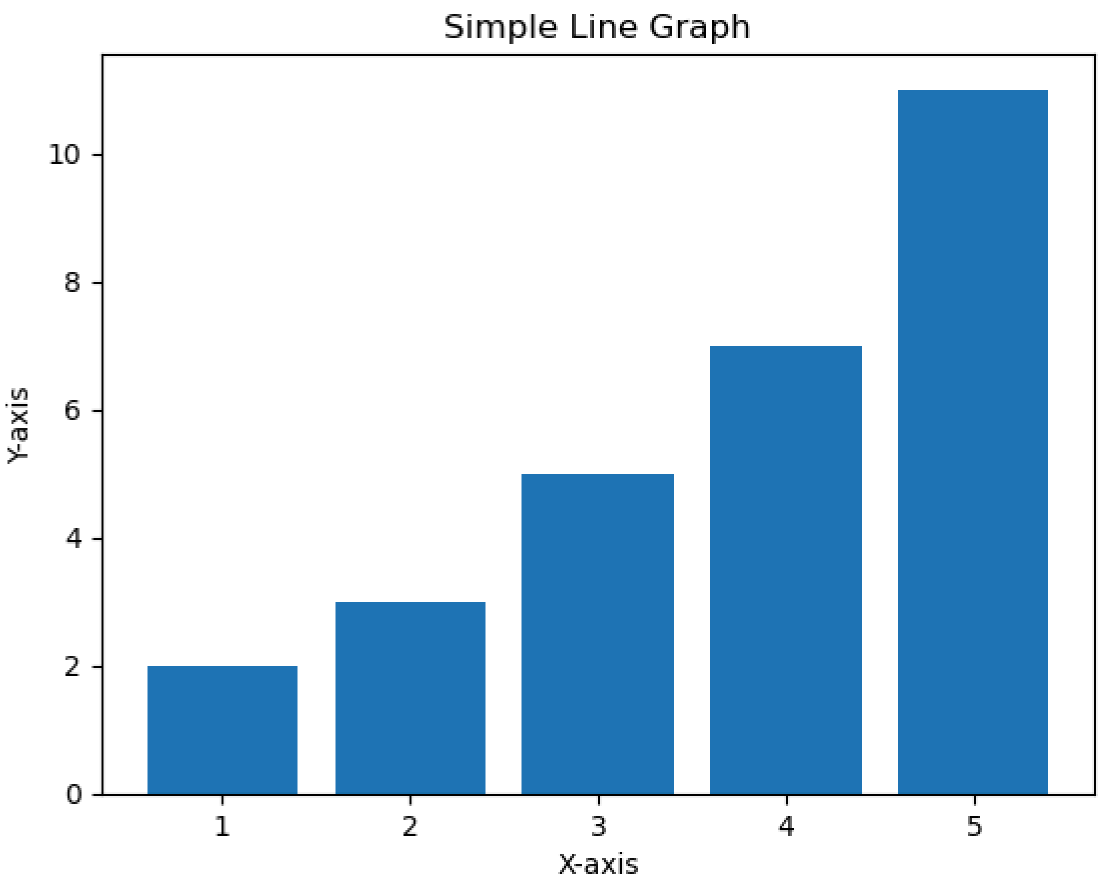
In [7]:
```
import matplotlib.pyplot as plt


x = [1, 2, 3, 4, 5]
y1 = [2, 3, 5, 7, 11]
y2 = [1, 4, 9, 16, 25]

plt.plot(x, y1, label='Line 1')
plt.plot(x, y2, label='Line 2')

plt.xlabel('X-axis')
plt.ylabel('Y-axis')
plt.title('Simple Line Graph')

plt.legend()

plt.show()

```



Bar graph

In [8]:

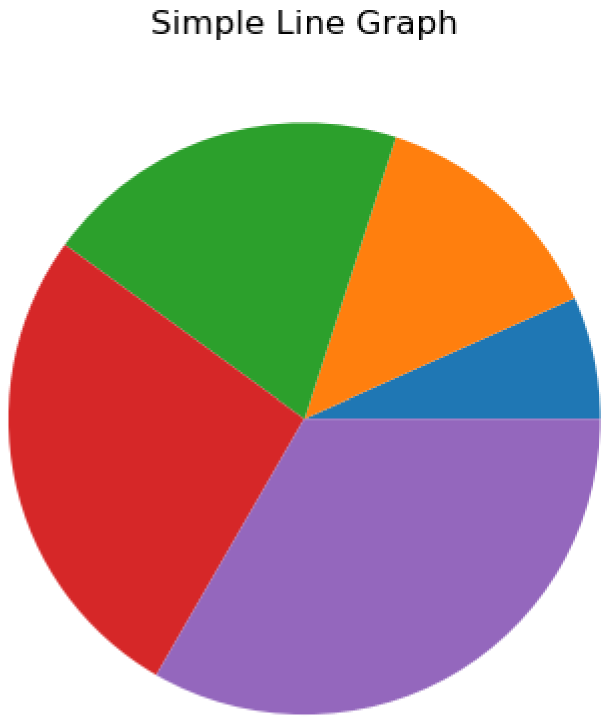
```
import matplotlib.pyplot as plt


x = [1, 2, 3, 4, 5]
y = [2, 3, 5, 7, 11]

plt.bar(x, y)

plt.xlabel('X-axis')
plt.ylabel('Y-axis')
plt.title('Simple Line Graph')


plt.show()
```



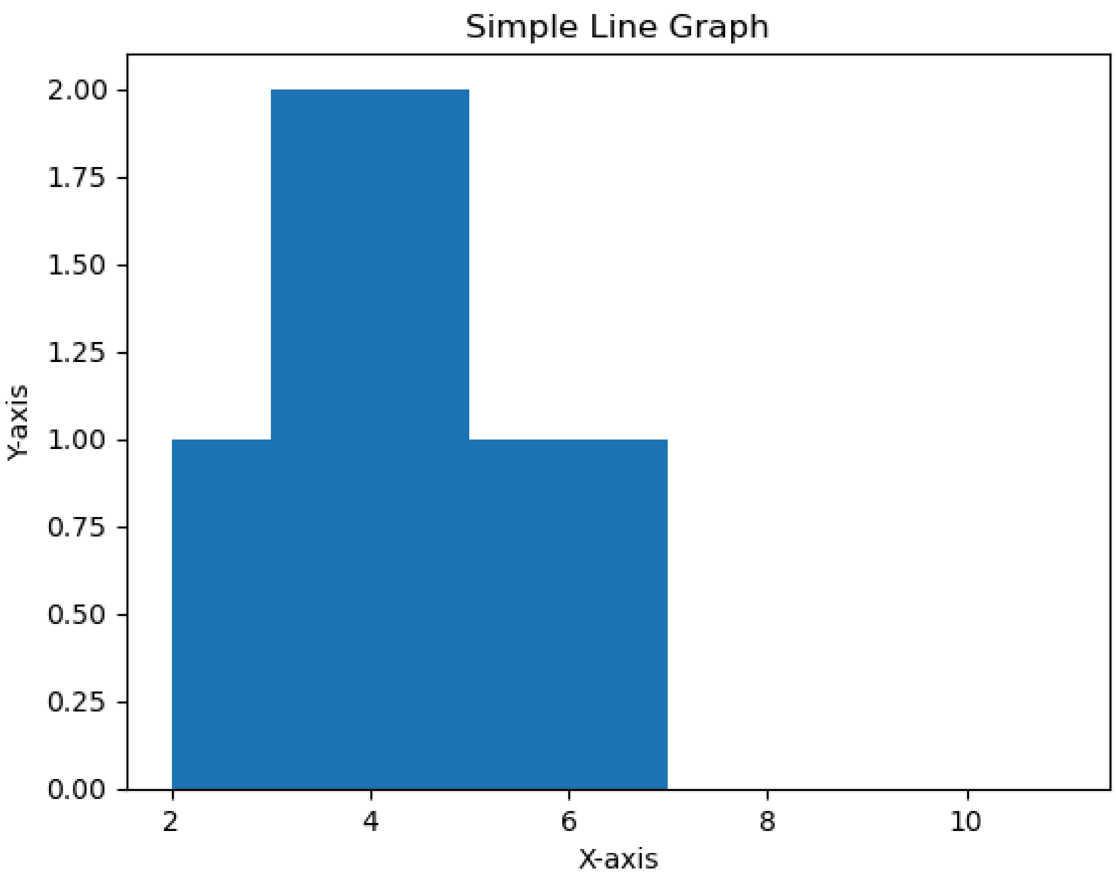
Pie chart

In [22]:

```
import matplotlib.pyplot as plt
x = [1, 2, 3, 4, 5]
plt.pie(x)
plt.title('Simple Line Graph')
plt.show()
```



In [10]:
```
import matplotlib.pyplot as plt


x = [1, 2, 3, 4, 5]
y = [2, 3, 5, 7, 11]

plt.hist(x, y)

plt.xlabel('X-axis')
plt.ylabel('Y-axis')
plt.title('Simple Line Graph')


plt.show()
```



Exp 2

Compute t-statistic, Residual standard error, F-statistic and residual sum of squares (RSS) errors.

In [ ]:

```
import numpy as np
import statsmodels.api as sm
X = np.array([1, 2, 3, 4, 5])
y = np.array([2, 3, 4, 5, 6])
X = sm.add_constant(X)
model = sm.OLS(y, X).fit()
t_statistic = model.tvalues
residual_standard_error = np.sqrt(model.mse_resid)
f_statistic = model.fvalue
rss = model.ssr
print("T-Statistic:", t_statistic)
print("Residual Standard Error:", residual_standard_error)
print("F-Statistic:", f_statistic)
print("Residual Sum of Squares (RSS):", rss)
```

Exp3

Do any of the predictors appear to be statistically significant? If so, which ones?

In [32]:
```python
import numpy as np
import statsmodels.api as sm

# Sample data
X = np.array([1, 2, 3, 4, 5])
y = np.array([2, 3, 4, 5, 6])

# Add constant for intercept term
X = sm.add_constant(X)

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

# Get model summary
print(model.summary())

# Extract p-values
p_values = model.pvalues[1:]  # Exclude intercept term
print("P-Values:", p_values)

# Identify statistically significant predictors
significant_predictors = np.where(p_values < 0.05)[0]
print("Significant Predictor Indices:", significant_predictors)

```

```
                            OLS Regression Results
==========================================================================
====
Dep. Variable:                      y   R-squared:
1.000
Model:                            OLS   Adj. R-squared:
1.000
Method:                 Least Squares   F-statistic:                 6.085
e+32
Date:                Fri, 03 May 2024   Prob (F-statistic):           1.47
e-49
Time:                        14:56:10   Log-Likelihood:                 17
7.15
No. Observations:                   5   AIC:                            -3
50.3
Df Residuals:                       3   BIC:                            -3
51.1
Df Model:                           1
Covariance Type:            nonrobust
==========================================================================
====
                 coef    std err          t      P>|t|      [0.025      0.
975]
--------------------------------------------------------------------------
----
const          1.0000   1.34e-16   7.44e+15      0.000       1.000
1.000
x1             1.0000   4.05e-17   2.47e+16      0.000       1.000
1.000
==========================================================================
====
Omnibus:                          nan   Durbin-Watson:
1.000
Prob(Omnibus):                    nan   Jarque-Bera (JB):
1.888
Skew:                           1.500   Prob(JB):
0.389
Kurtosis:                       3.250   Cond. No.
8.37
==========================================================================
====

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is cor
rectly specified.
P-Values: [1.46929683e-49]
Significant Predictor Indices: [0]

C:\Users\hites\anaconda3\Lib\site-packages\statsmodels\stats\stattools.py:
74: ValueWarning: omni_normtest is not valid with less than 8 observation
s; 5 samples were given.
  warn("omni_normtest is not valid with less than 8 observations; %i "
```

Exp4

Compute the confusion matrix and overall fraction of correct predictions. Explain what the confusion matrix is telling you about the types of mistakes made by KNN

In [17]:
```
from sklearn.metrics import confusion_matrix, accuracy_score


y_true = [0, 1, 0, 1, 0, 1, 1, 0, 1, 0]
y_pred = [0, 1, 1, 1, 0, 0, 1, 0, 1, 1]

# Compute confusion matrix
cm = confusion_matrix(y_true, y_pred)
print("Confusion Matrix:")
print(cm)

# Compute overall fraction of correct predictions
accuracy = accuracy_score(y_true, y_pred)
print("Overall Accuracy:", accuracy)


```

```
Confusion Matrix:
[[3 2]
 [1 4]]
Overall Accuracy: 0.7
```

Exp 5

Compute Mallow’s Cp, Akaike information criterion (AIC), adjusted R Squared and Bayesian information criterion (BIC)

In [18]:

```
import numpy as np
import statsmodels.api as sm


X = np.array([[1, 2, 3, 4, 5], [2, 3, 4, 5, 6]]).T
y = np.array([2, 3, 4, 5, 6])

model = sm.OLS(y, sm.add_constant(X)).fit()

residuals = model.resid
p = len(model.params)
n = len(y)
rss = np.sum(residuals ** 2)

Cp = (1/n) * (rss + 2 * p * (rss/n))
AIC = (2 * p) + (n * np.log(rss/n))
adj_r_squared = 1 - ((rss/(n - p - 1)) / (np.var(y)))
BIC = n * np.log(rss/n) + p * np.log(n)


print("Mallow's Cp:", Cp)
print("AIC:", AIC)
print("Adjusted R-squared:", adj_r_squared)
print("BIC:", BIC)

```

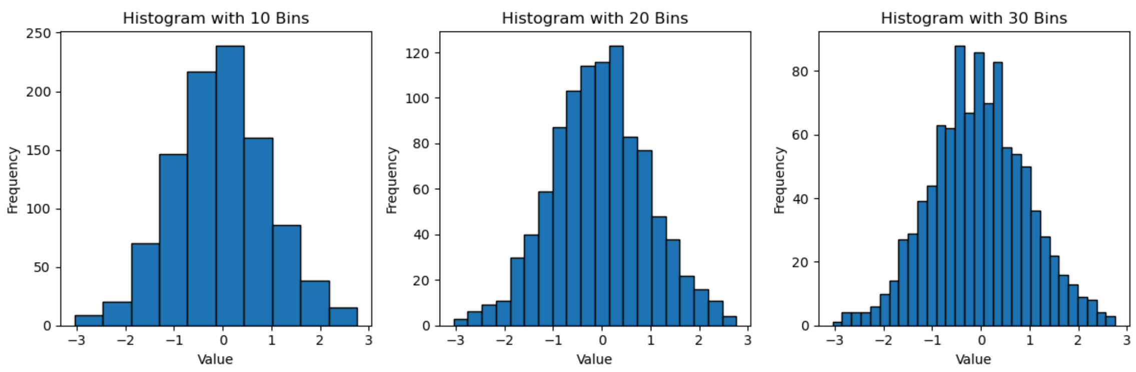
```
Mallow's Cp: 1.8591479383796198e-29
AIC: -328.71653386035314
Adjusted R-squared: 1.0
BIC: -329.8882201230508
```

Exp6

Produce some histograms with differing numbers of bins for a few of the quantitative variables

In [23]:

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

# Generate some example data
np.random.seed(0)
data = np.random.normal(loc=0, scale=1, size=1000)  # Example data with

# Create histograms with different numbers of bins
bins_list = [10, 20, 30]  # Different numbers of bins
titles = ['Histogram with 10 Bins', 'Histogram with 20 Bins', 'Histogra

plt.figure(figsize=(12, 4))
for i, bins in enumerate(bins_list, start=1):
    plt.subplot(1, len(bins_list), i)
    plt.hist(data, bins=bins, edgecolor='black')
    plt.title(titles[i-1])
    plt.xlabel('Value')
    plt.ylabel('Frequency')

plt.tight_layout()
plt.show()

```



Exp 7

Continue exploring the data, and provide a brief summary of what you discover.

Descriptive Statistics: Calculate summary statistics such as mean, median, standard deviation, minimum, maximum, and quartiles for quantitative variables. For categorical variables, count the frequency of each category. Data Visualization: Create visualizations such as histograms, box plots, scatter plots, and bar plots to understand the distribution, relationships, and patterns in the data. Correlation Analysis: Compute correlations between pairs of quantitative variables to identify any linear relationships. Visualize the correlation matrix using a heatmap. Outlier Detection: Identify any outliers in the data using statistical methods or visualization techniques. Missing Values: Check for missing values in the dataset and decide on appropriate strategies for handling them, such as imputation or removal. Feature Engineering: Create new features or transform existing features to better represent the underlying patterns in the data. Dimensionality Reduction: Apply techniques such as principal component analysis (PCA) or t-distributed stochastic neighbor embedding (t-SNE) to reduce the dimensionality of the data and visualize high-dimensional data in lower dimensions. Cluster Analysis: Explore whether there are natural groupings or clusters in the data using clustering algorithms such as K-means or hierarchical clustering. Based on the results of these analyses, we can provide a summary of the key findings, insights, and patterns observed in the data. This summary can help stakeholders better understand the

Exp 8

Carry out the pearson product moment correlation, spearman rho correlation and kendall’s tau

In [24]:

```
import numpy as np
from scipy.stats import pearsonr, spearmanr, kendalltau

x = np.array([1, 2, 3, 4, 5])
y = np.array([2, 3, 4, 5, 6])

pearson_corr, pearson_p_value = pearsonr(x, y)
print("Pearson correlation coefficient:", pearson_corr)
print("Pearson p-value:", pearson_p_value)

spearman_corr, spearman_p_value = spearmanr(x, y)
print("Spearman's rank correlation coefficient (Spearman rho):", spearm
print("Spearman p-value:", spearman_p_value)

kendall_corr, kendall_p_value = kendalltau(x, y)
print("Kendall's tau correlation coefficient:", kendall_corr)
print("Kendall p-value:", kendall_p_value)

```

```
Pearson correlation coefficient: 1.0
Pearson p-value: 0.0
Spearman's rank correlation coefficient (Spearman rho): 0.9999999999999999
Spearman p-value: 1.4042654220543672e-24
Kendall's tau correlation coefficient: 0.9999999999999999
Kendall p-value: 0.016666666666666666
```

Exp 9

Perform Simple Hypothesis testing, student’s t-test, paired t and u test, correlation and covariance, tests for association

In [26]:

```python
import numpy as np
from scipy.stats import ttest_ind, ttest_rel, wilcoxon, chi2_contingenc
import pandas as pd

# Example data
data = {
    'X': [1, 2, 3, 4, 5],
    'Y': [2, 3, 4, 5, 6],
    'Category': ['A', 'B', 'A', 'B', 'A']
}
df = pd.DataFrame(data)

# Simple Hypothesis Testing
mean_value = 3  # Known value for comparison
sample_mean = np.mean(df['X'])
t_statistic, p_value = ttest_ind(df['X'], [mean_value])
print("Simple Hypothesis Testing:")
print("Sample mean:", sample_mean)
print("t-statistic:", t_statistic)
print("p-value:", p_value)

# Student's t-test
t_statistic, p_value = ttest_ind(df[df['Category'] == 'A']['Y'], df[df[
print("\nStudent's t-test:")
print("t-statistic:", t_statistic)
print("p-value:", p_value)

# Paired t-test
t_statistic, p_value = ttest_rel(df['X'], df['Y'])
print("\nPaired t-test:")
print("t-statistic:", t_statistic)
print("p-value:", p_value)

# Wilcoxon Signed-Rank Test (U Test)
statistic, p_value = wilcoxon(df['X'], df['Y'])
print("\nWilcoxon Signed-Rank Test (U Test):")
print("Statistic:", statistic)
print("p-value:", p_value)

# Correlation and Covariance
correlation = df['X'].corr(df['Y'])
covariance = df['X'].cov(df['Y'])
print("\nCorrelation and Covariance:")
print("Correlation coefficient:", correlation)
print("Covariance:", covariance)

# Tests for Association
contingency_table = pd.crosstab(df['X'], df['Category'])
chi2, p_value, _, _ = chi2_contingency(contingency_table)
print("\nTests for Association:")
print("Chi-square statistic:", chi2)
print("p-value:", p_value)

```

```
Simple Hypothesis Testing:
Sample mean: 3.0
t-statistic: 0.0
p-value: 1.0

Student's t-test:
t-statistic: 0.0
p-value: 1.0

Paired t-test:
t-statistic: -inf
p-value: 0.0

Wilcoxon Signed-Rank Test (U Test):
Statistic: 0.0
p-value: 0.0625

Correlation and Covariance:
Correlation coefficient: 0.9999999999999999
Covariance: 2.5

Tests for Association:
Chi-square statistic: 5.000000000000001
p-value: 0.2872974951836456
```

```
C:\Users\hites\anaconda3\Lib\site-packages\scipy\stats\_axis_nan_policy.p
y:523: RuntimeWarning: Precision loss occurred in moment calculation due t
o catastrophic cancellation. This occurs when the data are nearly identica
l. Results may be unreliable.
  res = hypotest_fun_out(*samples, **kwds)
```

Exp 10

Perform Programming for Eigen values and Eigen vectors

In [28]:

```python
import numpy as np

# Example matrix
A = np.array([[1, 2],
              [2, 1]])

# Compute eigenvalues and eigenvectors
eigenvalues, eigenvectors = np.linalg.eig(A)

# Print eigenvalues and eigenvectors
print("Eigenvalues:")
print(eigenvalues)
print("\nEigenvectors:")
print(eigenvectors)

```

```
Eigenvalues:
[ 3. -1.]

Eigenvectors:
[[ 0.70710678 -0.70710678]
 [ 0.70710678  0.70710678]]
```