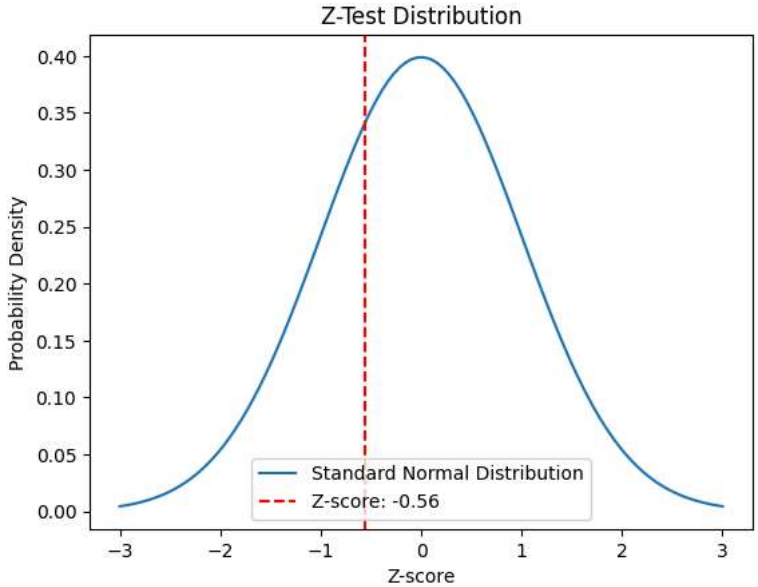
```python
#z-test

import numpy as np
import scipy.stats as stats
import matplotlib.pyplot as plt

# Sample Data
sample = np.array([10, 12, 14, 16, 18])
population_mean = 15
population_std = 4
sample_mean = np.mean(sample)
sample_size = len(sample)

# Compute Z-score
z_score = (sample_mean - population_mean) / (population_std / np.sqrt(sample_size))
p_value = 2 * (1 - stats.norm.cdf(abs(z_score)))

# Print Results
print(f"Z-Score: {z_score:.2f}")
print(f"P-Value: {p_value:.5f}")

# Plot Standard Normal Distribution
x = np.linspace(-3, 3, 100)
y = stats.norm.pdf(x, 0, 1)
plt.plot(x, y, label="Standard Normal Distribution")

# Highlight Z-score
plt.axvline(z_score, color='r', linestyle='dashed', label=f'Z-score: {z_score:.2f}')
plt.legend()
plt.title("Z-Test Distribution")
plt.xlabel("Z-score")
plt.ylabel("Probability Density")
plt.show()

```

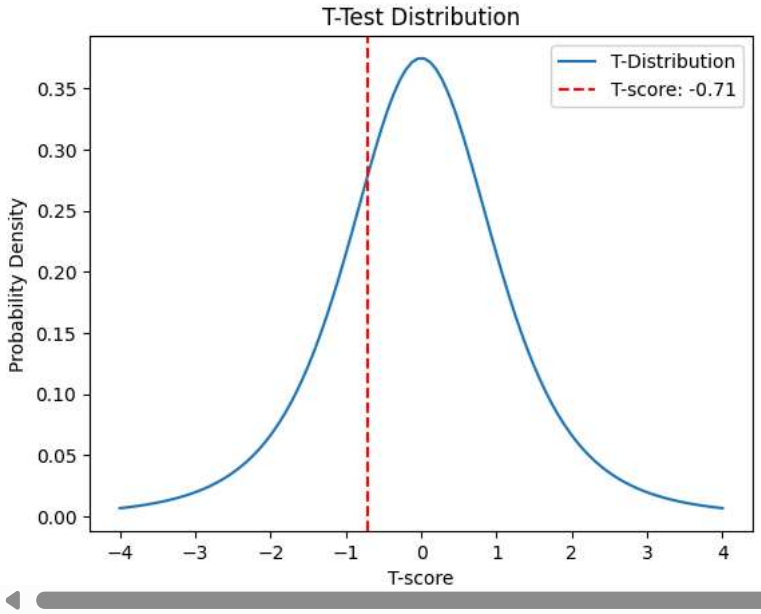
```
Z-Score: -0.56
P-Value: 0.57615
```




```python
#T-test

import seaborn as sns

# Compute T-test
```

```
t_stat, p_value = stats.ttest_1samp(sample, population_mean)

# Print Results
print(f"T-Statistic: {t_stat:.2f}")
print(f"P-Value: {p_value:.5f}")

# Plot T-distribution
x = np.linspace(-4, 4, 100)
y = stats.t.pdf(x, df=sample_size-1)
plt.plot(x, y, label="T-Distribution")

# Highlight T-score
plt.axvline(t_stat, color='r', linestyle='dashed', label=f'T-score: {t_stat:.2f}')
plt.legend()
plt.title("T-Test Distribution")
plt.xlabel("T-score")
plt.ylabel("Probability Density")
plt.show()

```

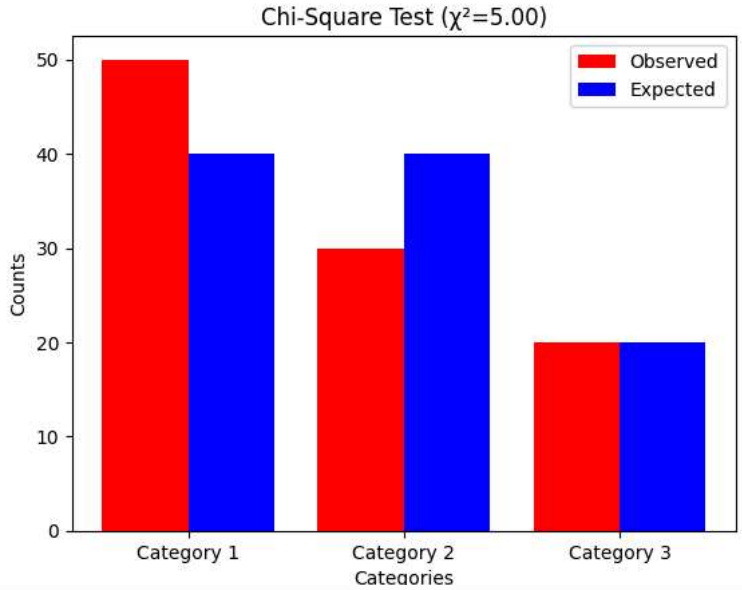
```
T-Statistic: -0.71
P-Value: 0.51852
```




```
#chi-square test

# Observed and Expected Data
observed = np.array([50, 30, 20])
expected = np.array([40, 40, 20])

# Compute Chi-Square
chi2_stat, p_value = stats.chisquare(observed, expected)

# Print Results
print(f"Chi-Square Statistic: {chi2_stat:.2f}")
print(f"P-Value: {p_value:.5f}")

# Bar Plot
labels = ["Category 1", "Category 2", "Category 3"]
x = np.arange(len(labels))

plt.bar(x - 0.2, observed, 0.4, label='Observed', color='red')
plt.bar(x + 0.2, expected, 0.4, label='Expected', color='blue')

plt.xticks(x, labels)
```

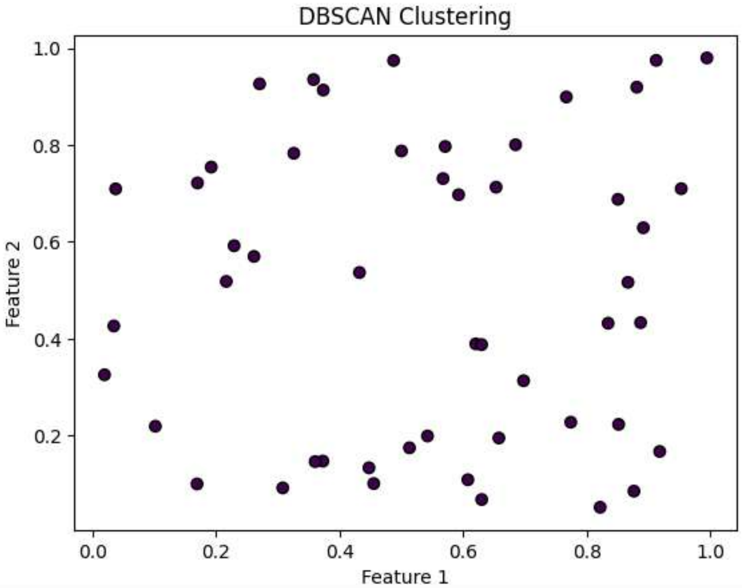
```
plt.xlabel("Categories")
plt.ylabel("Counts")
plt.title(f"Chi-Square Test (χ²={chi2_stat:.2f})")
plt.legend()
plt.show()

```

```
Chi-Square Statistic: 5.00
P-Value: 0.08208
```



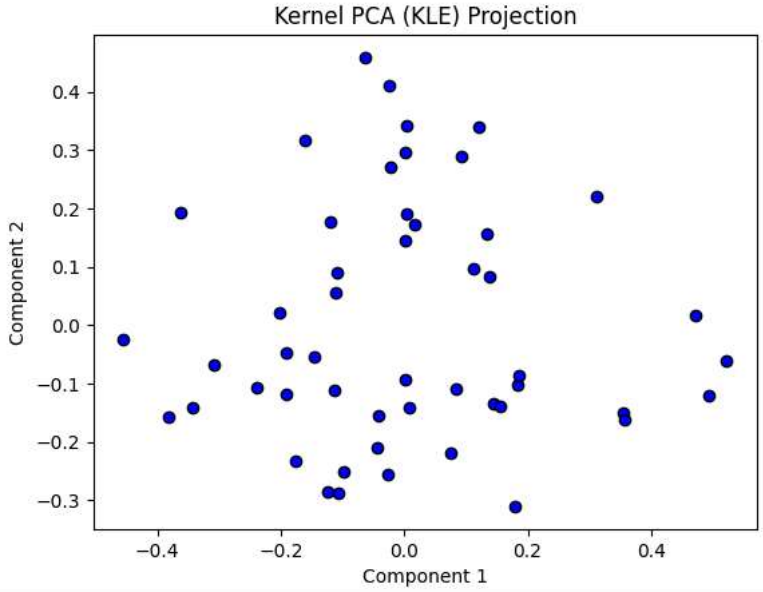

```
#DBSCAN

from sklearn.cluster import DBSCAN

# Generate Sample Data
data = np.random.rand(50, 2)

# Perform DBSCAN Clustering
clustering = DBSCAN(eps=0.2, min_samples=3).fit(data)
labels = clustering.labels_

# Print Cluster Labels
print("DBSCAN Cluster Labels:", labels)

# Scatter Plot
plt.scatter(data[:, 0], data[:, 1], c=labels, cmap='viridis', edgecolors='k')
plt.title("DBSCAN Clustering")
plt.xlabel("Feature 1")
plt.ylabel("Feature 2")
plt.show()

```

```
DBSCAN Cluster Labels: [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0]
```



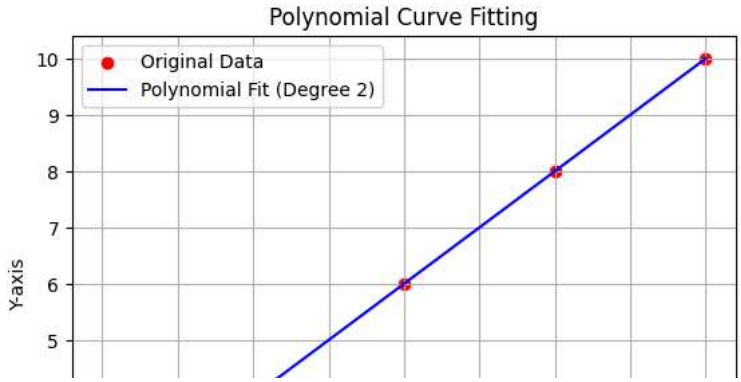

```python
#KLE

from sklearn.decomposition import KernelPCA

# Generate Sample Data
data = np.random.rand(50, 3)  # 3D data

# Apply Kernel PCA
kpca = KernelPCA(n_components=2, kernel='rbf')
transformed_data = kpca.fit_transform(data)

# Print Transformed Data
print("Kernel PCA Transformed Data:\n", transformed_data[:5])  # Display first 5 points

# Scatter Plot
plt.scatter(transformed_data[:, 0], transformed_data[:, 1], c='blue', edgecolors='k')
plt.title("Kernel PCA (KLE) Projection")
plt.xlabel("Component 1")
plt.ylabel("Component 2")
plt.show()
```

```
Kernel PCA Transformed Data:
 [[-0.04196128 -0.15457085]
 [ 0.00416047  0.34132979]
 [-0.36358496  0.19272669]
 [ 0.00381215  0.19088952]
 [-0.20170502  0.02154918]]
```




```python
#POLYNOMIAL CURVE FITTING

from numpy.polynomial.polynomial import Polynomial

# Sample Data
x = np.array([1, 2, 3, 4, 5])
y = np.array([2, 4, 6, 8, 10])

# Fit Polynomial
p = Polynomial.fit(x, y, 2)

# Generate Curve
x_curve = np.linspace(min(x), max(x), 100)
y_curve = p(x_curve)

# Print Polynomial Coefficients
print(f"Polynomial Coefficients: {p.convert().coef}")

# Plot
plt.scatter(x, y, color='red', label='Original Data')
plt.plot(x_curve, y_curve, color='blue', label=f'Polynomial Fit (Degree {p.degree()})')

plt.xlabel('X-axis')
plt.ylabel('Y-axis')
plt.title('Polynomial Curve Fitting')
plt.legend()
plt.grid()
plt.show()

```

```
Polynomial Coefficients: [3.55271368e-15 2.00000000e+00 7.61606712e-16]
```




```
Start coding or generate with AI.
```

Generate print hello world using rot13 Close

```python
#POLYNOMIAL CURVE FITTING

from numpy.polynomial.polynomial import Polynomial

# Sample Data
x = np.array([1, 2, 3, 4, 5])
y = np.array([2, 4, 6, 8, 10])

# Fit Polynomial
p = Polynomial.fit(x, y, 2)

# Generate Curve
x_curve = np.linspace(min(x), max(x), 100)
y_curve = p(x_curve)

# Print Polynomial Coefficients
print(f"Polynomial Coefficients: {p.convert().coef}")

# Plot
plt.scatter(x, y, color='red', label='Original Data')
plt.plot(x_curve, y_curve, color='blue', label=f'Polynomial Fit (Degree {p.degree()})')

plt.xlabel('X-axis')
plt.ylabel('Y-axis')
plt.title('Polynomial Curve Fitting')
plt.legend()
plt.grid()
plt.show()

```

```
Polynomial Coefficients: [3.55271368e-15 2.00000000e+00 7.61606712e-16]
```