correlation

July 2, 2024

1 Correlation Implementations

1.0.1 Imports

[157]:
```
import numpy as np
import statistics
import math
import matplotlib.pyplot as plt
import scipy.stats as sci
import pandas as pd
```

1.0.2 Pearson

Implementation

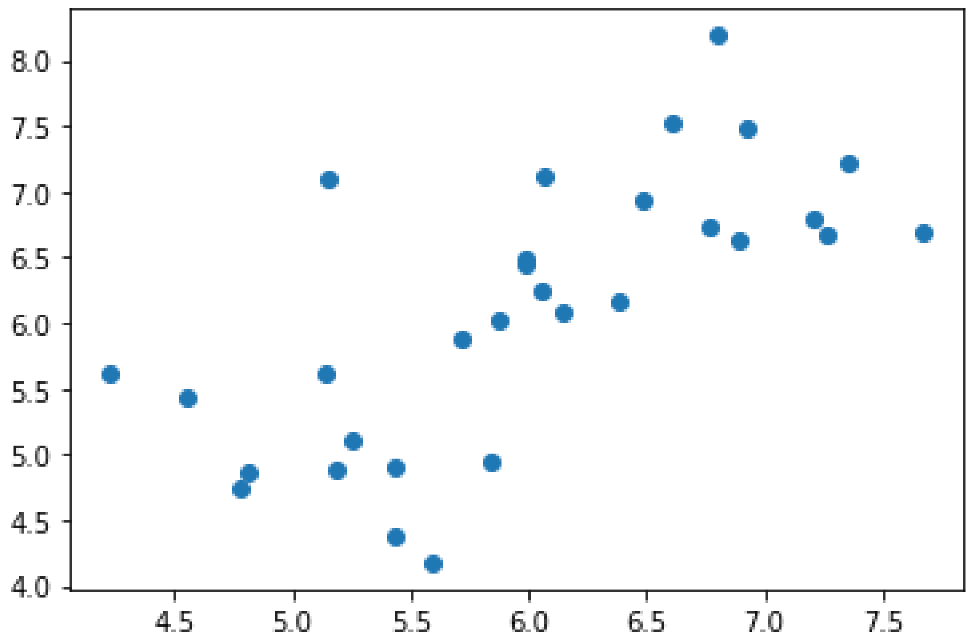
[162]:
```
def pearson(data):
    # Get mean for x and y
    x, y = list(zip(*data))
    x_mean = statistics.mean(x)
    y_mean = statistics.mean(y)

    # Compute numerator and denominator to get r
    numerator = sum([(point[0] - x_mean) * (point[1] - y_mean) for point in␣
 ↪data])
    denominator = math.sqrt(sum([(x[0] - x_mean)**2 for x in data]) *␣
 ↪sum([(y[1] - y_mean)**2 for y in data]))
    r = numerator / denominator

    # Get p-value
    n = len(data)
    t = r * math.sqrt(n - 2) / math.sqrt((1 - r**2))
    p = 2 * (1 - sci.t.cdf(t, n - 2))

    return r, p
```

Generating Test Dataset

[163]:
```
data = []

data.extend([np.random.normal(5, 0.5, 2) for x in range(10)])
data.extend([np.random.normal(6, 0.5, 2) for x in range(10)])
data.extend([np.random.normal(7, 0.5, 2) for x in range(10)])
```

[164]:
```
x_data = [x[0] for x in data]
y_data = [x[1] for x in data]

plt.scatter(x_data, y_data)
```

[164]: <matplotlib.collections.PathCollection at 0x1a1946d2910>



Compare Implementation VS. Library Function

[165]:
```
r, p = pearson(data)
print(f"IMPLEMENTATION\nr: {r}\np-value:{p}")
```

```
IMPLEMENTATION
r: 0.6874296947478136
p-value:2.7085236676382962e-05
```

[166]:
```
x, y = list(zip(*data))
r, p = sci.pearsonr(x, y)
print(f"LIBRARY FUNCTION\nr: {r}\np-value:{p}")
```

```
LIBRARY FUNCTION
r: 0.6874296947478133
p-value:2.7085236676395132e-05
```

1.0.3 Spearman

Implementation

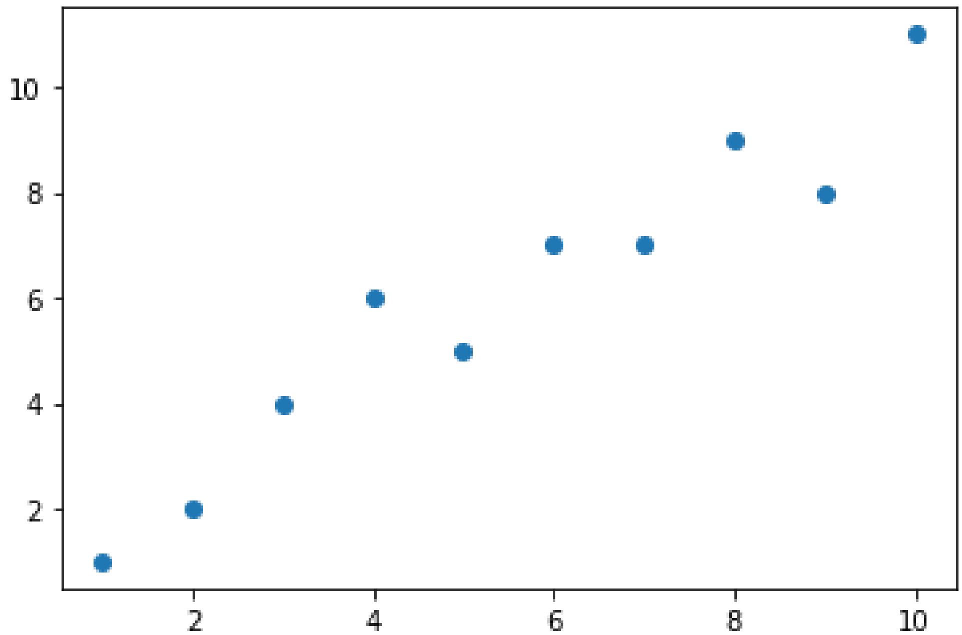
[168]:
```python
def spearman(data):
    # Get ranks for x and y
    x, y = list(zip(*data))
    ranks = []
    ranks.append(sci.rankdata(x))
    ranks.append(sci.rankdata(y))

    # Compute numerator and denominator to get r
    n = len(data)
    numerator = 6 * sum([(ranks[0][i] - ranks[1][i])**2 for i in␣
 ↪range(len(data))])
    denominator = n * (n**2 - 1)
    r = 1 - numerator / denominator

    # Get p-value
    t = r * math.sqrt(n - 2) / math.sqrt((1 - r**2))
    p = 2 * (1 - sci.t.cdf(t, n - 2))

    return r, p
```

Create Test Dataset

[169]:
```python
data = [[1, 1], [2, 2], [3, 4], [4, 6], [5, 5], [6, 7], [7, 7], [8, 9], [9, 8],␣
 ↪[10, 11]]

x_data = [x[0] for x in data]
y_data = [x[1] for x in data]
```

[170]:
```python
plt.scatter(x_data, y_data)
```

[170]:
```
<matplotlib.collections.PathCollection at 0x1a193c23220>
```



Compare Implementation VS. Library Function

[171]:
```
r, p = spearman(data)
print(f"IMPLEMENTATION\nr: {r}\np-value:{p}")
```

```
IMPLEMENTATION
r: 0.9727272727272728
p-value:2.3421115469268727e-06
```

[172]:
```
r, p = sci.spearmanr(data)
print(f"LIBRARY FUNCTION\nr: {r}\np-value:{p}")
```

```
LIBRARY FUNCTION
r: 0.9726488698881034
p-value:2.3689349006384747e-06
```

1.0.4 ANOVA

Implementation

[177]:
```
def anova(data):
    # Compute group means
    group_means = [np.mean(group) for group in data]
    grand_mean = np.mean(group_means)

    # Compute between-group sum of squares (SSB)
```

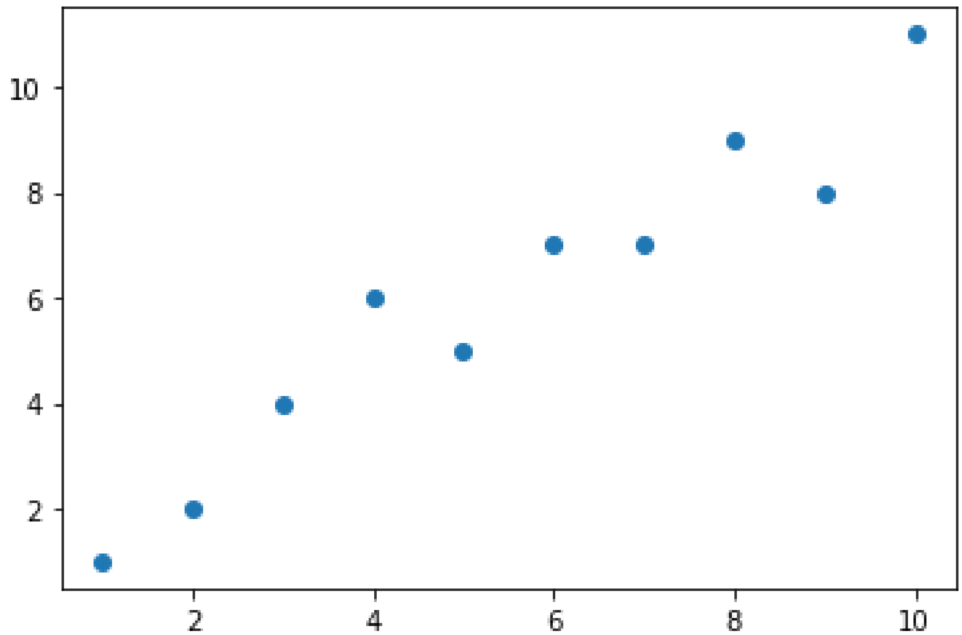
```python
    ssb = sum(len(group) * (mean - grand_mean)**2 for group, mean in zip(data,␣
↪group_means))

    # Compute within-group sum of squares (SSW)
    ssw = sum(sum((x - mean)**2 for x in group) for group, mean in zip(data,␣
↪group_means))

    # Degrees of freedom
    data_between = len(data) - 1
    data_within = sum(len(group) - 1 for group in data)

    # Calculate mean squares
    msb = ssb / data_between
    msw = ssw / data_within

    # Calculate F-statistic
    f = msb / msw

    # Calculate p-value
    p = 1 - sci.f.cdf(f, data_between, data_within)

    return f, p
```

Create Test Dataset

[178]:
```python
data = [[1, 1], [2, 2], [3, 4], [4, 6], [5, 5], [6, 7], [7, 7], [8, 9], [9, 8],␣
↪[10, 11]]
x_data, y_data = list(zip(*data))
```

[179]:
```python
plt.scatter(x_data, y_data)
```

[179]: <matplotlib.collections.PathCollection at 0x1a1940b4190>



Compare Implementation VS. Library Function

[180]:
```
f, p = anova([x_data, y_data])
print(f"IMPLEMENTATION\nf: {f}\np-value:{p}")
```

```
IMPLEMENTATION
f: 0.13353115727002968
p-value:0.7190572800056678
```

[181]:
```
f, p = sci.f_oneway(x_data, y_data)
print(f"LIBRARY FUNCTION\nf: {f}\np-value:{p}")
```

```
LIBRARY FUNCTION
f: 0.13353115727002968
p-value:0.7190572800056663
```

1.0.5 Chi Square

Implementation

[182]:
```
def chi2(data):
    # Get sums of columns and rows
    row_sums = data.sum(axis = 1)
    col_sums = data.sum(axis = 0)
    total = row_sums.sum()

    # Create expected frequency table
```

```
    data_expected = np.outer(row_sums, col_sums) / total

    # Calculate Chi-Square statistic
    x2 = np.sum((data.values - data_expected)**2 / data_expected)

    # Get degrees of freedom
    df = (data.shape[0] - 1) * (data.shape[1] - 1)

    # Get p-value
    p = 1 - sci.chi2.cdf(x2, df)

    return x2, p
```

Create Test Dataset

[183]:
```
data = pd.DataFrame([[35, 15, 50], [10, 30, 60]], ["Female", "Male"],
 ↪["Archery", "Boxing", "Cycling"])
data
```

[183]:

	Archery	Boxing	Cycling
Female	35	15	50
Male	10	30	60

Compare Implementation VS. Library Function

[184]:
```
x2, p = chi2(data)
print(f"IMPLEMENTATION\nx2: {x2}\np-value:{p}")
```

```
IMPLEMENTATION
x2: 19.7979797979798
p-value:5.022538915855357e-05
```

[185]:
```
x2, p, df, expected = sci.chi2_contingency(data.values.tolist())
print(f"LIBRARY FUNCTION\nx2: {x2}\np-value:{p}")
```

```
LIBRARY FUNCTION
x2: 19.7979797979798
p-value:5.022538915853797e-05
```