Data Science I

Overall Points: 97 **/ 100**

Exercise 2: Mathematical Preliminaries

Submission Deadline: November 03 2025, 07:00 UTC
University of Oldenburg
Winter 2025/2026
Instructors: Jannik Schröder, Wolfram "Wolle" Wingerath

Submitted by: Cansu Horata, Charleen Owiti, Suzine Ngiedom

Part 1: Probabilities

25 / 25

1.) Suppose that 80% of people like peanut butter, 89% like jelly, and 78% like both. Given that a randomly sampled person likes peanut butter, what is the probability that she also likes jelly?

$$P(J|PB) = \frac{P(J \cap PB)}{P(PB)} = \frac{0.78}{0.80} = 0.975$$

✓

In [1]:
```
P_PB = 0.80                    #probability of liking Peanut Butter
P_J = 0.89                     #probability of liking jelly
P_PB_and_J = 0.78              #probability of liking both

#Conditional Probability
P_J_given_PB = P_PB_and_J / P_PB
P_J_given_PB
```

Out[1]:
```
0.975
```

2.) Suppose that P(A) = 0.3 and P(B) = 0.7.
(a) Can you compute P(A and B) if you only know P(A) and P(B)?

We don't have any idea whether they are independent. We only know how often each thing happens, not how they affect each other. Maybe you take the bus more often when it rains, or maybe it doesn't matter at all. So we cannot compute them. ✓

(b) Assuming that events A and B arise from independent random processes:
(1) What is P(A and B)?

In [4]:

```python
P_A = 0.3
P_B = 0.7
P_A_and_B = P_A * P_B
P_A_and_B
```

✓

Out[4]:

```
0.21
```

(2) Again assuming that A and B are independent, what is P(A or B)?

$$P(A \cup B) = P(A) + P(B) - P(A \cap B)$$

In [5]:

```python
P_A = 0.3
P_B = 0.7
P_A_and_B = P_A*P_B
P_A_or_B = P_A + P_B - P_A_and_B
print(P_A_or_B)
```

✓

```
0.79
```

(3) Assuming again the independence of A and B, what is P(A|B)?

$$P(A|B) = \frac{P(A \cap B)}{P(B)}$$

In [6]:

```python
P_A = 0.3
P_B = 0.7
P_A_and_B = P_A*P_B
P_A_given_B = P_A*P_B/P_B
print(P_A_given_B)
```

```
0.3
```

3.) Consider a game where your score is the maximum value from two dice throws. Write a small python function that outputs the probability of each event from {1, ..., 6}.

In [1]:

```python
def max_dice_probabilities():
    probs = {}
    for k in range(1, 7):
        probs[k] = (k/6)**2 - ((k-1)/6)**2
    return probs

probs = max_dice_probabilities()
for k in range(1, 7):
    print(f"P(max = {k}) = {probs[k]:.4f}")
```

✓

```
P(max = 1) = 0.0278
P(max = 2) = 0.0833
P(max = 3) = 0.1389
P(max = 4) = 0.1944
P(max = 5) = 0.2500
P(max = 6) = 0.3056
```

4.) If two binary random variables X and Y are independent, is the complement of X and Y also independent? Give a proof or a counterexample.

latex \textbf{Question:} If two binary random variables X and Y are independent, is the complement of X and Y also independent?

\textbf{Answer:} Yes, if X and Y are independent, then their complements are also independent.

\textbf{Proof:}

Let's denote the complements as X^c and Y^c. We need to show that $P(X^c \cap Y^c) = P(X^c) \cdot P(Y^c)$.

First, recall that:

$$X^c = \text{the event that } X \text{ does not occur} \quad (1)$$
$$Y^c = \text{the event that } Y \text{ does not occur} \quad (2)$$

Given that X and Y are independent, we know:

$$P(X \cap Y) = P(X) \cdot P(Y) \quad (3)$$

For binary random variables, we have:

$$P(X^c) = 1 - P(X) \quad (4)$$
$$P(Y^c) = 1 - P(Y) \quad (5)$$

Now, let's compute $P(X^c \cap Y^c)$:

$$\begin{aligned} P(X^c \cap Y^c) &= P((X \cup Y)^c) \quad \text{(by De Morgan's laws)} \\ &= 1 - P(X \cup Y) \\ &= 1 - [P(X) + P(Y) - P(X \cap Y)] \\ &= 1 - [P(X) + P(Y) - P(X) \cdot P(Y)] \quad \text{(using independence of} \\ &= 1 - P(X) - P(Y) + P(X) \cdot P(Y) \\ &= (1 - P(X))(1 - P(Y)) \quad \text{(factoring)} \\ &= P(X^c) \cdot P(Y^c) \end{aligned}$$

✓

Therefore, $P(X^c \cap Y^c) = P(X^c) \cdot P(Y^c)$, which proves that X^c and Y^c are independent.

This result extends to other combinations as well: X and Y^c are independent, and X^c and Y are independent.

Part 2: Statistics

22 / 25

1.) Consider the following pair of distributions:

```
In [ ]: dist1 = [3, 5, 5, 5, 8, 11, 11, 11, 13]
        dist2 = [3, 5, 5, 5, 8, 11, 11, 11, 20]
```

(a) Decide which one has the greater mean and the greater standard deviation without computing either of those.

They are almost the same, except the last number. Probably dist2 has greater mean and the greater standard deviation.

(b) Compute mean and standard deviation of these distributions using built-in functions of the pandas library.

```
In [39…  import pandas as pd

         dist1 = pd.Series([3, 5, 5, 5, 8, 11, 11, 11, 13])
         dist2 = pd.Series([3, 5, 5, 5, 8, 11, 11, 11, 20])

         print("dist1 mean:", dist1.mean(), "std:", dist1.std())
         print("dist2 mean:", dist2.mean(), "std:", dist2.std())
```

✓

```
dist1 mean: 8.0 std: 3.605551275463989
dist2 mean: 8.777777777777779 std: 5.214829282387338
```

(c) Now define a function yourself to compute mean and standard deviation. Verify that the functions work correctly by comparing the output with the output of the built-in functions.

here you should do python code

$$\textbf{Given data:} \tag{13}$$
$$\text{dist1} = [3, 5, 5, 5, 8, 11, 11, 11, 13] \tag{14}$$
$$\text{dist2} = [3, 5, 5, 5, 8, 11, 11, 11, 20] \tag{15}$$

Calculate Mean calculation:

$$\text{mean(data)} = \frac{\sum_{i=1}^{n} x_i}{n} \tag{16}$$

Calculate Standard deviation:

$$\text{std(data)} = \sqrt{\frac{\sum_{i=1}^{n} (x_i - \text{mean})^2}{n-1}} \tag{17}$$

Results:

$$\begin{aligned}
\text{dist1 mean} &= \frac{3+5+5+5+8+11+11+11+13}{9} \\
&= \frac{72}{9} \\
&= 8.0
\end{aligned}$$

$$\begin{aligned}
\text{dist1 std} &= \sqrt{\frac{(3-8)^2+(5-8)^2+(5-8)^2+(5-8)^2+(8-8)^2+(11 -}{9-1}} \\
&= \sqrt{\frac{25+9+9+9+0+9+9+9+25}{8}} \\
&= \sqrt{\frac{104}{8}} \\
&= \sqrt{13} \\
&\approx 3.606
\end{aligned}$$

$$\begin{aligned}
\text{dist2 mean} &= \frac{3+5+5+5+8+11+11+11+20}{9} \\
&= \frac{79}{9} \\
&\approx 8.778
\end{aligned}$$

$$\begin{aligned}
\text{dist2 std} &= \sqrt{\frac{(3-8.778)^2+(5-8.778)^2+(5-8.778)^2+(5-8.778)^2+}{}} \\
&= \sqrt{\frac{33.395+14.284+14.284+14.284+0.605+4.926+4.926}{8}} \\
&= \sqrt{\frac{217.469}{8}} \\
&= \sqrt{27.184} \\
&\approx 5.214
\end{aligned}$$

2.) Consider the following distribution:
Distribution 1: [1, 1, 1, 1, 1, 1, 1, 1, 1]
How do the arithmetic and geometric mean compare?

Arithmetic Mean:

$$\begin{aligned}
\text{AM} &= \frac{\sum_{i=1}^{n} x_i}{n} && (37) \\
&= \frac{1+1+1+1+1+1+1+1+1}{9} && (38) \\
&= \frac{9}{9} && (39) \\
&= 1 && (40)
\end{aligned}$$

✓

Geometric Mean:

$$\begin{aligned}
\mathrm{GM} &= \left(\prod_{i=1}^{n} x_i\right)^{1/n} && (41)\\
&= (1 \times 1 \times 1 \times 1 \times 1 \times 1 \times 1 \times 1 \times 1)^{1/9} && (42)\\
&= (1)^{1/9} && (43)\\
&= 1 && (44)
\end{aligned}$$

Comparison: For this distribution, the arithmetic mean and geometric mean are equal:

$$\mathrm{AM} = \mathrm{GM} = 1 \qquad (45)$$

This is a special case where all values in the distribution are identical (all 1's). When all values in a distribution are equal, the arithmetic mean and geometric mean will always be equal to that value.

3.) How do the arithmetic and geometric mean compare on random integers that are not identical? Draw a sample distribution of size 10 a few times and write down your findings.

Note:
Please implement your own version of the geometric mean and use it in this task.

The arithmetic mean (average) is the sum of all values divided by their count, while the geometric mean is the nth root of their product. When numbers are not identical, the geometric mean is always less than or equal to the arithmetic mean.

In [2]:
```python
import numpy as np
import random

def arithmetic_mean(data):
    """Calculate the arithmetic mean of a list of numbers"""
    return sum(data) / len(data)

def geometric_mean(data):
    """Calculate the geometric mean of a list of numbers"""
    product = 1
    for value in data:
        product *= value
    return product ** (1/len(data))

# Set a random seed for reproducibility
np.random.seed(42)

# Function to generate a sample and compare means
def compare_means(sample_size=10, min_val=1, max_val=20, num_trials=5):
    print("Comparing Arithmetic Mean (AM) and Geometric Mean (GM) for rand
    print("-" * 80)
    for trial in range(num_trials):
        # Generate random integers
        sample = [random.randint(min_val, max_val) for _ in range(sample_si

        # Calculate means
        am = arithmetic_mean(sample)
        gm = geometric_mean(sample)

        # Print results
        print(f"Trial {trial+1}:")
        print(f"Sample: {sample}")
        print(f"Arithmetic Mean: {am:.4f}")
        print(f"Geometric Mean: {gm:.4f}")
        print(f"Difference (AM - GM): {am - gm:.4f}")
        print(f"Ratio (AM/GM): {am/gm:.4f}")
        print("-" * 80)

    # Summary of findings
    print("\nFindings:")
    print("1. The arithmetic mean is always greater than or equal to the ge
    print("2. The difference between AM and GM increases with greater varia
    print("3. When all values are identical, AM equals GM (as shown in the
    print("4. The AM-GM inequality is a well-known mathematical principle:
    print("   equality if and only if all values in the distribution are ec

# Run the comparison
compare_means(sample_size=10, min_val=1, max_val=20, num_trials=5)
```

```
Comparing Arithmetic Mean (AM) and Geometric Mean (GM) for random integer d
istributions:
---------------------------------------------------------------------------
-----
Trial 1:
Sample: [8, 12, 20, 9, 12, 16, 17, 7, 19, 11]
Arithmetic Mean: 13.1000
Geometric Mean: 12.3496
Difference (AM - GM): 0.7504
Ratio (AM/GM): 1.0608
---------------------------------------------------------------------------
-----
Trial 2:
Sample: [13, 8, 9, 8, 10, 2, 5, 19, 7, 2]
Arithmetic Mean: 8.3000
Geometric Mean: 6.7597
Difference (AM - GM): 1.5403
Ratio (AM/GM): 1.2279
---------------------------------------------------------------------------
-----
Trial 3:
Sample: [19, 18, 6, 7, 2, 6, 15, 1, 15, 4]
Arithmetic Mean: 9.3000
Geometric Mean: 6.5928
Difference (AM - GM): 2.7072
Ratio (AM/GM): 1.4106
---------------------------------------------------------------------------
-----
Trial 4:
Sample: [19, 16, 13, 1, 12, 17, 6, 19, 18, 7]
Arithmetic Mean: 12.8000
Geometric Mean: 10.1478
Difference (AM - GM): 2.6522
Ratio (AM/GM): 1.2614
---------------------------------------------------------------------------
-----
Trial 5:
Sample: [8, 5, 3, 6, 17, 17, 14, 11, 5, 6]
Arithmetic Mean: 9.2000
Geometric Mean: 7.9120
Difference (AM - GM): 1.2880
Ratio (AM/GM): 1.1628
---------------------------------------------------------------------------
-----

Findings:
1. The arithmetic mean is always greater than or equal to the geometric mea
n.
2. The difference between AM and GM increases with greater variance in the
data.
3. When all values are identical, AM equals GM (as shown in the previous qu
estion).
4. The AM-GM inequality is a well-known mathematical principle: AM ≥ GM wit
h
   equality if and only if all values in the distribution are equal.
```
✓

Part 3: Correlation Analysis

/ 25

1.) A correlation coefficient of –0.9 indicates a stronger linear relationship than a correlation coefficient of 0.5 – true or false? Explain why.

True. Even though –0.9 is negative, its absolute value is larger, so the linear relationship is stronger than in the 0.5 case just in the opposite direction.

2.) Compute the Pearson and Spearman Rank correlations for uniformly drawn samples of points (x, x^k) and answer the questions below.

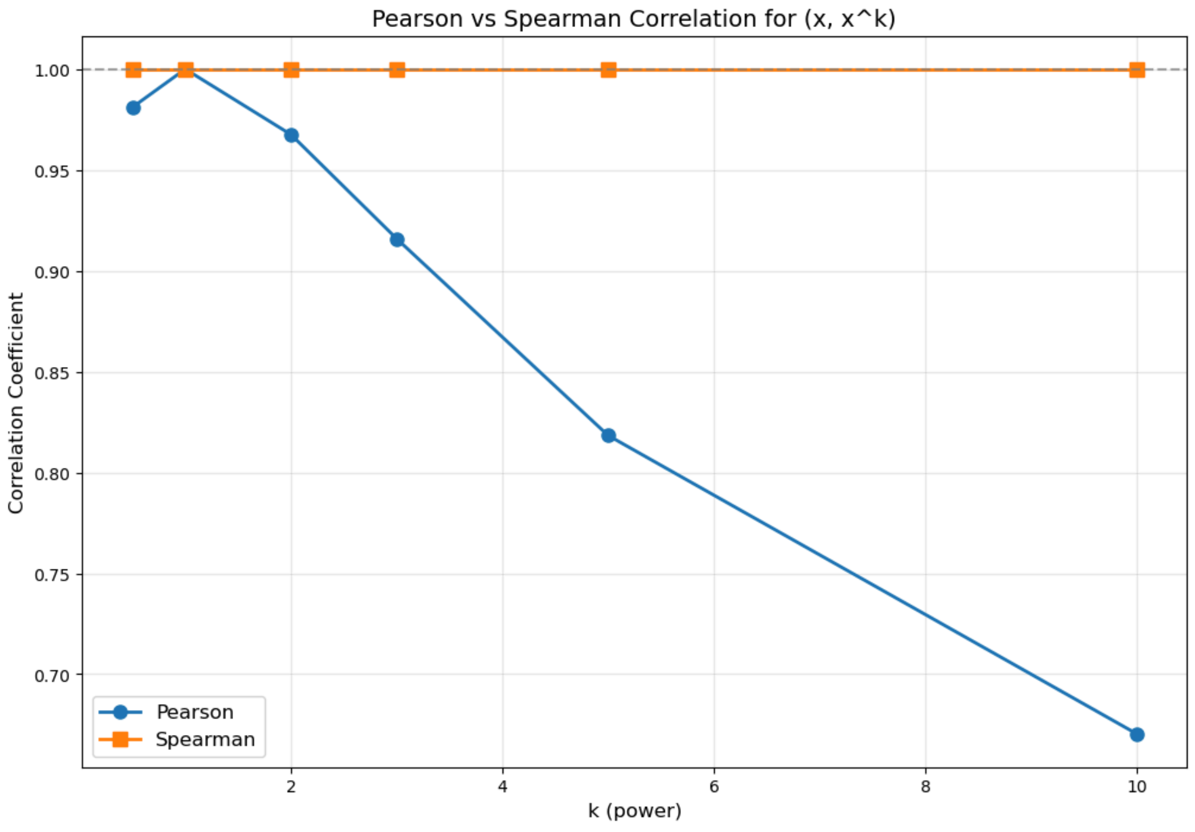
Note:
You can use scipy.stats.spearmanr and scipy.stats.pearsonr to compute the ranks.

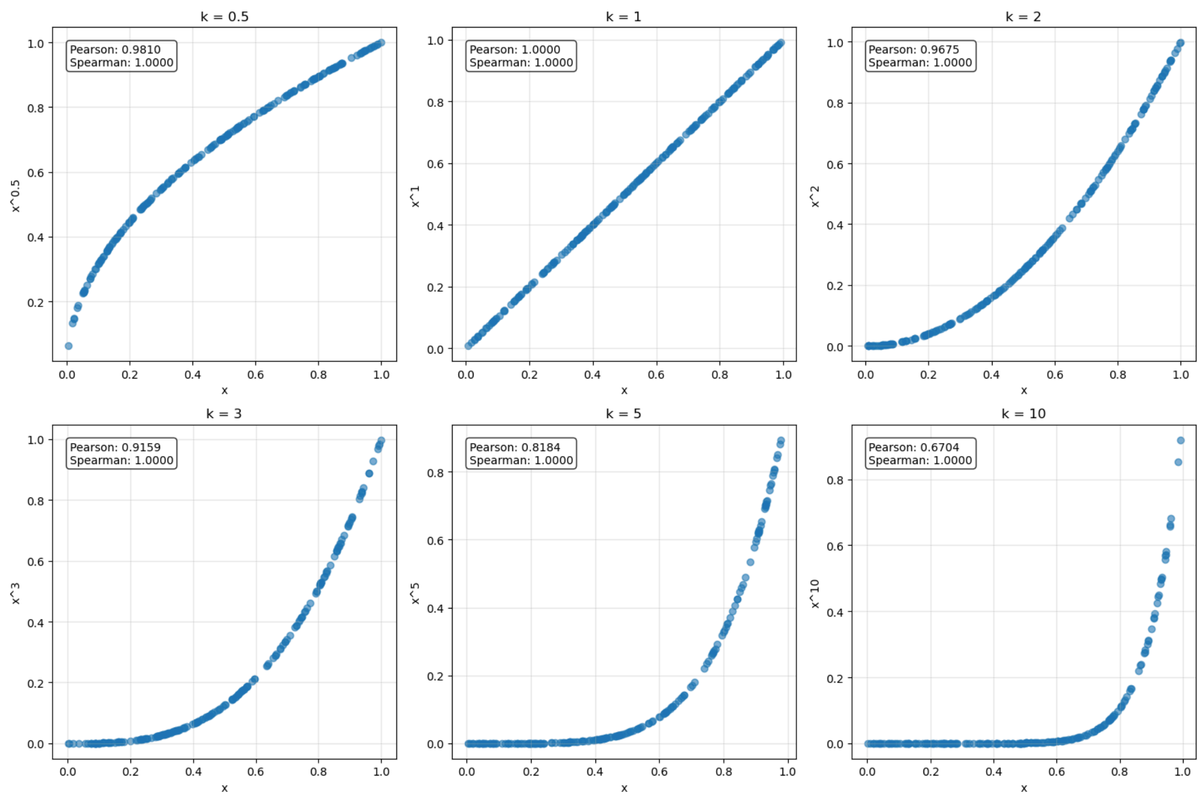
Pearson correlation measures how strongly two variables are related in a straight line — it checks if one increases proportionally when the other increases. Spearman correlation, on the other hand, measures how well the order or ranking between two variables is preserved, even if the relationship is not perfectly linear. In short, Pearson looks for a linear relationship, while Spearman looks for a monotonic (order-based) relationship.

In [3]:
```python
import numpy as np
import matplotlib.pyplot as plt
from scipy.stats import pearsonr, spearmanr

# Set random seed for reproducibility
np.random.seed(42)

# Function to generate data and compute correlations
def compute_correlations(k_values, sample_size=1000):
    results = []

    for k in k_values:
        # Generate uniform random samples for x between 0 and 1
        x = np.random.uniform(0, 1, sample_size)

        # Compute y = x^k
        y = x**k

        # Calculate Pearson correlation
        pearson_corr, pearson_p = pearsonr(x, y)

        # Calculate Spearman rank correlation
        spearman_corr, spearman_p = spearmanr(x, y)

        results.append({
            'k': k,
            'pearson': pearson_corr,
            'spearman': spearman_corr
        })

    return results

# Define k values to test
k_values = [0.5, 1, 2, 3, 5, 10]

# Compute correlations
results = compute_correlations(k_values)

# Display results in a table
print("Correlation coefficients for (x, x^k):")
print("-" * 60)
print(f"{'k':^10} | {'Pearson r':^15} | {'Spearman r':^15}")
print("-" * 60)
for r in results:
    print(f"{r['k']:^10} | {r['pearson']:^15.6f} | {r['spearman']:^15.6f}"
print("-" * 60)

# Create visualization
plt.figure(figsize=(12, 8))

# Plot correlation values
k_values_arr = np.array([r['k'] for r in results])
pearson_values = np.array([r['pearson'] for r in results])
spearman_values = np.array([r['spearman'] for r in results])

plt.plot(k_values_arr, pearson_values, 'o-', label='Pearson', linewidth=2,
plt.plot(k_values_arr, spearman_values, 's-', label='Spearman', linewidth=
plt.axhline(y=1.0, color='gray', linestyle='--', alpha=0.7)

# Add labels and title
plt.xlabel('k (power)', fontsize=12)
plt.ylabel('Correlation Coefficient', fontsize=12)
plt.title('Pearson vs Spearman Correlation for (x, x^k)', fontsize=14)
plt.grid(True, alpha=0.3)
plt.legend(fontsize=12)

# Create subplots to show example scatter plots
fig, axs = plt.subplots(2, 3, figsize=(15, 10))
axs = axs.flatten()

for i, k in enumerate(k_values):
    # Generate data for visualization
    x = np.random.uniform(0, 1, 200)
    y = x**k

    # Plot scatter
    axs[i].scatter(x, y, alpha=0.6)
    axs[i].set_title(f'k = {k}')
    axs[i].set_xlabel('x')
    axs[i].set_ylabel(f'x^{k}')
    axs[i].grid(alpha=0.3)

    # Add correlation values as text
    pearson = results[i]['pearson']
    spearman = results[i]['spearman']
    axs[i].text(0.05, 0.95, f'Pearson: {pearson:.4f}\nSpearman: {spearman:
                transform=axs[i].transAxes, verticalalignment='top',
                bbox=dict(boxstyle='round', facecolor='white', alpha=0.8))

plt.tight_layout()
plt.show()

# Analysis of results
print("\nAnalysis of Results:")
print("1. For k = 1, both correlations are exactly 1.0 because y = x (perfe
print("2. For k ≠ 1, Pearson correlation decreases as the relationship beco
print("3. Spearman correlation remains 1.0 for all positive k values becau
print("4. This demonstrates that Spearman is invariant to monotonic transfo
print("5. Pearson measures linear association, while Spearman measures mono
```

```
Correlation coefficients for (x, x^k):
------------------------------------------------------------
    k      |    Pearson r    |   Spearman r
------------------------------------------------------------
   0.5     |    0.981048     |    1.000000
    1      |    1.000000     |    1.000000
    2      |    0.967534     |    1.000000
    3      |    0.915905     |    1.000000
    5      |    0.818353     |    1.000000
    10     |    0.670430     |    1.000000
------------------------------------------------------------
```

✓





```
Analysis of Results:
1. For k = 1, both correlations are exactly 1.0 because y = x (perfect line
ar relationship).
2. For k ≠ 1, Pearson correlation decreases as the relationship becomes mor
e nonlinear.
3. Spearman correlation remains 1.0 for all positive k values because the r
ank order is preserved.
4. This demonstrates that Spearman is invariant to monotonic transformation
s.
5. Pearson measures linear association, while Spearman measures monotonic a
ssociation.
```

(a) How do these values change as a function of increasing k?

As k increases, the Pearson correlation decreases because the relationship between x and x^k becomes less linear. However, the Spearman correlation remains close to 1, since x^k is still a monotonic increasing function — the rank order of x and y does not change.

(b) Does it matter whether x can be positive and negative or positive only?

Yes it matters because when:

x is only positive

if x is only positive, then for any positive values of k, $y = x^k$, will increase when x increases. Example. given x is 1,2,3,4, and k is 2, then $y = x^k$ will be 1, 4, 9, 16 respectively and when k is 3, then $y = x^k$ will be 1, 8, 27, 64 respectively. This shows, regardless of whether k is is an even or odd number, when x is positive, the relationship is monotonic increasing.

x is positive and negative with even power

If x includes both positive and negative values, then, for even powers of k (like k = 2, 4, 6), the function $y = x^k$ becomes non monotonic(U shaped).For instance, if x is -3,-2,-1,0,1,2,3 and $y=x^2, the y becomes 9,4,1,0,1,4,9 respectively. That means, as x increases from -3 to -1, y decreases from 9 to 1 and as x increases from 1 to 3, y increases from 1 to 9).

x is positive and negative with odd power

If x includes both positive and negative values like -3,-2,-1,0,1,2,3 and for odd powers of k (like k = 3, 7, 9), in this case we use 3 The function $y = x^k$ yields:

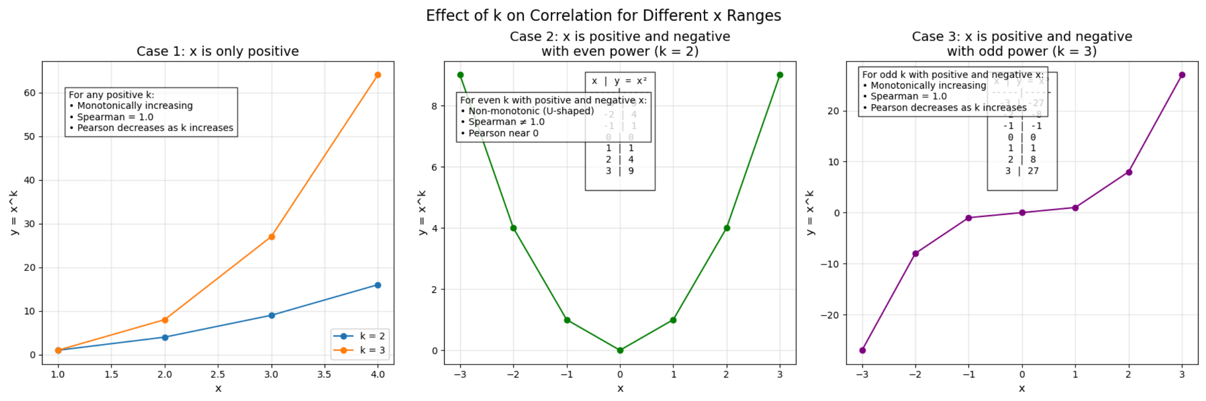
x	$y = x^3$
-3	-27
-2	-8
-1	-1
0	0
1	1
2	8
3	27

We Observe The function $y = x^k$ is strictly monotonically increasing. The code below shows the shape of the above three relationships

In [5]:
```python
import numpy as np
import matplotlib.pyplot as plt
from scipy.stats import pearsonr, spearmanr

# Set up the figure with 3 subplots
fig, axs = plt.subplots(1, 3, figsize=(18, 6))

# Case 1: x is only positive
x_positive = np.array([1, 2, 3, 4])
k_values = [2, 3]

# Plot for positive x values
axs[0].set_title('Case 1: x is only positive', fontsize=14)
axs[0].set_xlabel('x', fontsize=12)
axs[0].set_ylabel('y = x^k', fontsize=12)

for k in k_values:
    y = x_positive ** k
    axs[0].plot(x_positive, y, 'o-', label=f'k = {k}')
    # Calculate correlations
    pearson_corr, _ = pearsonr(x_positive, y)
    spearman_corr, _ = spearmanr(x_positive, y)

    print(f"Positive x only, k={k}: Pearson={pearson_corr:.4f}, Spearman={s

axs[0].grid(True, alpha=0.3)
axs[0].legend()
axs[0].text(1.1, 0.8*max(x_positive**max(k_values)),
           "For any positive k:\n• Monotonically increasing\n• Spearman = 1
           bbox=dict(facecolor='white', alpha=0.8))

# Case 2: x is positive and negative with even power
x_both = np.array([-3, -2, -1, 0, 1, 2, 3])
k_even = 2

# Plot for even power
y_even = x_both ** k_even
axs[1].set_title(f'Case 2: x is positive and negative\nwith even power (k =
axs[1].set_xlabel('x', fontsize=12)
axs[1].set_ylabel('y = x^k', fontsize=12)
axs[1].plot(x_both, y_even, 'o-', color='green')

# Calculate correlations
pearson_corr, _ = pearsonr(x_both, y_even)
spearman_corr, _ = spearmanr(x_both, y_even)

print(f"Positive and negative x, k={k_even}: Pearson={pearson_corr:.4f}, Sp

# Add a table to show values
table_text = "x | y = x²\n-----|-----\n"
for i in range(len(x_both)):
    table_text += f"{x_both[i]} | {y_even[i]}\n"

axs[1].text(0.5, 0.95, table_text, transform=axs[1].transAxes,
           verticalalignment='top', horizontalalignment='center',
           fontfamily='monospace', bbox=dict(facecolor='white', alpha=0.8)

axs[1].grid(True, alpha=0.3)
axs[1].text(-3, 7,
           "For even k with positive and negative x:\n• Non-monotonic (U-sh
           bbox=dict(facecolor='white', alpha=0.8))

# Case 3: x is positive and negative with odd power
k_odd = 3
y_odd = x_both ** k_odd

# Plot for odd power
axs[2].set_title(f'Case 3: x is positive and negative\nwith odd power (k =
axs[2].set_xlabel('x', fontsize=12)
axs[2].set_ylabel('y = x^k', fontsize=12)
axs[2].plot(x_both, y_odd, 'o-', color='purple')

# Calculate correlations
pearson_corr, _ = pearsonr(x_both, y_odd)
```

```
spearman_corr, _ = spearmanr(x_both, y_odd)

print(f"Positive and negative x, k={k_odd}: Pearson={pearson_corr:.4f}, Spe

# Add a table to show values
table_text = "x | y = x³\n-----|-----\n"
for i in range(len(x_both)):
    table_text += f"{x_both[i]} | {y_odd[i]}\n"

axs[2].text(0.5, 0.95, table_text, transform=axs[2].transAxes,
           verticalalignment='top', horizontalalignment='center',
           fontfamily='monospace', bbox=dict(facecolor='white', alpha=0.8)

axs[2].grid(True, alpha=0.3)
axs[2].text(-3, 20,
           "For odd k with positive and negative x:\n• Monotonically increa
           bbox=dict(facecolor='white', alpha=0.8))

# Add a main title
plt.suptitle('Effect of k on Correlation for Different x Ranges', fontsize

# Adjust layout
plt.tight_layout()
plt.subplots_adjust(top=0.85)

# Save the figure
plt.savefig('correlation_cases.png', dpi=300, bbox_inches='tight')
plt.show()

# Create a summary table of correlation values
print("\nSummary of Correlation Values:")
print("-" * 60)
print(f"{'Case':<30} | {'Pearson':<10} | {'Spearman':<10}")
print("-" * 60)

# Case 1: Positive x only
for k in k_values:
    y = x_positive ** k
    pearson_corr, _ = pearsonr(x_positive, y)
    spearman_corr, _ = spearmanr(x_positive, y)
    print(f"Positive x only, k={k:<20} | {pearson_corr:.4f}    | {spearman_

# Case 2: Positive and negative x, even k
pearson_corr, _ = pearsonr(x_both, y_even)
spearman_corr, _ = spearmanr(x_both, y_even)
print(f"Pos & neg x, even k={k_even:<16} | {pearson_corr:.4f}    | {spearma

# Case 3: Positive and negative x, odd k
pearson_corr, _ = pearsonr(x_both, y_odd)
spearman_corr, _ = spearmanr(x_both, y_odd)
print(f"Pos & neg x, odd k={k_odd:<17} | {pearson_corr:.4f}    | {spearman_
print("-" * 60)
```

```
Positive x only, k=2: Pearson=0.9844, Spearman=1.0000
Positive x only, k=3: Pearson=0.9514, Spearman=1.0000
Positive and negative x, k=2: Pearson=0.0000, Spearman=0.0000
Positive and negative x, k=3: Pearson=0.9295, Spearman=1.0000
```



```
Summary of Correlation Values:
------------------------------------------------------------
Case                           | Pearson    | Spearman
------------------------------------------------------------
Positive x only, k=2                    | 0.9844     | 1.0000
Positive x only, k=3                    | 0.9514     | 1.0000
Pos & neg x, even k=2                | 0.0000     | 0.0000
Pos & neg x, odd k=3                 | 0.9295     | 1.0000
------------------------------------------------------------
```

Part 4: Logarithms

/ 25

Recall the definition of the geometric mean:

$$\left(\prod_{i=1}^{n} a_i\right)^{1/n} = \sqrt[n]{a_1 \cdot a_2 \cdots a_n}$$

For large sample sets or small sample sets with very large numbers, the computed product will become very large. Chances are that your implementation of the geometric mean does not work with large sample sizes, either – go ahead and try it with a sample size of, let's say, 1000!

Assuming that your implementation does not work for a sample size of 1000, create one that does!

When we calculate the geometric mean, we multiply many numbers together. If there are many numbers, the product can get too big for the computer. Then how do we get the final result? 1)take the log of every number. 2)Find the average of all these log. 3)Take exponential() which is the reverse of log. That gives you the same result as multiplying and taking the root — but safer.

In [1]:
```
import numpy as np
```

```
def geometric_mean(x):
    x = np.array(x, dtype=float)
    if np.any(x <= 0):
        raise ValueError("All numbers must be positive for geometric mean.'
    return np.exp(np.mean(np.log(x)))

data = np.abs(np.random.rand(1000) * 10) + 1e-8
result = geometric_mean(data)
print("Geometric mean for 1000 numbers:", result)
```

```
Geometric mean for 1000 numbers: 3.599238885114391
```

Finally: Submission

Save your notebook and submit it (as both **notebook and PDF file**). And please don't forget to ...

- ... choose a **file name** according to convention (see Exercise Sheet 1) and to
- ... include the **execution output** in your submission!