Data Science I

Overall Points: 95 / 100

Exercise 3: Data Munging, Data Cleaning, Rankings & Scores

Submission Deadline: November 17 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: Data Munging & Data Cleaning 40 / 40

1.) Find a source for energy prices over time (e.g. electric energy).

a) How would you assess the quality of the data set you found? Do you need to do any preparation before doing the analysis? (If so, what exactly did you do?)

I obtained my dataset from IEA(International Energy Agency), which is a reliable official source. However, I still performed several cleaning steps, such as removing empty spaces, unnecessary characters, and formatting marks to prepare the data for analysis.

quite short. Maybe describe your steps more detailed and with examples

b) Analyze this data and make a projection about energy costs five years from now.

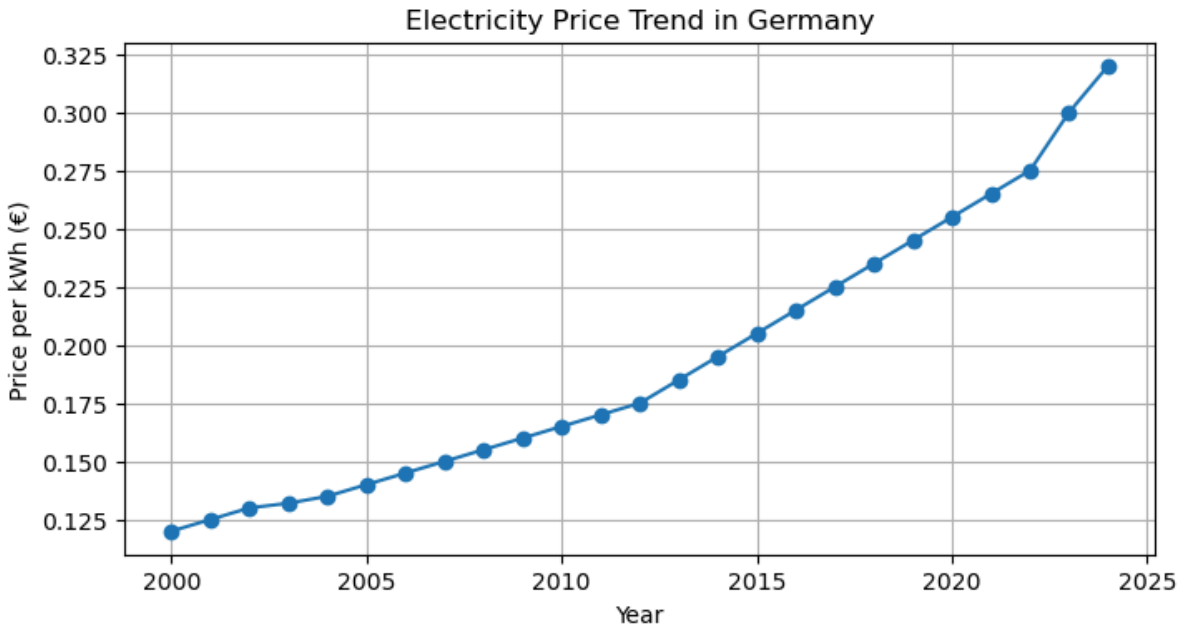
In [9]:
```
import pandas as pd
df = pd.read_csv("germany_electricity_sample.csv")
df
df.rename(columns={"Electricity_Price_EUR_per_kWh": "Price_kWh"}, inplace=
df
```

Out[9]:

	Year	Price_kWh
0	2000	0.120
1	2001	0.125
2	2002	0.130
3	2003	0.132
4	2004	0.135
5	2005	0.140
6	2006	0.145
7	2007	0.150
8	2008	0.155
9	2009	0.160
10	2010	0.165
11	2011	0.170
12	2012	0.175
13	2013	0.185
14	2014	0.195
15	2015	0.205
16	2016	0.215
17	2017	0.225
18	2018	0.235
19	2019	0.245
20	2020	0.255
21	2021	0.265
22	2022	0.275
23	2023	0.300
24	2024	0.320

In [21…

```
import matplotlib.pyplot as plt
plt.figure(figsize=(8,4))
plt.plot(df["Year"], df["Price_kWh"], marker="o")
plt.xlabel("Year")
plt.ylabel("Price per kWh (€)")
plt.title("Electricity Price Trend in Germany")
plt.grid()
plt.show()
```



In [23…
```
df.describe()
```

Out[23]:

	Year	Price_kWh
count	25.000000	25.00000
mean	2012.000000	0.19288
std	7.359801	0.05833
min	2000.000000	0.12000
25%	2006.000000	0.14500
50%	2012.000000	0.17500
75%	2018.000000	0.23500
max	2024.000000	0.32000

In [27…
```
import numpy as np
from sklearn.linear_model import LinearRegression

X = df["Year"].values.reshape(-1, 1)      # independent variable
y = df["Price_kWh"].values                # dependent variable
```

In [28…
```
model = LinearRegression()
model.fit(X, y)
```

Out[28]:
```
▾ LinearRegression
LinearRegression()
```

In [29…
```
last_year = df["Year"].max()
future_years = np.arange(last_year + 1, last_year + 6).reshape(-1, 1)
```

```
future_years
```

Out[29]:
```
array([[2025],
       [2026],
       [2027],
       [2028],
       [2029]])
```

In [30…
```
future_prices = model.predict(future_years)
future_prices
```

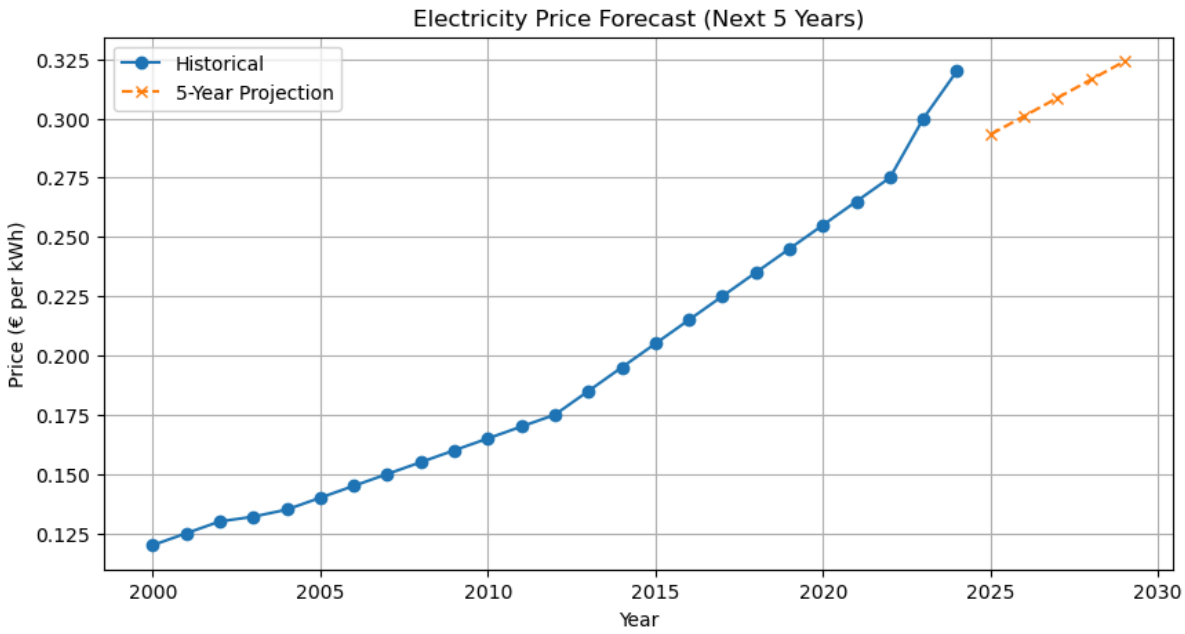
Out[30]:
```
array([0.2932    , 0.30091692, 0.30863385, 0.31635077, 0.32406769])
```

In [31…
```
forecast_df = pd.DataFrame({
    "Year": future_years.flatten(),
    "Predicted_Price_kWh": future_prices
})

forecast_df
```

Out[31]:

	Year	Predicted_Price_kWh
0	2025	0.293200
1	2026	0.300917
2	2027	0.308634
3	2028	0.316351
4	2029	0.324068

In [32…
```
import matplotlib.pyplot as plt

plt.figure(figsize=(10,5))

# historical
plt.plot(df["Year"], df["Price_kWh"], marker="o", label="Historical")

# predicted
plt.plot(forecast_df["Year"], forecast_df["Predicted_Price_kWh"],
         marker="x", linestyle="--", label="5-Year Projection")

plt.xlabel("Year")
plt.ylabel("Price (€ per kWh)")
plt.title("Electricity Price Forecast (Next 5 Years)")
plt.grid(True)
plt.legend()
plt.show()
```



c) What will energy prices be in 25 or 50 years?

In [34…
```
last_year = df["Year"].max()
future_years = np.arange(last_year + 1, last_year + 26).reshape(-1, 1)
future_years
```

Out[34]:
```
array([[2025],
       [2026],
       [2027],
       [2028],
       [2029],
       [2030],
       [2031],
       [2032],
       [2033],
       [2034],
       [2035],
       [2036],
       [2037],
       [2038],
       [2039],
       [2040],
       [2041],
       [2042],
       [2043],
       [2044],
       [2045],
       [2046],
       [2047],
       [2048],
       [2049]])
```

In [35…
```
future_prices = model.predict(future_years)
future_prices
```

Out[35]:
```
array([0.2932    , 0.30091692, 0.30863385, 0.31635077, 0.32406769,
       0.33178462, 0.33950154, 0.34721846, 0.35493538, 0.36265231,
       0.37036923, 0.37808615, 0.38580308, 0.39352   , 0.40123692,
       0.40895385, 0.41667077, 0.42438769, 0.43210462, 0.43982154,
       0.44753846, 0.45525538, 0.46297231, 0.47068923, 0.47840615])
```

In [36…
```
forecast_df = pd.DataFrame({
    "Year": future_years.flatten(),
    "Predicted_Price_kWh": future_prices
})

forecast_df
```

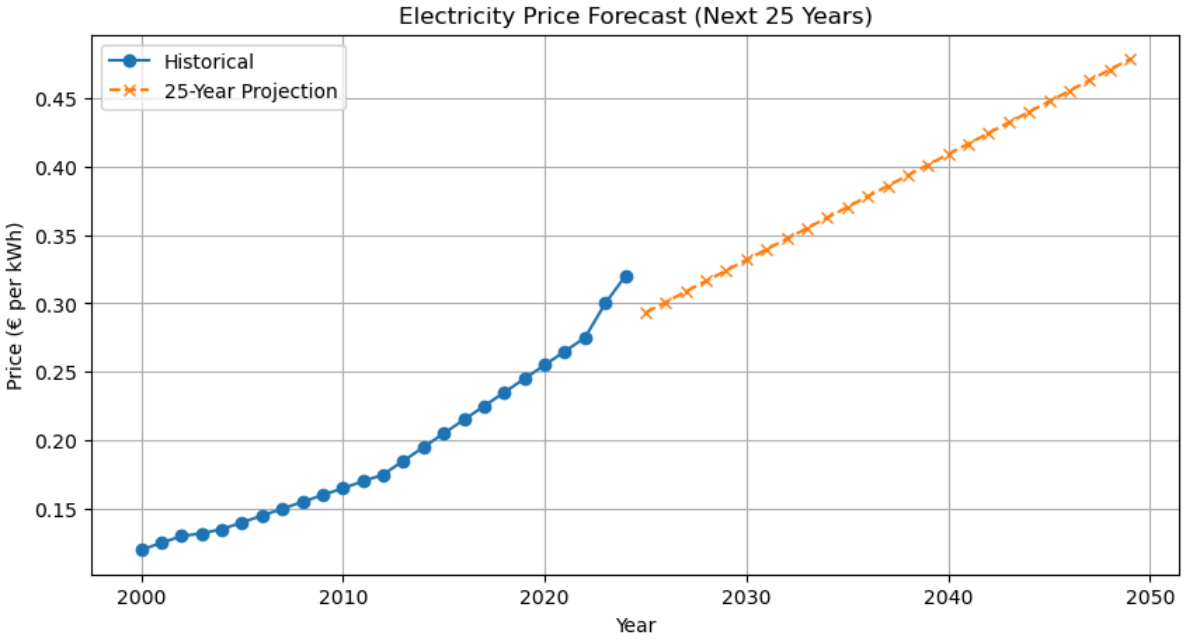
Out[36]:

	Year	Predicted_Price_kWh
0	2025	0.293200
1	2026	0.300917
2	2027	0.308634
3	2028	0.316351
4	2029	0.324068
5	2030	0.331785
6	2031	0.339502
7	2032	0.347218
8	2033	0.354935
9	2034	0.362652
10	2035	0.370369
11	2036	0.378086
12	2037	0.385803
13	2038	0.393520
14	2039	0.401237
15	2040	0.408954
16	2041	0.416671
17	2042	0.424388
18	2043	0.432105
19	2044	0.439822
20	2045	0.447538
21	2046	0.455255
22	2047	0.462972
23	2048	0.470689
24	2049	0.478406

In [37…

```
import matplotlib.pyplot as plt

plt.figure(figsize=(10,5))

# historical
plt.plot(df["Year"], df["Price_kWh"], marker="o", label="Historical")

# predicted
plt.plot(forecast_df["Year"], forecast_df["Predicted_Price_kWh"],
         marker="x", linestyle="--", label="25-Year Projection")
```

```
plt.xlabel("Year")
plt.ylabel("Price (€ per kWh)")
plt.title("Electricity Price Forecast (Next 25 Years)")
plt.grid(True)
plt.legend()
plt.show()
```



In [39…

```
last_year = df["Year"].max()
future_years = np.arange(last_year + 1, last_year + 51).reshape(-1, 1)
future_years
```

Out[39]:
```
array([[2025],
       [2026],
       [2027],
       [2028],
       [2029],
       [2030],
       [2031],
       [2032],
       [2033],
       [2034],
       [2035],
       [2036],
       [2037],
       [2038],
       [2039],
       [2040],
       [2041],
       [2042],
       [2043],
       [2044],
       [2045],
       [2046],
       [2047],
       [2048],
       [2049],
       [2050],
       [2051],
       [2052],
       [2053],
       [2054],
       [2055],
       [2056],
       [2057],
       [2058],
       [2059],
       [2060],
       [2061],
       [2062],
       [2063],
       [2064],
       [2065],
       [2066],
       [2067],
       [2068],
       [2069],
       [2070],
       [2071],
       [2072],
       [2073],
       [2074]])
```

In [40…
```
future_prices = model.predict(future_years)
future_prices
```

Out[40]:
```
array([0.2932    , 0.30091692, 0.30863385, 0.31635077, 0.32406769,
       0.33178462, 0.33950154, 0.34721846, 0.35493538, 0.36265231,
       0.37036923, 0.37808615, 0.38580308, 0.39352   , 0.40123692,
       0.40895385, 0.41667077, 0.42438769, 0.43210462, 0.43982154,
       0.44753846, 0.45525538, 0.46297231, 0.47068923, 0.47840615,
       0.48612308, 0.49384   , 0.50155692, 0.50927385, 0.51699077,
       0.52470769, 0.53242462, 0.54014154, 0.54785846, 0.55557538,
       0.56329231, 0.57100923, 0.57872615, 0.58644308, 0.59416   ,
       0.60187692, 0.60959385, 0.61731077, 0.62502769, 0.63274462,
       0.64046154, 0.64817846, 0.65589538, 0.66361231, 0.67132923])
```

In [41…
```
forecast_df = pd.DataFrame({
    "Year": future_years.flatten(),
    "Predicted_Price_kWh": future_prices
})

forecast_df
```

Out[41]:

	Year	Predicted_Price_kWh
0	2025	0.293200
1	2026	0.300917
2	2027	0.308634
3	2028	0.316351
4	2029	0.324068
5	2030	0.331785
6	2031	0.339502
7	2032	0.347218
8	2033	0.354935
9	2034	0.362652
10	2035	0.370369
11	2036	0.378086
12	2037	0.385803
13	2038	0.393520
14	2039	0.401237
15	2040	0.408954
16	2041	0.416671
17	2042	0.424388
18	2043	0.432105
19	2044	0.439822
20	2045	0.447538
21	2046	0.455255
22	2047	0.462972
23	2048	0.470689
24	2049	0.478406
25	2050	0.486123
26	2051	0.493840
27	2052	0.501557
28	2053	0.509274
29	2054	0.516991
30	2055	0.524708
31	2056	0.532425

	Year	Predicted_Price_kWh
32	2057	0.540142
33	2058	0.547858
34	2059	0.555575
35	2060	0.563292
36	2061	0.571009
37	2062	0.578726
38	2063	0.586443
39	2064	0.594160
40	2065	0.601877
41	2066	0.609594
42	2067	0.617311
43	2068	0.625028
44	2069	0.632745
45	2070	0.640462
46	2071	0.648178
47	2072	0.655895
48	2073	0.663612
49	2074	0.671329

2.) What types of outliers might you expect to occur in the following data sets?

a) Student grades

Very low grades (e.g. 0 or close to 0) from students who did not take the exam or did not prepare.

Very high grades (close to the maximum) from a few top students compared to the rest of the class.

Possibly data entry errors, e.g. grades outside the valid range (negative values or > 100).

b) Salary data

Extremely high salaries for a small number of people (CEOs, top managers, famous experts). These are strong upper outliers because salary distributions are very right-skewed.

Very low salaries for interns, part-time workers, or minimum-wage jobs.

Recording errors, e.g. an extra zero added so that 50,000 becomes 500,000.

c) Lifespans in Wikipedia

Very long lifespans (e.g. 110–120 years) for a few super-centenarians compared to the average of about 70–80 years.

Very short lifespans, for example people who died as babies or children.

Incorrect or uncertain historical data, e.g. implausible ages like 150 years due to wrong birth or death dates.

Part 2: Scores & Rankings

/ 60

3.) Let X represent a random variable drawn from the normal distribution defined by μ = 2 and σ = 3. Suppose we observe X = 5.08.

Find the Z-score of x, and determine how many standard deviations away from the mean that x is.

In [85…

```
import numpy as np
mu = 2
sigma = 3
x = 5.08

z = (x - mu) / sigma

z
```

Out[85]: 1.0266666666666666

4.) What percentage of the standard normal distribution (μ = 0, σ = 1) is found in each region?

a) Z > 1.13

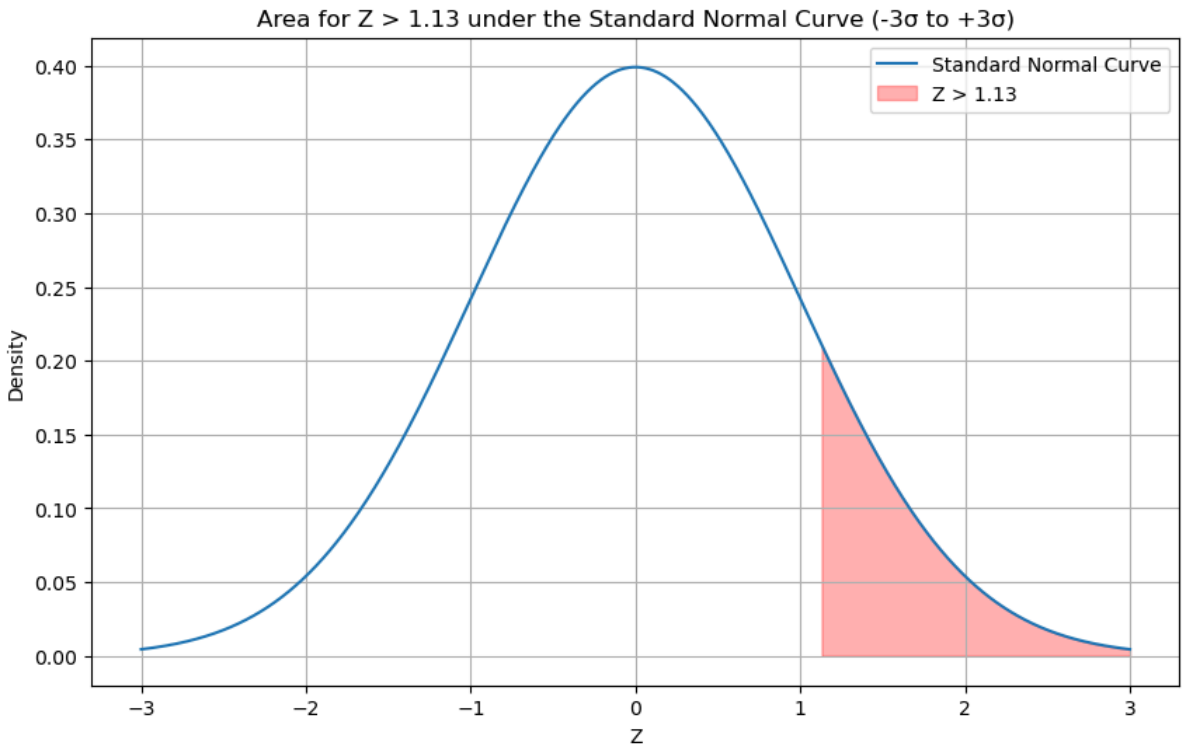
In [45…

```
from scipy.stats import norm

z = 1.13
prob = 1 - norm.cdf(z)

percentage = prob * 100
print(f"{percentage:.2f}%")
```

12.92%

In [91…

```python
import numpy as np
import matplotlib.pyplot as plt
from scipy.stats import norm

z = 1.13

# x-axis from -3σ to +3σ
x = np.linspace(-3, 3, 1000)
y = norm.pdf(x)

plt.figure(figsize=(10, 6))
plt.plot(x, y, label="Standard Normal Curve")

# Shade region Z > 1.13
x_shade = np.linspace(z, 3, 300)
y_shade = norm.pdf(x_shade)
plt.fill_between(x_shade, y_shade, color='red', alpha=0.3, label="Z > 1.13"

plt.title("Area for Z > 1.13 under the Standard Normal Curve (−3σ to +3σ)"
plt.xlabel("Z")
plt.ylabel("Density")
plt.legend()
plt.grid(True)
plt.show()
```



b) Z < 0.18

In [47…

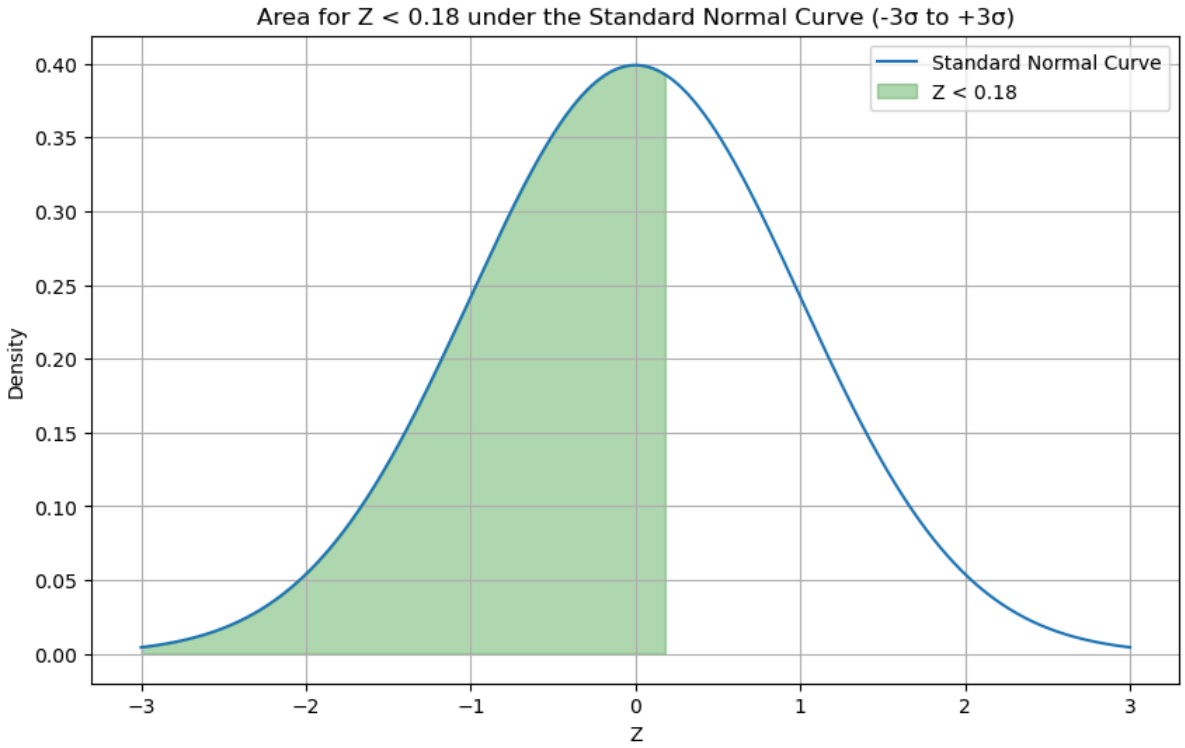
```python
from scipy.stats import norm

z = 0.18
prob = norm.cdf(z)
```

```
percentage = prob * 100
print(f"{percentage:.2f}%")
```

```
57.14%
```

In [90…

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.stats import norm

z = 0.18

# x-axis from -3σ to +3σ
x = np.linspace(-3, 3, 1000)
y = norm.pdf(x)

plt.figure(figsize=(10, 6))
plt.plot(x, y, label="Standard Normal Curve")

# Shade region Z < 0.18
x_shade = np.linspace(-3, z, 300)
y_shade = norm.pdf(x_shade)
plt.fill_between(x_shade, y_shade, color='green', alpha=0.3, label="Z < 0.1

plt.title("Area for Z < 0.18 under the Standard Normal Curve (−3σ to +3σ)"
plt.xlabel("Z")
plt.ylabel("Density")
plt.legend()
plt.grid(True)
plt.show()
```



c) Z > 8

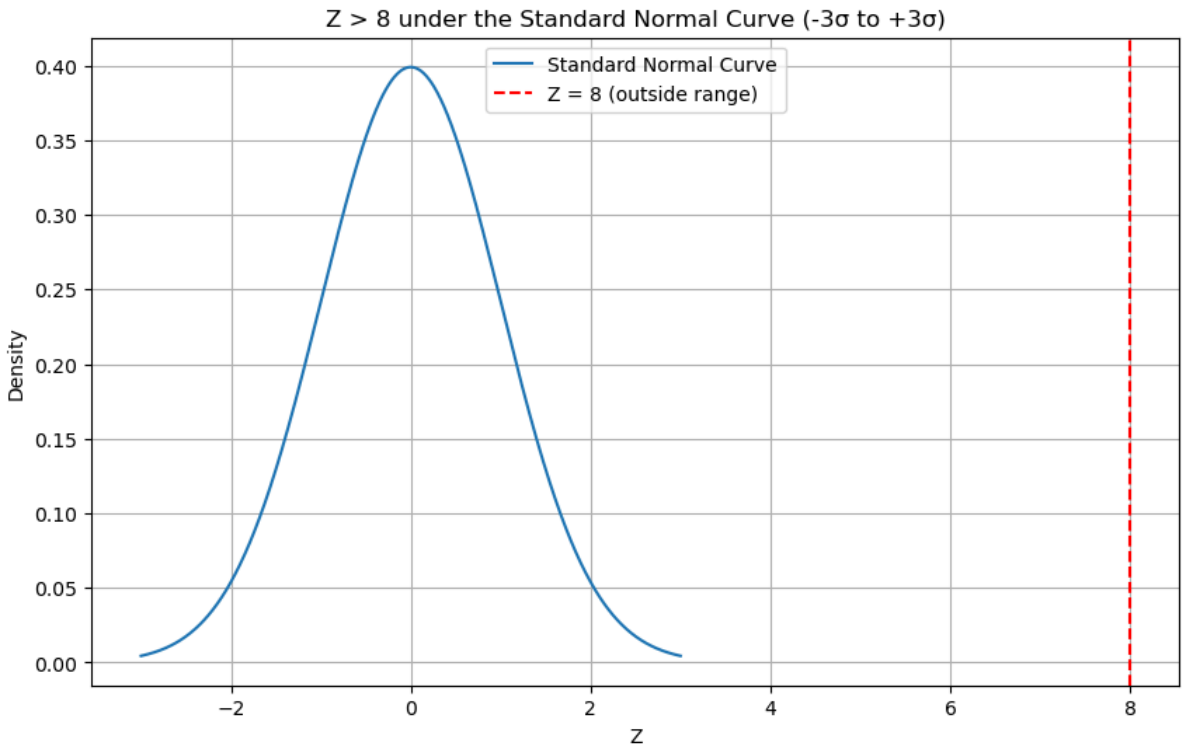
In [48…

```python
from scipy.stats import norm

z = 8
prob = 1 - norm.cdf(z)   # or norm.sf(z)
percentage = prob * 100
print(f"{percentage:.20f}%")
```

```
0.00000000000006661338%
```

In [92…

```python
import numpy as np
import matplotlib.pyplot as plt
from scipy.stats import norm

z = 8  # extremely far in the right tail

# x-axis only from -3σ to +3σ
x = np.linspace(-3, 3, 1000)
y = norm.pdf(x)

plt.figure(figsize=(10, 6))
plt.plot(x, y, label="Standard Normal Curve")

# Note: Z = 8 is far outside the visible range
plt.axvline(x=z, color='red', linestyle='--', label='Z = 8 (outside range)')

plt.title("Z > 8 under the Standard Normal Curve (−3σ to +3σ)")
plt.xlabel("Z")
plt.ylabel("Density")
plt.legend()
plt.grid(True)
plt.show()
```



d) |Z| < 0.5

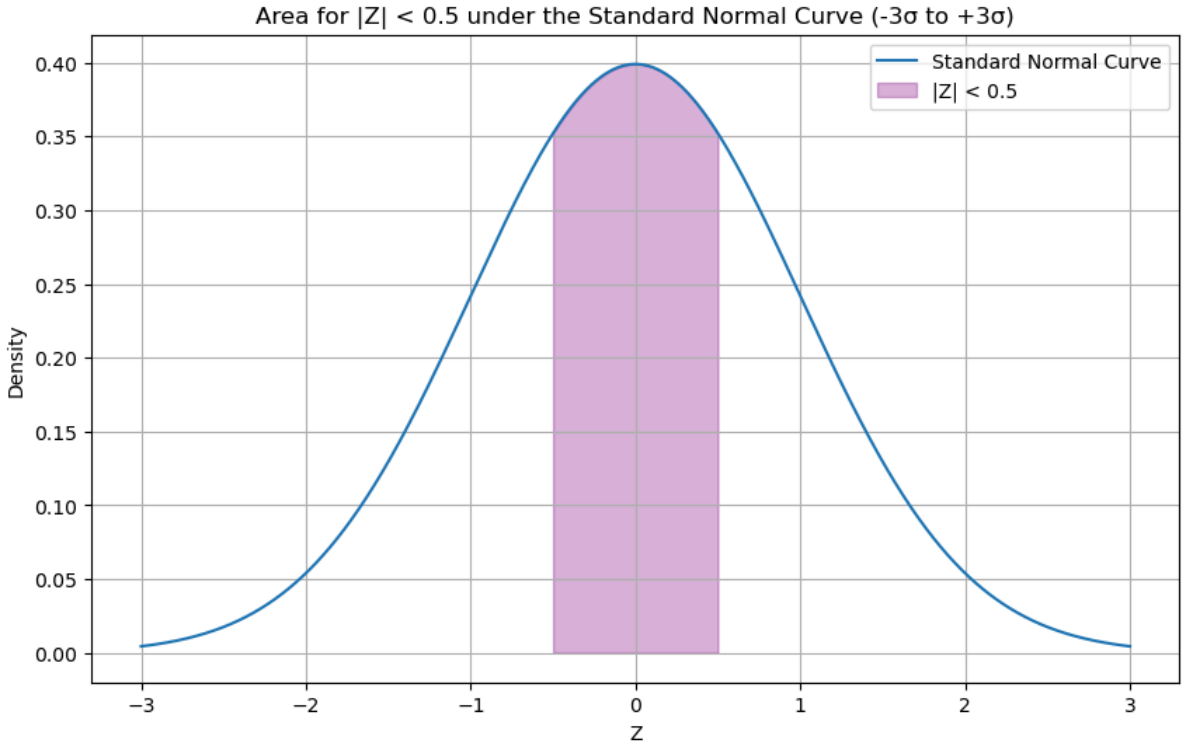
In [49…

```python
from scipy.stats import norm

lower = norm.cdf(-0.5)   # P(Z < -0.5)
upper = norm.cdf(0.5)    # P(Z < 0.5)

prob = upper - lower
percentage = prob * 100
print(f"{percentage:.2f}%")
```

```
38.29%
```

In [93…

```python
import numpy as np
import matplotlib.pyplot as plt
from scipy.stats import norm

# Bounds
left = -0.5
right = 0.5

# x-axis from -3σ to +3σ
x = np.linspace(-3, 3, 1000)
y = norm.pdf(x)

plt.figure(figsize=(10, 6))
plt.plot(x, y, label="Standard Normal Curve")

# Shade region between -0.5 and +0.5
x_shade = np.linspace(left, right, 300)
y_shade = norm.pdf(x_shade)
plt.fill_between(x_shade, y_shade, color='purple', alpha=0.3, label="|Z| <

plt.title("Area for |Z| < 0.5 under the Standard Normal Curve (−3σ to +3σ)'
plt.xlabel("Z")
plt.ylabel("Density")
plt.legend()
plt.grid(True)
plt.show()
```



5.) Amanda took the Graduate Record Examination (GRE), and scored 160 in verbal reasoning and 157 in quantitative reasoning. The mean score for verbal reasoning was 151 with a standard deviation of 7, compared with mean μ = 153 and σ = 7.67 for quantitative reasoning. Assume that both distributions are normal.

a) What were Amanda's Z-scores on these exam sections? Mark these scores on a standard normal distribution curve. Missing

In [96…

```
import numpy as np
mu_verbal=151
sigma_verbal=7
x_verbal=160

z_verbal=(x_verbal-mu_verbal)/sigma_verbal
z_verbal

mu_q=153
sigma_q=7.67
x_q=157

z_q=(x_q-mu_q)/sigma_q
z_q

print("Verbal Z-score:", z_verbal)
print("Quantitative Z-score:", z_q)
```

```
Verbal Z-score: 1.2857142857142858
Quantitative Z-score: 0.5215123859191656
```

b) Which section did she do better on, relative to other students?

She performed better on the verbal section because her Z-score in verbal (≈ 1.28) is higher than her quantitative Z-score (≈ 0.52). This means she is further above the average verbal score compared to the quantitative average.

c) Find her percentile scores for the two exams.

In [97…

```
from scipy.stats import norm

verbal_percentile = norm.cdf(z_verbal) * 100
quant_percentile = norm.cdf(z_q) * 100

verbal_percentile, quant_percentile
```

Out[97]:

```
(np.float64(90.0728603156669), np.float64(69.89950602269671))
```

Each percentile value shows the percentage of other students Amanda scored better than. For example, being in the 90th percentile means she performed better than 90% of all test takers.

6.) Identify three successful and well-used scoring functions in areas of personal interest to you. For each, explain what makes it a good scoring function and how it is used to create rankings in that domain.

1. Body Mass Index (BMI)

BMI combines a person's height and weight into one simple number to indicate whether they are underweight, normal, or overweight. It's useful because it's extremely easy to calculate and gives a quick, consistent way to compare people. Even though it's not perfect, it works well as a general health indicator.

It is crticized a lot.
I personally won't call the BMI successful :D

2. IMDB Weighted Rating

IMDB doesn't just use the simple average rating of a movie. Instead, it uses a weighted formula that also includes how many people voted. This prevents movies with only a few ratings from jumping to the top. I think this scoring function works well because it balances quality (rating) with reliability (vote count), which leads to fairer and more stable movie rankings.

3. Elo Rating

The Elo system scores players based on match results. Beating a strong opponent increases your score more than beating a weak one. This scoring function is useful because it updates continuously and reflects current skill, which is why it's widely used in chess, esports, and competitive games. Higher Elo simply means a stronger player.

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!

In [ ]: