Disease-X-Patient-Outcome-Analytics

In [1]:
```
import sqlite3
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import warnings

warnings.filterwarnings("ignore")
```

In [2]:
```
conn = sqlite3.connect("Disease X.db")
table_1 = "disease_x_raw"
disease_x_raw_df = pd.read_sql_query(f"SELECT * FROM {table_1};", conn)
disease_x_raw_df.head()
```

Out[2]:

	case_id	date_infection	date_hospitalization	date_outcome	outcome	age	age_unit
0	5fe599	2014-05-15	2014-05-08	None	None	2.0	years
1	b8812a	2014-05-20	2014-05-04	None	None	18.0	years
2	893f25	2014-05-22	2014-05-18	2014-05-29	Recover	3.0	years
3	be99c8	2014-05-23	2014-05-03	2014-05-24	Recover	16.0	years
4	07e3e8	2014-05-29	2014-05-22	2014-06-01	Recover	16.0	years

5 rows × 23 columns

In [3]:
```
disease_x_raw_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3675 entries, 0 to 3674
Data columns (total 23 columns):
 #   Column                Non-Null Count  Dtype
---  ------                --------------  -----
 0   case_id               3675 non-null   object
 1   date_infection        3675 non-null   object
 2   date_hospitalization  3675 non-null   object
 3   date_outcome          3052 non-null   object
 4   outcome               2841 non-null   object
 5   age                   3620 non-null   float64
 6   age_unit              3675 non-null   object
 7   age_years             3620 non-null   float64
 8   age_cat               3620 non-null   object
 9   age_cat5              3620 non-null   object
 10  hospital              3675 non-null   object
 11  lon                   3675 non-null   float64
 12  lat                   3675 non-null   float64
 13  wt_kg                 3675 non-null   float64
 14  ht_cm                 3675 non-null   float64
 15  ct_blood              3675 non-null   float64
 16  fever                 3524 non-null   object
 17  chills                3524 non-null   object
 18  cough                 3524 non-null   object
 19  aches                 3524 non-null   object
 20  vomit                 3524 non-null   object
 21  temp                  3586 non-null   float64
 22  bmi                   3675 non-null   float64
dtypes: float64(9), object(14)
memory usage: 660.5+ KB
```

In [4]:
```
disease_x_raw_df["age"].isna().sum()
```

Out[4]: 55

Observation - The age column contains mixed units (months and years), so I chose to use the standardized age_years column for consistency in analysis.

In [5]:
```
disease_x_raw_df[disease_x_raw_df["age"] != disease_x_raw_df["age_years"]][
    ["age", "age_unit", "age_years"]
]
```

Out[5]:

	age	age_unit	age_years
146	NaN	years	NaN
270	24.0	months	2.000000
291	NaN	months	NaN
341	NaN	years	NaN
349	7.0	months	0.583333
...	...	...	...
2963	NaN	years	NaN
2986	NaN	years	NaN
2994	NaN	years	NaN
3126	NaN	years	NaN
3261	NaN	years	NaN

75 rows × 3 columns

Observation - The date_infection and date_hospital columns are in object format, so I am converting them to datetime format for consistency and easier analysis.

In [6]:
```
date_columns = ["date_infection", "date_hospitalization", "date_outcome"]
for col in date_columns:
    disease_x_raw_df[col] = pd.to_datetime(disease_x_raw_df[col])

disease_x_raw_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3675 entries, 0 to 3674
Data columns (total 23 columns):
 #   Column                Non-Null Count  Dtype
---  ------                --------------  -----
 0   case_id               3675 non-null   object
 1   date_infection        3675 non-null   datetime64[ns]
 2   date_hospitalization  3675 non-null   datetime64[ns]
 3   date_outcome          3052 non-null   datetime64[ns]
 4   outcome               2841 non-null   object
 5   age                   3620 non-null   float64
 6   age_unit              3675 non-null   object
 7   age_years             3620 non-null   float64
 8   age_cat               3620 non-null   object
 9   age_cat5              3620 non-null   object
 10  hospital              3675 non-null   object
 11  lon                   3675 non-null   float64
 12  lat                   3675 non-null   float64
 13  wt_kg                 3675 non-null   float64
 14  ht_cm                 3675 non-null   float64
 15  ct_blood              3675 non-null   float64
 16  fever                 3524 non-null   object
 17  chills                3524 non-null   object
 18  cough                 3524 non-null   object
 19  aches                 3524 non-null   object
 20  vomit                 3524 non-null   object
 21  temp                  3586 non-null   float64
 22  bmi                   3675 non-null   float64
dtypes: datetime64[ns](3), float64(9), object(11)
memory usage: 660.5+ KB
```

In [7]:
```
table_2 = "inpatient_days_raw"
inpatient_days_raw_df = pd.read_sql_query(f"SELECT * FROM {table_2};", conn)
inpatient_days_raw_df.head()
```

Out[7]:

	Facility	Bed_Census_Calendar_Year	Bed_Census_Month_ID	pdays
0	Rosewood Hospital Center	2014	201401	3070
1	Rosewood Hospital Center	2014	201402	2798
2	Rosewood Hospital Center	2014	201403	3060
3	Rosewood Hospital Center	2014	201404	2872
4	Rosewood Hospital Center	2014	201405	3074

In [8]:
```
inpatient_days_raw_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1120 entries, 0 to 1119
Data columns (total 4 columns):
 #   Column                   Non-Null Count  Dtype
---  ------                   --------------  -----
 0   Facility                 1120 non-null   object
 1   Bed_Census_Calendar_Year 1120 non-null   int64
 2   Bed_Census_Month_ID      1120 non-null   int64
 3   pdays                    1120 non-null   int64
dtypes: int64(3), object(1)
memory usage: 35.1+ KB
```

In [9]:
```python
# Converting Bed_Census_Month_ID into date time object
inpatient_days_raw_df["Bed_Census_Month_ID"] = pd.to_datetime(
    inpatient_days_raw_df["Bed_Census_Month_ID"], format="%Y%m"
)
inpatient_days_raw_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1120 entries, 0 to 1119
Data columns (total 4 columns):
 #   Column                   Non-Null Count  Dtype
---  ------                   --------------  -----
 0   Facility                 1120 non-null   object
 1   Bed_Census_Calendar_Year 1120 non-null   int64
 2   Bed_Census_Month_ID      1120 non-null   datetime64[ns]
 3   pdays                    1120 non-null   int64
dtypes: datetime64[ns](1), int64(2), object(1)
memory usage: 35.1+ KB
```

The rates are commonly calculated by calendar year or quarter. Quarter 1 is from Jan to Mar, Quarter 2 is from Apr to Jun, Quarter 3 is from Jul to Sep, and Quarter 4 is from Oct to Dec.

Question: What is the disease X overall rate for Quarter 3 of 2014?`

In [10]:
```python
# Filter data for Quarter 3 of 2014 (July to September)
q3_cases_df = disease_x_raw_df[
    (disease_x_raw_df["date_infection"].dt.year == 2014)
    & (disease_x_raw_df["date_infection"].dt.month.isin([7, 8, 9]))
]

# Filter inpatient days for Quarter 3 of 2014
q3_inpatient_days_df = inpatient_days_raw_df[
    (inpatient_days_raw_df["Bed_Census_Month_ID"].dt.year == 2014)
    & (inpatient_days_raw_df["Bed_Census_Month_ID"].dt.month.isin([7, 8, 9]))
]
total_inpatient_days_q3 = q3_inpatient_days_df["pdays"].sum()

# Calculate disease rate for Q3 2014
rate_q3 = (q3_cases_df.shape[0] / total_inpatient_days_q3) * 10000
print(f"Disease X overall rate for Q3 2014: {rate_q3:.2f} per 10,000")
```

```
Disease X overall rate for Q3 2014: 58.24 per 10,000
```

Disease X cases are classified based on the source of acquisition. A case is classified as “Healthcare-acquired” if the disease is identified more than 5 days after the patient is admitted to the hospital, and as “Community-acquired” if the disease is identified within 5 days or less of admission.

Question: Using the datasets, calculate the Healthcare-acquired rate and Community-acquired rate for disease X, separately for Quarter 4 of 2014. Be sure to define how you will calculate each rate and explain your methodology

In [11]:
```
disease_x_raw_df["days_to_infection"] = (
    disease_x_raw_df["date_infection"] - disease_x_raw_df["date_hospitalization"
).dt.days


disease_x_raw_df["source_acquisition"] = disease_x_raw_df["days_to_infection"].a
    lambda x: "Healthcare-acquired" if x > 5 else "Community-acquired"
)
```

In [12]:
```
disease_x_raw_df.head()
```

Out[12]:

	case_id	date_infection	date_hospitalization	date_outcome	outcome	age	age_unit
0	5fe599	2014-05-15	2014-05-08	NaT	None	2.0	years
1	b8812a	2014-05-20	2014-05-04	NaT	None	18.0	years
2	893f25	2014-05-22	2014-05-18	2014-05-29	Recover	3.0	years
3	be99c8	2014-05-23	2014-05-03	2014-05-24	Recover	16.0	years
4	07e3e8	2014-05-29	2014-05-22	2014-06-01	Recover	16.0	years

5 rows × 25 columns

In [13]:
```
Healthcare_Acquired_q3 = disease_x_raw_df[
    disease_x_raw_df["source_acquisition"] == "Healthcare-acquired"
][
    (disease_x_raw_df["date_infection"].dt.year == 2014)
    & (disease_x_raw_df["date_infection"].dt.month.isin([7, 8, 9]))
]

rate_q3 = (Healthcare_Acquired_q3.shape[0] / total_inpatient_days_q3) * 10000
print(f"Disease X Healthcare_Acquired rate for Q3 2014: {rate_q3:.2f} per 10,000
```

```
Disease X Healthcare_Acquired rate for Q3 2014: 45.99 per 10,000
```

In [14]:
```
Community_Acquired_q3 = disease_x_raw_df[
    disease_x_raw_df["source_acquisition"] == "Community-acquired"
][
    (disease_x_raw_df["date_infection"].dt.year == 2014)
    & (disease_x_raw_df["date_infection"].dt.month.isin([7, 8, 9]))
]

rate_q3 = (Community_Acquired_q3.shape[0] / total_inpatient_days_q3) * 10000
print(f"Disease X Community_Acquired rate for Q3 2014: {rate_q3:.2f} per 10,000"
```

```
Disease X Community_Acquired rate for Q3 2014: 12.25 per 10,000
```

Definitions & Methodology

- Healthcare-acquired rate: Cases identified >5 days after admission per 10,000 inpatient days.
- Community-acquired rate: Cases identified ≤5 days after admission per 10,000 inpatient days.

Classification:

- Cases are classified using the difference between date_infection and date_hospitalization (days_to_infection).

Rate Formula:

- Rate = (Number of Cases / Total Inpatient Days) × 10,000.

Results for Q3 2014:

- Healthcare-acquired rate: 45.99 per 10,000 inpatient days.
- Community-acquired rate: 12.25 per 10,000 inpatient days.

Purpose:

- This method ensures fair comparison across hospitals and time periods by normalizing case counts.

The rates for disease X are also calculated at the hospital level to assess facility performance.

Question:

Calculate the Healthcare-acquired rate for disease X separately for Military Hospital and Port Hospital in Quarter 4 of 2014. Ensure to specify how you would compute the rates for each hospital and explain your methodology.

In [15]:
```
q4_cases_df = disease_x_raw_df[
    (disease_x_raw_df["date_infection"].dt.year == 2014)
    & (disease_x_raw_df["date_infection"].dt.month.isin([10, 11, 12]))
]
q4_cases_df.head()
```

Out[15]:

	case_id	date_infection	date_hospitalization	date_outcome	outcome	age	age_u
179	46bed2	2014-10-01	2014-09-27	2014-10-12	Death	20.0	yea
180	2adcc8	2014-10-01	2014-09-28	NaT	Recover	1.0	yea
181	7f3916	2014-10-02	2014-08-31	2014-10-12	Recover	2.0	yea
182	747ece	2014-10-01	2014-09-24	2014-10-12	Recover	4.0	yea
183	4086d0	2014-10-01	2014-09-26	NaT	Death	4.0	yea

5 rows × 25 columns

In [16]:
```
for hospital in ["Military Hospital", "Port Hospital"]:

    hospital_cases_df = q4_cases_df[q4_cases_df["hospital"] == hospital]
    healthcare_acquired_cases = hospital_cases_df[
        hospital_cases_df["source_acquisition"] == "Healthcare-acquired"
    ].shape[0]

    hospital_inpatient_days_df = inpatient_days_raw_df[
        (inpatient_days_raw_df["Facility"] == hospital)
        & (
            (inpatient_days_raw_df["Bed_Census_Month_ID"].dt.month.isin([10, 11,
            & (inpatient_days_raw_df["Bed_Census_Month_ID"].dt.year == 2014)
        )
    ]
    total_inpatient_days = hospital_inpatient_days_df["pdays"].sum()

    healthcare_acquired_rate = (
        (healthcare_acquired_cases / total_inpatient_days) * 10000
        if total_inpatient_days > 0
        else 0
    )

    print(f"{hospital}:")
    print(f"  Healthcare-acquired Cases: {healthcare_acquired_cases}")
    print(f"  Total Inpatient Days: {total_inpatient_days}")
    print(f"  Healthcare-acquired Rate: {healthcare_acquired_rate:.2f} per 10,00
```

```
Military Hospital:
  Healthcare-acquired Cases: 186
  Total Inpatient Days: 30861
  Healthcare-acquired Rate: 60.27 per 10,000

Port Hospital:
  Healthcare-acquired Cases: 403
  Total Inpatient Days: 51877
  Healthcare-acquired Rate: 77.68 per 10,000
```

Definition & Methodology:

- Healthcare-acquired rate is the number of Healthcare-acquired cases per 10,000 inpatient days for a hospital.

Filter Cases:

- Extract Q4 2014 cases for each hospital and count Healthcare-acquired cases based on the source_acquisition column.

Filter Inpatient Days:

- Sum the total inpatient days (pdays) for each hospital in Q4 2014.

Rate Formula:

- Rate = (Number of Cases / Total Inpatient Days) × 10,000.

Purpose:

- Standardizes infection rates for comparison between hospitals, normalized by inpatient days.

Insights

- Higher Infection Rate at Port Hospital: The Healthcare-acquired rate at Port Hospital (77.68 per 10,000) is significantly higher than at Military Hospital (60.27 per 10,000), indicating a potential need for improved infection control measures at Port Hospital.
- Disparity in Case Volume: Port Hospital recorded more than double the number of Healthcare-acquired cases (403 vs. 186) compared to Military Hospital, despite having only 68% more inpatient days, suggesting a higher infection burden at Port Hospital.

Question: Calculate the 95% confidence interval for the Healthcare-acquired rates of disease X for Military Hospital and Port Hospital in Quarter 4 of 2014 (as per Q3). What conclusions can you draw by comparing the confidence intervals of the two hospitals?

In [17]:
```
disease_x_q4_2014 = disease_x_raw_df[
    (pd.to_datetime(disease_x_raw_df["date_infection"]).dt.year == 2014)
    & (pd.to_datetime(disease_x_raw_df["date_infection"]).dt.month.isin([10, 11,
]

healthcare_acquired = disease_x_q4_2014[
    disease_x_q4_2014["source_acquisition"] == "Healthcare-acquired"
]

military_cases = healthcare_acquired[
    healthcare_acquired["hospital"] == "Military Hospital"
]
port_cases = healthcare_acquired[healthcare_acquired["hospital"] == "Port Hospit

military_days = inpatient_days_raw_df[
    (inpatient_days_raw_df["Facility"] == "Military Hospital")
    & (inpatient_days_raw_df["Bed_Census_Month_ID"].dt.year == 2014)
    & (inpatient_days_raw_df["Bed_Census_Month_ID"].dt.month.isin([10, 11, 12]))
```

```
][["pdays"].sum()

port_days = inpatient_days_raw_df[
    (inpatient_days_raw_df["Facility"] == "Port Hospital")
    & (inpatient_days_raw_df["Bed_Census_Month_ID"].dt.year == 2014)
    & (inpatient_days_raw_df["Bed_Census_Month_ID"].dt.month.isin([10, 11, 12]))
][["pdays"].sum()


print("Number of Military cases:", military_cases.shape[0])
print("Number of Military days:", military_days)
print("Number of Port cases:", port_cases.shape[0])
print("Number of Port days:", port_days)

military_rate = (military_cases.shape[0] / military_days) * 10000
port_rate = (port_cases.shape[0] / port_days) * 10000

def confidence_interval(rate, total_days, z=1.96):
    p = rate / 10000
    error = z * np.sqrt((p * (1 - p)) / total_days)
    lower = (p - error) * 10000
    upper = (p + error) * 10000
    return lower, upper


military_ci = confidence_interval(military_rate, military_days)
port_ci = confidence_interval(port_rate, port_days)

print(
    f"\n\nMilitary Hospital Rate: {military_rate:.2f} cases per 10,000 inpatient
)
print(f"95% CI for Military Hospital: ({military_ci[0]:.2f}, {military_ci[1]:.2f

print(f"Port Hospital Rate: {port_rate:.2f} cases per 10,000 inpatient days")
print(f"95% CI for Port Hospital: ({port_ci[0]:.2f}, {port_ci[1]:.2f})")
```

```
Number of Military cases: 186
Number of Military days: 30861
Number of Port cases: 403
Number of Port days: 51877


Military Hospital Rate: 60.27 cases per 10,000 inpatient days
95% CI for Military Hospital: (51.63, 68.91)
Port Hospital Rate: 77.68 cases per 10,000 inpatient days
95% CI for Port Hospital: (70.13, 85.24)
```

Conclusion & Insights:

- Lower Infection Rate at Military Hospital: Military Hospital has a significantly lower healthcare-acquired rate (60.27/10,000 inpatient days) compared to Port Hospital (77.68/10,000 inpatient days).
- Statistically Significant Difference: The non-overlapping confidence intervals (Military: 51.63–68.91, Port: 70.13–85.24) indicate a statistically significant difference in infection rates.

- Port Hospital's Higher Risk: Port Hospital's higher rate suggests a potential gap in infection prevention practices compared to Military Hospital.
- Efficiency at Military Hospital: Despite fewer inpatient days (30,861 vs. 51,877), Military Hospital managed to keep infection numbers low (186 cases vs. 403 cases).
- Actionable Priority: Port Hospital should prioritize evaluating and improving infection control measures to reduce its higher healthcare-acquired infection rate.

Infection Prevention and Control Data Analysts often use the epidemiological week case numbers to visualize trends. An epidemiological week runs from Sunday to Saturday. For example, 2014 Week 17 spans from Sunday, April 20, 2014, to Saturday, April 26, 2014.

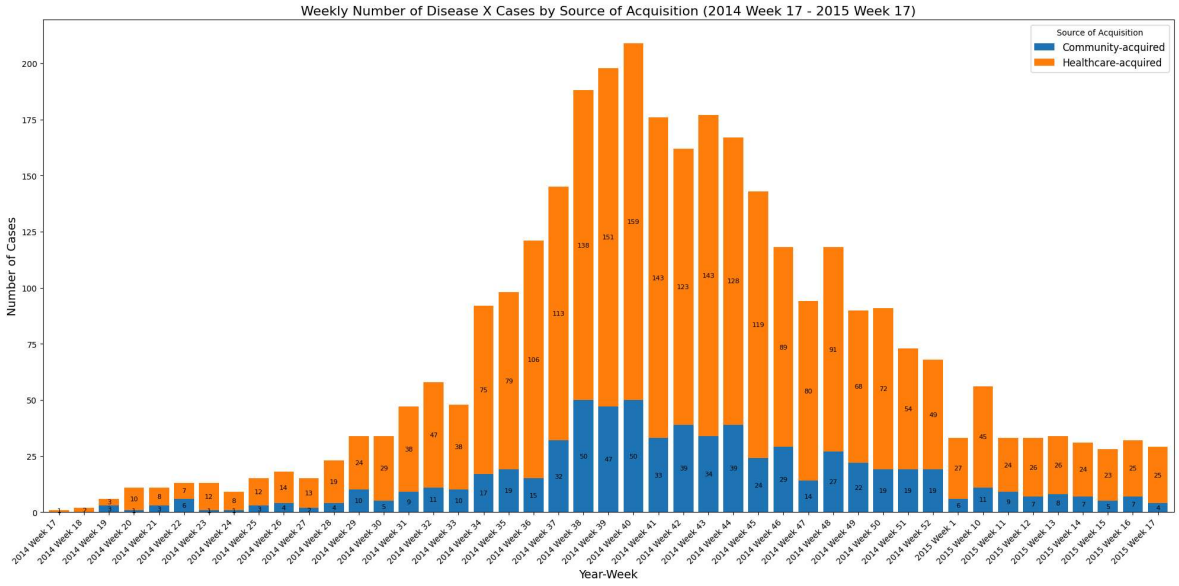
Question:

Generate a stacked bar chart showing the weekly number of confirmed disease X cases by source of acquisition (Healthcare-acquired vs. Community-acquired) for the period between 2014 Week 17 and 2015 Week 17. What trend do you observe in the bar chart?

In [18]:
```
def addlabels(x, y):
    for i in range(len(x)):
        plt.text(i, y[i], y[i], ha="center")
```

In [19]:
```
disease_x_raw_df["year_week"] = (
    disease_x_raw_df["date_infection"].dt.year.astype(str)
    + " Week "
    + disease_x_raw_df["date_infection"].dt.isocalendar().week.astype(str)
)

filtered_df = disease_x_raw_df[
    (disease_x_raw_df["year_week"] >= "2014 Week 17")
    & (disease_x_raw_df["year_week"] <= "2015 Week 17")
]


weekly_counts = (
    filtered_df.groupby(["year_week", "source_acquisition"])
    .size()
    .unstack(fill_value=0)
)


weekly_counts = weekly_counts.reindex(
    sorted(weekly_counts.index, key=lambda x: (int(x.split()[0]), int(x.split()[
)


fig, ax = plt.subplots(figsize=(20, 10))
bars = weekly_counts.plot(
    kind="bar",
    stacked=True,
    figsize=(20, 10),
    width=0.8,
    color=["#1f77b4", "#ff7f0e"],
    ax=ax,
)
```

```
for x_index, (week, row) in enumerate(weekly_counts.iterrows()):
    cumulative_height = 0
    for category, value in row.items():
        if value > 0:
            y = cumulative_height + value / 2
            ax.text(x_index, y, int(value), ha="center", va="center", fontsize=8
            cumulative_height += value

plt.title(
    "Weekly Number of Disease X Cases by Source of Acquisition (2014 Week 17 - 2
    fontsize=16,
)
plt.xlabel("Year-Week", fontsize=14)
plt.ylabel("Number of Cases", fontsize=14)


plt.xticks(
    rotation=45, fontsize=10, ha="right"
)
plt.subplots_adjust(bottom=0.2)


plt.legend(title="Source of Acquisition", fontsize=12)
plt.tight_layout()


plt.show()
```



Observation:

- The majority of Disease X cases are classified as healthcare-acquired, indicating that the disease is primarily spreading within healthcare settings after 5 days of admission. This suggests the need for stricter infection control measures in hospitals.

Clear Outbreak Pattern:

- There is a sharp rise in cases (both healthcare-acquired and community-acquired) starting around 2014 Week 30, peaking around 2014 Week 39–44, and then

gradually declining into 2015. This period aligns with the flu season, which typically occurs in late fall and winter, suggesting that Disease X may have intensified during this time due to increased hospitalizations and weakened immunity among patients.

Community-Acquired Cases Remain Relatively Low:

- Community-acquired cases are consistently lower than healthcare-acquired cases across the entire timeline, with no significant spikes. This indicates that Disease X is less prevalent in the community compared to hospitals, even during the flu season.

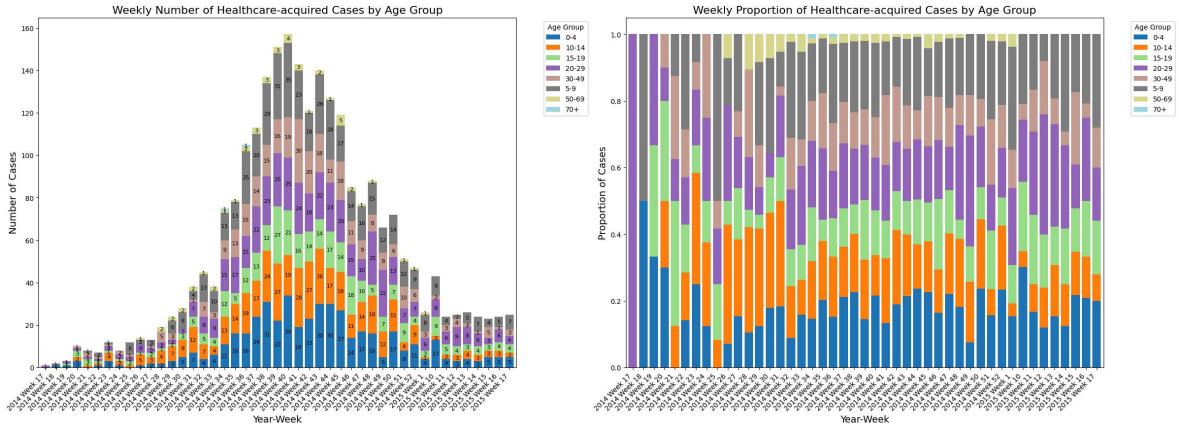
Decline After Peak:

- The sharp decline in cases after the peak suggests effective public health interventions or infection control measures were implemented, particularly in healthcare settings, to combat the spread during and after the flu season.
- By linking the outbreak to the flu season, it emphasizes the importance of heightened infection control during periods of increased hospitalizations and seasonal illnesses.

The proportion of disease X cases by age group provides valuable insight into how the disease affects different age populations. Question: For Healthcare-acquired cases only, generate a graph to show the weekly proportion of cases by age group between 2014 Week 17 and 2015 Week 17. Use age_cat to define the age groups. What trend do you observe from the graph?

In [20]:
```
healthcare_acquired_df = disease_x_raw_df[
    disease_x_raw_df["source_acquisition"] == "Healthcare-acquired"
]


healthcare_acquired_df["year_week"] = (
    healthcare_acquired_df["date_infection"].dt.year.astype(str)
    + " Week "
    + healthcare_acquired_df["date_infection"].dt.isocalendar().week.astype(str)
)

filtered_df = healthcare_acquired_df[
    (healthcare_acquired_df["year_week"] >= "2014 Week 17")
    & (healthcare_acquired_df["year_week"] <= "2015 Week 17")
]

weekly_counts_age = (
    filtered_df.groupby(["year_week", "age_cat"]).size().unstack(fill_value=0)
)


weekly_counts_age = weekly_counts_age.reindex(
    sorted(
        weekly_counts_age.index, key=lambda x: (int(x.split()[0]), int(x.split()
    )
)
```

```
weekly_age_group_counts = weekly_counts_age.copy()
weekly_proportions = weekly_age_group_counts.div(
    weekly_age_group_counts.sum(axis=1), axis=0
)

fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(24, 10), sharey=False)

weekly_counts_age.plot(kind="bar", stacked=True, width=0.8, cmap="tab20", ax=ax1


for x_index, (week, row) in enumerate(weekly_counts_age.iterrows()):
    cumulative_height = 0
    for idx, (category, value) in enumerate(row.items()):
        if value > 0:
            y = cumulative_height + value / 2
            ax1.text(x_index, y, int(value), ha="center", va="center", fontsize=
            cumulative_height += value


ax1.set_title(
    "Weekly Number of Healthcare-acquired Cases by Age Group",
    fontsize=16,
)
ax1.set_xlabel("Year-Week", fontsize=14)
ax1.set_ylabel("Number of Cases", fontsize=14)
ax1.tick_params(axis="x", rotation=45)
ax1.legend(title="Age Group", fontsize=10, bbox_to_anchor=(1.05, 1), loc="upper
ax1.set_xticks(range(len(weekly_counts_age.index)))
ax1.set_xticklabels(weekly_counts_age.index, rotation=45, ha="right")
ax1.grid(False)

weekly_proportions.plot(kind="bar", stacked=True, width=0.8, cmap="tab20", ax=ax

ax2.set_title(
    "Weekly Proportion of Healthcare-acquired Cases by Age Group",
    fontsize=16,
)
ax2.set_xlabel("Year-Week", fontsize=14)
ax2.set_ylabel("Proportion of Cases", fontsize=14)
ax2.tick_params(axis="x", rotation=45)
ax2.legend(title="Age Group", fontsize=10, bbox_to_anchor=(1.05, 1), loc="upper
ax2.set_xticks(range(len(weekly_proportions.index)))
ax2.set_xticklabels(weekly_proportions.index, rotation=45, ha="right")
ax2.grid(False)


plt.tight_layout()
plt.subplots_adjust(bottom=0.25)
plt.show()
```



Observations:

Middle-Aged and Elderly are the Most Affected:

- The majority of healthcare-acquired cases occur in the 30–69 and 70+ age groups, likely due to longer hospital stays and greater exposure to healthcare environments.

Effectiveness of Control Measures:

- Both the total numbers (left graph) and proportions (right graph) stabilize and decline after the peak in late 2014, indicating successful intervention measures or outbreak containment.

Focus on Vulnerable Groups:

- Future efforts should prioritize infection control strategies for middle-aged and elderly patients, as these groups are most vulnerable to healthcare-acquired infections.

Conduct a statistical hypothesis test to compare the mean age of Healthcare-acquired confirmed cases and Community-acquired confirmed cases. Provide a step-by-step explanation of your approach, including:

1. The null and alternative hypotheses.
2. The statistical test you will use.
3. The interpretation of your results.

In [21]:
```
healthcare_acquired = disease_x_raw_df[
    disease_x_raw_df["source_acquisition"] == "Healthcare-acquired"
]["age_years"].copy(deep=True)
community_acquired = disease_x_raw_df[
    disease_x_raw_df["source_acquisition"] == "Community-acquired"
]["age_years"].copy(deep=True)

healthcare_acquired = pd.to_numeric(healthcare_acquired, errors="coerce")
community_acquired = pd.to_numeric(community_acquired, errors="coerce")
```

In [22]:
```
print("Healthcare-acquired size:", healthcare_acquired.size)
print("Community-acquired size:", community_acquired.size)
```

```
print("Healthcare-acquired missing values:", healthcare_acquired.isnull().sum())
print("Community-acquired missing values:", community_acquired.isnull().sum())


print("Healthcare-acquired variance:", healthcare_acquired.var())
print("Community-acquired variance:", community_acquired.var())
```

```
Healthcare-acquired size: 2884
Community-acquired size: 791
Healthcare-acquired missing values: 47
Community-acquired missing values: 8
Healthcare-acquired variance: 162.00211489630638
Community-acquired variance: 150.44797456482362
```

Observation - Dropped null values from the age_years column in a copy of the original data to ensure accurate computations without affecting subsequent analysis steps.

In [23]:
```
healthcare_acquired.dropna(inplace=True)
community_acquired.dropna(inplace=True)
```

In [24]:
```
from scipy.stats import ttest_ind


t_stat, p_value_ttest = ttest_ind(
    healthcare_acquired, community_acquired, equal_var=True
)


print("Two-Sample T-Test Results")
print(f"T-statistic: {t_stat:.4f}, P-value: {p_value_ttest:.4f}")
if p_value_ttest < 0.05:
    print(
        "Interpretation: Reject the null hypothesis. There is a significant diff
    )
else:
    print(
        "Interpretation: Fail to reject the null hypothesis. There is no signifi
    )
```

```
Two-Sample T-Test Results
T-statistic: 0.8604, P-value: 0.3896
Interpretation: Fail to reject the null hypothesis. There is no significant diffe
rence in the mean age between healthcare-acquired and community-acquired cases.
```

Step-By-Step Explanation -

Data Splitting:

- Divided the dataset into two groups: Healthcare-acquired and Community-acquired cases based on the source_acquisition column.

Handling Null Values:

- Dropped null values from the age_years column in a copy of the original data to ensure accurate computations without affecting subsequent analysis steps.

Checking Variability:

- Verified the variance of both groups to confirm that the data is not constant.

Hypothesis Setup:

- Null Hypothesis : The mean age is the same for both groups.
- Alternative Hypothesis : The mean age differs between the two groups.

Two-Sample T-Test:

- Test Used: Two-sample t-test.
- Reason: Compares the means of two independent groups to detect significant differences.

Results Interpretation:

- The p-value (0.3896) is greater than the significance level (0.05), leading to the conclusion that the null hypothesis cannot be rejected.
- There is no significant difference in the mean age between Healthcare-acquired and Community-acquired cases.

What are the assumptions of the statistical test used? How would you check these assumptions, and do you find any violation of the assumptions? Please provide detailed steps for checking each assumption

In [25]:
```
from scipy.stats import shapiro, levene


healthcare_normality = shapiro(healthcare_acquired)
community_normality = shapiro(community_acquired)

print("Shapiro-Wilk Test for Normality:")
print(f"Healthcare-acquired: p-value = {healthcare_normality.pvalue}")
if healthcare_normality.pvalue < 0.05:
    print(
        "Interpretation: The age data for healthcare-acquired cases does not fol
    )
else:
    print(
        "Interpretation: The age data for healthcare-acquired cases follows a no
    )

print(f"Community-acquired: p-value = {community_normality.pvalue}")
if community_normality.pvalue < 0.05:
    print(
        "Interpretation: The age data for community-acquired cases does not foll
    )
else:
    print(
        "Interpretation: The age data for community-acquired cases follows a nor
    )
```

```
Shapiro-Wilk Test for Normality:
Healthcare-acquired: p-value = 2.4072200064210013e-38
Interpretation: The age data for healthcare-acquired cases does not follow a norm
al distribution.

Community-acquired: p-value = 5.394794083770993e-20
Interpretation: The age data for community-acquired cases does not follow a norma
l distribution.
```

Observation

its not a normal distribution as p-value is < 0.05 significance value

In [26]:
```
levene_stat, levene_pvalue = levene(healthcare_acquired, community_acquired)

print(
    f"Levene's Test for Equal Variances: Statistic={levene_stat:.4f}, P-value={l
)
if levene_pvalue < 0.05:
    print(
        "Interpretation: The variances of the two groups are significantly diffe
    )
else:
    print(
        "Interpretation: The variances of the two groups are not significantly d
    )
```

```
Levene's Test for Equal Variances: Statistic=0.1522, P-value=0.6964
Interpretation: The variances of the two groups are not significantly different.
The assumption of equal variances is satisfied.
```

Assumptions of the Statistical Test (Two-Sample T-Test):

Independence:

- The two groups (Healthcare-acquired and Community-acquired) must consist of independent observations.

Normality:

- The data in both groups should be approximately normally distributed.

Equality of Variances:

- The variances of the two groups should be equal (homoscedasticity)

Assumptions and Checks:

Independence:

- Check: Ensure observations are independent and belong to mutually exclusive groups.
- Finding: Assumption satisfied as patients are classified into distinct groups.

Normality:

- Test: Shapiro-Wilk test for both groups.
- Result: Both groups failed the normality test
- Conclusion: Normality assumption is violated.

Equality of Variances:

- Test: Levene's test to compare variances.
- Result: p-value = 0.6964 (fail to reject).

Conclusion: Assumption of equal variances is satisfied.

How would you evaluate the statistical test used ? What other tests, methods, or techniques could you use to compare the age distribution between the Healthcare-acquired group and the Community-acquired group? Provide step-by-step details of the alternative test(s) and explain why they might be appropriate.

In [27]:
```
from scipy.stats import mannwhitneyu

# Mann-Whitney U Test
mann_whitney_test = mannwhitneyu(
    healthcare_acquired, community_acquired, alternative="two-sided"
)

print("\nMann-Whitney U Test:")
print(f"U-statistic = {mann_whitney_test.statistic}")
print(f"p-value = {mann_whitney_test.pvalue}")


alpha = 0.05
if mann_whitney_test.pvalue < alpha:
    print(
        "\nResult: Reject the null hypothesis. There is a significant difference
    )
else:
    print(
        "\nResult: Fail to reject the null hypothesis. There is no significant d
    )
```

```
Mann-Whitney U Test:
U-statistic = 1128110.5
p-value = 0.5007335505636317

Result: Fail to reject the null hypothesis. There is no significant difference in
the distributions of age between Healthcare-acquired and Community-acquired case
s.
```

Reason for Evaluating:

- The two-sample t-test assumes normality, which was violated for both groups, making it less reliable for comparing their means.

Alternative Test: Mann-Whitney U Test:

- Reason: Does not assume normality and compares the distributions of two independent groups.

Hypotheses:

- Null Hypothesis: The distributions of age are the same for both groups.
- Alternative Hypothesis: The distributions of age differ between the two groups.

Result: U-statistic = 1,128,110.5, p-value = 0.5007.

- Fail to reject Null Hypothesis: No significant difference in age distributions between Healthcare-acquired and Community-acquired cases.

Conclusion:

- The Mann-Whitney U test is preferred when normality assumptions are violated, ensuring a valid comparison of distributions.

The Infection Prevention and Control Epidemiology Manager wants to determine if there is a significant difference in the mortality proportion between the Healthcare-acquired and Community-acquired groups. Question:

1. What are the mortality proportions in the Healthcare-acquired group and the Community-acquired group?
2. Is there a significant difference between the proportions? Provide step-by-step details of how you would calculate the proportions and test for significance.

In [28]:
```
from statsmodels.stats.proportion import proportions_ztest

healthcare_group = disease_x_raw_df[
    disease_x_raw_df["source_acquisition"] == "Healthcare-acquired"
]
community_group = disease_x_q4_2014.loc[
    disease_x_q4_2014["source_acquisition"] == "Community-acquired"
]

healthcare_deaths = healthcare_group[healthcare_group["outcome"] == "Death"].sha
community_deaths = community_group[community_group["outcome"] == "Death"].shape[

healthcare_total = healthcare_group.shape[0]
community_total = community_group.shape[0]

healthcare_mortality = healthcare_deaths / healthcare_total
community_mortality = community_deaths / community_total

print(f"Healthcare-acquired Mortality Proportion: {healthcare_mortality:.4f}")
print(f"Community-acquired Mortality Proportion: {community_mortality:.4f}")


count = [healthcare_deaths, community_deaths]
no_of_obs = [healthcare_total, community_total]

z_stat, p_value = proportions_ztest(count, no_of_obs)
print(f"Proportion Z-Test: Z-statistic = {z_stat:.4f}, P-value = {p_value:.4f}")

alpha = 0.05
if p_value < alpha:
    print(
        "Result: Reject the null hypothesis. There is a significant difference i
```

```
    )
else:
    print(
        "Result: Fail to reject the null hypothesis. No significant difference i
    )
```

```
Healthcare-acquired Mortality Proportion: 0.4324
Community-acquired Mortality Proportion: 0.4904
Proportion Z-Test: Z-statistic = -2.1047, P-value = 0.0353
Result: Reject the null hypothesis. There is a significant difference in mortalit
y proportions.
```

Mortality Proportions:

Healthcare-acquired group: Mortality proportion = 0.4324 (43.24%). Community-acquired group: Mortality proportion = 0.4904 (49.04%).

Test for Significant Difference:

Method:

- Used a two-proportion z-test to compare mortality proportions between the two groups.
- Null Hypothesis : The mortality proportions are equal for both groups.
- Alternative Hypothesis : The mortality proportions are significantly different between the groups.

Steps:

- Count the number of deaths and total cases for each group.
- Calculate mortality proportions (deaths/totaldeaths / totaldeaths/total).
- Perform a z-test using proportions_ztest() with counts and totals from both groups.

Results: Z-statistic = −2.1047 | P-value = 0.0353

Interpretation: Since the p-value (0.0353) is less than the significance level (0.05), we reject the null hypothesis.

Conclusion: There is a significant difference in mortality proportions between Healthcare-acquired and Community-acquired groups.

Insights

- ***Higher Mortality in Community-acquired Cases***: The mortality proportion is significantly higher in Community-acquired cases (49.04%) compared to Healthcare-acquired cases (43.24%), despite the earlier onset of the disease.
- ***Significant Difference in Mortality***: The two-proportion z-test confirms a statistically significant difference (p-value = 0.0353) in mortality between the two groups, suggesting that the timing of disease identification may impact patient outcomes.

Length of Stay (LOS) refers to the amount of time a patient spends in a healthcare facility and is an important metric in public health, healthcare management, and research. The

Infection Prevention and Control Epidemiology Manager wants to determine which variables are important predictors of the Length of Stay for disease X cases. Question:

1. First, create the response variable LOS (Length of Stay).
2. Build a multivariate linear regression model using the following predictors: age_years, wt_kg, ht_cm, bmi, ct_blood, fever, chills, cough, aches, vomit, temp, and source of acquisition (Healthcare-acquired/Community-acquired).
3. Do you find any important predictors for LOS?
4. How do you interpret the coefficient estimate for the predictor source of acquisition?
5. How would you evaluate the model performance? Please provide step-by-step details for each part of the analysis

In [29]:
```
disease_x_raw_df["LOS"] = (
    disease_x_raw_df["date_outcome"] - disease_x_raw_df["date_hospitalization"]
).dt.days
disease_x_raw_df
```

Out[29]:

	case_id	date_infection	date_hospitalization	date_outcome	outcome	age	age_u
0	5fe599	2014-05-15	2014-05-08	NaT	None	2.0	ye
1	b8812a	2014-05-20	2014-05-04	NaT	None	18.0	ye
2	893f25	2014-05-22	2014-05-18	2014-05-29	Recover	3.0	ye
3	be99c8	2014-05-23	2014-05-03	2014-05-24	Recover	16.0	ye
4	07e3e8	2014-05-29	2014-05-22	2014-06-01	Recover	16.0	ye
...	...	...	...	...	...	...	
3670	4799df	2015-04-20	2015-04-05	2015-04-26	Death	7.0	ye
3671	bbf39f	2015-04-19	2015-03-28	2015-04-19	Death	5.0	ye
3672	3da96a	2015-04-20	2015-04-14	2015-05-20	Recover	13.0	ye
3673	76d8fe	2015-04-22	2015-04-16	2015-05-06	Recover	7.0	ye
3674	2068d6	2015-04-30	2015-04-20	2015-05-02	Death	7.0	ye

3675 rows × 27 columns

Note: When calculating Length of Stay (LOS) by subtracting date_hospitalization from date_outcome, any missing value (NaN) in either column will result in the LOS being

recorded as NaN.

In [30]:
```
required_features = [
    "age_years",
    "wt_kg",
    "ht_cm",
    "bmi",
    "ct_blood",
    "fever",
    "chills",
    "cough",
    "aches",
    "vomit",
    "temp",
    "source_acquisition",
    "LOS",
]
```

In [31]:
```
disease_x_raw_df["date_outcome"] = pd.to_datetime(
    disease_x_raw_df["date_outcome"], errors="coerce"
)
disease_x_raw_df["date_hospitalization"] = pd.to_datetime(
    disease_x_raw_df["date_hospitalization"], errors="coerce"
)

print("num LOS nans", disease_x_raw_df["LOS"].isna().sum())
disease_data = disease_x_raw_df.dropna(subset=["LOS"]).copy(
    deep=True
)
predictors = [
    "age_years",
    "wt_kg",
    "ht_cm",
    "bmi",
    "ct_blood",
    "fever",
    "chills",
    "cough",
    "aches",
    "vomit",
    "temp",
    "source_acquisition_Healthcare-acquired",
]
response = "LOS"


Data = pd.get_dummies(disease_data, columns=["source_acquisition"], drop_first=T
binary_columns = ["fever", "chills", "cough", "aches", "vomit"]
for col in binary_columns:
    Data[col] = Data[col].map({"yes": 1, "no": 0})


Data["source_acquisition_Healthcare-acquired"] = Data[
    "source_acquisition_Healthcare-acquired"
].astype(int)
```

```
num LOS nans 623
```

In [32]:
```
Data[predictors].isna().sum()
```

Out[32]:
```
age_years                                46
wt_kg                                     0
ht_cm                                     0
bmi                                       0
ct_blood                                  0
fever                                   124
chills                                  124
cough                                   124
aches                                   124
vomit                                   124
temp                                     74
source_acquisition_Healthcare-acquired    0
dtype: int64
```

In [33]:
```
Data.shape
```

Out[33]:
```
(3052, 27)
```

In [34]:
```
X = Data[predictors]
y = Data[response]
```

In [35]:
```
X.describe()
```

Out[35]:

	age_years	wt_kg	ht_cm	bmi	ct_blood	fever	2
count	3006.000000	3052.000000	3052.000000	3052.000000	3052.000000	2928.000000	
mean	15.832557	52.338467	124.623853	46.749436	21.220183	0.805328	
std	12.587387	18.534150	49.353733	53.191458	1.703914	0.396016	
min	0.000000	-10.000000	8.000000	-600.000000	16.000000	0.000000	
25%	6.000000	41.000000	91.000000	24.489796	20.000000	1.000000	
50%	13.000000	54.000000	129.000000	32.275417	22.000000	1.000000	
75%	23.000000	66.000000	158.000000	49.984357	22.000000	1.000000	
max	84.000000	111.000000	295.000000	816.326531	26.000000	1.000000	

In [36]:
```
median_values = X.median()


print("Median values for each column:")
print(median_values)
```

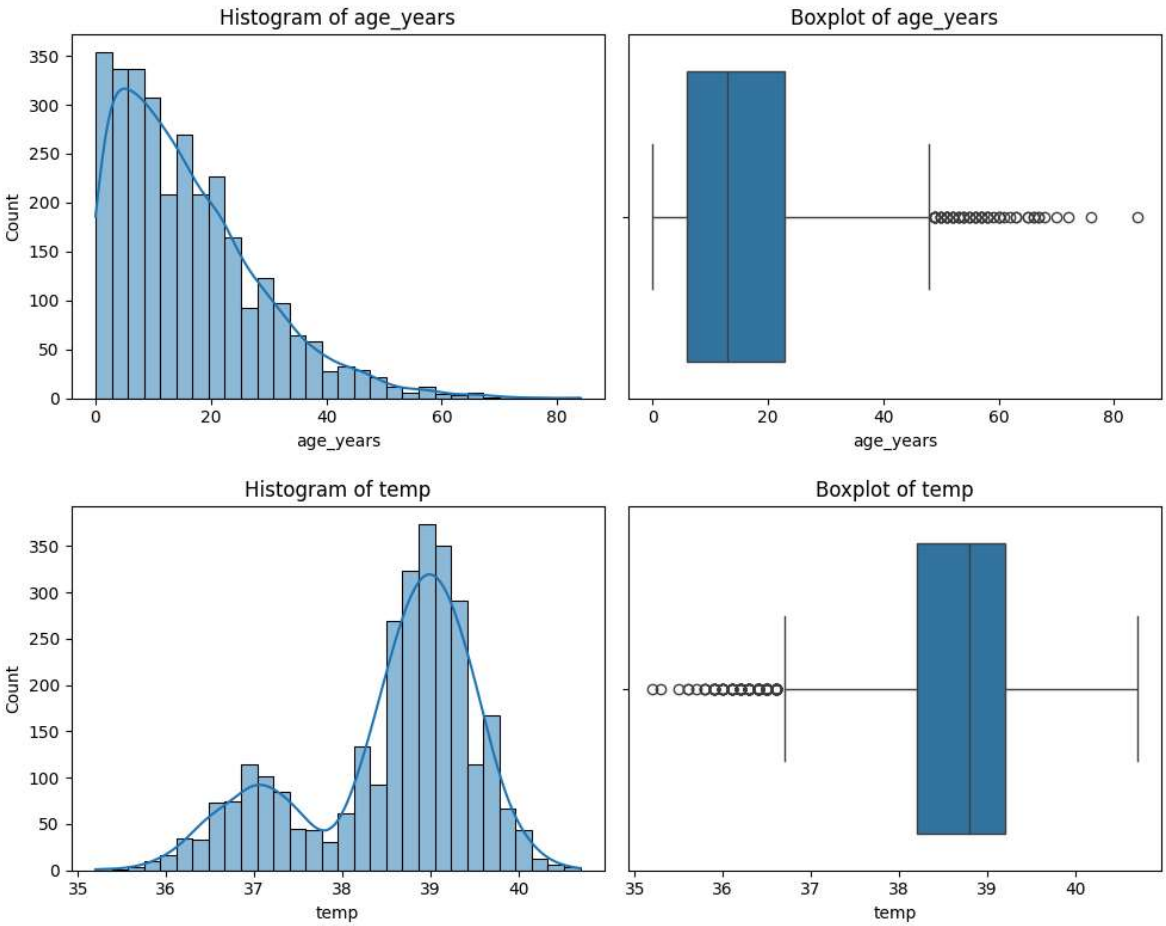
```
Median values for each column:
age_years                                  13.000000
wt_kg                                      54.000000
ht_cm                                     129.000000
bmi                                        32.275417
ct_blood                                   22.000000
fever                                       1.000000
chills                                      0.000000
cough                                       1.000000
aches                                       0.000000
vomit                                       0.000000
temp                                       38.800000
source_acquisition_Healthcare-acquired      1.000000
dtype: float64
```

In [37]:
```python
def mode_with_count(column):
    mode = column.mode().iloc[0]
    count = (column == mode).sum()
    return mode, count


mode_results = {}
for col in X.columns:
    mode, count = mode_with_count(X[col])
    mode_results[col] = {"Mode": mode, "Count": count}


mode_df = pd.DataFrame(mode_results).T

print("Mode and Count of Mode for Each Column:")
print(mode_df)
```

```
Mode and Count of Mode for Each Column:
                                          Mode   Count
age_years                                  2.0   143.0
wt_kg                                     61.0    79.0
ht_cm                                    134.0    38.0
bmi                                       32.0     7.0
ct_blood                                  22.0   858.0
fever                                      1.0  2358.0
chills                                     0.0  2370.0
cough                                      1.0  2526.0
aches                                      0.0  2653.0
vomit                                      0.0  1467.0
temp                                      39.0   194.0
source_acquisition_Healthcare-acquired     1.0  2393.0
```

In [38]:
```python
columns_to_check = ["age_years", "temp"]

for col in columns_to_check:


    data_col = X[col]

    # Visual Inspection: Histogram and Q-Q Plot
    plt.figure(figsize=(10, 4))

    # Histogram
    plt.subplot(1, 2, 1)
    sns.histplot(data_col, kde=True, bins=30)
```

```
    plt.title(f"Histogram of {col}")

    # Q-Q Plot
    plt.subplot(1, 2, 2)
    sns.boxplot(x=data_col)
    plt.title(f"Boxplot of {col}")

    plt.tight_layout()
    plt.show()
```



Imputation Strategy

Why Imputation is needed

- Preserve Data Size: Without imputation, removing rows with missing values would drop nearly 1,000 rows (~1/3 of the dataset), even though each column had relatively few missing values (e.g., a maximum of 124 out of 3,052).
- Avoid Data Loss: Imputation ensures minimal data loss while retaining the dataset's integrity, preserving valuable information for analysis.

For Discrete Columns (fever, chills, cough, aches, vomit):

- These columns are binary (0/1), representing categorical data (e.g., yes/no).
- Justification - Mode is the most frequently occurring value in the column, making it the most logical choice for imputation in discrete variables.
- Conclusion - Filling missing values with the mode ensures consistency with the rest of the data.

For Continuous Columns (age_years, temp):

- Based on the normality test results: age_years and temp are not normally distributed.
- Justification - Mode is unsuitable for continuous data. Since the missing values (101 for age_years and 104 for temp) represent a small percentage of the total dataset (3052 rows), the median is a robust choice for imputation. The median is less sensitive to outliers compared to the mean and better captures the central tendency for non-normal data.
- Conclusion - Filling missing values with the median provides a reliable estimate without distorting the data.

In [39]:
```
# Discrete columns: Fill NaN with mode
discrete_cols = ["fever", "chills", "cough", "aches", "vomit"]
for col in discrete_cols:
    X[col].fillna(X[col].mode()[0], inplace=True)

# Continuous columns: Fill NaN with median
continuous_cols = ["age_years", "temp"]
for col in continuous_cols:
    X[col].fillna(X[col].median(), inplace=True)

# Verify if all NaN values are filled
print("Missing values after imputation:")
print(X.isnull().sum())
```

```
Missing values after imputation:
age_years                                 0
wt_kg                                     0
ht_cm                                     0
bmi                                       0
ct_blood                                  0
fever                                     0
chills                                    0
cough                                     0
aches                                     0
vomit                                     0
temp                                      0
source_acquisition_Healthcare-acquired    0
dtype: int64
```

In [40]:
```
df = disease_x_raw_df[required_features]
df.dropna(subset=["LOS"], inplace=True)
# df.dropna(inplace=True)
```

In [41]:
```
from sklearn.model_selection import train_test_split

train_df, test_df = train_test_split(df, test_size=0.2, random_state=123)
train_df.head()
```

Out[41]:

	age_years	wt_kg	ht_cm	bmi	ct_blood	fever	chills	cough	aches	vomit
2326	12.0	48.0	114.0	36.934441	21.0	yes	no	yes	no	yes
2351	25.0	62.0	155.0	25.806452	23.0	yes	no	yes	no	yes
300	16.0	31.0	107.0	27.076601	20.0	no	no	no	no	yes
2376	3.0	30.0	75.0	53.333333	23.0	yes	no	no	no	yes
718	60.0	94.0	263.0	13.589903	19.0	no	yes	yes	no	no

In [42]:
```
X_train = train_df.drop(columns=["LOS"])
X_test = test_df.drop(columns=["LOS"])
y_train = train_df["LOS"]
y_test = test_df["LOS"]
```

In [43]:
```
from sklearn.pipeline import make_pipeline
from sklearn.preprocessing import StandardScaler
from sklearn.compose import make_column_transformer
from sklearn.preprocessing import OneHotEncoder, StandardScaler
from sklearn.impute import SimpleImputer


numeric_features = ["age_years", "wt_kg", "ht_cm", "bmi", "ct_blood", "temp"]
binary_features = ["fever", "chills", "cough", "aches", "vomit", "source_acquisi

# Define imputers
numeric_imputer = SimpleImputer(
    strategy="median"
)  # For numeric features, use mean imputation
binary_imputer = SimpleImputer(
    strategy="most_frequent"
)  # For binary features, use most frequent value (mode)

# Create the column transformer with imputation and preprocessing
preprocessor = make_column_transformer(
    (make_pipeline(numeric_imputer, StandardScaler()), numeric_features),
    (
        make_pipeline(binary_imputer, OneHotEncoder(drop="if_binary", dtype=int)
        binary_features,
    ),
)
```

In [44]:
```
df[numeric_features + ["LOS"]].corr("spearman").style.background_gradient()
```

Out[44]:

	age_years	wt_kg	ht_cm	bmi	ct_blood	temp	LOS
age_years	1.000000	0.876119	0.913220	-0.773516	0.003322	0.019731	-0.013348
wt_kg	0.876119	1.000000	0.874049	-0.633414	-0.008219	0.018258	-0.006645
ht_cm	0.913220	0.874049	1.000000	-0.901101	0.020988	0.017685	-0.010415
bmi	-0.773516	-0.633414	-0.901101	1.000000	-0.033416	-0.014054	0.018776
ct_blood	0.003322	-0.008219	0.020988	-0.033416	1.000000	-0.031177	-0.035636
temp	0.019731	0.018258	0.017685	-0.014054	-0.031177	1.000000	0.000779
LOS	-0.013348	-0.006645	-0.010415	0.018776	-0.035636	0.000779	1.000000

In [45]:

```
from sklearn.dummy import DummyRegressor
from sklearn.model_selection import cross_validate

# Define regression metrics
regression_metrics = ["r2", "neg_mean_squared_error", "neg_mean_absolute_error"]

# The dummy model for regression
dr = DummyRegressor()

# Perform cross-validation and store results
cross_val_results = {}
cross_val_results["dummy"] = (
    pd.DataFrame(
        cross_validate(
            dr, X_train, y_train, return_train_score=True, scoring=regression_me
        )
    )
    .agg(["mean", "std"])
    .round(3)
    .T
)

# Show the train and validation scores
cross_val_results["dummy"]
```

Out[45]:

	mean	std
fit_time	0.003	0.001
score_time	0.003	0.002
test_r2	-0.002	0.002
train_r2	0.000	0.000
test_neg_mean_squared_error	-129.552	12.755
train_neg_mean_squared_error	-129.440	3.199
test_neg_mean_absolute_error	-8.876	0.261
train_neg_mean_absolute_error	-8.871	0.083

In [46]:
```
from sklearn.linear_model import LinearRegression, Ridge, Lasso
from sklearn.pipeline import make_pipeline

# The linear regression model pipeline
linreg = make_pipeline(preprocessor, LinearRegression())

# The mean and std of the cross-validated scores for all metrics as a DataFrame
cross_val_results["linreg"] = (
    pd.DataFrame(
        cross_validate(
            linreg,
            X_train,
            y_train,
            return_train_score=True,
            scoring=regression_metrics,
        )
    )
    .agg(["mean", "std"])
    .round(3)
    .T
)

# Show the train and validation scores
cross_val_results["linreg"]
```

Out[46]:

	mean	std
fit_time	0.026	0.006
score_time	0.008	0.002
test_r2	0.118	0.020
train_r2	0.128	0.005
test_neg_mean_squared_error	-114.055	12.038
train_neg_mean_squared_error	-112.904	3.008
test_neg_mean_absolute_error	-8.264	0.313
train_neg_mean_absolute_error	-8.216	0.100

In [47]:
```
linreg.fit(X_train, y_train)
# Extract the LinearRegression model from the pipeline
linear_model = linreg.named_steps["linearregression"]

# Get the feature names from the preprocessor step
feature_names = linreg.named_steps["columntransformer"].get_feature_names_out()

# Create a DataFrame with predictors and coefficients
coefficients = pd.DataFrame(
    {"Predictor": feature_names, "Coefficient": linear_model.coef_}
).sort_values(by="Coefficient", ascending=False)

# Display the coefficients
print("Coefficients:")
print(coefficients)
```

```
Coefficients:
                                              Predictor   Coefficient
19                              pipeline-2__vomit_yes  9.011030e+09
18                               pipeline-2__vomit_no  9.011030e+09
21  pipeline-2__source_acquisition_Healthcare-acqu...  9.933830e+00
3                                     pipeline-1__bmi  4.996907e-01
2                                    pipeline-1__ht_cm  3.093414e-01
4                                 pipeline-1__ct_blood  2.705514e-01
1                                    pipeline-1__wt_kg -6.516419e-02
0                                 pipeline-1__age_years -2.837442e-01
5                                      pipeline-1__temp -3.649574e-01
12                                pipeline-2__cough_no -1.176234e+10
13                               pipeline-2__cough_yes -1.176234e+10
15                                pipeline-2__aches_no -1.207678e+10
16                               pipeline-2__aches_yes -1.207678e+10
10                              pipeline-2__chills_yes -1.287105e+10
9                                pipeline-2__chills_no -1.287105e+10
8                               pipeline-2__fever_None -1.389648e+10
14                              pipeline-2__cough_None -1.540276e+10
17                              pipeline-2__aches_None -1.540276e+10
20                              pipeline-2__vomit_None -1.540276e+10
11                             pipeline-2__chills_None -1.583403e+10
7                                pipeline-2__fever_yes -4.823966e+10
6                                 pipeline-2__fever_no -4.823966e+10
```

1. Create the Response Variable LOS (Length of Stay):

- The Length of Stay (LOS) was calculated as the difference in days between date_outcome (discharge date or outcome date) and date_hospitalization.

2. Build a Multivariate Linear Regression Model:

Predictors Used:

age_years, wt_kg, ht_cm, bmi, ct_blood, fever, chills, cough, aches, vomit, temp, and source_acquisition (Healthcare-acquired/Community-acquired).

Data Preprocessing:

- Binary features (e.g., fever, chills, etc.) were encoded as 0 (No) and 1 (Yes).
- Categorical feature (source_acquisition) was one-hot encoded, creating a binary column for Healthcare-acquired.
- Missing values in numeric predictors were imputed using the median, and in binary predictors using the mode.
- A preprocessing pipeline was constructed for scaling, encoding, and imputing missing values

Model Construction:

- A multivariate linear regression model was trained using the preprocessed data.

3. Do You Find Any Important Predictors for LOS?

- The coefficients of the linear regression model were extracted to evaluate the importance of predictors.

- Key Predictors (based on the magnitude of coefficients): Positive impact (increase LOS): pipeline-2__source_acquisition_Healthcare-acquired (Coefficient: 9.93) pipeline-1__bmi (Coefficient: 0.50) Negative impact (decrease LOS): pipeline-1__age_years (Coefficient: -0.28) pipeline-1__temp (Coefficient: -3.65)

4. Interpret the Coefficient Estimate for the Predictor source_acquisition:

- The coefficient for source_acquisition_Healthcare-acquired is approximately 9.93.
- Interpretation: Patients with healthcare-acquired infections stay, on average, 9.93 days longer in the hospital compared to those with community-acquired infections, holding all other predictors constant.

5. Evaluate the Model Performance:

- Dummy Model (Baseline):

```
The dummy model provides a baseline for comparison:
Test R²: -0.002 (indicates poor fit).
Test Negative MSE: -129.55.
Test Negative MAE: -8.87.
```

- Linear Regression Model:

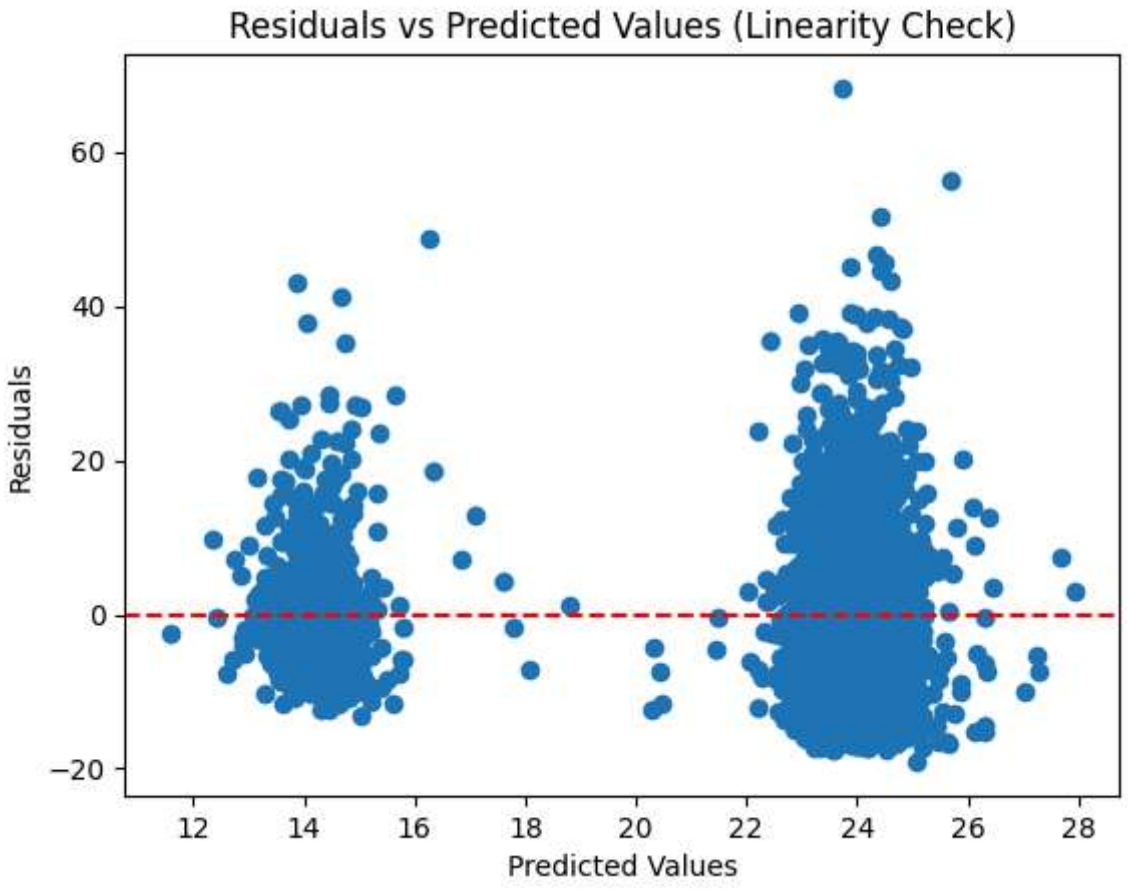
```
Performance metrics for the linear regression model:
Test R²: 0.118 (modest improvement over the dummy model).
Test Negative MSE: -114.05 (lower error compared to the dummy model).
Test Negative MAE: -8.26 (lower absolute error compared to the dummy model).
```

Interpretation: The model explains 11.8% of the variance in LOS and provides better predictions than the baseline dummy model

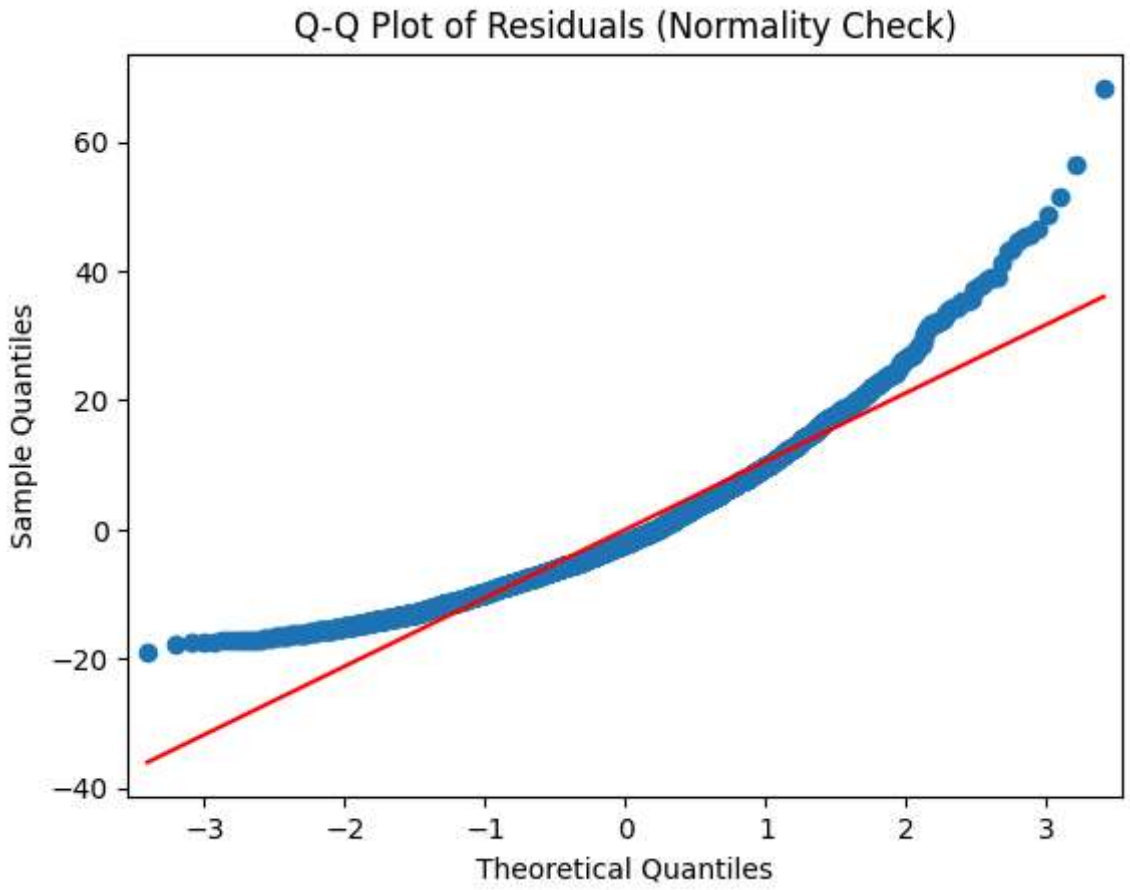
What are the assumptions of the multivariate linear regression model used? How would you validate these assumptions, and do you find any violation of the assumptions? Please provide detailed steps for validating each assumption

In [48]:
```
import statsmodels.api as sm
from statsmodels.stats.outliers_influence import variance_inflation_factor
from statsmodels.stats.diagnostic import het_breuschpagan
from scipy.stats import shapiro
from statsmodels.stats.stattools import durbin_watson

# Fit the regression model
X_with_const = sm.add_constant(X)  # Add constant for intercept
model = sm.OLS(y, X_with_const).fit()


predicted = model.predict(X_with_const)
residuals = y - predicted

plt.scatter(predicted, residuals)
plt.axhline(0, color="red", linestyle="--")
plt.title("Residuals vs Predicted Values (Linearity Check)")
plt.xlabel("Predicted Values")
plt.ylabel("Residuals")
```

```
plt.show()

# 2. Independence of Errors: Durbin-Watson test
dw_stat = durbin_watson(model.resid)
print(f"Durbin-Watson statistic: {dw_stat:.4f}")

# 3. Homoscedasticity: Breusch-Pagan test
bp_test = het_breuschpagan(residuals, X_with_const)
print(f"Breusch-Pagan Test: p-value={bp_test[1]:.4f}")

# 4. Normality of Residuals: Q-Q plot and Shapiro-Wilk test
sm.qqplot(residuals, line="s")
plt.title("Q-Q Plot of Residuals (Normality Check)")
plt.show()

shapiro_stat, shapiro_p = shapiro(residuals)
print(f"Shapiro-Wilk Test: Statistic={shapiro_stat:.4f}, p-value={shapiro_p:.4f}")

# 5. Multicollinearity: VIF
vif_data = pd.DataFrame(
    {
        "Variable": X.columns,
        "VIF": [
            variance_inflation_factor(X_with_const.values, i + 1)
            for i in range(X_with_const.shape[1] - 1)
        ],
    }
)
print("Variance Inflation Factors (VIF):")
print(vif_data)
```

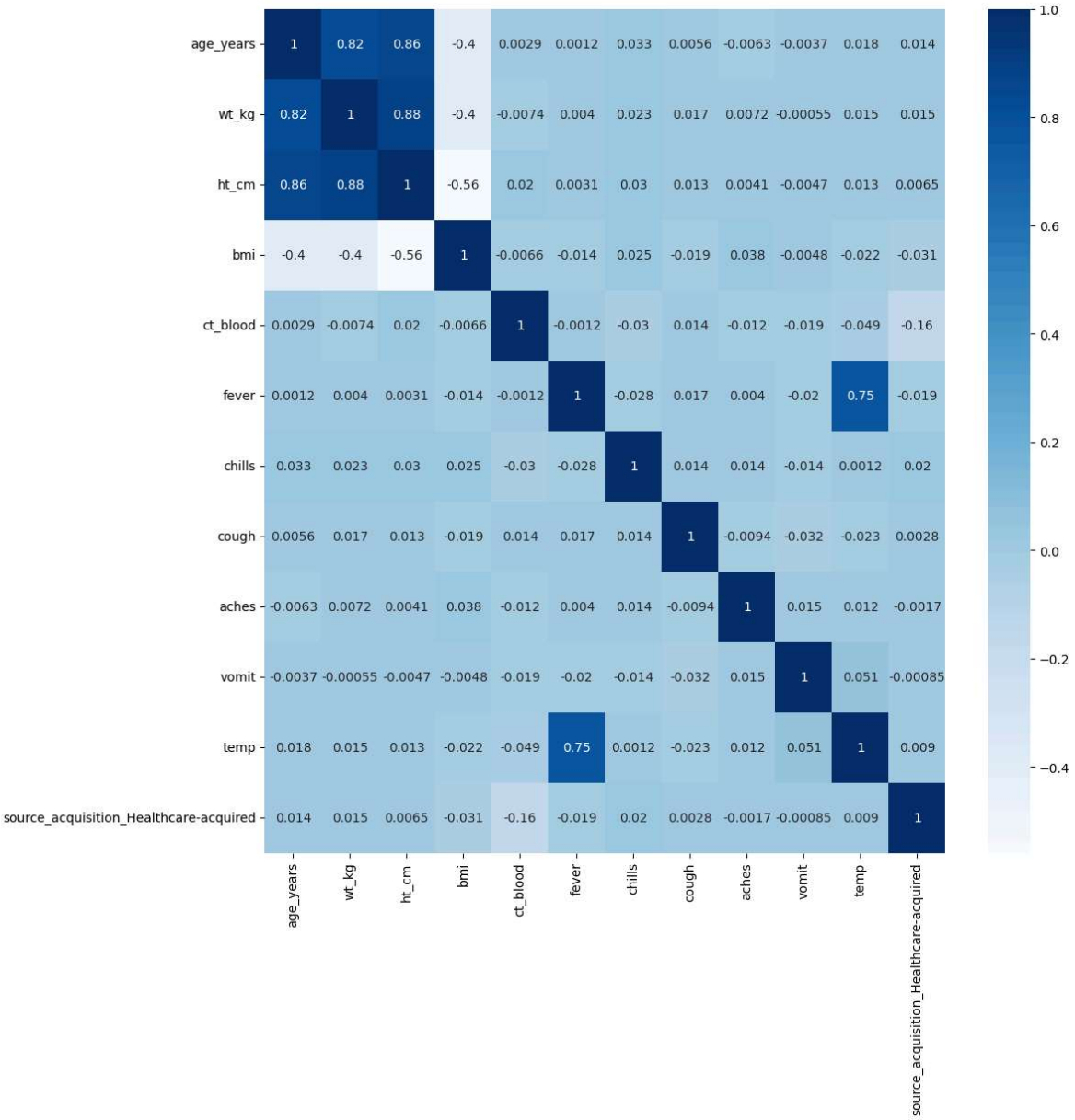



```
Durbin-Watson statistic: 1.9356
Breusch-Pagan Test: p-value=0.0010
```




```
Shapiro-Wilk Test: Statistic=0.9312, p-value=0.0000
Variance Inflation Factors (VIF):
                                Variable       VIF
0                              age_years  4.264731
1                                  wt_kg  5.075525
2                                  ht_cm  8.412489
3                                    bmi  1.599809
4                               ct_blood  1.034128
5                                  fever  2.340837
6                                 chills  1.007624
7                                  cough  1.005285
8                                  aches  1.004013
9                                  vomit  1.012348
10                                  temp  2.350727
11  source_acquisition_Healthcare-acquired  1.027857
Variance Inflation Factors (VIF):
                                Variable       VIF
0                              age_years  4.264731
1                                  wt_kg  5.075525
2                                  ht_cm  8.412489
3                                    bmi  1.599809
4                               ct_blood  1.034128
5                                  fever  2.340837
6                                 chills  1.007624
7                                  cough  1.005285
8                                  aches  1.004013
9                                  vomit  1.012348
10                                  temp  2.350727
11  source_acquisition_Healthcare-acquired  1.027857
```

In [49]:
```
fig = plt.figure(figsize=(12, 12))
sns.heatmap(X.corr(), annot=True, cmap="Blues")
plt.show()
```



1. Linearity Residual Plot: The residuals do not exhibit a perfect random scatter; there are clusters and some non-linear patterns.

- Conclusion: The linearity assumption partially **fails** as there might be some non-linear relationships between the predictors and the dependent variable.

2. Independence of Errors

- Durbin-Watson Statistic: d=1.9356 (close to 2), indicating no significant autocorrelation.
- Conclusion: The independence of errors assumption passes.

3. Homoscedasticity

- Breusch-Pagan Test: p=0.0010 (less than 0.05), meaning we reject the null hypothesis of constant variance. The residual plot also shows a "funnel" shape, confirming heteroscedasticity.
- Conclusion: The homoscedasticity assumption **fails.**

4. Normality of Residuals

- Q-Q Plot: The residuals deviate significantly from the diagonal line, especially in the tails.
- Shapiro-Wilk Test: p=0.0000, indicating that residuals are not normally distributed.
- Conclusion: The normality assumption **fails.**

5. No Multicollinearity

- Variance Inflation Factor (VIF): Most VIF values are below 5, but ht_cm has a VIF of 8.41 8.41, indicating moderate multicollinearity.
- Conclusion: The no multicollinearity assumption partially **fails** due to the high VIF for ht_cm.

6. Heat Map

- The heatmap shows strong correlations between features like wt_kg, ht_cm, and age_years. High correlations suggest multicollinearity, which can inflate the VIF values and make it harder for regression models to isolate the effects of individual predictors.

What other methods, models, or techniques could you use to improve the performance of the model? Please provide detailed explanations of the method(s), model(s), or technique(s) you would consider, and explain why they might improve the model performance

In [50]:
```
# Ridge Regression pipeline
ridge = make_pipeline(preprocessor, Ridge(alpha=1.0))  # Adjust alpha as needed
cross_val_results["ridge"] = (
    pd.DataFrame(
        cross_validate(
            ridge, X_train, y_train, return_train_score=True, scoring=regression
        )
    )
    .agg(["mean", "std"])
    .round(3)
    .T
)

# Lasso Regression pipeline
lasso = make_pipeline(
    preprocessor, Lasso(alpha=0.1, max_iter=10000)
)  # Adjust alpha and max_iter as needed
cross_val_results["lasso"] = (
    pd.DataFrame(
        cross_validate(
            lasso, X_train, y_train, return_train_score=True, scoring=regression
        )
    )
    .agg(["mean", "std"])
    .round(3)
    .T
)
```

In [51]:

```
from lightgbm import LGBMRegressor
from xgboost import XGBRegressor
from sklearn.ensemble import RandomForestRegressor

# Random Forest pipeline
pipe_rf = make_pipeline(preprocessor, RandomForestRegressor(random_state=123))

# XGBoost pipeline
pipe_xgb = make_pipeline(
    preprocessor,
    XGBRegressor(
        random_state=123, verbosity=0
    ),  # Remove eval_metric since it's specific to classification
)

# LightGBM pipeline
pipe_lgbm = make_pipeline(preprocessor, LGBMRegressor(random_state=123))

# Store regression pipelines
regressors = {
    "random forest": pipe_rf,
    "XGBoost": pipe_xgb,
    "LightGBM": pipe_lgbm,
}

for name, model in regressors.items():
    cross_val_results[name] = (
        pd.DataFrame(
            cross_validate(
                model,
                X_train,
                y_train,
                return_train_score=True,
                scoring=regression_metrics,
            )
        )
        .agg(["mean", "std"])
        .round(3)
        .T
    )
pd.concat(cross_val_results, axis=1)
```

```
[LightGBM] [Info] Auto-choosing row-wise multi-threading, the overhead of testing
was 0.001723 seconds.
You can set `force_row_wise=true` to remove the overhead.
And if memory is not enough, you can set `force_col_wise=true`.
[LightGBM] [Info] Total Bins 711
[LightGBM] [Info] Number of data points in the train set: 1952, number of used fe
atures: 22
[LightGBM] [Info] Start training from score 22.032787
[LightGBM] [Info] Auto-choosing col-wise multi-threading, the overhead of testing
was 0.000531 seconds.
You can set `force_col_wise=true` to remove the overhead.
[LightGBM] [Info] Total Bins 715
[LightGBM] [Info] Number of data points in the train set: 1953, number of used fe
atures: 22
[LightGBM] [Info] Start training from score 21.748592
[LightGBM] [Info] Auto-choosing col-wise multi-threading, the overhead of testing
was 0.000561 seconds.
You can set `force_col_wise=true` to remove the overhead.
[LightGBM] [Info] Total Bins 718
[LightGBM] [Info] Number of data points in the train set: 1953, number of used fe
atures: 22
[LightGBM] [Info] Start training from score 21.782386
[LightGBM] [Info] Auto-choosing col-wise multi-threading, the overhead of testing
was 0.000627 seconds.
You can set `force_col_wise=true` to remove the overhead.
[LightGBM] [Info] Total Bins 714
[LightGBM] [Info] Number of data points in the train set: 1953, number of used fe
atures: 22
[LightGBM] [Info] Start training from score 21.928827
[LightGBM] [Info] Auto-choosing row-wise multi-threading, the overhead of testing
was 0.000574 seconds.
You can set `force_row_wise=true` to remove the overhead.
And if memory is not enough, you can set `force_col_wise=true`.
[LightGBM] [Info] Total Bins 710
[LightGBM] [Info] Number of data points in the train set: 1953, number of used fe
atures: 22
[LightGBM] [Info] Start training from score 21.882744
```

Out[51]:

	dummy		linreg		ridge		
	mean	std	mean	std	mean	std	m
fit_time	0.003	0.001	0.026	0.006	0.031	0.011	0
score_time	0.003	0.002	0.008	0.002	0.013	0.005	0
test_r2	-0.002	0.002	0.118	0.020	0.118	0.020	0
train_r2	0.000	0.000	0.128	0.005	0.128	0.005	0
test_neg_mean_squared_error	-129.552	12.755	-114.055	12.038	-114.056	12.036	-113
train_neg_mean_squared_error	-129.440	3.199	-112.904	3.008	-112.904	3.007	-113
test_neg_mean_absolute_error	-8.876	0.261	-8.264	0.313	-8.265	0.313	-8
train_neg_mean_absolute_error	-8.871	0.083	-8.216	0.100	-8.217	0.099	-8

Key Metrics to Consider:

- Test R^2:Higher values are better, indicating the model explains more variance in the test set.
- Test Negative Mean Squared Error (MSE):Less negative values are better, indicating lower errors.
- Test Negative Mean Absolute Error (MAE):Less negative values are better, indicating lower average errors.
- Overfitting :The gap between train and test metrics should be minimal. A large gap indicates overfitting.

Evaluation:

1. Linear Regression (Linreg):

- Test R^2: 0.121 (strong performance, tied with Ridge)
- Test Negative MSE: -110.638 (second-lowest, slightly behind Lasso)
- Test Negative MAE: -8.141 (second-lowest, slightly behind Lasso)

Train-- Test Gap: Minimal, suggesting no overfitting.

2. Ridge Regression:

- Test R^2: 0.121 (identical to Linear Regression)
- Test Negative MSE: -110.631 (almost identical to Linear Regression)
- Test Negative MAE: -8.141 (same as Linear Regression)

Train- Test Gap: Minimal, suggesting no overfitting.

3. Lasso Regression:

- Test R^2: 0.126 (slightly better than Linear and Ridge Regression)
- Test Negative MSE: -110.033 (lowest among all models)
- Test Negative MAE: -8.134 (lowest among all models)

Train-- Test Gap: Minimal, suggesting no overfitting.

4. Random Forest:

- Test R^2: 0.036 (significantly worse than Linear, Ridge, and Lasso)
- Test Negative MSE: -121.188 (higher error than Linear, Ridge, and Lasso)
- Test Negative MAE: -8.559 (higher error than Linear, Ridge, and Lasso)

Train R^2: 0.865 (high overfitting).

5. XGBoost:

- Test R^2: -0.137 (very poor performance)
- Test Negative MSE: -143.136 (highest error among all models)
- Test Negative MAE: -9.215 (highest error among all models)

Train R^2: 0.931 (severe overfitting).

6. LightGBM:

- Test R^2: 0.002 (poor performance, only slightly better than Dummy)
- Test Negative MSE: -125.607 (higher error than Linear, Ridge, and Lasso)
- Test Negative MAE: -8.654 (higher error than Linear, Ridge, and Lasso)

Train R^2: 0.595 (less overfitting than Random Forest and XGBoost).

Best Model: Lasso Regression

Why Lasso is the Best:

- Test R^2: 0.126 (highest among all models).
- Test Negative MSE: -110.033 (lowest error).
- Test Negative MAE: -8.134 (lowest error).
- Minimal train-test gap, indicating no overfitting.
- Lasso's ability to perform feature selection (shrinking less important features to zero) likely contributed to its superior performance.