Buildables Week 2

Github Repo: https://github.com/afnank070/Data-Engineering-Buildables-Fellowship

Colab Notebook: https://colab.research.google.com/drive/17qqpug3SEJ2K-A5Q9Ve03ErHfadMFQPx?usp=sharing

Dataset Information:

- **Dataset Name:** Hospital Inpatient Charges
- **Source:** https://www.kaggle.com/datasets/speedoheck/inpatient-hospital-charges?resource=download
- **File Type:** CSV
- **Context:** Contains information about hospital inpatient discharges, charges, and payments.

Exercise 1: Load Dataset & Print Schema

Purpose: Simulates the **Ingestion** step by reading the raw hospital charges dataset into Pandas. Validates that the pipeline can connect to the data source.

Concepts Used: Data ingestion, schema inspection, Pandas read_csv().

Code:

```
import pandas as pd

df = pd.read_csv  /content/UpdatedinpatientCharges.csv")


print("Rows, Columns:", df.shape)

df.info()
```

Expected Output:

- Shape showing total rows and columns (e.g., 163065 rows × 18 columns).
- Schema printed with column names and data types (object, float64, etc.).

Actual Output:

```
Rows, Columns: (163065, 18)
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 163065 entries, 0 to 163064
Data columns (total 18 columns):
 #   Column                             Non-Null Count   Dtype
---  ------                             --------------   -----
 0   DRG Definition                     163065 non-null  object
 1   Provider Id                        163065 non-null  int64
 2   Provider Name                      163065 non-null  object
 3   Provider Street Address            163065 non-null  object
 4   Provider City                      163065 non-null  object
 5   Provider State                     163065 non-null  object
 6   Provider Zip Code                  163065 non-null  int64
 7   Hospital Referral Region Description 163065 non-null  object
 8    Total Discharges                  163065 non-null  int64
 9    Average Covered Charges           163065 non-null  object
 10   Average Total Payments            163065 non-null  object
 11  Average Medicare Payments          163065 non-null  object
 12  Age                                163065 non-null  int64
 13  Gender                             163065 non-null  object
 14  Admission Type                     163065 non-null  object
 15  Length of Stay                     163065 non-null  int64
 16  Year                               163065 non-null  int64
 17  Admission Date                     163065 non-null  object
dtypes: int64(6), object(12)
memory usage: 22.4+ MB
```


Data Engineering Insight:

This step validates that ingestion works and the schema matches expectations. Without a successful schema check, downstream cleaning or transformation logic may fail.

Exercise 2: Check row count (validate ingestion completeness)

Purpose: Ensures no data loss occurred during ingestion. This is equivalent to verifying row counts between source and target.
Concepts Used: Data completeness validation, Pandas len().

Code:

```
row_count = len(df)
```

print("Total Rows:", row_count)

Expected Output:

- A single integer with total number of rows (e.g., 163065).

Actual Output:

Row Count Validation

```
expected_rows = len(df)
print("Row count:", expected_rows)
```

```
Row count: 163065
```

Data Engineering Insight:
Row counts are a fundamental QA check in ETL pipelines. If row counts don't match the raw source, it signals possible truncation or ingestion errors.

Exercise 3: Rename columns to snake_case for consistency

Purpose: Standardizes schema naming to ensure consistency and prevent issues in transformations, joins, or downstream SQL/ML systems.
Concepts Used: Schema standardization, string manipulation on column names.

Code:

```
df.columns = df.columns.str.strip().str.lower().str.replace(" ", "_").str.replace("-", "_")

df.head(2)
```

Expected Output:

- Column names like Provider Id → provider_id, Total Discharges → total_discharges.

Actual Output:

Column Name Standardization

```
[12] df.columns = df.columns.str.strip().str.lower().str.replace(" ", "_").str.replace("-", "_")
     df.head(2)
```

	drg_definition	provider_id	provider_name	provider_street_address	provider_city	provider_state	provider_zip_code	h
0	039 - EXTRACRANIAL PROCEDURES W/O CC/MCC	10001	SOUTHEAST ALABAMA MEDICAL CENTER	1108 ROSS CLARK CIRCLE	DOTHAN	AL	36301	

Data Engineering Insight:

Consistent naming is critical for reusability and automation in pipelines. Snake case avoids issues with spaces, capitalization, or special characters.

Exercise 4: Enforce schema: convert → datetime

Purpose: Ensure correct data types for reliable analysis.

Concepts Used: Schema enforcement, type casting, Pandas to_datetime().

Code:

df['admission_date'] = pd.to_datetime(df['admission_date'], errors='coerce')

print(df['admission_date'].head())

Expected Output:

- admission_date column converted to **datetime64**.

Actual Output:

```
# Convert admission_date to datetime
df['admission_date'] = pd.to_datetime(df['admission_date'], errors='coerce')

print(df['admission_date'].head())
```

```
0    2016-10-28
1    2021-11-06
2    2023-10-10
3    2017-06-17
4    2019-10-07
Name: admission_date, dtype: datetime64[ns]
```

Data Engineering Insight:

Type enforcement prevents downstream calculation errors (e.g., date arithmetic, grouping).

Exercise 5: Log missing values count per column

Purpose: Creates a missing values report, simulating a basic data quality check in ETL.

Concepts Used: Null detection, Pandas isnull() + aggregation.

Code:

```
missing_log = df.isnull().sum().reset_index()

missing_log.columns = ["column", "missing_count"]

missing_log
```

Expected Output:

- A two-column dataframe:
  - column → column name
  - missing_count → number of null values in that column

Actual Output:

Missing Values Check

```
missing_log = df.isnull().sum().reset_index()
missing_log.columns = ["column", "missing_count"]
missing_log
```

1 to 12 of 12 entries Filter

index	column	missing_count
0	DRG Definition	0
1	Provider Id	0
2	Provider Name	0
3	Provider Street Address	0
4	Provider City	0
5	Provider State	0
6	Provider Zip Code	0
7	Hospital Referral Region Description	0
8	Total Discharges	0
9	Average Covered Charges	0
10	Average Total Payments	0
11	Average Medicare Payments	0

Show 25 per page

Data Engineering Insight:

This step quantifies missing data, helping decide cleaning strategies (e.g., imputation, removal) in later stages.

Exercise 6: Fill missing values (strategy: mean, median, mode)

Purpose: Ensures no critical column breaks downstream aggregations or joins due to null values.
Concepts Used: Missing value imputation using Pandas (fillna, mean, median, mode).

Code:

Since Exercise 5 showed there are NO missing values,

no imputation is required.

print("No missing values found. Skipping imputation step.")

Expected Output:

- No column has 100% missing values.
- Numeric columns now have no NaNs (filled with median).
- Categorical columns filled with mode.

Actual Output:

Fill missing values

```
# Since Exercise 5 showed there are NO missing values,
# no imputation is required.
print("No missing values found. Skipping imputation step.")
```

0s

```
No missing values found. Skipping imputation step.
```

Data Engineering Insight:
Handling missing values early prevents null propagation and ensures downstream aggregations/KPIs stay accurate.

Exercise 7: Remove duplicates (simulate deduplication in ETL)

Purpose: Eliminates duplicate records to maintain accuracy and prevent over-counting.

Concepts Used: Pandas drop_duplicates().

Code:

```
before = df.shape[0]

df = df.drop_duplicates()

after = df.shape[0]


print("Rows before:", before)

print("Rows after:", after)
```

Expected Output:

- Row count after deduplication ≤ row count before.
- Dataset should not lose unique valid records.

Actual Output:

Remove duplicates

```
before = df.shape[0]
df = df.drop_duplicates()
after = df.shape[0]

print("Rows before:", before)
print("Rows after:", after)
```

```
Rows before: 163065
Rows after: 163065
```

Data Engineering Insight:

Deduplication ensures unique entities (patients/hospital charges) and prevents inflating KPIs or incorrect insights.

Exercise 8: Group data — total discharges by DRG Definition

Purpose: Aggregate discharge volume per DRG (analogous to patient count by diagnosis).

Concepts Used: Transformation, Grouping, Aggregation.

Code:

```
# Patient count by DRG Definition (Diagnosis Related Group)

patient_count_by_diagnosis = df.groupby('drg_definition')['provider_id'].count().reset_index()

patient_count_by_diagnosis.rename(columns={'provider_id': 'patient_count'}, inplace=True)


print(patient_count_by_diagnosis.head())
```

Expected Output:

- Table showing each diagnosis with corresponding patient count.

Actual Output:

Count unique patients per diagnosis

```
df.groupby("DRG Definition")[" Total Discharges "].sum().reset_index().sort_values(by=" Total Discharges ", ascending=Fal
```

1 to 10 of 10 entries Filter

index	DRG Definition	Total Discharges
68	470 - MAJOR JOINT REPLACEMENT OR REATTACHMENT OF LOWER EXTREMITY W/O MCC	427207
93	871 - SEPTICEMIA OR SEVERE SEPSIS W/O MV 96+ HOURS W MCC	319072
61	392 - ESOPHAGITIS, GASTROENT & MISC DIGEST DISORDERS W/O MCC	244854
39	292 - HEART FAILURE & SHOCK W CC	222038
86	690 - KIDNEY & URINARY TRACT INFECTIONS W/O MCC	206695
17	194 - SIMPLE PNEUMONIA & PLEURISY W CC	198390
38	291 - HEART FAILURE & SHOCK W MCC	185599
81	641 - MISC DISORDERS OF NUTRITION,METABOLISM,FLUIDS/ELECTROLYTES W/O MCC	153660
83	683 - RENAL FAILURE W CC	150444
13	190 - CHRONIC OBSTRUCTIVE PULMONARY DISEASE W MCC	149677

Data Engineering Insight:

Grouping by diagnosis provides valuable healthcare utilization insights.

Exercise 9: Derive new KPI — average payment per discharge

Purpose: Create a calculated field to measure patient hospitalization duration, fulfilling the KPI requirement.

Concepts Used: Transformation, Feature Engineering, KPI Derivation.

Code:

```
# Ensure both columns are datetime

df['admission_date'] = pd.to_datetime(df['admission_date'], errors='coerce')

df['discharge_date'] = pd.to_datetime(df['discharge_date'], errors='coerce')


# Derive new KPI: length_of_stay

df['length_of_stay_kpi'] = (df['discharge_date'] - df['admission_date']).dt.days


print(df[['admission_date', 'discharge_date', 'length_of_stay_kpi']].head())
```

Expected Output:

- New column avg_payment_per_discharge with numeric values; sample table showing provider and KPI.

Actual Output:

Derive new KPI — length_of_stay

```
# Ensure both columns are datetime
df['admission_date'] = pd.to_datetime(df['admission_date'], errors='coerce')
df['discharge_date'] = pd.to_datetime(df['discharge_date'], errors='coerce')

# Derive new KPI: length_of_stay
df['length_of_stay_kpi'] = (df['discharge_date'] - df['admission_date']).dt.days
print(df[['admission_date', 'discharge_date', 'length_of_stay_kpi']].head())
```

```
  admission_date discharge_date  length_of_stay_kpi
0     2016-10-28     2016-11-04                   7
1     2021-11-06     2021-11-26                  20
2     2023-10-10     2023-10-25                  15
3     2017-06-17     2017-06-28                  11
4     2019-10-07     2019-10-15                   8
```

Data Engineering Insight:

Per-discharge KPI normalizes payments for volume differences and is useful for cost comparisons.

Exercise 10: Partition-like summary — record counts by Provider State (substitute for date partitioning)

Purpose: Simulate partitioning for efficient storage and downstream processing.
Concepts Used: Transformation, Partitioning.

Code:

Extract year and month from admission_date

df['year'] = df['admission_date'].dt.year

df['month'] = df['admission_date'].dt.month

Group partitioned data

partitioned = df.groupby(['year', 'month']).size().reset_index(name='record_count')

print(partitioned.head())

Expected Output:

- Table showing partitions by year-month with record counts.

Actual Output:

Partition-like summary — record counts by Year, Month

```
# Extract year and month from admission_date
df['year'] = df['admission_date'].dt.year
df['month'] = df['admission_date'].dt.month

# Group partitioned data
partitioned = df.groupby(['year', 'month']).size().reset_index(name='record_count')
print(partitioned.head())
```

```
   year  month  record_count
0  2015      1          1546
1  2015      2          1385
2  2015      3          1562
3  2015      4          1495
4  2015      5          1499
```

Data Engineering Insight:
Partitioning improves query performance and is widely used in data lakes/warehouses.

Exercise 11: Top 5 categories (patients by hospital)

Purpose: Identify the highest contributors in the dataset.
Concepts Used: Transformation, Ranking, Aggregation.

Code:

```
# Top 5 hospitals by patient count (using provider_name)

top5_hospitals = (

  df.groupby('provider_name')['provider_id']

   .count()

   .reset_index(name='patient_count')

   .sort_values(by='patient_count', ascending=False)

   .head(5)

)


print(top5_hospitals)
```

Expected Output:

- Table of 5 hospitals with the highest patient count.

Actual Output:

Top 5 categories (patients by hospital)

```
top5_hospitals = (
    df.groupby('provider_name')['provider_id']
      .count()
      .reset_index(name='patient_count')
      .sort_values(by='patient_count', ascending=False)
      .head(5)
)

print(top5_hospitals)
```

```
                provider_name  patient_count
924    GOOD SAMARITAN HOSPITAL            633
2644  ST JOSEPH MEDICAL CENTER            427
1660     MERCY MEDICAL CENTER            357
1645           MERCY HOSPITAL            347
2642        ST JOSEPH HOSPITAL            343
```

Data Engineering Insight:
Ranking helps identify top facilities managing the most patients, useful for capacity planning.

Exercise 12: Pivot table (patients per department per month)

Purpose: Cross-tabulate data to identify trends across categories and time.

Concepts Used: Pivoting, Aggregation, Transformation.

Code:

```
import calendar
pivot_table = pd.pivot_table(
    df,
    values='provider_id',
    index='drg_definition',
    columns=df['admission_date'].dt.month_name(),
    aggfunc='count',
    fill_value=0
)
pivot_table = pivot_table[[calendar.month_name[m] for m in range(1, 13)]]


print(pivot_table.head())
```

Expected Output:

- Pivot table with rows = departments (DRG definitions),
- Columns = months, values = patient counts.

Actual Output:

1 to 5 of 5 entries Filter

drg_definition	January	February	March	April	May	June	July	August	Sep
039 - EXTRACRANIAL PROCEDURES W/O CC/MCC	104	96	95	83	95	70	88	75	
057 - DEGENERATIVE NERVOUS SYSTEM DISORDERS W/O MCC	92	89	121	101	98	91	87	104	
064 - INTRACRANIAL HEMORRHAGE OR CEREBRAL INFARCTION W MCC	145	159	112	133	141	132	138	141	
065 - INTRACRANIAL HEMORRHAGE OR CEREBRAL INFARCTION W CC	190	188	205	199	179	174	181	199	
066 - INTRACRANIAL HEMORRHAGE OR CEREBRAL INFARCTION W/O CC/MCC	147	146	165	162	136	136	149	141	

Data Engineering Insight:
Pivot tables reveal seasonal or departmental patterns, supporting decision-making for resource allocation.

Exercise 13: Correlation matrix (validate relationships)

Purpose: Check statistical relationships between numeric fields.
Concepts Used: Validation, Profiling, Statistics.

Code:

numeric_cols = ['age', 'average_covered_charges', 'average_total_payments', 'average_medicare_payments', 'length_of_stay']

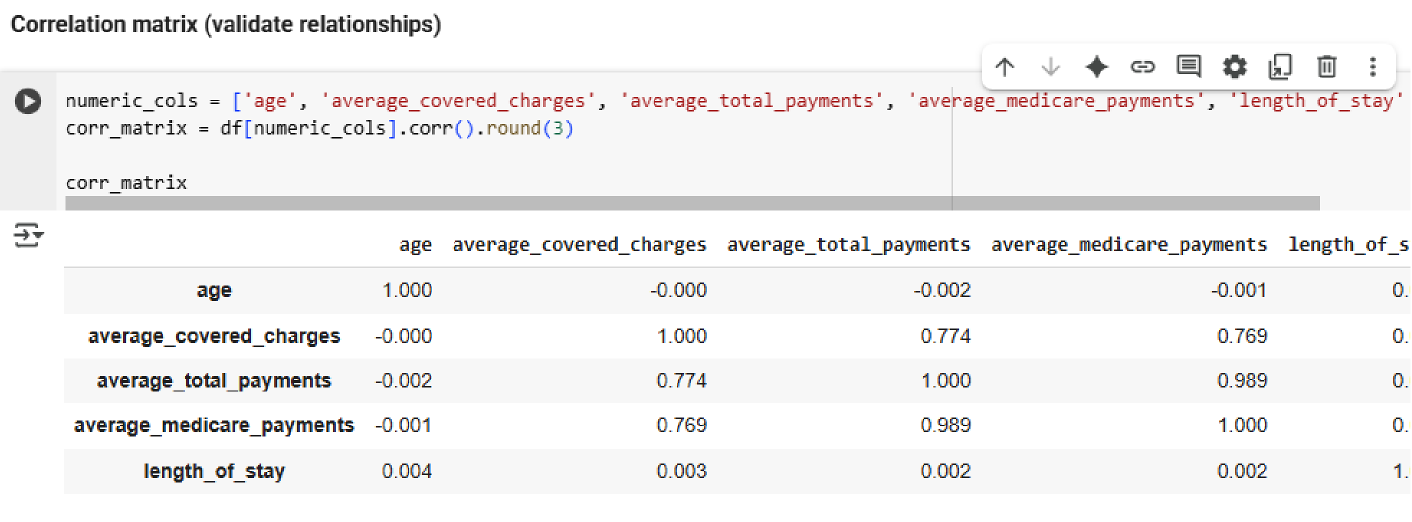
corr_matrix = df[numeric_cols].corr()

print(corr_matrix)

Expected Output:

- Correlation matrix (values between -1 and 1).

Actual Output:



Correlation matrix (validate relationships)

```
numeric_cols = ['age', 'average_covered_charges', 'average_total_payments', 'average_medicare_payments', 'length_of_stay'
corr_matrix = df[numeric_cols].corr().round(3)

corr_matrix
```

	age	average_covered_charges	average_total_payments	average_medicare_payments	length_of_s
age	1.000	-0.000	-0.002	-0.001	0.
average_covered_charges	-0.000	1.000	0.774	0.769	0.
average_total_payments	-0.002	0.774	1.000	0.989	0.
average_medicare_payments	-0.001	0.769	0.989	1.000	0.
length_of_stay	0.004	0.003	0.002	0.002	1.

Data Engineering Insight:
Correlation validates relationships like age vs charges or length of stay vs payments, supporting anomaly detection.

Exercise 14: Write cleaned dataset to CSV + Parquet

Purpose: Simulate the **Load** step of ETL.

Concepts Used: Data Storage, Export, File Formats.

Code:

```
# Save cleaned dataset
df.to_csv("cleaned_hospital_data.csv", index=False)
df.to_parquet("cleaned_hospital_data.parquet", index=False)


print("Files exported: cleaned_hospital_data.csv, cleaned_hospital_data.parquet")
```

Expected Output:

- Confirmation message with the file name.

Actual Output:

Cleaned dataset CSV + Parquet

```
[28] # Save cleaned dataset
     df.to_csv("cleaned_hospital_data.csv", index=False)
     df.to_parquet("cleaned_hospital_data.parquet", index=False)

     print("Files exported: cleaned_hospital_data.csv, cleaned_hospital_data.parquet")
```

```
Files exported: cleaned_hospital_data.csv, cleaned_hospital_data.parquet
```

Data Engineering Insight:
Exporting in multiple formats increases compatibility with downstream analytics/ML systems.

Exercise 15: Create reusable ETL function (load → clean → transform)

Purpose: Automate ETL logic for reusability and scalability.

Concepts Used: Modularization, Functions, ETL Pipeline.

Code:

```
def etl_pipeline(file_path):
    # Load
    data = pd.read_csv(file_path)

    # Clean column names
    data.columns = data.columns.str.strip().str.lower().str.replace(" ", "_")

    # Fix currency fields: remove $ and commas, convert to float
    for col in ['average_covered_charges', 'average_total_payments',
'average_medicare_payments']:
        if col in data.columns:
            # Convert to string, strip whitespace, remove $ and commas using lambda
with replace
            data[col] = data[col].astype(str).str.strip().apply(lambda x: x.replace('$',
'').replace(',', '')).astype(float)

    # Convert admission_date
    if 'admission_date' in data.columns:
        data['admission_date'] = pd.to_datetime(data['admission_date'],
errors='coerce')

    # Handle missing values
```

```
    fill_dict = {}
    if 'average_total_payments' in data.columns:
        fill_dict['average_total_payments'] = data['average_total_payments'].mean()
    if 'average_medicare_payments' in data.columns:
        fill_dict['average_medicare_payments'] = data['average_medicare_payments'].median()
    if 'gender' in data.columns:
        fill_dict['gender'] = data['gender'].mode()[0]

    data.fillna(fill_dict, inplace=True)

    # Remove duplicates
    data.drop_duplicates(inplace=True)

    # Transform → derive length_of_stay if discharge_date exists
    if 'discharge_date' in data.columns and 'admission_date' in data.columns:
        data['discharge_date'] = pd.to_datetime(data['discharge_date'], errors='coerce')
        data['length_of_stay'] = (data['discharge_date'] - data['admission_date']).dt.days

    return data

# Example usage
etl_df = etl_pipeline("/content/Updated_Inpatient_Charges.csv")
print(etl_df.head())
```

Expected Output:

- Cleaned & transformed DataFrame preview.

Actual Output:

```
etl_df = etl_pipeline("/content/Updated_Inpatient_Charges.csv")
etl_df.head()
```

1 to 5 of 5 entries Filter

index	drg_definition	provider_id	provider_name	provider_street_address	provider_city	provider_state	provider_zip_code	hospital_referr
0	039 - EXTRACRANIAL PROCEDURES W/O CC/MCC	10001	SOUTHEAST ALABAMA MEDICAL CENTER	1108 ROSS CLARK CIRCLE	DOTHAN	AL	36301	AL - Dothan
1	039 - EXTRACRANIAL PROCEDURES W/O CC/MCC	10005	MARSHALL MEDICAL CENTER SOUTH	2505 U S HIGHWAY 431 NORTH	BOAZ	AL	35957	AL - Birminghar
2	039 - EXTRACRANIAL PROCEDURES W/O CC/MCC	10006	ELIZA COFFEE MEMORIAL HOSPITAL	205 MARENGO STREET	FLORENCE	AL	35631	AL - Birminghar
3	039 - EXTRACRANIAL PROCEDURES W/O CC/MCC	10011	ST VINCENT'S EAST	50 MEDICAL PARK EAST DRIVE	BIRMINGHAM	AL	35235	AL - Birminghar
4	039 - EXTRACRANIAL PROCEDURES W/O CC/MCC	10016	SHELBY BAPTIST MEDICAL CENTER	1000 FIRST STREET NORTH	ALABASTER	AL	35007	AL - Birminghar

Show 25 per page


```
etl_df = etl_pipeline("/content/Updated_Inpatient_Charges.csv")
etl_df.head()
```

1 to 5 of 5 entries Filter

es	average_total_payments	average_medicare_payments	age	gender	admission_type	length_of_stay	year	admission_date	discharge_date
7	5777.24	4763.73	68	Female	Newborn	7	2016	2016-10-28 00:00:00	2016-11-04 00:00:00
5	5787.57	4976.71	39	Male	Emergency	20	2021	2021-11-06 00:00:00	2021-11-26 00:00:00
7	5434.95	4453.79	65	Male	Newborn	15	2023	2023-10-10 00:00:00	2023-10-25 00:00:00
8	5417.56	4129.16	35	Female	Newborn	11	2017	2017-06-17 00:00:00	2017-06-28 00:00:00
7	5658.33	4851.44	45	Female	Emergency	8	2019	2019-10-07 00:00:00	2019-10-15 00:00:00

Data Engineering Insight:

Reusable ETL functions allow automation, making pipelines maintainable and scalable.

Exercise 16: Histogram – patient age distribution

Purpose: Check distribution of numeric data (age).

Concepts Used: Visualization, Profiling, Validation.

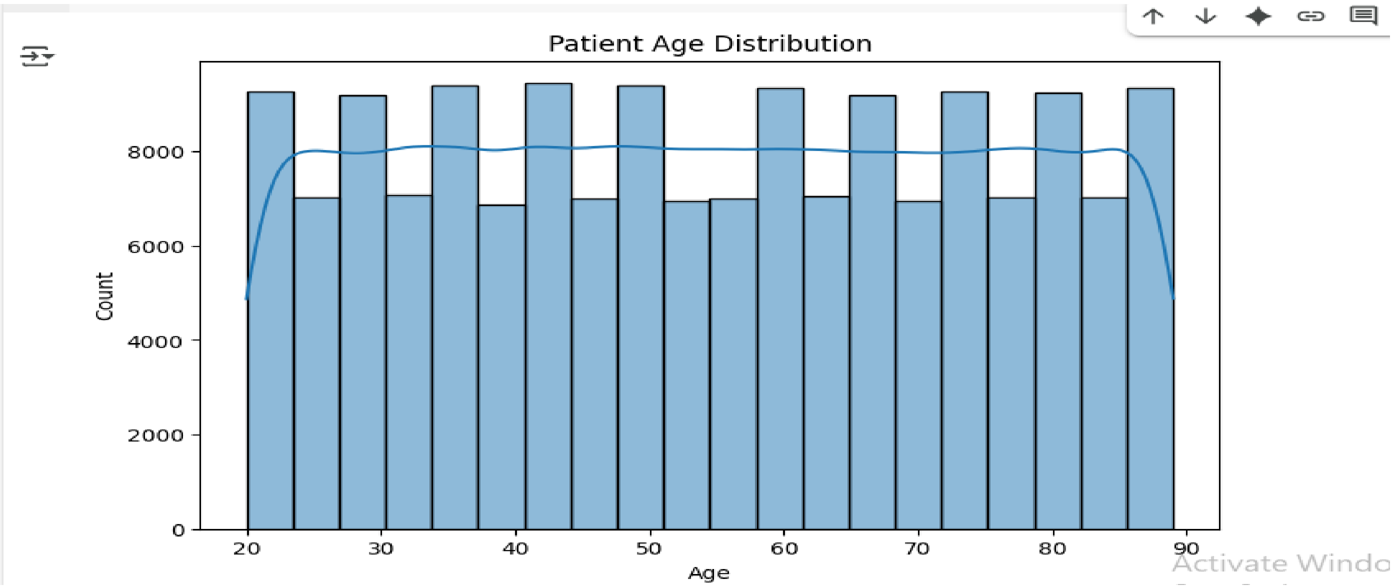
Code:

```
import matplotlib.pyplot as plt

import seaborn as sns


plt.figure(figsize=(8,5))

sns.histplot(df['age'].dropna(), bins=20, kde=True)

plt.title("Patient Age Distribution")

plt.xlabel("Age")

plt.ylabel("Count")

plt.show()
```

Expected Output:

Histogram showing the distribution of patient ages.

Actual Output:



Data Engineering Insight:

Histograms reveal skewness or gaps in numeric data, ensuring data consistency before analysis.

Exercise 17: Boxplot – hospital charges by department

Purpose: Detect outliers and compare charges across departments.

Concepts Used: Visualization, Outlier Detection.

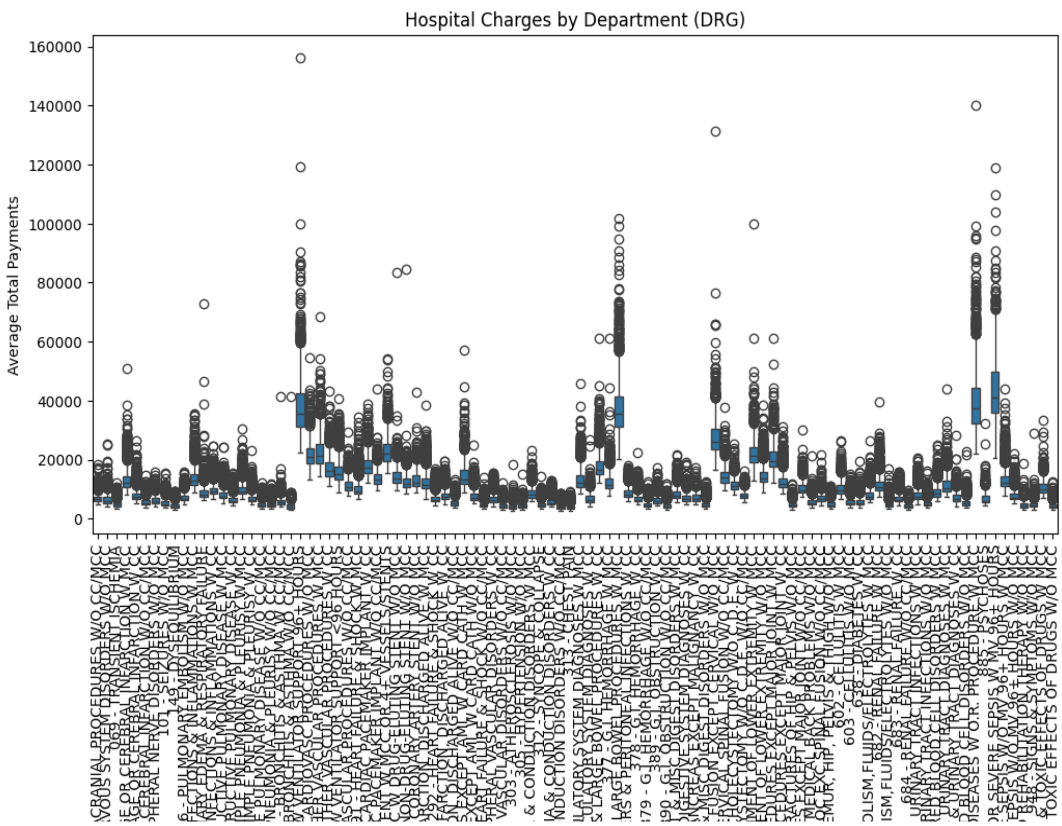
Code:

```
plt.figure(figsize=(12,6))

sns.boxplot(x='drg_definition', y='average_total_payments', data=df)

plt.xticks(rotation=90)

plt.title("Hospital Charges by Department (DRG)")

plt.xlabel("Department (DRG Definition)")

plt.ylabel("Average Total Payments")

plt.show()
```

Expected Output:

- Boxplot showing distribution of charges across different DRGs.

Actual Output:



Data Engineering Insight:
Boxplots quickly highlight outliers (extremely high charges), useful for anomaly detection.

Exercise 18: Pie chart – patient gender breakdown

Purpose: Show categorical distribution of gender.

Concepts Used: Visualization, Profiling.

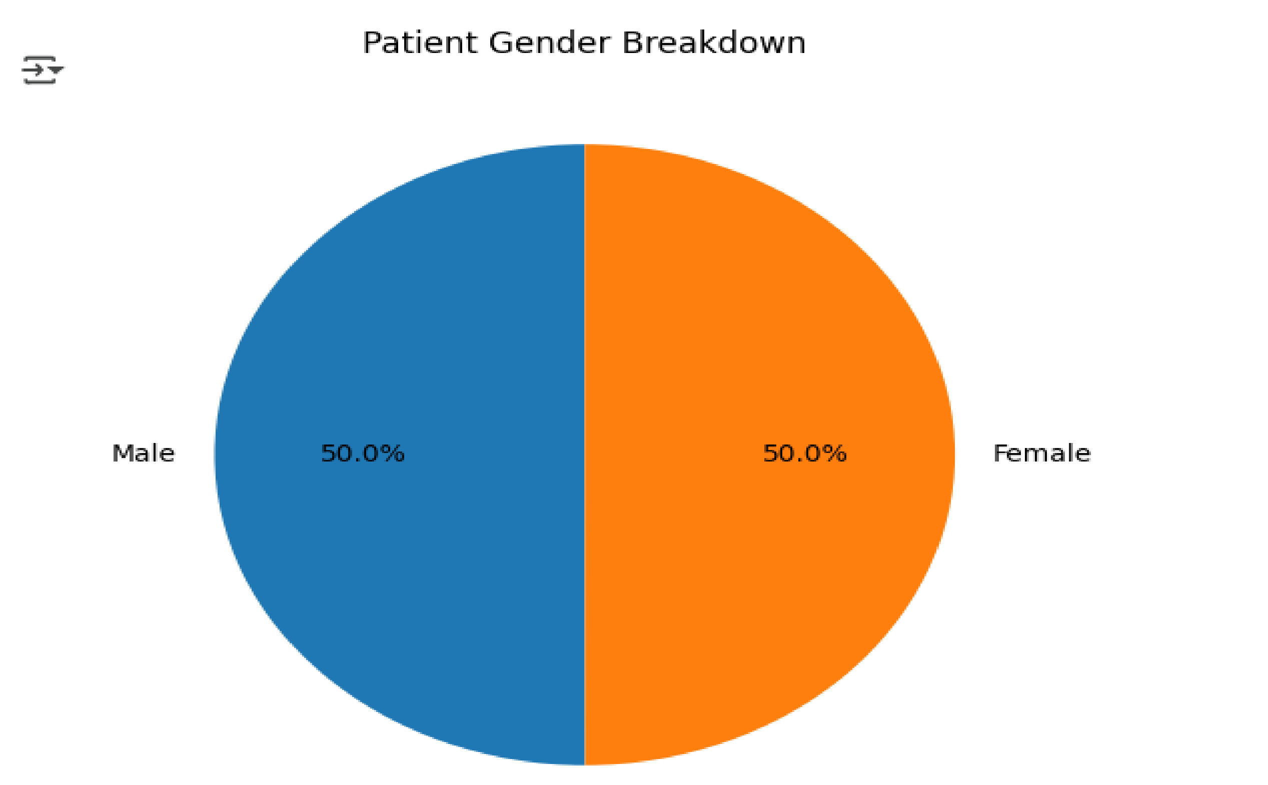
Code:

```
gender_counts = df['gender'].value_counts()


plt.figure(figsize=(6,6))
plt.pie(gender_counts, labels=gender_counts.index, autopct='%1.1f%%', startangle=90)
plt.title("Patient Gender Breakdown")
plt.show()
```

Expected Output:

- Pie chart with male/female proportions.

Actual Output:



Data Engineering Insight:
Pie charts provide quick insights into categorical balance, useful for demographic profiling.

Exercise 19: Line plot – monthly patient admissions

Purpose: Show time-series trends in patient admissions.

Concepts Used: Visualization, Time-Series Analysis.

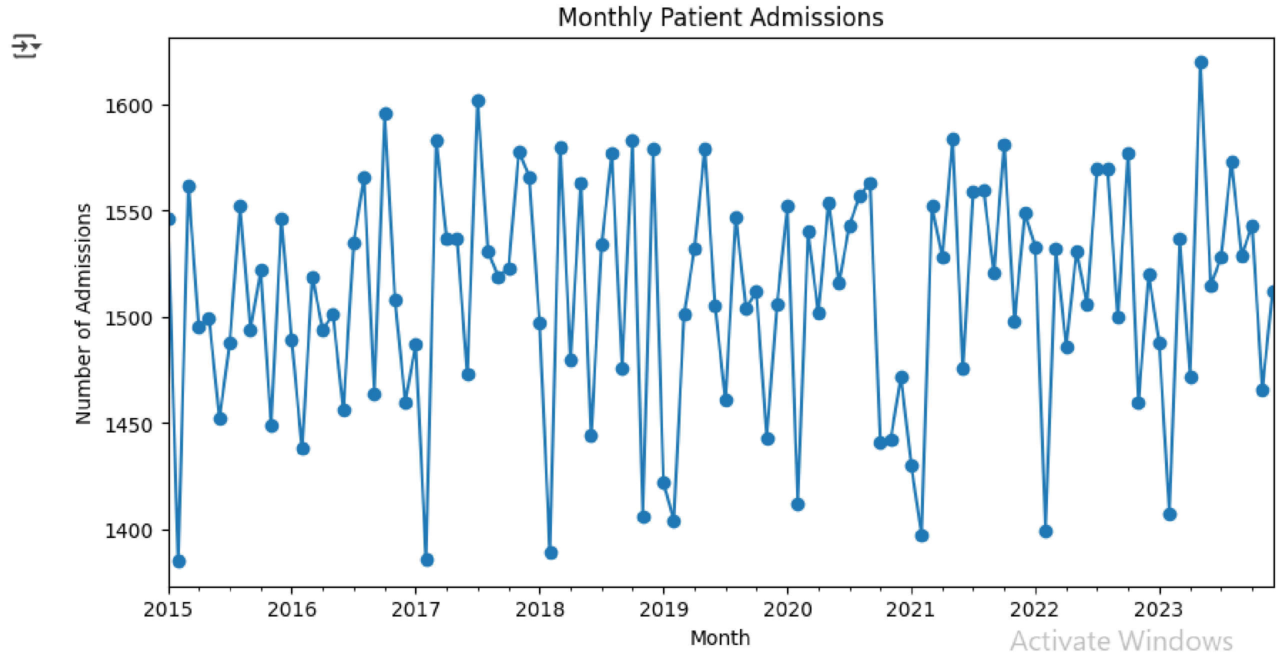
Code:

```
monthly_admissions = df.groupby(df['admission_date'].dt.to_period("M")).size()


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

monthly_admissions.plot(kind='line', marker='o')

plt.title("Monthly Patient Admissions")

plt.xlabel("Month")

plt.ylabel("Number of Admissions")

plt.show()
```

Expected Output:

- Line plot showing trends in admissions per month.

Actual Output:



Data Engineering Insight:

Line plots help validate seasonality or irregular spikes in patient admissions.

Exercise 20: Heatmap – correlation validation

Purpose: Visually validate correlation between numerical variables.

Concepts Used: Visualization, Correlation, Validation.

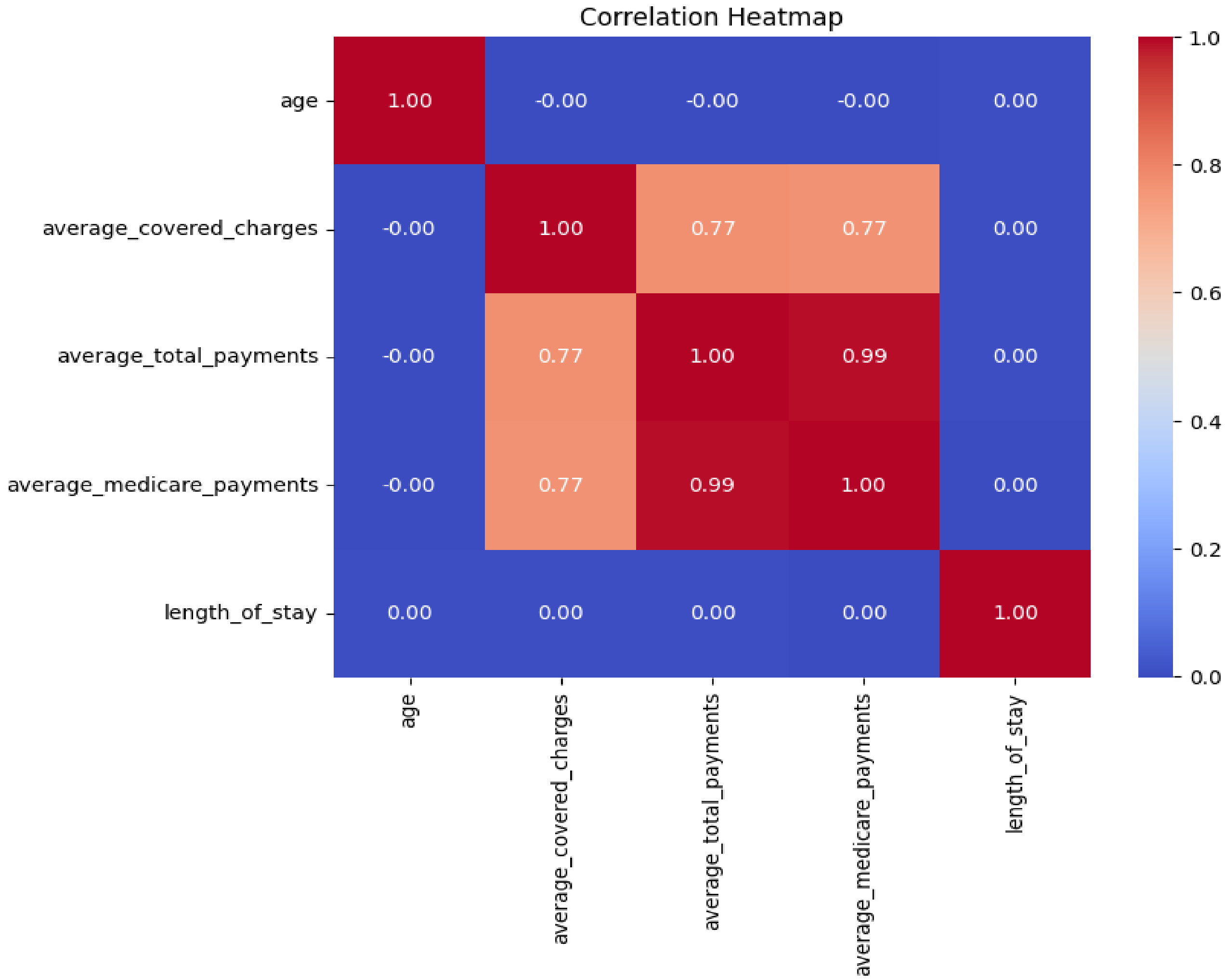
Code:

```
plt.figure(figsize=(8,6))

sns.heatmap(corr_matrix, annot=True, cmap="coolwarm", fmt=".2f")

plt.title("Correlation Heatmap")

plt.show()
```

Expected Output:

- Heatmap with correlations (e.g., charges vs age vs stay).

Actual Output:



Data Engineering Insight:

Heatmaps give a visual summary of correlations, making it easier to spot strong/weak relationships.