```
display(dbutils.fs.ls("/FileStore/tables/"))
```

path	name	size	modificationTime
dbfs:/FileStore/tables/ADMISSIONS.csv	ADMISSIONS.csv	26823	1749640575000
dbfs:/FileStore/tables/CALLOUT.csv	CALLOUT.csv	13820	1749640575000
dbfs:/FileStore/tables/CAREGIVERS.csv	CAREGIVERS.csv	178142	1749640576000
dbfs:/FileStore/tables/CHARTEVENTS.csv	CHARTEVENTS.csv	77730362	1749640703000
dbfs:/FileStore/tables/CPTEVENTS.csv	CPTEVENTS.csv	149024	1749640580000
dbfs:/FileStore/tables/DATETIMEEVENTS.csv	DATETIMEEVENTS.csv	1782801	1749640651000
dbfs:/FileStore/tables/DIAGNOSES_ICD.csv	DIAGNOSES_ICD.csv	48997	1749640656000
dbfs:/FileStore/tables/DRGCODES.csv	DRGCODES.csv	23122	1749640657000
dbfs:/FileStore/tables/D_CPT.csv	D_CPT.csv	12717	1749640582000

```
display(dbutils.fs.ls("/mnt/AzureBlobData1"))
```

```
---------------------------------------------------------------------------
ExecutionError                            Traceback (most recent call last)
File <command-4185528301637834>:1
----> 1 display(dbutils.fs.ls("/mnt/AzureBlobData1"))

File /databricks/python_shell/dbruntime/dbutils.py:364, in DBUtils.FSHandler.prettify_exception_message.<locals>.f_with_exception_handling(*args,
**kwargs)
    362 exc.__context__ = None
    363 exc.__cause__ = None
--> 364 raise exc

ExecutionError: An error occurred while calling o412.ls.
: java.io.FileNotFoundException: /mnt/AzureBlobData1
        at com.databricks.backend.daemon.data.client.DbfsClient.send0(DbfsClient.scala:121)
        at com.databricks.backend.daemon.data.client.DbfsClient.sendIdempotent(DbfsClient.scala:69)
        at com.databricks.backend.daemon.data.client.DatabricksFileSystemV1.listStatus(DatabricksFileSystemV1.scala:179)
        at com.databricks.backend.daemon.data.client.DatabricksFileSystem.listStatus(DatabricksFileSystem.scala:161)
        at com.databricks.backend.daemon.dbutils.FSUtils.lsWithLimit(DBUtilsCore.scala:274)
        at com.databricks.backend.daemon.dbutils.FSUtils.$anonfun$lsImpl$4(DBUtilsCore.scala:243)
        at com.databricks.backend.daemon.dbutils.FSUtils.withFsSafetyCheck(DBUtilsCore.scala:149)
        at com.databricks.backend.daemon.dbutils.FSUtils.$anonfun$lsImpl$3(DBUtilsCore.scala:243)
        at com.databricks.backend.daemon.dbutils.FSUtils.checkPermission(DBUtilsCore.scala:144)
        at com.databricks.backend.daemon.dbutils.FSUtils.lsImpl(DBUtilsCore.scala:242)
        at com.databricks.backend.daemon.dbutils.FSUtils.$anonfun$ls$1(DBUtilsCore.scala:215)
        at com.databricks.logging.UsageLogging.$anonfun$recordOperation$1(UsageLogging.scala:560)
        at com.databricks.logging.UsageLogging.executeThunkAndCaptureResultTags$1(UsageLogging.scala:657)
        at com.databricks.logging.UsageLogging.$anonfun$recordOperationWithResultTags$4(UsageLogging.scala:678)
        at com.databricks.logging.UsageLogging.$anonfun$withAttributionContext$1(UsageLogging.scala:414)
        at scala.util.DynamicVariable.withValue(DynamicVariable.scala:62)
        at com.databricks.logging.AttributionContext$.withValue(AttributionContext.scala:158)
        at com.databricks.logging.UsageLogging.withAttributionContext(UsageLogging.scala:412)
        at com.databricks.logging.UsageLogging.withAttributionContext$(UsageLogging.scala:409)
        at com.databricks.backend.daemon.dbutils.FSUtils.withAttributionContext(DBUtilsCore.scala:71)
        at com.databricks.logging.UsageLogging.withAttributionTags(UsageLogging.scala:457)
        at com.databricks.logging.UsageLogging.withAttributionTags$(UsageLogging.scala:442)
        at com.databricks.backend.daemon.dbutils.FSUtils.withAttributionTags(DBUtilsCore.scala:71)
        at com.databricks.logging.UsageLogging.recordOperationWithResultTags(UsageLogging.scala:652)
        at com.databricks.logging.UsageLogging.recordOperationWithResultTags$(UsageLogging.scala:569)
        at com.databricks.backend.daemon.dbutils.FSUtils.recordOperationWithResultTags(DBUtilsCore.scala:71)
        at com.databricks.logging.UsageLogging.recordOperation(UsageLogging.scala:560)
        at com.databricks.logging.UsageLogging.recordOperation$(UsageLogging.scala:528)
        at com.databricks.backend.daemon.dbutils.FSUtils.recordOperation(DBUtilsCore.scala:71)
        at com.databricks.backend.daemon.dbutils.FSUtils.recordDbutilsFsOp(DBUtilsCore.scala:135)
        at com.databricks.backend.daemon.dbutils.FSUtils.ls(DBUtilsCore.scala:215)
        at sun.reflect.NativeMethodAccessorImpl.invoke0(Native Method)
        at sun.reflect.NativeMethodAccessorImpl.invoke(NativeMethodAccessorImpl.java:62)
        at sun.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAccessorImpl.java:43)
        at java.lang.reflect.Method.invoke(Method.java:498)
        at py4j.reflection.MethodInvoker.invoke(MethodInvoker.java:244)
        at py4j.reflection.ReflectionEngine.invoke(ReflectionEngine.java:380)
        at py4j.Gateway.invoke(Gateway.java:306)
        at py4j.commands.AbstractCommand.invokeMethod(AbstractCommand.java:132)
        at py4j.commands.CallCommand.execute(CallCommand.java:79)
        at py4j.ClientServerConnection.waitForCommands(ClientServerConnection.java:195)
        at py4j.ClientServerConnection.run(ClientServerConnection.java:115)
        at java.lang.Thread.run(Thread.java:750)
```

```
import seaborn as sns
import pandas as pd
import matplotlib.pyplot as plt
from pyspark.sql.functions import col, datediff, to_date, when, count, mean, desc
from pyspark.sql.utils import AnalysisException
```

Load `CSV` files

```
def load_csv(file_name):
    path = f"dbfs:/FileStore/tables/{file_name}"
    try:
        df = spark.read.option("header", True).csv(path, inferSchema=True)
        print(f"Loaded: {file_name}")
        return df
    except AnalysisException as e:
        print(f" File not found or cannot be read: {file_name}")
        return None

# load data:
admissions = load_csv("ADMISSIONS.csv")
patients = load_csv("PATIENTS.csv")
icustays =  load_csv("ICUSTAYS.csv")
diagnoses =  load_csv("DIAGNOSES_ICD.csv")
icd_codes = load_csv("D_ICD_DIAGNOSES.csv")
```

```
Loaded: ADMISSIONS.csv
Loaded: PATIENTS.csv
Loaded: ICUSTAYS.csv
Loaded: DIAGNOSES_ICD.csv
Loaded: D_ICD_DIAGNOSES.csv
```

Data preprocessing

```
#convert to date objects from str format

admissions = admissions.withColumn("admittime", to_date("admittime"))
patients = patients.withColumn("dob", to_date("dob"))
```

```
# join & compute age

df = admissions.join(patients.select("subject_id", "gender", "dob"), on="subject_id", how="left")
df = df.withColumn("age", (datediff("admittime", "dob") / 365).cast("int"))
df = df.filter(col("age") <= 100)
df = df.withColumn("age_at_death", when(col("hospital_expire_flag") == 1, (datediff("admittime", "dob") / 365).cast("int")))
```

```
#join icu dataset and taking only imp columns
from pyspark.sql.functions import col

icustays_clean = icustays.select(
    col("subject_id"),
    col("hadm_id"),
    col("los").alias("icu_los")
)

df = df.join(icustays_clean, on=["subject_id", "hadm_id"], how="left")


display(df)
```

subject_id	hadm_id	row_id	admittime	dischtime	deathtime	admission_type	admission_location	discharge_location	insurance
10006	142345	12258	2164-10-23	2164-11-01T17:15:00.000+0000	null	EMERGENCY	EMERGENCY ROOM ADMIT	HOME HEALTH CARE	Medicare
10011	105331	12263	2126-08-14	2126-08-28T18:59:00.000+0000	2126-08-28T18:59:00.000+0000	EMERGENCY	TRANSFER FROM HOSP/EXTRAM	DEAD/EXPIRED	Private
10013	165520	12265	2125-10-04	2125-10-07T15:13:00.000+0000	2125-10-07T15:13:00.000+0000	EMERGENCY	TRANSFER FROM HOSP/EXTRAM	DEAD/EXPIRED	Medicare
10017	199207	12269	2149-05-26	2149-06-03T18:42:00.000+0000	null	EMERGENCY	EMERGENCY ROOM ADMIT	SNF	Medicare
10019	177759	12270	2163-05-14	2163-05-15T12:00:00.000+0000	2163-05-15T12:00:00.000+0000	EMERGENCY	TRANSFER FROM HOSP/EXTRAM	DEAD/EXPIRED	Medicare

```
# mortality colm(hospital_expire_flag)

df = df.withColumn("mortality", when(col("hospital_expire_flag") == 1, "Died").otherwise("Survived"))
```

Analysis

- First joins diagnosis codes (DIAGNOSES_ICD.csv) with descriptions (D_ICD_DIAGNOSES.csv) to get disease names.
- Then join with df to bring in hospital_expire_flag for mortality info.

```
diag = diagnoses.join(icd_codes.select("icd9_code", "long_title"), on="icd9_code", how="left") \
               .join(df.select("subject_id", "hadm_id", "hospital_expire_flag"), on=["subject_id", "hadm_id"], how="inner")
```

Top 15 Diagnosed Diseases

- Groups diagnoses by disease name (long_title) and counts frequency.
- Selects the top 15 most common diseases.
- .flatMap(lambda x: x).collect() converts the Spark output into a Python list of disease names.

```
top_disease_titles = diag.groupBy("long_title") \
                         .agg(count("*").alias("count")) \
                         .orderBy(desc("count")) \
                         .limit(15) \
                         .rdd.flatMap(lambda x: x).collect()
```

- Filters diag DataFrame to include only the top 15 most common diseases.

```
diag_top = diag.filter(col("long_title").isin(top_disease_titles))
```

- Groups by disease (long_title) and calculates average of hospital_expire_flag:
- Since hospital_expire_flag is 1 for death and 0 for survival, the mean = mortality rate.
- Sorts diseases by highest to lowest mortality rate.

```
disease_mortality = diag_top.groupBy("long_title") \
                            .agg(mean("hospital_expire_flag").alias("mortality_rate")) \
                            .orderBy(desc("mortality_rate"))
```

```
df_spark = spark.read.option("header", "true") \
    .option("inferSchema", "true") \
    .option("timestampFormat", "yyyy-MM-dd HH:mm:ss") \
    .csv("dbfs:/FileStore/tables/ADMISSIONS.csv")

df_spark.createOrReplaceTempView("admissions1")
```

```
admissions1 = df_spark.toPandas()
```

Visualization (via Pandas)

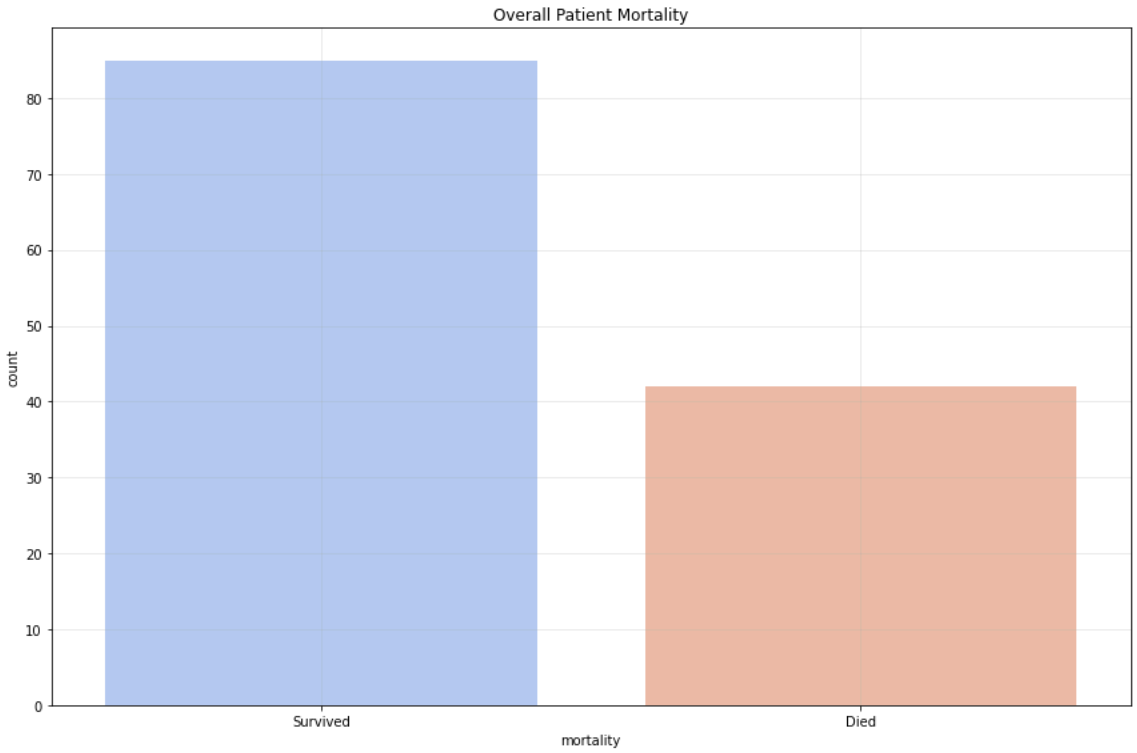
- why we convert PySpark DataFrames to pandas DataFrames using:
- Because most Python plotting libraries (like Matplotlib, Seaborn, Plotly, etc.) do not work directly with PySpark DataFrames. They expect the data to be in a format they understand, which is typically a pandas DataFrame.
- Library Compatibility: Seaborn, matplotlib, and most ML libraries require pandas, not Spark.

Best Practice:

1. Do heavy computation in PySpark.
2. Convert only final output or samples to pandas.
3. Use pandas for: Plotting

```
df_pd = df.select("age", "gender", "mortality",  "admission_type", "hospital_expire_flag").toPandas()
disease_mortality_pd = disease_mortality.toPandas()
```

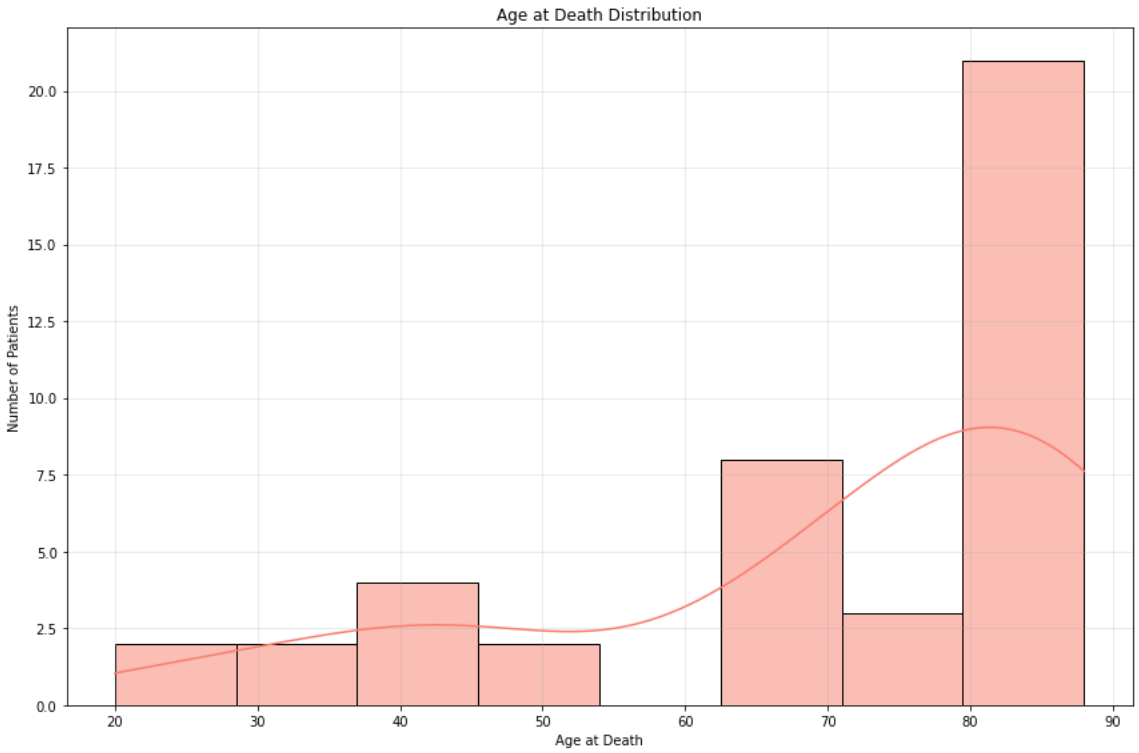
```
# Plot 1: Overall Mortality
plt.figure(figsize=(12, 8))
sns.countplot(data=df_pd, x='mortality', palette='coolwarm')
plt.grid(alpha=0.3)
plt.title("Overall Patient Mortality")
plt.tight_layout()
plt.show()
display(df_pd['mortality'])
```




```
0      Survived
1          Died
2          Died
3      Survived
4          Died
         ...
122    Survived
123    Survived
124    Survived
125    Survived
126    Survived
Name: mortality, Length: 127, dtype: object
```

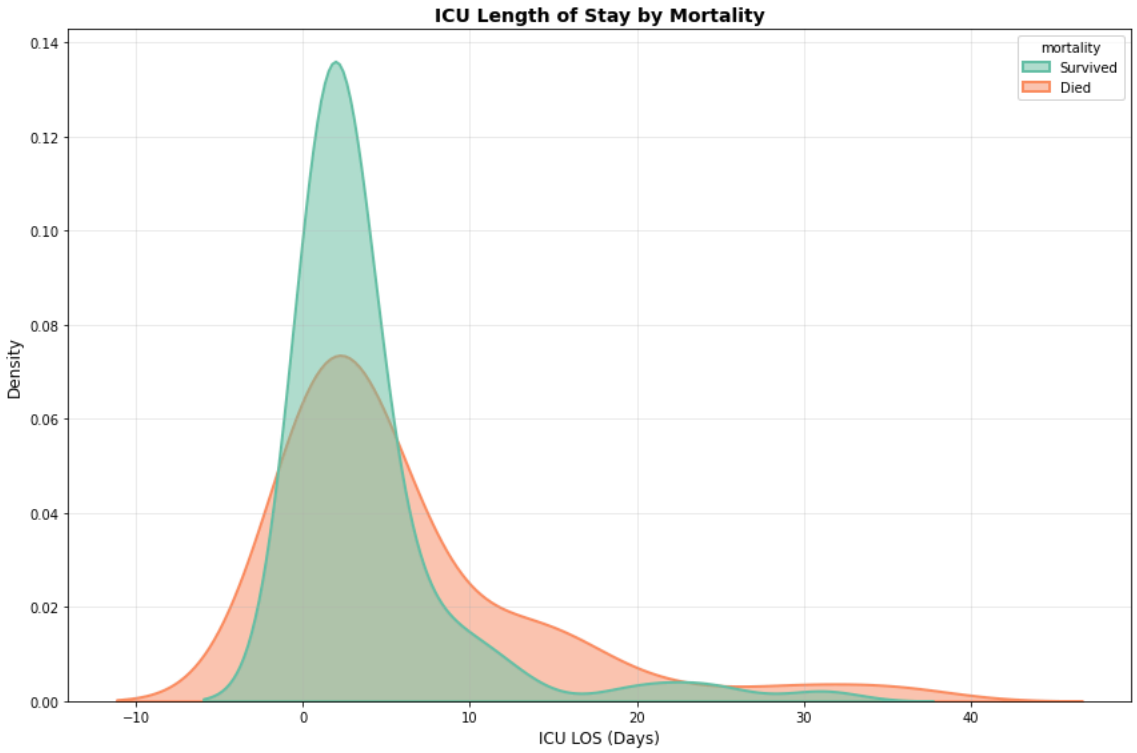
```
from pyspark.sql.functions import when, datediff, col

#age_at_death for patients who died
df = df.withColumn(
    "age_at_death",
    when(col("hospital_expire_flag") == 1, (datediff("admittime", "dob") / 365).cast("int"))
)

#update Pandas dataframe
df_pd = df.toPandas()
```

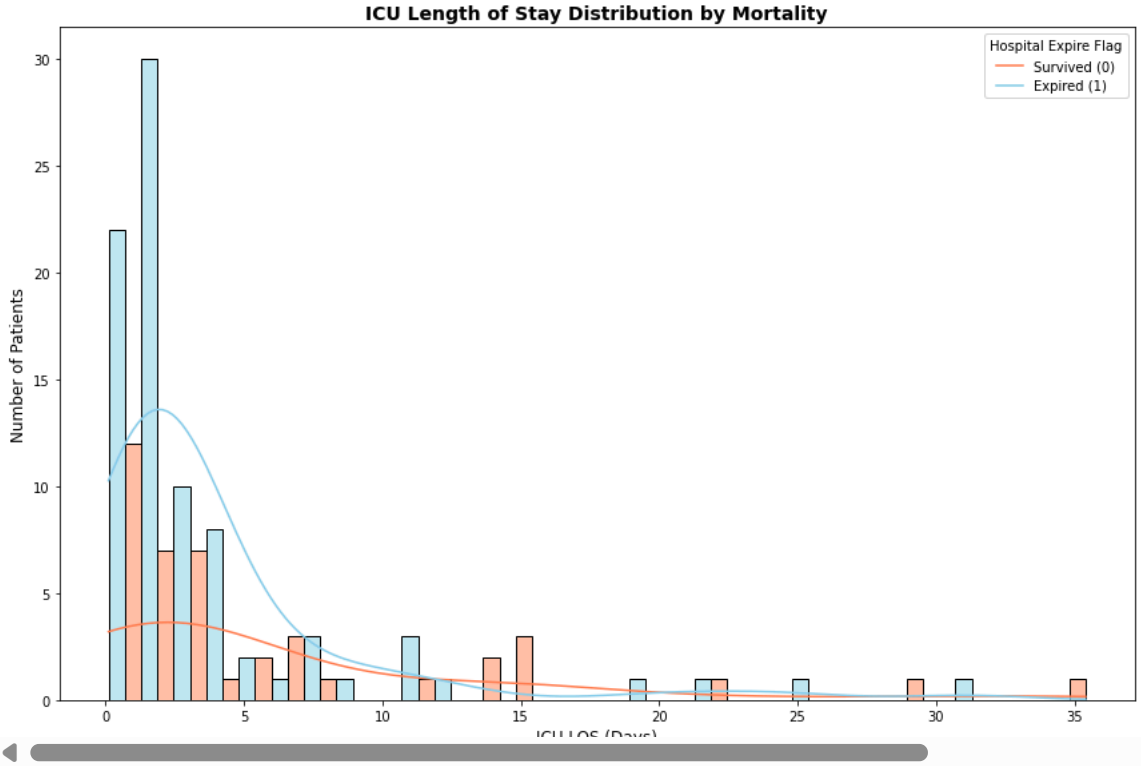
```
# Plot2: Age at Death Distribution
plt.figure(figsize=(12, 8))
sns.histplot(data=df_pd[df_pd['age_at_death'].notna()], x='age_at_death', bins='auto', kde=True, color='salmon')
plt.grid(alpha=0.3)

plt.title("Age at Death Distribution")
plt.xlabel("Age at Death")
plt.ylabel("Number of Patients")
plt.tight_layout()
plt.show()

display(df_pd['age_at_death'])
```



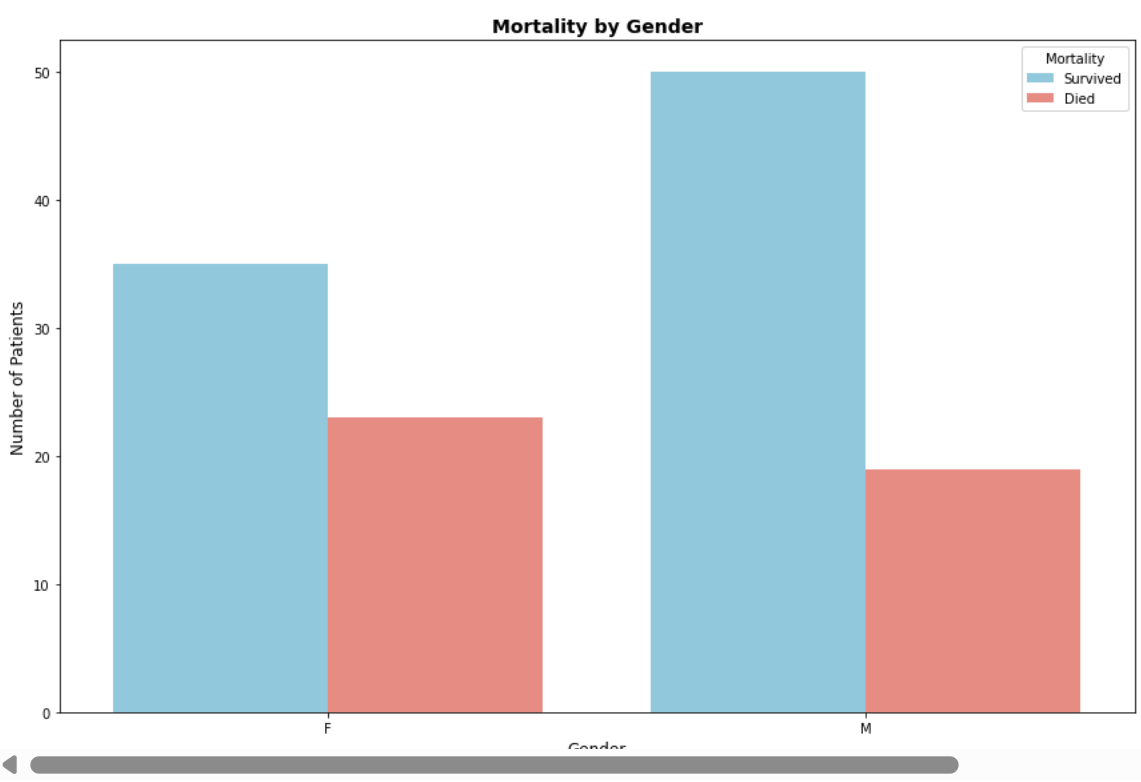

```
0       NaN
1      36.0
2      87.0
3       NaN
4      48.0
       ...
122     NaN
123     NaN
124     NaN
125     NaN
126     NaN
Name: age at death, Length: 127, dtype: float64
```

```
# Plot 3: ICU Length of Stay

plt.figure(figsize=(12, 8))
sns.kdeplot(
    data=df_pd,
    x="icu_los",
    hue="mortality",
    fill=True,
    common_norm=False,
    palette="Set2",
    alpha=0.5,
    linewidth=2
)

plt.title("ICU Length of Stay by Mortality", fontsize=14, fontweight='bold')
plt.xlabel("ICU LOS (Days)", fontsize=12)
plt.ylabel("Density", fontsize=12)
plt.grid(alpha=0.3)
plt.tight_layout()
plt.show()


# Plot 4: ICU Length of Stay by Mortality

plt.figure(figsize=(12, 8))

sns.histplot(
    data=df_pd,
    x="icu_los",
    hue="hospital_expire_flag",
    multiple="dodge",
    bins=30,
    kde=True,
    palette={0: "skyblue", 1: "coral"}
)

plt.title("ICU Length of Stay Distribution by Mortality", fontsize=14, fontweight="bold")
plt.xlabel("ICU LOS (Days)", fontsize=12)
plt.ylabel("Number of Patients", fontsize=12)
plt.legend(title="Hospital Expire Flag", labels=["Survived (0)", "Expired (1)"])
plt.tight_layout()
plt.show()
```






```
# Plot 5: Gender vs Mortality

palette = {
    "Survived": "skyblue",
    "Died": "salmon"
}


plt.figure(figsize=(12, 8))
sns.countplot(
    data=df_pd,
    x="gender",
    hue="mortality",
    palette=palette
)

plt.title("Mortality by Gender", fontsize=14, fontweight='bold')
plt.xlabel("Gender", fontsize=12)
plt.ylabel("Number of Patients", fontsize=12)
plt.legend(title="Mortality", labels=["Survived", "Died"])
plt.tight_layout()
plt.show()
```




```
from pyspark.sql.functions import datediff, to_date, col, when

admissions = admissions.withColumn("admittime", to_date("admittime"))
patients = patients.withColumn("dob", to_date("dob"))

#join and calculate age_at_death
patient_age_df = admissions.join(patients, on="subject_id", how="left") \
    .withColumn("age_at_death", when(col("hospital_expire_flag") == 1,
                                     (datediff(col("admittime"), col("dob")) / 365).cast("int")))
```

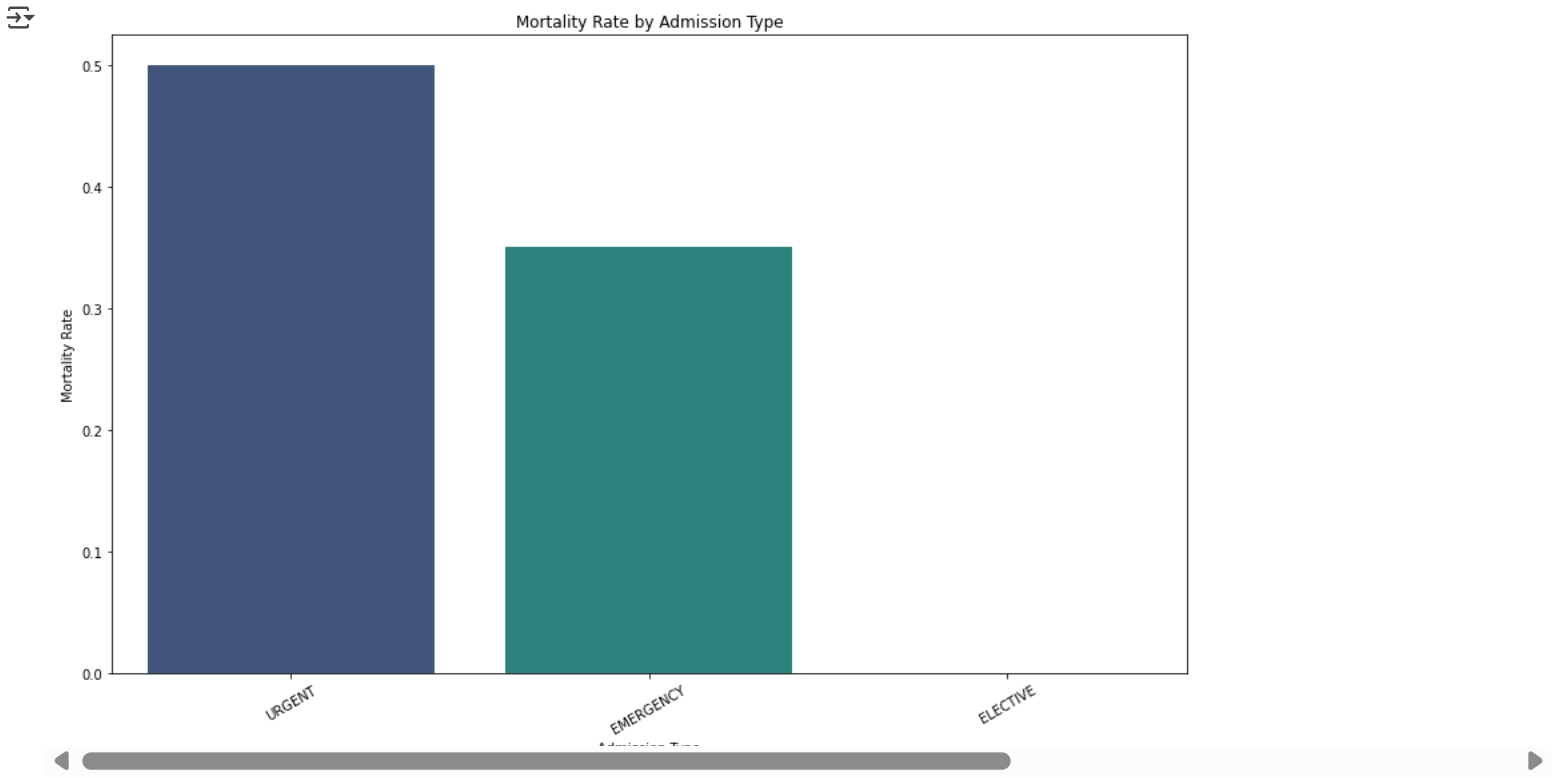
```
#Plot 6: Remove outliers > 30 days
deceased = deceased[deceased['time_to_death_days'] <= 30]

plt.figure(figsize=(12, 8))
sns.histplot(deceased['time_to_death_days'], bins=30, color='crimson', kde=True)
plt.title("Time of Death After Admission (<= 30 Days)")
plt.xlabel("Days Since Admission")
plt.ylabel("Number of Patients")
plt.tight_layout()
plt.show()
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
File <command-4185528301637865>:2
      1 #Plot 6: Remove outliers > 30 days
----> 2 deceased = deceased[deceased['time_to_death_days'] <= 30]
      4 plt.figure(figsize=(12, 8))
      5 sns.histplot(deceased['time_to_death_days'], bins=30, color='crimson', kde=True)

NameError: name 'deceased' is not defined
```

```
from pyspark.sql.functions import to_date, col, when, count, mean

#'admittime' and 'dob' to date format
admissions = admissions.withColumn("admittime", to_date("admittime"))
patients = patients.withColumn("dob", to_date("dob"))

#conv deathtime to date
df = df.withColumn("deathtime", to_date("deathtime"))

#create mortality_flag => 1 if deathtime is not null, else 0
df = df.withColumn("mortality_flag", when(col("deathtime").isNotNull(), 1).otherwise(0))

#group by insurance and calculate mortality rate
mortality_by_insurance = df.groupBy("insurance") \
    .agg(mean("mortality_flag").alias("mortality_rate")) \
    .withColumn("mortality_rate", (col("mortality_rate") * 100)) \
    .orderBy(col("mortality_rate").desc())

#convert to Pandas for plotting
mortality_by_insurance_pd = mortality_by_insurance.toPandas()
```

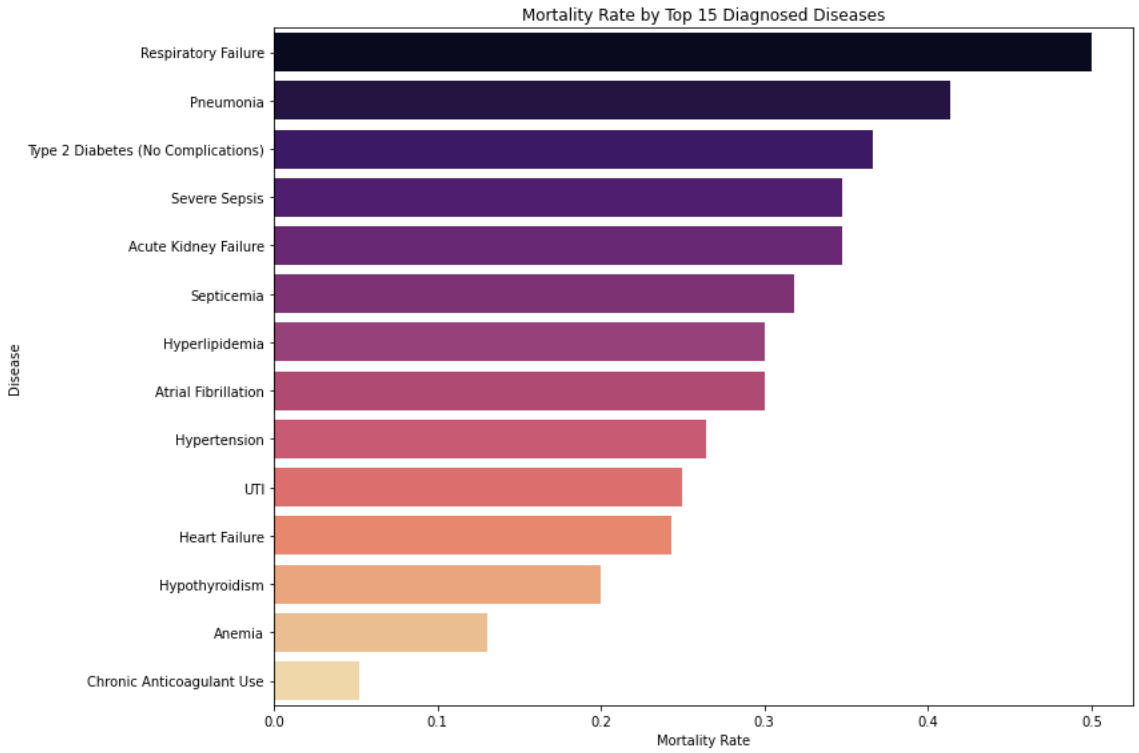
```
# Plot 7: Mortality by Admission Type
```

```
plt.figure(figsize=(12, 8))
adm_mortality = df_pd.groupby("admission_type")["hospital_expire_flag"].mean().sort_values(ascending=False)
sns.barplot(x=adm_mortality.index, y=adm_mortality.values, palette="viridis")
plt.title("Mortality Rate by Admission Type")
plt.xlabel("Admission Type")
plt.ylabel("Mortality Rate")
plt.xticks(rotation=30)
plt.tight_layout()
plt.show()
```

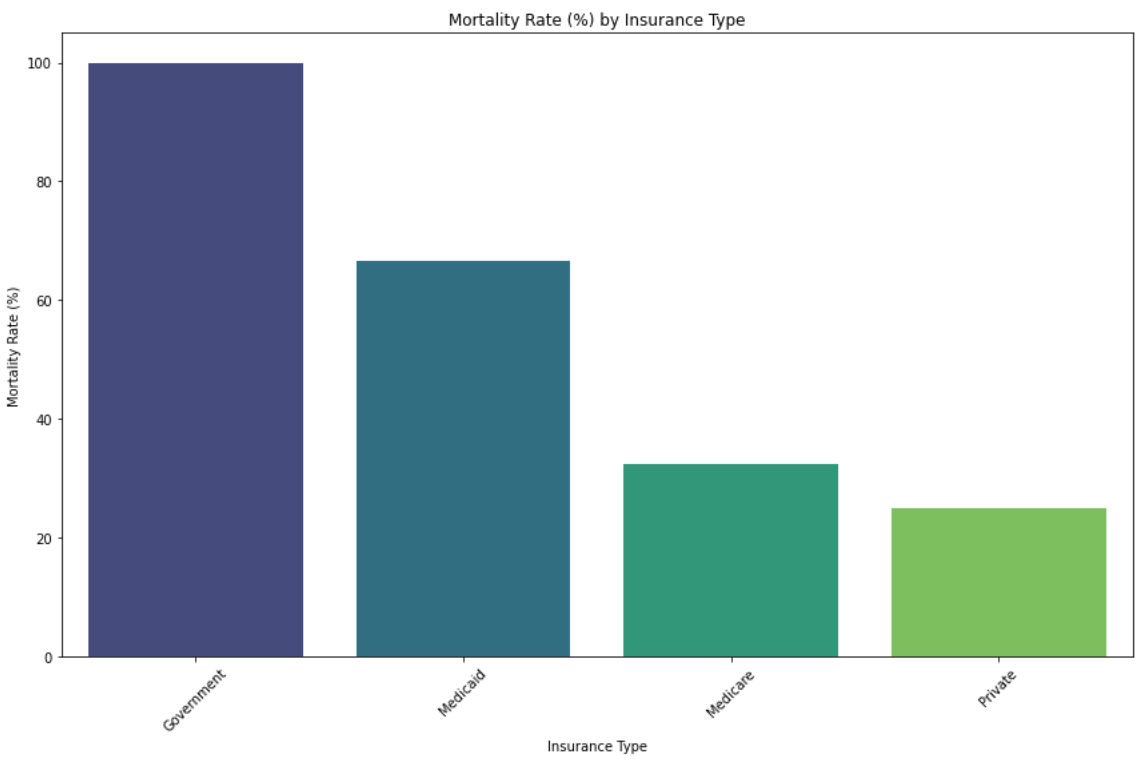



```
replacements = {
    "Acute respiratory failure": "Respiratory Failure",
    "Pneumonia, organism unspecified": "Pneumonia",
    "Diabetes mellitus without mention of complication, type II or unspecified type, not stated as uncontrolled": "Type 2 Diabetes (No Complications)"
    "Severe sepsis": "Severe Sepsis",
    "Acute kidney failure, unspecified": "Acute Kidney Failure",
    "Unspecified septicemia": "Septicemia",
    "Other and unspecified hyperlipidemia": "Hyperlipidemia",
    "Atrial fibrillation": "Atrial Fibrillation",
    "Unspecified essential hypertension": "Hypertension",
    "Urinary tract infection, site not specified": "UTI",
    "Congestive heart failure, unspecified": "Heart Failure",
    "Unspecified acquired hypothyroidism": "Hypothyroidism",
    "Anemia, unspecified": "Anemia",
    "Long-term (current) use of anticoagulants": "Chronic Anticoagulant Use"
}

disease_mortality_pd["short_title"] = disease_mortality_pd["long_title"].replace(replacements)
```

```
# Plot 8: Top 15 Disease Mortality Rates
plt.figure(figsize=(12, 8))
sns.barplot(data=disease_mortality_pd, x="mortality_rate", y="short_title", palette="magma")
plt.title("Mortality Rate by Top 15 Diagnosed Diseases")
plt.xlabel("Mortality Rate")
plt.ylabel("Disease")
plt.tight_layout()
plt.show()
```




```
# Plot 9: Mortality Rate (%) by Insurance Type
plt.figure(figsize=(12, 8))
sns.barplot(data=mortality_by_insurance_pd, x='insurance', y='mortality_rate', palette='viridis')
plt.title('Mortality Rate (%) by Insurance Type')
plt.xlabel('Insurance Type')
plt.ylabel('Mortality Rate (%)')
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()

print(mortality_by_insurance_pd.head())
```




```
    insurance  mortality_rate
0  Government      100.000000
1    Medicaid       66.666667
2    Medicare       32.291667
3     Private       25.000000
```

```
# Plot 10: Trends in 72 hours before death

# Use Spark to read
admissions_spark = spark.read.option("header", "true").option("inferSchema", "true") \
    .option("timestampFormat", "yyyy-MM-dd HH:mm:ss") \
    .csv("dbfs:/FileStore/tables/ADMISSIONS.csv")
```

```
chartevents_spark = spark.read.option("header", "true").option("inferSchema", "true") \
    .csv("dbfs:/FileStore/tables/CHARTEVENTS.csv")

# Convert to pandas
admissions2 = admissions_spark.toPandas()
chartevents2 = chartevents_spark.toPandas()

# Continue with your analysis
vitals_map = {
    'Heart Rate': 211,
    'Systolic BP': 51,
    'O2 Saturation': 220277
}

vitals = chartevents2[chartevents2['itemid'].isin(vitals_map.values())]
vitals['charttime'] = pd.to_datetime(vitals['charttime'])
admissions2['deathtime'] = pd.to_datetime(admissions2['deathtime'])

admissions2['mortality'] = admissions2['hospital_expire_flag']
vitals = vitals.merge(admissions2[['subject_id', 'hadm_id', 'mortality', 'deathtime']], on=['subject_id', 'hadm_id'], how='inner')
vital_death = vitals[vitals['mortality'] == 1].copy()
vital_death['hours_before_death'] = (vital_death['deathtime'] - vital_death['charttime']).dt.total_seconds() / 3600
vital_death = vital_death[(vital_death['hours_before_death'] >= 0) & (vital_death['hours_before_death'] <= 72)]

id_to_label = {v: k for k, v in vitals_map.items()}
vital_death['Vital Sign'] = vital_death['itemid'].map(id_to_label)

plt.figure(figsize=(14, 6))
sns.lineplot(data=vital_death, x='hours_before_death', y='valuenum', hue='Vital Sign', estimator='mean')
plt.gca().invert_xaxis()
plt.title("Vital Sign Trends in Last 72 Hours Before Death")
```