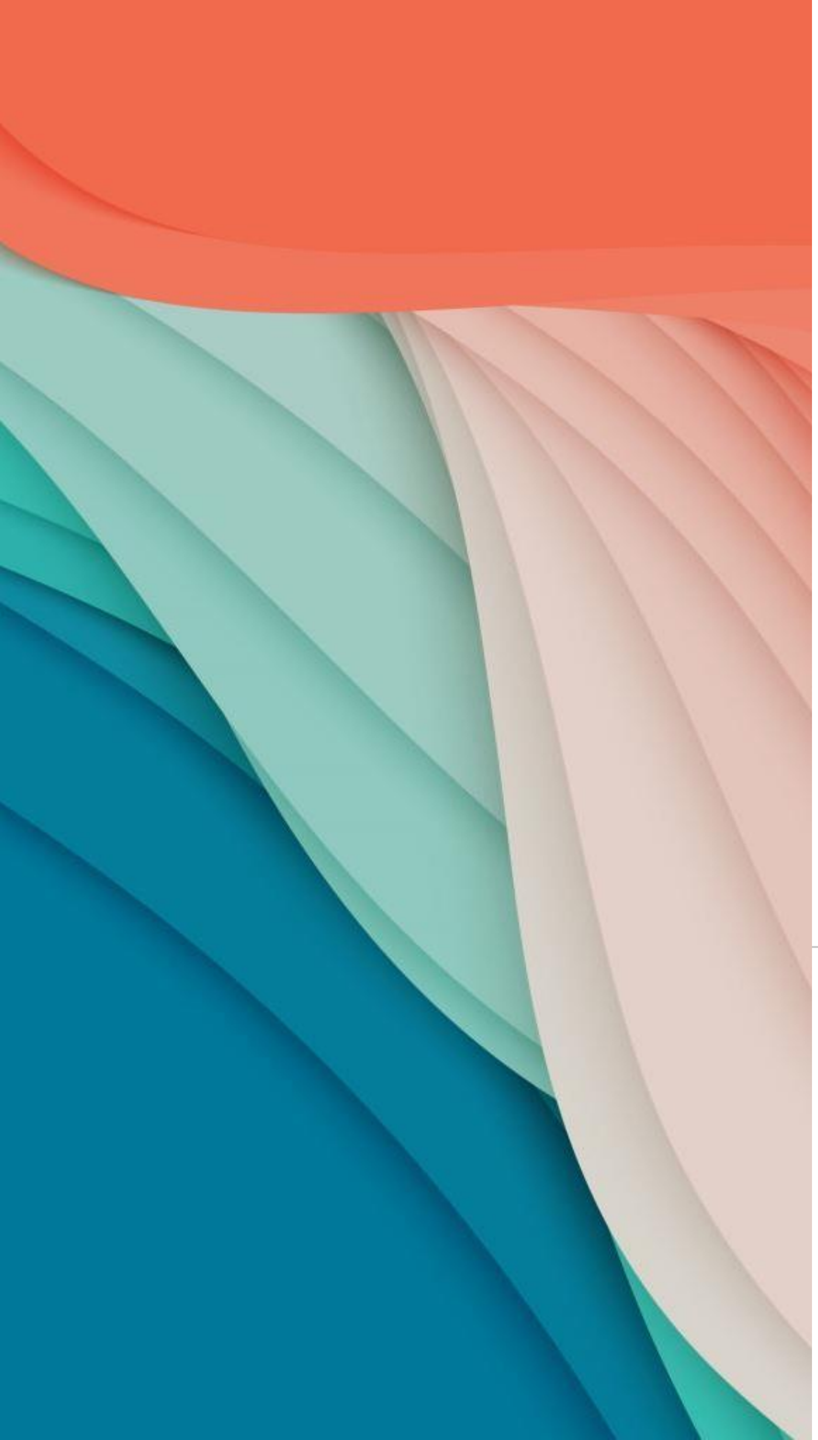


MIMIC Visualization

Ankita Savaliya

AI In Healthcare Course at UT Austin

Coding environment and packages

- Google Colab
- MIMIC-III Demo Dataset
- Used Google Drive for saving and loading MIMIC CSV data
- Libraries Used: matplotlib, seaborn, plotly, WordCloud, altair
- GitHub and Google Colab Links:

https://colab.research.google.com/github/AnkitaSavaliya/AIH/blob/main/MIMIC_Visualization.ipynb

https://github.com/AnkitaSavaliya/AIH/blob/main/MIMIC_Visualization.ipynb (Note: Visualizations from certain libraries may not render properly in the raw (blob) view on the GitHub link due to known limitations. Please use the provided Colab link to view all visuals correctly.)

https://github.com/AnkitaSavaliya/AIH/blob/main/MIMIC%20Visualization.pptx

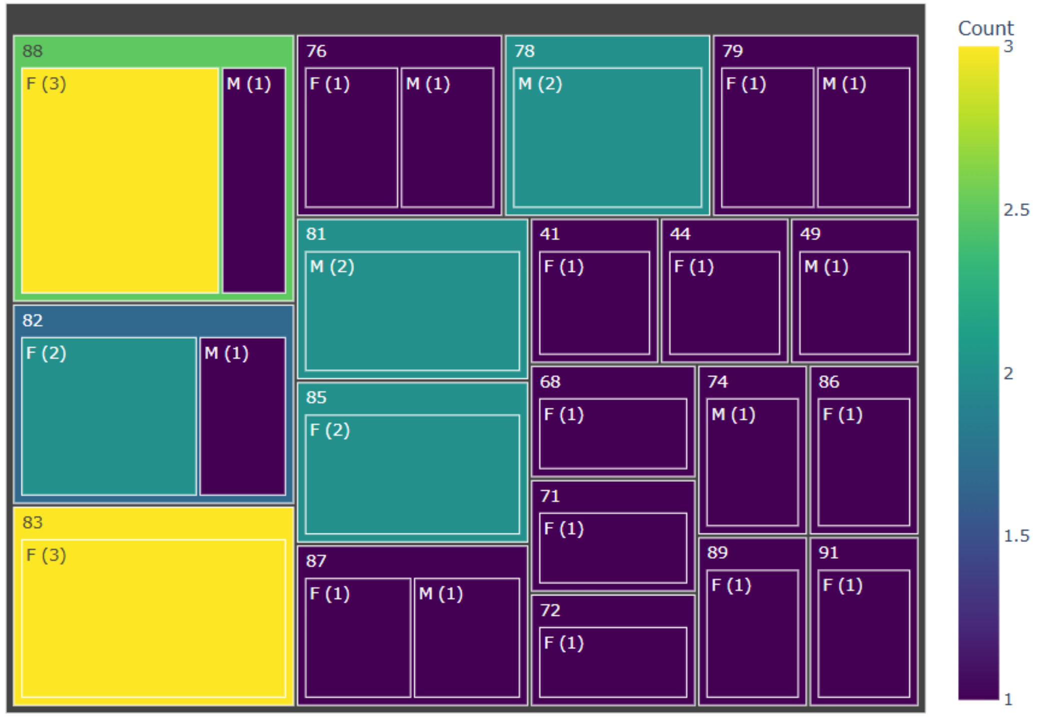
Data Load and Initial Setup

- Used PATIENTS.CSV, ADMISSIONS.CSV, DIAGNOSES_ICD.CSV, D_ICD_DIAGNOSES.CSV, ICUSTAYS.CSV, PRESCRIPTIONS.CSV, PROCEDUREEVENTS_MV, D_ITEMS, etc.
- Created the initial dataframe using PATIENTS, ADMISSIONS, and DIAGNOSES_ICD.
- Merged the dataframe with other CSV files as needed for visualization.
- Represented all visuals using different types of charts.

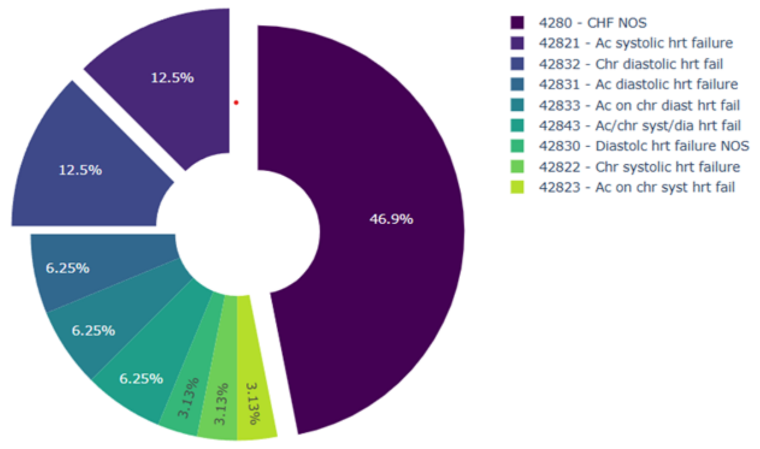
Visual 1- Heart Patient Analysis: Gender, Age, and ICD-9 Subtypes

This visualization represents heart patients, showcasing their age and gender distribution. It also includes an interactive pie chart displaying the proportional distribution of heart failure subtypes.

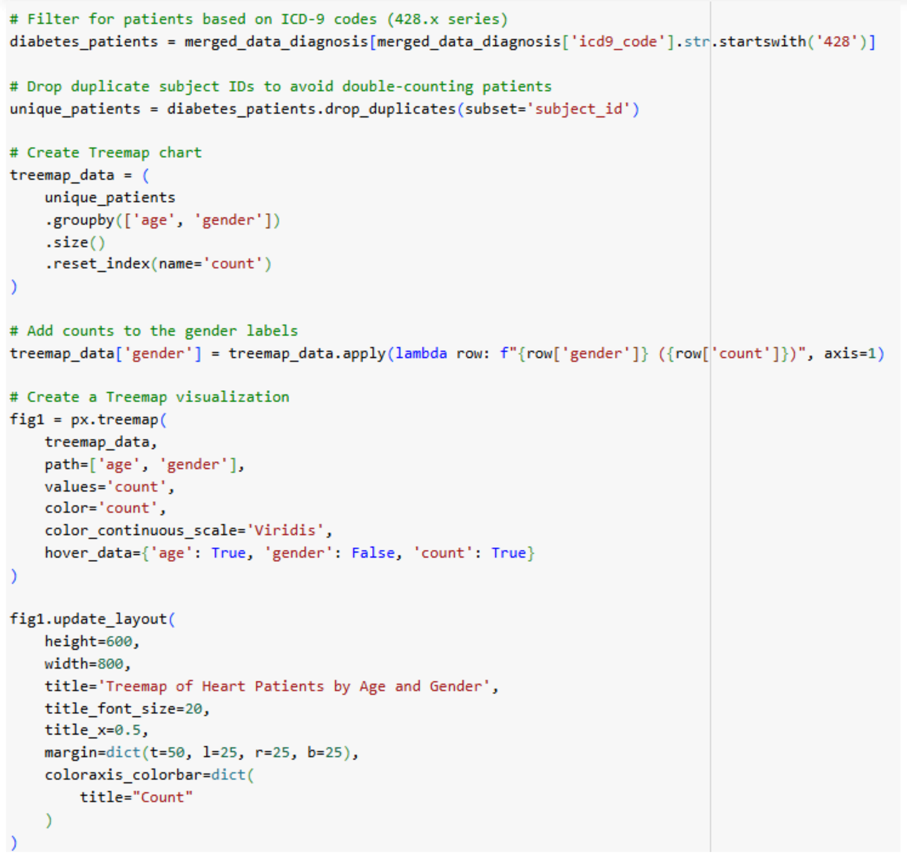
Treemap of Heart Patients by Age and Gender



Proportional Distribution of Heart Failure Subtypes by ICD-9 Code

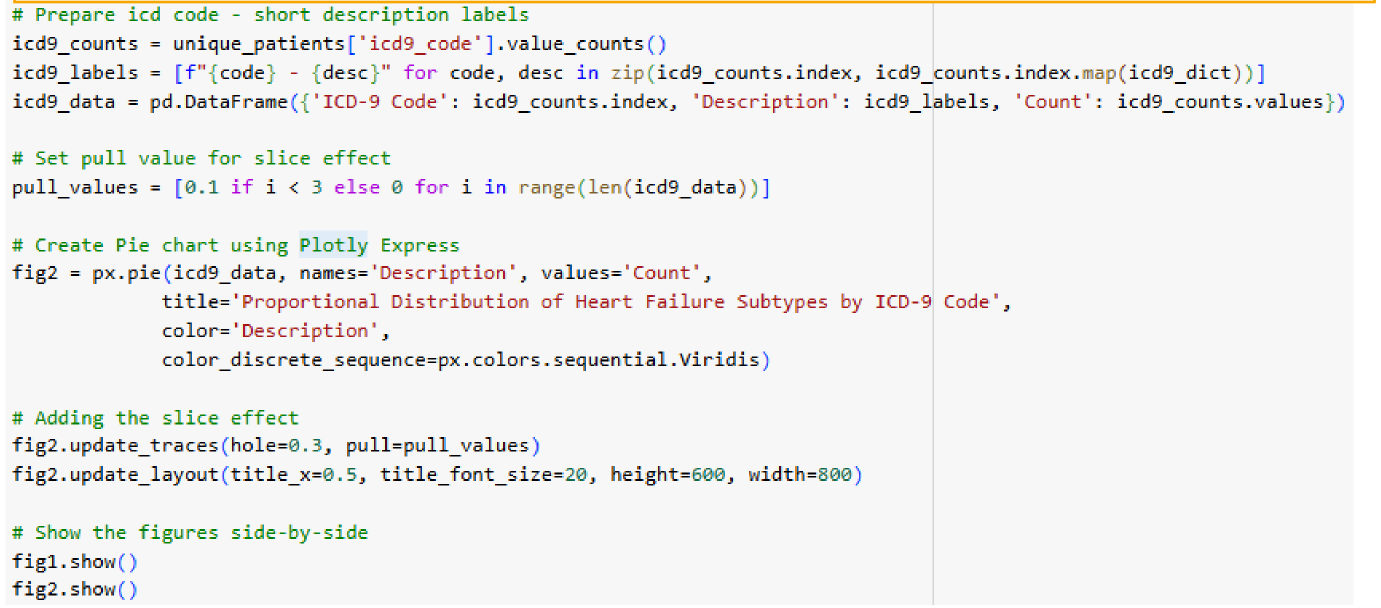


Visual 1 - How to recreate



```python
# Filter for patients based on ICD-9 codes (428.x series)
diabetes_patients = merged_data_diagnosis[merged_data_diagnosis['icd9_code'].str.startswith('428')]

# Drop duplicate subject IDs to avoid double-counting patients
unique_patients = diabetes_patients.drop_duplicates(subset='subject_id')

# Create Treemap chart
treemap_data = (
    unique_patients
    .groupby(['age', 'gender'])
    .size()
    .reset_index(name='count')
)

# Add counts to the gender labels
treemap_data['gender'] = treemap_data.apply(lambda row: f"{row['gender']} ({row['count']})", axis=1)

# Create a Treemap visualization
fig1 = px.treemap(
    treemap_data,
    path=['age', 'gender'],
    values='count',
    color='count',
    color_continuous_scale='Viridis',
    hover_data={'age': True, 'gender': False, 'count': True}
)

fig1.update_layout(
    height=600,
    width=800,
    title='Treemap of Heart Patients by Age and Gender',
    title_font_size=20,
    title_x=0.5,
    margin=dict(t=50, l=25, r=25, b=25),
    coloraxis_colorbar=dict(
        title="Count"
    )
)
```

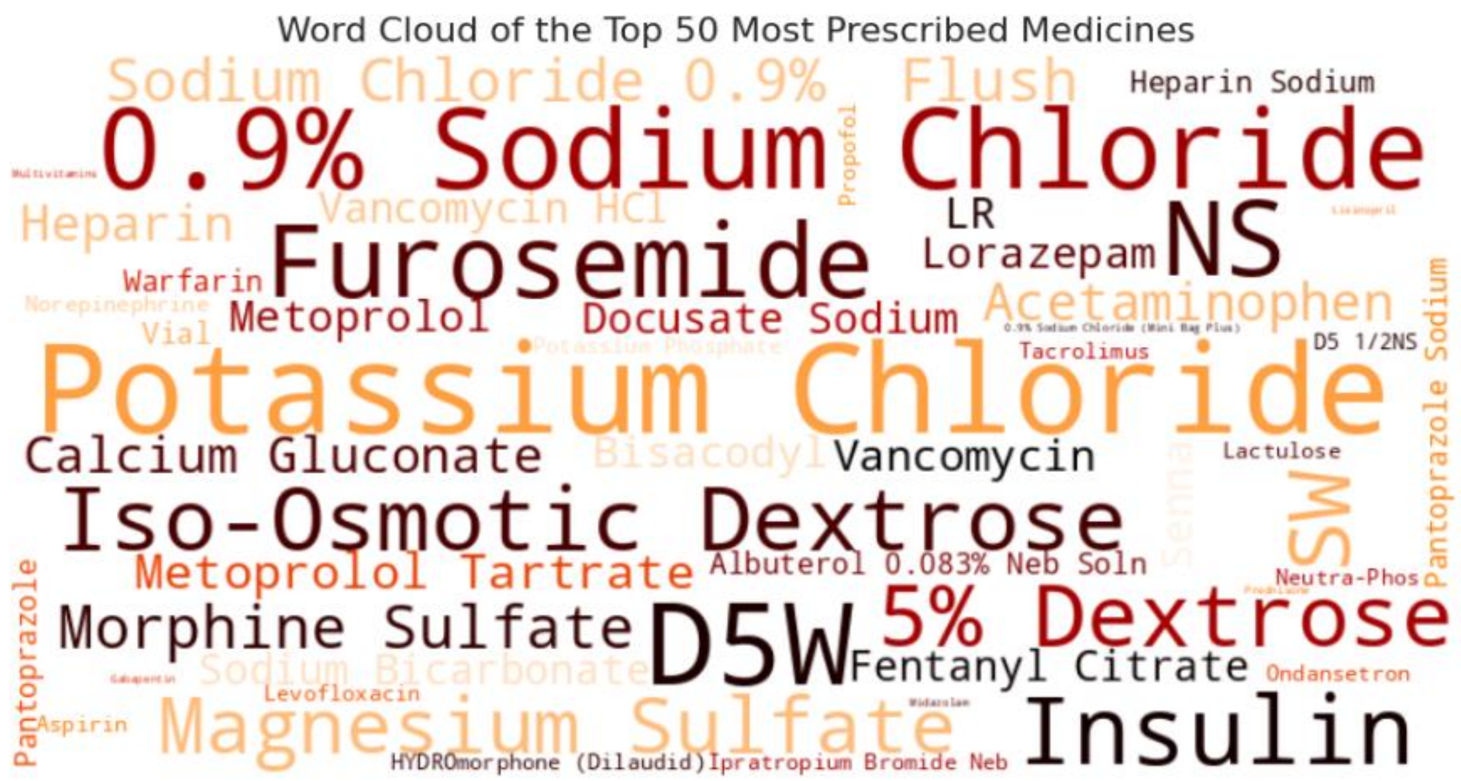
Filters patient records based on ICD-9 codes (428.x series) and drops duplicate patients using their unique IDs. Counts the gender distribution among unique patients and plots a treemap. Counts the occurrences of each ICD-9 code, prepares labels, and creates an interactive pie chart using Plotly Express.



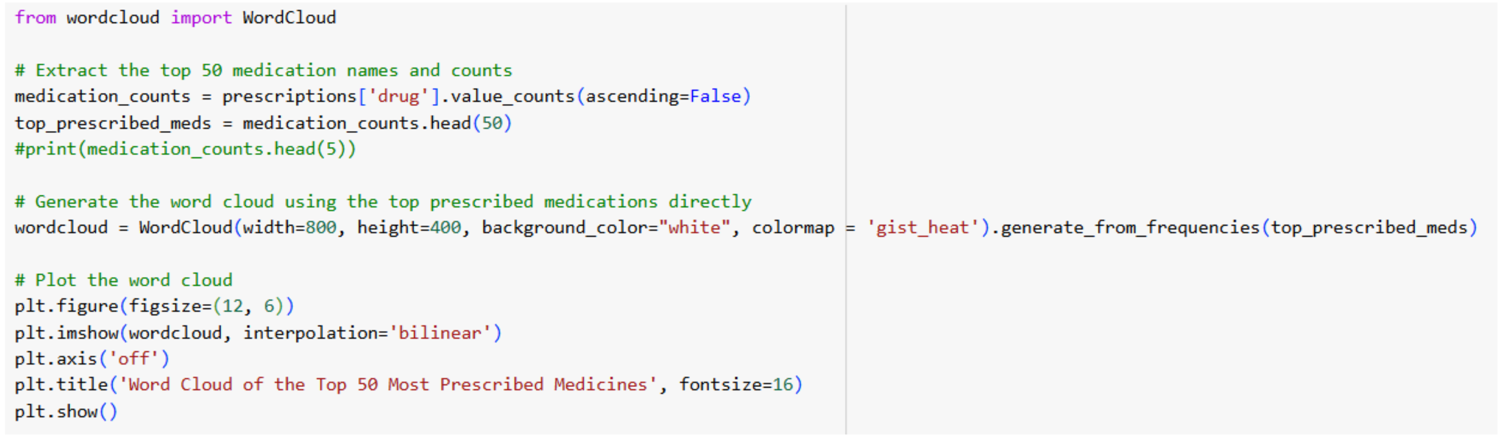
```python
# Prepare icd code - short description labels
icd9_counts = unique_patients['icd9_code'].value_counts()
icd9_labels = [f"{code} - {desc}" for code, desc in zip(icd9_counts.index, icd9_counts.index.map(icd9_dict))]
icd9_data = pd.DataFrame({'ICD-9 Code': icd9_counts.index, 'Description': icd9_labels, 'Count': icd9_counts.values})

# Set pull value for slice effect
pull_values = [0.1 if i < 3 else 0 for i in range(len(icd9_data))]

# Create Pie chart using Plotly Express
fig2 = px.pie(icd9_data, names='Description', values='Count',
              title='Proportional Distribution of Heart Failure Subtypes by ICD-9 Code',
              color='Description',
              color_discrete_sequence=px.colors.sequential.Viridis)

# Adding the slice effect
fig2.update_traces(hole=0.3, pull=pull_values)
fig2.update_layout(title_x=0.5, title_font_size=20, height=600, width=800)

# Show the figures side-by-side
fig1.show()
fig2.show()
```

Visual 2 - Word Cloud of Most Prescribed Medicines

This visualization is a word cloud showcasing the top 50 most prescribed medications.



Visual 2 - How to recreate

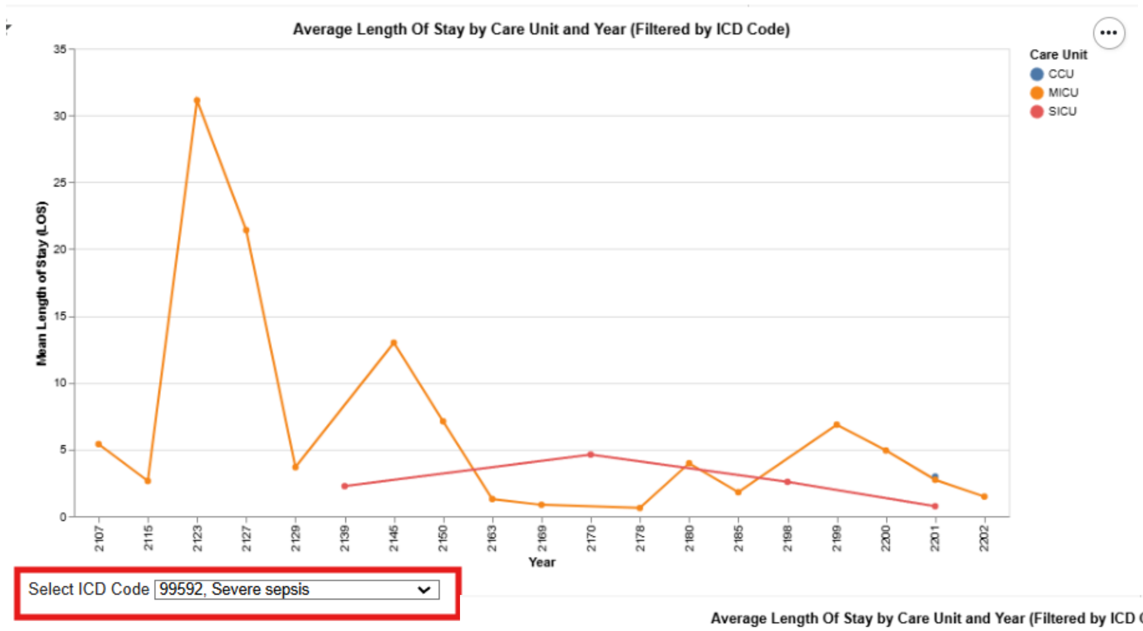


```
from wordcloud import WordCloud

# Extract the top 50 medication names and counts
medication_counts = prescriptions['drug'].value_counts(ascending=False)
top_prescribed_meds = medication_counts.head(50)
#print(medication_counts.head(5))

# Generate the word cloud using the top prescribed medications directly
wordcloud = WordCloud(width=800, height=400, background_color="white", colormap = 'gist_heat').generate_from_frequencies(top_prescribed_meds)

# Plot the word cloud
plt.figure(figsize=(12, 6))
plt.imshow(wordcloud, interpolation='bilinear')
plt.axis('off')
plt.title('Word Cloud of the Top 50 Most Prescribed Medicines', fontsize=16)
plt.show()
```

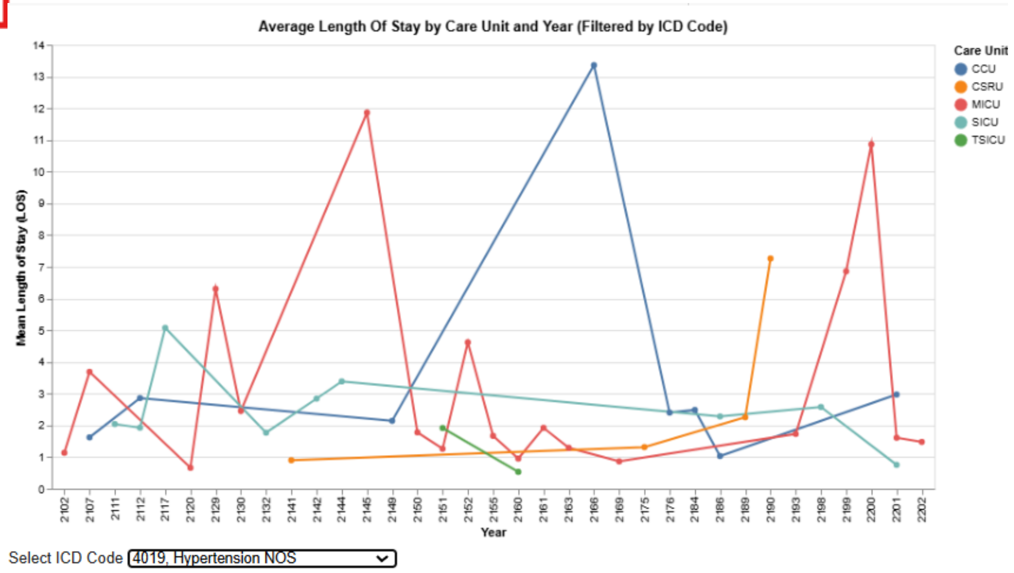
Extract the top 50 most prescribed medication names and their counts from the dataset. Create and display a word cloud using the frequency of these medications, with larger font sizes representing higher prescription counts.

Visual 3 - Average Length of Stay by Care Unit and Year (ICD Code Filtered)"

This visualization is an interactive line chart created with Altair, analyzing the average length of stay (LOS) by care unit and year, dynamically filtered by ICD-9 codes.

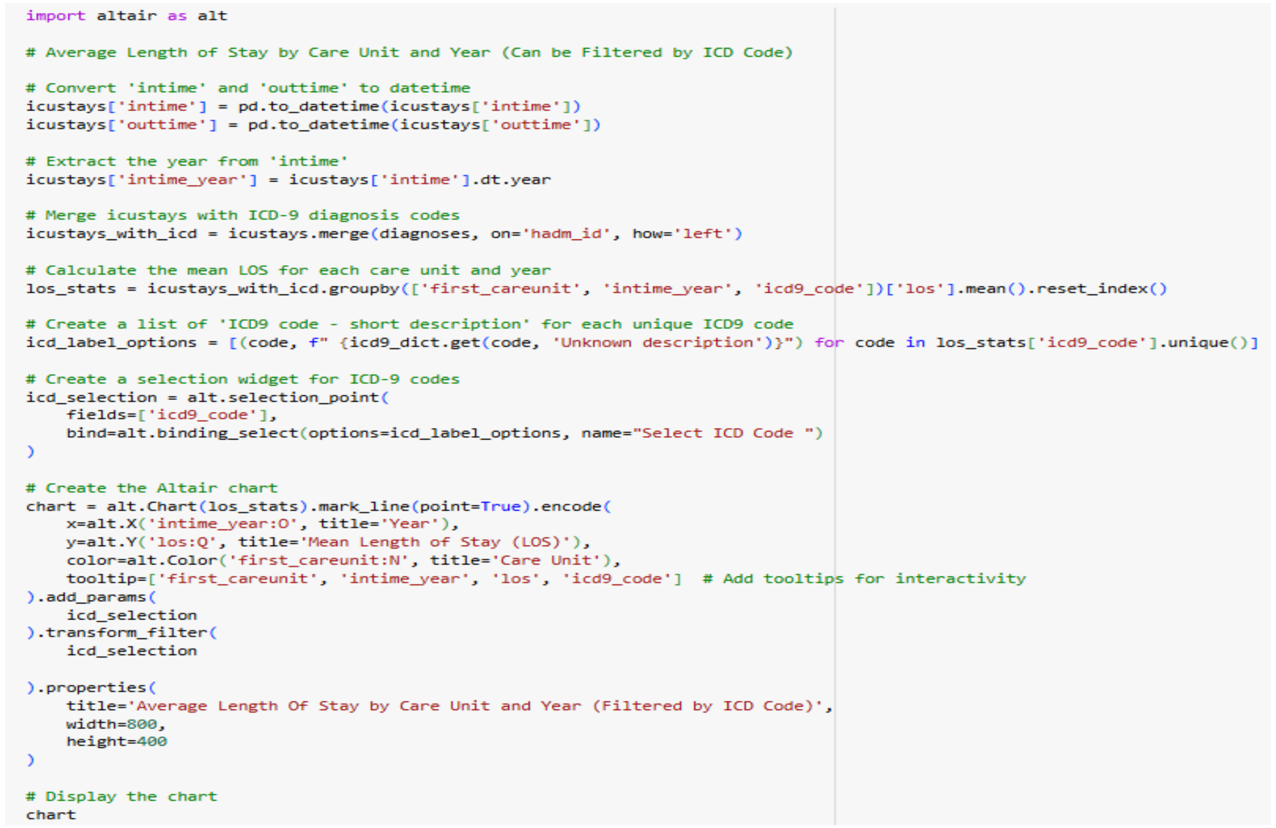


Select ICD Code 99592, Severe sepsis



Select ICD Code 4019, Hypertension NOS

Visual 3 - How to recreate

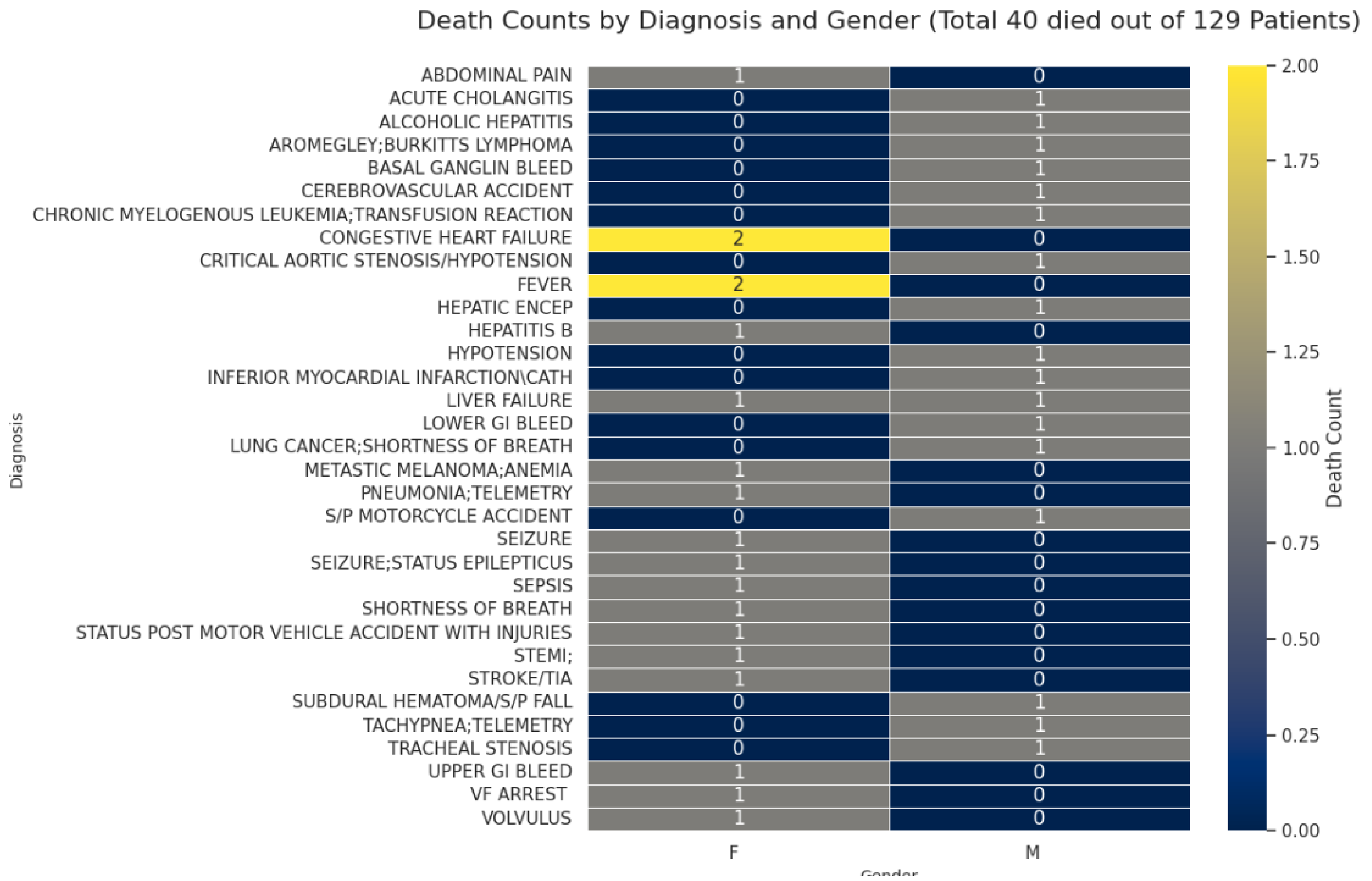


```python
import altair as alt

# Average Length of Stay by Care Unit and Year (Can be Filtered by ICD Code)

# Convert 'intime' and 'outtime' to datetime
icustays['intime'] = pd.to_datetime(icustays['intime'])
icustays['outtime'] = pd.to_datetime(icustays['outtime'])

# Extract the year from 'intime'
icustays['intime_year'] = icustays['intime'].dt.year

# Merge icustays with ICD-9 diagnosis codes
icustays_with_icd = icustays.merge(diagnoses, on='hadm_id', how='left')

# Calculate the mean LOS for each care unit and year
los_stats = icustays_with_icd.groupby(['first_careunit', 'intime_year', 'icd9_code'])['los'].mean().reset_index()

# Create a list of 'ICD9 code - short description' for each unique ICD9 code
icd_label_options = [(code, f" {icd9_dict.get(code, 'Unknown description')}") for code in los_stats['icd9_code'].unique()]

# Create a selection widget for ICD-9 codes
icd_selection = alt.selection_point(
    fields=['icd9_code'],
    bind=alt.binding_select(options=icd_label_options, name="Select ICD Code ")
)

# Create the Altair chart
chart = alt.Chart(los_stats).mark_line(point=True).encode(
    x=alt.X('intime_year:O', title='Year'),
    y=alt.Y('los:Q', title='Mean Length of Stay (LOS)'),
    color=alt.Color('first_careunit:N', title='Care Unit'),
    tooltip=['first_careunit', 'intime_year', 'los', 'icd9_code']  # Add tooltips for interactivity
).add_params(
    icd_selection
).transform_filter(
    icd_selection

).properties(
    title='Average Length Of Stay by Care Unit and Year (Filtered by ICD Code)',
    width=800,
    height=400
)

# Display the chart
chart
```

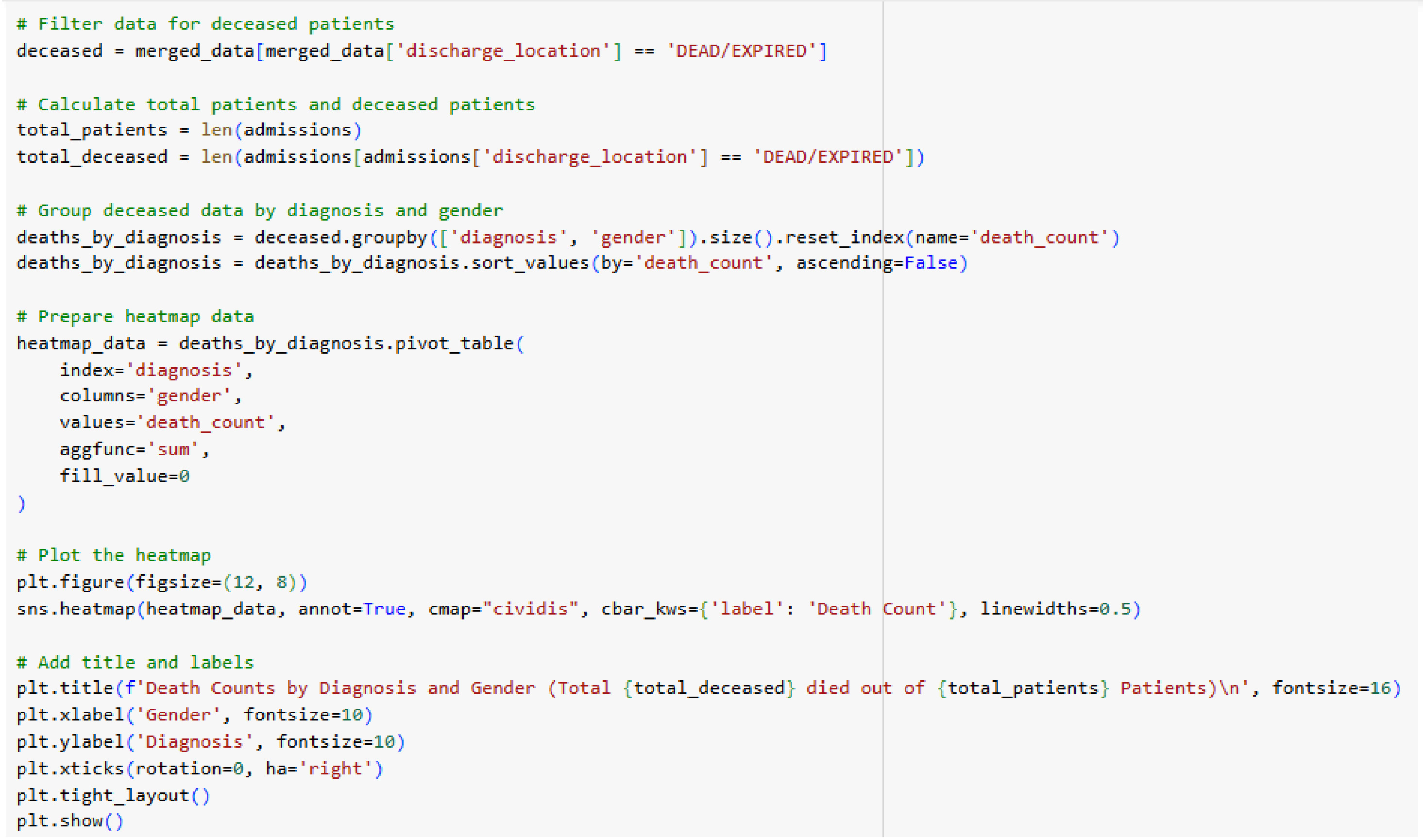
- Converts ICU stay times to datetime format and extracts the year and merges ICU stay data with ICD-9 diagnosis codes.
- Calculates the mean LOS for each care unit and year.
- Creates and display a line chart(altair) showing trends in LOS by care unit and year with selection box for ICD-9 code.

Visual 4 - Death Counts by Diagnosis and Gender

This visualization represents deceased patients categorized by their diagnosis and gender.



Visual 4 - How to recreate

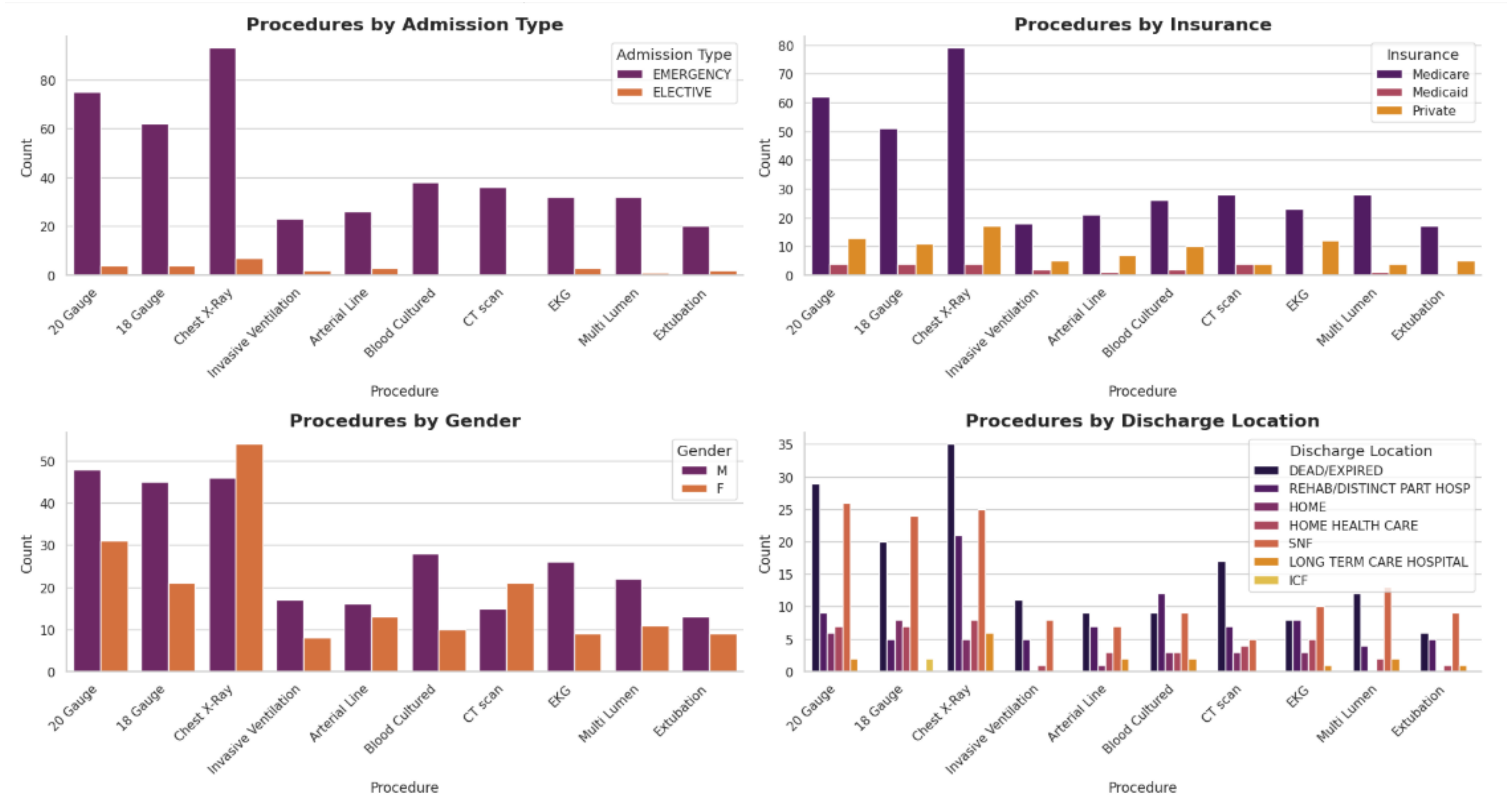


```
# Filter data for deceased patients
deceased = merged_data[merged_data['discharge_location'] == 'DEAD/EXPIRED']

# Calculate total patients and deceased patients
total_patients = len(admissions)
total_deceased = len(admissions[admissions['discharge_location'] == 'DEAD/EXPIRED'])

# Group deceased data by diagnosis and gender
deaths_by_diagnosis = deceased.groupby(['diagnosis', 'gender']).size().reset_index(name='death_count')
deaths_by_diagnosis = deaths_by_diagnosis.sort_values(by='death_count', ascending=False)

# Prepare heatmap data
heatmap_data = deaths_by_diagnosis.pivot_table(
    index='diagnosis',
    columns='gender',
    values='death_count',
    aggfunc='sum',
    fill_value=0
)

# Plot the heatmap
plt.figure(figsize=(12, 8))
sns.heatmap(heatmap_data, annot=True, cmap="cividis", cbar_kws={'label': 'Death Count'}, linewidths=0.5)

# Add title and labels
plt.title(f'Death Counts by Diagnosis and Gender (Total {total_deceased} died out of {total_patients} Patients)\n', fontsize=16)
plt.xlabel('Gender', fontsize=10)
plt.ylabel('Diagnosis', fontsize=10)
plt.xticks(rotation=0, ha='right')
plt.tight_layout()
plt.show()
```

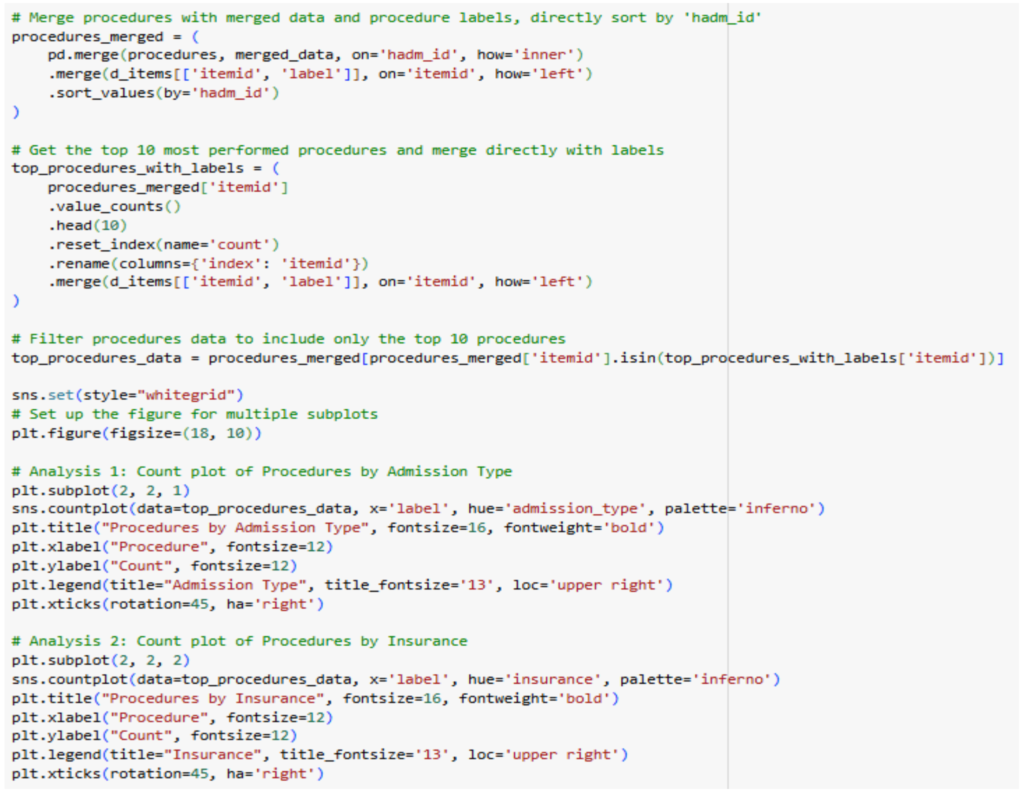
- Filter the data for deceased patients.
- Aggregate the death counts by diagnosis and gender.
- Create and plot a heatmap to visually compare death counts across diagnoses and genders.

Visual 5 - Analysis of Most Performed Procedures

This visualization represents the analysis of the 10 most performed procedures across admission type, insurance types, gender, and discharge location.

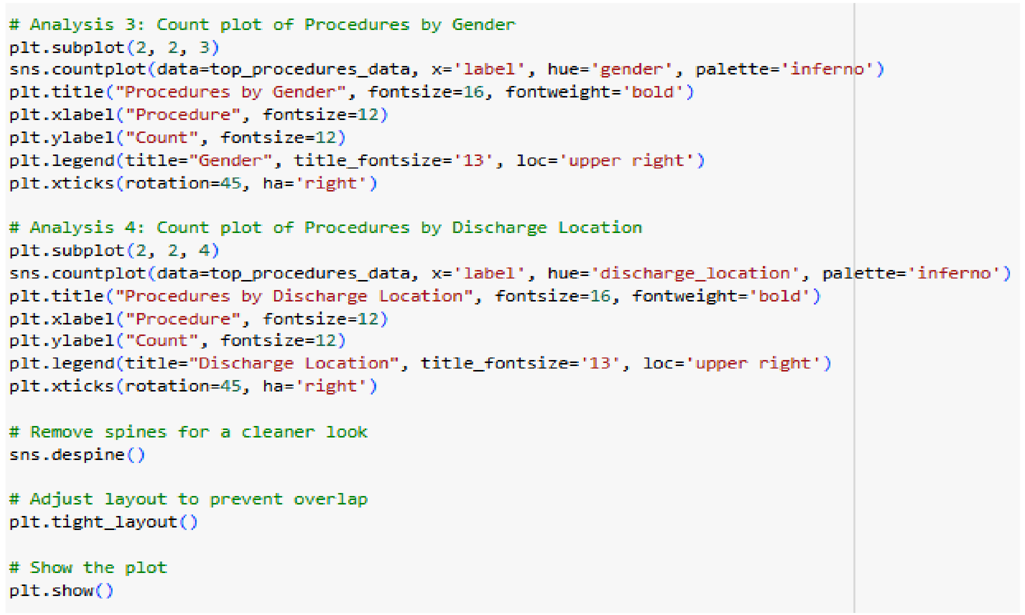


Visual 6 - How to build/recreate



```python
# Merge procedures with merged data and procedure labels, directly sort by 'hadm_id'
procedures_merged = (
    pd.merge(procedures, merged_data, on='hadm_id', how='inner')
    .merge(d_items[['itemid', 'label']], on='itemid', how='left')
    .sort_values(by='hadm_id')
)

# Get the top 10 most performed procedures and merge directly with labels
top_procedures_with_labels = (
    procedures_merged['itemid']
    .value_counts()
    .head(10)
    .reset_index(name='count')
    .rename(columns={'index': 'itemid'})
    .merge(d_items[['itemid', 'label']], on='itemid', how='left')
)

# Filter procedures data to include only the top 10 procedures
top_procedures_data = procedures_merged[procedures_merged['itemid'].isin(top_procedures_with_labels['itemid'])]

sns.set(style="whitegrid")
# Set up the figure for multiple subplots
plt.figure(figsize=(18, 10))

# Analysis 1: Count plot of Procedures by Admission Type
plt.subplot(2, 2, 1)
sns.countplot(data=top_procedures_data, x='label', hue='admission_type', palette='inferno')
plt.title("Procedures by Admission Type", fontsize=16, fontweight='bold')
plt.xlabel("Procedure", fontsize=12)
plt.ylabel("Count", fontsize=12)
plt.legend(title="Admission Type", title_fontsize='13', loc='upper right')
plt.xticks(rotation=45, ha='right')

# Analysis 2: Count plot of Procedures by Insurance
plt.subplot(2, 2, 2)
sns.countplot(data=top_procedures_data, x='label', hue='insurance', palette='inferno')
plt.title("Procedures by Insurance", fontsize=16, fontweight='bold')
plt.xlabel("Procedure", fontsize=12)
plt.ylabel("Count", fontsize=12)
plt.legend(title="Insurance", title_fontsize='13', loc='upper right')
plt.xticks(rotation=45, ha='right')
```

Create a dataframe with existing data and merge procedure data with procedure labels. Extract the top 10 procedures and plot a countplot using this information."



```python
# Analysis 3: Count plot of Procedures by Gender
plt.subplot(2, 2, 3)
sns.countplot(data=top_procedures_data, x='label', hue='gender', palette='inferno')
plt.title("Procedures by Gender", fontsize=16, fontweight='bold')
plt.xlabel("Procedure", fontsize=12)
plt.ylabel("Count", fontsize=12)
plt.legend(title="Gender", title_fontsize='13', loc='upper right')
plt.xticks(rotation=45, ha='right')

# Analysis 4: Count plot of Procedures by Discharge Location
plt.subplot(2, 2, 4)
sns.countplot(data=top_procedures_data, x='label', hue='discharge_location', palette='inferno')
plt.title("Procedures by Discharge Location", fontsize=16, fontweight='bold')
plt.xlabel("Procedure", fontsize=12)
plt.ylabel("Count", fontsize=12)
plt.legend(title="Discharge Location", title_fontsize='13', loc='upper right')
plt.xticks(rotation=45, ha='right')

# Remove spines for a cleaner look
sns.despine()

# Adjust layout to prevent overlap
plt.tight_layout()

# Show the plot
plt.show()
```