USING LLM TO PREDICTING ACUTE KIDNEY INJURY BASED ON THE MIMIC-IV CLINICAL DATABASE

Joseph Barnes

AI in Healthcare

University of Texas at Austin

OVERVIEW

This exercise outlines a comprehensive approach to predicting Acute Kidney Injury (AKI) using Large Language Models (LLMs), specifically Microsoft Azure OpenAI GPT-4, leveraging patient data from the MIMIC-IV database. The process encompasses the following steps:

Data Loading (MIMIC IV)

- Load various datasets from MIMIC-IV, including patient demographics, admissions, diagnoses, lab events, chart events, output events, and discharge notes, converting them to parquet format if not already done.

Data Preparation and Preprocessing

- Clean and preprocess the datasets by dropping missing values, converting date columns to datetime objects, and ensuring the integrity of the data (e.g., admission times before discharge times).

Feature Engineering

- Calculate patient age at admission and filter lab events for chemistry labs, applying conditions to create specific chemical indicators.
- Extract serum creatinine baselines and determine the presence of chronic kidney disease (CKD) based on ICD codes.
- Compile weight and urine output events, calculating relevant metrics like urine output over 6 hours adjusted for patient weight.

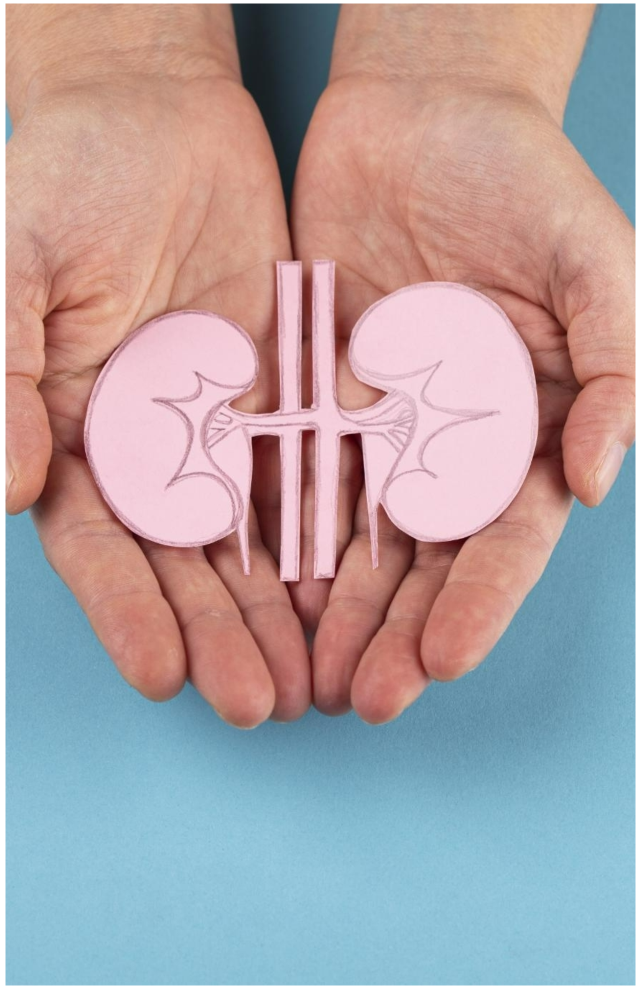
Using GPT-4 for AKI Prediction

- Initialize Azure OpenAI Client and create functions to interact with GPT-4, formulating questions and processing responses.
- Prepare patient data and discharge notes for input into GPT-4, asking it to predict the risk of AKI, explain its reasoning, and if positive, detail a timeline leading up to AKI.
- Evaluate the prediction accuracy against actual AKI risk based on KDIGO criteria and existing diagnoses, employing functions to predict AKI risk for a sample of patient admissions and add labels based on actual outcomes.

IMPORTANT NOTE ABOUT USING GPT-4 & MIMIC IV DATABASE

In this exercise, a private instance of the Microsoft Azure OpenAI GPT-4 model was used to provide chat completions for a set of prompts based on patient data provided by the MIMIC IV Clinical Database. In order to remain in compliance with terms and conditions for use of the MIMIC IV data in this manner, content logging was disable for this instance of Azure OpenAI.

THE CLINICAL CHALLENGE OF ACUTE KIDNEY INJURY (AKI)



Acute Kidney Injury is a common problem with a high impact on patient outcomes

Early detection can significantly improve patient outcomes

Early detection has the potential to reduce morbidity and healthcare costs

HOW IS AKI DIAGNOSED?

KDIGO, which stands for Kidney Disease: Improving Global Outcomes, is a global organization focused on improving patient outcomes in kidney disease through the development and implementation of global clinical practice guidelines. When it comes to Acute Kidney Injury (AKI), KDIGO has established specific criteria to standardize the diagnosis and staging of AKI, aiming to facilitate early detection, management, and interventions to improve outcomes.

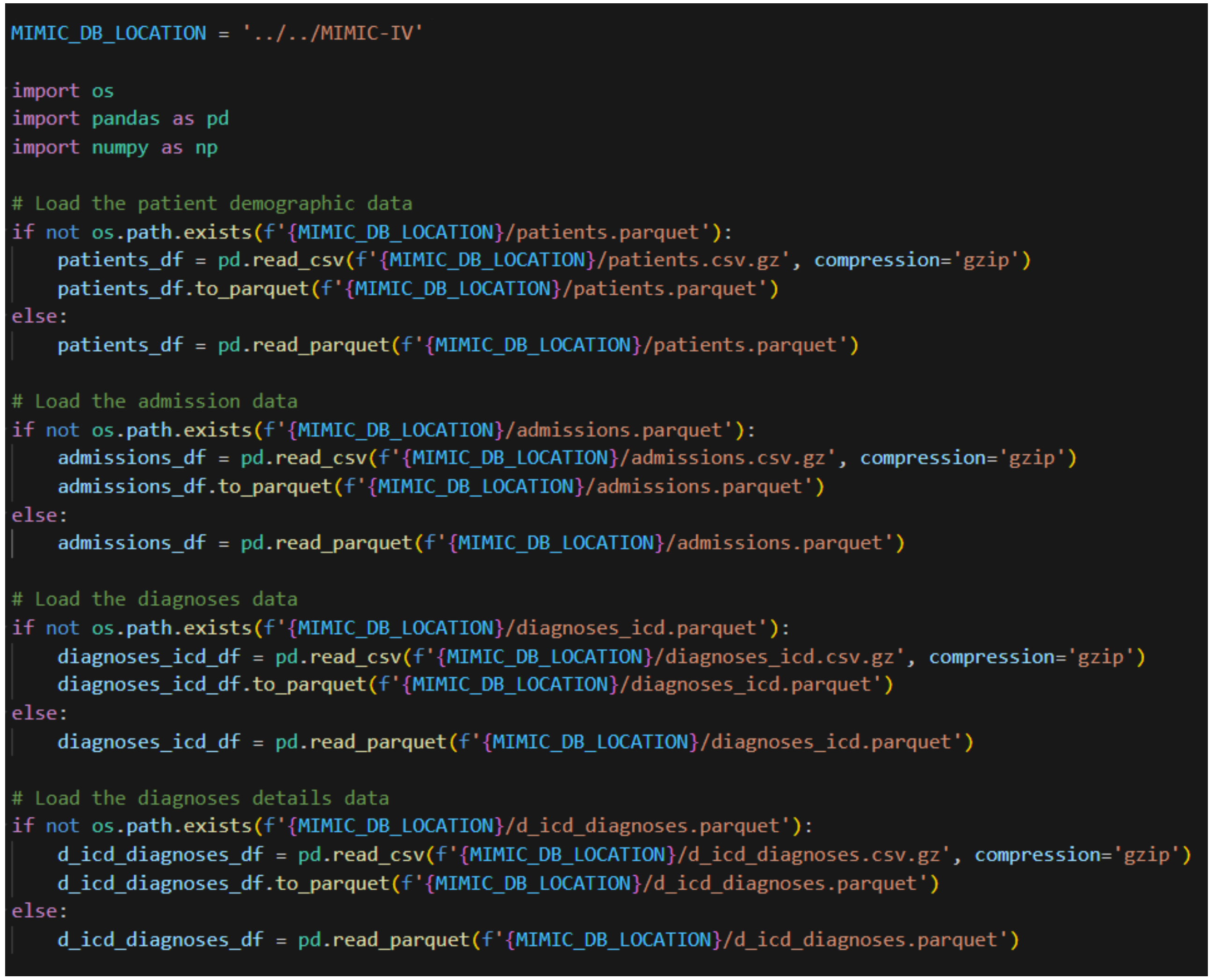
The KDIGO criteria for diagnosing AKI are based on changes in serum creatinine (SCr) levels, urine output, or both, and are divided into three main criteria:

1. **Increase in Serum Creatinine by ≥0.3 mg/dl (≥26.5 µmol/l) within 48 hours:**
   This criterion is based on the absolute increase in SCr over a short period. An increase of at least 0.3 mg/dl suggests a significant decline in kidney function, indicating AKI. This rapid change is critical for early detection and intervention.

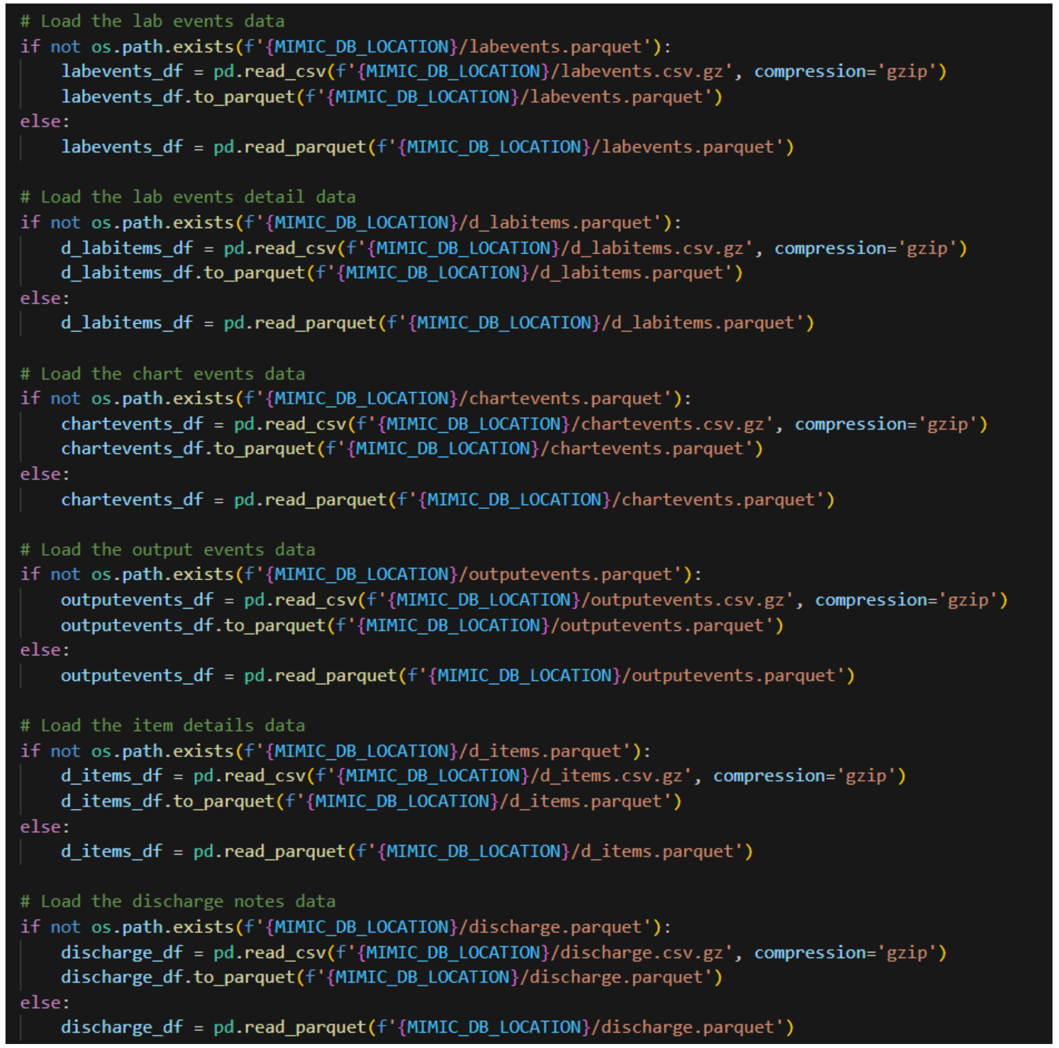
2. **Increase in Serum Creatinine to ≥1.5 times baseline, which is known or presumed to have occurred within the prior 7 days:**
   This criterion looks at the relative increase in SCr from the patient's baseline level. A 1.5-fold or greater increase in SCr within the last week is considered indicative of AKI. The baseline SCr may be the lowest SCr measured during the hospital admission or an estimated baseline SCr calculated using a formula (e.g., MDRD formula) if no prior SCr values are available.

3. **Urine volume < 0.5 ml/kg/hr for 6 hours:**
   This criterion focuses on the patient's urine output as an indicator of kidney function. A urine output of less than 0.5 ml/kg/hr for at least 6 hours is suggestive of AKI, reflecting decreased kidney filtration and excretion capabilities. Monitoring urine output is especially important in critical care settings where patients may have varying fluid balances and requirements.

LOADING THE MIMIC-IV DATA TABLES

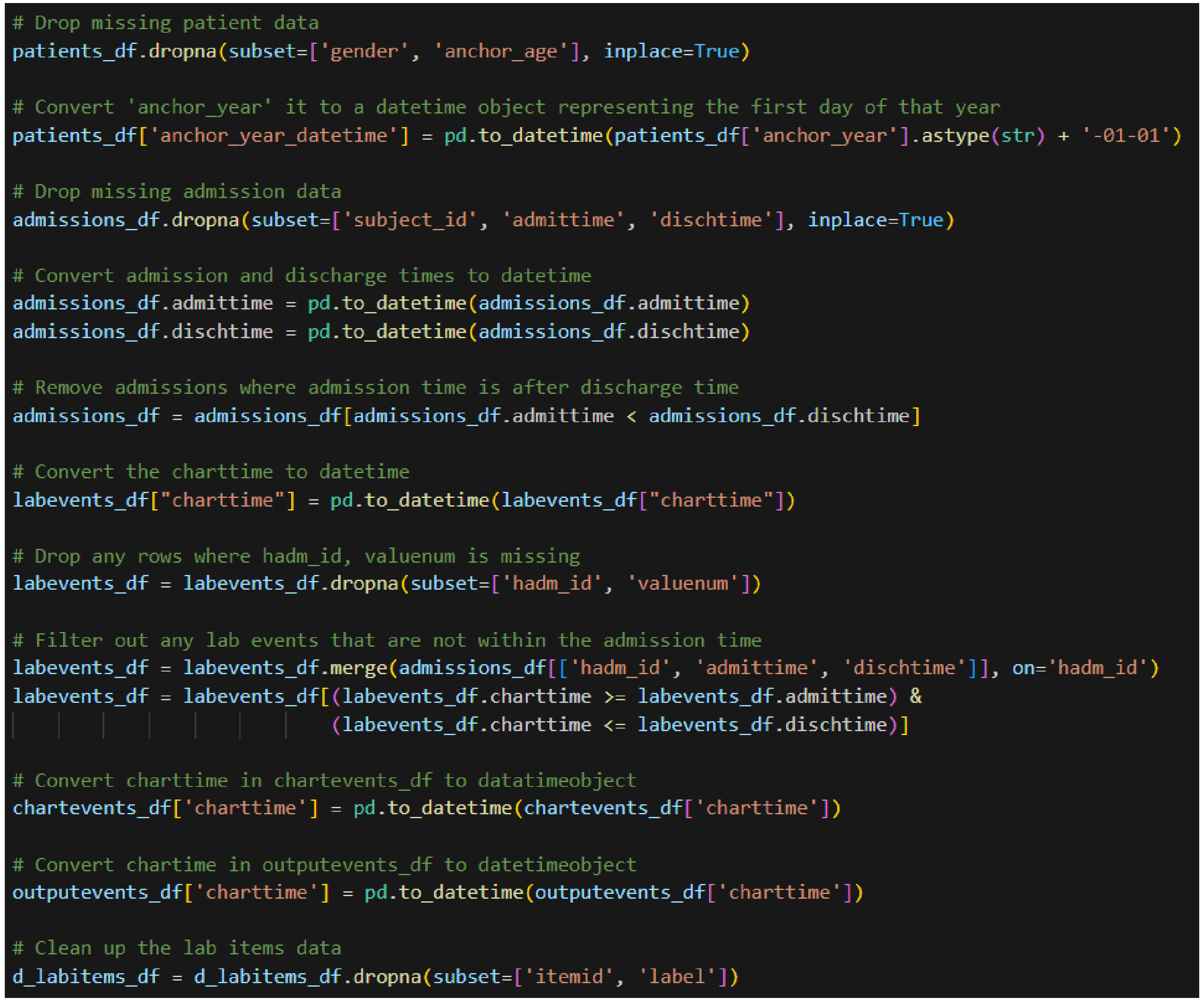


```python
MIMIC_DB_LOCATION = '../../MIMIC-IV'

import os
import pandas as pd
import numpy as np

# Load the patient demographic data
if not os.path.exists(f'{MIMIC_DB_LOCATION}/patients.parquet'):
    patients_df = pd.read_csv(f'{MIMIC_DB_LOCATION}/patients.csv.gz', compression='gzip')
    patients_df.to_parquet(f'{MIMIC_DB_LOCATION}/patients.parquet')
else:
    patients_df = pd.read_parquet(f'{MIMIC_DB_LOCATION}/patients.parquet')

# Load the admission data
if not os.path.exists(f'{MIMIC_DB_LOCATION}/admissions.parquet'):
    admissions_df = pd.read_csv(f'{MIMIC_DB_LOCATION}/admissions.csv.gz', compression='gzip')
    admissions_df.to_parquet(f'{MIMIC_DB_LOCATION}/admissions.parquet')
else:
    admissions_df = pd.read_parquet(f'{MIMIC_DB_LOCATION}/admissions.parquet')

# Load the diagnoses data
if not os.path.exists(f'{MIMIC_DB_LOCATION}/diagnoses_icd.parquet'):
    diagnoses_icd_df = pd.read_csv(f'{MIMIC_DB_LOCATION}/diagnoses_icd.csv.gz', compression='gzip')
    diagnoses_icd_df.to_parquet(f'{MIMIC_DB_LOCATION}/diagnoses_icd.parquet')
else:
    diagnoses_icd_df = pd.read_parquet(f'{MIMIC_DB_LOCATION}/diagnoses_icd.parquet')

# Load the diagnoses details data
if not os.path.exists(f'{MIMIC_DB_LOCATION}/d_icd_diagnoses.parquet'):
    d_icd_diagnoses_df = pd.read_csv(f'{MIMIC_DB_LOCATION}/d_icd_diagnoses.csv.gz', compression='gzip')
    d_icd_diagnoses_df.to_parquet(f'{MIMIC_DB_LOCATION}/d_icd_diagnoses.parquet')
else:
    d_icd_diagnoses_df = pd.read_parquet(f'{MIMIC_DB_LOCATION}/d_icd_diagnoses.parquet')
```



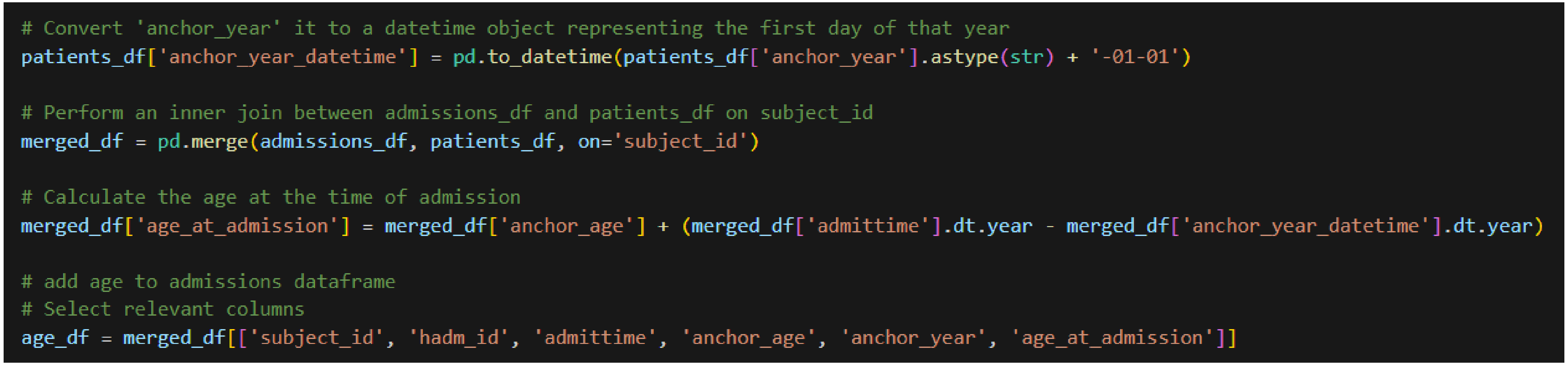
```python
# Load the lab events data
if not os.path.exists(f'{MIMIC_DB_LOCATION}/labevents.parquet'):
    labevents_df = pd.read_csv(f'{MIMIC_DB_LOCATION}/labevents.csv.gz', compression='gzip')
    labevents_df.to_parquet(f'{MIMIC_DB_LOCATION}/labevents.parquet')
else:
    labevents_df = pd.read_parquet(f'{MIMIC_DB_LOCATION}/labevents.parquet')

# Load the lab events detail data
if not os.path.exists(f'{MIMIC_DB_LOCATION}/d_labitems.parquet'):
    d_labitems_df = pd.read_csv(f'{MIMIC_DB_LOCATION}/d_labitems.csv.gz', compression='gzip')
    d_labitems_df.to_parquet(f'{MIMIC_DB_LOCATION}/d_labitems.parquet')
else:
    d_labitems_df = pd.read_parquet(f'{MIMIC_DB_LOCATION}/d_labitems.parquet')

# Load the chart events data
if not os.path.exists(f'{MIMIC_DB_LOCATION}/chartevents.parquet'):
    chartevents_df = pd.read_csv(f'{MIMIC_DB_LOCATION}/chartevents.csv.gz', compression='gzip')
    chartevents_df.to_parquet(f'{MIMIC_DB_LOCATION}/chartevents.parquet')
else:
    chartevents_df = pd.read_parquet(f'{MIMIC_DB_LOCATION}/chartevents.parquet')

# Load the output events data
if not os.path.exists(f'{MIMIC_DB_LOCATION}/outputevents.parquet'):
    outputevents_df = pd.read_csv(f'{MIMIC_DB_LOCATION}/outputevents.csv.gz', compression='gzip')
    outputevents_df.to_parquet(f'{MIMIC_DB_LOCATION}/outputevents.parquet')
else:
    outputevents_df = pd.read_parquet(f'{MIMIC_DB_LOCATION}/outputevents.parquet')

# Load the item details data
if not os.path.exists(f'{MIMIC_DB_LOCATION}/d_items.parquet'):
    d_items_df = pd.read_csv(f'{MIMIC_DB_LOCATION}/d_items.csv.gz', compression='gzip')
    d_items_df.to_parquet(f'{MIMIC_DB_LOCATION}/d_items.parquet')
else:
    d_items_df = pd.read_parquet(f'{MIMIC_DB_LOCATION}/d_items.parquet')

# Load the discharge notes data
if not os.path.exists(f'{MIMIC_DB_LOCATION}/discharge.parquet'):
    discharge_df = pd.read_csv(f'{MIMIC_DB_LOCATION}/discharge.csv.gz', compression='gzip')
    discharge_df.to_parquet(f'{MIMIC_DB_LOCATION}/discharge.parquet')
else:
    discharge_df = pd.read_parquet(f'{MIMIC_DB_LOCATION}/discharge.parquet')
```

CLEANING AND PREPROCESSING THE DATA



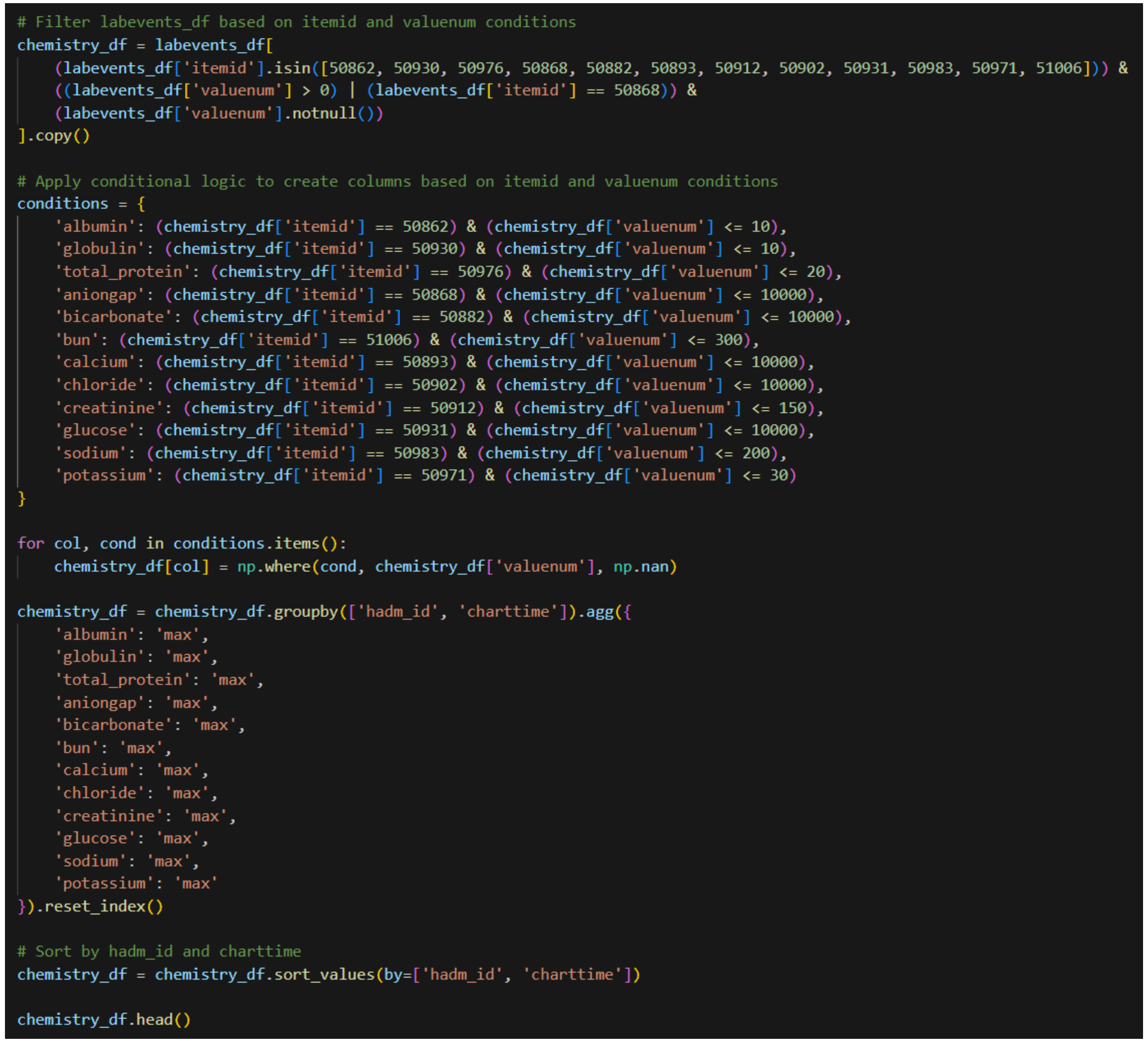
```
# Drop missing patient data
patients_df.dropna(subset=['gender', 'anchor_age'], inplace=True)

# Convert 'anchor_year' it to a datetime object representing the first day of that year
patients_df['anchor_year_datetime'] = pd.to_datetime(patients_df['anchor_year'].astype(str) + '-01-01')

# Drop missing admission data
admissions_df.dropna(subset=['subject_id', 'admittime', 'dischtime'], inplace=True)

# Convert admission and discharge times to datetime
admissions_df.admittime = pd.to_datetime(admissions_df.admittime)
admissions_df.dischtime = pd.to_datetime(admissions_df.dischtime)

# Remove admissions where admission time is after discharge time
admissions_df = admissions_df[admissions_df.admittime < admissions_df.dischtime]

# Convert the charttime to datetime
labevents_df["charttime"] = pd.to_datetime(labevents_df["charttime"])

# Drop any rows where hadm_id, valuenum is missing
labevents_df = labevents_df.dropna(subset=['hadm_id', 'valuenum'])

# Filter out any lab events that are not within the admission time
labevents_df = labevents_df.merge(admissions_df[['hadm_id', 'admittime', 'dischtime']], on='hadm_id')
labevents_df = labevents_df[(labevents_df.charttime >= labevents_df.admittime) &
                            (labevents_df.charttime <= labevents_df.dischtime)]

# Convert charttime in chartevents_df to datatimeobject
chartevents_df['charttime'] = pd.to_datetime(chartevents_df['charttime'])

# Convert chartime in outputevents_df to datetimeobject
outputevents_df['charttime'] = pd.to_datetime(outputevents_df['charttime'])

# Clean up the lab items data
d_labitems_df = d_labitems_df.dropna(subset=['itemid', 'label'])
```

FEATURE ENGINEERING: AGE AT TIME OF ADMISSIONS



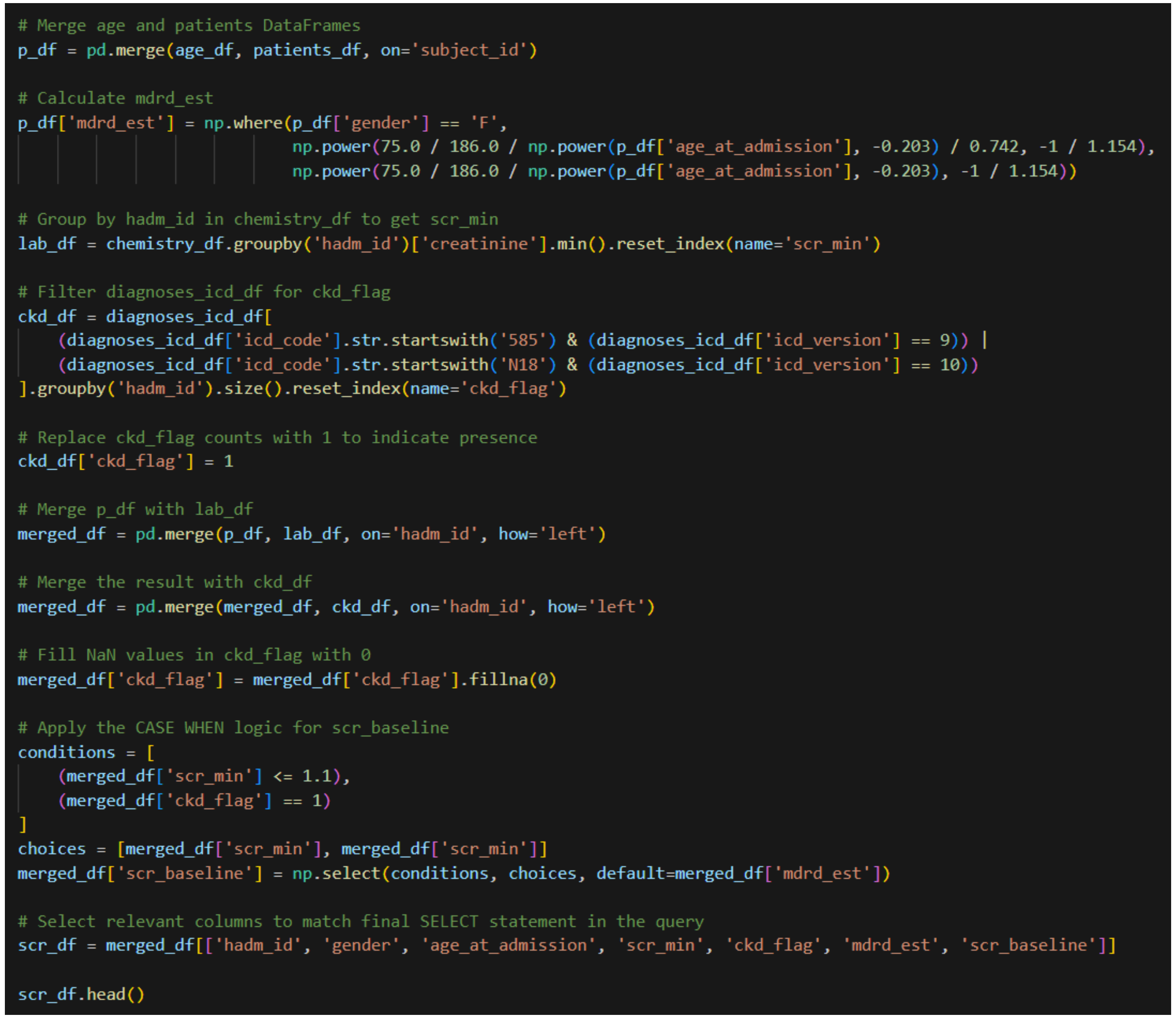
```python
# Convert 'anchor_year' it to a datetime object representing the first day of that year
patients_df['anchor_year_datetime'] = pd.to_datetime(patients_df['anchor_year'].astype(str) + '-01-01')

# Perform an inner join between admissions_df and patients_df on subject_id
merged_df = pd.merge(admissions_df, patients_df, on='subject_id')

# Calculate the age at the time of admission
merged_df['age_at_admission'] = merged_df['anchor_age'] + (merged_df['admittime'].dt.year - merged_df['anchor_year_datetime'].dt.year)

# add age to admissions dataframe
# Select relevant columns
age_df = merged_df[['subject_id', 'hadm_id', 'admittime', 'anchor_age', 'anchor_year', 'age_at_admission']]
```

FEATURE ENGINEERING: CHEMISTRY LAB EVENTS



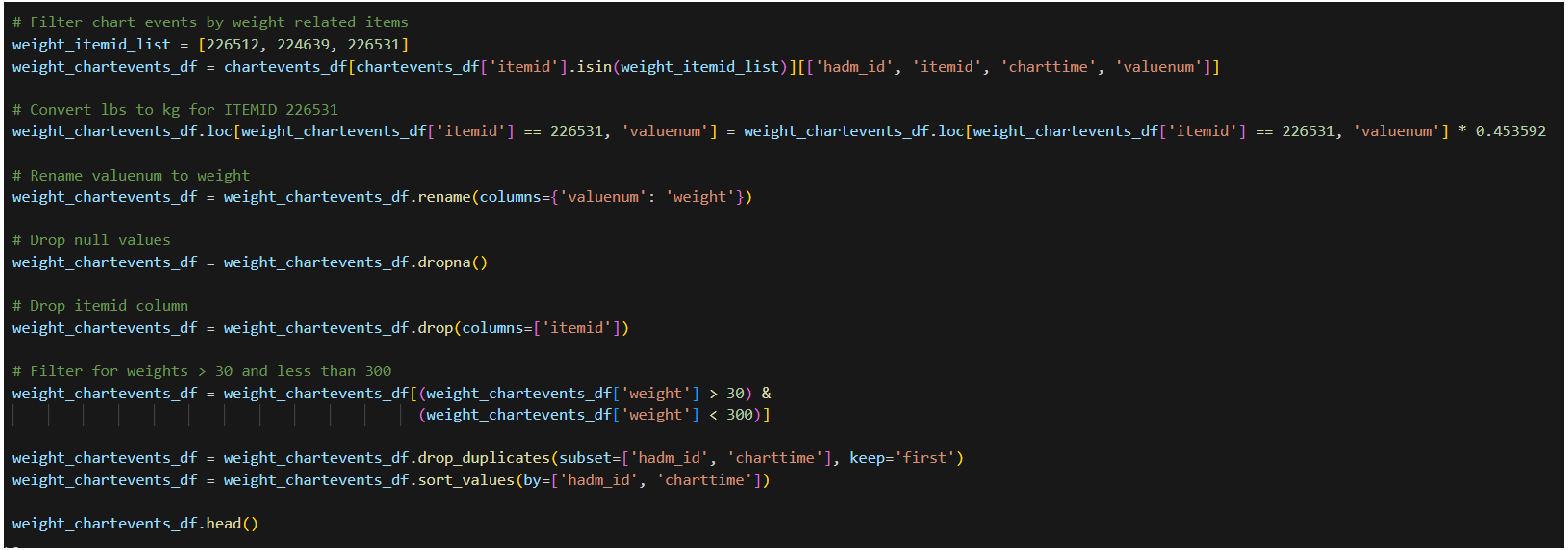
```python
# Filter labevents_df based on itemid and valuenum conditions
chemistry_df = labevents_df[
    (labevents_df['itemid'].isin([50862, 50930, 50976, 50868, 50882, 50893, 50912, 50902, 50931, 50983, 50971, 51006])) &
    ((labevents_df['valuenum'] > 0) | (labevents_df['itemid'] == 50868)) &
    (labevents_df['valuenum'].notnull())
].copy()

# Apply conditional logic to create columns based on itemid and valuenum conditions
conditions = {
    'albumin': (chemistry_df['itemid'] == 50862) & (chemistry_df['valuenum'] <= 10),
    'globulin': (chemistry_df['itemid'] == 50930) & (chemistry_df['valuenum'] <= 10),
    'total_protein': (chemistry_df['itemid'] == 50976) & (chemistry_df['valuenum'] <= 20),
    'aniongap': (chemistry_df['itemid'] == 50868) & (chemistry_df['valuenum'] <= 10000),
    'bicarbonate': (chemistry_df['itemid'] == 50882) & (chemistry_df['valuenum'] <= 10000),
    'bun': (chemistry_df['itemid'] == 51006) & (chemistry_df['valuenum'] <= 300),
    'calcium': (chemistry_df['itemid'] == 50893) & (chemistry_df['valuenum'] <= 10000),
    'chloride': (chemistry_df['itemid'] == 50902) & (chemistry_df['valuenum'] <= 10000),
    'creatinine': (chemistry_df['itemid'] == 50912) & (chemistry_df['valuenum'] <= 150),
    'glucose': (chemistry_df['itemid'] == 50931) & (chemistry_df['valuenum'] <= 10000),
    'sodium': (chemistry_df['itemid'] == 50983) & (chemistry_df['valuenum'] <= 200),
    'potassium': (chemistry_df['itemid'] == 50971) & (chemistry_df['valuenum'] <= 30)
}

for col, cond in conditions.items():
    chemistry_df[col] = np.where(cond, chemistry_df['valuenum'], np.nan)

chemistry_df = chemistry_df.groupby(['hadm_id', 'charttime']).agg({
    'albumin': 'max',
    'globulin': 'max',
    'total_protein': 'max',
    'aniongap': 'max',
    'bicarbonate': 'max',
    'bun': 'max',
    'calcium': 'max',
    'chloride': 'max',
    'creatinine': 'max',
    'glucose': 'max',
    'sodium': 'max',
    'potassium': 'max'
}).reset_index()

# Sort by hadm_id and charttime
chemistry_df = chemistry_df.sort_values(by=['hadm_id', 'charttime'])

chemistry_df.head()
```

FEATURE ENGINEERING: SERUM CREATININE BASELINES



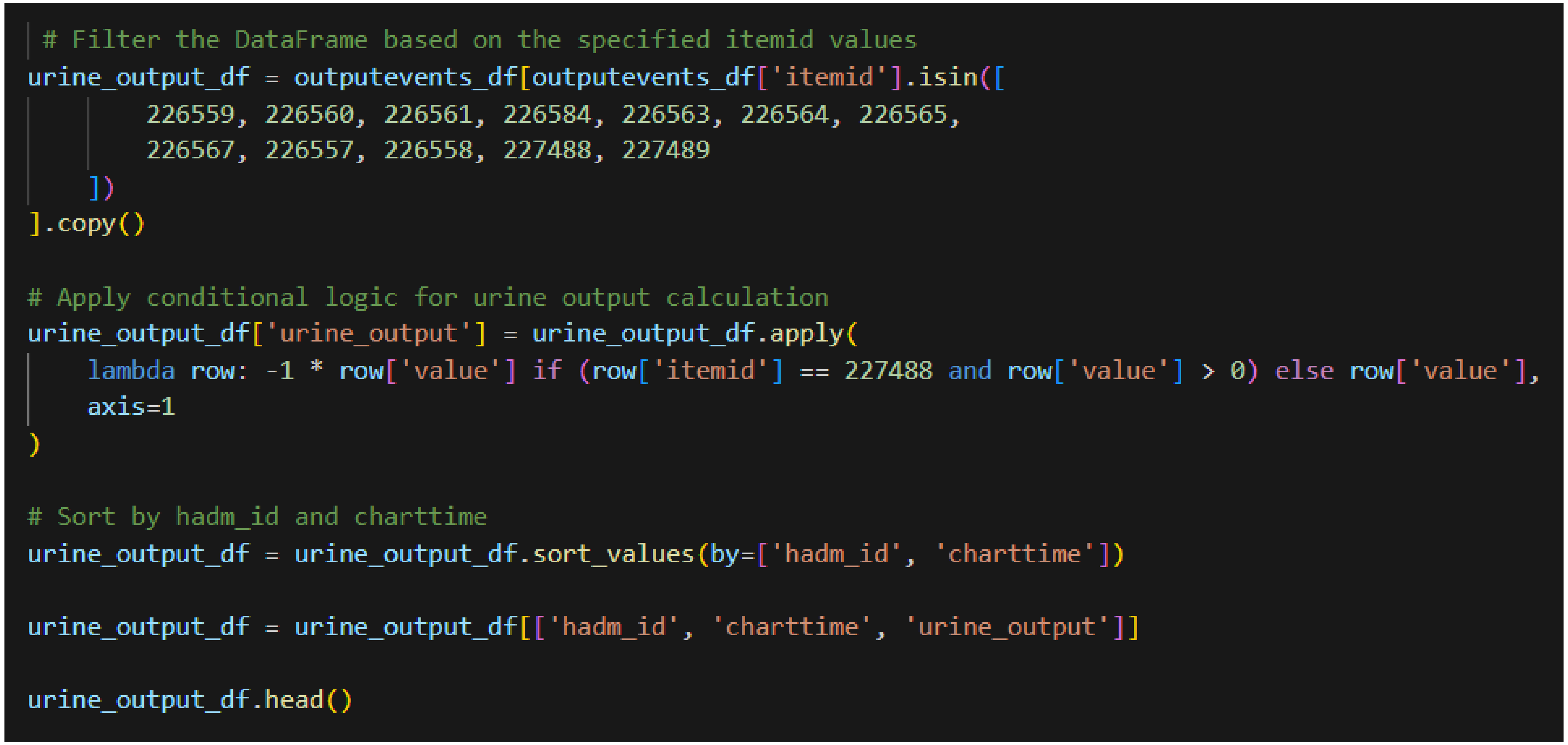
```python
# Merge age and patients DataFrames
p_df = pd.merge(age_df, patients_df, on='subject_id')

# Calculate mdrd_est
p_df['mdrd_est'] = np.where(p_df['gender'] == 'F',
                            np.power(75.0 / 186.0 / np.power(p_df['age_at_admission'], -0.203) / 0.742, -1 / 1.154),
                            np.power(75.0 / 186.0 / np.power(p_df['age_at_admission'], -0.203), -1 / 1.154))

# Group by hadm_id in chemistry_df to get scr_min
lab_df = chemistry_df.groupby('hadm_id')['creatinine'].min().reset_index(name='scr_min')

# Filter diagnoses_icd_df for ckd_flag
ckd_df = diagnoses_icd_df[
    (diagnoses_icd_df['icd_code'].str.startswith('585') & (diagnoses_icd_df['icd_version'] == 9)) |
    (diagnoses_icd_df['icd_code'].str.startswith('N18') & (diagnoses_icd_df['icd_version'] == 10))
].groupby('hadm_id').size().reset_index(name='ckd_flag')

# Replace ckd_flag counts with 1 to indicate presence
ckd_df['ckd_flag'] = 1

# Merge p_df with lab_df
merged_df = pd.merge(p_df, lab_df, on='hadm_id', how='left')

# Merge the result with ckd_df
merged_df = pd.merge(merged_df, ckd_df, on='hadm_id', how='left')

# Fill NaN values in ckd_flag with 0
merged_df['ckd_flag'] = merged_df['ckd_flag'].fillna(0)

# Apply the CASE WHEN logic for scr_baseline
conditions = [
    (merged_df['scr_min'] <= 1.1),
    (merged_df['ckd_flag'] == 1)
]
choices = [merged_df['scr_min'], merged_df['scr_min']]
merged_df['scr_baseline'] = np.select(conditions, choices, default=merged_df['mdrd_est'])

# Select relevant columns to match final SELECT statement in the query
scr_df = merged_df[['hadm_id', 'gender', 'age_at_admission', 'scr_min', 'ckd_flag', 'mdrd_est', 'scr_baseline']]

scr_df.head()
```

FEATURE ENGINEERING: WEIGHT EVENTS



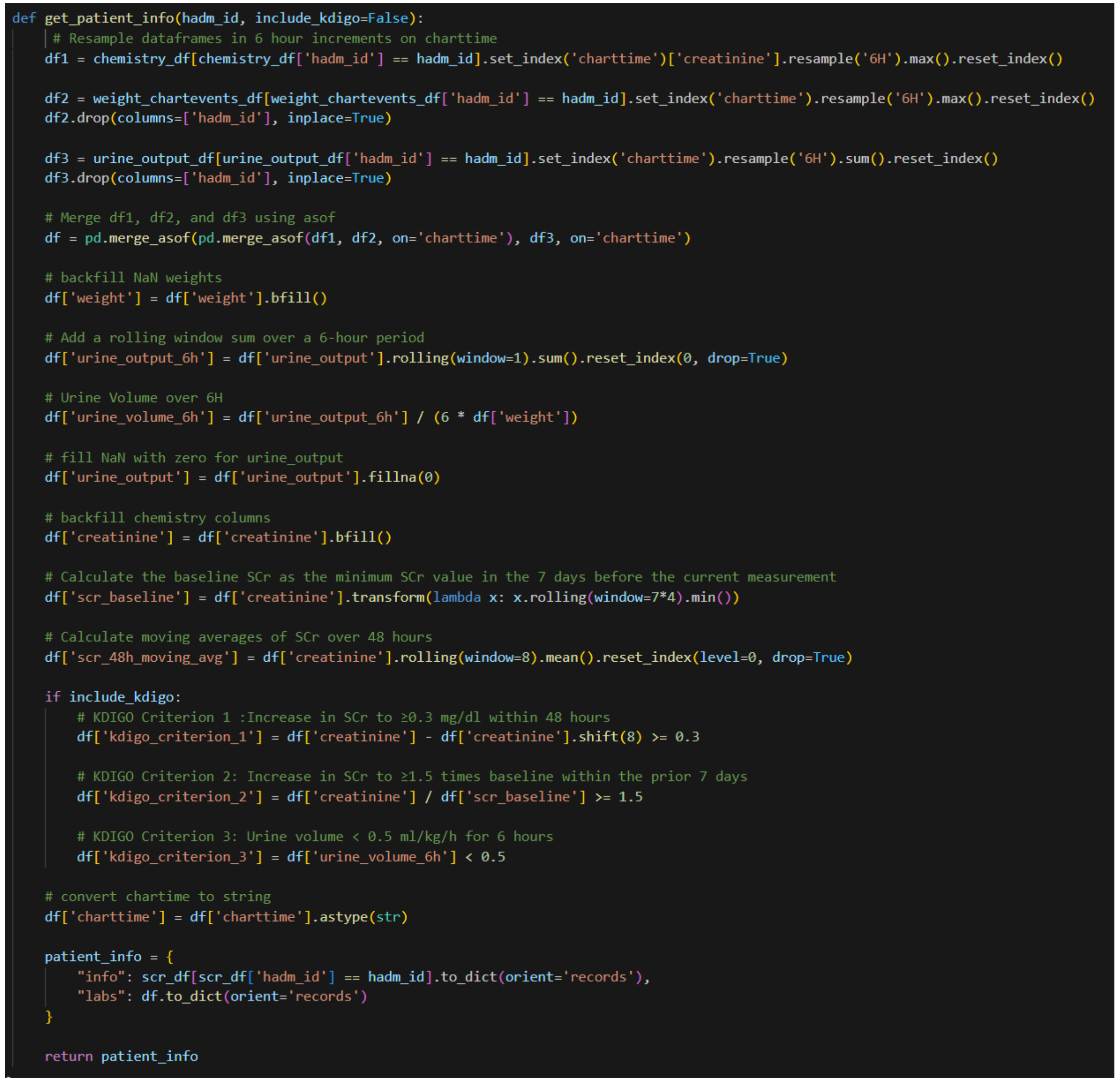
```python
# Filter chart events by weight related items
weight_itemid_list = [226512, 224639, 226531]
weight_chartevents_df = chartevents_df[chartevents_df['itemid'].isin(weight_itemid_list)][['hadm_id', 'itemid', 'charttime', 'valuenum']]

# Convert lbs to kg for ITEMID 226531
weight_chartevents_df.loc[weight_chartevents_df['itemid'] == 226531, 'valuenum'] = weight_chartevents_df.loc[weight_chartevents_df['itemid'] == 226531, 'valuenum'] * 0.453592

# Rename valuenum to weight
weight_chartevents_df = weight_chartevents_df.rename(columns={'valuenum': 'weight'})

# Drop null values
weight_chartevents_df = weight_chartevents_df.dropna()

# Drop itemid column
weight_chartevents_df = weight_chartevents_df.drop(columns=['itemid'])

# Filter for weights > 30 and less than 300
weight_chartevents_df = weight_chartevents_df[(weight_chartevents_df['weight'] > 30) &
                                              (weight_chartevents_df['weight'] < 300)]

weight_chartevents_df = weight_chartevents_df.drop_duplicates(subset=['hadm_id', 'charttime'], keep='first')
weight_chartevents_df = weight_chartevents_df.sort_values(by=['hadm_id', 'charttime'])

weight_chartevents_df.head()
```

FEATURE ENGINEERING: URINE OUTPUT EVENTS



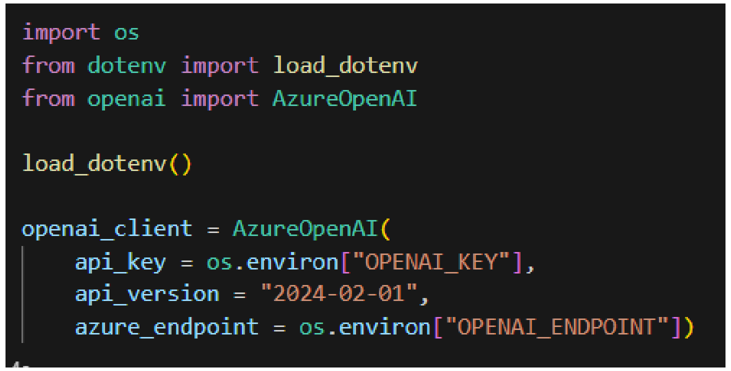
```python
# Filter the DataFrame based on the specified itemid values
urine_output_df = outputevents_df[outputevents_df['itemid'].isin([
        226559, 226560, 226561, 226584, 226563, 226564, 226565,
        226567, 226557, 226558, 227488, 227489
    ])
].copy()

# Apply conditional logic for urine output calculation
urine_output_df['urine_output'] = urine_output_df.apply(
    lambda row: -1 * row['value'] if (row['itemid'] == 227488 and row['value'] > 0) else row['value'],
    axis=1
)

# Sort by hadm_id and charttime
urine_output_df = urine_output_df.sort_values(by=['hadm_id', 'charttime'])

urine_output_df = urine_output_df[['hadm_id', 'charttime', 'urine_output']]

urine_output_df.head()
```

FEATURE ENGINEERING: PER PATIENT FEATURES

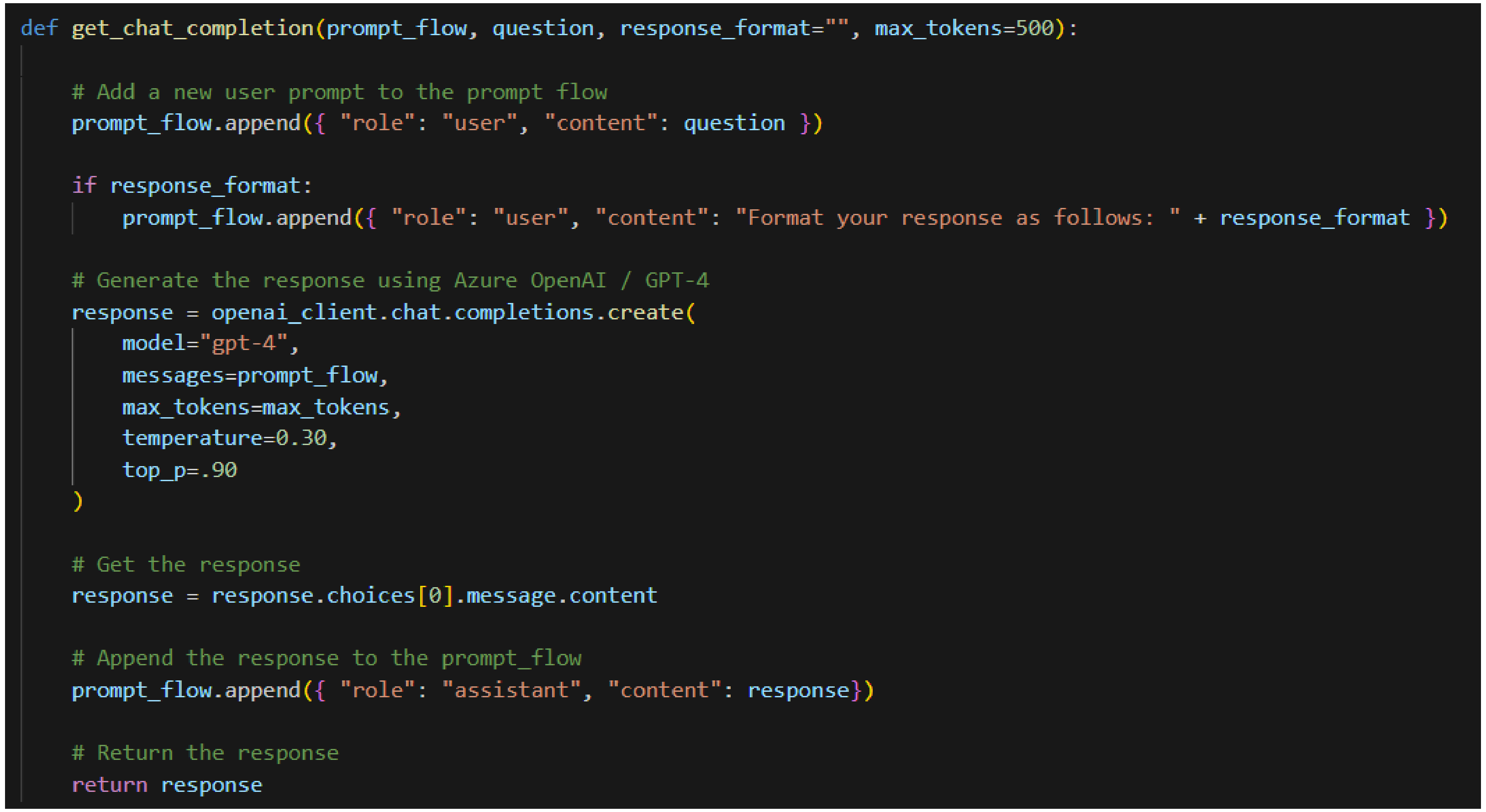


```python
def get_patient_info(hadm_id, include_kdigo=False):
    # Resample dataframes in 6 hour increments on charttime
    df1 = chemistry_df[chemistry_df['hadm_id'] == hadm_id].set_index('charttime')['creatinine'].resample('6H').max().reset_index()

    df2 = weight_chartevents_df[weight_chartevents_df['hadm_id'] == hadm_id].set_index('charttime').resample('6H').max().reset_index()
    df2.drop(columns=['hadm_id'], inplace=True)

    df3 = urine_output_df[urine_output_df['hadm_id'] == hadm_id].set_index('charttime').resample('6H').sum().reset_index()
    df3.drop(columns=['hadm_id'], inplace=True)

    # Merge df1, df2, and df3 using asof
    df = pd.merge_asof(pd.merge_asof(df1, df2, on='charttime'), df3, on='charttime')

    # backfill NaN weights
    df['weight'] = df['weight'].bfill()

    # Add a rolling window sum over a 6-hour period
    df['urine_output_6h'] = df['urine_output'].rolling(window=1).sum().reset_index(0, drop=True)

    # Urine Volume over 6H
    df['urine_volume_6h'] = df['urine_output_6h'] / (6 * df['weight'])

    # fill NaN with zero for urine_output
    df['urine_output'] = df['urine_output'].fillna(0)

    # backfill chemistry columns
    df['creatinine'] = df['creatinine'].bfill()

    # Calculate the baseline SCr as the minimum SCr value in the 7 days before the current measurement
    df['scr_baseline'] = df['creatinine'].transform(lambda x: x.rolling(window=7*4).min())

    # Calculate moving averages of SCr over 48 hours
    df['scr_48h_moving_avg'] = df['creatinine'].rolling(window=8).mean().reset_index(level=0, drop=True)

    if include_kdigo:
        # KDIGO Criterion 1 :Increase in SCr to ≥0.3 mg/dl within 48 hours
        df['kdigo_criterion_1'] = df['creatinine'] - df['creatinine'].shift(8) >= 0.3

        # KDIGO Criterion 2: Increase in SCr to ≥1.5 times baseline within the prior 7 days
        df['kdigo_criterion_2'] = df['creatinine'] / df['scr_baseline'] >= 1.5

        # KDIGO Criterion 3: Urine volume < 0.5 ml/kg/h for 6 hours
        df['kdigo_criterion_3'] = df['urine_volume_6h'] < 0.5

    # convert chartime to string
    df['charttime'] = df['charttime'].astype(str)

    patient_info = {
        "info": scr_df[scr_df['hadm_id'] == hadm_id].to_dict(orient='records'),
        "labs": df.to_dict(orient='records')
    }

    return patient_info
```

AZURE OPEN AI (GPT-4): INITIALIZATION AND CHAT COMPLETE FUNCTION

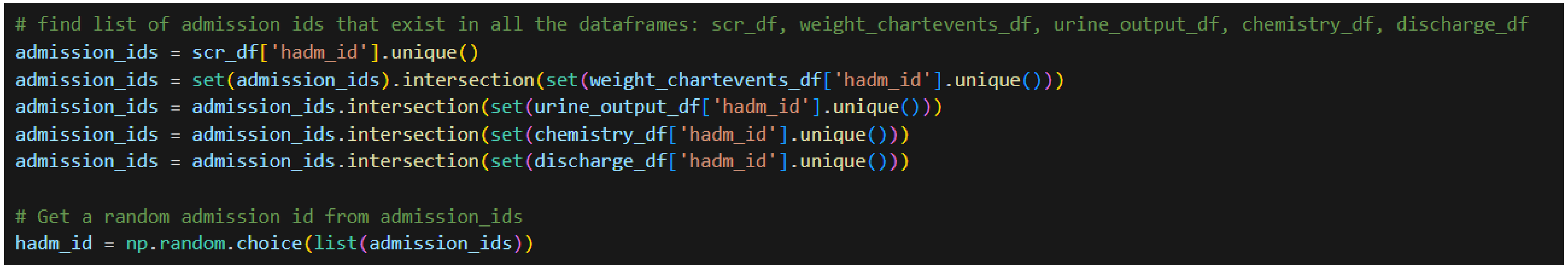


```python
import os
from dotenv import load_dotenv
from openai import AzureOpenAI

load_dotenv()

openai_client = AzureOpenAI(
    api_key = os.environ["OPENAI_KEY"],
    api_version = "2024-02-01",
    azure_endpoint = os.environ["OPENAI_ENDPOINT"])
```



```python
def get_chat_completion(prompt_flow, question, response_format="", max_tokens=500):

    # Add a new user prompt to the prompt flow
    prompt_flow.append({ "role": "user", "content": question })

    if response_format:
        prompt_flow.append({ "role": "user", "content": "Format your response as follows: " + response_format })

    # Generate the response using Azure OpenAI / GPT-4
    response = openai_client.chat.completions.create(
        model="gpt-4",
        messages=prompt_flow,
        max_tokens=max_tokens,
        temperature=0.30,
        top_p=.90
    )

    # Get the response
    response = response.choices[0].message.content

    # Append the response to the prompt_flow
    prompt_flow.append({ "role": "assistant", "content": response})

    # Return the response
    return response
```

GET LIST OF ADMISSIONS THAT EXIST ACROSS RELEVANT DATA FRAMES



```python
# find list of admission ids that exist in all the dataframes: scr_df, weight_chartevents_df, urine_output_df, chemistry_df, discharge_df
admission_ids = scr_df['hadm_id'].unique()
admission_ids = set(admission_ids).intersection(set(weight_chartevents_df['hadm_id'].unique()))
admission_ids = admission_ids.intersection(set(urine_output_df['hadm_id'].unique()))
admission_ids = admission_ids.intersection(set(chemistry_df['hadm_id'].unique()))
admission_ids = admission_ids.intersection(set(discharge_df['hadm_id'].unique()))

# Get a random admission id from admission_ids
hadm_id = np.random.choice(list(admission_ids))
```

PHASE I: SINGLE PATIENT PREDICTION AND ANALYSIS

Leveraging GPT-4 for AKI Prediction:

- Utilizing GPT-4 with Chain-of-Thought prompting.
- Analyzing patient data to forecast the onset of Acute Kidney Injury (AKI).

Explaining Prediction Rationale:

- Providing detailed explanations behind the AKI risk predictions.
- Enhancing understanding of the factors contributing to AKI.

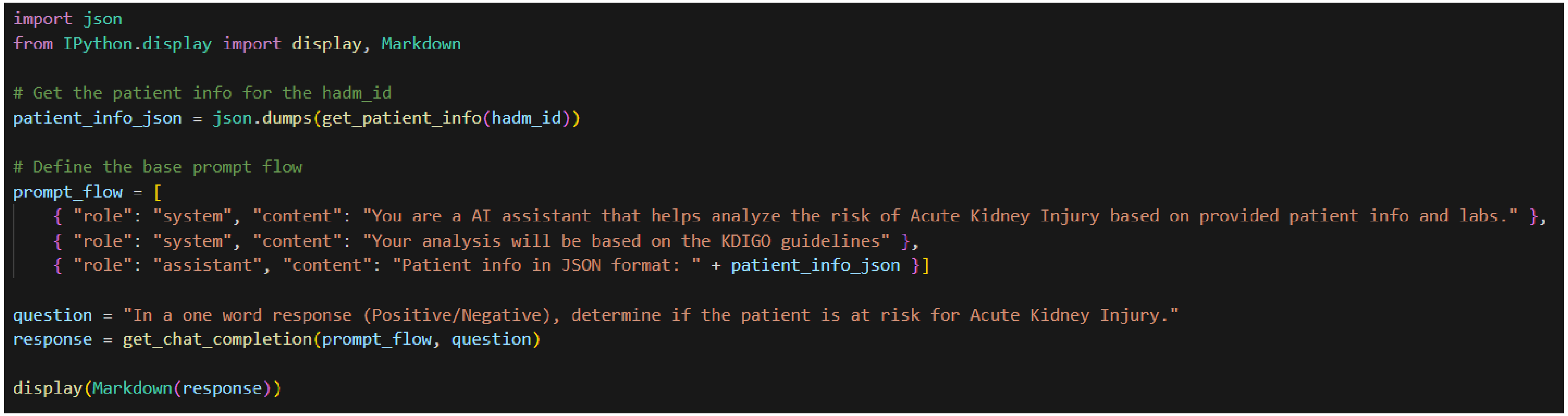
Creating a Timeline of Events:

- Constructing a sequence of clinical markers leading up to AKI.
- Offering insights into the progression and development of the condition.

Extracting Information from Discharge Notes:

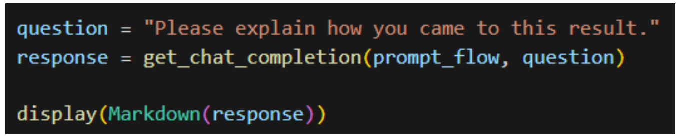
- Analyzing discharge summaries for relevant AKI-related information.
- Integrating clinical narrative with prediction models for a holistic view.

ASK GPT-4 FOR A SINGLE WORD PREDICTION (POSITIVE/NEGATIVE) FOR THE PATIENT'S RISK OF AKI

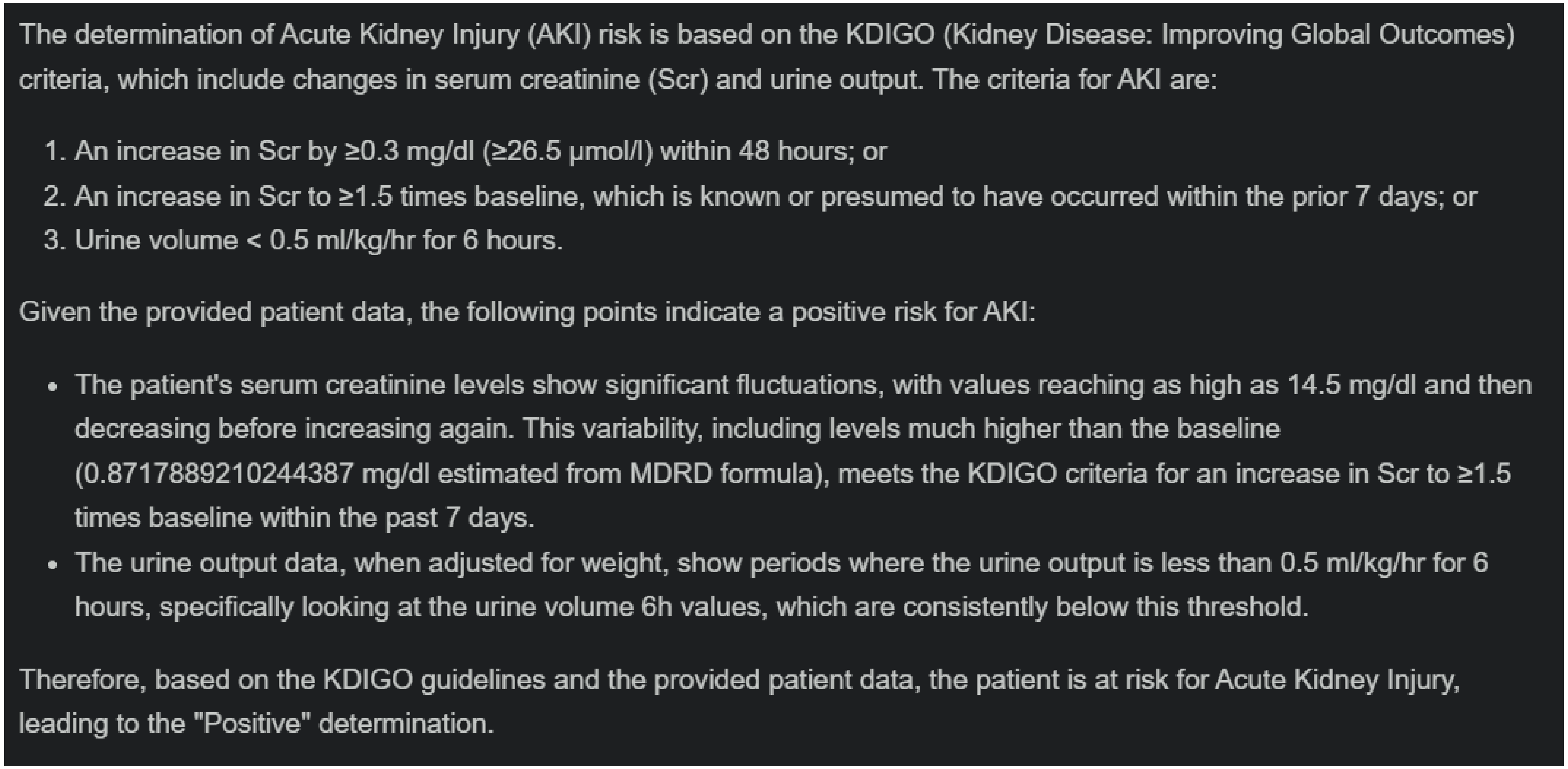


```python
import json
from IPython.display import display, Markdown

# Get the patient info for the hadm_id
patient_info_json = json.dumps(get_patient_info(hadm_id))

# Define the base prompt flow
prompt_flow = [
    { "role": "system", "content": "You are a AI assistant that helps analyze the risk of Acute Kidney Injury based on provided patient info and labs." },
    { "role": "system", "content": "Your analysis will be based on the KDIGO guidelines" },
    { "role": "assistant", "content": "Patient info in JSON format: " + patient_info_json }]

question = "In a one word response (Positive/Negative), determine if the patient is at risk for Acute Kidney Injury."
response = get_chat_completion(prompt_flow, question)

display(Markdown(response))
```

ASK GPT-4 TO EXPLAIN THE REASONING BEHIND THE PREDICTION



```
question = "Please explain how you came to this result."
response = get_chat_completion(prompt_flow, question)

display(Markdown(response))
```



The determination of Acute Kidney Injury (AKI) risk is based on the KDIGO (Kidney Disease: Improving Global Outcomes) criteria, which include changes in serum creatinine (Scr) and urine output. The criteria for AKI are:

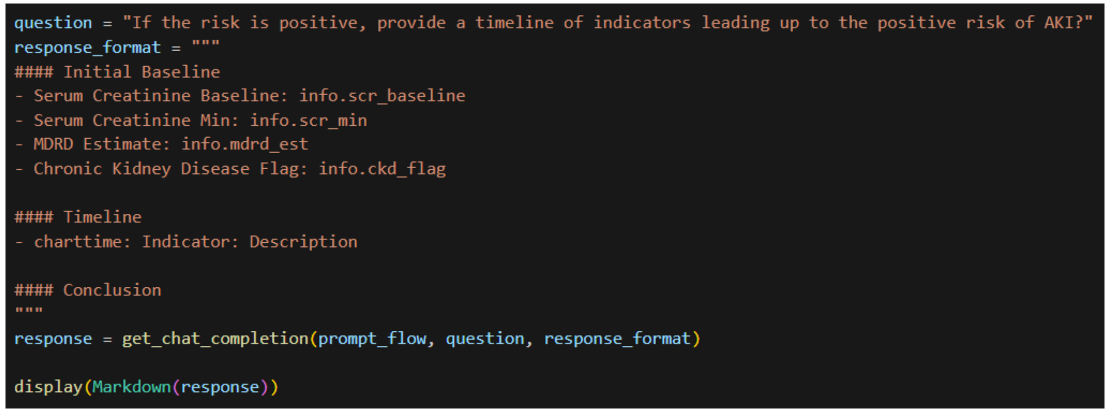
1. An increase in Scr by ≥0.3 mg/dl (≥26.5 μmol/l) within 48 hours; or
2. An increase in Scr to ≥1.5 times baseline, which is known or presumed to have occurred within the prior 7 days; or
3. Urine volume < 0.5 ml/kg/hr for 6 hours.

Given the provided patient data, the following points indicate a positive risk for AKI:

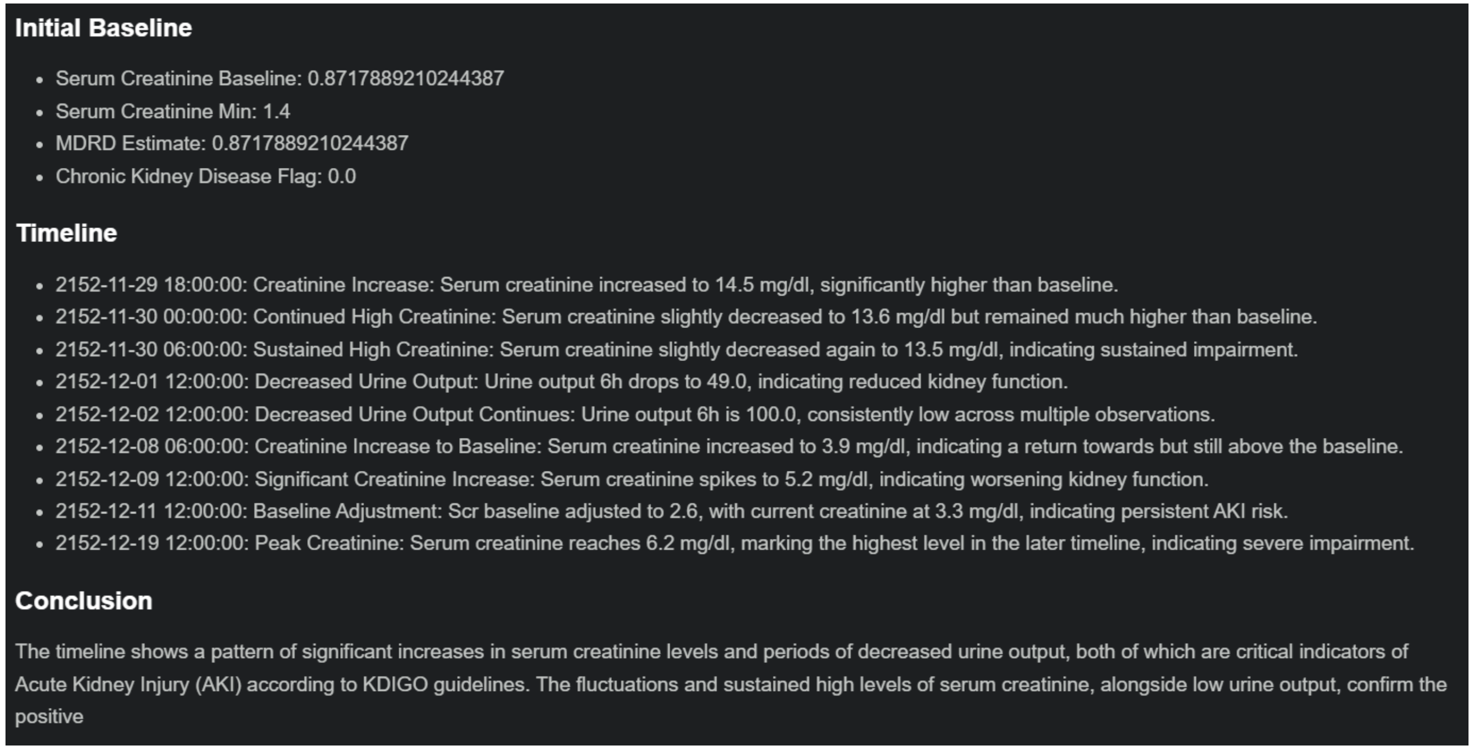
- The patient's serum creatinine levels show significant fluctuations, with values reaching as high as 14.5 mg/dl and then decreasing before increasing again. This variability, including levels much higher than the baseline (0.8717889210244387 mg/dl estimated from MDRD formula), meets the KDIGO criteria for an increase in Scr to ≥1.5 times baseline within the past 7 days.
- The urine output data, when adjusted for weight, show periods where the urine output is less than 0.5 ml/kg/hr for 6 hours, specifically looking at the urine volume 6h values, which are consistently below this threshold.

Therefore, based on the KDIGO guidelines and the provided patient data, the patient is at risk for Acute Kidney Injury, leading to the "Positive" determination.

ASK GPT-4 PROVIDE A TIMELINE OF INDICATORS LEADING UP TO THE POSITIVE RISK OF AKI



```
question = "If the risk is positive, provide a timeline of indicators leading up to the positive risk of AKI?"
response_format = """
#### Initial Baseline
- Serum Creatinine Baseline: info.scr_baseline
- Serum Creatinine Min: info.scr_min
- MDRD Estimate: info.mdrd_est
- Chronic Kidney Disease Flag: info.ckd_flag

#### Timeline
- charttime: Indicator: Description

#### Conclusion
"""
response = get_chat_completion(prompt_flow, question, response_format)

display(Markdown(response))
```



Initial Baseline

- Serum Creatinine Baseline: 0.8717889210244387
- Serum Creatinine Min: 1.4
- MDRD Estimate: 0.8717889210244387
- Chronic Kidney Disease Flag: 0.0

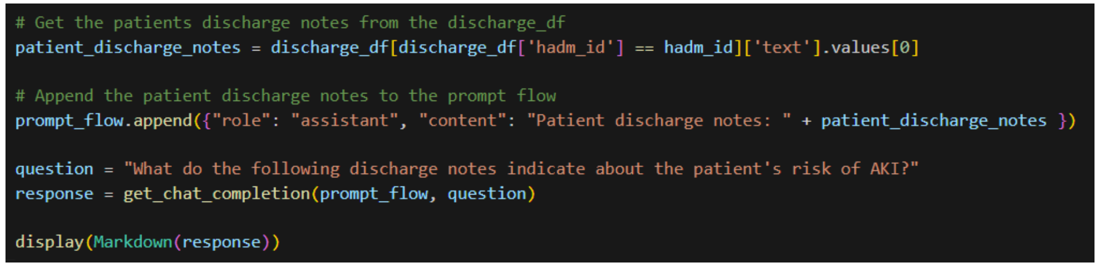
Timeline

- 2152-11-29 18:00:00: Creatinine Increase: Serum creatinine increased to 14.5 mg/dl, significantly higher than baseline.
- 2152-11-30 00:00:00: Continued High Creatinine: Serum creatinine slightly decreased to 13.6 mg/dl but remained much higher than baseline.
- 2152-11-30 06:00:00: Sustained High Creatinine: Serum creatinine slightly decreased again to 13.5 mg/dl, indicating sustained impairment.
- 2152-12-01 12:00:00: Decreased Urine Output: Urine output 6h drops to 49.0, indicating reduced kidney function.
- 2152-12-02 12:00:00: Decreased Urine Output Continues: Urine output 6h is 100.0, consistently low across multiple observations.
- 2152-12-08 06:00:00: Creatinine Increase to Baseline: Serum creatinine increased to 3.9 mg/dl, indicating a return towards but still above the baseline.
- 2152-12-09 12:00:00: Significant Creatinine Increase: Serum creatinine spikes to 5.2 mg/dl, indicating worsening kidney function.
- 2152-12-11 12:00:00: Baseline Adjustment: Scr baseline adjusted to 2.6, with current creatinine at 3.3 mg/dl, indicating persistent AKI risk.
- 2152-12-19 12:00:00: Peak Creatinine: Serum creatinine reaches 6.2 mg/dl, marking the highest level in the later timeline, indicating severe impairment.

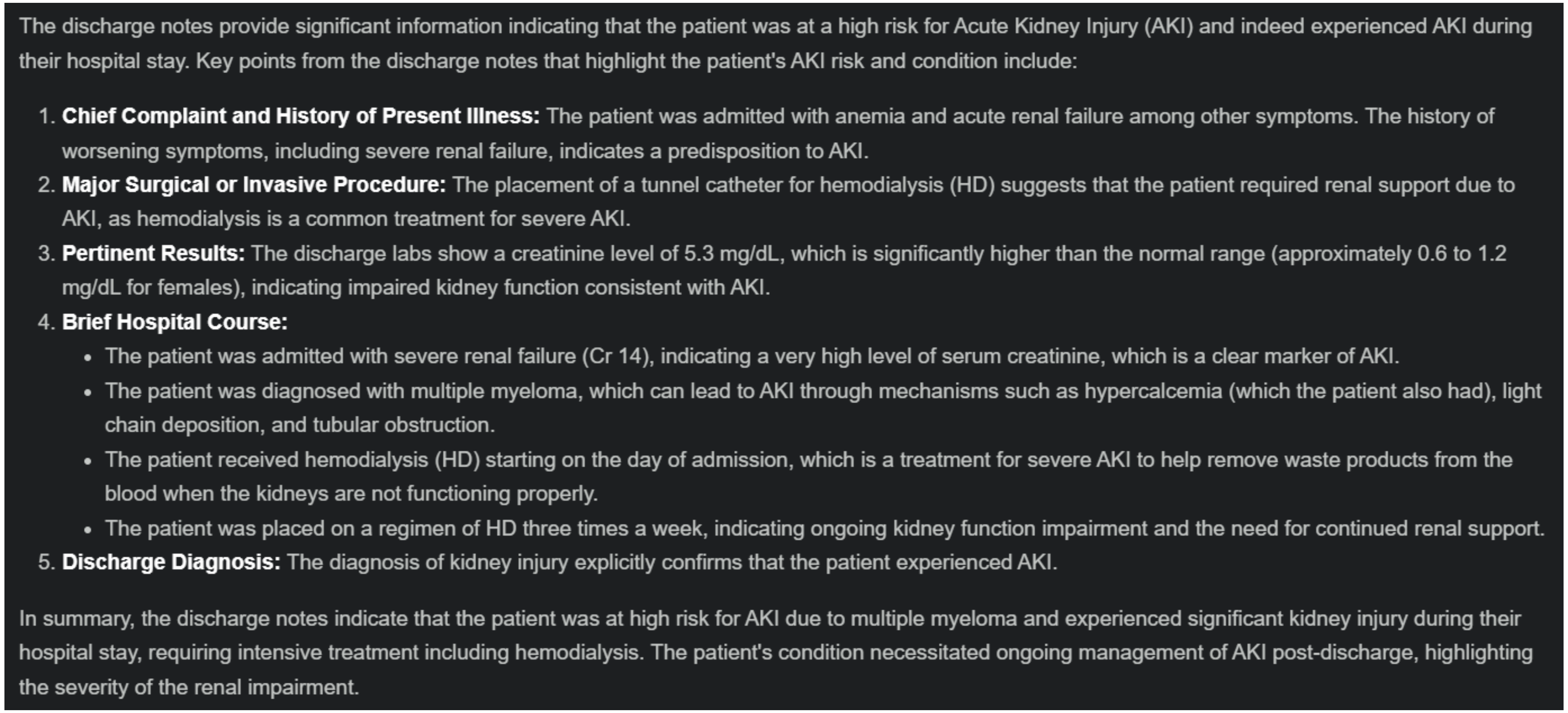
Conclusion

The timeline shows a pattern of significant increases in serum creatinine levels and periods of decreased urine output, both of which are critical indicators of Acute Kidney Injury (AKI) according to KDIGO guidelines. The fluctuations and sustained high levels of serum creatinine, alongside low urine output, confirm the positive

ASK GPT-4 TO DETERMINE WHAT THE DISCHARGE NOTES INDICATE ABOUT THE PATIENT'S RISK OF AKI



```python
# Get the patients discharge notes from the discharge_df
patient_discharge_notes = discharge_df[discharge_df['hadm_id'] == hadm_id]['text'].values[0]

# Append the patient discharge notes to the prompt flow
prompt_flow.append({"role": "assistant", "content": "Patient discharge notes: " + patient_discharge_notes })

question = "What do the following discharge notes indicate about the patient's risk of AKI?"
response = get_chat_completion(prompt_flow, question)

display(Markdown(response))
```



The discharge notes provide significant information indicating that the patient was at a high risk for Acute Kidney Injury (AKI) and indeed experienced AKI during their hospital stay. Key points from the discharge notes that highlight the patient's AKI risk and condition include:

1. **Chief Complaint and History of Present Illness:** The patient was admitted with anemia and acute renal failure among other symptoms. The history of worsening symptoms, including severe renal failure, indicates a predisposition to AKI.
2. **Major Surgical or Invasive Procedure:** The placement of a tunnel catheter for hemodialysis (HD) suggests that the patient required renal support due to AKI, as hemodialysis is a common treatment for severe AKI.
3. **Pertinent Results:** The discharge labs show a creatinine level of 5.3 mg/dL, which is significantly higher than the normal range (approximately 0.6 to 1.2 mg/dL for females), indicating impaired kidney function consistent with AKI.
4. **Brief Hospital Course:**
   - The patient was admitted with severe renal failure (Cr 14), indicating a very high level of serum creatinine, which is a clear marker of AKI.
   - The patient was diagnosed with multiple myeloma, which can lead to AKI through mechanisms such as hypercalcemia (which the patient also had), light chain deposition, and tubular obstruction.
   - The patient received hemodialysis (HD) starting on the day of admission, which is a treatment for severe AKI to help remove waste products from the blood when the kidneys are not functioning properly.
   - The patient was placed on a regimen of HD three times a week, indicating ongoing kidney function impairment and the need for continued renal support.
5. **Discharge Diagnosis:** The diagnosis of kidney injury explicitly confirms that the patient experienced AKI.

In summary, the discharge notes indicate that the patient was at high risk for AKI due to multiple myeloma and experienced significant kidney injury during their hospital stay, requiring intensive treatment including hemodialysis. The patient's condition necessitated ongoing management of AKI post-discharge, highlighting the severity of the renal impairment.

PHASE 1I: ASSESSING GPT-4'S AKI PREDICTION ACCURACY AGAINST KDIGO CRITERIA AND CLINICAL DIAGNOSES

Evaluating Prediction Accuracy:

- Assessing the performance of GPT-4 in predicting AKI risk.
- Comparing predictions against actual AKI risk as defined by KDIGO criteria.

Utilizing Actual Diagnoses for Validation:

- Leveraging existing AKI diagnoses as a benchmark for model accuracy.
- Ensuring validation is rooted in clinically verified outcomes.

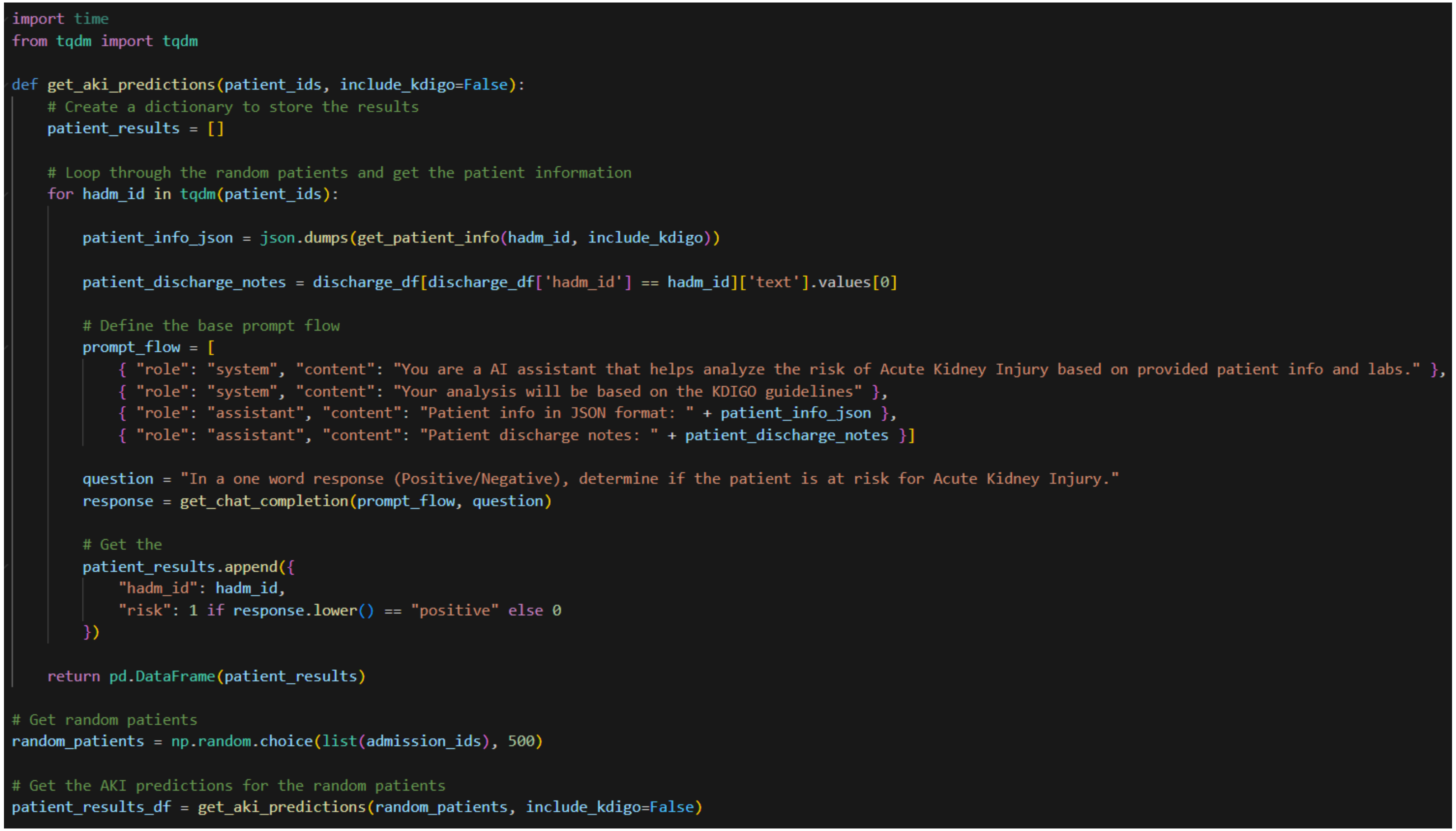
Incorporating KDIGO Criteria:

- Utilizing established KDIGO guidelines as a standard for AKI identification.
- Aligning model predictions with recognized clinical benchmarks for AKI.

Comprehensive Model Evaluation:

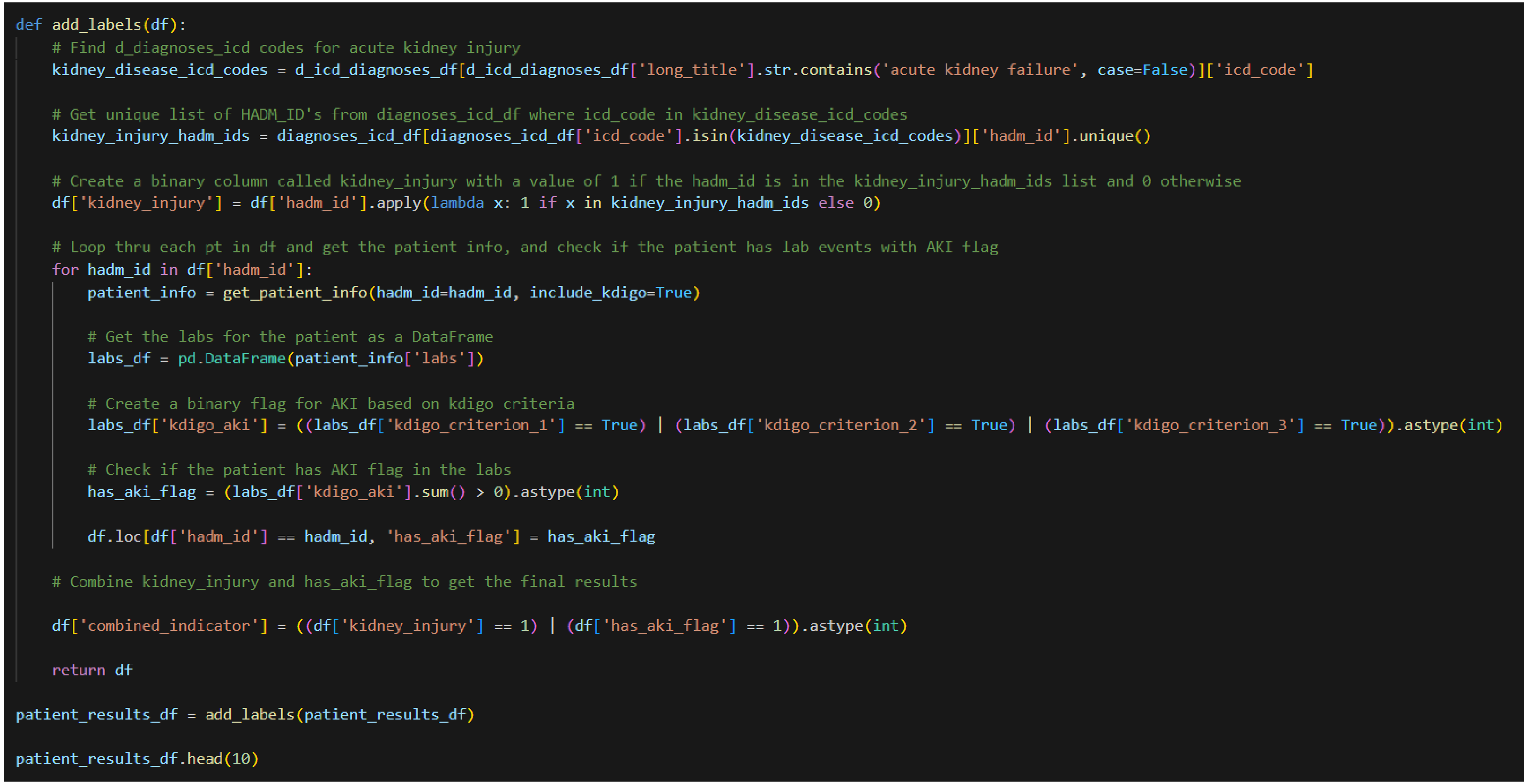
- Employing a variety of metrics to assess model performance comprehensively.
- Looking beyond accuracy to include precision, recall, and AUC-ROC scores.

PREDICT RISK OF AKI FOR A RANDOM SAMPLE OF PATIENT ADMISSIONS



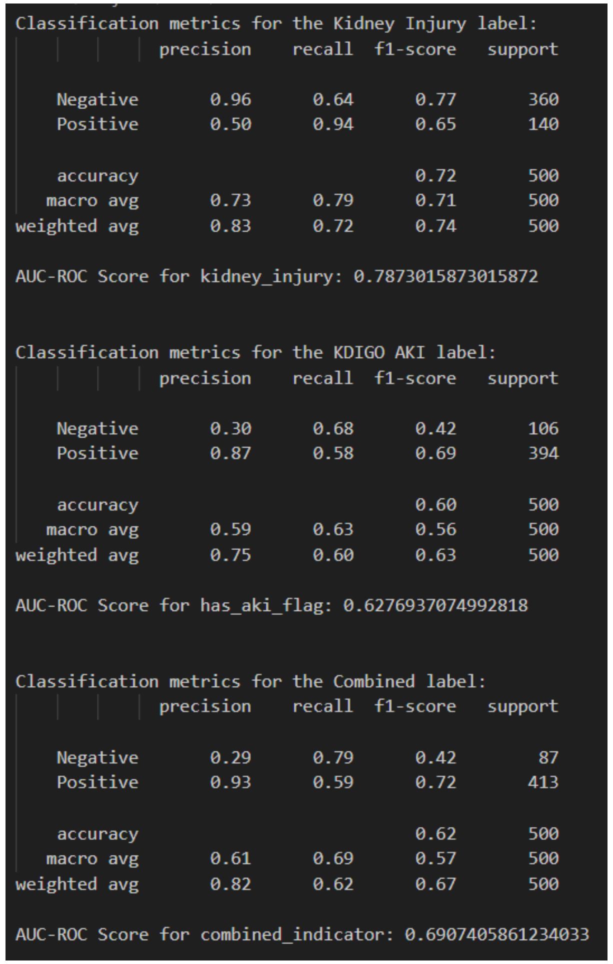
```python
import time
from tqdm import tqdm

def get_aki_predictions(patient_ids, include_kdigo=False):
    # Create a dictionary to store the results
    patient_results = []

    # Loop through the random patients and get the patient information
    for hadm_id in tqdm(patient_ids):

        patient_info_json = json.dumps(get_patient_info(hadm_id, include_kdigo))

        patient_discharge_notes = discharge_df[discharge_df['hadm_id'] == hadm_id]['text'].values[0]

        # Define the base prompt flow
        prompt_flow = [
            { "role": "system", "content": "You are a AI assistant that helps analyze the risk of Acute Kidney Injury based on provided patient info and labs." },
            { "role": "system", "content": "Your analysis will be based on the KDIGO guidelines" },
            { "role": "assistant", "content": "Patient info in JSON format: " + patient_info_json },
            { "role": "assistant", "content": "Patient discharge notes: " + patient_discharge_notes }]

        question = "In a one word response (Positive/Negative), determine if the patient is at risk for Acute Kidney Injury."
        response = get_chat_completion(prompt_flow, question)

        # Get the
        patient_results.append({
            "hadm_id": hadm_id,
            "risk": 1 if response.lower() == "positive" else 0
        })

    return pd.DataFrame(patient_results)

# Get random patients
random_patients = np.random.choice(list(admission_ids), 500)

# Get the AKI predictions for the random patients
patient_results_df = get_aki_predictions(random_patients, include_kdigo=False)
```

ADD LABELS TO THE PATIENT RESULTS BASED ON DIAGNOSES AND KDIGO CALCULATION

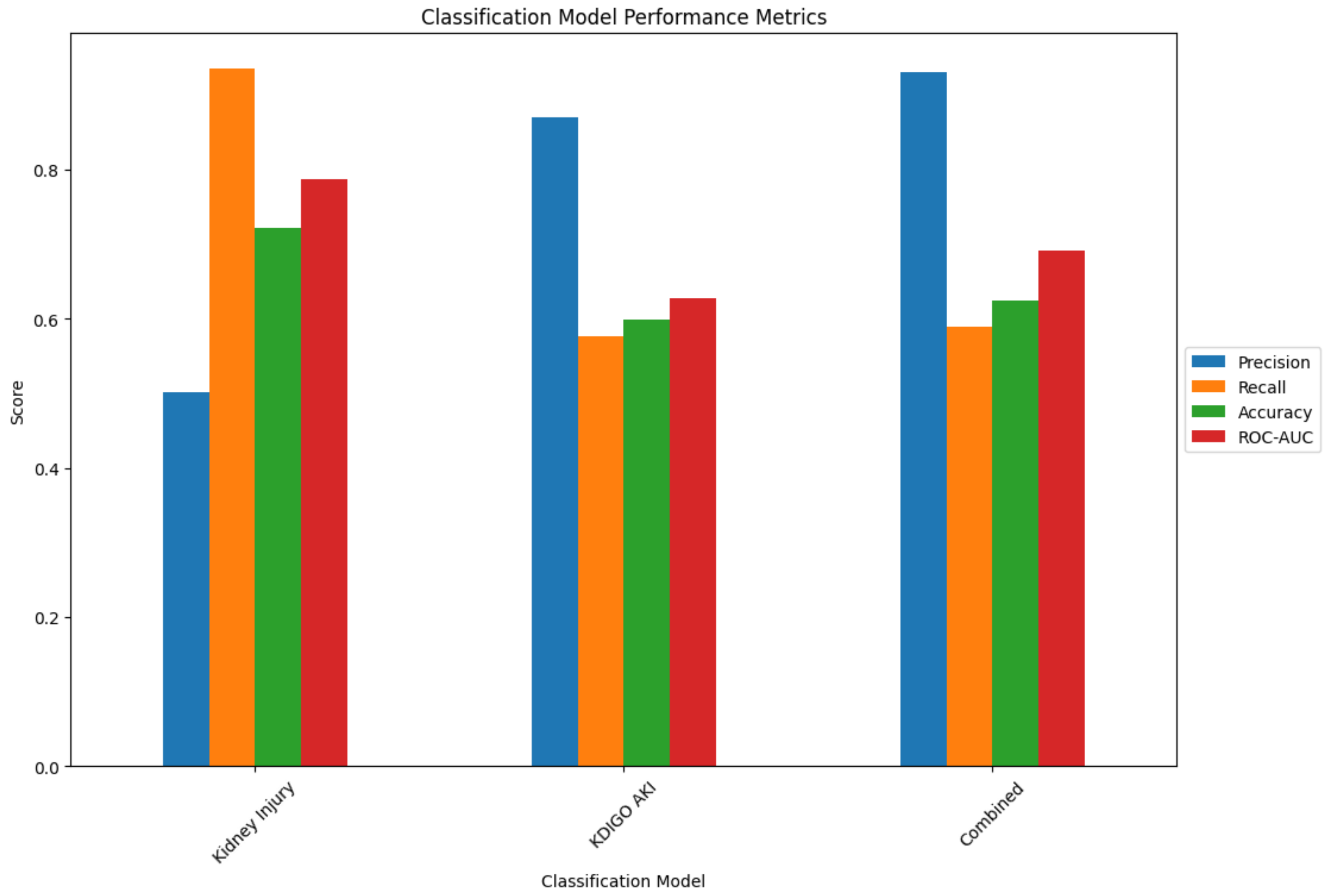


```python
def add_labels(df):
    # Find d_diagnoses_icd codes for acute kidney injury
    kidney_disease_icd_codes = d_icd_diagnoses_df[d_icd_diagnoses_df['long_title'].str.contains('acute kidney failure', case=False)]['icd_code']

    # Get unique list of HADM_ID's from diagnoses_icd_df where icd_code in kidney_disease_icd_codes
    kidney_injury_hadm_ids = diagnoses_icd_df[diagnoses_icd_df['icd_code'].isin(kidney_disease_icd_codes)]['hadm_id'].unique()

    # Create a binary column called kidney_injury with a value of 1 if the hadm_id is in the kidney_injury_hadm_ids list and 0 otherwise
    df['kidney_injury'] = df['hadm_id'].apply(lambda x: 1 if x in kidney_injury_hadm_ids else 0)

    # Loop thru each pt in df and get the patient info, and check if the patient has lab events with AKI flag
    for hadm_id in df['hadm_id']:
        patient_info = get_patient_info(hadm_id=hadm_id, include_kdigo=True)

        # Get the labs for the patient as a DataFrame
        labs_df = pd.DataFrame(patient_info['labs'])

        # Create a binary flag for AKI based on kdigo criteria
        labs_df['kdigo_aki'] = ((labs_df['kdigo_criterion_1'] == True) | (labs_df['kdigo_criterion_2'] == True) | (labs_df['kdigo_criterion_3'] == True)).astype(int)

        # Check if the patient has AKI flag in the labs
        has_aki_flag = (labs_df['kdigo_aki'].sum() > 0).astype(int)

        df.loc[df['hadm_id'] == hadm_id, 'has_aki_flag'] = has_aki_flag

    # Combine kidney_injury and has_aki_flag to get the final results

    df['combined_indicator'] = ((df['kidney_injury'] == 1) | (df['has_aki_flag'] == 1)).astype(int)

    return df

patient_results_df = add_labels(patient_results_df)

patient_results_df.head(10)
```

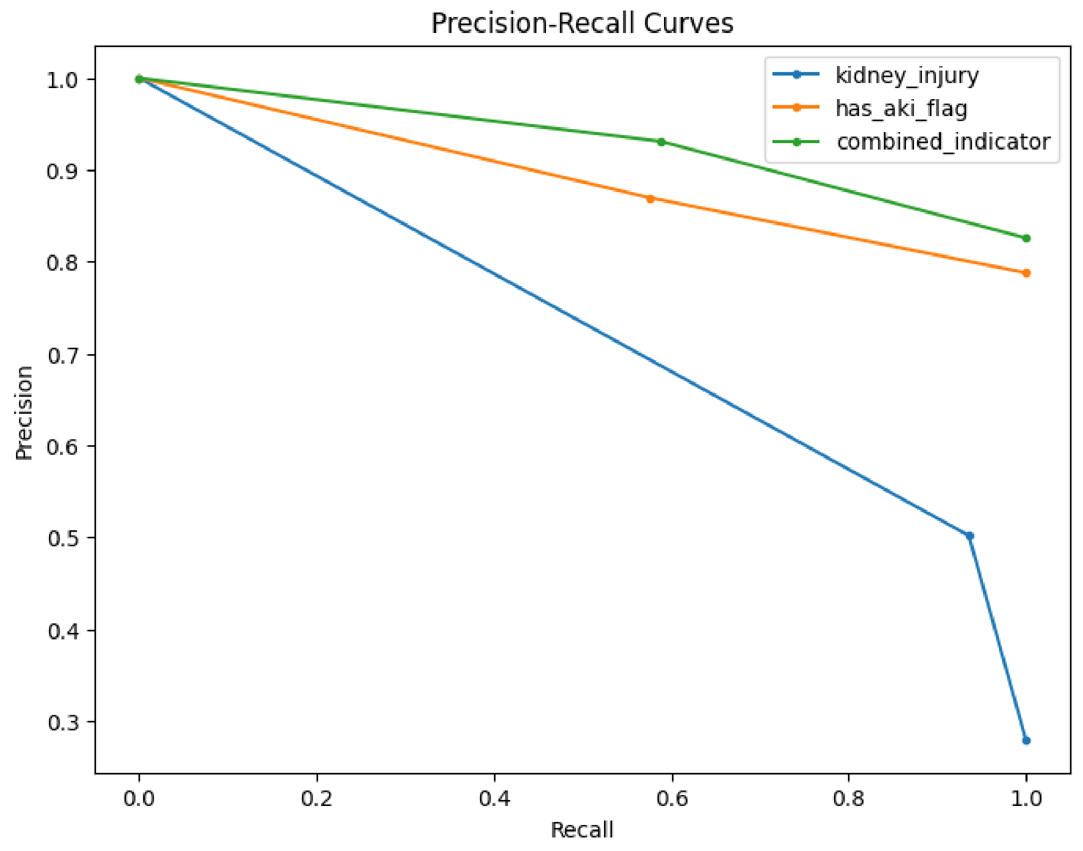
CALCULATE THE CLASSIFICATION METRICS AND AUC-ROC SCORE FOR EACH LABEL AND PLOT THEM

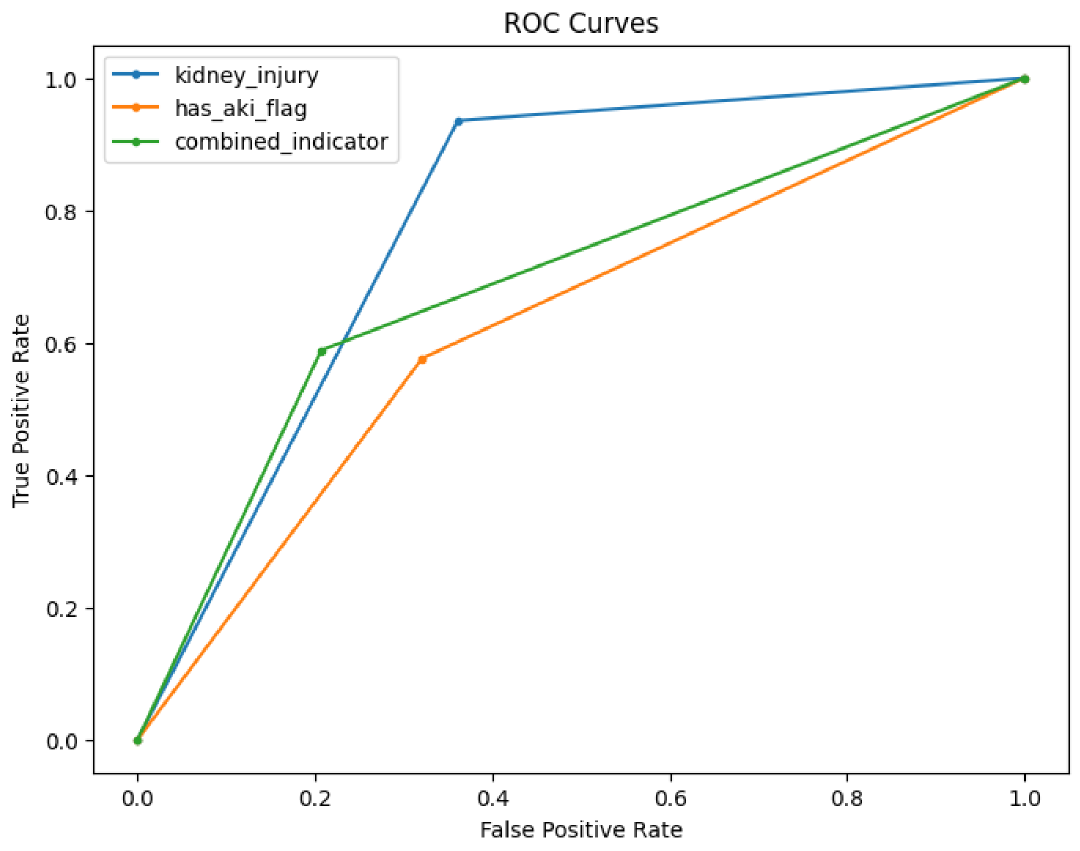


```
Classification metrics for the Kidney Injury label:
              precision    recall  f1-score   support

    Negative       0.96      0.64      0.77       360
    Positive       0.50      0.94      0.65       140

    accuracy                           0.72       500
   macro avg       0.73      0.79      0.71       500
weighted avg       0.83      0.72      0.74       500

AUC-ROC Score for kidney_injury: 0.7873015873015872


Classification metrics for the KDIGO AKI label:
              precision    recall  f1-score   support

    Negative       0.30      0.68      0.42       106
    Positive       0.87      0.58      0.69       394

    accuracy                           0.60       500
   macro avg       0.59      0.63      0.56       500
weighted avg       0.75      0.60      0.63       500

AUC-ROC Score for has_aki_flag: 0.6276937074992818


Classification metrics for the Combined label:
              precision    recall  f1-score   support

    Negative       0.29      0.79      0.42        87
    Positive       0.93      0.59      0.72       413

    accuracy                           0.62       500
   macro avg       0.61      0.69      0.57       500
weighted avg       0.82      0.62      0.67       500

AUC-ROC Score for combined_indicator: 0.6907405861234033
```

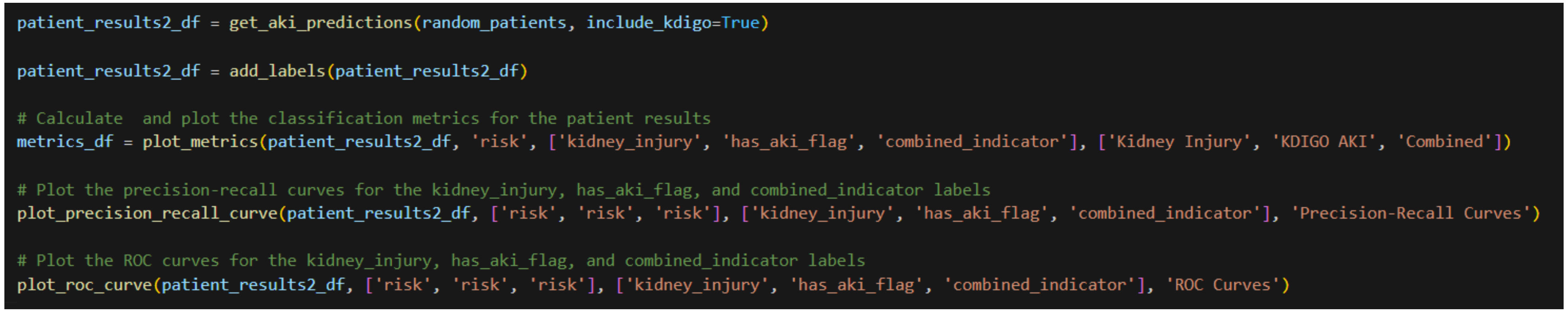


CALCULATE PRECISION-RECALL AND ROC CURVES FOR EACH LABEL AND PLOT THEM



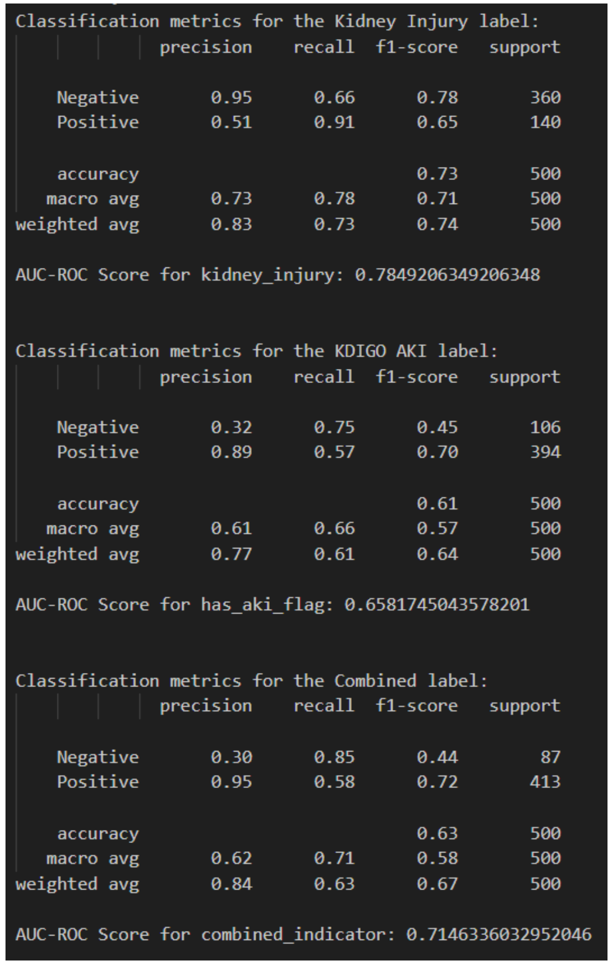


REGENERATE AND EVALUATE THE AKI PREDICTIONS ON THE SAME SAMPLE BUT INCLUDE KEY KDIGO CRITERION CALCULATIONS IN THE PATIENT INFO

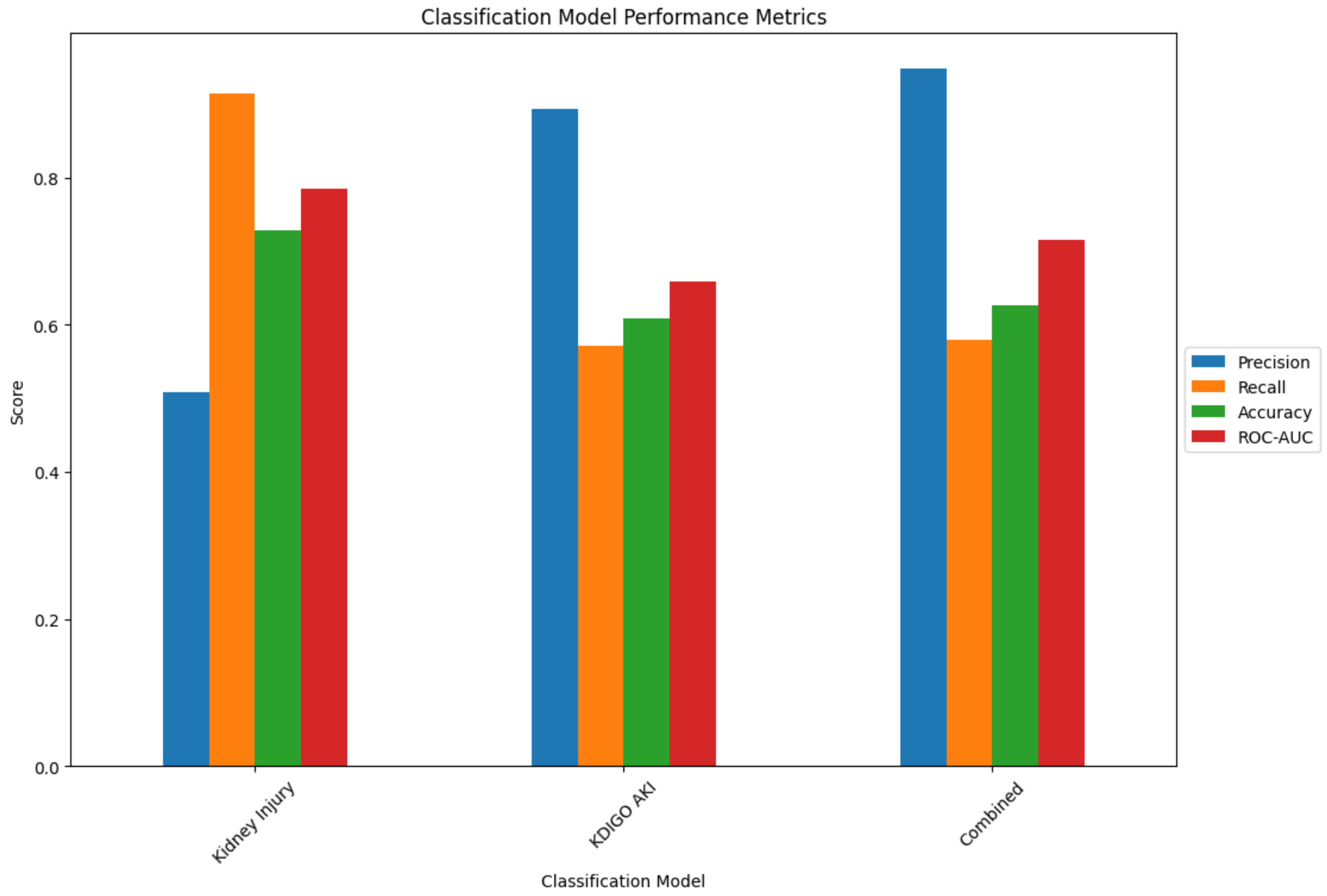


```python
patient_results2_df = get_aki_predictions(random_patients, include_kdigo=True)

patient_results2_df = add_labels(patient_results2_df)

# Calculate  and plot the classification metrics for the patient results
metrics_df = plot_metrics(patient_results2_df, 'risk', ['kidney_injury', 'has_aki_flag', 'combined_indicator'], ['Kidney Injury', 'KDIGO AKI', 'Combined'])

# Plot the precision-recall curves for the kidney_injury, has_aki_flag, and combined_indicator labels
plot_precision_recall_curve(patient_results2_df, ['risk', 'risk', 'risk'], ['kidney_injury', 'has_aki_flag', 'combined_indicator'], 'Precision-Recall Curves')

# Plot the ROC curves for the kidney_injury, has_aki_flag, and combined_indicator labels
plot_roc_curve(patient_results2_df, ['risk', 'risk', 'risk'], ['kidney_injury', 'has_aki_flag', 'combined_indicator'], 'ROC Curves')
```

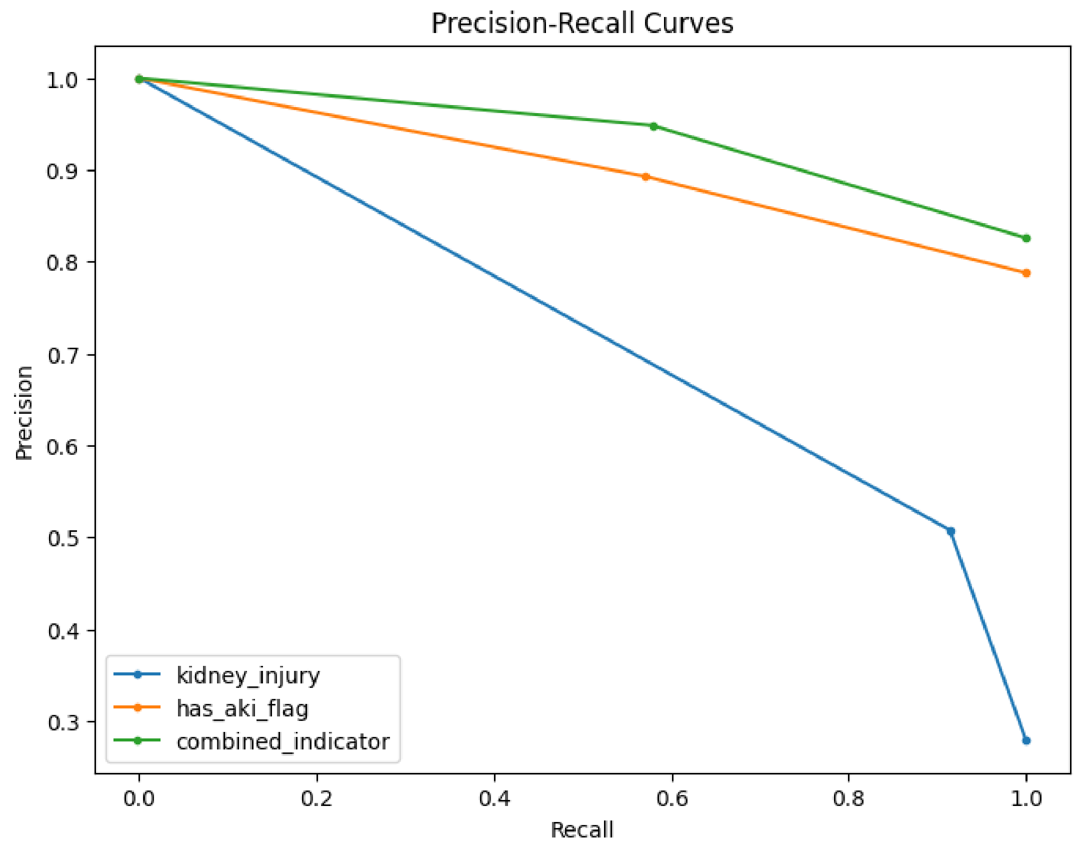
CALCULATE THE CLASSIFICATION METRICS AND AUC-ROC SCORE FOR EACH LABEL AND PLOT THEM

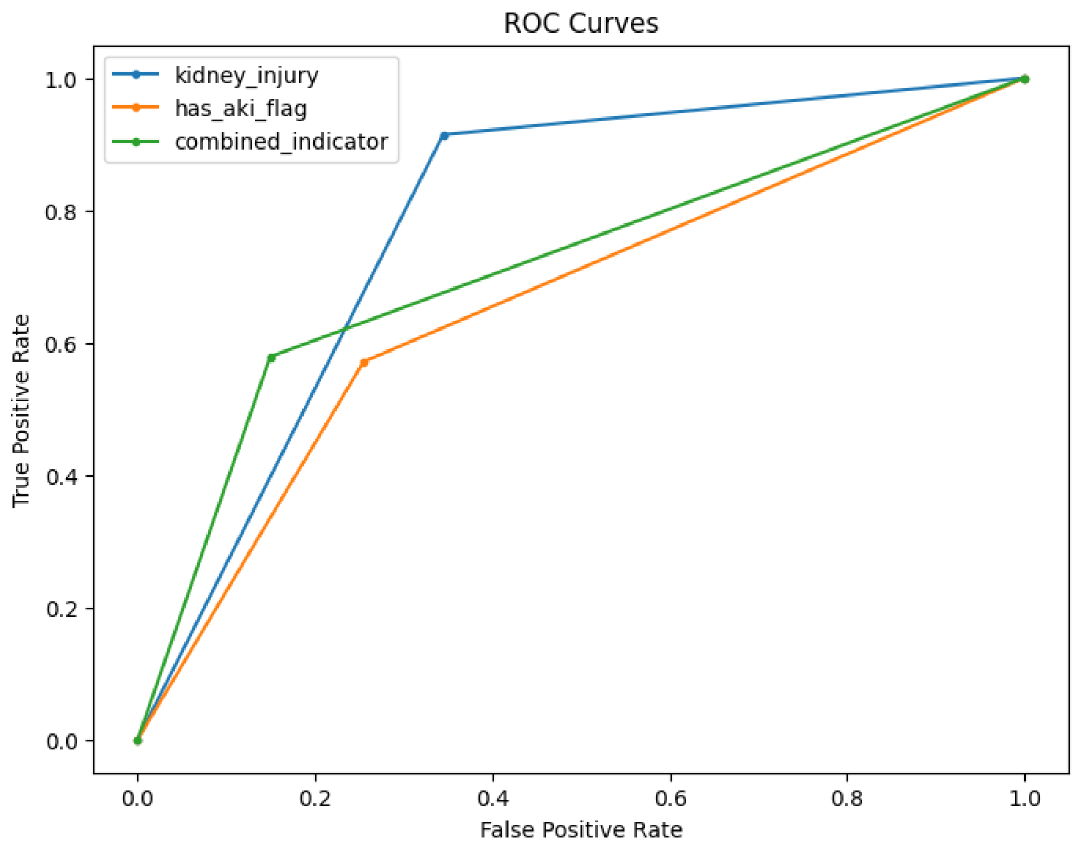


```
Classification metrics for the Kidney Injury label:
              precision    recall  f1-score   support

    Negative       0.95      0.66      0.78       360
    Positive       0.51      0.91      0.65       140

    accuracy                           0.73       500
   macro avg       0.73      0.78      0.71       500
weighted avg       0.83      0.73      0.74       500

AUC-ROC Score for kidney_injury: 0.7849206349206348


Classification metrics for the KDIGO AKI label:
              precision    recall  f1-score   support

    Negative       0.32      0.75      0.45       106
    Positive       0.89      0.57      0.70       394

    accuracy                           0.61       500
   macro avg       0.61      0.66      0.57       500
weighted avg       0.77      0.61      0.64       500

AUC-ROC Score for has_aki_flag: 0.6581745043578201


Classification metrics for the Combined label:
              precision    recall  f1-score   support

    Negative       0.30      0.85      0.44        87
    Positive       0.95      0.58      0.72       413

    accuracy                           0.63       500
   macro avg       0.62      0.71      0.58       500
weighted avg       0.84      0.63      0.67       500

AUC-ROC Score for combined_indicator: 0.7146336032952046
```



CALCULATE PRECISION-RECALL AND ROC CURVES FOR EACH LABEL AND PLOT THEM





EVALUATION OF MODEL PERFORMANCE

Kidney Injury Prediction:

- **Precision and Recall:** For predicting kidney injury based on discharge diagnoses, the model demonstrated moderate precision and high recall, indicating a tendency to identify most true cases of kidney injury but also to falsely label some patients as at risk.
- **AUC-ROC Score:** The Area Under the Receiver Operating Characteristic (AUC-ROC) score for kidney injury predictions was approximately 0.78, suggesting a good but not excellent ability to distinguish between patients with and without kidney injury.

KDIGO AKI Prediction:

- **Precision and Recall:** When evaluating the model based on KDIGO AKI criteria, the precision was lower, but recall remained moderately high. This indicates the model's capability to capture a significant portion of true AKI cases as defined by KDIGO, albeit with a higher false-positive rate.
- **AUC-ROC Score:** The AUC-ROC score for KDIGO AKI was around 0.63, reflecting a moderate discriminative power for this specific prediction task.

Combined Indicator Prediction:

- **Precision and Recall:** The combined indicator, which considers both discharge diagnoses of kidney injury and KDIGO criteria for AKI, saw a slightly higher precision and recall than the individual indicators. This approach leverages both clinical and laboratory data to assess AKI risk.
- **AUC-ROC Score:** The AUC-ROC score for the combined indicator reached approximately 0.69, indicating an improvement in the model's ability to classify patients accurately compared to using KDIGO criteria alone.

Clinical Implications:

- The model's high recall rates across different indicators suggest it could serve as an effective screening tool to identify patients at risk for AKI, ensuring that fewer cases go unnoticed. However, the moderate precision and AUC-ROC scores highlight a need for caution, as the model may generate false positives, potentially leading to unnecessary interventions or anxiety.

Enhancing Model Performance with Structured Clinical Criteria:

- Incorporating KDIGO criteria directly into the model's input significantly impacted its performance. Including key KDIGO criterion calculations resulted in improved precision and recall, especially for combined indicator predictions. This suggests that providing the model with structured clinical criteria and guidelines can enhance its predictive accuracy and clinical relevance.

SUMMARY

This presentation outlined and evaluated the innovative use of GPT-4 in predicting the risk of Acute Kidney Injury (AKI) among patients. The exercise involved several key steps, each contributing to the overall aim of enhancing early detection and understanding of AKI. Here's a summary of the critical components and outcomes:

Prediction Using GPT-4:

- The process leveraged GPT-4, combined with Chain-of-Thought prompting and patient data, to anticipate AKI onset. This approach aimed to harness the power of advanced AI for predictive healthcare analytics.

Evaluation Against Clinical Standards:

- The accuracy of GPT-4 predictions was rigorously evaluated against actual AKI risks based on the Kidney Disease: Improving Global Outcomes (KDIGO) criteria and existing AKI diagnoses. This comparison ensured that the AI model's predictions were grounded in established medical standards and real-world outcomes.

Comprehensive Model Assessment:

- The evaluation extended beyond simple accuracy metrics to include a variety of measures, such as precision, recall, and the Area Under the Receiver Operating Characteristic Curve (AUC-ROC). These metrics provided a nuanced understanding of the model's performance.

Outcomes and Insights:

- The exercise revealed GPT-4's potential in accurately predicting AKI risk, highlighting the model's ability to align closely with clinical diagnoses and criteria. It underscored the importance of integrating AI with established healthcare protocols to enhance patient care.

IDEAS FOR IMPROVEMENTS

Data Quality and Diversity: Prioritize the enhancement of input data quality and diversity for training the LLM. This includes incorporating a wider range of demographic information, clinical parameters, and longitudinal patient data to improve the model's understanding and prediction of AKI risk.

Advanced Natural Language Processing (NLP) Techniques: Implement more sophisticated NLP techniques to better interpret and utilize the unstructured data from patient records. Techniques such as entity recognition, sentiment analysis, and contextual embeddings could provide deeper insights into patient conditions, potentially improving prediction accuracy.

Customized Model Training: Explore the potential of custom training the LLM specifically for AKI prediction by fine-tuning it with a dataset comprised of AKI cases. This could help the model to better learn the nuances and complexities associated with AKI risk factors.

Interpretability and Explainability Enhancements: Work on increasing the interpretability of the LLM's predictions. Developing methods to provide clear explanations for the AI's decision-making process can build trust among clinicians and help them understand the rationale behind the AI's AKI risk assessments.

Integration of Dynamic Data: Investigate ways to incorporate dynamic, real-time patient data into the LLM's predictive framework. The ability to analyze changes in patient conditions over time could significantly enhance the model's predictive capabilities and timeliness.

Robust Validation Mechanisms: Enhance the validation process by employing more rigorous and diverse methodologies. This could involve cross-validation techniques, external validation on separate datasets, or real-world testing in clinical settings to ensure the model's reliability and generalizability.

Collaboration with Clinical Experts: Strengthen collaboration with nephrologists and other healthcare professionals to continuously refine the LLM's predictive model. Their insights can help identify clinical nuances and emerging AKI risk factors that might improve the model's performance.

Ethical and Bias Considerations: Actively address potential ethical concerns and biases within the AI model. This involves regular audits for bias, ensuring the model's equitable performance across different patient populations, and aligning the model's use with ethical guidelines in healthcare.