In [3]:
```
!pip install pdf2image
!pip install pytesseract
!pip install pillow
!apt-get install -y poppler-utils
!apt-get install -y tesseract-ocr
!pip install pytesseract
from pdf2image import convert_from_path
from pytesseract import image_to_string
from PIL import Image
import cv2
import pytesseract
from pdf2image import convert_from_path
from pytesseract import image_to_string
from PIL import Image
import nltk
nltk.download('punkt')
nltk.download('words')
nltk.download('stopwords')
nltk.download('maxent_ne_chunker')
nltk.download('averaged_perceptron_tagger')
import re
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
import pandas as pd
```

```
Requirement already satisfied: pdf2image in ./opt/anaconda3/lib/python
3.9/site-packages (1.16.3)
Requirement already satisfied: pillow in ./opt/anaconda3/lib/python3.9/
site-packages (from pdf2image) (9.2.0)
Requirement already satisfied: pytesseract in ./opt/anaconda3/lib/pytho
n3.9/site-packages (0.3.10)
Requirement already satisfied: packaging>=21.3 in ./opt/anaconda3/lib/p
ython3.9/site-packages (from pytesseract) (21.3)
Requirement already satisfied: Pillow>=8.0.0 in ./opt/anaconda3/lib/pyt
hon3.9/site-packages (from pytesseract) (9.2.0)
Requirement already satisfied: pyparsing!=3.0.5,>=2.0.2 in ./opt/anacon
da3/lib/python3.9/site-packages (from packaging>=21.3->pytesseract) (3.
0.9)
Requirement already satisfied: pillow in ./opt/anaconda3/lib/python3.9/
site-packages (9.2.0)
zsh:1: command not found: apt-get
zsh:1: command not found: apt-get
Requirement already satisfied: pytesseract in ./opt/anaconda3/lib/pytho
n3.9/site-packages (0.3.10)
Requirement already satisfied: Pillow>=8.0.0 in ./opt/anaconda3/lib/pyt
hon3.9/site-packages (from pytesseract) (9.2.0)
Requirement already satisfied: packaging>=21.3 in ./opt/anaconda3/lib/p
ython3.9/site-packages (from pytesseract) (21.3)
Requirement already satisfied: pyparsing!=3.0.5,>=2.0.2 in ./opt/anacon
da3/lib/python3.9/site-packages (from packaging>=21.3->pytesseract) (3.
0.9)
```

```
[nltk_data] Downloading package punkt to
[nltk_data]     /Users/prathyushavajinepally/nltk_data...
[nltk_data]   Package punkt is already up-to-date!
[nltk_data] Downloading package words to
[nltk_data]     /Users/prathyushavajinepally/nltk_data...
[nltk_data]   Package words is already up-to-date!
[nltk_data] Downloading package stopwords to
[nltk_data]     /Users/prathyushavajinepally/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
[nltk_data] Downloading package maxent_ne_chunker to
[nltk_data]     /Users/prathyushavajinepally/nltk_data...
[nltk_data]   Package maxent_ne_chunker is already up-to-date!
[nltk_data] Downloading package averaged_perceptron_tagger to
[nltk_data]     /Users/prathyushavajinepally/nltk_data...
[nltk_data]   Package averaged_perceptron_tagger is already up-to-
[nltk_data]       date!
```

In [4]:

```python
from pdf2image import convert_from_path
from PIL import Image

def split_pdf_pages(input_file_path):
    # Convert the PDF into a list of PIL Image objects
    images = convert_from_path(input_file_path)

    for i, image in enumerate(images):
        # Save each image as a separate file
        output_file_path = f'page_{i+1}.png'
        image.save(output_file_path, 'PNG')
        print(f'Saved page {i+1} as {output_file_path}')



# Specify the path to your PDF file
pdf_file_path = '/Users/prathyushavajinepally/Desktop/Ptreports.pdf'

# Split the PDF into separate pages
split_pdf_pages(pdf_file_path)
```

```
Saved page 1 as page_1.png
Saved page 2 as page_2.png
Saved page 3 as page_3.png
Saved page 4 as page_4.png
Saved page 5 as page_5.png
Saved page 6 as page_6.png
Saved page 7 as page_7.png
Saved page 8 as page_8.png
Saved page 9 as page_9.png
Saved page 10 as page_10.png
Saved page 11 as page_11.png
Saved page 12 as page_12.png
Saved page 13 as page_13.png
Saved page 14 as page_14.png
Saved page 15 as page_15.png
Saved page 16 as page_16.png
Saved page 17 as page_17.png
```

In [5]:
```
import pytesseract
from PIL import Image

# Set the path to the Tesseract executable (replace with your own path)
pytesseract.tesseract_cmd = r'/opt/homebrew/bin/tesseract'
# Open the image file
image3 = Image.open('/Users/prathyushavajinepally/Downloads/page_3.png')
image4 = Image.open('/Users/prathyushavajinepally/Downloads/page_4.png')

# Perform OCR to extract text from the image
text3 = pytesseract.image_to_string(image3)
text4 = pytesseract.image_to_string(image4)


# Print the extracted text
#print(text3)
#print(text4)
data=text3+text4
print(data)
```

```
Date of Service: 12/10/2022 12:17:00 PM
Addressee: [redacted]

CLICK HERE TO VIEW THE IMAGES (4)

This report is for: [redacted]
Referred By:
Dr G. [redacted]

XR - RIGHT KNEE 12/10/2022 Reference: 3346017

X-RAY RIGHT KNEE

CLINICAL HISTORY: Pain and restricted movement

REPORT:

Tibiofemoral joint space narrowing noted, with moderate marginal
osteophytes seen. There is also spurring at the intercondylar spine.
```

In [6]:
```
import re
import pandas as pd
```

In [7]:
```
pattern = r'(.+?)\s+((?:[\d.]+\s+)*[\d.]+)\s+(\S+)\s+(.+)$'
matches = re.findall(pattern, data, re.MULTILINE)

# Extract the data with units, test names, and values along with symbols
result = []
for match in matches:
    test_name = match[0].strip()
    values = match[1].strip().split()
    unit = match[2].strip()
    reference = match[3].strip()

    if len(values) < 3:
        values.extend([None] * (3 - len(values)))  # Add None values if

    # Remove 'H' from the values if it exists
    values = [value.replace('H', '') if value else None for value in val

    # Append each set of data as a dictionary to the result list
    row_data = {
        'Test': test_name,
        '16/01/21': values[0],
        '18/09/21': values[1],
        '10/10/22': values[2],
        'Units': unit,
        'Reference': reference
    }
    result.append(row_data)

df = pd.DataFrame(result)

# Modify the data for index 25
df.at[25, 'Test'] = "Cholesterol"
df.at[25, '16/01/21'] = "5.3"
df.at[25, '18/09/21'] = "5.7"
df.at[25, '10/10/22'] = "5.6"
df.at[25, 'Units'] = "mmol/L"
df.at[25, 'Reference'] = "<5.5"

# Modify the data for index 11
df.at[11, 'Test'] = "eGFR"
df.at[11, '16/01/21'] = ">90"
df.at[11, '18/09/21'] = "88"
df.at[11, '10/10/22'] = "76"
df.at[11, 'Units'] = "mL/min/1.73m2"
df.at[11, 'Reference'] = "(>59)"
# Slice the data from 4 to 26 index
Slicedf1 = df.iloc[5:27, :]
Slicedf1
```

Out[7]:

	Test	16/01/21	18/09/21	10/10/22	Units	Reference
5	Sodium	141	136	140	mmol/L	(135-145)
6	Potassium	4.5	4.4	4.4	mmol/L	(3.5-5.5)
7	Chloride	105	104	101	mmol/L	(95-110)
8	Bicarbonate	27	24	28	mmol/L	(20-32)
9	Urea	3.0	4.4	5.2	mmol/L	(3.0-8.5)
10	Creatinine	60	65	70	umol/L	(45-85)
11	eGFR	>90	88	76	mL/min/1.73m2	(>59)
12	Urate	0.30	0.30	0.29	mmol/L	(0.15-0.40)
13	Calcium	2.34	2.25	2.51	mmol/L	(2.15-2.55)
14	Corr Calcium	2.40	2.37	2.51	mmol/L	(2.15-2.55)
15	Phosphate.	0.95	1.07	1.03	mmol/L	(0.8-1.5)
16	Bili.Total	3	8	7	umol/L	(3-15)
17	ALP	64	77	76	U/L	(30-115)
18	GGT	11	11	12	U/L	(5-35)
19	LD	130	133	151	U/L	(120-250)
20	AST	14	15	17	U/L	(10-35)
21	ALT	8	8	6	U/L	(5-30)
22	Total Protein	71	68	71	g/L	(64-83)
23	Albumin	40	37	43	g/L	(36-47)
24	Globulin	31	31	28	g/L	(23-39)
25	Cholesterol	5.3	5.7	5.6	mmol/L	<5.5
26	Triglycerides	1.2	1.2	1.0	mmol/L	(<2.0)

In [8]:
```
#Reset the index from 0 to 22
df1 = Slicedf1.reset_index(drop=True)
df1
```

Out[8]:

	Test	16/01/21	18/09/21	10/10/22	Units	Reference
0	Sodium	141	136	140	mmol/L	(135-145)
1	Potassium	4.5	4.4	4.4	mmol/L	(3.5-5.5)
2	Chloride	105	104	101	mmol/L	(95-110)
3	Bicarbonate	27	24	28	mmol/L	(20-32)
4	Urea	3.0	4.4	5.2	mmol/L	(3.0-8.5)
5	Creatinine	60	65	70	umol/L	(45-85)
6	eGFR	>90	88	76	mL/min/1.73m2	(>59)
7	Urate	0.30	0.30	0.29	mmol/L	(0.15-0.40)
8	Calcium	2.34	2.25	2.51	mmol/L	(2.15-2.55)
9	Corr Calcium	2.40	2.37	2.51	mmol/L	(2.15-2.55)
10	Phosphate.	0.95	1.07	1.03	mmol/L	(0.8-1.5)
11	Bili.Total	3	8	7	umol/L	(3-15)
12	ALP	64	77	76	U/L	(30-115)
13	GGT	11	11	12	U/L	(5-35)
14	LD	130	133	151	U/L	(120-250)
15	AST	14	15	17	U/L	(10-35)
16	ALT	8	8	6	U/L	(5-30)
17	Total Protein	71	68	71	g/L	(64-83)
18	Albumin	40	37	43	g/L	(36-47)
19	Globulin	31	31	28	g/L	(23-39)
20	Cholesterol	5.3	5.7	5.6	mmol/L	<5.5
21	Triglycerides	1.2	1.2	1.0	mmol/L	(<2.0)

In [9]:
```
import pytesseract
from PIL import Image

# Set the path to the Tesseract executable (replace with your own path)
pytesseract.tesseract_cmd = r'/opt/homebrew/bin/tesseract'
# Open the image file
image13 = Image.open('/Users/prathyushavajinepally/Downloads/page_13.png
image14 = Image.open('/Users/prathyushavajinepally/Downloads/page_14.png

# Perform OCR to extract text from the image
text13 = pytesseract.image_to_string(image13)
text14 = pytesseract.image_to_string(image14)


# Print the extracted text
print(text13)
print(text14)
```

```
Reference intervals are included for referenc
treatment goals should be guided by patient-s
assessment (see Australian Cardiovascular Ris
web-site www.cvdcheck.org.au
risk assessment for individual patients.)

e only, and interpretation /
pecific cardiovascular risk
k Charts. Alternatively, the

can be accessed in order to complete a

Haemolysis Nil Nil Nil
Icterus Nil Nil Nil
Lipaemia Nil Nil Nil
Fasting status Fasting Fasting Fasting
Chol (3.9-5.2) mmol/L 4.7 4.5 4.1
Trig (0.5-1.7) mmol/L 1.5 1.2 1.0
HDL (1.0-2.0) mmol/L 1.0 1.2 1.3
```

In [26]:
```
test_patterns = [
    r"(\w+)\s+(\([\w\s\.\-<]+\))\s+(\w+/\w+)?\s+([\d\.\-<]+)\s+([\d\.\-<
    r"eGFR\s+mL/min/1\.73m\*2\s+>\s+(\d+)\s+(\d+)\s+(\d+)",  # Index 8
    r"(\w+)\s+\((\d+\.\d+-\d+\.\d+)\)(\s+\w+)?\s+([\d\.\-<]+)\s+([\d\.\-
]

data_dict = {
    'Test': [],
    'Reference': [],
    'Units': [],
    'Result1': [],
    'Result2': [],
    'Result3': []
}

for i, pattern in enumerate(test_patterns):
    matches = re.findall(pattern, text14)
    for match in matches:
        if i == 0:
            test = match[0]
            reference = match[1].strip('()')
            units = match[2].strip() if match[2] else ""
            result1 = match[3] if len(match) > 3 else ""
            result2 = match[4] if len(match) > 4 else ""
            result3 = match[5] if len(match) > 5 else ""
        elif i == 1:
            test = "eGFR"
            reference = ""
            units = "mL/min/1.73m*2"
            result1, result2, result3 = match
        else:
            test = match[0]
            reference = match[1].strip('()')
            units = match[2].strip() if match[2] else ""
            result1 = match[3]
            result2 = match[4]
            result3 = match[5] if len(match) > 5 else ""

        # Exclude the specific line "Serum    3.4-5.4          mmol/L
        if test != "Serum" or reference != "3.4-5.4" or units != "mmol/L
            data_dict['Test'].append(test)
            data_dict['Reference'].append(reference)
            data_dict['Units'].append(units)
            data_dict['Result1'].append(result1)
            data_dict['Result2'].append(result2)
            data_dict['Result3'].append(result3)

df2 = pd.DataFrame(data_dict)
df2
```

Out[26]:

	Test	Reference	Units	Result1	Result2	Result3
0	Na	135-145	mmol/L	141	140	142
1	K	3.6-5.4	mmol/L	4.3	4.3	4.1
2	cl	95-110	mmol/L	106	100	100
3	HCO3	22-32	mmol/L	26	24	24
4	Gap	10-20	mmol/L	13	20	22
5	Urea	2.5-9.0	mmol/L	4.6	5.3	6.0
6	Creat	45-90	umol/L	60	65	70
7	Urate	0.14-0.36	mmol/L	0.32	0.30	
8	Bili	< 15	umol/L	5	4	6
9	AST	< 35	U/L	11	16	14
10	ALT	< 30	U/L	<7	9	7
11	GGT	< 35	U/L	11	8	10
12	Phos	30-115	U/L	72	64	67
13	Protein	60-82	g/L	66	70	68
14	Albumin	36-48	g/L	39	42	42
15	Glob	20-39	g/L	27	28	26
16	Ca	2.10-2.60	mmol/L	2.43	2.57	
17	PO4	0.75-1.50	mmol/L	1.10	1.01	
18	eGFR		mL/min/1.73m*2	90	84	78
19	Ca	2.10-2.60		2.51	2.59	

In [27]:
```
import pandas as pd

#combined_df = pd.concat([df1, df2], ignore_index=True)
combined_df = pd.concat([df1, df2], axis=0)
transposed_df = combined_df.set_index('Test').transpose().reset_index(dr
new_row_names = ['16 Jan 21', '18 Sep 21', '10 Nov 22', 'Units', 'Refere
transposed_df = transposed_df.set_axis(new_row_names, axis=0)
```

In [28]:
```
# Check if 'Units' and 'Reference' rows exist before dropping
rows_to_drop = ['Units', 'Reference']
if all(row_name in transposed_df.index for row_name in rows_to_drop):
    transposed_df = transposed_df.drop(rows_to_drop)

# Add time column at index 0
time_values = ['09.10', '09.03', '09.02', '08:15', '07:30', '08:57']  # 
transposed_df.insert(0, 'Time', time_values)

# Reset the index to move the 'Test' column to a regular column
transposed_df = transposed_df.reset_index()

# Rename the 'index' column to 'Dates'
transposed_df = transposed_df.rename(columns={'index': 'Dates'})

# Convert the 'Dates' column to datetime format
transposed_df['Dates'] = pd.to_datetime(transposed_df['Dates'], format='
# Sort the DataFrame by the 'Dates' column in ascending order
transposed_df = transposed_df.sort_values('Dates', ascending=True)

# Reset the index
transposed_df = transposed_df.reset_index(drop=True)
```

In [29]:
```
transposed_df
```

Out[29]:

Test	Dates	Time	Sodium	Potassium	Chloride	Bicarbonate	Urea	Creatinine	eGFR	Urate	...
0	2020-11-19	08:15	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...
1	2021-01-16	09.10	141	4.5	105	27	3.0	60	>90	0.30	...
2	2021-09-18	09.03	136	4.4	104	24	4.4	65	88	0.30	...
3	2022-11-10	09.02	140	4.4	101	28	5.2	70	76	0.29	...
4	2022-11-16	07:30	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...
5	2023-02-16	08:57	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...

6 rows × 44 columns

In [48]:
```
transposeddf=transposed_df.to_csv('combinedserum.csv', index=False)
transposeddf
```

In [18]:
```
column_names = [(idx, name) for idx, name in enumerate(transposed_df.col

# Print column names with indices
for idx, name in column_names:
    print("Index: {}, Column Name: '{}'".format(idx, name))
```

```
Index: 0, Column Name: 'Dates'
Index: 1, Column Name: 'Time'
Index: 2, Column Name: 'Sodium'
Index: 3, Column Name: 'Potassium'
Index: 4, Column Name: 'Chloride'
Index: 5, Column Name: 'Bicarbonate'
Index: 6, Column Name: 'Urea'
Index: 7, Column Name: 'Creatinine'
Index: 8, Column Name: 'eGFR'
Index: 9, Column Name: 'Urate'
Index: 10, Column Name: 'Calcium'
Index: 11, Column Name: 'Corr Calcium'
Index: 12, Column Name: 'Phosphate.'
Index: 13, Column Name: 'Bili.Total'
Index: 14, Column Name: 'ALP'
Index: 15, Column Name: 'GGT'
Index: 16, Column Name: 'LD'
Index: 17, Column Name: 'AST'
Index: 18, Column Name: 'ALT'
Index: 19, Column Name: 'Total Protein'
Index: 20, Column Name: 'Albumin'
Index: 21, Column Name: 'Globulin'
Index: 22, Column Name: 'Cholesterol'
Index: 23, Column Name: 'Triglycerides'
Index: 24, Column Name: 'Na'
Index: 25, Column Name: 'K'
Index: 26, Column Name: 'cl'
Index: 27, Column Name: 'HCO3'
Index: 28, Column Name: 'Gap'
Index: 29, Column Name: 'Urea'
Index: 30, Column Name: 'Creat'
Index: 31, Column Name: 'Urate'
Index: 32, Column Name: 'Bili'
Index: 33, Column Name: 'AST'
Index: 34, Column Name: 'ALT'
Index: 35, Column Name: 'GGT'
Index: 36, Column Name: 'Phos'
Index: 37, Column Name: 'Protein'
Index: 38, Column Name: 'Albumin'
Index: 39, Column Name: 'Glob'
Index: 40, Column Name: 'Ca'
Index: 41, Column Name: 'PO4'
Index: 42, Column Name: 'eGFR'
Index: 43, Column Name: 'Ca'
```

In [30]:
```
transposed_df.columns.values[43] = 'Corr ca'
transposed_df.columns.values[31] = 'Urt'
transposed_df.columns.values[40] = 'Ca2'
transposed_df.columns.values[38] = 'Albu'
transposed_df.columns.values[42] = 'egfr'
transposed_df.columns.values[34] = 'alt'
transposed_df.columns.values[35] = 'ggt'
transposed_df.columns.values[33] = 'ast'
transposed_df.columns.values[29] = 'urea'
transposed_df
```

Out[30]:

Test	Dates	Time	Sodium	Potassium	Chloride	Bicarbonate	Urea	Creatinine	eGFR	Urate	...
0	2020-11-19	08:15	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...
1	2021-01-16	09.10	141	4.5	105	27	3.0	60	>90	0.30	...
2	2021-09-18	09.03	136	4.4	104	24	4.4	65	88	0.30	...
3	2022-11-10	09.02	140	4.4	101	28	5.2	70	76	0.29	...
4	2022-11-16	07:30	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...
5	2023-02-16	08:57	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...

6 rows × 44 columns

In [21]:
```
column_names = [(idx, name) for idx, name in enumerate(transposed_df.col

# Print column names with indices
for idx, name in column_names:
    print("Index: {}, Column Name: '{}'".format(idx, name))
```

```
Index: 0, Column Name: 'Dates'
Index: 1, Column Name: 'Time'
Index: 2, Column Name: 'Sodium'
Index: 3, Column Name: 'Potassium'
Index: 4, Column Name: 'Chloride'
Index: 5, Column Name: 'Bicarbonate'
Index: 6, Column Name: 'Urea'
Index: 7, Column Name: 'Creatinine'
Index: 8, Column Name: 'eGFR'
Index: 9, Column Name: 'Urate'
Index: 10, Column Name: 'Calcium'
Index: 11, Column Name: 'Corr Calcium'
Index: 12, Column Name: 'Phosphate.'
Index: 13, Column Name: 'Bili.Total'
Index: 14, Column Name: 'ALP'
Index: 15, Column Name: 'GGT'
Index: 16, Column Name: 'LD'
Index: 17, Column Name: 'AST'
Index: 18, Column Name: 'ALT'
Index: 19, Column Name: 'Total Protein'
Index: 20, Column Name: 'Albumin'
Index: 21, Column Name: 'Globulin'
Index: 22, Column Name: 'Cholesterol'
Index: 23, Column Name: 'Triglycerides'
Index: 24, Column Name: 'K'
Index: 25, Column Name: 'cl'
Index: 26, Column Name: 'HCO3'
Index: 27, Column Name: 'Gap'
Index: 28, Column Name: 'urea'
Index: 29, Column Name: 'Creat'
Index: 30, Column Name: 'Urt'
Index: 31, Column Name: 'Bili'
Index: 32, Column Name: 'ast'
Index: 33, Column Name: 'alt'
Index: 34, Column Name: 'ggt'
Index: 35, Column Name: 'Phos'
Index: 36, Column Name: 'Protein'
Index: 37, Column Name: 'Albu'
Index: 38, Column Name: 'Glob'
Index: 39, Column Name: 'Ca2'
Index: 40, Column Name: 'PO4'
Index: 41, Column Name: 'egfr'
Index: 42, Column Name: 'Corr ca'
```

In [31]:
```
column_to_fill = 'Sodium'   # Specify the column to fill with values
column_to_drop = 'Na'
## Replace NaN values in 'column_to_fill' with values from 'column_to_dr
transposed_df[column_to_fill] = transposed_df[column_to_fill].fillna(tra
transposed_df.drop(column_to_drop, axis=1, inplace=True)

column_to_drop = 'K'
column_to_fill = 'Potassium'
## Replace NaN values in 'column_to_fill' with values from 'column_to_dr
transposed_df[column_to_fill] = transposed_df[column_to_fill].fillna(tra
transposed_df.drop(column_to_drop, axis=1, inplace=True)

column_to_drop = 'HCO3'       # Specify the column to drop
column_to_fill = 'Bicarbonate'
## Replace NaN values in 'column_to_fill' with values from 'column_to_dr
transposed_df[column_to_fill] = transposed_df[column_to_fill].fillna(tra
transposed_df.drop(column_to_drop, axis=1, inplace=True)

column_to_drop = 'cl'       # Specify the column to drop
column_to_fill = 'Chloride'
## Replace NaN values in 'column_to_fill' with values from 'column_to_dr
transposed_df[column_to_fill] = transposed_df[column_to_fill].fillna(tra
transposed_df.drop(column_to_drop, axis=1, inplace=True)

column_to_drop = 'Creat'       # Specify the column to drop
column_to_fill = 'Creatinine'
## Replace NaN values in 'column_to_fill' with values from 'column_to_dr
transposed_df[column_to_fill] = transposed_df[column_to_fill].fillna(tra
transposed_df.drop(column_to_drop, axis=1, inplace=True)

column_to_drop = 'Protein'       # Specify the column to drop
column_to_fill = 'Total Protein'
## Replace NaN values in 'column_to_fill' with values from 'column_to_dr
transposed_df[column_to_fill] = transposed_df[column_to_fill].fillna(tra
transposed_df.drop(column_to_drop, axis=1, inplace=True)

column_to_drop = 'Bili'       # Specify the column to drop
column_to_fill = 'Bili.Total'
## Replace NaN values in 'column_to_fill' with values from 'column_to_dr
transposed_df[column_to_fill] = transposed_df[column_to_fill].fillna(tra
transposed_df.drop(column_to_drop, axis=1, inplace=True)

column_to_drop = 'urea'       # Specify the column to drop
column_to_fill = 'Urea'
## Replace NaN values in 'column_to_fill' with values from 'column_to_dr
transposed_df[column_to_fill] = transposed_df[column_to_fill].fillna(tra
transposed_df.drop(column_to_drop, axis=1, inplace=True)

column_to_drop = 'Corr ca'       # Specify the column to drop
column_to_fill = 'Corr Calcium'
## Replace NaN values in 'column_to_fill' with values from 'column_to_dr
transposed_df[column_to_fill] = transposed_df[column_to_fill].fillna(tra
transposed_df.drop(column_to_drop, axis=1, inplace=True)

column_to_drop = 'Urt'       # Specify the column to drop
column_to_fill = 'Urate'
## Replace NaN values in 'column_to_fill' with values from 'column_to_dr
```

```
transposed_df[column_to_fill] = transposed_df[column_to_fill].fillna(tra
transposed_df.drop(column_to_drop, axis=1, inplace=True)

column_to_drop = 'Ca2'       # Specify the column to drop
column_to_fill = 'Calcium'
## Replace NaN values in 'column_to_fill' with values from 'column_to_dr
transposed_df[column_to_fill] = transposed_df[column_to_fill].fillna(tra
transposed_df.drop(column_to_drop, axis=1, inplace=True)

column_to_drop = 'ast'       # Specify the column to drop
column_to_fill = 'AST'
## Replace NaN values in 'column_to_fill' with values from 'column_to_dr
transposed_df[column_to_fill] = transposed_df[column_to_fill].fillna(tra
transposed_df.drop(column_to_drop, axis=1, inplace=True)

column_to_drop = 'Glob'       # Specify the column to drop
column_to_fill = 'Globulin'
## Replace NaN values in 'column_to_fill' with values from 'column_to_dr
transposed_df[column_to_fill] = transposed_df[column_to_fill].fillna(tra
transposed_df.drop(column_to_drop, axis=1, inplace=True)

column_to_drop = 'alt'       # Specify the column to drop
column_to_fill = 'ALT'
## Replace NaN values in 'column_to_fill' with values from 'column_to_dr
transposed_df[column_to_fill] = transposed_df[column_to_fill].fillna(tra
transposed_df.drop(column_to_drop, axis=1, inplace=True)

column_to_drop = 'Albu'       # Specify the column to drop
column_to_fill = 'Albumin'
## Replace NaN values in 'column_to_fill' with values from 'column_to_dr
transposed_df[column_to_fill] = transposed_df[column_to_fill].fillna(tra
transposed_df.drop(column_to_drop, axis=1, inplace=True)

column_to_drop = 'egfr'       # Specify the column to drop
column_to_fill = 'eGFR'
## Replace NaN values in 'column_to_fill' with values from 'column_to_dr
transposed_df[column_to_fill] = transposed_df[column_to_fill].fillna(tra
transposed_df.drop(column_to_drop, axis=1, inplace=True)



transposed_df
```

Out[31]:

	Test	Dates	Time	Sodium	Potassium	Chloride	Bicarbonate	Urea	Creatinine	eGFR	Urate	...
0		2020-11-19	08:15	141	4.3	106	26	4.6	60	90	0.32	...
1		2021-01-16	09.10	141	4.5	105	27	3.0	60	>90	0.30	...
2		2021-09-18	09.03	136	4.4	104	24	4.4	65	88	0.30	...
3		2022-11-10	09.02	140	4.4	101	28	5.2	70	76	0.29	...
4		2022-11-16	07:30	140	4.3	100	24	5.3	65	84	0.30	...
5		2023-02-16	08:57	142	4.1	100	24	6.0	70	78		...

6 rows × 28 columns

In [32]:
```
column_to_drop = 'ggt'        # Specify the column to drop
column_to_fill = 'GGT'
## Replace NaN values in 'column_to_fill' with values from 'column_to_dr
transposed_df[column_to_fill] = transposed_df[column_to_fill].fillna(tra
column_to_drop = 'Phos'       # Specify the column to drop
column_to_fill = 'Phosphate.'
## Replace NaN values in 'column_to_fill' with values from 'column_to_dr
transposed_df[column_to_fill] = transposed_df[column_to_fill].fillna(tra

# Drop the 'Na' column
transposed_df.drop(column_to_drop, axis=1, inplace=True)
transposed_df
```

Out[32]:

	Test	Dates	Time	Sodium	Potassium	Chloride	Bicarbonate	Urea	Creatinine	eGFR	Urate	...
0		2020-11-19	08:15	141	4.3	106	26	4.6	60	90	0.32	...
1		2021-01-16	09.10	141	4.5	105	27	3.0	60	>90	0.30	...
2		2021-09-18	09.03	136	4.4	104	24	4.4	65	88	0.30	...
3		2022-11-10	09.02	140	4.4	101	28	5.2	70	76	0.29	...
4		2022-11-16	07:30	140	4.3	100	24	5.3	65	84	0.30	...
5		2023-02-16	08:57	142	4.1	100	24	6.0	70	78		...

6 rows × 27 columns

In [ ]:
```
column_names = [(idx, name) for idx, name in enumerate(transposed_df.col

# Print column names with indices
for idx, name in column_names:
    print("Index: {}, Column Name: '{}'".format(idx, name))
```

In [ ]:
```
column_names = [(idx, name) for idx, name in enumerate(transposed_df.col

# Print column names with indices
for idx, name in column_names:
    print("Index: {}, Column Name: '{}'".format(idx, name))
```

In [ ]:
```
import seaborn as sns
sns.distplot(transposed_df['Sodium'])
```

In [2]:
```
transposeddf=transposed_df.to_csv('Serum_chemistry.csv', index=False)
transposeddf
```

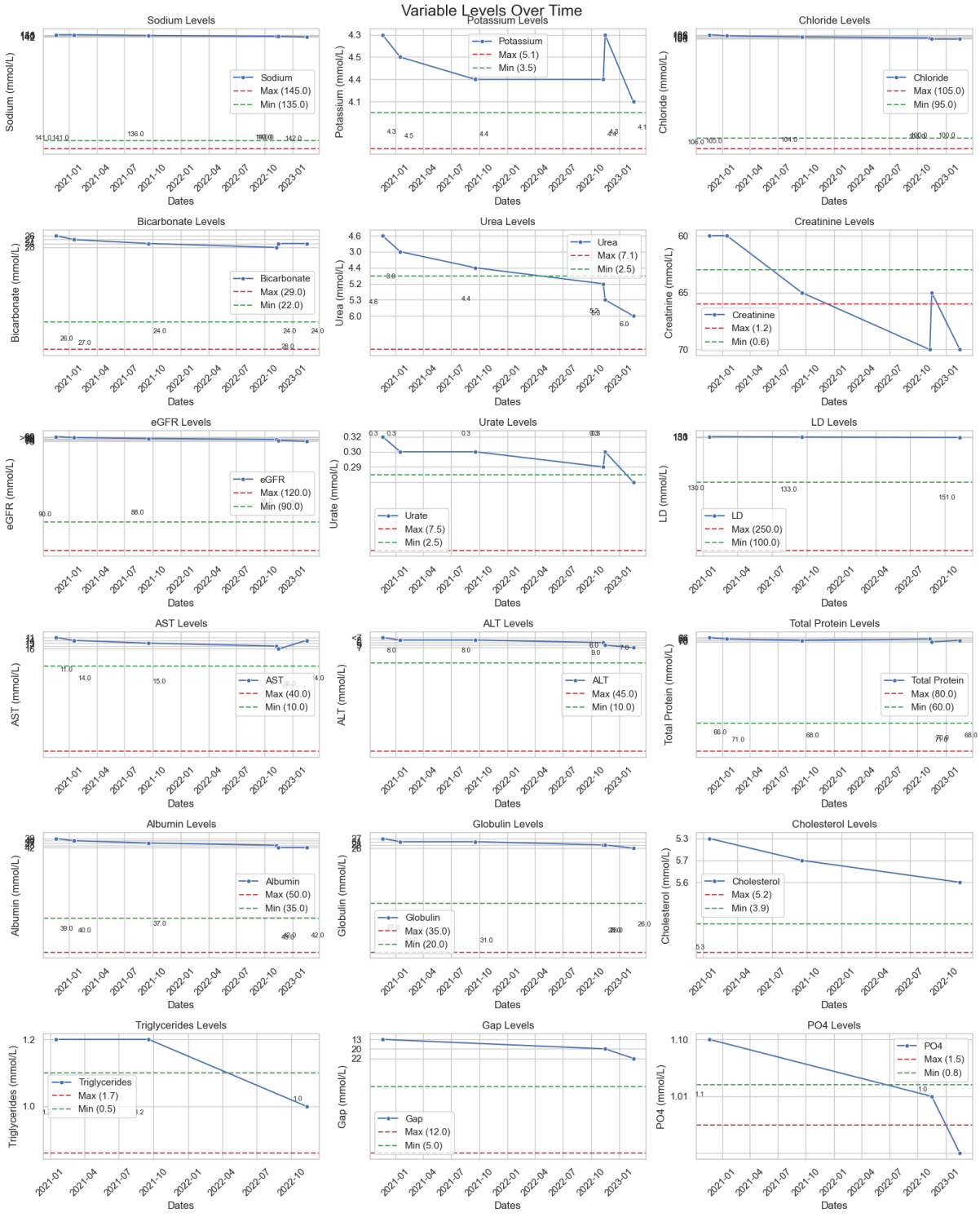
```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call l
ast)
/var/folders/4v/mp4s48nd11q6nq8xyr15v52w0000gn/T/ipykernel_7237/2015673
751.py in <module>
----> 1 transposeddf=transposed_df.to_csv('Serum_chemistry.csv', index=
False)
      2 transposeddf

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

In [1]:
```
transposeddf
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call l
ast)
/var/folders/4v/mp4s48nd11q6nq8xyr15v52w0000gn/T/ipykernel_7237/1619083
30.py in <module>
----> 1 transposeddf

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

In [33]:
```
import seaborn as sns
import matplotlib.pyplot as plt
import pandas as pd
from matplotlib.dates import AutoDateLocator, AutoDateFormatter

sns.set(style="whitegrid")

# Convert date strings to datetime format
transposed_df['Dates'] = pd.to_datetime(transposed_df['Dates'],format='%

# Create a list of variables to plot
variables = ['Sodium', 'Potassium', 'Chloride', 'Bicarbonate', 'Urea', '
             'eGFR', 'Urate', 'LD', 'AST', 'ALT', 'Total Protein', 'Albu
             'Globulin', 'Cholesterol', 'Triglycerides', 'Gap', 'PO4']

# Define thresholds for specific variables (replace these with appropria
thresholds = {'Sodium': (135, 145),
              'Potassium': (3.5, 5.1),
              'Chloride': (95, 105),
              'Bicarbonate': (22, 29),
              'Urea': (2.5, 7.1),
              'Creatinine': (0.6, 1.2),
              'eGFR': (90, 120),
              'Urate': (2.5, 7.5),
              'LD': (100, 250),
              'AST': (10, 40),
              'ALT': (10, 45),
              'Total Protein': (60, 80),
              'Albumin': (35, 50),
              'Globulin': (20, 35),
              'Cholesterol': (3.9, 5.2),
              'Triglycerides': (0.5, 1.7),
              'Gap': (5, 12),
              'PO4': (0.8, 1.5)}
num_plots = len(variables)
num_cols = 3
num_rows = (num_plots + num_cols - 1) // num_cols
# Calculate the number of rows and columns for the subplots
#num_plots = len(variables)
#num_rows = int(num_plots ** 0.5)
#num_cols = int(num_plots / num_rows) if num_plots % num_rows == 0 else

# Create subplots in a grid layout with larger size
fig, axes = plt.subplots(num_rows, num_cols, figsize=(16, 20))
fig.suptitle('Variable Levels Over Time', fontsize=18)

# Plot each variable in a separate subplot using Seaborn's lineplot
for i, var in enumerate(variables):
    row = i // num_cols
    col = i % num_cols
    ax = axes[row, col] if num_rows > 1 else axes[col]

    # Plot line chart using Seaborn's lineplot with markers and labels
    sns.lineplot(data=transposed_df, x='Dates', y=var, ax=ax, marker='o'

    # Convert the marker values column to numeric type
    transposed_df[var] = pd.to_numeric(transposed_df[var], errors='coerc
```

```
    # Annotate marker values
    for x_val, y_val in zip(transposed_df['Dates'], transposed_df[var]):
        if not pd.isnull(y_val):
            if transposed_df[var].diff().iloc[1] > 0:
                xytext = (5, -5)
                ha = 'left'
            else:
                xytext = (-5, 5)
                ha = 'right'
            ax.annotate(f'{y_val:.1f}', (x_val, y_val), xytext=xytext, t

    ax.set_xlabel('Dates')
    ax.set_ylabel(var + ' (mmol/L)')  # Replace 'mmol/L' with appropriat
    ax.set_title(var + ' Levels')

    # Specify horizontal line types for thresholds
    if var in thresholds:
        min_val, max_val = thresholds[var]
        ax.axhline(y=max_val, color='r', linestyle='--', label=f'Max ({m
        ax.axhline(y=min_val, color='g', linestyle='--', label=f'Min ({m
        ax.legend()

    # Configure x-axis tick locations and labels
    locator = AutoDateLocator()
    formatter = AutoDateFormatter(locator)
    ax.xaxis.set_major_locator(locator)
    ax.xaxis.set_major_formatter(formatter)
    plt.setp(ax.xaxis.get_majorticklabels(), rotation=45)

# Remove any empty subplots
if num_plots < num_rows * num_cols:
    if num_rows > 1:
        axes[-1, -1].axis('off')
    else:
        axes[-1].axis('off')

plt.tight_layout(rect=[0, 0, 1, 0.96])
plt.tight_layout()
plt.show()
```


Variable Levels Over Time
Sodium Levels
Sodium (mmol/L)
Sodium
Max (145.0)
Min (135.0)
Potassium Levels
Potassium (mmol/L)
Potassium
Max (5.1)
Min (3.5)
Chloride Levels
Chloride (mmol/L)
Chloride
Max (105.0)
Min (95.0)
Bicarbonate Levels
Bicarbonate (mmol/L)
Bicarbonate
Max (29.0)
Min (22.0)
Urea Levels
Urea (mmol/L)
Urea
Max (7.1)
Min (2.5)
Creatinine Levels
Creatinine (mmol/L)
Creatinine
Max (1.2)
Min (0.6)
eGFR Levels
eGFR (mmol/L)
eGFR
Max (120.0)
Min (90.0)
Urate Levels
Urate (mmol/L)
Urate
Max (7.5)
Min (2.5)
LD Levels
LD (mmol/L)
LD
Max (250.0)
Min (100.0)
AST Levels
AST (mmol/L)
AST
Max (40.0)
Min (10.0)
ALT Levels
ALT (mmol/L)
ALT
Max (45.0)
Min (10.0)
Total Protein Levels
Total Protein (mmol/L)
Total Protein
Max (80.0)
Min (60.0)
Albumin Levels
Albumin (mmol/L)
Albumin
Max (50.0)
Min (35.0)
Globulin Levels
Globulin (mmol/L)
Globulin
Max (35.0)
Min (20.0)
Cholesterol Levels
Cholesterol (mmol/L)
Cholesterol
Max (5.2)
Min (3.9)
Triglycerides Levels
Triglycerides (mmol/L)
Triglycerides
Max (1.7)
Min (0.5)
Gap Levels
Gap (mmol/L)
Gap
Max (12.0)
Min (5.0)
PO4 Levels
PO4 (mmol/L)
PO4
Max (1.5)
Min (0.8)
Dates

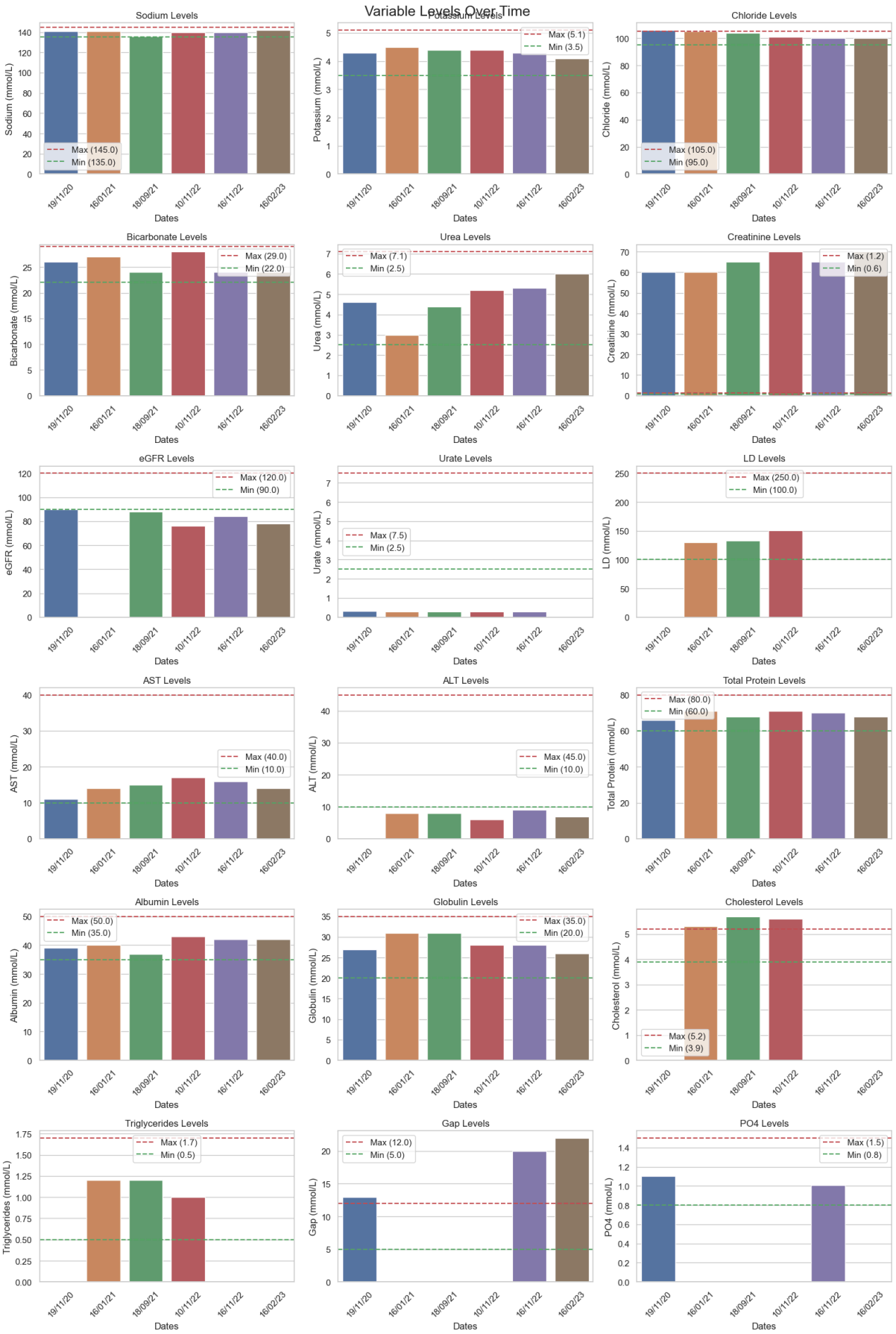
In [43]:
```
import seaborn as sns
import matplotlib.pyplot as plt
import pandas as pd
from matplotlib.dates import AutoDateLocator, AutoDateFormatter

sns.set(style="darkgrid")

# Convert date strings to datetime format
transposed_df['Dates'] = pd.to_datetime(transposed_df['Dates'], format='
# Create a list of variables to plot
variables = ['Sodium', 'Potassium', 'Chloride', 'Bicarbonate', 'Urea', '
             'eGFR', 'Urate', 'LD', 'AST', 'ALT', 'Total Protein', 'Albu
             'Globulin', 'Cholesterol', 'Triglycerides', 'Gap', 'PO4']

# Define thresholds for specific variables (replace these with appropria
thresholds = {'Sodium': (135, 145),
              'Potassium': (3.5, 5.1),
              'Chloride': (95, 105),
              'Bicarbonate': (22, 29),
              'Urea': (2.5, 7.1),
              'Creatinine': (0.6, 1.2),
              'eGFR': (90, 120),
              'Urate': (2.5, 7.5),
              'LD': (100, 250),
              'AST': (10, 40),
              'ALT': (10, 45),
              'Total Protein': (60, 80),
              'Albumin': (35, 50),
              'Globulin': (20, 35),
              'Cholesterol': (3.9, 5.2),
              'Triglycerides': (0.5, 1.7),
              'Gap': (5, 12),
              'PO4': (0.8, 1.5)}

num_plots = len(variables)
num_cols = 3
num_rows = (num_plots + num_cols - 1) // num_cols

# Create subplots in a grid layout with larger size
fig, axes = plt.subplots(num_rows, num_cols, figsize=(16, 4*num_rows))
fig.suptitle('Variable Levels Over Time', fontsize=18)

# Plot each variable in a separate subplot using Seaborn's distplot
for i, var in enumerate(variables):
    row = i // num_cols
    col = i % num_cols
    ax = axes[row, col] if num_rows > 1 else axes[col]

    # Plot histogram and kernel density estimate using Seaborn's distplo
    sns.distplot(transposed_df[var], ax=ax, bins=15, kde=True, hist_kws=

    # Annotate threshold lines
    if var in thresholds:
        min_val, max_val = thresholds[var]
        ax.axvline(x=max_val, color='r', linestyle='--', label=f'Max ({m
        ax.axvline(x=min_val, color='g', linestyle='--', label=f'Min ({m
```

```
        ax.legend()

    ax.set_xlabel(var + ' (mmol/L)')  # Replace 'mmol/L' with appropriat
    ax.set_title(var + ' Levels')

plt.tight_layout(rect=[0, 0, 1, 0.96])
plt.tight_layout()
plt.show()
```

```
/Users/prathyushavajinepally/opt/anaconda3/lib/python3.9/site-packages/
seaborn/distributions.py:2619: FutureWarning: `distplot` is a deprecate
d function and will be removed in a future version. Please adapt your c
ode to use either `displot` (a figure-level function with similar flexi
bility) or `histplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
/Users/prathyushavajinepally/opt/anaconda3/lib/python3.9/site-packages/
seaborn/distributions.py:2619: FutureWarning: `distplot` is a deprecate
d function and will be removed in a future version. Please adapt your c
ode to use either `displot` (a figure-level function with similar flexi
bility) or `histplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
/Users/prathyushavajinepally/opt/anaconda3/lib/python3.9/site-packages/
seaborn/distributions.py:2619: FutureWarning: `distplot` is a deprecate
d function and will be removed in a future version. Please adapt your c
ode to use either `displot` (a figure-level function with similar flexi
bility) or `histplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
/Users/prathyushavajinepally/opt/anaconda3/lib/python3.9/site-packages/
```

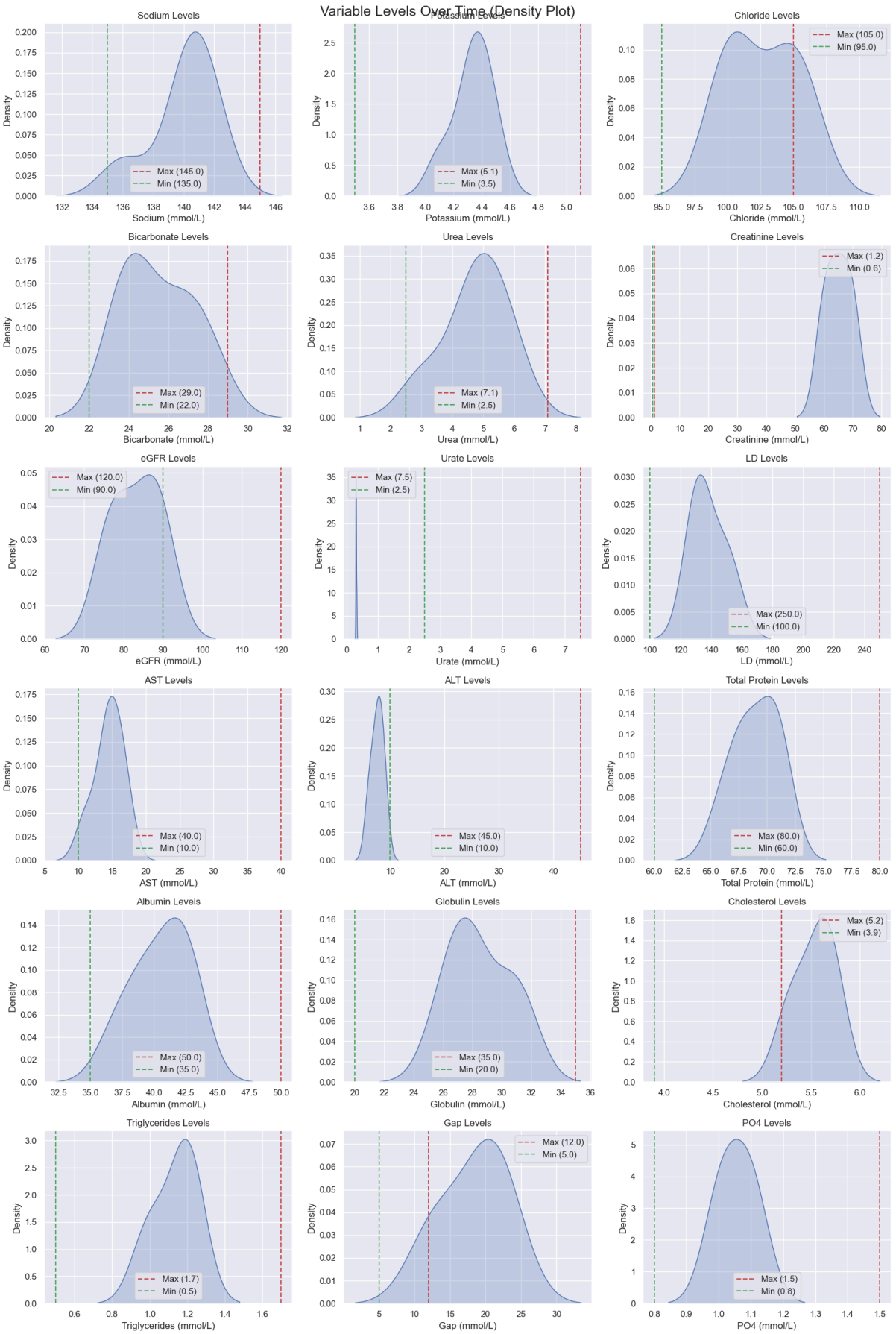
In [53]:
```
import seaborn as sns
import matplotlib.pyplot as plt
import pandas as pd
from matplotlib.dates import AutoDateLocator, AutoDateFormatter

sns.set(style="whitegrid")

# Convert date strings to datetime format
transposed_df['Dates'] = pd.to_datetime(transposed_df['Dates'], format='
# Create a list of variables to plot
variables = ['Sodium', 'Potassium', 'Chloride', 'Bicarbonate', 'Urea', '
             'eGFR', 'Urate', 'LD', 'AST', 'ALT', 'Total Protein', 'Albu
             'Globulin', 'Cholesterol', 'Triglycerides', 'Gap', 'PO4']

# Define thresholds for specific variables (replace these with appropria
thresholds = {'Sodium': (135, 145),
              'Potassium': (3.5, 5.1),
              'Chloride': (95, 105),
              'Bicarbonate': (22, 29),
              'Urea': (2.5, 7.1),
              'Creatinine': (0.6, 1.2),
              'eGFR': (90, 120),
              'Urate': (2.5, 7.5),
              'LD': (100, 250),
              'AST': (10, 40),
              'ALT': (10, 45),
              'Total Protein': (60, 80),
              'Albumin': (35, 50),
              'Globulin': (20, 35),
              'Cholesterol': (3.9, 5.2),
              'Triglycerides': (0.5, 1.7),
              'Gap': (5, 12),
              'PO4': (0.8, 1.5)}

num_plots = len(variables)
num_cols = 3
num_rows = (num_plots + num_cols - 1) // num_cols

# Create subplots in a grid layout with larger size
fig, axes = plt.subplots(num_rows, num_cols, figsize=(16, 4*num_rows))
fig.suptitle('Variable Levels Over Time', fontsize=18)

# Create bar plot for each variable showing the mean value
for i, var in enumerate(variables):
    row = i // num_cols
    col = i % num_cols
    ax = axes[row, col] if num_rows > 1 else axes[col]

    # Plot bar plot using Seaborn's barplot
    sns.barplot(data=transposed_df, x='Dates', y=var, ax=ax)

    # Annotate threshold lines
    if var in thresholds:
        min_val, max_val = thresholds[var]
        ax.axhline(y=max_val, color='r', linestyle='--', label=f'Max ({m
        ax.axhline(y=min_val, color='g', linestyle='--', label=f'Min ({m
```

```
        ax.legend()

    ax.set_xlabel('Dates')
    ax.set_ylabel(var + ' (mmol/L)')  # Replace 'mmol/L' with appropriat
    ax.set_title(var + ' Levels')
    # Rotate x-axis tick labels by 45 degrees
    ax.set_xticklabels([date.strftime('%d/%m/%y') for date in transposed_

plt.tight_layout(rect=[0, 0, 1, 0.96])
plt.tight_layout()
plt.show()
```


Variable Levels Over Time
Sodium Levels
Sodium (mmol/L)
Max (145.0)
Min (135.0)
Potassium Levels
Potassium (mmol/L)
Max (5.1)
Min (3.5)
Chloride Levels
Chloride (mmol/L)
Max (105.0)
Min (95.0)
Bicarbonate Levels
Bicarbonate (mmol/L)
Max (29.0)
Min (22.0)
Urea Levels
Urea (mmol/L)
Max (7.1)
Min (2.5)
Creatinine Levels
Creatinine (mmol/L)
Max (1.2)
Min (0.6)
eGFR Levels
eGFR (mmol/L)
Max (120.0)
Min (90.0)
Urate Levels
Urate (mmol/L)
Max (7.5)
Min (2.5)
LD Levels
LD (mmol/L)
Max (250.0)
Min (100.0)
AST Levels
AST (mmol/L)
Max (40.0)
Min (10.0)
ALT Levels
ALT (mmol/L)
Max (45.0)
Min (10.0)
Total Protein Levels
Total Protein (mmol/L)
Max (80.0)
Min (60.0)
Albumin Levels
Albumin (mmol/L)
Max (50.0)
Min (35.0)
Globulin Levels
Globulin (mmol/L)
Max (35.0)
Min (20.0)
Cholesterol Levels
Cholesterol (mmol/L)
Max (5.2)
Min (3.9)
Triglycerides Levels
Triglycerides (mmol/L)
Max (1.7)
Min (0.5)
Gap Levels
Gap (mmol/L)
Max (12.0)
Min (5.0)
PO4 Levels
PO4 (mmol/L)
Max (1.5)
Min (0.8)
19/11/20
16/01/21
18/09/21
10/11/22
16/11/22
16/02/23
Dates

In [56]:
```
import seaborn as sns
import matplotlib.pyplot as plt
import pandas as pd
from matplotlib.dates import AutoDateLocator, AutoDateFormatter

sns.set(style="darkgrid")

# Convert date strings to datetime format
transposed_df['Dates'] = pd.to_datetime(transposed_df['Dates'], format='
# Create a list of variables to plot
variables = ['Sodium', 'Potassium', 'Chloride', 'Bicarbonate', 'Urea', '
             'eGFR', 'Urate', 'LD', 'AST', 'ALT', 'Total Protein', 'Albu
             'Globulin', 'Cholesterol', 'Triglycerides', 'Gap', 'PO4']

# Define thresholds for specific variables (replace these with appropria
thresholds = {'Sodium': (135, 145),
              'Potassium': (3.5, 5.1),
              'Chloride': (95, 105),
              'Bicarbonate': (22, 29),
              'Urea': (2.5, 7.1),
              'Creatinine': (0.6, 1.2),
              'eGFR': (90, 120),
              'Urate': (2.5, 7.5),
              'LD': (100, 250),
              'AST': (10, 40),
              'ALT': (10, 45),
              'Total Protein': (60, 80),
              'Albumin': (35, 50),
              'Globulin': (20, 35),
              'Cholesterol': (3.9, 5.2),
              'Triglycerides': (0.5, 1.7),
              'Gap': (5, 12),
              'PO4': (0.8, 1.5)}

num_plots = len(variables)
num_cols = 3
num_rows = (num_plots + num_cols - 1) // num_cols

# Create subplots in a grid layout with larger size
fig, axes = plt.subplots(num_rows, num_cols, figsize=(16, 4*num_rows))
fig.suptitle('Variable Levels Over Time (Density Plot)', fontsize=18)

# Create density plot for each variable
for i, var in enumerate(variables):
    row = i // num_cols
    col = i % num_cols
    ax = axes[row, col] if num_rows > 1 else axes[col]

    # Plot density plot using Seaborn's kdeplot
    sns.kdeplot(data=transposed_df, x=var, ax=ax, fill=True, common_norm

    # Annotate threshold lines
    if var in thresholds:
        min_val, max_val = thresholds[var]
        ax.axvline(x=max_val, color='r', linestyle='--', label=f'Max ({m
        ax.axvline(x=min_val, color='g', linestyle='--', label=f'Min ({m
```

```
        ax.legend()

    ax.set_xlabel(var + ' (mmol/L)')  # Replace 'mmol/L' with appropriat
    ax.set_title(var + ' Levels')

plt.tight_layout(rect=[0, 0, 1, 0.96])
plt.tight_layout()
plt.show()
```


Variable Levels Over Time (Density Plot)
Sodium Levels
Max (145.0)
Min (135.0)
Sodium (mmol/L)
Potassium Levels
Max (5.1)
Min (3.5)
Potassium (mmol/L)
Chloride Levels
Max (105.0)
Min (95.0)
Chloride (mmol/L)
Bicarbonate Levels
Max (29.0)
Min (22.0)
Bicarbonate (mmol/L)
Urea Levels
Max (7.1)
Min (2.5)
Urea (mmol/L)
Creatinine Levels
Max (1.2)
Min (0.6)
Creatinine (mmol/L)
eGFR Levels
Max (120.0)
Min (90.0)
eGFR (mmol/L)
Urate Levels
Max (7.5)
Min (2.5)
Urate (mmol/L)
LD Levels
Max (250.0)
Min (100.0)
LD (mmol/L)
AST Levels
Max (40.0)
Min (10.0)
AST (mmol/L)
ALT Levels
Max (45.0)
Min (10.0)
ALT (mmol/L)
Total Protein Levels
Max (80.0)
Min (60.0)
Total Protein (mmol/L)
Albumin Levels
Max (50.0)
Min (35.0)
Albumin (mmol/L)
Globulin Levels
Max (35.0)
Min (20.0)
Globulin (mmol/L)
Cholesterol Levels
Max (5.2)
Min (3.9)
Cholesterol (mmol/L)
Triglycerides Levels
Max (1.7)
Min (0.5)
Triglycerides (mmol/L)
Gap Levels
Max (12.0)
Min (5.0)
Gap (mmol/L)
PO4 Levels
Max (1.5)
Min (0.8)
PO4 (mmol/L)
Density

In [ ]: