In [10]:
```
pip install pandoc
```

```
Collecting pandoc
  Downloading pandoc-2.4.tar.gz (34 kB)
  Preparing metadata (setup.py): started
  Preparing metadata (setup.py): finished with status 'done'
Collecting plumbum (from pandoc)
  Obtaining dependency information for plumbum from https://files.pythonhosted.org/packages/fa/08/53cf4fb6bebdfd2598e9d
620a587229c3bfcc8df1a202289da07e5b282cd/plumbum-1.8.3-py3-none-any.whl.metadata
  Downloading plumbum-1.8.3-py3-none-any.whl.metadata (10 kB)
Requirement already satisfied: ply in c:\users\saisn\anaconda\lib\site-packages (from pandoc) (3.11)
Requirement already satisfied: pywin32 in c:\users\saisn\anaconda\lib\site-packages (from plumbum->pandoc) (305.1)
Downloading plumbum-1.8.3-py3-none-any.whl (127 kB)
   ---------------------------------------- 0.0/127.6 kB ? eta -:--:--
   ------------------------ -------------- 81.9/127.6 kB 1.6 MB/s eta 0:00:01
   ---------------------------------------- 127.6/127.6 kB 1.5 MB/s eta 0:00:00
Building wheels for collected packages: pandoc
  Building wheel for pandoc (setup.py): started
  Building wheel for pandoc (setup.py): finished with status 'done'
  Created wheel for pandoc: filename=pandoc-2.4-py3-none-any.whl size=34819 sha256=d3e09c98d1988b55477370cf80437ff53464
1f929337ddc368daf1fde995fd0d
  Stored in directory: c:\users\saisn\appdata\local\pip\cache\wheels\4f\d7\32\c6c9b7b05e852e920fd72174487be3a0f18e633a7
adcc303be
Successfully built pandoc
Installing collected packages: plumbum, pandoc
Successfully installed pandoc-2.4 plumbum-1.8.3
Note: you may need to restart the kernel to use updated packages.
```

In [4]:
```
import pandas as pd
import plotly.express as px
import plotly.graph_objects as go
import ipywidgets as widgets
from IPython.display import display
import io
import random
import string
from googletrans import Translator

# Function to generate a password
def generate_password(maxlength):
    return ''.join(random.choices(string.ascii_letters + string.digits + string.punctuation, k=maxlength))

# Generate passwords
upload_password = generate_password(12)
```

```
process_password = generate_password(12)

# Display username and password inputs
username_input = widgets.Text(description='Username:')
upload_password_input = widgets.Password(description='Upload Password:')
process_password_input = widgets.Password(description='Process Password:')

print("\nEnter your username and upload password before proceeding:")
display(username_input)
display(upload_password_input)

# Function to handle file upload and perform analysis
def on_upload_change(change):
    if upload_password_input.value == upload_password:
        print("Upload password verified.")

        if upload.value:
            # Get the uploaded file content
            uploaded_file = upload.value[0]

            # Read the content into a DataFrame
            df = pd.read_csv(io.BytesIO(uploaded_file['content']))

            # Display the DataFrame
            print("Data Preview:")
            display(df.head())

            # Print column names for debugging
            print("Columns in DataFrame:")
            print(df.columns.tolist())

            # Existing graphs
            # 1. Bar Plot: Frequency of Each Drug (only if column exists)
            if 'Name of Drug' in df.columns:
                print("Bar Plot of Frequency of Each Drug:")
                drug_counts = df['Name of Drug'].value_counts().reset_index()
                drug_counts.columns = ['Drug Name', 'Count']
                fig_bar_drug = px.bar(
                    drug_counts,
                    x='Drug Name',
                    y='Count',
                    title='Frequency of Each Drug'
                )
                fig_bar_drug.show()
            else:
```

```
        print("Column 'Name of Drug' not found in dataset.")

    # 2. Histogram: Dosage Distribution (only if column exists)
    if 'Dosage (gram)' in df.columns:
        print("Histogram of Dosage Distribution:")
        fig_hist_dosage = px.histogram(
            df,
            x='Dosage (gram)',
            title='Dosage Distribution'
        )
        fig_hist_dosage.show()
    else:
        print("Column 'Dosage (gram)' not found in dataset.")

    # 3. Pie Chart: Gender Distribution (only if column exists)
    if 'Gender' in df.columns:
        print("Pie Chart of Gender Distribution:")
        gender_distribution = df['Gender'].value_counts().reset_index()
        gender_distribution.columns = ['Gender', 'Count']
        fig_pie_gender = px.pie(
            gender_distribution,
            names='Gender',
            values='Count',
            title='Gender Distribution'
        )
        fig_pie_gender.show()
    else:
        print("Column 'Gender' not found in dataset.")

    # 4. Bar Plot: Count of Each Type of Facility (only if column exists)
    if 'Type  (Hospital / Nursing Home / Lab)' in df.columns:
        print("Bar Plot of Each Type of Facility:")
        type_counts = df['Type  (Hospital / Nursing Home / Lab)'].value_counts().reset_index()
        type_counts.columns = ['Type of Facility', 'Count']
        fig_bar_type = px.bar(
            type_counts,
            x='Type of Facility',
            y='Count',
            title='Count of Each Type of Facility'
        )
        fig_bar_type.show()
    else:
        print("Column 'Type  (Hospital / Nursing Home / Lab)' not found in dataset.")

    # 5. Pie Chart: Distribution by Facility Class (only if column exists)
```

```
if ' Class : (Public / Private)' in df.columns:
    print("Pie Chart of Facility Class Distribution:")
    class_distribution = df[' Class : (Public / Private)'].value_counts().reset_index()
    class_distribution.columns = ['Class', 'Count']
    fig_pie_class = px.pie(
        class_distribution,
        names='Class',
        values='Count',
        title='Facility Class Distribution'
    )
    fig_pie_class.show()
else:
    print("Column ' Class : (Public / Private)' not found in dataset.")

# 6. Bar Plot: Pharmacy Availability (only if column exists)
if 'Pharmacy Available : Yes/No' in df.columns:
    print("Bar Plot of Pharmacy Availability:")
    pharmacy_counts = df['Pharmacy Available : Yes/No'].value_counts().reset_index()
    pharmacy_counts.columns = ['Pharmacy Availability', 'Count']
    fig_bar_pharmacy = px.bar(
        pharmacy_counts,
        x='Pharmacy Availability',
        y='Count',
        title='Pharmacy Availability'
    )
    fig_bar_pharmacy.show()
else:
    print("Column 'Pharmacy Available : Yes/No' not found in dataset.")

# 7. Pie Chart: Ambulance Service Availability (only if column exists)
if 'Ambulance Service Available' in df.columns:
    print("Pie Chart of Ambulance Service Availability:")
    ambulance_distribution = df['Ambulance Service Available'].value_counts().reset_index()
    ambulance_distribution.columns = ['Ambulance Service', 'Count']
    fig_pie_ambulance = px.pie(
        ambulance_distribution,
        names='Ambulance Service',
        values='Count',
        title='Ambulance Service Availability'
    )
    fig_pie_ambulance.show()
else:
    print("Column 'Ambulance Service Available' not found in dataset.")

# 8. Correlation Matrix Heatmap (only if numeric columns exist)
```

```
            numeric_df = df.select_dtypes(include=['number'])
            if not numeric_df.empty:
                print("Correlation Matrix Heatmap:")
                correlation_matrix = numeric_df.corr()
                fig_heatmap = go.Figure(data=go.Heatmap(
                    z=correlation_matrix.values,
                    x=correlation_matrix.columns,
                    y=correlation_matrix.columns,
                    colorscale='Viridis'
                ))
                fig_heatmap.update_layout(title='Correlation Matrix Heatmap')
                fig_heatmap.show()
            else:
                print("No numeric columns found for correlation matrix.")

            # New dataset graphs
            # 1. Bar Plot: Organisation Type
            if 'OrganisationType' in df.columns:
                print("Bar Plot of Organisation Type:")
                org_type_counts = df['OrganisationType'].value_counts().reset_index()
                org_type_counts.columns = ['Organisation Type', 'Count']
                fig_bar_org_type = px.bar(
                    org_type_counts,
                    x='Organisation Type',
                    y='Count',
                    title='Organisation Type Distribution'
                )
                fig_bar_org_type.show()
            else:
                print("Column 'OrganisationType' not found in dataset.")

            # 2. Bar Plot: Sector
            if 'Sector' in df.columns:
                print("Bar Plot of Sector Distribution:")
                sector_counts = df['Sector'].value_counts().reset_index()
                sector_counts.columns = ['Sector', 'Count']
                fig_bar_sector = px.bar(
                    sector_counts,
                    x='Sector',
                    y='Count',
                    title='Sector Distribution'
                )
                fig_bar_sector.show()
            else:
                print("Column 'Sector' not found in dataset.")
```

```
        # 3. Pie Chart: Organisation Status
        if 'OrganisationStatus' in df.columns:
            print("Pie Chart of Organisation Status Distribution:")
            status_distribution = df['OrganisationStatus'].value_counts().reset_index()
            status_distribution.columns = ['Organisation Status', 'Count']
            fig_pie_status = px.pie(
                status_distribution,
                names='Organisation Status',
                values='Count',
                title='Organisation Status Distribution'
            )
            fig_pie_status.show()
        else:
            print("Column 'OrganisationStatus' not found in dataset.")

        # Display process password input for translation
        print("\nEnter process password for translation:")
        display(process_password_input)

        # Translation text inputs using ipywidgets
        text_input = widgets.Text(description='Text to Translate:')
        language_input = widgets.Text(description='Destination Language:')
        translate_button = widgets.Button(description='Translate')

        def on_translate_button_clicked(b):
            if process_password_input.value == process_password:
                print("Process password verified.")
                text_to_translate = text_input.value
                dest_language = language_input.value

                translate_text(text_to_translate, dest_language)
            else:
                print("Incorrect process password. Translation canceled.")

        translate_button.on_click(on_translate_button_clicked)
        display(text_input)
        display(language_input)
        display(translate_button)

    else:
        print("Upload password verified. Please upload a file.")
else:
    print("Incorrect upload password. Upload process canceled.")
```

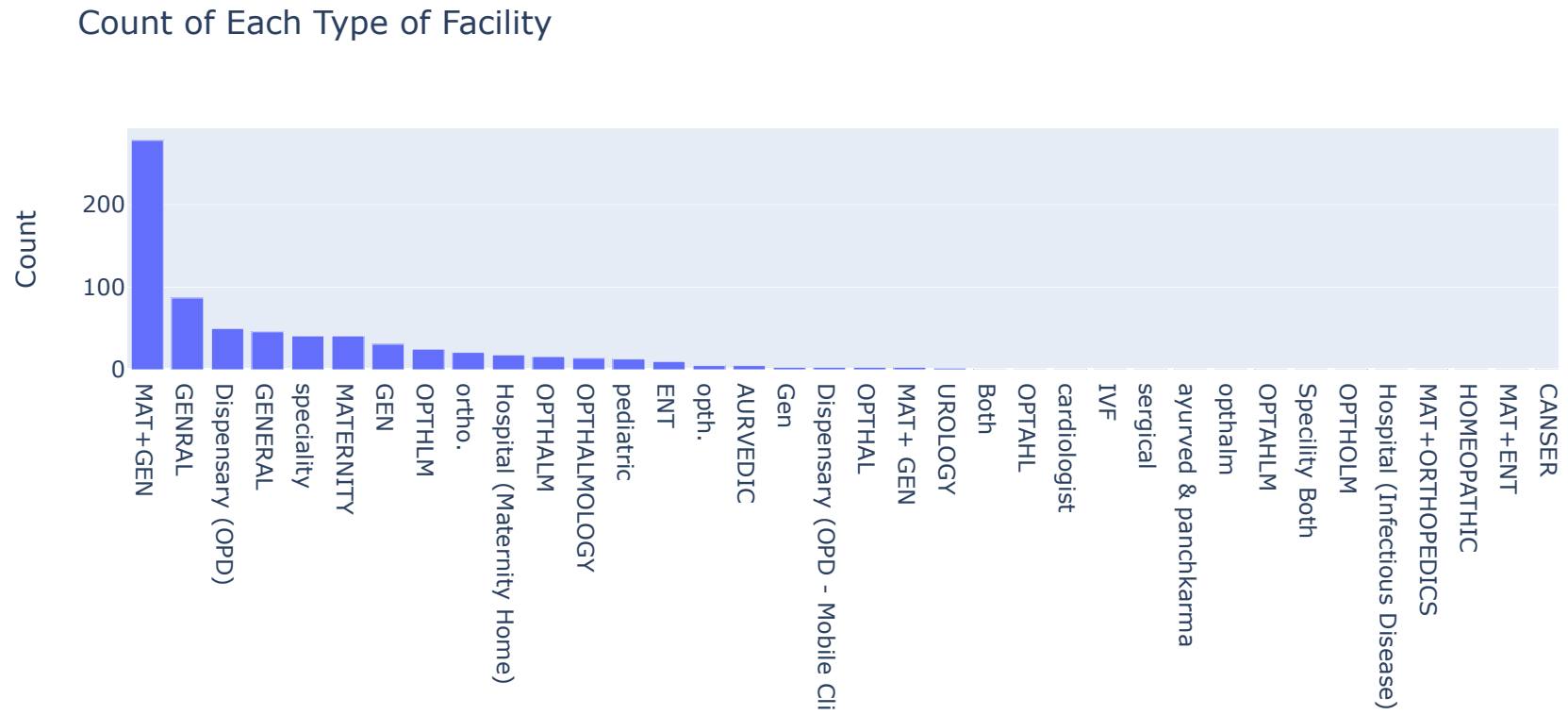
```
# Create a file upload widget
upload = widgets.FileUpload(accept='.csv', multiple=False)
upload.observe(on_upload_change, names='value')
display(upload)

# Language dictionary with language codes and names
language = {
    "bn": "Bangla",
    "en": "English",
    "ko": "Korean",
    "fr": "French",
    "de": "German",
    "he": "Hebrew",
    "hi": "Hindi",
    "it": "Italian",
    "ja": "Japanese",
    'la': "Latin",
    "ms": "Malay",
    "ne": "Nepali",
    "ru": "Russian",
    "ar": "Arabic",
    "zh": "Chinese",
    "es": "Spanish"
}

# Translation function
translator = Translator()

def translate_text(text, dest_language):
    try:
        translated = translator.translate(text, dest=dest_language)
        print(f"\n{language.get(dest_language, 'Unknown')} translation: {translated.text}")

        if translated.pronunciation:
            print(f"Pronunciation: {translated.pronunciation}")

        print(f"Translated from: {language.get(translated.src, 'Unknown')}")

    except Exception as e:
        print(f"Translation error: {str(e)}")

# Function to generate passwords for specific usernames
def generate_user_passwords(change):
    if username_input.value in ['sai', 'thilak', 'sabri']:
        global upload_password, process_password
```

```
        upload_password = generate_password(12)
        process_password = generate_password(12)
        print(f"Generated Upload Password for {username_input.value}: {upload_password}")
        print(f"Generated Process Password for {username_input.value}: {process_password}")

username_input.observe(generate_user_passwords, names='value')
```

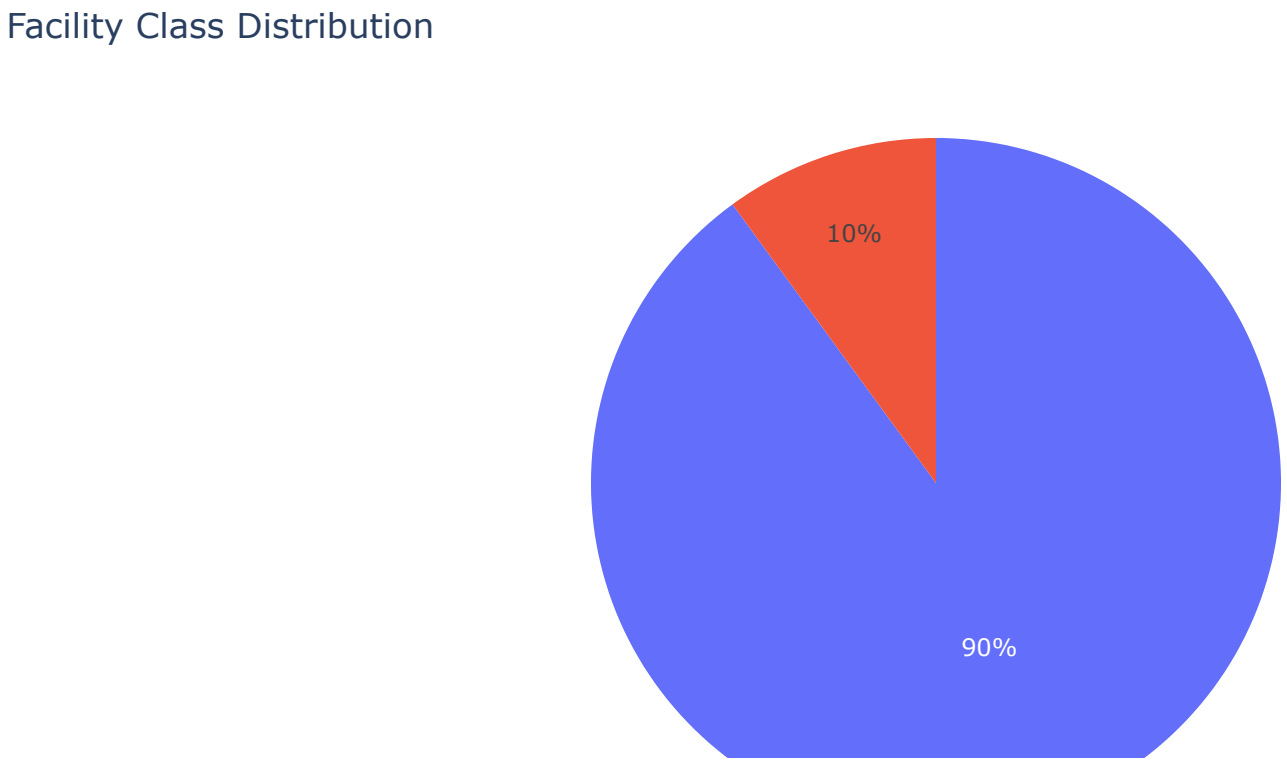
```
Enter your username and upload password before proceeding:
Text(value='', description='Username:')
Password(description='Upload Password:')
FileUpload(value=(), accept='.csv', description='Upload')
Generated Upload Password for sai: $4ZB&<|EAr]z
Generated Process Password for sai: Kmhw<|pM8GG)
Upload password verified.
Data Preview:
```

	City Name	Zone Name	Ward Name	Zone No.	Ward No.	Facility Name	Type (Hospital / Nursing Home / Lab)	Class : (Public / Private)	Pharmacy Available : Yes/No	Number of Beds in Emergency Wards	Number of Beds in facility type	Number of Doctors / Physicians	Number of Nurses	N Pr
0	Pune	kasba - vishrambagwada WO	Ambil Odha	5.0	NaN	Late Matoshri Ramabai Ambedkar Maternity Home,...	Hospital (Maternity Home)	Public	Yes	0	10	2	4	
1	Pune	Aundh - Baner WO	Aundh	2.0	NaN	Aundh Kuti Maternity Home, Aundh	Hospital (Maternity Home)	Public	Yes	0	14	2	5	
2	Pune	Aundh - Baner WO	Bopodi	2.0	NaN	Late Draupadabai Murlidhar Khedekar Maternity ...	Hospital (Maternity Home)	Public	Yes	0	16	2	4	
3	Pune	Ghole Road - Shivaji Nagar WO	Shivaji Nagar Station	2.0	NaN	Dr. Dalvi, PMC Joint Project	Hospital (Maternity Home)	Public	Yes	0	30	4	7	
4	Pune	Hadapsar - Mundhwa WO	Hadapsar	4.0	NaN	Late Anna Saheb Magar Maternity Home, Hadapsar	Hospital (Maternity Home)	Public	Yes	0	20	2	3	

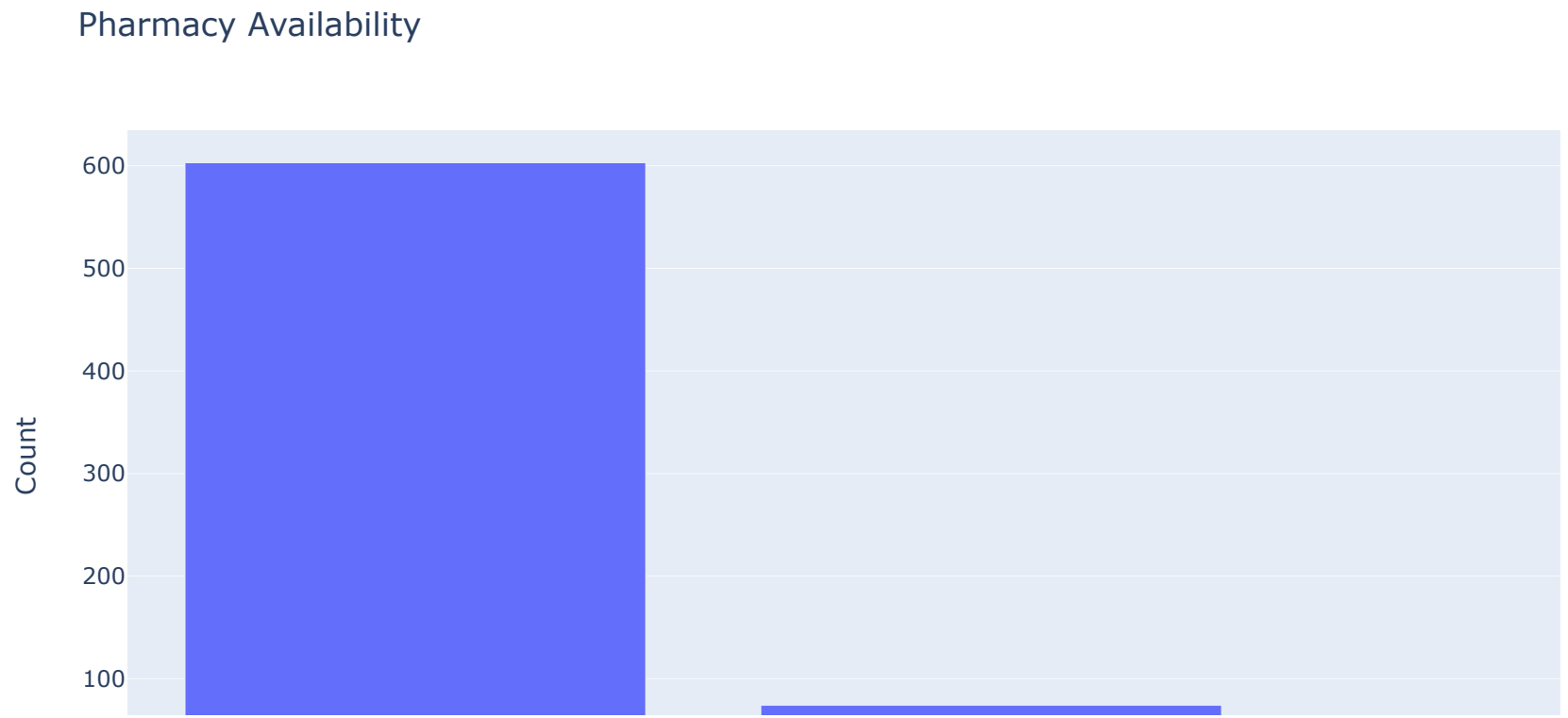
```
Columns in DataFrame:
['City Name', 'Zone Name', 'Ward Name', 'Zone No.', 'Ward No.', 'Facility Name', 'Type  (Hospital / Nursing Home / La
b)', ' Class : (Public / Private)', 'Pharmacy Available : Yes/No', 'Number of Beds in Emergency Wards ', 'Number of Bed
s in facility type', 'Number of Doctors / Physicians', 'Number of Nurses', 'Number of Midwives Professional ', 'Average
Monthly Patient Footfall', 'Ambulance Service Available', 'Count of Ambulance']
Column 'Name of Drug' not found in dataset.
Column 'Dosage (gram)' not found in dataset.
Column 'Gender' not found in dataset.
Bar Plot of Each Type of Facility:
```



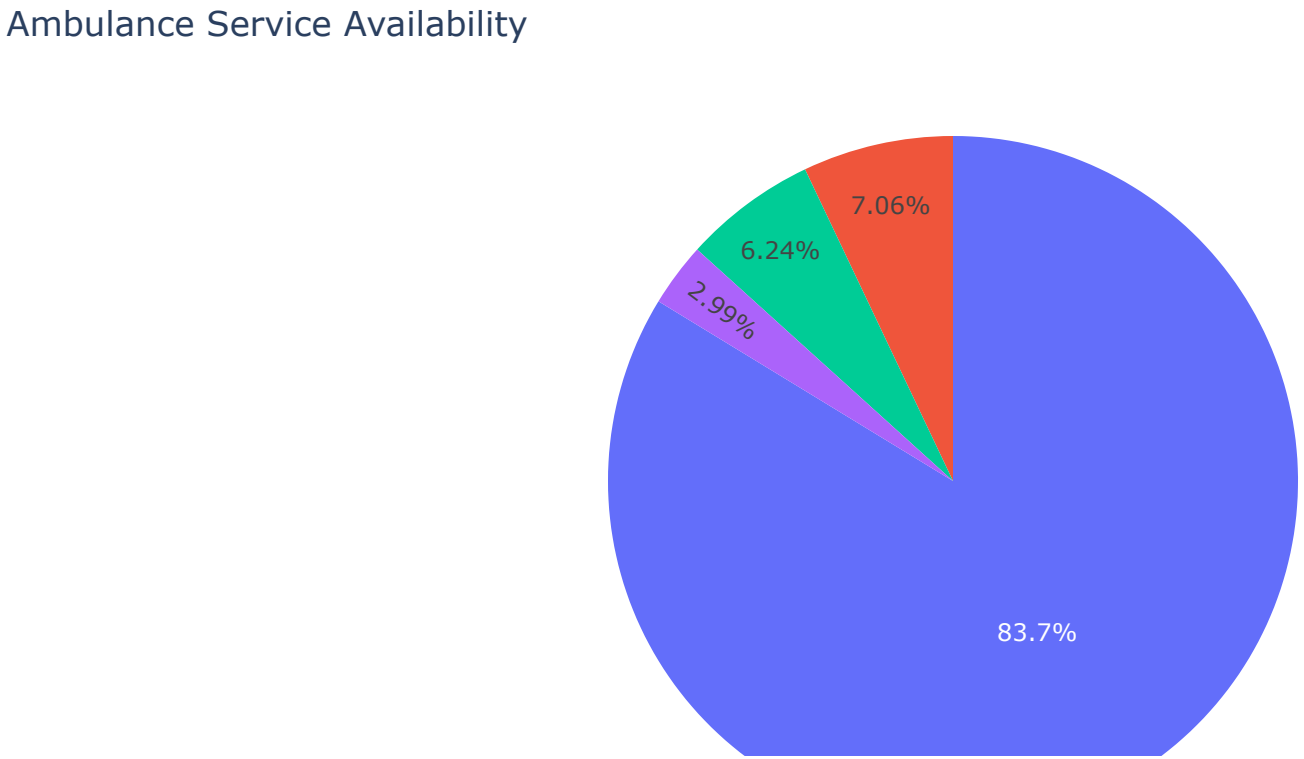
Pie Chart of Facility Class Distribution:



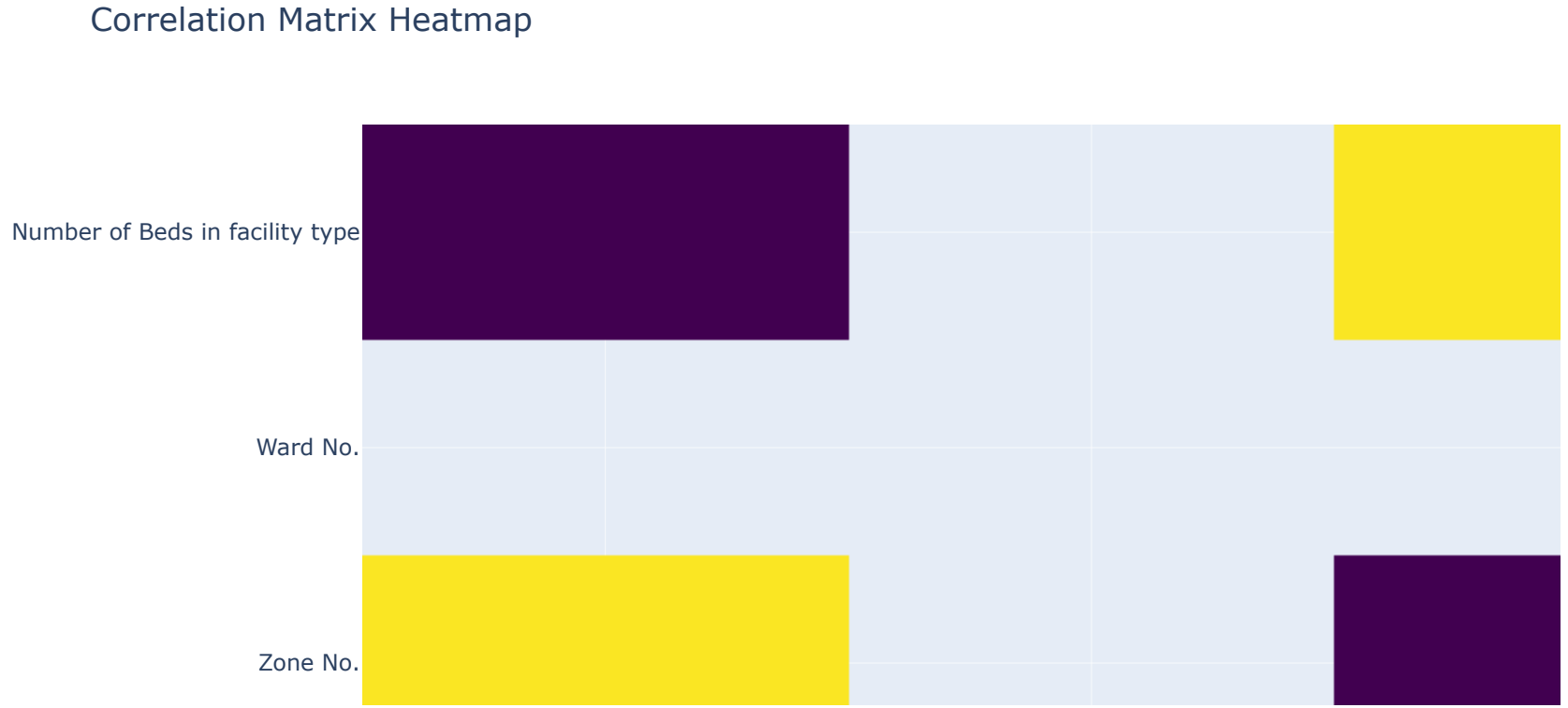
Bar Plot of Pharmacy Availability:



Pie Chart of Ambulance Service Availability:



Correlation Matrix Heatmap:




```
Column 'OrganisationType' not found in dataset.
Column 'Sector' not found in dataset.
Column 'OrganisationStatus' not found in dataset.

Enter process password for translation:
Password(description='Process Password:')
Text(value='', description='Text to Translate:')
Text(value='', description='Destination Language:')
Button(description='Translate', style=ButtonStyle())
```

```
Process password verified.

French translation: Il y avait 7,8% du lit à l'hôpital
```

In [9]:

```python
import pandas as pd
import plotly.express as px
import plotly.graph_objects as go
import ipywidgets as widgets
from IPython.display import display
import io
import random
import string
from googletrans import Translator

# Function to generate a password
def generate_password(maxlength):
    return ''.join(random.choices(string.ascii_letters + string.digits + string.punctuation, k=maxlength))

# Generate passwords
upload_password = generate_password(12)
process_password = generate_password(12)

# Display username and password inputs
username_input = widgets.Text(description='Username:')
upload_password_input = widgets.Password(description='Upload Password:')
process_password_input = widgets.Password(description='Process Password:')

print("\nEnter your username and upload password before proceeding:")
display(username_input)
display(upload_password_input)

# Function to handle file upload and perform analysis
def on_upload_change(change):
    if upload_password_input.value == upload_password:
        print("Upload password verified.")

        if upload.value:
            # Get the uploaded file content
            uploaded_file = upload.value[0]

            # Read the content into a DataFrame
            df = pd.read_csv(io.BytesIO(uploaded_file['content']))

            # Display the DataFrame
```

```
print("Data Preview:")
display(df.head())

# Print column names for debugging
print("Columns in DataFrame:")
print(df.columns.tolist())

# Existing graphs
# 1. Bar Plot: Frequency of Each Drug (only if column exists)
if 'Name of Drug' in df.columns:
    print("Bar Plot of Frequency of Each Drug:")
    drug_counts = df['Name of Drug'].value_counts().reset_index()
    drug_counts.columns = ['Drug Name', 'Count']
    fig_bar_drug = px.bar(
        drug_counts,
        x='Drug Name',
        y='Count',
        title='Frequency of Each Drug'
    )
    fig_bar_drug.show()
else:
    print("Column 'Name of Drug' not found in dataset.")

# 2. Histogram: Dosage Distribution (only if column exists)
if 'Dosage (gram)' in df.columns:
    print("Histogram of Dosage Distribution:")
    fig_hist_dosage = px.histogram(
        df,
        x='Dosage (gram)',
        title='Dosage Distribution'
    )
    fig_hist_dosage.show()
else:
    print("Column 'Dosage (gram)' not found in dataset.")

# 3. Pie Chart: Gender Distribution (only if column exists)
if 'Gender' in df.columns:
    print("Pie Chart of Gender Distribution:")
    gender_distribution = df['Gender'].value_counts().reset_index()
    gender_distribution.columns = ['Gender', 'Count']
    fig_pie_gender = px.pie(
        gender_distribution,
        names='Gender',
        values='Count',
        title='Gender Distribution'
```

```
        )
        fig_pie_gender.show()
    else:
        print("Column 'Gender' not found in dataset.")

    # 4. Bar Plot: Count of Each Type of Facility (only if column exists)
    if 'Type  (Hospital / Nursing Home / Lab)' in df.columns:
        print("Bar Plot of Each Type of Facility:")
        type_counts = df['Type  (Hospital / Nursing Home / Lab)'].value_counts().reset_index()
        type_counts.columns = ['Type of Facility', 'Count']
        fig_bar_type = px.bar(
            type_counts,
            x='Type of Facility',
            y='Count',
            title='Count of Each Type of Facility'
        )
        fig_bar_type.show()
    else:
        print("Column 'Type  (Hospital / Nursing Home / Lab)' not found in dataset.")

    # 5. Pie Chart: Distribution by Facility Class (only if column exists)
    if ' Class : (Public / Private)' in df.columns:
        print("Pie Chart of Facility Class Distribution:")
        class_distribution = df[' Class : (Public / Private)'].value_counts().reset_index()
        class_distribution.columns = ['Class', 'Count']
        fig_pie_class = px.pie(
            class_distribution,
            names='Class',
            values='Count',
            title='Facility Class Distribution'
        )
        fig_pie_class.show()
    else:
        print("Column ' Class : (Public / Private)' not found in dataset.")

    # 6. Bar Plot: Pharmacy Availability (only if column exists)
    if 'Pharmacy Available : Yes/No' in df.columns:
        print("Bar Plot of Pharmacy Availability:")
        pharmacy_counts = df['Pharmacy Available : Yes/No'].value_counts().reset_index()
        pharmacy_counts.columns = ['Pharmacy Availability', 'Count']
        fig_bar_pharmacy = px.bar(
            pharmacy_counts,
            x='Pharmacy Availability',
            y='Count',
            title='Pharmacy Availability'
```

```
        )
        fig_bar_pharmacy.show()
    else:
        print("Column 'Pharmacy Available : Yes/No' not found in dataset.")

    # 7. Pie Chart: Ambulance Service Availability (only if column exists)
    if 'Ambulance Service Available' in df.columns:
        print("Pie Chart of Ambulance Service Availability:")
        ambulance_distribution = df['Ambulance Service Available'].value_counts().reset_index()
        ambulance_distribution.columns = ['Ambulance Service', 'Count']
        fig_pie_ambulance = px.pie(
            ambulance_distribution,
            names='Ambulance Service',
            values='Count',
            title='Ambulance Service Availability'
        )
        fig_pie_ambulance.show()
    else:
        print("Column 'Ambulance Service Available' not found in dataset.")

    # 8. Correlation Matrix Heatmap (only if numeric columns exist)
    numeric_df = df.select_dtypes(include=['number'])
    if not numeric_df.empty:
        print("Correlation Matrix Heatmap:")
        correlation_matrix = numeric_df.corr()
        fig_heatmap = go.Figure(data=go.Heatmap(
            z=correlation_matrix.values,
            x=correlation_matrix.columns,
            y=correlation_matrix.columns,
            colorscale='Viridis'
        ))
        fig_heatmap.update_layout(title='Correlation Matrix Heatmap')
        fig_heatmap.show()
    else:
        print("No numeric columns found for correlation matrix.")

    # New dataset graphs
    # 1. Bar Plot: Organisation Type
    if 'OrganisationType' in df.columns:
        print("Bar Plot of Organisation Type:")
        org_type_counts = df['OrganisationType'].value_counts().reset_index()
        org_type_counts.columns = ['Organisation Type', 'Count']
        fig_bar_org_type = px.bar(
            org_type_counts,
            x='Organisation Type',
```

```
            y='Count',
            title='Organisation Type Distribution'
        )
        fig_bar_org_type.show()
    else:
        print("Column 'OrganisationType' not found in dataset.")

    # 2. Bar Plot: Sector
    if 'Sector' in df.columns:
        print("Bar Plot of Sector Distribution:")
        sector_counts = df['Sector'].value_counts().reset_index()
        sector_counts.columns = ['Sector', 'Count']
        fig_bar_sector = px.bar(
            sector_counts,
            x='Sector',
            y='Count',
            title='Sector Distribution'
        )
        fig_bar_sector.show()
    else:
        print("Column 'Sector' not found in dataset.")

    # 3. Pie Chart: Organisation Status
    if 'OrganisationStatus' in df.columns:
        print("Pie Chart of Organisation Status Distribution:")
        status_distribution = df['OrganisationStatus'].value_counts().reset_index()
        status_distribution.columns = ['Organisation Status', 'Count']
        fig_pie_status = px.pie(
            status_distribution,
            names='Organisation Status',
            values='Count',
            title='Organisation Status Distribution'
        )
        fig_pie_status.show()
    else:
        print("Column 'OrganisationStatus' not found in dataset.")

    # Display process password input for translation
    print("\nEnter process password for translation:")
    display(process_password_input)

    # Translation text inputs using ipywidgets
    text_input = widgets.Text(description='Text to Translate:')
    language_input = widgets.Text(description='Destination Language:')
    translate_button = widgets.Button(description='Translate')
```

```
            def on_translate_button_clicked(b):
                if process_password_input.value == process_password:
                    print("Process password verified.")
                    text_to_translate = text_input.value
                    dest_language = language_input.value

                    translate_text(text_to_translate, dest_language)
                else:
                    print("Incorrect process password. Translation canceled.")

            translate_button.on_click(on_translate_button_clicked)
            display(text_input)
            display(language_input)
            display(translate_button)

        else:
            print("Upload password verified. Please upload a file.")
    else:
        print("Incorrect upload password. Upload process canceled.")

# Create a file upload widget
upload = widgets.FileUpload(accept='.csv', multiple=False)
upload.observe(on_upload_change, names='value')
display(upload)

# Language dictionary with language codes and names
language = {
    "bn": "Bangla",
    "en": "English",
    "ko": "Korean",
    "fr": "French",
    "de": "German",
    "he": "Hebrew",
    "hi": "Hindi",
    "it": "Italian",
    "ja": "Japanese",
    'la': "Latin",
    "ms": "Malay",
    "ne": "Nepali",
    "ru": "Russian",
    "ar": "Arabic",
    "zh": "Chinese",
    "es": "Spanish"
}
```

```
# Translation function
translator = Translator()

def translate_text(text, dest_language):
    try:
        translated = translator.translate(text, dest=dest_language)
        print(f"\n{language.get(dest_language, 'Unknown')} translation: {translated.text}")

        if translated.pronunciation:
            print(f"Pronunciation: {translated.pronunciation}")

        print(f"Translated from: {language.get(translated.src, 'Unknown')}")

    except Exception as e:
        print(f"Translation error: {str(e)}")

# Function to generate passwords for specific usernames
def generate_user_passwords(change):
    if username_input.value in ['sai', 'thilak', 'sabri']:
        global upload_password, process_password
        upload_password = generate_password(12)
        process_password = generate_password(12)
        print(f"Generated Upload Password for {username_input.value}: {upload_password}")
        print(f"Generated Process Password for {username_input.value}: {process_password}")

username_input.observe(generate_user_passwords, names='value')
```

```
Enter your username and upload password before proceeding:
Text(value='', description='Username:')
Password(description='Upload Password:')
FileUpload(value=(), accept='.csv', description='Upload')
Generated Upload Password for thilak: F[PB2z[q)pm'
Generated Process Password for thilak: }qN^#5J8AH4f
Upload password verified.
Data Preview:
```

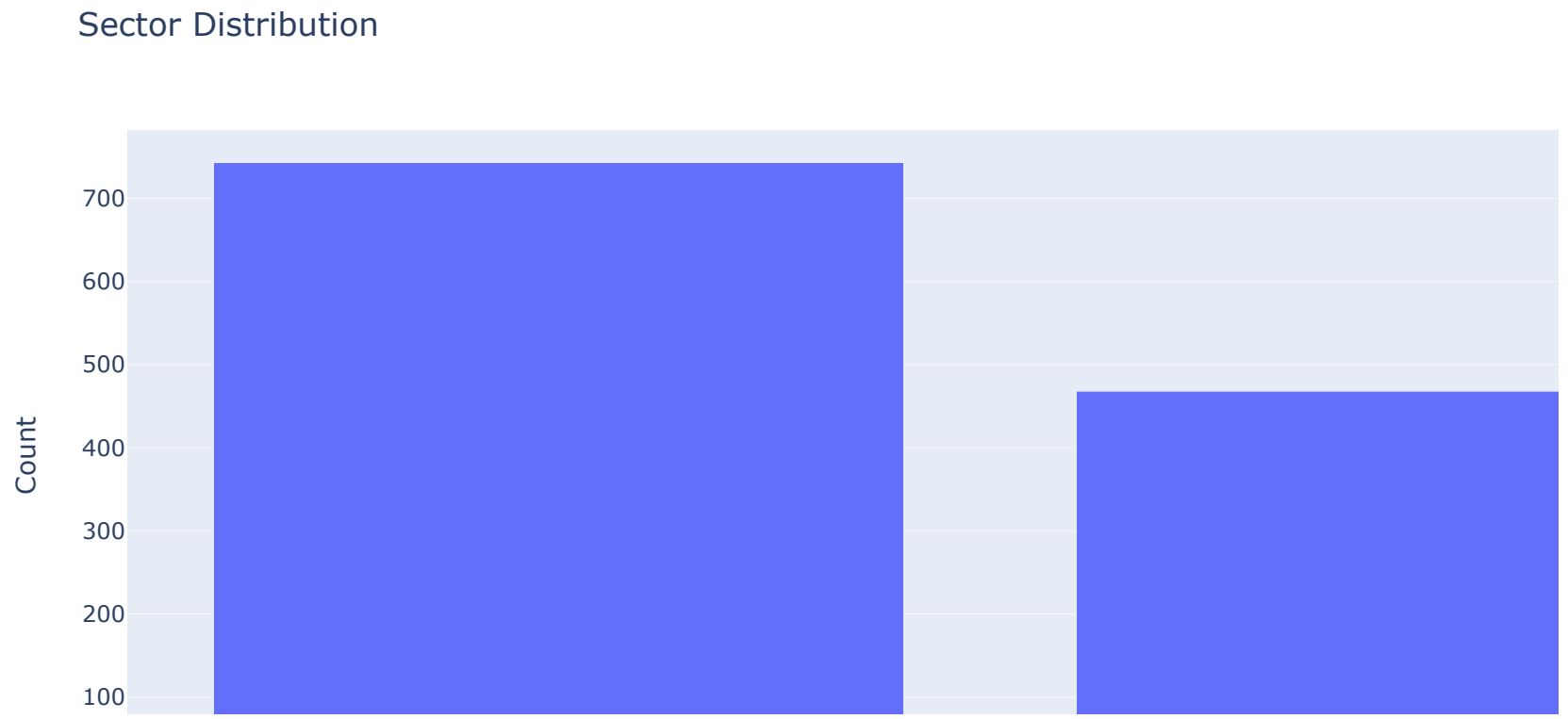
	OrganisationID	OrganisationCode	OrganisationType	SubType	Sector	OrganisationStatus	IsPimsManaged	OrganisationName	Add
0	17970	NDA07	Hospital	Hospital	Independent Sector	Visible	True	Walton Community Hospital - Virgin Care Servic...	
1	17981	NDA18	Hospital	Hospital	Independent Sector	Visible	True	Woking Community Hospital (Virgin Care)	
2	18102	NLT02	Hospital	Hospital	NHS Sector	Visible	True	North Somerset Community Hospital	I Som Comm Ho
3	18138	NMP01	Hospital	Hospital	Independent Sector	Visible	False	Bridgewater Hospital	Pri
4	18142	NMV01	Hospital	Hospital	Independent Sector	Visible	True	Kneesworth House	Old I

5 rows × 22 columns

```
Columns in DataFrame:
['OrganisationID', 'OrganisationCode', 'OrganisationType', 'SubType', 'Sector', 'OrganisationStatus', 'IsPimsManaged',
'OrganisationName', 'Address1', 'Address2', 'Address3', 'City', 'County', 'Postcode', 'Latitude', 'Longitude', 'ParentO
DSCode', 'ParentName', 'Phone', 'Email', 'Website', 'Fax,,,']
Column 'Name of Drug' not found in dataset.
Column 'Dosage (gram)' not found in dataset.
Column 'Gender' not found in dataset.
Column 'Type  (Hospital / Nursing Home / Lab)' not found in dataset.
Column ' Class : (Public / Private)' not found in dataset.
Column 'Pharmacy Available : Yes/No' not found in dataset.
Column 'Ambulance Service Available' not found in dataset.
Correlation Matrix Heatmap:
```

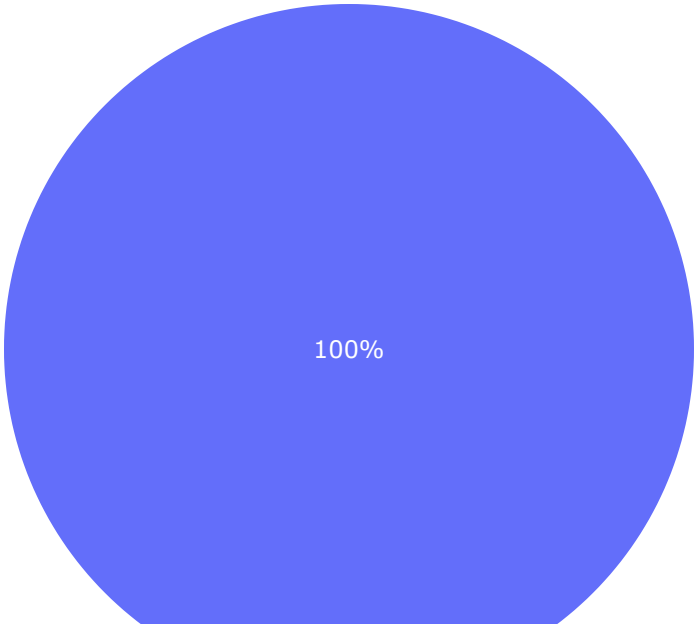
Bar Plot of Organisation Type:

Bar Plot of Sector Distribution:



Pie Chart of Organisation Status Distribution:

Organisation Status Distribution




```
Enter process password for translation:
Password(description='Process Password:')
Text(value='', description='Text to Translate:')
Text(value='', description='Destination Language:')
Button(description='Translate', style=ButtonStyle())
Process password verified.

Russian translation: Статус организации - счет 1211
Pronunciation: Status organizatsii - schet 1211
```

In [6]:
```
import pandas as pd
import plotly.express as px
import plotly.graph_objects as go
import ipywidgets as widgets
from IPython.display import display
import io
import random
import string
from googletrans import Translator

# Function to generate a password
def generate_password(maxlength):
    return ''.join(random.choices(string.ascii_letters + string.digits + string.punctuation, k=maxlength))

# Generate passwords
upload_password = generate_password(12)
process_password = generate_password(12)

# Display username and password inputs
username_input = widgets.Text(description='Username:')
upload_password_input = widgets.Password(description='Upload Password:')
process_password_input = widgets.Password(description='Process Password:')

print("\nEnter your username and upload password before proceeding:")
display(username_input)
display(upload_password_input)

# Function to handle file upload and perform analysis
def on_upload_change(change):
    if upload_password_input.value == upload_password:
        print("Upload password verified.")

        if upload.value:
            # Get the uploaded file content
            uploaded_file = upload.value[0]

            # Read the content into a DataFrame
            df = pd.read_csv(io.BytesIO(uploaded_file['content']))

            # Display the DataFrame
            print("Data Preview:")
            display(df.head())

            # Print column names for debugging
```

```
print("Columns in DataFrame:")
print(df.columns.tolist())

# Existing graphs
# 1. Bar Plot: Frequency of Each Drug (only if column exists)
if 'Name of Drug' in df.columns:
    print("Bar Plot of Frequency of Each Drug:")
    drug_counts = df['Name of Drug'].value_counts().reset_index()
    drug_counts.columns = ['Drug Name', 'Count']
    fig_bar_drug = px.bar(
        drug_counts,
        x='Drug Name',
        y='Count',
        title='Frequency of Each Drug'
    )
    fig_bar_drug.show()
else:
    print("Column 'Name of Drug' not found in dataset.")

# 2. Histogram: Dosage Distribution (only if column exists)
if 'Dosage (gram)' in df.columns:
    print("Histogram of Dosage Distribution:")
    fig_hist_dosage = px.histogram(
        df,
        x='Dosage (gram)',
        title='Dosage Distribution'
    )
    fig_hist_dosage.show()
else:
    print("Column 'Dosage (gram)' not found in dataset.")

# 3. Pie Chart: Gender Distribution (only if column exists)
if 'Gender' in df.columns:
    print("Pie Chart of Gender Distribution:")
    gender_distribution = df['Gender'].value_counts().reset_index()
    gender_distribution.columns = ['Gender', 'Count']
    fig_pie_gender = px.pie(
        gender_distribution,
        names='Gender',
        values='Count',
        title='Gender Distribution'
    )
    fig_pie_gender.show()
else:
    print("Column 'Gender' not found in dataset.")
```

```
# 4. Bar Plot: Count of Each Type of Facility (only if column exists)
if 'Type  (Hospital / Nursing Home / Lab)' in df.columns:
    print("Bar Plot of Each Type of Facility:")
    type_counts = df['Type  (Hospital / Nursing Home / Lab)'].value_counts().reset_index()
    type_counts.columns = ['Type of Facility', 'Count']
    fig_bar_type = px.bar(
        type_counts,
        x='Type of Facility',
        y='Count',
        title='Count of Each Type of Facility'
    )
    fig_bar_type.show()
else:
    print("Column 'Type  (Hospital / Nursing Home / Lab)' not found in dataset.")

# 5. Pie Chart: Distribution by Facility Class (only if column exists)
if ' Class : (Public / Private)' in df.columns:
    print("Pie Chart of Facility Class Distribution:")
    class_distribution = df[' Class : (Public / Private)'].value_counts().reset_index()
    class_distribution.columns = ['Class', 'Count']
    fig_pie_class = px.pie(
        class_distribution,
        names='Class',
        values='Count',
        title='Facility Class Distribution'
    )
    fig_pie_class.show()
else:
    print("Column ' Class : (Public / Private)' not found in dataset.")

# 6. Bar Plot: Pharmacy Availability (only if column exists)
if 'Pharmacy Available : Yes/No' in df.columns:
    print("Bar Plot of Pharmacy Availability:")
    pharmacy_counts = df['Pharmacy Available : Yes/No'].value_counts().reset_index()
    pharmacy_counts.columns = ['Pharmacy Availability', 'Count']
    fig_bar_pharmacy = px.bar(
        pharmacy_counts,
        x='Pharmacy Availability',
        y='Count',
        title='Pharmacy Availability'
    )
    fig_bar_pharmacy.show()
else:
    print("Column 'Pharmacy Available : Yes/No' not found in dataset.")
```

```
# 7. Pie Chart: Ambulance Service Availability (only if column exists)
if 'Ambulance Service Available' in df.columns:
    print("Pie Chart of Ambulance Service Availability:")
    ambulance_distribution = df['Ambulance Service Available'].value_counts().reset_index()
    ambulance_distribution.columns = ['Ambulance Service', 'Count']
    fig_pie_ambulance = px.pie(
        ambulance_distribution,
        names='Ambulance Service',
        values='Count',
        title='Ambulance Service Availability'
    )
    fig_pie_ambulance.show()
else:
    print("Column 'Ambulance Service Available' not found in dataset.")

# 8. Correlation Matrix Heatmap (only if numeric columns exist)
numeric_df = df.select_dtypes(include=['number'])
if not numeric_df.empty:
    print("Correlation Matrix Heatmap:")
    correlation_matrix = numeric_df.corr()
    fig_heatmap = go.Figure(data=go.Heatmap(
        z=correlation_matrix.values,
        x=correlation_matrix.columns,
        y=correlation_matrix.columns,
        colorscale='Viridis'
    ))
    fig_heatmap.update_layout(title='Correlation Matrix Heatmap')
    fig_heatmap.show()
else:
    print("No numeric columns found for correlation matrix.")

# New dataset graphs
# 1. Bar Plot: Organisation Type
if 'OrganisationType' in df.columns:
    print("Bar Plot of Organisation Type:")
    org_type_counts = df['OrganisationType'].value_counts().reset_index()
    org_type_counts.columns = ['Organisation Type', 'Count']
    fig_bar_org_type = px.bar(
        org_type_counts,
        x='Organisation Type',
        y='Count',
        title='Organisation Type Distribution'
    )
    fig_bar_org_type.show()
```

```
        else:
            print("Column 'OrganisationType' not found in dataset.")

        # 2. Bar Plot: Sector
        if 'Sector' in df.columns:
            print("Bar Plot of Sector Distribution:")
            sector_counts = df['Sector'].value_counts().reset_index()
            sector_counts.columns = ['Sector', 'Count']
            fig_bar_sector = px.bar(
                sector_counts,
                x='Sector',
                y='Count',
                title='Sector Distribution'
            )
            fig_bar_sector.show()
        else:
            print("Column 'Sector' not found in dataset.")

        # 3. Pie Chart: Organisation Status
        if 'OrganisationStatus' in df.columns:
            print("Pie Chart of Organisation Status Distribution:")
            status_distribution = df['OrganisationStatus'].value_counts().reset_index()
            status_distribution.columns = ['Organisation Status', 'Count']
            fig_pie_status = px.pie(
                status_distribution,
                names='Organisation Status',
                values='Count',
                title='Organisation Status Distribution'
            )
            fig_pie_status.show()
        else:
            print("Column 'OrganisationStatus' not found in dataset.")

        # Display process password input for translation
        print("\nEnter process password for translation:")
        display(process_password_input)

        # Translation text inputs using ipywidgets
        text_input = widgets.Text(description='Text to Translate:')
        language_input = widgets.Text(description='Destination Language:')
        translate_button = widgets.Button(description='Translate')

        def on_translate_button_clicked(b):
            if process_password_input.value == process_password:
                print("Process password verified.")
```

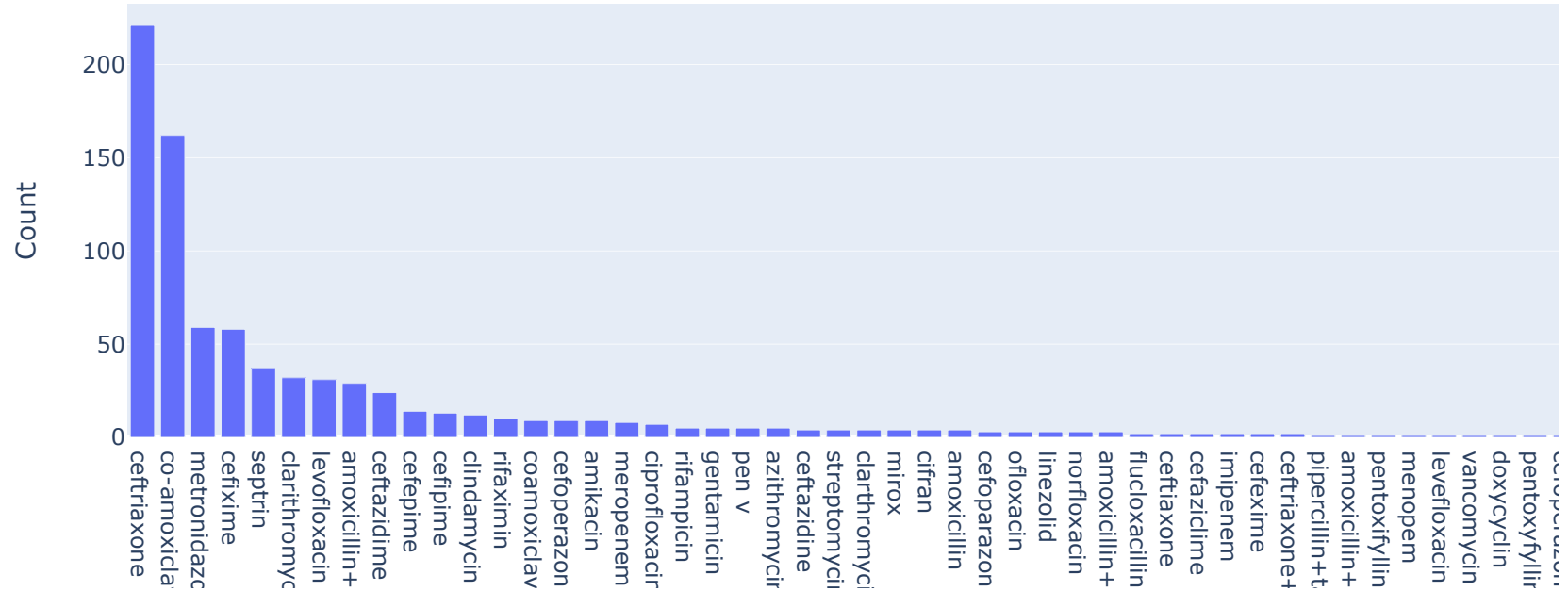
```
                    text_to_translate = text_input.value
                    dest_language = language_input.value

                    translate_text(text_to_translate, dest_language)
                else:
                    print("Incorrect process password. Translation canceled.")

            translate_button.on_click(on_translate_button_clicked)
            display(text_input)
            display(language_input)
            display(translate_button)

        else:
            print("Upload password verified. Please upload a file.")
    else:
        print("Incorrect upload password. Upload process canceled.")

# Create a file upload widget
upload = widgets.FileUpload(accept='.csv', multiple=False)
upload.observe(on_upload_change, names='value')
display(upload)

# Language dictionary with language codes and names
language = {
    "bn": "Bangla",
    "en": "English",
    "ko": "Korean",
    "fr": "French",
    "de": "German",
    "he": "Hebrew",
    "hi": "Hindi",
    "it": "Italian",
    "ja": "Japanese",
    'la': "Latin",
    "ms": "Malay",
    "ne": "Nepali",
    "ru": "Russian",
    "ar": "Arabic",
    "zh": "Chinese",
    "es": "Spanish"
}

# Translation function
translator = Translator()
```

```
def translate_text(text, dest_language):
    try:
        translated = translator.translate(text, dest=dest_language)
        print(f"\n{language.get(dest_language, 'Unknown')} translation: {translated.text}")

        if translated.pronunciation:
            print(f"Pronunciation: {translated.pronunciation}")

        print(f"Translated from: {language.get(translated.src, 'Unknown')}")

    except Exception as e:
        print(f"Translation error: {str(e)}")

# Function to generate passwords for specific usernames
def generate_user_passwords(change):
    if username_input.value in ['sai', 'thilak', 'sabri']:
        global upload_password, process_password
        upload_password = generate_password(12)
        process_password = generate_password(12)
        print(f"Generated Upload Password for {username_input.value}: {upload_password}")
        print(f"Generated Process Password for {username_input.value}: {process_password}")

username_input.observe(generate_user_passwords, names='value')
```

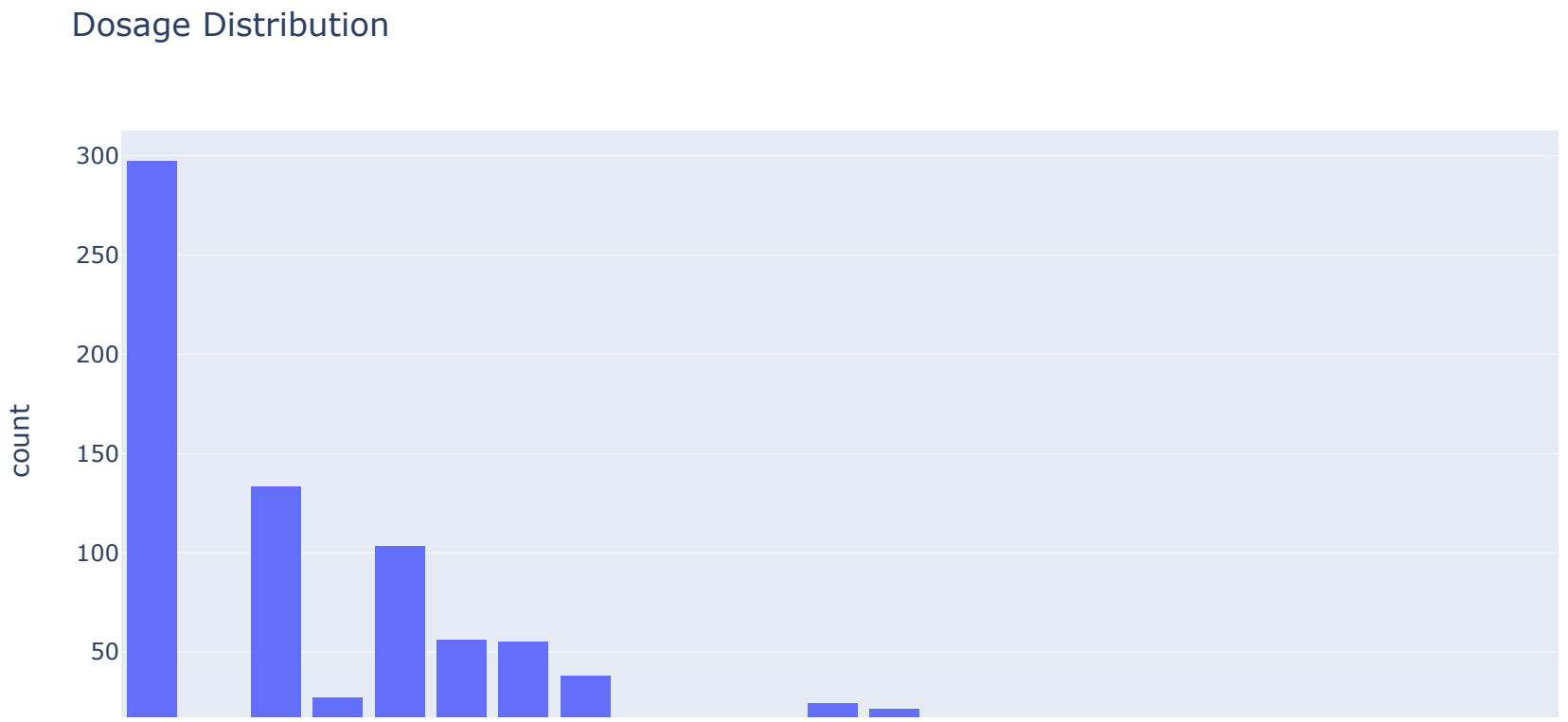
```
Enter your username and upload password before proceeding:
Text(value='', description='Username:')
Password(description='Upload Password:')
FileUpload(value=(), accept='.csv', description='Upload')
Generated Upload Password for sabri: zz<8hd=AbdAq
Generated Process Password for sabri: SyO7k4|}GIqN
Upload password verified.
Data Preview:
```

	Age	Date of Data Entry	Gender	Diagnosis	Name of Drug	Dosage (gram)	Route	Frequency	Duration (days)	Indication
0	85	19/12/2019 14:41:49	Female	ccf, hypertension, ida, ckd(stage 5), ?icm,	ceftriaxone	1	IV	BD	7	icm
1	87	19/12/2019 16:35:25	Female	pad(lt u.l), be amputation,/post op, akt	ceftriaxone	1	IV	BD	1	post op
2	82	19/12/2019 15:48:49	Male	type-2dm, ihd, col, copd, ht	ofloxacin	0.4	IV	BD	3	abd distension with leg swelling
3	82	19/12/2019 15:50:33	Male	type-2 dm, ihd, col, copd, ht	cefipime	1	IV	BD	5	abd distension with leg swelling
4	82	19/12/2019 15:52:20	Male	type-2 dm, ihd, col, copd, ht	azithromycin	0.5	Oral	OD	3	abd distension with leg swelling

```
Columns in DataFrame:
['Age', 'Date of Data Entry', 'Gender', 'Diagnosis', 'Name of Drug', 'Dosage (gram)', 'Route', 'Frequency', 'Duration
(days)', 'Indication']
Bar Plot of Frequency of Each Drug:
```

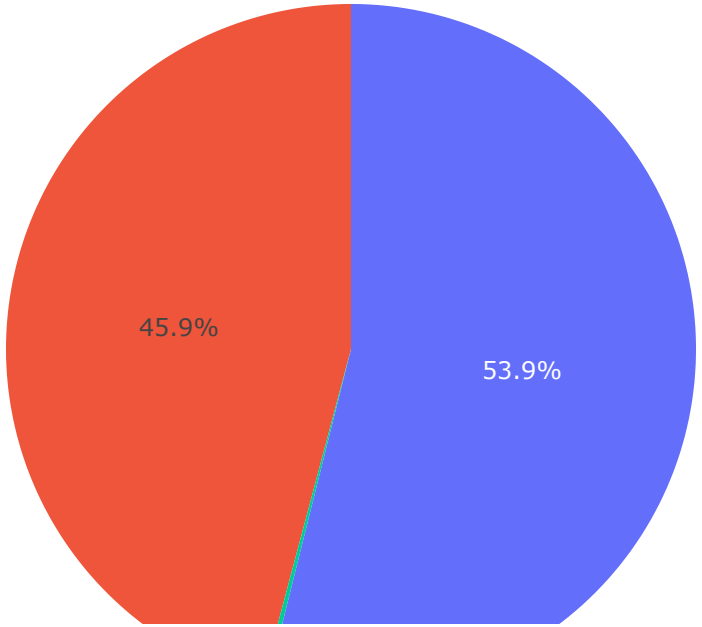


Histogram of Dosage Distribution:



Pie Chart of Gender Distribution:

Gender Distribution




```
Column 'Type  (Hospital / Nursing Home / Lab)' not found in dataset.
Column ' Class : (Public / Private)' not found in dataset.
Column 'Pharmacy Available : Yes/No' not found in dataset.
Column 'Ambulance Service Available' not found in dataset.
No numeric columns found for correlation matrix.
Column 'OrganisationType' not found in dataset.
Column 'Sector' not found in dataset.
Column 'OrganisationStatus' not found in dataset.

Enter process password for translation:
Password(description='Process Password:')
Text(value='', description='Text to Translate:')
```

```
Text(value='', description='Destination Language:')
Button(description='Translate', style=ButtonStyle())
Process password verified.

Japanese translation: 男性は男性の病院で39.9です
Pronunciation: Dansei wa dansei no byōin de 39. 9Desu
Translated from: English
```

In [ ]: