Question

```
This work intend to answering the question “How well do DRG-predicted LOS values match actual values? Do any factors explain significant variability in the closeness of these values?”
```

Import module

In [1]:
```
import pprint
import sqlite3
import pathlib

import pandas as pd
import numpy as np
import seaborn as sns
sns.set_style('whitegrid')
import matplotlib as plt
from scipy import stats

import matplotlib.pyplot as plt
plt.style.use('default')
plt.style.use ('seaborn-darkgrid')

import plotly.express as px
from statsmodels.graphics.gofplots import qqplot
from statsmodels.stats.weightstats import ztest


pd.set_option('display.max_rows', 500)
pd.set_option('display.max_columns', 500)
```

Read from database

In [2]:
```
def query_sqlite_with_pandas(query):

    #with sqlite3.Connection('LOS_data_cd41.sqlite') as conn:
    with sqlite3.Connection('LOS_data_2023-02-01.sqlite') as conn:
        query_results = pd.read_sql(query, conn)
    conn.close()

    return query_results
```

Query the variables of interest

In [3]:
```
query = """
SELECT e.patient_encounter_tk, e.admission_datetime, e.discharge_datetime,              --Encounter information
            pe.drg_code, d.drg_description, d.gmlos, d.clinical_type, d.mdc,             --DRG information
            p.patient_tk, p.DOB, g.gender_abbreviation, p.ethnicity_tk, p.race_tk,      --Patient descriptives
            e.last_inpatient_admission_datetime, e.patient_class_tk,                    --Encounter descriptives
            f.facility_tk, f.hospital_system_tk, f.analytics_display_name               --Facility information

FROM cd_patient_encounters e
        LEFT JOIN cd_patient_encounter_drgs pe ON e.patient_encounter_tk=pe.patient_encounter_tk
        LEFT JOIN ref_drg_code_history d ON pe.drg_code=d.drg_code
        LEFT JOIN cd_patients p ON e.patient_tk=p.patient_tk
        LEFT JOIN ref_facilities f ON e.facility_tk=f.facility_tk
        LEFT JOIN cd_genders g2 ON p.gender_tk=g2.gender_tk and f.facility_tk=g2.facility_tk
        LEFT JOIN ref_genders  g ON g2.master_gender_tk=g.master_gender_tk;
"""

los_calc_df = query_sqlite_with_pandas(query)
```

In [4]:
```
#### Make datetime datatype
los_calc_df['admission_datetime'] = pd.to_datetime(los_calc_df['admission_datetime'])
los_calc_df['discharge_datetime'] = pd.to_datetime(los_calc_df['discharge_datetime'],errors='coerce')
los_calc_df['last_inpatient_admission_datetime'] = pd.to_datetime(los_calc_df['last_inpatient_admission_datetime'],errors=
los_calc_df['dob'] = pd.to_datetime(los_calc_df['dob'])
```

Initial Data Exploration and Understanding and Data Transformstion

Create New Fields

Lenght of stay LOS is the actual duration of time used by a recorder patient encounter. This is calculated by subtracting the discharge time from admission time

In [5]:
```
los_calc_df['encounter_los'] = los_calc_df['discharge_datetime']-los_calc_df['admission_datetime']
```

In [6]:
```
los_calc_df['encounter_los'] = los_calc_df['encounter_los'] / pd.Timedelta(days=1)
```

In [293]:
```
#check for unique values in gender column
los_calc_df.gender_abbreviation.unique()
```

Out[293]:
```
array(['M', 'F', 'U'], dtype=object)
```

Difference between DRG LOS and ENC LOS, Meet_it criteria and return_visit

LOS Difference is the calculated value as a difference between encounter_los from the expected or given Geometric Mean Lenght of stay GMLOS. There is the MDC that group DRGs into different categories. The DRGs are mapped to MDC using a file called MDC mapper. Return flag is to code any patient that has been to the hospital system before i.e a returning patient.There are instances where admission date is less than date of birth this is specific for foetus who are not yet born as at admission. There is a meet_it binary criteria that shows for each patient encounter if they exceed the GMLOS or not.

In [8]:
```
#calculate difference
los_calc_df['los_difference'] = los_calc_df['encounter_los']-los_calc_df['gmlos']
```

In [9]:
```
### MDC Description
mdc_mapper = pd.read_csv("mdc_mapper.csv", dtype='object').set_index('mdc')
los_calc_df = los_calc_df.join(mdc_mapper, on='mdc')
los_calc_df['mdc_description'] = los_calc_df.mdc_description.fillna('')
```

In [10]:
```
# Return Visit Flag
los_calc_df['return_visit'] = 1*(~los_calc_df['last_inpatient_admission_datetime'].isna())
```

In [11]:
```
# patient : foetus i.e having DOB to be less than admission date
(los_calc_df['admission_datetime']<los_calc_df['dob']).any()
```

Out[11]:
```
True
```

In [ ]:
```
los_calc_df[(los_calc_df['admission_datetime']<los_calc_df['dob'])]
```

In [13]:
```
los_calc_df.dtypes
```

Out[13]:
```
patient_encounter_tk                    int64
admission_datetime             datetime64[ns]
discharge_datetime             datetime64[ns]
drg_code                               object
drg_description                        object
gmlos                                 float64
clinical_type                          object
mdc                                    object
patient_tk                              int64
dob                            datetime64[ns]
gender_abbreviation                    object
ethnicity_tk                            int64
race_tk                                 int64
last_inpatient_admission_datetime    datetime64[ns]
patient_class_tk                        int64
facility_tk                             int64
hospital_system_tk                      int64
analytics_display_name                 object
encounter_los                         float64
los_difference                        float64
mdc_description                        object
return_visit                            int32
dtype: object
```

In [14]:
```
los_calc_df.nunique()
```

Out[14]:
```
patient_encounter_tk                160205
admission_datetime                  133779
discharge_datetime                  101921
drg_code                               759
drg_description                        740
gmlos                                  106
clinical_type                            2
mdc                                     27
patient_tk                          130242
dob                                  29487
gender_abbreviation                      3
ethnicity_tk                            23
race_tk                                240
last_inpatient_admission_datetime     3360
patient_class_tk                        76
facility_tk                             31
hospital_system_tk                       4
analytics_display_name                  31
encounter_los                        29220
los_difference                       52522
mdc_description                         26
return_visit                             2
dtype: int64
```

In [15]:
```
# Age at Admission
los_calc_df['age_at_admit'] = los_calc_df['admission_datetime']-los_calc_df['dob']
los_calc_df['age_at_admit'] = los_calc_df['age_at_admit'].astype('timedelta64[Y]').astype(int)
```

In [16]:
```
# meet_it (Is gmlos met?)
los_calc_df['meet_it'] = 1*(los_calc_df['encounter_los'] <= los_calc_df['gmlos'])
```

Data Cleaning

In [17]:
```
los_calc_df[los_calc_df['encounter_los'] == 0].shape
#446 rows with exact same admit and discharge time
```

Out[17]: (446, 24)

In [18]:
```
encounter_records = los_calc_df[los_calc_df['encounter_los']> 0]
#1279 rows have discharge at or before admit - these are dropped
#also drops 497 n/a discharges
```

In [19]:
```
encounter_records = encounter_records[encounter_records['encounter_los'] <= 180]
#drop 22 extreme outliers
```

In [20]:
```
encounter_records[encounter_records['gmlos'].isna()]['drg_code'].unique()
#8880 records in 18 drg codes do not map to a gmlos
```

Out[20]:
```
array(['767', '', '765', '775', '133', '766', '774', '777', '238', '131',
       '484', '782', '129', '490', '778', '130', '781', '998', '132'],
      dtype=object)
```

In [21]:
```
encounter_records[encounter_records['drg_code']==''].shape
#8615 of those records have a blank drg
```

Out[21]: (8615, 24)

In [22]:
```
encounter_records_final = encounter_records[~encounter_records['encounter_los'].isna()]
```

Create DRG Grouped Data

In [23]:
```
agg_funcs = {'encounter_los':stats.gmean,
             'age_at_admit':'mean',
             'return_visit':'mean',
             'meet_it':'mean',
             'patient_encounter_tk':'count'}
group_by_columns = ['drg_code','drg_description','mdc','mdc_description','clinical_type','gmlos',]#,
```

In [24]:
```
drg_records_prelim = encounter_records_final.groupby(group_by_columns).agg(agg_funcs).reset_index()
```

In [25]:
```
rename_cols = {"encounter_los":"actual_gmlos",
               "age_at_admit":"average_age",
               "return_visit":"pct_return_visits",
               "meet_it":"pct_meet_it",
               "patient_encounter_tk":"num_encounters"}
```

In [40]:
```
drg_records = drg_records_prelim.dropna().rename(columns=rename_cols)
drg_records['gmlos_difference'] = drg_records.eval("actual_gmlos-gmlos")
drg_records['drg_meet_it'] = 1*(drg_records['actual_gmlos'] <= drg_records['gmlos'])
drg_records.head()
drg_records_final = drg_records.copy()
```

In [ ]:
```

```

In [43]:
```
#select columns od interest
column = ['drg_code', 'mdc', 'clinical_type','average_age', 'pct_return_visits', 'num_encounters',
          'gmlos','actual_gmlos','gmlos_difference','drg_meet_it']
df2=drg_records_final[column]
```

In [45]:
```
df2
```

Out[45]:

	drg_code	mdc	clinical_type	average_age	pct_return_visits	num_encounters	gmlos	actual_gmlos	gmlos_difference	drg_meet_it
0	001	PRE	SURG	60.571429	0.190476	21	30.1	20.212580	-9.887420	1
1	003	PRE	SURG	55.146341	0.012195	164	22.4	18.259437	-4.140563	1
2	004	PRE	SURG	60.181159	0.028986	138	20.0	22.124499	2.124499	0
3	011	PRE	SURG	72.421053	0.000000	19	10.9	0.771884	-10.128116	1
4	012	PRE	SURG	64.666667	0.000000	9	8.3	4.845043	-3.454957	1
...	...	...	...	...	...	...	...	...	...	...
734	982		SURG	60.717791	0.036810	163	4.6	4.288287	-0.311713	1
735	983		SURG	60.648649	0.000000	37	2.3	2.085249	-0.214751	1
736	987		SURG	65.827273	0.036364	110	7.7	6.624850	-1.075150	1
737	988		SURG	63.370690	0.017241	116	4.3	3.754208	-0.545792	1
738	989		SURG	56.647059	0.000000	17	2.3	2.481734	0.181734	0

739 rows × 10 columns

In [ ]:
```

```

In [ ]:
```

```

In [46]:
```
#select columns of intereest
columns = ['drg_code', 'mdc','gender_abbreviation', 'clinical_type','age_at_admit', 'return_visit', 'gmlos','encounter_los
```

In [47]:
```
encounter_records[columns].isna().sum()
```

Out[47]:
```
drg_code                0
mdc                  8880
gender_abbreviation     0
clinical_type        8880
age_at_admit            0
return_visit            0
gmlos                8880
encounter_los           0
los_difference       8880
meet_it                 0
dtype: int64
```

In [312]:
```
# drop all rows that have no value, not a number
df1=encounter_records[columns].dropna().reset_index(drop='Index')
df1
```

Out[312]:

	drg_code	mdc	gender_abbreviation	clinical_type	age_at_admit	return_visit	gmlos	encounter_los	los_difference	meet_it
0	872	18	M	MED	55	0	3.5	4.162500	0.662500	0
1	441	07	F	MED	42	0	4.7	2.091667	-2.608333	1
2	871	18	F	MED	46	0	4.8	7.886806	3.086806	0
3	442	07	F	MED	47	0	3.2	1.006250	-2.193750	1
4	708	12	M	SURG	51	0	1.4	1.203472	-0.196528	1
...	...	...	...	...	...	...	...	...	...	...
149520	794	15	U	MED	0	0	3.4	1.873681	-1.526319	1
149521	026	01	F	SURG	64	0	3.8	1.264468	-2.535532	1
149522	787	14	F	SURG	37	0	3.5	2.150914	-1.349086	1
149523	982		F	SURG	35	0	4.6	1.933727	-2.666273	1
149524	795	15	M	MED	0	0	3.1	2.054375	-1.045625	1

149525 rows × 10 columns

In [313]:
```
df1.rename(columns={'gender_abbreviation':'gender','age_at_admit':'age','mdc':'mdc','encounter_los':'LOS','los_difference'
df1
```

Out[313]:

	drg_code	mdc	gender	clinical_type	age	return_visit	gmlos	LOS	LOS_Diff	meet_it
0	872	18	M	MED	55	0	3.5	4.162500	0.662500	0
1	441	07	F	MED	42	0	4.7	2.091667	-2.608333	1
2	871	18	F	MED	46	0	4.8	7.886806	3.086806	0
3	442	07	F	MED	47	0	3.2	1.006250	-2.193750	1
4	708	12	M	SURG	51	0	1.4	1.203472	-0.196528	1
...	...	...	...	...	...	...	...	...	...	...
149520	794	15	U	MED	0	0	3.4	1.873681	-1.526319	1
149521	026	01	F	SURG	64	0	3.8	1.264468	-2.535532	1
149522	787	14	F	SURG	37	0	3.5	2.150914	-1.349086	1
149523	982		F	SURG	35	0	4.6	1.933727	-2.666273	1
149524	795	15	M	MED	0	0	3.1	2.054375	-1.045625	1

149525 rows × 10 columns

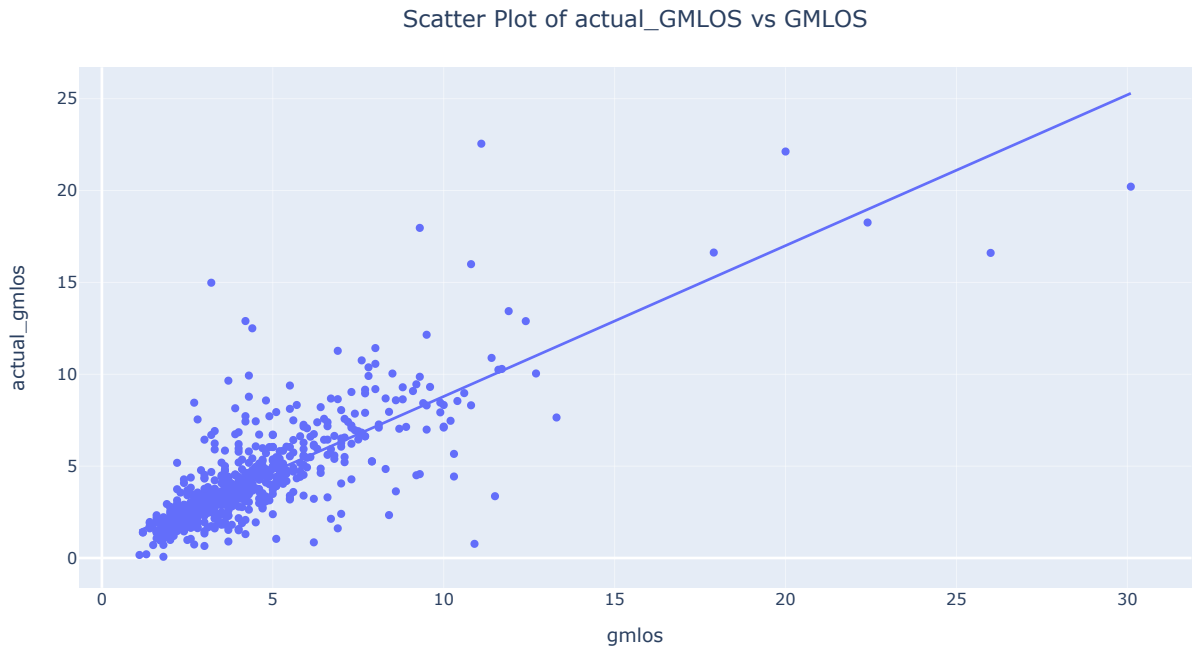
In [ ]:
```

```

Business Intelligence

```
### How well does GMLOS predict length of stay for DRG encounters?•
We like to understand how the given GMLON accurately explains the actual variation in hospital patients encounter lenght
of stay (Actual_GMLOS). We plot a linear fit of the two using a scatter plot and a line of fit
```

In [ ]:
```

```

In [177]:
```
def scatter(df,x,y,title):
    '''scatter plot to plot a linear line of best fit between two continuos values x and y'''
    fig = px.scatter(df, x=x, y=y, trendline='ols')
    # Set plot title and center-align it
    fig.update_layout(
        title=title,
        title_x=0.5  # Value of 0.5 centers the title
    )
    fig.show()
scatter(df=df2,x='gmlos',y='actual_gmlos',title='Scatter Plot of actual_GMLOS vs GMLOS')
```



Difference between Encounter LOS(Patient's Actual GMLOS) and given Geometric Mean LOS

At DRG level

A histogram showing the distribution of Grouped DRGs and showing how many DRGs meet the government GMLOS. We expect data point to be at the mean of zero , slightly left skewed and a decent kurtois. Against our thought there are lots of DRGs that do not meet the GMLOS as the distribution is slightly more skewed to the right.

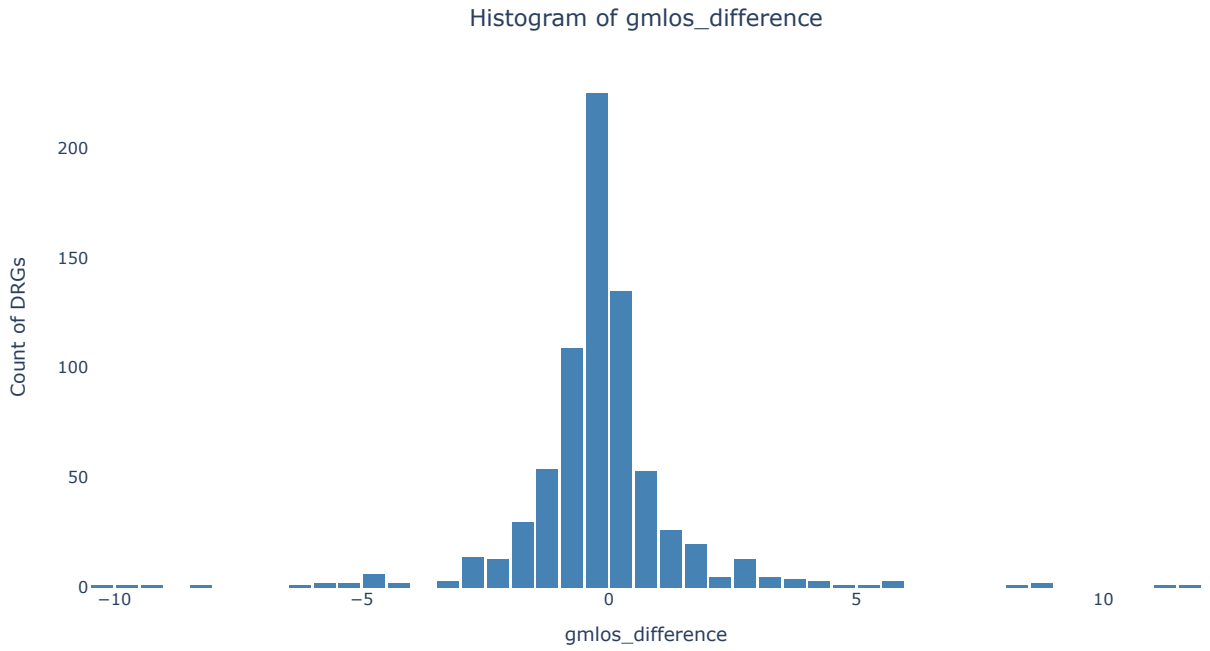
In [96]:
```
#aggregated dataframe used (grouped DRGs)
df2
```

Out[96]:

	drg_code	mdc	clinical_type	average_age	pct_return_visits	num_encounters	gmlos	actual_gmlos	gmlos_difference	drg_meet_it
0	001	PRE	SURG	60.571429	0.190476	21	30.1	20.212580	-9.887420	1
1	003	PRE	SURG	55.146341	0.012195	164	22.4	18.259437	-4.140563	1
2	004	PRE	SURG	60.181159	0.028986	138	20.0	22.124499	2.124499	0
3	011	PRE	SURG	72.421053	0.000000	19	10.9	0.771884	-10.128116	1
4	012	PRE	SURG	64.666667	0.000000	9	8.3	4.845043	-3.454957	1
...	...	...	...	...	...	...	...	...	...	...
734	982		SURG	60.717791	0.036810	163	4.6	4.288287	-0.311713	1
735	983		SURG	60.648649	0.000000	37	2.3	2.085249	-0.214751	1
736	987		SURG	65.827273	0.036364	110	7.7	6.624850	-1.075150	1
737	988		SURG	63.370690	0.017241	116	4.3	3.754208	-0.545792	1
738	989		SURG	56.647059	0.000000	17	2.3	2.481734	0.181734	0

739 rows × 10 columns

In [52]:
```
fig = px.histogram(df2, x='gmlos_difference')

# Set plot title and center-align it
fig.update_layout(
    title='Histogram of gmlos_difference',
    title_x=0.5  # Value of 0.5 centers the title
)

# Customize the bar colors
fig.update_traces(marker_color='steelblue')

# Set x-axis label
fig.update_xaxes(title='gmlos_difference')

# Set y-axis label
fig.update_yaxes(title='Count of DRGs')

# Customize the layout
fig.update_layout(
    plot_bgcolor='white',   # Set plot background color
    barmode='overlay',      # Display bars in overlay mode
    bargap=0.1,             # Set gap between bars
    bargroupgap=0.05,       # Set gap between bar groups
)

fig.show()
```

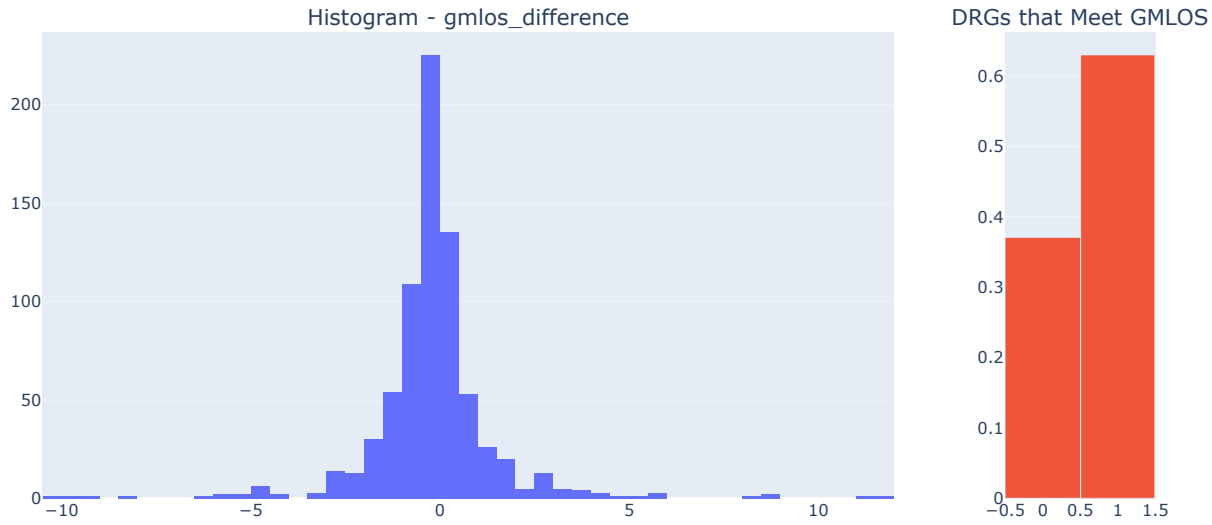


In [ ]:
```
1
```

Plot distributtion side by side to a bar chart showing the proportion

In [104]:

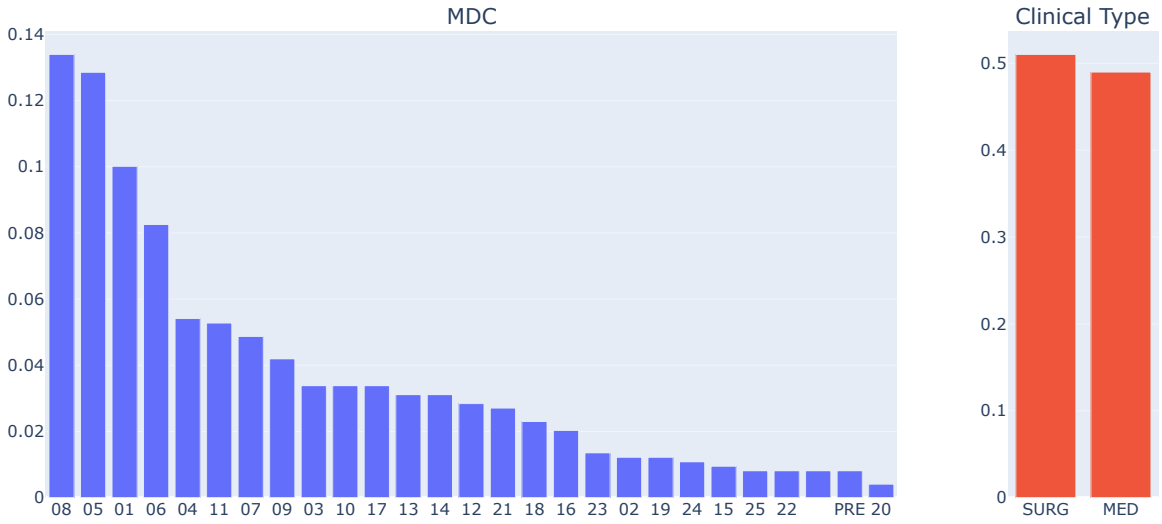
```
# Create a subplot with 1 row and 2 columns
fig = sp.make_subplots(rows=1, cols=2, subplot_titles=('Histogram - gmlos_difference', 'DRGs that Meet GMLOS'),
                       column_widths=[0.85, 0.15])

# Histogram - gmlos_difference
fig.add_trace(go.Histogram(x=df2['gmlos_difference']), row=1, col=1)

# Bar Chart - drg_meet_it
meet_counts = df2['drg_meet_it'].value_counts(normalize=True)
fig.add_trace(go.Bar(x=meet_counts.index, y=meet_counts.values, #marker=dict(color=['skyblue', 'skyblue'])
#                  name='drg_meet_it',
                   ), row=1, col=2)

# Update subplot layout
fig.update_layout(showlegend=False)

fig.update_layout(legend=dict(x=0, y=0))

fig.show()
```



Plot bar chat of MDCs side by side to a bar chart showing the proportion of clinical type of patient

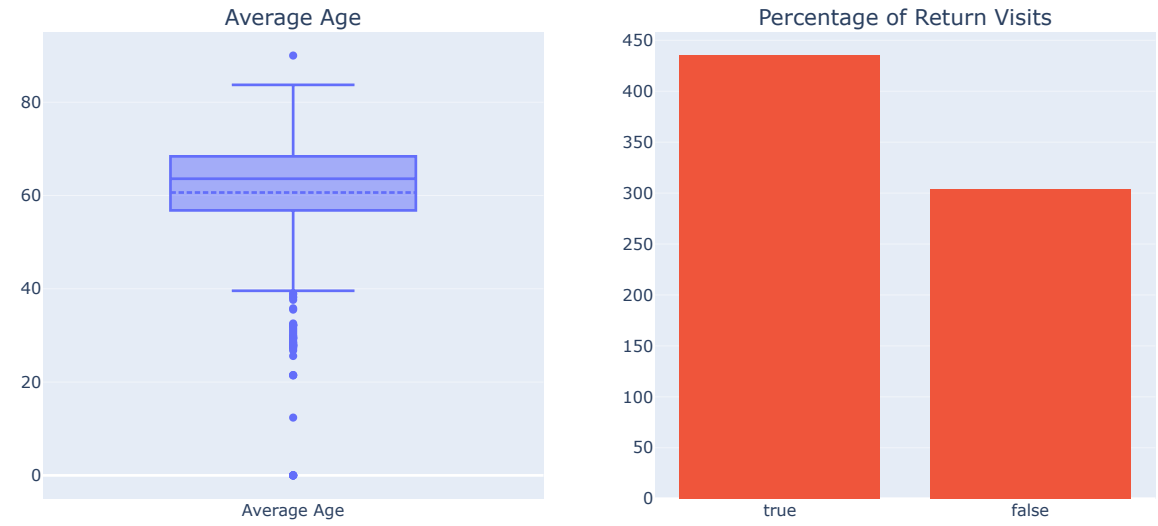
In [107]:

```
# Create a subplot with 1 row and 2 columns
fig = sp.make_subplots(rows=1, cols=2, subplot_titles=('Counts of MDC', 'Counts of Clinical Type'),
                       column_widths=[0.85, 0.15])


# Bar Chart - count of MDCs
meet_counts = df2['mdc'].value_counts(normalize=True)
fig.add_trace(go.Bar(x=meet_counts.index, y=meet_counts.values, #marker=dict(color=['skyblue', 'skyblue'])
#                 name='drg_meet_it',
                  ), row=1, col=1)

# Bar Chart - count of Clinical type
meet_count = df2['clinical_type'].value_counts(normalize=True)
fig.add_trace(go.Bar(x=meet_count.index, y=meet_count.values, #marker=dict(color=['skyblue', 'skyblue'])
#                 name='drg_meet_it',
                  ), row=1, col=2)

# Update subplot layout
fig.update_layout(showlegend=False)

fig.update_layout(legend=dict(x=0, y=0))

fig.show()

```



Plot Box plot of Age side by side to a bar chart showing the proportion of Return visit

In [121]:
```
# Create a subplot with 2 rows and 2 columns
fig = sp.make_subplots(rows=1, cols=2, subplot_titles=('Average Age', 'Percentage of Return Visits',
                                                       ))

# Average Age
fig.add_trace(go.Box(y=df2['average_age'], boxmean=True, name='Average Age'), row=1, col=1)

# Percentage of Return Visits
fig.add_trace(go.Histogram(x=df2['pct_return_visits']>0, name=' Return Visits'), row=1, col=2)



# Update subplot layout
fig.update_layout(showlegend=False)

fig.show()

```



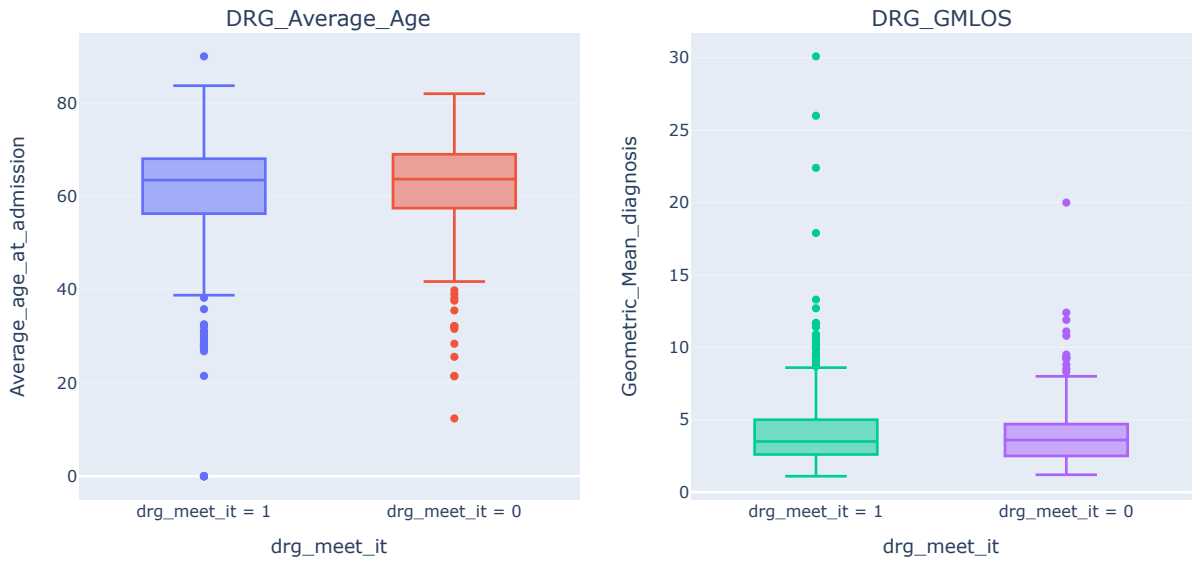
In [ ]:
```
1
```

Boxlot of average age and GMLOS grouped by meet_it(0and 1)

In [ ]:
```
def plot(df1,title1,title2,y_text1,y_text2,target,col1,col2):

    # Create a subplot with 1 row and 2 columns
    fig = sp.make_subplots(rows=1, cols=2, subplot_titles=(title1, title2))

    # Box plot for Average Age, grouped by drg_meet_it
    for value in df1[target].unique():
        fig.add_trace(go.Box(y=df1[df1[target] == value][col1],
                             name='{} = {}'.format(target,value)), row=1, col=1)

    # Box plot for Actual GMLOS, grouped by drg_meet_it
    for value in df1[target].unique():
        fig.add_trace(go.Box(y=df1[df1[target] == value][col2],
                             name='{} = {}'.format(target,value)), row=1, col=2)

    # Update subplot layout
    fig.update_layout(showlegend=False)

    # Set titles for the subplots
    fig.update_annotations(
        dict(text=title1, x=0.17, y=1.06, showarrow=False, font=dict(size=12)),
        dict(text=title2, x=0.78, y=1.06, showarrow=False, font=dict(size=12))
    )

    # Update layout for each subplot
    fig.update_xaxes(title_text=target, row=1, col=1)
    fig.update_xaxes(title_text=target, row=1, col=2)
    fig.update_yaxes(title_text=y_text1, row=1, col=1)
    fig.update_yaxes(title_text=y_text2, row=1, col=2)

    # Show the figure
    fig.show()
```

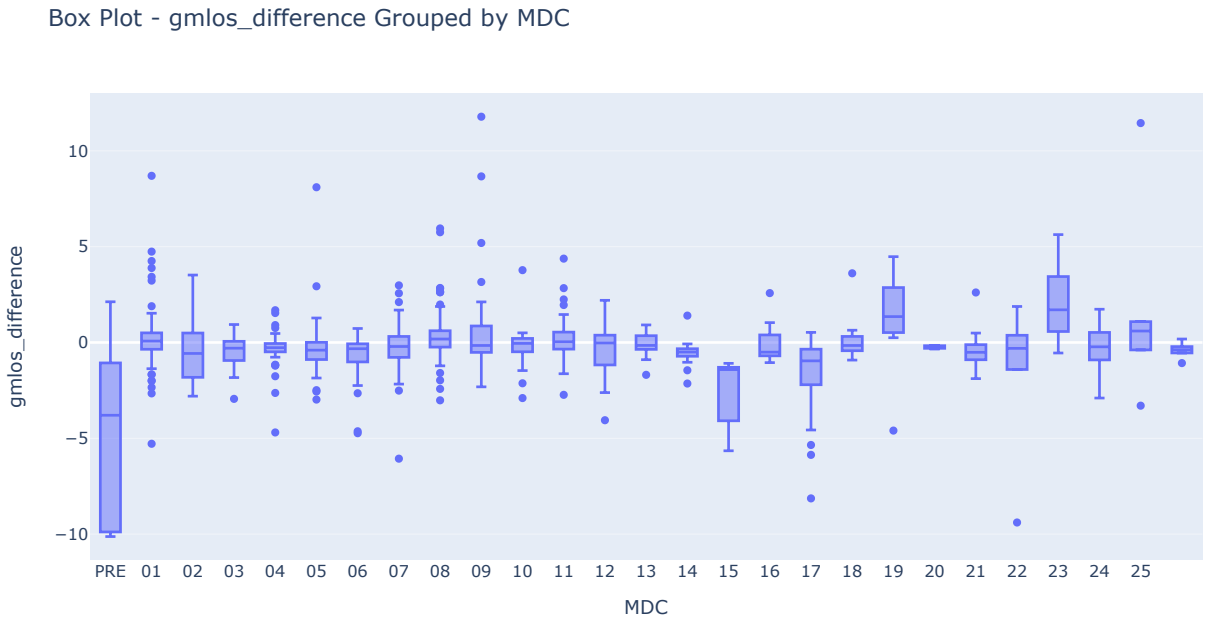
Boxplot of AGE and GMLOS grouped by drg_meet_it at DRG level

In [146]:
```
plot(df2,'DRG_Average_Age','DRG_GMLOS','Average_age_at_admission','Geometric_Mean_diagnosis','drg_meet_it','average_age','
```



Boxlot of GMLOS difference grouped by MDC

In [156]:
```
def cat_box(df,x,y,title,xaxes_title,yaxes_title):
    # Create the box plot
    fig = px.box(df, x=x , y=y , title=title ,)

    # Update layout
    fig.update_xaxes(title=xaxes_title)
    fig.update_yaxes(title=yaxes_title)

    # Show the figure
    fig.show()
cat_box(df2,'mdc','gmlos_difference','Box Plot - gmlos_difference Grouped by MDC','MDC','gmlos_difference')
```



In [ ]:
```
1
```

Difference between Encounter LOS(Patient's Actual GMLOS) and given Geometric Mean LOS

- ENCOUNTER LEVEL

We checked for the distribution for each encounter in the as recorded system in the most granular form

In [54]:
```
1 df1
```

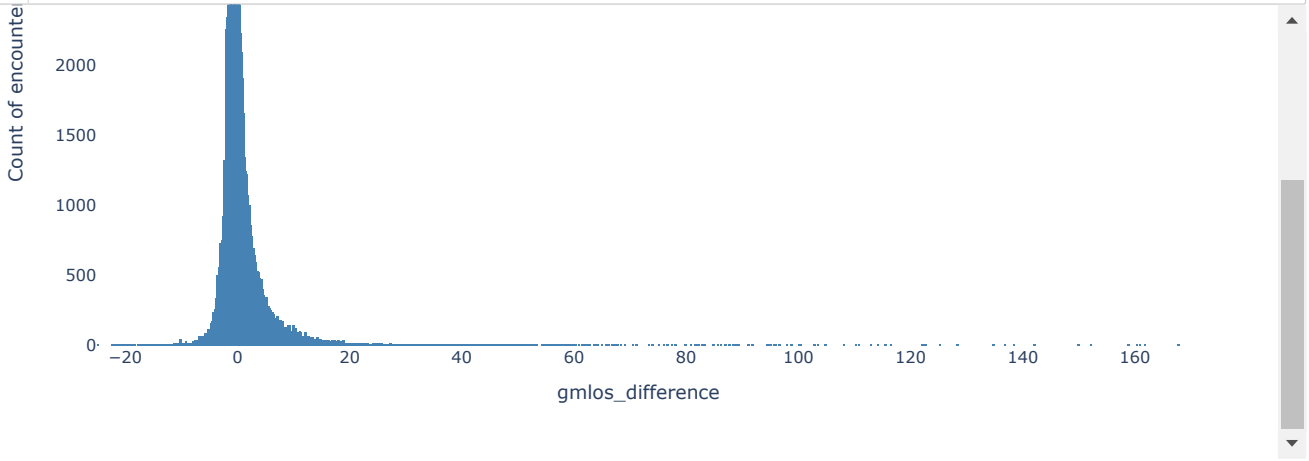
	drg_code	mdc	gender	clinical_type	age	return_visit	gmlos	LOS	LOS_Diff	meet_it
0	872	18	M	MED	55	0	3.5	4.162500	0.662500	0
1	441	07	F	MED	42	0	4.7	2.091667	-2.608333	1
2	871	18	F	MED	46	0	4.8	7.886806	3.086806	0
3	442	07	F	MED	47	0	3.2	1.006250	-2.193750	1
4	708	12	M	SURG	51	0	1.4	1.203472	-0.196528	1
...	...	...	...	...	...	...	...	...	...	...
149520	794	15	U	MED	0	0	3.4	1.873681	-1.526319	1
149521	026	01	F	SURG	64	0	3.8	1.264468	-2.535532	1
149522	787	14	F	SURG	37	0	3.5	2.150914	-1.349086	1
149523	982		F	SURG	35	0	4.6	1.933727	-2.666273	1
149524	795	15	M	MED	0	0	3.1	2.054375	-1.045625	1

149525 rows × 10 columns

The distibution is very right skewed as a lot of patients are not meeting the specific GMLOS for their ailment as suggested by the government

In [ ]:
```
def hist(df,x,xaxes_title,yaxes_title,layout_title):
    fig = px.histogram(df, x)


    # Customize the bar colors
    fig.update_traces(marker_color='steelblue')

    # Set x-axis label
    fig.update_xaxes(title=xaxes_title)

    # Set y-axis label
    fig.update_yaxes(title=yaxes_title)

    # Set plot title and center-align it
    fig.update_layout(
        title=layout_title,
        title_x=0.5  # Value of 0.5 centers the title
    )

    # Customize the layout
    fig.update_layout(
        plot_bgcolor='white',   # Set plot background color
    )

    fig.show()

```

In [153]:
```

hist(df1,'LOS_Diff','gmlos_difference','Count of encounters','Histogram of gmlos_difference-Encounter Level')
```




```
To understand what is happening we reduced the skewness by eliminating those who exceed the limit over 25 days differnt
```

In [150]:
```
#slice the frame limiting to those patient who do not exceed their GMLOS more than 25 days
data = df1[df1.LOS_Diff<25]
data
```

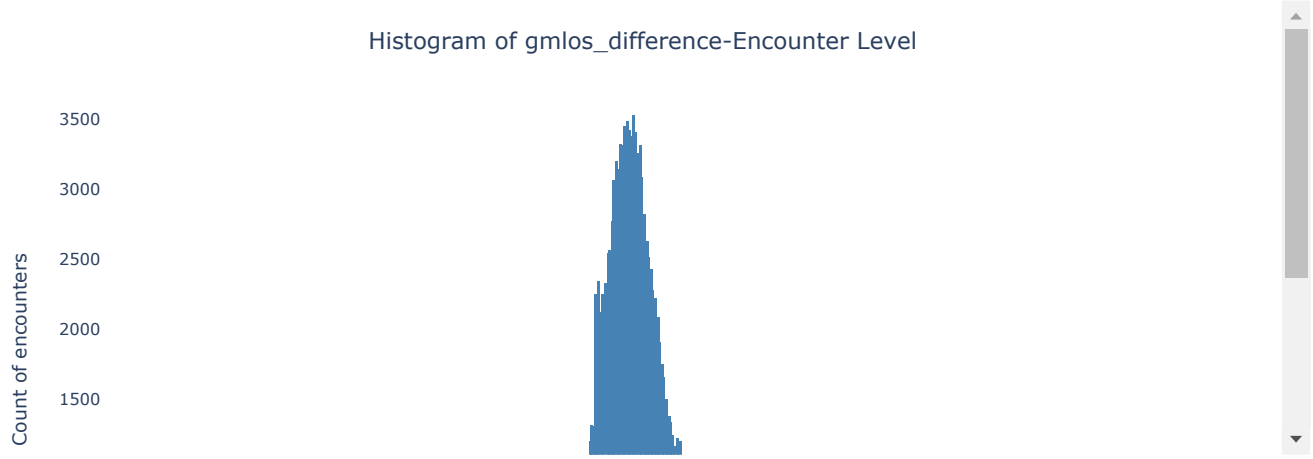
Out[150]:

	drg_code	mdc	gender	clinical_type	age	return_visit	gmlos	LOS	LOS_Diff	meet_it
0	872	18	M	MED	55	0	3.5	4.162500	0.662500	0
1	441	07	F	MED	42	0	4.7	2.091667	-2.608333	1
2	871	18	F	MED	46	0	4.8	7.886806	3.086806	0
3	442	07	F	MED	47	0	3.2	1.006250	-2.193750	1
4	708	12	M	SURG	51	0	1.4	1.203472	-0.196528	1
...	...	...	...	...	...	...	...	...	...	...
149520	794	15	U	MED	0	0	3.4	1.873681	-1.526319	1
149521	026	01	F	SURG	64	0	3.8	1.264468	-2.535532	1
149522	787	14	F	SURG	37	0	3.5	2.150914	-1.349086	1
149523	982		F	SURG	35	0	4.6	1.933727	-2.666273	1
149524	795	15	M	MED	0	0	3.1	2.054375	-1.045625	1

148603 rows × 10 columns

In [ ]:
```

```

In [152]:
```
hist(data,'LOS_Diff','gmlos_difference','Count of encounters','Histogram of gmlos_difference-Encounter Level')
```



In [ ]:
```

```

Scatter plot of age vs los_difference

This does not give a particular correlation the data point is clustered around zero level as expected

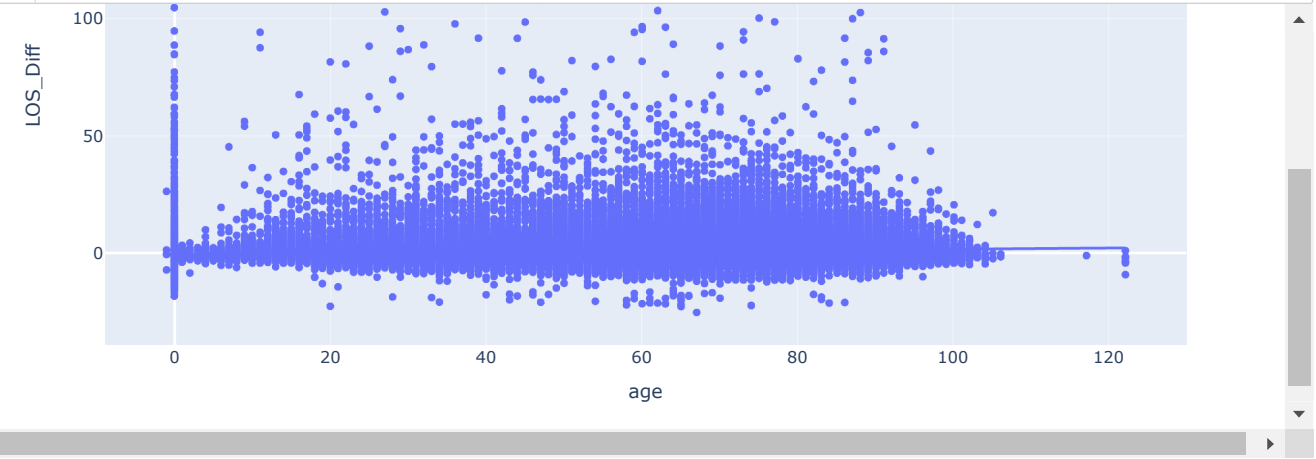
In [315]:
```
df1
```

Out[315]:

	drg_code	mdc	gender	clinical_type	age	return_visit	gmlos	LOS	LOS_Diff	meet_it
0	872	18	M	MED	55	0	3.5	4.162500	0.662500	0
1	441	07	F	MED	42	0	4.7	2.091667	-2.608333	1
2	871	18	F	MED	46	0	4.8	7.886806	3.086806	0
3	442	07	F	MED	47	0	3.2	1.006250	-2.193750	1
4	708	12	M	SURG	51	0	1.4	1.203472	-0.196528	1
...	...	...	...	...	...	...	...	...	...	...
149520	794	15	U	MED	0	0	3.4	1.873681	-1.526319	1
149521	026	01	F	SURG	64	0	3.8	1.264468	-2.535532	1
149522	787	14	F	SURG	37	0	3.5	2.150914	-1.349086	1
149523	982		F	SURG	35	0	4.6	1.933727	-2.666273	1
149524	795	15	M	MED	0	0	3.1	2.054375	-1.045625	1

149525 rows × 10 columns

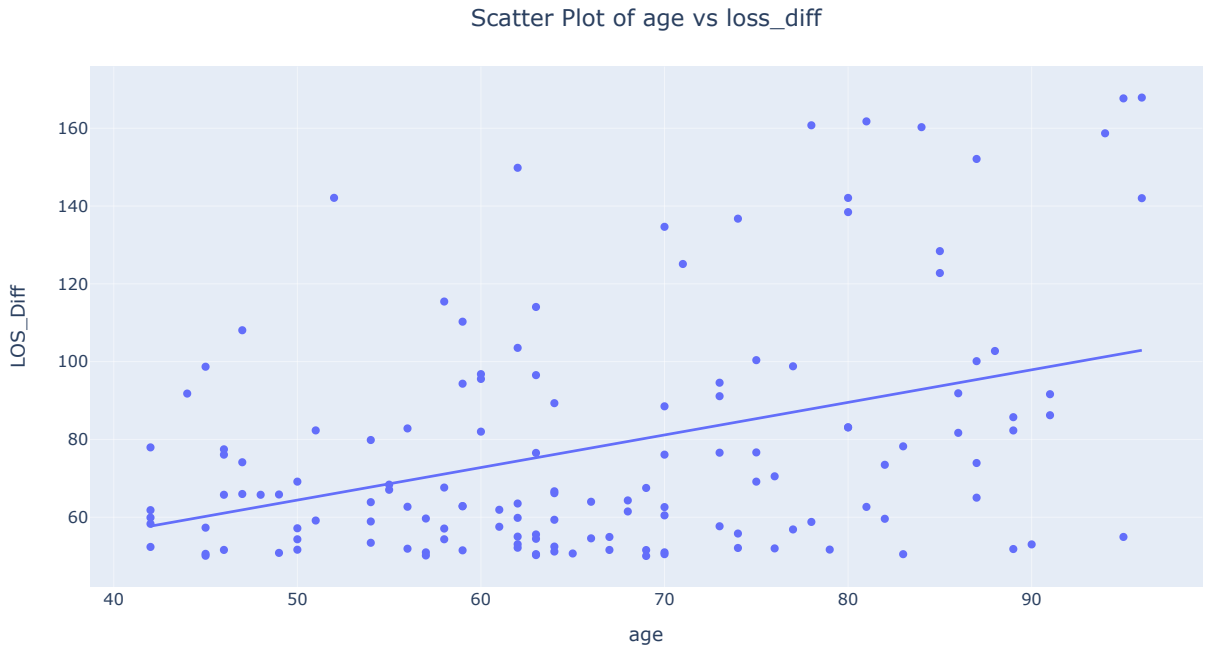
In [316]:
```

scatter(df1,'age','LOS_Diff','Scatter Plot of age vs loss_diff')
```



Scatter plot for slized Age vs Difference in Lenght of stay (Actual GMLOS -GMLOS)

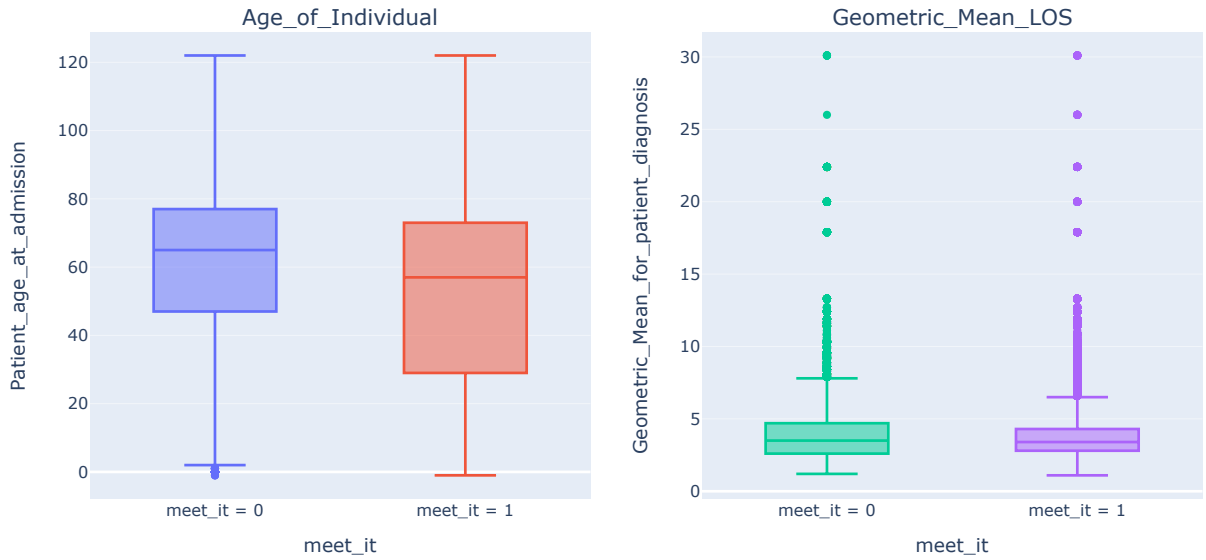
When we removed those who constantly meet it e.g foetus we see a clearer linear correlation between LOS_Difference and age of patient. It shows that as patient get older there is a higher tendency not to meet the GMLOS

In [319]:
```
##sliced dataframe for those LOS
df_sliced = df1[(df1['LOS_Diff'] > 50) & (df1['age'] >40)]
scatter(df_sliced,'age','LOS_Diff','Scatter Plot of age vs loss_diff')
```



Boxplot of AGE and GMLOS grouped by drg_meet_it at Encounter level

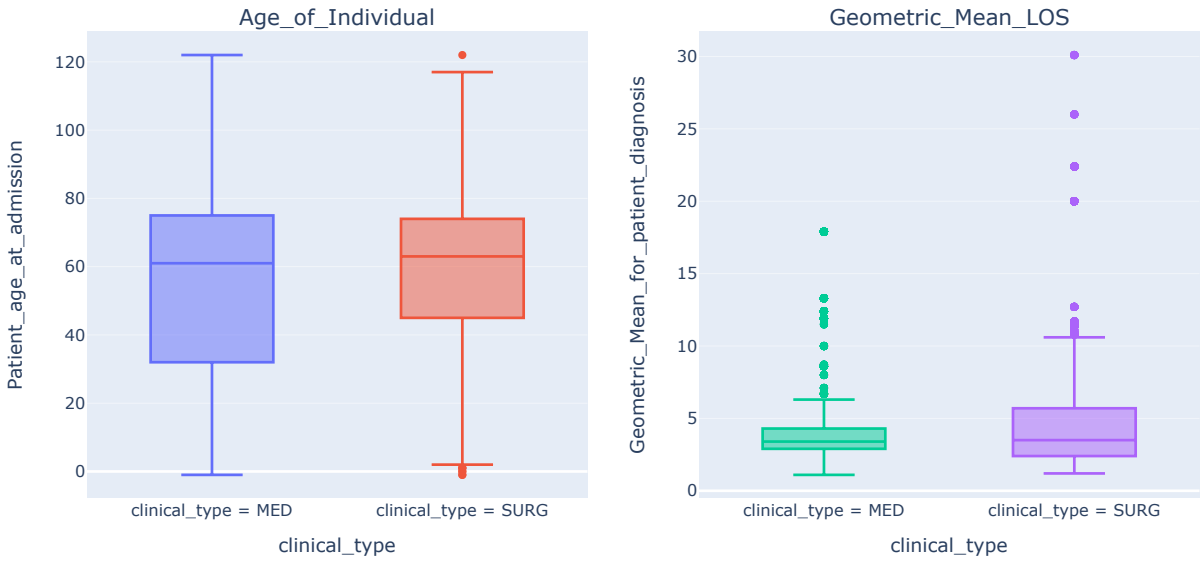
In [147]:
```
plot(df1,'Age_of_Individual','Geometric_Mean_LOS','Patient_age_at_admission','Geometric_Mean_for_patient_diagnosis','meet_
```



In [ ]:
```
1
```

Boxplot of AGE and GMLOS grouped by clinical_type at Encounter level

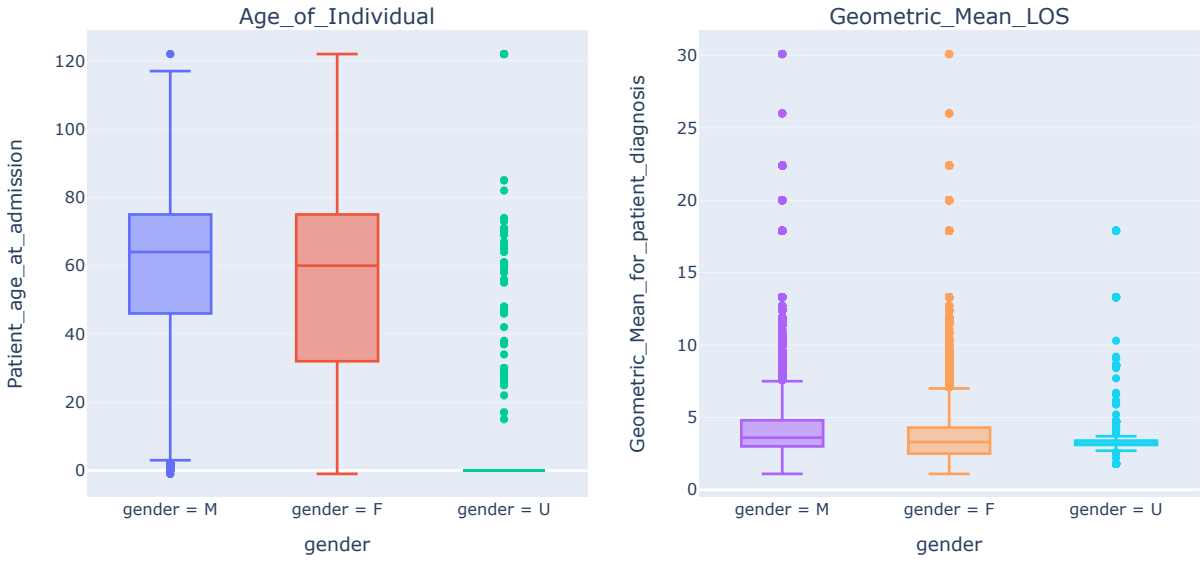
In [148]:
```
1 plot(df1,'Age_of_Individual','Geometric_Mean_LOS','Patient_age_at_admission','Geometric_Mean_for_patient_diagnosis','clini
```



In [ ]:
```
1
```

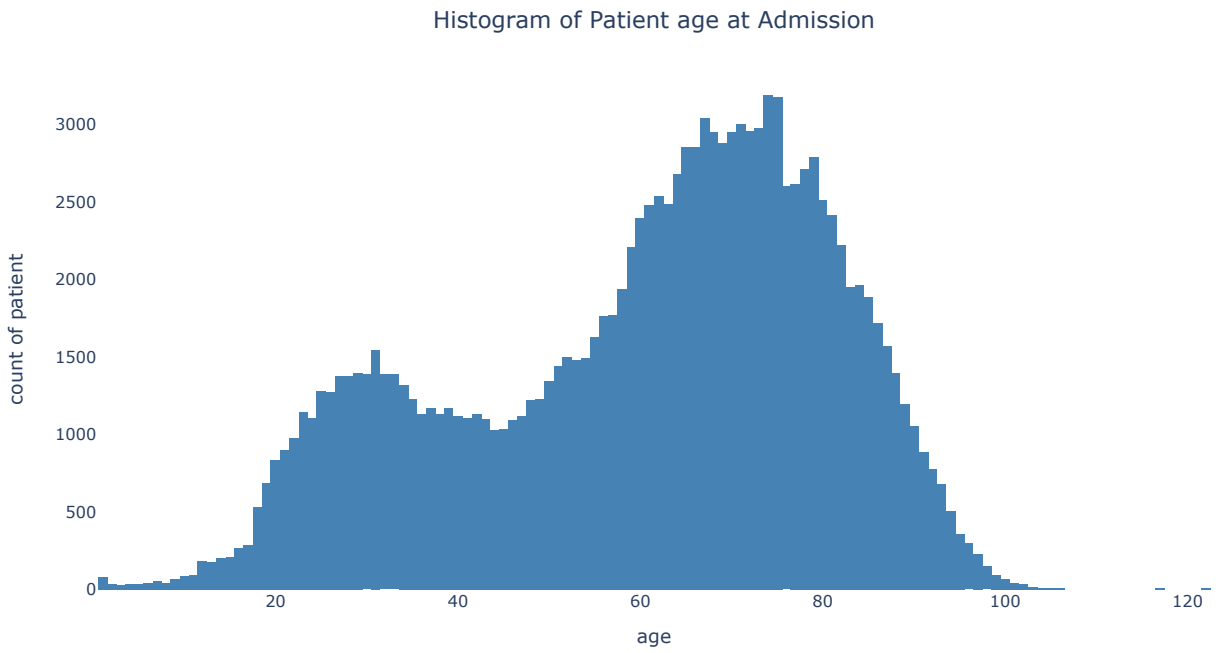
Boxplot of AGE and GMLOS grouped by Gender(Female , Male and Unknown) at Encounter level

In [149]:
```
1 plot(df1,'Age_of_Individual','Geometric_Mean_LOS','Patient_age_at_admission','Geometric_Mean_for_patient_diagnosis','gende
```



Histogram of Age for patients Greater than 0 yrs

In [216]:
```
hist(df1[df1.age>0],'age','age','count of patient','Histogram of Patient age at Admission')
```



In [ ]:
```

```

In [ ]:
```
df=pd.read_csv('encounter_records.csv')
df.drop(columns=['Unnamed: 0'],inplace=True)

###replace the MDC without name  with 00
df.mdc.replace(to_replace= ' ', value='00',inplace=True)

```

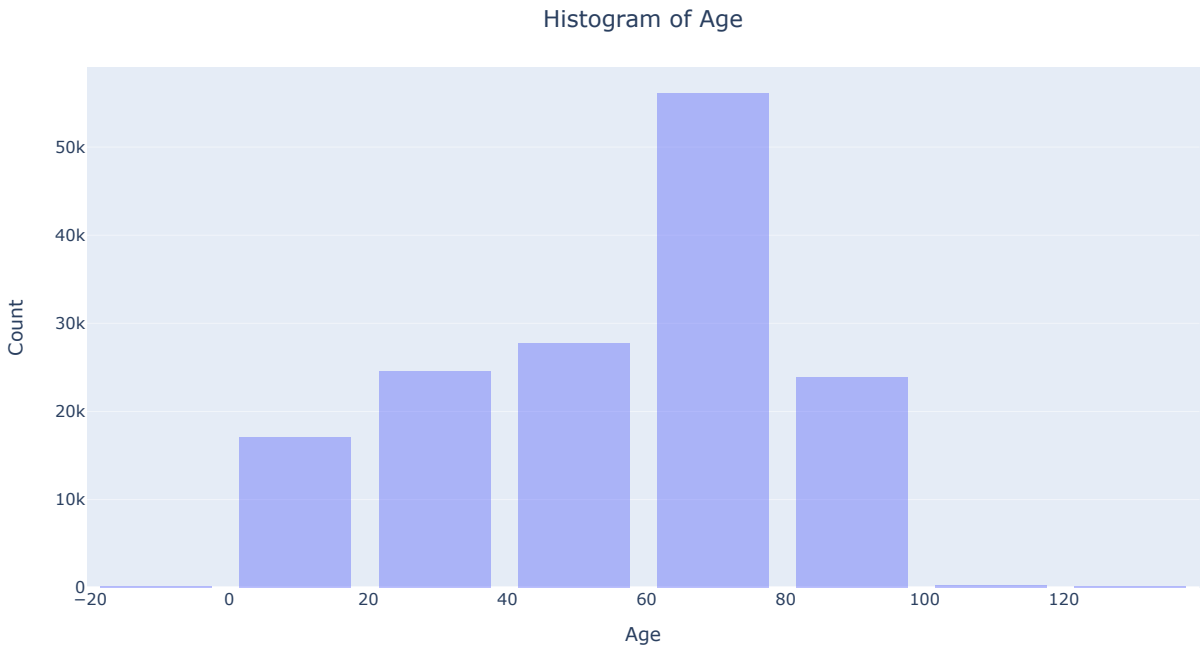
In [220]:
```
#Create new column for flaging patients with complication
df['with_cc_mcc']=((df['drg_description'].str.contains('WITH MCC')== True) | (df['drg_description'].str.contains('WITH CC/
```

In [221]:
```
#Create new column for flagging patient using ventilator
df['with_vent']=((df['drg_description'].str.contains('WITH MV')== True) | (df['drg_description'].str.contains('WITH VENTIL
```

In [222]:
```
df1=df[['gender_abbreviation','clinical_type','drg_code',
 'mdc','age_at_admit','with_cc_mcc','with_vent','return_visit','meet_it']]

df=df1.copy()
df.rename(columns={'gender_abbreviation':'gender','age_at_admit':'age','mdc':'mdc','with_cc_mcc':'cc_mc','with_vent':'vent
          ,inplace=True)

df['gender']=df.gender.map({'M':'M','U':'M','F':'F'})

bins = [-np.inf, 20, 40, 60, 70, 80, np.inf]
names = ['<20', '20_40', '40_60', '60_70', '70_80','80more']

df['age_bin'] = pd.cut(df['age'], bins, labels=names)
```

Histogram of Binned Age

In [262]:
```
def hist_bin(df,x,nbins,title,xaxis_title):
    # Create a histogram using Plotly with adjusted bar space
    fig = px.histogram(df1, x=x, nbins= nbins, barmode='overlay')

    # Update layout and axis labels
    fig.update_layout(title=title,
                      title_x=0.5,  # Set the title position to the center
                      xaxis_title=xaxis_title,
                      yaxis_title='Count',
                      bargap=0.2)  # Adjust the value to increase/decrease the bar space

    # Update aesthetic styles
    fig.update_traces(opacity=0.90)  # Change the bar color and opacity
    fig.update_layout( showlegend=False)  # Set background color and hide legend

    # Show the histogram
    fig.show()
hist_bin(df,x='age_at_admit',nbins=10,title='Histogram of Age',xaxis_title='Age')
```

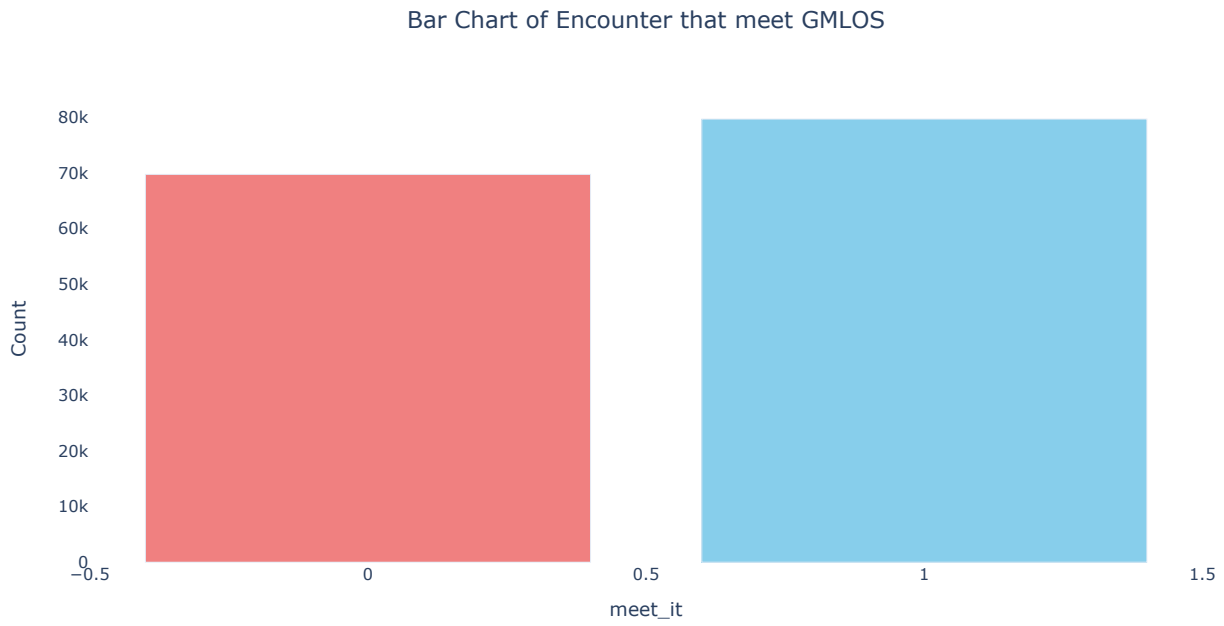


In [ ]:
```
1
```

Bar chart of proportion of encounters that meet it or not (1 or 0)

In [277]:
```
def barchart(df,target,subject):
    # Count the number of 1s and 0s in the 'meet_it' column
    meet_it_counts = df1[target].value_counts()

    # Create a bar chart using Plotly
    fig = px.bar(x=meet_it_counts.index, y=meet_it_counts.values,
                 color=meet_it_counts.index, labels={'x': target, 'y': 'Count'},
                 title='Bar Chart of Encounter that {}'.format(subject))

    # Update aesthetic styles
    fig.update_traces(marker_color=['skyblue', 'lightcoral'])  # Set custom colors for the bars
    fig.update_layout(plot_bgcolor='white', title_x=0.5, showlegend=False)  # Set background color and hide legend
    fig.update_xaxes(title_text=target)  # Set x-axis label

    # Show the bar chart
    fig.show()
barchart(df1,'meet_it','meet GMLOS')
```

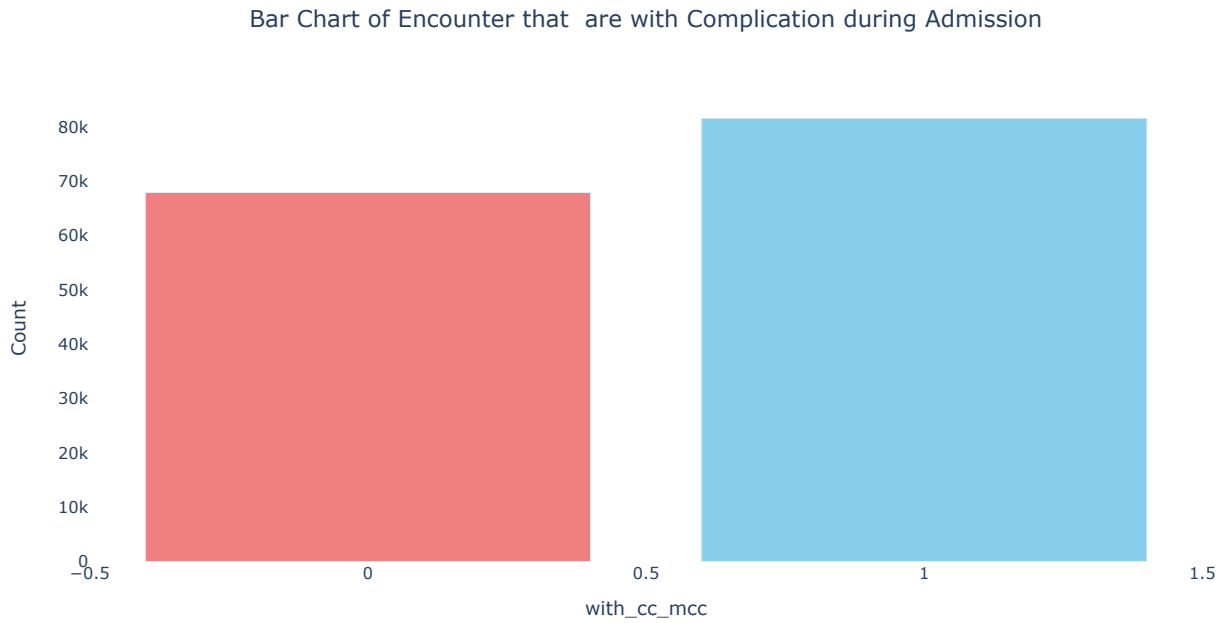
```
C:\ProgramData\Anaconda3\lib\site-packages\numpy\core\numeric.py:2327: FutureWarning:

elementwise comparison failed; returning scalar instead, but in the future will perform elementwise comparison
```



In [ ]:
```

```

Bar chart of proportion of encounters that has complications or not (1 or 0)

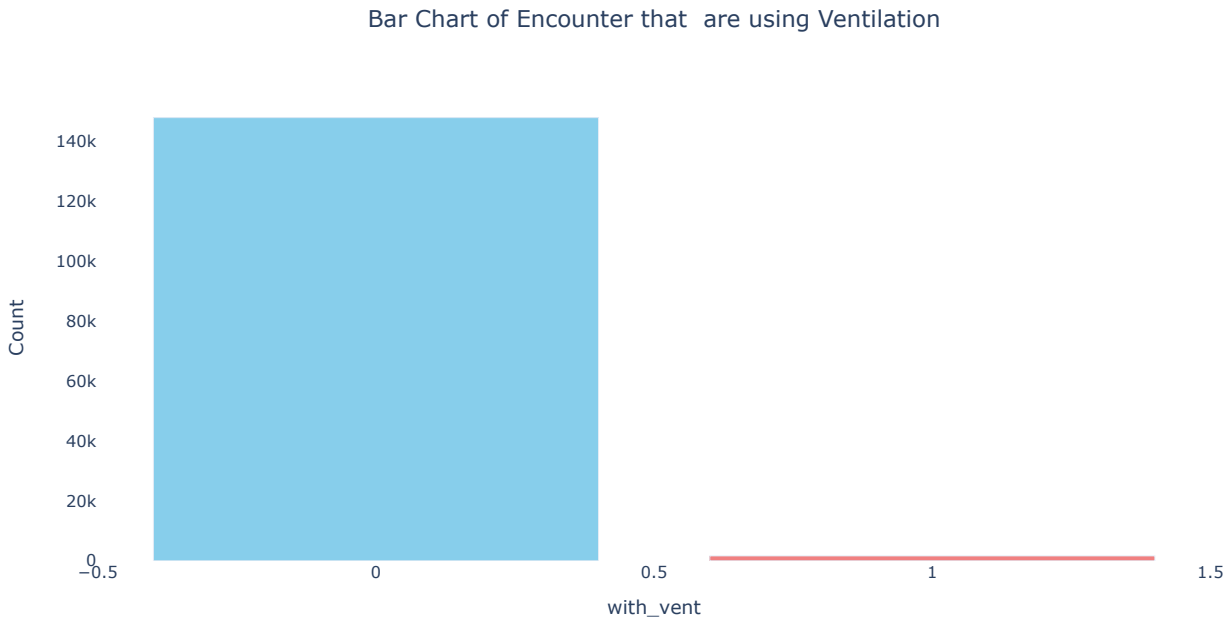
In [279]:
```
1 barchart(df1,'with_cc_mcc', " are with Complication during Admission")
```



Bar chart of proportion of encounters that use ventilator or not (1 or 0)

In [281]:
```
1 barchart(df1,'with_vent', " are using Ventilation")
```

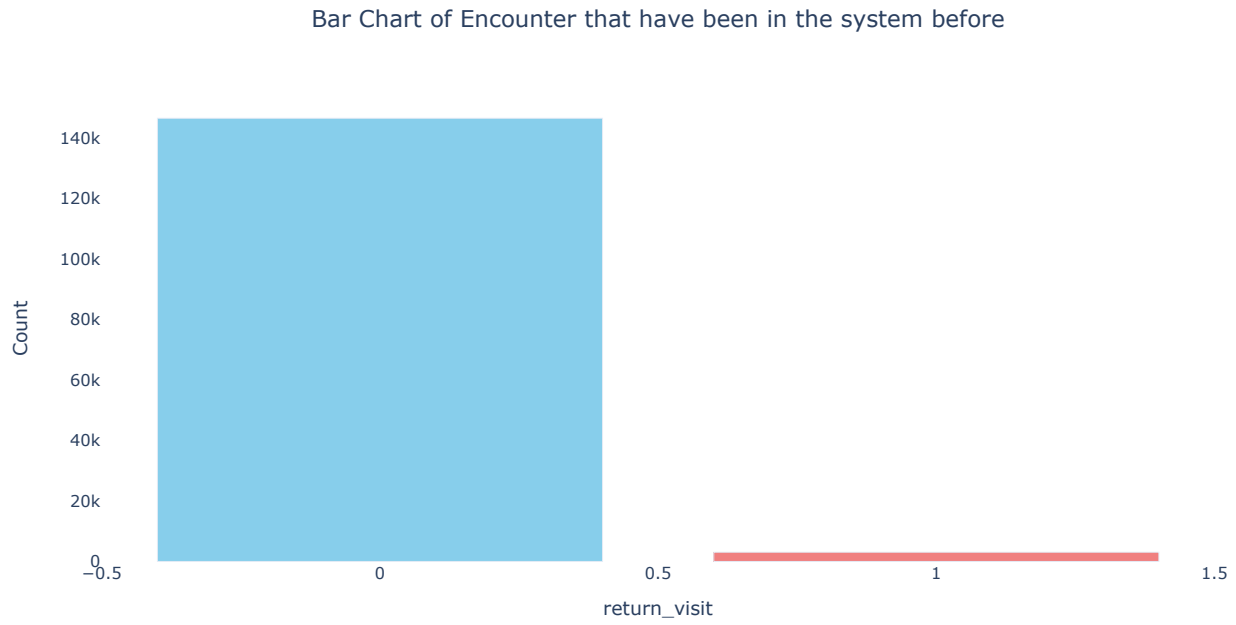
```
C:\ProgramData\Anaconda3\lib\site-packages\numpy\core\numeric.py:2327: FutureWarning:

elementwise comparison failed; returning scalar instead, but in the future will perform elementwise comparison
```



In [ ]:
```
1
```

Bar chart of proportion of encounters that return to the hospital or not (1 or 0)

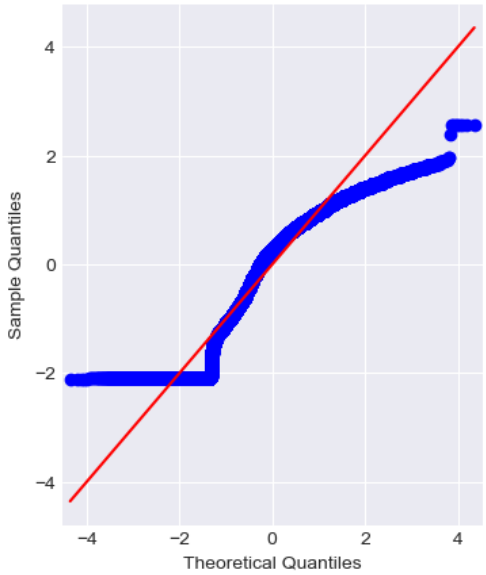
In [282]:
```
barchart(df1,'return_visit', 'have been in the system before')
```



Check for normality in age distribution

As we have known by now , there is a violation of normality for the distribution of age . A lot of outliers present

In [244]:
```
# check normality
fig, ax = plt.subplots(figsize=(4,5))
qqplot(df1['age_at_admit'], fit=True, line='s',ax=ax)
plt.show()
```



Modelling

```
Used a variety of statistical and machine learning models to assess the data, with the goal of building an accurate model
that could predict whether a patient will meet or not meet the GMLOS. Some of these models gave a likelihood to meet the
GMLOS. In all of the models, we were attempting to predict for each encounter a binary variable that represented whether
the encounter met the GMLOS or not.
```

```
We first tried logistic regression. This model was 59.1% accurate. We tried another logistic regression but removed any patients with an admit age of 0 or below, just to avoid the extreme violation of normality, to remove newborn patients, which exhibited different behavior than the rest of our patients. This model was less accurate than the first. Then tried a standard decision tree model on the full data, which was 59.26% accurate, and a gradient boosted classifier, which was 59.48% accurate.Finally tried a random forest model which returned our most accurate model by a thin margin at 59.5%. For reference, 53.23% should be used as a benchmark as this is what percentage of our data met the GMLOS. It can be seen that these models performed marginally better than a blanket guess of “Yes”, meaning that their interpretation could provide some value, but may not be extremely accurate as a predictor at admission for a patient as there is significant lack of fit .
```

Grouping the Categories

In [ ]:
```
df['mdc']=df.mdc.astype('category')
df['gender']=df.gender.astype('category')
df['clinical_type']=df.clinical_type.astype('category')
df['drg_code']=df.drg_code.astype('category')

df1=df.copy()
df.drop(columns=['drg_code'],inplace=True)
```

Creating Dummies

In [227]:
```
dummies_age_bin = pd.get_dummies(df['age_bin'])
dummies_age_bin.drop(columns='<20',inplace=True)

dummies_mdc = pd.get_dummies(df['mdc'],prefix='mdc')
dummies_mdc.drop(columns='mdc_00',inplace=True)

dummies_clinical_type = pd.get_dummies(df['clinical_type'])
dummies_clinical_type.drop(columns='SURG',inplace=True)

dummies_gender = pd.get_dummies(df['gender'])
dummies_gender.drop(columns='M',inplace=True)

categorical_features = ['clinical_type', 'gender','mdc', 'age_bin']
continuous_features = ['age']
binary_features =['return_visit', 'cc_mc', 'vent']
target =['meet_it']
target=df[target]
```

Final dataframe

In [228]:
```
data = pd.concat([df[binary_features], dummies_gender], axis=1)
data_1 = pd.concat([data, dummies_clinical_type], axis=1)
data_2 = pd.concat([data_1, dummies_mdc], axis=1)
data_3 = pd.concat([data_2, dummies_age_bin], axis=1)
df = pd.concat([data_3, target], axis=1)

```

In [258]:
```
df
```

Out[258]:

	return_visit	cc_mc	vent	F	MED	mdc_01	mdc_02	mdc_03	mdc_04	mdc_05	mdc_06	mdc_07	mdc_08	mdc_09	mdc_10	mdc_11	mdc_12	m
0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	
1	0	1	0	1	1	0	0	0	0	0	0	1	0	0	0	0	0	
2	0	1	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	
3	0	1	0	1	1	0	0	0	0	0	0	1	0	0	0	0	0	
4	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
149520	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	
149521	0	1	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	
149522	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	
149523	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	
149524	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	

149525 rows × 37 columns

In [ ]:
```

```

In [283]:
```
# rename columns
df.rename(columns={'20-40':'20_40','40-60':'40_60','60-70':'60_70','70-80':'70_80','80+':'80more'})
```

Out[283]:

	return_visit	cc_mc	vent	F	MED	mdc_01	mdc_02	mdc_03	mdc_04	mdc_05	mdc_06	mdc_07	mdc_08	mdc_09	mdc_10	mdc_11	mdc_12	mc
0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	
1	0	1	0	1	1	0	0	0	0	0	0	1	0	0	0	0	0	
2	0	1	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	
3	0	1	0	1	1	0	0	0	0	0	0	1	0	0	0	0	0	
4	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
149520	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	
149521	0	1	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	
149522	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	
149523	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	
149524	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	

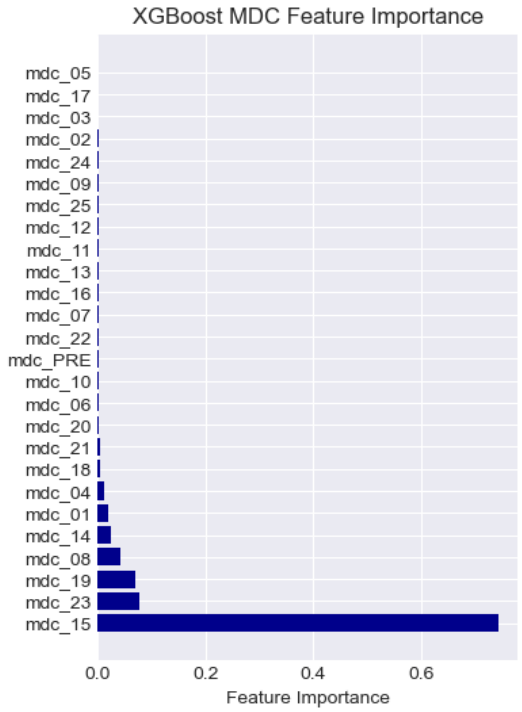
149525 rows × 37 columns

In [ ]:
```
1
```

Check for feature Importance

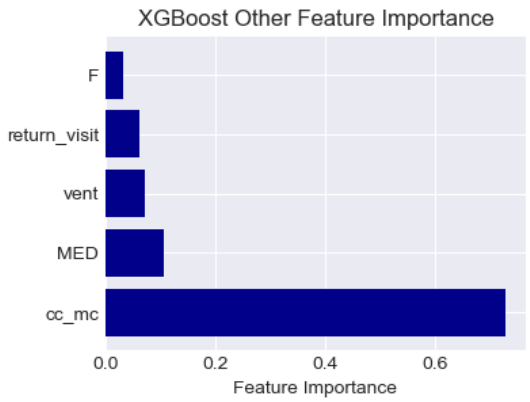
The features were grouped into different sects and then checked their importance using XGbosst feature importance capability. The Features was latter combined and checked for the top 10 most important feature in the data

In [287]:
```
# plot feature importance manually
from numpy import loadtxt
from xgboost import XGBClassifier
import matplotlib.pyplot as plt




#### MDC features

features_mdc=['mdc_01', 'mdc_02',
       'mdc_03', 'mdc_04', 'mdc_05', 'mdc_06', 'mdc_07', 'mdc_08', 'mdc_09',
       'mdc_10', 'mdc_11', 'mdc_12', 'mdc_13', 'mdc_14', 'mdc_15', 'mdc_16',
       'mdc_17', 'mdc_18', 'mdc_19', 'mdc_20', 'mdc_21', 'mdc_22', 'mdc_23',
       'mdc_24', 'mdc_25', 'mdc_PRE',
       ]
target =['meet_it']

X = df[features_mdc]
y = df[target]

# fit model on training data
model = XGBClassifier()
model.fit(X, y)

# feature importance
feature_importance = model.feature_importances_

# sort feature importance in descending order
sorted_idx = feature_importance.argsort()[::-1]
sorted_features = [features_mdc[i] for i in sorted_idx]
sorted_importance = feature_importance[sorted_idx]

# create a horizontal bar plot
fig, ax = plt.subplots(figsize=(4, 6))
ax.barh(sorted_features, sorted_importance, color='darkblue')
ax.set_xlabel('Feature Importance')
ax.set_title('XGBoost MDC Feature Importance')

plt.show()
#plt.savefig('MDC_feat_import.png',dpi=360)

```



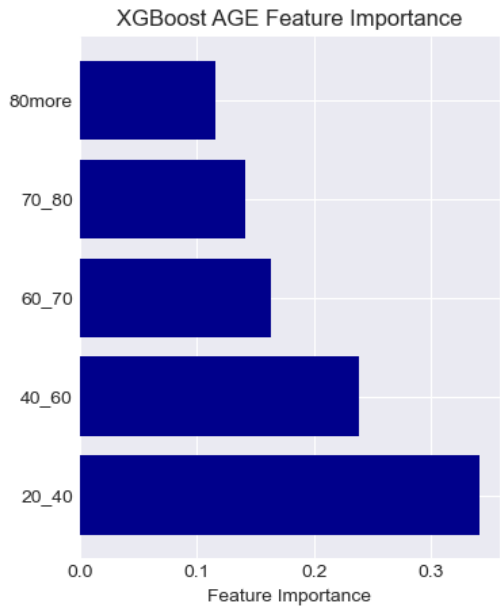
In [321]:
```
#### Other feature group
```

In [288]:
```
features_others=['return_visit', 'cc_mc', 'vent', 'F', 'MED']


X = df[features_others]
y = df[target]

# fit model on training data
model = XGBClassifier()
model.fit(X, y)

# feature importance
feature_importance = model.feature_importances_

# sort feature importance in descending order
sorted_idx = feature_importance.argsort()[::-1]
sorted_features = [features_others[i] for i in sorted_idx]
sorted_importance = feature_importance[sorted_idx]

# create a horizontal bar plot
fig, ax = plt.subplots(figsize=(4, 3))
ax.barh(sorted_features, sorted_importance, color='darkblue')
ax.set_xlabel('Feature Importance')
ax.set_title('XGBoost Other Feature Importance')

plt.show()
#plt.savefig('other_feat_import.png',dpi=360)
```



In [322]:
```
#### Age feature group
```

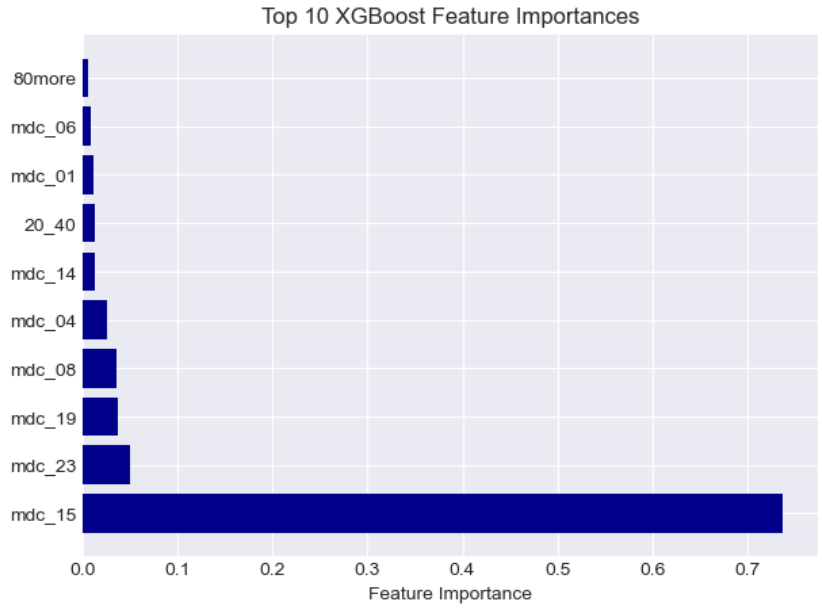
In [290]:
```
features_age=['20_40', '40_60', '60_70', '70_80', '80more']


X = df[features_age]
y = df[target]

# fit model on training data
model = XGBClassifier()
model.fit(X, y)

# feature importance
feature_importance = model.feature_importances_

# sort feature importance in descending order
sorted_idx = feature_importance.argsort()[::-1]
sorted_features = [features_age[i] for i in sorted_idx]
sorted_importance = feature_importance[sorted_idx]

# create a horizontal bar plot
fig, ax = plt.subplots(figsize=(4, 5))
ax.barh(sorted_features, sorted_importance, color='darkblue')
ax.set_xlabel('Feature Importance')
ax.set_title('XGBoost AGE Feature Importance')

plt.show()
#plt.savefig('age_feat_import.png',dpi=360)
```



In [ ]:
```

```

All features combined

In [291]:
```
features=['return_visit', 'cc_mc', 'vent', 'F', 'MED', 'mdc_01', 'mdc_02',
       'mdc_03', 'mdc_04', 'mdc_05', 'mdc_06', 'mdc_07', 'mdc_08', 'mdc_09',
       'mdc_10', 'mdc_11', 'mdc_12', 'mdc_13', 'mdc_14', 'mdc_15', 'mdc_16',
       'mdc_17', 'mdc_18', 'mdc_19', 'mdc_20', 'mdc_21', 'mdc_22', 'mdc_23',
       'mdc_24', 'mdc_25', 'mdc_PRE', '20_40', '40_60', '60_70', '70_80',
       '80more']
target =['meet_it']

df[features]

X = df[features]
y = df[target]

# fit model on training data
model = XGBClassifier()
model.fit(X, y)

# feature importance
feature_importance = model.feature_importances_

# sort feature importance in descending order
sorted_idx = feature_importance.argsort()[::-1]
sorted_features = [features[i] for i in sorted_idx]
sorted_importance = feature_importance[sorted_idx]

# create a horizontal bar plot of the top 10 feature importances
top_k = 10
fig, ax = plt.subplots(figsize=(7, 5))
ax.barh(sorted_features[:top_k], sorted_importance[:top_k], color='darkblue')
ax.set_xlabel('Feature Importance')
ax.set_title(f'Top {top_k} XGBoost Feature Importances')

plt.show()
#plt.savefig('Top_10_allfeat_import.png',dpi=360)
```



Model Building

In [72]:
```
from sklearn.model_selection import train_test_split
from sklearn import tree

X = df[features].copy()
y = df[target].copy()

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.30, random_state=0)
```

In [106]:
```
## Decision Tree

clf = tree.DecisionTreeClassifier(random_state=0,)
clf = clf.fit(X_train, y_train)
clf.score(X_train, y_train)
```

Out[106]: 0.6077830008407857

In [107]:
```
clf.score(X_test, y_test)
```

Out[107]:
```
0.5926074549670055
```

Random Forest

In [123]:
```
from sklearn.ensemble import RandomForestClassifier

#y_train=y_train.meet_it.ravel()
clf = RandomForestClassifier(random_state=70,max_depth=13)
clf = clf.fit(X_train, y_train)
clf.score(X_train, y_train)
```

Out[123]:
```
0.6037701597492929
```

In [124]:
```
clf.score(X_test, y_test)
```

Out[124]:
```
0.5954164437310505
```

Logistic Regression

In [131]:
```
from sklearn.linear_model import LogisticRegression
from sklearn.preprocessing import StandardScaler

X = df[features].copy()
y = df[target].copy()

scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

y_train = y_train.meet_it.ravel()
y_test = y_test.meet_it.ravel()

logisticRegr = LogisticRegression(max_iter=1000)
logisticRegr.fit(X_train, y_train)

predictions = logisticRegr.predict(X_test)

```

In [132]:
```
# Use score method to get accuracy of model
score = logisticRegr.score(X_test, y_test)
print(score)
```

```
0.5911137863385054
```

In [133]:
```
from sklearn.metrics import confusion_matrix
y_true = y_test
y_pred = predictions
confusion_matrix(y_true, y_pred)
```

Out[133]:
```
array([[11080,  9906],
       [ 8435, 15435]], dtype=int64)
```

In [135]:
```
# lst=[[11107,  9879],
#        [ 8468, 15402]]
lst=confusion_matrix(y_true, y_pred).tolist()
lst
```

Out[135]:
```
[[11080, 9906], [8435, 15435]]
```

In [140]:
```
tot_P=(y_test==1).sum()
tot_N=(y_test==0).sum()
true_Neg = lst[0][0]/tot_N
false_Neg = lst[0][1]/tot_N
false_Pos = lst[1][0]/tot_P
true_Pos = lst[1][1]/tot_P
print(f'true positive rate is: {true_Pos},false positive rate is: {false_Pos}')
print(f'true negative rate is: {true_Neg}, false negative rate is: {false_Neg}')

```

```
true positive rate is: 0.6466275659824047,false positive rate is: 0.3533724340175953
true negative rate is: 0.527971028304584, false negative rate is: 0.47202897169541597
```

In [151]:
```
precision=(true_Pos/(true_Pos+false_Pos))
```

In [155]:
```
recall=(true_Pos/(true_Pos+false_Neg))
```

In [156]:
```
f1=2*(precision*recall/(precision+recall))
```

In [157]:
```
f1
```

Out[157]:
```
0.6104128295293665
```

XGBoost

In [143]:
```
from numpy import loadtxt
from xgboost import XGBClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
```

In [144]:
```
X = df[features]
Y = df[target]

```

In [145]:
```
# split data into train and test sets
seed = 7
test_size = 0.33
X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=test_size, random_state=seed)

```

Train Model

In [146]:
```

# fit model no training data
model = XGBClassifier()
model.fit(X_train, y_train)
```

Out[146]:
```
XGBClassifier(base_score=None, booster=None, callbacks=None,
              colsample_bylevel=None, colsample_bynode=None,
              colsample_bytree=None, early_stopping_rounds=None,
              enable_categorical=False, eval_metric=None, feature_types=None,
              gamma=None, gpu_id=None, grow_policy=None, importance_type=None,
              interaction_constraints=None, learning_rate=None, max_bin=None,
              max_cat_threshold=None, max_cat_to_onehot=None,
              max_delta_step=None, max_depth=None, max_leaves=None,
              min_child_weight=None, missing=nan, monotone_constraints=None,
              n_estimators=100, n_jobs=None, num_parallel_tree=None,
              predictor=None, random_state=None, ...)
```

In [147]:
```

# make predictions for test data
y_pred = model.predict(X_test)
predictions = [round(value) for value in y_pred]
```

In [148]:
```

# evaluate predictions
accuracy = accuracy_score(y_test, predictions)
print("Accuracy: %.2f%%" % (accuracy * 100.0))
```

```
Accuracy: 59.48%
```

In [ ]:
```

```

In [ ]:
```

```

Conclusion

In this work, it was found that in overall, some DRGs do better at meeting these expected GMLOS than others. We found that DRGs with a high expected length of stay are less likely to be met than those with short stays.

insights were given into what the data is made up,which will help healthcare professionals better understand which patients overstay their GMLOS. The visualizations allow for detailed investigation of how different aspects of a patient and their condition contribute to their ability to meet the DRG-defined GMLOS. The machine earning models, can be used to essentially triage new patients that may require more attention to meet the GMLOS. These insights will provide value to both the patient and hospital

In [ ]:
```

```