# Question 1

Python code :

*#------------------------------------------------------------------------------------------------------------------  
# Question1 a  
#------------------------------------------------------------------------------------------------------------------*print("-----Question1 a------")  
integer1 = int(80)  
print("integer 1 is ",integer1)  
print("type of integer 1 is ",type(integer1))  
mult\_by = int(200)  
dvyd\_by = int(40)  
var2 = integer1\*mult\_by  
var3 = integer1//dvyd\_by  
var4 = integer1\*mult\_by//dvyd\_by  
print("Var2 = integer1\* 200 :" ,var2,"with type",type(var2), "Var3 integer1/ 40:" ,var3, "with type", type(var3))  
print("Var4 = integer1 \* 200 / 40:" ,var4,"with type",type(var4))  
print("-----------------")  
*#------------------------------------------------------------------------------------------------------------------  
  
#------------------------------------------------------------------------------------------------------------------  
# Question1 b  
#------------------------------------------------------------------------------------------------------------------*print("-----Question1 b------")  
float\_var = float(100.0)  
print("Variable float\_var :", float\_var,"has Type :", type(float\_var))  
float\_var = float\_var \*\* 0.5  
print( "after square root it is:", float\_var)  
print("-----------------")  
*#------------------------------------------------------------------------------------------------------------------  
  
#------------------------------------------------------------------------------------------------------------------  
# Question1 c  
#------------------------------------------------------------------------------------------------------------------*print("-----Question1 c------")  
strng = "HELLO WORLD"  
print("original string->", strng)  
strng = strng.lower()  
print("which has Type :", type(strng), "after lower case:", strng)  
print("-----------------")  
*#------------------------------------------------------------------------------------------------------------------  
  
#------------------------------------------------------------------------------------------------------------------  
# Question1 d  
#------------------------------------------------------------------------------------------------------------------*print("-----Question1 d------")  
my\_tuple = (1, -2, -3, -4, -5)  
max\_from\_tuple = max(my\_tuple)  
print("my tuple has values : ", my\_tuple, "and type", type(my\_tuple), " and max number from my tuple is ", max\_from\_tuple  
 ," and max\_from\_tuple has type of :", type(max\_from\_tuple))  
print("-----------------")  
*#------------------------------------------------------------------------------------------------------------------  
  
#------------------------------------------------------------------------------------------------------------------  
# Question1 e  
#------------------------------------------------------------------------------------------------------------------*print("-----Question1 e------")  
mylist = [0,1,2,3,4,5,6]  
third\_position = mylist[2] *#third position has index 2*print("my list is :",mylist, "with type of :",type(mylist) ,"third position is :",third\_position, " with type",  
 type(third\_position))  
print("-----------------")  
*#------------------------------------------------------------------------------------------------------------------  
  
#------------------------------------------------------------------------------------------------------------------  
# Question1 f  
#------------------------------------------------------------------------------------------------------------------*print("-----Question1 f------")  
mydict = {"key1": 1, "key2": 2}  
print("my dict is ->", mydict, "with type :", type(mydict))  
mydict["key3"] = 3  
print("my dict after adding key3:", mydict, "and key3 has type :", type(mydict["key3"]))  
print("-----------------")  
*#------------------------------------------------------------------------------------------------------------------*

Pycharm Screenshot of output :

Graphical user interface, text, application, email

Description automatically generated

# Question 2

Python code :

import pandas as pd  
import matplotlib.pyplot as plt  
  
desired\_width = 320  
pd.set\_option('display.width', desired\_width)  
pd.set\_option('display.max\_columns', 10)  
df = pd.read\_csv('SchData2013.csv', low\_memory=False)  
*#---------------------*print(df.shape)  
print(df.info())  
print(df.describe())  
*#---------------------  
  
#---------------------  
  
# Fill missing values in int64 columns with the mean value for that column*int64\_columns = df.select\_dtypes(include=['int64']).columns  
df[int64\_columns] = df[int64\_columns].fillna(df.mean())  
  
*# Fill missing values in float64 columns with the mean value for that column*float64\_columns = df.select\_dtypes(include=['float64']).columns  
df[float64\_columns] = df[float64\_columns].fillna(df.mean())  
  
print(df['PatientInRoomMin'].isnull().sum())  
print(df['SchCreateDays'].isnull().sum())  
*#---------------------  
  
  
# #--to be run in the end as we are dropping categorical data---------------  
 df = df.drop('CaseMonth', axis=1)  
 df = df.dropna()  
 cat\_cols = [col for col in df.columns if df[col].dtype == 'object']  
 df = df.drop(cat\_cols, axis=1)  
 df = df.reset\_index(drop=True)  
 print(df.info())  
# #---- to be run in the end as we are dropping categorical data -----------*

diced\_df = df[(df["SurgicalArea"] == "HHOR") & (df["SSSNo"] == 9) & (df["PatientType"] == 2)]  
  
*# calculate the mean and median of PatientInRoomMin for the diced dataframe*mean = diced\_df["PatientInRoomMin"].mean()  
median = diced\_df["PatientInRoomMin"].median()  
  
print(diced\_df.head(10))  
print("Mean of PatientInRoomMin: {:.2f}".format(mean))  
print("Median of PatientInRoomMin: {:.2f}".format(median))  
  
*# create a boxplot for PatientInRoomMin*diced\_df.boxplot(column=["PatientInRoomMin"])  
plt.show()  
  
*# calculate the IQR and remove outliers*Q1 = diced\_df["PatientInRoomMin"].quantile(0.25)  
Q3 = diced\_df["PatientInRoomMin"].quantile(0.75)  
IQR = Q3 - Q1  
lower\_bound = Q1 - 1.5\*IQR  
upper\_bound = Q3 + 1.5\*IQR  
diced\_df = diced\_df[(diced\_df["PatientInRoomMin"] >= lower\_bound) & (diced\_df["PatientInRoomMin"] <= upper\_bound)]  
  
*# create a boxplot for PatientInRoomMin after removing outliers*diced\_df.boxplot(column=["PatientInRoomMin"])  
plt.show()

Output :

Graphical user interface, text

Description automatically generated

Table

Description automatically generated

Proof that null data replaced in data types int64 and float64

Text

Description automatically generated

Deleting categorical data

Graphical user interface, text

Description automatically generated

Table

Description automatically generated with medium confidence

Text

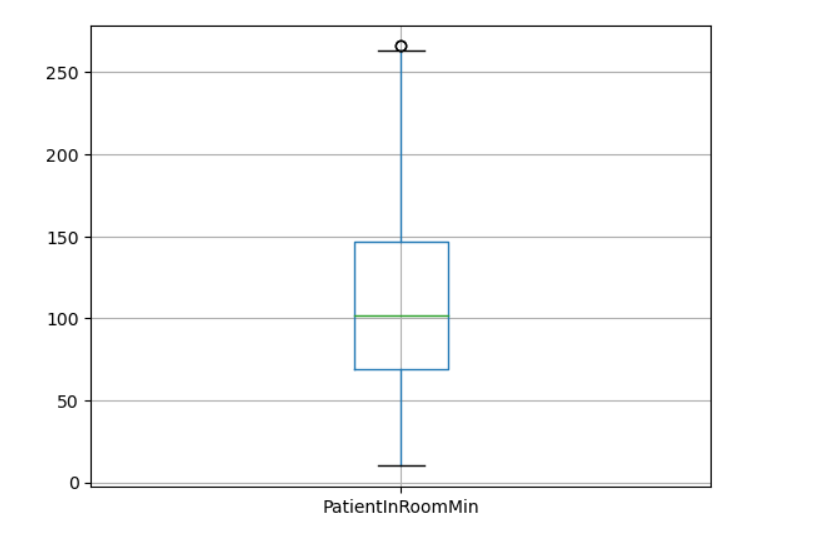
Description automatically generated with low confidence

Before reducing outliers

Chart, box and whisker chart

Description automatically generated

After reducing outliers :



# Question3:

Python Code :

import pandas as pd  
import matplotlib.pyplot as plt  
  
desired\_width = 400  
pd.set\_option('display.width', desired\_width)  
pd.set\_option('display.max\_columns', 10)  
df = pd.read\_csv('SchData2013.csv', low\_memory=False)  
print(df.info())  
  
df2 = df.groupby('SurgicalSpecialty')['ScheduledCaseDuration'].median().reset\_index()  
df2.columns = ['SurgicalSpecialty', 'ScheduledCaseDurationMedian']  
  
merged\_df = pd.merge(df, df2, on='SurgicalSpecialty')  
filtered\_df = merged\_df[(merged\_df['SurgicalSpecialty'] == 'URO') & (merged\_df['TotalSurgeryMin'] <= 250)]  
  
print("Mean of TotalSurgeryMin:", filtered\_df['TotalSurgeryMin'].mean())  
print("Median of TotalSurgeryMin:", filtered\_df['TotalSurgeryMin'].median())  
  
plt.boxplot(filtered\_df['TotalSurgeryMin'])  
plt.title("Boxplot of TotalSurgeryMin")  
plt.show()

Output:

Table

Description automatically generated

Graphical user interface, text, application

Description automatically generated

Chart, box and whisker chart

Description automatically generated

# Question 4

Python code :

import pandas as pd  
import matplotlib.pyplot as plt  
from sklearn.model\_selection import train\_test\_split  
from sklearn.linear\_model import LinearRegression  
from sklearn.metrics import r2\_score, mean\_absolute\_error, mean\_squared\_error  
  
desired\_width = 320  
pd.set\_option('display.width', desired\_width)  
pd.set\_option('display.max\_columns', 18)  
df = pd.read\_csv('SchData2013.csv', low\_memory=False)  
print(df.info())  
  
*# Identify int64 and float64 variables*int\_vars = df.select\_dtypes(include=['int64']).columns  
float\_vars = df.select\_dtypes(include=['float64']).columns  
print('Int Variables:', int\_vars)  
print('Float Variables:', float\_vars)  
  
*# Identify all numeric columns*numeric\_cols = df.select\_dtypes(include=['int64', 'float64']).columns  
*#AND  
# Replace missing values with column mean*df[numeric\_cols] = df[numeric\_cols].fillna(df[numeric\_cols].mean())  
  
*#verifying if there is no null values in any columns*print(df.isnull().sum())  
  
*# Slice X*X = df.loc[:, ['SSSNo', 'PatientType', 'SchedulePriority', 'PatientAge', 'ScheduledCaseDuration', 'TotalSurgeryMin', 'PatientInRoomMin', 'SetUpMin']]  
  
*# Slice y*y = df.loc[:, ['SchCreateDays']]  
  
*# Split X and y into training and test data subsets*X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)  
  
*# Fit a linear regression model on the training data*reg = LinearRegression().fit(X\_train, y\_train)  
  
*# Predict y values for test data*y\_pred = reg.predict(X\_test)  
  
*# Calculate the R2 score, MAE, MSE, and RMSE*r2 = r2\_score(y\_test, y\_pred)  
mae = mean\_absolute\_error(y\_test, y\_pred)  
mse = mean\_squared\_error(y\_test, y\_pred)  
rmse = mean\_squared\_error(y\_test, y\_pred, squared=False)  
  
print("R2 Score:", r2)  
print("Mean Absolute Error (MAE):", mae)  
print("Mean Squared Error (MSE):", mse)  
print("Root Mean Squared Error (RMSE):", rmse)  
  
*# Scatterplot of X variables and y*for col in X.columns:  
 fig, ax = plt.subplots()  
 ax.scatter(X\_test[col], y\_test)  
 ax.set\_xlabel(col)  
 ax.set\_ylabel('SchCreateDays')  
 plt.show()

output:

Table

Description automatically generated

Graphical user interface, text

Description automatically generated

Graphical user interface, application

Description automatically generated

# Chart, scatter chart Description automatically generated

Chart

Description automatically generated

Chart

Description automatically generated

Chart

Description automatically generated

Chart, scatter chart

Description automatically generated

Chart, scatter chart

Description automatically generated

Chart, scatter chart

Description automatically generated

Chart, scatter chart

Description automatically generated

Chart, scatter chart

Description automatically generated

# Question 5

The method I chose, which is K Nearest Neighbors (KNN) regression, is useful for this dataset because it is a non-parametric method that can handle both numerical and categorical data, K Nearest Neighbors (KNN) regression is a good choice for this dataset.

Python code :

import pandas as pd  
from sklearn.model\_selection import train\_test\_split  
from sklearn.linear\_model import LinearRegression  
from sklearn.metrics import r2\_score, mean\_absolute\_error, mean\_squared\_error  
from sklearn.cluster import KMeans  
from sklearn.metrics import adjusted\_rand\_score  
from sklearn.neighbors import KNeighborsRegressor  
  
  
desired\_width = 320  
pd.set\_option('display.width', desired\_width)  
pd.set\_option('display.max\_columns', 18)  
df = pd.read\_csv('SchData2013.csv', low\_memory=False)  
print(df.info())  
  
*# Identify int64 and float64 variables*int\_vars = df.select\_dtypes(include=['int64']).columns  
float\_vars = df.select\_dtypes(include=['float64']).columns  
print('Int Variables:', int\_vars)  
print('Float Variables:', float\_vars)  
*# Identify all numeric columns*numeric\_cols = df.select\_dtypes(include=['int64', 'float64']).columns  
*#AND  
# Replace missing values with column mean*df[numeric\_cols] = df[numeric\_cols].fillna(df[numeric\_cols].mean())  
*#verifying if there is no null values in any columns*print(df.isnull().sum())  
df = df.drop('CaseMonth', axis=1)  
df = df.dropna()  
  
*#----------Linear Regression-------*num\_cols = ['SSSNo', 'PatientType', 'SchedulePriority', 'PatientAge', 'ScheduledCaseDuration',  
 'TotalSurgeryMin', 'PatientInRoomMin', 'SetUpMin', 'SchCreateDays']  
data = df[num\_cols]  
X\_train, X\_test, y\_train, y\_test = train\_test\_split(data.iloc[:, :-1], data.iloc[:, -1], test\_size=0.2, random\_state=42)  
*# Create a linear regression object*reg = LinearRegression()  
*# Fit the model using the training data*reg.fit(X\_train, y\_train)  
*# Make predictions on the test data*y\_pred = reg.predict(X\_test)  
print("R-squared:", r2\_score(y\_test, y\_pred))  
print("Mean Absolute Error:", mean\_absolute\_error(y\_test, y\_pred))  
print("Mean Squared Error:", mean\_squared\_error(y\_test, y\_pred))  
  
*#-------------K means clustering analysis---------  
# Split the dataset into training and testing sets*X = df[['SSSNo', 'PatientType', 'SchedulePriority', 'PatientAge', 'ScheduledCaseDuration', 'TotalSurgeryMin', 'PatientInRoomMin', 'SetUpMin']]  
y = df['SchCreateDays']  
X\_train2, X\_test2, y\_train2, y\_test2 = train\_test\_split(X, y, test\_size=0.2, random\_state=0)  
*# Apply KMeans clustering to the data*kmeans = KMeans(n\_clusters=3, random\_state=0)  
kmeans.fit(X\_train2)  
*# Predict the clusters for the test data*y\_pred = kmeans.predict(X\_test2)  
*# Evaluate the clustering performance using adjusted rand score*print('Adjusted Rand Score:', adjusted\_rand\_score(y\_test2, y\_pred))  
*#----------------------------------------------  
  
  
# Apply KNN Regression to the data -----------------*knn = KNeighborsRegressor(n\_neighbors=5)  
knn.fit(X\_train, y\_train)  
*# Predict the values for the test data*y\_pred = knn.predict(X\_test)  
*# Evaluate the model performance using R2 score, mean absolute error, and root mean squared error*print('R2 Score:', r2\_score(y\_test, y\_pred))  
print('Mean Absolute Error:', mean\_absolute\_error(y\_test, y\_pred))  
print('Root Mean Squared Error:', mean\_squared\_error(y\_test, y\_pred, squared=False))  
*#----------------------------------------------*

Screenshot

Table

Description automatically generated

Graphical user interface, text, application, email

Description automatically generated

Graphical user interface

Description automatically generated with low confidence

# Question 6

Python Code :

*# sqlite import*import sqlite3  
  
*# establish connection to SQLite test.db*con = sqlite3.connect("HMIS.db")  
  
*# assign variable to con.cursor() function*cur = con.cursor()  
  
*# result of SQL script #1 execution iterated by row*for row in cur.execute(  
 """  
 SELECT Application || " --- " || Status AS App\_Stat\_Rural\_MA, COUNT(\*)  
 FROM (  
 SELECT Name, Application, Status, City, State, LENGTH(Name) AS name\_length, UPPER(State) AS state\_upper  
 FROM (  
 SELECT Name, Application, Status, City, State  
 FROM LEADS  
 WHERE State = "MA"  
 )  
 WHERE City NOT IN ("Boston", "Cambridge", "Lowell", "Springfield", "Worcester", "Newton")  
 ORDER BY City, Name  
 )  
 GROUP BY App\_Stat\_Rural\_MA, name\_length, state\_upper  
 ORDER BY COUNT(\*) DESC  
 LIMIT 10;  
"""  
):  
 *# print SQL projection row by row with the 'for loop'* print(row)  
*# close the database connection*con.close()  
*# using a user defined function to execute a SQL script*def exec\_sql():  
 *# print the SQL script to execute* print(SQL\_script)  
 *# assign cur to the connection.cursor() function* cur = con.cursor()  
 *# 'for loop' to receive SQL projection by row* for row in cur.execute(SQL\_script):  
 *# print row by row of projection* print(row)  
  
  
*# establishes connection to SQLite database (open db)*con = sqlite3.connect("test.db")  
SQL\_script = print(  
 """  
 SELECT Application || " --- " || Status AS App\_Stat\_Rural\_MA, COUNT(\*)  
 FROM (  
 SELECT Name, Application, Status, City, State, LENGTH(Name) AS name\_length, UPPER(State) AS state\_upper  
 FROM (  
 SELECT Name, Application, Status, City, State  
 FROM LEADS  
 WHERE State = "MA"  
 )  
 WHERE City NOT IN ("Boston", "Cambridge", "Lowell", "Springfield", "Worcester", "Newton")  
 ORDER BY City, Name  
 )  
 GROUP BY App\_Stat\_Rural\_MA, name\_length, state\_upper  
 ORDER BY COUNT(\*) DESC  
 LIMIT 10;  
""" )  
*# execute user defined function exec\_sql() for SQL script*exec\_sql()  
*# closes SQLite database connection*con.close()

Pycharm screenshot :

# Question 7

Python code :

import pandas as pd  
  
*# initialize variables for chunking*chunk\_size = 3000  
batch\_num = 1  
  
*# initialize CSV output file*csv\_out\_file = 'question7output.csv'  
  
for chunk in pd.read\_csv('SchData2013.csv', chunksize=chunk\_size, iterator=True):  
  
 *# filter for SurgicalArea and SurgicalSpecialty* filtered\_chunk = chunk[(chunk['SurgicalArea'] == 'CVOR') | (chunk['SurgicalSpecialty'] == 'CV')]  
  
 *# slice and dice the filtered chunk* sliced\_chunk = filtered\_chunk[['SurgicalArea', 'SurgicalSpecialty', 'SSSNo', 'PatientAge',  
 'TotalSurgeryMin', 'PrimaryProcedure', 'OperatingRoom']]  
  
 *# write the sliced and diced chunk to CSV* if batch\_num == 1:  
 sliced\_chunk.to\_csv(csv\_out\_file, header=True, index=False)  
 else:  
 sliced\_chunk.to\_csv(csv\_out\_file, mode='a', header=False, index=False)  
  
 print(f'Chunk {batch\_num} written to CSV. Type: {type(sliced\_chunk)}')  
 batch\_num += 1  
  
print(chunk.head())  
print(filtered\_chunk.head())  
print(sliced\_chunk.head())

screenshot :

Table

Description automatically generated

Table

Description automatically generated

Output csv

Text

Description automatically generated

# Question8

Python code :

import pandas as pd  
import seaborn as sns  
import matplotlib.pyplot as plt  
  
  
desired\_width = 320  
pd.set\_option('display.width', desired\_width)  
pd.set\_option('display.max\_columns', 18)  
df = pd.read\_csv('SchData2013.csv', low\_memory=False)  
print(df.info())  
  
*# Identify int64 and float64 variables*int\_vars = df.select\_dtypes(include=['int64']).columns  
float\_vars = df.select\_dtypes(include=['float64']).columns  
print('Int Variables:', int\_vars)  
print('Float Variables:', float\_vars)  
  
df = df.drop('CaseMonth', axis=1)  
df = df.dropna()  
*# Identify all numeric columns*numeric\_cols = df.select\_dtypes(include=['int64', 'float64']).columns  
*#AND  
# Replace missing values with column mean*df[numeric\_cols] = df[numeric\_cols].fillna(df[numeric\_cols].mean())  
*#verifying if there is no null values in any columns*print(df.isnull().sum())  
  
  
df2 = df[numeric\_cols]  
print(df2.info())  
  
corr\_matrix = df2.corr()  
print(corr\_matrix)  
  
*# Create a heatmap using Seaborn*sns.set(font\_scale=0.8)  
sns.heatmap(corr\_matrix, cmap="coolwarm", annot=True, annot\_kws={"size": 5})  
plt.title("Correlation Matrix Heatmap")  
plt.xticks(rotation=45)  
plt.yticks(rotation=0)  
plt.tight\_layout()  
  
*# Save the plot to disk*plt.savefig("correlation\_heatmap.png")

Output :

Graphical user interface, application, table

Description automatically generated

Graphical user interface

Description automatically generated with medium confidence

Table

Description automatically generated

Graphical user interface, table

Description automatically generated

Chart

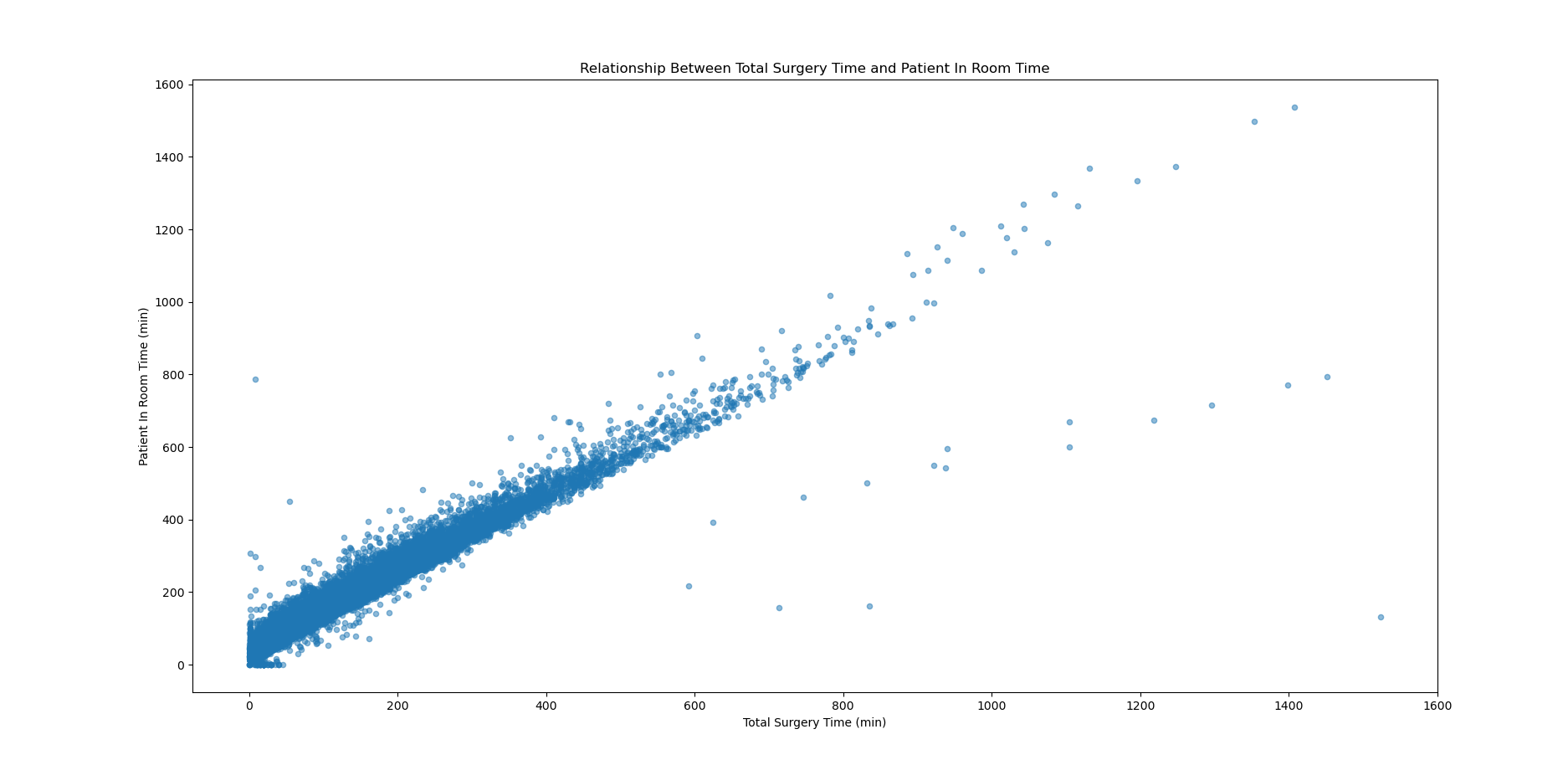
Description automatically generated

# Question 9

Python code :

import pandas as pd  
import seaborn as sns  
import matplotlib.pyplot as plt  
  
df = pd.read\_csv('SchData2013.csv', low\_memory=False)  
  
*# create a scatter plot of TotalSurgeryMin vs. PatientInRoomMin*print('-------------------')  
df.plot(kind='scatter', x='TotalSurgeryMin', y='PatientInRoomMin', alpha=0.5)  
plt.xlabel('Total Surgery Time (min)')  
plt.ylabel('Patient In Room Time (min)')  
plt.title('Relationship Between Total Surgery Time and Patient In Room Time')  
plt.show()  
  
*# create a bar chart of PatientType counts*df['PatientType'].value\_counts().plot(kind='bar')  
plt.xlabel('Patient Type')  
plt.ylabel('Count')  
plt.title('Number of Patients by Type')  
print('-------------------')  
plt.show()  
  
df2= df[(df.PatientAge >= 20) & (df.PatientAge <= 30)]  
sns.boxplot(df2.PatientAge).set\_title('Box Plot :Young Patient Operated')  
print('-------------------')  
plt.show()  
  
*# create a boxplot of ScheduledCaseDuration by SurgicalSpecialty and PatientType using Seaborn*sns.boxplot(x='SurgicalSpecialty', y='ScheduledCaseDuration', hue='PatientType', data=df)  
plt.xlabel('Surgical Specialty')  
plt.ylabel('Scheduled Case Duration (min)')  
plt.title('Scheduled Case Duration by Surgical Specialty and Patient Type')  
print('-------------------')  
plt.show()  
  
  
*# create a line plot of TotalSurgeryMin over time*df.plot(kind='line', x='CaseStartDate\_Time', y='TotalSurgeryMin')  
plt.xlabel('Case Start Date')  
plt.ylabel('Total Surgery Time (min)')  
plt.title('Total Surgery Time Over Time')  
print('-------------------')  
plt.show()  
  
*# Create a series of the counts of each surgical specialty*specialty\_counts = df['SurgicalSpecialty'].value\_counts()  
  
*# Create a pie chart with labels*plt.pie(specialty\_counts, labels=specialty\_counts.index, autopct='%1.1f%%')  
plt.title('Distribution of Surgical Specialties')  
plt.axis('equal')  
plt.show()  
  
*# Create a figure with 3 subplots*fig, axs = plt.subplots(1, 3, figsize=(12, 4))  
  
*# Plot the histograms for each column*axs[0].hist(df["PatientAge"], bins=20, color="orange")  
axs[0].set\_title("Patient Age")  
axs[0].set\_xlabel("Age (years)")  
axs[0].set\_ylabel("Frequency")  
  
axs[1].hist(df["ScheduledCaseDuration"], bins=20, color="purple")  
axs[1].set\_title("Scheduled Case Duration")  
axs[1].set\_xlabel("Duration (minutes)")  
axs[1].set\_ylabel("Frequency")  
  
axs[2].hist(df["TotalSurgeryMin"], bins=20, color="green")  
axs[2].set\_title("Total Surgery Time")  
axs[2].set\_xlabel("Time (minutes)")  
axs[2].set\_ylabel("Frequency")  
  
*# Adjust the layout and spacing between subplots*plt.tight\_layout()  
  
*# Display the plot*plt.show()

Output:



Chart, bar chart

Description automatically generated

Chart

Description automatically generated

Chart

Description automatically generated

Chart, histogram

Description automatically generated

Chart, pie chart

Description automatically generated

Chart, histogram

Description automatically generated