PYTHON CLEANING AND INSIGHTS SUMMARY

1. Importing the python libraries.

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
```

2. Load and merge all three datasets, handle missing values

Loading Treatment Records:

```
Treatment =
pd.read_csv(r"C:\Users\soura\Downloads\TreatmentRecords_Cleaned.csv",
parse_dates = ["Treatment_Date"])

Treatment.head()
```

Loading Doctor Details:

```
Doctors = pd.read_csv(r"C:\Users\soura\Downloads\DoctorDetails_Cleaned.csv")

Doctors.head()
```

Loading Patient Info:

```
Patients = pd.read_csv(r"C:\Users\soura\Downloads\PatientInfo_Cleaned.csv")

Patients.head()
```

Merging all 3 dataset:

```
df = (Treatment
    .merge(Doctors, on= "Doctor_ID", how= "left")
    .merge(Patients, on= "Patient_ID", how= "left") )

df
```

3. Identify duplicate patient IDs and remove them.¶

Checking the null counts

```
df.isnull().sum()
```

Confirming whether there is no nulls

```
Treatment[~Treatment["Doctor_ID"].isin(Doctors["Doctor_ID"])]

Treatment[~Treatment["Patient_ID"].isin(Patients["Patient_ID"])]
```

4. Finding Duplicate row counts

```
df.duplicated().sum()
```

confirming whether no duplicates exist

```
duplicates_rows = df[df.duplicated(keep=False)]

duplicates_rows
```

5. Create a bar chart showing patient count by department.

```
Patient_count_Dept = df.groupby("Specialty")["Patient_ID"].nunique().sort_values(ascending= False)
plt.figure(figsize=(8,6))
plt.bar(Patient_count_Dept.index, Patient_count_Dept.values, color= "Blue", width= 0.5)

font1 = {"family": "serif", "color": "Blue", "fontsize":20}
plt.ylabel("No of Patients", fontdict= font1, weight= "bold")
plt.xlabel("Departments", fontdict= font1, weight= "bold")
plt.title("Patient count by Department", loc= "center", weight= "bold")
plt.show()
```

6. Visualize average treatment duration by condition.¶

```
Avg_duration_condition = df.groupby("Disease")["Treatment_Duration_Days"].mean().sort_values(ascending= False)
plt.figure(figsize=(8,6))
plt.barh(Avg_duration_condition.index, Avg_duration_condition.values, color= "green", height= 0.4)
plt.ylabel("Condition", fontdict= font1, weight= "bold")
plt.xlabel("Treatment Duration (Days)", fontdict= font1, weight= "bold")
plt.title("Average Treatment Duration by Condition", loc= "center", weight= "bold")
plt.show()
```

7. Identify which age group has the highest readmission rate.

Creating age groups

```
df['Age_Group'] = pd.cut(
   df['Age'], bins= [0,17,59,120], labels= ['Child', 'Adult', 'Senior'])
```

Sorting by patient and date

```
df_sorted = df.sort_values(by= ['Patient_ID', 'Treatment_Date'])
```

Count number of admissions within 30 days for each patient

```
df_sorted['Prev_Admission'] =
df_sorted.groupby('Patient_ID')['Treatment_Date'].shift(1)
```

finding the day difference between current Treatment Date & Previous Treatment Date

```
df_sorted['Days_since_last_visit'] = (df_sorted['Treatment_Date'] -
df_sorted['Prev_Admission']).dt.days

df_sorted['Readmission'] = df_sorted['Days_since_last_visit'].apply(lambda x: 1 if x is
not pd.NaT and x <= 30 else 0)
```

calculating readmission rate per age group

```
Readmission_rate =
df_sorted.groupby('Age_Group')['Readmission'].mean().sort_values(ascending= False)
print(Readmission_rate)
```

8. Generate a correlation matrix for cost, duration, and satisfaction.

```
corr_cols = ['Treatment_Cost', 'Treatment_Duration_Days', 'Satisfaction_Score']
corr_matrix = df[corr_cols].corr()
corr_matrix

plt.figure(figsize=(6,4))
plt.imshow(corr_matrix, cmap='coolwarm', interpolation='nearest')
plt.colorbar()
plt.yticks(range(len(corr_matrix.columns)), corr_matrix.columns)
plt.title("Correlation Matrix")
plt.show()
```

9. Extract patients who had poor outcomes despite high costs.

```
avg_cost = df['Treatment_Cost'].mean()
AvgCost_PoorOut = df[(df['Treatment_Cost'] > avg_cost) & (df['Outcome'] == 'Critical')]
AvgCost_PoorOut
```

10. Save cleaned and merged data to CSV for dashboard use.

```
df.to_csv(r"Downloads/Cleaned_TreatmentRecords_Merged.csv")
```

11. Show monthly trend of admissions using a line plot.

```
df['Month'] = df['Treatment_Date'].dt.strftime('%b')
Monthly_Admissions = df.groupby('Month')['Record_ID'].count()

months_order = ['Jan','Feb','Mar','Apr','May','Jun','Jul','Aug','Sep','Oct','Nov','Dec']
Monthly_Admissions = Monthly_Admissions.reindex(months_order)
Monthly_Admissions

plt.figure(figsize=(10,5))
plt.plot(Monthly_Admissions.index.astype(str), Monthly_Admissions.values,
marker='o', ms= 10, mfc= 'red')
plt.title("Monthly Admissions Trend", fontsize=16, weight="bold")
plt.xlabel("Month", fontsize=12)
plt.ylabel("Number of Admissions", fontsize=12)
plt.grid(True)
plt.show()
```

12. Group data to find doctors with best outcome-to-cost ratio.

```
outcome_map = {'Recovered': 3, 'Ongoing': 2, 'Critical': 1}
df['Outcome_Score'] = df['Outcome'].map(outcome_map)

best_doctors = df.groupby('Doctor_ID').apply(
    lambda x: x['Outcome_Score'].mean() / x['Treatment_Cost'].mean()
).reset_index(name='Outcome_to_Cost')

# Sort descending to get top doctors
best_doctors = best_doctors.sort_values('Outcome_to_Cost', ascending=False)
best_doctors.head(10)
```