Technical Report: Final Project DS 5110: Introduction to Data Management and Processing

Team Members: Mia Khan and Onn Ye Young
Khoury College of Computer Sciences
Data Science Program
khan.mia@northeastern.edu and young.on@northeastern.edu

November 17, 2024

Contents

1 Introduction **3**

2 Literature Review **3**

3 Methodology **4**
3.1 Data Collection 4
3.2 Data Preprocessing 4
3.3 Analysis Techniques 4

4 Results and Analysis **5**
4.1 Age Distribution 5
4.2 Income Distribution 6
4.3 Religion Distribution 7
4.4 Marriage Status Distribution 8
4.5 Procedure Cost Distribution 9
4.6 Procedure Types Distribution 9
4.7 Average Procedure Cost by Type 10
4.8 Average Age per Procedure 10
4.9 Income Distribution by Marriage Status 11
4.10 Income by Occupation 12

5 Discussion **12**

6 Conclusion **13**

7 References **13**

A Appendix A: Code **14**
A.1 Data Cleaning and Preprocessing 14
A.2 Data Visualization 20
A.3 Additional Code for Aid Type and Amount 23

B Appendix B: Additional Figures **25**
B.1 Correlation Matrix Heatmap 25
B.2 Procedure Types by Occupation 25
B.3 Procedure Types by Occupation pc 26

1 Introduction

The goal of this project is to create a database for a charity medical foundation, called the Dow Patient Care Association (DPCA), which is a student-led NGO that aims to financially assist the non-affording patients of Dow Hospital in Pakistan. This charity foundation helps patients pay for their medical procedures via a reimbursement process, taking into account factors such as occupation, income, and alternate forms of financial support.

The creation of this database would allow members of the foundation to easily perform data analysis and run queries on the data, creating static and dynamic visualizations and utilizing a dashboard to filter the data accordingly. This would provide the foundation valuable information on procedure requests and trends, as well as analysis of patient demographics and financials.

The scope of this project includes a populated database in SQL that has been subsequently cleaned, standardized, transformed, and validated in Python, as well as queries on the data, data visualizations that are both static and interactive, and an HTML dashboard displaying the visualizations.

2 Literature Review

The Global Burden of Diseases, injuries, and Risk Factors Study (GBD) was done in 2019 for Pakistan, comparing health indicators since 1990 to provide insights to strengthen the health-care system, reduce inequalities, and improve patient outcomes. In order to obtain these health indicators, the GBD 2019 utilized data inputs to estimate factors including socio-demographic index, healthy life expectancy, and risk factors [1]. This was done at the national and subnational levels, for Pakistan as a whole and its four provinces and three territories [1].

From these estimates, it was found that "Pakistan is showing signs of undergoing an epidemiological transition as the burden shifts to NCDs" [1]. NCDs are noncommunicable diseases, which largely consist of "cardiovascular diseases, diabetes, cancers, and chronic respiratory diseases" [2]. Worldwide, NCDs are about 74% of all deaths globally, with 85% of premature NCD deaths occurring in low- and middle-income countries [2]. To reduce the risk of NCDs, efforts focused on informing young populations about lifestyle choices and medical interventions, as well as investments in preventative medicine and health systems are key [1].

As part of these efforts in bolstering preventative medicine and health systems, the Pakistan Government launched a program in 2022, awarding residents of Punjab a health card, which would cover a majority of medical expenses, increasing access to healthcare [1]. Programs such as this one, in addition to foundations such as the DPCA, are imperative in reducing present and future healthcare burdens in Pakistan. Data analysis on patient demographics, resource allocation, and healthcare costs allow for evaluation of program success and provide pathways for improvements in healthcare systems and program outreach.

3 Methodology

3.1 Data Collection

The data in the dataset is collected from patients who go to the DPCA. Employees of the DPCA will ask patients various questions, filling out a Google Form that gets inputted into a Google Sheet that contains all the data.

3.2 Data Preprocessing

After the database was created and populated in SQL, each table was imported into Python for cleaning and preprocessing. To clean the data, the duplicate headers were first removed and the proper data types were set. Then, the dataframes were combined into a dataframe with all of the data from the database.

With all of the data in a single dataframe, duplicate rows and columns were checked for and dropped. Following this step, some of the data was then mapped, changing text into numbers for easier processing.

The next step was then to find the number of null values in each column and replace them with the appropriate values, such as "N.A", "Unknown", or 0, depending on the category.

Once there were no longer any null values, functions to clean different columns were created, replacing non-numeric values and strings with NaN, and handling range values by taking the midpoint of the range. Once the column was cleaned, the values were set to the proper data type.

With the data cleaned, standardized, and transformed, it was then validated, ensuring that the values contained within the database made logical sense. This includes validating that there are no values less than 0 for categories such as procedure cost, income, or age.

3.3 Analysis Techniques

The analysis of the data was conducted using various visualization methods to uncover trends and patterns. These techniques included:

- **Histograms:** Used to understand the distribution of numerical variables such as procedure costs and patient ages.
- **Bar Plots:** Applied to compare categorical variables like procedure types and their average costs.
- **Pie Charts:** Utilized to show proportional distributions, such as the breakdown of patients by religion or marriage status.
- **Box Plots:** Used for examining the spread and outliers in income distributions across categories like occupation and marriage status.

The visualizations were created using Python libraries such as Matplotlib and Seaborn, ensuring clarity and effective communication of results.

4 Results and Analysis

4.1 Age Distribution

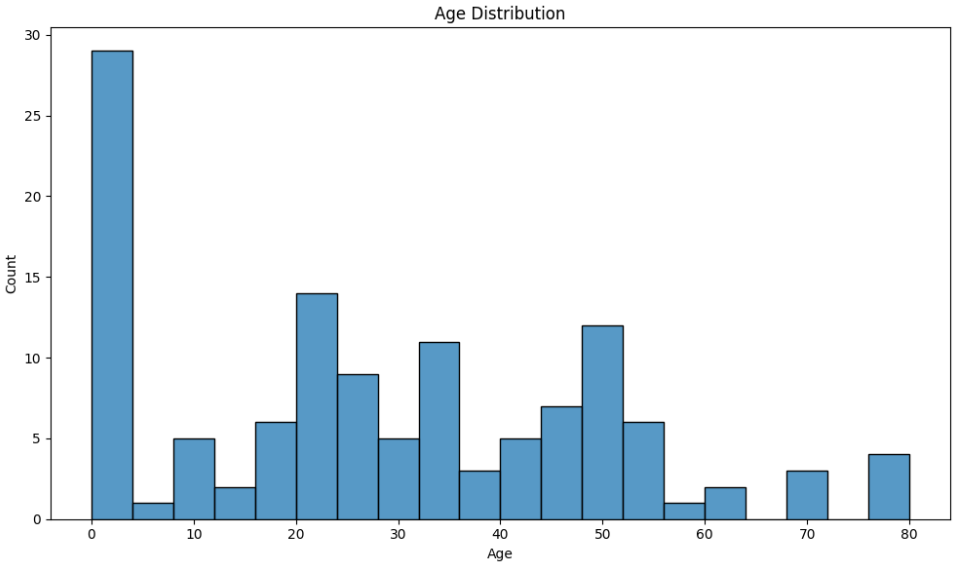

Figure 1: Age Distribution

Figure 1 highlights the age distribution of patients. The population is relatively young, with the largest group being 0–10 years old. This suggests a need for pediatric coverage. The working-age population (20–50 years) might benefit from partial reimbursement schemes, while the smaller elderly population (60+) may require specialized care.

4.2 Income Distribution

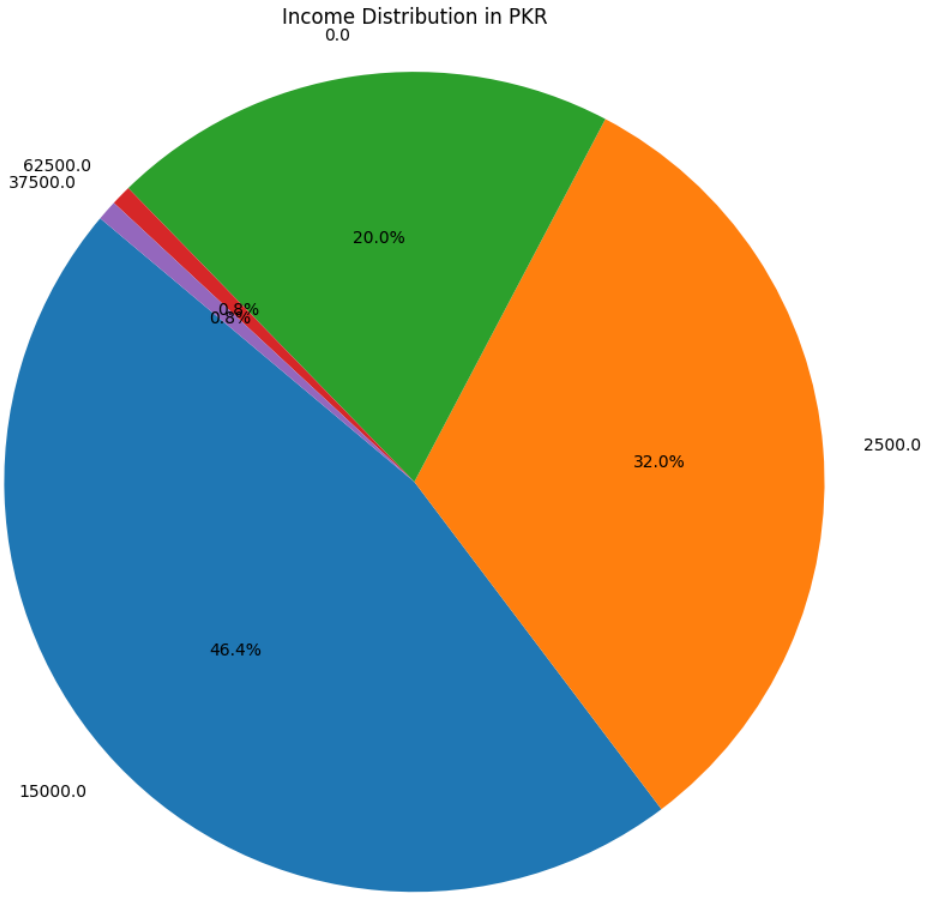

Figure 2: Income Distribution in PKR/month

The pie chart in Figure 2 illustrates the distribution of monthly income across patients. 98.4% of the population earns 15,000 PKR or less. As of November 2024, 15,000 PKR equates to 54 USD. Included in this 98.4% is 20% of the total population that earns no income, requiring substantial support, and 32% that earn only 2,500 PKR, likely needing full reimbursement. The 46.4% in the 15,000 PKR bracket could qualify for partial reimbursement, while less than 2% in highest brackets of 37,500 to 62,500 PKR might need minimal support. A large percentage of the population falls into lower-income brackets, highlighting economic disparities.

4.3 Religion Distribution

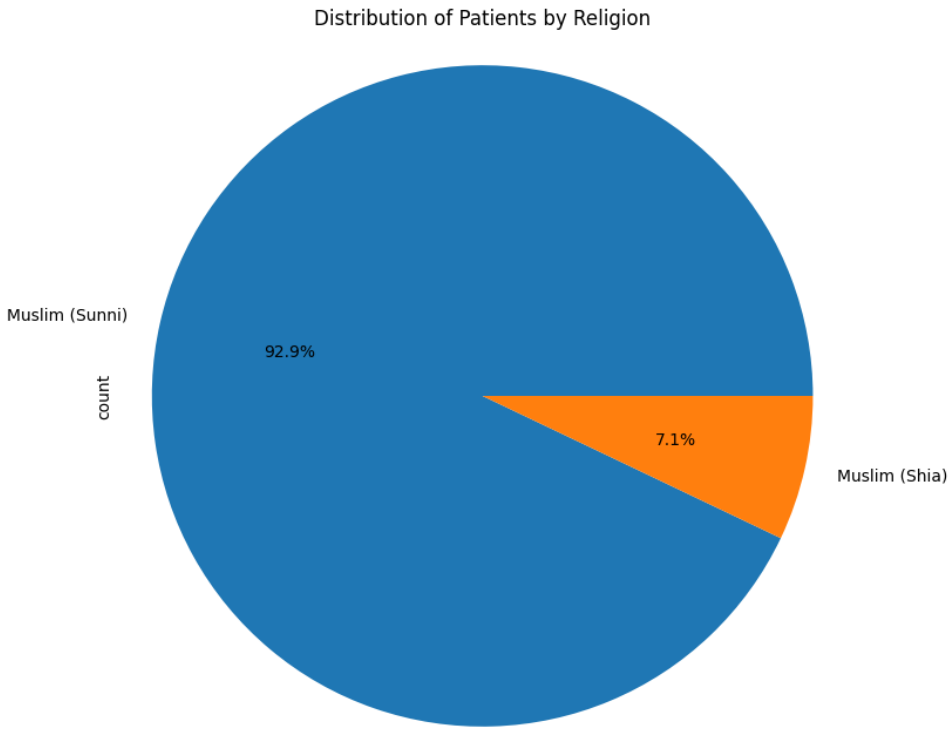

Figure 3: Distribution of Patients by Religion

Figure 3 shows the proportion of patients belonging to different religions.The population served is predominantly Sunni (92.9%), with 7.1% Shia representation. This understanding could guide culturally sensitive outreach programs and identify potential gaps in service delivery to other communities.

4.4 Marriage Status Distribution

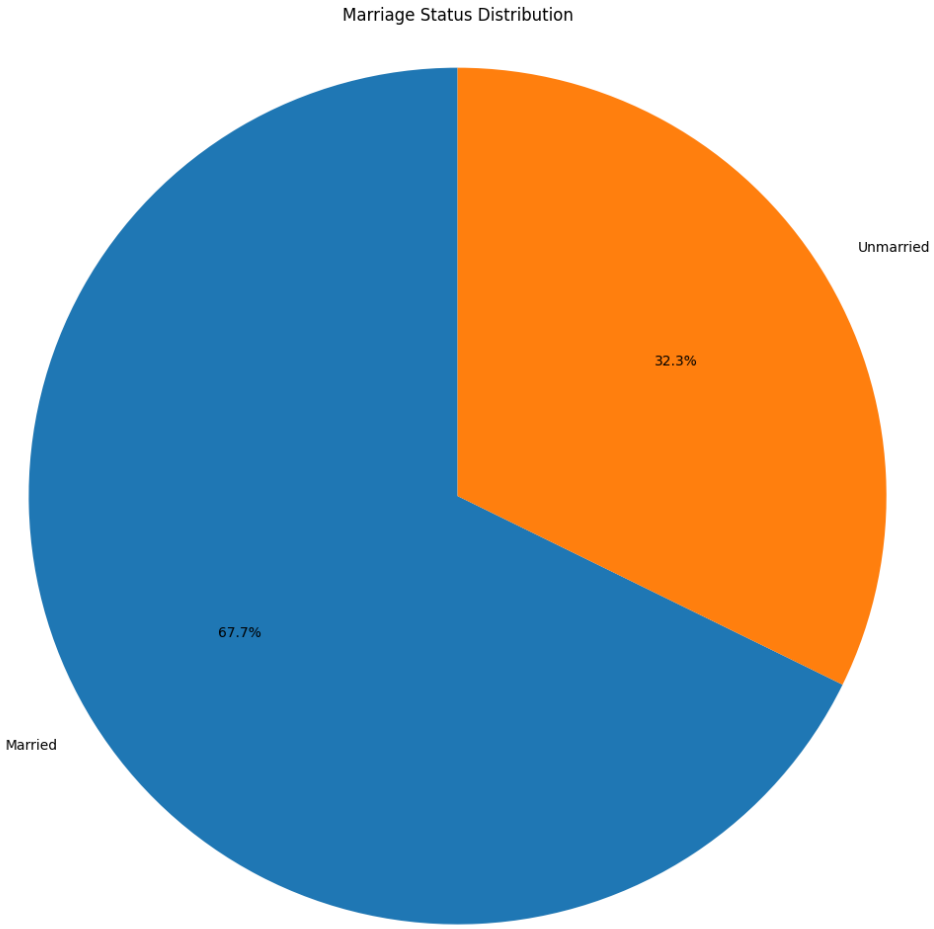

Figure 4: Marriage Status Distribution

In Figure 4, the marriage status distribution of patients is presented. A majority of patients (67.7%) are married, indicating the potential for family-based coverage plans. However, unmarried patients (32.3%) might need more individualized support options.

4.5 Procedure Cost Distribution

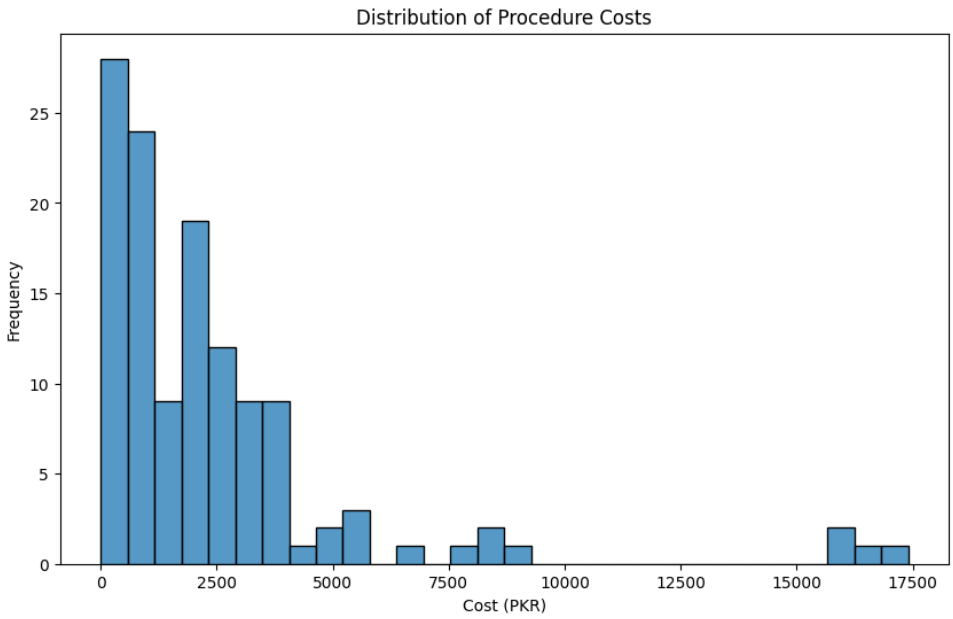

Figure 5: Distribution of Procedure Costs

The histogram in Figure 5 shows the distribution of procedure costs. Most procedures fall in the low-cost range (0 to 5,000 PKR), highlighting the foundation's focus on basic medical needs. Fewer procedures fall into high-cost ranges (15,000 to 17,500 PKR), suggesting tiered reimbursement thresholds could be implemented to allocate funds effectively.

4.6 Procedure Types Distribution

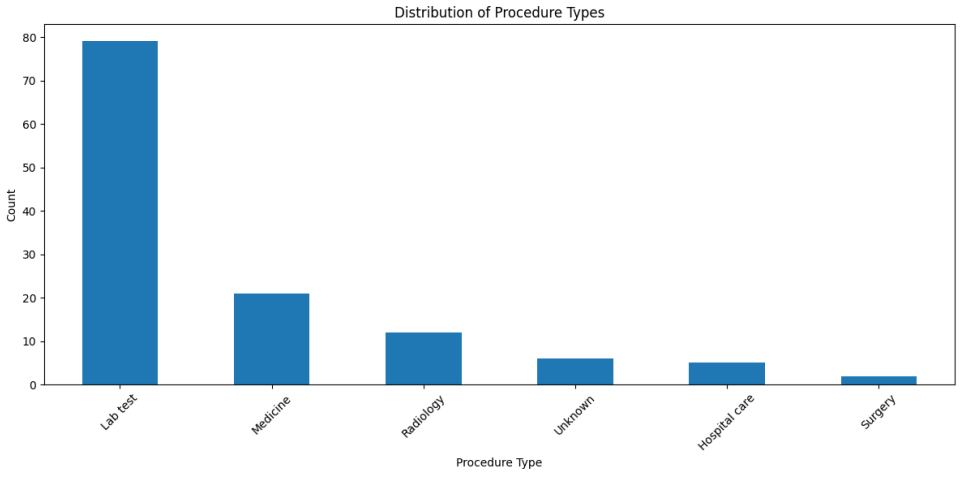

Figure 6: Distribution of Procedure Types

Figure 6 shows the different procedure types. Lab tests are the most common procedures (78 cases), reflecting high demand for diagnostic services. Medicine-related procedures are the second most common, while surgeries are the least utilized. This trend

suggests prioritization of diagnostic service reimbursements and potential bulk negotiation with labs.

4.7 Average Procedure Cost by Type

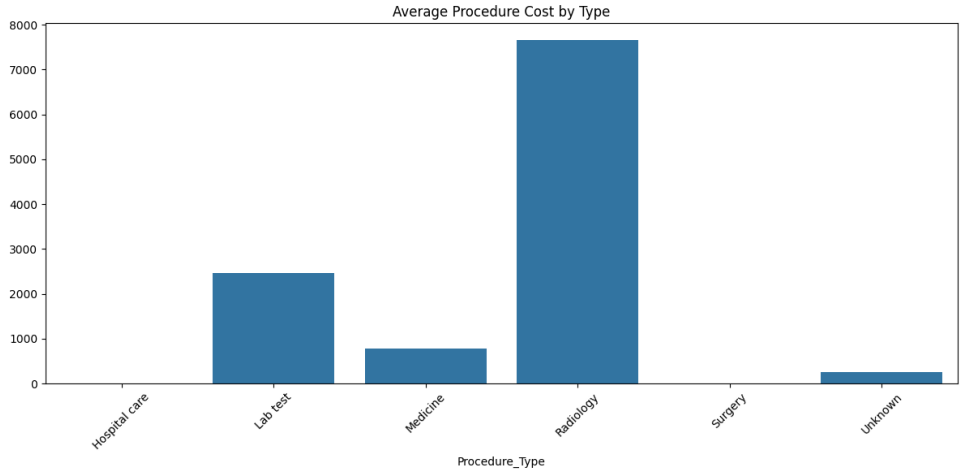

Figure 7: Average Procedure Cost by Type

In Figure 7, the average cost per procedure type is shown. Radiology procedures are the most expensive ($\approx 7,500$ PKR), followed by lab tests ($\approx 2,500$ PKR). Medicine and surgery are more affordable, potentially requiring less financial aid. This helps identify which procedures are costlier on average and could guide resource allocation or cost management strategies.

4.8 Average Age per Procedure

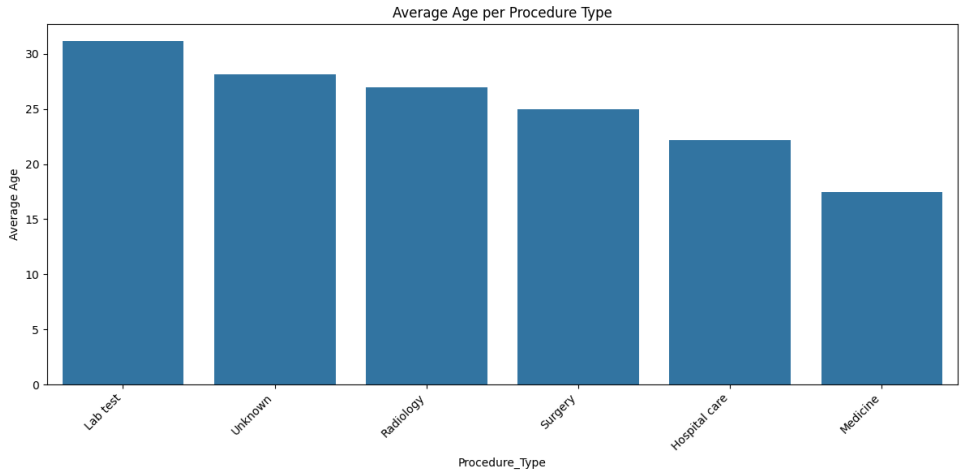

Figure 8: Average Age per Procedure Type

Figure 8 displays the average age of patients for each procedure type. Older patients (average age around 30) tend to undergo lab tests, reflecting a need for diagnostic services in this age group. Medicine-related procedures are most common among younger patients

(average age around 17), indicating their relevance for pediatric and adolescent care. Radiology and other procedures fall within a moderate age range (20 to 31), suggesting a balanced demand across age groups. As a result of this data, age-specific outreach and reimbursement plans can be tailored for different procedure types, as younger patients may benefit from medicine-focused coverage and older patients may require enhanced support for diagnostic and advanced procedures.

4.9 Income Distribution by Marriage Status

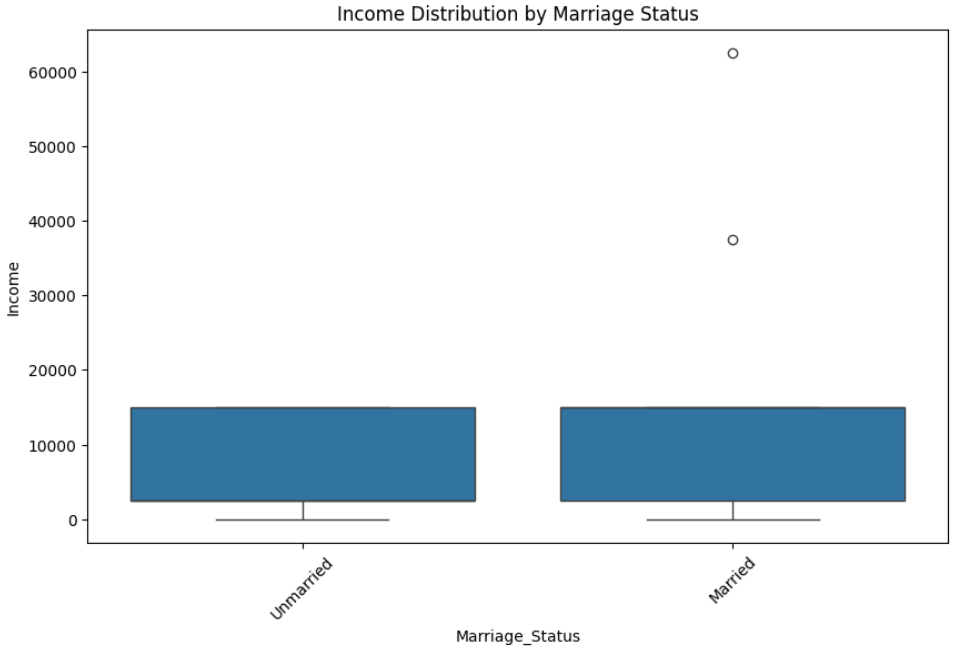

Figure 9: Income Distribution by Marriage Status

The box plot in Figure 9 compares income distribution based on marital status. Both married and unmarried groups display similar median income levels, suggesting that marital status alone may not strongly influence income. The married group tends to have slightly more outliers than the unmarried group, possibly due to dual-income households or shared financial resources.

4.10 Income by Occupation

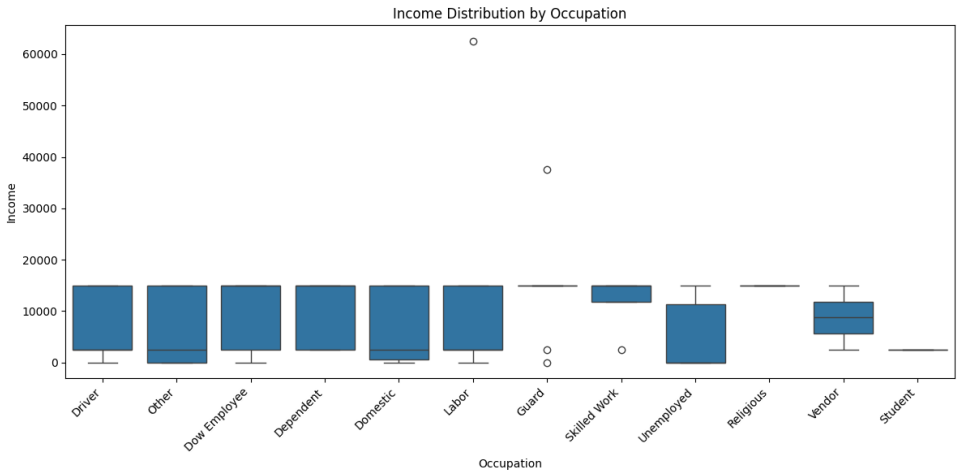

Figure 10: Income Distribution by Occupation

The box plot in Figure 10 examines incomes across various occupations. The box plot shows considerable variation in income across different occupations. Occupations like Skilled Workers have a noticeably higher median income compared to other groups, which could suggest a more stable or specialized labor market for these roles. Guards, Laborers, and Domestic Workers display consistently lower median incomes, highlighting the potential need for additional financial support. Significant income inequality is observed across occupations, especially in categories like Guards and Religious occupations, where some individuals earn much higher than others (outliers). This could point to the presence of a few high earners within these occupations, but the majority earn low wages.

5 Discussion

In analyzing general patient demographic information, such as age, religion, and marriage status, a better image of who the DPCA serves is created. The most abundant patient age falls below 10 years old. However, there is a wide spread of data, with the DPCA providing funds to those in all stages of life (Figure 1). When looking at income, more than 98% of the patients make less than 15,000 PKR a month, indicating a dire need for financial support and highlighting the importance of programs and foundations that provide it (Figure 2). An evaluation of patient religions show that 92.9% are Sunni, which could guide the creation of outreach programs to expand awareness of healthcare related financial aid programs to other communities (Figure 3). About a third of patients are unmarried, while the rest of the population is married (Figure 4). Providing various support options depending on patients marital status could be beneficial, offering a family-based coverage plan in addition to more individualized support.

To better evaluate the success of resource allocation to patients, distributions of procedure costs and procedure types were created. The most common procedure type is lab tests (Figure 6), which are important for preventative screening, potentially lowering the healthcare burden of NCDs, as discussed in the literature review. Procedure costs show that financial support for patients is a dire need. Most procedure costs are below 2500 PKR (Figure 5), but with 52% of patients making 2500 PKR or less a month (Figure 2),

the cost of healthcare would create a serious financial burden. Looking at the average procedure cost by type, lab tests cost 2500 PKR on average (Figure 7), further displaying the financial burden that would be imparted on patients if they paid for procedures themselves. Lab tests also have the highest average age of patients, reflecting a need for preventative and diagnostic services (Figure 8). Thus, it is shown that the DPCA is providing much needed financial support for lower income patients, allowing them to take preventative measures against disease.

Understanding how income varies with factors such as marital status and occupation is key for identifying groups that might need greater financial support. In an evaluation of the effect of marital status on income, both married and unmarried groups show similar median incomes, but there are a few high-earning outliers in married group, potentially indicating dual-income households or shared financial resources (Figure 9). Evaluating how income varies with occupation, there are considerable variations in median income (Figure 10). Domestic workers and students display lower median incomes, providing a potential area of outreach for the foundation. Additional income inequality is seen among groups like labor, guard work, and skilled work, with a wide variation of incomes within the occupations themselves.

6 Conclusion

The creation of the database and subsequent data visualizations has provided important insights on current resource allocation of the foundation and has highlighted potential areas of outreach for certain groups. With lab tests being the most common procedure type, utilizing funds to support preventative care and diagnostic services is a key initiative to prevent the future healthcare burden of NCDs. Further, highlighting the disparities in income by occupation could lead to outreach programs for low-income groups to reduce the financial burden of healthcare they may face. Additional outreach programs could also be created for those of different communities, as an evaluation of patient religions show that more than 90% of patients utilizing the foundation's services are Sunni.

While the analysis of the data in this database has already generated some insights on the success of the foundation and areas for potential improvement, this project was limited by the quantity of data, with only the information from 124 patients. As the DPCA continues serving patients, more data will be added to the dataset and additional analyses can be performed, allowing for continuous monitoring of success in reducing the financial burden of low-income patients. Additional future work for this project includes a re-evaluation of the amount of aid allocated to each patient depending on factors such as income, monthly budgets, and patient eligibility, potentially allowing the DPCA to serve a larger quantity of patients.

7 References

References

[1] Hafeez, Assad et al., *The state of health in Pakistan and its provinces and territories, 1990–2019: a systematic analysis for the Global Burden of Disease Study 2019*, The Lancet Global Health, Volume 11, Issue 2, e229 - e243, 2023.

[2] World Health Organization, *Communicable and Noncommunicable Diseases and Mental Health*, Available: https://www.who.int/our-work/communicable-and-noncommunicable-diseases-and-mental-health, Accessed: 16-Nov-2024.

A Appendix A: Code

A.1 Data Cleaning and Preprocessing

```
# import libraries
import numpy as np
import pandas as pd
import sqlite3
import re
import matplotlib.pyplot as plt
import seaborn as sns

# import database
def ReadDB(table_name):
    conn = sqlite3.connect('/content/DPCA.db')

    query = f'SELECT * FROM {table_name}'
    df = pd.read_sql(query, conn)

    conn.close()

    return df

df_patient = ReadDB('Patient')
df_procedure = ReadDB('Procedure')
df_patient_financials = ReadDB('Patient_Financials')
df_financial_aid = ReadDB('Financial_Aid')
df_zakat_eligibility = ReadDB('Zakat_Eligibility')

# Clean Patient DataFrame - Remove duplicate headers and set proper data types
df_patient = df_patient.iloc[1:]  # Remove duplicate header row
df_patient.loc[:, 'Age'] = pd.to_numeric(df_patient['Age'], errors='coerce')
df_patient.loc[:, 'Patient_ID'] = pd.to_numeric(df_patient['Patient_ID'], errors='coerce')

# Clean Procedure DataFrame
df_procedure = df_procedure.iloc[1:]  # Remove duplicate header row
df_procedure.loc[:, 'Procedure_ID'] = pd.to_numeric(df_procedure['Procedure_ID'], errors='coerce')
df_procedure.loc[:, 'Patient_ID'] = pd.to_numeric(df_procedure['Patient_ID'], errors='coerce')
df_procedure.loc[:, 'Procedure_Cost'] = pd.to_numeric(df_procedure['Procedure_Cost'], errors='coerce')
df_procedure.loc[:, 'Amount_Paid_DPCA'] = pd.to_numeric(df_procedure['Amount_Paid_DPCA'], errors='coerce')

# Clean Patient Financials DataFrame
df_patient_financials = df_patient_financials.iloc[1:]  # Remove duplicate header row
```

```
df_patient_financials.loc[:, 'Financial_Info_ID'] = pd.to_numeric(df_patient_financials['Financial_Info_ID'], errors='coerce')
df_patient_financials.loc[:, 'Patient_ID'] = pd.to_numeric(df_patient_financials['Patient_ID'], errors='coerce')
df_patient_financials.loc[:, 'Dependent_Members'] = pd.to_numeric(df_patient_financials['Dependent_Members'], errors='coerce')

# Clean Financial Aid DataFrame
df_financial_aid = df_financial_aid.iloc[1:]  # Remove duplicate header row
df_financial_aid.loc[:, 'Financial_Aid_ID'] = pd.to_numeric(df_financial_aid['Financial_Aid_ID'], errors='coerce')
df_financial_aid.loc[:, 'Patient_ID'] = pd.to_numeric(df_financial_aid['Patient_ID'], errors='coerce')
df_financial_aid.loc[:, 'Aid_Date'] = pd.to_datetime(df_financial_aid['Aid_Date'], errors='coerce')

# combine dataframes
df_all = pd.concat([df_patient, df_procedure, df_patient_financials, df_financial_aid, df_zakat_eligibility], axis=1)

# check for row duplicates
df_all.duplicated().sum()

# check for column duplicates
transposed_df = df_all.transpose()              # transpose dataframe
transposed_df.duplicated().sum()                # find column duplicates

# drop column duplicates (no row duplicates)
transposed_df.drop_duplicates(inplace = True)
transposed_df.duplicated().sum()

# map the data - change text to numbers
df_all['Religion'] = df_all['Religion'].map({'Muslim (Shia)':0, 'Muslim (Sunni)':1})
df_all['Syed_NonSyed'] = df_all['Syed_NonSyed'].map({'Syed':0, 'Non-Syed':1})
df_all['Marriage_Status'] = df_all['Marriage_Status'].map({'Married':0, 'Unmarried':1})
df_all['Income'] = df_all['Income'].map({'No Income': 0, 'No income': 0, '1-10000/month': 5000, '10001-50000/month': 30000, '50001-100000/month': 75000, '100001-150000/month': 125000}).fillna(df_all['Income'])

# return # of each column's none values
df_all.replace('', pd.NA, inplace=True)  # Replace empty strings with NaN (if needed)
df_all.isnull().sum()

#Transforming data/ filling or dropping null values
#df_all['Income'] = df_all['Income'].fillna('N.A')          # I think it might be better to keep numerical values as NaN instead of 'N.A' because it's a string? - Onn Ye
df_all['Procedure_Cost'] = df_all['Procedure_Cost'].fillna('N.A')
df_all['Aid_Type'] = df_all['Aid_Type'].fillna('N.A')
df_all['Eligibility_Criteria'] = df_all['Eligibility_Criteria'].fillna('N.A')
```

```
79
80 df_all['Religion'] = df_all['Religion'].fillna('Unknown')
81 df_all['Syed_NonSyed'] = df_all['Syed_NonSyed'].fillna('Unknown')
82 df_all['Marriage_Status'] = df_all['Marriage_Status'].fillna('Unknown')
83 df_all['Contact_No'] = df_all['Contact_No'].fillna('Unknown')
84 df_all['Address'] = df_all['Address'].fillna('Unknown')
85 df_all['Occupation'] = df_all['Occupation'].fillna('Unknown')
86 df_all['Procedure_Date'] = df_all['Procedure_Date'].fillna('Unknown')
87
88 df_all['Dependent_Members'] = df_all['Dependent_Members'].fillna(0)
89 df_all['Properties_Owned'] = df_all['Properties_Owned'].fillna(0)
90 df_all['Amount_Paid_DPCA'] = df_all['Amount_Paid_DPCA'].fillna(0)
91 df_all['Financial_Support_Method'] = df_all['Financial_Support_Method'
   ].fillna('Unknown')
92 df_all['Is_Eligible'] = df_all['Is_Eligible'].fillna('No')
93
94 df_all['Patient_ID'] = df_all['Patient_ID'].fillna('Unknown')  # or use
    a default ID if needed
95 df_all['Patient_Name'] = df_all['Patient_Name'].fillna('Unknown')
96 df_all['Age'] = df_all['Age'].fillna(0)  # assuming age 0 for missing
   data
97 df_all['CNIC_No'] = df_all['CNIC_No'].fillna('Unknown')
98 df_all['Procedure_Type'] = df_all['Procedure_Type'].fillna('Unknown')
99 df_all['MR_number'] = df_all['MR_number'].fillna('Unknown')
100 df_all['Aid_Date'] = df_all['Aid_Date'].fillna('Unknown')
101
102 df_all.isnull().sum()
103
104 # Check non-numeric entries
105 def check_non_numeric(column):
106     return df_all[column][pd.to_numeric(df_all[column], errors='coerce'
   ).isna()]
107
108 print("Non-numeric entries in Amount_Paid_DPCA:", check_non_numeric('
   Amount_Paid_DPCA'))
109 print("Non-numeric entries in Procedure_Cost:", check_non_numeric('
   Procedure_Cost'))
110 print("Non-numeric entries in Income:", check_non_numeric('Income'))
111
112 # Function to clean the 'Income' and 'Procedure_Cost' columns
113 def clean_numeric_column(column):
114     # Replace non-numeric values with NaN to handle them separately
115     df_all[column] = pd.to_numeric(df_all[column], errors='coerce')
116
117     # Replace specific strings with NaN (or you could replace with 0 if
    you prefer)
118     df_all[column] = df_all[column].replace(['No income', 'N.A'], np.
   nan)
119
120     # Handle range values by taking the midpoint of the range
121     def convert_range(value):
122         if isinstance(value, str) and '-' in value:
123             parts = value.split('-')
124             try:
125                 # Convert each part of the range to an integer and take
    the average
126                 return (int(parts[0]) + int(parts[1].split('/')[0])) /
   2
```

```
            except ValueError:
                return np.nan
        return value

    # Apply range conversion function
    df_all[column] = df_all[column].apply(convert_range)

    # Convert remaining non-numeric entries to NaN
    df_all[column] = pd.to_numeric(df_all[column], errors='coerce')

    # Fill NaN with 0 or another default value if needed
    df_all[column] = df_all[column].fillna(0)

# Clean 'Income' and 'Procedure_Cost' columns
clean_numeric_column('Income')
clean_numeric_column('Procedure_Cost')

# Verify that non-numeric values are handled
print("Non-numeric entries in Amount_Paid_DPCA:", check_non_numeric('
    Amount_Paid_DPCA'))
print("Non-numeric entries in Procedure_Cost:", check_non_numeric('
    Procedure_Cost'))
print("Non-numeric entries in Income:", check_non_numeric('Income'))

# Function to clean Financial column
def clean_financial_column(value):
    if pd.isna(value):  # If the value is NaN, keep it as NaN
        return value

    # Remove any non-numeric characters except for decimal points
    value = re.sub(r'[^\d.]', '', str(value))

    # Handle cases where value is empty after removing non-numeric
    characters
    if value == '':
        return pd.NA

    # Convert the cleaned value to a float
    try:
        return float(value)
    except ValueError:
        return pd.NA  # Return NaN if conversion fails

df_all['Amount_Paid_DPCA'] = df_all['Amount_Paid_DPCA'].apply(
    clean_financial_column)
df_all['Procedure_Cost'] = df_all['Procedure_Cost'].apply(
    clean_financial_column)
df_all['Income'] = df_all['Income'].apply(clean_financial_column)

# Function to clean Income column
def clean_income(value):
    if pd.isna(value):
        return value

    # Handle "No income" case
    if "no income" in str(value).lower():
        return 0

```

```
    # Handle ranges (e.g., "1-10000/month" -> take the midpoint)
    range_match = re.match(r'(\d+)[\D]+(\d+)', str(value))
    if range_match:
        lower = int(range_match.group(1))
        upper = int(range_match.group(2))
        return (lower + upper) / 2  # Return the midpoint of the range

    # Clean and convert as a regular financial value
    return clean_financial_column(value)

# Apply to the Income column
df_all['Income'] = df_all['Income'].apply(clean_income)

# Check data types
print(df_all[['Amount_Paid_DPCA', 'Procedure_Cost', 'Income']].dtypes)

# Display any remaining non-numeric entries
print("Remaining non-numeric in Amount_Paid_DPCA:", df_all['
    Amount_Paid_DPCA'][pd.to_numeric(df_all['Amount_Paid_DPCA'], errors=
    'coerce').isna()])
print("Remaining non-numeric in Procedure_Cost:", df_all['
    Procedure_Cost'][pd.to_numeric(df_all['Procedure_Cost'], errors='
    coerce').isna()])
print("Remaining non-numeric in Income:", df_all['Income'][pd.
    to_numeric(df_all['Income'], errors='coerce').isna()])

# Convert columns to numeric, coercing any remaining non-numeric values
    to NaN
df_all['Amount_Paid_DPCA'] = pd.to_numeric(df_all['Amount_Paid_DPCA'],
    errors='coerce')
df_all['Procedure_Cost'] = pd.to_numeric(df_all['Procedure_Cost'],
    errors='coerce')
df_all['Income'] = pd.to_numeric(df_all['Income'], errors='coerce')

# Check data types again
print(df_all[['Amount_Paid_DPCA', 'Procedure_Cost', 'Income']].dtypes)

# Validate financial columns
invalid_amount_paid = df_all[df_all['Amount_Paid_DPCA'] < 0]
invalid_procedure_cost = df_all[df_all['Procedure_Cost'] < 0]
invalid_income = df_all[df_all['Income'] < 0]

print("Invalid Amount_Paid_DPCA:", invalid_amount_paid)
print("Invalid Procedure_Cost:", invalid_procedure_cost)
print("Invalid Income:", invalid_income)

# Further clean the 'Age' column to ensure all values are numeric

def clean_age(age):
    if isinstance(age, str):
        # Extract numeric part
        numbers = re.findall(r'\d+\.?\d*', age)
        if numbers:
            return float(numbers[0])
    try:
        return float(age)
    except ValueError:
        return np.nan  # Return NaN for any non-convertible values
```

```
230
231 # Apply the cleaning function
232 df_all['Age'] = df_all['Age'].apply(clean_age)
233
234 # Drop any rows with NaN values in 'Age' or 'Procedure_Cost' to ensure
    clean calculations
235 df_all.dropna(subset=['Age', 'Procedure_Cost'], inplace=True)
236
237 # Clean Occupation column
238 def clean_occupation(value):
239     # Convert the value to a string for consistent handling
240     value_str = str(value).lower()
241
242     # Handle "Dow" case
243     if "dow" in value_str:
244         return "Dow Employee"
245
246     # Handle "Driver" case
247     if "rickshaw" in value_str or "driver" in value_str or "rikshaw" in
    value_str:
248         return "Driver"
249
250     # Handle "Guard" case
251     if "guard" in value_str or "gaurd" in value_str or "security" in
    value_str:
252         return "Guard"
253
254     # Handle "Domestic" case
255     if "house" in value_str or "maid" in value_str or "cook" in
    value_str or "clean" in value_str:
256         return "Domestic"
257
258     # Handle "Labor" case
259     if any(term in value_str for term in ["labour", "labor", "mazdoor",
    "sweep", "janitor", "cement", "block maker", "trash", "vendor"]):
260         return "Labor"
261
262     # Handle "Dependent" case
263     if any(term in value_str for term in ["father", "husband", "brother
    ", "son", "wife"]):
264         return "Dependent"
265
266     # Handle "Vendor" case
267     if any(term in value_str for term in ["sell", "vendor", "stall"]):
268         return "Vendor"
269
270     # Handle "Student" case
271     if "student" in value_str:
272         return "Student"
273
274     # Handle "Unemployed" case
275     if any(term in value_str for term in ["unemployed", "not", "none",
    "jobless", "does nothing", "-", "N/A"]):
276         return "Unemployed"
277
278     # Handle "Religious" case
279     if "imam" in value_str:
280         return "Religious"
```

```
281
282     # Handle "Skilled Work" case
283     if any(term in value_str for term in ["sewing", "tailor", "packager
    ", "chai hotel", "operator", "machine"]):
284         return "Skilled Work"
285
286     # Default case
287     return "Other"
288
289 # Apply to the Income column
290 df_all['Occupation'] = df_all['Occupation'].apply(clean_occupation)
291
292 # Clean Procedure Type column
293 def clean_procedure(value):
294     # Convert the value to a string for consistent handling
295     value_str = str(value).lower()
296
297     # Handle "Lab test" case
298     if "lab test" in value_str:
299         return "Lab test"
300
301     # Handle "Surgery" case
302     if "surgery" in value_str:
303         return "Surgery"
304
305     # Handle "Medicine" case
306     if "med" in value_str:
307         return "Medicine"
308
309     # Default case: return the original value
310     return value
311
312 # Apply to the Income column
313 df_all['Procedure_Type'] = df_all['Procedure_Type'].apply(
    clean_procedure)
314
315 # Print out cleaned and processed database for checking
316 df_all.head(50)
```

Listing 1: Code for Data Cleaning and Preprocessing

A.2 Data Visualization

```
1
2 # Undo mapped data for charts
3 df_all['Religion'] = df_all['Religion'].map({0:'Muslim (Shia)', 1:'
    Muslim (Sunni)'})
4 df_all['Syed_NonSyed'] = df_all['Syed_NonSyed'].map({0:'Syed', 1:'Non-
    Syed'})
5 df_all['Marriage_Status'] = df_all['Marriage_Status'].map({0:'Married',
     1:'Unmarried'})
6
7 # Visualizations
8 sns.set_style('whitegrid')
9 sns.set_palette("husl")
10
11 # Procedure Cost Distribution
```

```
plt.figure(figsize=(10, 6))
sns.histplot(data=df_all, x='Procedure_Cost', bins=30)
plt.title('Distribution of Procedure Costs')
plt.xlabel('Cost (PKR)')
plt.ylabel('Frequency')
plt.show()

# Procedure Types Distribution
plt.figure(figsize=(12, 6))
df_all['Procedure_Type'].value_counts().plot(kind='bar')
plt.title('Distribution of Procedure Types')
plt.xlabel('Procedure Type')
plt.ylabel('Count')
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()

# Average Procedure Cost by Type
plt.figure(figsize=(12, 6))
avg_cost = df_all.groupby('Procedure_Type')['Procedure_Cost'].mean()
sns.barplot(x=avg_cost.index, y=avg_cost.values)
plt.title('Average Procedure Cost by Type')
plt.xticks(rotation=45)
plt.tight_layout()
#plt.savefig('avg_procedure_cost.png')
plt.show()

# Average Age per Procedure
plt.figure(figsize=(12, 6))
avg_age = df_all.groupby('Procedure_Type')['Age'].mean().sort_values(
    ascending=False)
sns.barplot(x=avg_age.index, y=avg_age.values)
plt.title('Average Age per Procedure Type')
plt.xticks(rotation=45, ha='right')
plt.ylabel('Average Age')
plt.tight_layout()
#plt.savefig('avg_age_per_procedure.png')
plt.show()

# Religion Distribution
plt.figure(figsize=(8, 8))
df_all['Religion'].value_counts().plot(kind='pie', autopct='%1.1f%%')
plt.title('Distribution of Patients by Religion')
plt.axis('equal')
plt.show()


# Income Distribution by Marriage Status
plt.figure(figsize=(10, 6))
sns.boxplot(data=df_all, x='Marriage_Status', y='Income')
plt.title('Income Distribution by Marriage Status')
plt.xticks(rotation=45)
plt.show()

# Pie chart for income distribution
plt.figure(figsize=(8, 8))
income_counts = df_all['Income'].value_counts()
plt.pie(income_counts, labels=income_counts.index, autopct='%1.1f%%',
```

```
    startangle=140)
plt.title('Income Distribution in PKR')
plt.axis('equal')
plt.tight_layout()
plt.show()

# Pie chart for marriage status
plt.figure(figsize=(10, 10))
marriage_counts = df_all['Marriage_Status'].value_counts()
plt.pie(marriage_counts, labels=marriage_counts.index, autopct='%1.1f%%
    ', startangle=90)
plt.title('Marriage Status Distribution')
plt.axis('equal')
plt.tight_layout()
plt.show()

# Histogram for age distribution
plt.figure(figsize=(10, 6))
sns.histplot(data=df_all, x='Age', bins=20)
plt.title('Age Distribution')
plt.xlabel('Age')
plt.ylabel('Count')
plt.tight_layout()
plt.show()

# Box plots for income by occupation
plt.figure(figsize=(12, 6))
sns.boxplot(x='Occupation', y='Income', data=df_all)
plt.xticks(rotation=45, ha='right')
plt.title('Income Distribution by Occupation')
plt.tight_layout()
plt.show()

numeric_df = df_all.select_dtypes(include=[np.number])

# Calculate the correlation matrix for numeric columns
correlation_matrix = numeric_df.corr()

# Create a heatmap for the correlation matrix
plt.figure(figsize=(12, 8))
sns.heatmap(correlation_matrix, annot=True, cmap='crest', fmt='.2f')
plt.title('Correlation Matrix Heatmap')
plt.tight_layout()
plt.show()

print("Correlation matrix heatmap created")

# Analyze job impact on procedure type
job_procedure_crosstab = pd.crosstab(df_all['Occupation'], df_all['
    Procedure_Type'])

# Create a heatmap of the job-procedure relationship
plt.figure(figsize=(12, 8))
sns.heatmap(job_procedure_crosstab, annot=True, fmt='d', cmap='YlOrRd')
plt.title('Procedure Types by Occupation')
plt.xlabel('Procedure Type')
plt.ylabel('Occupation')
plt.xticks(rotation=45, ha='right')
```

```
plt.tight_layout()
plt.show()

# Calculate percentage distribution of procedures within each occupation
procedure_pct = job_procedure_crosstab.div(job_procedure_crosstab.sum(axis=1), axis=0) * 100

# Create a heatmap of the percentage distribution
plt.figure(figsize=(12, 8))
sns.heatmap(procedure_pct, annot=True, fmt='.1f', cmap='YlOrRd')
plt.title('Procedure Types Distribution by Occupation (%)')
plt.xlabel('Procedure Type')
plt.ylabel('Occupation')
plt.xticks(rotation=45, ha='right')
plt.tight_layout()
plt.show()
```

Listing 2: Code for Data Visualization

A.3 Additional Code for Aid Type and Amount

```

print("\
Top 3 most common occupation-procedure combinations:")
# Get the top 3 occupation-procedure combinations
flat_crosstab = job_procedure_crosstab.unstack()
top_combinations = flat_crosstab.sort_values(ascending=False)[:3]
for (occupation, procedure), count in top_combinations.items():
    print(f"{occupation} - {procedure}: {count} cases")

# Create aid_type column based on eligibility criteria
def determine_aid_type(criteria):
    if isinstance(criteria, str):
        criteria = criteria.lower()
        # Check for all required conditions for Zakat
        muslim_condition = 'muslim' in criteria
        non_syed_condition = 'non-syed' in criteria or 'non syed' in criteria
        wealth_condition = '52.5 tola' in criteria or 'does not own' in criteria

        if muslim_condition and non_syed_condition and wealth_condition:
            return 'Zakat Funds'
    return 'General Funds'

# Apply the function to create aid_type column
df_all['Aid_Type'] = df_all['Eligibility_Criteria'].apply(determine_aid_type)

# Create is_eligible column based on aid_type
df_all['Is_Eligible'] = df_all['Aid_Type'].apply(lambda x: 'Yes' if x == 'Zakat Funds' else 'No')

# Display sample results
print("Sample of Eligibility Criteria, Aid Type, and Eligibility:")
```

```
print(df_all[['Eligibility_Criteria', 'Aid_Type', 'Is_Eligible']].head(10))

# Display distribution of aid types
print("Distribution of Aid Types:")
print(df_all['Aid_Type'].value_counts())

# Display eligibility counts
print("Distribution of Eligibility:")
print(df_all['Is_Eligible'].value_counts())

# Calculation of Aid Amounts based on eligibility

# monthly_budget = 168000  # Rupees

# # Calculate total aid by type
# aid_summary = df_all.groupby('Aid_Type').agg({
#     'Aid_Amount': ['sum', 'count', 'mean']
# }).round(2)

# aid_summary.columns = ['Total_Amount', 'Number_of_Patients', 'Average_Amount']
# aid_summary = aid_summary.reset_index()

# print("Aid Allocation Summary:")
# print(aid_summary)

# # Calculate percentages
# total_aid = aid_summary['Total_Amount'].sum()
# aid_summary['Percentage_of_Total_Aid'] = (aid_summary['Total_Amount'] / total_aid * 100).round(2)

# print("\
# Detailed Aid Distribution:")
# print(df_all.head())
```

Listing 3: Additional Code for Filling in Aid Type and Amount based on Eligibility

B Appendix B: Additional Figures

B.1 Correlation Matrix Heatmap

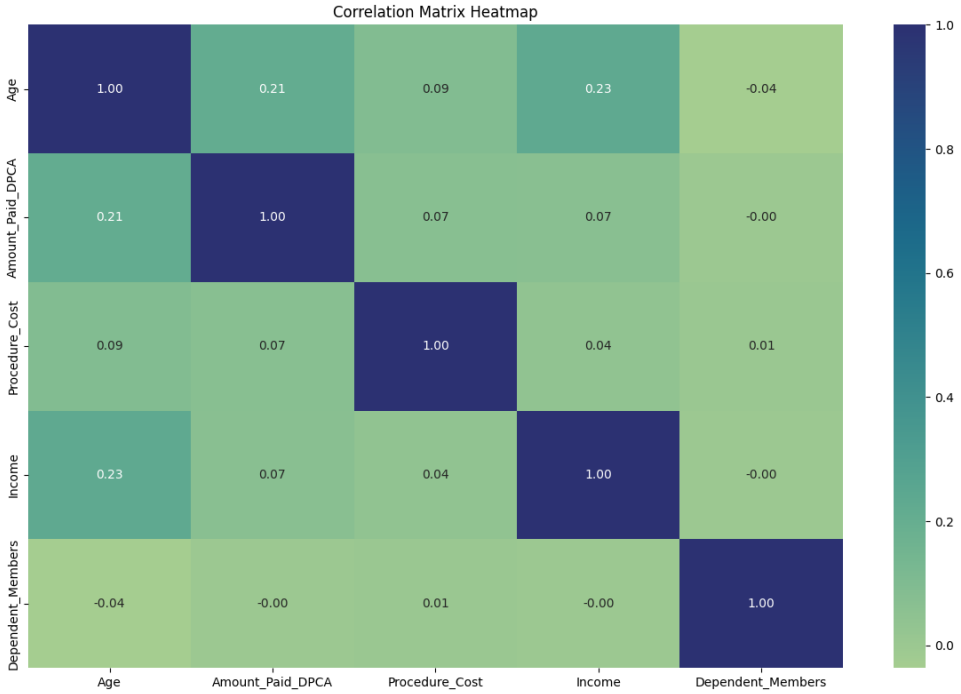

Figure 11: Correlation Matrix Heatmap

B.2 Procedure Types by Occupation

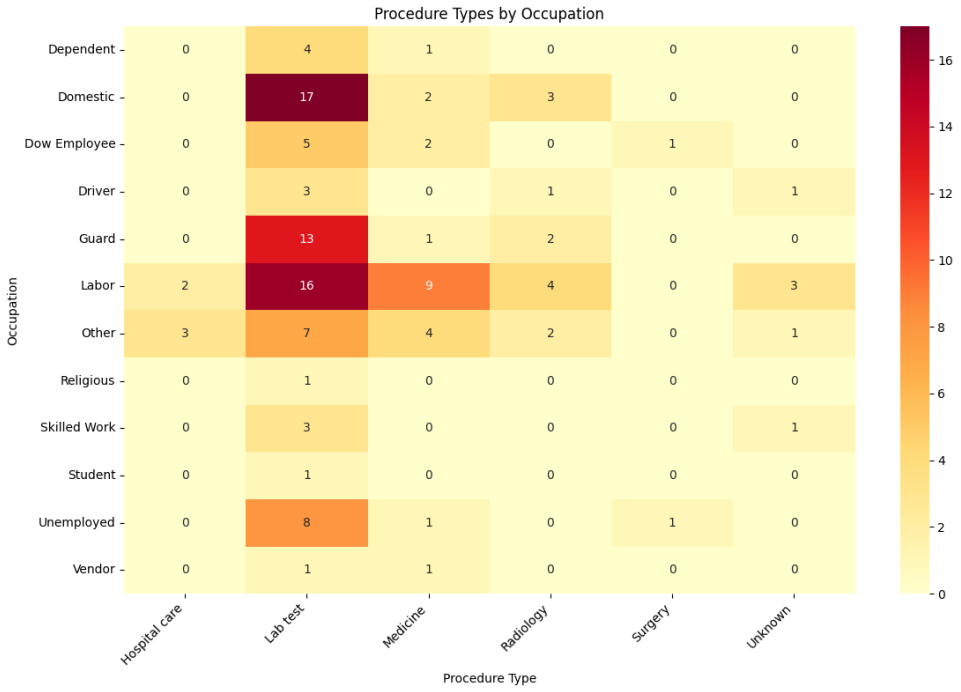

Figure 12: Procedure Types by Occupation

B.3 Procedure Types by Occupation pc

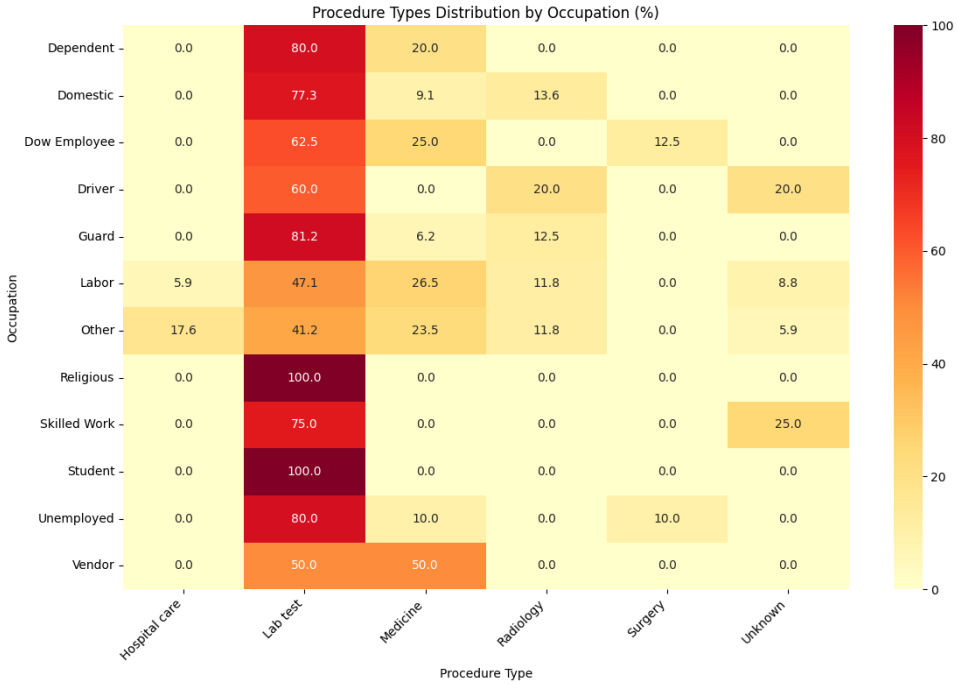

Figure 13: Procedure Types by Occupation pc