Homework 2

Aden Benson
Data Engineering 300

Part 1

Analysis Question 1

A -

```
SELECT ethnicity as ETHNICITY,drug as DRUG, count(*) as NUM_ORDERS
FROM PRESCRIPTIONS
JOIN ADMISSIONS ON ADMISSIONS.hadm_id=prescriptions.hadm_id
GROUP BY ethnicity, drug
ORDER BY ethnicity;
```

Figure 1: Initial query to find overall summary

```
WITH totals AS (
SELECT ethnicity,drug, count(*) as NUM_ORDERS
    FROM PRESCRIPTIONS
    JOIN ADMISSIONS ON ADMISSIONS.hadm_id=prescriptions.hadm_id
    GROUP BY ethnicity, drug),
max_per_ethnicity AS (
    SELECT ethnicity, max(NUM_ORDERS) as MAX_AMOUNT
    From totals GROUP BY ethnicity)
SELECT totals.ethnicity as ETHNICITY, drug as DRUG, MAX_AMOUNT as MAX_AMOUNT
From totals
JOIN max_per_ethnicity
ON totals.ethnicity = max_per_ethnicity.ethnicity
AND totals.NUM_ORDERS=max_per_ethnicity.MAX_AMOUNT
ORDER BY ETHNICITY;
```

Figure 2: Advanced query to find most prominent drug for each ethnicity

B - In Fig. 1, we are first creating a general summary of the information to prepare us for answering the overall analysis question 1. To start, we are joining the admissions table onto the prescriptions table by hospital admission id. This allows us to see both drug and ethnicity information for each row in prescriptions. By grouping on both ethnicity and drug and adding a count(*) column, we see the total number of rows for each combination of ethnicity and drug.

In Fig. 2, we are answering analysis question 1 by finding the most common three prescriptions for each combination of ethnicity and drug. It does this by first creating the table in Figure 1, and naming it “totals”. Then, we create a new table titled “max_per_ethnicity”, which simply finds the maximum number of occurrences for each ethnicity. Finally, we join these two tables to find all rows where ethnicity and the number of orders lines up. This provides us with the maximum prescription orders for each ethnicity in the dataset.

C -

ETHNICITY varchar	DRUG varchar	NUM_ORDERS int64
AMERICAN INDIAN/ALASKA NATIVE FEDERALLY RECOGNIZED TRIBE	D5W	2
AMERICAN INDIAN/ALASKA NATIVE FEDERALLY RECOGNIZED TRIBE	LeVETiracetam	5
AMERICAN INDIAN/ALASKA NATIVE FEDERALLY RECOGNIZED TRIBE	Phenylephrine	1
AMERICAN INDIAN/ALASKA NATIVE FEDERALLY RECOGNIZED TRIBE	Pantoprazole Sodium	2
AMERICAN INDIAN/ALASKA NATIVE FEDERALLY RECOGNIZED TRIBE	Bacitracin Ointment	1
AMERICAN INDIAN/ALASKA NATIVE FEDERALLY RECOGNIZED TRIBE	Fentanyl Citrate	2
AMERICAN INDIAN/ALASKA NATIVE FEDERALLY RECOGNIZED TRIBE	Ranitidine	2
AMERICAN INDIAN/ALASKA NATIVE FEDERALLY RECOGNIZED TRIBE	Lorazepam	3

Figure 3: Head of query shown in Fig. 1

ETHNICITY varchar	DRUG varchar	MAX_AMOUNT int64
AMERICAN INDIAN/ALASKA NATIVE FEDERALLY RECOGNIZED TRIBE	5% Dextrose	27
ASIAN	D5W	27
BLACK/AFRICAN AMERICAN	Insulin	38
HISPANIC OR LATINO	5% Dextrose	28
HISPANIC/LATINO – PUERTO RICAN	0.9% Sodium Chloride	86
OTHER	NS	11
UNABLE TO OBTAIN	0.9% Sodium Chloride	28
UNKNOWN/NOT SPECIFIED	D5W	37
WHITE	Potassium Chloride	381

Figure 4: Result of query shown in Fig. 2

D - This analysis shows several patterns in drug prescription frequencies across different ethnicities. For example, Potassium Chloride emerged as the most common prescription for White patients, while 5% Dextrose was most common for American Indian/Alaska Native and Hispanic or Latino. Both Puerto Rican and “Unable to Obtain” rows had 0.9% Sodium Chloride as the most ordered prescription. With deeper investigation, we may be able to see either differences in health conditions or systematic differences in care among these populations.

To qualify drug use, I originally planned on using the form_val_disp column, which has the amount of medication ordered. However, this column contains values that are in different units. Most columns are expressed in mL, however others are expressed as PUFFS, CAPS, grams, etc. This heavily skewed the most common drugs to those which had the smallest unit of measurement. As a result, I decided to base the analysis on single prescription counts, using the number of times each drug appeared in the data as frequency of use.

Analysis Question 2

A -

```
WITH actual_age AS (
SELECT
    date_diff('year',
        CAST(PATIENTS.DOB AS DATE),
        CAST(ADMISSIONS.admittime AS DATE)) as AGE,
    D_ICD_PROCEDURES.short_title as Procedure,
    count(*) as procedure_count
FROM PROCEDURES_ICD
JOIN PATIENTS ON PATIENTS.subject_id=PROCEDURES_ICD.subject_id
Join ADMISSIONS ON PROCEDURES_ICD.hadm_id=ADMISSIONS.hadm_id
JOIN D_ICD_PROCEDURES ON PROCEDURES_ICD.icd9_code = D_ICD_PROCEDURES.icd9_code
GROUP BY AGE, D_ICD_PROCEDURES.short_title),
counts AS (
    Select CASE
            WHEN AGE <=19 THEN '<=19'
            WHEN AGE BETWEEN 20 AND 49 THEN '20-49'
            WHEN AGE BETWEEN 50 AND 79 THEN '50-79'
            ELSE '>80' END AS Age_Group,
        Procedure,
        SUM(procedure_count) as Total_Procedures,
    FROM actual_age
    WHERE AGE BETWEEN 0 AND 120
    GROUP BY Age_Group, procedure)
Select age_group, Procedure, total_procedures FROM counts
QUALIFY
ROW_NUMBER() OVER (
    PARTITION BY age_group
    ORDER BY Total_Procedures DESC
) <= 3
Order by
    CASE
        WHEN Age_Group = '<=19' THEN 1
        WHEN Age_Group = '20-49' THEN 2
        WHEN Age_Group = '50-79' THEN 3
        WHEN Age_Group = '>80' THEN 4
    END, total_procedures DESC;
```

Figure 5: Query to find the top three procedures performed on each age group

B - The query shown in Fig. 5 finds the top three most frequently performed procedures for patients within four distinct age groups. The first table created known as “actual_age” finds the ages of each patient as an integer. We are only given the patient's date of birth and admission time as datetime variables, but we can use the date_diff function to find the difference between these values. We must cast the columns as DATE types to make the date_diff function work. To ensure we are getting the correct patient information, admission time, and procedure information, we merge the tables of procedures_icd, patients, admissions, and d_icd_procedures. This table shows the number of occurrences for each d_icd_procedures.short_title, which is a short standardized description for each procedure.

The next table known as “count” simply takes the above information and sorts it into counts over age groups instead of raw ages. By using the CASE function, we separate age into a column known as “Age_group”. We also set a restriction that the maximum age is only 120, as dates of

birth are anonymized, creating instances where ages are set to be over 300. I decided to simply ignore this data, as it is unknown which age group they would fit in.

Finally, we use the QUALIFY function on the "counts" table to find the top three procedures by number of occurrences. This is done with the ROW_NUMBER() function, which ranks procedures partitioned across each age group and only selects those within the top three. Finally, the output is ordered in a way where the youngest age group is shown first and the oldest age group is shown last using a CASE function.

C -

Age_Group varchar	Procedure varchar	Total_Procedures int128
<=19	Venous cath NEC	2
<=19	Vertebral fx repair	1
<=19	Interruption vena cava	1
20-49	Venous cath NEC	9
20-49	Entral infus nutrit sub	7
20-49	Insert endotracheal tube	6
50-79	Venous cath NEC	25
50-79	Entral infus nutrit sub	22
50-79	Packed cell transfusion	13
>80	Venous cath NEC	16
>80	Packed cell transfusion	13
>80	Insert endotracheal tube	8
12 rows		3 columns

Figure 6: Query to find the top three procedures for each age group

D - The SQL query finds the top three most frequently performed procedures under the four distinct age groups: <20, 20-49, 50-79, and >79. With the given data shown in Figure 6, we see that the most common procedures in this specific dataset tend to be constant across the different age groups. I added a column total_procedures so we can see the sample size in each of our groups.

For the <20 group, we see that there is not a large enough sample size to create accurate insights on the most common procedures. The only procedure that is listed more than one time in this age group is Venous cath NEC, which comes up 2 times. However, in the other three groups, we have large enough sample sizes to make distinct conclusions.

The most common procedure among the three larger groups is again a Venous cath NEC. If this dataset is representative of the general population, then we can assume that the most common procedure at most hospitals is also the Venous catheterization NEC. We also see that the insertion

of an endotracheal tube is a common procedure, appearing in the top three of all three major groups. Finally, Enteral nutrition infusion is also a common procedure, appearing in the top three of both the 20-49 group and the 50-79 group.

Next, we can look at the number of procedures performed for each age group. We see that there is a large amount of procedures performed among the 50-79 and >79 age groups, particularly the 50-79 group. This shows that extensive care is needed among the 50-79 age group, while the general population of >79 year olds is much less. Another explanation for this drop off is that many of the >80 rows were disregarded, as some of the anonymized data seemed to generally be older individuals.

Analysis Question 3

A -

```
WITH STAY_LENGTHS AS (
SELECT subject_id, date_diff('hour',
                            CAST(intime AS TIMESTAMP),
                            CAST(outtime AS TIMESTAMP)) as Hours FROM ICUSTAYS)
SELECT
    percentile_cont(0) WITHIN GROUP (ORDER BY Hours) as Min_Hours,
    percentile_cont(.25) WITHIN GROUP (ORDER BY Hours) as Q1_Hours,
    percentile_cont(.5) WITHIN GROUP (ORDER BY Hours) as Median_Hours,
    percentile_cont(.75) WITHIN GROUP (ORDER BY Hours) as Q3_Hours,
    percentile_cont(1) WITHIN GROUP (ORDER BY Hours) as Max_Hours,
From STAY_LENGTHS
```

Figure 7: SQL query to find the quantiles for hours spent in the ICU

```
SELECT ADMISSIONS.ethnicity, PATIENTS.gender,
                    ROUND(AVG(date_diff('hour',
                    CAST(intime AS TIMESTAMP),
                    CAST(outtime AS TIMESTAMP))),2) as Average_Stay,
                    count(*) as Num_Records
FROM ICUSTAYS
JOIN ADMISSIONS on ADMISSIONS.hadm_id = ICUSTAYS.hadm_id
JOIN PATIENTS on ADMISSIONS.subject_id = PATIENTS.subject_id
GROUP BY ADMISSIONS.ethnicity, PATIENTS.gender
ORDER BY ADMISSIONS.ethnicity, PATIENTS.gender;
```

Figure 8: SQL query to find average ICU stay by ethnicity and gender

B - The first query shown in Fig. 7 finds the quantiles for the number of hours spent in the ICU in our dataset. First, we create a table titled “Stay_lengths” which simply finds the number of hours spent for each ICU stay. To do this, we use the date_diff function, casting both ICU intime and outtime as TIMESTAMP types. This creates a column known as Hours that tracks the number of hours spent for an ICU stay. Then, we use the function percentile_cont() to find the

percentile of the Hours column, ordered in a descending order. Thus, this gives us some values we can use to find the range of our dataset.

The second query shown in Fig. 8 goes through each combination of ethnicity and gender and finds the average ICU stay length among the dataset. This begins by joining ICUstays, Admissions, and Patients among hadm_id and subject_id. This provides us with information on patient ethnicity (admissions), gender (patients), and total time in the hospital (ICUstays). We then select the rounded average of the date_diff function used above to find the average stay length for each ethnicity and gender combination, along with the number of records for each combination using count(*). After grouping over ethnicity and gender and ordering by ethnicity and gender, we successfully find the average icu stay length for each combination along with the sample size for each combination.

C -

Min_Hours double	Q1_Hours double	Median_Hours double	Q3_Hours double	Max_Hours double
3.0	29.5	50.5	104.0	849.0

Figure 9: Quantile information for ICU stay length

ethnicity varchar	gender varchar	Average_Stay double	Num_Records int64
AMERICAN INDIAN/ALASKA NATIVE FEDERALLY RECOGNIZED TRIBE	M	272.0	2
ASIAN	F	16.0	1
ASIAN	M	170.0	1
BLACK/AFRICAN AMERICAN	F	269.0	4
BLACK/AFRICAN AMERICAN	M	71.33	3
HISPANIC OR LATINO	F	179.33	3
HISPANIC/LATINO – PUERTO RICAN	M	78.0	15
OTHER	F	32.0	2
OTHER	M	3.0	1
UNABLE TO OBTAIN	M	321.0	1
UNKNOWN/NOT SPECIFIED	F	132.89	9
UNKNOWN/NOT SPECIFIED	M	51.5	2
WHITE	F	124.73	44
WHITE	M	75.63	48
14 rows			4 columns

Figure 10: Average ICU stay for each combination of ethnicity and gender.

D - Based on the results from Fig. 9, we see that the range of ICU stays go from as short as 3 hours to as long as 849 hours, which is over 35 days. However, the median ICU stay is 50.5 hours, a little over two days, where the first and third quartiles are half and double this. This suggests that while most ICU stays are relatively short (2 days or less), a large number of patients require much longer care, with 25% of stays exceeding four days. This shows variability in patient needs and the importance of flexible ICU planning.

The results from Fig. 10 explain how ICU stays differ between ethnicities and gender. While I don't see much noticeable variation between ethnicities, it is very common for women to spend a longer time in the ICU than men. In particular, the ethnicities with a larger sample size are black/African American, hispanic/latino, and white. In these three groups, the average ICU stay for a woman ranges from 124 hours to 269 hours, while the average ICU stay for a male ranges from 71 to 78 hours. While I first thought this was because women gave birth in the ICU, a quick Google search told me that this was not the case, as this is not a regular occurrence. However, women with complications during pregnancy may be admitted into the ICU, which could explain the rise in average hours. I also found a US News article that explains women of color are more likely to be admitted to the ICU during pregnancy, which could explain the major increase in average stay length for black women (https://www.usnews.com/news/health-news/articles/2023-12-07/mothers-of-color-at-higher-risk-for-icu-admission-during-birth-hospitalization#:~:text=ICU%20admission%20rates%20rose%20significantly,mothers%20delivering%20a%20single%20child).

Part 2

1 -

```python
from cassandra.cluster import ExecutionProfile, EXEC_PROFILE_DEFAULT
from cassandra import ConsistencyLevel


# Define execution profile with LOCAL_QUORUM
execution_profile = ExecutionProfile(
    consistency_level=ConsistencyLevel.LOCAL_QUORUM
)

# Cluster setup with correct profile
cluster = Cluster(
    ['cassandra.us-east-2.amazonaws.com'],
    ssl_context=ssl_context,
    auth_provider=auth_provider,
    port=9142,
    execution_profiles={EXEC_PROFILE_DEFAULT: execution_profile}
)

# establishing connection to Keyspace
session = cluster.connect()
session.set_keyspace('atb2199_ks')  # Replace with your keyspace
```

Figure 11: Successful connection to AWS keyspace

In Fig. 11, we see that the set_keyspace(atb2199_ks) function works successfully. This shows that our docker container is connected to our .aws folder in the EC2 instance and shows that we can successfully connect to the keyspace.

2 - *No copies of the AWS credentials file are stored on any publicly accessible location, nor is the file in any way shared with anyone outside of DATA_ENG 300 (Spring 2025).*

3 -

Analysis Question 1

A -

```
session.execute("""
CREATE TABLE IF NOT EXISTS question_1 (
    id UUID PRIMARY KEY,
    ethnicity text,
    drug text
)
""")
```

Figure 12: Table construction for Analysis Question 1

B -

```
import uuid
import csv

# Create merge csv files together using pandas
drugs_df=pd.read_csv('hw2_data/PRESCRIPTIONS.csv', usecols=['hadm_id','drug'])
admissions_df=pd.read_csv('hw2_data/ADMISSIONS.csv', usecols=['hadm_id','ethnicity'])
df1=drugs_df.merge(admissions_df, on='hadm_id', how='inner')

# Upload information to Cassandra
for i in range(df1.shape[0]):
    session.execute("""
        INSERT INTO question_1 (id, ethnicity, drug)
        VALUES (%s, %s, %s)
    """, (uuid.uuid4(), df1.loc[i,'ethnicity'], df1.loc[i, 'drug']))
```

Figure 13: Uploading data into Cassandra for Analysis Question 1

C -

```
# Extract data from cassandra, then perform post-extraction analysis using python to aggregate.
rows=session.execute("SELECT ethnicity, drug FROM question_1")
extracted_df=pd.DataFrame(rows, columns=['ethnicity', 'drug'])

# Create counts per prescription using pandas.groupby function
counts=extracted_df.groupby(['ethnicity', 'drug']).size().reset_index(name='total_orders')

# Pick drug with max total_orders in each ethnicity
# First, find id number for each maximum grouped over ethnicity using idxmax
max_ids=counts.groupby('ethnicity')['total_orders'].idxmax()
# Then, create a dataframe using these ids
q1_df=counts.loc[max_ids]
```

Figure 14: Pull data from Cassandra, use Pandas to aggregate

D -

	ethnicity	drug	total_orders
2	AMERICAN INDIAN/ALASKA NATIVE FEDERALLY RECOGN...	5% Dextrose	27
89	ASIAN	D5W	27
205	BLACK/AFRICAN AMERICAN	Insulin	38
294	HISPANIC OR LATINO	5% Dextrose	28
377	HISPANIC/LATINO - PUERTO RICAN	0.9% Sodium Chloride	86
555	OTHER	NS	11
572	UNABLE TO OBTAIN	0.9% Sodium Chloride	28
655	UNKNOWN/NOT SPECIFIED	D5W	37
1152	WHITE	Potassium Chloride	381

Figure 15: Resulting dataframe from Analysis question 1

As we can see above in Fig. 15, the resulting dataframe is the same as the data shown in Fig. 4. Thus, the information can be validated.

Analysis Question 2

A -

```
session.execute("""
CREATE TABLE IF NOT EXISTS question_2 (
    id UUID PRIMARY KEY,
    hadm_id int,
    subject_id int,
    short_title text,
    dob timestamp,
    admittime timestamp,
)
""")
```

Figure 16: Table construction for Analysis Question 2

B -

```
# Load csvs
proc_icd=pd.read_csv('hw2_data/PROCEDURES_ICD.csv', usecols=['subject_id','hadm_id','icd9_code'])
d_icd=pd.read_csv('hw2_data/D_ICD_PROCEDURES.csv', usecols=['icd9_code','short_title'])
admissions=pd.read_csv('hw2_data/ADMISSIONS.csv', usecols=['subject_id','hadm_id','admittime'])
patients=pd.read_csv('hw2_data/PATIENTS.csv', usecols=['subject_id','dob'])

# Merge dfs together
df=proc_icd.merge(d_icd, on='icd9_code',how='left')
df=df.merge(admissions, on=['subject_id','hadm_id'], how='left')
df2=df.merge(patients, on='subject_id',how='left')

# Fill nan values with a string, was getting a type error
df2['short_title'] = df2['short_title'].fillna('UNKNOWN_PROCEDURE')

# Insert into Cassandra
for i in range(df2.shape[0]):
    session.execute("""
        INSERT INTO question_2 (id, hadm_id, subject_id, short_title, dob, admittime)
        VALUES (%s, %s, %s, %s, %s, %s)
    """, (uuid.uuid4(), df2.loc[i,'hadm_id'], df2.loc[i, 'subject_id'], df2.loc[i, 'short_title'], df2.loc[i, 'dob'], df2.loc[i, 'admittime']))
```

Figure 17: Uploading data into Cassandra for Analysis Question 2

C -

```
# Extract data
rows=session.execute("""SELECT hadm_id, short_title AS procedure_name, admittime, dob FROM question_2""")
extracted_df2=pd.DataFrame(rows, columns=['hadm_id','procedure_name','admittime','dob'])

# Create age column
extracted_df2['admittime']=pd.to_datetime(extracted_df2['admittime'])
extracted_df2['dob']=pd.to_datetime(extracted_df2['dob'])
# For the age column, first set both admittime and dob as numpy arrays, then subtract, then turn into years.
# Was getting int overflow errors before
extracted_df2['age']=((extracted_df2['admittime'].to_numpy(dtype='datetime64[D]')
                       -extracted_df2['dob'].to_numpy(dtype='datetime64[D]')).astype(int))//365

# Now, we can turn the age col into age buckets
# Define a helper function
def helper_bucket(age):
    if age <= 19:
        return '<=19'
    elif age <= 49:
        return '20-49'
    elif age <= 79:
        return '50-79'
    else:
        return '>80'
extracted_df2['age_group']=extracted_df2['age'].map(helper_bucket)

# Now, we can aggregate and pick the top three, similar to the aggregation done in analysis question 1
# First, get a column of counts grouped by age group and procedure name
counts=extracted_df2.groupby(['age_group','procedure_name']).size().reset_index(name='total_count')

# Now, select the top three from counts using .head(i), which displays the top i entries for each groupby. Sort_values makes this easy
top3=counts.sort_values(['age_group','total_count'], ascending=[True, False]).groupby('age_group').head(3)
```

Figure 18: Pull data from Cassandra, use Pandas to aggregate

D -

	age_group	procedure_name	total_count
47	20-49	Venous cath NEC	9
13	20-49	Entral infus nutrit sub	7
9	20-49	Cont inv mec ven 96+ hrs	6
148	50-79	Venous cath NEC	26
80	50-79	Entral infus nutrit sub	22
111	50-79	Packed cell transfusion	13
166	<=19	Venous cath NEC	2
149	<=19	Applic ext fix dev-femur	1
150	<=19	Atlas-axis fusion	1
238	>80	Venous cath NEC	19
216	>80	Packed cell transfusion	13
197	>80	Insert endotracheal tube	8

Figure 19: Resulting dataframe from Analysis Question 2

As we can see above in Fig. 19, the resulting dataframe is very similar to the data shown in Fig. 6. Thus, the information can be validated.

Analysis Question 3

A -

```
session.execute("""
CREATE TABLE IF NOT EXISTS question_3 (
    id UUID PRIMARY KEY,
    hadm_id int,
    subject_id int,
    intime timestamp,
    outtime timestamp,
    ethnicity text,
    gender text
)
""")
```

Figure 20: Table construction for Analysis Question 3

B -

```
# Upload csv files
icustays=pd.read_csv('hw2_data/ICUSTAYS.csv', usecols=['subject_id','hadm_id','intime','outtime'])
admissions=pd.read_csv('hw2_data/ADMISSIONS.csv', usecols=['hadm_id','ethnicity'])
patients=pd.read_csv('hw2_data/PATIENTS.csv', usecols=['subject_id','gender'])

# Merge dfs together
df3=icustays.merge(admissions, on='hadm_id',how='left').merge(patients, on='subject_id',how='left')

# Insert into Cassandra
for i in range(df3.shape[0]):
    session.execute("""
        INSERT INTO question_3 (id, hadm_id, subject_id, intime, outtime, ethnicity, gender)
        VALUES (%s, %s, %s, %s, %s, %s, %s)
    """, (uuid.uuid4(), df3.loc[i,'hadm_id'], df3.loc[i, 'subject_id'], df3.loc[i, 'intime'],
          df3.loc[i, 'outtime'], df3.loc[i, 'ethnicity'], df3.loc[i, 'gender']))
```

Figure 21: Uploading data into Cassandra for Analysis Question 3

C -

```
# Extract data from Cassandra
rows=session.execute("""SELECT intime, outtime, ethnicity, gender FROM question_3""")
extracted_df3=pd.DataFrame(rows, columns=['intime','outtime','ethnicity','gender'])

# Compute hours column in python, subtracting intime from outtime
extracted_df3['hours']=(pd.to_datetime(extracted_df3['outtime'])-
                        pd.to_datetime(extracted_df3['intime'])).dt.total_seconds()/3600

# First, create percentile dataframe from Part 1 using np.percentile
import numpy as np
percentiles=np.percentile(extracted_df3['hours'], [0,25,50,75,100])
percentile_df=pd.DataFrame({
    'Metric': ['Min','Q1','Median','Q3','Max'],
    'Value': percentiles})

# Next, create dataframe that shows average icu stay length for each combination of gender and ethnicity.
average_df3=extracted_df3[['hours','ethnicity','gender']].groupby(['ethnicity','gender']).mean()
```

Figure 22: Pull data from Cassandra, use Pandas to aggregate

D -

	Metric	Value
0	Min	2.542222
1	Q1	29.604097
2	Median	50.674722
3	Q3	103.897500
4	Max	849.756389

Figure 23: Quantile dataframe for Analysis Question 3

ethnicity	gender	hours
AMERICAN INDIAN/ALASKA NATIVE FEDERALLY RECOGNIZED TRIBE	M	272.091806
ASIAN	F	15.907222
	M	170.815278
BLACK/AFRICAN AMERICAN	F	268.829792
	M	71.453889
HISPANIC OR LATINO	F	179.031296
HISPANIC/LATINO - PUERTO RICAN	M	77.833481
OTHER	F	32.067500
	M	2.542222
UNABLE TO OBTAIN	M	320.568889
UNKNOWN/NOT SPECIFIED	F	133.032407
	M	51.491389
WHITE	F	124.701162
	M	75.693299

Figure 24: Average ICU length stay for Analysis Question 3

As seen above in both Figures 23 and 24, our outputs match the outputs gained from Part 1, validating our methods.