QUEEN'S UNIVERSITY BELFAST

DSA8002 ASSIGNMENT

Object-Oriented Programming meets Database

John Poole
40206201
Jpoole07@qub.ac.uk

Table of Contents

List of Figures

1. Introduction

The assignment brief outlines a problem that requires both object-oriented programming and database implementation. The problem was solved using python and this report describes each implementation stage, justifying why certain decisions were made and how this reflects the overall project design. First, the design of the database and classes will be outlined - specifically how they implement the criteria in the brief. A description of the important functions and operations of both the database and classes will follow before a section on how various class methods were tested.

An additional section showing how information from the database was analysed using both SQL and pandas will be detailed. This section is outside the scope of the project brief but is something I implemented to gain a deeper understanding of the hospital data.

As detailed in the brief a final section will discuss creating a draft plan to migrate the current project implementation to QUB's Azure HDInsight. All project source files can be found in the *Code Appendix* section.

2. Database Design

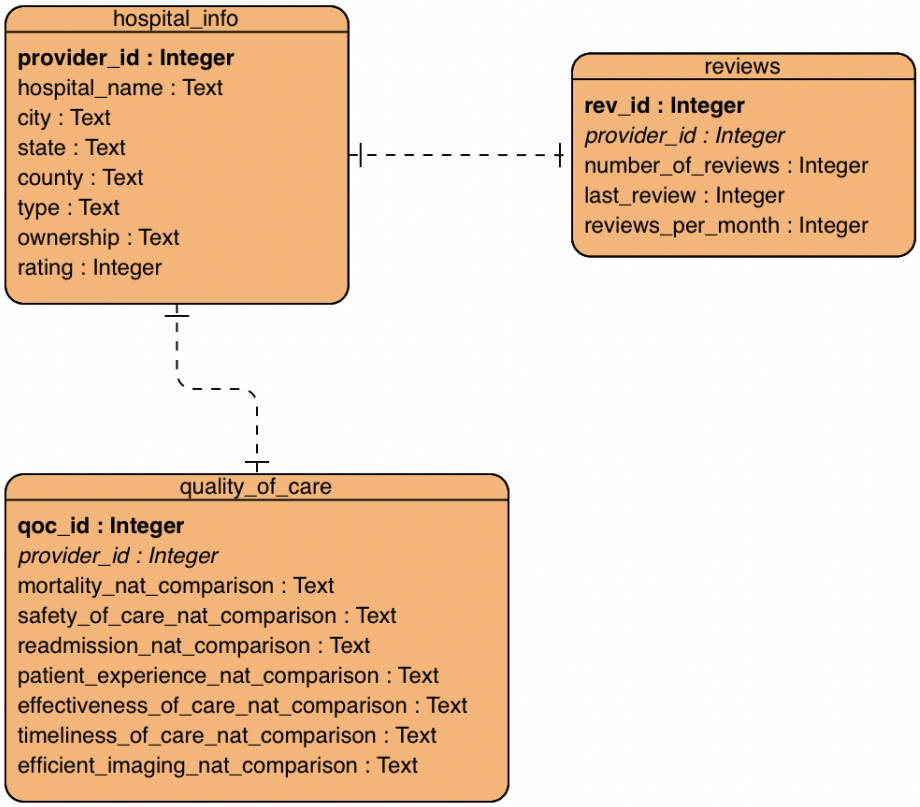


Figure 1 Database Design

3. Database Functionality

Figure 1 shows the database based on the original hospital dataset. Within the database, three tables are created – hospital_info, reviews, and quality_of_care. The process of creating these tables is shown in Figures 2, 3 and 4.

```
c.execute('''CREATE TABLE IF NOT EXISTS hospital_info
        (
        provider_id INTEGER PRIMARY KEY,
        hospital_name TEXT,
        city TEXT,
        state TEXT,
        county TEXT,
        type TEXT,
        ownership TEXT,
        rating INTEGER
        );''')
```

Figure 2 Hospital info table

Figure 2 creates the hospital info table which describes general information about each hospital as well as the overall hospital rating scored out of 5. The hospital provider id is assigned as the table primary key which uniquely identifies each hospital record.

```
c.execute('''CREATE TABLE IF NOT EXISTS quality_of_care
        (
        qoc_id INTEGER PRIMARY KEY AUTOINCREMENT NOT NULL,
        provider_id INTEGER,
        mortality_nat_comparison TEXT,
        safety_of_care_nat_comparison TEXT,
        readmission_nat_comparison TEXT,
        patient_experience_nat_comparison TEXT,
        effectiveness_of_care_nat_comparison TEXT,
        timeliness_of_care_nat_comparison TEXT,
        efficient_imaging_nat_comparison TEXT,
        FOREIGN KEY (provider_id) REFERENCES hospital_info(provider_id)
        );''')
```

Figure 3 Quality of care table

Figure 3 creates the quality-of-care table which compares the various quality of care measurements to the national average. qoc_id is assigned as the primary key with provider id inserted as a foreign key that refers to the primary key of the hospital info table. These two tables are therefore linked in a one-to-one relationship i.e., one record in hospital info can only be linked to one record in the quality-of-care table.

```
c.execute('''CREATE TABLE IF NOT EXISTS reviews
        (
         rev_id INTEGER PRIMARY KEY AUTOINCREMENT NOT NULL,
         provider_id INTEGER,
         number_of_reviews INTEGER,
         last_review INTEGER,
         reviews_per_month INTEGER,
         FOREIGN KEY (provider_id) REFERENCES hospital_info(provider_id)
        );''')
```

Figure 4 Reviews table

Figure 4 creates the reviews table which contains information about reviews received by each hospital. rev_id is assigned as the primary key and like in *Figure 3* provider id is inserted as a foreign key linking the review and hospital info table in a one-to-one relationship.

```
c.execute('''
INSERT INTO hospital_info (provider_id, hospital_name, city, state,
    county, type, ownership, rating)
SELECT DISTINCT Hospital.`Provider ID`, Hospital.`Hospital Name`, Hospital.City,
    Hospital.State, Hospital.`County Name`, Hospital.`Hospital Type`,
    Hospital.`Hospital Ownership`, Hospital.`Hospital overall rating`
FROM Hospital
''')
```

Figure 5 Inserting data into the hospital info table

Figure 5 shows how the hospital info table is populated by data from the original hospital dataset. This process is similar for both the quality of care and reviews table. A complete view of the database creation is shown in the *Code Appendix* section.

This implementation satisfies both the proper SQL operation wrapper and database file creation criteria outlined in the brief. The database elements of the project were programmed in python with the database file being viewed using DB Browser for SQLite.

4. Class Design

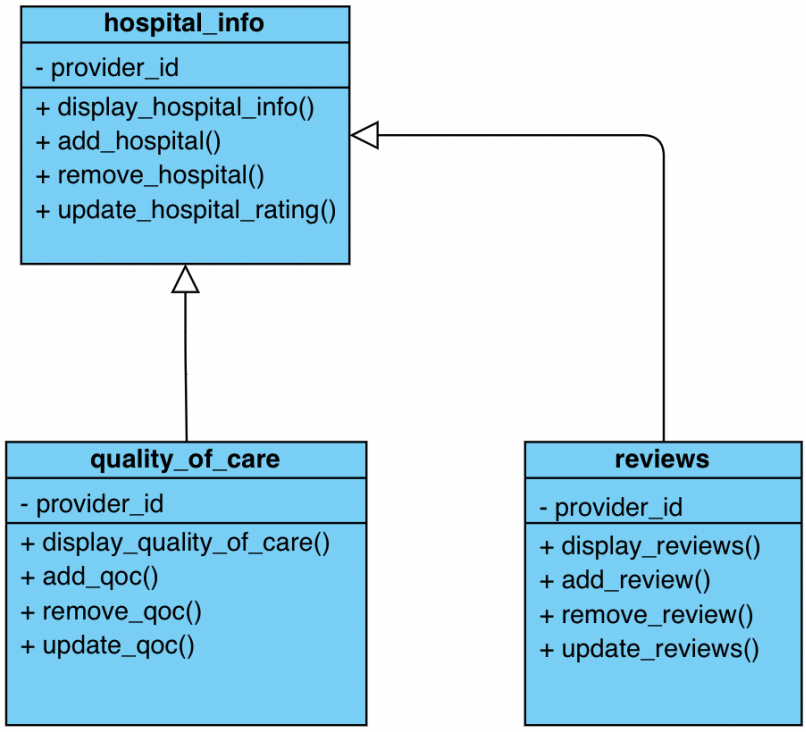


Figure 6 Class Design

5. Class Functionality

The three classes hospital, quality-of-care, and reviews each contain methods that communicate with the database to display, add, remove, and update data in the corresponding tables. Hospital is the base class with quality-of-care and reviews the two child classes. The child classes are set up each with an inheritance relationship to the base hospital class. An overall view of the class architecture is displayed in *Figure 6.* This section will explain the functionality of the base hospital class and how the class is implemented using code snippets of essential methods.

```
def __init__(self, providerid):
    self.providerid = providerid

def display_hospital_info(self):
    c.execute('''SELECT provider_id, hospital_name, city, state, county,
                     type, ownership, rating
                 FROM hospital_info WHERE provider_id=:providerid''',
                     {'providerid': self.providerid})
    hos_info = c.fetchall()
    return hos_info
```

Figure 7 Display hospital info class

The __init__ is a constructor method that is called when a hospital object is created and initialises the provider id class attribute. The display_hospital_info class uses the provider id passed in as a parameter by the user to fetch records from the hospital_info table that match the provider id. The matching record is fetched using a wrapped SQL query in python and returns the hospital provider id, name, city, state, county, type, ownership, and rating.

```
def add_hospital(self):
    hos_id = input("Enter hospital provider id: ")
    c.execute('SELECT provider_id FROM hospital_info WHERE provider_id=:hos_id',
                {'hos_id': hos_id})
    fetch_id = c.fetchone()

    if fetch_id != None:
        print('Hospital with id:', hos_id, "already exists")
    else:
        hos_name = input("Enter hospital name: ")
        hos_city = input("Enter hospital city: ")
        hos_state = input("Enter hospital state: ")
        hos_county = input("Enter hospital county: ")
        hos_type = input("Enter hospital type: ")
        hos_owner = input("Enter hospital owner: ")
        hos_rating = input("Enter hospital rating: ")

        c.execute('''INSERT INTO hospital_info(provider_id, hospital_name,
            city, state, county,
            type, ownership, rating)
            VALUES(?,?,?,?,?,?,?,?)''', (hos_id, hos_name, hos_city, hos_state,
            hos_county, hos_type, hos_owner, hos_rating))
        conn.commit()
```

Figure 8 Add hospital class

The add_hospital method in *Figure 8* adds a record to the hospital_info table using information inputted by the user. First, the user is asked to enter the provider id of the hospital they wish to add. A query is then performed to find records that match the provider id and if the result of that query returns an output a check informs the user a hospital with that id already exists - this increases the overall robustness of the code. If the provider id check returns no results the user is required to enter values for hospital name, city, county, type, owner and rating as input rather than parameters. Once values corresponding to each column of the hospital_info table are entered a SQL insert query is executed to add the new record to the database.

```
def remove_hospital(self):
    hos_id = input("Enter hospital provider id: ")

    c.execute('SELECT provider_id FROM hospital_info WHERE provider_id=:hos_id',
              {'hos_id': hos_id})
    fetch_id = c.fetchone()

    if fetch_id == None:
        print('Hospital information cannot be removed as it does not exist')
    else:
        c.execute('DELETE FROM hospital_info WHERE provider_id=:hos_id',
                  {'hos_id': hos_id})
        print("Hospital information sucessfully removed")
    conn.commit()
```

Figure 9 Remove hospital class

The remove_hospital method in *Figure 9* allows the user to remove a particular hospital record from the hospital_info table. The user is first asked for the provider id of the hospital they wish to remove. Another check runs the same SQL query as *Figure 8* to fetch any records matching the inputted provider id. However, this check ensures that the query output is empty as you cannot remove an entry that does not already exist in the table. If the check passes and the query returns a record matching the user entered provider id, a SQL delete query is executed to remove that hospital record from the table. A print statement provides feedback to the user informing them that the hospital information was successfully removed.

```
def update_hospital_rating(self):
    hos_id = input("Enter hospital provider id: ")
    c.execute('SELECT provider_id FROM hospital_info WHERE provider_id=:hos_id',
              {'hos_id': hos_id})
    fetch_id = c.fetchone()

    if fetch_id == None:
        print('Hospital with id:', hos_id, "doesn't exist")
    else:
        hos_rating = input("Enter new hospital rating: ")

        c.execute('UPDATE hospital_info SET rating=? WHERE provider_id=?',
                  (hos_rating, hos_id))
        conn.commit()
        print("Hospital rating sucessfully updated")
```

Figure 10 Update hospital class

The update_hospital_rating method in *Figure 10* allows to user to update an existing record from the hospital_info table. The user is asked to enter the provider id of the record they wish to update. A check ensures the user can't update a record that doesn't exist. If the SQL query check for that provider id returns an existing record the user is asked to enter a new value for only the hospital rating. The decision to only allow users to update the hospital rating was made as the other values associated with the hospital_info table aren't likely to be subject to change i.e., a hospital isn't going to change state. Once the user enters the new hospital rating a SQL update query is executed to commit this change to the record matching the provider id. Additional feedback informs the user that the rating was successfully updated.

The quality-of-care and review classes are like the hospital class in their implementation. They each contain methods for displaying, adding, removing, and updating records from their associated table in the database. The main difference is the SQL queries and input values which differ to match the columns and information in the reviews and quality-of-care tables. These classes also behave differently as being the child classes they implement the meaningful inheritance relationship outlined in the brief.

```
class reviews(hospital):
    def __init__(self, hospit):
        self.providerid = hospit.providerid

class quality_of_care(hospital):
    def __init__(self, hospit):
        self.providerid = hospit.providerid
```

Figure 11 Reviews and quality of care constructor methods

Figure 11 shows how the variable provider id from the base class hospital is inherited by the child classes reviews and quality-of-care. Apart from these differences the reviews and quality-of-care classes behave mostly the same as parent hospital class, implementing the same code checks and SQL queries. Therefore, screenshots with descriptions of their class methods are unnecessary as the same logic outlined throughout this section applies to both the reviews and quality-of-care classes. Instead, the full code listings for all three classes will be provided in the *Code Appendix* section.

```
if __name__ == "__main__":

    hos = hospital(10055)
    print(hos.display_hospital_info())
    #hos.add_hospital()
    #hos.remove_hospital()
    #hos.update_hospital_rating()

    rev = reviews(hos)
    print(rev.display_reviews())
    #rev.add_review()
    #rev.remove_review()
    #rev.update_reviews()

    qoc = quality_of_care(hos)
    print(qoc.display_quality_of_care())
    #qoc.add_qoc()
    #qoc.remove_qoc()
    #qoc.update_qoc()
```

Figure 12 Class functions

```
[(10055, 'FLOWERS HOSPITAL', 'DOTHAN', 'AL', 'HOUSTON', 'Acute Care Hospitals', 'Proprietary', 4)]
[(10055, 99, '24/08/2019', 1.28)]
[(10055, 'Below the national average', 'Above the national average', 'Above the national average', 'Above the natio
nal average', 'Same as the national average', 'Above the national average', 'Same as the national average')]
```

Figure 13 Demo function output

Figure 12 shows the functions associated with each of the classes. In this case, only the display function for each of the classes is run for demo purposes with the result of each displayed in *Figure 13.* This section, therefore, shows how both the classes implementation and demo function criteria in the project brief was fulfilled.

6. Code Testing

Testing some code snippets was another requirement outlined in the project brief. Unit testing allows users to understand the functionality provided by individual units and functions. To demonstrate how unit testing could be applied to this project, a unit test was run for one method of each class. *Figure 14* shows how the assertEqual() from Pythons unit test library was used to check the equality of two values. In this case, the test checks that the actual result of calling the display method of each class matches the expected result of each class, returning true if the test passes. *Figure 14* also displays the results of the three tests which indicate the tests pass and the class methods function as expected.

```
import unittest

class testing(unittest.TestCase):

    def test_displaying_hospital(self):
        expected = [(10055, 'FLOWERS HOSPITAL', 'DOTHAN', 'AL', 'HOUSTON', 'Acute Care Hospitals', 'Proprietary', 4)
        actual = hos.display_hospital_info()

        self.assertEqual(expected, actual)

    def test_displaying_reviews(self):
        expected = [(10055, 99, '24/08/2019', 1.28)]
        actual = rev.display_reviews()

        self.assertEqual(expected, actual)

    def test_displaying_qoc(self):
        expected = [(10055, 'Below the national average', 'Above the national average', 'Above the national average'
        actual = qoc.display_quality_of_care()

        self.assertEqual(expected, actual)

if __name__ == '__main__':
    unittest.main(argv=['first-arg-is-ignored'], exit=False)
```

```
...
----------------------------------------------------------------------
Ran 3 tests in 0.004s

OK
```

Figure 14 Unit testing class

7. Data Analysis

Analysing the data in the database file allows us to discover useful information and suggest conclusions about hospitals throughout the USA. The first section describes some basic analyses of hospitals and their associated overall rating using SQL. The following section will propose a question about the data which will be explored using pandas. Pandas will be utilised to inspect, clean, and transform the data with the hope of informing a conclusion regarding the initial question posed.

7.1 Data Analysis using SQL

The analysis using SQL proposed three questions, answered using three separate queries which were:

- Which hospital ownership group had the highest overall rating
- What were the top 5 states for overall hospital rating?
- For the state with the highest average rating, what were the top three highest-rated counties?

```
c.execute('''
SELECT ownership, AVG(rating) AS avg_rating
FROM hospital_info
GROUP BY ownership
ORDER BY avg_rating DESC
LIMIT 1
''')
print(c.fetchall())

c.execute('''
SELECT state, AVG(rating) AS avg_rating
FROM hospital_info
GROUP BY state
ORDER BY avg_rating DESC
LIMIT 5
''')
print(c.fetchall())

c.execute('''
SELECT state, county, AVG(rating) AS avg_rating
FROM hospital_info
WHERE state = 'SD'
GROUP BY county
ORDER BY avg_rating DESC
LIMIT 3
''')
print(c.fetchall())
```

Query Results:

- Physician-owned hospitals had the highest average rating
- The states with the highest average rating (in order) are South Dakota, Delaware, Wisconsin, Minnesota, and Idaho (top 5)
- Within South Dakota, the counties with the highest average rating (in order) are Brown, Minnehaha, and Yankton (top 3)

7.2 Data Analysis using Pandas

After the initial analysis in *Section 7.1,* I wanted to perform a deeper analysis of the data to establish which ownership group has the highest proportion of its hospitals with the overall quality of care better than the national average. The analysis utilises Pandas which is a powerful and flexible data analysis and manipulation tool in python. The data analysis unlike the rest of the source code is written in Jupyter Notebook.

```
hospital_df = pd.read_sql_query('''
SELECT hospital_info.provider_id, hospital_info.hospital_name, hospital_info.ownership, hospital_info.rating,
   quality_of_care.mortality_nat_comparison, quality_of_care.safety_of_care_nat_comparison,
   quality_of_care.readmission_nat_comparison, quality_of_care.patient_experience_nat_comparison,
   quality_of_care.effectiveness_of_care_nat_comparison, quality_of_care.timeliness_of_care_nat_comparison,
   quality_of_care.efficient_imaging_nat_comparison
FROM hospital_info
LEFT JOIN quality_of_care on hospital_info.provider_id = quality_of_care.provider_id
''', conn)
```

The read_sql_query() function is used to return a DataFrame corresponding to the result set of the query string. The query string joins the hospital_info and quality_of_care table and returns the relevant column values necessary to answer the proposed analysis question. The query result is saved into a new DataFrame hospital_df.

```
hospital_df.head()
```

	provider_id	hospital_name	ownership	rating	mortality_nat_comparison	safety_of_care_nat_comparison	readmission_nat_comparison	patient_experience_n
0	10001	SOUTHEAST ALABAMA MEDICAL CENTER	Government - Hospital District or Authority	3	Same as the national average	Above the national average	Same as the national average	Below the r
1	10005	MARSHALL MEDICAL CENTER SOUTH	Government - Hospital District or Authority	3	Below the national average	Same as the national average	Above the national average	Same as the r
2	10006	ELIZA COFFEE MEMORIAL HOSPITAL	Government - Hospital District or Authority	2	Below the national average	Same as the national average	Same as the national average	Below the r
3	10007	MIZELL MEMORIAL HOSPITAL	Voluntary non-profit - Private	2	Same as the national average	Not Available	Below the national average	Same as the r
4	10008	CRENSHAW COMMUNITY	Proprietary	3	Same as the national	Not Available	Same as the national average	

The head() function by default returns the first 5 rows of hospital_df and is useful to give the user an insight into the structure and values present in the DataFrame.

3.1 Dealing with missing data

```
hospital_df.replace("Not Available", np.nan, inplace=True)

hospital_df.dropna(how='all', inplace=True)
```

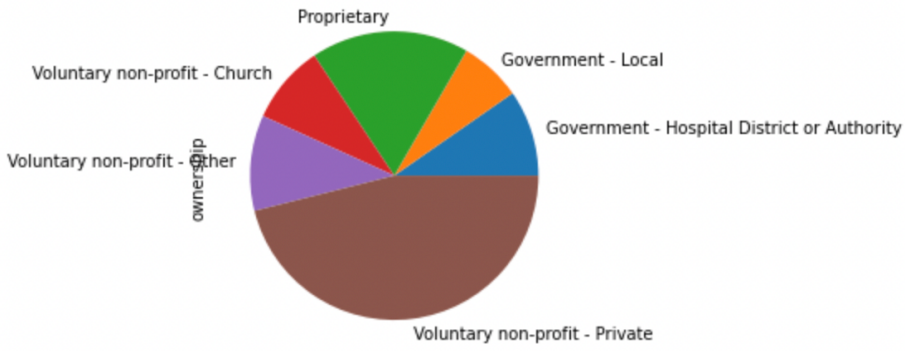
3.2 Removing hospital ownerships with small sample size

```
df2 = hospital_df.groupby('ownership')['ownership'].count()
df200 = df2[df2 < 200]

df200
```

```
ownership
Government - Federal    16
Government - State      45
Physician               20
Tribal                   2
Name: ownership, dtype: int64
```

```
remove_owners = ['Government - Federal', 'Government - State', 'Physician', 'Tribal']

clean_data = hospital_df[hospital_df.ownership.isin(remove_owners) == False]
```

The next data analysis stage is to deal with missing values and data cleaning. Frist any rows with missing values for all columns are removed. Next, groups that own less than 200 hospitals are removed from the next stages of analysis as they each have a small sample size compared with the remaining hospital owners. Not removing these owners from the analysis would reduce the effectiveness of the study and increase the margin of error which could render the results of the analysis meaningless. The distribution of the remaining hospital owner groups is visualised below.



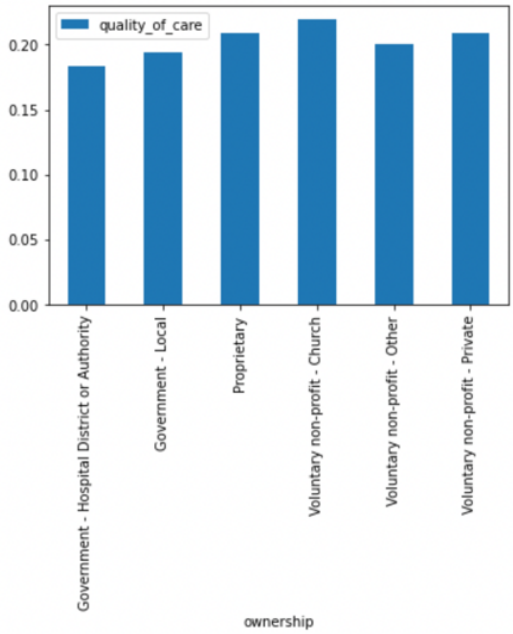
After creating, inspecting, and cleaning the hospital_df DataFrame, the next stage is to transform and manipulate the data. First, the proportion of each quality-of-care measurement (patient experience for example) above, below, and same as the national average grouped by hospital ownership is calculated. Next only the proportions above the national average are selected except in the case of mortality and readmission. As we wish to analyse quality-of-care better than the national average being above the national average is better for measurements such as patient experience but for mortality and readmission having these measurements below the national average is classified as being better.

	ownership	patient_experience_nat_comparison	pat_exp
1	Government - Hospital District or Authority	Above the national average	0.305970
3	Government - Local	Above the national average	0.368421
8	Proprietary	Above the national average	0.179966
9	Voluntary non-profit - Church	Above the national average	0.383562
12	Voluntary non-profit - Other	Above the national average	0.379501
16	Voluntary non-profit - Private	Above the national average	0.360161

The national comparison column is then dropped for each of the seven quality-of-care measurement DataFrames which are then combined to produce a single DataFrame with the ownership group and the proportion of quality-of-care better than the national average for each group. To find the average proportion of quality-of-care better than the national average the individual proportions are summed together and divided by 7 for each ownership group

	ownership	pat_exp	soc	eoc	time	eff_img	mort	read	sum	quality_of_care
0	Government - Hospital District or Authority	0.305970	0.262295	0.028846	0.301587	0.113122	0.132013	0.143713	1.287547	0.183935
1	Government - Local	0.368421	0.198413	0.009901	0.411215	0.065868	0.095455	0.208333	1.357606	0.193944
2	Proprietary	0.179966	0.348837	0.028239	0.377551	0.065022	0.112263	0.347754	1.459632	0.208519
3	Voluntary non-profit - Church	0.383562	0.301961	0.050336	0.273038	0.143396	0.137124	0.245955	1.535370	0.219339
4	Voluntary non-profit - Other	0.379501	0.265781	0.057377	0.261765	0.163399	0.089337	0.187151	1.404311	0.200616
5	Voluntary non-profit - Private	0.360161	0.297479	0.040640	0.279303	0.153485	0.091628	0.240554	1.463250	0.209036

This average quality-of-care proportion for each ownership group is plotted to give the final analysis results. They suggest hospitals owned by Voluntary non-profit - Church have the highest proportion of its hospitals with the overall quality of care better than the national average. This section details some essential code result snippets from the analysis and an interpretation of the results in the context of the research question. A complete version of the analysis with code can be found in the *Code Appendix* Section of the project.



8. Draft QUB Azure HDInsight Plan

This section creates a draft plan to migrate the current project implementation to QUB's Azure HDInsight. The first step would be to configure a new cluster by creating a new resource group, unique cluster name and cluster type – qubAssignment, hospitals, and Hadoop 2.7.3 for example. Other options such as region, subscription, username, and password can also be configured. To finish configuring a new cluster the storage options must be completed which include setting a primary storage type and account, selection method and container. After reviewing the security and networking, configuration and price screens which can be left unchanged the cluster build can be initiated.

Once the cluster build is complete, the progress of which can be monitored from a link in the Azure portal home screen, a deployment succeeded message is received. Upon entering the new resource, you should see the HDInsight build screen. Go to the Hadoop portal for the created cluster and when in the Ambari view select Hive View 2.0.

In the Hive view, the tables for hospital_info, reviews, and quality_of_care can be created. Click on the green plus sign then upload each of the tables by selecting the file source from the appropriate location on your computer. Since data file uploading requires files to be in an appropriate format the current project implementation would need to be changed to save the three data tables to CSV format for example. Once this formatting step is complete and the files are uploaded the tables can be created. Data analysis and run queries, like those in *Section 7,* can now be performed on the dataset.

Code Appendix

create_database.py

```
import sqlite3
import pandas as pd
from pandas import DataFrame

conn = sqlite3.connect("Hospital.db")
c = conn.cursor()

c.executescript('''
   DROP TABLE IF EXISTS hospital_info;
   DROP TABLE IF EXISTS quality_of_care;
   DROP TABLE IF EXISTS reviews;
   ''')

c.execute('''CREATE TABLE IF NOT EXISTS hospital_info
      (
      provider_id INTEGER PRIMARY KEY,
      hospital_name TEXT,
      city TEXT,
      state TEXT,
      county TEXT,
      type TEXT,
      ownership TEXT,
      rating INTEGER
      );''')

c.execute('''CREATE TABLE IF NOT EXISTS quality_of_care
      (
      qoc_id INTEGER PRIMARY KEY AUTOINCREMENT NOT NULL,
      provider_id INTEGER,
      mortality_nat_comparison TEXT,
      safety_of_care_nat_comparison TEXT,
      readmission_nat_comparison TEXT,
      patient_experience_nat_comparison TEXT,
      effectiveness_of_care_nat_comparison TEXT,
      timeliness_of_care_nat_comparison TEXT,
      efficient_imaging_nat_comparison TEXT,
      FOREIGN KEY (provider_id) REFERENCES hospital_info(provider_id)
      );''')
```

DSA8002 (2021-2022)

```
c.execute('''CREATE TABLE IF NOT EXISTS reviews
       (
        rev_id INTEGER PRIMARY KEY AUTOINCREMENT NOT NULL,
        provider_id INTEGER,
        number_of_reviews INTEGER,
        last_review INTEGER,
        reviews_per_month INTEGER,
        FOREIGN KEY (provider_id) REFERENCES hospital_info(provider_id)
       );''')

conn.commit()

read_hospital_data = pd.read_csv(
   'DSA8002 (2021-2022)-dataset.csv', encoding='unicode_escape')
read_hospital_data.to_sql("Hospital", conn, if_exists='append', index=False)

conn.commit()

c.execute('''
INSERT INTO hospital_info (provider_id, hospital_name, city, state,
   county, type, ownership, rating)
SELECT DISTINCT Hospital.`Provider ID`, Hospital.`Hospital Name`, Hospital.City,
   Hospital.State, Hospital.`County Name`, Hospital.`Hospital Type`,
   Hospital.`Hospital Ownership`, Hospital.`Hospital overall rating`
FROM Hospital
''')

c.execute('''
INSERT INTO quality_of_care(provider_id,
   mortality_nat_comparison, safety_of_care_nat_comparison,
   readmission_nat_comparison, patient_experience_nat_comparison,
   effectiveness_of_care_nat_comparison, timeliness_of_care_nat_comparison,
   efficient_imaging_nat_comparison)
SELECT DISTINCT Hospital.`Provider ID`,
   Hospital.`Mortality national comparison`, Hospital.`Safety of care national comparison`,
   Hospital.`Readmission national comparison`, Hospital.`Patient experience national
comparison`,
   Hospital.`Effectiveness of care national comparison`, Hospital.`Timeliness of care national
comparison`,
   Hospital.`Efficient use of medical imaging national comparison`
FROM Hospital
''')
```

```
c.execute('''
INSERT INTO reviews(provider_id, number_of_reviews, last_review, reviews_per_month)
SELECT DISTINCT Hospital.`Provider ID`, Hospital.`Hospital-number_of_reviews`,
   Hospital.last_review, Hospital.reviews_per_month
FROM Hospital
''')

c.execute('''
DELETE FROM hospital_info
WHERE rating ='Not Available'
''')

conn.commit()
```

classes+testing.py

```
import sqlite3

conn = sqlite3.connect("Hospital.db")
c = conn.cursor()

class hospital:
   def __init__(self, providerid):
      self.providerid = providerid

   def display_hospital_info(self):
      c.execute('''SELECT provider_id, hospital_name, city, state, county,
                  type, ownership, rating
               FROM hospital_info WHERE provider_id=:providerid''',
                  {'providerid': self.providerid})
      hos_info = c.fetchall()
      return hos_info

   def add_hospital(self):
      hos_id = input("Enter hospital provider id: ")
      c.execute('SELECT provider_id FROM hospital_info WHERE provider_id=:hos_id',
                {'hos_id': hos_id})
      fetch_id = c.fetchone()

      if fetch_id != None:
         print('Hospital with id:', hos_id, "already exists")
      else:
         hos_name = input("Enter hospital name: ")
         hos_city = input("Enter hospital city: ")
         hos_state = input("Enter hospital state: ")
         hos_county = input("Enter hospital county: ")
         hos_type = input("Enter hospital type: ")
```

```
        hos_owner = input("Enter hospital owner: ")
        hos_rating = input("Enter hospital rating: ")

        c.execute('''INSERT INTO hospital_info(provider_id, hospital_name,
          city, state, county,
          type, ownership, rating)
          VALUES(?,?,?,?,?,?,?,?)''', (hos_id, hos_name, hos_city, hos_state,
          hos_county, hos_type, hos_owner, hos_rating))
        conn.commit()

    def remove_hospital(self):
      hos_id = input("Enter hospital provider id: ")

      c.execute('SELECT provider_id FROM hospital_info WHERE provider_id=:hos_id',
                {'hos_id': hos_id})
      fetch_id = c.fetchone()

      if fetch_id == None:
        print('Hospital information cannot be removed as it does not exist')
      else:
        c.execute('DELETE FROM hospital_info WHERE provider_id=:hos_id',
                {'hos_id': hos_id})
        print("Hospital information sucessfully removed")
      conn.commit()

    def update_hospital_rating(self):
      hos_id = input("Enter hospital provider id: ")
      c.execute('SELECT provider_id FROM hospital_info WHERE provider_id=:hos_id',
                {'hos_id': hos_id})
      fetch_id = c.fetchone()

      if fetch_id == None:
        print('Hospital with id:', hos_id, "doesn't exist")
      else:
        hos_rating = input("Enter new hospital rating: ")

        c.execute('UPDATE hospital_info SET rating=? WHERE provider_id=?',
                  (hos_rating, hos_id))
        conn.commit()
        print("Hospital rating sucessfully updated")

class reviews(hospital):
    def __init__(self, hospit):
      self.providerid = hospit.providerid

    def display_reviews(self):
      c.execute('''SELECT provider_id, number_of_reviews, last_review, reviews_per_month
```

```
        FROM reviews WHERE provider_id=:providerid''',
          {'providerid': self.providerid})
    rev_info = c.fetchall()
    return rev_info

  def add_review(self):
    rev_id = input("Enter hospital provider id: ")
    c.execute('SELECT provider_id FROM reviews WHERE provider_id=:rev_id',
              {'rev_id': rev_id})
    fetch_id = c.fetchone()

    if fetch_id != None:
      print('Hospital with id:', rev_id, "already exists")
    else:
      rev_num_reviews = input("Enter number of hospital reviews: ")
      rev_last_review = input("Enter last hospital review: ")
      rev_per_month = input("Enter hospital reviews per month: ")

      c.execute('''INSERT INTO reviews(provider_id, number_of_reviews,
        last_review, reviews_per_month)
        VALUES(?,?,?,?)''', (rev_id, rev_num_reviews, rev_last_review, rev_per_month))
      conn.commit()

  def remove_review(self):
    rev_id = input("Enter hospital provider id: ")

    c.execute('SELECT provider_id FROM reviews WHERE provider_id=:rev_id',
              {'rev_id': rev_id})
    fetch_id = c.fetchone()

    if fetch_id == None:
      print('Hospital reviews cannot be removed as it does not exist')
    else:
      c.execute('DELETE FROM reviews WHERE provider_id=:rev_id',
              {'rev_id': rev_id})
      print("Hospital reviews sucessfully removed")
    conn.commit()

  def update_reviews(self):
    rev_id = input("Enter hospital provider id: ")
    c.execute('SELECT provider_id FROM reviews WHERE provider_id=:rev_id',
              {'rev_id': rev_id})
    fetch_id = c.fetchone()

    if fetch_id == None:
      print('Hospital with id:', rev_id, "doesn't exist")
    else:
```

```
        rev_num_reviews = input("Enter number of hospital reviews: ")
        rev_last_review = input("Enter last hospital review: ")
        rev_per_month = input("Enter hospital reviews per month: ")

        c.execute('''UPDATE reviews
                SET number_of_reviews=?, last_review=?, reviews_per_month=?
                WHERE provider_id=?''',
                (rev_num_reviews, rev_last_review, rev_per_month, rev_id))
        conn.commit()
        print("Hospital reviews sucessfully updated")

class quality_of_care(hospital):
   def __init__(self, hospit):
      self.providerid = hospit.providerid

   def display_quality_of_care(self):
      c.execute('''SELECT provider_id,mortality_nat_comparison,
safety_of_care_nat_comparison,
            readmission_nat_comparison, patient_experience_nat_comparison,
            effectiveness_of_care_nat_comparison, timeliness_of_care_nat_comparison,
            efficient_imaging_nat_comparison
          FROM quality_of_care WHERE provider_id=:providerid''',
          {'providerid': self.providerid})
      qoc_info = c.fetchall()
      return qoc_info

   def add_qoc(self):
      qoc_id = input("Enter hospital provider id: ")
      c.execute('SELECT provider_id FROM quality_of_care WHERE provider_id=:qoc_id',
              {'qoc_id': qoc_id})
      fetch_id = c.fetchone()

      if fetch_id != None:
         print('Hospital with id:', qoc_id, "already exists")
      else:
         qoc_mortality = input("Enter hospital mortaility national comparison: ")
         qoc_safety = input("Enter hospital safety national comparison: ")
         qoc_readmission = input("Enter hospital readmission national comparison: ")
         qoc_pat_exp = input("Enter hospital patient experience national comparison: ")
         qoc_eff_of_care = input("Enter hospital effectiveness of care national comparison: ")
         qoc_timeliness = input("Enter hospital timeliness national comparison: ")
         qoc_eff_img = input("Enter hospital efficient imaging national comparison: ")

         c.execute('''INSERT INTO quality_of_care(provider_id,mortality_nat_comparison,
              safety_of_care_nat_comparison, readmission_nat_comparison,
              patient_experience_nat_comparison, effectiveness_of_care_nat_comparison,
              timeliness_of_care_nat_comparison, efficient_imaging_nat_comparison)
```

```
            VALUES(?,?,?,?,?,?,?,?)''', (qoc_id, qoc_mortality, qoc_safety, qoc_readmission,
            qoc_pat_exp, qoc_eff_of_care, qoc_timeliness, qoc_eff_img))
        print('Hospital quality of care with id:', qoc_id, "successfully added")
        conn.commit()

    def remove_qoc(self):
        qoc_id = input("Enter hospital provider id: ")

        c.execute('SELECT provider_id FROM quality_of_care WHERE provider_id=:qoc_id',
                  {'qoc_id': qoc_id})
        fetch_id = c.fetchone()

        if fetch_id == None:
            print('Hospital quality of care information cannot be removed as it does not exist')
        else:
            c.execute('DELETE FROM quality_of_care WHERE provider_id=:qoc_id',
                  {'qoc_id': qoc_id})
            print("Hospital quality of care information sucessfully removed")
        conn.commit()

    def update_qoc(self):
        qoc_id = input("Enter hospital provider id: ")
        c.execute('SELECT provider_id FROM quality_of_care WHERE provider_id=:qoc_id',
                  {'qoc_id': qoc_id})
        fetch_id = c.fetchone()

        if fetch_id == None:
            print('Hospital with id:', qoc_id, "doesn't exist")
        else:
            qoc_mortality = input("Enter new mortaility national comparison: ")
            qoc_safety = input("Enter new safety national comparison: ")
            qoc_readmission = input("Enter new readmission national comparison: ")
            qoc_pat_exp = input("Enter new patient experience national comparison: ")
            qoc_eff_of_care = input("Enter new effectiveness of care national comparison: ")
            qoc_timeliness = input("Enter new timeliness national comparison: ")
            qoc_eff_img = input("Enter new efficient imaging national comparison: ")

            c.execute('''UPDATE quality_of_care
                      SET mortality_nat_comparison=?, safety_of_care_nat_comparison=?,
                        readmission_nat_comparison=?, patient_experience_nat_comparison=?,
                        effectiveness_of_care_nat_comparison=?,
timeliness_of_care_nat_comparison=?,
                        efficient_imaging_nat_comparison=?
                      WHERE provider_id=?''',
                      (qoc_mortality, qoc_safety, qoc_readmission, qoc_pat_exp, qoc_eff_of_care,
qoc_timeliness, qoc_eff_img, qoc_id))
        conn.commit()
```

```
        print("Hospital quality of care sucessfully updated")

if __name__ == "__main__":

    hos = hospital(10055)
    print(hos.display_hospital_info())
    #hos.add_hospital()
    #hos.remove_hospital()
    #hos.update_hospital_rating()

    rev = reviews(hos)
    print(rev.display_reviews())
    #rev.add_review()
    #rev.remove_review()
    #rev.update_reviews()

    qoc = quality_of_care(hos)
    print(qoc.display_quality_of_care())
    #qoc.add_qoc()
    #qoc.remove_qoc()
    #qoc.update_qoc()

import unittest

class testing(unittest.TestCase):

    def test_displaying_hospital(self):
        expected = [(10055, 'FLOWERS HOSPITAL', 'DOTHAN', 'AL', 'HOUSTON', 'Acute Care
Hospitals', 'Proprietary', 4)]
        actual = hos.display_hospital_info()

        self.assertEqual(expected, actual)

    def test_displaying_reviews(self):
        expected = [(10055, 99, '24/08/2019', 1.28)]
        actual = rev.display_reviews()

        self.assertEqual(expected, actual)

    def test_displaying_qoc(self):
        expected = [(10055, 'Below the national average', 'Above the national average', 'Above
the national average', 'Above the national average', 'Same as the national average', 'Above
the national average', 'Same as the national average')]
        actual = qoc.display_quality_of_care()

        self.assertEqual(expected, actual)
```

```
if __name__ == '__main__':
    unittest.main(argv=['first-arg-is-ignored'], exit=False)
```

data_analysis_sql.py

```
import sqlite3

conn = sqlite3.connect("Hospital.db")
c = conn.cursor()

c.execute('''
SELECT ownership, AVG(rating) AS avg_rating
FROM hospital_info
GROUP BY ownership
ORDER BY avg_rating DESC
LIMIT 1
''')
print(c.fetchall())

c.execute('''
SELECT state, AVG(rating) AS avg_rating
FROM hospital_info
GROUP BY state
ORDER BY avg_rating DESC
LIMIT 5
''')
print(c.fetchall())

c.execute('''
SELECT state, county, AVG(rating) AS avg_rating
FROM hospital_info
WHERE state = 'SD'
GROUP BY county
ORDER BY avg_rating DESC
LIMIT 3
''')
print(c.fetchall())

conn.close()
```

data_analysis_pandas.ipynb

```
#!/usr/bin/env python
# coding: utf-8

# # Hospital Ownership and Quality of Care

# ## Aim of analysis

# To assess which ownership group has the highest proportion of its hospitals with overall quality of care better than the national average

# ### 1. Processing relevant data from database

# In[ ]:
import sqlite3
import pandas as pd
import numpy as np
from functools import reduce
pd.options.mode.chained_assignment = None

conn = sqlite3.connect("Hospital.db")
c = conn.cursor()

hospital_df = pd.read_sql_query('''
SELECT hospital_info.provider_id, hospital_info.hospital_name, hospital_info.ownership,
hospital_info.rating,
  quality_of_care.mortality_nat_comparison,
quality_of_care.safety_of_care_nat_comparison,
  quality_of_care.readmission_nat_comparison,
quality_of_care.patient_experience_nat_comparison,
  quality_of_care.effectiveness_of_care_nat_comparison,
quality_of_care.timeliness_of_care_nat_comparison,
  quality_of_care.efficient_imaging_nat_comparison
FROM hospital_info
LEFT JOIN quality_of_care on hospital_info.provider_id = quality_of_care.provider_id
''', conn)

conn.close()
```

2. Data information

# In[ ]:
hospital_df.head()

# In[ ]:
hospital_df.info()

In[ ]:
hospital_df.shape

2.1 Displaying and visualising the average rating of each ownership group

# In[ ]:
avg_rating = hospital_df.groupby('ownership')['rating'].mean()
avg_rating.sort_values(ascending = False)

# In[ ]:
avg_rating.plot(kind = 'bar')

3. Cleaning data

3.1 Dealing with missing data

# In[ ]:
hospital_df.replace("Not Available", np.nan, inplace=True)

hospital_df.dropna(how='all', inplace=True)

3.2 Removing hospital ownerships with small sample size

# In[ ]:
df2 = hospital_df.groupby('ownership')['ownership'].count()
df200 = df2[df2 < 200]
df200

# In[ ]:
remove_owners = ['Government - Federal', 'Government - State', 'Physician', 'Tribal']

clean_data = hospital_df[hospital_df.ownership.isin(remove_owners) == False]

3.3 Displaying the number of hospitals owned by the remaining groups

# In[ ]:
count_owners = clean_data.groupby('ownership')['ownership'].count()

count_owners
3.4 Visualising the number of hospitals owned by each group

# In[ ]:
count_owners.plot(kind='pie')

4. Data analysis

4.1 Calculating the proportion of each quality of care measurement below, above and same as the national average grouped by hospital ownership

In[ ]:
pat_exp_df =
clean_data.groupby('ownership')['patient_experience_nat_comparison'].value_counts(normalize=True)
mort_df =
clean_data.groupby('ownership')['mortality_nat_comparison'].value_counts(normalize=True)
soc_df =
clean_data.groupby('ownership')['safety_of_care_nat_comparison'].value_counts(normalize=True)
read_df =
clean_data.groupby('ownership')['readmission_nat_comparison'].value_counts(normalize=True)
eoc_df =
clean_data.groupby('ownership')['effectiveness_of_care_nat_comparison'].value_counts(normalize=True)
time_df =
clean_data.groupby('ownership')['timeliness_of_care_nat_comparison'].value_counts(normalize=True)
eff_img_df =
clean_data.groupby('ownership')['efficient_imaging_nat_comparison'].value_counts(normalize=True)

n1 = pat_exp_df.reset_index(name = 'pat_exp')
n2 = soc_df.reset_index(name = 'soc')
n3 = eoc_df.reset_index(name = 'eoc')
n4 = time_df.reset_index(name = 'time')
n5 = eff_img_df.reset_index(name = 'eff_img')

n6 = mort_df.reset_index(name = 'mort')
n7 = read_df.reset_index(name = 'read')
n1

4.2 Selecting proportions for 5 quality of care measurements that are above the national average

In[ ]:
```
above1 = n1[n1['patient_experience_nat_comparison'] == "Above the national average"]
above2 = n2[n2['safety_of_care_nat_comparison'] == "Above the national average"]
above3 = n3[n3['effectiveness_of_care_nat_comparison'] == "Above the national average"]
above4 = n4[n4['timeliness_of_care_nat_comparison'] == "Above the national average"]
above5 = n5[n5['efficient_imaging_nat_comparison'] == "Above the national average"]

above1
```

4.3 Selecting proportions for the 2 remaining quality of care measurements that are below the national average

In[ ]:
```
below6 = n6[n6['mortality_nat_comparison'] == "Below the national average"]
below7 = n7[n7['readmission_nat_comparison'] == "Below the national average"]

below6
```

4.4 Dropping the national comparison for each quality of care measurement

In[ ]:
```
above1.drop('patient_experience_nat_comparison', axis=1, inplace=True)
above2.drop('safety_of_care_nat_comparison', axis = 1, inplace=True)
above3.drop('effectiveness_of_care_nat_comparison', axis = 1, inplace=True)
above4.drop('timeliness_of_care_nat_comparison', axis = 1, inplace=True)
above5.drop('efficient_imaging_nat_comparison', axis = 1, inplace=True)

below6.drop('mortality_nat_comparison', axis = 1, inplace=True)
below7.drop('readmission_nat_comparison', axis = 1, inplace=True)
above1
```

4.5 Merging the quality of care proportion for each measurement into one dataframe

In[ ]:
```
# referenced from https://stackoverflow.com/questions/23668427/pandas-three-way-joining-multiple-dataframes-on-columns

data_frames = [above1, above2, above3, above4, above5, below6, below7]

df_merged = reduce(lambda left,right: pd.merge(left,right,on=['ownership'],
                                    how='outer'), data_frames)

df_merged
```

DSA8002 (2021-2022)

4.6 Calculating the average quality of care better than the national average for each hospital group

In[ ]:
df_merged['sum'] = df_merged['pat_exp'] + df_merged['soc'] + df_merged['eoc'] + df_merged['time'] + df_merged['eff_img'] + df_merged['mort'] + df_merged['read']

df_merged['quality_of_care'] = df_merged['sum'] / 7

df_merged

4.7 Displaying the final dataframe containing each hospital group and the associated quality of care better than national average proportion

# In[ ]:
df_final = df_merged[['ownership','quality_of_care']]

df_final.sort_values(by='quality_of_care', ascending = False)

4.8 Visualising the final result

# In[ ]:
df_final.plot.bar(x='ownership')

5. Conclusion

- hospitals owned by Voluntary non-profit - Church have the highest proportion of its hospitals with overall quality of care better than the national average
- further information will be detailed in the project report

readme.txt
DSA8002 (2021 - 2022) Assignment

Author:
 John Poole

Student ID:
 40206201

Project Files:
 -> DSA8002 (2021-2022) -dataset.csv
 -> Hospital.db
 -> create_database.py
 -> classes+testing.py
 -> data_analysis_sql.py
 -> data_analysis_pandas.ipynb

DSA8002 (2021-2022)

File Description:

DSA8002 (2021-2022)-dataset.csv
-> the original dataset containing information about hospitals in the USA

create_database.py
-> creates the database file and three tables (hospital_info, quality of care and reviews)
-> populates the tables based on data from the Hospital dataset

Hospital.db
-> the database file containing the tables hospital_info, quality of care and reviews

classes+testing.py
-> implements three classes based on the database tables
-> classes allow users to add, display, remove and update information from each of the tables
-> unit testing code from the three classes to ensure it functions as expected

data_analysis_sql.py
-> a general analysis of the data using SQL
-> gathering information from the data using the overall hospital rating column

data_analysis_pandas.ipynb
-> a more detailed analysis of the data using pandas
-> used to determine which hospital ownership group has the highest proportion of its hospitals with overall quality of care better than the national average

How to use code:

- first, the user should run the create_database.py file to create the database file Hospital.db
- the Hospital.db file can be inspected using DB Browser for SQLite
- now the user can test the various functions in the classes.py depending on the specific table in the database file they wish to interact with. class methods will allow the user to add, update, remove and display information from the hospital_info, quality_of_care and reviews tables
- after the user is familiar with the classes.py class they can run the unit tests to understand how the classes function as expected
- the data analysis files can be viewed to gain an understanding into the processes and methods used to obtain conclusions to questions posed in the report document