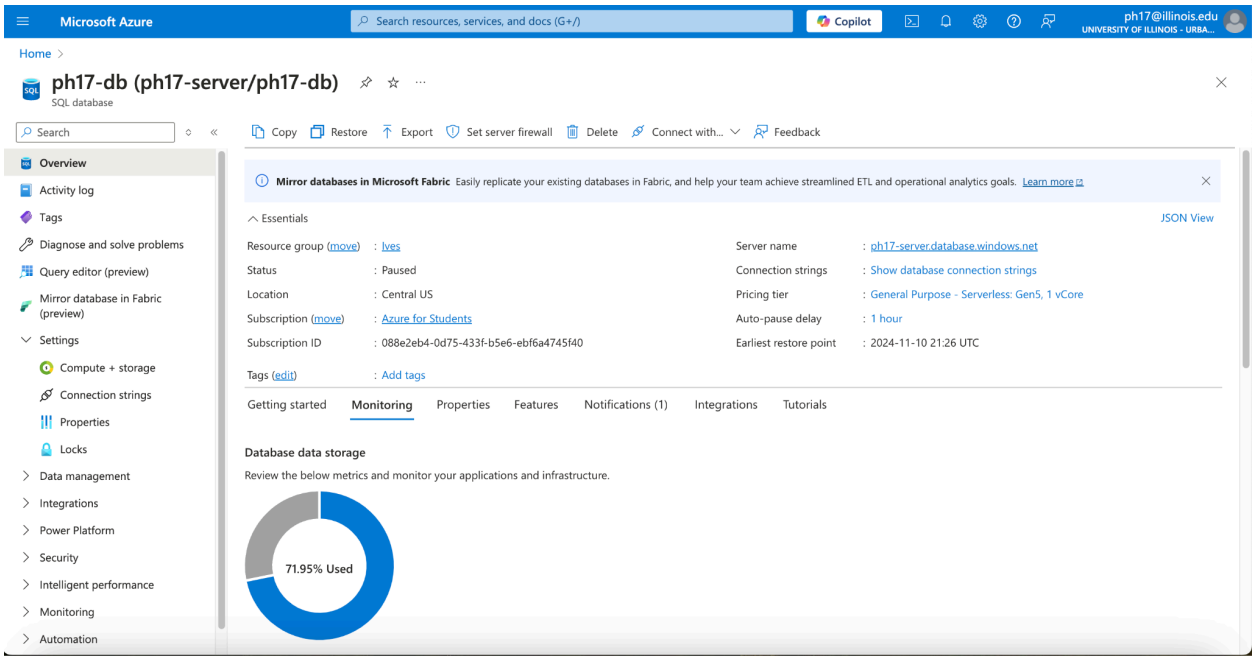
Mini Project 1 Documentation Group 13

This project involves converting semi-structured Yelp data in JSON format into a structured SQL format on Azure SQL Server. The document should include screenshots, code, and explanations of each step, allowing evaluators to assess technical understanding.You can copy this document and insert screenshots/explanations here and then make it a pdf to submit.

1.Setting up the Azure Server

- Screenshots Needed:
  - Connection String: A screenshot of the final connection string, which should show the database server name, username, and relevant connection parameters.

Example:



2. Initial Data Inspection and Loading

- Screenshots Needed: (you can add this directly from your python notebook but make sure that your output is clearly visible)
  - The original JSON data viewed in Pandas or SQL Workbench.
  - Code showing data loading into Pandas or SQL Workbench.
  - The JSON structure (highlighting nested elements or multi-valued attributes) before any transformations.

Example:

Review Data Sample:

	review_id	user_id	business_id	\
0	KU_O5udG6zpxOg-VcAEodg	mh_-eMZ6K5RLWhZyISBhwA	XQfwVwDr-v0ZS3_CbbE5Xw	
1	BiTunyQ73aT9WBnpR9DZGw	OyoGAe7OKpv6SyGZT5g77Q	7ATYjTIgM3jUlt4UM3IypQ	
2	saUsX_uimxRlCVr67Z4Jig	8g_iMtfSiwikVnbP2etR0A	YjUWPpI6HXG530lwP-fb2A	
3	AqPFMleE6RsU23_auESxiA	_7bHUi9Uuf5__HHc_Q8guQ	kxX2SOes4o-D3ZQBkiMRfA	
4	Sx8TMOWLNuJBWer-0pcmoA	bcjbaE6dDog4jkNY91ncLQ	e4Vwtrqf-wpJfwesgvdgxQ	

	stars	useful	funny	cool	\
0	3	0	0	0	
1	5	1	0	1	
2	3	0	0	0	
3	5	1	0	1	
4	4	1	0	1	

	text	date
0	If you decide to eat here, just be aware it is...	2018-07-07 22:09:11
1	I've taken a lot of spin classes over the year...	2012-01-03 15:28:18
2	Family diner. Had the buffet. Eclectic assortm...	2014-02-05 20:30:30
3	Wow! Yummy, different, delicious. Our favo...	2015-01-04 00:01:03
4	Cute interior and owner (?) gave us tour of up...	2017-01-14 20:54:15

Review Data Columns:

```
Index(['review_id', 'user_id', 'business_id', 'stars', 'useful', 'funny',
       'cool', 'text', 'date'],
      dtype='object')
```

Check-in Data Sample:

	business_id	date
0	---kPU91CF4Lq2-WlRu9Lw	2020-03-13 21:10:56, 2020-06-02 22:18:06, 2020...
1	--0iUa4sNDFiZFrAdIWhZQ	2010-09-13 21:43:09, 2011-05-04 23:08:15, 2011...
2	--30_8IhuyMHbSOcNWd6DQ	2013-06-14 23:29:17, 2014-08-13 23:20:22
3	--7PUidqRWpRSpXebiyxTg	2011-02-15 17:12:00, 2011-07-28 02:46:10, 2012...
4	--7jw19RH9JKXgFohspgQw	2014-04-21 20:42:11, 2014-04-28 21:04:46, 2014...

Check-in Data Columns:

```
Index(['business_id', 'date'], dtype='object')
```

Tip Data Sample:

	user_id	business_id	\
0	AGNUgVwnZUey3gcPCJ76iw	3uLgwr0qeCNMjKenHJwPGQ	
1	NBN4MgHP9D3cw--SnauTkA	QoezRbYQncpRqyrLH6Iqjg	
2	-copOvldyKh1qr-vzkDEvw	MYoRNLb5chwjQe3c_k37Gg	
3	FjMQVZjSqY8syIO-53KFKw	hV-bABTK-glh5wj31ps_Jw	
4	ld0AperBXk1h6UbqmM80zw	_uN0OudeJ3Zl_tf6nxg5ww	

	text	date	\
0	Avengers time with the ladies.	2012-05-18 02:17:21	
1	They have lots of good deserts and tasty cuban...	2013-02-05 18:35:10	
2	It's open even when you think it isn't	2013-08-18 00:56:08	
3	Very decent fried chicken	2017-06-27 23:05:38	
4	Appetizers.. platter special for lunch	2012-10-06 19:43:09	

	compliment_count
0	0
1	0
2	0
3	0
4	0

Tip Data Columns:

```
Index(['user_id', 'business_id', 'text', 'date', 'compliment_count'], dtype='object')
```

Business Data Sample:

	business_id	name	\
0	Pns2l4eNsfO8kk83dixA6A	Abby Rappoport, LAC, CMQ	
1	mpf3x-BjTdTEA3yCZrAYPw	The UPS Store	
2	tUFrWirKiKi_TAnsVWINQQ	Target	
3	MTSW4McQd7CbVtyjqoe9mw	St Honore Pastries	
4	mWMc6_wTdE0EUBKIGXDVfA	Perkiomen Valley Brewery	

	address	city	state	postal_code	\
0	1616 Chapala St, Ste 2	Santa Barbara	CA	93101	
1	87 Grasso Plaza Shopping Center	Affton	MO	63123	
2	5255 E Broadway Blvd	Tucson	AZ	85711	
3	935 Race St	Philadelphia	PA	19107	
4	101 Walnut St	Green Lane	PA	18054	

	latitude	longitude	stars	review_count	is_open	\
0	34.426679	-119.711197	5.0	7	0	
1	38.551126	-90.335695	3.0	15	1	
2	32.223236	-110.880452	3.5	22	0	
3	39.955505	-75.155564	4.0	80	1	
4	40.338183	-75.471659	4.5	13	1	

	attributes	\
0	{'ByAppointmentOnly': 'True'}	
1	{'BusinessAcceptsCreditCards': 'True'}	
2	{'BikeParking': 'True', 'BusinessAcceptsCredit...	
3	{'RestaurantsDelivery': 'False', 'OutdoorSeati...	
4	{'BusinessAcceptsCreditCards': 'True', 'Wheelc...	

	categories	\
0	Doctors, Traditional Chinese Medicine, Naturop...	
1	Shipping Centers, Local Services, Notaries, Ma...	
2	Department Stores, Shopping, Fashion, Home & G...	
3	Restaurants, Food, Bubble Tea, Coffee & Tea, B...	
4	Brewpubs, Breweries, Food	

	hours
0	None
1	{'Monday': '0:0-0:0', 'Tuesday': '8:0-18:30', ...
2	{'Monday': '8:0-22:0', 'Tuesday': '8:0-22:0', ...
3	{'Monday': '7:0-20:0', 'Tuesday': '7:0-20:0', ...
4	{'Wednesday': '14:0-22:0', 'Thursday': '16:0-2...

Business Data Columns:

```
Index(['business_id', 'name', 'address', 'city', 'state', 'postal_code',
       'latitude', 'longitude', 'stars', 'review_count', 'is_open',
       'attributes', 'categories', 'hours'],
      dtype='object')
```

```
User Data Sample:
                user_id    name  review_count       yelping_since  useful  \
0  qVc8ODYU5SZjKXVBgXdI7w  Walker           585  2007-01-25 16:47:26    7217
1  j14WgRoU_-2ZE1aw1dXrJg  Daniel          4333  2009-01-25 04:35:42   43091
2  ZWnXYQFK0hXEoTxPtV2zvg   Steph           665  2008-07-25 10:41:00    2086
3  SZDeASXq7o05mMNLshsdIA    Gwen           224  2005-11-29 04:38:33     512
4  hA5lMy-EnncsH4JoR-hFGQ   Karen            79  2007-01-05 19:40:59      29

   funny   cool                                        elite  \
0   1259   5994                                         2007
1  13066  27281  2009,2010,2011,2012,2013,2014,2015,2016,2017,2...
2   1010   1003                       2009,2010,2011,2012,2013
3    330    299                                2009,2010,2011
4     15      7

                                      friends  fans  ...  \
0  NSCy54eWehBJyZdG2iE84w, pe42u7DcCH2QmI81NX-8qA...   267  ...
1  ueRPE0CX75ePGMqOFVj6IQ, SZoH4DrRvzzl8wh5UXyU0A...  3138  ...
2  LuO3Bn4f3rlhyHIaNfTlnA, j9B4XdHUhDfTKVecyWQgyA...    52  ...
3  enx1vVPnfdNUdPho6PH_wg, 4wOcvMLtU6a9Lslggq74Vg...    28  ...
4  PBK4q9KEEBHhFvSXCUirIw, 3FWPpM7KU1gXeOM_ZbYMbA...     1  ...

   compliment_more  compliment_profile  compliment_cute  compliment_list  \
0               65                  55               56               18
1              264                 184              157              251
2               13                  10               17                3
3                4                   1                6                2
4                1                   0                0                0

   compliment_note  compliment_plain  compliment_cool  compliment_funny  \
0              232               844              467               467
1             1847              7054             3131              3131
2               66                96              119               119
3               12                16               26                26
4                1                 1                0                 0

   compliment_writer  compliment_photos
0                239                180
1               1521               1946
2                 35                 18
3                 10                  9
4                  0                  0

[5 rows x 22 columns]

User Data Columns:
Index(['user_id', 'name', 'review_count', 'yelping_since', 'useful', 'funny',
       'cool', 'elite', 'friends', 'fans', 'average_stars', 'compliment_hot',
       'compliment_more', 'compliment_profile', 'compliment_cute',
       'compliment_list', 'compliment_note', 'compliment_plain',
       'compliment_cool', 'compliment_funny', 'compliment_writer',
       'compliment_photos'],
      dtype='object')
```

```
Exploring the structure of the User DataFrame:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1987897 entries, 0 to 1987896
Data columns (total 22 columns):
 #   Column              Dtype
---  ------              -----
 0   user_id             object
 1   name                object
 2   review_count        int64
 3   yelping_since       object
 4   useful              int64
 5   funny               int64
 6   cool                int64
 7   elite               object
 8   friends             object
 9   fans                int64
 10  average_stars       float64
 11  compliment_hot      int64
 12  compliment_more     int64
 13  compliment_profile  int64
 14  compliment_cute     int64
 15  compliment_list     int64
 16  compliment_note     int64
 17  compliment_plain    int64
 18  compliment_cool     int64
 19  compliment_funny    int64
 20  compliment_writer   int64
 21  compliment_photos   int64
dtypes: float64(1), int64(16), object(5)
memory usage: 333.7+ MB
None
```

```
Exploring the structure of the Check-in DataFrame:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 131930 entries, 0 to 131929
Data columns (total 2 columns):
 #   Column       Non-Null Count   Dtype
---  ------       --------------   -----
 0   business_id  131930 non-null  object
 1   date         131930 non-null  object
dtypes: object(2)
memory usage: 2.0+ MB
None
```

```
Exploring the structure of the Tip DataFrame:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 908915 entries, 0 to 908914
Data columns (total 5 columns):
 #   Column            Non-Null Count   Dtype
---  ------            --------------   -----
 0   user_id           908915 non-null  object
 1   business_id       908915 non-null  object
 2   text              908915 non-null  object
 3   date              908915 non-null  datetime64[ns]
 4   compliment_count  908915 non-null  int64
dtypes: datetime64[ns](1), int64(1), object(3)
memory usage: 34.7+ MB
None
```

```
Exploring the structure of the Review DataFrame:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 6990280 entries, 0 to 6990279
Data columns (total 9 columns):
 #   Column       Dtype
---  ------       -----
 0   review_id    object
 1   user_id      object
 2   business_id  object
 3   stars        int64
 4   useful       int64
 5   funny        int64
 6   cool         int64
 7   text         object
 8   date         datetime64[ns]
dtypes: datetime64[ns](1), int64(4), object(4)
memory usage: 480.0+ MB
```

```
Exploring the structure of the Business DataFrame:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 150346 entries, 0 to 150345
Data columns (total 14 columns):
 #   Column        Non-Null Count   Dtype
---  ------        --------------   -----
 0   business_id   150346 non-null  object
 1   name          150346 non-null  object
 2   address       150346 non-null  object
 3   city          150346 non-null  object
 4   state         150346 non-null  object
 5   postal_code   150346 non-null  object
 6   latitude      150346 non-null  float64
 7   longitude     150346 non-null  float64
 8   stars         150346 non-null  float64
 9   review_count  150346 non-null  int64
 10  is_open       150346 non-null  int64
 11  attributes    136602 non-null  object
 12  categories    150243 non-null  object
 13  hours         127123 non-null  object
dtypes: float64(3), int64(2), object(9)
memory usage: 16.1+ MB
None
```

1. Loading and exploring the JSON data in the notebook

```
# Import necessary libraries
import pandas as pd

# Define file paths
review_path = '/kaggle/input/yelp-dataset/yelp_academic_dataset_review.json'
checkin_path = '/kaggle/input/yelp-dataset/yelp_academic_dataset_checkin.json'
business_path = '/kaggle/input/yelp-dataset/yelp_academic_dataset_business.json'
tip_path = '/kaggle/input/yelp-dataset/yelp_academic_dataset_tip.json'
user_path = '/kaggle/input/yelp-dataset/yelp_academic_dataset_user.json'

# Load the data with chunksize for large files to avoid memory issues
chunksize = 50000  # Adjust based on memory availability

# Load a sample chunk to inspect the structure of the review data
review_chunk = pd.read_json(review_path, lines=True, chunksize=chunksize)
review_df = next(review_chunk)
print("Review Data Sample:")
print(review_df.head())
print("\nReview Data Columns:")
print(review_df.columns)

# Load a sample chunk to inspect the structure of the check-in data
checkin_df = pd.read_json(checkin_path, lines=True)
print("\nCheck-in Data Sample:")
print(checkin_df.head())
print("\nCheck-in Data Columns:")
print(checkin_df.columns)

# Load a sample chunk to inspect the structure of the business data
business_chunk = pd.read_json(business_path, lines=True, chunksize=chunksize)
business_df = next(business_chunk)
print("\nBusiness Data Sample:")
print(business_df.head())
print("\nBusiness Data Columns:")
print(business_df.columns)

# Load the entire tip data (smaller file)
tip_df = pd.read_json(tip_path, lines=True)
print("\nTip Data Sample:")
print(tip_df.head())
print("\nTip Data Columns:")
print(tip_df.columns)

# Load the entire user data (smaller file)
user_chunk = pd.read_json(user_path, lines=True, chunksize=chunksize)
user_df = next(user_chunk)
print("\nUser Data Sample:")
print(user_df.head())
print("\nUser Data Columns:")
print(user_df.columns)

# Additional exploration
print("\nSummary of the Review DataFrame:")
print(review_df.info())

print("\nSummary of the Business DataFrame:")
print(business_df.info())
```

```
# Load complete Review data in chunks and concatenate into a single DataFrame
review_df = pd.concat(pd.read_json(review_path, lines=True, chunksize=chunksize))
print("\nReview DataFrame Loaded:")
print(review_df.head())
print("\nReview DataFrame Info:")
print(review_df.info())

# Load complete Business data in chunks and concatenate into a single DataFrame
business_df = pd.concat(pd.read_json(business_path, lines=True, chunksize=chunksize))
print("\nBusiness DataFrame Loaded:")
print(business_df.head())
print("\nBusiness DataFrame Info:")
print(business_df.info())

# Load Check-in data (smaller dataset, can be loaded directly)
checkin_df = pd.read_json(checkin_path, lines=True)
print("\nCheck-in DataFrame Loaded:")
print(checkin_df.head())
print("\nCheck-in DataFrame Info:")
print(checkin_df.info())

# Load Tip data (smaller dataset, can be loaded directly)
tip_df = pd.read_json(tip_path, lines=True)
print("\nTip DataFrame Loaded:")
print(tip_df.head())
print("\nTip DataFrame Info:")
print(tip_df.info())

# Load complete User data in chunks and concatenate into a single DataFrame
user_df = pd.concat(pd.read_json(user_path, lines=True, chunksize=chunksize))
print("\nUser DataFrame Loaded:")
print(user_df.head())
print("\nUser DataFrame Info:")
print(user_df.info())

# Check for missing values in each DataFrame
print("\nMissing Values in Review DataFrame:")
print(review_df.isnull().sum())

print("\nMissing Values in Business DataFrame:")
print(business_df.isnull().sum())

print("\nMissing Values in Check-in DataFrame:")
print(checkin_df.isnull().sum())

print("\nMissing Values in Tip DataFrame:")
print(tip_df.isnull().sum())

print("\nMissing Values in User DataFrame:")
print(user_df.isnull().sum())
```

- Key Insights
  - Business attributes and categories have hierarchical data that may need flattening for analysis.
  - Reviews and tips are rich in user-generated text, offering opportunities for sentiment analysis.
- Challenges
  - Managing large data sizes requires chunking and memory optimization.
  - Dealing with nested or multi-valued JSON attributes presents transformation complexities.
- Lessons Learned
  - Early identification of data structure saves time in later transformations.
  - Efficient loading strategies like chunking are critical for large-scale data.

3. Data Transformation and Flattening Process

- Screenshots Needed:
  - Code used to flatten or transform nested data structures.
  - The result after transformations, showing how nested attributes were normalized (e.g., pd.json_normalize() or using explode() for multi-valued lists).

Example:

```python
# Normalize the 'attributes' column
if 'attributes' in business_df.columns:
    # Drop NoneType values and normalize the dictionary
    attributes_df = pd.json_normalize(business_df['attributes'].dropna())
    # Concatenate the normalized attributes back to the main DataFrame
    business_df = pd.concat([business_df.drop(columns=['attributes']), attributes_df], axis=1)

# Normalize the 'hours' column
if 'hours' in business_df.columns:
    # Drop NoneType values and normalize the dictionary
    hours_df = pd.json_normalize(business_df['hours'].dropna())
    # Concatenate the normalized hours back to the main DataFrame
    business_df = pd.concat([business_df.drop(columns=['hours']), hours_df], axis=1)

# Display the processed data
print("\nFlattened Business DataFrame Sample:")
print(business_df.head())

# Check column names and data types
print("\nBusiness DataFrame columns after normalization:")
print(business_df.info())
```

```
Flattened Business DataFrame Sample:
              business_id                     name  \
0  Pns2l4eNsfO8kk83dixA6A  Abby Rappoport, LAC, CMQ
1  mpf3x-BjTdTEA3yCZrAYPw             The UPS Store
2  tUFrWirKiKi_TAnsVWINQQ                    Target
3  MTSW4McQd7CbVtyjqoe9mw       St Honore Pastries
4  mWMc6_wTdE0EUBKIGXDVfA  Perkiomen Valley Brewery

                      address           city state postal_code  \
0          1616 Chapala St, Ste 2  Santa Barbara    CA       93101
1  87 Grasso Plaza Shopping Center         Affton    MO       63123
2            5255 E Broadway Blvd         Tucson    AZ       85711
3                   935 Race St   Philadelphia    PA       19107
4                 101 Walnut St     Green Lane    PA       18054

    latitude   longitude  stars  review_count  ...  RestaurantsCounterService  \
0  34.426679 -119.711197    5.0             7  ...                        NaN
1  38.551126  -90.335695    3.0            15  ...                        NaN
2  32.223236 -110.880452    3.5            22  ...                        NaN
3  39.955505  -75.155564    4.0            80  ...                        NaN
4  40.338183  -75.471659    4.5            13  ...                        NaN

  AgesAllowed DietaryRestrictions    Monday    Tuesday  Wednesday   Thursday  \
0         NaN                 NaN   0:0-0:0  8:0-18:30  8:0-18:30  8:0-18:30
1         NaN                 NaN  8:0-22:0   8:0-22:0   8:0-22:0   8:0-22:0
2         NaN                 NaN  7:0-20:0   7:0-20:0   7:0-20:0   7:0-20:0
3         NaN                 NaN       NaN        NaN  14:0-22:0  16:0-22:0
4         NaN                 NaN   0:0-0:0   6:0-22:0   6:0-22:0   6:0-22:0

      Friday   Saturday     Sunday
0  8:0-18:30   8:0-14:0        NaN
1   8:0-23:0   8:0-23:0   8:0-22:0
2   7:0-21:0   7:0-21:0   7:0-21:0
3  12:0-22:0  12:0-22:0  12:0-18:0
4   9:0-0:0    9:0-22:0   8:0-22:0

[5 rows x 58 columns]
```

```
Business DataFrame columns after normalization:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 150346 entries, 0 to 150345
Data columns (total 58 columns):
 #   Column                      Non-Null Count   Dtype
---  ------                      --------------   -----
 0   business_id                 150346 non-null  object
 1   name                        150346 non-null  object
 2   address                     150346 non-null  object
 3   city                        150346 non-null  object
 4   state                       150346 non-null  object
 5   postal_code                 150346 non-null  object
 6   latitude                    150346 non-null  float64
 7   longitude                   150346 non-null  float64
 8   stars                       150346 non-null  float64
 9   review_count                150346 non-null  int64
 10  is_open                     150346 non-null  int64
 11  categories                  150243 non-null  object
 12  ByAppointmentOnly           42339 non-null   object
 13  BusinessAcceptsCreditCards  119765 non-null  object
 14  BikeParking                 72638 non-null   object
 15  RestaurantsPriceRange2      85314 non-null   object
 16  CoatCheck                   5584 non-null    object
 17  RestaurantsTakeOut          59857 non-null   object
 18  RestaurantsDelivery         56282 non-null   object
 19  Caters                      40127 non-null   object
 20  WiFi                        56914 non-null   object
 21  BusinessParking             91085 non-null   object
 22  WheelchairAccessible        28953 non-null   object
 23  HappyHour                   15171 non-null   object
 24  OutdoorSeating              48802 non-null   object
 25  HasTV                       45084 non-null   object
 26  RestaurantsReservations     45247 non-null   object
 27  DogsAllowed                 18284 non-null   object
 28  Alcohol                     43189 non-null   object
 29  GoodForKids                 53375 non-null   object
 30  RestaurantsAttire           39255 non-null   object
 31  Ambience                    44279 non-null   object
 32  RestaurantsTableService     19982 non-null   object
 33  RestaurantsGoodForGroups    44170 non-null   object
 34  DriveThru                   7760 non-null    object
 35  NoiseLevel                  37993 non-null   object
 36  GoodForMeal                 29087 non-null   object
 37  BusinessAcceptsBitcoin      17430 non-null   object
 38  Smoking                     4567 non-null    object
 39  Music                       7521 non-null    object
 40  GoodForDancing              4628 non-null    object
 41  AcceptsInsurance            5713 non-null    object
 42  BestNights                  5694 non-null    object
 43  BYOB                        4451 non-null    object
 44  Corkage                     3553 non-null    object
 45  BYOBCorkage                 1444 non-null    object
 46  HairSpecializesIn           1065 non-null    object
 47  Open24Hours                 39 non-null      object
 48  RestaurantsCounterService   19 non-null      object
 49  AgesAllowed                 129 non-null     object
 50  DietaryRestrictions         31 non-null      object
 51  Monday                      114474 non-null  object
 52  Tuesday                     120631 non-null  object
 53  Wednesday                   123771 non-null  object
 54  Thursday                    125198 non-null  object
 55  Friday                      124999 non-null  object
 56  Saturday                    110770 non-null  object
 57  Sunday                      81172 non-null   object
dtypes: float64(3), int64(2), object(53)
memory usage: 66.5+ MB
None
```

- Key Insights
  - Flattening nested data enables straightforward analysis and visualization.
  - The expanded dataset provides granular insights into business features and operational hours.
- Challenges

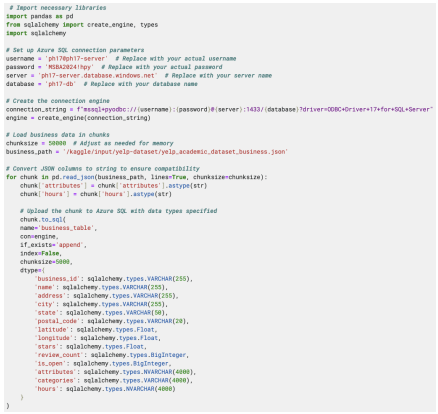
- Flattening complex nested dictionaries required careful handling to maintain consistency.
- Null values for certain attributes added complexity to merging.

- Lessons Learned
  - Flattening nested structures simplifies downstream data processing.

4. Creating and Uploading SQL Tables

- Screenshots Needed:
  - SQLAlchemy code for creating the connection engine and loading data into SQL tables.
  - Azure data studio view showing tables created from the JSON data in Azure.(I want to see the count for TOTAL ROWS and COLUMNS) in all the tables [if this is not shown properly your team might end up losing the grade)
  - Logs of any successful uploads or completed tasks.

Example:



```
# Import necessary libraries
import pandas as pd
from sqlalchemy import create_engine, types
import sqlalchemy

# Set up Azure SQL connection parameters
username = 'ph17@ph17-server'  # Replace with your actual username
password = 'MSBA2024!hpy'  # Replace with your actual password
server = 'ph17-server.database.windows.net'  # Replace with your server name
database = 'ph17-db'  # Replace with your database name

# Create the connection engine
connection_string = f"mssql+pyodbc://{username}:{password}@{server}:1433/{database}?driver=ODBC+Driver+17+for+SQL+Server"
engine = create_engine(connection_string)

# Load business data in chunks
chunksize = 50000  # Adjust as needed for memory
business_path = '/kaggle/input/yelp-dataset/yelp_academic_dataset_business.json'

# Convert JSON columns to string to ensure compatibility
for chunk in pd.read_json(business_path, lines=True, chunksize=chunksize):
    chunk['attributes'] = chunk['attributes'].astype(str)
    chunk['hours'] = chunk['hours'].astype(str)

    # Upload the chunk to Azure SQL with data types specified
    chunk.to_sql(
    name='business_table',
    con=engine,
    if_exists='append',
    index=False,
    chunksize=5000,
    dtype={
        'business_id': sqlalchemy.types.VARCHAR(255),
        'name': sqlalchemy.types.VARCHAR(255),
        'address': sqlalchemy.types.VARCHAR(255),
        'city': sqlalchemy.types.VARCHAR(255),
        'state': sqlalchemy.types.VARCHAR(50),
        'postal_code': sqlalchemy.types.VARCHAR(20),
        'latitude': sqlalchemy.types.Float,
        'longitude': sqlalchemy.types.Float,
        'stars': sqlalchemy.types.Float,
        'review_count': sqlalchemy.types.BigInteger,
        'is_open': sqlalchemy.types.BigInteger,
        'attributes': sqlalchemy.types.NVARCHAR(4000),
        'categories': sqlalchemy.types.VARCHAR(4000),
        'hours': sqlalchemy.types.NVARCHAR(4000)
    }
)
```

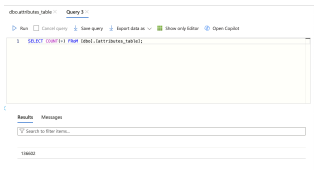
```
import pandas as pd
import sqlalchemy
from sqlalchemy import create_engine

# Database connection details
username = 'ph17@ph17-server'  # Your actual username
password = 'MSBA2024!hpy'  # Your actual password
server = 'ph17-server.database.windows.net'  # Your server name
database = 'ph17-db'  # Your database name

# Creating the connection string
connection_string = f"mssql+pyodbc://{username}:{password}@{server}/{database}?driver=ODBC+Driver+17+for+SQL+Server"

# Create the engine
engine = create_engine(connection_string)

# Load the check-in data from JSON file
checkin_path = '/kaggle/input/yelp-dataset/yelp_academic_dataset_checkin.json'
checkin_df = pd.read_json(checkin_path, lines=True)

# Split and explode the date column so each timestamp is in a separate row
checkin_df['date'] = checkin_df['date'].str.split(', ')
checkin_df = checkin_df.explode('date').reset_index(drop=True)  # Reset index to avoid duplicate labels

# Convert 'date' column to datetime format
checkin_df['date'] = pd.to_datetime(checkin_df['date'], errors='coerce')

# Upload the data to Azure SQL
checkin_df.to_sql(
    name='checkin_table',
    con=engine,
    if_exists='replace',  # 'replace' will overwrite the table, use 'append' to add data
    index=False,
    chunksize=50000,  # Upload data in larger chunks for efficiency
    dtype={
        'business_id': sqlalchemy.types.VARCHAR(255),
        'date': sqlalchemy.types.DATETIME  # Ensure the date column is stored as a DATETIME type
    }
)
```

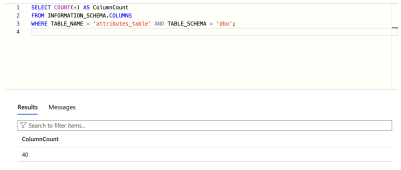
```
import pandas as pd
import sqlalchemy
from sqlalchemy import create_engine

# Database connection details
username = 'ph17@ph17-server'  # Your actual username
password = 'MSBA2024!hpy'  # Your actual password
server = 'ph17-server.database.windows.net'  # Your server name
database = 'ph17-db'  # Your database name

# Creating the connection string
connection_string = f"mssql+pyodbc://{username}:{password}@{server}/{database}?driver=ODBC+Driver+17+for+SQL+Server"

# Create the engine
engine = create_engine(connection_string)

# Load the tip data from JSON file
tip_path = '/kaggle/input/yelp-dataset/yelp_academic_dataset_tip.json'
tip_df = pd.read_json(tip_path, lines=True)

# Convert 'date' column to datetime format
tip_df['date'] = pd.to_datetime(tip_df['date'], errors='coerce')

# Upload the data to Azure SQL
tip_df.to_sql(
    name='tip_table',
    con=engine,
    if_exists='replace',  # 'replace' will overwrite the table, use 'append' to add data
    index=False,
    chunksize=50000,  # Upload data in larger chunks for efficiency
    dtype={
        'user_id': sqlalchemy.types.VARCHAR(255),
        'business_id': sqlalchemy.types.VARCHAR(255),
        'text': sqlalchemy.types.TEXT,
        'date': sqlalchemy.types.DATETIME,
        'compliment_count': sqlalchemy.types.INTEGER
    }
)
```

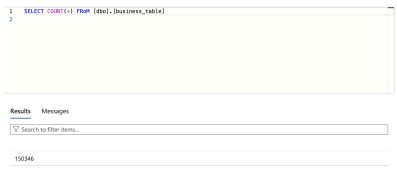
```python
import pandas as pd
import sqlalchemy
from sqlalchemy import create_engine

# Database connection details
username = 'ph170ph17-server'  # Your actual username
password = 'MSBA2024!hpy'  # Your actual password
server = 'ph17-server.database.windows.net'  # Your server name
database = 'ph17-db'  # Your database name

# Creating the connection string
connection_string = f"mssql+pyodbc://{username}:{password}@{server}/{database}?driver=ODBC+Driver+17+for+SQL+Server"

# Create the engine
engine = create_engine(connection_string)

# Load the user data from JSON file
user_path = '/kaggle/input/yelp-dataset/yelp_academic_dataset_user.json'
user_df = pd.read_json(user_path, lines=True)

# Convert any datetime columns if needed and handle large text data
user_df['yelping_since'] = pd.to_datetime(user_df['yelping_since'], errors='coerce')

# Upload the data to Azure SQL
user_df.to_sql(
    name='user_table',
    con=engine,
    if_exists='replace',  # 'replace' will overwrite the table, use 'append' to add data
    index=False,
    chunksize=10000,  # Upload data in larger chunks for efficiency
    dtype={
        'user_id': sqlalchemy.types.VARCHAR(255),
        'name': sqlalchemy.types.VARCHAR(255),
        'review_count': sqlalchemy.types.INTEGER,
        'yelping_since': sqlalchemy.types.DATETIME,
        'useful': sqlalchemy.types.INTEGER,
        'funny': sqlalchemy.types.INTEGER,
        'cool': sqlalchemy.types.INTEGER,
        'fans': sqlalchemy.types.INTEGER,
        'elite': sqlalchemy.types.TEXT,
        'average_stars': sqlalchemy.types.FLOAT,
        'compliment_hot': sqlalchemy.types.INTEGER,
        'compliment_more': sqlalchemy.types.INTEGER,
        'compliment_profile': sqlalchemy.types.INTEGER,
        'compliment_cute': sqlalchemy.types.INTEGER,
        'compliment_list': sqlalchemy.types.INTEGER,
        'compliment_note': sqlalchemy.types.INTEGER,
        'compliment_plain': sqlalchemy.types.INTEGER,
        'compliment_cool': sqlalchemy.types.INTEGER,
        'compliment_funny': sqlalchemy.types.INTEGER,
        'compliment_writer': sqlalchemy.types.INTEGER,
        'compliment_photos': sqlalchemy.types.INTEGER
    }
)
```

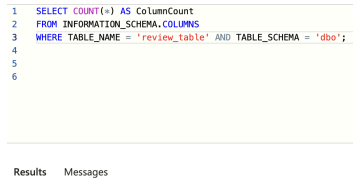
```python
import pandas as pd
import sqlalchemy
from sqlalchemy import create_engine

# Database connection details
username = 'ph170ph17-server'  # Your actual username
password = 'MSBA2024!hpy'  # Your actual password
server = 'ph17-server.database.windows.net'  # Your server name
database = 'ph17-db'  # Your database name

# Creating the connection string
connection_string = f"mssql+pyodbc://{username}:{password}@{server}/{database}?driver=ODBC+Driver+17+for+SQL+Server"

# Create the engine
engine = create_engine(connection_string)

# Define the path to the review data
review_path = '/kaggle/input/yelp-dataset/yelp_academic_dataset_review.json'

# Define the chunk size for reading the JSON file
chunk_size = 100000  # You can adjust this size based on available memory

# Read and upload data in chunks
reader = pd.read_json(review_path, lines=True, chunksize=chunk_size)

for chunk in reader:
    # Convert 'date' column to datetime format
    chunk['date'] = pd.to_datetime(chunk['date'], errors='coerce')

    # Upload the chunk to Azure SQL
    chunk.to_sql(
        name='review_table',
        con=engine,
        if_exists='append',  # Append data to avoid replacing the table each time
        index=False,
        dtype={
            'review_id': sqlalchemy.types.VARCHAR(255),
            'user_id': sqlalchemy.types.VARCHAR(255),
            'business_id': sqlalchemy.types.VARCHAR(255),
            'stars': sqlalchemy.types.INTEGER,
            'useful': sqlalchemy.types.INTEGER,
            'funny': sqlalchemy.types.INTEGER,
            'cool': sqlalchemy.types.INTEGER,
            'text': sqlalchemy.types.TEXT,
            'date': sqlalchemy.types.DATETIME
        }
    )
```

```python
import pandas as pd
import json
import sqlalchemy
from sqlalchemy import create_engine

# Database connection details
username = 'ph170ph17-server'  # Your actual username
password = 'MSBA2024!hpy'  # Your actual password
server = 'ph17-server.database.windows.net'  # Your server name
database = 'ph17-db'  # Your database name

# Creating the connection string
connection_string = f"mssql+pyodbc://{username}:{password}@{server}/{database}?driver=ODBC+Driver+17+for+SQL+Server"

# Create the engine
engine = create_engine(connection_string)

# Load the business data
business_path = '/kaggle/input/yelp-dataset/yelp_academic_dataset_business.json'
business_df = pd.read_json(business_path, lines=True)

# Safely parse the 'attributes' column if it's a string
business_df['attributes'] = business_df['attributes'].apply(lambda x: json.loads(x) if isinstance(x, str) else x)

# Normalize the 'attributes' column into separate columns
attributes_df = pd.json_normalize(business_df['attributes'].dropna())
attributes_df['business_id'] = business_df.loc[attributes_df.index, 'business_id']  # Add business_id for referencing

# Upload the attributes table to Azure
attributes_df.to_sql(
    name='attributes_table',
    con=engine,
    if_exists='replace',
    index=False,
    chunksize=100000,
    dtype={col: sqlalchemy.types.VARCHAR(255) for col in attributes_df.columns}
)

# Safely parse the 'hours' column if it's a string
business_df['hours'] = business_df['hours'].apply(lambda x: json.loads(x) if isinstance(x, str) else x)

# Normalize the 'hours' column into separate columns
hours_df = pd.json_normalize(business_df['hours'].dropna())
hours_df['business_id'] = business_df.loc[hours_df.index, 'business_id']  # Add business_id for referencing

# Upload the hours table to Azure
hours_df.to_sql(
    name='hours_table',
    con=engine,
    if_exists='replace',
    index=False,
    chunksize=5000,
    dtype={col: sqlalchemy.types.VARCHAR(255) for col in hours_df.columns}
)
```









```sql
SELECT COUNT(*) FROM [dbo].[tip_table]
```

Results | Messages

Search to filter items...

908915

```sql
SELECT COUNT(*) AS ColumnCount
FROM INFORMATION_SCHEMA.COLUMNS
WHERE TABLE_NAME = 'tip_table' AND TABLE_SCHEMA = 'dbo';
```

Results | Messages

Search to filter items...

ColumnCount
5

```sql
SELECT COUNT(*) FROM [dbo].[user_table]
```

Results | Messages

Search to filter items...

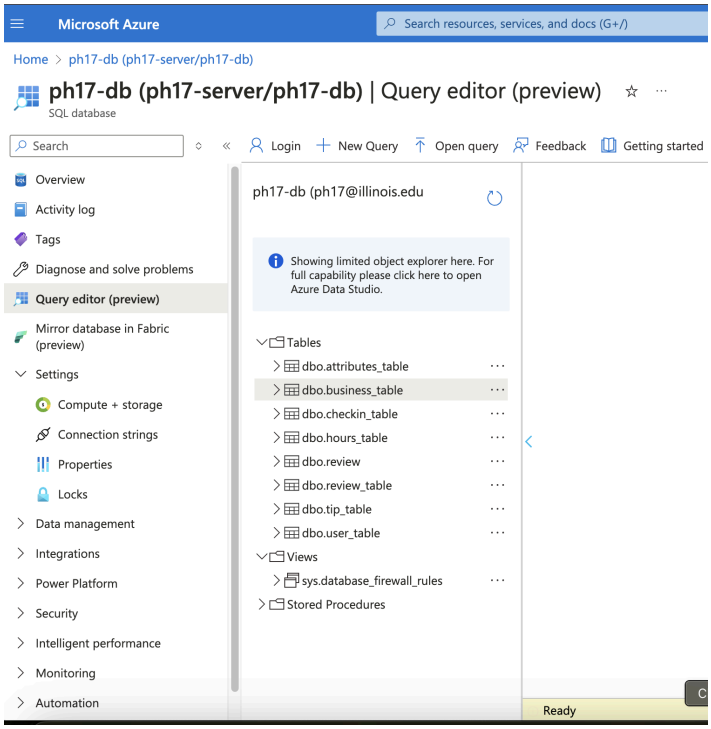
1987897

```sql
SELECT COUNT(*) AS ColumnCount
FROM INFORMATION_SCHEMA.COLUMNS
WHERE TABLE_NAME = 'user_table' AND TABLE_SCHEMA = 'dbo';
```

Results | Messages

Search to filter items...

ColumnCount
22



- Key Insights
- Successfully uploaded normalized JSON data to SQL tables for analysis.
- SQL table structures allow for efficient querying and scalability.
- Challenges

- Mapping complex data types (e.g., nested attributes) to SQL columns.
- Handling large data uploads while maintaining integrity.

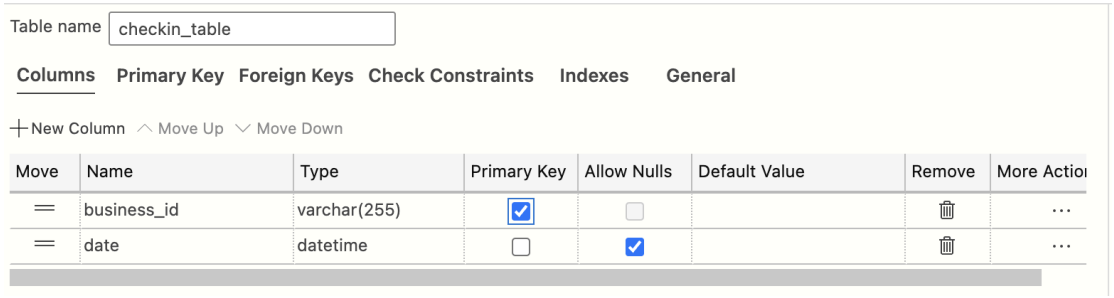
- Lessons Learned

- SQLAlchemy simplifies data integration but requires attention to data type mappings.
- Verifying data consistency post-upload ensures reliability for analysis.

5. Azure Data Studio – Primary and Foreign Key Definitions

- Screenshots Needed:
  - Azure Data Studio showing table structures with defined primary and foreign keys.
  - Code or interface view where keys are defined.
  - Execution logs or confirmation messages for successful PK/FK creation

Example:



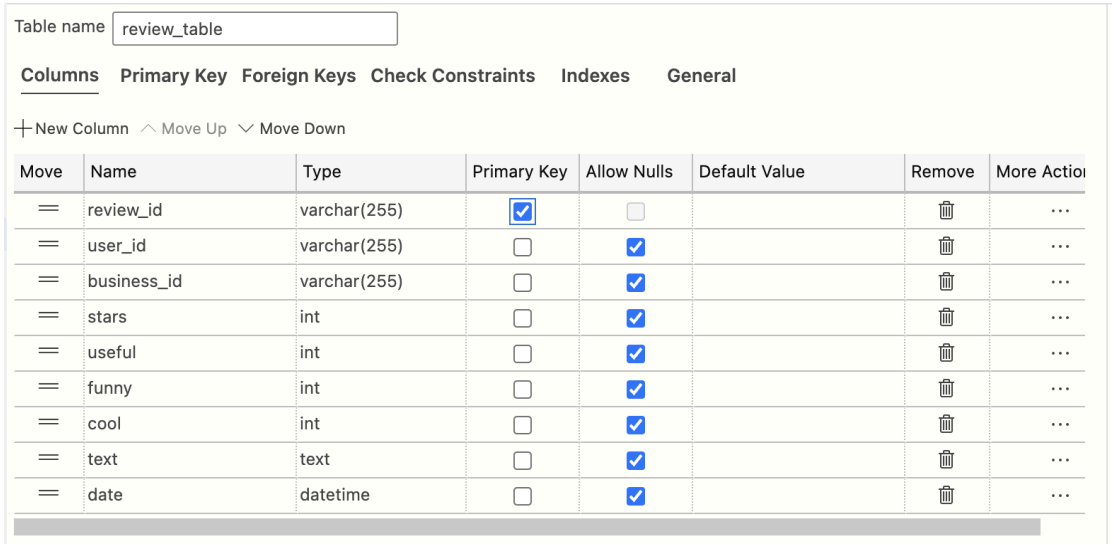


Table name: business_table

Columns | Primary Key | Foreign Keys | Check Constraints | Indexes | General

+ New Column | Move Up | Move Down

Move	Name	Type	Primary Key	Allow Nulls	Default Value	Remove	More Actio
=	business_id	varchar(255)	☑	☐		🗑	···
=	name	varchar(255)	☐	☑		🗑	···
=	address	varchar(255)	☐	☑		🗑	···
=	city	varchar(255)	☐	☑		🗑	···
=	state	varchar(50)	☐	☑		🗑	···
=	postal_code	varchar(20)	☐	☑		🗑	···
=	latitude	float	☐	☑		🗑	···
=	longitude	float	☐	☑		🗑	···
=	stars	float	☐	☑		🗑	···
=	review_count	bigint	☐	☑		🗑	···
=	is_open	bigint	☐	☑		🗑	···
=	attributes	nvarchar(4000)	☐	☑		🗑	···
=	categories	varchar(4000)	☐	☑		🗑	···
=	hours	nvarchar(4000)	☐	☑		🗑	···

Table name: user_table

Columns | Primary Key | Foreign Keys | Check Constraints | Indexes | General

+ New Column | Move Up | Move Down

Move	Name	Type	Primary Key	Allow Nulls	Default Value	Remove	More Actio
=	user_id	varchar(255)	☑	☐		🗑	···
=	name	varchar(255)	☐	☑		🗑	···
=	review_count	int	☐	☑		🗑	···
=	yelping_since	datetime	☐	☑		🗑	···
=	useful	int	☐	☑		🗑	···
=	funny	int	☐	☑		🗑	···
=	cool	int	☐	☑		🗑	···
=	elite	text	☐	☑		🗑	···
=	friends	varchar(MAX)	☐	☑		🗑	···
=	fans	int	☐	☑		🗑	···
=	average_stars	float	☐	☑		🗑	···
=	compliment_hot	int	☐	☑		🗑	···
=	compliment_more	int	☐	☑		🗑	···
=	compliment_profile	int	☐	☑		🗑	···
=	compliment_cute	int	☐	☑		🗑	···

Table name tip_table

Columns Primary Key Foreign Keys Check Constraints Indexes General

New Column Move Up Move Down

Move	Name	Type	Primary Key	Allow Nulls	Default Value	Remove	More Actio
=	user_id	varchar(255)	☐	☑		🗑	···
=	business_id	varchar(255)	☐	☑		🗑	···
=	text	text	☐	☑		🗑	···
=	date	datetime	☐	☑		🗑	···
=	compliment_count	int	☐	☑		🗑	···

Table name tip_table

Columns Primary Key Foreign Keys Check Constraints Indexes General

New Foreign Key

Name	Foreign Table	Remove
FK_tip_table_user_id_user_table	dbo.user_table	🗑
FK_tip_table_business_id_business_table	dbo.business_table	🗑

Table name review_table

Columns Primary Key Foreign Keys Check Constraints Indexes General

New Foreign Key

Name	Foreign Table	Remove
FK_review_table_user_id_user_table	dbo.user_table	🗑
FK_review_table_business_id_business_table	dbo.business_table	🗑

Table name checkin_table

Columns Primary Key Foreign Keys Check Constraints Indexes General

New Foreign Key

Name	Foreign Table	Remove
FK_checkin_table_business_id_business_table	dbo.business_table	

Table name attributes_table

Columns Primary Key Foreign Keys Check Constraints Indexes General

New Foreign Key

Name	Foreign Table	Remove
FK_attributes_table_business_id_business_table	dbo.business_table	

Table name hours_table

Columns Primary Key Foreign Keys Check Constraints Indexes General

New Foreign Key

Name	Foreign Table	Remove
FK_hours_table_business_id_business_table	dbo.business_table	

- Key Insights
- Primary and foreign key relationships enforce data integrity and support complex queries.
- Relational database structure efficiently connects user, business, and review data.
- Challenges
- Ensuring data consistency before applying keys to avoid constraint violations.
- Identifying appropriate foreign key references for each table.
- Lessons Learned
- Proper key definitions streamline relational database operations.
- Cross-table references enhance query flexibility and scalability.

6.Data Manipulation on Azure data studio

- Screenshots Needed:
  - Queries used for basic data manipulation .
  - Results of these queries, showing changes in the data where if relevant, try to avoid inserting and deleting data from the dataset as it might end up causing issues with your final output

Example:

```
# Query 1: Find the most common categories of restaurants
query1 = """SELECT TOP 10 categories, COUNT(*) AS num_restaurants
FROM business_table WHERE categories LIKE '%Restaurant%' GROUP BY categories
ORDER BY num_restaurants DESC;"""
df1 = pd.read_sql_query(query1, engine)
print("Most common restaurant categories:"); print(df1)
```

```
Most common restaurant categories:
                   categories  num_restaurants
0          Restaurants, Pizza              935
1          Pizza, Restaurants              823
2        Restaurants, Mexican              728
3        Restaurants, Chinese              708
4        Mexican, Restaurants              672
5        Chinese, Restaurants              651
6        Italian, Restaurants              328
7        Restaurants, Italian              285
8  Restaurants, American (New)             247
9  American (New), Restaurants             235
```

```
# Query 2: Identify the top-rated restaurants
query2 = """SELECT name, stars, review_count FROM business_table
WHERE categories LIKE '%Restaurant%' ORDER BY stars DESC, review_count DESC
OFFSET 0 ROWS FETCH NEXT 10 ROWS ONLY;"""
df2 = pd.read_sql_query(query2, engine)
print("Top-rated restaurants:"); print(df2)
```

```
Top-rated restaurants:
                                name  stars  review_count
0                       Blues City Deli    5.0           991
1                     Carlillos Cocina     5.0           799
2                             Tumerico     5.0           705
3                                 Yats     5.0           623
4              Smiling With Hope Pizza     5.0           526
5  Barracuda Deli Cafe St. Pete Beach     5.0           521
6                           Cafe Soleil    5.0           468
7                           Kaffe Crepe    5.0           454
8                            Buena Onda    5.0           414
9                          Big Al's Deli    5.0           390
```

```
# Query 3: Find the restaurants with the highest number of reviews
query3 = """SELECT name, review_count, stars FROM business_table
WHERE categories LIKE '%Restaurant%' ORDER BY review_count DESC
OFFSET 0 ROWS FETCH NEXT 10 ROWS ONLY;"""
df3 = pd.read_sql_query(query3, engine)
print("Restaurants with the highest number of reviews:"); print(df3)
```

```
Restaurants with the highest number of reviews:
                               name  review_count  stars
0                  Acme Oyster House          7568    4.0
1                       Oceana Grill          7400    4.0
2  Hattie B's Hot Chicken - Nashville         6093    4.5
3             Reading Terminal Market          5721    4.5
4          Ruby Slipper - New Orleans          5193    4.5
5                Mother's Restaurant           5185    3.5
6                       Royal House          5070    4.0
7                Commander's Palace           4876    4.5
8                              Luke           4554    4.0
9                            Cochon           4421    4.0
```

```
# Query 4: Get average star ratings for restaurants in specific cities
query4 = """SELECT city, AVG(stars) AS avg_rating FROM business_table
WHERE categories LIKE '%Restaurant%' GROUP BY city ORDER BY avg_rating DESC;"""
df4 = pd.read_sql_query(query4, engine)
print("Average star ratings for restaurants in specific cities:"); print(df4)
```

```
Average star ratings for restaurants in specific cities:
                city  avg_rating
0          Bryn Athyn         5.0
1            Glendale         5.0
2            Hilltown         5.0
3           Kalispell         5.0
4    LOWER PROVIDENCE         5.0
..                ...         ...
843        Chalemette         1.5
844        Black Jack         1.5
845         Bellville         1.5
846           Algiers         1.5
847     Peerless Park         1.0

[848 rows x 2 columns]
```

```
# Query 5: Identify peak check-in hours for popular restaurants
query5 = """SELECT business_id, FORMAT(date, 'HH') AS hour, COUNT(*) AS checkin_count
FROM checkin_table GROUP BY business_id, FORMAT(date, 'HH')
ORDER BY checkin_count DESC OFFSET 0 ROWS FETCH NEXT 10 ROWS ONLY;"""
df5 = pd.read_sql_query(query5, engine)
print("Peak check-in hours for popular restaurants:"); print(df5)
```

```
Peak check-in hours for popular restaurants:
              business_id hour  checkin_count
0  -QI8Qi8XWH3D8y8ethnajA   20           4186
1  -QI8Qi8XWH3D8y8ethnajA   21           4162
2  VQcCL9PiNL_wkGf-uF3fjg   00           3845
3  VQcCL9PiNL_wkGf-uF3fjg   01           3715
4  -QI8Qi8XWH3D8y8ethnajA   19           3497
5  VQcCL9PiNL_wkGf-uF3fjg   23           3315
6  -QI8Qi8XWH3D8y8ethnajA   18           3234
7  Eb1XmmLWyt_way5NNZ7-Pw   21           3170
8  -QI8Qi8XWH3D8y8ethnajA   22           3149
9  Eb1XmmLWyt_way5NNZ7-Pw   20           3138
```

```
# Query 6: Find the most active users who visit restaurants
query6 = """SELECT user_id, COUNT(*) AS review_count FROM review_table
GROUP BY user_id ORDER BY review_count DESC OFFSET 0 ROWS FETCH NEXT 10 ROWS ONLY;"""
df6 = pd.read_sql_query(query6, engine)
print("Most active users who visit restaurants:"); print(df6)
```

```
Most active users who visit restaurants:
                 user_id  review_count
0  _BcWyKQL16ndpBdggh2kNA          8016
1  Xw7ZjaGfr0WNVt6s_5KZfA          4835
2  0Igx-a1wAstiBDerGxXk2A          4576
3  -G7Zkl1wIWBBmD0KRy_sCw          4372
4  ET8n-r7glWYqZhuR6GcdNw          4178
5  bYENop4BuQepBjM1-BI3fA          4135
6  1HM81n6n4iPIFU5d2Lokhw          4037
7  fr1Hz2acAb3OaL3l6DyKNg          3757
8  wXdbkFZsfDR7utJvbWElyA          3651
9  Um5bfs5DH6eizgjH3xZsvg          3641
```

```
# Query 7: Identify common positive attributes of highly rated restaurants
query7 = """
SELECT 'BusinessAcceptsCreditCards' AS attribute_name, COUNT(*) AS count_positive
FROM attributes_table
JOIN business_table ON attributes_table.business_id = business_table.business_id
WHERE business_table.stars >= 4 AND attributes_table.BusinessAcceptsCreditCards = 'True'

UNION ALL

SELECT 'RestaurantsTakeOut' AS attribute_name, COUNT(*) AS count_positive
FROM attributes_table
JOIN business_table ON attributes_table.business_id = business_table.business_id
WHERE business_table.stars >= 4 AND attributes_table.RestaurantsTakeOut = 'True'

ORDER BY count_positive DESC;
"""
df7 = pd.read_sql_query(query7, engine)
print("Common positive attributes of highly rated restaurants:")
print(df7)
```

```
Common positive attributes of highly rated restaurants:
              attribute_name  count_positive
0  BusinessAcceptsCreditCards           56674
1          RestaurantsTakeOut           26295
```

```
# Query 9: List restaurants that are currently open
query9 = """SELECT name, city, stars, review_count FROM business_table
WHERE is_open = 1 AND categories LIKE '%Restaurant%'
ORDER BY stars DESC, review_count DESC OFFSET 0 ROWS FETCH NEXT 10 ROWS ONLY;"""
df9 = pd.read_sql_query(query9, engine)
print("Restaurants that are currently open:"); print(df9)
```

```
Restaurants that are currently open:
                                  name            city  stars  review_count
0                    Blues City Deli      Saint Louis    5.0           991
1                    Carlillos Cocina          Sparks    5.0           799
2                           Tumerico          Tucson    5.0           705
3                               Yats        Franklin    5.0           623
4            Smiling With Hope Pizza            Reno    5.0           526
5  Barracuda Deli Cafe St. Pete Beach   St Pete Beach    5.0           521
6                        Cafe Soleil  St. Pete Beach    5.0           468
7                        Kaffe Crepe            Reno    5.0           454
8                         Buena Onda   Santa Barbara    5.0           414
9                       Big Al's Deli       Nashville    5.0           390
```

```
# Query 10: Get the number of restaurants in each state
query10 = """SELECT state, COUNT(*) AS num_restaurants FROM business_table
WHERE categories LIKE '%Restaurant%' GROUP BY state ORDER BY num_restaurants DESC;"""
df10 = pd.read_sql_query(query10, engine)
print("Number of restaurants in each state:"); print(df10)
```

```
Number of restaurants in each state:
   state  num_restaurants
0     PA            12644
1     FL             8732
2     TN             4353
3     MO             4248
4     IN             4150
5     LA             3641
6     NJ             3343
7     AZ             2675
8     AB             2410
9     NV             1675
10    ID             1304
11    CA             1161
12    IL              983
13    DE              962
14    CO                1
15    HI                1
16   XMS                1
17    MT                1
18    NC                1
```

```
# Query 8: Find restaurants with extended opening hours
query8 = """
SELECT business_id,
       day_of_week,
       opening_time,
       closing_time,
       DATEDIFF(HOUR, opening_time, closing_time) AS hours_open
FROM (
    SELECT business_id,
           'Monday' AS day_of_week,
           CONVERT(TIME, SUBSTRING(Monday, 1, CHARINDEX('-', Monday) - 1)) AS opening_time,
           CONVERT(TIME, SUBSTRING(Monday, CHARINDEX('-', Monday) + 1, LEN(Monday))) AS closing_time
    FROM hours_table
    WHERE Monday IS NOT NULL
    UNION ALL
    SELECT business_id,
           'Tuesday',
           CONVERT(TIME, SUBSTRING(Tuesday, 1, CHARINDEX('-', Tuesday) - 1)),
           CONVERT(TIME, SUBSTRING(Tuesday, CHARINDEX('-', Tuesday) + 1, LEN(Tuesday)))
    FROM hours_table
    WHERE Tuesday IS NOT NULL
    UNION ALL
    SELECT business_id,
           'Wednesday',
           CONVERT(TIME, SUBSTRING(Wednesday, 1, CHARINDEX('-', Wednesday) - 1)),
           CONVERT(TIME, SUBSTRING(Wednesday, CHARINDEX('-', Wednesday) + 1, LEN(Wednesday)))
    FROM hours_table
    WHERE Wednesday IS NOT NULL
    UNION ALL
    SELECT business_id,
           'Thursday',
           CONVERT(TIME, SUBSTRING(Thursday, 1, CHARINDEX('-', Thursday) - 1)),
           CONVERT(TIME, SUBSTRING(Thursday, CHARINDEX('-', Thursday) + 1, LEN(Thursday)))
    FROM hours_table
    WHERE Thursday IS NOT NULL
    UNION ALL
    SELECT business_id,
           'Friday',
           CONVERT(TIME, SUBSTRING(Friday, 1, CHARINDEX('-', Friday) - 1)),
           CONVERT(TIME, SUBSTRING(Friday, CHARINDEX('-', Friday) + 1, LEN(Friday)))
    FROM hours_table
    WHERE Friday IS NOT NULL
    UNION ALL
    SELECT business_id,
           'Saturday',
           CONVERT(TIME, SUBSTRING(Saturday, 1, CHARINDEX('-', Saturday) - 1)),
           CONVERT(TIME, SUBSTRING(Saturday, CHARINDEX('-', Saturday) + 1, LEN(Saturday)))
    FROM hours_table
    WHERE Saturday IS NOT NULL
    UNION ALL
    SELECT business_id,
           'Sunday',
           CONVERT(TIME, SUBSTRING(Sunday, 1, CHARINDEX('-', Sunday) - 1)),
           CONVERT(TIME, SUBSTRING(Sunday, CHARINDEX('-', Sunday) + 1, LEN(Sunday)))
    FROM hours_table
    WHERE Sunday IS NOT NULL
) AS parsed_hours
WHERE DATEDIFF(HOUR, opening_time, closing_time) >= 12;
"""

# Run the query and load the results into a DataFrame
df8 = pd.read_sql_query(query8, engine)
print("Restaurants with extended opening hours:")
print(df8)
```

```
Restaurants with extended opening hours:
                business_id day_of_week opening_time closing_time  \
0       mpf3x-BjTdTEA3yCZrAYPw      Monday     08:00:00     22:00:00
1       tUFrWirKiKi_TAnsVWINQQ      Monday     07:00:00     20:00:00
2       k0hlBqXX-Bt0vf1op7Jr1w      Monday     09:30:00     21:30:00
3       UJsufbvfyfONHeWdvAHKjA      Monday     06:00:00     22:00:00
4       gbN4RuUi1BxQ2GBFQ2p3Cw      Monday     08:00:00     23:00:00
...                        ...         ...          ...          ...
146082  ckE45_JrmPgJ-DKYbV6IwA      Sunday     11:00:00     23:00:00
146083  SBpuja7Z_xcmCiWtUHPaKQ      Sunday     08:00:00     21:00:00
146084  x2sxfGcFKXoAUEXbZTaTow      Sunday     06:00:00     21:00:00
146085  9SCZdPbyJHD1fF1k957DiQ      Sunday     07:00:00     20:00:00
146086  jxLA8ikCHUJkvI8qWU89aw      Sunday     00:00:00     16:00:00

        hours_open
0               14
1               13
2               12
3               16
4               15
...            ...
146082          12
146083          13
146084          15
146085          13
146086          16

[146087 rows x 5 columns]
```

1. Decision Our Team Made for Each Query Purpose and Insights

Below are the decisions our team made based on the insights and decision-making value from each query:

Query 1: Most Common Restaurant Categories

- *Decision: Focus on unique cuisines or restaurant styles not heavily represented in the top categories (e.g., fusion cuisines or vegan restaurants) to stand out from market saturation.*

Query 2: Top-Rated Restaurants

- *Decision: Benchmark successful restaurants by studying their operational practices, menu offerings, and customer service to replicate their high ratings.*

Query 3: Restaurants with the Highest Number of Reviews

- *Decision: Leverage marketing strategies like loyalty programs and influencer partnerships to increase customer reviews and brand visibility.*

Query 4: Average Star Ratings for Restaurants in Specific Cities

- *Decision: Target cities with higher average star ratings for launching new restaurants, as this indicates a customer base willing to support quality establishments.*

Query 5: Peak Check-In Hours for Popular Restaurants

- *Decision: Align operational hours and staff shifts with peak hours to maximize revenue and customer satisfaction during high-demand times.*

Query 6: Most Active Users Who Visit Restaurants

- *Decision: Create targeted promotions and engagement campaigns with top reviewers, offering perks for their continued patronage and leveraging their influence.*

Query 7: Common Positive Attributes of Highly Rated Restaurants

- *Decision: Ensure the inclusion of essential attributes like accepting credit cards and offering takeout to align with customer expectations.*

Query 8: Restaurants with Extended Opening Hours

- *Decision: Offer extended hours on weekends or holidays to fill unmet demand for late-night or early-morning dining options.*

Query 9: Restaurants That Are Currently Open

- *Decision: Highlight "currently open" statuses in marketing materials or apps to attract customers during real-time searches, differentiating from competitors.*

Query 10: Number of Restaurants in Each State

- *Decision: Explore underrepresented states or regions for expansion opportunities, focusing on areas with fewer restaurants to reduce competition.*

2. Overall Conclusion for the Restaurant Strategy

***Overall Conclusion**:*
Our analysis highlights key areas for differentiation and strategic growth in the restaurant industry. To maximize success, we recommend:

1. *Targeting underserved cuisines or niche markets identified in Query 1.*
2. *Replicating the success factors of top-rated restaurants (Query 2) and highly reviewed establishments (Query 3).*
3. *Launching in cities or regions with high customer satisfaction (Query 4) and low restaurant density (Query 10).*
4. *Optimizing operational hours (Query 5) and incorporating customer-valued features (Query 7).*
5. *Leveraging customer engagement through influencers and real-time marketing (Queries 6 and 9).*