ETL Project

ETL: Extract, Transform, Load

For this project, I wanted to see if the amount of sleep I got affected my blood glucose values.

Extract:

Two data sources:

1. Dexcom glucose values
2. Fitbit sleep values

Transform:

Transform data and data cleaning.

In [1]:

```
# Import dependencies

import pandas as pd
from sqlalchemy import create_engine
```

In [2]:

```
# Read the CSV file

csv_file_1 = "Resources/DEXCOM_CGM_08082019_to_11052019.csv"
cgm_data_df = pd.read_csv(csv_file_1)

# Display the data

cgm_data_df.head()
```

Out[2]:

	Index	Timestamp (YYYY-MM-DDThh:mm:ss)	Event Type	Event Subtype	Patient Info	Device Info	Source Device ID	Glucose Value (mg/dL)	Insulin Value (u)	Carb Value (grams)	(I
0	1	2019-08-08T00:01:13	EGV	NaN	NaN	NaN	iPhone G6	140	NaN	NaN	
1	2	2019-08-08T00:06:13	EGV	NaN	NaN	NaN	iPhone G6	138	NaN	NaN	
2	3	2019-08-08T00:11:13	EGV	NaN	NaN	NaN	iPhone G6	136	NaN	NaN	
3	4	2019-08-08T00:16:13	EGV	NaN	NaN	NaN	iPhone G6	134	NaN	NaN	
4	5	2019-08-08T00:21:13	EGV	NaN	NaN	NaN	iPhone G6	131	NaN	NaN	

In [3]:

```
# Get only the columns needed from the glucose values CSV

new_cgm_data_df = cgm_data_df[['Timestamp (YYYY-MM-DDThh:mm:ss)', 'Event Type'
, 'Glucose Value (mg/dL)']].copy()

# Display the data

new_cgm_data_df.head()
```

Out[3]:

	Timestamp (YYYY-MM-DDThh:mm:ss)	Event Type	Glucose Value (mg/dL)
0	2019-08-08T00:01:13	EGV	140
1	2019-08-08T00:06:13	EGV	138
2	2019-08-08T00:11:13	EGV	136
3	2019-08-08T00:16:13	EGV	134
4	2019-08-08T00:21:13	EGV	131

In [4]:

```
# Do the same for the sleep CSV

csv_file_2 = "Resources/fitbit_sleep_08082019_to_11052019.csv"
sleep_data_df = pd.read_csv(csv_file_2)

# Display the data

sleep_data_df.head()
```

Out[4]:

	Start Time	End Time	Minutes Asleep	Minutes Awake	Number of Awakenings	Time in Bed	Minutes REM Sleep	Minutes Light Sleep	Minutes Deep Sleep
0	2019-11-03 12:18AM	2019-11-03 8:13AM	429	46	30	475	131.0	210.0	88.0
1	2019-11-02 3:47PM	2019-11-02 7:30PM	192	31	19	223	29.0	135.0	28.0
2	2019-11-01 8:12PM	2019-11-02 7:21AM	574	95	40	669	120.0	379.0	75.0
3	2019-11-01 2:08AM	2019-11-01 7:00AM	247	45	16	292	50.0	161.0	36.0
4	2019-10-30 8:35PM	2019-10-31 6:36AM	508	93	2	601	NaN	NaN	NaN

In [5]:
```
# Get only the columns needed from the sleep CSV

new_sleep_data_df = sleep_data_df[['End Time', 'Minutes Asleep', 'Minutes Awake', 'Number of Awakenings','Time in Bed','Minutes REM Sleep','Minutes Light Sleep','Minutes Deep Sleep']].copy()

# Display the data

new_sleep_data_df.head()
```

Out[5]:

	End Time	Minutes Asleep	Minutes Awake	Number of Awakenings	Time in Bed	Minutes REM Sleep	Minutes Light Sleep	Minutes Deep Sleep
0	2019-11-03 8:13AM	429	46	30	475	131.0	210.0	88.0
1	2019-11-02 7:30PM	192	31	19	223	29.0	135.0	28.0
2	2019-11-02 7:21AM	574	95	40	669	120.0	379.0	75.0
3	2019-11-01 7:00AM	247	45	16	292	50.0	161.0	36.0
4	2019-10-31 6:36AM	508	93	2	601	NaN	NaN	NaN

In [6]:
```
# Convert the time stamp to display only the date
# Because I will be combining the data to do analysis later, the date will be the joining column
# The time stamp will be too specific so I need to make sure data displays date only

new_cgm_data_df['Timestamp (YYYY-MM-DDThh:mm:ss)'] = pd.to_datetime(new_cgm_data_df['Timestamp (YYYY-MM-DDThh:mm:ss)']).dt.strftime('%Y-%m-%d')

# Display the data

new_cgm_data_df.head()
```

Out[6]:

	Timestamp (YYYY-MM-DDThh:mm:ss)	Event Type	Glucose Value (mg/dL)
0	2019-08-08	EGV	140
1	2019-08-08	EGV	138
2	2019-08-08	EGV	136
3	2019-08-08	EGV	134
4	2019-08-08	EGV	131

In [7]:
```
# Check the amount of data

len(new_cgm_data_df)
```

Out[7]:
```
25772
```

In [8]:

```
# Drop all Event Type values that are not equal to EGV (estimated glucose valu
e)

new_cgm_data_df = new_cgm_data_df[~(new_cgm_data_df['Event Type'] != 'EGV')]

# Dipslay the data

new_cgm_data_df.head()
```

Out[8]:

	Timestamp (YYYY-MM-DDThh:mm:ss)	Event Type	Glucose Value (mg/dL)
0	2019-08-08	EGV	140
1	2019-08-08	EGV	138
2	2019-08-08	EGV	136
3	2019-08-08	EGV	134
4	2019-08-08	EGV	131

In [9]:

```
# Check count to see if rows were dropped

len(new_cgm_data_df)
```

Out[9]:

```
25449
```

In [10]:

```
# Rename columns for clean data

new_cgm_data_df = new_cgm_data_df.rename(columns={'Timestamp (YYYY-MM-DDThh:m
m:ss)':'date_entry', 'Event Type':'event_type', 'Glucose Value (mg/dL)':'gluco
se_value'})

# Display the data

new_cgm_data_df.head()
```

Out[10]:

	date_entry	event_type	glucose_value
0	2019-08-08	EGV	140
1	2019-08-08	EGV	138
2	2019-08-08	EGV	136
3	2019-08-08	EGV	134
4	2019-08-08	EGV	131

In [11]:

```
# Drop the event_type column because I will not need it for further analysis

new_cgm_data_df = new_cgm_data_df.drop(columns=['event_type'])

# Drop NaN values from the table

new_cgm_data_df = new_cgm_data_df.dropna()
```

In [13]:

```
# Check data types of the values - will need to convert later

new_cgm_data_df.dtypes
```

Out[13]:

```
date_entry       object
glucose_value    object
dtype: object
```

In [14]:

```
# Get rid of all non-values such as "High" and "Low"

new_cgm_data_df = new_cgm_data_df[(new_cgm_data_df['glucose_value'] != 'High'
)]
new_cgm_data_df = new_cgm_data_df[(new_cgm_data_df['glucose_value'] != 'Low')]

# Display the data

new_cgm_data_df.head()
```

Out[14]:

	date_entry	glucose_value
0	2019-08-08	140
1	2019-08-08	138
2	2019-08-08	136
3	2019-08-08	134
4	2019-08-08	131

In [16]:

```
# Convert the glucose values to float for calculations later

new_cgm_data_df['glucose_value'] = new_cgm_data_df['glucose_value'].astype('fl
oat')

# Check to make sure everything looks ok

new_cgm_data_df.head()
```

Out[16]:

	date_entry	glucose_value
0	2019-08-08	140.0
1	2019-08-08	138.0
2	2019-08-08	136.0
3	2019-08-08	134.0
4	2019-08-08	131.0

In [18]:

```
# In orer to do aggregate calculations, set the index to the date

new_cgm_data_df = new_cgm_data_df.set_index('date_entry')

# Find the average glucose value by date

clean_cgm_df = new_cgm_data_df.groupby(new_cgm_data_df.index).mean()

# Display the data

clean_cgm_df.head()
```

Out[18]:

	glucose_value
date_entry	
2019-08-08	114.712803
2019-08-09	151.347222
2019-08-10	203.204861
2019-08-11	176.045139
2019-08-12	194.965278

In [19]:

```
# Reset the index so that the data can be loaded to pgAdmin4

clean_cgm_df = clean_cgm_df.reset_index()

# Display the data

clean_cgm_df.head()
```

Out[19]:

	date_entry	glucose_value
0	2019-08-08	114.712803
1	2019-08-09	151.347222
2	2019-08-10	203.204861
3	2019-08-11	176.045139
4	2019-08-12	194.965278

In [21]:

```
# Convert the sleep date time stamp to date only so that the date is in the sa
me format as the glucose values table
# In order to join the two data frames, the date needs to be in the same forma
t

new_sleep_data_df['End Time'] = pd.to_datetime(new_sleep_data_df['End Time']).
dt.strftime('%Y-%m-%d')

# Display the data

new_sleep_data_df.head()
```

Out[21]:

	End Time	Minutes Asleep	Minutes Awake	Number of Awakenings	Time in Bed	Minutes REM Sleep	Minutes Light Sleep	Minutes Deep Sleep
0	2019-11-03	429	46	30	475	131.0	210.0	88.0
1	2019-11-02	192	31	19	223	29.0	135.0	28.0
2	2019-11-02	574	95	40	669	120.0	379.0	75.0
3	2019-11-01	247	45	16	292	50.0	161.0	36.0
4	2019-10-31	508	93	2	601	NaN	NaN	NaN

In [22]:

```
# Rename the columns for clean data

new_sleep_data_df = new_sleep_data_df.rename(columns={'End Time':'date_entry',
'Minutes Asleep':'min_asleep','Minutes Awake':'min_awake','Number of Awakening
s':'num_times_awake','Time in Bed':'total_time_in_bed','Minutes REM Sleep':'mi
n_REM','Minutes Light Sleep':'min_light','Minutes Deep Sleep':'min_deep'})

# Display the data

new_sleep_data_df.head()
```

Out[22]:

	date_entry	min_asleep	min_awake	num_times_awake	total_time_in_bed	min_REM	min_ligh
0	2019-11-03	429	46	30	475	131.0	210.(
1	2019-11-02	192	31	19	223	29.0	135.(
2	2019-11-02	574	95	40	669	120.0	379.(
3	2019-11-01	247	45	16	292	50.0	161.(
4	2019-10-31	508	93	2	601	NaN	Na

In [24]:

```
# Drop NaN values

new_sleep_data_df = new_sleep_data_df.dropna()

# Display the data

new_sleep_data_df.head()
```

Out[24]:

	date_entry	min_asleep	min_awake	num_times_awake	total_time_in_bed	min_REM	min_ligh
0	2019-11-03	429	46	30	475	131.0	210.(
1	2019-11-02	192	31	19	223	29.0	135.(
2	2019-11-02	574	95	40	669	120.0	379.(
3	2019-11-01	247	45	16	292	50.0	161.(
5	2019-10-30	221	31	16	252	39.0	138.(

In [25]:

```
# Check the count of data

len(new_sleep_data_df)
```

Out[25]:

```
73
```

Load:

Create database and tables in pgAdmin4 and load the data above.

In [26]:

```
# Create the connection to pgAdmin4

rds_connection_string = "postgres:*******@localhost:5432/T1D_db"
engine = create_engine(f'postgresql://{rds_connection_string}')
```

In [27]:

```
# Check table names to make sure correct tables are there

engine.table_names()
```

Out[27]:

```
['cgm_data', 'sleep_data']
```

In [28]:

```
# Load the data into the appropriate tables

clean_cgm_df.to_sql(name='cgm_data', con=engine, if_exists='replace', index=False)
new_sleep_data_df.to_sql(name='sleep_data', con=engine, if_exists='replace', index=False)
```

In [29]:
```
# Read the data back to make sure data was loaded correctly

pd.read_sql_query('select * from cgm_data', con=engine).head()
```

Out[29]:

	date_entry	glucose_value
0	2019-08-08	114.712803
1	2019-08-09	151.347222
2	2019-08-10	203.204861
3	2019-08-11	176.045139
4	2019-08-12	194.965278

In [30]:
```
# Read the data back to make sure data was loaded correctly

pd.read_sql_query('select * from sleep_data', con=engine).head()
```

Out[30]:

	date_entry	min_asleep	min_awake	num_times_awake	total_time_in_bed	min_REM	min_ligh
0	2019-11-03	429	46	30	475	131.0	210.(
1	2019-11-02	192	31	19	223	29.0	135.(
2	2019-11-02	574	95	40	669	120.0	379.(
3	2019-11-01	247	45	16	292	50.0	161.(
4	2019-10-30	221	31	16	252	39.0	138.(

Analysis:

Analysis on the clean data.

In [31]:

```
# Join the two data frames for analysis
# Join on the date column

sleep_cgm_df = pd.merge(new_sleep_data_df, clean_cgm_df, on='date_entry', how=
'inner')

# Display data

sleep_cgm_df.head()
```

Out[31]:

	date_entry	min_asleep	min_awake	num_times_awake	total_time_in_bed	min_REM	min_ligh
0	2019-11-03	429	46	30	475	131.0	210.(
1	2019-11-02	192	31	19	223	29.0	135.(
2	2019-11-02	574	95	40	669	120.0	379.(
3	2019-11-01	247	45	16	292	50.0	161.(
4	2019-10-30	221	31	16	252	39.0	138.(

In [38]:

```
# Create bins for the amount of sleep in minutes
# Each bin will have 120 minutes or 2 hours

bins = [0, 120, 240, 360, 480, 10000]

# Create labels for the bins
# Labels show the amount in hours

labels = ['<=2','2<=4', '4<=6', '6<=8', '>8']

# Create a new column that shows where each value falls for the amount of slee
p

sleep_cgm_df['hours'] = pd.cut(sleep_cgm_df['min_asleep'], bins=bins, labels=l
abels)

# Display the data

sleep_cgm_df.head()
```

Out[38]:

	date_entry	min_asleep	min_awake	num_times_awake	total_time_in_bed	min_REM	min_ligh
0	2019-11-03	429	46	30	475	131.0	210.(
1	2019-11-02	192	31	19	223	29.0	135.(
2	2019-11-02	574	95	40	669	120.0	379.(
3	2019-11-01	247	45	16	292	50.0	161.(
4	2019-10-30	221	31	16	252	39.0	138.(

In [42]:

```
# Since data is in bins, find the average glucose value by amount of sleep

cgm_by_hours_sleep = pd.DataFrame(sleep_cgm_df.groupby(sleep_cgm_df['hours']).
mean()['glucose_value'])

# Display the data

cgm_by_hours_sleep.head()
```

Out[42]:

	glucose_value
hours	
<=2	NaN
2<=4	185.907280
4<=6	178.811851
6<=8	166.626095
>8	182.428819

In [43]:

```
# Find the max minutes of REM sleep in order to create appropriate bins

sleep_cgm_df['min_REM'].max()
```

Out[43]:

```
174.0
```

In [44]:

```
# Create bins and labels for the REM sleep data

REM_bins = [0, 60, 120, 180]
REM_labels = ['<=1', '1<=2', '>2']

# Label the data into the appropriate bin

sleep_cgm_df['REM_hours'] = pd.cut(sleep_cgm_df['min_REM'], bins=REM_bins, labels=REM_labels)

# Display the data

sleep_cgm_df.head()
```

Out[44]:

	date_entry	min_asleep	min_awake	num_times_awake	total_time_in_bed	min_REM	min_ligh
0	2019-11-03	429	46	30	475	131.0	210.(
1	2019-11-02	192	31	19	223	29.0	135.(
2	2019-11-02	574	95	40	669	120.0	379.(
3	2019-11-01	247	45	16	292	50.0	161.(
4	2019-10-30	221	31	16	252	39.0	138.(

In [45]:

```
# Find the average glucose value by REM sleep amount

cgm_by_hours_sleep_2 = pd.DataFrame(sleep_cgm_df.groupby(sleep_cgm_df['REM_hours']).mean()['glucose_value'])

# Display the data

cgm_by_hours_sleep_2
```

Out[45]:

	glucose_value
REM_hours	
<=1	179.996548
1<=2	173.700413
>2	175.453982

See PDF for further information