UPDATED RDB SCHEMA FINAL VERSION

Table: major	Type
agency_code	INT NOT NULL
agency_name	VARCHAR NOT NULL
current_uii	VARCHAR NOT NULL
investment_name	VARCHAR NOT NULL
project_id - **key**	VARCHAR
agency_project_id	VARCHAR NOT NULL
project_name - **key**	VARCHAR
project_goal	VARCHAR NOT NULL
infrastructure_management_categor	INT NOT NULL
project_status	VARCHAR
tmf_initiative	VARCHAR
software_project	BOOLEAN
incremental_development	BOOLEAN
iteration_frequency_amount	FLOAT
iteration_frequency_units	VARCHAR
iterative_description	VARCHAR
planned_start_date	DATE
projected_start_date	DATE
actual_start_date	DATE
planned_end_date	DATE
projected_end_date	DATE
actual_end_date	DATE
project_length	FLOAT
planned_cost	FLOAT
projected_cost	FLOAT
actual_cost	FLOAT
schedule_variance_days	INT
schedule_variance_percent	FLOAT
schedule_variance_color	VARCHAR
cost_variance_dollars_mill	FLOAT
cost_variance_percent	FLOAT
cost_variance_color	VARCHAR
updated_time	TIMESTAMP

Table: departments	Type
agency_name - **key**	VARCHAR PRIMARY KEY
employee_total	FLOAT
project_count	INT NOT NULL
major_prop	FLOAT NOT NULL
standard_prop	FLOAT NOT NULL
non_major_prop	FLOAT NOT NULL
mean_length	FLOAT NOT NULL
median_cost	FLOAT NOT NULL
median_cost_variance_percent	FLOAT NOT NULL

Table: standard	Type
agency_code	INT NOT NULL
agency_name	VARCHAR NOT NULL
current_uii	VARCHAR NOT NULL
investment_name	VARCHAR NOT NULL
project_id - **key**	VARCHAR
agency_project_id	VARCHAR NOT NULL
project_name - **key**	VARCHAR
project_goal	VARCHAR NOT NULL
infrastructure_management_categor	INT NOT NULL
project_status	VARCHAR
tmf_initiative	VARCHAR
software_project	BOOLEAN
incremental_development	BOOLEAN
iteration_frequency_amount	FLOAT
iteration_frequency_units	VARCHAR
iterative_description	VARCHAR
planned_start_date	DATE
projected_start_date	DATE
actual_start_date	DATE
planned_end_date	DATE
projected_end_date	DATE
actual_end_date	DATE
project_length	FLOAT
planned_cost	FLOAT
projected_cost	FLOAT
actual_cost	FLOAT
schedule_variance_days	INT
schedule_variance_percent	FLOAT
schedule_variance_color	VARCHAR
cost_variance_dollars_mill	FLOAT
cost_variance_percent	FLOAT
cost_variance_color	VARCHAR
updated_time	TIMESTAMP

Table: non_major	Type
agency_code	INT NOT NULL
agency_name	VARCHAR NOT NULL
current_uii	VARCHAR NOT NULL
investment_name	VARCHAR NOT NULL
project_id - **key**	VARCHAR
agency_project_id	VARCHAR NOT NULL
project_name - **key**	VARCHAR
project_goal	VARCHAR NOT NULL
infrastructure_management_categor	INT NOT NULL
project_status	VARCHAR
tmf_initiative	VARCHAR
software_project	BOOLEAN
incremental_development	BOOLEAN
iteration_frequency_amount	FLOAT
iteration_frequency_units	VARCHAR
iterative_description	VARCHAR
planned_start_date	DATE
projected_start_date	DATE
actual_start_date	DATE
planned_end_date	DATE
projected_end_date	DATE
actual_end_date	DATE
project_length	FLOAT
planned_cost	FLOAT
projected_cost	FLOAT
actual_cost	FLOAT
schedule_variance_days	INT
schedule_variance_percent	FLOAT
schedule_variance_color	VARCHAR
cost_variance_dollars_mill	FLOAT
cost_variance_percent	FLOAT
cost_variance_color	VARCHAR
updated_time	TIMESTAMP

EXTRACT STAGE

```
import pandas as pd
import numpy as np
pd.set_option('display.max_columns', None)

#CSV files hosted on my github
departmentsUrl = 'https://raw.githubusercontent.com/Efws777/ITProjects/main/depar
projectsUrl = 'https://raw.githubusercontent.com/Efws777/ITProjects/main/projects

#Read in two CSVs
departments = pd.read_csv(departmentsUrl)
projects = pd.read_csv(projectsUrl)
```

TRANSFORM STAGE

Add column for length of (completed) projects

```
#Turn columns in to datetime types and turn missing dates and other values into N
projects['plannedStartDate'] = pd.to_datetime(projects['plannedStartDate'])
projects['projectedStartDate'] = pd.to_datetime(projects['projectedStartDate'])
projects['actualStartDate'] = pd.to_datetime(projects['actualStartDate'])
projects['plannedEndDate'] = pd.to_datetime(projects['plannedEndDate'])
projects['projectedEndDate'] = pd.to_datetime(projects['projectedEndDate'])
projects['actualEndDate'] = pd.to_datetime(projects['actualEndDate'])
projects['updatedTime'] = pd.to_datetime(projects['updatedTime'], format='%Y-%m-%
projects['plannedStartDate'] = projects['plannedStartDate'].replace({pd.NaT: None
projects['projectedStartDate'] = projects['projectedStartDate'].replace({pd.NaT: N
projects['actualStartDate'] = projects['actualStartDate'].replace({pd.NaT: None})
projects['plannedEndDate'] = projects['plannedEndDate'].replace({pd.NaT: None})
projects['projectedEndDate'] = projects['projectedEndDate'].replace({pd.NaT: None
projects['actualEndDate'] = projects['actualEndDate'].replace({pd.NaT: None})
projects['updatedTime'] = projects['updatedTime'].replace({pd.NaT: None})
projects['incrementalDevelopment'] = projects['incrementalDevelopment'].astype('o
projects['incrementalDevelopment'] = projects['incrementalDevelopment'].replace({

#Calculate difference in start and end dates
projects['projectLength'] = pd.to_numeric((projects['actualEndDate'] - projects['

#Move projectLength column near dates section of data table
projects.insert(23, 'projectLength', projects.pop('projectLength'))

###Make agencyName column all caps to match agency names in departments table
projects['agencyName'] = projects['agencyName'].str.upper()
```

Split projects data frame into three different data frames based on investment type

```
#Get list of investment types
print(projects['investmentType'].unique())

#Data frame of Major IT Investments only
projectsMajor = projects[projects['investmentType'] == 'Major IT Investments']
projectsMajor = projectsMajor.drop(columns='investmentType')

#Data frame of Standard IT Investments only
projectsStandard = projects[projects['investmentType'] == 'Standard IT Investment
projectsStandard = projectsStandard.drop(columns='investmentType')

#Data frame of Non-major IT Investments only
projectsNonMajor = projects[projects['investmentType'] == 'Non-major IT Investmen
projectsNonMajor = projectsNonMajor.drop(columns='investmentType')
```

```
['Major IT Investments' 'Standard IT Investments'
 'Non-major IT Investments']
```

Add total employment column for each department - It is important to note that the total employment column includes ALL employees within a cabinet level department, not just those assigned to IT projects. Relative department size can still be gleaned from this data.

```
#Drop unnecessary columns.
#type column is unneccessary because all the data is the same type: 'Cabinet Leve
#dod_aggregate is unneccessary because it is an extra column full of blanks creat
#used an XML to CSV file transformer
departments = departments.drop(columns=['type', 'dod_aggregate'])

#For each department, calculate the sum of all employees in subdepartments
departmentEmployment = departments.groupby(['name']).agg({'employment': ['sum']})

#Keep one row for each agency
departments = departments.drop_duplicates(subset=['name'])

#Make name not the index column in order to conduct join in next line
departmentEmployment.reset_index(inplace=True)

#Add employment column to departments data frame
departments = departments.join(departmentEmployment.set_index('name'), on='name')

#Drop unnecessary columns
#employment is uneccessary because we have already calculated the total employmen
```

```
#for each of the cabinet level agencies
#agency_subelement is unnecessary because we only care about each cabinet level
#agency as a whole
departments = departments.drop(columns=['employment', 'agency_subelement'])

#Rename column to employmentTotal
departments.rename(columns={('employment', 'sum'): 'employmentTotal'}, inplace=Tr

#Reset index
departments.reset_index(inplace=True)
departments = departments.drop(columns=['index'])

#Drop agencies not represented in projects data frame
departments = departments.drop([0, 1, 2, 3])

#Get agencies that are in the projects data frame but not in the departments data
projects['agencyName'][~projects['agencyName'].isin(departments['name'])].unique(

#Add missing agencies to departments data frame
#They will just have missing data for employeeTotal
departments = pd.concat([departments, pd.DataFrame([{'name': 'ENVIRONMENTAL PROTE
departments = pd.concat([departments, pd.DataFrame([{'name': 'GENERAL SERVICES AD
departments = pd.concat([departments, pd.DataFrame([{'name': 'NATIONAL AERONAUTIC
departments = pd.concat([departments, pd.DataFrame([{'name': 'NUCLEAR REGULATORY
departments = pd.concat([departments, pd.DataFrame([{'name': 'NATIONAL SCIENCE FO
departments = pd.concat([departments, pd.DataFrame([{'name': 'OFFICE OF PERSONNEL
departments = pd.concat([departments, pd.DataFrame([{'name': 'SMALL BUSINESS ADMI
departments = pd.concat([departments, pd.DataFrame([{'name': 'SOCIAL SECURITY ADM
departments = pd.concat([departments, pd.DataFrame([{'name': 'U.S. AGENCY FOR INT
departments = pd.concat([departments, pd.DataFrame([{'name': 'U.S. ARMY CORPS OF
departments = pd.concat([departments, pd.DataFrame([{'name': 'NATIONAL ARCHIVES A

#Turn employment column into int type
departments['employmentTotal'] = pd.to_numeric(departments['employmentTotal'])

#Rename name column to agencyName to match projects data frame
departments.rename(columns={'name': 'agencyName'}, inplace=True)
```

```
<ipython-input-437-9cc1b8d65658>:17: FutureWarning: merging between different
  departments = departments.join(departmentEmployment.set_index('name'), on='
```

Add column for number of projects

```
#Calculate number of projects in each department
numProjects = projects.groupby(['agencyName']).agg({'projectId': ['count']})

#Make agencyName not the index column in order to conduct join in next line
numProjects.reset_index(inplace=True)

#Add project count column to departments data frame
departments = departments.join(numProjects.set_index('agencyName'), on='agencyNam

#Rename column to projectCount
departments.rename(columns={('projectId', 'count'): 'projectCount'}, inplace=True
```

```
<ipython-input-438-3da012a9cb40>:8: FutureWarning: merging between different
  departments = departments.join(numProjects.set_index('agencyName'), on='ager
```

Add three columns for proportion of projects of each investment type.Proportions and not counts were chosen because we are curious about the distribution of the three investment types among each agency.

```
#Calculate number of projects that with an investment type of 'major investment'
numMajor = projectsMajor.groupby(['agencyName']).agg({'projectId': ['count']})

#Make agencyName not the index column in order to conduct join in next line
numMajor.reset_index(inplace=True)

#Add project count column to departments data frame
departments = departments.join(numMajor.set_index('agencyName'), on='agencyName')

#Rename column to majorProp
departments.rename(columns={('projectId', 'count'): 'majorProp'}, inplace=True)

#Calculate number of projects that with an investment type of 'standard investment'
numStandard = projectsStandard.groupby(['agencyName']).agg({'projectId': ['count']}

#Make agencyName not the index column in order to conduct join in next line
numStandard.reset_index(inplace=True)

#Add project count column to departments data frame
departments = departments.join(numStandard.set_index('agencyName'), on='agencyName'

#Rename column to standardProp
departments.rename(columns={('projectId', 'count'): 'standardProp'}, inplace=True)
```

```
#Turn NaNs into 0s and turn column into int type
departments['standardProp'].fillna(0, inplace=True)
departments['standardProp'] = departments['standardProp'].astype(int)

#Calculate number of projects that with an investment type of 'standard investment'
numNonMajor = projectsNonMajor.groupby(['agencyName']).agg({'projectId': ['count']}

#Make agencyName not the index column in order to conduct join in next line
numNonMajor.reset_index(inplace=True)

#Add project count column to departments data frame
departments = departments.join(numNonMajor.set_index('agencyName'), on='agencyName'

#Rename column to nonMajorProp
departments.rename(columns={('projectId', 'count'): 'nonMajorProp'}, inplace=True)

#Turn NaNs into 0s and turn column into int type
departments['nonMajorProp'].fillna(0, inplace=True)
departments['nonMajorProp'] = departments['nonMajorProp'].astype(int)

#Turn counts into proportions and round 4 decimal digits
departments['majorProp'] = departments['majorProp'] / departments['projectCount']
departments['majorProp'] = departments['majorProp'].round(4)
departments['standardProp'] = departments['standardProp'] / departments['projectCou
departments['standardProp'] = departments['standardProp'].round(4)
departments['nonMajorProp'] = departments['nonMajorProp'] / departments['projectCou
departments['nonMajorProp'] = departments['nonMajorProp'].round(4)
```

```
<ipython-input-439-98033abd1f53>:8: FutureWarning: merging between different
  departments = departments.join(numMajor.set_index('agencyName'), on='agencyN
<ipython-input-439-98033abd1f53>:20: FutureWarning: merging between different
  departments = departments.join(numStandard.set_index('agencyName'), on='ager
<ipython-input-439-98033abd1f53>:36: FutureWarning: merging between different
  departments = departments.join(numNonMajor.set_index('agencyName'), on='ager
```

Add average project length column to departments data frame. Mean was chosen because there were no clear cluster of outliers influencing the mean of the data so it wouldn't be necessary to use median in this case.

```
#Get mean project length for each department
aveLength = projects.groupby(['agencyName']).agg({'projectLength': 'mean'}).round(2

#Make agencyName not the index column in order to conduct join in next line
aveLength.reset_index(inplace=True)

#Add column to departments data frame
departments = departments.join(aveLength.set_index('agencyName'), on='agencyName')

#Rename column to meanLength
departments.rename(columns={'projectLength': 'meanLength'}, inplace=True)
```

Add median cost column and median cost variance percentage column to departments data frame. Median was chosen for cost and cost variance because many values in the actual_cost column were 0 for projects that were not done yet. This causes the data to have a right skew, so using a median would give us a more accurate center of the data.

```
#Get median project cost (for completed projects only) for each department
aveCost = projects.groupby(['agencyName']).agg({'actualCost': 'median'}).round(2)

#Make agencyName not the index column in order to conduct join in next line
aveCost.reset_index(inplace=True)

#Add column to departments data frame
departments = departments.join(aveCost.set_index('agencyName'), on='agencyName')

#Rename column to medianCost
departments.rename(columns={'actualCost': 'medianCost'}, inplace=True)

#Replace average cost variances of 0 (ongoing projects) with NAs so that the mean
projects['costVariance(%)'] = projects['costVariance(%)'].replace(0, np.nan)

#Get median project cost (for completed projects only) for each department
aveCost = projects.groupby(['agencyName']).agg({'costVariance(%)': 'median'}).rou

#Make agencyName not the index column in order to conduct join in next line
aveCost.reset_index(inplace=True)

#Add column to departments data frame
departments = departments.join(aveCost.set_index('agencyName'), on='agencyName')

#Rename column to medianCostVariance(%)
departments.rename(columns={'costVariance(%)': 'medianCostVariance(%)'}, inplace=
```

LOAD STAGE

Import necessary functions to connect and work with AWS database

```
# run_query function
def run_query(query_string):

 conn, cur = get_conn_cur() # get connection and cursor

 cur.execute(query_string) # executing string as before

 my_data = cur.fetchall() # fetch query data as before

 # here we're extracting the 0th element for each item in cur.description
 colnames = [desc[0] for desc in cur.description]
```

```
    cur.close() # close
    conn.close() # close

    return(colnames, my_data) # return column names AND data

  # Column name function for checking out what's in a table
  def get_column_names(table_name): # arguement of table_name
    conn, cur = get_conn_cur() # get connection and cursor

    # Now select column names while inserting the table name into the WERE
    column_name_query = """SELECT column_name FROM information_schema.columns
       WHERE table_name = '%s' """ %table_name

    cur.execute(column_name_query) # exectue
    my_data = cur.fetchall() # store

    cur.close() # close
    conn.close() # close

    return(my_data) # return

  # Check table_names
  def get_table_names():
    conn, cur = get_conn_cur() # get connection and cursor

    # query to get table names
    table_name_query = """SELECT table_name FROM information_schema.tables
         WHERE table_schema = 'public' """

    cur.execute(table_name_query) # execute
    my_data = cur.fetchall() # fetch results

    cur.close() #close cursor
    conn.close() # close connection

    return(my_data) # return your fetched results

  # make sql_head function
  def sql_head(table_name):
    conn, cur = get_conn_cur() # get connection and cursor

    # Now select column names while inserting the table name into the WERE
    head_query = """SELECT * FROM %s LIMIT 5; """ %table_name
```

```
    cur.execute(head_query) # exectue
    colnames = [desc[0] for desc in cur.description] # get column names
    my_data = cur.fetchall() # store first five rows

    cur.close() # close
    conn.close() # close

    df = pd.DataFrame(data = my_data, columns = colnames) # make into df

    return(df) # return

# drop a table from your rdb (if you try to create a table that already exists, i
def my_drop_table(tab_name):
  conn, cur = get_conn_cur()
  tq = """DROP TABLE IF EXISTS %s CASCADE;""" %tab_name
  cur.execute(tq)
  conn.commit()


###Connect to database
import psycopg2
def get_conn_cur():
  conn = psycopg2.connect(
    host="final-proj-db.czeyu24ewxmh.us-east-2.rds.amazonaws.com",
    database="final_proj",
    user="postgres",
    password="12345678",
    port='5432')
  cur = conn.cursor()   # Make a cursor after

  return(conn, cur)   # Return both the connection and the cursor


# Drop major table if it already exists
my_drop_table('major')
```

```
###Create table for major investment projects
tq = """CREATE TABLE major (
     agency_code INT NOT NULL,
     agency_name VARCHAR NOT NULL,
     current_uii VARCHAR NOT NULL,
     investment_name VARCHAR NOT NULL,
     project_id VARCHAR,
     agency_project_id VARCHAR NOT NULL,
     project_name VARCHAR,
     PRIMARY KEY (project_id, project_name),
     project_goal VARCHAR NOT NULL,
     infrastructure_management_category INT NOT NULL,
     project_status VARCHAR,
     tmf_initiative VARCHAR,
     software_project BOOLEAN,
     incremental_development BOOLEAN,
     iteration_frequency_amount FLOAT,
     iteration_frequency_units VARCHAR,
     iterative_description VARCHAR,
     planned_start_date DATE,
     projected_start_date DATE,
     actual_start_date DATE,
     planned_end_date DATE,
     projected_end_date DATE,
     actual_end_date DATE,
     project_length FLOAT,
     planned_cost FLOAT,
     projected_cost FLOAT,
     actual_cost FLOAT,
     schedule_variance_days INT,
     schedule_variance_percent FLOAT,
     schedule_variance_color VARCHAR,
     cost_variance_dollars_mill FLOAT,
     cost_variance_percent FLOAT,
     cost_variance_color VARCHAR,
     updated_time TIMESTAMP
     );"""

conn, cur = get_conn_cur()
cur.execute(tq)
conn.commit()
```

```
#Make sure the table is in the database
get_table_names()
```

```
[('standard',), ('non_major',), ('departments',), ('major',)]
```

```
#Make sure all the columns are in table
get_column_names(table_name='major')
```

```
[('updated_time',),
 ('actual_start_date',),
 ('planned_end_date',),
 ('projected_end_date',),
 ('actual_end_date',),
 ('project_length',),
 ('planned_cost',),
 ('projected_cost',),
 ('actual_cost',),
 ('schedule_variance_days',),
 ('schedule_variance_percent',),
 ('cost_variance_dollars_mill',),
 ('cost_variance_percent',),
 ('agency_code',),
 ('infrastructure_management_category',),
 ('software_project',),
 ('incremental_development',),
 ('iteration_frequency_amount',),
 ('planned_start_date',),
 ('projected_start_date',),
 ('agency_name',),
 ('current_uii',),
 ('investment_name',),
 ('project_id',),
 ('agency_project_id',),
 ('project_name',),
 ('project_goal',),
 ('iteration_frequency_units',),
 ('project_status',),
 ('tmf_initiative',),
 ('iterative_description',),
 ('schedule_variance_color',),
 ('cost_variance_color',)]
```

```
major_np = projectsMajor.to_numpy();
major_np[:,1] = np.vectorize(lambda x: str(x))(major_np[:,1])
data_tups = [tuple(x) for x in major_np]

iq = """INSERT INTO major(agency_code,agency_name,current_uii,investment_name,pro
iq
```

```
'INSERT INTO major(agency_code,agency_name,current_uii,investment_name,projec
t_id,agency_project_id,project_name,project_goal,infrastructure_management_ca
tegory,project_status,tmf_initiative,software_project,incremental_developmen
t,iteration_frequency_amount,iteration_frequency_units,iterative_description,
planned_start_date,projected_start_date,actual_start_date,planned_end_date,pr
ojected_end_date,actual_end_date,project_length,planned_cost,projected_cost,a
```

```
#Load data into table
conn, cur = get_conn_cur()
cur.executemany(iq, data_tups)
conn.commit()
conn.close()
```

```
#Check table
sql_head(table_name='major')
```

	agency_code	agency_name	current_uii	investment_name	project_id	age
0	6	DEPARTMENT OF COMMERCE	006-000000136	Census - Enterprise Data Lake (EDL)	0001D21002	6037e537
1	6	DEPARTMENT OF COMMERCE	006-000000136	Census - Enterprise Data Lake (EDL)	0001P22003	61bbba08
2	6	DEPARTMENT OF COMMERCE	006-000000136	Census - Enterprise Data Lake (EDL)	0001P22004	61bbbfcd
3	6	DEPARTMENT OF COMMERCE	006-000000136	Census - Enterprise Data Lake (EDL)	0001P22006	61bbc519
4	6	DEPARTMENT OF COMMERCE	006-000000136	Census - Enterprise Data Lake (EDL)	0001P22002	61bbab91

```
# Drop standard table if it already exists
my_drop_table('standard')
```

```
###Create table for standard investment projects
tq = """CREATE TABLE standard (
     agency_code INT NOT NULL,
     agency_name VARCHAR NOT NULL,
     current_uii VARCHAR NOT NULL,
     investment_name VARCHAR NOT NULL,
     project_id VARCHAR,
     agency_project_id VARCHAR NOT NULL,
     project_name VARCHAR,
     PRIMARY KEY (project_id, project_name),
     project_goal VARCHAR NOT NULL,
     infrastructure_management_category INT NOT NULL,
     project_status VARCHAR,
     tmf_initiative VARCHAR,
     software_project BOOLEAN,
     incremental_development BOOLEAN,
     iteration_frequency_amount FLOAT,
     iteration_frequency_units VARCHAR,
     iterative_description VARCHAR,
     planned_start_date DATE,
     projected_start_date DATE,
     actual_start_date DATE,
     planned_end_date DATE,
     projected_end_date DATE,
     actual_end_date DATE,
     project_length FLOAT,
     planned_cost FLOAT,
     projected_cost FLOAT,
     actual_cost FLOAT,
     schedule_variance_days INT,
     schedule_variance_percent FLOAT,
     schedule_variance_color VARCHAR,
     cost_variance_dollars_mill FLOAT,
     cost_variance_percent FLOAT,
     cost_variance_color VARCHAR,
     updated_time TIMESTAMP
     );"""

conn, cur = get_conn_cur()
cur.execute(tq)
conn.commit()
```

```
#Make sure the table is in the database
get_table_names()
```

```
[('non_major',), ('departments',), ('major',), ('standard',)]
```

```
#Make sure all the columns are in table
get_column_names(table_name='standard')
```

```
[('updated_time',),
 ('actual_start_date',),
 ('planned_end_date',),
 ('projected_end_date',),
 ('actual_end_date',),
 ('project_length',),
 ('planned_cost',),
 ('projected_cost',),
 ('actual_cost',),
 ('schedule_variance_days',),
 ('schedule_variance_percent',),
 ('cost_variance_dollars_mill',),
 ('cost_variance_percent',),
 ('agency_code',),
 ('infrastructure_management_category',),
 ('software_project',),
 ('incremental_development',),
 ('iteration_frequency_amount',),
 ('planned_start_date',),
 ('projected_start_date',),
 ('agency_name',),
 ('current_uii',),
 ('investment_name',),
 ('project_id',),
 ('agency_project_id',),
 ('project_name',),
 ('project_goal',),
 ('iteration_frequency_units',),
 ('project_status',),
 ('tmf_initiative',),
 ('iterative_description',),
 ('schedule_variance_color',),
 ('cost_variance_color',)]
```

```
#Turn each row of data from dataframe into a tuple for insertion into table
standard_np = projectsStandard.to_numpy();
standard_np[:,1] = np.vectorize(lambda x: str(x))(standard_np[:,1])
data_tups = [tuple(x) for x in standard_np]

iq = """INSERT INTO standard(agency_code,agency_name,current_uii,investment_name,
iq
```

```
'INSERT INTO standard(agency_code,agency_name,current_uii,investment_name,pro
ject_id,agency_project_id,project_name,project_goal,infrastructure_management
_category,project_status,tmf_initiative,software_project,incremental_developm
ent,iteration_frequency_amount,iteration_frequency_units,iterative_descriptio
n,planned_start_date,projected_start_date,actual_start_date,planned_end_date,
projected_end_date,actual_end_date,project_length,planned_cost,projected_cos
```

```
#Load data into table
conn, cur = get_conn_cur()
cur.executemany(iq, data_tups)
conn.commit()
conn.close()
```

```
#Check table
sql_head(table_name='standard')
```

	agency_code	agency_name	current_uii	investment_name	project_id	age
0	6	DEPARTMENT OF COMMERCE	006-000000132	NIST Application (APP)	Cyberlock	5ed04b8d
1	6	DEPARTMENT OF COMMERCE	006-000000117	NIST Network (NET)	PACS	5ed0569b
2	6	DEPARTMENT OF COMMERCE	006-000000117	NIST Network (NET)	NW2021	5feb564a
3	6	DEPARTMENT OF COMMERCE	006-000000117	NIST Network (NET)	NISTHSN	614b6f7
4	6	DEPARTMENT OF COMMERCE	006-000370600	NOAA/NOAA/ N-WAVE (Network Report)	3511M11012	5ed054dd

```
# Drop nonMajor table if it already exists
my_drop_table('non_major')
```

```
###Create table for non major investment projects
tq = """CREATE TABLE non_major (
     agency_code INT NOT NULL,
     agency_name VARCHAR NOT NULL,
     current_uii VARCHAR NOT NULL,
     investment_name VARCHAR NOT NULL,
     project_id VARCHAR,
     agency_project_id VARCHAR NOT NULL,
     project_name VARCHAR,
     PRIMARY KEY (project_id, project_name),
     project_goal VARCHAR NOT NULL,
     infrastructure_management_category INT NOT NULL,
     project_status VARCHAR,
     tmf_initiative VARCHAR,
     software_project BOOLEAN,
     incremental_development BOOLEAN,
     iteration_frequency_amount FLOAT,
     iteration_frequency_units VARCHAR,
     iterative_description VARCHAR,
     planned_start_date DATE,
     projected_start_date DATE,
     actual_start_date DATE,
     planned_end_date DATE,
     projected_end_date DATE,
     actual_end_date DATE,
     project_length FLOAT,
     planned_cost FLOAT,
     projected_cost FLOAT,
     actual_cost FLOAT,
     schedule_variance_days INT,
     schedule_variance_percent FLOAT,
     schedule_variance_color VARCHAR,
     cost_variance_dollars_mill FLOAT,
     cost_variance_percent FLOAT,
     cost_variance_color VARCHAR,
     updated_time TIMESTAMP
     );"""

conn, cur = get_conn_cur()
cur.execute(tq)
conn.commit()
```

```
#Make sure the table is in the database
get_table_names()
```

```
[('departments',), ('major',), ('standard',), ('non_major',)]
```

```
#Make sure all the columns are in table
get_column_names(table_name='non_major')
```

```
[('updated_time',),
 ('actual_start_date',),
 ('planned_end_date',),
 ('projected_end_date',),
 ('actual_end_date',),
 ('project_length',),
 ('planned_cost',),
 ('projected_cost',),
 ('actual_cost',),
 ('schedule_variance_days',),
 ('schedule_variance_percent',),
 ('cost_variance_dollars_mill',),
 ('cost_variance_percent',),
 ('agency_code',),
 ('infrastructure_management_category',),
 ('software_project',),
 ('incremental_development',),
 ('iteration_frequency_amount',),
 ('planned_start_date',),
 ('projected_start_date',),
 ('agency_name',),
 ('current_uii',),
 ('investment_name',),
 ('project_id',),
 ('agency_project_id',),
 ('project_name',),
 ('project_goal',),
 ('iteration_frequency_units',),
 ('project_status',),
 ('tmf_initiative',),
 ('iterative_description',),
 ('schedule_variance_color',),
 ('cost_variance_color',)]
```

```
#Turn each row of data from dataframe into a tuple for insertion into table
non_major_np = projectsNonMajor.to_numpy();
non_major_np[:,1] = np.vectorize(lambda x: str(x))(non_major_np[:,1])
data_tups = [tuple(x) for x in non_major_np]

iq = """INSERT INTO non_major(agency_code,agency_name,current_uii,investment_name
iq
```

```
'INSERT INTO non_major(agency_code,agency_name,current_uii,investment_name,pr
oject_id,agency_project_id,project_name,project_goal,infrastructure_managemen
t_category,project_status,tmf_initiative,software_project,incremental_develop
ment,iteration_frequency_amount,iteration_frequency_units,iterative_descripti
on,planned_start_date,projected_start_date,actual_start_date,planned_end_dat
e,projected_end_date,actual_end_date,project_length,planned_cost,projected_co
```

```
#Load data into table
conn, cur = get_conn_cur()
cur.executemany(iq, data_tups)
conn.commit()
conn.close()
```

```
#Check table
sql_head(table_name='non_major')
```

	agency_code	agency_name	current_uii	investment_name	project_id	age
0	6	DEPARTMENT OF COMMERCE	006-000400800	Census - Field Support Systems	4008D20001	5ed050d5
1	6	DEPARTMENT OF COMMERCE	006-000400800	Census - Field Support Systems	4008D23001	61fbf8a0
2	6	DEPARTMENT OF COMMERCE	006-000400800	Census - Field Support Systems	4008D24001	61fbf94
3	6	DEPARTMENT OF COMMERCE	006-000404200	Census - Personnel & Employment Check Systems	4042D21001	6022f037
4	6	DEPARTMENT OF COMMERCE	006-000551500	BIS Mission Applications	2	5f60d7c

```
# Drop nonMajor table if it already exists
my_drop_table('departments')
```

```
###Create table for non federal departments
tq = """CREATE TABLE departments (
     agency_name VARCHAR PRIMARY KEY,
     employment_total FLOAT,
     project_count INT NOT NULL,
     major_prop FLOAT NOT NULL,
     standard_prop FLOAT NOT NULL,
     non_major_prop FLOAT NOT NULL,
     mean_length FLOAT NOT NULL,
     median_cost FLOAT NOT NULL,
     median_cost_variance_percent FLOAT NOT NULL
     );"""

conn, cur = get_conn_cur()
cur.execute(tq)
conn.commit()


#Make sure the table is in the database
get_table_names()

     [('departments',), ('major',), ('standard',), ('non_major',)]


#Make sure all the columns are in table
get_column_names(table_name='departments')

     [('median_cost_variance_percent',),
      ('employment_total',),
      ('project_count',),
      ('major_prop',),
      ('standard_prop',),
      ('non_major_prop',),
      ('mean_length',),
      ('median_cost',),
      ('agency_name',)]
```

```
#Turn each row of data from dataframe into a tuple for insertion into table
departments_np = departments.to_numpy();
departments_np[:,1] = np.vectorize(lambda x: str(x))(departments_np[:,1])
data_tups = [tuple(x) for x in departments_np]

iq = """INSERT INTO departments(agency_name,employment_total,project_count,major_
iq
```

```
'INSERT INTO departments(agency_name,employment_total,project_count,major_pro
p,standard_prop,non_major_prop,mean_length,median_cost,median_cost_variance_p
ercent) VALUES (%s, %s, %s, %s, %s, %s, %s, %s, %s)'
```

```
#Load data into table
conn, cur = get_conn_cur()
cur.executemany(iq, data_tups)
conn.commit()
conn.close()


#Check table
sql_head(table_name='departments')
```

	agency_name	employment_total	project_count	major_prop	standard_prop	no
0	DEPARTMENT OF AGRICULTURE	96340.0	1015	0.4650	0.4335	
1	DEPARTMENT OF COMMERCE	50846.0	301	0.7409	0.2326	
2	DEPARTMENT OF JUSTICE	113694.0	145	0.7724	0.2276	
3	DEPARTMENT OF LABOR	16246.0	145	0.8759	0.1241	

```
#Project cost data for major investment projects where the cost variance is less
run_query(
"""SELECT major.actual_cost, major.cost_variance_dollars_mill, major.cost_varianc
FROM major
JOIN departments ON major.agency_name = departments.agency_name
WHERE major.cost_variance_percent < -100;""")
```

```
    (['actual_cost',
      'cost_variance_dollars_mill',
      'cost_variance_percent',
      'median_cost_variance_percent'],
     [(2.29, 3.27261, -142.9087, 2.7),
      (1.0, 6.411105, -641.1105, 2.7),
      (19.77865, -10.010066, -102.472027, 2.7),
      (7.022, 9.9135, -141.1777, 2.7),
      (2.308698, -1.366518, -145.037891, 2.7),
      (3.80713, -2.359366, -162.966202, 2.7),
      (0.0, -50.599849, -419.910497, 4.86),
      (0.0, -104.08, -488.638498, 4.86),
      (5.874, 1.509602, -125.4343, 18.6),
      (0.539304, -0.354094, -191.185141, 18.6),
      (3.822, 0.649, -144.2222, 18.6),
      (0.465, 0.135835, -100.6185, 18.6),
      (0.10005, 0.12165, -121.5892, 2.32),
      (0.00757, 0.01763, -232.893, 2.32),
      (0.623, 0.221, -107.8049, 0.34),
      (12.0937, -7.5201, -164.424086, -14.0),
      (7.181051, -6.952451, -3041.317148, -14.0),
      (0.0, -4.459, -289.35756, -14.0),
      (0.428778, -0.286156, -200.639453, -14.0),
      (2.099286, -1.584921, -308.13158, 3.04),
      (0.643703, 0.291147, -142.3235, 3.73),
      (8.430194, -4.786256, -131.348448, 3.73),
      (nan, -5.942328, -621.956512, 3.73)])
```

```
#Department data for each standard investment project projected to cost over $40 m
run_query(
"""SELECT departments.agency_name, departments.employment_total, departments.proj
FROM departments
JOIN standard ON departments.agency_name = standard.agency_name
WHERE standard.projected_cost > 40 AND standard.projected_cost <> 'nan';""")
```

```
(['agency_name',
  'employment_total',
  'project_count',
  'mean_length',
  'project_id',
  'projected_cost'],
 [('GENERAL SERVICES ADMINISTRATION', nan, 267, 339.19, 'FY23-1', 66.987341),
  ('GENERAL SERVICES ADMINISTRATION', nan, 267, 339.19, 'FY24-1', 67.665678),
  ('SOCIAL SECURITY ADMINISTRATION', nan, 25, 939.86, '01', 685.66572),
  ('DEPARTMENT OF THE TREASURY', 109145.0, 303, 362.76, '200656', 60.5),
  ('DEPARTMENT OF THE TREASURY', 109145.0, 303, 362.76, '1446', 208.0),
  ('DEPARTMENT OF THE TREASURY', 109145.0, 303, 362.76, '202134', 134.452),
  ('DEPARTMENT OF THE TREASURY', 109145.0, 303, 362.76, '202722', 83.265223),
  ('DEPARTMENT OF AGRICULTURE', 96340.0, 1015, 355.16, '197896', 108.0),
  ('DEPARTMENT OF AGRICULTURE', 96340.0, 1015, 355.16, '246416', 60.909),
  ('DEPARTMENT OF AGRICULTURE', 96340.0, 1015, 355.16, '268306', 63.924),
  ('DEPARTMENT OF AGRICULTURE', 96340.0, 1015, 355.16, '224326', 127.337),
  ('DEPARTMENT OF AGRICULTURE', 96340.0, 1015, 355.16, '294721', 108.055)])
```

```
#Summary statistics: # of major investment projects, average planned cost of stan
run_query(
"""SELECT
    (SELECT COUNT(*) FROM major) AS major_count,
    (SELECT AVG(planned_cost) FROM standard WHERE planned_cost <> 'nan') AS avg_s
    (SELECT MAX(project_length) FROM non_major WHERE project_length <> 'nan') AS m

    """)
```

```
(['major_count', 'avg_standard_planned_cost', 'max_non_major_length'],
 [(3445, 7.1813975567451775, 1460.0)])
```