Part 1:

1.

a. Please find the code on this github link. Additionally, I have attached the code files in the email as well.

b. As per the assignment, I used the state level dataset provided here. The reason why I used only state specific data because the metrics (name popularity, etc.) that I wanted to calculate could be obtained at a national level by aggregating the state level data.

2.

a. Firstly there were multiple txt files by state in the raw dataset, and to be able to leverage the geography aspect of the data, I decided to read and combine these files into one csv file.

```
with open(os.path.join(root_dir, 'output_file.csv'), 'w+') as csv_file:
    for path in glob.glob('./*.TXT'):
        with open(path) as txt_file:
            txt = txt_file.read() + '\n'
            csv_file.write(txt)

os.chdir(root_dir)

data = pd.read_csv('output_file.csv', header=None)
data.columns = ["state", "gender", "year", "name", "count"]
```

b. I then tried to understand the basics of my dataset, like how does the data look like, what the shape of the dataset is like, what are the column types and if it has any null values, the distribution of the numerical variables. During this exploration, I found that the all of the columns are object data types except year and count. I decided to change the year into a string column except the count column. The dataset has 64,08,041 rows and I found that the min value in this dataset is 5, and max value is 10,027 and mean is 50. This tells me that the data is highly skewed due to the big range of the data.

```python
# Understanding the data
print(data.head())
print(data.shape)
print(data.info())

data["year"] = data["year"].astype(str)

print(data.describe())
print(data["name"].value_counts().shape)
```

```
  state gender  year      name  count
0    ID      F  1910      Mary     53
1    ID      F  1910   Dorothy     31
2    ID      F  1910     Helen     30
3    ID      F  1910  Margaret     24
4    ID      F  1910      Ruth     24
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 6408041 entries, 0 to 6408040
Data columns (total 5 columns):
 #   Column  Dtype
---  ------  -----
 0   state   object
 1   gender  object
 2   year    int64
 3   name    object
 4   count   int64
dtypes: int64(2), object(3)
```

```
              count
count  6.408041e+06
mean   5.033971e+01
std    1.720529e+02
min    5.000000e+00
25%    7.000000e+00
50%    1.200000e+01
75%    3.300000e+01
max    1.002700e+04
```

c. According the readME provided in the raw dataset, there were some conditions like state code is always 2 digits, year is always 4 digits, hence I wanted to run some quality checks on the data as well. Therefore, I created some functions that does those checks for me.

```
# Data checks
def isnull(data):
    print(data.isnull().any())

isnull(data)

def check_state_length(data):
    state_length = data['state'].str.len()
    state_length = state_length.to_numpy()
    if((state_length[0] == state_length).all()) == True:
        print("Check pass")
    else:
        print("Data quality issue found")

check_state_length(data)
```

```
def check_year_length(data):
    year_length = data['state'].str.len()
    year_length = year_length.to_numpy()
    if((year_length[0] == year_length).all()) == True:
        print("Check pass")
    else:
        print("Data quality issue found")

check_year_length(data)
```

d. I was then interested in finding popular gender neutral names as well. How I defined popular gender neutral was if the name has more than 500 occurrences of it, and the absolute difference of female proportions and male proportions is < 0.3 (Range from 0 and 1 in which 0 is an equal split among the genders, and 1 being a name used only by 1 gender).

```python
# Function to create gender neutral names
def gender_neutral_names(data):
    """
    Returns gender neutral names

    data: dataframe
    """

    df_m=data[data['gender']=='M'].groupby(['name','gender'])['count'].sum()
    df_f=data[data['gender']=='F'].groupby(['name','gender'])['count'].sum()
    df_unisex=pd.merge(df_f,df_m,how='inner',on='name').reset_index()
    df_unisex["total"] = df_unisex['count_y']+ df_unisex['count_x']
    df_unisex['prop_x']=df_unisex['count_x']/df_unisex['total']
    df_unisex['prop_y']=df_unisex['count_y']/df_unisex['total']
    df_unisex['diff']=abs(df_unisex['prop_x']-df_unisex['prop_y'])
    df_unisex[(df_unisex['total']>=500) & (df_unisex['diff']<0.3)]
    print(df_unisex['name'])

gender_neutral_names(data)
```

Results:

```
0        Aaliyah
1          Aaron
2          Aarya
3          Aaryn
4           Abby
          ...
1705        Zuri
1706      Zuriel
1707      Zyaire
1708        Zyan
1709        Zyon
Name: name, Length: 1710, dtype: object
```

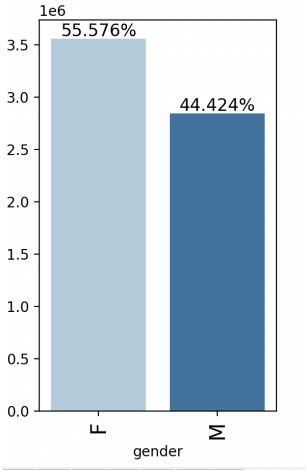
This tells me that there were 3184 gender neutral names that fits the conditions I specified above.

e. I then wanted to leverage matplotlib and seaborn to do some exploratory visualization as well on the data. Hence I made a function so that I could visualize labelled barplots.

```
# Function to create labeled barplots
def labeled_barplot(data, feature, perc=False, n=None):
    """
    Barplot with percentage at the top

    data: dataframe
    feature: dataframe column
    perc: whether to display percentages instead of count (default is False)
    n: displays the top n category levels (default is None, i.e., display all levels)
    """

    total = len(data[feature])  # length of the column
    count = data[feature].nunique()
    if n is None:
        plt.figure(figsize=(count + 1, 5))
    else:
        plt.figure(figsize=(n + 1, 5))

    plt.xticks(rotation=90, fontsize=15)
    ax = sns.countplot(
        data=data,
        x=feature,
        palette="Paired",
        order=data[feature].value_counts().index[:n].sort_values(),
    )

    for p in ax.patches:
        if perc == True:
            label = "{:.3f}%".format(
                100 * p.get_height() / total
            )  # percentage of each class of the category
        else:
            label = p.get_height()  # count of each level of the category

        x = p.get_x() + p.get_width() / 2  # width of the plot
        y = p.get_height()  # height of the plot

        ax.annotate(
            label,
            (x, y),
            ha="center",
            va="center",
            size=12,
            xytext=(0, 5),
            textcoords="offset points",
        )  # annotate the percentage

    plt.show()  # show the plot
```

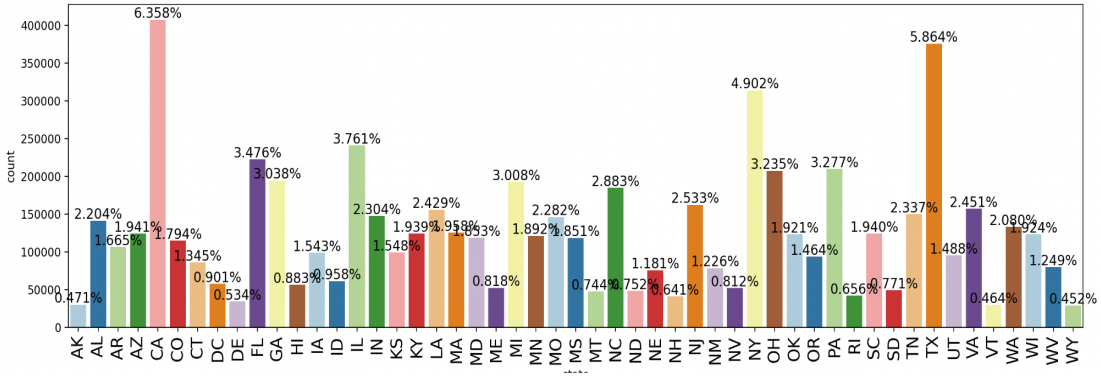
Here are the results for gender and state:

Gender:



This shows that 55.576% of the names in this dataset are female names.

State:



This shows that highest proportion of names are coming from California.

3. I got the following error that I encountered in steps 1 and 2:
    a. The first error was when I was combining the txt files, and in the loop where I was combing them as a csv file: `with open(os.path.join(root_dir, 'output_file.csv'), 'w+') as csv_file:`
    There is an argument “w+” which means write and overwrite, however, this was “a” before which meant write and append, in which it kept appending to “output_file.csv” with multiple runs instead of overwriting, hence it was duplicating the results, and also increasing the file size immensely which resulted in the script running very slow.
4. I believe if we had population data over the name data, I think it would have been very interesting to see the parallel trends between the population and name data. For instance, if there is jump in ratio of unique names, is there also a jump in the population. We could have a separate population table, and we could combine the names table on state and year.
5. Please find the powerBI dashboard to see some of the trends and findings I found. I have determined top 10 trending names, top 10 female and male trending names, and it can be filtered on year and state. I have also found the trend of distinct names per year. Additionally, I have also found the male and female names split in each state.

Part 2:

1.
    a. When we are building models in PowerBI, setting the storage mode is very important as it gives the developer control if they want to cache table data in memory or not. Setting storage provides many benefits such as boosting query performance, enable analysis over large datasets, get real time data to reduce data latency where needed. The schema provided shows us that its a Product and Store has a one to many relationship with Sales where Store/Product have one instance of a particular value, and the Sales table can have more than one instance of a value. Depending on how the storage mode is set for these tables, if it is a direct query for all tables, it is going to take a lot of time to query all the data in Product, Stores and Sales data, decrease query performance and interactivity with the visuals.
    b. To avoid any data issues, I have the following recommendations for each table:
        i. Sales: Report queries based on Sales tables should run in DirectQuery mode. These queries may take longer since they are closer to real time, no caching latency exists.
        ii. Storage and Product: These tables should store in Import mode. Queries based on Storage and Product tables will be returned from the in-memory cache and will be relatively fast.

2. There should be 2 dimension tables - Customer and Date, where the Customer Table, Date table and Sales table should have an extra field called “OrderID”, and these tables will join the sales on again a one-to-many relationship. These dimension tables should be stored in Dual mode, so load times of initial reports are fast when they retrieve slicer values to display.