<u>Title:</u> PYTHON DATA STRUCTURES

<u>Objective</u>: Develop an application that uses different python data Structures

<u>Source Code:</u>

```
Dict = {1: 'Data', 2: 'Information', 3: 'Processed Data'}
print("Dictionary with the use of Integer Keys: ")
print(Dict)

#Add Elements
Dict[4]="New"

#Display Elements
print(Dict)

#Access Keys
print(Dict.keys())

#Access Values
print(Dict.values())
# Access Key Value Pairs
print(Dict)
#List
# create a list
prime_numbers = [2, 3, 5]

# create another list
numbers = [1, 4]

# add all elements of prime_numbers to numbers
numbers.extend(prime_numbers)

print('List after extend():', numbers)

#Add elements by extend
# languages list
languages = ['French', 'English']

# another list of language
languages1 = ['Spanish', 'Portuguese']
```

```
# appending language1 elements to language
languages.extend(languages1)


print('Languages List:', languages)

#Add elements by insert
numbers.insert(6,89)

print(numbers)

#delete the elements
numbers.remove(89)

print(numbers)

#TUPLE

tup=(1,2,3,4,5,6,7,8,9,10)

l1=list(tup)

l1.append(89)

tup=tuple(l1)

print(tup)

#SET

A={"Apple","Banana","Mango"}

B={"Apple","Chickoo"}

A.add("Guava")

print("Union :", A | B)

print("Intersection :", A & B)

print("Difference :", A - B)

print("Symmetric difference :", A ^ B)
```

Output:

```
Dictionary with the use of Integer Keys:
{1: 'Data', 2: 'Information', 3: 'Processed Data'}
{1: 'Data', 2: 'Information', 3: 'Processed Data', 4: 'New'}
dict_keys([1, 2, 3, 4])
dict_values(['Data', 'Information', 'Processed Data', 'New'])
{1: 'Data', 2: 'Information', 3: 'Processed Data', 4: 'New'}
List after extend(): [1, 4, 2, 3, 5]
Languages List: ['French', 'English', 'Spanish', 'Portuguese']
[1, 4, 2, 3, 5, 89]
[1, 4, 2, 3, 5]
(1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 89)
Union : {'Chickoo', 'Apple', 'Guava', 'Banana', 'Mango'}
Intersection : {'Apple'}
Difference : {'Guava', 'Banana', 'Mango'}
Symmetric difference : {'Chickoo', 'Banana', 'Mango', 'Guava'}
```

Result: The experiment is successfully completed with the desired output.

Date: **Signature:**

Title: **PYTHON STRING FUNCTIONS**

Objective: Write a program to explore different string functions

Source Code:

```
str1="Varun"
str1=str1.capitalize()
print(str1)
str2="R"
print(str2)
print(str2.lower())
txt = "DAP Experiment - 2"
x = txt.title()
print(x)
print(x.casefold())
x=x.upper()
print(x)
print(x.count('a'))
print(x.count('E'))
print(x.find('-'))
print(x.find('A'))
print(x.find('D'))
print(x)
x=x.replace('2','II')
print(x)
print(x.swapcase())

str3="au"
"f".join(str3)
```

Output:

```
Varun
R
r
Dap Experiment - 2
dap experiment - 2
DAP EXPERIMENT - 2
0
3
15
1
0
DAP EXPERIMENT - 2
DAP EXPERIMENT - II
dap experiment - ii

Out[2]: 'afu'
```

Result: The experiment is successfully completed with the desired output.

Date: **Signature:**

Title: **REGULAR EXPRESSION**

Objective: Demonstrate usage of regular expression

Source Code:

```
import re
t = "The rain in Spain"
x = re.search("^The.*Spain$", t)
print(x)
txt = "The rain in Spain"
x = re.findall("ai", txt)
print(x)
txt = "The rain in Spain"
x = re.findall("Portugal", txt)
print(x)
#Search for the first white-space character in the string:
txt = "The rain in Spain"
x = re.search("\s", txt)
print("The first white-space character is located in position:", x.start())
#Make a search that returns no match:
txt = "The rain in Spain"
x = re.search("Portugal", txt)
print(x)
#split function
txt = "The rain in Spain"
x = re.split("\s", txt)
print(x)
#control the maxsplit parameter
txt = "The rain in Spain"
x = re.split("\s", txt, 1)
```

```
print(x)

#sub function: The sub() function replaces the matches with the text of your choice:
txt = "The rain in Spain"
x = re.sub("\s", "9", txt)
print(x)
#Do a search that will return a Match Object:
txt = "The rain in Spain"
x = re.search("ai", txt)
print(x) #this will print an object
txt = "The rain in Spain"
x = re.search(r"\bS\w+", txt)
print(x.span())
txt = "The rain in Spain"
x = re.search(r"\bS\w+", txt)
print(x.string)
txt = "The rain in Spain"
x = re.search(r"\bS\w+", txt)
print(x.group())
```

Output:

```
<re.Match object; span=(0, 17), match='The rain in Spain'>
['ai', 'ai']
[]
The first white-space character is located in position: 3
None
['The', 'rain', 'in', 'Spain']
['The', 'rain in Spain']
The9rain9in9Spain
<re.Match object; span=(5, 7), match='ai'>
(12, 17)
The rain in Spain
Spain
```

```
The decision tree for the dataset using ID3 algorithm is
 Outlook
    rain
       Wind
          strong
             no
          weak
             yes
    overcast
       yes
    sunny
       Humidity
          high
             no
          normal
             yes
The test instance: ['rain', 'cool', 'normal', 'strong']
The label for test instance:  no
The test instance: ['sunny', 'mild', 'normal', 'strong']
The label for test instance:  yes
```

Result: The experiment is successfully completed with the desired output

Date: **Signature:**

Title: **NUMPY LIBRARIES**

Objective: Demonstrate the usage of numpy libraries

Source Code:

```
import numpy as np
arr = np.array(42)
print(arr)
arr = np.array([1, 2, 3, 4, 5])
print(arr)
arr = np.array([[1, 2, 3], [4, 5, 6]])
print(arr)
arr = np.array([[[1, 2, 3], [4, 5, 6]], [[1, 2, 3], [4, 5, 6]]])
print(arr)
a = np.array(42)
b = np.array([1, 2, 3, 4, 5])
c = np.array([[1, 2, 3], [4, 5, 6]])
d = np.array([[[1, 2, 3], [4, 5, 6]], [[1, 2, 3], [4, 5, 6]]])
print(a.ndim)
print(b.ndim)
print(c.ndim)
print(d.ndim)
arr = np.array([1, 2, 3, 4], ndmin=5)
print(arr)
print('number of dimensions :', arr.ndim)
arr = np.array([1.1, 2.1, 3.1])
newarr = arr.astype('i')
print(newarr)
print(newarr.dtype)
arr = np.array([[1, 2, 3, 4], [5, 6, 7, 8]])
```

```
print(arr.shape)
arr = np.array([1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12])
newarr = arr.reshape(4, 3)
print(newarr)
arr = np.array([1, 2, 3, 4, 5, 6, 7, 8])
print(arr.reshape(2, 4).base)
arr1 = np.array([1, 2, 3])
arr2 = np.array([4, 5, 6])
arr = np.concatenate((arr1, arr2))
print(arr)
arr1 = np.array([[1, 2], [3, 4]])
arr2 = np.array([[5, 6], [7, 8]])
arr = np.concatenate((arr1, arr2), axis=1)
print(arr)
arr1 = np.array([1, 2, 3])
arr2 = np.array([4, 5, 6])
arr = np.stack((arr1, arr2), axis=1)
print(arr)
arr1 = np.array([1, 2, 3])
arr2 = np.array([4, 5, 6])
arr = np.hstack((arr1, arr2))
print(arr)
arr1 = np.array([1, 2, 3]
arr2 = np.array([4, 5, 6])
arr = np.vstack((arr1, arr2))
print(arr)
arr1 = np.array([1, 2, 3])
arr2 = np.array([4, 5, 6])
arr = np.dstack((arr1, arr2))
print(arr)
```

```
arr = np.array([1, 2, 3, 4, 5, 6])
newarr = np.array_split(arr, 3)
print(newarr)
arr = np.array([1, 2, 3, 4, 5, 6])
newarr = np.array_split(arr, 3)
print(newarr[0])
print(newarr[1])
print(newarr[2])
arr = np.array([[1, 2], [3, 4], [5, 6], [7, 8], [9, 10], [11, 12]])
newarr = np.array_split(arr, 3)
print(newarr)
#Split the 2-D array into three 2-D arrays along rows.
arr = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9], [10, 11, 12], [13, 14, 15], [16, 17, 18]])
newarr = np.array_split(arr, 3, axis=1)
print(newarr)
#Use the hsplit() method to split the 2-D array into three 2-D arrays along rows.
arr = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9], [10, 11, 12], [13, 14, 15], [16, 17, 18]])
newarr = np.hsplit(arr, 3)
print(newarr)
arr = np.array([1, 2, 3, 4, 5, 4, 4])
x = np.where(arr == 4)
print(x)
#Find the indexes where the values are even:
arr = np.array([1, 2, 3, 4, 5, 6, 7, 8])
x = np.where(arr%2 == 0)
print(x)
arr = np.array([6, 7, 8, 9])
x = np.searchsorted(arr, 7)
print(x)
arr = np.array([6, 7, 8, 9])
```

```
x = np.searchsorted(arr, 7, side='right')
print(x)
arr = np.array([3, 2, 0, 1])
print(np.sort(arr))
arr = np.array(['banana', 'cherry', 'apple'])
print(np.sort(arr))
arr = np.array([[3, 2, 4], [5, 0, 1]])
print(np.sort(arr))
arr = np.array([41, 42, 43, 44])
x = [True, False, True, False]
newarr = arr[x]
print(newarr)
#Create a filter array that will return only even elements from the original array:
arr = np.array([1, 2, 3, 4, 5, 6, 7])
# Create an empty list
filter_arr = []
# go through each element in arr
for element in arr:
  # if the element is completely divisble by 2, set the value to True, otherwise False
    if element % 2 == 0:
        filter_arr.append(True)
    else:
        filter_arr.append(False)
newarr = arr[filter_arr]
print(filter_arr)
print(newarr)
from numpy import random
arr = np.array([1, 2, 3, 4, 5])
random.shuffle(arr)
print(arr)
```

arr = np.array([1, 2, 3, 4, 5])

print(random.permutation(arr))

import seaborn as sns

import matplotlib.pyplot as plt

sns.histplot([0, 1, 2, 3, 4, 5])

plt.show()

Output:

```
42
[1 2 3 4 5]
[[1 2 3]
 [4 5 6]]
[[[1 2 3]
  [4 5 6]]

 [[1 2 3]
  [4 5 6]]]
0
1
2
3
[[[[[1 2 3 4]]]]]
number of dimensions : 5


[1 2 3]
int32
(2, 4)


[[ 1  2  3]
 [ 4  5  6]
 [ 7  8  9]
 [10 11 12]]
[1 2 3 4 5 6 7 8]


[1 2 3 4 5 6]


[[1 2 5 6]
 [3 4 7 8]]


[[1 4]
 [2 5]
 [3 6]]


[1 2 3 4 5 6]


[[1 2 3]
 [4 5 6]]


[[[1 4]
  [2 5]
  [3 6]]]
```

```
[array([1, 2]), array([3, 4]), array([5, 6])]
```

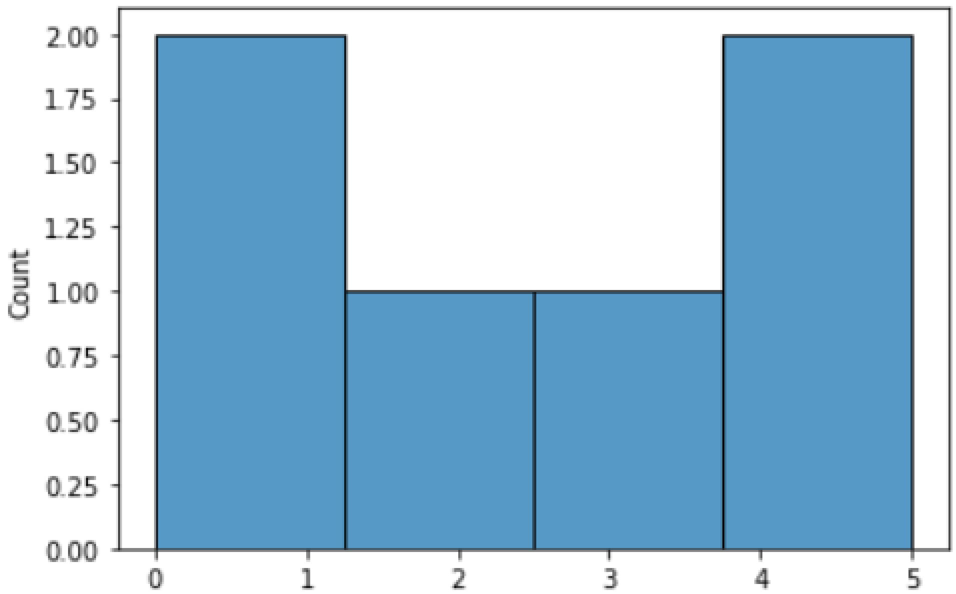
```
[1 2]
[3 4]
[5 6]
[array([[1, 2],
       [3, 4]]), array([[5, 6],
       [7, 8]]), array([[ 9, 10],
       [11, 12]])]
```

```
[array([[ 1],
       [ 4],
       [ 7],
       [10],
       [13],
       [16]]), array([[ 2],
       [ 5],
       [ 8],
       [11],
       [14],
       [17]]), array([[ 3],
       [ 6],
       [ 9],
       [12],
       [15],
       [18]])]
```

```
[array([[ 1],
       [ 4],
       [ 7],
       [10],
       [13],
       [16]]), array([[ 2],
       [ 5],
       [ 8],
       [11],
       [14],
       [17]]), array([[ 3],
       [ 6],
       [ 9],
       [12],
       [15],
       [18]])]
(array([3, 5, 6], dtype=int64),)
(array([1, 3, 5, 7], dtype=int64),)
1
2
[0 1 2 3]
['apple' 'banana' 'cherry']
[[2 3 4]
 [0 1 5]]
```

```
[41 43]
[False, True, False, True, False, True, False]
[2 4 6]
```

```
[4 5 2 1 3]
[3 2 5 1 4]
```



Result: The experiment is successfully completed with the desired output

Date: **Signature:**

Title: PANDAS LIBRARY

Objective: Demonstrate the usage of pandas library

Source Code:

```
import pandas

mydataset = {
  'cars': ["BMW", "Volvo", "Ford"],
  'passings': [3, 7, 2]
}

myvar = pandas.DataFrame(mydataset)

print(myvar)
import pandas as pd

a = [1, 7, 2]

myvar = pd.Series(a)

print(myvar)

print(myvar[0]) # Return the first value of the Series

a = [1, 7, 2]

myvar = pd.Series(a, index = ["x", "y", "z"])

print(myvar)
```

```
print(myvar["y"]) #Return the value of "y"

calories = {"day1": 420, "day2": 380, "day3": 390}
myvar = pd.Series(calories)
print(myvar)
data = {
  "calories": [420, 380, 390],
  "duration": [50, 40, 45]
}
df= pd.DataFrame(data)
print(myvar)
#refer to the row index:
print(df.loc[0])
#refer to the row index:
print(df.loc[0])
#use a list of indexes:
print(df.loc[[0, 1]])

#named indexes
data = {
  "calories": [420, 380, 390],
  "duration": [50, 40, 45]
}

df = pd.DataFrame(data, index = ["day1", "day2", "day3"])
print("\n")
print(df.head())

#refer to the named index:
print(df.loc["day2"])
```

```
#Load a Python Dictionary into a DataFrame:
data = {
  "Duration":{
    "0":60,
    "1":60,
    "2":60,
    "3":45,
    "4":45,
    "5":60
  },
  "Pulse":{
    "0":110,
    "1":117,
    "2":103,
    "3":109,
    "4":117,
    "5":102
  },
  "Maxpulse":{
    "0":130,
    "1":145,
    "2":135,
    "3":175,
    "4":148,
    "5":127
  },
  "Calories":{
    "0":409,
    "1":479,
```

```
    "2":340,
    "3":282,
    "4":406,
    "5":300
  }
}
print("\n")
df = pd.DataFrame(data)
print(df.head())
df = pd.read_csv('data.csv')
df = pd.read_csv('data.csv')
print(df.head())
print("\n")
new_df = df.dropna()
df.dropna(inplace = True)
df.fillna(130, inplace = True)
df["Calories"].fillna(130, inplace = True)
x = df["Calories"].mean()
df["Calories"].fillna(x, inplace = True)
x = df["Calories"].median()
df["Calories"].fillna(x, inplace = True)
print(df.head())
```

Output:

```
    cars  passings
0    BMW         3
1  Volvo         7
2   Ford         2
0    1
1    7
2    2
dtype: int64
1
x    1
y    7
z    2
dtype: int64
7
day1    420
day2    380
day3    390
dtype: int64
day1    420
day2    380
day3    390
dtype: int64
calories    420
duration     50
Name: 0, dtype: int64
calories    420
duration     50
Name: 0, dtype: int64
   calories  duration
0       420        50
1       380        40


      calories  duration
day1       420        50
day2       380        40
day3       390        45
calories    380
duration     40
Name: day2, dtype: int64


   Duration  Pulse  Maxpulse  Calories
0        60    110       130       409
1        60    117       145       479
2        60    103       135       340
3        45    109       175       282
4        45    117       148       406
   Duration  Pulse  Maxpulse  Calories
0        60    110       130     409.1
1        60    117       145     479.0
2        60    103       135     340.0
3        45    109       175     282.4
4        45    117       148     406.0


   Duration  Pulse  Maxpulse  Calories
0        60    110       130     409.1
1        60    117       145     479.0
2        60    103       135     340.0
3        45    109       175     282.4
4        45    117       148     406.0
```

Result: The experiment is successfully completed with the desired output

Date: **Signature:**

Title: **USA gov Data from Bitly:** In 2011, URL shortening service Bitly partnered with the US government website USA.gov to provide a feed of anonymous data gathered from users who shorten links ending with .gov or .mil. In 2011, a live feed as well as hourly snapshots were available as download able text files.

i. Load the data file
ii. Convert a JSON string into a Python dictionary object
iii. Counting Time Zones in Pure Python
iv. Counting Time Zones in Pandas
v. Visualize this data using matplotlib and Seaborn

Source Code:

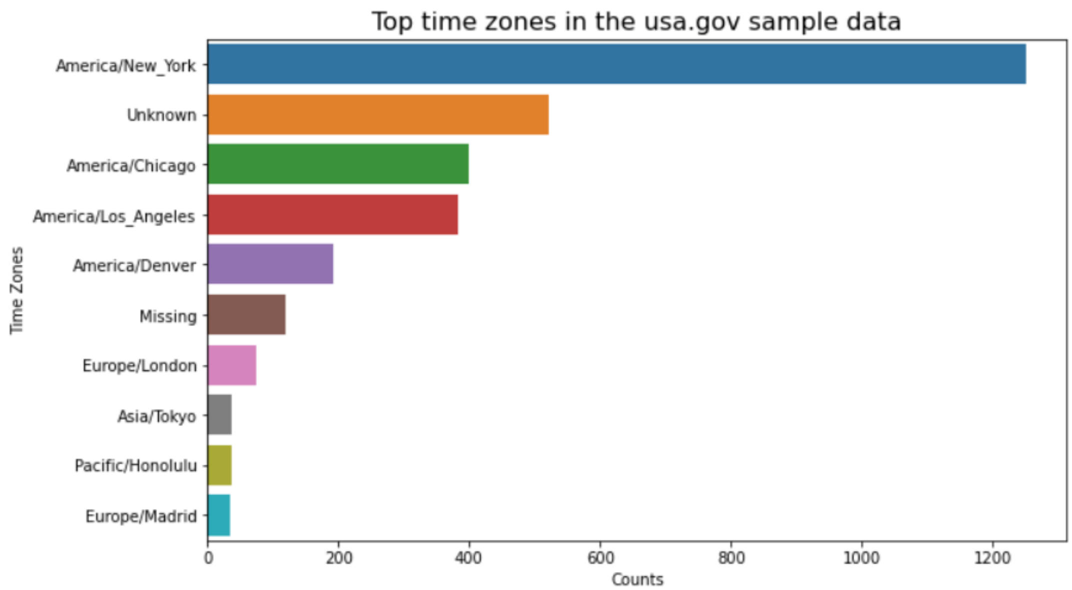
```
from numpy.random import randn
import numpy as np
import seaborn as sns
np.random.seed(123)
import os
import matplotlib.pyplot as plt
import pandas as pd
plt.rc('figure',figsize=(10, 6))
np.set_printoptions(precision=4)
pd.options.display.max_rows = 20

path = 'example.txt'
open(path).readline()

#The database is in json format, so we extract the data following this format.

import json
records = [json.loads(line) for line in open(path)]

records[0]
```

```
data = pd.DataFrame(records)
data.info()

# Time Zones
#First, we are going to analyze the time zones (*tz* in the database).

data['tz'][:10]

tz_counts = data['tz'].value_counts()
tz_counts

#- Cleaning of data

clean_tz = data['tz'].fillna('Missing')
clean_tz[clean_tz == ''] = 'Unknown'
tz_counts = clean_tz.value_counts()
tz_counts[:10]

#- Visualization of data

subset = tz_counts[:10]
sns.barplot(y=subset.index, x=subset.values)
plt.title('Top time zones in the usa.gov sample data', fontsize=16)
plt.xlabel("Counts")
plt.ylabel("Time Zones")
plt.show()
data['a'][:10]
browser = pd.Series([x.split(' ')[0] for x in data['a'].dropna()])
browser.value_counts()[:10]
```

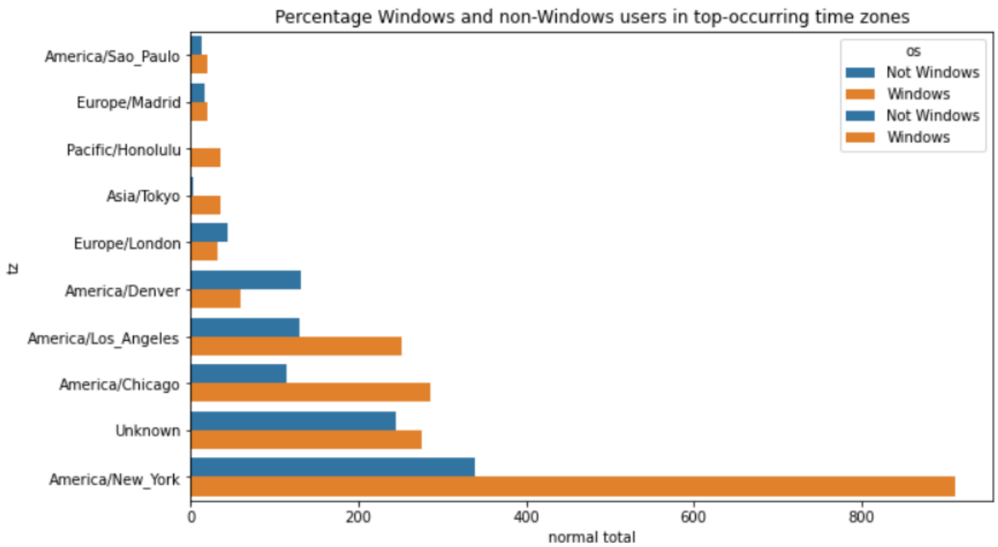
```
clean_data = data[data['a'].notnull()]
clean_data['os'] = np.where(clean_data['a'].str.contains('Windows'), 'Windows', 'Not Windows')
clean_data['os'][:5]

clean_data['tz'] = clean_data['tz'].fillna('Missing')
clean_data['tz'][clean_data['tz'] == ''] = 'Unknown'

#We group time zone and operating system

group_tz_os = clean_data.groupby(['tz', 'os'])
counts_tz_os = group_tz_os.size().unstack().fillna(0)
indexer = counts_tz_os.sum(1).argsort()
subset = counts_tz_os.take(indexer[-10:])
subset = subset.stack()
subset.name = 'total'
subset = subset.reset_index()
sns.barplot(x = 'total', y = 'tz', hue = 'os', data = subset)
plt.title('Top time zones by Windows and non-Windows users')

def normal_total(group):
    group['normal total'] = group.total/group.total.sum()
    return group

subset_normal = subset.groupby('tz').apply(normal_total)
subset_normal

sns.barplot(x = 'normal total', y = 'tz', hue = 'os', data = subset_normal)
plt.title('Percentage Windows and non-Windows users in top-occurring time zones')
```

Output:

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3560 entries, 0 to 3559
Data columns (total 18 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   a            3440 non-null   object
 1   c            2919 non-null   object
 2   nk           3440 non-null   float64
 3   tz           3440 non-null   object
 4   gr           2919 non-null   object
 5   g            3440 non-null   object
 6   h            3440 non-null   object
 7   l            3440 non-null   object
 8   al           3094 non-null   object
 9   hh           3440 non-null   object
 10  r            3440 non-null   object
 11  u            3440 non-null   object
 12  t            3440 non-null   float64
 13  hc           3440 non-null   float64
 14  cy           2919 non-null   object
 15  ll           2919 non-null   object
 16  _heartbeat_  120 non-null    float64
 17  kw           93 non-null     object
dtypes: float64(4), object(14)
memory usage: 500.8+ KB
```



```
Text(0.5, 1.0, 'Percentage Windows and non-Windows users in top-occurring time zones')
```



<u>Result:</u> The experiment is successfully completed with the desired output

Date: **Signature:**

Title: **MovieLens 1M Dataset:** GroupLens Research provides a number of collections of movie ratings data collected from users of MovieLens in the late 1990s and early 2000s. The data provide movie ratings, movie metadata (genres and year), and demographic data about the users (age, zip code, gender identification, and occupation). Such data is often of interest in the development of recommendation systems based on machine learning algorithms. The MovieLens 1M dataset contains 1 million ratings collected from 6,000 users on movie information.

i. Load each table into a pandas data frame object using pandas.read_table
ii. Merge ratings with users and then merge that result with the movies data
iii. Calculate mean movie ratings for each film grouped by gender
iv. Find top films among female viewers
v. Find the movies that are most divisive between male and female viewers

Source Code:

```
import pandas as pd

# Make display smaller

pd.options.display.max_rows = 10

unames = ['user_id', 'gender', 'age', 'occupation', 'zip']

users = pd.read_table('ml-1m/users.dat', sep='::',header=None, names=unames)

print(users.head())

rnames = ['user_id', 'movie_id', 'rating', 'timestamp']

ratings = pd.read_table('ml-1m/ratings.dat', sep='::',header=None, names=rnames)

print(ratings.head())

mnames = ['movie_id', 'title', 'genres']

movies = pd.read_table(r'ml-1m/movies.dat', sep='::',header=None, names=mnames,
encoding="ISO-8859-1")


movies

data = pd.merge(pd.merge(ratings, users), movies)

print(data.head())

mean_ratings = data.pivot_table('rating', index='title',columns='gender', aggfunc='mean')

print(mean_ratings.head())
```

```
ratings_by_title = data.groupby('title').size()
print(ratings_by_title.head())
active_titles = ratings_by_title.index[ratings_by_title >= 250]
print(active_titles)
mean_ratings = mean_ratings.loc[active_titles]
print(mean_ratings.head())
top_female_ratings = mean_ratings.sort_values(by='F', ascending=False)
print(top_female_ratings.head())
mean_ratings['diff'] = mean_ratings['M'] - mean_ratings['F']
print(mean_ratings.head())

sorted_by_diff = mean_ratings.sort_values(by='diff')
print(sorted_by_diff.head())
sorted_by_diff[::-1][:10]
rating_std_by_title = data.groupby('title')['rating'].std()
print(rating_std_by_title)
rating_std_by_title = rating_std_by_title.loc[active_titles]
rating_std_by_title
rating_std_by_title.sort_values(ascending=False)[:10]
```

Output:

```
   user_id gender  age  occupation    zip
0        1      F    1          10  48067
1        2      M   56          16  70072
2        3      M   25          15  55117
3        4      M   45           7  02460
4        5      M   25          20  55455
   user_id  movie_id  rating  timestamp
0        1      1193       5  978300760
1        1       661       3  978302109
2        1       914       3  978301968
3        1      3408       4  978300275
4        1      2355       5  978824291
```

```
   user_id  movie_id  rating  timestamp gender  age  occupation    zip  \
0        1      1193       5  978300760      F    1          10  48067
1        2      1193       5  978298413      M   56          16  70072
2       12      1193       4  978220179      M   25          12  32793
3       15      1193       4  978199279      M   25           7  22903
4       17      1193       5  978158471      M   50           1  95350

                                   title genres
0  One Flew Over the Cuckoo's Nest (1975)  Drama
1  One Flew Over the Cuckoo's Nest (1975)  Drama
2  One Flew Over the Cuckoo's Nest (1975)  Drama
3  One Flew Over the Cuckoo's Nest (1975)  Drama
4  One Flew Over the Cuckoo's Nest (1975)  Drama
gender                                F         M
title
$1,000,000 Duck (1971)         3.375000  2.761905
'Night Mother (1986)           3.388889  3.352941
'Til There Was You (1997)      2.675676  2.733333
'burbs, The (1989)             2.793478  2.962085
...And Justice for All (1979)  3.828571  3.689024
title
$1,000,000 Duck (1971)            37
'Night Mother (1986)              70
'Til There Was You (1997)         52
'burbs, The (1989)               303
...And Justice for All (1979)    199
dtype: int64
Index([''burbs, The (1989)', '10 Things I Hate About You (1999)',
       '101 Dalmatians (1961)', '101 Dalmatians (1996)', '12 Angry Men (1957)',
       '13th Warrior, The (1999)', '2 Days in the Valley (1996)',
       '20,000 Leagues Under the Sea (1954)', '2001: A Space Odyssey (1968)',
       '2010 (1984)',
       ...
       'X-Men (2000)', 'Year of Living Dangerously (1982)',
       'Yellow Submarine (1968)', 'You've Got Mail (1998)',
       'Young Frankenstein (1974)', 'Young Guns (1988)',
       'Young Guns II (1990)', 'Young Sherlock Holmes (1985)',
       'Zero Effect (1998)', 'eXistenZ (1999)'],
      dtype='object', name='title', length=1216)
```

```
gender                                     F         M
title
'burbs, The (1989)                  2.793478  2.962085
10 Things I Hate About You (1999)  3.646552  3.311966
101 Dalmatians (1961)               3.791444  3.500000
101 Dalmatians (1996)               3.240000  2.911215
12 Angry Men (1957)                 4.184397  4.328421
gender                                                     F         M
title
Close Shave, A (1995)                               4.644444  4.473795
Wrong Trousers, The (1993)                          4.588235  4.478261
Sunset Blvd. (a.k.a. Sunset Boulevard) (1950)       4.572650  4.464589
Wallace & Gromit: The Best of Aardman Animation...  4.563107  4.385075
Schindler's List (1993)                             4.562602  4.491415
gender                                     F         M      diff
title
'burbs, The (1989)                  2.793478  2.962085  0.168607
10 Things I Hate About You (1999)  3.646552  3.311966 -0.334586
101 Dalmatians (1961)               3.791444  3.500000 -0.291444
101 Dalmatians (1996)               3.240000  2.911215 -0.328785
12 Angry Men (1957)                 4.184397  4.328421  0.144024
gender                             F         M      diff
title
Dirty Dancing (1987)       3.790378  2.959596 -0.830782
Jumpin' Jack Flash (1986)  3.254717  2.578358 -0.676359
Grease (1978)              3.975265  3.367041 -0.608224
Little Women (1994)        3.870588  3.321739 -0.548849
Steel Magnolias (1989)     3.901734  3.365957 -0.535777
title
$1,000,000 Duck (1971)                       1.092563
'Night Mother (1986)                         1.118636
'Til There Was You (1997)                    1.020159
'burbs, The (1989)                           1.107760
...And Justice for All (1979)                0.878110
                                               ...
Zed & Two Noughts, A (1985)                  1.052794
Zero Effect (1998)                           1.042932
Zero Kelvin (Kjærlighetens kjøtere) (1995)    0.707107
Zeus and Roxanne (1997)                      1.122884
eXistenZ (1999)                              1.178568
Name: rating, Length: 3706, dtype: float64
```

```
title
Dumb & Dumber (1994)                         1.321333
Blair Witch Project, The (1999)              1.316368
Natural Born Killers (1994)                  1.307198
Tank Girl (1995)                             1.277695
Rocky Horror Picture Show, The (1975)        1.260177
Eyes Wide Shut (1999)                        1.259624
Evita (1996)                                 1.253631
Billy Madison (1995)                         1.249970
Fear and Loathing in Las Vegas (1998)        1.246408
Bicentennial Man (1999)                      1.245533
Name: rating, dtype: float64
```

Result: The experiment is successfully completed with the desired output

Date: **Signature:**

Title: **US Baby Names 1880 2010:** The United States Social Security Administration (SSA) has made available data on the frequency of baby names from 1880 to 2010

i. Use Data Wrangling to load this dataset

ii. as the total number of births in that year

iii. Assemble all of the data into a single Data Frame and further add a year field

iv. Visualize total births by sex and year

v. Analyze Naming Trends

Source Code:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
names1880 = pd.read_csv('names/yob1880.txt' ,names=['name', 'sex', 'births'])
print(names1880.head())
print("\n")
print("\nSum")
print(names1880.groupby('sex').births.sum())
years = range(1880, 2011)
pieces = []
columns = ['name', 'sex', 'births']
for year in years:
    path = 'names/yob%d.txt' % year
    frame = pd.read_csv(path, names=columns)
    frame['year'] = year
    pieces.append(frame)
```

```
# Concatenate everything into a single DataFrame
names = pd.concat(pieces, ignore_index=True)
print("\n")
print(names.head())
total_births = names.pivot_table('births', index='year',columns='sex', aggfunc=sum)
print("\n")
print(total_births.head())
total_births = names.pivot_table('births', index='year',columns='sex', aggfunc=sum)
print("\n")
print(total_births.head())
def add_prop(group):
    group['prop'] = group.births / group.births.sum()
    return group
names = names.groupby(['year', 'sex']).apply(add_prop)
print("\n")print(names.head())
names.groupby(['year', 'sex']).prop.sum()
def get_top1000(group):
    return group.sort_values(by='births', ascending=False)[:1000]
grouped = names.groupby(['year', 'sex'])
top1000 = grouped.apply(get_top1000)
# Drop the group index, not needed
top1000.reset_index(inplace=True, drop=True)
pieces = []
for year, group in names.groupby(['year', 'sex']):
    pieces.append(group.sort_values(by='births', ascending=False)[:1000])
    top1000 = pd.concat(pieces, ignore_index=True)
print("\n")
print(top1000.head())
#### Analyzing Naming Trends
boys = top1000[top1000.sex == 'M']
```

```
girls = top1000[top1000.sex == 'F']
total_births = top1000.pivot_table('births', index='year',columns='name',aggfunc=sum)
total_births.info()
subset = total_births[['John', 'Harry', 'Mary', 'Marilyn']]
subset.plot(subplots=True, figsize=(12, 10), grid=False, title="Number of births per year")
plt.show()

#### Measuring the increase in naming diversity
table = top1000.pivot_table('prop', index='year', columns='sex', aggfunc=sum)
table.plot(title='Sum of table1000.prop by year and sex', yticks=np.linspace(0, 1.2, 13),
xticks=range(1880, 2020, 10))
plt.show()
df = boys[boys.year == 2010]
print("\n")
print(df.head())
prop_cumsum = df.sort_values(by='prop', ascending=False).prop.cumsum()
print("\n")
print(prop_cumsum[:10])
prop_cumsum.values.searchsorted(0.5)
df = boys[boys.year == 1900]
in1900 = df.sort_values(by='prop', ascending=False).prop.cumsum()
in1900.values.searchsorted(0.5) + 1
def get_quantile_count(group, q=0.5):
    group = group.sort_values(by='prop', ascending=False)
    return group.prop.cumsum().values.searchsorted(q) + 1
diversity = top1000.groupby(['year', 'sex']).apply(get_quantile_count)
diversity = diversity.unstack('sex')
diversity.head()
diversity.plot(title="Number of popular names in top 50%")
plt.show()
```

```
##### The “last letter” revolution
# extract last letter from name column
get_last_letter = lambda x: x[-1]
last_letters = names.name.map(get_last_letter)
last_letters.name = 'last_letter'
table = names.pivot_table('births', index=last_letters,
columns=['sex', 'year'], aggfunc=sum)
subtable = table.reindex(columns=[1910, 1960, 2010], level='year')
print("\n")
print(subtable.head())
print("\nSum ")
print(subtable.sum())
print("\n")
print(subtable.sum())
letter_prop = subtable / subtable.sum()
print("\n")
print(letter_prop.head())
import matplotlib.pyplot as plt
fig, axes = plt.subplots(2, 1, figsize=(10, 8))
letter_prop['M'].plot(kind='bar', rot=0, ax=axes[0], title='Male')
plt.show()
letter_prop['F'].plot(kind='bar', rot=0, ax=axes[1], title='Female',legend=False)
plt.show()
letter_prop = table / table.sum()
dny_ts = letter_prop.loc[['d', 'n', 'y'], 'M'].T
print("\n")
print(dny_ts.head())
dny_ts.plot()
plt.show()
##### Boy names that became girl names (and vice versa)
```

```
all_names = pd.Series(top1000.name.unique())
lesley_like = all_names[all_names.str.lower().str.contains('lesl')]
print("\n")
print(lesley_like)
filtered = top1000[top1000.name.isin(lesley_like)]
print("\n")
print(filtered)
filtered.groupby('name').births.sum()
table = filtered.pivot_table('births', index='year', columns='sex', aggfunc='sum')
table = table.div(table.sum(1), axis=0)
print("\n")
print(table.tail())
table.plot(style={'M': 'k-', 'F': 'k--'})
plt.show()
```

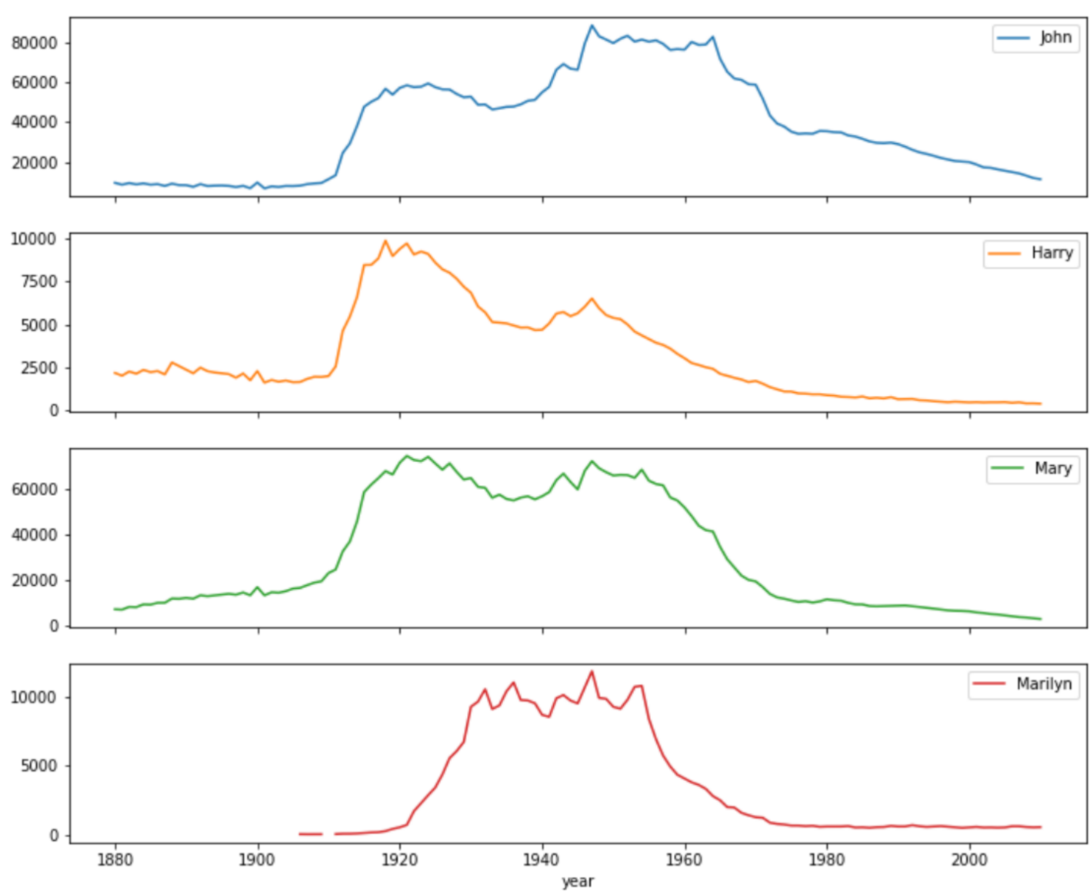
Output:

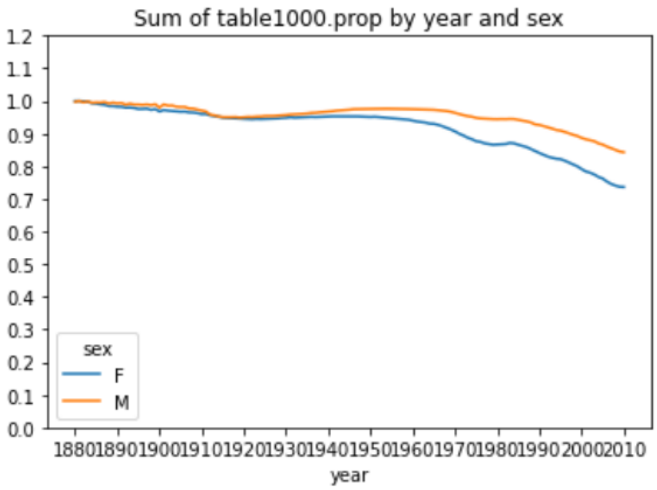
```
         name sex  births
0        Mary   F    7065
1        Anna   F    2604
2        Emma   F    2003
3   Elizabeth   F    1939
4      Minnie   F    1746



Sum
sex
F     90994
M    110490
Name: births, dtype: int64


         name sex  births  year
0        Mary   F    7065  1880
1        Anna   F    2604  1880
2        Emma   F    2003  1880
3   Elizabeth   F    1939  1880
4      Minnie   F    1746  1880


sex         F       M
year
1880    90994  110490
1881    91953  100738
1882   107847  113686
1883   112319  104625
1884   129019  114442
```

```
        name sex  births  year      prop
0       Mary   F    7065  1880  0.077642
1       Anna   F    2604  1880  0.028617
2       Emma   F    2003  1880  0.022012
3  Elizabeth   F    1939  1880  0.021309
4     Minnie   F    1746  1880  0.019188
```

```
        name sex  births  year      prop
0       Mary   F    7065  1880  0.077642
1       Anna   F    2604  1880  0.028617
2       Emma   F    2003  1880  0.022012
3  Elizabeth   F    1939  1880  0.021309
4     Minnie   F    1746  1880  0.019188
<class 'pandas.core.frame.DataFrame'>
Int64Index: 131 entries, 1880 to 2010
Columns: 6873 entries, Aaden to Zuri
dtypes: float64(6873)
memory usage: 6.9 MB
```

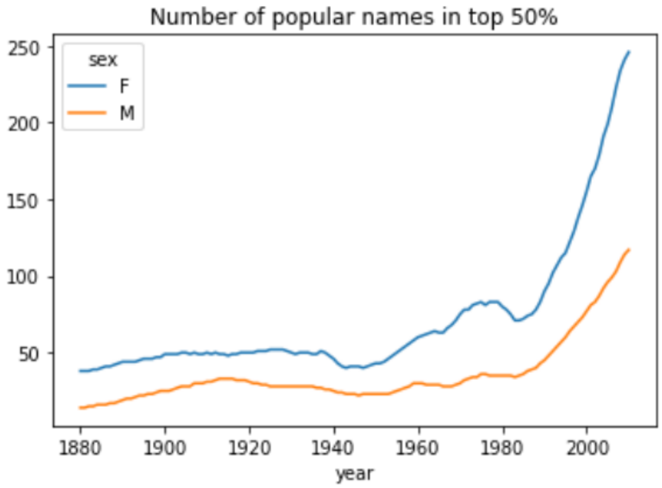
Number of births per year






```
          name sex  births  year      prop
260876    Jacob   M   22139  2010  0.011548
260877    Ethan   M   18006  2010  0.009392
260878  Michael   M   17361  2010  0.009056
260879   Jayden   M   17189  2010  0.008966
260880  William   M   17058  2010  0.008897
```

```
260876    0.011548
260877    0.020940
260878    0.029995
260879    0.038961
260880    0.047858
260881    0.056599
260882    0.065184
260883    0.073451
260884    0.081558
260885    0.089643
Name: prop, dtype: float64
```

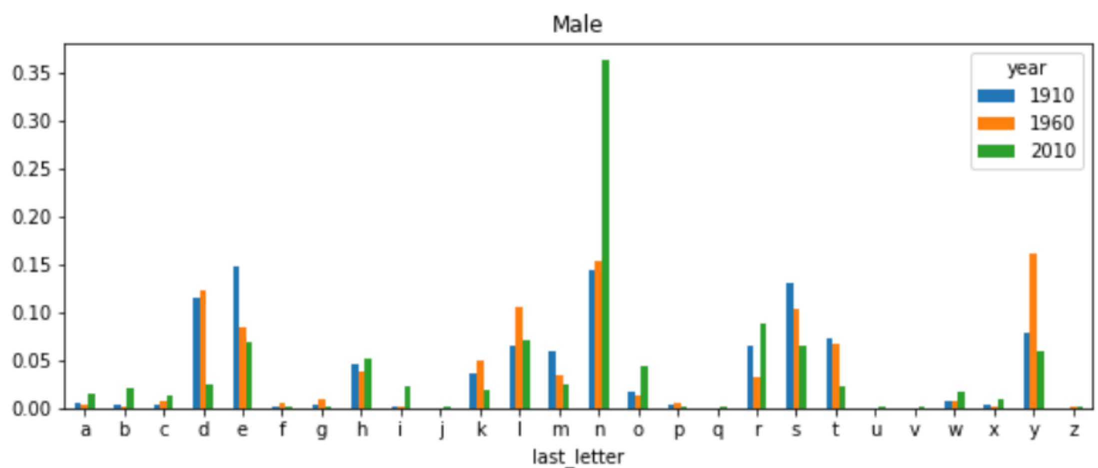


```
sex                 F                          M
year             1910      1960      2010     1910      1960      2010
last_letter
a            108399.0  691278.0  677100.0    977.0    5212.0   28882.0
b                 NaN     694.0     455.0    411.0    3911.0   39294.0
c                 5.0      49.0     957.0    482.0   15457.0   23357.0
d              6751.0    3731.0    2645.0  22113.0  262120.0   44851.0
e            133601.0  435023.0  316878.0  28665.0  178760.0  130307.0
```

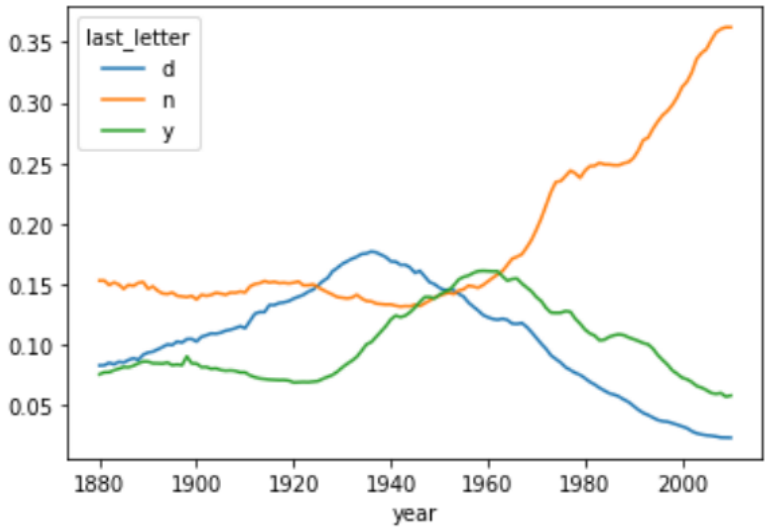
```
Sum
sex  year
F    1910     396505.0
     1960    2022012.0
     2010    1775986.0
M    1910     194210.0
     1960    2132115.0
     2010    1917177.0
dtype: float64
```

```
sex  year
F    1910     396505.0
     1960    2022012.0
     2010    1775986.0
M    1910     194210.0
     1960    2132115.0
     2010    1917177.0
dtype: float64
```

```
sex                 F                              M
year             1910      1960      2010      1910      1960      2010
last_letter
a            0.273386  0.341876  0.381253  0.005031  0.002445  0.015065
b                 NaN  0.000343  0.000256  0.002116  0.001834  0.020496
c            0.000013  0.000024  0.000539  0.002482  0.007250  0.012183
d            0.017026  0.001845  0.001489  0.113861  0.122939  0.023394
e            0.336947  0.215144  0.178424  0.147598  0.083842  0.067968
```



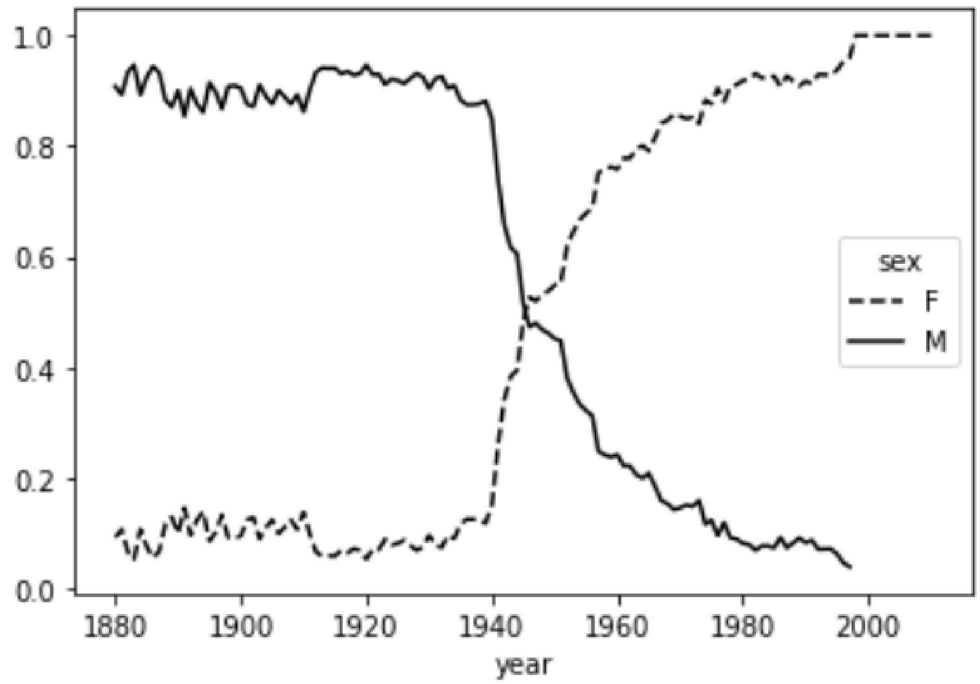
```
last_letter         d         n         y
year
1880         0.083057  0.153217  0.075763
1881         0.083246  0.153219  0.077458
1882         0.085332  0.149561  0.077538
1883         0.084053  0.151656  0.079149
1884         0.086122  0.149927  0.080408
```




```
632      Leslie
2293     Lesley
4265     Leslee
4733      Lesli
6109      Lesly
dtype: object
```

```
          name sex  births  year      prop
632     Leslie   F       8  1880  0.000088
1108    Leslie   M      79  1880  0.000715
2461    Leslie   F      11  1881  0.000120
3014    Leslie   M      92  1881  0.000913
4511    Leslie   F       9  1882  0.000083
...        ...  ..     ...   ...       ...
256326   Lesly   F     699  2008  0.000370
258035  Leslie   F    1982  2009  0.001080
258380   Lesly   F     598  2009  0.000326
260074  Leslie   F    1565  2010  0.000881
260456   Lesly   F     505  2010  0.000284

[400 rows x 5 columns]
```

```
sex       F    M
year
2006  1.0 NaN
2007  1.0 NaN
2008  1.0 NaN
2009  1.0 NaN
2010  1.0 NaN
```



Result: The experiment is successfully completed with the desired output

Date: **Signature:**

<u>Title</u>: USDA Food Database: The US Department of Agriculture makes available a database of food nutrient Information

i. Load dataset into python with any JSON library
ii. Convert a list of dicts to a data frame
iii. Plot median values by food group and nutrient type
iv. Display result Amino Acids' nutrient group

<u>Source Code:</u>

```
import numpy as np
import pandas as pd

fec = pd.read_csv('ALL.csv')

fec.info()
print(fec.head())

fec.iloc[123456]

unique_cands = fec.cand_nm.unique()

print("\n\n")
print(unique_cands)

unique_cands[2]

parties = {'Bachmann, Michelle': 'Republican',
'Cain, Herman': 'Republican',
'Gingrich, Newt': 'Republican',
'Huntsman, Jon': 'Republican',
'Johnson, Gary Earl': 'Republican',
'McCotter, Thaddeus G': 'Republican',
'Obama, Barack': 'Democrat',
'Paul, Ron': 'Republican',
'Pawlenty, Timothy': 'Republican',
'Perry, Rick': 'Republican',
"Roemer, Charles E. 'Buddy' III": 'Republican',
'Romney, Mitt': 'Republican',
'Santorum, Rick': 'Republican'}

fec.cand_nm[123456:123461]

fec.cand_nm[123456:123461].map(parties)

fec['party'] = fec.cand_nm.map(parties)
```

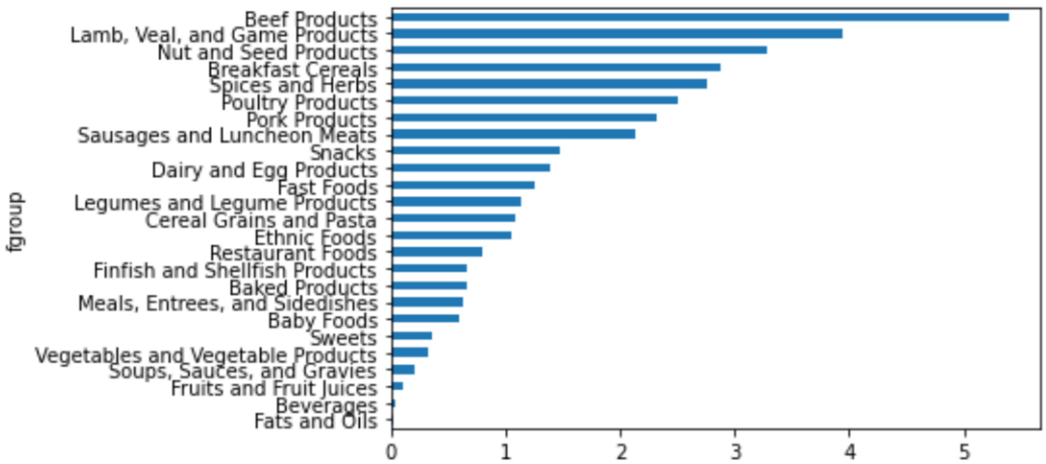
```
fec['party']

fec['party'].value_counts()

(fec.contb_receipt_amt > 0).value_counts()

fec = fec[fec.contb_receipt_amt > 0]

fec_mrbo = fec[fec.cand_nm.isin(['Obama, Barack', 'Romney, Mitt'])]

#### Donation Statistics by Occupation and Employer

fec.contbr_occupation.value_counts()[:10]

occ_mapping = {
'INFORMATION REQUESTED PER BEST EFFORTS' : 'NOT PROVIDED',
'INFORMATION REQUESTED' : 'NOT PROVIDED',
'INFORMATION REQUESTED (BEST EFFORTS)' : 'NOT PROVIDED',
'C.E.O.': 'CEO'
}

# If no mapping provided, return x
f = lambda x: occ_mapping.get(x, x)
fec.contbr_occupation = fec.contbr_occupation.map(f)

emp_mapping = {
'INFORMATION REQUESTED PER BEST EFFORTS' : 'NOT PROVIDED',
'INFORMATION REQUESTED' : 'NOT PROVIDED',
'SELF' : 'SELF-EMPLOYED',
'SELF EMPLOYED' : 'SELF-EMPLOYED',
}

# If no mapping provided, return x
f = lambda x: emp_mapping.get(x, x)
fec.contbr_employer = fec.contbr_employer.map(f)

by_occupation = fec.pivot_table('contb_receipt_amt',index='contbr_occupation',columns='party',
aggfunc='sum')

over_2mm = by_occupation[by_occupation.sum(1) > 2000000]

print("\n\n")
print(over_2mm)

over_2mm.plot(kind='barh')
```

```
def get_top_amounts(group, key, n=5):
    totals = group.groupby(key)['contb_receipt_amt'].sum()
    return totals.nlargest(n)

grouped = fec_mrbo.groupby('cand_nm')

grouped.apply(get_top_amounts, 'contbr_occupation', n=7)

grouped.apply(get_top_amounts, 'contbr_employer', n=10)

#### Bucketing Donation Amounts

bins = np.array([0, 1, 10, 100, 1000, 10000,100000, 1000000, 10000000])

labels = pd.cut(fec_mrbo.contb_receipt_amt, bins)

print(labels)

grouped = fec_mrbo.groupby(['cand_nm', labels])

grouped.size().unstack(0)

bucket_sums = grouped.contb_receipt_amt.sum().unstack(0)

normed_sums = bucket_sums.div(bucket_sums.sum(axis=1), axis=0)

print("\n\n")
print(normed_sums)

normed_sums[:-2].plot(kind='barh')
plt.show()

#### Donation Statistics by State

grouped = fec_mrbo.groupby(['cand_nm', 'contbr_st'])

totals = grouped.contb_receipt_amt.sum().unstack(0).fillna(0)

totals = totals[totals.sum(1) > 100000]

print(totals[:10])

percent = totals.div(totals.sum(1), axis=0)
print("\n\n")
print(percent[:10])
```

Output:

```
6636
dict_keys(['id', 'description', 'tags', 'manufacturer', 'group', 'portions', 'nutrients'])
{'value': 25.18, 'units': 'g', 'description': 'Protein', 'group': 'Composition'}
    value units                  description        group
0   25.18     g                      Protein  Composition
1   29.20     g            Total lipid (fat)  Composition
2    3.06     g  Carbohydrate, by difference  Composition
3    3.28     g                          Ash        Other
4  376.00  kcal                       Energy       Energy
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 6636 entries, 0 to 6635
Data columns (total 4 columns):
 #   Column        Non-Null Count  Dtype
---  ------        --------------  -----
 0   description   6636 non-null   object
 1   group         6636 non-null   object
 2   id            6636 non-null   int64
 3   manufacturer  5195 non-null   object
dtypes: int64(1), object(3)
memory usage: 207.5+ KB
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 6636 entries, 0 to 6635
Data columns (total 4 columns):
 #   Column        Non-Null Count  Dtype
---  ------        --------------  -----
 0   food          6636 non-null   object
 1   fgroup        6636 non-null   object
 2   id            6636 non-null   int64
 3   manufacturer  5195 non-null   object
dtypes: int64(1), object(3)
memory usage: 207.5+ KB
<class 'pandas.core.frame.DataFrame'>
Int64Index: 375176 entries, 0 to 375175
Data columns (total 8 columns):
 #   Column        Non-Null Count   Dtype
---  ------        --------------   -----
 0   value         375176 non-null  float64
 1   units         375176 non-null  object
 2   nutrient      375176 non-null  object
 3   nutgroup      375176 non-null  object
 4   id            375176 non-null  int64
 5   food          375176 non-null  object
 6   fgroup        375176 non-null  object
 7   manufacturer  293054 non-null  object
dtypes: float64(1), int64(1), object(6)
memory usage: 25.8+ MB
```

```
nutrient
Alanine                            Gelatins, dry powder, unsweetened
Arginine                                Seeds, sesame flour, low-fat
Aspartic acid                                    Soy protein isolate
Cystine                 Seeds, cottonseed flour, low fat (glandless)
Glutamic acid                                    Soy protein isolate
Glycine                            Gelatins, dry powder, unsweetened
Histidine                 Whale, beluga, meat, dried (Alaska Native)
Hydroxyproline    KENTUCKY FRIED CHICKEN, Fried Chicken, ORIGINA...
Isoleucine        Soy protein isolate, PROTEIN TECHNOLOGIES INTE...
Leucine           Soy protein isolate, PROTEIN TECHNOLOGIES INTE...
Lysine            Seal, bearded (Oogruk), meat, dried (Alaska Na...
Methionine                     Fish, cod, Atlantic, dried and salted
Phenylalanine     Soy protein isolate, PROTEIN TECHNOLOGIES INTE...
Proline                            Gelatins, dry powder, unsweetened
Serine            Soy protein isolate, PROTEIN TECHNOLOGIES INTE...
Threonine         Soy protein isolate, PROTEIN TECHNOLOGIES INTE...
Tryptophan         Sea lion, Steller, meat with fat (Alaska Native)
Tyrosine          Soy protein isolate, PROTEIN TECHNOLOGIES INTE...
Valine            Soy protein isolate, PROTEIN TECHNOLOGIES INTE...
Name: food, dtype: object
```



Result: The experiment is successfully completed with the desired output

Date: **Signature:**

<u>Title</u>: 2012 Federal Election Commission Database: The US Federal Election Commission publishes data on contributions to political campaigns. This includes contributor names, occupation and employer, address, and contribution amount. An interesting dataset is from the 2012 US presidential election

i. Load CSV file and convert into data frame

ii. Compute an array of political parties from the candidate names

iii. Analyze donation statistics by occupation and employer

iv. Use pivot_table to aggregate the data by party and occupation

v. Plot total donations by party for top occupations

vi. Bucketing donation amounts

vii. Plot Percentage of total donations received by candidates for each donation size

viii. Analyze donation statistics by state

<u>Source Code:</u>

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
fec = pd.read_csv('ALL.csv')

fec.info()

print(fec.head())

fec.iloc[123456]

unique_cands = fec.cand_nm.unique()
```

```
print("\n\n")
print(unique_cands)

unique_cands[2]

parties = {'Bachmann, Michelle': 'Republican',
'Cain, Herman': 'Republican',
'Gingrich, Newt': 'Republican',
'Huntsman, Jon': 'Republican',
'Johnson, Gary Earl': 'Republican',
'McCotter, Thaddeus G': 'Republican',
'Obama, Barack': 'Democrat',
'Paul, Ron': 'Republican',
'Pawlenty, Timothy': 'Republican',
'Perry, Rick': 'Republican',
"Roemer, Charles E. 'Buddy' III": 'Republican',
'Romney, Mitt': 'Republican',
'Santorum, Rick': 'Republican'}

fec.cand_nm[123456:123461]

fec.cand_nm[123456:123461].map(parties)

fec['party'] = fec.cand_nm.map(parties)

fec['party']

fec['party'].value_counts()
```

```
(fec.contb_receipt_amt > 0).value_counts()

fec = fec[fec.contb_receipt_amt > 0]

fec_mrbo = fec[fec.cand_nm.isin(['Obama, Barack', 'Romney, Mitt'])]

#### Donation Statistics by Occupation and Employer

fec.contbr_occupation.value_counts()[:10]

occ_mapping = {
'INFORMATION REQUESTED PER BEST EFFORTS' : 'NOT PROVIDED',
'INFORMATION REQUESTED' : 'NOT PROVIDED',
'INFORMATION REQUESTED (BEST EFFORTS)' : 'NOT PROVIDED',
'C.E.O.': 'CEO'
}

# If no mapping provided, return x
f = lambda x: occ_mapping.get(x, x)
fec.contbr_occupation = fec.contbr_occupation.map(f)

emp_mapping = {
'INFORMATION REQUESTED PER BEST EFFORTS' : 'NOT PROVIDED',
'INFORMATION REQUESTED' : 'NOT PROVIDED',
'SELF' : 'SELF-EMPLOYED',
'SELF EMPLOYED' : 'SELF-EMPLOYED',
}

# If no mapping provided, return x
f = lambda x: emp_mapping.get(x, x)
```

```
fec.contbr_employer = fec.contbr_employer.map(f)

by_occupation = fec.pivot_table('contb_receipt_amt',index='contbr_occupation',columns='party',
aggfunc='sum')

over_2mm = by_occupation[by_occupation.sum(1) > 2000000]



print("\n\n")
print(over_2mm)

over_2mm.plot(kind='barh')

def get_top_amounts(group, key, n=5):
    totals = group.groupby(key)['contb_receipt_amt'].sum()
    return totals.nlargest(n)

grouped = fec_mrbo.groupby('cand_nm')

grouped.apply(get_top_amounts, 'contbr_occupation', n=7)

grouped.apply(get_top_amounts, 'contbr_employer', n=10)

#### Bucketing Donation Amounts

bins = np.array([0, 1, 10, 100, 1000, 10000,100000, 1000000, 10000000])

labels = pd.cut(fec_mrbo.contb_receipt_amt, bins)

print("\n\n")
```

```
print(labels)

grouped = fec_mrbo.groupby(['cand_nm', labels])

grouped.size().unstack(0)

bucket_sums = grouped.contb_receipt_amt.sum().unstack(0)

normed_sums = bucket_sums.div(bucket_sums.sum(axis=1), axis=0)

print("\n\n")
print(normed_sums)

normed_sums[:-2].plot(kind='barh')
plt.show()

#### Donation Statistics by State

grouped = fec_mrbo.groupby(['cand_nm', 'contbr_st'])

totals = grouped.contb_receipt_amt.sum().unstack(0).fillna(0)

totals = totals[totals.sum(1) > 100000]

print(totals[:10])

percent = totals.div(totals.sum(1), axis=0)
print("\n\n")
print(percent[:10])
```

<u>Output:</u>

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1001731 entries, 0 to 1001730
Data columns (total 16 columns):
 #   Column             Non-Null Count    Dtype  
---  ------             --------------    -----  
 0   cmte_id            1001731 non-null  object 
 1   cand_id            1001731 non-null  object 
 2   cand_nm            1001731 non-null  object 
 3   contbr_nm          1001731 non-null  object 
 4   contbr_city        1001712 non-null  object 
 5   contbr_st          1001727 non-null  object 
 6   contbr_zip         1001620 non-null  object 
 7   contbr_employer    988002 non-null   object 
 8   contbr_occupation  993301 non-null   object 
 9   contb_receipt_amt  1001731 non-null  float64
 10  contb_receipt_dt   1001731 non-null  object 
 11  receipt_desc       14166 non-null    object 
 12  memo_cd            92482 non-null    object 
 13  memo_text          97770 non-null    object 
 14  form_tp            1001731 non-null  object 
 15  file_num           1001731 non-null  int64  
dtypes: float64(1), int64(1), object(14)
memory usage: 122.3+ MB
     cmte_id    cand_id             cand_nm           contbr_nm  \
0  C00410118  P20002978  Bachmann, Michelle     HARVEY, WILLIAM   
1  C00410118  P20002978  Bachmann, Michelle     HARVEY, WILLIAM   
2  C00410118  P20002978  Bachmann, Michelle       SMITH, LANIER   
3  C00410118  P20002978  Bachmann, Michelle    BLEVINS, DARONDA   
4  C00410118  P20002978  Bachmann, Michelle  WARDENBURG, HAROLD   

          contbr_city contbr_st   contbr_zip        contbr_employer  \
0              MOBILE        AL  366010290.0                RETIRED   
1              MOBILE        AL  366010290.0                RETIRED   
2              LANETT        AL  368633403.0  INFORMATION REQUESTED   
3             PIGGOTT        AR  724548253.0                   NONE   
4  HOT SPRINGS NATION        AR  719016467.0                   NONE   

       contbr_occupation  contb_receipt_amt contb_receipt_dt receipt_desc  \
0                RETIRED              250.0        20-JUN-11          NaN   
1                RETIRED               50.0        23-JUN-11          NaN   
2  INFORMATION REQUESTED              250.0        05-JUL-11          NaN   
3                RETIRED              250.0        01-AUG-11          NaN   
4                RETIRED              300.0        20-JUN-11          NaN   

  memo_cd memo_text form_tp  file_num  
0     NaN       NaN   SA17A    736166  
1     NaN       NaN   SA17A    736166  
2     NaN       NaN   SA17A    749073  
3     NaN       NaN   SA17A    749073  
4     NaN       NaN   SA17A    736166  
```

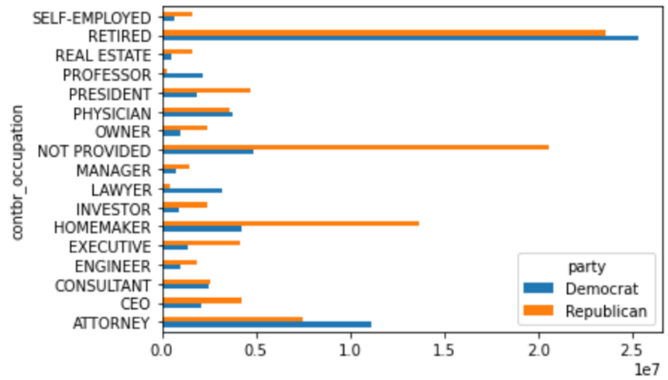
Date: **Signature:**

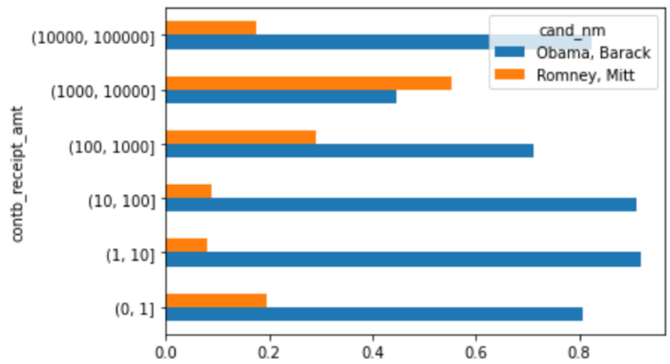
```
['Bachmann, Michelle' 'Romney, Mitt' 'Obama, Barack'
 "Roemer, Charles E. 'Buddy' III" 'Pawlenty, Timothy' 'Johnson, Gary Earl'
 'Paul, Ron' 'Santorum, Rick' 'Cain, Herman' 'Gingrich, Newt'
 'McCotter, Thaddeus G' 'Huntsman, Jon' 'Perry, Rick']
```

```
party                 Democrat   Republican
contbr_occupation
ATTORNEY           11141982.97   7477194.43
CEO                 2074974.79   4211040.52
CONSULTANT          2459912.71   2544725.45
ENGINEER             951525.55   1818373.70
EXECUTIVE           1355161.05   4138850.09
HOMEMAKER           4248875.80  13634275.78
INVESTOR             884133.00   2431768.92
LAWYER              3160478.87    391224.32
MANAGER              762883.22   1444532.37
NOT PROVIDED        4866973.96  20565473.01
OWNER               1001567.36   2408286.92
PHYSICIAN           3735124.94   3594320.24
PRESIDENT           1878509.95   4720923.76
PROFESSOR           2165071.08    296702.73
REAL ESTATE          528902.09   1625902.25
RETIRED            25305116.38  23561244.49
SELF-EMPLOYED        672393.40   1640252.54
```

```
411         (10, 100]
412       (100, 1000]
413       (100, 1000]
414         (10, 100]
415         (10, 100]
             ...
701381      (10, 100]
701382    (100, 1000]
701383        (1, 10]
701384      (10, 100]
701385    (100, 1000]
Name: contb_receipt_amt, Length: 694282, dtype: category
Categories (8, interval[int64, right]): [(0, 1] < (1, 10] < (10, 100] < (100, 1000] < (1000, 10000] < (10000, 100000] < (10000
0, 1000000] < (1000000, 10000000]]
```

```
cand_nm              Obama, Barack  Romney, Mitt
contb_receipt_amt
(0, 1]                    0.805182      0.194818
(1, 10]                   0.918767      0.081233
(10, 100]                 0.910769      0.089231
(100, 1000]               0.710176      0.289824
(1000, 10000]             0.447326      0.552674
(10000, 100000]           0.823120      0.176880
(100000, 1000000]         1.000000      0.000000
(1000000, 10000000]       1.000000      0.000000
```






```
cand_nm     Obama, Barack  Romney, Mitt
contbr_st
AK              281840.15      86204.24
AL              543123.48     527303.51
AR              359247.28     105556.00
AZ             1506476.98    1888436.23
CA            23824984.24   11237636.60
CO             2132429.49    1506714.12
CT             2068291.26    3499475.45
DC             4373538.80    1025137.50
DE              336669.14      82712.00
FL             7318178.58    8338458.81
```

```
cand_nm     Obama, Barack  Romney, Mitt
contbr_st
AK               0.765778      0.234222
AL               0.507390      0.492610
AR               0.772902      0.227098
AZ               0.443745      0.556255
CA               0.679498      0.320502
CO               0.585970      0.414030
CT               0.371476      0.628524
DC               0.810113      0.189887
DE               0.802776      0.197224
FL               0.467417      0.532583
```

Result: The experiment is successfully completed with the desired output

Date: **Signature:**