```python
# -*- coding: utf-8 -*-
"""
Created on Fri May 20 18:07:23 2022

@author: carlj
"""

import pandas as pd

"""
Pandas Lecture 4: Creating Pandas Series
"""
# series
# with series, you can assign index labels
groceries = pd.Series(data = [30, 6, 'Yes', 'No'],
                      index = ['eggs', 'apples', 'milk', 'bread'])
"""
eggs       30
apples      6
milk      Yes
bread      No
dtype: object
"""
groceries.shape
# (4,)
groceries.ndim
# 1
groceries.size
# 4

groceries.index
# Index(['eggs', 'apples', 'milk', 'bread'], dtype='object')

groceries.values
# array([30, 6, 'Yes', 'No'], dtype=object)

# can check index labels like the following
'eggs' in groceries # True
'fruit' in groceries # False

"""
Pandas Lecture 5: Accessing and Deleting Elements in Pandas Series
"""
groceries['eggs']
# 30
groceries[['eggs', 'bread']]
"""
eggs     30
bread    No
dtype: object
"""
groceries[0]
# 30
# can grab multiple entries using list of indices

# loc and iloc
# loc: location (labeled index)
```

```python
# iloc: integer location (numerical index)

# changing elements
groceries['eggs'] = 2

# deleting elements
groceries.drop('apples') # this will not modify the original series
groceries.drop('apples', inplace = True) # this will modify the original series

"""
Pandas Lecture 6: Arithmetic Operations on Pandas Series
"""
fruits = pd.Series([10, 6, 3], ['apples', 'oranges', 'bananas'])

# can use mathematical operations (normal arithmetic) or mathematic functions from NumPy
import numpy as np

fruits * 2
np.sqrt(fruits)
fruits[bananas] + 2
fruits.iloc[0] - 2
fruits[['apples', 'oranges']] * 2

# can use arthimetic operations on a Pandas series of mixed datapoints...
# recall groceries
groceries = pd.Series([30, 6, 'Yes', 'No'], ['eggs', 'apples', 'milk', 'bread'])
groceries * 2
"""
eggs          60
apples        12
milk      YesYes
bread       NoNo
"""
# for instance, multiplication is defined by strings but division is not

"""
Pandas Lecture 7: Manipulate a Series (challenge)
"""

# DO NOT CHANGE THE VARIABLE NAMES

# Given a list representing a few planets
planets = ['Earth', 'Saturn', 'Venus', 'Mars', 'Jupiter']

# Given another list representing the distance of each of these planets from the Sun
# The distance from the Sun is in units of 10^6 km
distance_from_sun = [149.6, 1433.5, 108.2, 227.9, 778.6]


# TO DO: Create a Pandas Series "dist_planets" using the lists above,
# representing the distance of the planet from the Sun.
# Use the `distance_from_sun` as your data, and `planets` as your index.
dist_planets = pd.Series(distance_from_sun, planets)


# TO DO: Calculate the time (minutes) it takes light from the Sun to reach each planet.
# You can do this by dividing each planet's distance from the Sun by the speed of light.
```

```
# Use the speed of light, c = 18, since light travels 18 x 10^6 km/minute.
time_light = dist_planets / 18


# TO DO: Use Boolean indexing to select only those planets for which sunlight takes less
# than 40 minutes to reach them.
# We'll check your work by printing out these close planets.
close_planets = time_light[time_light < 40]

"""
Earth     8.311111
Venus     6.011111
Mars     12.661111
dtype: float64
"""

"""
Pandas Lecture 8: Creating Pandas DataFrames
"""
# think of dataframe as really powerful spreadsheet
items = {'Bob' : pd.Series(data = [245, 25, 55],
        index = ['bike', 'pants', 'watch']),
        'Alice' : pd.Series(data = [40, 110, 500, 45],
        index = ['book', 'glasses', 'bike', 'pants'])}

print(type(items))
# <class 'dict'>

shopping_carts = pd.DataFrame(items)
shopping_carts
"""
           Bob  Alice
bike     245.0  500.0
book       NaN   40.0
glasses    NaN  110.0
pants     25.0   45.0
watch     55.0    NaN
"""

# Let's see what this looks like without index labels
items = {'Bob' : pd.Series([245, 25, 55]),
         'Alice' : pd.Series([40, 110, 500, 45])}

shopping_carts = pd.DataFrame(items)
shopping_carts

"""
     Bob  Alice
0  245.0     40
1   25.0    110
2   55.0    500
3    NaN     45
"""

# can use shape (n, m), ndim (n), size (n x m)

# DataFrame Subsets
```

```
items = {'Bob' : pd.Series(data = [245, 25, 55],
        index = ['bike', 'pants', 'watch']),
        'Alice' : pd.Series(data = [40, 110, 500, 45],
        index = ['book', 'glasses', 'bike', 'pants'])}
shopping_carts = pd.DataFrame(items)

bob_shopping_cart = pd.DataFrame(items, columns = ['Bob'])
bob_shopping_cart

"""
       Bob
bike   245
pants   25
watch   55
"""

ex_cart = pd.DataFrame(items, index = ['pants', 'book'])
ex_cart

"""
        Bob  Alice
pants  25.0     45
book    NaN     40
"""

ex_cart = pd.DataFrame(items, index = ['glasses', 'bike'], columns = ['Alice'])
ex_cart

"""
         Alice
glasses    110
bike       500
"""

# we can alter DataFrame after the fact
data = {'Integers': [1, 2, 3], 'Floats': [4.5, 8.2, 9.6]}
df = pd.DataFrame(data, index = ['label 1', 'label 2', 'label 3'])
df

"""
         Integers  Floats
label 1         1     4.5
label 2         2     8.2
label 3         3     9.6
"""

# creating a DataFrame using a list of dictionaries
# We create a list of Python dictionaries
items2 = [{'bikes': 20, 'pants': 30, 'watches': 35},
          {'watches': 10, 'glasses': 50, 'bikes': 15, 'pants':5}]

# We create a DataFrame  and provide the row index
store_items = pd.DataFrame(items2, index = ['store 1', 'store 2'])

# We display the DataFrame
store_items
```

```
"""
         bikes  pants  watches  glasses
store 1     20     30       35      NaN
store 2     15      5       10     50.0
"""

"""
Pandas Lecture 9: Accessing Elements in Pandas DataFrames
"""

# can access rows, columns and elements
# columns first, then rows for accessing elements
store_items[['bikes', 'pants']]
'''
         bikes  pants
store 1     20     30
store 2     15      5
'''

# recall loc is rows & columns by name, and iloc is by integer index/position
store_items.loc[['store 1']]
'''
         bikes  pants  watches  glasses
store 1     20     30       35      NaN
'''

store_items['bikes']['store 2']
# 15

# add items
store_items['shirts'] = [15, 2]
print(store_items)

'''
         bikes  pants  watches  glasses  shirts
store 1     20     30       35      NaN      15
store 2     15      5       10     50.0       2
'''

store_items['suits'] = store_items['shirts'] + store_items['pants']
print(store_items)
'''
         bikes  pants  watches  glasses  shirts  suits
store 1     20     30       35      NaN      15     45
store 2     15      5       10     50.0       2      7
'''

new_items = [{'bikes': 20, 'pants': 30, 'watches': 35, 'glasses': 4}]
new_store = pd.DataFrame(new_items, index = ['store 3'])
new_store

# append new store
store_items = store_items.append(new_store)

# lets stock new watches in stores 2 and 3 with same prices as old
store_items['new_watches'] = store_items['watches'][1:]
print(store_items)
```

```
'''
         bikes  pants  watches  glasses  shirts  suits  new_watches
store 1     20     30       35      NaN    15.0   45.0          NaN
store 2     15      5       10     50.0     2.0    7.0         10.0
store 3     20     30       35      4.0     NaN    NaN         35.0
'''

# insert formula
store_items.insert(5, 'shoes', [8, 5, 0])
store_items

'''
         bikes  pants  watches  glasses  shirts  shoes  suits  new_watches
store 1     20     30       35      NaN    15.0      8   45.0          NaN
store 2     15      5       10     50.0     2.0      5    7.0         10.0
store 3     20     30       35      4.0     NaN      0    NaN         35.0
'''

# pop and drop methods to delete columns, rows
# pop to remove columns
# drop to remove columns (axis = 1) and rows (axis = 0)

store_items.pop('new_watches')

store_items = store_items.drop(['watches', 'shoes'], axis=1)
store_items = store_items.drop(['store 1', 'store 2'], axis=0)

store_items
'''
         bikes  pants  glasses  shirts  suits
store 3     20     30      4.0     NaN    NaN
'''

# changes column names
store_items = store_items.rename(columns = {'bikes': 'hats'})
store_items
'''
         hats  pants  glasses  shirts  suits
store 3    20     30      4.0     NaN    NaN
'''

# changes row names
store_items = store_items.rename(index = {'store 3': 'last store'})
store_items

'''
            hats  pants  glasses  shirts  suits
last store    20     30      4.0     NaN    NaN
'''

"""
Pandas Lecture 10: Dealing with NaN
"""

# time to clean some data, most common error is missing values
```

```
# We create a list of Python dictionaries
items2 = [{'bikes': 20, 'pants': 30, 'watches': 35, 'shirts': 15, 'shoes':8, 'suits':45},
{'watches': 10, 'glasses': 50, 'bikes': 15, 'pants':5, 'shirts': 2, 'shoes':5, 'suits':7},
{'bikes': 20, 'pants': 30, 'watches': 35, 'glasses': 4, 'shoes':10}]

# We create a DataFrame  and provide the row index
store_items = pd.DataFrame(items2, index = ['store 1', 'store 2', 'store 3'])

# We display the DataFrame
store_items

'''
         bikes  pants  watches  shirts  shoes  suits  glasses
store 1     20     30       35    15.0      8   45.0      NaN
store 2     15      5       10     2.0      5    7.0     50.0
store 3     20     30       35     NaN     10    NaN      4.0
'''

# x = store_items.isnull().sum().sum()
x = store_items.isnull()
'''
         bikes  pants  watches  shirts  shoes  suits  glasses
store 1  False  False    False   False  False  False     True
store 2  False  False    False   False  False  False    False
store 3  False  False    False    True  False   True    False
'''
x = store_items.isnull().sum()
'''
bikes      0
pants      0
watches    0
shirts     1
shoes      0
suits      1
glasses    1
dtype: int64
'''
x = store_items.isnull().sum().sum()
# 3

# OR x = store_items.count()
x = store_items.count()
'''
bikes      3
pants      3
watches    3
shirts     2
shoes      3
suits      2
glasses    2
dtype: int64
'''

# We can drop both entire rows and columns that contain NaN values
# drop rows (axis = 0), drop columns (axis = 1)
store_items.dropna(axis=0)
'''
```

```
         bikes  pants  watches  shirts  shoes  suits  glasses
store 2     15      5       10     2.0      5    7.0     50.0
'''

store_items.dropna(axis=1)
'''
         bikes  pants  watches  shoes
store 1     20     30       35      8
store 2     15      5       10      5
store 3     20     30       35     10
'''

# the above examples do not modify the original dataframe, using inplace would
# i.e. store_items.dropna(axis=1, inplace = True)

# instead of eliminating, let's replace
# fill with 0s
store_items.fillna(0)

# forwarding filling: replaces each NaN with the value prior to NaN
# forward filling won't affect any elements in first row / column
# switch row and column using axis specification
# backward filling won't affect any elements in last row / column

store_items.fillna(method = 'ffill', axis = 0)
'''
         bikes  pants  watches  shirts  shoes  suits  glasses
store 1     20     30       35    15.0      8   45.0      NaN
store 2     15      5       10     2.0      5    7.0     50.0
store 3     20     30       35     2.0     10    7.0      4.0
'''
store_items.fillna(method = 'ffill', axis = 1)
'''
         bikes  pants  watches  shirts  shoes  suits  glasses
store 1   20.0   30.0     35.0    15.0    8.0   45.0     45.0
store 2   15.0    5.0     10.0     2.0    5.0    7.0     50.0
store 3   20.0   30.0     35.0    35.0   10.0   10.0      4.0
'''

store_items.fillna(method = 'backfill', axis = 0)
'''
         bikes  pants  watches  shirts  shoes  suits  glasses
store 1     20     30       35    15.0      8   45.0     50.0
store 2     15      5       10     2.0      5    7.0     50.0
store 3     20     30       35     NaN     10    NaN      4.0
'''

store_items.fillna(method = 'backfill', axis = 1)
'''
         bikes  pants  watches  shirts  shoes  suits  glasses
store 1   20.0   30.0     35.0    15.0    8.0   45.0      NaN
store 2   15.0    5.0     10.0     2.0    5.0    7.0     50.0
store 3   20.0   30.0     35.0    10.0   10.0    4.0      4.0
'''

# can fill NaN values using interpolations...
store_items.interpolate(method = 'linear', axis = 1)
```

```
'''
         bikes  pants  watches  shirts  shoes  suits  glasses
store 1   20.0   30.0     35.0    15.0    8.0   45.0     45.0
store 2   15.0    5.0     10.0     2.0    5.0    7.0     50.0
store 3   20.0   30.0     35.0    22.5   10.0    7.0      4.0
'''


"""
Pandas Lecture 11: Manipulating a DataFrame (Excersize)
"""

# DO NOT CHANGE THE VARIABLE NAMES

# Set the precision of our dataframes to one decimal place.
pd.set_option('precision', 1)

# Create a Pandas DataFrame that contains the ratings some users have given to a series of books.
# The ratings given are in the range from 1 to 5, with 5 being the best score.
# The names of the books, the corresponding authors, and the ratings of each user are given below:

books = pd.Series(data = ['Great Expectations', 'Of Mice and Men', 'Romeo and Juliet',
                          'The Time Machine', 'Alice in Wonderland' ])
authors = pd.Series(data = ['Charles Dickens', 'John Steinbeck', 'William Shakespeare',
                        ' H. G. Wells', 'Lewis Carroll' ])

# User ratings are in the order of the book titles mentioned above
# If a user has not rated all books, Pandas will automatically consider the missing values as NaN.
# If a user has mentioned `np.nan` value, then also it means that the user has not yet rated that b
user_1 = pd.Series(data = [3.2, np.nan ,2.5])
user_2 = pd.Series(data = [5., 1.3, 4.0, 3.8])
user_3 = pd.Series(data = [2.0, 2.3, np.nan, 4])
user_4 = pd.Series(data = [4, 3.5, 4, 5, 4.2])


# Use the data above to create a Pandas DataFrame that has the following column
# labels: 'Author', 'Book Title', 'User 1', 'User 2', 'User 3', 'User 4'.
# Let Pandas automatically assign numerical row indices to the DataFrame.

# TO DO: Create a dictionary with the data given above
dat = {'Author': authors, 'Book Title': books, 'User 1': user_1,
       'User 2': user_2, 'User 3': user_3, 'User 4': user_4}

# TO DO: Create a Pandas DataFrame using the dictionary created above
book_ratings = pd.DataFrame(dat)

# TO DO:
# If you created the dictionary correctly you should have a Pandas DataFrame
# that has column labels:
# 'Author', 'Book Title', 'User 1', 'User 2', 'User 3', 'User 4'
# and row indices 0 through 4.

# Now replace all the NaN values in your DataFrame with the average rating in
# each column. Replace the NaN values in place.
# HINT: Use the `pandas.DataFrame.fillna(value, inplace = True)` function for
# substituting the NaN values.
# Write your code below:
```

```
book_ratings.fillna(book_ratings.mean(), inplace = True)


"""
Pandas Lecture 12: Loading Data into a Pandas DataFrame
"""
# loading a csv file
example = pd.read_csv('file.csv')
# check for NaN values
example.isnull().any()
# this will return a list of the columns, and either True or False
example['Column X'].describe()
# will return:
# count, mean, std, min, 25%, 50%, 75%, max, name

# can apply .max(), min(), .mean(), etc. on entire set and will return stats by column

# .gropuby() method
# ex/ data.groupby(['Year'])['Salary'].sum()
# will give the sum of the salaries
# ex/ data.groupby(['Year', 'Department'])['Salary'].sum()

"""
Pandas Lecture 15: Mini-Project: Statistics From Stock Data
"""

aapl_path = 'C:/Users/carlj/OneDrive/Documents/Continued Education/Data Science/Udacity/Electives/F
amzn_path = 'C:/Users/carlj/OneDrive/Documents/Continued Education/Data Science/Udacity/Electives/F
goog_path = 'C:/Users/carlj/OneDrive/Documents/Continued Education/Data Science/Udacity/Electives/F

import pandas as pd

'''
You will now load the stock data from Google, Apple, and Amazon into separte DataFrames.
However, for each stock data you will only be interested in loading the Date and Adj Close columns
into the Dataframe. In addtion, you want to use the Date column as your row index.
Finally, you want the DataFrame to recognize the dates as actual dates (year/month/day) and not as
For each stock, you can accomplish all theses things in just one line of code by using the appropia
keywords in the pd.read_csv() function. Here are a few hints:

Use the index_col keyword to indicate which column you want to use as an index.
For example index_col = ['Open']

Set the parse_dates keyword equal to True to convert the Dates into real dates of the form year/mor

Use the usecols keyword to select which columns you want to load into the DataFrame.
For example usecols = ['Open', 'High']
'''
google_stock = pd.read_csv(goog_path, index_col = ['Date'], parse_dates = True, usecols = ['Date',
apple_stock = pd.read_csv(aapl_path, index_col = ['Date'], parse_dates = True, usecols = ['Date', '
amazon_stock = pd.read_csv(amzn_path, index_col = ['Date'], parse_dates = True, usecols = ['Date',

# We create calendar dates between '2000-01-01' and  '2016-12-31'
dates = pd.date_range('2000-01-01', '2016-12-31')

# We create and empty DataFrame that uses the above dates as indices
```

```
all_stocks = pd.DataFrame(index = dates)

# preparing to join the datasets together

# Change the Adj Close column label to Google
google_stock = google_stock.rename(columns = {'Adj Close': 'Google'})

# Change the Adj Close column label to Apple
apple_stock = apple_stock.rename(columns = {'Adj Close': 'Apple'})

# Change the Adj Close column label to Amazon
amazon_stock = amazon_stock.rename(columns = {'Adj Close': 'Amazon'})

# now time to join the dataframe
# We join the Google stock to all_stocks
all_stocks = all_stocks.join(google_stock)

# We join the Apple stock to all_stocks
all_stocks = all_stocks.join(apple_stock)

# We join the Amazon stock to all_stocks
all_stocks =all_stocks.join(amazon_stock)

# let's describe the stocks
all_stocks.describe()

# NaN values dealio
# Print the column-wise count of NaN values, if any, in the all_stocks dataframe.
all_stocks.isnull().sum()
'''
Google    3095
Apple     1933
Amazon    1933
dtype: int64
'''

# Remove any rows that contain NaN values. Do this operation inplace.
all_stocks.dropna(axis=0, inplace = True)


# let's get some basic statistics
# Print the average stock price for each stock
all_stocks.mean()
'''
Google    347.420229
Apple      47.736018
Amazon    216.598177
'''


# Print the median stock price for each stock
all_stocks.median()
'''
Google    286.397247
Apple      39.461483
Amazon    161.820007
'''
```

```
# Print the standard deviation of the stock price for each stock
all_stocks.std()
'''
Google    187.671596
Apple      37.421555
Amazon    199.129792
'''

# Print the correlation between stocks
all_stocks.corr()
'''
          Google     Apple    Amazon
Google  1.000000  0.900242  0.952444
Apple   0.900242  1.000000  0.886321
Amazon  0.952444  0.886321  1.000000
'''

'''
We will now look at how we can compute some rolling statistics,
also known as moving statistics.
We can calculate for example the rolling mean (moving average) of the Google
stock price by using the Pandas dataframe.rolling().mean() method.
The dataframe.rolling(N).mean() calculates the rolling mean over
an N-day window. In other words, we can take a look at the average stock price
every N days using
the above method. Fill in the code below to calculate the average stock price
every 150 days for
Google stock
'''

rollingMean = all_stocks['Google'].rolling(150).mean()

# time to visualize the data
import matplotlib.pyplot as plt


# We plot the Google stock data
plt.plot(all_stocks['Google'])

# We plot the rolling mean ontop of our Google stock data
plt.plot(rollingMean)
plt.legend(['Google Stock Price', 'Rolling Mean'])
```