```
# %%

# ======================================================================================
# PANDAS EXCERCISES
# ======================================================================================


# %%

# excercise 1

# importing the panadas

import numpy as np
import pandas as pd

# %%

# creating a dataframe fruits

fruits = pd.DataFrame({'Apples': [30], 'Bananas': [21]})

fruits

# %%

# creating a dataframe fruit_sales

fruit_sales = pd.DataFrame({"Apples": [35, 41], "Bananas": [
                           21, 34]}, index=(["2017 Sales", "2018 Sales"]))

fruit_sales


# %%


# creating a series ingredients
# creating a series ingredients

ingredients = pd.Series(
    {"Flour": "4 Cups", "Milk": "1 Cup", "Eggs": "2 large", "Spam": "1 Can"})

ingredients

# %%


# reading the csv dataset of wine reviews into  a Dataframe reviews


reviews = pd.read_csv("~/KUBUNTU/DATASET/CSV_FILES/winemag-data_first150k.csv")

reviews.head()

# %%


# describe the dataset

desc = reviews.describe()


# %%

# create and display a DataFrame called `animals`

animals = pd.DataFrame({'Cows': [12, 20], 'Goats': [22, 19]}, index=[
                       'Year 1', 'Year 2'])

animals


# %%


# code to save this DataFrame to disk as a csv file with the name `cows_and_goats.csv`
```

```
animals.to_csv("~/KUBUNTU/DATASET/CSV_FILES/cows_and_goats.csv")


# %%

# excercise 2


# select the description column from reviews and assign it to variable desc

desc = reviews["description"]

desc

# %%


# check the type of the variable desc is


print(type(desc))


# %%


# select first column from description column of reviews and assign it to ariable first_description


first_description = reviews["description"][0]

first_description

# %%

# select first row of the data from reviews and assign it to first_row

first_row = reviews.iloc[0]
first_row = reviews.iloc[0]

first_row


# %%


# select the first 10 values from the description column in reviews and assign to variable first_descriptions

first_descriptions = reviews["description"][:10]

first_descriptions


# %%


# select the records with index labels 1,2,3,5,8 and assign to varible sample_reviews

sample_reviews = reviews.iloc[[1, 2, 3, 5, 8], :]


sample_reviews


# %%


# create a variable df have country, province, region_1, region_2, columns with index labels 0,1,10,100

df = reviews[["description", "country", "province",
             "region_1", "region_2"]].iloc[[0, 1, 10, 100]]

df


# %%


# a variable df containing country and variety column first 100 records

df = reviews[["country", "variety"]].iloc[:100, ]

df


# %%


#  craete a datframe italian_wines having reviews of wines made in Italy

italian_wines = reviews.loc[reviews["country"] == "Italy"]


italian_wines


# %%


# create a datafrme top_oceania_wines with all reviews having atleast 95 points out of 100 for wines from Australia and New Zealand


top_oceania_wines = reviews.loc[(reviews["country"].isin(
    ["Australia", "New Zealand"])) & (reviews["points"] >= 95)]


top_oceania_wines


# %%


# excercse 3


# find tjhe median of the points column


median_points = reviews["points"].median()


median_points

# %%


# what countries in dataset

countries_unique = reviews["country"].unique()


countries_unique


# %%


# create reviews_per_country variable mapping countries to the count of reviews of wines from that country


reviews_per_country = reviews["country"].value_counts()


reviews_per_country


# %%


# craete centered_price having version of price column with mean malue subtarcted


centered_price = reviews["price"]-reviews["price"].mean()
```

```python

centered_price


# %%


# create variable bargain_wine with the title of the wine with the highest points-to-price ratio


bargain_idx = (reviews["points"]/reviews["price"]).idxmax()

bargain_wine = reviews.loc[bargain_idx]["winery"]

bargain_wine


# %%


# create a series descriptor_counts to count how many the words "tropical" or "fruity" appear


trop = reviews["description"].map(lambda desc: "tropical" in desc).sum()


fruit = reviews["description"].map(lambda desc: "fruity" in desc).sum()


descriptor_counts = pd.Series([trop, fruit], index=["tropical", "fruity"])


descriptor_counts


# %%


# create variable star_ratings with number of strs regarding the each review in dataset


def rating(row):

    if row.country == "Canada":

        return 3
    elif row.points >= 95:

        return 3
    elif row.points >= 85:

        return 2

    else:

        return 1


star_ratings = reviews.apply(rating, axis="columns")

star_ratings


# %%


# excercise 4


# create series whose index is variety and whose value counts how many review each person wrote

reviews_written = reviews.groupby('variety').variety.count()

reviews_written


# %%
# %%

# Create a Series whose index is wine prices and whose values is the maximum number of points a wine costing that much was given in a revie

best_rating_per_price = reviews.groupby('price')['points'].max().sort_index()

best_rating_per_price


# %%

# Create a DataFrame whose index is the variety category from the dataset and whose values are the min and max values thereof.

price_extremes = reviews.groupby('variety').price.agg([min, max])


price_extremes

# %%

# Create a variable sorted_varieties containing a copy of the dataframe from the previous question where varieties are sorted in descending ord

sorted_varieties = price_extremes.sort_values(
    by=['min', 'max'], ascending=False)


sorted_varieties


# %%

# Create a Series whose index is reviewers and whose values is the average review score given out by that reviewer.

reviewer_mean_ratings = reviews.groupby('designation').points.mean()


reviewer_mean_ratings
```

```
# %%

# Create a Series whose index is a MultiIndexof {country, variety} pairs.


country_variety_counts = reviews.groupby(
    ['country', 'variety']).size().sort_values(ascending=False)


country_variety_counts


# %%

# excercise 5


# What is the data type of the points column in the dataset


dtype = reviews.points.dtype


dtype


# %%


# Create a Series from entries in the `points` column, but convert the entries to strings.


point_strings = reviews.points.astype(str)


point_strings


# %%
# %%


# How many reviews in the dataset are missing a price?


n_missing_prices = reviews.price.isnull().sum()


n_missing_prices


# %%


# Create a Series counting the number of times each value occurs in the region_1 field. This field is often missing data, so replace missing value


reviews_per_region = reviews.region_1.fillna(
    'Unknown').value_counts().sort_values(ascending=False)


reviews_per_region


# %%

# `region_1` and `region_2` are pretty uninformative names for locale columns in the dataset. Create a copy of `reviews` with these columns ren

renamed = reviews.rename(columns=dict(region_1='region', region_2='locale'))


renamed


# %%

# Set the index name in the dataset to `wines`.

```

```python

reindexed = reviews.rename_axis('wines', axis='rows')


reindexed

# %%

# ======================================================================================
# NUMPY EXCERCISES
# ======================================================================================


# %%

# NumPy Creating Array

# importing the numpy library


# %%

# Create a NumPy ndarray Object


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

print(arr)

print(type(arr))


# %%

# Use a tuple to create a NumPy array:


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

```

print(arr)


# %%

# Create a 0-D array with value 42


arr = np.array(42)

print(arr)


# %%

# Create a 1-D array containing the values 1,2,3,4,5:


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

print(arr)


# %%

# Create a 2-D array containing two arrays with the values 1,2,3 and 4,5,6:


arr = np.array([[1, 2, 3], [4, 5, 6]])

print(arr)


# %%

# Create a 3-D array with two 2-D arrays, both containing two arrays with the values 1,2,3 and 4,5,6:


```

```
arr = np.array([[[1, 2, 3], [4, 5, 6]], [[1, 2, 3], [4, 5, 6]]])

print(arr)


# %%


# Check how many dimensions the arrays have:


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)


# %%

# Create an array with 5 dimensions and verify that it has 5 dimensions:


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

print(arr)
print('number of dimensions :', arr.ndim)


# %%

# NumPy Array Indexing


# Get the first element from the following array:


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

print(arr[0])


# %%

# Get the second element from the following array.


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

print(arr[1])


# %%

# Get third and fourth elements from the following array and add them.

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

print(arr[2] + arr[3])


# %%

# Access the 2nd element on 1st dim:


arr = np.array([[1, 2, 3, 4, 5], [6, 7, 8, 9, 10]])

print('2nd element on 1st dim: ', arr[0, 1])


# %%
# Access the 5th element on 2nd dim:


arr = np.array([[1, 2, 3, 4, 5], [6, 7, 8, 9, 10]])

print('5th element on 2nd dim: ', arr[1, 4])

# %%

# Access the third element of the second array of the first array:


arr = np.array([[[1, 2, 3], [4, 5, 6]], [[7, 8, 9], [10, 11, 12]]])

print(arr[0, 1, 2])


# %%

# Print the last element from the 2nd dim:


arr = np.array([[1, 2, 3, 4, 5], [6, 7, 8, 9, 10]])

print('Last element from 2nd dim: ', arr[1, -1])


# %%


# NumPy Array Slicing


# Slice elements from index 1 to index 5 from the following array:


arr = np.array([1, 2, 3, 4, 5, 6, 7])

```

```python
print(arr[1:5])


# %%

# Slice elements from index 4 to the end of the array:


arr = np.array([1, 2, 3, 4, 5, 6, 7])

print(arr[4:])

# %%

# Slice elements from the beginning to index 4 (not included):


arr = np.array([1, 2, 3, 4, 5, 6, 7])

print(arr[:4])


# %%


# Slice from the index 3 from the end to index 1 from the end:


arr = np.array([1, 2, 3, 4, 5, 6, 7])

print(arr[-3:-1])


# %%


# Return every other element from index 1 to index 5:

```

```
arr = np.array([1, 2, 3, 4, 5, 6, 7])

print(arr[1:5:2])


# %%


# Return every other element from the entire array:


arr = np.array([1, 2, 3, 4, 5, 6, 7])

print(arr[::2])


# %%

# From the second element, slice elements from index 1 to index 4 (not included):


arr = np.array([[1, 2, 3, 4, 5], [6, 7, 8, 9, 10]])

print(arr[1, 1:4])

# %%

# From both elements, return index 2:


arr = np.array([[1, 2, 3, 4, 5], [6, 7, 8, 9, 10]])

print(arr[0:2, 2])


# %%

# From both elements, slice index 1 to index 4 (not included), this will return a 2-D array:
```

```python


arr = np.array([[1, 2, 3, 4, 5], [6, 7, 8, 9, 10]])

print(arr[0:2, 1:4])


# %%

# NumPy Data Types


# Get the data type of an array object:


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

print(arr.dtype)


# %%

# Get the data type of an array containing strings:


arr = np.array(['apple', 'banana', 'cherry'])

print(arr.dtype)


# %%

# Create an array with data type string:


print(arr)
print(arr.dtype)

```

```
# %%

# Create an array with data type 4 bytes integer:


arr = np.array([1, 2, 3, 4], dtype='i4')

print(arr)
print(arr.dtype)


# %%

# Change data type from float to integer by using 'i' as parameter value:


arr = np.array([1.1, 2.1, 3.1])

newarr = arr.astype('i')

print(newarr)
print(newarr.dtype)

# %%

# Change data type from float to integer by using int as parameter value:


arr = np.array([1.1, 2.1, 3.1])

newarr = arr.astype(int)

print(newarr)
print(newarr.dtype)


# %%
# Change data type from integer to boolean:


arr = np.array([1, 0, 3])

newarr = arr.astype(bool)

print(newarr)
print(newarr.dtype)


# %%

# NumPy Array Copy vs View


# Make a copy, change the original array, and display both arrays:


arr = np.array([1, 2, 3, 4, 5])
x = arr.copy()
arr[0] = 42

print(arr)
print(x)


# %%

# Make a view, change the original array, and display both arrays:


arr = np.array([1, 2, 3, 4, 5])
x = arr.view()
arr[0] = 42

print(arr)
print(x)
```

```python


# %%

# Make a view, change the view, and display both arrays:


arr = np.array([1, 2, 3, 4, 5])
x = arr.view()
x[0] = 31

print(arr)
print(x)


# %%

# Print the value of the base attribute to check if an array owns it's data or not:


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

x = arr.copy()
y = arr.view()

print(x.base)
print(y.base)


# %%

# NumPy Array Shape


# Print the shape of a 2-D array:


arr = np.array([[1, 2, 3, 4], [5, 6, 7, 8]])
```

```
print(arr.shape)


# %%

# Create an array with 5 dimensions using ndmin using a vector with values 1,2,3,4 and verify that last dimension has value 4:


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

print(arr)
print('shape of array :', arr.shape)


# %%

# NumPy Array Reshaping


# Convert the following 1-D array with 12 elements into a 2-D array.The outermost dimension will have 4 arrays, each with 3 elements:


arr = np.array([1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12])

newarr = arr.reshape(4, 3)

print(newarr)


# %%


# Convert the following 1-D array with 12 elements into a 3-D array.The outermost dimension will have 2 arrays that contains 3 arrays, each with

arr = np.array([1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12])

newarr = arr.reshape(2, 3, 2)

```

```python
print(newarr)


# %%

# Check if the returned array is a copy or a view:


arr = np.array([1, 2, 3, 4, 5, 6, 7, 8])

print(arr.reshape(2, 4).base)


# %%


# Convert 1D array with 8 elements to 3D array with 2x2 elements:


arr = np.array([1, 2, 3, 4, 5, 6, 7, 8])

newarr = arr.reshape(2, 2, -1)

print(newarr)


# %%

# Convert the array into a 1D array:


arr = np.array([[1, 2, 3], [4, 5, 6]])

newarr = arr.reshape(-1)

print(newarr)


# %%
# %%

# NumPy Array Iterating


# Iterate on the elements of the following 1-D array:


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

for x in arr:
    print(x)


# %%


# Iterate on the elements of the following 2-D array:


arr = np.array([[1, 2, 3], [4, 5, 6]])

for x in arr:
    print(x)


# %%

# Iterate on each scalar element of the 2-D array:


arr = np.array([[1, 2, 3], [4, 5, 6]])

for x in arr:
    for y in x:
        print(y)


# %%

# Iterate on the elements of the following 3-D array:


arr = np.array([[[1, 2, 3], [4, 5, 6]], [[7, 8, 9], [10, 11, 12]]])

for x in arr:
    print(x)


# %%

# Iterate down to the scalars:


arr = np.array([[[1, 2, 3], [4, 5, 6]], [[7, 8, 9], [10, 11, 12]]])

for x in arr:
    for y in x:
        for z in y:
            print(z)


# %%

# Iterate through the following 3-D array:


arr = np.array([[[1, 2], [3, 4]], [[5, 6], [7, 8]]])

for x in np.nditer(arr):
    print(x)


# %%


# Iterate through the array as a string:
```

```python

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

for x in np.nditer(arr, flags=['buffered'], op_dtypes=['S']):
    print(x)


# %%

# Iterate through every scalar element of the 2D array skipping 1 element:


arr = np.array([[1, 2, 3, 4], [5, 6, 7, 8]])

for x in np.nditer(arr[:, ::2]):
    print(x)


# %%

# Enumerate on following 1D arrays elements:


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

for idx, x in np.ndenumerate(arr):
    print(idx, x)


# %%

# Enumerate on following 2D array's elements:


arr = np.array([[1, 2, 3, 4], [5, 6, 7, 8]])

for idx, x in np.ndenumerate(arr):
    print(idx, x)
```

```python


# %%

# NumPy Joining Array


# Join two arrays


arr1 = np.array([1, 2, 3])

arr2 = np.array([4, 5, 6])

arr = np.concatenate((arr1, arr2))

print(arr)


# %%


# Join two 2-D arrays along rows (axis=1):


arr1 = np.array([[1, 2], [3, 4]])

arr2 = np.array([[5, 6], [7, 8]])

arr = np.concatenate((arr1, arr2), axis=1)

print(arr)


# %%

# NumPy Splitting Array

# Split the array in 3 parts:


arr = np.array([1, 2, 3, 4, 5, 6])

newarr = np.array_split(arr, 3)

print(newarr)


# %%

# Split the array in 4 parts:


arr = np.array([1, 2, 3, 4, 5, 6])

newarr = np.array_split(arr, 4)

print(newarr)


# %%

# Access the splitted arrays:


arr = np.array([1, 2, 3, 4, 5, 6])

newarr = np.array_split(arr, 3)

print(newarr[0])
print(newarr[1])
print(newarr[2])


# %%
# Split the 2-D array into three 2-D arrays.


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.


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)

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)


# %%

# NumPy Searching Arrays


# Find the indexes where the value is 4:


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)


# %%

# Find the indexes where the values are odd:

arr = np.array([1, 2, 3, 4, 5, 6, 7, 8])

x = np.where(arr % 2 == 1)

print(x)

# %%

# NumPy Sorting Arrays

```

```python
# Sort the array:

arr = np.array([3, 2, 0, 1])

print(np.sort(arr))


# %%

# Sort the array alphabetically:

arr = np.array(['banana', 'cherry', 'apple'])

print(np.sort(arr))


# %%

# Sort a boolean array:

arr = np.array([True, False, True])

print(np.sort(arr))


# %%

# Sort a 2-D array:

arr = np.array([[3, 2, 4], [5, 0, 1]])

print(np.sort(arr))


# %%


# ==============================================================================
# EXPERIMENT COMPLETED
# ================================================================================
```