|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| |  |  |  | | --- | --- | --- | | **Kingdom of Saudi Arabia**  **Ministry of Education**  **University of Jeddah**  **College of Computer Science and Engineering**  **Department of Computer Science and Artificial Intelligence** | Logo, company name  Description automatically generated | **المملكة العربية السعودية**  **وزارة التعليم**  **جامعة جدّة**  **كلية علوم وهندسة الحاسب**  **قسم علوم الحاسب والذكاء الاصطناعي** | |  |  |

|  |
| --- |
| **Lab 2** |
| **CCAI 312 Pattern Recognition** |
| **Third Trimester 2023**   |  |  | | --- | --- | | **Lab Date/Time:**  **Lab assignment submission Date/Time:** |  | | **Student Name: Bushra Dajam**  **Student ID: 2110054** | | |

|  |  |
| --- | --- |
| **Instructor Name** | **Section** |
|  |  |

**Instructions**:

The lab assignments must be submitted before the allocated Date/Time.

The lab assignments must by uploaded on LMS / sent by email to teacher@uj.edu.sa.

Plagiarism will be punished according to university rules.

|  |  |
| --- | --- |
| **PLO/CLO** | **SO** |
| **PLO S2 (CLO 2):** **Implement** a suitable pattern recognition technique for a given problem using Python | **SO 2:** Design, implement, and evaluate a computing-based solution to meet a given set of computing requirements in the context of the program’s discipline |
| **PLO C4 (CLO 3):** Discover and apply new knowledge as needed, using appropriate learning strategies. | **SO 7:** An ability to acquire and apply new knowledge as needed, using appropriate learning strategies. |

|  |  |  |  |
| --- | --- | --- | --- |
|  |  | **Max Score** | **Student Score** |
| **PLO S2 / CLO 2 / SO 2** | **Task 1** | **2** |  |
| **PLO C4 / CLO 3 / SO 7** | **Task 2** | **2** |  |
| **Total** | |  |  |

# Lab Description

In this lab you will learn about python panda library and perform exploratory dataset analysis (EDA) using two publicly available datasets.

# Objectives

* Introduce Pandas – The Python Data Analysis Library
* Read and Explore Data

# Lab Tool(s)

<https://www.kaggle.com/>

or

<https://colab.research.google.com/>

# Lab Deliverables

Submit A notebook to Blackboard containing your solution to the lab assessment at the end of this document.

# References:

<https://github.com/PacktPublishing/Python-Data-Cleaning-Cookbook>

<https://github.com/wesm/pydata-book>

Other Resources:

Free Panda Online courses

<https://www.udemy.com/course/python-pandas-for-your-grandpa/?ranMID=39197&ranEAID=JVFxdTr9V80&ranSiteID=JVFxdTr9V80-M0RwieWcFkDehKouZziExw&LSNPUBID=JVFxdTr9V80&utm_source=aff-campaign&utm_medium=udemyads>

<https://www.coursera.org/learn/python-data-analysis?irclickid=UHnwmyW0nxyNT08wnKTznzdyUkARMJ0Qb2Fyyw0&irgwc=1&utm_medium=partners&utm_source=impact&utm_campaign=3294490&utm_content=b2c>

# Exploratory Data Analysis

# 1-Introducing Pandas

**pandas** is a fast, powerful, flexible and easy to use open source data analysis and manipulation tool, built on top of the Python programming language.

To use pandas, we import:

import pandas as pd

# **1.1 Creating Data**

Pandas has two core objects: **DataFrame** and **Series**. Summarized in the table below

|  |  |  |
| --- | --- | --- |
| *Object* | *Description* | *Example* |
| Series | A Series is a sequence of data values (i.e., **list**). | pd.Series([1, 2, 3, 4, 5]) |
| DataFrame | A DataFrame is a **table**. It contains an array of individual entries, each of which has a certain value. Each entry corresponds to a row (or record) and a column. | pd.DataFrame({'Yes': [50, 21], 'No': [131, 2]}) |

## **DataFrame**

Consider this example

pd.DataFrame({'Bob': ['I liked it.', 'It was awful.'], 'Sue': ['Pretty good.', 'Bland.']})

We are using the pd.DataFrame() constructor to generate these DataFrame objects. The syntax for declaring a new one is a dictionary whose keys are the column names (Bob and Sue in this example), and whose values are a list of entries. This is the standard way of constructing a new DataFrame, and the one you are most likely to encounter.

The dictionary-list constructor assigns values to the column labels, but just uses an ascending count from 0 (0, 1, 2, 3, ...) for the row labels. Sometimes this is OK, but oftentimes we will want to assign these labels ourselves.

The list of row labels used in a DataFrame is known as an Index. We can assign values to it by using an index parameter in our constructor:

pd.DataFrame({'Bob': ['I liked it.', 'It was awful.'],   
 'Sue': ['Pretty good.', 'Bland.']},  
 index=['Product A', 'Product B'])

## **Series**

A Series is, in essence, a single column of a DataFrame. So you can assign column values to the Series the same way as before, using an index parameter. However, a Series does not have a column name, it only has one overall name:

pd.Series([30, 35, 40], index=['2015 Sales', '2016 Sales', '2017 Sales'], name='Product A')

# **1.2 Reading Data Files**

**Datasets**

We will work with two datasets in this chapter: The National Longitudinal Survey of Youth for 1997, a survey conducted by the United States government that surveyed the same group of individuals from 1997 through 2017; and the counts of COVID cases and deaths by country from Our World in Data.

We will use pandas tools to take a closer look at the **National Longitudinal Survey** (**NLS**) and COVID-19 case data.

**Importing CSV files**

The **read\_csv** method of the **pandas** library can be used to read a file with **comma separated values** (**CSV**) and load it into memory as a pandas data frame. In this recipe, we read a CSV file and address some common issues: creating column names that make sense to us, parsing dates, and dropping rows with critical missing data.

We will import a CSV file into pandas, taking advantage of some very useful **read\_csv** options:

1. Import the **pandas** library and set up the environment to make viewing the output easier:

import pandas as pd

2- Read the data file, set new names for the headings, and parse the date column.

Pass an argument of **1** to the **skiprows** parameter to skip the first row, pass a list of columns to **parse\_dates** to create a pandas datetime column from those columns, and set **low\_memory** to **False** to reduce the usage of memory during the import process:

landtemps = pd.read\_csv('/content/landtempssample.csv',  
names=['stationid','year','month','avgtemp','latitude','longitude','elevation','station','countryid','cou try'],  
skiprows=1,  
parse\_dates=[['month','year']],  
low\_memory=False  
)

3- check the type

type(landtemps)

## **1.3** **Get a quick glimpse of the data**

* View the first few rows:

landtemps.head()

* Show the data type for all columns:

landtemps.dtypes

* Show the number of rows and columns:

landtemps.shape

* Give the date column a better name and view the summary statistics for average monthly temperature:

landtemps.rename(columns={'month\_year':'measuredate'},

 inplace=True)

landtemps.dtypes

* view the summary statistics for average monthly temperature:

landtemps.avgtemp.describe()

* view the summary statistics for all continues columns:

landtemps.describe()

### Look for missing values for each column:

### Use isnull, which returns True for each value that is missing for each column, and False when not missing. Chain this with sum to count the missings for each column.

landtemps.isnull().sum()

* Remove rows with missing data for avgtemp:

landtemps.dropna(subset=['avgtemp'], inplace=True)  
landtemps.shape

### **1.3 Taking the Measure of Your Data**

Ask yourself what the first few things you try to find out in this situation are; that is, when you first get data about which you know little. It is probably something like this:

* How are the rows of the dataset uniquely identified? (What is the unit of analysis?)
* How many rows and columns are in the dataset?
* What are the key categorical variables and the frequencies of each value?
* How are important continuous variables distributed?
* How might variables be related to each other – for example, how might the distribution of continuous variables vary according to categories in the data?
* What variable values are out of expected ranges, and how are missing values distributed?

1.3.1Getting a first look at your data

import pandas as pd  
  
import numpy as np  
  
nls97 = pd.read\_csv("/content/nls97.csv")

nls97.set\_index("personid", inplace=True)  
nls97.index

nls97.shape

nls97.index.nunique()

# Show the data types and non-null value counts:  
nls97.info()

# show the first row of the nls97 data.Use transpose to show a little more of the output:  
nls97.head(2).T

* + 1. Selecting and organizing columns
* Select a column using the pandas **[]** bracket operator, and the **loc** and **iloc** accessors.

analysisdemo = nls97['gender']  
type(analysisdemo)

analysisdemo = nls97[['gender']]  
type(analysisdemo)

analysisdemo = nls97.loc[:,['gender']]  
type(analysisdemo)

analysisdemo = nls97.iloc[:,[0]]  
type(analysisdemo)

* Select multiple columns from a pandas DataFrame: Use the bracket operator and **loc** to select a few columns.

analysisdemo = nls97[['gender','maritalstatus','highestgradecompleted']]  
analysisdemo.shape

* Select multiple columns based on a list of columns.

keyvars = ['gender','maritalstatus','highestgradecompleted','wageincome','gpaoverall','weeksworked17','colenroct17']  
analysiskeys = nls97[keyvars]  
analysiskeys.info()

* Select all columns with specific data type:Use the **select\_dtypes** method to select columns by data type:

analysisnums = nls97.select\_dtypes(include=["number"])  
analysisnums.info()

* + 1. Selecting rows
* Use slicing to start at the 1001st row and go to the 1004th row

nls97[1000:1004].T

Use slicing to start at the 1001st row and go to the 1004th row, skipping every other row.

nls97[1000:1004:2].T

Select the first three rows using head and [] operator slicing

nls97.head(3).T

Select the last three rows using tail and [] operator slicing

nls97.tail(3).T

Select a few rows using the loc data accessor

nls97.loc[[195884,195891,195970]].T

Select a row from the beginning of the DataFrame with the iloc data access

nls97.iloc[[0]].T

Select a few rows from the beginning of the DataFrame with the iloc data accessor.

nls97.iloc[[0,1,2]].T

nls97.iloc[0:3].T

Select a few rows from the end of the DataFrame with the iloc data accessor.

nls97.iloc[[-3,-2,-1]].T

Select multiple rows conditionally using boolean indexing

nls97.nightlyhrssleep.count()  
sleepcheckbool = nls97.nightlyhrssleep<=4  
sleepcheckbool

lowsleep = nls97.loc[sleepcheckbool]  
lowsleep.shape

Select rows based on multiple conditions

lowsleep.childathome.describe()

lowsleep3pluschildren =nls97.loc[(nls97.nightlyhrssleep<=4) & (nls97.childathome>=3)]  
lowsleep3pluschildren.shape

Select rows and columns based on multiple conditions

lowsleep3pluschildren = nls97.loc[(nls97.nightlyhrssleep<=4) & (nls97.childathome>=3), ['nightlyhrssleep','childathome']]  
lowsleep3pluschildren

* + 1. Generating frequencies for categorical variables

We use pandas tools to generate frequencies, particularly the very handy value\_counts:

Most of the columns in the **nls97** DataFrame (57 out of 88) have the object data type. If we are working with data that is logically categorical, but does not have a category data type in pandas, there are good reasons to convert it to the category type. Not only does this save memory, it also makes data cleaning a little easier, as we saw in this recipe.

nls97.loc[:, nls97.dtypes == 'object'] =nls97.select\_dtypes(['object']).apply(lambda x: x.astype('category'))

Show the names for columns with the category data type and check for the number of missing values.

catcols = nls97.select\_dtypes(include=["category"]).columns  
nls97[catcols].isnull().sum()

Show the frequencies for marital status:

nls97.maritalstatus.value\_counts()

Turn off sorting by frequency:

nls97.maritalstatus.value\_counts(sort=False)

Show percentages instead of counts

nls97.maritalstatus.value\_counts(sort=False, normalize=True)

Show the percentages for all government responsibility columns

nls97.filter(like="gov").apply(pd.value\_counts, normalize=True)

Find the percentages for all government responsibility columns of people who are married.

nls97[nls97.maritalstatus=="Married"].filter(like="gov").apply(pd.value\_counts, normalize=True)

Find the frequencies and percentages for all category columns in the DataFrame.

#First, open a file to write out the frequencies:

freqout = open('frequencies.txt', 'w')  
for col in nls97.select\_dtypes(include=["category"]):  
  print(col, "----------------------", "frequencies",  
        nls97[col].value\_counts(sort=False),"percentages",  
        nls97[col].value\_counts(normalize=True, sort=False),  
        sep="\n\n", end="\n\n\n", file=freqout)  
freqout.close()

* + 1. Generating statistics for continuous variables

Pandas has a good number of tools we can use to get a sense of the distribution of continuous variables. We will focus on the splendid functionality of **describe** in this recipe and demonstrate the usefulness of histograms for visualizing variable distributions.

Before doing any analysis with a continuous variable it is important to have a good understanding of how it is distributed – its central tendency, its spread, and its skewness. This understanding greatly informs our efforts to identify outliers and unexpected values. But it is also crucial information in and of itself.

* Get the descriptive statistics on the COVID totals and demographic columns

nls97.describe()

It is often a red flag when the mean and median (50%) have dramatically different values. Cases and deaths are heavily skewed to the right (reflected in the mean being much higher than the median). This alerts us to the presence of outliers at the upper end.

* Take a closer look at the distribution of values for the wageincome column: Use NumPy's arange method to pass a list of floats from 0 to 1.0 to the quantile method of the DataFrame:

nls97["wageincome"].quantile(np.arange(0.0, 1.1, 0.1))

The more detailed percentile data in step 4 further supports this sense of skewness. For instance, the gap between the 90th-percentile and 100th-percentile values for cases and deaths is substantial. These are good first indicators that we are not dealing with normally distributed data here

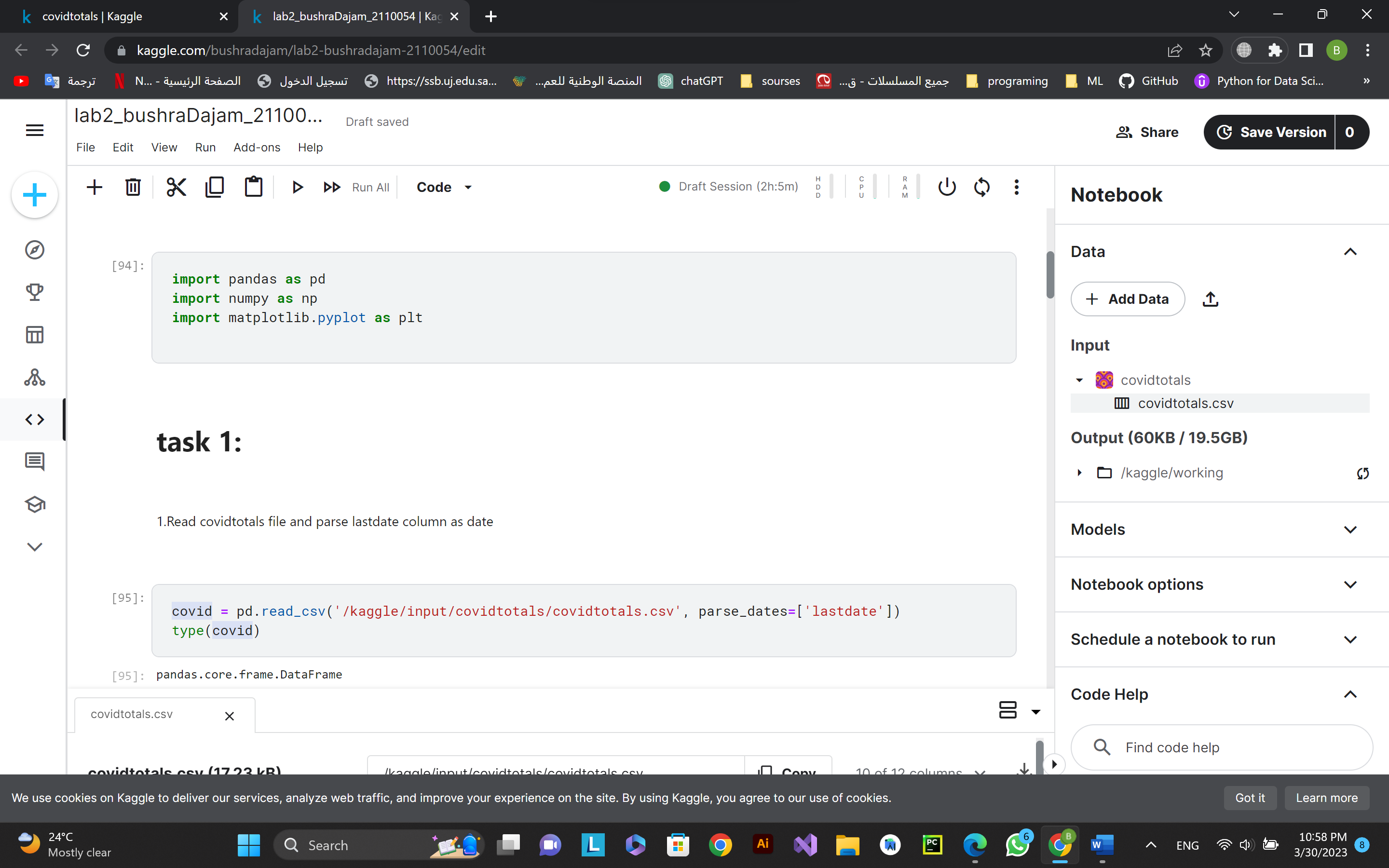
* We take a look at the distribution of a few key continuous variables: View the distribution of income.

import matplotlib.pyplot as plt

plt.hist(nls97['wageincome'], bins=10)  
  
plt.title("Total wage income")  
  
plt.xlabel('income')  
  
plt.ylabel("Number of individuals")  
  
plt.show()

Task **Task 1: [PLO S2 / CLO 2 / SO 2] [2 marks]**

1. Read covidtotals file and parse lastdate column as date

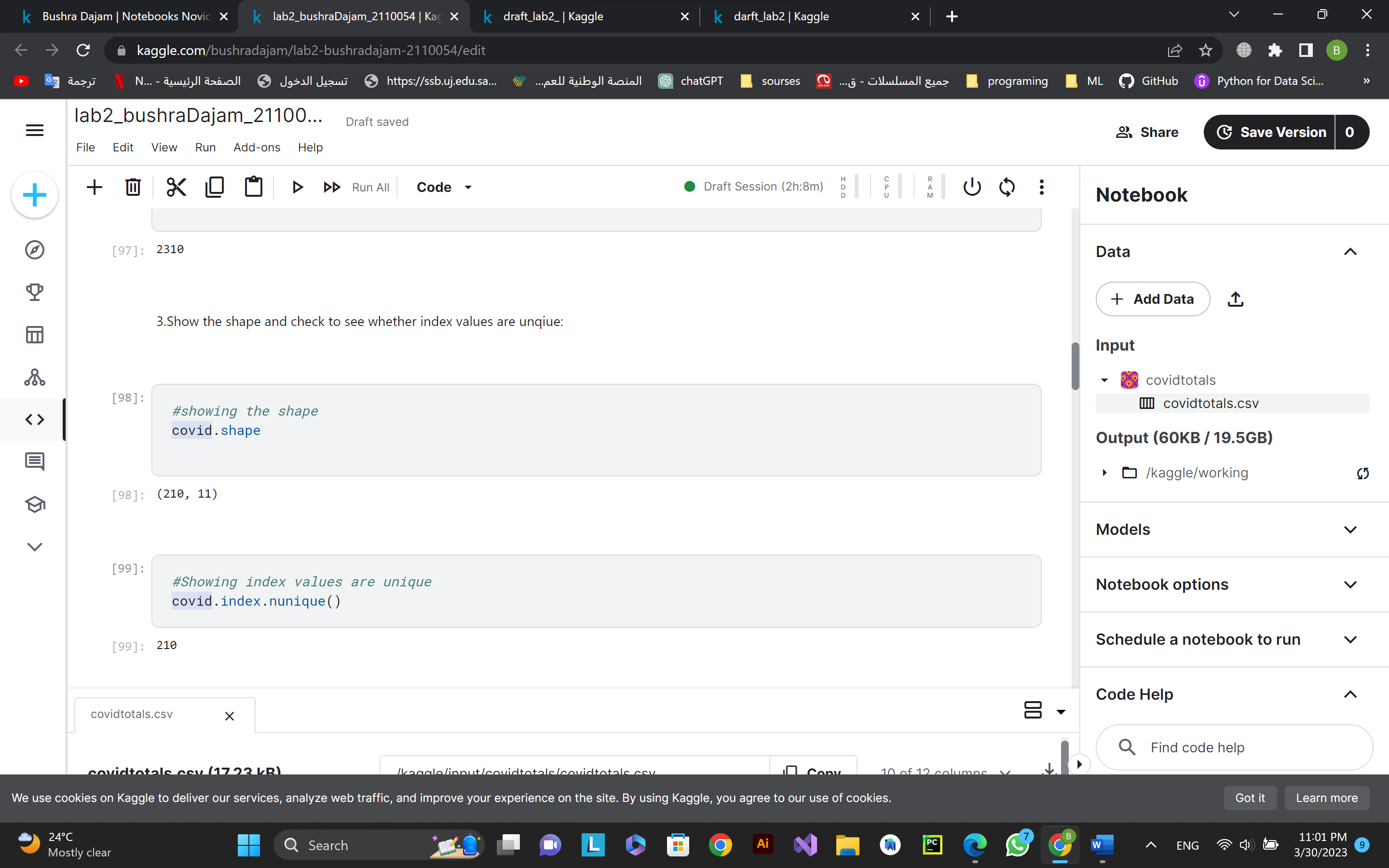


1. Set and show the index and size for the COVID data (use iso\_code as index).

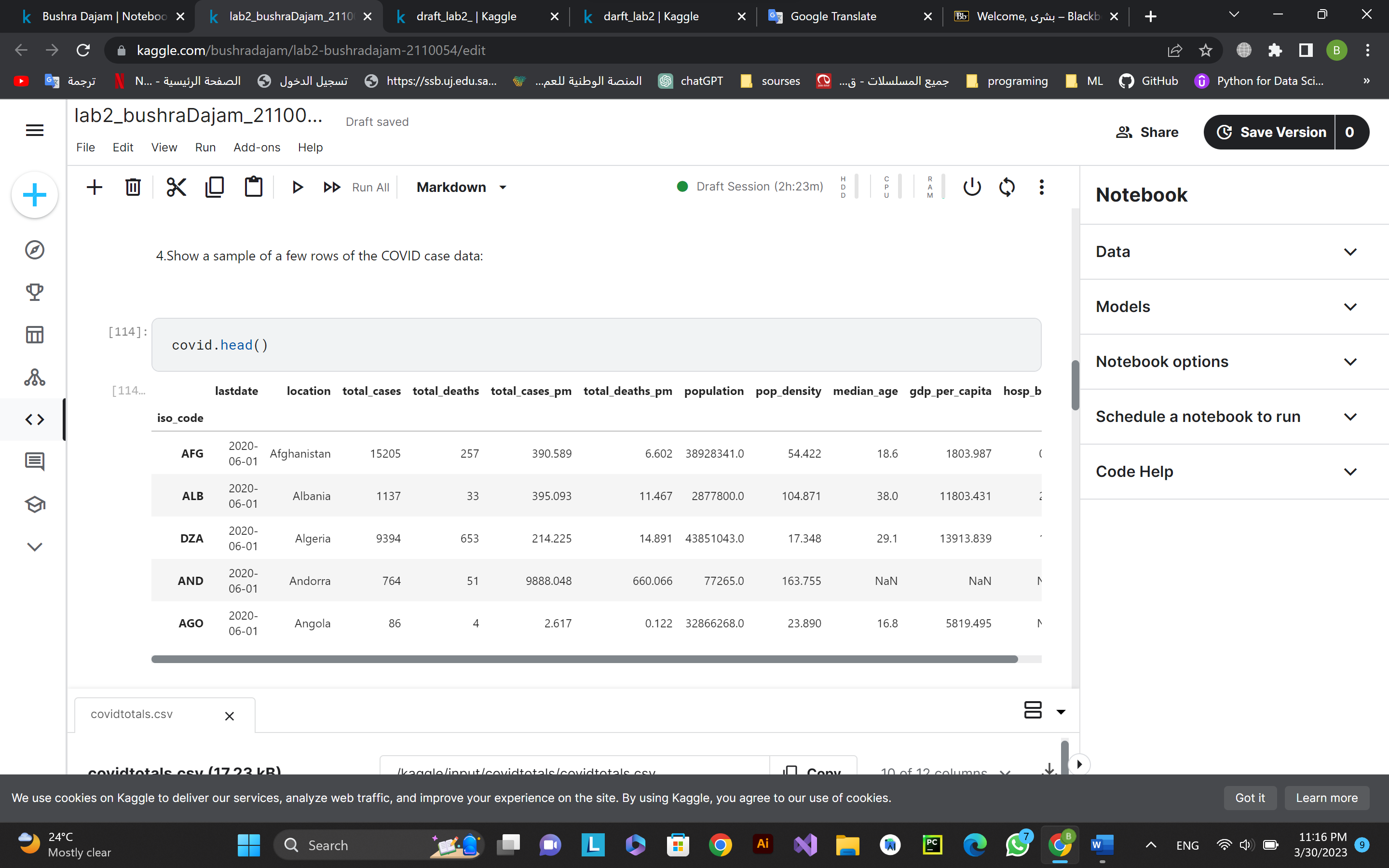
A screenshot of a computer

Description automatically generated

1. Show the shape and check to see whether index values are unqiue:



1. Show a sample of a few rows of the COVID case data:



1. Get the descriptive statistics on the COVID totals and demographic columns

A screenshot of a computer

Description automatically generated

**Task 2: [PLO C4 / CLO 3 / SO 7] [2 mark]**

1. Take a closer look at the distribution of values for the cases and deaths columns.

Hint : Use NumPy's **arange** method to pass a list of floats from 0 to 1.0 to the **quantile** method of the DataFrame.

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

1. Draw a histogram of the distribution of total cases.

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

1. Based on your analysis are the 'total\_cases' and ‘total\_deaths’ distributions skewed or not?  If skewed, to which direction? justify your answer.

yes it is positive/right-skewed.

It seems that both the 'total\_cases' and 'total\_deaths' columns are heavily right-skewed. This is indicated by the fact that the median (50th percentile) for both columns is significantly smaller than the mean, and the upper quantiles (0.8, 0.9, 1.0) have much larger values than the lower quantiles (0.0 - 0.6). This indicates that there are a small number of countries with very high numbers of cases and deaths, while the majority of countries have relatively low numbers. Therefore, the direction of skewness is towards the right.

A screenshot of a computer

Description automatically generated