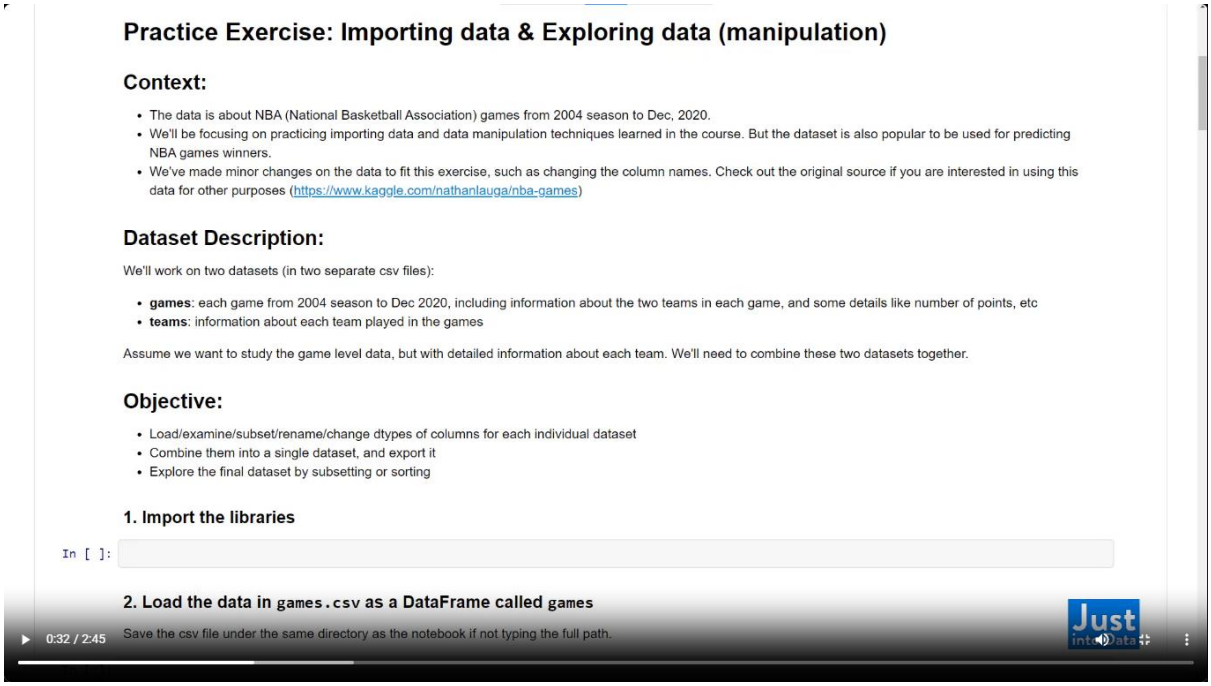
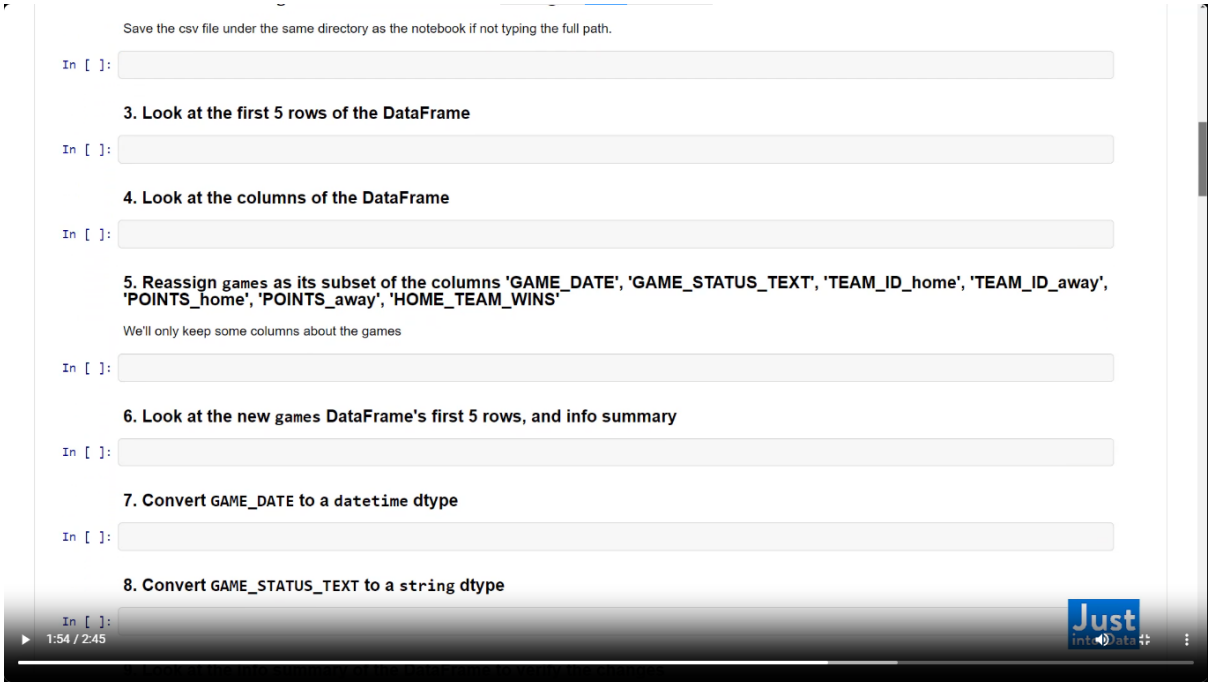

Practice Exercise: Importing data & Exploring data (manipulation)
Context:
The data is about NBA (National Basketball Association) games from 2004 season to Dec, 2020.
We'll be focusing on practicing importing data and data manipulation techniques learned in the course. But the dataset is also popular to be used for predicting NBA games winners.
We've made minor changes on the data to fit this exercise, such as changing the column names. Check out the original source if you are interested in using this data for other purposes (https://www.kaggle.com/nathanlauga/nba-games)
Dataset Description:
We'll work on two datasets (in two separate csv files):
games: each game from 2004 season to Dec 2020, including information about the two teams in each game, and some details like number of points, etc
teams: information about each team played in the games
Assume we want to study the game level data, but with detailed information about each team. We'll need to combine these two datasets together.
Objective:
Load/examine/subset/rename/change dtypes of columns for each individual dataset
Combine them into a single dataset, and export it
Explore the final dataset by subsetting or sorting
1. Import the libraries
In [ ]:
2. Load the data in games.csv as a DataFrame called games
Save the csv file under the same directory as the notebook if not typing the full path.
0:32 / 2:45
Just


Save the csv file under the same directory as the notebook if not typing the full path.
In [ ]:
3. Look at the first 5 rows of the DataFrame
In [ ]:
4. Look at the columns of the DataFrame
In [ ]:
5. Reassign games as its subset of the columns 'GAME_DATE', 'GAME_STATUS_TEXT', 'TEAM_ID_home', 'TEAM_ID_away', 'POINTS_home', 'POINTS_away', 'HOME_TEAM_WINS'
We'll only keep some columns about the games
In [ ]:
6. Look at the new games DataFrame's first 5 rows, and info summary
In [ ]:
7. Convert GAME_DATE to a datetime dtype
In [ ]:
8. Convert GAME_STATUS_TEXT to a string dtype
In [ ]:
1:54 / 2:45
Just

9. Look at the info summary of the DataFrame to verify the changes

In [ ]:

10. Load the data in `teams.csv` as a DataFrame called `teams`, and look at its first 5 rows, and its columns

In [ ]:

11. Reassign `teams` as a subset of its columns 'TEAM_ID', 'CITY', 'NICKNAME', and look at its first 5 rows and info summary

We'll only keep some columns about the teams

In [ ]:

12. Convert both columns `CITY` and `NICKNAME` to a `string` dtype

In [ ]:

13. Verify the changes with the `dtypes` attribute

In [ ]:

14. Print out the first two rows of `games` and `teams`, how can we combine them?

In [ ]:

Hint:

Within the `games` DataFrame, there are two columns `TEAM_ID_home` and `TEAM_ID_away`. This is because each game involves two teams playing against each other. The team that played in its own location, is called the 'home' team, the team that played outside its location, is called the 'away' team. Each game has one 'home' team and one 'away' team.

1:57 / 2:45

Within the `games` DataFrame, there are two columns `TEAM_ID_home` and `TEAM_ID_away`. This is because each game involves two teams playing against each other. The team that played in its own location, is called the 'home' team, the team that played outside its location, is called the 'away' team. Each game has one 'home' team and one 'away' team.

While the `teams` DataFrame stores the information about each team, the identifier for each team is the column `TEAM_ID`.

We can merge the two DataFrames based on:

- *`TEAM_ID_home` in `games` and `TEAM_ID` in `teams`: to get the team information for the 'home' team*
- *`TEAM_ID_away` in `games` and `TEAM_ID` in `teams`: to get the team information for the 'away' team*

15. Merge (inner) `games` and `teams` based on 'TEAM_ID_home' and 'TEAM_ID', call the merged DataFrame `games_with_home_team`

In [ ]:

16. Print out the first 5 rows of the new DataFrame

Since we used the column `TEAM_ID_home` when merging, the two columns `CITY` and `NICKNAME` are storing the city and nickname of the 'home' team. So let's rename them.

In [ ]:

17. Rename the column `CITY` as 'city_home', `NICKNAME` as 'nickname_home'

In [ ]:

18. Merge (inner) `games_with_home_team` and `teams` based on 'TEAM_ID_away' and 'TEAM_ID', call the merged DataFrame `games_with_both_teams`

In [ ]:

2:25 / 2:45

19. Print out the first two rows of the new DataFrame

19. Print out the first two rows of the new DataFrame

Since we used the column `TEAM_ID_away` when merging, the two columns `CITY` and `NICKNAME` are storing the city and nickname of the 'away' team. So let's rename them.

In [ ]:

20. Rename the column `CITY` as 'city_away', `NICKNAME` as 'nickname_away'

In [ ]:

21. Print out the first 5 rows of the new DataFrame

You probably have noticed that there are two columns called `TEAM_ID_x` and `TEAM_ID_y`. This is because we merged the DataFrame twice with the `teams` DataFrame. So the `TEAM_ID` column from `teams` is added twice to the merged DataFrame.

To avoide duplicate column names, `pandas` added the suffixes of '_x' and '_y' to distinguish them. But due to inner joins, `TEAM_ID_x` is the same as `TEAM_ID_home`, and `TEAM_ID_y` is the same as `TEAM_ID_away`.

In [ ]:

The team ID columns are not needed after the merge of the DataFrames.

The below code is provided to drop the columns 'TEAM_ID_home', 'TEAM_ID_away', 'TEAM_ID_x', 'TEAM_ID_y' from `games_with_both_teams`. We'll learn about this `drop` method in a later section

In [ ]:
```
games_with_both_teams = games_with_both_teams.drop(columns=['TEAM_ID_home', 'TEAM_ID_away', 'TEAM_ID_x', 'TEAM_ID_y'])
games_with_both_teams.head()
```

22. Make a copy of `games_with_both_teams` and assign it as `games`

In [ ]:

2:26 / 2:45

23. Change the column names in `games` to all lowercase

In [ ]:

24. Print out the columns of `games` to verify the changes

In [ ]:

25. Print out the columns of `games_with_both_teams` to verify that the original DataFrame wasn't impacted by the copy

In [ ]:

26. Check the dimensionality of `games`

In [ ]:

27. Export `games` as a csv file called 'games_transformed.csv', and open the csv file to look at it

Feel free to test the difference of the csv files with or without the argument `index=False`

In [ ]:

28. Select all the columns of 'number' dtypes from `games`

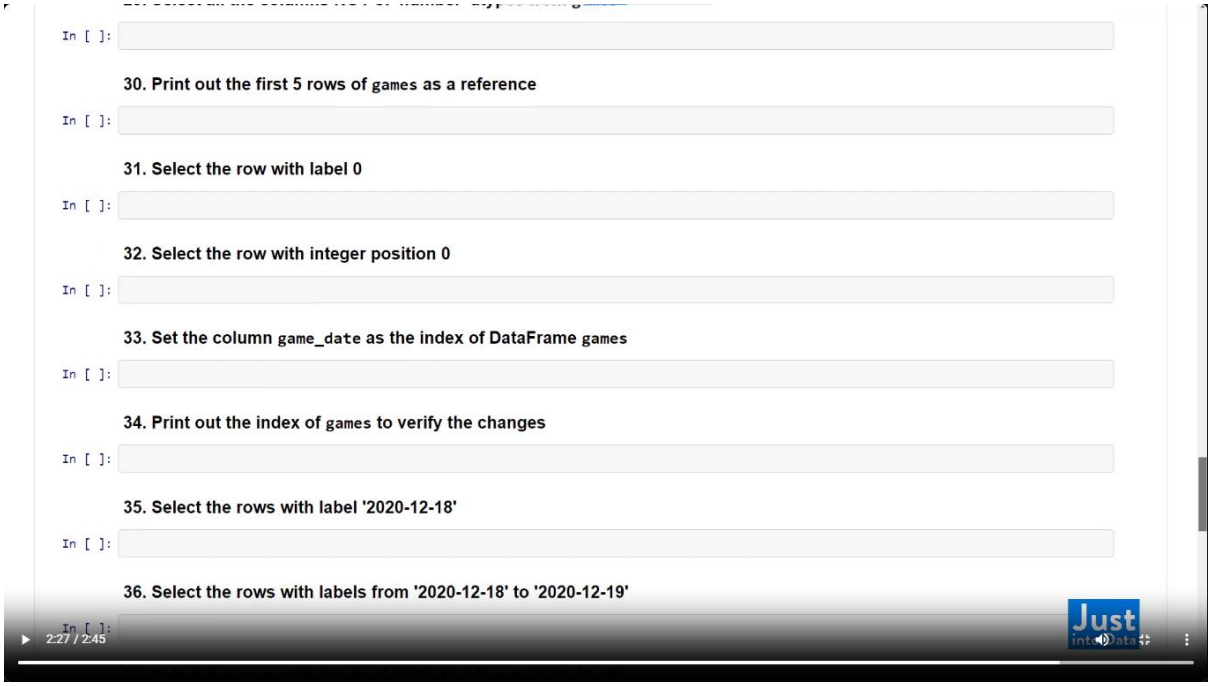
In [ ]:

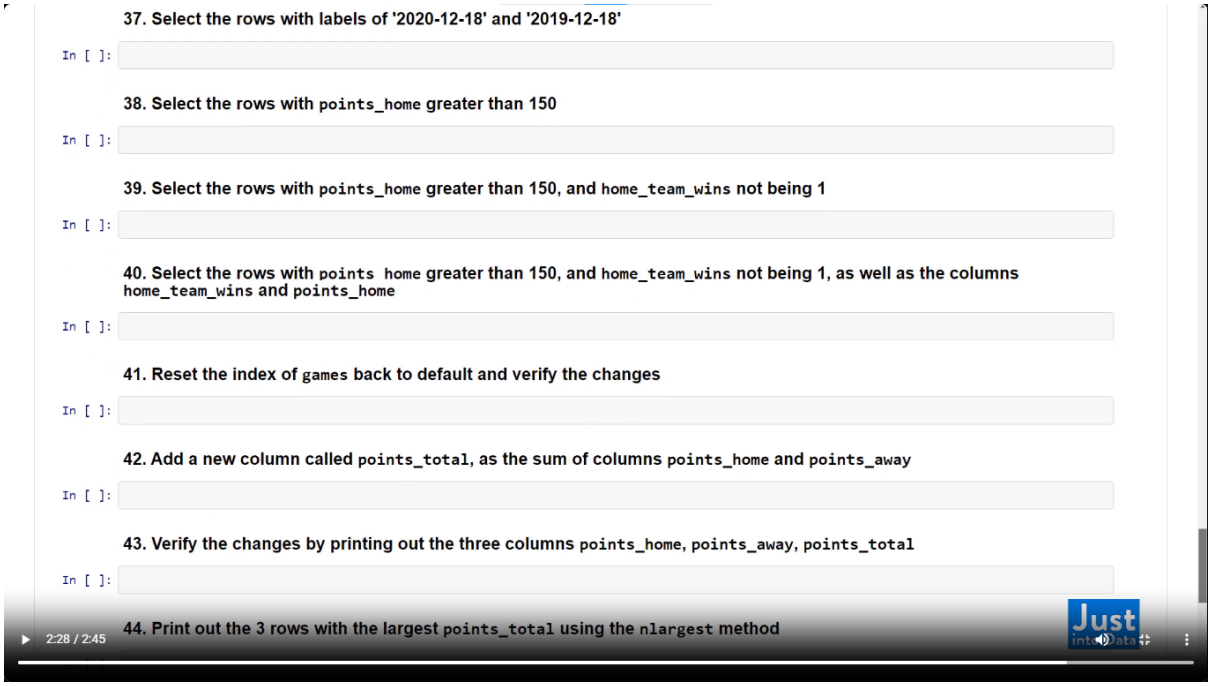
29. Select all the columns NOT of 'number' dtypes from `games`

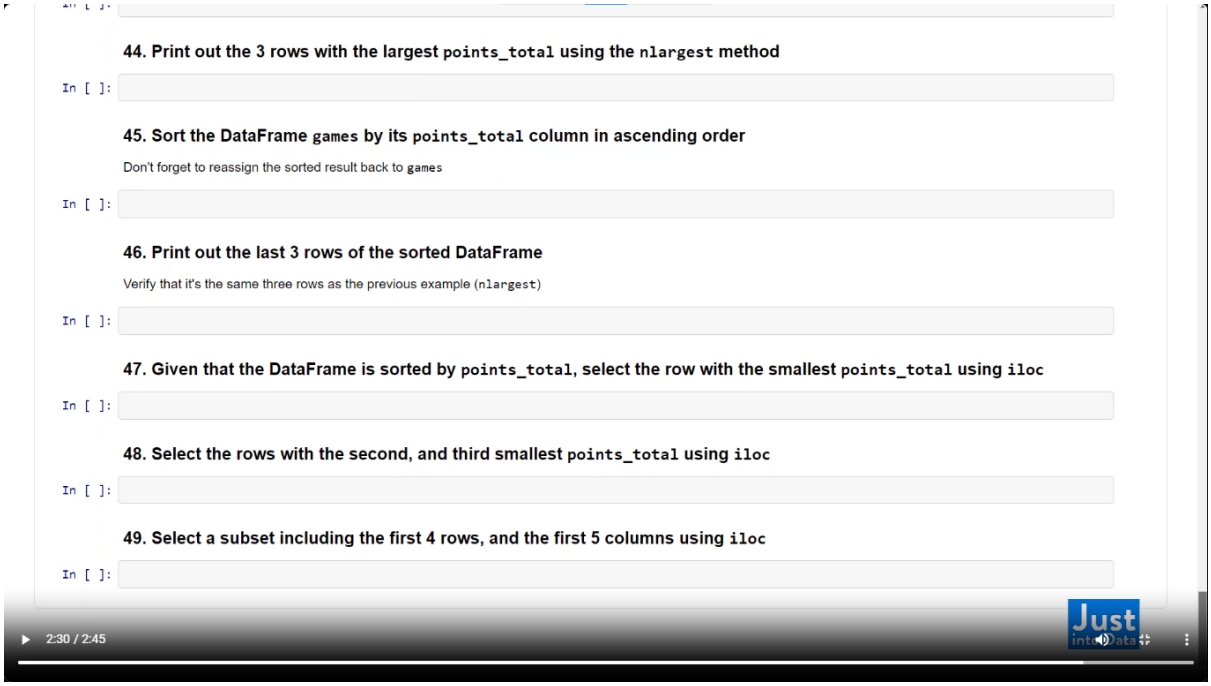
In [ ]:

2:26 / 2:45

30. Print out the first 5 rows of games as a reference


In [ ]:
30. Print out the first 5 rows of games as a reference
In [ ]:
31. Select the row with label 0
In [ ]:
32. Select the row with integer position 0
In [ ]:
33. Set the column game_date as the index of DataFrame games
In [ ]:
34. Print out the index of games to verify the changes
In [ ]:
35. Select the rows with label '2020-12-18'
In [ ]:
36. Select the rows with labels from '2020-12-18' to '2020-12-19'
In [ ]:
2:27 / 2:45


37. Select the rows with labels of '2020-12-18' and '2019-12-18'
In [ ]:
38. Select the rows with points_home greater than 150
In [ ]:
39. Select the rows with points_home greater than 150, and home_team_wins not being 1
In [ ]:
40. Select the rows with points_home greater than 150, and home_team_wins not being 1, as well as the columns home_team_wins and points_home
In [ ]:
41. Reset the index of games back to default and verify the changes
In [ ]:
42. Add a new column called points_total, as the sum of columns points_home and points_away
In [ ]:
43. Verify the changes by printing out the three columns points_home, points_away, points_total
In [ ]:
44. Print out the 3 rows with the largest points_total using the nlargest method
2:28 / 2:45


44. Print out the 3 rows with the largest points_total using the nlargest method
In [ ]:
45. Sort the DataFrame games by its points_total column in ascending order
Don't forget to reassign the sorted result back to games
In [ ]:
46. Print out the last 3 rows of the sorted DataFrame
Verify that it's the same three rows as the previous example (nlargest)
In [ ]:
47. Given that the DataFrame is sorted by points_total, select the row with the smallest points_total using iloc
In [ ]:
48. Select the rows with the second, and third smallest points_total using iloc
In [ ]:
49. Select a subset including the first 4 rows, and the first 5 columns using iloc
In [ ]:
2:30 / 2:45
Just