Here are the instructions for this week's independent lab, for context within this chat. Do not solve any of the questions or write any code as of yet; assist me in learning, but only output code responses when I explicitly ask. "# Independent Lab: Manipulating Data

Context of Assignment

You currently work in the Information Systems department for a consulting firm working with the state government agency that oversees the healthcare system in California. Your administrator reports directly to the CIO of your company. You have been asked to join a team charged with assessing the condition of the healthcare system in California. Several surveys were emailed to a random sample of 61 hospitals and the results have been recorded in the file [CaliforniaHospitalData.csv](data/CaliforniaHospitalData.csv). Additionally, you have been provided with personnel data containing employee information within the file [CaliforniaHospitalData_Personnel.txt](data/CaliforniaHospitalData_Personnel.txt). See the tables below for more metadata.

This first table provides the variables in the hospital data.

Variable	Description
HospitalID	The primary key of each hospital
Name	The legal name of the hospital
Zip	Zipcode where the hospital is located
Website	The url for the hospital’s website
TypeControl	Indicates the primary managing entity of the hospital
Teaching	Indicates teaching status
DonorType	This field indicates the most prominent group of donors
NoFTE	Number of full-time employees registered at the hospital
NetPatRev	Net patient revenue
InOperExp	Estimate of the inpatient operating costs
OutPerExp	Estimate of the outpatient operating costs
OperRev	Operating revenue of the hospital
OperInc	Operating Income is the operating revenue less the operating expenses
AvlBeds	The number of available beds in the hospital

This second table provides the data for the personnel data.

Variable	Description
HospitalID	The foreign key of the hospital where position is held
Work_ID	Primary key of the personnel
LastName	The last name of the personnel
FirstName	First name of the personnel
Gender	Gender of the individual
PositionID	The foreign key for the position held
PositionTitle	The title of this position
Compensation	The annual amount the position is compensated for service
MaxTerm	The maximum number of years an individual can serve in this position
StartDate	The beginning of service for this position

Your goal for this assignment is to pre-process this data for the Business Intelligence team. They need to perform descriptive analyses on the data and have requested you to provide a clean start for them.

Merging the Data

Using Python, import the data and combine both into a single DataFrame. Note, both files have the column HospitalID. Use this column to merge the files. That is, each employee has a unique position at one or more hospitals; these positions need to be combined with the correct hospital. Thus, each hospital has one unique employee listed in the data, but an employee can be assigned to more than one hospital. Use the method [merge()](https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.merge.html) from pandas.

After the data has been merged, please remove the following columns of data:

* duplicate columns
* `Work_ID`
* `PositionID`
* `Website`

Exporting the Data

Select only those hospitals that are *Small/Rural* and have 15 or more available beds. Exclude hospitals with a negative operating income. Export your data as tab-delimited and name the file `hospital_data_new.txt`. Save this to the directory `data`. Be sure to not export the index using the argument `index = False` for the pandas' function `to_csv()`.

Renaming Columns

Open the newly created file in Python as a new DataFrame. Change the name of the following columns:

* `NoFTE` to `FullTimeCount`
* `NetPatRev` to `NetPatientRevenue`
* `InOperExp` to `InpatientOperExp`
* `OutOperExp` to `OutpatientOperExp`
* `OperRev` to `Operating_Revenue`
* `OperInc` to `Operating_Income`

Inserting Records

Select two of the existing hospitals in the newly created DataFrame and create a new position for each hospital. Insert yourself as the new employee at those two hospitals; put in your first name and last name. Put today’s date as the start date. Select one of the positions as shown in the table below and fill out the data accordingly. Fill in the rest of the columns as you choose. Use the method `concat()`. You should have two new rows of data. Save the DataFrame as `new_merge`.

PositionTitle	Compensation	MaxTerm
Regional Representative	46978	4
State Board Representative	89473	3
Acting Director	248904	8
Safety Inspection Member	23987	2

Filtering Data

Perform the following tasks:

* Using the `new_merge` data, select all hospitals that are non-profit with more than 250 employees, unless the net patient revenue is smaller than $109,000. Remove the columns containing employee information and save it as a new DataFrame.
* Using the `new_merge` data, select all the *Regional Representatives* who work at a hospital with operating income greater than $100,000. Save this as a new

DataFrame.

Convert Date-Time Data

Using the `new_merge` DataFrame, convert any date-time columns into a datetime datatype. Confirm your changes by outputting the data types for all columns. Additionally, output the first 5 records of the column(s) you converted.

Save your notebook with output displayed within it and submit for grading."

Reasoned for 10 seconds >

Got it—the lab assignment involves several important data manipulation tasks using pandas. To summarize conceptually without writing any code yet:

1. **Importing and Merging Data:**
    - You need to read two files (a CSV for hospital data and a TXT for personnel data) and merge them based on the common key `HospitalID`.
    - Understanding the merge method in pandas is key here, particularly how inner, left, or right joins work depending on your dataset relationships.
2. **Data Cleaning:**
    - After merging, you'll remove duplicate columns that might result from the merge and specifically drop columns like `Work_ID`, `PositionID`, and `Website`.
    - This step is critical to prevent redundancy and maintain clarity in your consolidated DataFrame.
3. **Filtering and Exporting Data:**
    - The assignment asks you to filter hospitals by specific criteria: selecting those classified as *Small/Rural* that have at least 15 available beds and non-negative operating income.
    - You will then export the filtered data as a tab-delimited file without the DataFrame index.
4. **Renaming Columns:**
    - Once you load the exported file as a new DataFrame, renaming several columns is required. This ensures that the variable names clearly indicate what the values represent (e.g., `NoFTE` becomes `FullTimeCount`).
5. **Inserting New Records:**
    - You'll add two new records for hospital positions. This involves selecting two existing hospitals and concatenating new rows with your own employee details

    (like first name, last name, and today's date) along with specific position information such as compensation and max term.

6. **Additional Filtering Tasks:**
    - One task requires filtering based on multiple conditions (e.g., nonprofit hospitals with more than 250 employees, unless revenue is too low).
    - Another filtering task focuses on selecting all “Regional Representatives” at hospitals with an operating income above $100,000.
7. **Date-Time Conversion:**
    - Finally, you'll convert columns containing date-time information into the proper datetime dtype in pandas and verify these changes by outputting the data types and a sample of the converted columns.

This assignment touches on various data transformation skills that are very useful for preparing data before analysis. If you’d like, I can help clarify any concepts, walk through the steps more deeply, or explain specific functions and methods (like `merge()`, `concat()`, and `to_csv()`) without providing code until you ask for it.

Let me know how you’d like to proceed or if there’s any particular concept you’d like to delve into further!

In my first cell, I should be loading both the CSV and a TXT file. I am loading the CSV file with the following command: "CA_Hospital = pd.read_csv("/Users/AlexMcKenzie/Library/CloudStorage/GoogleDrive-alex.c.mckenzie@gmail.com/My Drive/Education/UM Graduate School/BGEN632/Week 7/data/CaliforniaHospitalData.csv")"; can i use a similar command to load a TXT file in python?

Reasoned for 14 seconds

Yes, you can use a similar command to load a TXT file with pandas. The key is ensuring you specify the correct delimiter if the file isn't comma-separated. For example, if your TXT file is tab-delimited, you could use:

```python
CA_Hospital_Personnel =
pd.read_csv("path/to/CaliforniaHospitalData_Personnel.txt",
delimiter='\t')
```

Alternatively, you can use `pd.read_table()`, which by default assumes tab-delimited data:

```python
```

```
CA_Hospital_Personnel =
pd.read_table("path/to/CaliforniaHospitalData_Personnel.txt")
```

These commands work similarly to reading a CSV file, with the difference being the delimiter used to parse the data.

how can i tell if my file is tab-delimited?

Reasoned about file delimiter detection for 8 seconds >

You can determine if a file is tab-delimited in a couple of ways:

1. **Manually Inspect the File:**
   Open the file in a simple text editor (like Notepad, TextEdit, or VSCode). Look at a few lines and check whether columns are separated by the tab character. In many editors, tabs might appear as extra spaces, or you might enable visible whitespace to see them. If you see the separation happening where you expect a tab character (typically a significant gap or an indicator if whitespace is shown), it's likely tab-delimited.
2. **Programmatically Detect the Delimiter:**
   Python's built-in `csv` module includes a `Sniffer` class which can analyze a sample of the file and infer the delimiter. For example:

   python Copy

   ```python
   import csv filename =
   "/path/to/CaliforniaHospitalData_Personnel.txt" with open(filename,
   "r") as file: sample = file.read(1024) dialect =
   csv.Sniffer().sniff(sample) print("Detected delimiter:",
   repr(dialect.delimiter))
   ```

   If the output shows `'\t'` (the tab character), then your file is tab-delimited.

Using these approaches, you can confidently determine the format of your TXT file before loading it with `pd.read_csv()` or `pd.read_table()`.

next I need to merge/join two csv and txt files, using this method: "DataFrame.merge(right, how='inner', on=None, left_on=None, right_on=None, left_index=False, right_index=False, sort=False, suffixes=('_x', '_y'), copy=None, indicator=False, validate=None". My assignment instructions read "Using Python, import the data and combine both into a single DataFrame. Note, both files have the column HospitalID. Use this column to merge the files. That is, each employee has a unique position at one or more hospitals; these positions need to be combined with the correct hospital. Thus, each hospital has one unique employee listed in the data, but an employee can be assigned to more than one hospital." One thing I'm confused on is how a txt file can have a "column"?

Reasoned for 6 seconds

A file with a ".txt" extension can absolutely have columns as long as its contents are structured (for example, using tabs, commas, or another delimiter to separate the values). In this context, even though the file is called a TXT file, it's likely formatted in a delimited way—similar to a CSV—so that pandas can parse the content into a DataFrame with columns.

When you use a function such as `pd.read_csv()` or `pd.read_table()` with the appropriate delimiter (e.g., `delimiter='\t'` for tab-delimited), pandas reads the first line as the header (unless specified otherwise), creating column names for your DataFrame. Thus, if your text file includes a header row and well-organized data rows, it will have columns just like any other table.

After you load both files into DataFrames, you can merge them using the `merge()` method on a common column, in your case `HospitalID`. This way, even though one file is a CSV and the other is a TXT, as long as they are both structured, you can treat them similarly for data manipulation purposes.

Looking at my tutorial instructions, it seems perhaps I should first load the CSV, and then in a subsequent cell load the txt file via the merge command?

Reasoned for 7 seconds

Yes, that’s a common and recommended approach. Typically, you’d first load your CSV file into a DataFrame and then, in a subsequent cell, load your TXT file (using the appropriate delimiter) into another DataFrame. Once you have both DataFrames, you can merge or join them using the `merge()` method on the common column, which is `HospitalID` in your case.

A suggested workflow could be:

1. **Load the CSV:**
   Read in your CSV file using `pd.read_csv()` to create your first DataFrame.
2. **Load the TXT File:**
   Use `pd.read_csv()` or `pd.read_table()` (with the correct delimiter) to read in your TXT file into a second DataFrame.

3. **Merge the DataFrames:**
   With both DataFrames available, merge them using the `merge()` method on `HospitalID`.

This way, you keep your workflow modular and easier to debug if needed, and you ensure that each DataFrame is fully read and inspected before performing the merge.

this was my second cell and i got an error: "CA_Hospital_Personnel = pd.read_csv("/Users/AlexMcKenzie/Library/CloudStorage/GoogleDrive-alex.c.mckenzie@gmail.com/My Drive/Education/UM Graduate School/BGEN632/Week 7/data/CaliforniaHospitalData_Personnel.txt", delimiter='\t')
DataFrame.merge(on = 'HospitalID')"

Reasoned about merge operation and errors for 16 seconds

The error is occurring because you're trying to call the merge method on the DataFrame class itself rather than on an instance of a DataFrame. In your second cell, you successfully loaded the TXT file into the variable `CA_Hospital_Personnel`, and presumably you have your CSV data loaded into the variable `CA_Hospital`. To merge these two DataFrames using the common column "HospitalID", you need to call `merge()` on one of them and pass the other as an argument. For example:

python Copy

```python
merged_df = CA_Hospital.merge(CA_Hospital_Personnel, on='HospitalID')
```

Alternatively, you can use the pandas function `pd.merge()`:

python Copy

```python
merged_df = pd.merge(CA_Hospital, CA_Hospital_Personnel,
on='HospitalID')
```

Either of these approaches will merge the two DataFrames on the specified column. Remember, the error arose because of calling `DataFrame.merge(on='HospitalID')` without specifying which DataFrame instance to merge.

Next i have to drop some unnecessary columns. I think I know how to use the drop command for named columns, with "CA_Hospital.drop(['Word_ID', 'PositionID', 'Website'], axis = 1)", but I'm not sure how to drop duplicate columns?

Reasoned about dropping duplicate columns for 18 seconds >

When you merge two DataFrames, sometimes you can end up with duplicate columns—especially if both DataFrames have columns (other than the merge key) with the same names. In such cases, you can drop duplicate columns by selecting only the first occurrence of each column name.

A common method is to use the DataFrame's column indexer along with the duplicated function. For example:

python Copy

```python
# This line selects only the first occurrence of each column name.
merged_df = merged_df.loc[:, ~merged_df.columns.duplicated()]
```

Here's a breakdown of what’s happening:

1. `merged_df.columns.duplicated()` creates a Boolean array where each element is `True` if the column name has already been encountered.
2. Using `~` (the negation operator) inverts this Boolean array so that the first occurrence of a column name is marked as `True` (to keep), while duplicates are marked as `False`.
3. Finally, `merged_df.loc[:, ...]` selects all rows and only those columns for which the condition is `True`.

This method ensures that your DataFrame contains only one instance of each column name, effectively dropping any duplicates.

If you have additional columns to drop by name—such as `Work_ID`, `PositionID`, and `Website`—you can chain the drop command as follows:

python Copy

```python
merged_df = merged_df.drop(['Work_ID', 'PositionID', 'Website'],
axis=1) merged_df = merged_df.loc[:, ~merged_df.columns.duplicated()]
```

This way, you remove both the specifically named columns and any duplicate columns that might remain.

for cell 8 i wrote "# Here I got the first line of code myself, but needed help in dropping duplicate columns.

CA_Hospital.drop(['Word_ID', 'PositionID', 'Website'], axis = 1)
merged_df = merged_df.loc[:,

```
~merged_df.columns.duplicated()]" And got the
response "KeyError                              Traceback (most
recent call last)
Cell In[8], line 3
      1 # Here I got the first line of code myself, but
needed help in dropping duplicate columns.
----> 3 CA_Hospital.drop(['Word_ID', 'PositionID',
'Website'], axis = 1)
      4 merged_df = merged_df.loc[:,
~merged_df.columns.duplicated()]

File /opt/anaconda3/lib/python3.12/site-
packages/pandas/core/frame.py:5581, in
DataFrame.drop(self, labels, axis, index, columns, level,
inplace, errors)
   5433 def drop(
   5434     self,
   5435     labels: IndexLabel | None = None,
   (...)
   5442     errors: IgnoreRaise = "raise",
   5443 ) -> DataFrame | None:
   5444     """
   5445     Drop specified labels from rows or columns.
   5446
   (...)
   5579             weight  1.0     0.8
   5580     """
-> 5581     return super().drop(
   5582         labels=labels,
   5583         axis=axis,
   5584         index=index,
   5585         columns=columns,
   5586         level=level,
   5587         inplace=inplace,
   5588         errors=errors,
   5589     )

File /opt/anaconda3/lib/python3.12/site-
packages/pandas/core/generic.py:4788, in
NDFrame.drop(self, labels, axis, index, columns, level,
inplace, errors)
   4786 for axis, labels in axes.items():
   4787     if labels is not None:
-> 4788         obj = obj._drop_axis(labels, axis,
level=level, errors=errors)
   4790 if inplace:
   4791     self._update_inplace(obj)

File /opt/anaconda3/lib/python3.12/site-
packages/pandas/core/generic.py:4830, in
NDFrame._drop_axis(self, labels, axis, level, errors,
only_slice)
   4828         new_axis = axis.drop(labels, level=level,
errors=errors)
   4829     else:
-> 4830         new_axis = axis.drop(labels, errors=errors)
   4831     indexer = axis.get_indexer(new_axis)
   4833 # Case for non-unique axis
   4834 else:

File /opt/anaconda3/lib/python3.12/site-
packages/pandas/core/indexes/base.py:7070, in
Index.drop(self, labels, errors)
   7068 if mask.any():
   7069     if errors != "ignore":
-> 7070         raise KeyError(f"{labels[mask].tolist()} not
found in axis")
   7071     indexer = indexer[~mask]
   7072 return self.delete(indexer)

KeyError: "['Word_ID', 'PositionID'] not found in axis""
```

Reasoned for 21 seconds >

The error indicates that one or more of the column names you specified for dropping aren't present in the DataFrame where you're trying to drop them. Here are a few points to check and adjust:

1. **Check Column Names Carefully:**
   - In your code, you're attempting to drop `"Word_ID"`, but the assignment instructions mention `"Work_ID"`. This typo would cause a KeyError.
   - Verify the actual column names using a command like `print(CA_Hospital.columns)` or `print(merged_df.columns)`.
2. **Drop from the Correct DataFrame:**
   - The additional columns you want to drop— `Work_ID`, `PositionID`, and `Website` —are likely present in the merged DataFrame (since they come from the personnel data).
   - Instead of calling `drop` on `CA_Hospital` (which may not have all these columns), apply the `drop` method to your merged DataFrame.
3. **Using the `errors` Parameter (Optional):**
   - If you want to avoid an error when a column isn't present, you can use the parameter `errors='ignore'`:

```python
merged_df = merged_df.drop(['Work_ID', 'PositionID', 'Website'],
axis=1, errors='ignore')
```

Here’s how you could adjust your code:

```python
# Correct column names and apply drop on the merged DataFrame.
merged_df = merged_df.drop(['Work_ID', 'PositionID', 'Website'],
axis=1, errors='ignore') # Then drop duplicate columns. merged_df =
merged_df.loc[:, ~merged_df.columns.duplicated()]
```

By ensuring you're using the correct column names (e.g., "Work_ID" instead of "Word_ID") and applying the drop to the merged DataFrame, you should avoid the KeyError.

how do I apply the drop method to your merged DataFrame instead of CA_Hospital?

Reasoned for 7 seconds >

To apply the drop method to the merged DataFrame instead of CA_Hospital, you should call the drop method on the variable that holds your merged data (e.g., merged_df). For instance, if you merged your CSV and TXT data like this:

```python
merged_df = CA_Hospital.merge(CA_Hospital_Personnel, on='HospitalID')
```

Then you can drop the unnecessary columns from merged_df by doing:

```python
merged_df = merged_df.drop(['Work_ID', 'PositionID', 'Website'],
axis=1, errors='ignore')
```

This command removes the specified columns from merged_df.
The `errors='ignore'` parameter helps avoid errors if any of the columns are not present in the DataFrame.

Finally, if you want to remove duplicate columns, you can chain that command as follows:

```python
merged_df = merged_df.loc[:, ~merged_df.columns.duplicated()]
```

This way, you’re applying the drop method directly to your merged DataFrame.

how can i see the# For the next step, first I ran the following, all without ChatGPT:

# merged_df[(merged_df.TypeControl == 'Small/Rural') & (merged_df.AvlBeds >= 15)]

# This returned my column names and "0 rows × 22 columns", so something's not right... all the data?

Reasoned for 9 seconds >

When filtering your DataFrame and getting 0 rows, it might mean that none of the rows match both conditions, or there could be an issue with how the conditions are written. To diagnose the problem, here are some steps you can take:

1. **Inspect the Entire DataFrame:**
   Instead of filtering, view a sample of your data to see the actual values. For example, you can run:

   python Copy

   ```python
   merged_df.head(10)
   ```

   This will display the first 10 rows of your DataFrame so you can see what values are in the `TypeControl` and `AvlBeds` columns.

2. **Examine Unique Values:**
   Check the unique values present in the `TypeControl` column. It's possible the value isn't exactly `'Small/Rural'` (e.g., it could be spelled differently, have extra spaces, or use different casing):

   python Copy

   ```python
   print(merged_df['TypeControl'].unique())
   ```

   This will help you confirm whether there are any rows with exactly `'Small/Rural'` or if you might need to adjust your condition.

3. **Check the Numerical Data:**
   Verify that the `AvlBeds` column is numeric. If it's stored as a string, then the condition (`>= 15`) may not work as expected. You can check the data type by running:

   python Copy

   ```python
   print(merged_df['AvlBeds'].dtype)
   ```

   If necessary, you can convert it to a numeric type using `pd.to_numeric()` .

4. **Test Conditions Separately:**
   It often helps to test one condition at a time. For instance, first check how many rows meet the bed condition:

   ```python
   print(merged_df[merged_df['AvlBeds'] >= 15].shape)
   ```

   Then test the `TypeControl` condition:

   ```python
   print(merged_df[merged_df['TypeControl'] == 'Small/Rural'].shape)
   ```

   This will let you know if one of the conditions is filtering out all the data, or if both have non-zero counts individually but not when combined.

By following these steps, you should be able to identify whether the issue is with the filter condition itself or with how the data is formatted. Once you've identified the problem (for example, if the value in `TypeControl` is slightly different), you can adjust your filter accordingly.

the two tests returned: "(60, 22)
(0, 22)"

Reasoned for 14 seconds

The results indicate that while all 60 rows meet the bed condition (AvlBeds ≥ 15), none of the rows have a TypeControl value that exactly matches "Small/Rural". This suggests one of two possibilities:

1. **Different Value in TypeControl:**
   The actual values in the TypeControl column might be different from "Small/Rural". It could be due to alternate spellings, extra spaces, or even different terminology. To confirm the actual values, run:

   ```python
   print(merged_df['TypeControl'].unique())
   ```

   This command will show you all unique values present in the TypeControl column. Look carefully to see if there is a value that should be considered equivalent to "Small/Rural".

2. **Data Formatting Issues:**
   Sometimes, extra spaces or differences in case can prevent an exact match. For example, a value like `" Small/Rural"` (with a leading space)

or `"small/rural"` (all lowercase) won't match `"Small/Rural"`. You can clean or adjust the condition by stripping white space and lowering the case:

```python
filtered_df = merged_df[
(merged_df['TypeControl'].str.strip().str.lower() == 'small/rural')
& (merged_df['AvlBeds'] >= 15) ] print(filtered_df.shape)
```

This approach ensures that minor formatting differences won't cause the condition to fail.

By checking the unique values in the TypeControl column and possibly adjusting your filtering criteria, you should be able to correctly isolate the rows that match your intended filter.

ok, i realized i was referring to the wrong column name. But now that I'm getting results, I'm stuck on my next question: To exclude hospitals that have a negative operating income, with the given column header 'OperInc', though I'm not seeing that here: "HospitalID Name Zip TypeControl Teaching DonorType NoFTE NetPatRev InOperExp OutOperExp ... AvlBeds LastName FirstName Gender PositionTitle Compensation MaxTerm StartDate Phone Email"

Reasoned for 25 seconds

If you're not seeing the 'OperInc' column in your merged DataFrame, here are some steps to diagnose and resolve the issue:

1. **Inspect All Column Names:**
   First, check exactly which columns exist in your DataFrame. Run:

   ```python
   print(merged_df.columns)
   ```

   This will list every column, so you can verify whether 'OperInc' appears or if a similar column (perhaps with a slightly different name) exists.

2. **Review the Original CSV File:**
   Confirm that your CSV file (`CaliforniaHospitalData.csv`) actually contains a column named 'OperInc'. It's possible that the column name might be different (for example, it might be misspelled, or maybe it's named something like 'Operating_Income'). Opening the CSV in a text editor or using:

   ```python
   ```

```
df = pd.read_csv("path/to/CaliforniaHospitalData.csv")
print(df.columns)
```

can help verify this.

3. **Check the Merge Process:**
   When you merged the DataFrames, if both had columns with the same name, pandas appends suffixes (like `'_x'` or `'_y'`).
   - For example, if both DataFrames contained an 'OperInc' column, you might have ended up with 'OperInc_x' and 'OperInc_y'.
   - If you then dropped duplicate columns, one of them might have been removed depending on your column selection. Verify if any column that looks like 'OperInc_x' or 'OperInc_y' exists.
4. **Filter Based on the Correct Column:**
   Once you identify the correct column name for operating income (whether it's 'OperInc' or something similar), you can exclude hospitals with a negative operating income. For example, if the correct name is 'OperInc', you would do:

   python Copy

   ```python
   merged_df = merged_df[merged_df['OperInc'] >= 0]
   ```
5. **Consider Recomputing Operating Income:**
   If the column is truly missing from your data, check whether you have the ingredients (such as operating revenue and operating expenses) to calculate it. According to your assignment instructions, operating income is defined as operating revenue less operating expenses. If you have, say, an 'OperRev' column along with expense columns like 'InOperExp' and 'OutOperExp', you might need to compute it with:

   python Copy

   ```python
   # Example calculation; adjust column names as necessary.
   merged_df['OperInc'] = merged_df['OperRev'] -
   (merged_df['InOperExp'] + merged_df['OutOperExp']) # Now filter the
   DataFrame: merged_df = merged_df[merged_df['OperInc'] >= 0]
   ```

By taking these steps, you should be able to locate the operating income data correctly, whether it exists in the file or needs to be computed, and then filter out hospitals with negative values accordingly.

i'm trying to exclude hospitals with a negative operating income and this isn't working, what am i doing wrong? "merged_df.drop(['OperInc' <= '0')"

Reasoned for 4 seconds >

The `drop` method is used to remove columns or rows by their labels or indices, not for filtering based on a condition. When you write `['OperInc' <= '0']`, you're comparing a string literal (`'OperInc'`) to `'0'`, which isn't the intended comparison, and the syntax is also incorrect.

To filter out (exclude) hospitals with a negative operating income, you should use boolean indexing. For example, if you want to keep only the rows where the `OperInc` column has a value greater than or equal to zero, you can do:

```python
merged_df = merged_df[merged_df['OperInc'] >= 0]
```

This code creates a boolean mask where the condition `merged_df['OperInc'] >= 0` is `True` for rows you want to keep, and then uses that mask to filter `merged_df`.

So, instead of using `drop`, use boolean indexing to filter the rows that meet your condition.

Next I am to "Export your data as tab-delimited and name the file `hospital_data_new.txt`. Save this to the directory `/Users/AlexMcKenzie/Library/CloudStorage/GoogleDrive-alex.c.mckenzie@gmail.com/My Drive/Education/UM Graduate School/BGEN632/Week 7/data`. Be sure to not export the index using the argument `index = False` for the pandas' function `to_csv()`"

Reasoned for 14 seconds >

To export your merged DataFrame as a tab-delimited text file and ensure that the index is not included, you'll use the DataFrame's `to_csv()` method. The key points are:

1. **File Path:**
   Save the file to the specified directory with the name `hospital_data_new.txt`.
2. **Delimiter:**
   Use the `sep='\t'` parameter to indicate that the file should be tab-delimited.
3. **Index Exclusion:**
   Use `index=False` so that the row labels (index) are not written to the file.

The command would look something like this:

```python
```

```
merged_df.to_csv("/Users/AlexMcKenzie/Library/CloudStorage/GoogleDrive-
alex.c.mckenzie@gmail.com/My Drive/Education/UM Graduate
School/BGEN632/Week 7/data/hospital_data_new.txt", sep='\t',
index=False)
```

This line of code tells pandas to write `merged_df` to a file at your specified location, using tabs as the delimiter and leaving out the index column. Make sure that the DataFrame `merged_df` contains the processed data you want to export before running this command.

Select two of the existing hospitals in the newly created DataFrame and create a new position for each hospital. Insert yourself as the new employee at those two hospitals; put in your first name and last name. Put today’s date as the start date. Select one of the positions as shown in the table below and fill out the data accordingly. Fill in the rest of the columns as you choose. Use the method `concat()`. You should have two new rows of data. Save the DataFrame as `new_merge`.

Reasoned about adding new employee records for 18 seconds >

You can think of the process in two parts:

1. **Create a DataFrame for the Two New Rows:**
   First, select the two hospitals you want to add new personnel records to (for example, by extracting their HospitalID values from your merged DataFrame). Then, create a new DataFrame with two rows. In this DataFrame, include all the columns that your merged DataFrame contains. For the personnel-related columns, you’ll set:

   - Your first and last name (for example, `'Alex'` and `'McKenzie'`).
   - For the start date, use today's date (you can generate it with Python’s datetime functions).
   - For the position, choose one of the positions from the provided table (e.g., `'Regional Representative'` with its associated compensation and max term).

   The values you don’t need to worry about as much (like possibly employee IDs if they are generated) can be filled with placeholder values or adjusted as needed.

2. **Combine This DataFrame with the Existing One Using `concat()`:**
   With the two-row DataFrame ready, you can use pandas’ `concat()` function to add those rows to your existing merged DataFrame. For example, if your current DataFrame is called `merged_df` and your new DataFrame (with the two new

personnel entries) is called `new_entries` , you would do something like this conceptually:

python Copy

```
new_merge = pd.concat([merged_df, new_entries], ignore_index=True)
```

Here, `ignore_index=True` reassigns a new sequential index after concatenation, which is often convenient for a new merged DataFrame.

In summary, you will:

- Identify two hospitals from your existing data.
- Manually create two rows (as a new DataFrame) with your personalized personnel information and any other required column values.
- Use `pd.concat()` to append these new rows to the original DataFrame, saving the result as `new_merge` .

Let me know if you would like more detailed guidance or need help writing a specific part of the code!

just give me the code for this one

Reasoned for 24 seconds >

python Copy

```
import pandas as pd from datetime import date # Assuming 'merged_df' is
your DataFrame that you created and saved from previous steps # Select
two existing hospitals – here we take the first two rows as examples
selected_hospitals = merged_df.iloc[[0, 1]].copy() # For each selected
hospital, update the personnel details to add a new entry for yourself
# (Feel free to adjust other columns as needed for your assignment)
selected_hospitals.loc[:, 'FirstName'] = 'Alex'
selected_hospitals.loc[:, 'LastName'] = 'McKenzie'
selected_hospitals.loc[:, 'PositionTitle'] = 'Regional Representative'
selected_hospitals.loc[:, 'Compensation'] = 46978
selected_hospitals.loc[:, 'MaxTerm'] = 4 selected_hospitals.loc[:,
'StartDate'] = date.today().strftime('%Y-%m-%d') # Concatenate the new
rows with the original merged DataFrame new_merge =
pd.concat([merged_df, selected_hospitals], ignore_index=True)
```

let's do it in these hospitals
[(row),HospitalID,Name,Zip]: 21 37436
Fallbrook Hospital 92028, 31 29805
Alameda Hospital 94501

Reasoned for 15 seconds >

Below is an example code snippet that selects the two hospitals using their HospitalID values (37436 for Fallbrook Hospital and 29805 for Alameda Hospital), updates the personnel fields to insert yourself as the new employee with the 'Regional Representative' position, and then concatenates these new rows to the existing merged DataFrame. Make sure your DataFrame `merged_df` is already defined from your previous steps:

python Copy

```python
import pandas as pd from datetime import date # Select the hospitals
with HospitalID 37436 and 29805. selected_hospitals =
merged_df[merged_df['HospitalID'].isin([37436, 29805])].copy() # Update
the personnel information for the new entries.
selected_hospitals.loc[:, 'FirstName'] = 'Alex'
selected_hospitals.loc[:, 'LastName'] = 'McKenzie'
selected_hospitals.loc[:, 'PositionTitle'] = 'Regional Representative'
selected_hospitals.loc[:, 'Compensation'] = 46978
selected_hospitals.loc[:, 'MaxTerm'] = 4 selected_hospitals.loc[:,
'StartDate'] = date.today().strftime('%Y-%m-%d') # Concatenate these
new rows with the original merged DataFrame. new_merge =
pd.concat([merged_df, selected_hospitals], ignore_index=True)
```

This code does the following:

- **Filtering:** Selects rows from `merged_df` where the `HospitalID` is either 37436 or 29805.
- **Updating:** Replaces the `FirstName`, `LastName`, `PositionTitle`, `Compensation`, `MaxTerm`, and `StartDate` fields for the new entries.
- **Concatenation:** Uses `pd.concat()` to add the new rows to your original DataFrame, resulting in a new DataFrame called `new_merge`.

You can adjust additional fields as needed to complete your assignment requirements.

it said "

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Cell In[28], line 9
      7 selected_hospitals.loc[:, 'Compensation'] = 46978
      8 selected_hospitals.loc[:, 'MaxTerm'] = 4
----> 9 selected_hospitals.loc[:, 'StartDate'] = date.today().strftime('%Y-%m-%d')
     10 new_merge = pd.concat([merged_df, selected_hospitals], ignore_index=True)
```

NameError: name 'date' is not defined"

Reasoned for 5 seconds >

The error indicates that Python doesn't know what `date` is because it's not been imported. To fix this, you need to add an import statement at the top of your code. Here's the revised code with the necessary import:

python Copy

```
import pandas as pd from datetime import date # Import date from the
datetime module # Select hospitals with HospitalID 37436 (Fallbrook
Hospital) and 29805 (Alameda Hospital) selected_hospitals =
merged_df[merged_df['HospitalID'].isin([37436, 29805])].copy() # Update
the personnel information for the new entries.
selected_hospitals.loc[:, 'FirstName'] = 'Alex'
selected_hospitals.loc[:, 'LastName'] = 'McKenzie'
selected_hospitals.loc[:, 'PositionTitle'] = 'Regional Representative'
selected_hospitals.loc[:, 'Compensation'] = 46978
selected_hospitals.loc[:, 'MaxTerm'] = 4 selected_hospitals.loc[:,
'StartDate'] = date.today().strftime('%Y-%m-%d') # Concatenate these
new rows with the original merged DataFrame. new_merge =
pd.concat([merged_df, selected_hospitals], ignore_index=True)
```

Make sure that `merged_df` is already defined in your notebook from previous steps. This updated code will now import `date` correctly, which should resolve the `NameError`.

Using the new_merge data, select all hospitals that are non-profit with more than 250 employees, unless the net patient revenue is smaller than $109,000. Remove the columns containing employee information and save it as a new DataFrame.
* Using the new_merge data, select all the *Regional Representatives* who work at a hospital with operating income greater than $100,000. Save this as a new DataFrame.

Reasoned for 28 seconds >

Below is an example of how you can create the two new DataFrames from your `new_merge` DataFrame. Adjust the column names if needed to match your actual data.

1. Hospitals that are Non-profit, Have More Than 250 Employees, and Net Patient Revenue ≥ $109,000

In this first DataFrame, we're filtering based on those conditions and then removing the columns that hold employee information (for example: last name, first name, etc.). In this example, I assumed that `"TypeControl"` contains the value `'Non-profit'` for non-profit hospitals, and that the employee-related columns are:

- `'LastName'`
- `'FirstName'`
- `'Gender'`
- `'PositionTitle'`
- `'Compensation'`
- `'MaxTerm'`
- `'StartDate'`
- `'Phone'`
- `'Email'`

```python
# List of columns with employee information employee_cols =
['LastName', 'FirstName', 'Gender', 'PositionTitle', 'Compensation',
'MaxTerm', 'StartDate', 'Phone', 'Email'] # Filter for non-profit
hospitals with more than 250 employees and NetPatRev >= 109000. #
Adjust the condition for 'Non-profit' if your actual data uses a
different string. df_nonprofit = new_merge[ (new_merge['TypeControl']
== 'Non-profit') & (new_merge['NoFTE'] > 250) & (new_merge['NetPatRev']
>= 109000) ] # Drop the employee-related columns df_nonprofit =
df_nonprofit.drop(columns=employee_cols, errors='ignore')
```

2. Regional Representatives at Hospitals with Operating Income Greater Than $100,000

In the second DataFrame, we select rows where the `PositionTitle` equals `"Regional Representative"` and the operating income (assumed to be in the column `"OperInc"`) is greater than $100,000.

```python
df_regional = new_merge[ (new_merge['PositionTitle'] == 'Regional
Representative') & (new_merge['OperInc'] > 100000) ]
```

Explanation

- **Filtering:**
    - In the first block, the conditions check for non-profit hospitals (`TypeControl == 'Non-profit'`), that have more than 250 employees (`NoFTE > 250`), and net patient revenue of at least $109,000 (`NetPatRev >= 109000`).

  - In the second block, we filter based on the personnel's `PositionTitle` and the hospital's operating income (`OperInc > 100000`).
- **Removing Columns:**
  - The `drop()` method removes all employee-related columns from the first filtered DataFrame (`df_nonprofit`).

Be sure to examine your DataFrame's actual column names and values to make necessary adjustments if they differ from this example.

Using the `new_merge` DataFrame, convert any date-time columns into a datetime datatype. Confirm your changes by outputting the data types for all columns. Additionally, output the first 5 records of the column(s) you converted.

Save your notebook with output displayed within it and submit for grading.

Reasoned for 14 seconds ›

Below is an example code snippet that converts any date-time columns in your `new_merge` DataFrame (in this case, we assume that the `StartDate` column contains date-time data) into a datetime datatype. The code then prints out the data types of all columns and shows the first 5 records of the converted column(s):

```python
import pandas as pd # Option 1: If you only have a specific date-time
column, such as StartDate: new_merge['StartDate'] =
pd.to_datetime(new_merge['StartDate']) # Verify the conversion: Print
the data types of all columns print(new_merge.dtypes) # Print the first
5 records of the StartDate column print(new_merge['StartDate'].head(5))
# Option 2: If you want to automate for any column that contains "Date"
in its name: date_cols = [col for col in new_merge.columns if "Date" in
col] # Convert each date-like column into datetime for col in
date_cols: new_merge[col] = pd.to_datetime(new_merge[col]) # Confirm
the data types after conversion print(new_merge.dtypes) # Print the
first 5 records of each date-time column converted for col in
date_cols: print(f"First 5 records of {col}:")
print(new_merge[col].head(5))
```

Explanation

- **Conversion to Datetime:**
  The `pd.to_datetime()` function is used to convert a column (or columns) from its current type to a datetime format.

- **Verifying Data Types:**
  `new_merge.dtypes` prints out the datatype for every column so you can confirm that your date columns are now of type `datetime64[ns]`.
- **Output First 5 Records:**
  Using the `.head(5)` method, the code prints the first five entries of the converted date column(s).

Remember to run this after your `new_merge` DataFrame has been created. Also, if you have additional date-time columns, Option 2 automatically converts and displays those.

Save your notebook with output displayed within it before submitting for grading.

i am short on time, just give me the cell code please

Reasoned for 11 seconds >

python

Copy

```python
# Convert the 'StartDate' column to datetime new_merge['StartDate'] =
pd.to_datetime(new_merge['StartDate']) # Confirm the changes by
printing the data types of all columns print(new_merge.dtypes) # Output
the first 5 records of the 'StartDate' column
print(new_merge['StartDate'].head(5))
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent
call last)
Cell In[34], line 1
----> 1 new_merge['StartDate'] =
pd.to_datetime(new_merge['StartDate'])

File /opt/anaconda3/lib/python3.12/site-
packages/pandas/core/tools/datetimes.py:1067, in
to_datetime(arg, errors, dayfirst, yearfirst, utc, format,
exact, unit, infer_datetime_format, origin, cache)
   1065         result = arg.map(cache_array)
   1066     else:
-> 1067         values = convert_listlike(arg._values,
format)
   1068         result = arg._constructor(values,
index=arg.index, name=arg.name)
   1069 elif isinstance(arg, (ABCDataFrame,
abc.MutableMapping)):

File /opt/anaconda3/lib/python3.12/site-
```

```
packages/pandas/core/tools/datetimes.py:433, in
_convert_listlike_datetimes(arg, format, name, utc, unit,
errors, dayfirst, yearfirst, exact)
    431 # format could be inferred, or user didn't ask for
mixed-format parsing.
    432 if format is not None and format != "mixed":
--> 433     return _array_strptime_with_fallback(arg,
name, utc, format, exact, errors)
    435 result, tz_parsed = objects_to_datetime64(
    436     arg,
    437     dayfirst=dayfirst,
   (...)
    441     allow_object=True,
    442 )
    444 if tz_parsed is not None:
    445     # We can take a shortcut since the datetime64
numpy array
    446     # is in UTC

File /opt/anaconda3/lib/python3.12/site-
packages/pandas/core/tools/datetimes.py:467, in
_array_strptime_with_fallback(arg, name, utc, fmt,
exact, errors)
    456 def _array_strptime_with_fallback(
    457     arg,
    458     name,
   (...)
    462     errors: str,
    463 ) -> Index:
    464     """
    465     Call array_strptime, with fallback behavior
depending on 'errors'.
    466     """
--> 467     result, tz_out = array_strptime(arg, fmt,
exact=exact, errors=errors, utc=utc)
    468     if tz_out is not None:
    469         unit = np.datetime_data(result.dtype)[0]

File strptime.pyx:501, in
pandas._libs.tslibs.strptime.array_strptime()

File strptime.pyx:451, in
pandas._libs.tslibs.strptime.array_strptime()

File strptime.pyx:583, in
pandas._libs.tslibs.strptime._parse_with_format()

ValueError: time data "2025-04-13" doesn't match
format "%m/%d/%Y", at position 41. You might want to
try:
    - passing format if your strings have a consistent
```

format;
  - passing `format='ISO8601'` if your strings are all ISO8601 but not necessarily in exactly the same format;
  - passing `format='mixed'`, and the format will be inferred for each element individually. You might want to use `dayfirst` alongside this.

Reasoned for 21 seconds >

The error occurs because the conversion function is expecting dates in the format `%m/%d/%Y` , but your dates (like `"2025-04-13"`) are in ISO format (`%Y-%m-%d`). To fix this, you can specify the correct format when converting. For example:

python

```python
# Convert the 'StartDate' column to datetime using the ISO format
new_merge['StartDate'] = pd.to_datetime(new_merge['StartDate'],
format='%Y-%m-%d') # Confirm the changes by printing the data types of
all columns print(new_merge.dtypes) # Output the first 5 records of the
'StartDate' column print(new_merge['StartDate'].head(5))
```

This code explicitly tells pandas that the dates are in the `YYYY-MM-DD` format, which should resolve the conversion error.

your first line of code returned: "

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
Cell In[35], line 1
----> 1 new_merge['StartDate'] = pd.to_datetime(new_merge['StartDate'], format='%Y-%m-%d')

File /opt/anaconda3/lib/python3.12/site-packages/pandas/core/tools/datetimes.py:1067, in to_datetime(arg, errors, dayfirst, yearfirst, utc, format, exact, unit, infer_datetime_format, origin, cache)
   1065         result = arg.map(cache_array)
   1066     else:
-> 1067         values = convert_listlike(arg._values, format)
   1068         result = arg._constructor(values, index=arg.index, name=arg.name)
   1069 elif isinstance(arg, (ABCDataFrame, abc.MutableMapping)):

File /opt/anaconda3/lib/python3.12/site-packages/pandas/core/tools/datetimes.py:433, in
```

```
_convert_listlike_datetimes(arg, format, name, utc, unit,
errors, dayfirst, yearfirst, exact)
    431 # format could be inferred, or user didn't ask for
mixed-format parsing.
    432 if format is not None and format != "mixed":
--> 433     return _array_strptime_with_fallback(arg,
name, utc, format, exact, errors)
    435 result, tz_parsed = objects_to_datetime64(
    436     arg,
    437     dayfirst=dayfirst,
   (...)
    441     allow_object=True,
    442 )
    444 if tz_parsed is not None:
    445     # We can take a shortcut since the datetime64
numpy array
    446     # is in UTC

File /opt/anaconda3/lib/python3.12/site-
packages/pandas/core/tools/datetimes.py:467, in
_array_strptime_with_fallback(arg, name, utc, fmt,
exact, errors)
    456 def _array_strptime_with_fallback(
    457     arg,
    458     name,
   (...)
    462     errors: str,
    463 ) -> Index:
    464     """
    465     Call array_strptime, with fallback behavior
depending on 'errors'.
    466     """
--> 467     result, tz_out = array_strptime(arg, fmt,
exact=exact, errors=errors, utc=utc)
    468     if tz_out is not None:
    469         unit = np.datetime_data(result.dtype)[0]

File strptime.pyx:501, in
pandas._libs.tslibs.strptime.array_strptime()

File strptime.pyx:451, in
pandas._libs.tslibs.strptime.array_strptime()

File strptime.pyx:583, in
pandas._libs.tslibs.strptime._parse_with_format()

ValueError: time data "1/1/2009" doesn't match format
"%Y-%m-%d", at position 0. You might want to try:
    - passing format if your strings have a consistent
format;
    - passing format='ISO8601' if your strings are all
```

ISO8601 but not necessarily in exactly the same format;
  - passing `format='mixed'`, and the format will be inferred for each element individually. You might want to use `dayfirst` alongside this."

Reasoned for 14 seconds >

Since your "StartDate" column contains dates in more than one format (for example, some values like "1/1/2009" and others like "2025-04-13"), specifying a single format (such as `%Y-%m-%d`) won't work. Instead, you can let pandas automatically infer the format by using the `infer_datetime_format=True` argument and removing the format parameter altogether.

Try this cell:

```python
# Convert the 'StartDate' column to datetime using automatic inference
new_merge['StartDate'] = pd.to_datetime(new_merge['StartDate'],
infer_datetime_format=True) # Confirm the conversion by printing the
datatypes of all columns print(new_merge.dtypes) # Output the first 5
records of the 'StartDate' column print(new_merge['StartDate'].head(5))
```

This should correctly convert all date strings in "StartDate" to datetime objects regardless of their original format.

```
"/var/folders/xt/1ps45zhn76962p4cclc2wk0h0000gn/T/i
pykernel_35521/1833757124.py:1: UserWarning: The
argument 'infer_datetime_format' is deprecated and will
be removed in a future version. A strict version of it is
now the default, see
https://pandas.pydata.org/pdeps/0004-consistent-to-
datetime-parsing.html. You can safely remove this
argument.
  new_merge['StartDate'] =
pd.to_datetime(new_merge['StartDate'],
infer_datetime_format=True)
----------------------------------------------------------
------------------
ValueError                         Traceback (most recent
call last)
Cell In[36], line 1
----> 1 new_merge['StartDate'] =
pd.to_datetime(new_merge['StartDate'],
infer_datetime_format=True)

File /opt/anaconda3/lib/python3.12/site-
packages/pandas/core/tools/datetimes.py:1067, in
to_datetime(arg, errors, dayfirst, yearfirst, utc, format,
exact, unit, infer_datetime_format, origin, cache)
   1065         result = arg.map(cache_array)
   1066     else:
-> 1067         values = convert_listlike(arg._values,
format)
   1068         result = arg._constructor(values,
index=arg.index, name=arg.name)
   1069 elif isinstance(arg, (ABCDataFrame,
abc.MutableMapping)):

File /opt/anaconda3/lib/python3.12/site-
packages/pandas/core/tools/datetimes.py:433, in
_convert_listlike_datetimes(arg, format, name, utc, unit,
errors, dayfirst, yearfirst, exact)
    431 # format could be inferred, or user didn't ask for
mixed-format parsing.
    432 if format is not None and format != "mixed":
--> 433     return _array_strptime_with_fallback(arg,
name, utc, format, exact, errors)
    435 result, tz_parsed = objects_to_datetime64(
    436     arg,
    437     dayfirst=dayfirst,
   (...)
    441     allow_object=True,
    442 )
    444 if tz_parsed is not None:
    445     # We can take a shortcut since the datetime64
numpy array
    446     # is in UTC

File /opt/anaconda3/lib/python3.12/site-
packages/pandas/core/tools/datetimes.py:467, in
_array_strptime_with_fallback(arg, name, utc, fmt,
exact, errors)
    456 def _array_strptime_with_fallback(
    457     arg,
    458     name,
   (...)
    462     errors: str,
    463 ) -> Index:
    464     """
    465     Call array_strptime, with fallback behavior
depending on 'errors'.
    466     """
--> 467     result, tz_out = array_strptime(arg, fmt,
exact=exact, errors=errors, utc=utc)
    468     if tz_out is not None:
    469         unit = np.datetime_data(result.dtype)[0]

File strptime.pyx:501, in
pandas._libs.tslibs.strptime.array_strptime()
```

File strptime.pyx:451, in pandas._libs.tslibs.strptime.array_strptime()

File strptime.pyx:583, in pandas._libs.tslibs.strptime._parse_with_format()

ValueError: time data "2025-04-13" doesn't match format "%m/%d/%Y", at position 41. You might want to try:
- passing `format` if your strings have a consistent format;
- passing `format='ISO8601'` if your strings are all ISO8601 but not necessarily in exactly the same format;
- passing `format='mixed'`, and the format will be inferred for each element individually. You might want to use `dayfirst` alongside this."

Reasoned for 31 seconds >

Try converting the date column with the `errors='coerce'` parameter, which will convert any unparseable date to `NaT` (Not a Time). Once converted, you can inspect if there are any `NaT` values to ensure everything parsed correctly. Here's the cell code:

python Copy

```python
# Convert the 'StartDate' column to datetime, coercing errors to NaT if
any value doesn't match expected formats new_merge['StartDate'] =
pd.to_datetime(new_merge['StartDate'], errors='coerce') # Confirm the
conversion by printing the data types of all columns
print(new_merge.dtypes) # Output the first 5 records of the 'StartDate'
column print(new_merge['StartDate'].head(5))
```

If you find that some entries become `NaT` unexpectedly, you can investigate further by checking those specific rows.