Exercise1

Performing an anti join

In our music streaming company dataset, each customer is assigned an employee representative to assist them. In this exercise, filter the employee table by a table of top customers, returning only those employees who are not assigned to a customer. The results should resemble the results of an anti join. The company's leadership will assign these employees additional training so that they can work with high valued customers.

The top_cust and employees tables have been provided for you.

Instructions

- Merge employees and top_cust with a left join, setting indicator argument to True. Save the result to empl_cust
- Select the srid column of empl_cust and the rows where _merge is 'left_only'. Save the result to srid_list.
- Subset the employees table and select those rows where the srid is in the variable srid_list and print the results.

In [ ]:

```
# Merge employees and top_cust
empl_cust = ____.merge(____, on=____,
                            how=____, indicator=____)

# Select the srid column where _merge is left_only
srid_list = empl_cust.loc[____, 'srid']

# Get employees not working with top customers
print(employees[____.isin(____)])

#____________________#
#Solutions

# Merge employees and top_cust
empl_cust = employees.merge(top_cust, on='srid',
                            how='left', indicator=True)

# Select the srid column where _merge is left_only
srid_list = empl_cust.loc[empl_cust['_merge']=='left_only', 'srid']


# Get employees not working with top customers
print(employees[employees['srid'].isin(srid_list)])
```

Exercise2

Performing a semi join

Some of the tracks that have generated the most significant amount of revenue are from TV-shows or are other non-musical audio. You have been given a table of invoices that include top revenue-generating items. Additionally, you have a table of non-musical tracks from the streaming service. In this exercise, you'll use a semi join to find the top revenue-generating non-musical tracks..

The tables non_mus_tcks, top_invoices, and genres have been loaded for you.

Instructions

- Merge non_mus_tcks and top_invoices on tid using an inner join. Save the result as tracks_invoices.
- Use .isin() to subset the rows of non_mus_tck where tid is in the tid column of tracks_invoices. Save the result as top_tracks.
- Group top_tracks by gid and count the tid rows. Save the result to cnt_by_gid.
- Merge cnt_by_gid with the genres table on gid and print the result.

In [ ]:

```
# Merge the non_mus_tck and top_invoices tables on tid
tracks_invoices = ____.merge(____)

# Use .isin() to subset non_mus_tcks to rows with tid in tracks_invoices
top_tracks = _____[non_mus_tcks['tid'].isin(____)]

# Group the top_tracks by gid and count the tid rows
cnt_by_gid = top_tracks.groupby(['gid'], as_index=False).agg({'tid':____})

# Merge the genres table to cnt_by_gid on gid and print
print(____)

#_____________________#
#Solutions

# Merge the non_mus_tck and top_invoices tables on tid
tracks_invoices = non_mus_tcks.merge(top_invoices, on='tid')

# Use .isin() to subset non_mus_tcsk to rows with tid in tracks_invoices
top_tracks = non_mus_tcks[non_mus_tcks['tid'].isin(tracks_invoices['tid'])]

# Group the top_tracks by gid and count the tid rows
cnt_by_gid = top_tracks.groupby(['gid'], as_index=False).agg({'tid':'count'})

# Merge the genres table to cnt_by_gid on gid and print
print(cnt_by_gid.merge(genres, on='gid'))
```

Exercise3

Concatenation basics

You have been given a few tables of data with musical track info for different albums from the metal band, Metallica. The track info comes from their Ride The Lightning, Master Of Puppets, and St. Anger albums. Try various features of the .concat() method by concatenating the tables vertically together in different ways.

The tables tracks_master, tracks_ride, and tracks_st have loaded for you.

Instructions

- Concatenate tracks_master, tracks_ride, and tracks_st, in that order, setting sort to True.
- Concatenate tracks_master, tracks_ride, and tracks_st, where the index goes from 0 to n-1.
- Concatenate tracks_master, tracks_ride, and tracks_st, showing only columns that are in all tables.

In [ ]:

```
# Concatenate the tracks
tracks_from_albums = pd.concat(____,
                               sort=True)
print(tracks_from_albums)

#____________________#
#Solutions

# Concatenate the tracks
tracks_from_albums = pd.concat([tracks_master,tracks_ride,tracks_st],
                               sort=True)
print(tracks_from_albums)

# Concatenate the tracks so the index goes from 0 to n-1
tracks_from_albums = pd.concat([tracks_master, tracks_ride, tracks_st],
                               ignore_index=True,
                               sort=True)
print(tracks_from_albums)

# Concatenate the tracks, show only columns names that are in all tables
tracks_from_albums = pd.concat([tracks_master, tracks_ride, tracks_st],
                               join='inner',
                               sort=True)
print(tracks_from_albums)
```

Exercise4

Concatenating with keys

The leadership of the music streaming company has come to you and asked you for assistance in analyzing sales for a recent business quarter. They would like to know which month in the quarter saw the highest average invoice total. You have been given three tables with invoice data named inv_jul, inv_aug, and inv_sep. Concatenate these tables into one to create a graph of the average monthly invoice total.

Instructions

- Concatenate the three tables together vertically in order with the oldest month first, adding '7Jul', '8Aug', and '9Sep' as keys for their respective months, and save to variable avg_inv_by_month.
- Use the .agg() method to find the average of the total column from the grouped invoices.
- Create a bar chart of avg_inv_by_month.

In [ ]:

```
# Concatenate the tables and add keys
inv_jul_thr_sep = pd.concat(____,
                            keys=____)

# Group the invoices by the index keys and find avg of the total column
avg_inv_by_month = inv_jul_thr_sep.groupby(level=0).agg(____)

# Bar plot of avg_inv_by_month
avg_inv_by_month.____
plt.show()

#____________________#
#Solutions

# Concatenate the tables and add keys
inv_jul_thr_sep = pd.concat([inv_jul, inv_aug,inv_sep],
                            keys=['7Jul','8Aug','9Sep'])

# Group the invoices by the index keys and find avg of the total column
avg_inv_by_month = inv_jul_thr_sep.groupby(level=0).agg({'total':'mean'})

# Bar plot of avg_inv_by_month
avg_inv_by_month.plot(kind='bar')
plt.show()
```

Exercise5

Using the append method

The .concat() method is excellent when you need a lot of control over how concatenation is performed. However, if you do not need as much control, then the .append() method is another option. You'll try this method out by appending the track lists together from different Metallica albums. From there, you will merge it with the invoice_items table to determine which track sold the most.

The tables tracks_master, tracks_ride, tracks_st, and invoice_items have loaded for you.

Instructions

- Use the .append() method to combine (in this order) tracks_ride, tracks_master, and tracks_st together vertically, and save to metallica_tracks.
- Merge metallica_tracks and invoice_items on tid with an inner join, and save to tracks_invoices.
- For each tid and name in tracks_invoices, sum the quantity sold column, and save as tracks_sold.
- Sort tracks_sold in descending order by the quantity column, and print the table.

In [ ]:

```
# Use the .append() method to combine the tracks tables
metallica_tracks = ____.append(____, sort=False)

# Merge metallica_tracks and invoice_items
tracks_invoices = ____

# For each tid and name sum the quantity sold
tracks_sold = tracks_invoices.groupby(['tid','name']).agg(____)

# Sort in decending order by quantity and print the results
print(tracks_sold.sort_values(____))

#____________________#
#Solutions

# Use the .append() method to combine the tracks tables
metallica_tracks = tracks_ride.append([tracks_master,tracks_st], sort=False)

# Merge metallica_tracks and invoice_items
tracks_invoices = metallica_tracks.merge(invoice_items, on='tid')

# For each tid and name sum the quantity sold
tracks_sold = tracks_invoices.groupby(['tid','name']).agg({'quantity':'sum'})

# Sort in decending order by quantity and print the results
print(tracks_sold.sort_values('quantity', ascending=False))
```

Exercise6

Concatenate and merge to find common songs

The senior leadership of the streaming service is requesting your help again. You are given the historical files for a popular playlist in the classical music genre in 2018 and 2019. Additionally, you are given a similar set of files for the most popular pop music genre playlist on the streaming service in 2018 and 2019. Your goal is to concatenate the respective files to make a large classical playlist table and overall popular music table. Then filter the classical music table using a semi join to return only the most popular classical music tracks.

The tables classic_18, classic_19, and pop_18, pop_19 have been loaded for you. Additionally, pandas has been loaded as pd.

Instructions

- Concatenate the classic_18 and classic_19 tables vertically where the index goes from 0 to n-1, and save to classic_18_19.
- Concatenate the pop_18 and pop_19 tables vertically where the index goes from 0 to n-1, and save to pop_18_19.
- With classic_18_19 on the left, merge it with pop_18_19 on tid using an inner join.
- Use .isin() to filter classic_18_19 where tid is in classic_pop.

In [ ]:

```
# Concatenate the classic tables vertically
classic_18_19 = ____

# Concatenate the pop tables vertically
pop_18_19 = ____

# Merge classic_18_19 with pop_18_19
classic_pop = ____

# Using .isin(), filter classic_18_19 rows where tid is in classic_pop
popular_classic = classic_18_19[classic_18_19[____].isin(____)]

# Print popular chart
print(popular_classic)

#____________________#
#Solutions

# Concatenate the classic tables vertically
classic_18_19 = pd.concat([classic_18,classic_19], sort=True, ignore_index=True)

# Concatenate the pop tables vertically
pop_18_19 = pd.concat([pop_18,pop_19], sort=True, ignore_index=True)

# Merge classic_18_19 with pop_18_19
classic_pop = classic_18_19.merge(pop_18_19, on='tid')

# Using .isin(), filter classic_18_19 rows where tid is in classic_pop
popular_classic = classic_18_19[classic_18_19['tid'].isin(classic_pop['tid'])]

# Print popular chart
print(popular_classic)
```