Module 3: Python & teradataml Labs

Teradata Vantage Analytics Workshop

ADVANCED

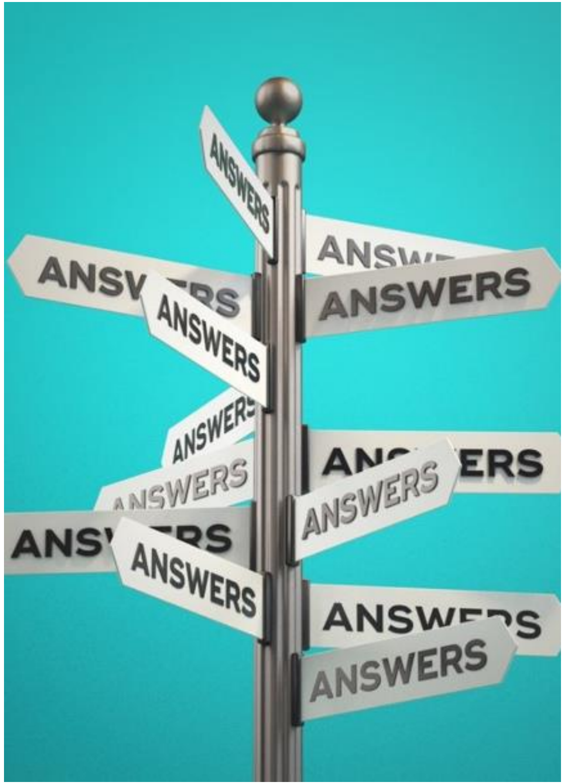
Objectives

After completing this module, you will be able to:

- Load Teradata Python library 'teradataml'
- Connect to a Teradata Vantage context within JupyterLab
- Convert Tables into Teradata Dataframes
- Convert Teradata DataFrames into Pandas
- Convert Teradata DataFrames into Teradata Tables
- View underlying Advanced SQL query from Python code
- Run following Python functions:
  - Sessionize
  - Attribution
  - nPath
  - VAL Association

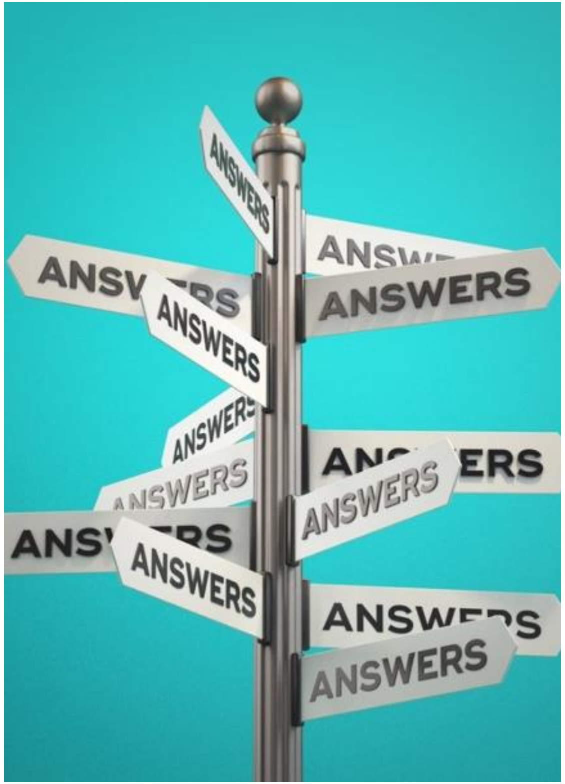
Topics

- Use Case 01: Recommend shows for customers who watch 'Game of Thrones'
  - Sessionize
  - Attribution
  - Npath
- Use Case 02: Recommend products for customers who purchase Chicken Nuggets
  - VAL Association
- Review & Summary



Current Topic – Use Case: Game of Thrones®

- **Use Case 01: Recommend shows for customers who watch 'Game of Thrones'**
  - **Sessionize**
  - **Attribution**
  - **Npath**
- Use Case 02: Recommend products for customers who purchase Chicken Nuggets
  - VAL Association
- Review & Summary



Here's the 'GameOfThrones' Scenario

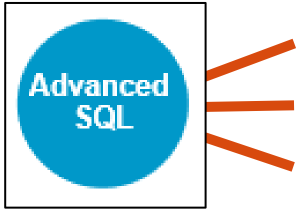
Goal: Find all TV shows surfed 1-Day before the User settles on 'GameOfThrones'. These shows will be basis for recommending to 'GameOfThrones' viewers

TV Service Provider has black box for its customer’s so they can purchase on-line TV shows on demand as well as surf their normal pay channels

Provider wants to know which channels were surfed before the on-demand show 'GameOfThrones' was purchased for viewing. This will help Provider make recommendations of other channels like 'GameOfThrones'

Functions/ML Utilized

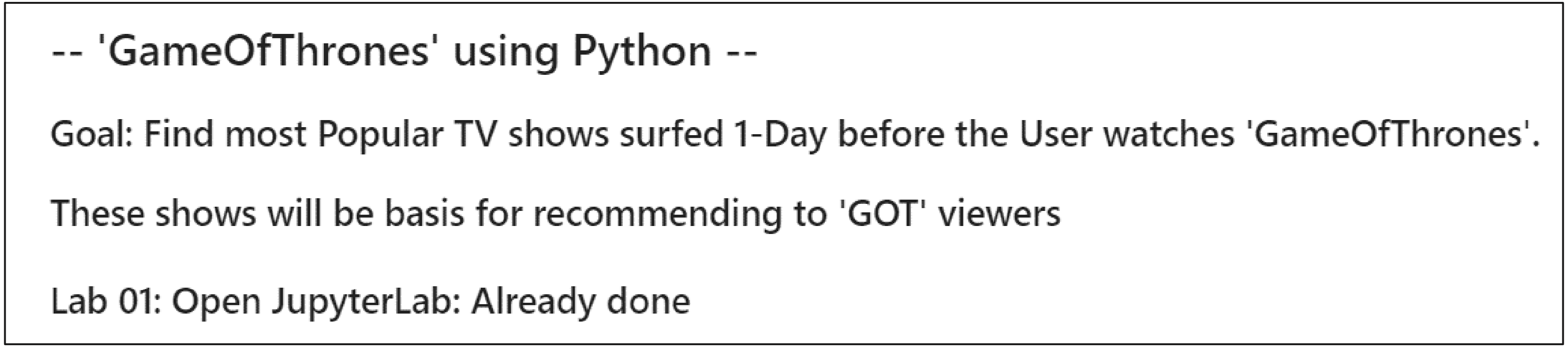
If functions utilized, which algorithms and which order of execution?



1. Sessionize	Find Visit Ct	(Sessionid) per Partition (customer)
2. Attribution	Attribute the Data	(Filter by 1-Day increments)
3. Npath	Pattern Detection	(Find shows prior to GOT to make tv show recommendation)

Lab 1: Open JupyterLab

1. Open file: 08c) GOT.ipynb
2. Highlight the 1st Cell (you'll get a blue vertical bar for that Cell)



-- 'GameOfThrones' using Python --

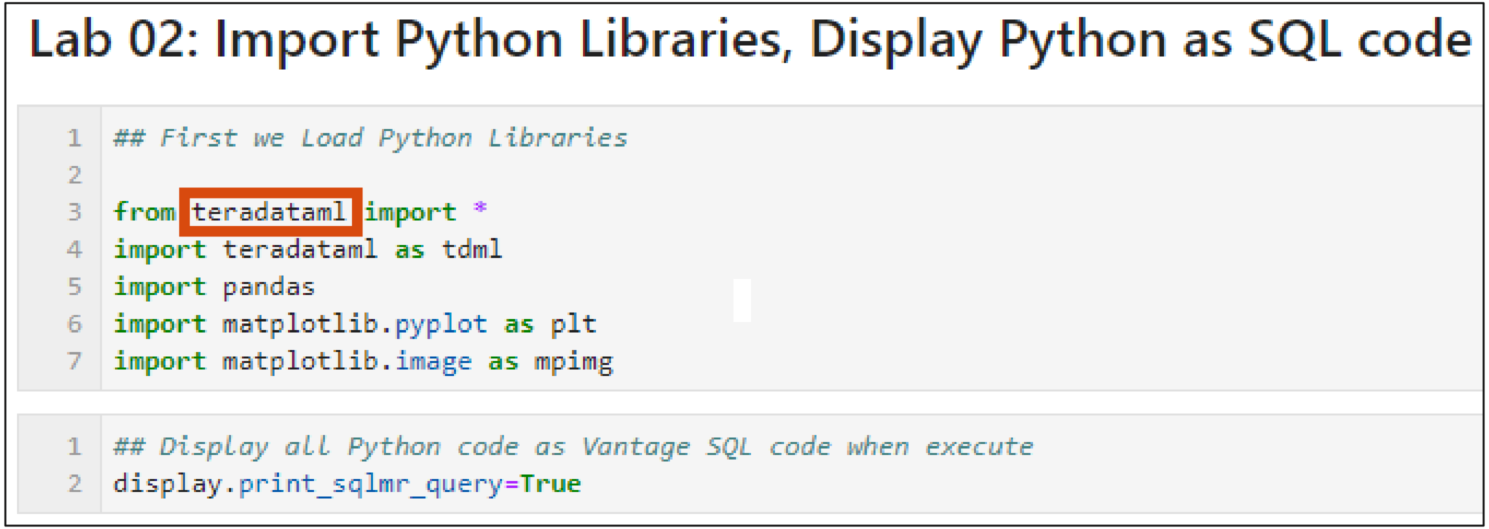
Goal: Find most Popular TV shows surfed 1-Day before the User watches 'GameOfThrones'.

These shows will be basis for recommending to 'GOT' viewers

Lab 01: Open JupyterLab: Already done

Lab 2a: Import Python Libraries

1. Highlight Cell 1 (you'll get a blue vertical bar for that Cell)
2. Click Run button ▶. Kernel indicator circle will fill in. When finished it will be White again (sometimes it happens so fast won't see circle fill in)
3. Run Cell 2 to Display all Python code as SQL code

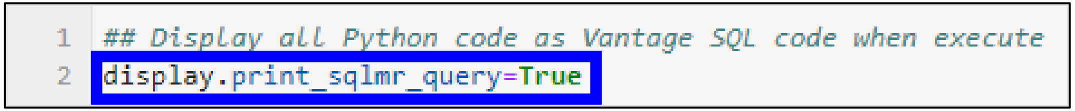


Lab 02: Import Python Libraries, Display Python as SQL code

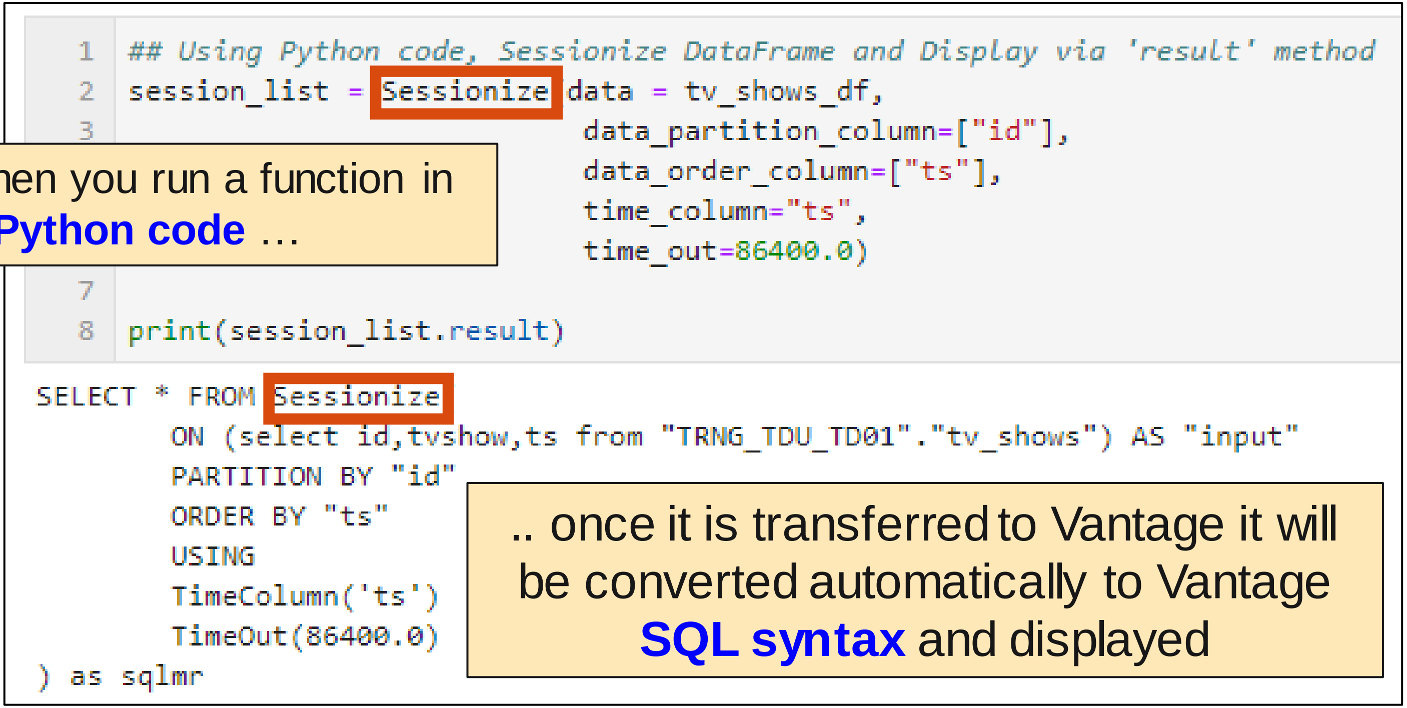
```
## First we Load Python Libraries

from teradataml import *
import teradataml as tdml
import pandas
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
```

```
## Display all Python code as Vantage SQL code when execute
display.print_sqlmr_query=True
```

Lab 2b: View Underlying Vantage SQL Query



```
## Display all Python code as Vantage SQL code when execute
display.print_sqlmr_query=True
```



```
## Using Python code, Sessionize DataFrame and Display via 'result' method
session_list = Sessionize(data = tv_shows_df,
                           data_partition_column=["id"],
                           data_order_column=["ts"],
                           time_column="ts",
                           time_out=86400.0)

print(session_list.result)
```

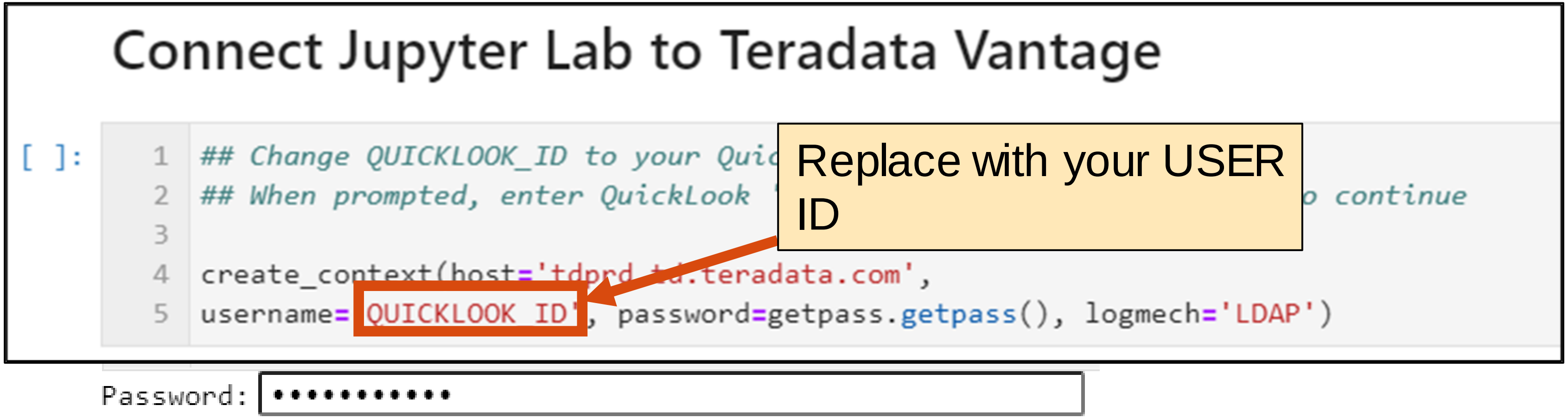
```
SELECT * FROM Sessionize
        ON (select id,tvshow,ts from "TRNG_TDU_TD01"."tv_shows") AS "input"
        PARTITION BY "id"
        ORDER BY "ts"
        USING
        TimeColumn('ts')
        TimeOut(86400.0)
) as sqlmr
```

Later, when you run a function in **Python code** …

.. once it is transferred to Vantage it will be converted automatically to Vantage **SQL syntax** and displayed

Lab 3: Connect JupyterLab to Vantage

Run 'create_context' method to connect Python client to Vantage Cluster via JDBC



Connect Jupyter Lab to Teradata Vantage

[ ]:

```
## Change QUICKLOOK_ID to your Quic
## When prompted, enter QuickLook '                              o continue

create_context(host='tdprd.td.teradata.com',
username='QUICKLOOK_ID', password=getpass.getpass(), logmech='LDAP')
```

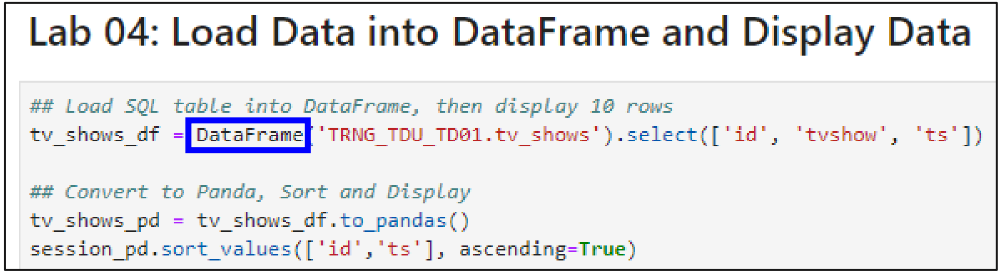
```
Password: ••••••••••••
```

Replace with your USER ID

Must enter your 'password' followed by the Enter key to proceed

Lab 4: Load Table into Dataframe and Display

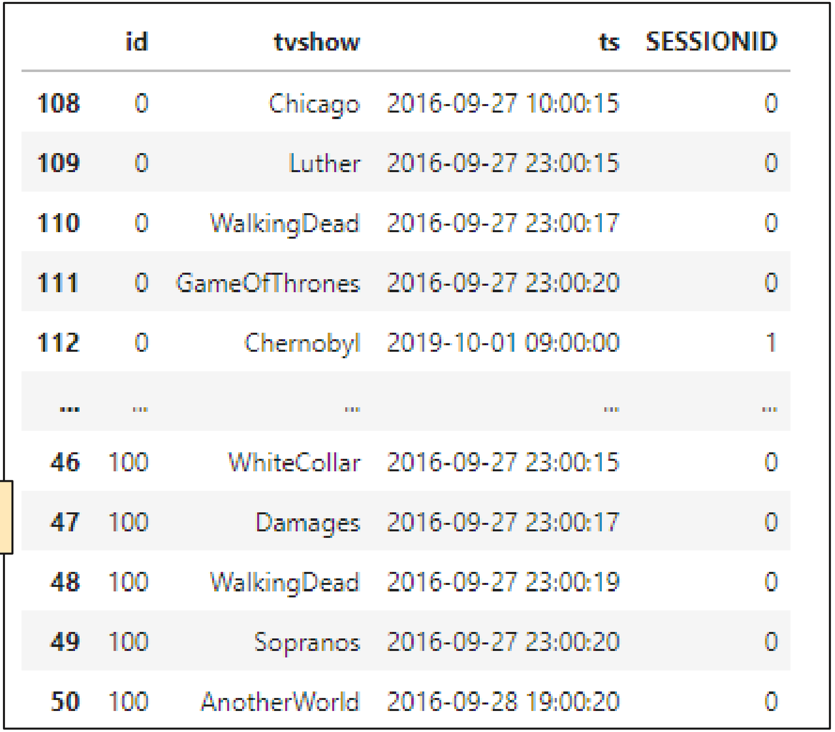
Use 'DataFrame' command to load SQL Table into a DataFrame.
Convert to Panda, Sort and Display



Lab 04: Load Data into DataFrame and Display Data

```
## Load SQL table into DataFrame, then display 10 rows
tv_shows_df = DataFrame('TRNG_TDU_TD01.tv_shows').select(['id', 'tvshow', 'ts'])

## Convert to Panda, Sort and Display
tv_shows_pd = tv_shows_df.to_pandas()
session_pd.sort_values(['id','ts'], ascending=True)
```

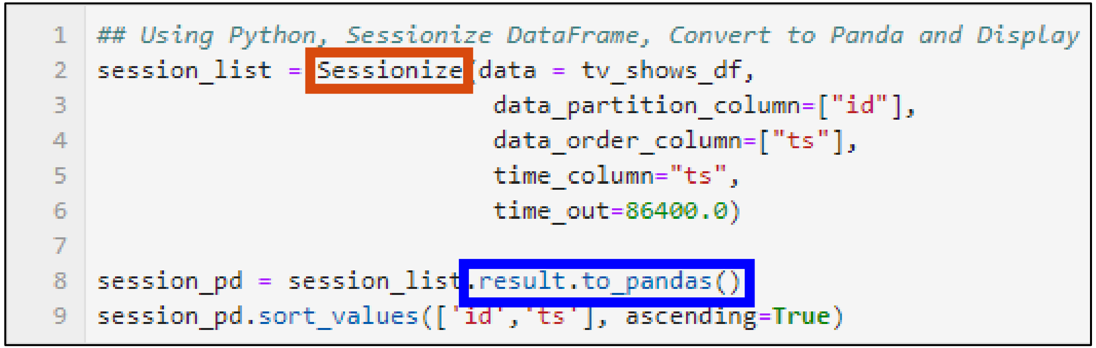
Here's 10 rows of the Panda



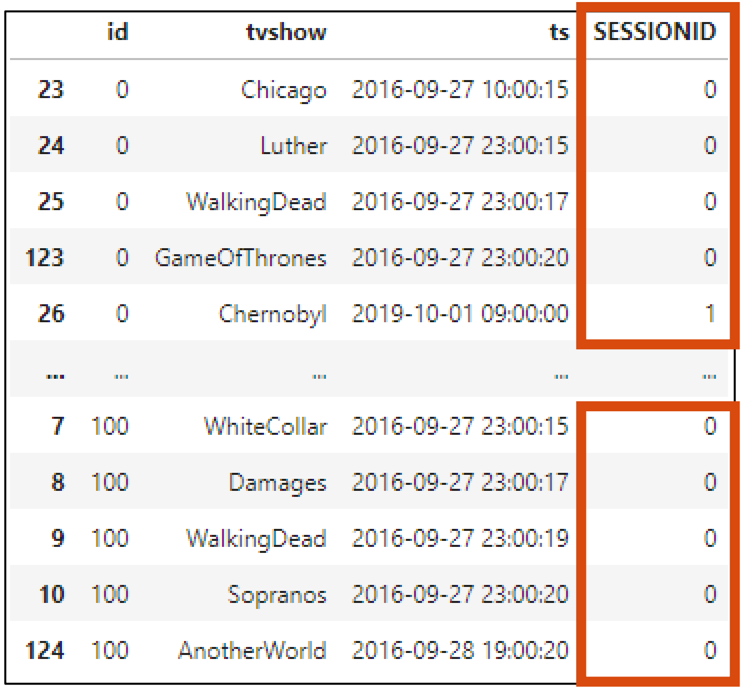
	id	tvshow	ts	SESSIONID
108	0	Chicago	2016-09-27 10:00:15	0
109	0	Luther	2016-09-27 23:00:15	0
110	0	WalkingDead	2016-09-27 23:00:17	0
111	0	GameOfThrones	2016-09-27 23:00:20	0
112	0	Chernobyl	2019-10-01 09:00:00	1
...	...	...	...	...
46	100	WhiteCollar	2016-09-27 23:00:15	0
47	100	Damages	2016-09-27 23:00:17	0
48	100	WalkingDead	2016-09-27 23:00:19	0
49	100	Sopranos	2016-09-27 23:00:20	0
50	100	AnotherWorld	2016-09-28 19:00:20	0

Lab 5a: Sessionize the DataFrame

Note use of the 'result' and 'to_pandas'. This converts DataFrame into a Panda

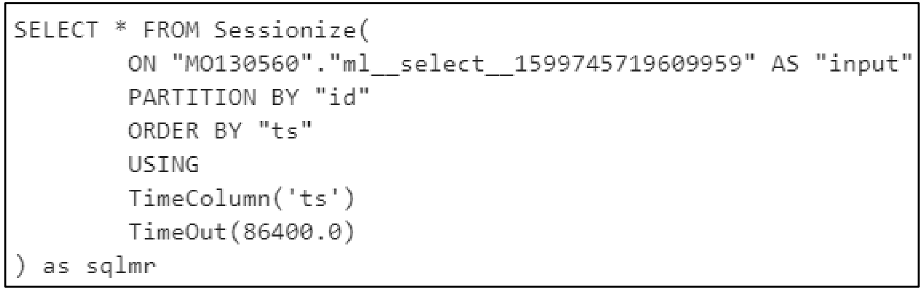


```
## Using Python, Sessionize DataFrame, Convert to Panda and Display
session_list = Sessionize(data = tv_shows_df,
                          data_partition_column=["id"],
                          data_order_column=["ts"],
                          time_column="ts",
                          time_out=86400.0)

session_pd = session_list.result.to_pandas()
session_pd.sort_values(['id','ts'], ascending=True)
```



	id	tvshow	ts	SESSIONID
23	0	Chicago	2016-09-27 10:00:15	0
24	0	Luther	2016-09-27 23:00:15	0
25	0	WalkingDead	2016-09-27 23:00:17	0
123	0	GameOfThrones	2016-09-27 23:00:20	0
26	0	Chernobyl	2019-10-01 09:00:00	1
...	...	...	...	...
7	100	WhiteCollar	2016-09-27 23:00:15	0
8	100	Damages	2016-09-27 23:00:17	0
9	100	WalkingDead	2016-09-27 23:00:19	0
10	100	Sopranos	2016-09-27 23:00:20	0
124	100	AnotherWorld	2016-09-28 19:00:20	0

SQL conversion

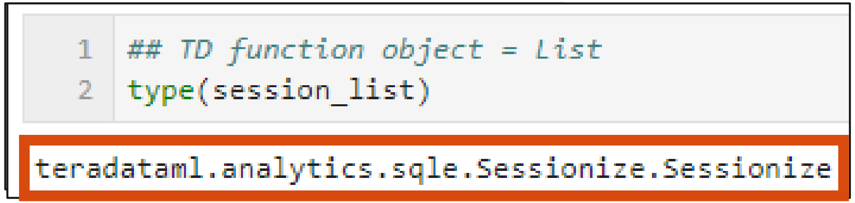


```
SELECT * FROM Sessionize(
        ON "MO130560"."ml__select__1599745719609959" AS "input"
        PARTITION BY "id"
        ORDER BY "ts"
        USING
        TimeColumn('ts')
        TimeOut(86400.0)
) as sqlmr
```

Lab 5b: 'type' and 'result' Methods

'type' command is used to display Python object type.

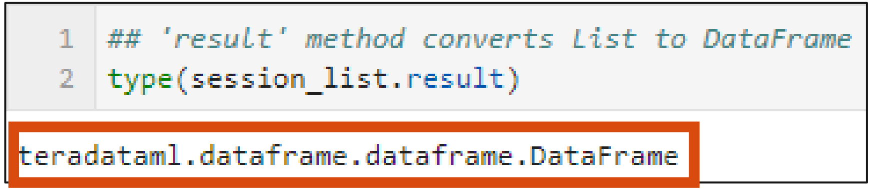
Vantage function output will typically be a List object and display like this



```
## TD function object = List
type(session_list)
```

```
teradataml.analytics.sqle.Sessionize.Sessionize
```

'result' method converts a List object to a DataFrame object so it can be further processed by another function

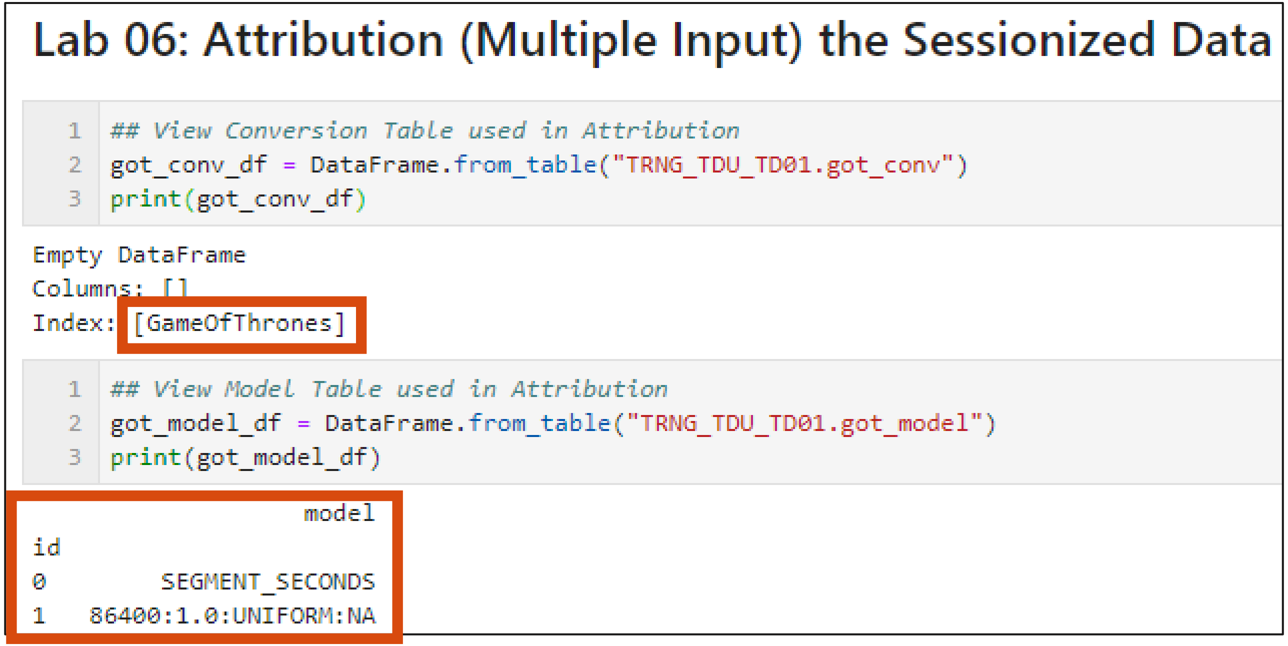


```
## 'result' method converts List to DataFrame
type(session_list.result)
```

```
teradataml.dataframe.dataframe.DataFrame
```

Lab 6: Attribution the Sessionized Data

We'll be using Multiple-Input Attribution. As such we'll point to three DataFrames (Input, Conversion and Model). Here's content of Conversion and Model DataFrames (Input will be the Sessionized DataFrame)



Lab 06: Attribution (Multiple Input) the Sessionized Data

```
## View Conversion Table used in Attribution
got_conv_df = DataFrame.from_table("TRNG_TDU_TD01.got_conv")
print(got_conv_df)
```

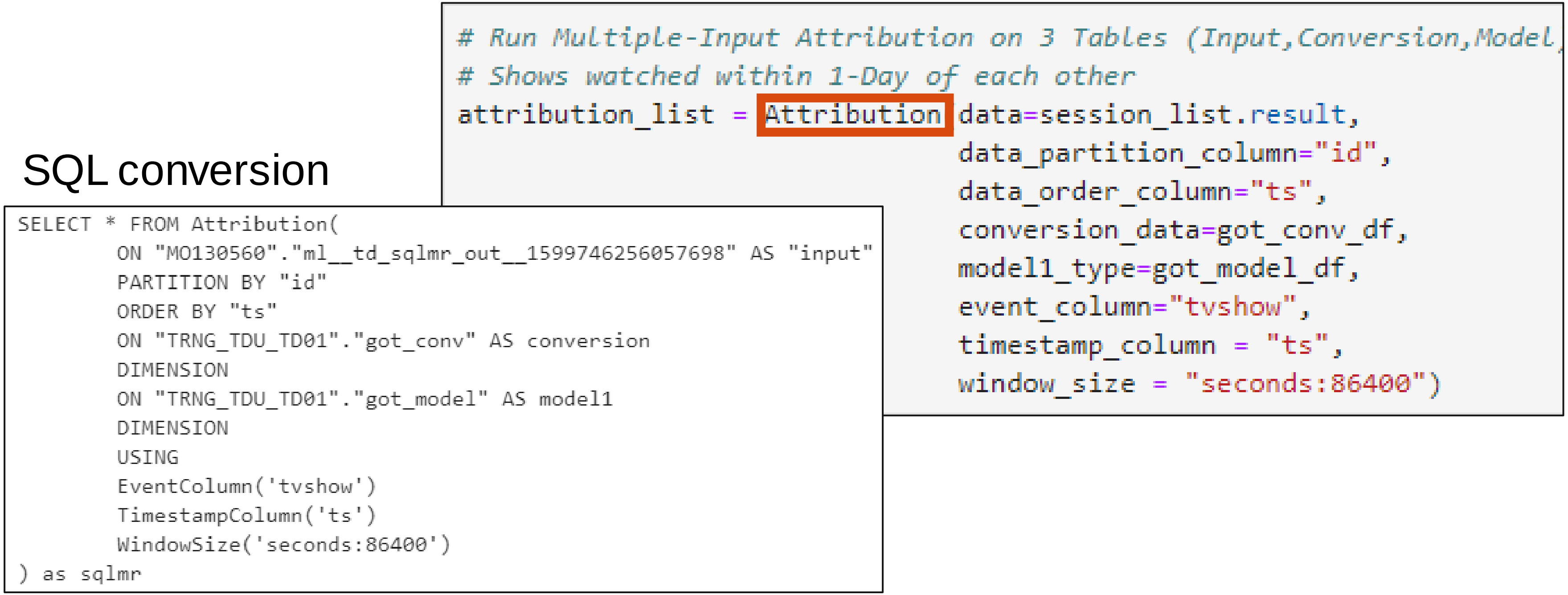
```
Empty DataFrame
Columns: []
Index: [GameOfThrones]
```

```
## View Model Table used in Attribution
got_model_df = DataFrame.from_table("TRNG_TDU_TD01.got_model")
print(got_model_df)
```

```
                        model
id
0             SEGMENT_SECONDS
1      86400:1.0:UNIFORM:NA
```

Lab 6: Attribution the Sessionized Data (cont.)

After we run Attribution, we now have Weights for those tv shows watched 1-day prior to 'GameOfThrones' (see next Slide for Output)



```
# Run Multiple-Input Attribution on 3 Tables (Input,Conversion,Model,
# Shows watched within 1-Day of each other
attribution_list = Attribution data=session_list.result,
                               data_partition_column="id",
                               data_order_column="ts",
                               conversion_data=got_conv_df,
                               model1_type=got_model_df,
                               event_column="tvshow",
                               timestamp_column = "ts",
                               window_size = "seconds:86400")
```

SQL conversion

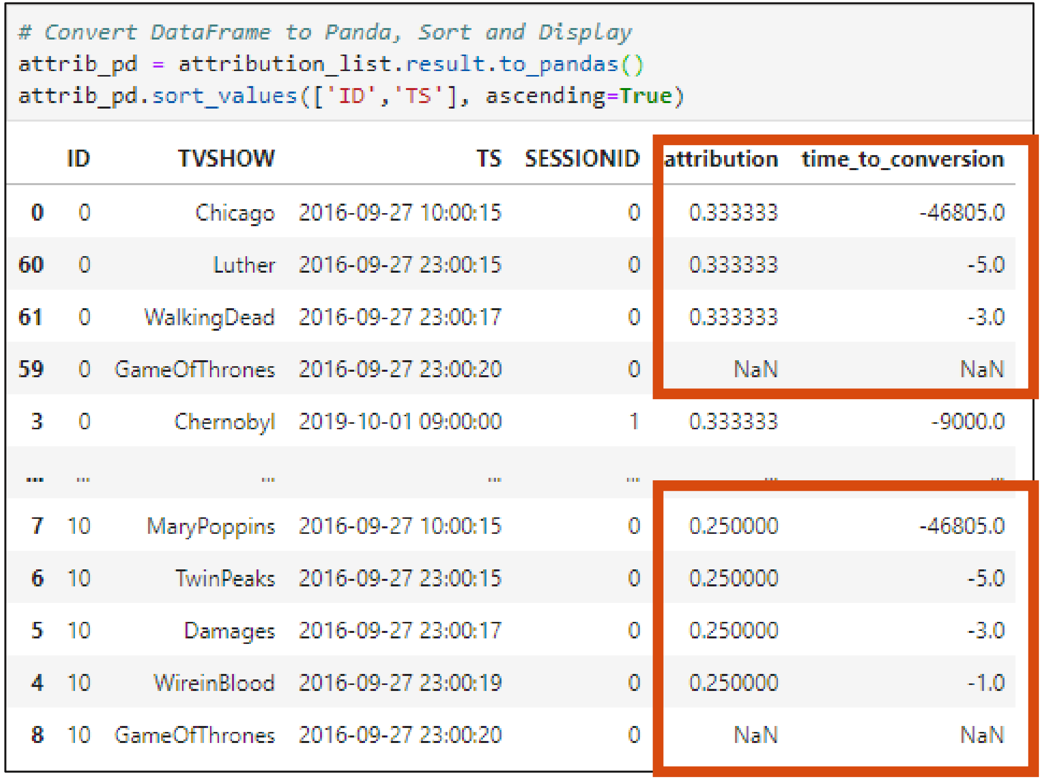
```
SELECT * FROM Attribution(
        ON "MO130560"."ml__td_sqlmr_out__1599746256057698" AS "input"
        PARTITION BY "id"
        ORDER BY "ts"
        ON "TRNG_TDU_TD01"."got_conv" AS conversion
        DIMENSION
        ON "TRNG_TDU_TD01"."got_model" AS model1
        DIMENSION
        USING
        EventColumn('tvshow')
        TimestampColumn('ts')
        WindowSize('seconds:86400')
) as sqlmr
```

Lab 6: Attribution the Sessionized Data (cont.)

We convert the DataFrame into a Panda via 'to_pandas' command

Then we sort by 'ID' and 'TS' using the 'sort_values()' command

Here we see which tv shows were watches for User (ID) 0 and 10 prior to watching 'GOT'



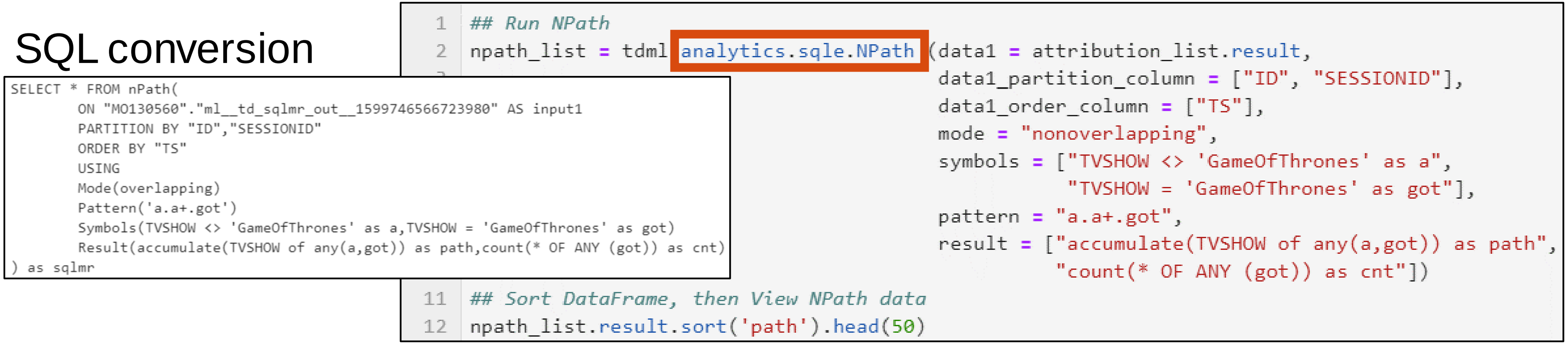
```
# Convert DataFrame to Panda, Sort and Display
attrib_pd = attribution_list.result.to_pandas()
attrib_pd.sort_values(['ID','TS'], ascending=True)
```

	ID	TVSHOW	TS	SESSIONID	attribution	time_to_conversion
0	0	Chicago	2016-09-27 10:00:15	0	0.333333	-46805.0
60	0	Luther	2016-09-27 23:00:15	0	0.333333	-5.0
61	0	WalkingDead	2016-09-27 23:00:17	0	0.333333	-3.0
59	0	GameOfThrones	2016-09-27 23:00:20	0	NaN	NaN
3	0	Chernobyl	2019-10-01 09:00:00	1	0.333333	-9000.0
...	...	...	...	...	...	...
7	10	MaryPoppins	2016-09-27 10:00:15	0	0.250000	-46805.0
6	10	TwinPeaks	2016-09-27 23:00:15	0	0.250000	-5.0
5	10	Damages	2016-09-27 23:00:17	0	0.250000	-3.0
4	10	WireinBlood	2016-09-27 23:00:19	0	0.250000	-1.0
8	10	GameOfThrones	2016-09-27 23:00:20	0	NaN	NaN

Lab 7: NPath the Attribution Data

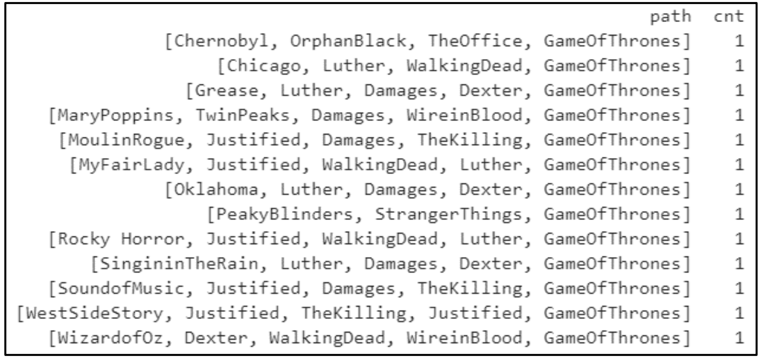
Next, we run NPath to find Pattern of tv shows watched 24 hours prior to 'GOT'

SQL conversion



```
## Run NPath
npath_list = tdml analytics.sqle.NPath(data1 = attribution_list.result,
                                       data1_partition_column = ["ID", "SESSIONID"],
                                       data1_order_column = ["TS"],
                                       mode = "nonoverlapping",
                                       symbols = ["TVSHOW <> 'GameOfThrones' as a",
                                                  "TVSHOW = 'GameOfThrones' as got"],
                                       pattern = "a.a+.got",
                                       result = ["accumulate(TVSHOW of any(a,got)) as path",
                                                 "count(* OF ANY (got)) as cnt"])

## Sort DataFrame, then View NPath data
npath_list.result.sort('path').head(50)
```

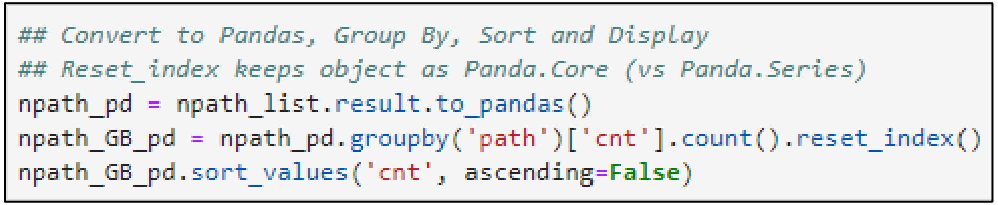
```
SELECT * FROM nPath(
        ON "MO130560"."ml__td_sqlmr_out__1599746566723980" AS input1
        PARTITION BY "ID","SESSIONID"
        ORDER BY "TS"
        USING
        Mode(overlapping)
        Pattern('a.a+.got')
        Symbols(TVSHOW <> 'GameOfThrones' as a,TVSHOW = 'GameOfThrones' as got)
        Result(accumulate(TVSHOW of any(a,got)) as path,count(* OF ANY (got)) as cnt)
) as sqlmr
```



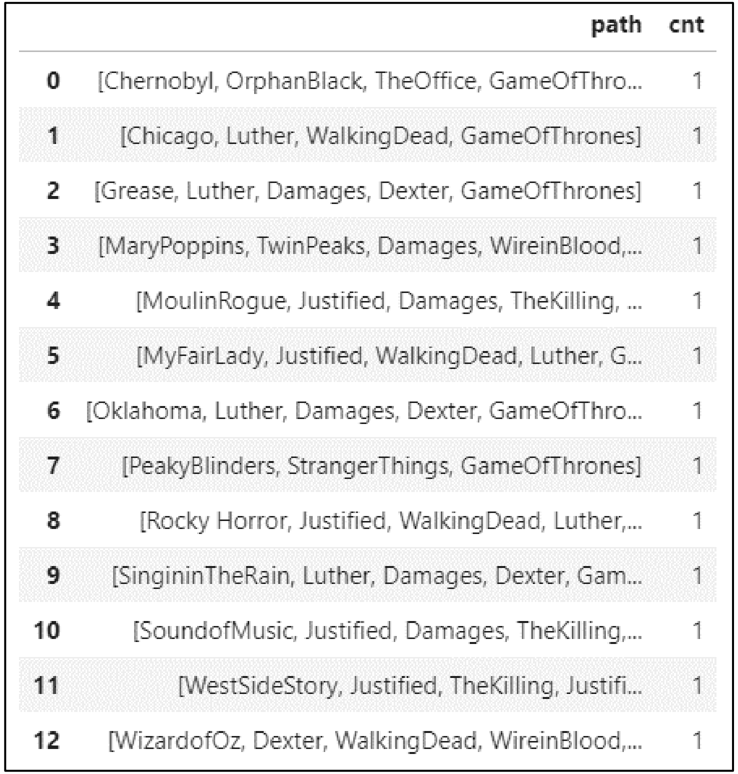
```
                                                   path  cnt
          [Chernobyl, OrphanBlack, TheOffice, GameOfThrones]    1
               [Chicago, Luther, WalkingDead, GameOfThrones]    1
             [Grease, Luther, Damages, Dexter, GameOfThrones]    1
    [MaryPoppins, TwinPeaks, Damages, WireinBlood, GameOfThrones]    1
     [MoulinRogue, Justified, Damages, TheKilling, GameOfThrones]    1
      [MyFairLady, Justified, WalkingDead, Luther, GameOfThrones]    1
           [Oklahoma, Luther, Damages, Dexter, GameOfThrones]    1
               [PeakyBlinders, StrangerThings, GameOfThrones]    1
    [Rocky Horror, Justified, WalkingDead, Luther, GameOfThrones]    1
       [SinginTheRain, Luther, Damages, Dexter, GameOfThrones]    1
    [SoundofMusic, Justified, Damages, TheKilling, GameOfThrones]    1
  [WestSideStory, Justified, TheKilling, Justified, GameOfThrones]    1
    [WizardofOz, Dexter, WalkingDead, WireinBlood, GameOfThrones]    1
```

Lab 8: Convert DataFrame to Panda and Display Paths in Descending Order

The tv shows in these Paths should be Recommended to all those customers who watch 'GOT'



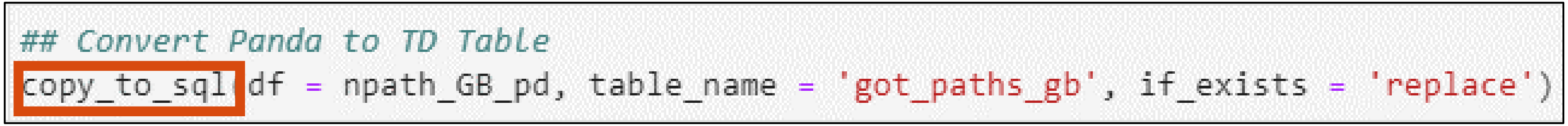
```
## Convert to Pandas, Group By, Sort and Display
## Reset_index keeps object as Panda.Core (vs Panda.Series)
npath_pd = npath_list.result.to_pandas()
npath_GB_pd = npath_pd.groupby('path')['cnt'].count().reset_index()
npath_GB_pd.sort_values('cnt', ascending=False)
```



	path	cnt
0	[Chernobyl, OrphanBlack, TheOffice, GameOfThro...	1
1	[Chicago, Luther, WalkingDead, GameOfThrones]	1
2	[Grease, Luther, Damages, Dexter, GameOfThrones]	1
3	[MaryPoppins, TwinPeaks, Damages, WireinBlood,...	1
4	[MoulinRogue, Justified, Damages, TheKilling, ...	1
5	[MyFairLady, Justified, WalkingDead, Luther, G...	1
6	[Oklahoma, Luther, Damages, Dexter, GameOfThro...	1
7	[PeakyBlinders, StrangerThings, GameOfThrones]	1
8	[Rocky Horror, Justified, WalkingDead, Luther,...	1
9	[SinginTheRain, Luther, Damages, Dexter, Gam...	1
10	[SoundofMusic, Justified, Damages, TheKilling,...	1
11	[WestSideStory, Justified, TheKilling, Justifi...	1
12	[WizardofOz, Dexter, WalkingDead, WireinBlood,...	1

Lab 9: Convert Panda to TD Table using 'copy_to_sql'

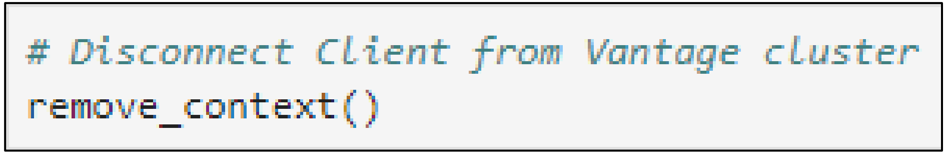
Use 'copy_to_sql' method to convert DataFrame into Teradata table



```
## Convert Panda to TD Table
copy_to_sql(df = npath_GB_pd, table_name = 'got_paths_gb', if_exists = 'replace')
```

Lab 10: Remove Context

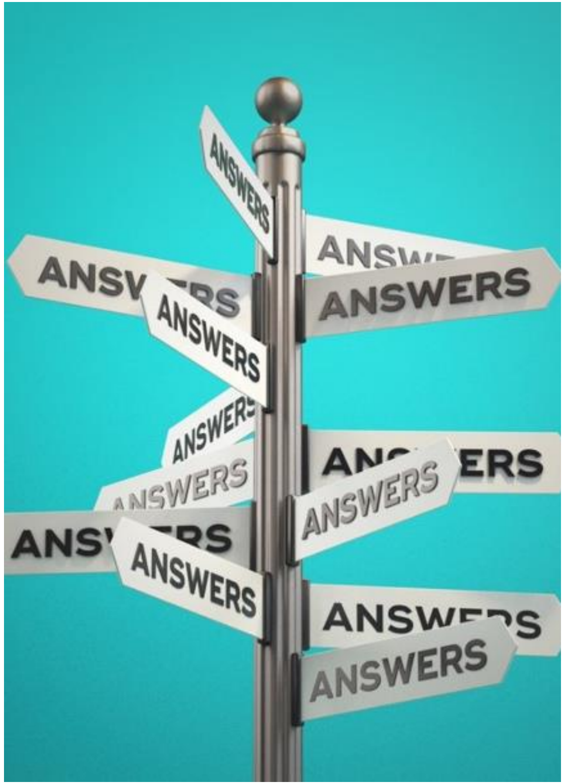
This command also does any Garbage Collection that is needed (removes Temporary tables)



```
# Disconnect Client from Vantage cluster
remove_context()
```

Current Topic – Use Case: Chicken Nuggets

- Use Case 01: Recommend shows for customers who watch 'Game of Thrones'
  - – Sessionize
  - – Attribution
  - – Npath
- **Use Case 02: Recommend products for customers who purchase Chicken Nuggets**
  - **– VAL Association**
- Review & Summary



Association Overview

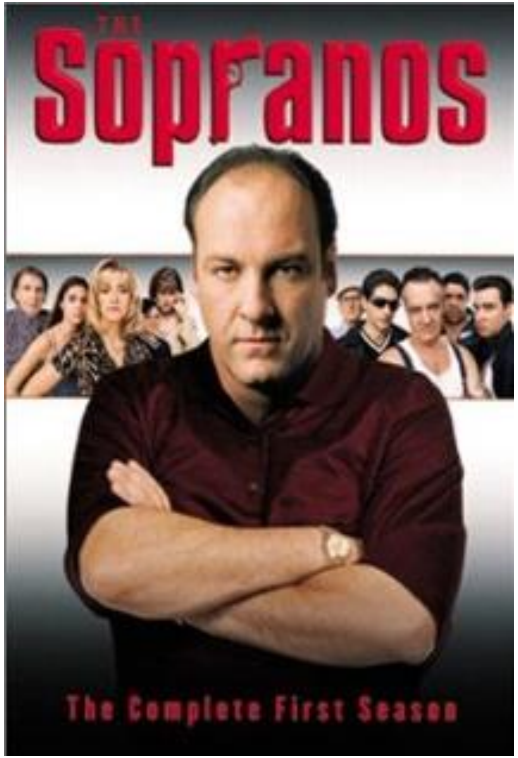
- Very common use case for retailers, on-line retailers, internet and consumer-focused financial institutions

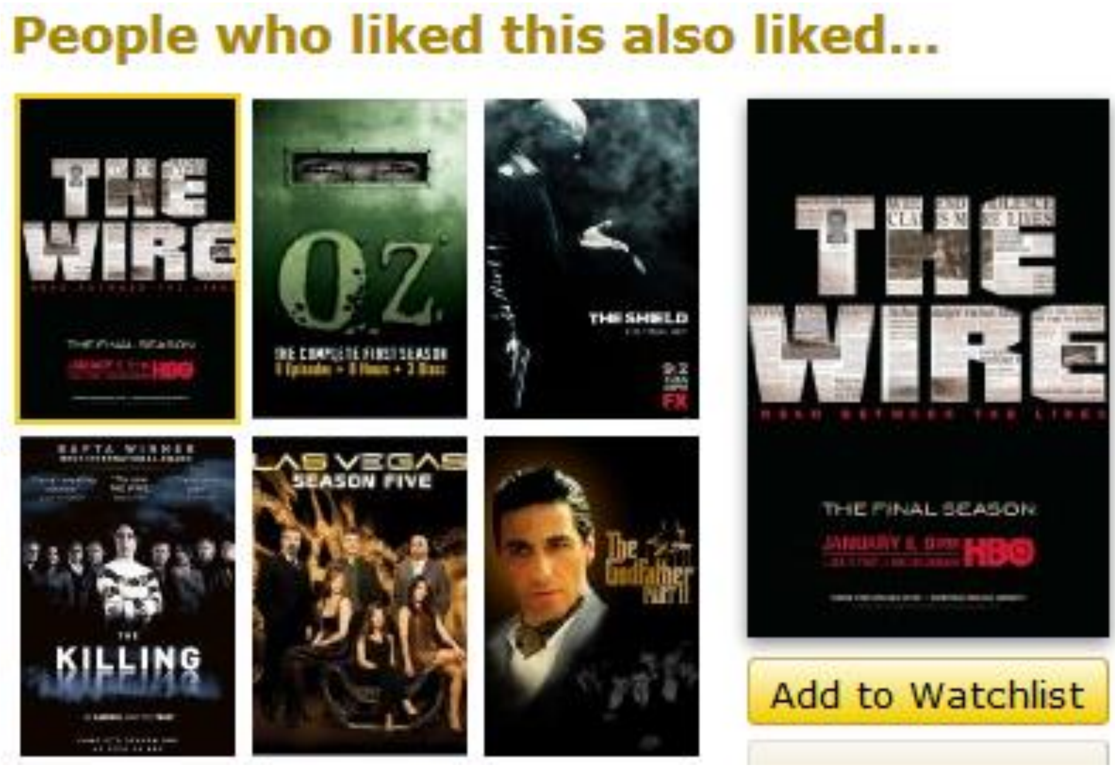
- Source data could be:
  - Retail purchase data
  - On-line purchase data
  - Activity data
  - Credit card purchase data

Association is a method of making automatic predictions (filtering) about the interests of a user by collecting preferences from many users (collaborating). The underlying assumption is that if a person *A* has the same opinion as a person *B* on an issue, A is more likely to have B's opinion on a different issue *x* than to have the opinion on x of a person chosen randomly.

- Output could fuel "analytics products" like:
  - *"People who bought this also bought …"*
  - *"People who viewed this profile also viewed…"*
  - *"People who liked this job also liked …"*

Association Example



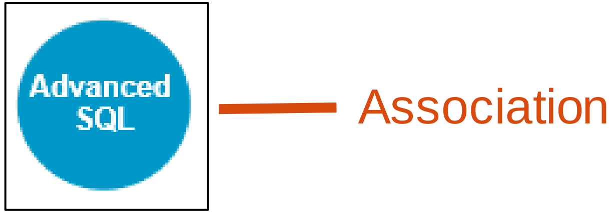

People who liked this also liked...

Allows for Merchandise opportunities such as cross-sell, and co-location of product

Here's the Chicken Nuggets Scenario

Goal – Chicken Nuggets has given promotional dollars to fund a grocery advertisement. To maximize sales, use Collaborative Filtering to find which products have strongest affinity with the Nuggets. Advertise these products along with Chicken Nuggets.

Vantage Analytics Library Algorithms Utilized

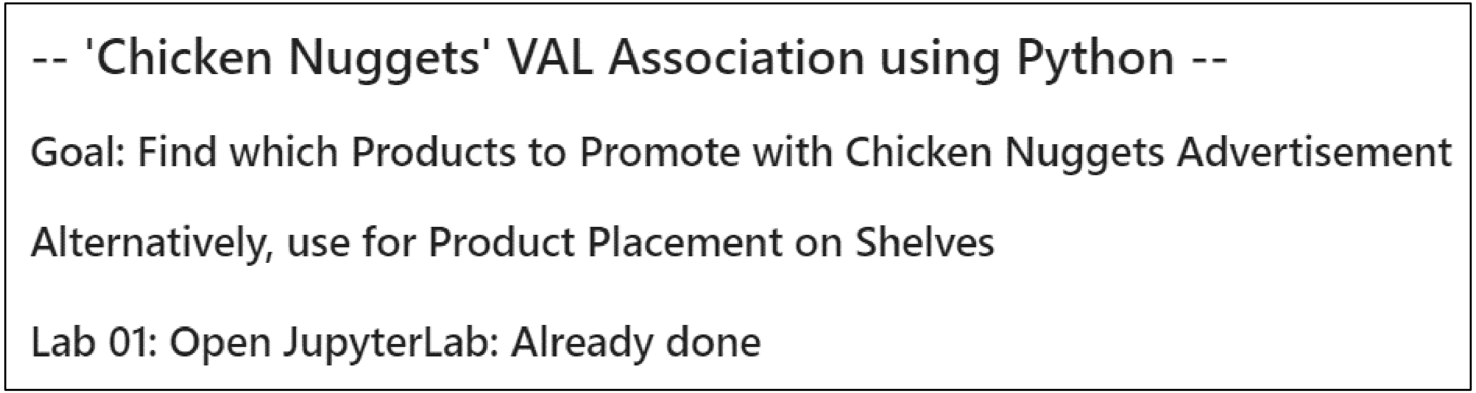


Advanced SQL

Association

Lab 11: Open JupyterLab

1. Open file: 08d) Nuggets.ipynb
2. Highlight the 1st Cell (you'll get a blue vertical bar for that Cell)



-- 'Chicken Nuggets' VAL Association using Python --

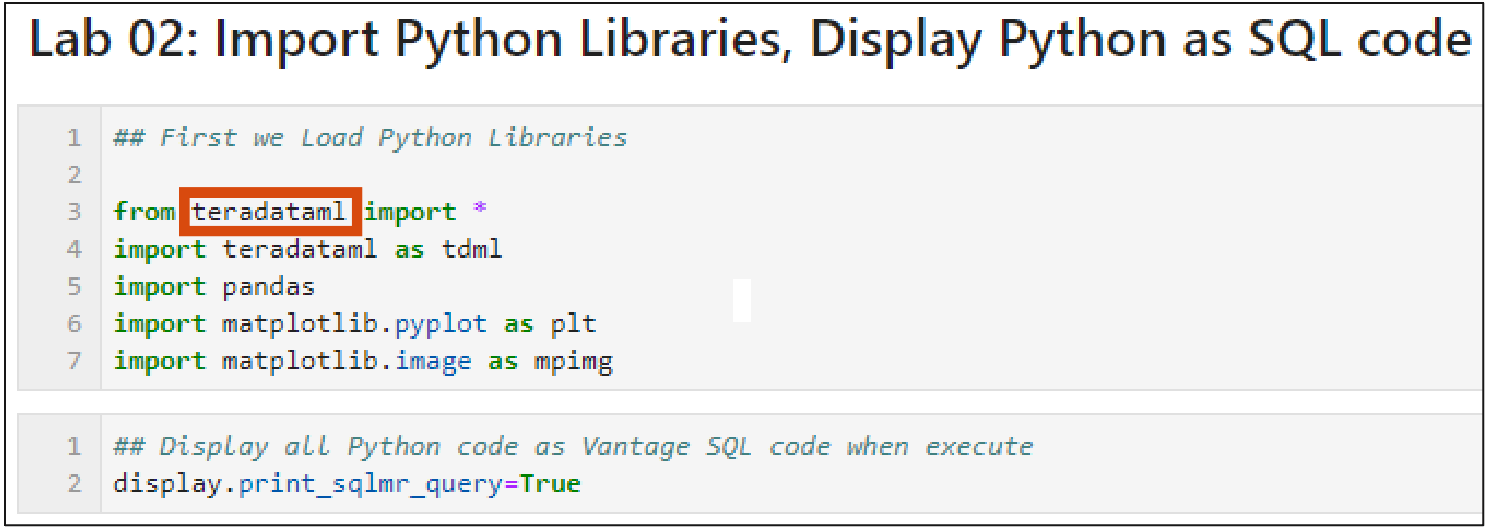
Goal: Find which Products to Promote with Chicken Nuggets Advertisement

Alternatively, use for Product Placement on Shelves

Lab 01: Open JupyterLab: Already done

Lab 12a: Load Python Libraries

1. Highlight Cell 1 (you'll get a blue vertical bar for that Cell)
2. Click Run button ▶. Kernel indicator circle will fill in. When finished it will be White again (sometimes it happens so fast won't see circle fill in)
3. Run Cell 2 to Display all Python code as SQL code

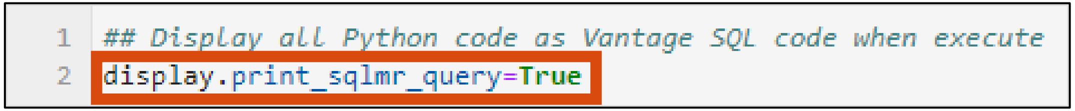


Lab 02: Import Python Libraries, Display Python as SQL code

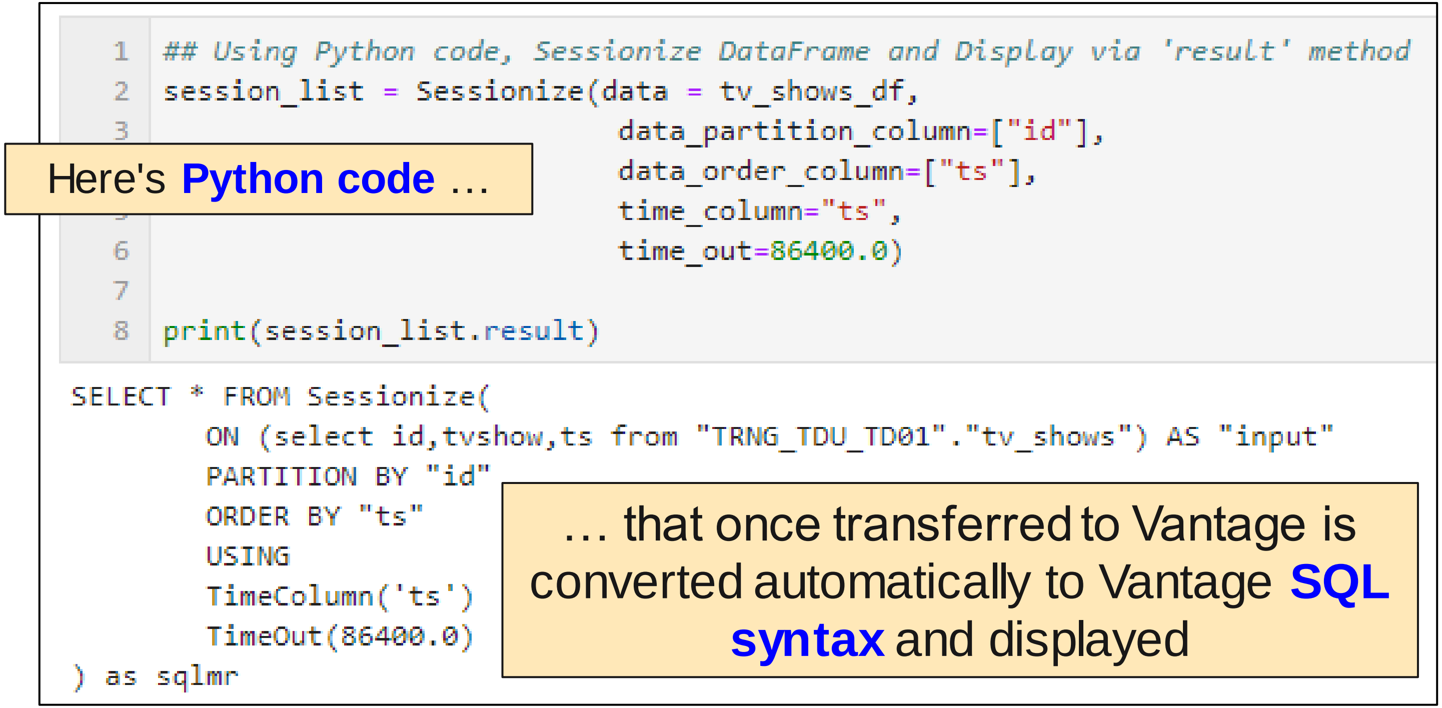
```
## First we Load Python Libraries

from teradataml import *
import teradataml as tdml
import pandas
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
```

```
## Display all Python code as Vantage SQL code when execute
display.print_sqlmr_query=True
```

Lab 12b: View Underlying Vantage SQL Query



```
## Display all Python code as Vantage SQL code when execute
display.print_sqlmr_query=True
```



```
## Using Python code, Sessionize DataFrame and Display via 'result' method
session_list = Sessionize(data = tv_shows_df,
                          data_partition_column=["id"],
                          data_order_column=["ts"],
                          time_column="ts",
                          time_out=86400.0)

print(session_list.result)
```

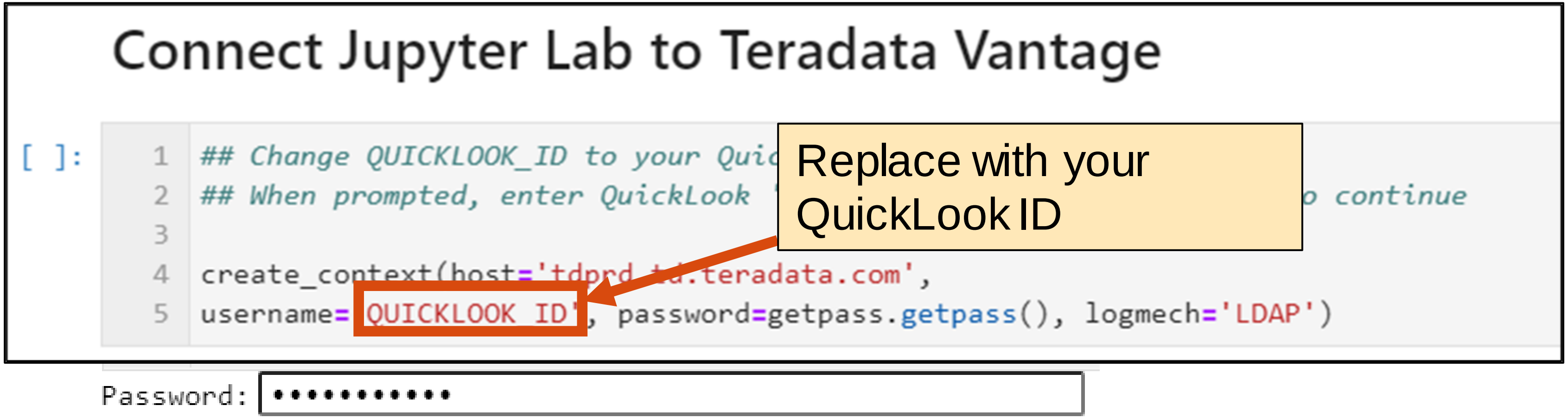
```
SELECT * FROM Sessionize(
        ON (select id,tvshow,ts from "TRNG_TDU_TD01"."tv_shows") AS "input"
        PARTITION BY "id"
        ORDER BY "ts"
        USING
        TimeColumn('ts')
        TimeOut(86400.0)
) as sqlmr
```

Here's **Python code** …

… that once transferred to Vantage is converted automatically to Vantage **SQL syntax** and displayed

Lab 13: Connect JupyterLab to Vantage

Run 'create_context' method to connect Python client to Vantage Cluster via JDBC



Connect Jupyter Lab to Teradata Vantage

[ ]:

```
## Change QUICKLOOK_ID to your Quic
## When prompted, enter QuickLook '                              o continue

create_context(host='tdprd.td.teradata.com',
username='QUICKLOOK_ID', password=getpass.getpass(), logmech='LDAP')
```

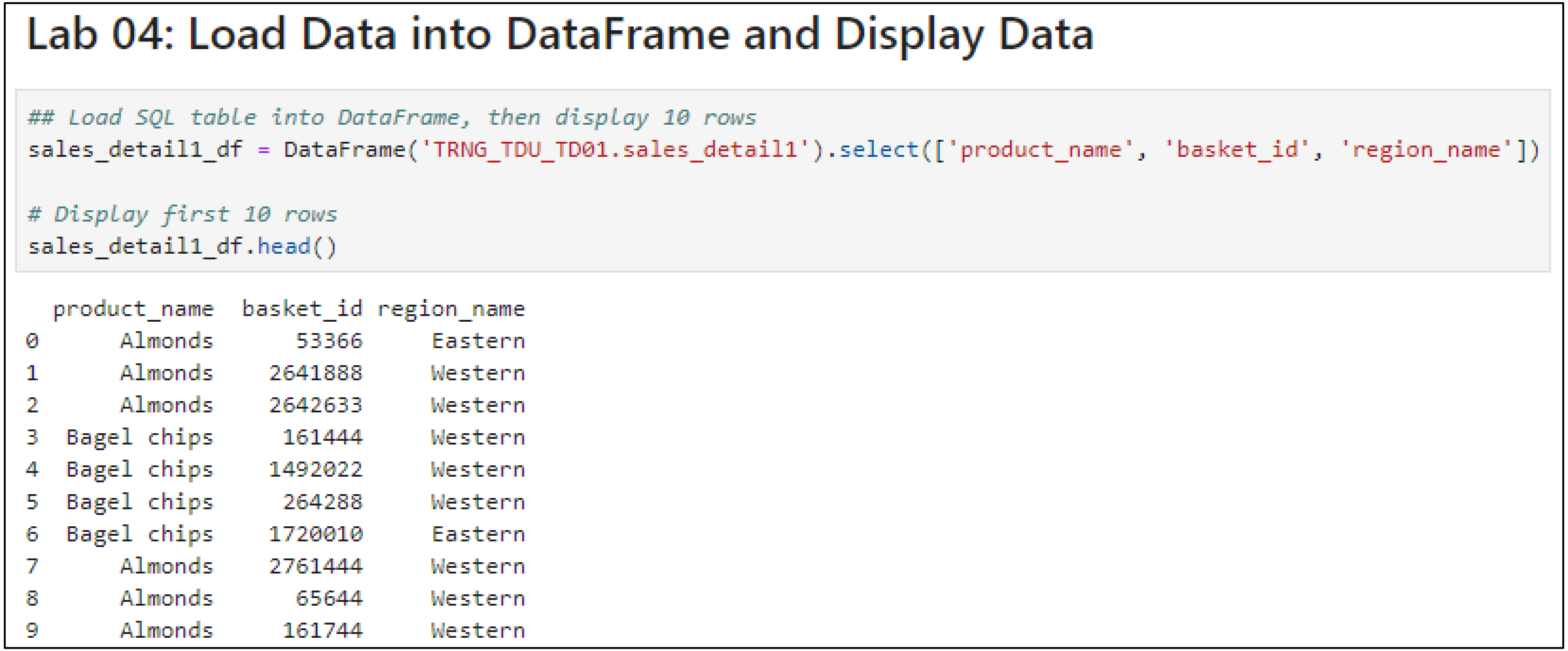
Password: ••••••••••••

Replace with your QuickLook ID

Must enter your LDAP 'password' followed by the Enter key to proceed

Lab 14a: Load Table into Dataframe and Display

Use 'DataFrame' command to load SQL Table into a DataFrame, then display

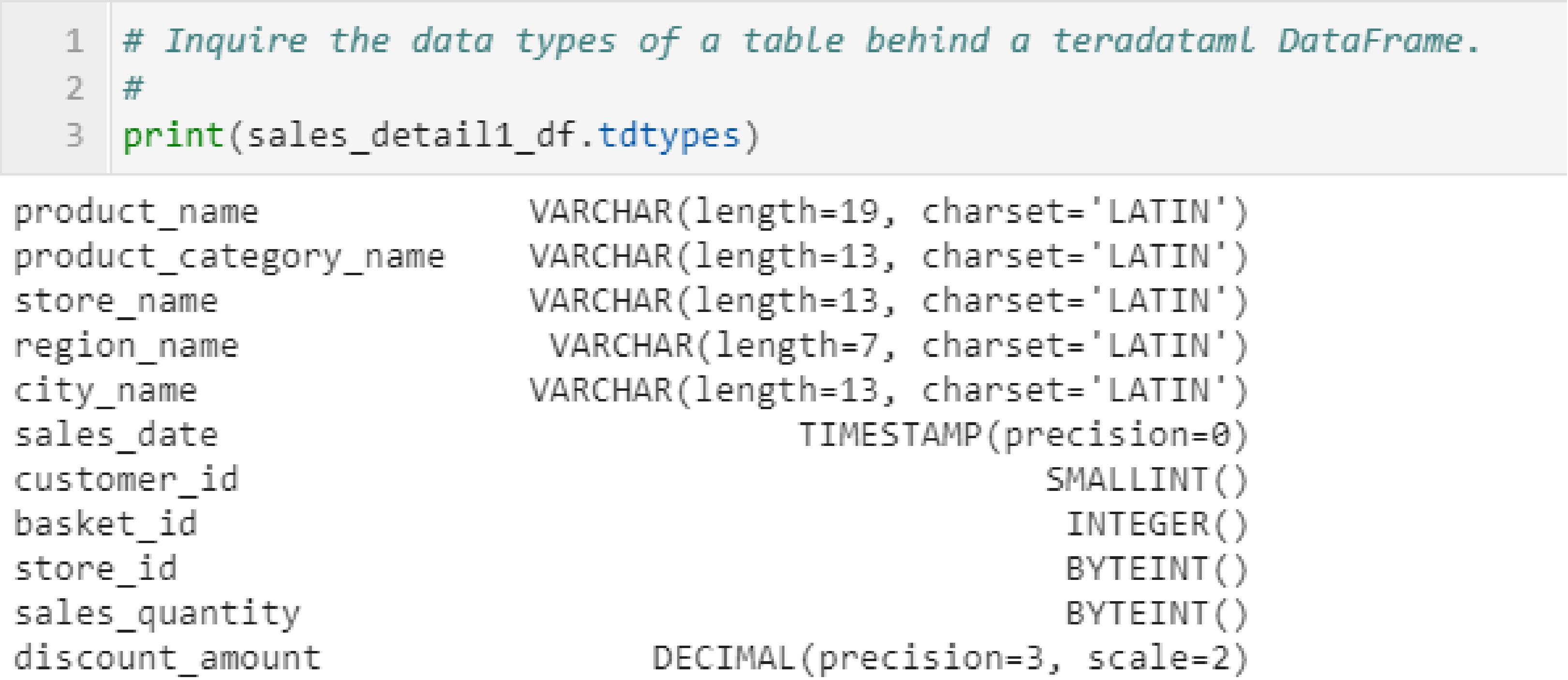


Lab 04: Load Data into DataFrame and Display Data

```
## Load SQL table into DataFrame, then display 10 rows
sales_detail1_df = DataFrame('TRNG_TDU_TD01.sales_detail1').select(['product_name', 'basket_id', 'region_name'])

# Display first 10 rows
sales_detail1_df.head()
```

```
  product_name  basket_id region_name
0      Almonds      53366     Eastern
1      Almonds    2641888     Western
2      Almonds    2642633     Western
3  Bagel chips     161444     Western
4  Bagel chips    1492022     Western
5  Bagel chips     264288     Western
6  Bagel chips    1720010     Eastern
7      Almonds    2761444     Western
8      Almonds      65644     Western
9      Almonds     161744     Western
```

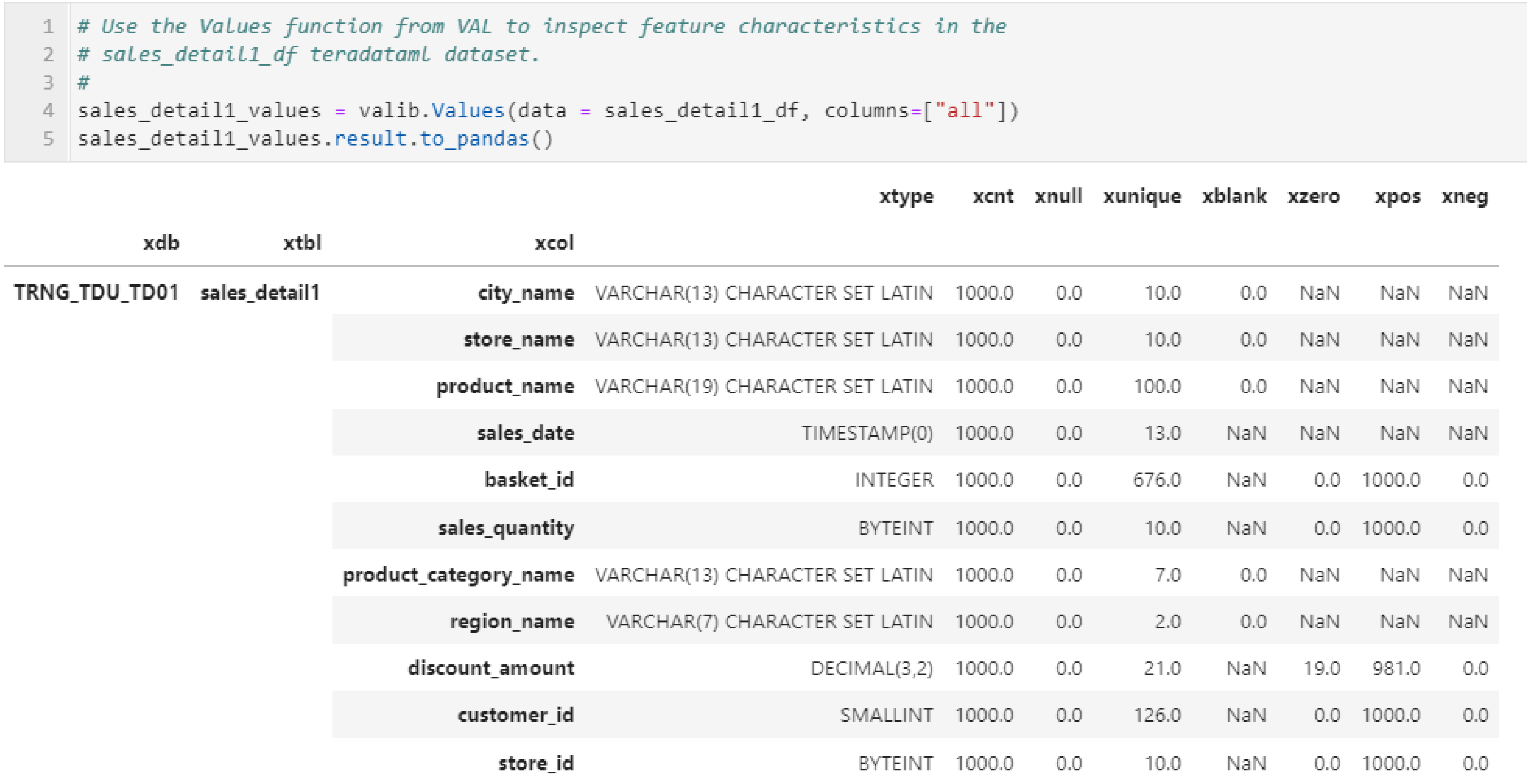
Lab 14b: Load Table into Dataframe and inspect



```
# Inquire the data types of a table behind a teradataml DataFrame.
#
print(sales_detail1_df.tdtypes)
```

```
product_name             VARCHAR(length=19, charset='LATIN')
product_category_name    VARCHAR(length=13, charset='LATIN')
store_name               VARCHAR(length=13, charset='LATIN')
region_name               VARCHAR(length=7, charset='LATIN')
city_name                VARCHAR(length=13, charset='LATIN')
sales_date                            TIMESTAMP(precision=0)
customer_id                                       SMALLINT()
basket_id                                          INTEGER()
store_id                                           BYTEINT()
sales_quantity                                     BYTEINT()
discount_amount                DECIMAL(precision=3, scale=2)
```

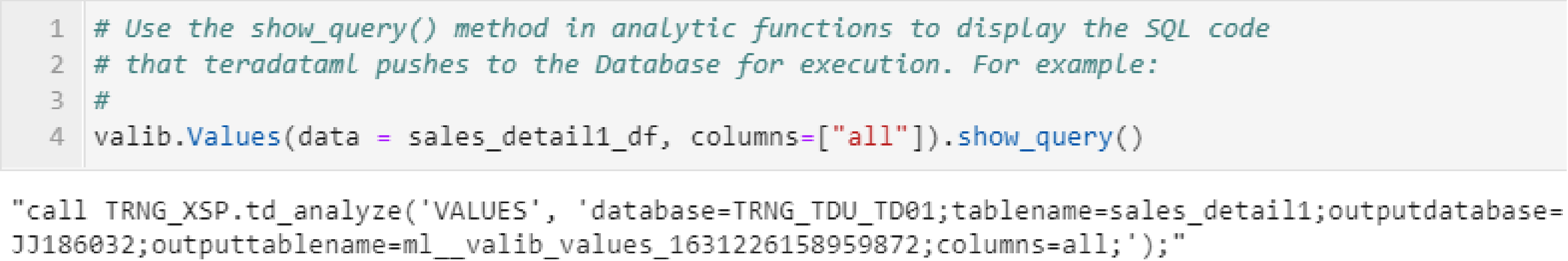
Lab 14c: VAL Values



```python
# Use the Values function from VAL to inspect feature characteristics in the
# sales_detail1_df teradataml dataset.
#
sales_detail1_values = valib.Values(data = sales_detail1_df, columns=["all"])
sales_detail1_values.result.to_pandas()
```

			xtype	xcnt	xnull	xunique	xblank	xzero	xpos	xneg
xdb	**xtbl**	**xcol**								
TRNG_TDU_TD01	**sales_detail1**	**city_name**	VARCHAR(13) CHARACTER SET LATIN	1000.0	0.0	10.0	0.0	NaN	NaN	NaN
		store_name	VARCHAR(13) CHARACTER SET LATIN	1000.0	0.0	10.0	0.0	NaN	NaN	NaN
		product_name	VARCHAR(19) CHARACTER SET LATIN	1000.0	0.0	100.0	0.0	NaN	NaN	NaN
		sales_date	TIMESTAMP(0)	1000.0	0.0	13.0	NaN	NaN	NaN	NaN
		basket_id	INTEGER	1000.0	0.0	676.0	NaN	0.0	1000.0	0.0
		sales_quantity	BYTEINT	1000.0	0.0	10.0	NaN	0.0	1000.0	0.0
		product_category_name	VARCHAR(13) CHARACTER SET LATIN	1000.0	0.0	7.0	0.0	NaN	NaN	NaN
		region_name	VARCHAR(7) CHARACTER SET LATIN	1000.0	0.0	2.0	0.0	NaN	NaN	NaN
		discount_amount	DECIMAL(3,2)	1000.0	0.0	21.0	NaN	19.0	981.0	0.0
		customer_id	SMALLINT	1000.0	0.0	126.0	NaN	0.0	1000.0	0.0
		store_id	BYTEINT	1000.0	0.0	10.0	NaN	0.0	1000.0	0.0

Lab 14d: VAL show.query()



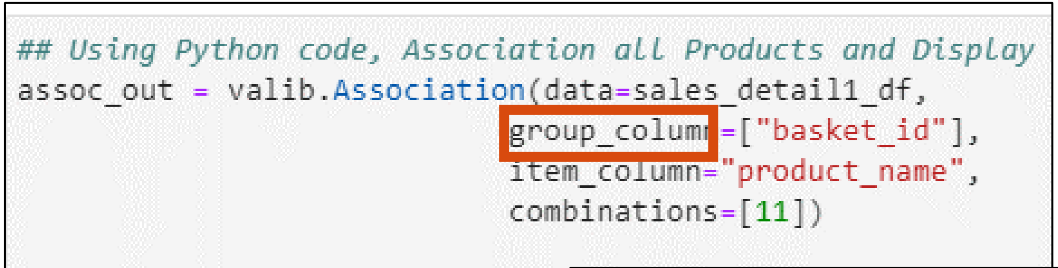
```
# Use the show_query() method in analytic functions to display the SQL code
# that teradataml pushes to the Database for execution. For example:
#
valib.Values(data = sales_detail1_df, columns=["all"]).show_query()
```

```
"call TRNG_XSP.td_analyze('VALUES', 'database=TRNG_TDU_TD01;tablename=sales_detail1;outputdatabase=
JJ186032;outputtablename=ml__valib_values_1631226158959872;columns=all;');"
```

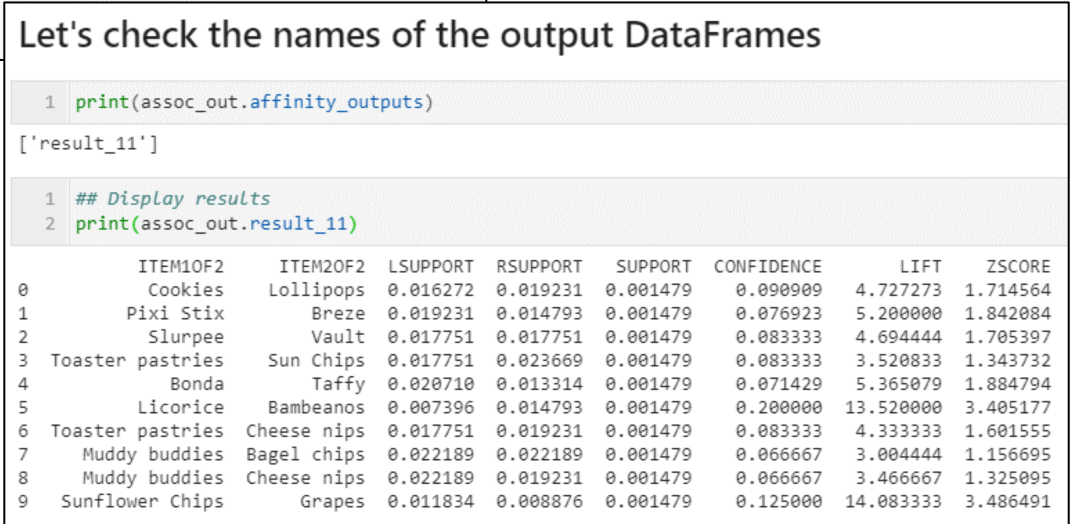
SQL conversion

Lab 15: Association on the Data

First, we run Association against all the Data



```
## Using Python code, Association all Products and Display
assoc_out = valib.Association(data=sales_detail1_df,
                              group_column=["basket_id"],
                              item_column="product_name",
                              combinations=[11])
```



Let's check the names of the output DataFrames

```
print(assoc_out.affinity_outputs)
```

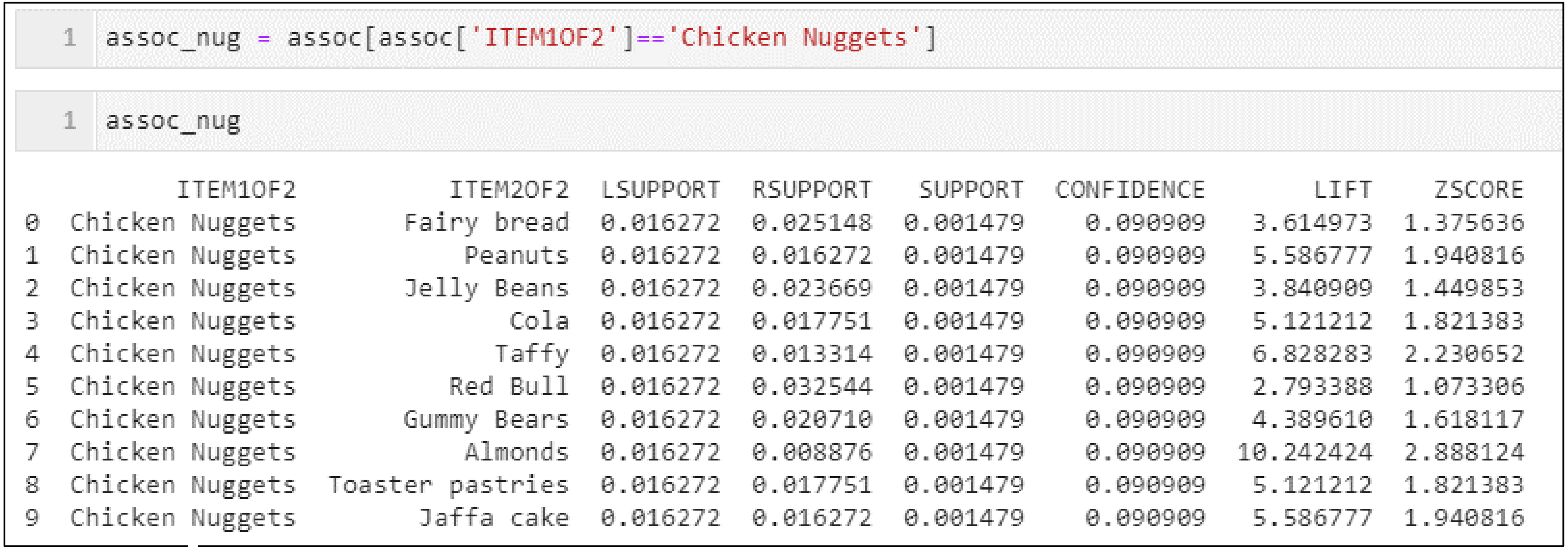
```
['result_11']
```

```
## Display results
print(assoc_out.result_11)
```

```
         ITEM1OF2     ITEM2OF2  LSUPPORT  RSUPPORT   SUPPORT  CONFIDENCE       LIFT    ZSCORE
0         Cookies    Lollipops  0.016272  0.019231  0.001479    0.090909   4.727273  1.714564
1       Pixi Stix        Breze  0.019231  0.014793  0.001479    0.076923   5.200000  1.842084
2         Slurpee        Vault  0.017751  0.017751  0.001479    0.083333   4.694444  1.705397
3  Toaster pastries    Sun Chips  0.017751  0.023669  0.001479    0.083333   3.520833  1.343732
4           Bonda        Taffy  0.020710  0.013314  0.001479    0.071429   5.365079  1.884794
5         Licorice    Bambeanos  0.007396  0.014793  0.001479    0.200000  13.520000  3.405177
6  Toaster pastries  Cheese nips  0.017751  0.019231  0.001479    0.083333   4.333333  1.601555
7    Muddy buddies  Bagel chips  0.022189  0.022189  0.001479    0.066667   3.004444  1.156695
8    Muddy buddies  Cheese nips  0.022189  0.019231  0.001479    0.066667   3.466667  1.325095
9   Sunflower Chips       Grapes  0.011834  0.008876  0.001479    0.125000  14.083333  3.486491
```

Lab 15: Association on the Data (cont.)

Select only Chicken Nuggets in first column of teradataml dataframe



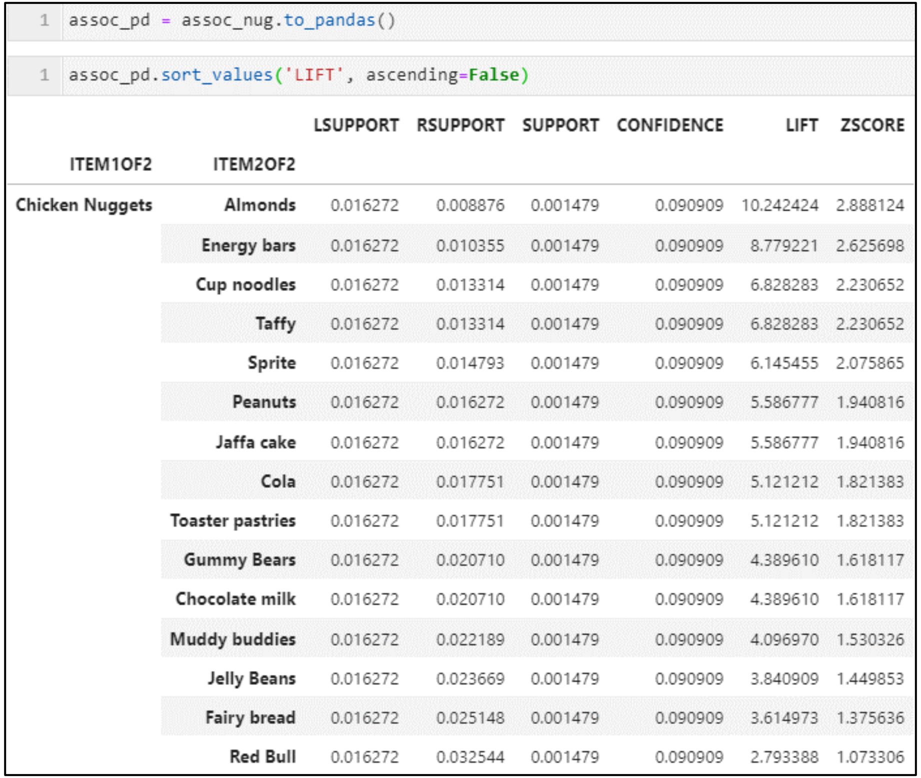
```
assoc_nug = assoc[assoc['ITEM1OF2']=='Chicken Nuggets']
```

```
assoc_nug
```

```
          ITEM1OF2           ITEM2OF2  LSUPPORT  RSUPPORT   SUPPORT  CONFIDENCE       LIFT    ZSCORE
0  Chicken Nuggets        Fairy bread  0.016272  0.025148  0.001479    0.090909   3.614973  1.375636
1  Chicken Nuggets            Peanuts  0.016272  0.016272  0.001479    0.090909   5.586777  1.940816
2  Chicken Nuggets        Jelly Beans  0.016272  0.023669  0.001479    0.090909   3.840909  1.449853
3  Chicken Nuggets               Cola  0.016272  0.017751  0.001479    0.090909   5.121212  1.821383
4  Chicken Nuggets              Taffy  0.016272  0.013314  0.001479    0.090909   6.828283  2.230652
5  Chicken Nuggets           Red Bull  0.016272  0.032544  0.001479    0.090909   2.793388  1.073306
6  Chicken Nuggets        Gummy Bears  0.016272  0.020710  0.001479    0.090909   4.389610  1.618117
7  Chicken Nuggets            Almonds  0.016272  0.008876  0.001479    0.090909  10.242424  2.888124
8  Chicken Nuggets  Toaster pastries  0.016272  0.017751  0.001479    0.090909   5.121212  1.821383
9  Chicken Nuggets         Jaffa cake  0.016272  0.016272  0.001479    0.090909   5.586777  1.940816
```

Lab 15: Association on the Data (cont.)

Convert teradataml dataframe to pandas dataframe and sort by LIFT



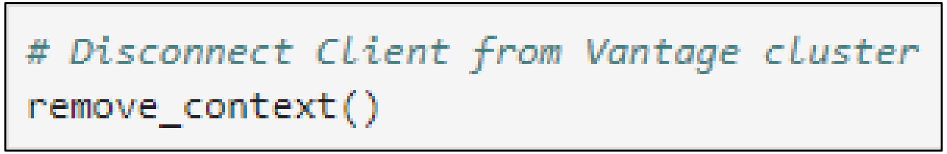
```
assoc_pd = assoc_nug.to_pandas()
```

```
assoc_pd.sort_values('LIFT', ascending=False)
```

		LSUPPORT	RSUPPORT	SUPPORT	CONFIDENCE	LIFT	ZSCORE
ITEM1OF2	ITEM2OF2						
Chicken Nuggets	Almonds	0.016272	0.008876	0.001479	0.090909	10.242424	2.888124
	Energy bars	0.016272	0.010355	0.001479	0.090909	8.779221	2.625698
	Cup noodles	0.016272	0.013314	0.001479	0.090909	6.828283	2.230652
	Taffy	0.016272	0.013314	0.001479	0.090909	6.828283	2.230652
	Sprite	0.016272	0.014793	0.001479	0.090909	6.145455	2.075865
	Peanuts	0.016272	0.016272	0.001479	0.090909	5.586777	1.940816
	Jaffa cake	0.016272	0.016272	0.001479	0.090909	5.586777	1.940816
	Cola	0.016272	0.017751	0.001479	0.090909	5.121212	1.821383
	Toaster pastries	0.016272	0.017751	0.001479	0.090909	5.121212	1.821383
	Gummy Bears	0.016272	0.020710	0.001479	0.090909	4.389610	1.618117
	Chocolate milk	0.016272	0.020710	0.001479	0.090909	4.389610	1.618117
	Muddy buddies	0.016272	0.022189	0.001479	0.090909	4.096970	1.530326
	Jelly Beans	0.016272	0.023669	0.001479	0.090909	3.840909	1.449853
	Fairy bread	0.016272	0.025148	0.001479	0.090909	3.614973	1.375636
	Red Bull	0.016272	0.032544	0.001479	0.090909	2.793388	1.073306

Lab 16: Remove Context

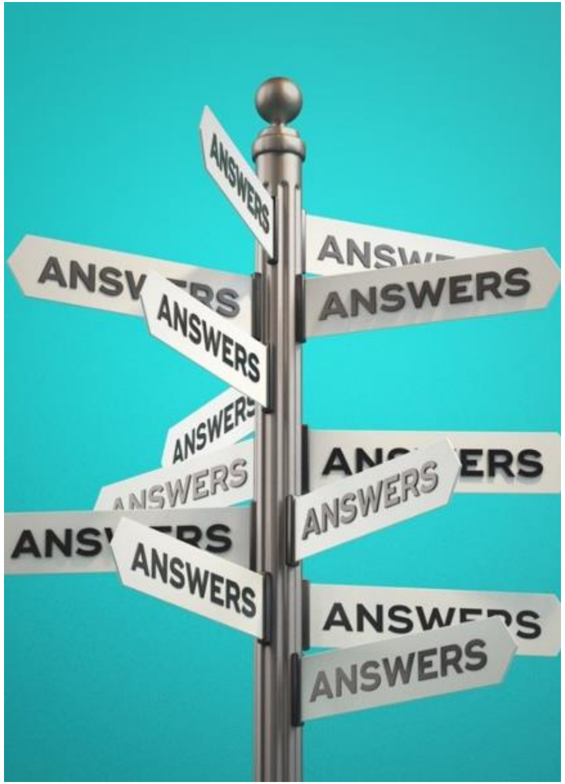
This command also does any Garbage Collection that is needed (removes Temporary tables)



```
# Disconnect Client from Vantage cluster
remove_context()
```

Current Topic – Review & Summary

- Use Case 01: Recommend shows for customers who watch 'Game of Thrones'
  - Sessionize
  - Attribution
  - Npath
- Use Case 02: Recommend products for customers who purchase Chicken Nuggets
  - VAL Association
- **Review & Summary**



Review & Summary

- In this module, we learned how to code in Python in the JupyterLab application
- We covered the following functions:
  - Sessionize
  - Attribution
  - nPath
  - Association

Thank you.

teradata.