etl

April 11, 2023

——————— transforming evWest data ———————

[ ]:
```
import pandas as pd
import numpy as np
```

[ ]:
```
evWest_initial_df = pd.read_csv("data/evWest.csv")
print("columns", evWest_initial_df.columns.to_list())
print("rows", len(evWest_initial_df))
evWest_initial_df.head(10)
```

```
columns ['category', 'model', 'model_name', 'manufacturer', 'weight', 'price']
rows 187
```

[ ]:

	category	model	\
0	Batteries	Lithium 2170 21700 Battery Cell\n5000mAh 14…	
1	Batteries	Lithium Ion Battery - 60.8V , 2.6kWh\nThese …	
2	Batteries	Lithium Super Cells 1.6 kWh - JH3\n63Ah 7S H…	
3	Batteries	Lithium Super Cells 3.2 kWh - JP3\n128Ah 2P7…	
4	Batteries	Samsung INR 18650 25R Lithium\nBattery Cells…	
5	Batteries	Samsung SDI 60Ah Lithium Ion\nBattery Cell\n…	
6	Batteries	Samsung SDI ESS Energy Storage\nBattery 16S …	
7	Batteries	Samsung SDI ESS Energy Storage\nBattery 22S …	
8	Batteries	Tesla	
9	Batteries	$799.00$990.00Tesla Smart Lithium Ion Batter…	

	model_name	manufacturer	weight	\
0	: INR2170M50L T\n	: EV West\n	: 0.00	
1	: BAT-2.6-\n16S\n	:\n	: 38.00	
2	: BAT-1.7-7S\n	: EV West\n	: 20.00	
3	: BAT-3.4-\n7S2P\n	:\n	: 40.00	
4	: INR25R\n	: Samsung\n	: 0.00	
5	: SM-SDI-60\n	: Samsung\n	: 5.00	
6	: ESS-3.5\n	: Samsung\n	: 110.00	
7	: ESS-7.6\n	: Samsung\n	: 140.00	
8	S Lithium Ion Battery\n18650 EV Module - 22.8…	: Tesla\n	: 55.00	
9	: 18650-3\n	: Tesla\n	: 42.00	

	price

```
0    : 0.00
1   : 38.00
2   : 20.00
3   : 40.00
4    : 0.00
5    : 5.00
6  : 110.00
7  : 140.00
8   : 55.00
9   : 42.00
```

removing unnessary chrachers and adjusting the data types

[ ]:
```
evWest_df_1= evWest_initial_df.replace(':','', regex=True)
#evWest_df_1.head()
evWest_df_2 = evWest_df_1.replace('\n','', regex=True)
# evWest_df_2.head(20)
evWest_df_3 = evWest_df_2.apply(lambda x : x.str.strip() if x.dtype == 'object'␣
 ↪else x)
evWest_df_3.dtypes
evWest_df_3[["price", "weight"]]= evWest_df_3[["price", "weight"]].
 ↪astype('float')
evWest_df_3.dtypes
evWest_df_3.head(10)
# evWest_df_3['manufacture_effective_date']= pd.Timestamp('2022-04-10')
evWest_df_3.head()
```

[ ]:
```
    category                                              model  \
0  Batteries  Lithium 2170 21700 Battery Cell5000mAh 14.4A  …
1  Batteries  Lithium Ion Battery - 60.8V , 2.6kWhThese batt…
2  Batteries  Lithium Super Cells 1.6 kWh - JH363Ah 7S High …
3  Batteries  Lithium Super Cells 3.2 kWh - JP3128Ah 2P7S Hi…
4  Batteries  Samsung INR 18650 25R LithiumBattery Cells - B…

      model_name manufacturer  weight  price
0  INR2170M50L T      EV West     0.0    0.0
1    BAT-2.6-16S                38.0   38.0
2     BAT-1.7-7S      EV West    20.0   20.0
3   BAT-3.4-7S2P                40.0   40.0
4         INR25R      Samsung     0.0    0.0
```

[ ]:
```
len(evWest_df_3)
```

[ ]: 187

adding vendor_since and product_release_Date

Sinse the pdf data is limited , I have populadated this fields using random dates , I made sure that

product_release_date is after the vendor since

[ ]:
```
evWest_df_4 = evWest_df_3.copy()
# evWest_df_4["manufacturer"]=evWest_df_4["manufacturer"].
 ↪mask(evWest_df_4["manufacturer"] == '')
evWest_df_4["manufacturer"]=evWest_df_4["manufacturer"].fillna("NULL")
unique_manufacturers = evWest_df_4.manufacturer.unique()
start_date = '2018-01-01'
end_date = '2023-04-11'
num_rows = len(evWest_df_3.manufacturer.unique())# number of rows in the␣
 ↪DataFrame
# Randomly sample dates from the sequence
# Generate a sequence of dates within the range of years
dates = pd.date_range(start=start_date, end=end_date)
random_dates = np.random.choice(dates, size=len(unique_manufacturers),␣
 ↪replace=True)


# Create a new DataFrame with "manufacturer" and "vendor_since" columns
df_man = pd.DataFrame({"manufacturer": unique_manufacturers, "vendor_since":␣
 ↪random_dates})

# Merge the new DataFrame with the original DataFrame
evWest_df_5 = pd.merge(evWest_df_4, df_man, on="manufacturer", how="left")
# evWest_df_5.columns
# len(evWest_df_5)

#define a function to generate random dates
def random_date(start, end):
    return pd.to_datetime(np.random.randint(start.value, end.value, size=1)[0],␣
 ↪unit='ns')

# product release date
evWest_df_5["product_release_Date"] =   evWest_df_5.apply(lambda row:␣
 ↪random_date(row['vendor_since'], pd.Timestamp.now()), axis=1)
evWest_df_5.head(20)
```

[ ]:

	category	model	\
0	Batteries	Lithium 2170 21700 Battery Cell5000mAh 14.4A	…
1	Batteries	Lithium Ion Battery - 60.8V , 2.6kWhThese batt…	
2	Batteries	Lithium Super Cells 1.6 kWh - JH363Ah 7S High …	
3	Batteries	Lithium Super Cells 3.2 kWh - JP3128Ah 2P7S Hi…	
4	Batteries	Samsung INR 18650 25R LithiumBattery Cells - B…	
5	Batteries	Samsung SDI 60Ah Lithium IonBattery CellThis i…	
6	Batteries	Samsung SDI ESS Energy StorageBattery 16S 60 V…	
7	Batteries	Samsung SDI ESS Energy StorageBattery 22S 80 V…	
8	Batteries	Tesla	

```
9              Batteries  $799.00$990.00Tesla Smart Lithium Ion Battery1…
10  BatteryEnclosuresAcc  MX150 Bulkhead and Connector -Battery Box CAN …
11  BatteryEnclosuresAcc  Rincon Power HVBD4AXR - 400 AContinuous - 1000…
12  BatteryEnclosuresAcc  VW Beetle Front Battery Box - '58-71Aluminum F…
13  BatteryEnclosuresAcc  VW Beetle Rear Battery Box - '58-71Aluminum Re…
14              Chargers  Elcon 3.3kW UF CAN Bus Chargerwith EVCCThe UFC…
15              Chargers  Elcon PFC1500 ChargerThis item has been discon…
16              Chargers  Manzanita Micro PFC 20-XMChargerThis product i…
17              Chargers  Manzanita Micro PFC 30-XMChargerThe PFC-30M is…
18              Chargers  Manzanita Micro PFC 40-XMChargerThe PFC-40M is…
19  Charging_Accessories  EVCC 3.0 - CAN Bus ChargerController by Dilith…

                                          model_name  manufacturer  weight  \
0                                      INR2170M50L T       EV West     0.0
1                                        BAT-2.6-16S                  38.0
2                                        BAT-1.7-7S       EV West    20.0
3                                       BAT-3.4-7S2P                  40.0
4                                             INR25R       Samsung     0.0
5                                          SM-SDI-60       Samsung     5.0
6                                            ESS-3.5       Samsung   110.0
7                                            ESS-7.6       Samsung   140.0
8   S Lithium Ion Battery18650 EV Module - 22.8 V …         Tesla    55.0
9                                            18650-3         Tesla    42.0
10                                        EVW -MX150       EV West     1.0
11                                           RP-HVBD   RinconPower     3.0
12                                        EVW -FBB1       EV West    20.0
13                                        EVW -RBB1       EV West    20.0
14                                        UFC33-CAN         Elcon    19.0
15                                           ELC1500         Elcon    15.0
16                                       MM PFC20-XM     Manzanita    16.0
17                                       MM PFC30-XM     Manzanita    17.2
18                                       MM PFC40-XM     Manzanita    20.0
19                                          TS-EVCC3  DilithiumBMS     1.0

    price vendor_since          product_release_Date
0     0.0   2018-09-02 2023-03-20 10:26:34.972912087
1    38.0   2020-12-24 2021-11-06 03:53:27.751711652
2    20.0   2018-09-02 2021-06-29 00:42:43.602005710
3    40.0   2020-12-24 2022-02-23 05:49:15.271129746
4     0.0   2021-08-28 2022-11-24 03:52:21.507046407
5     5.0   2021-08-28 2023-03-18 06:16:21.504488165
6   110.0   2021-08-28 2023-03-05 03:19:07.004449389
7   140.0   2021-08-28 2022-02-23 23:30:26.732679502
8    55.0   2022-09-25 2023-02-25 02:32:25.631017830
9    42.0   2022-09-25 2022-10-08 04:33:13.889636927
10    1.0   2018-09-02 2019-01-13 08:35:05.664007469
11    3.0   2022-09-28 2022-11-24 02:34:40.772600749
```

```
12   20.0   2018-09-02 2021-12-26 20:55:39.946738725
13   20.0   2018-09-02 2021-06-28 19:13:47.906863544
14   19.0   2021-03-19 2023-03-03 15:25:04.880349576
15   15.0   2021-03-19 2021-06-26 20:13:54.329377143
16   16.0   2019-09-18 2022-12-01 07:10:17.633051141
17   17.2   2019-09-18 2021-02-11 23:47:40.393722434
18   20.0   2019-09-18 2022-10-24 12:32:56.450851859
19    1.0   2019-05-05 2022-05-15 15:55:44.739341712
```

adding product status flag, in here we can think as a business rule if it has been more than 2 years since the product release that this product is no longer available

[ ]:
```
def status_flag_maker(product_release_Date):
    #a= evWest_df_4["product_release_Date"][0]
    # b = pd.Timestamp.now()
    # diff = b - a
    # diff= diff.days
    #730 is two years
    current_date = pd.Timestamp.now()
    diff = product_release_Date- current_date
    diff = diff.days
    if diff > 730:
        status = "inactive"
        # print("inactive")
    else:
        status = "active"
        # print("active")
    return status
```

we also add in here product updated date scd field as same as the product release date

[ ]:
```
evWest_df_5.apply(lambda row: random_date(row['vendor_since'], pd.Timestamp.
 ↪now()), axis=1)
evWest_df_5["product_status_flag"] = evWest_df_5.apply(lambda row:␣
 ↪status_flag_maker(row["product_release_Date"]), axis=1)
# initial values we set the product release date
evWest_df_5["product_updated_at"] = evWest_df_5["product_release_Date"]
evWest_df_5.head()
```

[ ]:
```
    category                                              model  \
0  Batteries  Lithium 2170 21700 Battery Cell5000mAh 14.4A  …
1  Batteries  Lithium Ion Battery - 60.8V , 2.6kWhThese batt…
2  Batteries  Lithium Super Cells 1.6 kWh - JH363Ah 7S High …
3  Batteries  Lithium Super Cells 3.2 kWh - JP3128Ah 2P7S Hi…
4  Batteries  Samsung INR 18650 25R LithiumBattery Cells - B…

      model_name manufacturer  weight  price vendor_since  \
```

```
0  INR2170M50L T      EV West     0.0    0.0   2018-09-02
1    BAT-2.6-16S                 38.0   38.0   2020-12-24
2     BAT-1.7-7S      EV West    20.0   20.0   2018-09-02
3   BAT-3.4-7S2P                 40.0   40.0   2020-12-24
4         INR25R      Samsung     0.0    0.0   2021-08-28

           product_release_Date product_status_flag  \
0 2023-03-20 10:26:34.972912087              active
1 2021-11-06 03:53:27.751711652              active
2 2021-06-29 00:42:43.602005710              active
3 2022-02-23 05:49:15.271129746              active
4 2022-11-24 03:52:21.507046407              active

             product_updated_at
0 2023-03-20 10:26:34.972912087
1 2021-11-06 03:53:27.751711652
2 2021-06-29 00:42:43.602005710
3 2022-02-23 05:49:15.271129746
4 2022-11-24 03:52:21.507046407
```

I am going to base my scd fields management based on the logic below. - company sells same product for only two years, so if the product release so if there has been two years since the product release date we assign inactive and viseversa - manufacturer_status: if the manufacturer does not have active product we will set the manufacturer status inactive and vise versa active

———————— LOAD———————————-

0.1 creating manufacturer dimensions

1- find unique manufacturers

2- create id

3- set manufacture status active

[ ]:
```
evWest_df_5["manufacturer"].isna().sum()
```

[ ]: 0

[ ]:
```
evWest_df_5["manufacturer"]=evWest_df_5["manufacturer"].
 ↪mask(evWest_df_5["manufacturer"] == '')
evWest_df_5["model_name"]= evWest_df_5["model_name"].
 ↪mask(evWest_df_5["model_name"] == '')
```

[ ]:
```
print(evWest_df_5["model_name"].isna().sum())
evWest_df_5["manufacturer"].isna().sum()
```

26

[ ]:
```
12
```

[ ]:
```
# first fill all emty values with null
evWest_df_6 = evWest_df_5.fillna("NULL")
evWest_df_6.isna().sum()
```

[ ]:
```
category                0
model                   0
model_name              0
manufacturer            0
weight                  0
price                   0
vendor_since            0
product_release_Date    0
product_status_flag     0
product_updated_at      0
dtype: int64
```

[ ]:
```
evWest_df_6.columns
```

[ ]:
```
Index(['category', 'model', 'model_name', 'manufacturer', 'weight', 'price',
       'vendor_since', 'product_release_Date', 'product_status_flag',
       'product_updated_at'],
      dtype='object')
```

[ ]:
```
evWest_df_6["manufacturer"].value_counts()
```

[ ]:
```
EV West                 75
Canadian EV             24
HPEVS                   15
NULL                    12
Tesla                   11
Smart                    5
Elcon                    5
Manzanita                5
Curtis                   5
Samsung                  4
Deltec                   3
AEM                      3
NetGainMotors            3
AM Racing                2
TBSElectronics           2
RinehartMotionSystems    2
DilithiumBMS             2
QuickChargePower         1
Modular EV               1
Chennic                  1
```

```
MSD Ignition              1
SSBC                      1
Tyco Kilovac              1
EmproShunts               1
RinconPower               1
Behr                      1
Name: manufacturer, dtype: int64
```

[ ]:
```
distinct_manufacturer = evWest_df_6.drop_duplicates(subset=["manufacturer",␣
 ↪"vendor_since"], keep='first')
distinct_manufacturer = distinct_manufacturer.reset_index(drop=True)
distinct_manufacturer["manufacture_id"]=distinct_manufacturer.index+1
distinct_manufacturer= distinct_manufacturer[["manufacture_id","manufacturer",␣
 ↪"vendor_since"]]
#setting all initial values are as active
distinct_manufacturer["manu_status_flag"]= "active"
#setting updated time to the vendor since
distinct_manufacturer["manu_updated_at"]= distinct_manufacturer["vendor_since"]
#initial sk we just set to the index
distinct_manufacturer= distinct_manufacturer.reset_index(drop=True)
distinct_manufacturer["sk_manufature"]=distinct_manufacturer.index+1
distinct_manufacturer
```

[ ]:

	manufacture_id	manufacturer	vendor_since	manu_status_flag	\
0	1	EV West	2018-09-02	active	
1	2	NULL	2020-12-24	active	
2	3	Samsung	2021-08-28	active	
3	4	Tesla	2022-09-25	active	
4	5	RinconPower	2022-09-28	active	
5	6	Elcon	2021-03-19	active	
6	7	Manzanita	2019-09-18	active	
7	8	DilithiumBMS	2019-05-05	active	
8	9	Chennic	2021-04-12	active	
9	10	Modular EV	2018-12-21	active	
10	11	QuickChargePower	2019-02-19	active	
11	12	HPEVS	2018-12-09	active	
12	13	Curtis	2021-09-27	active	
13	14	RinehartMotionSystems	2023-03-23	active	
14	15	NetGainMotors	2022-03-07	active	
15	16	Deltec	2019-09-19	active	
16	17	AEM	2023-02-22	active	
17	18	TBSElectronics	2020-03-30	active	
18	19	Canadian EV	2021-07-27	active	
19	20	MSD Ignition	2020-02-09	active	
20	21	SSBC	2022-09-09	active	
21	22	AM Racing	2019-03-25	active	
22	23	Tyco Kilovac	2019-10-31	active	

```
23              24            EmproShunts   2021-06-18          active
24              25                  Smart   2022-06-02          active
25              26                   Behr   2019-08-22          active
```

```
   manu_updated_at  sk_manufature
0       2018-09-02              1
1       2020-12-24              2
2       2021-08-28              3
3       2022-09-25              4
4       2022-09-28              5
5       2021-03-19              6
6       2019-09-18              7
7       2019-05-05              8
8       2021-04-12              9
9       2018-12-21             10
10      2019-02-19             11
11      2018-12-09             12
12      2021-09-27             13
13      2023-03-23             14
14      2022-03-07             15
15      2019-09-19             16
16      2023-02-22             17
17      2020-03-30             18
18      2021-07-27             19
19      2020-02-09             20
20      2022-09-09             21
21      2019-03-25             22
22      2019-10-31             23
23      2021-06-18             24
24      2022-06-02             25
25      2019-08-22             26
```

1 adding product field

All the product names shodul be unique, I noticed an issue in the pdf extraction, I will have to go back and fix the issue so that model names are unique

[ ]:
```
distinct_product = evWest_df_6.
 ↪drop_duplicates(subset=["model_name","model","category","product_status_flag","product_upda
 ↪keep='first')
distinct_product  = distinct_product.reset_index(drop=True)
distinct_product["product_id"]=distinct_product.index+1
distinct_product =␣
 ↪distinct_product[["product_id","model_name","model","category","product_status_flag","produ
distinct_product= distinct_product.reset_index(drop=True)
distinct_product["sk_product"]=distinct_product.index+1
```

```
distinct_product
distinct_product.head()
```

[ ]:
```
   product_id     model_name  \
0           1  INR2170M50L T
1           2    BAT-2.6-16S
2           3     BAT-1.7-7S
3           4   BAT-3.4-7S2P
4           5         INR25R

                                               model   category  \
0  Lithium 2170 21700 Battery Cell5000mAh 14.4A  …  Batteries
1  Lithium Ion Battery - 60.8V , 2.6kWhThese batt…  Batteries
2  Lithium Super Cells 1.6 kWh - JH363Ah 7S High …  Batteries
3  Lithium Super Cells 3.2 kWh - JP3128Ah 2P7S Hi…  Batteries
4  Samsung INR 18650 25R LithiumBattery Cells - B…  Batteries

  product_status_flag            product_updated_at  sk_product
0              active 2023-03-20 10:26:34.972912087           1
1              active 2021-11-06 03:53:27.751711652           2
2              active 2021-06-29 00:42:43.602005710           3
3              active 2022-02-23 05:49:15.271129746           4
4              active 2022-11-24 03:52:21.507046407           5
```

here as we can see we have dublicates values, next week I will fix this issue

[ ]:
```
distinct = evWest_df_6.drop_duplicates(subset=["manufacturer",␣
↪"model_name","model","category","product_status_flag","product_updated_at"],␣
↪keep='first').sort_values(by = ["manufacturer", "category", 'model_name',␣
↪"weight", "price"])
# distinct= distinct.reset_index(drop=True)
# # we add +1 to index column to start from 1
# distinct['product_id']= distinct.index+1
distinct_df2= distinct.groupby(distinct["model_name"], as_index=False).size()
distinct_df2.loc[distinct_df2['size']>1]
# distinct
# reorg the fields
# products =distinct[["product_id", "model_name", 'model', 'category']]
# products.head()
```

[ ]:
```
                                    model_name  size
33                                  Cable Grip     4
34                                 ChargePlate     2
36                                  Controller     2
66                                     Fitting     5
73                                      Heater     2
90                                       Motor     5
```

```
92                                                  NULL    26
103  S Lithium Ion Battery18650 EV Module - 22.8 V …     2
123                                           Slip Yoke     2
```

creating the manufacture-fact table

we will bring manufacture id and product id

[ ]:
```
manufacturer_fact_a= evWest_df_6.merge(distinct_manufacturer,␣
 ↪on=["manufacturer", "vendor_since"] )
manufacturer_fact_a.head()
manufacturer_fact_b = manufacturer_fact_a.merge(distinct_product,␣
 ↪on=["model_name","model","category","product_status_flag","product_updated_at"]␣
 ↪)
manufacturer_fact_b.head()
manufacturer_fact_b  = manufacturer_fact_b.reset_index(drop=True)
manufacturer_fact_b["manu_fact_id"]=manufacturer_fact_b.index+1
```

[ ]:
```
print("rows", len(manufacturer_fact_b))
manufacturer_fact_b.columns
```

```
rows 187
```

[ ]:
```
Index(['category', 'model', 'model_name', 'manufacturer', 'weight', 'price',
       'vendor_since', 'product_release_Date', 'product_status_flag',
       'product_updated_at', 'manufacture_id', 'manu_status_flag',
       'manu_updated_at', 'sk_manufature', 'product_id', 'sk_product',
       'manu_fact_id'],
      dtype='object')
```

[ ]:
```
manufacturer_fact = manufacturer_fact_b[["manu_fact_id",␣
 ↪"manufacture_id","product_id", "weight"        ,"price" ]]
len(manufacturer_fact)
```

[ ]:
```
187
```

——————— LOAD ———————

connect the data base

[ ]:
```
# import psycopg2 as pg # PostgreSQL
# from psycopg2 import  extensions
# #establishing the connection
# auto_commit =extensions.ISOLATION_LEVEL_AUTOCOMMIT
# conn = pg.connect(
#     host="localhost",
#     database="postgres",
#     user="postgres",
#     password="arnold")
```

```
# #Creating a cursor object using the cursor() method
# conn.set_isolation_level(auto_commit)
# cursor = conn.cursor()
# query = "CREATE database partsUnlimited"
# #Creating a database
# cursor.execute(query)
# print("Database created successfully........")

# #Closing the connection
# conn.close()
```

creating manufacturers table

[ ]:
```
distinct_manufacturer.columns
```

[ ]:
```
Index(['manufacture_id', 'manufacturer', 'vendor_since', 'manu_status_flag',
       'manu_updated_at', 'sk_manufature'],
      dtype='object')
```

[ ]:
```
distinct_manufacturer.rename(columns={'manufacturer':"manufacturer_name"})
```

[ ]:

	manufacture_id	manufacturer_name	vendor_since	manu_status_flag	\
0	1	EV West	2018-01-04	active	
1	2	NULL	2019-10-21	active	
2	3	Samsung	2020-01-24	active	
3	4	Tesla	2020-03-08	active	
4	5	RinconPower	2019-10-12	active	
5	6	Elcon	2020-05-24	active	
6	7	Manzanita	2020-01-31	active	
7	8	DilithiumBMS	2021-12-11	active	
8	9	Chennic	2019-04-09	active	
9	10	Modular EV	2020-11-22	active	
10	11	QuickChargePower	2019-10-14	active	
11	12	HPEVS	2020-04-11	active	
12	13	Curtis	2021-04-14	active	
13	14	RinehartMotionSystems	2022-04-02	active	
14	15	NetGainMotors	2020-09-02	active	
15	16	Deltec	2018-01-01	active	
16	17	AEM	2021-01-20	active	
17	18	TBSElectronics	2018-02-27	active	
18	19	Canadian EV	2019-06-10	active	
19	20	MSD Ignition	2020-04-05	active	
20	21	SSBC	2022-07-27	active	
21	22	AM Racing	2022-08-08	active	
22	23	Tyco Kilovac	2018-07-24	active	
23	24	EmproShunts	2023-03-30	active	
24	25	Smart	2019-06-17	active	

```
25              26                    Behr   2022-12-10           active

   manu_updated_at  sk_manufature
0       2018-01-04              1
1       2019-10-21              2
2       2020-01-24              3
3       2020-03-08              4
4       2019-10-12              5
5       2020-05-24              6
6       2020-01-31              7
7       2021-12-11              8
8       2019-04-09              9
9       2020-11-22             10
10      2019-10-14             11
11      2020-04-11             12
12      2021-04-14             13
13      2022-04-02             14
14      2020-09-02             15
15      2018-01-01             16
16      2021-01-20             17
17      2018-02-27             18
18      2019-06-10             19
19      2020-04-05             20
20      2022-07-27             21
21      2022-08-08             22
22      2018-07-24             23
23      2023-03-30             24
24      2019-06-17             25
25      2022-12-10             26
```

[ ]:
```
distinct_manufacturer.columns
```

[ ]:
```
Index(['manufacture_id', 'manufacturer', 'vendor_since', 'manu_status_flag',
       'manu_updated_at', 'sk_manufature'],
      dtype='object')
```

[ ]:
```
import psycopg2 as pg # PostgreSQL
# from psycopg2 import  extensions
#establishing the connection
conn = pg.connect(
    host="localhost",
    database="partsunlimited",
    user="postgres",
    password="arnold")
#Creating a cursor object using the cursor() method
cursor = conn.cursor()
#Droping if the table if already exists.
```

```
cursor.execute("DROP TABLE IF EXISTS manufacturers")

#Creating table as per requirement
query ='''CREATE TABLE manufacturers(
    manufacture_id integer UNIQUE NOT NULL,
    sk_manufature integer,
   manufacturer_name VARCHAR(250),
   manu_status_flag VARCHAR(50) ,
   vendor_since DATE,
   manu_updated_at DATE ,
   PRIMARY KEY(manufacture_id)
)'''

##Creating a database
cursor.execute(query)
print("Table has been created successfully........")
conn.commit()
# Get the updated list of tables
sqlGetTableList = "SELECT table_schema,table_name FROM information_schema.
 ↪tables where table_schema='public' ORDER BY table_schema,table_name ;"
#sqlGetTableList = "\dt"


# Retrieve all the rows from the cursor

cursor.execute(sqlGetTableList)
tables = cursor.fetchall()


# Print the names of the tables
print("list of tables in partsUnlimited database ")
print([table for table in tables])

#Closing the connection
cursor.close()
conn.close()
```

```
Table has been created successfully…
list of tables in partsUnlimited database
[('public', 'manufacturers'), ('public', 'products')]
```

Loading the manufacture data

[ ]:
```
from sqlalchemy import create_engine
# establish connections
conn_string = 'postgresql://postgres:arnold@localhost/partsunlimited'

db = create_engine(conn_string)
```

```
conn = db.connect()
print
#converting data to sql
distinct_manufacturer.to_sql('manufacturers', conn, if_exists= 'replace',␣
 ↪index=False)
conn.commit()
db.dispose()
conn.close()
```

[ ]:
```
import psycopg2 as pg # PostgreSQL
# from psycopg2 import  extensions
#establishing the connection
conn = pg.connect(
    host="localhost",
    database="partsunlimited",
    user="postgres",
    password="arnold")
#Creating a cursor object using the cursor() method
cursor = conn.cursor()
# create the SQL query to insert the data into the table
query = " select * from manufacturers LIMIT 5"
cursor.execute(query)
tables = cursor.fetchall()

temp = pd.DataFrame(tables, columns=['manufacture_id', 'manufacturer',␣
 ↪'vendor_since', 'manu_status_flag',
       'manu_updated_at', 'sk_manufature'])
print(temp)
cursor.close()
conn.close()
```

```
   manufacture_id manufacturer vendor_since manu_status_flag manu_updated_at  \
0               1      EV West   2018-01-04           active      2018-01-04
1               2         NULL   2019-10-21           active      2019-10-21
2               3      Samsung   2020-01-24           active      2020-01-24
3               4        Tesla   2020-03-08           active      2020-03-08
4               5  RinconPower   2019-10-12           active      2019-10-12

   sk_manufature
0              1
1              2
2              3
3              4
4              5
```

creating product table

[ ]:
```
distinct_product.columns
```

[ ]:
```
Index(['product_id', 'model_name', 'model', 'category', 'product_status_flag',
       'product_updated_at', 'sk_product'],
      dtype='object')
```

[ ]:
```
distinct_product = distinct_product.rename(columns={'model':
 ↪"product_description","model_name":"product_name", "category":␣
 ↪"product_category"})
```

[ ]:
```
distinct_product.columns
```

[ ]:
```
Index(['product_id', 'product_name', 'product_description', 'product_category',
       'product_status_flag', 'product_updated_at', 'sk_product'],
      dtype='object')
```

[ ]:
```
import psycopg2 as pg # PostgreSQL
# from psycopg2 import  extensions
#establishing the connection
conn = pg.connect(
    host="localhost",
    database="partsunlimited",
    user="postgres",
    password="arnold")
#Creating a cursor object using the cursor() method
cursor = conn.cursor()
#Droping if the table if already exists.
cursor.execute("DROP TABLE IF EXISTS products")

#Creating table as per requirement
query ='''CREATE TABLE products(
    product_id integer UNIQUE NOT NULL,
    sk_product integer,
   product_name VARCHAR(2000),
   product_description VARCHAR(2000) ,
   product_category VARCHAR(50),
   product_status_flag VARCHAR(50),
   product_updated_at DATE,
   PRIMARY KEY(product_id)
)'''

##Creating a database
cursor.execute(query)
print("Table has been created successfully........")
conn.commit()
# Get the updated list of tables
sqlGetTableList = "SELECT table_schema,table_name FROM information_schema.
 ↪tables where table_schema='public' ORDER BY table_schema,table_name ;"
#sqlGetTableList = "\dt"
```

```
# Retrieve all the rows from the cursor

cursor.execute(sqlGetTableList)
tables = cursor.fetchall()


# Print the names of the tables
print("list of tables in partsUnlimited database ")
print([table for table in tables])

#Closing the connection
cursor.close()
conn.close()
```

```
Table has been created successfully…
list of tables in partsUnlimited database
[('public', 'manufacturers'), ('public', 'products')]
```

loading to the product table

[ ]:
```
from sqlalchemy import create_engine
# establish connections
conn_string = 'postgresql://postgres:arnold@localhost/partsunlimited'

db = create_engine(conn_string)
conn = db.connect()
print
#converting data to sql
distinct_product.to_sql('products', conn, if_exists= 'replace', index=False)
conn.commit()
db.dispose()
conn.close()
```

[ ]:
```
import psycopg2 as pg # PostgreSQL
# from psycopg2 import  extensions
#establishing the connection
conn = pg.connect(
    host="localhost",
    database="partsunlimited",
    user="postgres",
    password="arnold")
#Creating a cursor object using the cursor() method
cursor = conn.cursor()
# create the SQL query to insert the data into the table
query = " select * from products LIMIT 5"
cursor.execute(query)
```

```
tables = cursor.fetchall()

temp = pd.DataFrame(tables, columns=['product_id', 'product_name',␣
 ↪'product_description', 'product_category',
       'product_status_flag', 'product_updated_at', 'sk_product'])
print(temp)
cursor.close()
conn.close()
```

```
   product_id   product_name  \
0           1  INR2170M50L T
1           2    BAT-2.6-16S
2           3     BAT-1.7-7S
3           4   BAT-3.4-7S2P
4           5         INR25R

                              product_description product_category  \
0  Lithium 2170 21700 Battery Cell5000mAh 14.4A  …        Batteries
1  Lithium Ion Battery - 60.8V , 2.6kWhThese batt…        Batteries
2  Lithium Super Cells 1.6 kWh - JH363Ah 7S High …        Batteries
3  Lithium Super Cells 3.2 kWh - JP3128Ah 2P7S Hi…        Batteries
4  Samsung INR 18650 25R LithiumBattery Cells - B…        Batteries

  product_status_flag         product_updated_at  sk_product
0              active 2020-10-06 22:13:20.630156           1
1              active 2021-10-27 17:27:00.677187           2
2              active 2021-09-05 15:25:58.465624           3
3              active 2020-05-22 16:20:06.218943           4
4              active 2020-04-02 17:18:48.480817           5
```

creating manufacture fact table

[ ]:
```
manufacturer_fact.columns
```

[ ]:
```
Index(['manu_fact_id', 'manufacture_id', 'product_id', 'weight', 'price'],
dtype='object')
```

[ ]:
```
import psycopg2 as pg # PostgreSQL
# from psycopg2 import  extensions
#establishing the connection
conn = pg.connect(
    host="localhost",
    database="partsunlimited",
    user="postgres",
    password="arnold")
#Creating a cursor object using the cursor() method
cursor = conn.cursor()
```

```
# alterin the table for the constrain
query =''' ALTER TABLE manufacturers
ADD CONSTRAINT unique_manufacturer_id
UNIQUE (manufacture_id);
'''
##Creating a database
cursor.execute(query)
print("Table has been created successfully........")
conn.commit()
# Get the updated list of tables
sqlGetTableList = "SELECT table_schema,table_name FROM information_schema.
 ↪tables where table_schema='public' ORDER BY table_schema,table_name ;"
#sqlGetTableList = "\dt"
# Retrieve all the rows from the cursor

cursor.execute(sqlGetTableList)
tables = cursor.fetchall()


# Print the names of the tables
print("list of tables in partsUnlimited database ")
print([table for table in tables])

#Closing the connection
cursor.close()
conn.close()
```

```
Table has been created successfully…
list of tables in partsUnlimited database
[('public', 'manufacturers'), ('public', 'products')]
```

[ ]:
```
import psycopg2 as pg # PostgreSQL
# from psycopg2 import  extensions
#establishing the connection
conn = pg.connect(
    host="localhost",
    database="partsunlimited",
    user="postgres",
    password="arnold")
#Creating a cursor object using the cursor() method
cursor = conn.cursor()
#Droping if the table if already exists.

#altering the table for the constraint
query =''' ALTER TABLE products
ADD CONSTRAINT unique_product_id
UNIQUE (product_id);
```

```
'''
##Creating a database
cursor.execute(query)
print("Table has been created successfully........")
conn.commit()
# Get the updated list of tables
sqlGetTableList = "SELECT table_schema,table_name FROM information_schema.
 ↪tables where table_schema='public' ORDER BY table_schema,table_name ;"
#sqlGetTableList = "\dt"
# Retrieve all the rows from the cursor

cursor.execute(sqlGetTableList)
tables = cursor.fetchall()


# Print the names of the tables
print("list of tables in partsUnlimited database ")
print([table for table in tables])

#Closing the connection
cursor.close()
conn.close()
```

```
Table has been created successfully…
list of tables in partsUnlimited database
[('public', 'manufacturers'), ('public', 'products')]
```

[ ]:
```
import psycopg2 as pg # PostgreSQL
# from psycopg2 import  extensions
#establishing the connection
conn = pg.connect(
    host="localhost",
    database="partsunlimited",
    user="postgres",
    password="arnold")
#Creating a cursor object using the cursor() method
cursor = conn.cursor()
#Droping if the table if already exists.
cursor.execute("DROP TABLE IF EXISTS manufacturer_fact")

#Creating table as per requirement
query ='''CREATE TABLE manufacturer_fact(
    manu_fact_id integer UNIQUE NOT NULL PRIMARY KEY,
    manufacture_id integer,
    product_id integer,
    weight float,
    price float,
```

```
    CONSTRAINT fk_manufacture_id
        FOREIGN KEY (manufacture_id)
        REFERENCES manufacturers (manufacture_id),
    CONSTRAINT fk_products_id
        FOREIGN KEY (product_id)
        REFERENCES products (product_id)
)'''

##Creating a database
cursor.execute(query)
print("Table has been created successfully........")
conn.commit()
# Get the updated list of tables
sqlGetTableList = "SELECT table_schema,table_name FROM information_schema.
 ↪tables where table_schema='public' ORDER BY table_schema,table_name ;"
#sqlGetTableList = "\dt"


# Retrieve all the rows from the cursor

cursor.execute(sqlGetTableList)
tables = cursor.fetchall()


# Print the names of the tables
print("list of tables in partsUnlimited database ")
print([table for table in tables])

#Closing the connection
cursor.close()
conn.close()
```

```
Table has been created successfully…
list of tables in partsUnlimited database
[('public', 'manufacturer_fact'), ('public', 'manufacturers'), ('public',
'products')]
```

[ ]:
```
from sqlalchemy import create_engine
# establish connections
conn_string = 'postgresql://postgres:arnold@localhost/partsunlimited'

db = create_engine(conn_string)
conn = db.connect()
print
#converting data to sql
manufacturer_fact.to_sql('manufacturer_fact', conn, if_exists= 'replace',␣
 ↪index=False)
```

```
conn.commit()
db.dispose()
conn.close()
```

[ ]:
```
import psycopg2 as pg # PostgreSQL
# from psycopg2 import  extensions
#establishing the connection
conn = pg.connect(
    host="localhost",
    database="partsunlimited",
    user="postgres",
    password="arnold")
#Creating a cursor object using the cursor() method
cursor = conn.cursor()
# create the SQL query to insert the data into the table
query = " select * from manufacturer_fact LIMIT 5"
cursor.execute(query)
tables = cursor.fetchall()

temp = pd.DataFrame(tables, columns=['manu_fact_id', 'manufacture_id',␣
 ↪'product_id', 'weight', 'price'])
print(temp)
cursor.close()
conn.close()
```

	manu_fact_id	manufacture_id	product_id	weight	price
0	1	1	1	0.0	0.0
1	2	1	3	20.0	20.0
2	3	1	11	1.0	1.0
3	4	1	13	20.0	20.0
4	5	1	14	20.0	20.0