project1_dataWranglingPython_electronicStoreSales

August 16, 2023

1 Data Wrangling

Data wrangling describes a series of processes designed to explore, transform, and validate raw datasets from their messy and complex forms into high-quality data with good integrity and consistency into produce valuable insights and guide business decisions in later analytics purposes.

For this Data Wrangling Project, I'm using electronic store sales dataset from Kaggle: https://www.kaggle.com/datasets/saumaydhaundiyal/electronic-store-sales-data

For the steps I'm using for this project are: 1. Data Discovery 2. Data Cleaning 3. Data Transformation 4. Data Enriching 5. Data Validating 6. Data Publishing

1.1 Data Discovery

The main purposes in this step will be: - Importing necessary python library - Import data from our local machine - Gather useful insight & information for future step.

[1]:
```
#Import Library

import pandas as pd
import numpy as np
import os
import glob
```

[2]:
```
#Check and list all file contained in the folder
file_list=os.listdir("C:/Users/renal/Documents/Renaldo's File/Data Analyst␣
 ↪Portofolio -Renaldo Livando/Project1 Data Wrangling using Python/dataSource/
 ↪Electronic Stores Sales Raw Data 2019")
file_list
```

[2]:
```
['dummy dataset.csv',
 'Sales_April_2019.csv',
 'Sales_August_2019.csv',
 'Sales_December_2019.csv',
 'Sales_February_2019.csv',
 'Sales_January_2019.csv',
 'Sales_July_2019.csv',
 'Sales_June_2019.csv',
 'Sales_March_2019.csv',
```

```
 'Sales_May_2019.csv',
 'Sales_November_2019.csv',
 'Sales_October_2019.csv',
 'Sales_September_2019.csv',
 'z NOT A DATA TO IMPORT.csv']
```

[3]:
```
# Select only the file that we want to merge from file_list; -r" stands for raw␣
 ↪string
path1 = r"C:/Users/renal/Documents/Renaldo's File/Data Analyst Portofolio␣
 ↪-Renaldo Livando/Project1 Data Wrangling using Python/dataSource/Electronic␣
 ↪Stores Sales Raw Data 2019"
all_files = glob.glob(os.path.join(path1 , "Sales*.csv"))
all_files
```

[3]:
```
["C:/Users/renal/Documents/Renaldo's File/Data Analyst Portofolio -Renaldo
Livando/Project1 Data Wrangling using Python/dataSource/Electronic Stores Sales
Raw Data 2019\\Sales_April_2019.csv",
 "C:/Users/renal/Documents/Renaldo's File/Data Analyst Portofolio -Renaldo
Livando/Project1 Data Wrangling using Python/dataSource/Electronic Stores Sales
Raw Data 2019\\Sales_August_2019.csv",
 "C:/Users/renal/Documents/Renaldo's File/Data Analyst Portofolio -Renaldo
Livando/Project1 Data Wrangling using Python/dataSource/Electronic Stores Sales
Raw Data 2019\\Sales_December_2019.csv",
 "C:/Users/renal/Documents/Renaldo's File/Data Analyst Portofolio -Renaldo
Livando/Project1 Data Wrangling using Python/dataSource/Electronic Stores Sales
Raw Data 2019\\Sales_February_2019.csv",
 "C:/Users/renal/Documents/Renaldo's File/Data Analyst Portofolio -Renaldo
Livando/Project1 Data Wrangling using Python/dataSource/Electronic Stores Sales
Raw Data 2019\\Sales_January_2019.csv",
 "C:/Users/renal/Documents/Renaldo's File/Data Analyst Portofolio -Renaldo
Livando/Project1 Data Wrangling using Python/dataSource/Electronic Stores Sales
Raw Data 2019\\Sales_July_2019.csv",
 "C:/Users/renal/Documents/Renaldo's File/Data Analyst Portofolio -Renaldo
Livando/Project1 Data Wrangling using Python/dataSource/Electronic Stores Sales
Raw Data 2019\\Sales_June_2019.csv",
 "C:/Users/renal/Documents/Renaldo's File/Data Analyst Portofolio -Renaldo
Livando/Project1 Data Wrangling using Python/dataSource/Electronic Stores Sales
Raw Data 2019\\Sales_March_2019.csv",
 "C:/Users/renal/Documents/Renaldo's File/Data Analyst Portofolio -Renaldo
Livando/Project1 Data Wrangling using Python/dataSource/Electronic Stores Sales
Raw Data 2019\\Sales_May_2019.csv",
 "C:/Users/renal/Documents/Renaldo's File/Data Analyst Portofolio -Renaldo
Livando/Project1 Data Wrangling using Python/dataSource/Electronic Stores Sales
Raw Data 2019\\Sales_November_2019.csv",
 "C:/Users/renal/Documents/Renaldo's File/Data Analyst Portofolio -Renaldo
Livando/Project1 Data Wrangling using Python/dataSource/Electronic Stores Sales
Raw Data 2019\\Sales_October_2019.csv",
```

```
 "C:/Users/renal/Documents/Renaldo's File/Data Analyst Portofolio -Renaldo
Livando/Project1 Data Wrangling using Python/dataSource/Electronic Stores Sales
Raw Data 2019\\Sales_September_2019.csv"]
```

[4]:
```
# Creating an empty dataframe to store the new concatenated dataframe
empty_df = []

# Concatenate while reading through files in the directory
for filename in all_files:
    df = pd.read_csv(filename, index_col=None, header=0) #header=0, because the␣
 ↪csv file has header
    empty_df.append(df)

merged_frame = pd.concat(empty_df, axis=0, ignore_index=True)

# Store our merged_frame in a new csv file
merged_frame.to_csv(r"C:/Users/renal/Documents/Renaldo's File/Data Analyst␣
 ↪Portofolio -Renaldo Livando/Project1 Data Wrangling using Python/out/
 ↪merged_electronic_sales_data.csv", index = False)
```

[5]:
```
# Read the new csv file in a new DataFrame
sales = pd.read_csv("C:/Users/renal/Documents/Renaldo's File/Data Analyst␣
 ↪Portofolio -Renaldo Livando/Project1 Data Wrangling using Python/out/
 ↪merged_electronic_sales_data.csv")

# Check and review our dataframe
sales.head()
```

[5]:

	Order ID	Product	Quantity Ordered	Price Each	\
0	176558	USB-C Charging Cable	2	11.95	
1	NaN	NaN	NaN	NaN	
2	176559	Bose SoundSport Headphones	1	99.99	
3	176560	Google Phone	1	600	
4	176560	Wired Headphones	1	11.99	

	Order Date	Purchase Address
0	04/19/19 08:46	917 1st St, Dallas, TX 75001
1	NaN	NaN
2	04/07/19 22:30	682 Chestnut St, Boston, MA 02215
3	04/12/19 14:38	669 Spruce St, Los Angeles, CA 90001
4	04/12/19 14:38	669 Spruce St, Los Angeles, CA 90001

Insight from dataframe review: 1. All the column in the dataset is necessary for our later analysis, theres no need to commit column filtering 2. That seems we need to extract the data from column “Order Date” and “Purchase Address” to make new column like “City”,“Postal Code”, “Month”, etc 3. We can add new calculated column from column “Quantity Ordered” and “Price Each”

[6]:
```
# Check and Review Information from our data
sales.info()
sales.describe()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 186850 entries, 0 to 186849
Data columns (total 6 columns):
 #   Column            Non-Null Count   Dtype
---  ------            --------------   -----
 0   Order ID          186305 non-null  object
 1   Product           186305 non-null  object
 2   Quantity Ordered  186305 non-null  object
 3   Price Each        186305 non-null  object
 4   Order Date        186305 non-null  object
 5   Purchase Address  186305 non-null  object
dtypes: object(6)
memory usage: 8.6+ MB
```

[6]:

	Order ID	Product	Quantity Ordered	Price Each	\
count	186305	186305	186305	186305	
unique	178438	20	10	24	
top	Order ID	USB-C Charging Cable	1	11.95	
freq	355	21903	168552	21903	

	Order Date	Purchase Address
count	186305	186305
unique	142396	140788
top	Order Date	Purchase Address
freq	355	355

Insight from data information review: 1. Theres about hundred difference between entries and Non-Null Count. Means we need to handle this missing values 2. The Non-Null Count for each column have the same number, we can conclude all NULL/NaN values is occur in a same row, so the best decision to handle this is by remove the rows instead replacing it 3. All the data type was object, we need to change some column data type to do further data transformation (perform mathematical operation, etc.) 4. count of Order ID and unique of Order ID is not match, means this [id]column is not primary key with unique value

1.2 Data Cleaning

The main purposes in this step will be remove errors that might distort or damage the accuracy of your analysis. This includes tasks like standardizing inputs, handling empty cells or missing values, handling duplicate values and fixing data inaccuracies. Because we use pandas dataframe, it already automatically trim double space and space at the start and the end of our string values. so we dont need trim step.

With insight from our Data Discovery, we already decide way to handle our data missing values is by removing it.

[7]:
```
#Remove Missing Values
sales=sales.dropna()
sales
```

[7]:

	Order ID	Product	Quantity Ordered	Price Each	Order Date	Purchase Address
0	176558	USB-C Charging Cable	2	11.95	04/19/19 08:46	917 1st St, Dallas, TX 75001
2	176559	Bose SoundSport Headphones	1	99.99	04/07/19 22:30	682 Chestnut St, Boston, MA 02215
3	176560	Google Phone	1	600	04/12/19 14:38	669 Spruce St, Los Angeles, CA 90001
4	176560	Wired Headphones	1	11.99	04/12/19 14:38	669 Spruce St, Los Angeles, CA 90001
5	176561	Wired Headphones	1	11.99	04/30/19 09:27	333 8th St, Los Angeles, CA 90001
…	…	…	…	…	…	…
186845	259353	AAA Batteries (4-pack)	3	2.99	09/17/19 20:56	840 Highland St, Los Angeles, CA 90001
186846	259354	iPhone	1	700	09/01/19 16:00	216 Dogwood St, San Francisco, CA 94016
186847	259355	iPhone	1	700	09/23/19 07:39	220 12th St, San Francisco, CA 94016
186848	259356	34in Ultrawide Monitor	1	379.99	09/19/19 17:30	511 Forest St, San Francisco, CA 94016
186849	259357	USB-C Charging Cable	1	11.95	09/30/19 00:18	250 Meadow St, San Francisco, CA 94016

[186305 rows x 6 columns]

If we take a look in the raw file, we can found that heading (string data type) of merged files is randomly repeated in data rows. So we can fix data inaccuracies by make sure all the data contained in Order ID is number not a string.

[8]:
```
# Change string value in column Order ID to NaN
sales[['Order ID']] = sales[['Order ID']].apply(pd.to_numeric, errors='coerce')

# Remove the NaN values
sales=sales.dropna()

# Change data type to int so the decimal will removed
sales['Order ID']=sales['Order ID'].astype(int)
```

```
C:\Users\renal\AppData\Local\Temp\ipykernel_11692\2231170803.py:8:
SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
```

```
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  sales['Order ID']=sales['Order ID'].astype(int)
```

There seems the code get the Warning: SettingWithCopyWarning, so we will get rid this by warning by creating a copy

[9]:
```
# Create copy to fix SettingWithCopyWarning
sales_cleaned=sales.copy()
sales_cleaned['Order ID']=sales_cleaned['Order ID'].astype(int)
sales_cleaned.shape[0]
```

[9]:
```
185950
```

We could see the rows is decreasing from 186305 to 185950, means we already clean that unwanted value

Check count of duplicate values in the dataframe

[10]:
```
duplicateRows = sales_cleaned[sales_cleaned.duplicated()]
duplicateRows.shape[0]
```

[10]:
```
264
```

[11]:
```
ids2= duplicateRows['Order ID']
sales_cleaned[sales_cleaned['Order ID'].isin(ids2)].sort_values('Order ID').
 ↪head(6)
```

[11]:

	Order ID	Product	Quantity Ordered	Price Each	Order Date	Purchase Address
68422	142071	AA Batteries (4-pack)	1	3.84	01/17/19 23:02	131 2nd St, Boston, MA 02215
68421	142071	AA Batteries (4-pack)	1	3.84	01/17/19 23:02	131 2nd St, Boston, MA 02215
71673	145143	Lightning Charging Cable	1	14.95	01/06/19 03:01	182 Jefferson St, San Francisco, CA 94016
71672	145143	Lightning Charging Cable	1	14.95	01/06/19 03:01	182 Jefferson St, San Francisco, CA 94016
73358	146765	Google Phone	1	600	01/21/19 11:23	918 Highland St, New York City, NY 10001
73357	146765	Google Phone	1	600	01/21/19 11:23	918 Highland St, New York City, NY 10001

With this check, we found there's occur several duplicate values. Before we decide a way to handle these duplicate values, we need to understand and interpet the dataset. 1. The data is record any purchasement into a row. 2. One Order ID should have One Order Date. 3. One Order ID should have One Purchase Address. 4. One Order ID could have several different products.

With these 4 points, we can conclude that if Order ID have same Product recorded different rows doesnt make sense. So we decide to remove duplicate rows that duplicated accross all column.

[12]:
```
sales_cleaned.shape[0]
```

[12]:
```
185950
```

[13]:
```
# Remove duplicate values
sales_cleaned=sales_cleaned.drop_duplicates()
sales_cleaned.shape[0]
```

[13]:
```
185686
```

Our dataset rows decreased by 264. and now has 185686 rows after deletion.

1.3 Data Transformation

The main purposes in this step will be: - Data Structuring: make sure various datasets are in compatible formats. - Attribute Construction: in which new attributes are added or created from existing attributes - Generalization: where low-level data attributes are converted into high-level data attributes (in this project is "Purchase Address" column).

[14]:
```
# Change data type for some columns
# Change 'Order Date' to datetime
sales_cleaned['Order Date'] = pd.to_datetime(sales_cleaned['Order Date'])

# Change 'Price Each' &  'Quantity Ordered' to float
sales_cleaned['Price Each'] = sales_cleaned['Price Each'].astype(float)
sales_cleaned['Quantity Ordered'] = sales_cleaned['Quantity Ordered'].
 ↪astype(float)

# Check new data type information
sales_cleaned.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 185686 entries, 0 to 186849
Data columns (total 6 columns):
 #   Column            Non-Null Count   Dtype
---  ------            --------------   -----
 0   Order ID          185686 non-null  int32
 1   Product           185686 non-null  object
 2   Quantity Ordered  185686 non-null  float64
 3   Price Each        185686 non-null  float64
 4   Order Date        185686 non-null  datetime64[ns]
 5   Purchase Address  185686 non-null  object
dtypes: datetime64[ns](1), float64(2), int32(1), object(2)
memory usage: 9.2+ MB
```

Creating new column "Month" by extracting from existing column "Order Date":

[15]:
```
# Create a dictionary for month mapping
month_map = {
    1: 'January',
    2: 'February',
    3: 'March',
    4: 'April',
    5: 'May',
    6: 'June',
    7: 'July',
    8: 'August',
    9: 'September',
    10: 'October',
    11: 'November',
    12: 'December'
}

# Create Month column
sales_cleaned['Month'] = sales_cleaned['Order Date'].dt.month.map(month_map)
sales_cleaned.head()
```

[15]:

	Order ID	Product	Quantity Ordered	Price Each	\
0	176558	USB-C Charging Cable	2.0	11.95	
2	176559	Bose SoundSport Headphones	1.0	99.99	
3	176560	Google Phone	1.0	600.00	
4	176560	Wired Headphones	1.0	11.99	
5	176561	Wired Headphones	1.0	11.99	

	Order Date	Purchase Address	Month
0	2019-04-19 08:46:00	917 1st St, Dallas, TX 75001	April
2	2019-04-07 22:30:00	682 Chestnut St, Boston, MA 02215	April
3	2019-04-12 14:38:00	669 Spruce St, Los Angeles, CA 90001	April
4	2019-04-12 14:38:00	669 Spruce St, Los Angeles, CA 90001	April
5	2019-04-30 09:27:00	333 8th St, Los Angeles, CA 90001	April

Creating new column “Total Cost” by multiplying from existing column “Quantity Ordered” & “Price Each”:

[16]:
```
sales_cleaned["Total Cost"]=sales_cleaned["Quantity␣
 ↪Ordered"]*sales_cleaned["Price Each"]
sales_cleaned.head()
```

[16]:

	Order ID	Product	Quantity Ordered	Price Each	\
0	176558	USB-C Charging Cable	2.0	11.95	
2	176559	Bose SoundSport Headphones	1.0	99.99	
3	176560	Google Phone	1.0	600.00	
4	176560	Wired Headphones	1.0	11.99	
5	176561	Wired Headphones	1.0	11.99	

```
           Order Date                      Purchase Address  Month  Total Cost
0 2019-04-19 08:46:00          917 1st St, Dallas, TX 75001  April       23.90
2 2019-04-07 22:30:00     682 Chestnut St, Boston, MA 02215  April       99.99
3 2019-04-12 14:38:00  669 Spruce St, Los Angeles, CA 90001  April      600.00
4 2019-04-12 14:38:00  669 Spruce St, Los Angeles, CA 90001  April       11.99
5 2019-04-30 09:27:00     333 8th St, Los Angeles, CA 90001  April       11.99
```

[17]:
```
#Re-Arrange our dataframe
sales_cleaned = sales_cleaned[['Order ID', 'Order␣
 ↪Date','Month','Product','Quantity Ordered','Price Each','Total Cost',
                               'Purchase Address']]
sales_cleaned.head()
```

[17]:
```
   Order ID          Order Date  Month                    Product  \
0    176558 2019-04-19 08:46:00  April        USB-C Charging Cable
2    176559 2019-04-07 22:30:00  April  Bose SoundSport Headphones
3    176560 2019-04-12 14:38:00  April                Google Phone
4    176560 2019-04-12 14:38:00  April            Wired Headphones
5    176561 2019-04-30 09:27:00  April            Wired Headphones

   Quantity Ordered  Price Each  Total Cost  \
0               2.0       11.95       23.90
2               1.0       99.99       99.99
3               1.0      600.00      600.00
4               1.0       11.99       11.99
5               1.0       11.99       11.99

                     Purchase Address
0          917 1st St, Dallas, TX 75001
2     682 Chestnut St, Boston, MA 02215
3  669 Spruce St, Los Angeles, CA 90001
4  669 Spruce St, Los Angeles, CA 90001
5     333 8th St, Los Angeles, CA 90001
```

Do generalization for column “Purchase Address” into column “Street”, “City”, “State”, “Postal Code”:

[18]:
```
# Extract Purchase Address by comma (,) delimited
sales_cleaned[['Street','City','State_X']]=sales_cleaned['Purchase Address'].
 ↪str.split(",",expand=True)
# Split the State and Postal Code
sales_cleaned[['EMPTY','State','Postal Code']]=sales_cleaned['State_X'].str.
 ↪split(" ",expand=True)
# Drop unnecessary column generated
sales_cleaned = sales_cleaned.drop(columns=['EMPTY','State_X'])
sales_cleaned.head()
```

[18]:
```
   Order ID          Order Date  Month                    Product  \
0    176558 2019-04-19 08:46:00  April        USB-C Charging Cable
2    176559 2019-04-07 22:30:00  April  Bose SoundSport Headphones
3    176560 2019-04-12 14:38:00  April                Google Phone
4    176560 2019-04-12 14:38:00  April            Wired Headphones
5    176561 2019-04-30 09:27:00  April            Wired Headphones

   Quantity Ordered  Price Each  Total Cost  \
0               2.0       11.95       23.90
2               1.0       99.99       99.99
3               1.0      600.00      600.00
4               1.0       11.99       11.99
5               1.0       11.99       11.99

                     Purchase Address           Street          City State  \
0         917 1st St, Dallas, TX 75001       917 1st St        Dallas    TX
2    682 Chestnut St, Boston, MA 02215  682 Chestnut St        Boston    MA
3  669 Spruce St, Los Angeles, CA 90001    669 Spruce St   Los Angeles    CA
4  669 Spruce St, Los Angeles, CA 90001    669 Spruce St   Los Angeles    CA
5    333 8th St, Los Angeles, CA 90001       333 8th St   Los Angeles    CA

  Postal Code
0       75001
2       02215
3       90001
4       90001
5       90001
```

1.4 Data Enriching

The main purpose in this step will be: enrich the dataset by adding values from other datasets.

[19]:
```python
# Create a dictionary for season mapping
season_map = {
    1: 'Winter',
    2: 'Winter',
    3: 'Spring',
    4: 'Spring',
    5: 'Spring',
    6: 'Summer',
    7: 'Summer',
    8: 'Summer',
    9: 'Fall',
    10: 'Fall',
    11: 'Fall',
    12: 'Winter'
}
```

```
# Create Season colum
sales_cleaned['Season'] = sales_cleaned['Order Date'].dt.month.map(season_map)
sales_cleaned.head()
```

[19]:
```
   Order ID           Order Date  Month                       Product  \
0    176558 2019-04-19 08:46:00  April          USB-C Charging Cable
2    176559 2019-04-07 22:30:00  April  Bose SoundSport Headphones
3    176560 2019-04-12 14:38:00  April                  Google Phone
4    176560 2019-04-12 14:38:00  April              Wired Headphones
5    176561 2019-04-30 09:27:00  April              Wired Headphones

   Quantity Ordered  Price Each  Total Cost  \
0               2.0       11.95       23.90
2               1.0       99.99       99.99
3               1.0      600.00      600.00
4               1.0       11.99       11.99
5               1.0       11.99       11.99

                      Purchase Address           Street          City State  \
0          917 1st St, Dallas, TX 75001       917 1st St        Dallas    TX
2     682 Chestnut St, Boston, MA 02215  682 Chestnut St        Boston    MA
3  669 Spruce St, Los Angeles, CA 90001    669 Spruce St   Los Angeles    CA
4  669 Spruce St, Los Angeles, CA 90001    669 Spruce St   Los Angeles    CA
5     333 8th St, Los Angeles, CA 90001       333 8th St   Los Angeles    CA

  Postal Code  Season
0       75001  Spring
2       02215  Spring
3       90001  Spring
4       90001  Spring
5       90001  Spring
```

1.5 Data Validating

The main purpose in this step will be: essentially check the work I did during the transformation stage, verifying the data is consistent.

[20]:
```
#This will return NaN if there any Postal Code with digits other than 5 digits

sales_cleaned['Postal Code']=sales_cleaned['Postal Code'].apply(lambda x: x if␣
 ↪len(x)== 5 else np.nan)
sales_cleaned.head()
```

[20]:
```
   Order ID           Order Date  Month                       Product  \
0    176558 2019-04-19 08:46:00  April          USB-C Charging Cable
2    176559 2019-04-07 22:30:00  April  Bose SoundSport Headphones
```

```
3    176560 2019-04-12 14:38:00  April              Google Phone
4    176560 2019-04-12 14:38:00  April            Wired Headphones
5    176561 2019-04-30 09:27:00  April            Wired Headphones

   Quantity Ordered  Price Each  Total Cost  \
0               2.0       11.95       23.90
2               1.0       99.99       99.99
3               1.0      600.00      600.00
4               1.0       11.99       11.99
5               1.0       11.99       11.99

                     Purchase Address           Street          City State  \
0          917 1st St, Dallas, TX 75001       917 1st St        Dallas    TX
2     682 Chestnut St, Boston, MA 02215  682 Chestnut St        Boston    MA
3  669 Spruce St, Los Angeles, CA 90001    669 Spruce St   Los Angeles    CA
4  669 Spruce St, Los Angeles, CA 90001    669 Spruce St   Los Angeles    CA
5     333 8th St, Los Angeles, CA 90001       333 8th St   Los Angeles    CA

  Postal Code  Season
0       75001  Spring
2       02215  Spring
3       90001  Spring
4       90001  Spring
5       90001  Spring
```

[21]:
```
# We find the rows that Postal Code got NaN value
df1 = sales_cleaned[sales_cleaned.isna().any(axis=1)]
df1
```

[21]:
```
Empty DataFrame
Columns: [Order ID, Order Date, Month, Product, Quantity Ordered, Price Each,
Total Cost, Purchase Address, Street, City, State, Postal Code, Season]
Index: []
```

Dataframe df1 doesnt have any rows, so we can conclude that Postal Code column already have valid data.

1.6 Data Publishing

The main purpose in this step is to publish our data into file format we prefer for sharing with other team members for downstream analysis purposes.

[22]:
```
sales_cleaned.to_csv(r"C:/Users/renal/Documents/Renaldo's File/Data Analyst␣
↪Portofolio -Renaldo Livando/Project1 Data Wrangling using Python/out/
↪cleaned_electronic_sales_data.csv", index = False)
```