# Final Project - Analyzing Sales Data

**Date**: 16 July 2023

**Author**: Monsicha Tamee

**Course**: `Pandas Foundation`

```
# import data
import pandas as pd
import numpy as np
df = pd.read_csv("sample-store.csv")
```

```
# preview top 5 rows
df.head()
```

| | Row ID | Order ID | Order Date | Ship Date | Ship Mode | Customer ID | Customer Name | Segment | Country/Region | City |
|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 1 | CA-2019-152156 | 11/8/2019 | 11/11/2019 | Second Class | CG-12520 | Claire Gute | Consumer | United States | Henderson |
| 1 | 2 | CA-2019-152156 | 11/8/2019 | 11/11/2019 | Second Class | CG-12520 | Claire Gute | Consumer | United States | Henderson |
| 2 | 3 | CA-2019-138688 | 6/12/2019 | 6/16/2019 | Second Class | DV-13045 | Darrin Van Huff | Corporate | United States | Los Angeles |
| 3 | 4 | US-2018-108966 | 10/11/2018 | 10/18/2018 | Standard Class | SO-20335 | Sean O'Donnell | Consumer | United States | Fort Lauderdale |
| 4 | 5 | US-2018-108966 | 10/11/2018 | 10/18/2018 | Standard Class | SO-20335 | Sean O'Donnell | Consumer | United States | Fort Lauderdale |

5 rows × 21 columns

```
# shape of dataframe
df.shape
```

```
(9994, 21)
```

```
# see data frame information using .info()
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 9994 entries, 0 to 9993
Data columns (total 21 columns):
 #   Column          Non-Null Count  Dtype  
---  ------          --------------  -----  
 0   Row ID          9994 non-null   int64  
 1   Order ID        9994 non-null   object 
 2   Order Date      9994 non-null   object 
 3   Ship Date       9994 non-null   object 
 4   Ship Mode       9994 non-null   object 
 5   Customer ID     9994 non-null   object 
 6   Customer Name   9994 non-null   object 
 7   Segment         9994 non-null   object 
 8   Country/Region  9994 non-null   object 
 9   City            9994 non-null   object 
 10  State           9994 non-null   object 
 11  Postal Code     9983 non-null   float64
 12  Region          9994 non-null   object 
 13  Product ID      9994 non-null   object 
 14  Category        9994 non-null   object 
```

We can use `pd.to_datetime()` function to convert columns 'Order Date' and 'Ship Date' to datetime.

```
# example of pd.to_datetime() function
pd.to_datetime(df['Order Date'].head(), format='%m/%d/%Y')
```

```
0   2019-11-08
1   2019-11-08
2   2019-06-12
3   2018-10-11
4   2018-10-11
Name: Order Date, dtype: datetime64[ns]
```

```
# TODO - convert order date and ship date to datetime in the original dataframe
df['Order Date'] = pd.to_datetime(df['Order Date'], format= '%m/%d/%Y')
df['Ship Date'] = pd.to_datetime(df['Ship Date'], format= '%m/%d/%Y')
df
```

| | Row ID | Order ID | Order Date | Ship Date | Ship Mode | Customer ID | Customer Name | Segment | Country/Region | City | ... |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 1 | CA-2019-152156 | 2019-11-08 | 2019-11-11 | Second Class | CG-12520 | Claire Gute | Consumer | United States | Henderson | ... |
| 1 | 2 | CA-2019-152156 | 2019-11-08 | 2019-11-11 | Second Class | CG-12520 | Claire Gute | Consumer | United States | Henderson | ... |
| 2 | 3 | CA-2019-138688 | 2019-06-12 | 2019-06-16 | Second Class | DV-13045 | Darrin Van Huff | Corporate | United States | Los Angeles | ... |
| 3 | 4 | US-2018-108966 | 2018-10-11 | 2018-10-18 | Standard Class | SO-20335 | Sean O'Donnell | Consumer | United States | Fort Lauderdale | ... |
| 4 | 5 | US-2018-108966 | 2018-10-11 | 2018-10-18 | Standard Class | SO-20335 | Sean O'Donnell | Consumer | United States | Fort Lauderdale | ... |
| ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... |
| 9989 | 9990 | CA-2017-110422 | 2017-01-21 | 2017-01-23 | Second Class | TB-21400 | Tom Boeckenhauer | Consumer | United States | Miami | ... |
| 9990 | 9991 | CA-2020-121258 | 2020-02-26 | 2020-03-03 | Standard Class | DB-13060 | Dave Brooks | Consumer | United States | Costa Mesa | ... |
| 9991 | 9992 | CA-2020-121258 | 2020-02-26 | 2020-03-03 | Standard Class | DB-13060 | Dave Brooks | Consumer | United States | Costa Mesa | ... |
| 9992 | 9993 | CA-2020-121258 | 2020-02-26 | 2020-03-03 | Standard Class | DB-13060 | Dave Brooks | Consumer | United States | Costa Mesa | ... |
| 9993 | 9994 | CA-2020-119914 | 2020-05-04 | 2020-05-09 | Second Class | CC-12220 | Chris Cortes | Consumer | United States | Westminster | ... |

9994 rows × 21 columns

```
# TODO - count nan in postal code column
df['Postal Code'].isna().sum()
```

11

```
# TODO - filter rows with missing values
df.isna().sum()
dfPostal_na = df[df['Postal Code'].isna()]
```

```
# TODO - Explore this dataset on your owns, ask your own questions
df.tail()
```

| | Row ID | Order ID | Order Date | Ship Date | Ship Mode | Customer ID | Customer Name | Segment | Country/Region | City | ... |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 9989 | 9990 | CA-2017-110422 | 2017-01-21 | 2017-01-23 | Second Class | TB-21400 | Tom Boeckenhauer | Consumer | United States | Miami | ... |
| 9990 | 9991 | CA-2020-121258 | 2020-02-26 | 2020-03-03 | Standard Class | DB-13060 | Dave Brooks | Consumer | United States | Costa Mesa | ... |
| 9991 | 9992 | CA-2020-121258 | 2020-02-26 | 2020-03-03 | Standard Class | DB-13060 | Dave Brooks | Consumer | United States | Costa Mesa | ... |
| 9992 | 9993 | CA-2020-121258 | 2020-02-26 | 2020-03-03 | Standard Class | DB-13060 | Dave Brooks | Consumer | United States | Costa Mesa | ... |
| 9993 | 9994 | CA-2020-119914 | 2020-05-04 | 2020-05-09 | Second Class | CC-12220 | Chris Cortes | Consumer | United States | Westminster | ... |

5 rows × 21 columns

```
# Summarises numberic columns
df.describe()
```

| | Row ID | Postal Code | Sales | Quantity | Discount | Profit |
|---|---|---|---|---|---|---|
| count | 9994.000000 | 9983.000000 | 9994.000000 | 9994.000000 | 9994.000000 | 9994.000000 |
| mean | 4997.500000 | 55245.233297 | 229.858001 | 3.789574 | 0.156203 | 28.656896 |
| std | 2885.163629 | 32038.715955 | 623.245101 | 2.225110 | 0.206452 | 234.260108 |
| min | 1.000000 | 1040.000000 | 0.444000 | 1.000000 | 0.000000 | -6599.978000 |
| 25% | 2499.250000 | 23223.000000 | 17.280000 | 2.000000 | 0.000000 | 1.728750 |
| 50% | 4997.500000 | 57103.000000 | 54.490000 | 3.000000 | 0.200000 | 8.666500 |
| 75% | 7495.750000 | 90008.000000 | 209.940000 | 5.000000 | 0.200000 | 29.364000 |
| max | 9994.000000 | 99301.000000 | 22638.480000 | 14.000000 | 0.800000 | 8399.976000 |

```
# Summarises string columns
df.describe(include = "object")
```

| | Order ID | Ship Mode | Customer ID | Customer Name | Segment | Country/Region | City | State | Region | Product ID | Ca |
|---|---|---|---|---|---|---|---|---|---|---|---|
| count | 9994 | 9994 | 9994 | 9994 | 9994 | 9994 | 9994 | 9994 | 9994 | 9994 | 99 |
| unique | 5009 | 4 | 793 | 793 | 3 | 1 | 531 | 49 | 4 | 1862 | 3 |
| top | CA-2020-100111 | Standard Class | WB-21850 | William Brown | Consumer | United States | New York City | California | West | OFF-PA-10001970 | Of Su |
| freq | 14 | 5968 | 37 | 37 | 5191 | 9994 | 915 | 2001 | 3203 | 19 | 60 |

In my opinion, separating data into two parts is easier to explore.

```
# Summarise all column including nan
df.describe(include = "all")
```

| | Row ID | Order ID | Order Date | Ship Date | Ship Mode | Customer ID | Customer Name | Segment | Country/Region | City |
|---|---|---|---|---|---|---|---|---|---|---|
| count | 9994.000000 | 9994 | 9994 | 9994 | 9994 | 9994 | 9994 | 9994 | 9994 | 9994 |
| unique | NaN | 5009 | 1236 | 1334 | 4 | 793 | 793 | 3 | 1 | 531 |
| top | NaN | CA-2020-100111 | 2019-09-05 00:00:00 | 2018-12-16 00:00:00 | Standard Class | WB-21850 | William Brown | Consumer | United States | New York City |
| freq | NaN | 14 | 38 | 35 | 5968 | 37 | 37 | 5191 | 9994 | 915 |
| first | NaN | NaN | 2017-01-03 00:00:00 | 2017-01-07 00:00:00 | NaN | NaN | NaN | NaN | NaN | NaN |
| last | NaN | NaN | 2020-12-30 00:00:00 | 2021-01-05 00:00:00 | NaN | NaN | NaN | NaN | NaN | NaN |
| mean | 4997.500000 | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN |
| std | 2885.163629 | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN |
| min | 1.000000 | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN |
| 25% | 2499.250000 | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN |
| 50% | 4997.500000 | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN |
| 75% | 7495.750000 | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN |
| max | 9994.000000 | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN |

13 rows × 21 columns

```
<ipython-input-46-219af8f44548>:2: FutureWarning: Treating datetime data as categ
  df.describe(include = "all")
<ipython-input-46-219af8f44548>:2: FutureWarning: Treating datetime data as categ
  df.describe(include = "all")
```

```
df["Ship Mode"].value_counts()
df["Segment"].value_counts()
```

```
Consumer       5191
Corporate      3020
Home Office    1783
Name: Segment, dtype: int64
```

```
# Explore category and sub-category
dfCate = df[["Category","Sub-Category"]].value_counts().reset_index()
dfCate.columns = ["Category", "Sub-Category","count"]
dfCate.sort_values("Category").reset_index()
```

| | index | Category | Sub-Category | count |
|---|---|---|---|---|
| 0 | 2 | Furniture | Furnishings | 957 |
| 1 | 12 | Furniture | Bookcases | 228 |
| 2 | 10 | Furniture | Tables | 319 |
| 3 | 7 | Furniture | Chairs | 617 |
| 4 | 0 | Office Supplies | Binders | 1523 |
| 5 | 14 | Office Supplies | Supplies | 190 |
| 6 | 13 | Office Supplies | Fasteners | 217 |
| 7 | 11 | Office Supplies | Envelopes | 254 |
| 8 | 9 | Office Supplies | Labels | 364 |
| 9 | 8 | Office Supplies | Appliances | 466 |
| 10 | 5 | Office Supplies | Art | 796 |
| 11 | 4 | Office Supplies | Storage | 846 |
| 12 | 1 | Office Supplies | Paper | 1370 |
| 13 | 6 | Technology | Accessories | 775 |
| 14 | 3 | Technology | Phones | 889 |
| 15 | 15 | Technology | Machines | 115 |
| 16 | 16 | Technology | Copiers | 68 |

```
# Explore state and city
state_city = df[["State","City"]].value_counts().reset_index()
state_city.columns =["state","city","count"]
state_city.sort_values(["count","state"], ascending = False)
```

| | state | city | count |
|---|---|---|---|
| 0 | New York | New York City | 915 |
| 1 | California | Los Angeles | 747 |
| 2 | Pennsylvania | Philadelphia | 537 |
| 3 | California | San Francisco | 510 |
| 4 | Washington | Seattle | 428 |
| ... | ... | ... | ... |
| 565 | California | Pico Rivera | 1 |
| 566 | California | Redding | 1 |
| 553 | Arkansas | Conway | 1 |
| 554 | Arkansas | Rogers | 1 |
| 555 | Arkansas | Springdale | 1 |

604 rows × 3 columns

```
# Check duplicated data
df.duplicated().sum()
```

```
0
```

## Data Analysis Part

Answer 10 below questions to get credit from this course. Write `pandas` code to find answers.

### TODO 01 - how many columns, rows in this dataset

```
df.shape

# Answer: There are 9994 rows and 21 columns
```

```
(9994, 21)
```

### TODO 02 - is there any missing values?, if there is, which colunm? how many nan values?

```
df.isna().sum()

# Answer: There are 11 rows of missing values in the 'Postal Code' column.
```

```
Row ID           0
Order ID         0
Order Date       0
Ship Date        0
Ship Mode        0
Customer ID      0
Customer Name    0
Segment          0
Country/Region   0
City             0
State            0
Postal Code     11
Region           0
Product ID       0
Category         0
Sub-Category     0
Product Name     0
Sales            0
Quantity         0
Discount         0
Profit           0
dtype: int64
```

```
# Remove nan
df = df.dropna()
df.isna().sum()
```

```
Row ID           0
Order ID         0
Order Date       0
Ship Date        0
Ship Mode        0
Customer ID      0
Customer Name    0
Segment          0
Country/Region   0
City             0
State            0
Postal Code      0
Region           0
Product ID       0
Category         0
Sub-Category     0
Product Name     0
Sales            0
Quantity         0
Discount         0
Profit           0
dtype: int64
```

### TODO 03 - your friend ask for `California` data, filter it and export csv for him

```
# Answer:
dfCalifornia = df[df['State'] == 'California'].dropna()
dfCalifornia.to_csv("California.csv")
```

### TODO 04 - your friend ask for all order data in `California` and `Texas` in 2017 (look at Order Date), send him csv file

```
# Filter 'California' and 'Texas': df[(df['State'] == 'California') | (df['State']
# Filter 'Order date 2017' use dt.strftime() to filter date: df[(df['Order Date'].d
# Answer:
CaliforniaTexas = df[(df['State'] == 'California') | (df['State'] == 'Texas')]
CaliforniaTexas_2017 = CaliforniaTexas[CaliforniaTexas['Order Date'].dt.strftime('%
CaliforniaTexas_2017.to_csv("CaliforniaTexas2017.csv")
```

### TODO 05 - how much total sales, average sales, and standard deviation of sales

```
# df['Sales'] .agg(['sum','mean','std'] )
Orders_2017 = df[df['Order Date'].dt.strftime('%Y') == '2017']
Orders_2017['Sales'].agg(['sum','mean','std']).round(2)

# Answer: Total sales is 484247.5, average sales is 242.97 and standard deviation i
```

```
sum     484247.50
mean       242.97
std        754.05
Name: Sales, dtype: float64
```

### TODO 06 - which Segment has the highest profit in 2018

```
df[df['Order Date'].dt.strftime('%Y') == '2018'].dropna()\
    .groupby('Segment')['Profit'].sum().sort_values(ascending= False).round(2)

# Answer: Consumer has the hihest profit at 28281.37 in 2018.
```

```
Segment
Consumer       28281.37
Corporate      19675.20
Home Office    12470.11
Name: Profit, dtype: float64
```

### TODO 07 - which top 5 States have the least total sales between 15 April 2019 - 31 December 2019

```
df[(df['Order Date'] >= '2019-04-15') & (df['Order Date'] <= '2019-12-31')]\
    .groupby('State')['Sales'].sum().sort_values(ascending= False).round(2).head(5)

# Answer: California, New York, Texas, Pennsylvania and Michigan
```

```
State
California      105632.96
New York         56873.93
Texas            31114.34
Pennsylvania     28207.29
Michigan         26675.81
Name: Sales, dtype: float64
```

### TODO 08 - what is the proportion of total sales (%) in West + Central in 2019 e.g. 25%

```
# Create order year column
df["order_year"] = df["Order Date"].dt.strftime('%Y')

# Total sales in 2019
Orders_2019 = df[df["order_year"] == '2019']
Sales_2019 = Orders_2019["Sales"].sum()

# Total sales of West + Central in 2019
OrdersWC_2019 = df[df["order_year"] == '2019'].query("Region == 'West' | Region ==
SalesWC_2019 = OrdersWC_2019["Sales"].sum()

# Calculte proportion of total sales (%) in West + Central in 2019
((SalesWC_2019/Sales_2019)*100).round(2)

# Answer: 55.24%
```

```
55.24

<ipython-input-59-87c5ea4a1f24>:2: 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/stabl
  df["order_year"] = df["Order Date"].dt.strftime('%Y')
```

### TODO 09 - find top 10 popular products in terms of number of orders vs. total sales during 2019-2020

```
# Filter orders during 2019-2020
Orders_2019_2020 = df[(df['order_year'] >= '2019') & (df['order_year'] <= '2020')]

# Find top 10 products
topSales_2019_2020 = Orders_2019_2020.groupby("Product Name")[["Quantity","Sales"]]
    .sum().sort_values("Quantity",ascending= False).round(2).head(10)

# Answer:
topSales_2019_2020
```

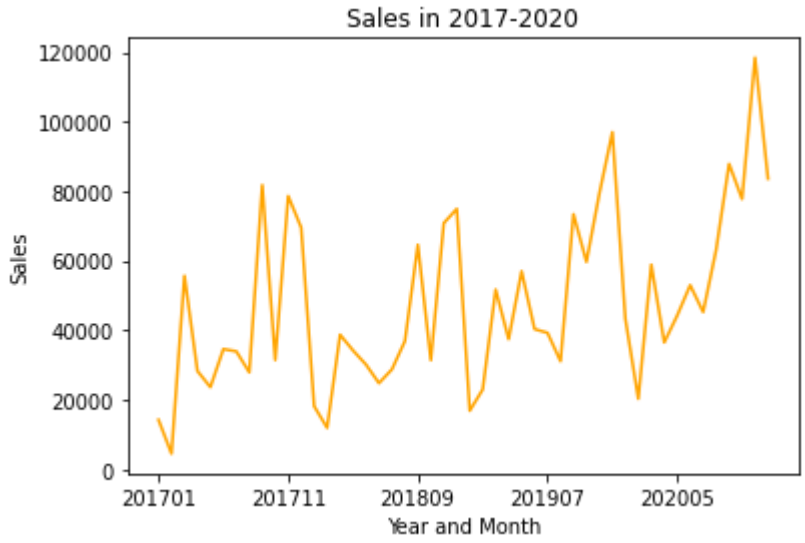
| | Quantity | Sales |
|---|---|---|
| Product Name | | |
| Staples | 124 | 462.07 |
| Easy-staple paper | 89 | 1481.73 |
| Staple envelope | 73 | 644.94 |
| Staples in misc. colors | 60 | 357.16 |
| Chromcraft Round Conference Tables | 59 | 7965.05 |
| Storex Dura Pro Binders | 49 | 176.42 |
| Situations Contoured Folding Chairs, 4/Set | 47 | 2612.06 |
| Wilson Jones Clip & Carry Folder Binder Tool for Ring Binders, Clear | 44 | 178.06 |
| Avery Non-Stick Binders | 43 | 122.13 |
| Wilson Jones Turn Tabs Binder Tool for Ring Binders | 42 | 182.20 |

### TODO 10 - plot at least 2 plots, any plot you think interesting :)

```
# Answer:
import matplotlib.pyplot as plt
df["monthid"] = df["Order Date"].dt.strftime('%Y%m')
df.groupby('monthid')['Sales'].sum().plot(kind="line",color = "orange")

plt.title('Sales in 2017-2020')
plt.xlabel('Year and Month')
plt.ylabel('Sales')
```

```
Text(0, 0.5, 'Sales')
```

Download

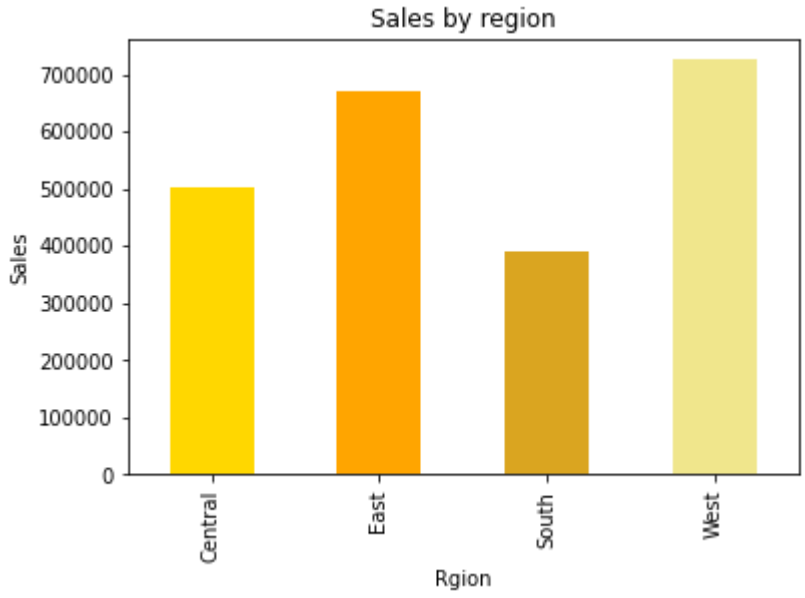



```
<ipython-input-28-7e49d9477e6a>:3: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stabl
  df["monthid"] = df["Order Date"].dt.strftime('%Y%m')
```

```
#df[["monthid","State"]].value_counts()
df.groupby("Region")["Sales"].sum().plot(kind= "bar",color = [ "gold","orange", "go

plt.title('Sales by region')
plt.xlabel('Rgion')
plt.ylabel('Sales')
```

```
Text(0, 0.5, 'Sales')
```

Download



**TODO Bonus - use np.where() to create new column in dataframe to help you answer your own questions**

```
# Which order is profitable?
df["gain profit"] =np.where(df["Profit"]>0,True,False)
df
```

| | Row ID | Order ID | Order Date | Ship Date | Ship Mode | Customer ID | Customer Name | Segment | Country/Region | City | ... |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 1 | CA-2019-152156 | 2019-11-08 | 2019-11-11 | Second Class | CG-12520 | Claire Gute | Consumer | United States | Henderson | ... |
| 1 | 2 | CA-2019-152156 | 2019-11-08 | 2019-11-11 | Second Class | CG-12520 | Claire Gute | Consumer | United States | Henderson | ... |
| 2 | 3 | CA-2019-138688 | 2019-06-12 | 2019-06-16 | Second Class | DV-13045 | Darrin Van Huff | Corporate | United States | Los Angeles | ... |
| 3 | 4 | US-2018-108966 | 2018-10-11 | 2018-10-18 | Standard Class | SO-20335 | Sean O'Donnell | Consumer | United States | Fort Lauderdale | ... |
| 4 | 5 | US-2018-108966 | 2018-10-11 | 2018-10-18 | Standard Class | SO-20335 | Sean O'Donnell | Consumer | United States | Fort Lauderdale | ... |
| ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... |
| 9989 | 9990 | CA-2017-110422 | 2017-01-21 | 2017-01-23 | Second Class | TB-21400 | Tom Boeckenhauer | Consumer | United States | Miami | ... |
| 9990 | 9991 | CA-2020-121258 | 2020-02-26 | 2020-03-03 | Standard Class | DB-13060 | Dave Brooks | Consumer | United States | Costa Mesa | ... |
| 9991 | 9992 | CA-2020-121258 | 2020-02-26 | 2020-03-03 | Standard Class | DB-13060 | Dave Brooks | Consumer | United States | Costa Mesa | ... |
| 9992 | 9993 | CA-2020-121258 | 2020-02-26 | 2020-03-03 | Standard Class | DB-13060 | Dave Brooks | Consumer | United States | Costa Mesa | ... |
| 9993 | 9994 | CA-2020-119914 | 2020-05-04 | 2020-05-09 | Second Class | CC-12220 | Chris Cortes | Consumer | United States | Westminster | ... |

9983 rows × 23 columns

```
<ipython-input-65-4801ac03c54c>:2: 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/stabl
  df["gain profit"] =np.where(df["Profit"]>0,True,False)
```

## end page

```
!git clone https://github.com/Gnampet/data-science-bootcampt7
```

```
Cloning into 'data-science-bootcampt7'...
remote: Enumerating objects: 165, done.
remote: Counting objects: 100% (87/87), done.
remote: Compressing objects: 100% (68/68), done.
remote: Total 165 (delta 50), reused 19 (delta 19), pack-reused 78
Receiving objects: 100% (165/165), 1.50 MiB | 9.23 MiB/s, done.
Resolving deltas: 100% (59/59), done.
```