Importing Libraries

```
import pandas as pd
import numpy as np
import  os
from matplotlib import pyplot as plt
import seaborn as sns
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
from scipy.stats import ttest_1samp
from sklearn.cluster import KMeans
from sklearn.preprocessing import StandardScaler , LabelEncoder
import datetime
from pmdarima import auto_arima
from statsmodels.tsa.arima_model import ARIMA
```

Loading Datasets

```
folder_path="sales_pre"
file_names=os.listdir(folder_path)
data=[]
for file_name in file_names:
    file_path = os.path.join(folder_path, file_name)
    if file_name.endswith(".csv"):
        df = pd.read_csv(file_path)
    elif file_name.endswith(".xlsx") or file_name.endswith(".xls"):
        df = pd.read_excel(file_path)
    data.append(df)
sale=pd.concat(data,ignore_index=True)
```

Examining data

```
sale.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
```

```
sale.shape
```

```
(186850, 6)
```

```
sale.info()
```

```
<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
```

Feature selection

```
sale["State"]=sale["Purchase Address"].str.split(",").str[-1].str.strip()
sale["State"]=sale["State"].str[:2]
sale["State"]
```

```
0          TX
1         NaN
2          MA
3          CA
4          CA
         ... 
186845     CA
186846     CA
186847     CA
186848     CA
186849     CA
Name: State, Length: 186850, dtype: object
```

```
sale.columns
```

```
Index(['Order ID', 'Product', 'Quantity Ordered', 'Price Each', 'Order
Date',
       'Purchase Address', 'State'],
      dtype='object')
```

Data Cleansing

```
sale.dtypes
```

```
Order ID            object
Product             object
Quantity Ordered    object
Price Each          object
Order Date          object
Purchase Address    object
State               object
dtype: object
```

```
pd.isnull(sale).sum()
```

```
Order ID            545
Product             545
Quantity Ordered    545
Price Each          545
Order Date          545
Purchase Address    545
State               545
dtype: int64
```

```
sale.dropna(inplace=True)
```

The dataset contains string values, so that we could not convert its data types

```
sale[sale["Order ID"]=="Order ID"]
```

```
        Order ID  Product  Quantity Ordered  Price Each  Order Date  \
519     Order ID  Product  Quantity Ordered  Price Each  Order Date
1149    Order ID  Product  Quantity Ordered  Price Each  Order Date
1155    Order ID  Product  Quantity Ordered  Price Each  Order Date
2878    Order ID  Product  Quantity Ordered  Price Each  Order Date
2893    Order ID  Product  Quantity Ordered  Price Each  Order Date
...          ...      ...               ...         ...         ...
185164  Order ID  Product  Quantity Ordered  Price Each  Order Date
185551  Order ID  Product  Quantity Ordered  Price Each  Order Date
186563  Order ID  Product  Quantity Ordered  Price Each  Order Date
186632  Order ID  Product  Quantity Ordered  Price Each  Order Date
186738  Order ID  Product  Quantity Ordered  Price Each  Order Date
```

```
        Purchase Address State
519     Purchase Address    Pu
1149    Purchase Address    Pu
1155    Purchase Address    Pu
2878    Purchase Address    Pu
2893    Purchase Address    Pu
...                  ...   ...
185164  Purchase Address    Pu
185551  Purchase Address    Pu
186563  Purchase Address    Pu
186632  Purchase Address    Pu
186738  Purchase Address    Pu

[355 rows x 7 columns]
```

```
def string_process(value):
    if value=="Order ID":
        return np.nan
    else:
        return value


sale["Order ID"]=sale["Order ID"].apply(string_process)

pd.isnull(sale).sum()
```

```
Order ID            355
Product               0
Quantity Ordered      0
Price Each            0
Order Date            0
Purchase Address      0
State                 0
dtype: int64
```

```
sale.dropna(inplace=True)

#Let's check
sale[sale["Quantity Ordered"]=="Quantity Ordered"]
#BINGOO
```

```
Empty DataFrame
Columns: [Order ID, Product, Quantity Ordered, Price Each, Order Date,
Purchase Address, State]
Index: []
```

Column Deletion

```
#sale.drop("Purchase Address",axis=1,inplace=True)

sale.tail(n=5)
```

```
       Order ID                 Product  Quantity Ordered  Price Each  \
186845   259353  AAA Batteries (4-pack)                 3        2.99   
186846   259354                  iPhone                 1         700   
186847   259355                  iPhone                 1         700   
186848   259356  34in Ultrawide Monitor                 1      379.99   
186849   259357    USB-C Charging Cable                 1       11.95   

            Order Date                         Purchase Address State  

186845  09/17/19 20:56   840 Highland St, Los Angeles, CA 90001    CA  

186846  09/01/19 16:00  216 Dogwood St, San Francisco, CA 94016    CA  

186847  09/23/19 07:39     220 12th St, San Francisco, CA 94016    CA  

186848  09/19/19 17:30   511 Forest St, San Francisco, CA 94016    CA  

186849  09/30/19 00:18   250 Meadow St, San Francisco, CA 94016    CA  
```

Data type casting

```
sale["Order ID"]=sale["Order ID"].astype(int)
sale["Quantity Ordered"]=sale["Quantity Ordered"].astype(int)
sale["Price Each"]=sale["Price Each"].astype(float)
sale["Total_sale"]=sale["Price Each"] * sale["Quantity Ordered"]
sale["Total_sale"]=sale["Total_sale"].astype(float)
sale["Order Date"]=pd.to_datetime(sale["Order Date"])
sale["State"]=sale["State"].astype("category")

sale.State.unique()
```

```
['TX', 'MA', 'CA', 'WA', 'GA', 'NY', 'OR', 'ME']
Categories (8, object): ['CA', 'GA', 'MA', 'ME', 'NY', 'OR', 'TX', 
'WA']
```

Again Feature Selection

In data analysis, the order of steps is flexible, allowing for freedom to perform various tasks at any point in the analysis process. The sequence of operations in data analysis is not fixed, providing the flexibility to perform tasks in any order based on specific needs and requirements.

```
sale["Month"]=sale["Order Date"].dt.strftime("%B")
sale["Month"]

0             April
2             April
3             April
4             April
5             April
            ...
186845    September
186846    September
186847    September
186848    September
186849    September
Name: Month, Length: 185950, dtype: object

sale["Year"]=sale["Order Date"].dt.year
sale["Year"].unique()

array([2019, 2020], dtype=int64)

quarter_map={1:"Q1",2:"Q2",3:"Q3",4:"Q4"}
sale["Quarter"]=sale["Order Date"].dt.quarter.map(quarter_map)

sale["Quarter"].unique()

array(['Q2', 'Q3', 'Q4', 'Q1'], dtype=object)

sale["Day"]=sale["Order Date"].dt.day_name()
sale["Day"].unique()

array(['Friday', 'Sunday', 'Tuesday', 'Monday', 'Wednesday',
'Thursday',
       'Saturday'], dtype=object)

sale.drop(["Order Date","Orde ID"],axis=1,inplace=True)

----------------------------------------------------------------------
-----
KeyError                                  Traceback (most recent call
last)
Cell In[33], line 1
----> 1 sale.drop(["Order Date","Orde ID"],axis=1,inplace=True)

File ~\anaconda3\envs\pandas_playground\Lib\site-packages\pandas\util\
_decorators.py:331, in
deprecate_nonkeyword_arguments.<locals>.decorate.<locals>.wrapper(*arg
s, **kwargs)
    325 if len(args) > num_allow_args:
    326     warnings.warn(
    327
msg.format(arguments=_format_argument_list(allow_args)),
    328         FutureWarning,
    329         stacklevel=find_stack_level(),
    330     )
--> 331 return func(*args, **kwargs)

File ~\anaconda3\envs\pandas_playground\Lib\site-packages\pandas\core\
frame.py:5399, in DataFrame.drop(self, labels, axis, index, columns,
level, inplace, errors)
   5251 @deprecate_nonkeyword_arguments(version=None,
allowed_args=["self", "labels"])
   5252 def drop(  # type: ignore[override]
   5253     self,
   (...)
   5260     errors: IgnoreRaise = "raise",
   5261 ) -> DataFrame | None:
   5262     """
   5263     Drop specified labels from rows or columns.
   5264
   (...)
   5397             weight  1.0     0.8
   5398     """
-> 5399     return super().drop(
   5400         labels=labels,
   5401         axis=axis,
   5402         index=index,
   5403         columns=columns,
   5404         level=level,
   5405         inplace=inplace,
   5406         errors=errors,
   5407     )

File ~\anaconda3\envs\pandas_playground\Lib\site-packages\pandas\util\
_decorators.py:331, in
deprecate_nonkeyword_arguments.<locals>.decorate.<locals>.wrapper(*arg
s, **kwargs)
    325 if len(args) > num_allow_args:
    326     warnings.warn(
    327
msg.format(arguments=_format_argument_list(allow_args)),
    328         FutureWarning,
    329         stacklevel=find_stack_level(),
    330     )
--> 331 return func(*args, **kwargs)

File ~\anaconda3\envs\pandas_playground\Lib\site-packages\pandas\core\
generic.py:4505, in NDFrame.drop(self, labels, axis, index, columns,
level, inplace, errors)
   4503 for axis, labels in axes.items():
   4504     if labels is not None:
-> 4505         obj = obj._drop_axis(labels, axis, level=level,
errors=errors)
   4507 if inplace:
   4508     self._update_inplace(obj)

File ~\anaconda3\envs\pandas_playground\Lib\site-packages\pandas\core\
generic.py:4546, in NDFrame._drop_axis(self, labels, axis, level,
errors, only_slice)
   4544         new_axis = axis.drop(labels, level=level,
errors=errors)
   4545     else:
-> 4546         new_axis = axis.drop(labels, errors=errors)
   4547     indexer = axis.get_indexer(new_axis)
   4549 # Case for non-unique axis
   4550 else:

File ~\anaconda3\envs\pandas_playground\Lib\site-packages\pandas\core\
indexes\base.py:6934, in Index.drop(self, labels, errors)
   6932 if mask.any():
   6933     if errors != "ignore":
-> 6934         raise KeyError(f"{list(labels[mask])} not found in
axis")
   6935     indexer = indexer[~mask]
   6936 return self.delete(indexer)

KeyError: "['Orde ID'] not found in axis"

sale
```

Data Visualization

Let's Utilize Power BI for interactive data visualization

Machine Learning

1.Predictive Modeling: Can we build a predictive model to estimate the total sales based on the quantity ordered?

Let's eliminate outliers

```
sale["z_score"]=np.abs((sale["Total_sale"]-
sale["Total_sale"].mean())/sale["Total_sale"].std())

sale.head()

z_score_limit=3
```

```
new_sales=sale[sale["z_score"]<= z_score_limit]
new_sales.shape

sale.shape

a=len(sale)
b=len(new_sales)

no_of_outliers=a-b
no_of_outliers

sample_sales=new_sales.sample(500)
sample_mean=sample_sales["Total_sale"].mean()

x=sample_sales["Price Each"].values.reshape(-1,1)
y=sample_sales["Total_sale"]

model=LinearRegression()

x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.5)

model.fit(x_train,y_train)

model.coef_

y_pred=model.predict(x_train)

plt.plot(x_train,y_train,color="blue",label="training data")
plt.plot(x_train,y_pred,color="red",label="predicted_values")
plt.xlabel("Quantity Ordered")
plt.ylabel("Total Sale")
plt.title("Linear Regression: Quantity Ordered vs. Total Sale")
plt.show()
```

let's check p value

```
pop_mean=new_sales["Quantity Ordered"].mean()
pop_mean

sample_size=200
samp1_mean=np.random.choice(new_sales["Quantity Ordered"],sample_size)

ttest,pvalue=ttest_1samp(samp1_mean,1.1275 )

pvalue < 0.05
```

Hence our linear regression analysis is statistically significnat

```
new_sales.columns
```

2.Can we segment customers based on their purchasing behavior, such as the quantity ordered, price each, and the state where the order was placed?

```
lr=LabelEncoder()

new_sales["State_labels"]=lr.fit_transform(new_sales["State"])

new_sales["State_labels"].unique()

scaler=StandardScaler()

segmented_data=new_sales[["State_labels","Price Each","Quantity
Ordered"]]

scaled_data=scaler.fit_transform(segmented_data)

kmeans=KMeans(n_clusters=3)
kmeans.fit(scaled_data)

new_sales["Cluster"]=kmeans.labels_

new_sales["Cluster"].unique()

plt.scatter(new_sales["Quantity Ordered"], new_sales["Price Each"],
c=new_sales["Cluster"], cmap="viridis")
plt.xlabel("Quantity Ordered")
plt.ylabel("Price Each")
plt.title("K-means Clustering")
plt.colorbar(label="Cluster")
plt.show()

new_sales["Cluster"]=new_sales["Cluster"].astype(int)

cluster_sizes = new_sales["Cluster"].value_counts().sort_index()
cluster_sizes

segment_0=new_sales[new_sales["Cluster"]==0]
segment_1=new_sales[new_sales["Cluster"]==1]
segment_2=new_sales[new_sales["Cluster"]==2]
```