Greetings for the day all..! here we will work on Super Market Dataset and see in detail applying EDA with Machine learning application.

```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline
import warnings
warnings.filterwarnings("ignore")
```

These are the libraries for application or use

```
df=pd.read_csv(r'../archive (3)/SampleSuperstore.csv')
df.head()
```

```
        Ship Mode    Segment        Country             City
State  \
0    Second Class   Consumer  United States        Henderson
Kentucky
1    Second Class   Consumer  United States        Henderson
Kentucky
2    Second Class  Corporate  United States      Los Angeles
California
3  Standard Class   Consumer  United States  Fort Lauderdale
Florida
4  Standard Class   Consumer  United States  Fort Lauderdale
Florida

   Postal Code Region         Category Sub-Category     Sales
Quantity  \
0        42420  South        Furniture    Bookcases  261.9600
2
1        42420  South        Furniture       Chairs  731.9400
3
2        90036   West  Office Supplies       Labels   14.6200
2
3        33311  South        Furniture       Tables  957.5775
5
4        33311  South  Office Supplies      Storage   22.3680
2

   Discount    Profit
0      0.00   41.9136
1      0.00  219.5820
2      0.00    6.8714
3      0.45 -383.0310
4      0.20    2.5164
```

here we have imported & read dataframe and assigned with variaable df we will check for the dataframe labels, column values and its datatypes to know more about DataFrame with the help of df.info()

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 9994 entries, 0 to 9993
Data columns (total 13 columns):
 #   Column         Non-Null Count  Dtype  
---  ------         --------------  -----  
 0   Ship Mode      9994 non-null   object 
 1   Segment        9994 non-null   object 
 2   Country        9994 non-null   object 
 3   City           9994 non-null   object 
 4   State          9994 non-null   object 
 5   Postal Code    9994 non-null   int64  
 6   Region         9994 non-null   object 
 7   Category       9994 non-null   object 
 8   Sub-Category   9994 non-null   object 
 9   Sales          9994 non-null   float64
 10  Quantity       9994 non-null   int64  
 11  Discount       9994 non-null   float64
 12  Profit         9994 non-null   float64
dtypes: float64(3), int64(2), object(8)
memory usage: 1015.1+ KB
```

we will check unique values in each columns for application of visualization & know more about dataset

```
df['Ship Mode'].unique()
```

```
array(['Second Class', 'Standard Class', 'First Class', 'Same Day'],
      dtype=object)
```

```
df['Segment'].unique()
```

```
array(['Consumer', 'Corporate', 'Home Office'], dtype=object)
```

```
df['Country'].unique()
```

```
array(['United States'], dtype=object)
```

```
df['State'].unique()[:10]
```

```
array(['Kentucky', 'California', 'Florida', 'North Carolina',
       'Washington', 'Texas', 'Wisconsin', 'Utah', 'Nebraska',
       'Pennsylvania'], dtype=object)
```

```
df['Postal Code'].unique()[:50]
```

```
array([42420, 90036, 33311, 90032, 28027, 98103, 76106, 53711, 84084,
       94109, 68025, 19140, 84057, 90049, 77095, 75080, 77041, 60540,
       32935, 55122, 48185, 19901, 47150, 10024, 12180, 90004, 60610,
```

```
       85234, 22153, 10009, 49201, 38109, 77070, 35601, 94122, 27707,
       60623, 29203, 55901, 55407, 97206, 55106, 80013, 28205, 60462,
       10035, 50322, 43229, 37620, 19805], dtype=int64)
```

```
df['Region'].unique()
```

```
array(['South', 'West', 'Central', 'East'], dtype=object)
```

```
df['Category'].unique()
```

```
array(['Furniture', 'Office Supplies', 'Technology'], dtype=object)
```

```
df['Sub-Category'].unique()
```

```
array(['Bookcases', 'Chairs', 'Labels', 'Tables', 'Storage',
       'Furnishings', 'Art', 'Phones', 'Binders', 'Appliances',
'Paper',
       'Accessories', 'Envelopes', 'Fasteners', 'Supplies',
'Machines',
       'Copiers'], dtype=object)
```

```
df['Sales'].unique()
```

```
array([261.96 , 731.94 ,  14.62 , ..., 437.472,  97.98 , 243.16 ])
```

```
df['Quantity'].unique()
```

```
array([ 2,  3,  5,  7,  4,  6,  9,  1,  8, 14, 11, 13, 10, 12],
      dtype=int64)
```

```
df['Discount']
```

```
0       0.00
1       0.00
2       0.00
3       0.45
4       0.20
        ...
9989    0.20
9990    0.00
9991    0.20
9992    0.00
9993    0.00
Name: Discount, Length: 9994, dtype: float64
```

```
df['Profit'].unique()
```

```
array([ 41.9136, 219.582 ,   6.8714, ...,  16.124 ,   4.1028,
72.948 ])
```

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 9994 entries, 0 to 9993
```

```
Data columns (total 13 columns):
 #   Column        Non-Null Count  Dtype  
---  ------        --------------  -----  
 0   Ship Mode     9994 non-null   object 
 1   Segment       9994 non-null   object 
 2   Country       9994 non-null   object 
 3   City          9994 non-null   object 
 4   State         9994 non-null   object 
 5   Postal Code   9994 non-null   int64  
 6   Region        9994 non-null   object 
 7   Category      9994 non-null   object 
 8   Sub-Category  9994 non-null   object 
 9   Sales         9994 non-null   float64
 10  Quantity      9994 non-null   int64  
 11  Discount      9994 non-null   float64
 12  Profit        9994 non-null   float64
dtypes: float64(3), int64(2), object(8)
memory usage: 1015.1+ KB
```

we will use groupby function to know more about relationship of columns in detaframe

```
df.groupby('Ship Mode')['Segment','City','Region','Postal
Code'].value_counts()
```

```
Ship Mode       Segment      City              Region   Postal Code
First Class     Consumer     New York City     East     10024          
30
                             Philadelphia      East     19143          
22
                             New York City     East     10035          
17
                                                        10009          
16
                Home Office  New York City     East     10035          
16
                                                                      
..
Standard Class  Consumer     Rochester Hills   Central  48307          
1
                             Rogers            South    72756          
1
                             Roseville         Central  48066          
1
                             San Luis Obispo   West     93405          
1
                Home Office  Yuma              West     85364          
1
Length: 2130, dtype: int64
```

we will use groupby function to know more about relationship of columns in detaframe

```
df.groupby('Category')['Sub-
Category','Quantity','Sales'].value_counts()
```

```
Category    Sub-Category  Quantity  Sales
Furniture   Furnishings   3         18.840      6
                          2         6.160       6
                                    41.960      5
            Chairs        2         301.960     4
            Furnishings   2         40.480      4
                                                ..
Technology  Accessories   1         78.150      1
                                    72.000      1
                                    63.992      1
                                    55.200      1
            Phones        14        1075.088    1
Length: 7284, dtype: int64
```

here we will sort the dataframe using sales columns to see proper graph in descending order

```
sorted_sales=df.sort_values(by='Sales',ascending=False)
```

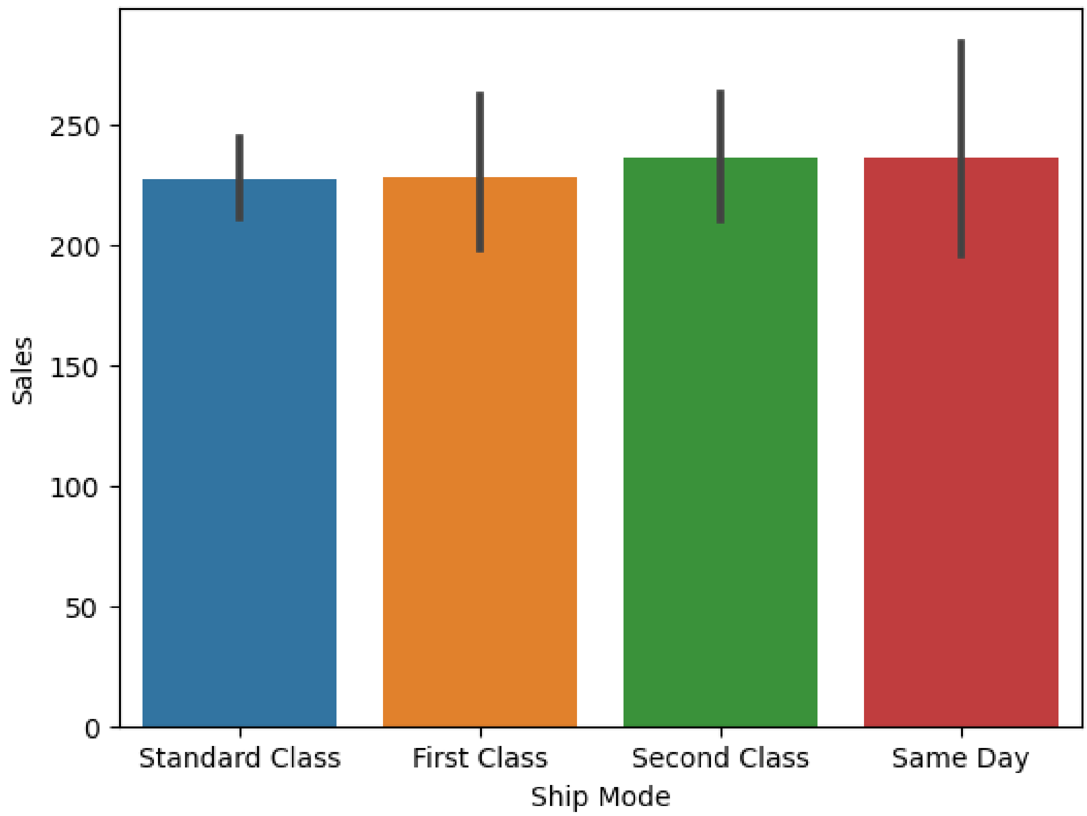
```
sorted_sales.head(2)
```

```
           Ship Mode      Segment        Country          City
State  \
2697  Standard Class  Home Office  United States  Jacksonville
Florida
6826  Standard Class    Corporate  United States     Lafayette
Indiana

      Postal Code   Region    Category Sub-Category     Sales
Quantity  \
2697        32216    South  Technology     Machines  22638.48
6
6826        47905  Central  Technology      Copiers  17499.95
5

      Discount     Profit
2697       0.5 -1811.0784
6826       0.0  8399.9760
```

we will plot barplot using Ship mode on X axis and sales on Y axis

```
sns.barplot(x='Ship
Mode',y='Sales',data=df.sort_values(by='Sales',ascending=False))
```

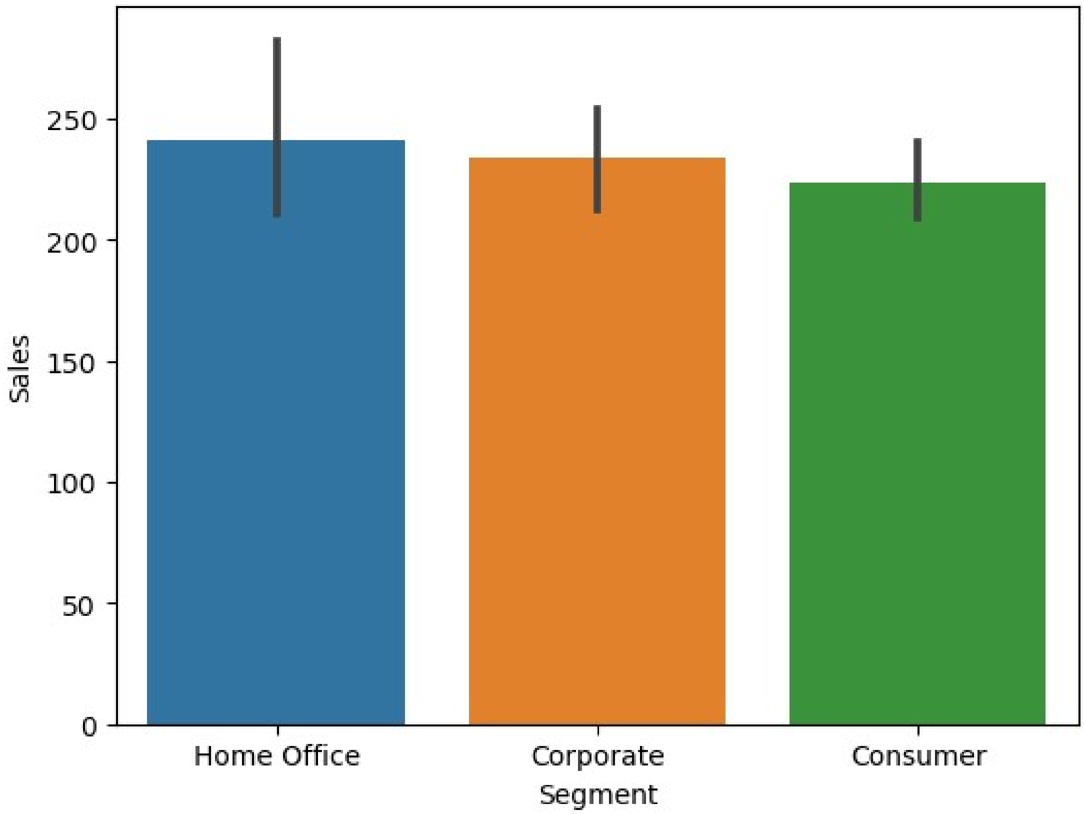
```
<AxesSubplot:xlabel='Ship Mode', ylabel='Sales'>
```



we can conclude that from above graph is range of ship mode of sales is 225 to 230 in the range.

```
sns.barplot(x='Segment',y='Sales',data=df.sort_values(by='Sales',ascending=False))
```

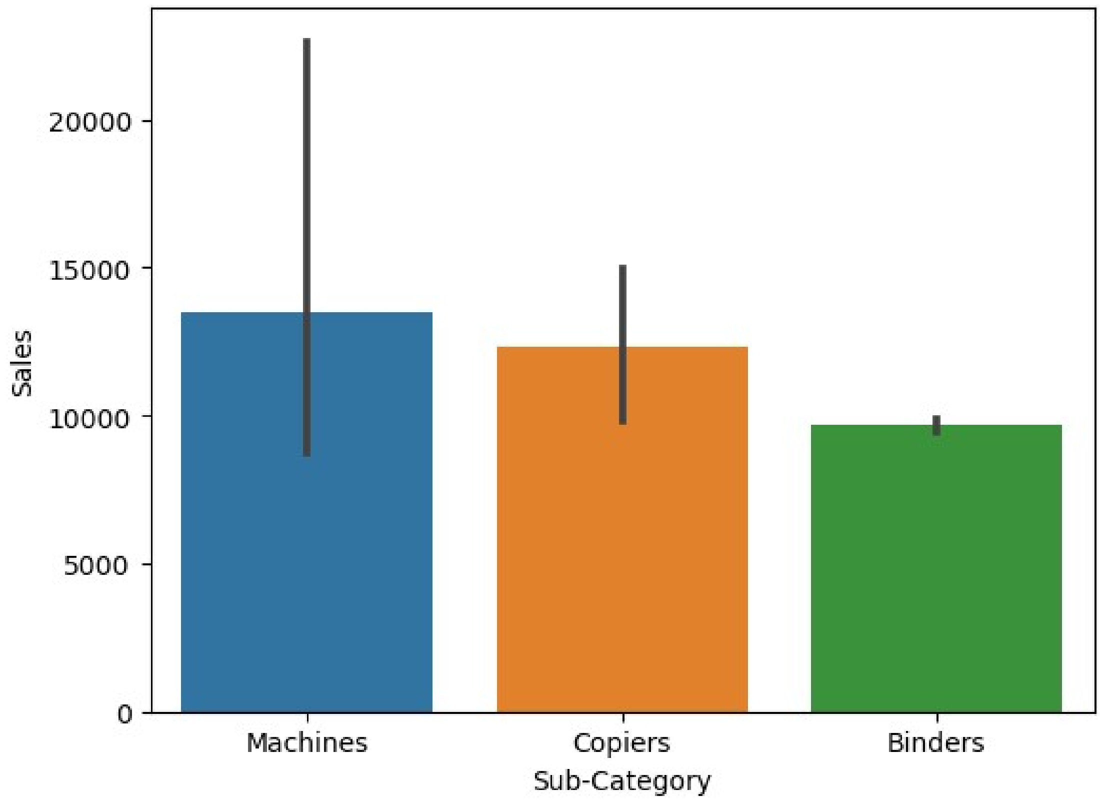
```
<AxesSubplot:xlabel='Segment', ylabel='Sales'>
```



In the above graph we have took segment on X axis and Sales on Y axis so we can conclude that segment home office have highest sales & segment Customer is lowest sales

```
sns.barplot(x='Sub-Category',y='Sales', data=sorted_sales[:10])
```

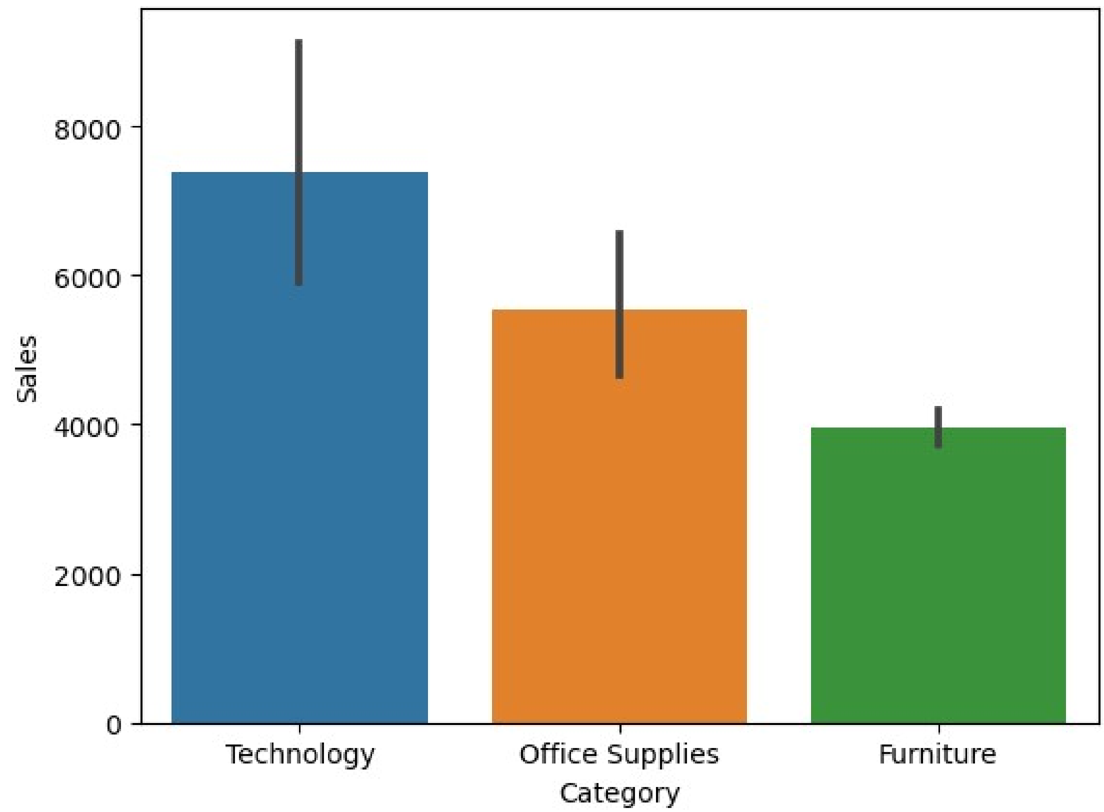
```
<AxesSubplot:xlabel='Sub-Category', ylabel='Sales'>
```



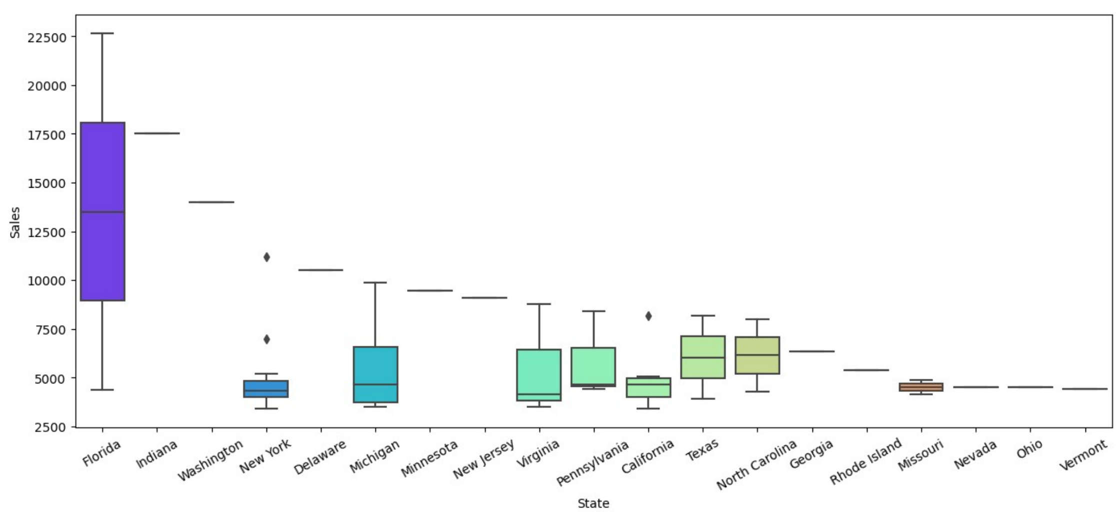
In the above graph we have took sub-catagery on X axis & Sales on Y axis so we ean conclude that machines subcatagery have highet sales than the other & supplies sub Catagery has lowest sales.

```
sns.barplot(x='Category',y='Sales', data=sorted_sales[:50])
```

```
<AxesSubplot:xlabel='Category', ylabel='Sales'>
```

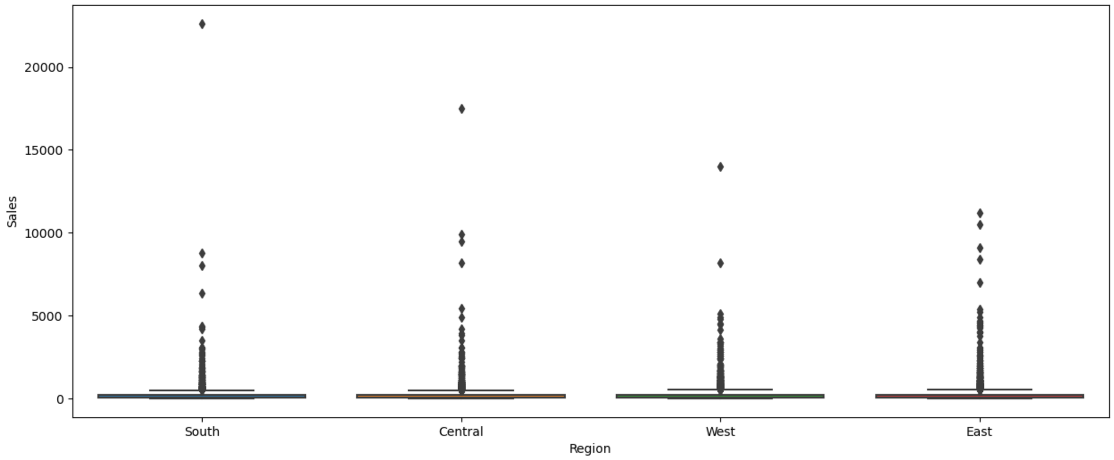



```
# df['State'].boxplotplt.figure(figsize=(20,8))
plt.figure(figsize=(15,6))
sns.boxplot(x="State", y="Sales",
data=df.sort_values(by='Sales',ascending=False)[:50],
palette='rainbow');
plt.xticks(rotation=30);
```

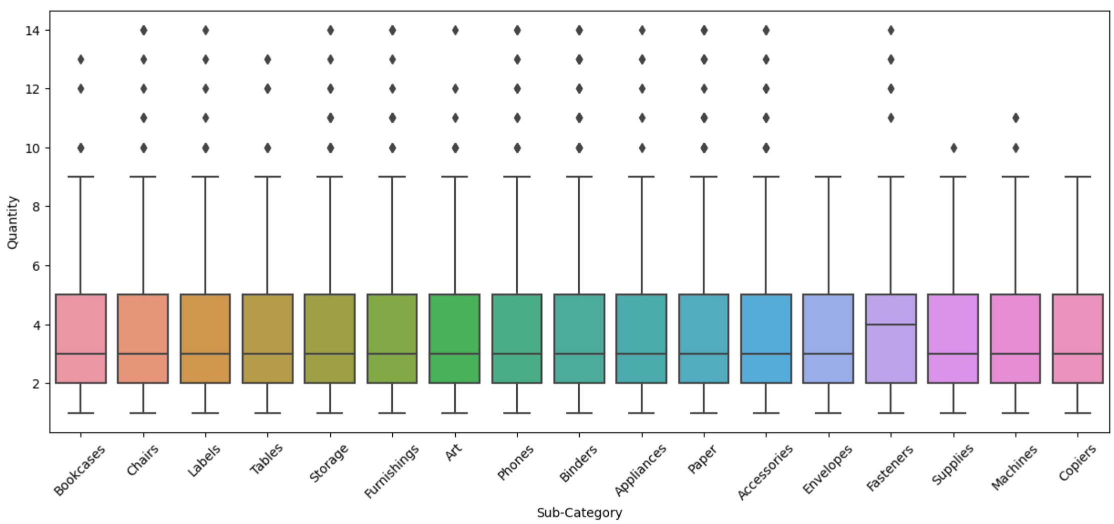



```
plt.figure(figsize=(15,6))
sns.boxplot(x='Region',y='Sales',data=df.sort_values(by='Sales',ascend
ing=False))
```

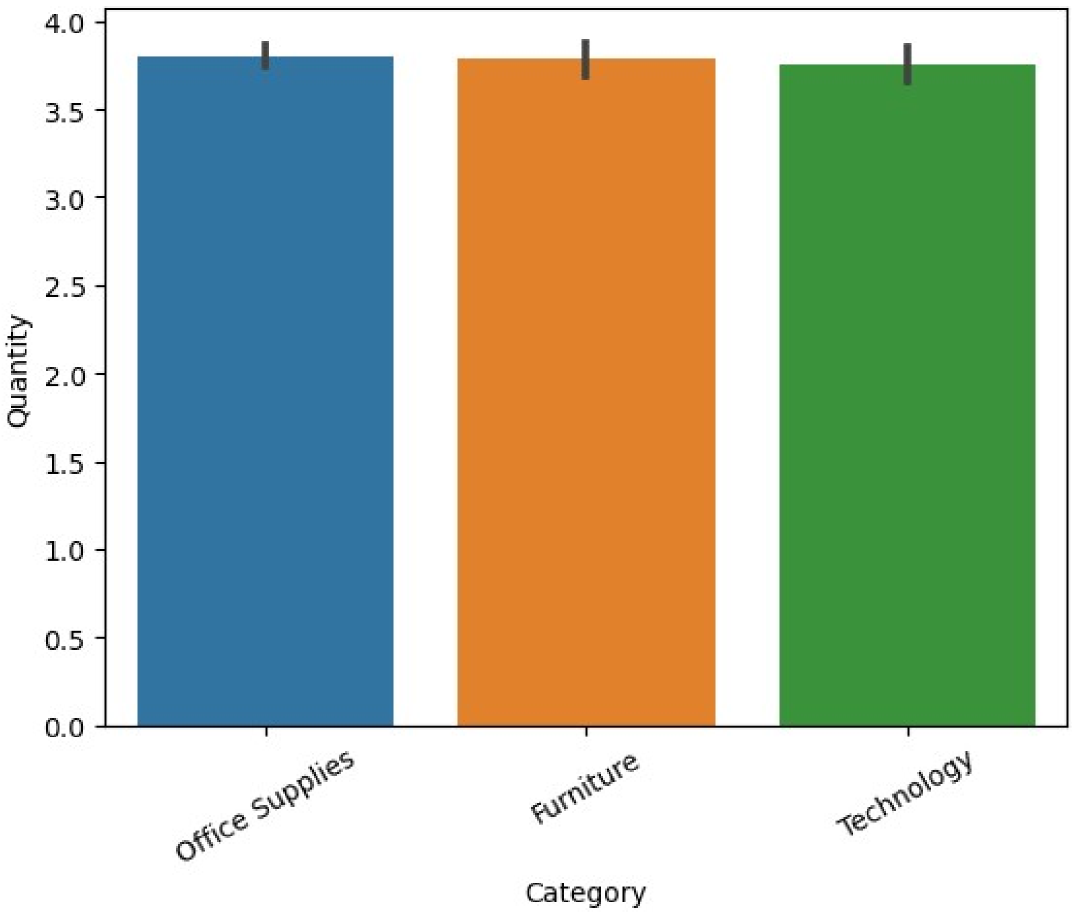
```
<AxesSubplot:xlabel='Region', ylabel='Sales'>
```



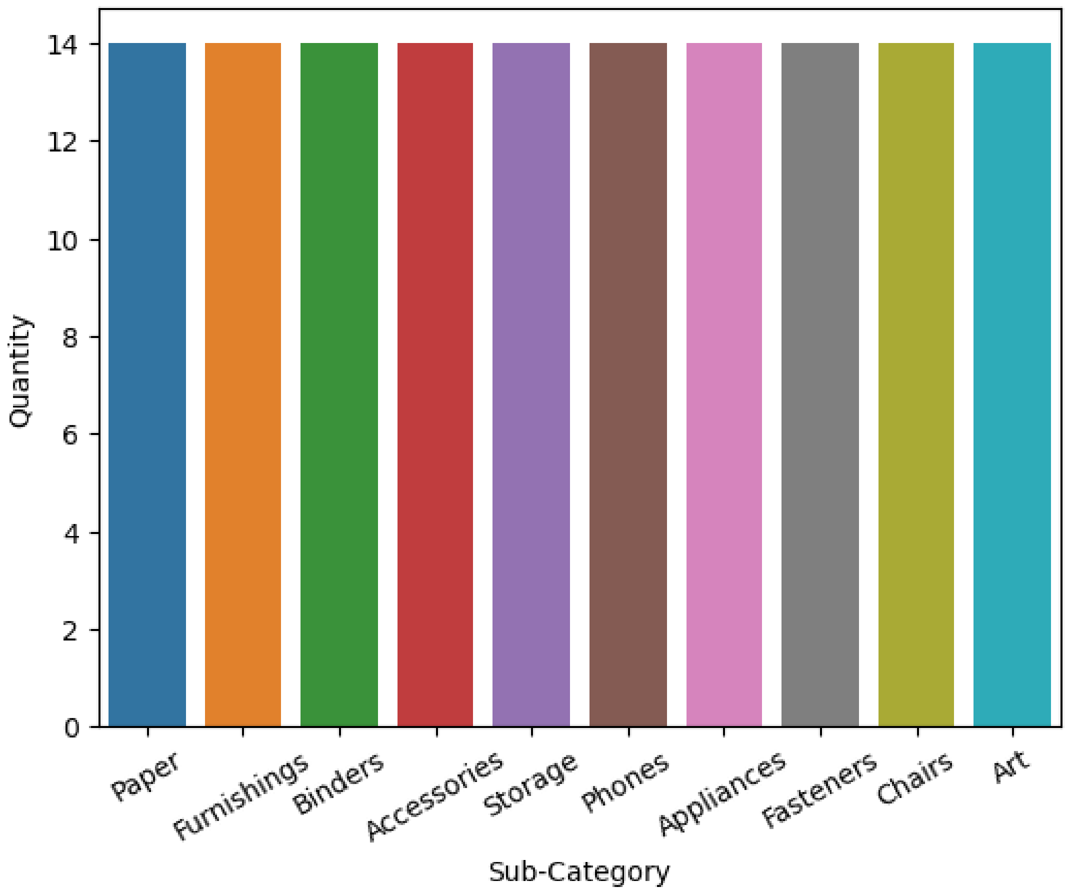

```
plt.figure(figsize=(15,6))
sns.boxplot(x='Sub-Category',y='Quantity',data=df);
plt.xticks(rotation=45);
```




```
sns.barplot(x='Category',y='Quantity',data=df.sort_values(by='Quantity
',ascending=False));
plt.xticks(rotation=30);
```

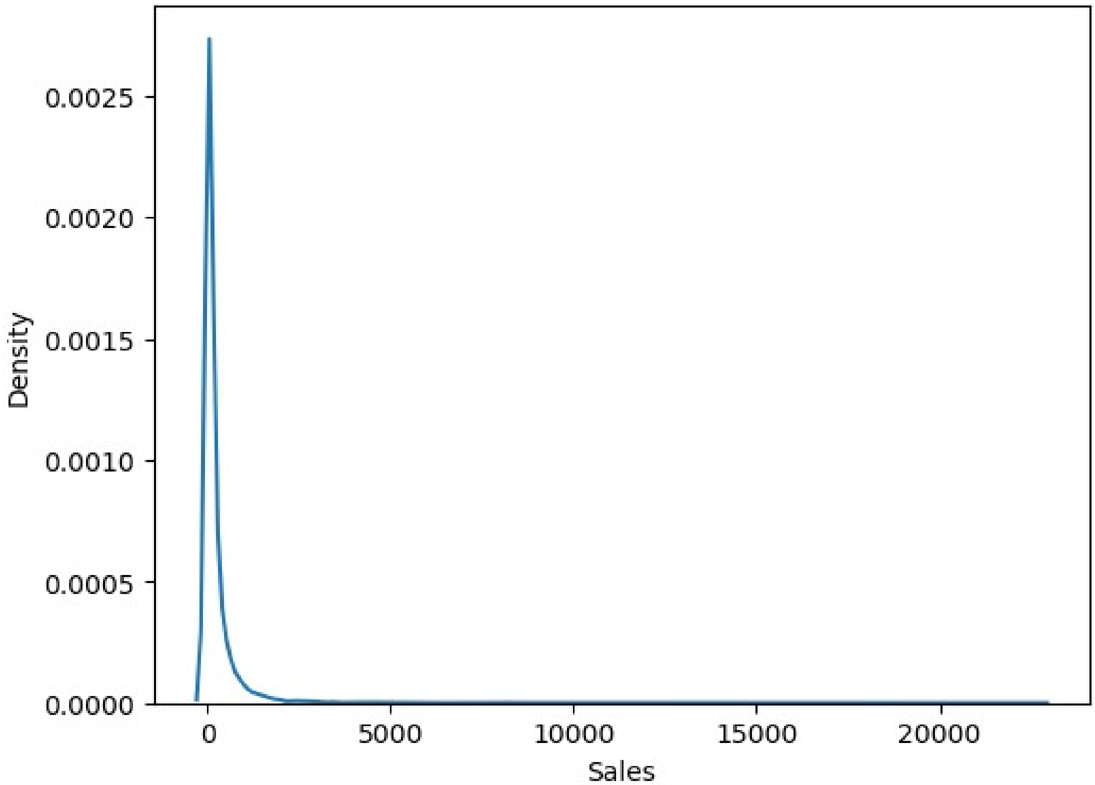



```
sns.barplot(x='Sub-
Category',y='Quantity',data=df.sort_values(by='Quantity',ascending=Fal
se)[:20]);
plt.xticks(rotation=30);
```




```
sns.kdeplot(x='Sales',data=df)
```

```
<AxesSubplot:xlabel='Sales', ylabel='Density'>
```



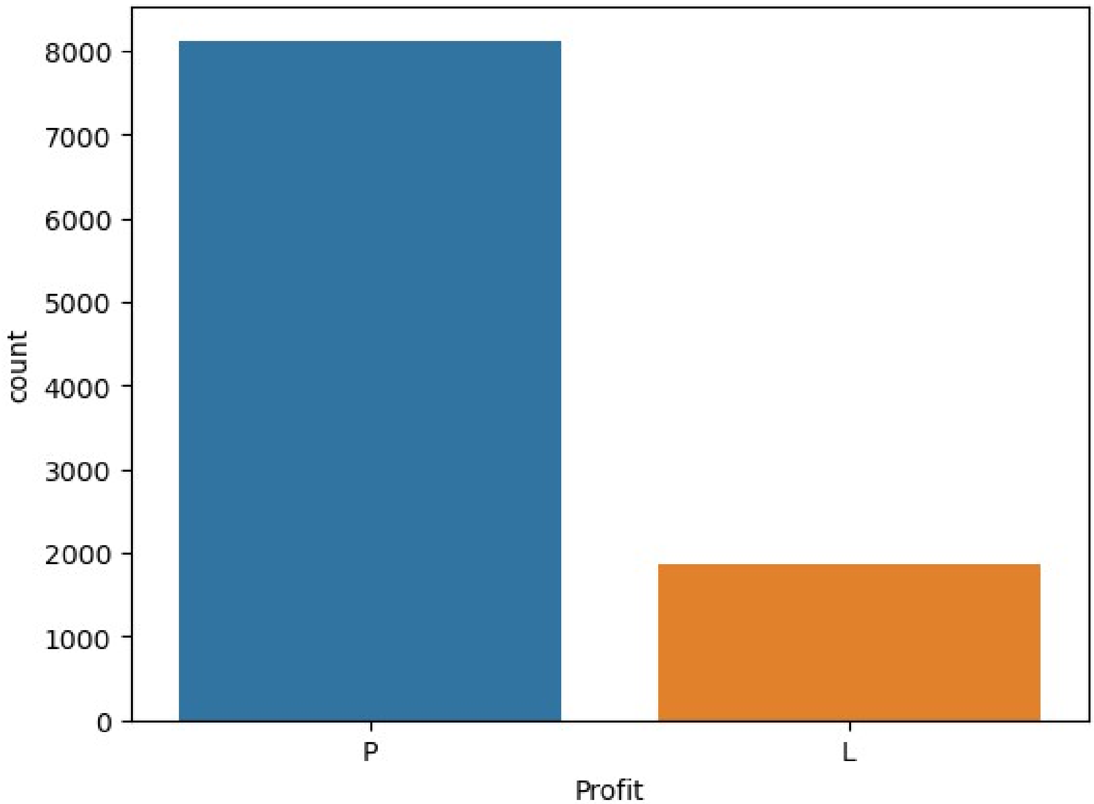
As we have seen our dataset is in continuous form, now we are working on classification alforithm so we need to convert Continuous values of target/ Label to Categorical. thats why we took Positive Values of profit is Profit (P) & negative values of Profit is loss (L)

```
def num_to_str(x):
    if x<0:
        return 'L'
    else:
        return 'P'

df['Profit'] = df['Profit'].apply(num_to_str)
# df['Profit'] = df['Profit'].apply(lambda x : 'L' if x<0  else 'P')
df.head(2)
```

```
      Ship Mode   Segment        Country       City     State  Postal
Code  \
0  Second Class  Consumer  United States  Henderson  Kentucky
42420
1  Second Class  Consumer  United States  Henderson  Kentucky
42420

  Region    Category Sub-Category   Sales  Quantity  Discount Profit
0  South  Furniture    Bookcases  261.96         2       0.0      P
1  South  Furniture       Chairs  731.94         3       0.0      P
```

```
sns.countplot(x='Profit',data=df)
```

```
<AxesSubplot:xlabel='Profit', ylabel='count'>
```



The count of Profin is 8000 & Loss is 1090

For application of ML Algorithm We need to assign Features & Labels se we have assigned All columns except profit as Features & Profit column as Label / Target

```
x=df.drop('Profit',axis=1)
y=df.Profit
```

```
for i in x.columns:
    print(i)
```

```
Ship Mode
Segment
Country
City
State
Postal Code
Region
Category
Sub-Category
Sales
Quantity
Discount
```

```
x['Ship Mode']
```

```
0         Second Class
1         Second Class
2         Second Class
3       Standard Class
4       Standard Class
             ...
9989      Second Class
9990    Standard Class
9991    Standard Class
9992    Standard Class
9993      Second Class
Name: Ship Mode, Length: 9994, dtype: object
```

```
# df.head()
x.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 9994 entries, 0 to 9993
Data columns (total 12 columns):
 #   Column        Non-Null Count  Dtype
---  ------        --------------  -----
 0   Ship Mode     9994 non-null   object
 1   Segment       9994 non-null   object
 2   Country       9994 non-null   object
 3   City          9994 non-null   object
 4   State         9994 non-null   object
 5   Postal Code   9994 non-null   int64
 6   Region        9994 non-null   object
 7   Category      9994 non-null   object
 8   Sub-Category  9994 non-null   object
 9   Sales         9994 non-null   float64
 10  Quantity      9994 non-null   int64
 11  Discount      9994 non-null   float64
dtypes: float64(2), int64(2), object(8)
memory usage: 937.1+ KB
```

for application of Machine Learning algorithm we need to all columns datatypes are in integer in our dataframe some are in Object, we will convert them in to integer by Label Encoding.

```
from sklearn.preprocessing import LabelEncoder
import pickle
# x['Postal Code']=x['Postal Code'].astype('str')
for i in x.select_dtypes('object'):
    lb=LabelEncoder()
    x[i]=lb.fit_transform(x[i])
```

```
x.head()
```

```
   Ship Mode  Segment  Country  City  State  Postal Code  Region  
Category  \
0          2        0        0   194     15        42420       2  
0   
1          2        0        0   194     15        42420       2  
0   
2          2        1        0   266      3        90036       3  
1   
3          3        0        0   153      8        33311       2  
0   
4          3        0        0   153      8        33311       2  
1   

   Sub-Category     Sales  Quantity  Discount  
0             4  261.9600         2      0.00  
1             5  731.9400         3      0.00  
2            10   14.6200         2      0.00  
3            16  957.5775         5      0.45  
4            14   22.3680         2      0.20  
```

```
x.shape, y.shape
```

```
((9994, 12), (9994,))
```

1) Application of KNN to Supermarket algorithm

we have build training & Testing model for application of ML Algorithm

```
from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.2,rando
m_state=0)
x_train.shape,x_test.shape,y_train.shape,y_test.shape
```

```
((7995, 12), (1999, 12), (7995,), (1999,))
```

```
from sklearn.neighbors import KNeighborsClassifier
knn = KNeighborsClassifier( n_neighbors =5 , metric ='manhattan')
knn.fit(x_train, y_train)
```

```
KNeighborsClassifier(metric='manhattan')
```

```
knn_train=knn.score(x_train,y_train)
knn_train
```

```
0.8915572232645403
```

```
knn_test=knn.score(x_test,y_test)
knn_test
```

```
0.8254127063531765
```

Training & Testing Csore of model using KNN is 89% & 82% respectively which is Good.

```
sample=x.iloc[9992]
knn.predict([sample])
```

```
array(['P'], dtype=object)
```

```
y.iloc[9992]
```

```
'P'
```

```
y_test_predict=knn.predict(x_test)
```

```
y_test_predict
```

```
array(['P', 'P', 'P', ..., 'P', 'P', 'P'], dtype=object)
```

We will see Confusion Matrics to see Accurecy Score also

```
from sklearn.metrics import confusion_matrix
conf_matrix = confusion_matrix(y_test, y_test_predict)
conf_matrix
```

```
array([[ 174,  195],
       [ 154, 1476]], dtype=int64)
```

```
true_neg, false_pos, false_neg, true_pos = conf_matrix.ravel()
true_neg, false_pos, false_neg, true_pos
```

```
(174, 195, 154, 1476)
```

```
from sklearn.metrics import accuracy_score
accuracy_score(y_test, y_test_predict)
```

```
0.8254127063531765
```

Accurecy Score Using Confusion Matrics is 82%.

```
Accurecy=(true_pos+true_neg)/(true_neg + false_pos + false_neg +
true_pos)
Accurecy
```

```
0.8254127063531765
```

```
Error_rate=(1-Accurecy)
Error_rate
```

```
0.17458729364682346
```

```
precision=(true_pos)/(true_pos+false_pos)
precision
```

```
0.8833034111310593
```

```
Reacll=(true_pos)/(true_pos+false_neg)
Reacll
```

```
0.905521472392638
```

Precision is 88% & Recall is 90%

2) Application of SVM to Super_market_df

```
# from sklearn.svm import SVC
# svm=SVC(kernel='linear',C=10E10)
# svm.fit(x_train.head(500),y_train.head(500))

# svm.score(x_train.head(500),y_train.head(500))

# svm.score(x_train.head(500),y_train.head(500))
```

As we know SVM Takes too much of time to run the code. here also it is taking lot of time, so we just skip this algorithm & keep its code in comment.

3) Application of Naive bayers to Super_market_df

```
from sklearn.naive_bayes import GaussianNB
nb=GaussianNB()
nb.fit(x_train,y_train)
```

```
GaussianNB()
```

```
nb_train=nb.score(x_train,y_train)
nb_train
```

```
0.8043777360850531
```

```
nb_test=nb.score(x_test,y_test)
nb_test
```

```
0.8009004502251126
```

Training & Testing Models Score Using Naive Bayes Algorithm is 80% & 80% Respectevely.

4) Application of Decision Tree to Super_market_df

```
from sklearn.tree import DecisionTreeClassifier
dt = DecisionTreeClassifier(criterion='gini', max_depth=10)
dt.fit(x_train,y_train)
```

```
DecisionTreeClassifier(max_depth=10)
```

```
dt_train_g=dt.score(x_train,y_train)
dt_train_g
```

```
0.9698561601000626
```

```
dt_test_g=dt.score(x_test,y_test)
dt_test_g
```

```
0.9399699849924963
```

Training & Testing Models Score Using Decision Tree Algorithm is 97% & 93% Respectevely.(Criterion -Gini)

```
dt = DecisionTreeClassifier(criterion='entropy',
max_depth=10,random_state=0)
dt.fit(x_train,y_train)
```

```
DecisionTreeClassifier(criterion='entropy', max_depth=10,
random_state=0)
```

```
dt_train_e=dt.score(x_train,y_train)
dt_train_e
```

```
0.9654784240150094
```

```
dt_test_e=dt.score(x_test,y_test)
dt_test_e
```

```
0.9424712356178089
```

Training & Testing Models Score Using Decision Tree Algorithm is 96% & 94% Respectevely.(Criterion -Entropy)

5) Application of Random Forest to Super_market_df

```
from sklearn.ensemble import RandomForestClassifier
```

```
clf=RandomForestClassifier()
```

```
clf.fit(x_train,y_train)
```

```
RandomForestClassifier()
```

```
clf_train=clf.score(x_train,y_train)
clf_train
```

```
1.0
```

```
clf_test=clf.score(x_test,y_test)
clf_test
```

```
0.9479739869934968
```

Using Random Forest Algorithm Trainig Models Score is 1% & Testing Models score is 94%. Training Models Score is 1% which means model is Overfit. which is not good.

we will do feature inportance & Then apply algorithm to reduce the Overfit Score & make it normalize

```
clf.feature_importances_
```

```
array([0.0173363 , 0.01665007, 0.        , 0.04246421, 0.06688963,
       0.07753466, 0.02349926, 0.03058708, 0.09835788, 0.09037368,
       0.03629759, 0.50000963])
```

```
x.columns
```

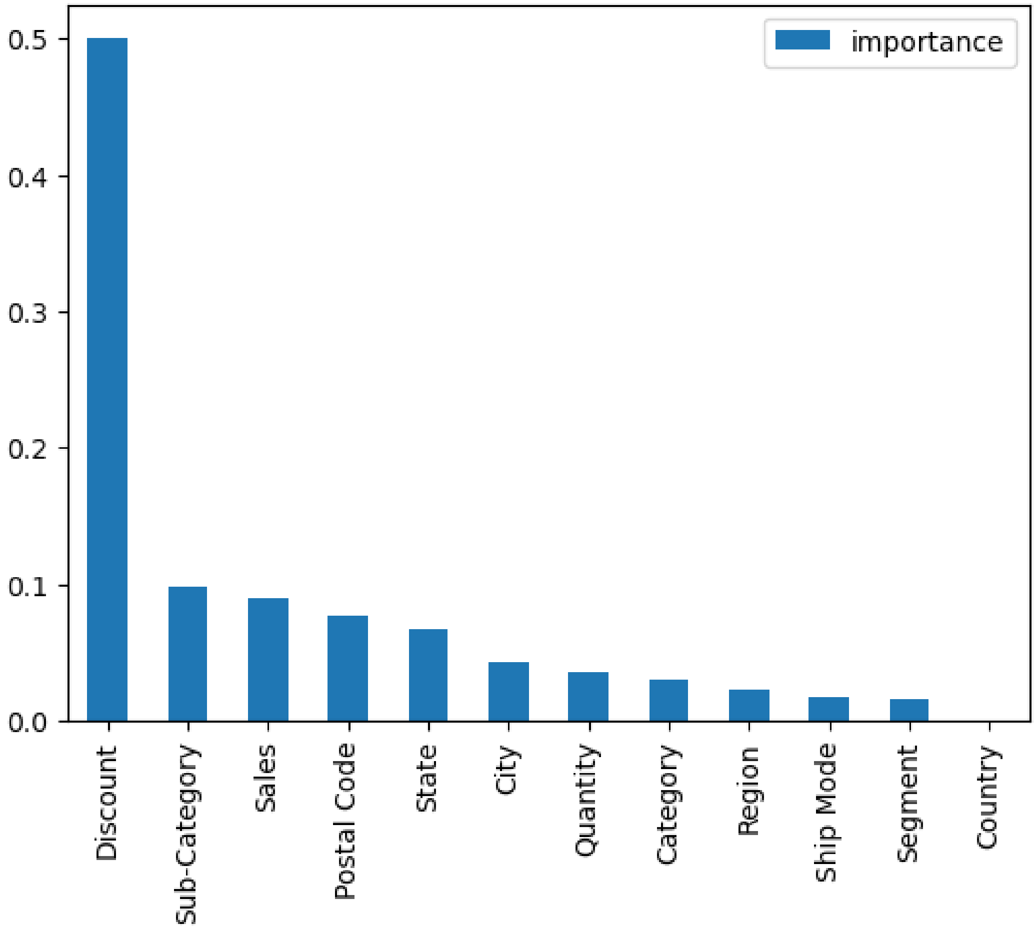
```
Index(['Ship Mode', 'Segment', 'Country', 'City', 'State', 'Postal
Code',
       'Region', 'Category', 'Sub-Category', 'Sales', 'Quantity',
'Discount'],
      dtype='object')
```

```
df2=pd.DataFrame({'importance':clf.feature_importances_},index=x.colum
ns).sort_values(by='importance',ascending=False)
df2.head()
```

	importance
Discount	0.500010
Sub-Category	0.098358
Sales	0.090374
Postal Code	0.077535
State	0.066890

```
df2.plot.bar()
```

```
<AxesSubplot:>
```




```
imp_feat=df2[df2['importance']>=0.05]
imp_feat
```

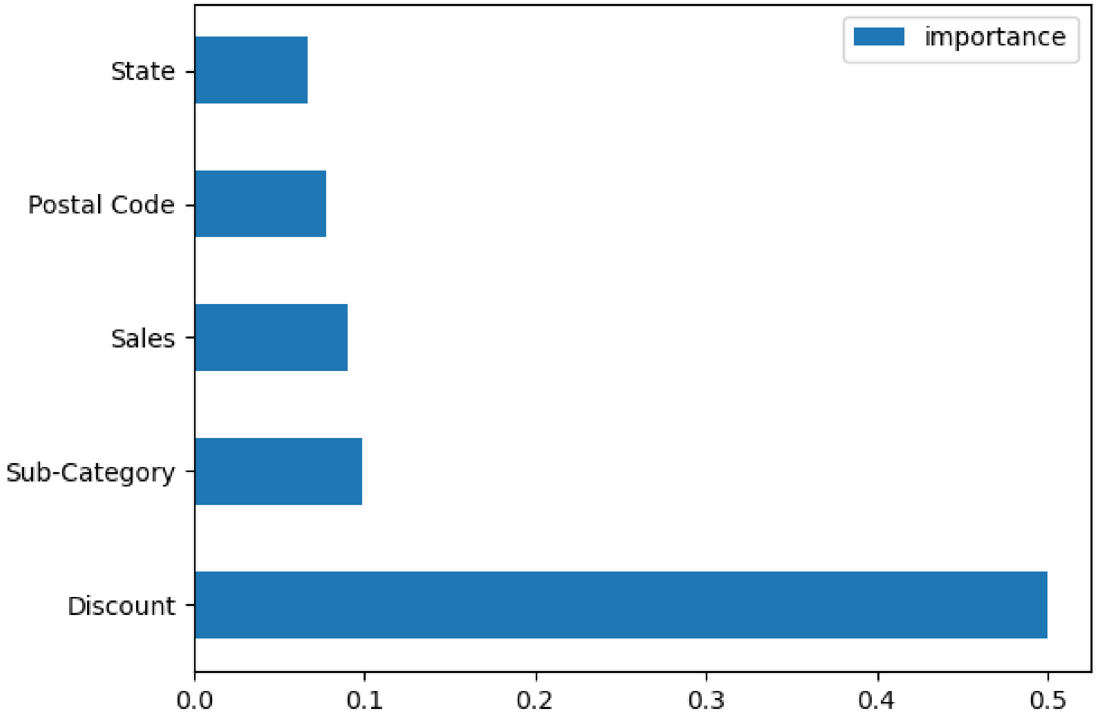
```
              importance
Discount        0.500010
Sub-Category    0.098358
Sales           0.090374
Postal Code     0.077535
State           0.066890
```

```
imp_feat.index
```

```
Index(['Discount', 'Sub-Category', 'Sales', 'Postal Code', 'State'],
dtype='object')
```

```
imp_feat.plot.barh()
```

```
<AxesSubplot:>
```



Here We will See Hyperparameter Tuning to All The algorithms One by one.

6) Application of randomizedsearch cv to KNN of Super_market_df

```
from sklearn.model_selection import GridSearchCV, RandomizedSearchCV # for hyperparameter tuning
knn=KNeighborsClassifier()

param_dist = {'n_neighbors':range(1,15),"metric":['euclidean','manhattan']}

random_search = RandomizedSearchCV(knn, param_distributions=param_dist, n_iter=10, cv=5)
random_search.fit(x_train,y_train)

RandomizedSearchCV(cv=5, estimator=KNeighborsClassifier(),
                   param_distributions={'metric': ['euclidean', 'manhattan'],
                                        'n_neighbors': range(1, 15)})

knn_random_train=random_search.score(x_train,y_train)
knn_random_train

0.8915572232645403

knn_random_test=random_search.score(x_test,y_test)
knn_random_test

0.8254127063531765
```

Score Of model Using Hyperparameters Randomsearchcv is 86 & 82 Training & Testing

7) Application of gridsearch cv to KNN Super_market_df

```
grid_search={'n_neighbors':range(1,15),"metric":
['euclidean','manhattan']}

grid_search = GridSearchCV(knn, param_grid=grid_search, cv=5)
grid_search.fit(x_train,y_train)

GridSearchCV(cv=5, estimator=KNeighborsClassifier(),
             param_grid={'metric': ['euclidean', 'manhattan'],
                         'n_neighbors': range(1, 15)})

knn_grid_train=grid_search.score(x_train,y_train)
knn_grid_train

0.8915572232645403

knn_grid_test=grid_search.score(x_test,y_test)
knn_grid_test

0.8254127063531765
```

Score Of model Using Hyperparameters Gridsearchcv is 89% & 82% Training & Testing

8) Application of Hyper parameter to decision Tree algoritham of Supermarket_df

```
clf=DecisionTreeClassifier()

param_dist = {'max_depth': [3,None],
             'max_features':range(1,11),
             'min_samples_split':range(2,11),
              'criterion':['gini','entropy']}

random_search=RandomizedSearchCV(clf,param_distributions=param_dist,n_
iter=10,cv=5)

random_search.fit(x_train,y_train)

RandomizedSearchCV(cv=5, estimator=DecisionTreeClassifier(),
                   param_distributions={'criterion': ['gini',
'entropy'],
                                        'max_depth': [3, None],
                                        'max_features': range(1, 11),
                                        'min_samples_split': range(2,
11)})

random_search.get_params().keys()
```

```
dict_keys(['cv', 'error_score', 'estimator__ccp_alpha',
'estimator__class_weight', 'estimator__criterion',
'estimator__max_depth', 'estimator__max_features',
'estimator__max_leaf_nodes', 'estimator__min_impurity_decrease',
'estimator__min_samples_leaf', 'estimator__min_samples_split',
'estimator__min_weight_fraction_leaf', 'estimator__random_state',
'estimator__splitter', 'estimator', 'n_iter', 'n_jobs',
'param_distributions', 'pre_dispatch', 'random_state', 'refit',
'return_train_score', 'scoring', 'verbose'])
```

```
dt_random_train=random_search.score(x_train,y_train)
dt_random_train
```

```
0.9428392745465917
```

```
dt_random_test=random_search.score(x_test,y_test)
dt_random_test
```

```
0.9394697348674337
```

Score Of model Using Hyperparameters Randomsearchcv is 94% & 93% Training & Testing

9) Application of Hyper parameter to Ensembling algoritham of Supermarket_df

```
rfc=RandomForestClassifier()
```

```
from sklearn.model_selection import GridSearchCV, RandomizedSearchCV #
for hyperparameter tuning
param_dist={"max_depth": [5, None],
              "max_features": range(1, 11),
              "min_samples_split": range(2, 11),
              "criterion": ["gini", "entropy"]}
```

```
randomens=RandomizedSearchCV(rfc,param_distributions=param_dist,n_iter
=10,cv=5)
randomens
```

```
RandomizedSearchCV(cv=5, estimator=RandomForestClassifier(),
                   param_distributions={'criterion': ['gini',
'entropy'],
                                        'max_depth': [5, None],
                                        'max_features': range(1, 11),
                                        'min_samples_split': range(2,
11)})
```

```
randomens.fit(x_train,y_train)
```

```
RandomizedSearchCV(cv=5, estimator=RandomForestClassifier(),
                   param_distributions={'criterion': ['gini',
'entropy'],
```

```
                                        'max_depth': [5, None],
                                        'max_features': range(1, 11),
                                        'min_samples_split': range(2,
11)})
```

```
ens_random_train=randomens.score(x_train,y_train)
ens_random_train
```

```
0.9923702313946217
```

```
ens_random_test=randomens.score(x_test,y_test)
ens_random_test
```

```
0.9479739869934968
```

Score Of model Using Hyperparameters Randomsearchcv is 1% & 95 Training & Testing

```
# gsc=GridSearchCV()
param_dist={"max_depth": [2, None],
              "max_features": range(1, 12),
              "min_samples_split": range(2, 10),
              "criterion": ["gini", "entropy"]}
```

```
grid_ens=GridSearchCV(rfc,param_grid=param_dist,cv=10)
grid_ens
```

```
GridSearchCV(cv=10, estimator=RandomForestClassifier(),
             param_grid={'criterion': ['gini', 'entropy'],
                         'max_depth': [2, None], 'max_features':
range(1, 12),
                         'min_samples_split': range(2, 10)})
```

```
# grid_ens.fit(x_train,y_train)
```

```
# grid_ens.score(x_train,y_train)
```

```
# grid_ens.score(x_test,y_test)
```

```
from sklearn.ensemble import AdaBoostClassifier
mod=AdaBoostClassifier()
model=mod.fit(x_train,y_train)
y_test_pred=model.predict(x_test)
x_train.shape,x_test.shape,y_train.shape,y_test.shape,y_test_pred.shap
e
```

```
((7995, 12), (1999, 12), (7995,), (1999,), (1999,))
```

```
y_test_pred
```

```
array(['P', 'P', 'P', ..., 'P', 'P', 'P'], dtype=object)
```

```
y_test_pred.shape,y_test.shape
```

```
((1999,), (1999,))
```

```
from sklearn.metrics import confusion_matrix, classification_report
conf_matrix = confusion_matrix(y_test, y_test_pred)
conf_matrix
```

```
array([[ 293,   76],
       [  36, 1594]], dtype=int64)
```

```
true_neg, false_pos, false_neg, true_pos = conf_matrix.ravel()
true_neg, false_pos, false_neg, true_pos
```

```
(293, 76, 36, 1594)
```

```
print(classification_report(y_test, y_test_pred))
```

```
              precision    recall  f1-score   support

           L       0.89      0.79      0.84       369
           P       0.95      0.98      0.97      1630

    accuracy                           0.94      1999
   macro avg       0.92      0.89      0.90      1999
weighted avg       0.94      0.94      0.94      1999
```

```
mod.score(x_train,y_train)
```

```
0.9463414634146341
```

```
mod.score(x_test,y_test)
```

```
0.9439719859929965
```

feature importance to df

```
imp_frea_x=x.loc[:,imp_feat.index]
imp_frea_x
```

```
      Discount  Sub-Category     Sales  Postal Code  State
0         0.00             4  261.9600        42420     15
1         0.00             5  731.9400        42420     15
2         0.00            10   14.6200        90036      3
3         0.45            16  957.5775        33311      8
4         0.20            14   22.3680        33311      8
...        ...           ...       ...          ...    ...
9989      0.20             9   25.2480        33180      8
9990      0.00             9   91.9600        92627      3
9991      0.20            13  258.5760        92627      3
9992      0.00            12   29.6000        92627      3
9993      0.00             1  243.1600        92683      3

[9994 rows x 5 columns]
```

```
y
```

```
0       P
1       P
2       P
3       L
4       P
       ..
9989    P
9990    P
9991    P
9992    P
9993    P
Name: Profit, Length: 9994, dtype: object
```

```
imp_frea_x.shape,y.shape
```

```
((9994, 5), (9994,))
```

KNN algorithm to important features

We Will See applying KNN algorithm for important features

```
x_train,x_test,y_train,y_test=train_test_split(imp_frea_x,y,random_sta
te=0)
```

```
fea_knn=KNeighborsClassifier(n_neighbors=25,metric='euclidean')
```

```
fea_knn.fit(x_train,y_train)
```

```
KNeighborsClassifier(metric='euclidean', n_neighbors=25)
```

```
fea_knn_train=fea_knn.score(x_train,y_train)
fea_knn_train
```

```
0.8414943295530354
```

```
fea_knn_test=fea_knn.score(x_test,y_test)
fea_knn_test
```

```
0.82953181272509
```

```
y_test_predict=fea_knn.predict(x_test)
y_test_predict
```

```
array(['P', 'P', 'P', ..., 'P', 'P', 'P'], dtype=object)
```

```
y_test_predict,y_test_predict.shape
```

```
(array(['P', 'P', 'P', ..., 'P', 'P', 'P'], dtype=object), (2499,))
```

```
from sklearn.metrics import confusion_matrix, classification_report
conf_matrix = confusion_matrix(y_test, y_test_predict)
conf_matrix
```

```
array([[ 161,  289],
       [ 137, 1912]], dtype=int64)
```

```
true_neg, false_pos, false_neg, true_pos = conf_matrix.ravel()
true_neg, false_pos, false_neg, true_pos
```

```
(161, 289, 137, 1912)
```

```
print(classification_report(y_test, y_test_predict))
```

```
              precision    recall  f1-score   support

           L       0.54      0.36      0.43       450
           P       0.87      0.93      0.90      2049

    accuracy                           0.83      2499
   macro avg       0.70      0.65      0.67      2499
weighted avg       0.81      0.83      0.82      2499
```

```
fea_knn.score(x_train,y_train)
```

```
0.8414943295530354
```

```
fea_knn.score(x_test,y_test)
```

```
0.82953181272509
```

Score of models knn algorithm if important features is 84% & 82%

Naive bayes algorithm to important features

```
from sklearn.naive_bayes import GaussianNB
fea_nb=GaussianNB()
fea_nb.fit(x_train,y_train)
```

```
GaussianNB()
```

```
fea_nb_train=fea_nb.score(x_train,y_train)
fea_nb_train
```

```
0.8010673782521681
```

```
fea_nb_test=fea_nb.score(x_test,y_test)
fea_nb_test
```

```
0.8047218887555022
```

```
fea_nb_y_pred=fea_nb.predict(x_test)
fea_nb_y_pred
```

```
array(['P', 'P', 'P', ..., 'P', 'P', 'P'], dtype='<U1')
```

```
conf_matrix = confusion_matrix(y_test, fea_nb_y_pred)
conf_matrix
```

```
array([[   7,  443],
       [  45, 2004]], dtype=int64)
```

Decision tree to features importance

```
from sklearn.tree import DecisionTreeClassifier
fea_dtc=DecisionTreeClassifier(criterion='gini', max_depth=10)
fea_dtc.fit(x_train,y_train)
```

```
DecisionTreeClassifier(max_depth=10)
```

```
fea_dtc_train=fea_dtc.score(x_train,y_train)
fea_dtc_train
```

```
0.962374916611074
```

```
fea_dtc_test=fea_dtc.score(x_test,y_test)
fea_dtc_test
```

```
0.9431772709083633
```

```
fea_dtc_y_pred=fea_dtc.predict(x_test)
fea_dtc_y_pred
```

```
array(['P', 'P', 'P', ..., 'P', 'P', 'P'], dtype=object)
```

```
conf_matrix = confusion_matrix(y_test, fea_dtc_y_pred)
conf_matrix
```

```
array([[ 362,   88],
       [  54, 1995]], dtype=int64)
```

```
param_dist={'max_depth': [3,None],
             'max_features':range(1,11),
             'min_samples_split':range(2,11),
              'criterion':['gini','entropy']}
```

```
fea_random=RandomizedSearchCV(fea_dtc,param_distributions=param_dist,n
_iter=10,cv=10)
fea_random.fit(x_train,y_train)
```

```
RandomizedSearchCV(cv=10,
estimator=DecisionTreeClassifier(max_depth=10),
                   param_distributions={'criterion': ['gini',
'entropy'],
```

```
                                        'max_depth': [3, None],
                                        'max_features': range(1, 11),
                                        'min_samples_split': range(2, 
11)})
```

```
fea_random_dtc_train=fea_random.score(x_train,y_train)
fea_random_dtc_train
```

```
0.9419613075383589
```

```
fea_random_dtc_test=fea_random.score(x_test,y_test)
fea_random_dtc_test
```

```
0.9427771108443377
```

Application of Ensembling to Imp_features

```
fea_rfc=RandomForestClassifier()
```

```
fea_rfc.fit(x_train,y_train)
```

```
RandomForestClassifier()
```

```
fea_rfc_train=fea_rfc.score(x_train,y_train)
fea_rfc_train
```

```
0.999866577718479
```

```
fea_rfc_test=fea_rfc.score(x_test,y_test)
fea_rfc_test
```

```
0.9387755102040817
```

```
fea_rfc_y_pred=fea_rfc.predict(x_test)
fea_rfc_y_pred
```

```
array(['P', 'P', 'P', ..., 'P', 'P', 'P'], dtype=object)
```

```
conf_matrix = confusion_matrix(y_test, fea_rfc_y_pred)
conf_matrix
```

```
array([[ 369,   81],
       [  72, 1977]], dtype=int64)
```

```
param_dist={"max_depth": [5, None],
              "max_features": range(1, 15),
              "min_samples_split": range(2, 15),
              "criterion": ["gini", "entropy"]}
```

```
fea_random_sea=RandomizedSearchCV(fea_rfc,param_distributions=param_di
st,n_iter=10,cv=10)
fea_random_sea
```

```
RandomizedSearchCV(cv=10, estimator=RandomForestClassifier(),
                   param_distributions={'criterion': ['gini',
'entropy'],
                                        'max_depth': [5, None],
                                        'max_features': range(1, 15),
                                        'min_samples_split': range(2,
15)})
```

```
fea_random_sea.fit(x_train,y_train)
```

```
RandomizedSearchCV(cv=10, estimator=RandomForestClassifier(),
                   param_distributions={'criterion': ['gini',
'entropy'],
                                        'max_depth': [5, None],
                                        'max_features': range(1, 15),
                                        'min_samples_split': range(2,
15)})
```

```
fea_random_sea_train=fea_random_sea.score(x_train,y_train)
fea_random_sea_train
```

```
0.9751834556370914
```

```
fea_random_sea_test=fea_random_sea.score(x_test,y_test)
fea_random_sea_test
```

```
0.9431772709083633
```

```
from sklearn.ensemble import AdaBoostClassifier
fea_ad=AdaBoostClassifier()
fea_model=fea_ad.fit(x_train,y_train)
y_test_pred=fea_model.predict(x_test)
y_test_pred
```

```
array(['P', 'P', 'P', ..., 'P', 'P', 'P'], dtype=object)
```

```
conf_matrix = confusion_matrix(y_test, fea_rfc_y_pred)
conf_matrix
```

```
array([[ 369,   81],
       [  72, 1977]], dtype=int64)
```

```
fea_model_ada_train=fea_model.score(x_train,y_train)
fea_model_ada_train
```

```
0.9454302868579053
```

```
fea_model_ada_test=fea_model.score(x_test,y_test)
fea_model_ada_test
```

```
0.9443777511004402
```

```
import pandas as pd
knn=KNeighborsClassifier()
```

Score After Hyper Parameter Tunning

```
df_hyper_T=pd.DataFrame([knn_random_train,knn_grid_train,dt_random_train,ens_random_train],index=['Knn_random','Knn_grid','dt_random','ens_random'])

df_hyper_T[0]

Knn_random    0.891557
Knn_grid      0.891557
dt_random     0.942839
ens_random    0.992370
Name: 0, dtype: float64
```

Score after models feature Importance

```
imp_features=pd.DataFrame([fea_knn_train,fea_nb_train,fea_dtc_train,fea_dtc_train,fea_rfc_train,fea_random_sea_train],index=['Knn','naive bayes','Decision Tree','rand_dtc','ensembling','random_ens'])

imp_features

                      0
Knn            0.841494
naive bayes    0.801067
Decision Tree  0.962375
rand_dtc       0.962375
ensembling     0.999867
random_ens     0.975183

imp_fea_score_df = 
pd.DataFrame(imp_features,columns=['Train_score','Test_score'])

imp_fea_score_df['Test_score']=[fea_knn_test,fea_nb_test,fea_dtc_test,fea_dtc_test,fea_rfc_test,fea_random_sea_test]

imp_fea_score_df['Train_score']=imp_features[0]
imp_fea_score_df

               Train_score  Test_score
Knn               0.841494    0.829532
naive bayes       0.801067    0.804722
Decision Tree     0.962375    0.943177
rand_dtc          0.962375    0.943177
ensembling        0.999867    0.938776
random_ens        0.975183    0.943177

imp_fea_score_df.style.highlight_max(color='green')
```

```
<pandas.io.formats.style.Styler at 0x262f8019940>
```

```
imp_fea_score_df.style.highlight_min(color='pink')
```

```
<pandas.io.formats.style.Styler at 0x262fc1d32e0>
```

```
imp_fea_score_df.style.set_properties(**{'border': '1.3px solid
black',
                          'color': 'Black'})
```

```
<pandas.io.formats.style.Styler at 0x262fc1d3ca0>
```

```
diff=imp_fea_score_df['Train_score']-imp_fea_score_df['Test_score']
diff.idxmax(),diff.max()
```

```
('ensembling', 0.0610910675143973)
```

Table Shows Training & Testing Models Score After Hyperparameter tunung Using All The Algorithms.