**Phase 3 – product demand prediction using machine learning.**

* **Importing the required libraries (data.csv)**

import numpy as np

import pandas as pd

import os

for dirname, \_, filenames in os.walk('/content/PoductDemand.csv.zip'):

    for filename in filenames:

        print(os.path.join(dirname, filename))

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

from sklearn.model\_selection import train\_test\_split

from sklearn.metrics import f1\_score, accuracy\_score, confusion\_matrix ,classification\_report

from sklearn.ensemble import RandomForestClassifier

from sklearn.svm import SVC

from sklearn.tree import DecisionTreeClassifier

from sklearn.ensemble import RandomForestClassifier, AdaBoostClassifier, GradientBoostingClassifier

from plotly.offline import iplot

import plotly as py

import plotly.tools as tls

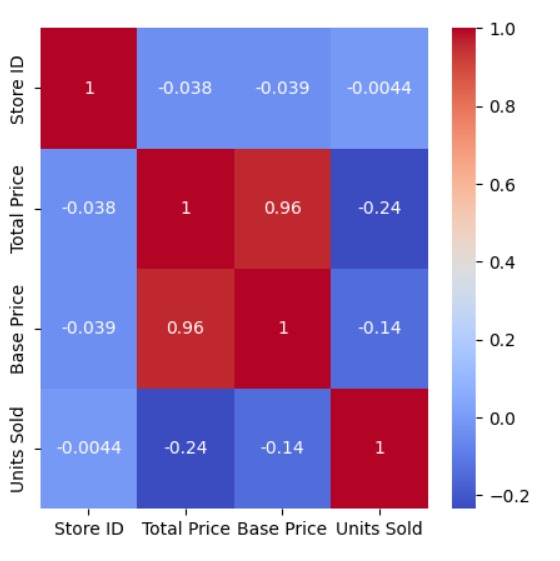
import pickle

* **Importing the dataset (read dataset, create matrix)**

data = pd.read\_csv("/content/PoductDemand.csv.zip")

for ind in data.index:

    continue



* **Handling the missing data (sklearn.preprocessing , libraries contains class called , imputer )**

def remove\_space\_between\_word(dataset):

    for col in dataset.columns:

        for i in range(len(dataset[col])):

            if (type(dataset[col][i]) == str ):

                dataset[col][i] = dataset[col][i].strip()

                dataset[col][i] = dataset[col][i].replace(" ", "\_")

    return data

new\_df = remove\_space\_between\_word(data)

new\_df.head()

fig = plt.figure(figsize=(5, 5))

ax = fig.gca()

data["ID"].hist(ax =ax)

plt.ylabel("number of apparution in data")

plt.xlabel("UNITS SOLD")

new\_df.hist()

* **Encoding categorical data**

One-Hot Encoding is another popular technique for treating categorical variables. It simply creates additional features based on the number of unique values in the categorical feature. Every unique value in the category will be added as a feature. One-Hot Encoding is the process of creating dummy variables.

def encode\_data(dataset , data\_dict\_weigth):

    cols = dataset.columns

    for columnName in cols:

        for i in range(len(dataset[columnName])):

            try:

            #print(data\_dict[data2[columnName][i]]["weight"])

                dataset[columnName][i] = data\_dict[dataset[columnName][i]]["weight"]

            except:

                pass

    dataset = dataset.fillna(0) # put empty cell to 0

    dataset = dataset.replace("Store ID" , 5)

    dataset = dataset.replace("Total Price" , 6)

    dataset = dataset.replace("Base price" , 6)

    return dataset

data\_dict = new\_df.set\_index('ID').T.to\_dict()

df = encode\_data(new\_df , data\_dict)

df.head()

new\_df\_data = df.drop('ID' , axis =1)

label = data["ID"]

import plotly.express as px

import seaborn as sns

correlations = new\_df\_data.corr(method='pearson')

plt.figure(figsize=(5, 5))

sns.heatmap(correlations, cmap="coolwarm", annot=True)

plt.show()

* **Split the dataset into test set and training set**

**TRAIN AND TEST DATASET:**

Creating train and test datasets is a fundamental step in building and evaluating machine learning models. Here's how you can create these datasets:

1. **Data Splitting** : Start with your complete dataset, which contains both the features (input variables) and the target variable (the variable you want to predict, like product demand).

2. **Randomization** : Shuffle the dataset randomly. This is important to ensure that there's no inherent order or bias in the data.

3. **Split Ratio** : Decide on a split ratio. A common choice is an 80/20 or 70/30 split, meaning you allocate 80% (or 70%) of the data to the training set and 20% (or 30%) to the test set. The training set is used to train the model, and the test set is used to evaluate its performance.

4. **Splitting** : Divide the shuffled dataset into the training and test sets based on the chosen ratio. Make sure both sets are representative of the overall data.

Where, X represents your features, y represents your target variable

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

In this code:

`X` is your feature data.

- `y` is your target variable.

- `test\_size` is set to 0.2, which means an 80/20 split.

- `random\_state` is used to ensure reproducibility. You can set it to any integer.

After splitting your data, you can use `X\_train` and `y\_train` to train your machine learning model, and `X\_test` and `y\_test` to evaluate its performance. Remember to keep your test set separate and only use it for evaluation to assess how well your model will perform on unseen data.

from sklearn.svm import SVC

from sklearn.tree import DecisionTreeClassifier

X\_train, X\_test, y\_train, y\_test = train\_test\_split(new\_df\_data, label, shuffle=True, train\_size = 0.70 )

def fit\_eval\_model(model, train\_features, y\_train, test\_features, y\_test):

    """

    Function: train and evaluate a machine learning classifier.

    Args:

      model: machine learning classifier

      train\_features: train data extracted features

      y\_train: train data lables

      test\_features: train data extracted features

      y\_test: train data lables

    Return:

      results(dictionary): a dictionary of classification report

    """

    results = {}

    # Train the model

    model.fit(train\_features, y\_train)

    # Test the model

    train\_predicted = model.predict(train\_features)

    test\_predicted = model.predict(test\_features)

     # Classification report and Confusion Matrix

    results['classification\_report'] = classification\_report(y\_test, test\_predicted)

    results['confusion\_matrix'] = confusion\_matrix(y\_test, test\_predicted)

    return results

rf = RandomForestClassifier(random\_state = 1)

ab = AdaBoostClassifier(random\_state = 1)

gb = GradientBoostingClassifier(random\_state = 1)

# Fit and evaluate models

results = {}

for cls in [ rf, ab, gb]:

    cls\_name = cls.\_\_class\_\_.\_\_name\_\_

    results[cls\_name] = {}

    results[cls\_name] = fit\_eval\_model(cls, X\_train, y\_train, X\_test, y\_test)

for result in results:

    print (result)

    print()

    for i in results[result]:

        print (i, ':')

        print(results[result][i])

        print()

    print ('-----')

    print()

* **Standard Scaler**

One of the most commonly used feature scaling techniques is Standard Scaler. It scales the data such that the mean is 0 and the standard deviation is 1. This means that the scaled data will have a normal distribution with a mean of 0 and a standard deviation of 1.