**Part2:** Programming questions

1. Reads data into pandas dataframe

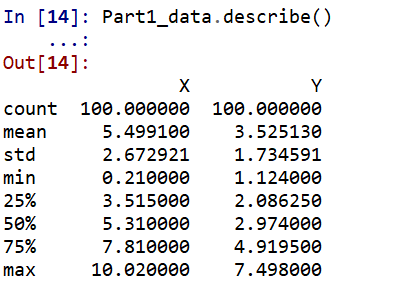
Part1\_data = pd.read\_csv('E:\TTIChallenge\Programming\_Part1\_TTI\_Challenge.csv', sep= '\t')



1. Compute min, max, average, standard deviation, and the geometric mean for X, Y.

Code and results for finding the min, max, average, std-standard deviation.

Code: Part1\_data.describe()



Code and results to find the geometric mean.

**Code:**

def geo\_mean(X):

'''x: 1darray(numeric)

Returns: Gemotric mean of given elements'''

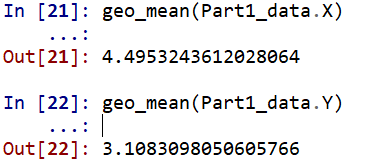
a = np.array(X)

return a.prod()\*\*(1.0/len(a))

geo\_mean(Part1\_data.X)

geo\_mean(Part1\_data.Y)

Results:



1. How would you write out the dataframe to a SQL table?

Assuming that ‘PartNo’ table is not present in the database TTI. I have used create table in sql-alchemy to create schema first and then used dataframe.to\_sql function to export data from our dataframe to ‘PartNO’ table inorder to retain the datatypes in our dataframe.

#Code

import pyodbc

from sqlalchemy import \*

#from sql alchemy import create\_engine, Table, Integer, Sequence, Column, MetaData

engine= create\_engine('mssql+pyodbc://user\_name:password@host:port/database?driver=SQL+Server')

conn= engine.connect()

#creating ‘PartNo’ database table

metadata= MetaData()

PartNo =Table('PartNo', metadata,

Column('Class', String(20), nullable=False),

Column('Part\_No', String(20), primary\_key=True),

Column('X', Integer, nullable=False),

Column('Y', Integer, nullable=False)

)

PartNo.create(engine, checkfirst=True)

#Exporting data from ‘Part1\_data’ dataframe to ‘PartNo’ table in MSsql

Part1\_data.to\_sql(name='[servername].[dbname].[PartNo]',con=engine, if\_exists='append')

1. Generates a predictive model of Y based on X (submit code and results); describe your approach.

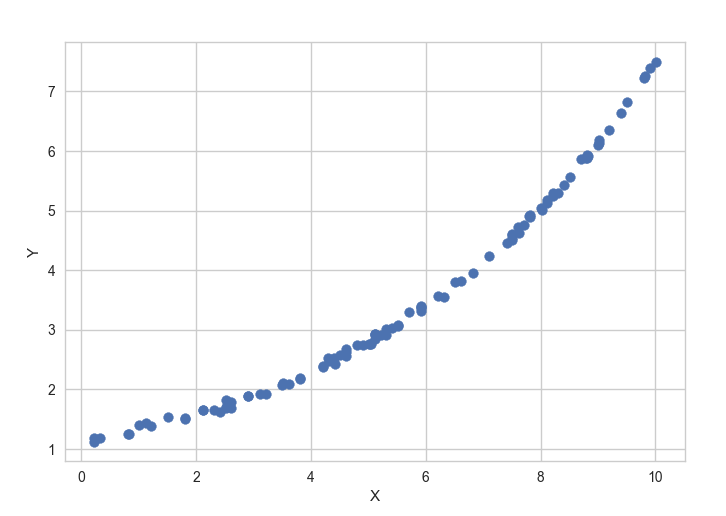
#Scatter plot of Y vs X

plt.scatter(Part1\_data.X, Part1\_data.Y )

plt.xlabel('X')

plt.ylabel('Y')

plt.show()



As Y vs X scatter plot shows that y is a function of power of x (checking which power sets best 2, 3,4, 5 or more).

# Plotting a linear regression of order 1 between 'X' and 'Y'

sns.regplot(x='X', y='Y', data=Part1\_data, scatter=None, color='grey', label='order 1')

# Plotting in green a linear regression of order 2,3,4,5,6 between 'X' and 'Y'

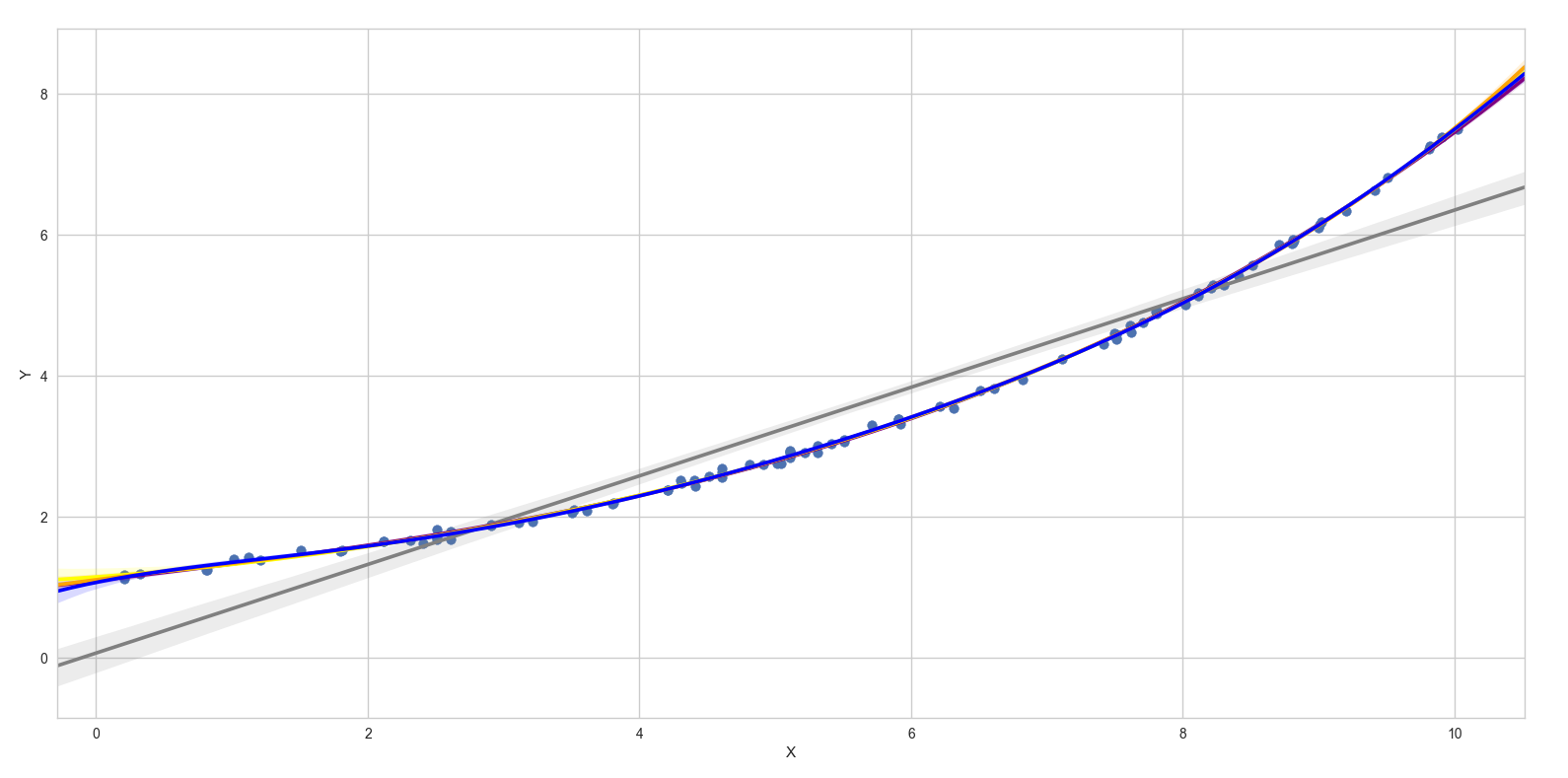
#sns.regplot(x='X', y='Y', data=Part1\_data, scatter=None, order=2, color='green', label='order 2')

sns.regplot(x='X', y='Y', data=Part1\_data, scatter=None, order=3, color='purple', label='order 3')

sns.regplot(x='X', y='Y', data=Part1\_data, scatter=None, order=4, color='yellow', label='order 4')

sns.regplot(x='X', y='Y', data=Part1\_data, scatter=None, order=5, color='orange', label='order 5')

sns.regplot(x='X', y='Y', data=Part1\_data, scatter=None, order=6, color='blue', label='order 6')



Plotting linear regression plot of various orders of X with Y. But as we see the regression graphs for various orders of X, they are all seem to fit the data. The Linear regression of 1st order of X is not definitely a good fit from the above graph.

Fitting a Linear regression for higher orders of X. Code for predicting Y using X is given below.

#Fitting Polynomial Regression to the dataset

X= Part1\_data.iloc[:,2:3].values

y= Part1\_data.iloc[:,3].values

K = 9

#generating polynomial features for X orders ranging from 2 to 10 before splitting the data

from sklearn.preprocessing import PolynomialFeatures

X\_poly = {}

for i in range(2,K):

poly\_reg = PolynomialFeatures(i)

X\_poly[i] = poly\_reg.fit\_transform(X)

# Splitting the dataset into the Training set and Test set for each dataset

X\_train = {}

X\_test = {}

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

for i in range(2,K):

X\_temp\_arr = X\_poly[i]

X\_train[i], X\_test[i], y\_train, y\_test= train\_test\_split(X\_temp\_arr, y, test\_size = 0.25, random\_state = 0)

# Fitting Linear regression for each order of X from 2 to 8 to the training sets

lin\_reg = {}

from sklearn.linear\_model import LinearRegression

for i in range(2,K):

lin\_reg[i] =LinearRegression()

lin\_reg[i].fit(X\_train[i], y\_train)

# Predicting the Test set results and finding the mse and rmse for each ordered polynomial

from sklearn.metrics import r2\_score

from sklearn.metrics import mean\_squared\_error

y\_pred = {}

error = {}

r2\_coef={}

mse={}

rmse={}

for i in range(2,K):

y\_pred[i] = lin\_reg[i].predict(X\_test[i])

error[i] = y\_pred[i]-y\_test

r2\_coef[i]=r2\_score(y\_test, y\_pred[i])

mse[i]= mean\_squared\_error(y\_test, y\_pred[i])

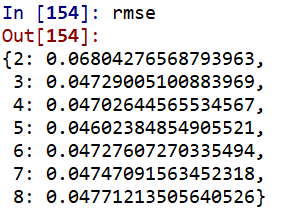
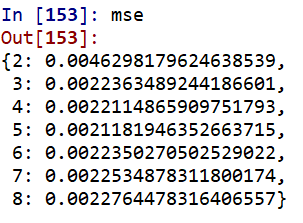
rmse[i]=math.sqrt(mse[i])

print(lin\_reg[3].coef\_)

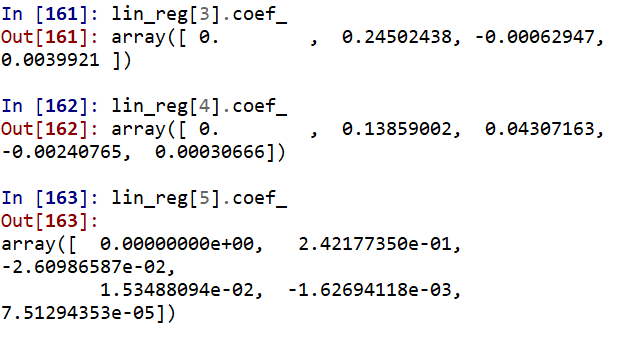
print(lin\_reg[4].coef\_)

print(lin\_reg[5].coef\_)

The ‘mean squared error’ and ‘Root mean squared error’ values for various orders of Polynomial features of x are shown below:



From the above results, it is evident that the polynomial regression of order 5 is performing best compared to all other models. But, the mean squared error values for all the models are differ by e-4, which is very minute. We can choose polynomial regression of order 3 if we want the simplest model with less complexity. As shown below, the coefficients of polynomial regression of order 5 is given by lin\_reg[5].coef\_ and of order 3 is given by lin\_reg[3].coef\_.



To get precise and accurate results, let’s check the accuracy consistency with the k fold cross validation to decide among order 3, 4 or 5.

1. Assess the accuracy consistency of your predictive model

Using K Fold cross validation to access the accuracy of the model.

# Applying k-Fold Cross Validation

from sklearn.model\_selection import cross\_val\_score

accuracies = {}

for i in range(2,K):

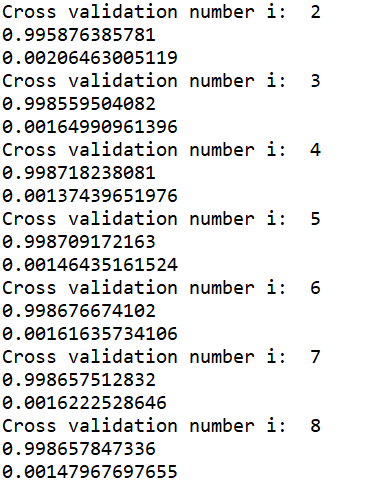
accuracies[i] = cross\_val\_score(estimator = lin\_reg[i], X = X\_train[i], y = y\_train, cv = 10)

print("Cross validation number i: ", i)

print (accuracies[i].mean())

print (accuracies[i].std())

Results:



Used the Kfold cross validation to test the accuracy of the results by comparing the mean accuracy and std deviation of various polynomial regressions. From, the above results the polynomial regression of order 4 has the highest accuracy and least standard deviation. As the differences are very minute which are of order 10 power -4, we can use any polynomial of order 3, 4, 5 interchangebly. However, polynomial regression of order 3 is the simplest model and order 4 is the accurate model.

**Part-3**

1. Display Full\_MFG\_Name in Table B without the MFG Code ( Example: ‘Amphenol’)

select b.\*, SUBSTRING(Full\_MFG\_Name,CHARINDEX('|',Full\_MFG\_Name)+2,LEN(Full\_MFG\_Name)) as Full\_MFG\_Name

from TableB b left join TableA a

on b.MFG\_Code=a.MFG\_code;

2. Calculate Total Revenue from Table B

select sum(Quantity\*Unit\_Price) as Total\_Revenue

from dbo.TableB;

3. Display the top 10 Products from Table B which made highest profit

select Top 10 Product, sum(Quantity\*(Unit\_Price-Unit\_Cost)) as profit

from dbo.tableB

group by Product

order by Profit Desc;

4. Display total cost, total Price and Margins grouped by Parent\_MFG in table A

select k.Parent\_MFG, sum(Quantity\*Unit\_Cost) as Total\_cost,

sum(Quantity\*Unit\_Price) as Total\_Price,

sum((Unit\_Price-Unit\_cost)/Unit\_Price) as Margin

from (select a.Parent\_MFG, b.\* from TableA a Left Join TableB b on a.MFG\_code=B.MFG\_code) k

group by Parent\_MFG;

5. Display the highest selling product and the second highest selling product

Select Top 2 Product, sum(Quantity\*(Unit\_Price-Unit\_Cost)) as Total\_Revenue

from tableB

group by product

order by sum(Quantity\*(Unit\_Price-Unit\_Cost)) desc;

6. Display the Total Cost and Total Revenue based on Type from Table C and order it in a

descending order.

select k.Type, sum(Quantity\*Unit\_Cost) as Total\_cost,

sum(Quantity\*Unit\_Price) as Total\_Revenue

from (select v.Type, b.\* from

(select distinct Type, Product from TableC) v

left join TableB b on v.Product=b.Product) k

group by k.type

order by k.Type Desc;

7. Find which Quarter sold highest number of products

Quarter 2 has the highest number of products sold

select top 1 v.Quarter, sum(Quantity) as NoofUnits

from(SELECT b.\*, YEAR([Date]) AS [Year],

CASE

WHEN MONTH([Date]) BETWEEN 1 AND 3 THEN 'Q1'

WHEN MONTH([Date]) BETWEEN 4 AND 6 THEN 'Q2'

WHEN MONTH([Date]) BETWEEN 7 AND 9 THEN 'Q3'

WHEN MONTH([Date]) BETWEEN 10 AND 12 THEN 'Q4'

END AS [Quarter]

from tableB b) v

group by Quarter

order by sum(Quantity) desc;

8. Find which quarter made the highest sale in ‘AUTOMOTIVE’ category In the last year

a. Quarter3 has highest no of ‘Automative’ units sold in the last year with 1000 units.

select top 1 v.Quarter, sum(Quantity) as [TotalNoofUnits]

from

(select c. Category, b.\*,

CASE

WHEN MONTH([Date]) BETWEEN 1 AND 3 THEN 'Q1'

WHEN MONTH([Date]) BETWEEN 4 AND 6 THEN 'Q2'

WHEN MONTH([Date]) BETWEEN 7 AND 9 THEN 'Q3'

WHEN MONTH([Date]) BETWEEN 10 AND 12 THEN 'Q4'

END AS [Quarter]

from TableC c left join TableB b on c.Product = b.Product

where c.Category='Automotive' and year([date])=2017) v

group by v.Quarter

order by sum(Quantity) desc;

b. Quarter 4 has the highest total revenue generated by ‘Automative’ products which is 1843$.

select top 1 v.Quarter, sum(Quantity\*Unit\_Price) as [TotalRevenue]

from

(select c. Category, b.\*,

CASE

WHEN MONTH([Date]) BETWEEN 1 AND 3 THEN 'Q1'

WHEN MONTH([Date]) BETWEEN 4 AND 6 THEN 'Q2'

WHEN MONTH([Date]) BETWEEN 7 AND 9 THEN 'Q3'

WHEN MONTH([Date]) BETWEEN 10 AND 12 THEN 'Q4'

END AS [Quarter]

from TableC c left join TableB b on c.Product = b.Product

where c.Category='Automotive' and year([date])=2017) v

group by v.Quarter

order by sum(Quantity\*Unit\_Price) desc;

9. Find the Products in table C that haven’t sold anything ever

Product P

select product from TableC

except

select product from TableB;

**Part -4**

1. In Python (or Pandas) write a code to import the transaction table

data2= pd.read\_excel('E:\TTIChallenge\ModelingDataSet.xlsx', sheet\_name='Transactions')

data2.columns= data2.columns.str.replace(' ','')# Removing any spaces if present in colmn names.

1. In Python (or Pandas), write a code that will cluster the extended costs into bins (example: bin 1: extended cost between $1 and $100; bin2: extended cost between $100 and $350, etc.) with the minimum variance within each bin and the maximum average margin difference across bins. The number of bins, nb\_b, is a parameter of your model.

As we have to cluster the extended costs into bins based on minimum variance of margin and average margin difference across the bins, I am using k means clustering on the given transaction margins.

Before proceeding to clustering step, the number of clusters for the kmeans should be known. Hence, to find the optimal number of clusters is identified from the elbow diagram (WCSS namely within-cluster sums of squares**)** using K means++ initialization.

Code:

# Using the elbow method to find the optimal number of clusters

from sklearn.cluster import KMeans

wcss = []

for i in range(1, 11):

kmeans = KMeans(n\_clusters = i, init = 'k-means++', random\_state = 42)

kmeans.fit(X1)

wcss.append(kmeans.inertia\_)

plt.plot(range(1, 11), wcss)

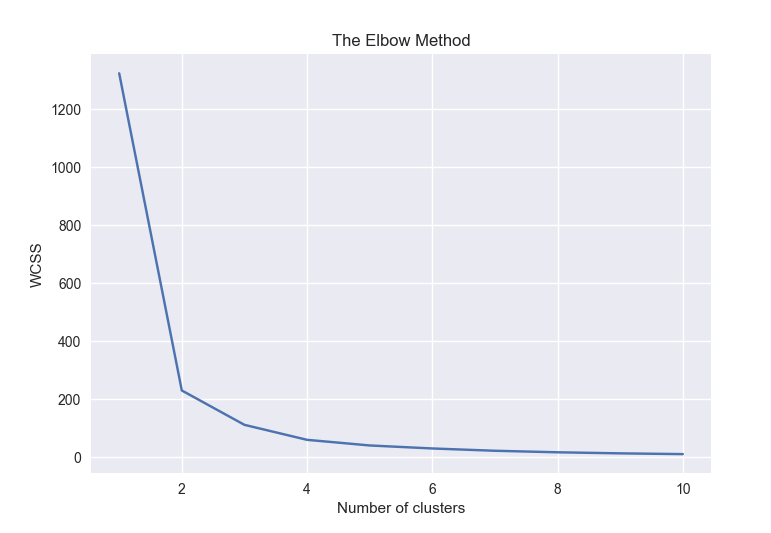
plt.title('The Elbow Method')

plt.xlabel('Number of clusters')

plt.ylabel('WCSS')

plt.show()

Below is the Elbow plot showing WCSS vs Number of Clusters.

 As the figure shows, 2 is the optimal best number of clusters with minimum WCSS and maximum average margin difference across different bins. But 2 clusters might not be ideal number in real world to cluster margins as they only represent high and low margin clusters. So, considering the next optimal number of clusters as 4 and clustering the Margin with 4 no of clusters is shown below.

# Fitting K-Means to the Margin of the dataset

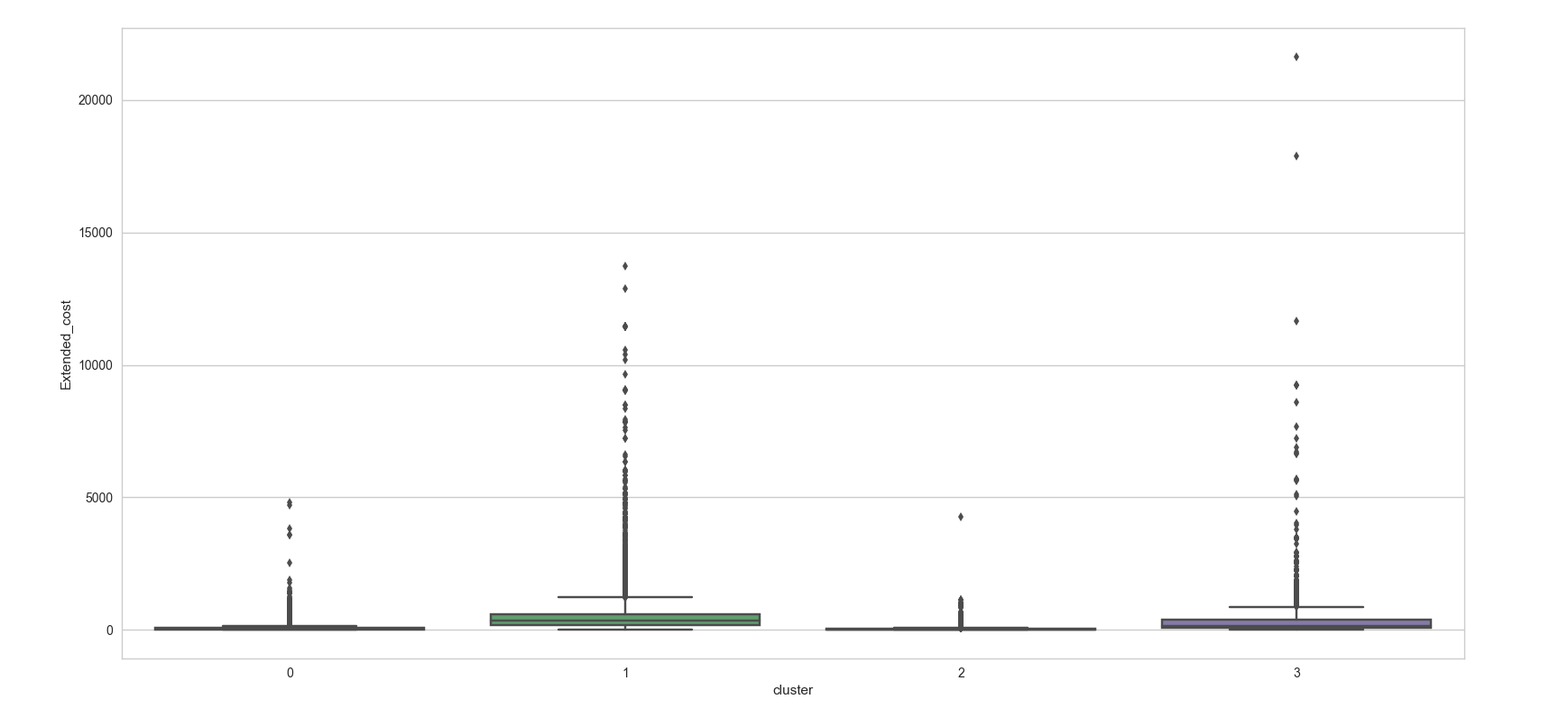
kmeans = KMeans(n\_clusters = 4, init = 'k-means++', random\_state = 42)

y\_kmeans = kmeans.fit\_predict(X1)

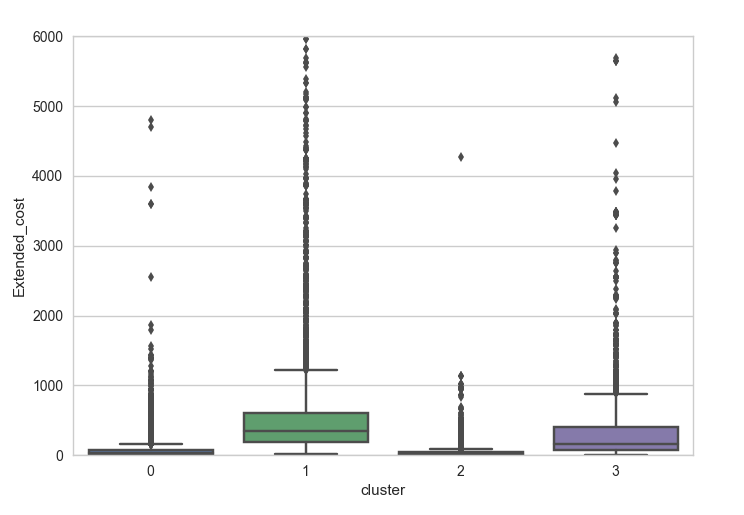
#Appending the clusters created back to the dataset

data2['cluster']= y\_kmeans

#Box plot for the extended cost across newly formed clusters



As few of the extended costs are beyond 20,000 the default scale does not show the clear distribution of extended costs. Let’s view the distribution of extended costs below 6000$ in detail.



From the figure it is clear that all the clusters have records with extended cost lying between 1 to 1000 and cluster 1 and cluster 3 have major overlap above 1000$ of extended cost. So, only margin cannot differentiate the extended cost bins as every bin we take will have same values of extended cost belonging to more than 1 cluster.

1. Use your favorite classification model to segment the extended costs (Explain your code)
2. Solution showing the iteration of number of bins from 3 to 10 and displaying the average margin % and coefficient for each cluster based on no of bins clustered.

kmeans={}

for nb\_b in range(3,11):

kmeans[nb\_b] = KMeans(n\_clusters = nb\_b, init = 'k-means++', random\_state = 42)

data2['cluster{0}'.format(nb\_b)] = kmeans[nb\_b].fit\_predict(X1)

#Displaying the results of number of bins and clusters and cluster average margin % and coefficient of variation for each cluster.

for nb\_b in range(3,11):

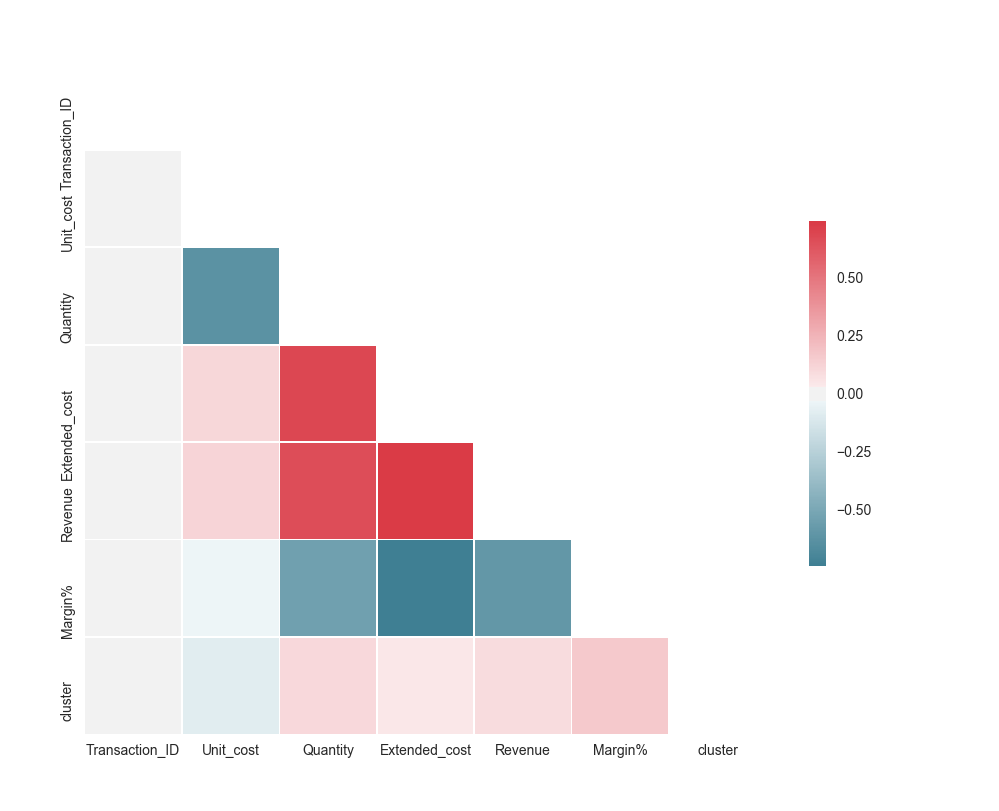
print("cluster "," with ",str(nb\_b),"bins")

print("clusternumber","average\_margin%","coefficent\_of\_variation\_margin%")

for i in range(nb\_b):

print('cluster{0}'.format(i),data2[data2['cluster{0}'.format(nb\_b)] == i]['Margin%'].mean(), (data2[data2['cluster{0}'.format(nb\_b)] == i]['Margin%'].std()/data2[data2['cluster{0}'.format(nb\_b)] == i]['Margin%'].mean()))

Let’s try to figure out which variables in the transaction table are classifying the clusters better.



The predictor variables extended cost and revenue are highly(positively) correlated and this is expected the no.of units price and cost are dependent on one other. Also we see that extended cost, quantity and extended\_cost, margin% are correlated to higher level. As of these correlations makes sense as Extended\_cost= Quantity\*Unit\_cost.

Margin= (Revenue-Extended\_cost)/Revenue\*100.