**Part2: Programming questions**

1. Reads data into pandas dataframe

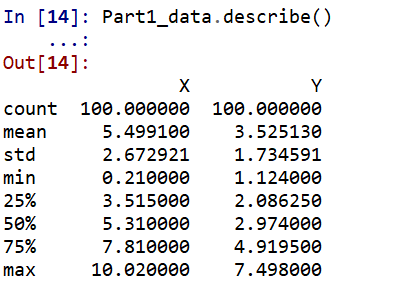
Part1\_data = pd.read\_csv('E:\TTIChallenge\Programming\_Part1\_TTI\_Challenge.csv', sep= '\t')



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'''

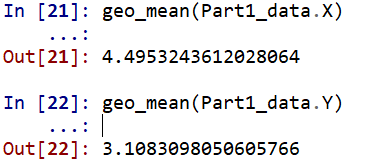
a = np.array(X)

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 a table schema using sql-alchemy connection in Ms Sql 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.

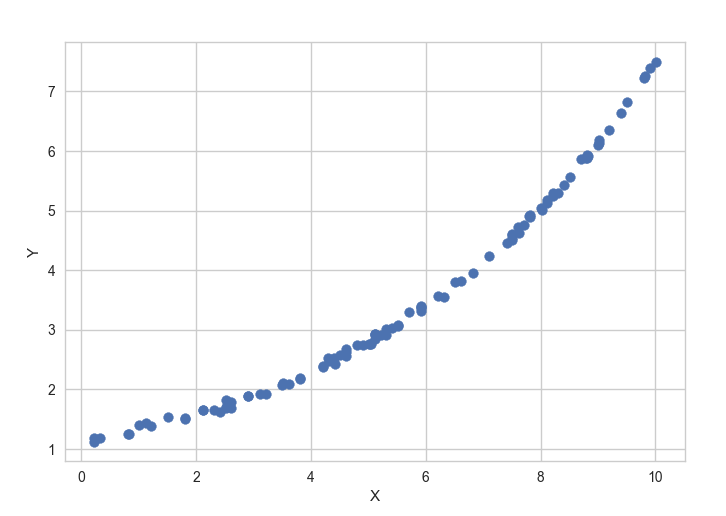
#Scatter plot of Y vs X

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'

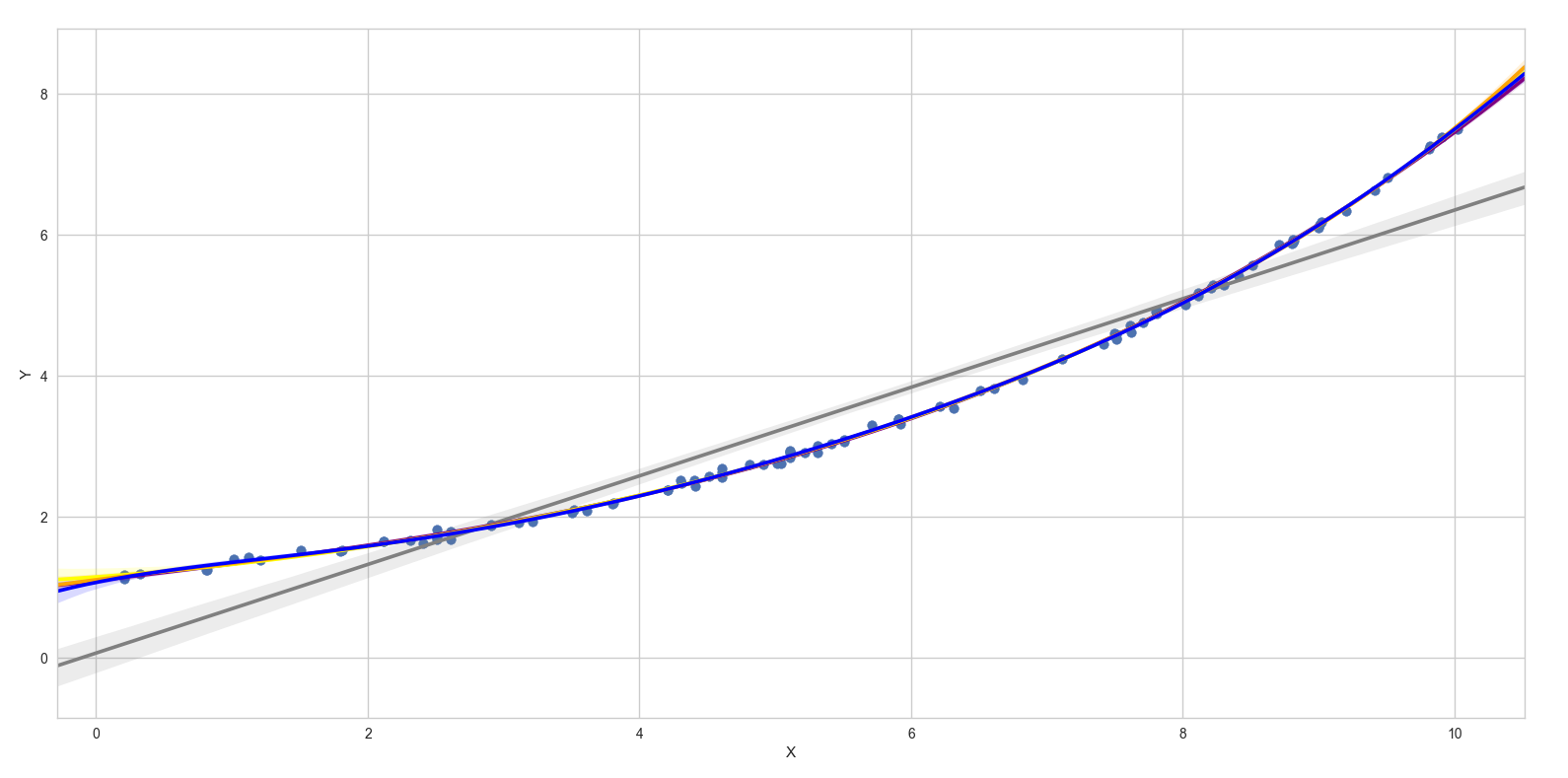
#sns.regplot(x='X', y='Y', data=Part1\_data, scatter=None, order=2, color='green', label='order 2')

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])

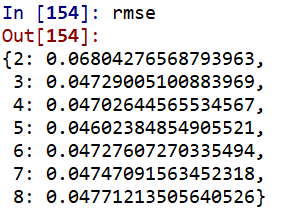
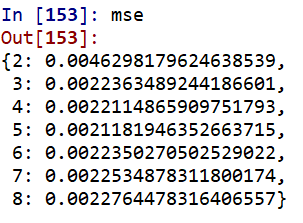
rmse[i]=math.sqrt(mse[i])

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

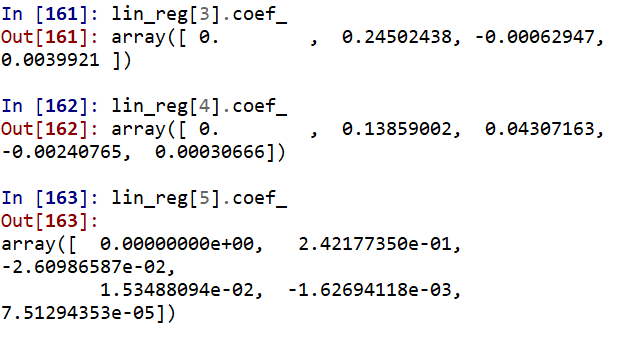
print(lin\_reg[4].coef\_)

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):

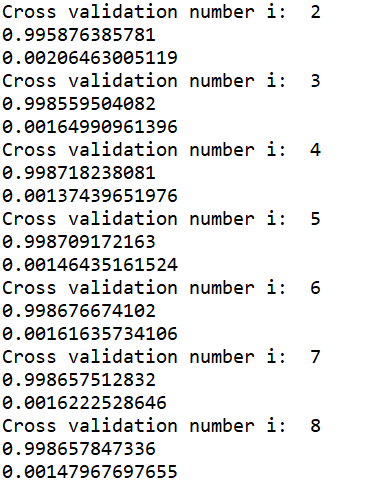
accuracies[i] = cross\_val\_score(estimator = lin\_reg[i], X = X\_train[i], y = y\_train, cv = 10)

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)

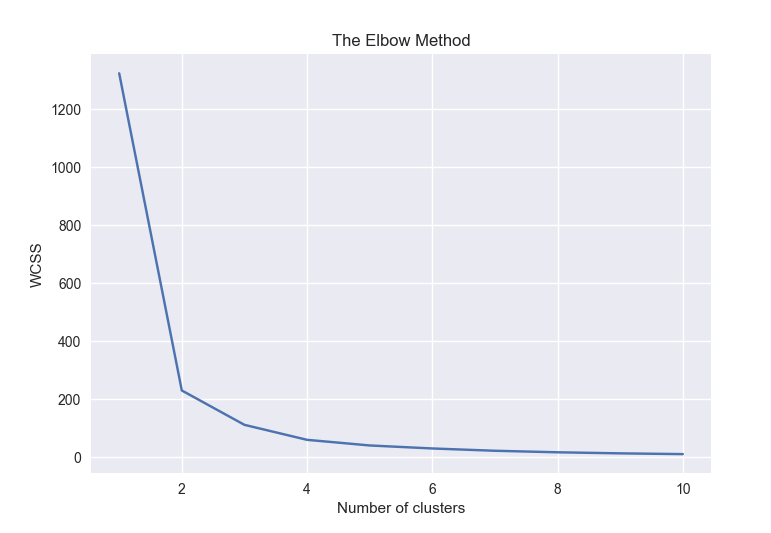
plt.title('The Elbow Method')

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, 4 is the optimal best number of clusters with minimum WCSS and maximum average margin difference across different bins. Beyond 4 clusters though the WCSS is decreasing the decrease is not leading to much difference between the average margin across difference.

#Fitting K-Means to the Margin of the dataset and appending formed clusternumbers(0 to 3) to dataset

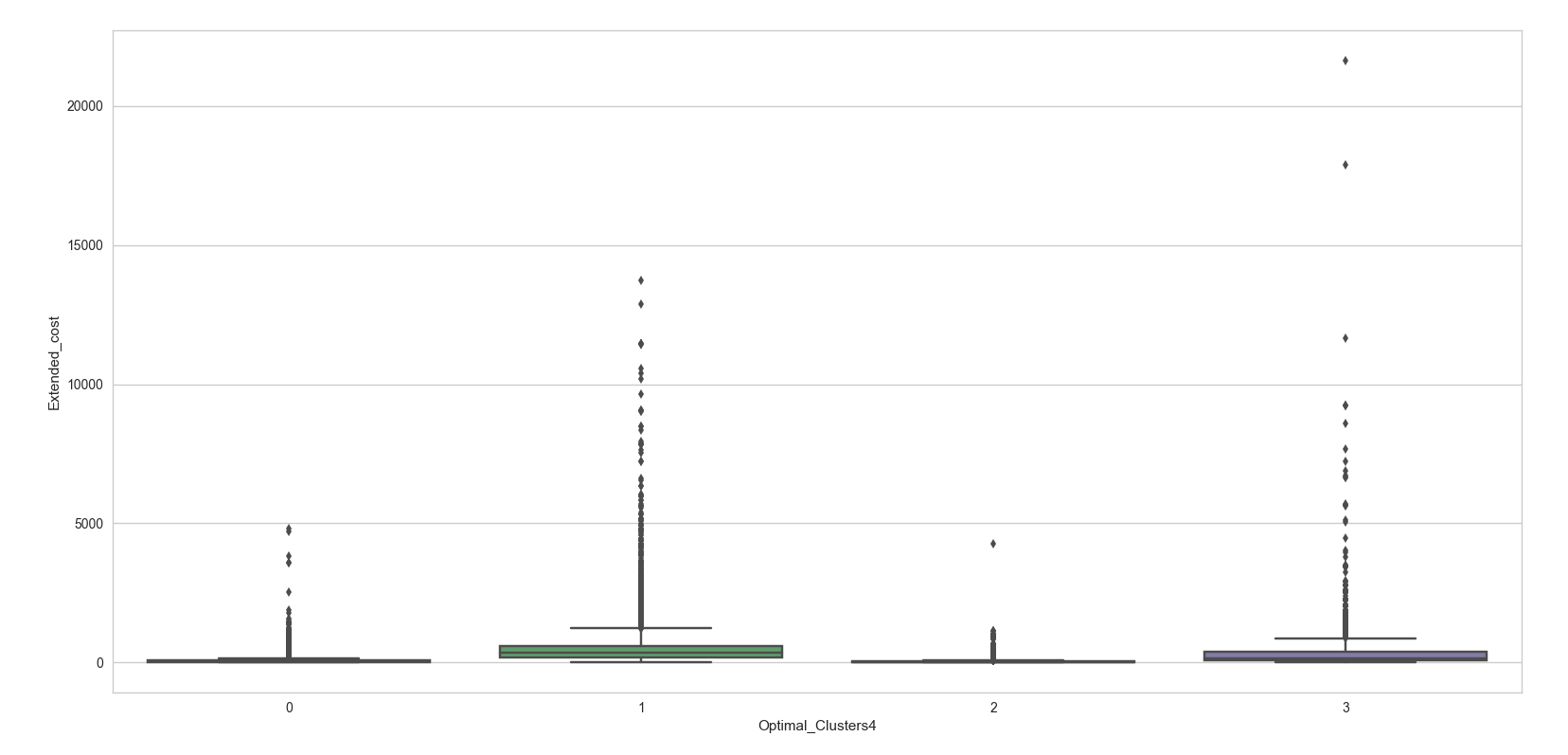
kmeans\_4 = KMeans(n\_clusters = 4, init = 'k-means++', random\_state = 42)

y\_kmeans\_4= kmeans\_4.fit\_predict(X1)

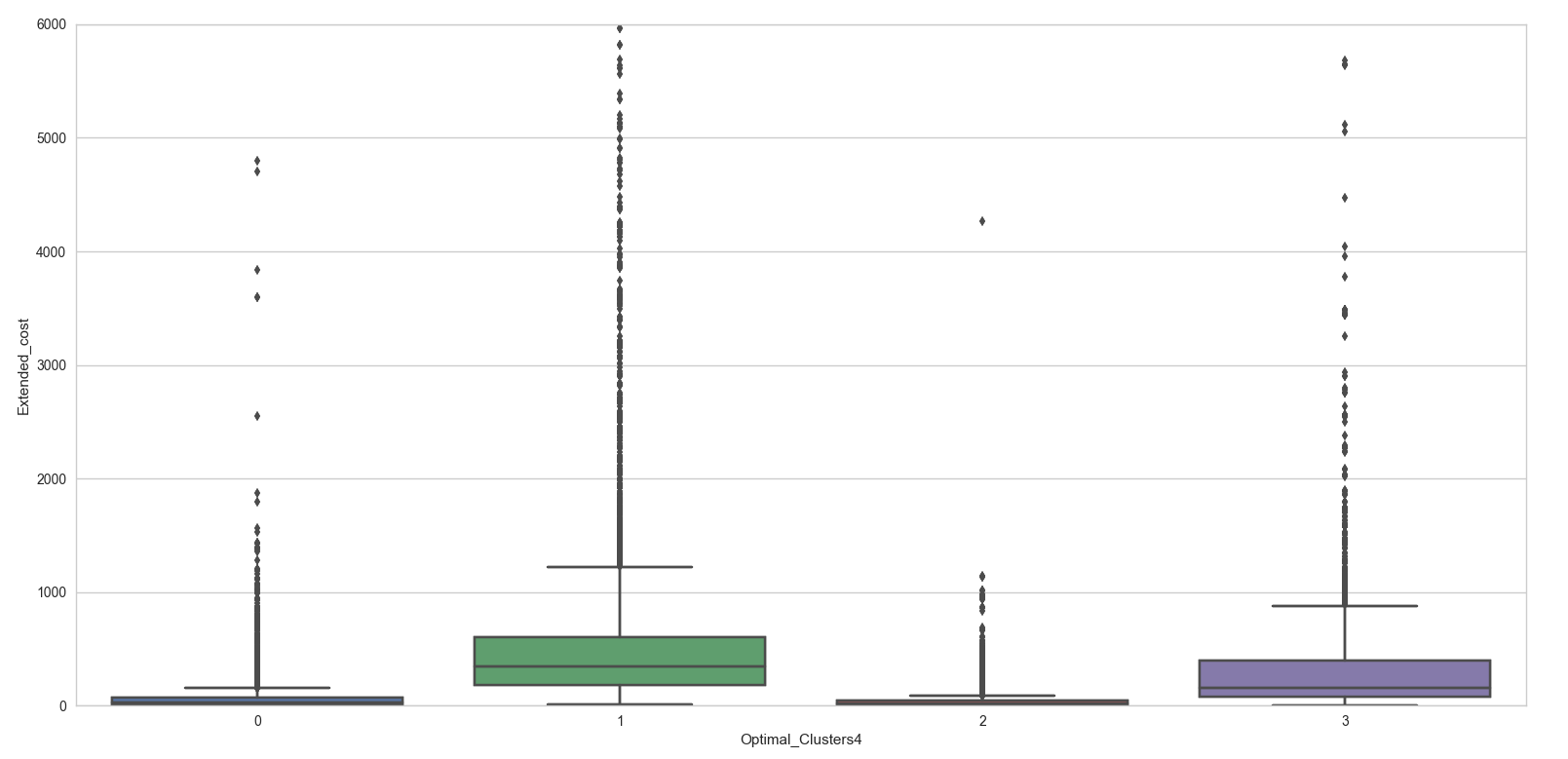
#Assigning the clusters created to the dataset

data2['Optimal\_Clusters6'] = y\_kmeans\_4

#Box plot for the extended cost across newly formed clusters



As very few of the extended costs are beyond 10,000 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. cluster 0, Cluster2 have major overlap below 100$ of extended cost. Also, cluster 1 and cluster 3 have values in similar range. 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 which is as expected because margin is not only dependent on extended cost but also on revenue.

1. Use your favorite classification model to segment the extended costs (Explain your code)

Using Random Forest classification to segment the extended cost.

#Random Forest classification

X = data2.iloc[:, 3:4].values

y = data2.iloc[:, 6].values

# Feature Scaling

from sklearn.preprocessing import StandardScaler

sc\_X = StandardScaler()

X = sc\_X.fit\_transform(X)

# Splitting the dataset into the Training set and Test set

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, random\_state = 0)

# Fitting Random Forest Classification to the Training set

from sklearn.ensemble import RandomForestClassifier

classifier\_RF = RandomForestClassifier(n\_estimators = 100, criterion = 'entropy', random\_state = 0)

classifier\_RF.fit(X\_train, y\_train)

# Predicting the Test set results

y\_pred\_RF = classifier\_RF.predict(X\_test)

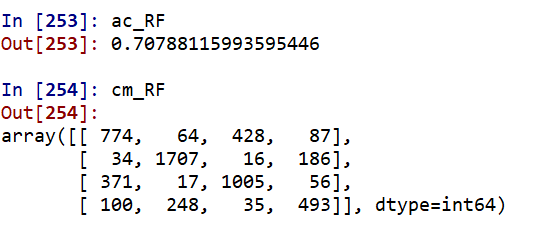
# Making the Confusion Matrix

from sklearn.metrics import confusion\_matrix

from sklearn.metrics import accuracy\_score

cm\_RF = confusion\_matrix(y\_test, y\_pred\_RF)

ac\_RF = accuracy\_score(y\_test, y\_pred\_RF)

The accuracy for the Random Forest classifier is 70.79%. The accuracy of the classifier using only extended\_cost is low because it Is not the only factor leading to margin%.

1. 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.

# Fitting K-Means to the Margin of the dataset and assigning the appending formed clusternumbers to dataset

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.

avg\_margin={}

coef\_var={}

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

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

print("clusternumber","average\_margin%","coef\_of\_var")

avg\_margin\_temp = [data2[data2['cluster{0}'.format(nb\_b)] == value]['Margin%'].mean() for value in range(nb\_b)]

coef\_var\_temp = [data2[data2['cluster{0}'.format(nb\_b)] == value]['Margin%'].std()\*100/avg\_margin\_temp[value] for value in range(nb\_b)]

for i in range(nb\_b):

print("cluster", str(i), avg\_margin\_temp[i], coef\_var\_temp[i])

avg\_margin[nb\_b] = avg\_margin\_temp

coef\_var[nb\_b] = coef\_var\_temp

cluster with 3 bins

clusternumber average\_margin% coef\_of\_var

cluster 0 0.6412912085901062 10.147599085324389

cluster 1 0.16893685069699316 32.9966611193018

cluster 2 0.4381221373092914 16.180915743932577

cluster with 4 bins

clusternumber average\_margin% coef\_of\_var

cluster 0 0.5345188526875511 8.142779094981245

cluster 1 0.15381231443245033 26.692612976172143

cluster 2 0.6761967698843874 7.319754230067148

cluster 3 0.3417330470056515 16.043460355780958

cluster with 5 bins

clusternumber average\_margin% coef\_of\_var

cluster 0 0.547744583636649 6.541697645059955

cluster 1 0.13706625324250066 18.519398340147415

cluster 2 0.6784103976406295 7.184682022063742

cluster 3 0.402143480622434 10.234082470325125

cluster 4 0.2531720371639419 14.96641513882245

cluster with 6 bins

clusternumber average\_margin% coef\_of\_var

cluster 0 0.6547121470633105 4.535983979244181

cluster 1 0.1370435577981661 18.50726139420284

cluster 2 0.39518796432964565 9.916075055629472

cluster 3 0.25151887291329983 14.614669839475031

cluster 4 0.5374218932574568 6.281079781209655

cluster 5 0.7634071379011278 6.773208058802564

cluster with 7 bins

clusternumber average\_margin% coef\_of\_var

cluster 0 0.23689441220758173 14.247579831788684

cluster 1 0.49524203983716464 5.90920413473205

cluster 2 0.6698535064393272 3.481814831166049

cluster 3 0.13376160069589535 16.685783634579593

cluster 4 0.3697352021490958 9.650665346380675

cluster 5 0.5784241004494357 4.316585637349223

cluster 6 0.7815689352869352 6.543991266122893

cluster with 8 bins

clusternumber average\_margin% coef\_of\_var

cluster 0 0.5786816924681308 4.189062632058937

cluster 1 0.13126427126876747 15.450572284111868

cluster 2 0.396403432181973 6.5572313137026494

cluster 3 0.6677163275190227 3.283657439529175

cluster 4 0.2983692316602306 8.567950316426478

cluster 5 0.501418176543595 5.092328164575703

cluster 6 0.7712773826787507 6.6830364726949885

cluster 7 0.21233929391777098 10.999806514302431

cluster with 9 bins

clusternumber average\_margin% coef\_of\_var

cluster 0 0.2984256270772227 8.454414027302457

cluster 1 0.6147825871957303 2.9748483361025695

cluster 2 0.1316372769836649 15.621219026525758

cluster 3 0.47664987536092496 4.614077421985161

cluster 4 0.6800649729316749 2.8309981995233993

cluster 5 0.5470938735513766 3.59169695071882

cluster 6 0.21341781767955798 10.778180268712264

cluster 7 0.7927313154761919 6.2971621796746735

cluster 8 0.3898082184512441 5.428391417551174

cluster with 10 bins

clusternumber average\_margin% coef\_of\_var

cluster 0 0.47958771048951154 4.617537919376113

cluster 1 0.12017943019105591 10.21532800102315

cluster 2 0.6819394653758547 2.7488740270408574

cluster 3 0.30390728987898663 7.403854333274005

cluster 4 0.5492366503981492 3.573788404616396

cluster 5 0.2283534641771606 8.485360929592696

cluster 6 0.39111844475787616 5.551617539837077

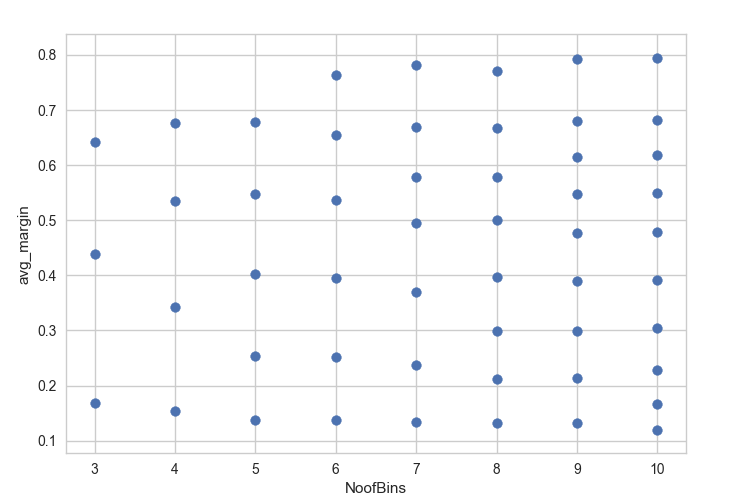
cluster 7 0.6184479792746093 3.0037262886588585

cluster 8 0.794783041666668 6.248538407696058

cluster 9 0.16570042042349628 8.932927868382139

The distribution of avg\_margin% across different clusters for different bins distribution is shown below:

Also, from the above results it is evident that as the noofbins increase each cluster is reducing its standard deviation. But, the difference between the avg margin across the clusters decreasing which leads to increase in misclassification rate. So, selecting value of number of bins to be 4 gives us clusters which can identify different margins with optimal average margin % difference across clusters.



c) What column in the Transaction table could help you to differentiate the margins even better?

Figuring out which variables other than Extended\_cost in the transaction table are classifying the margin clusters better. As, margin% is ‘(Revenue-Extended\_cost)/Revenue’ it is evident that revenue should be the other parameter in classifying the margin% by 100. Let’s not go with the assumption that Margin% can be written as a function of Revenue and Extended cost.

#Plotting scatter plot for numerical data

from pandas.plotting import scatter\_matrix

scatter\_matrix(data2)

#Finding if there is any correlation between the variables

correlations= data2.corr(method='spearman')

mask = np.zeros\_like(correlations, dtype=np.bool)

mask[np.triu\_indices\_from(mask)] = True

# Set up the matplotlib figure

f, ax = plt.subplots(figsize=(11, 9))

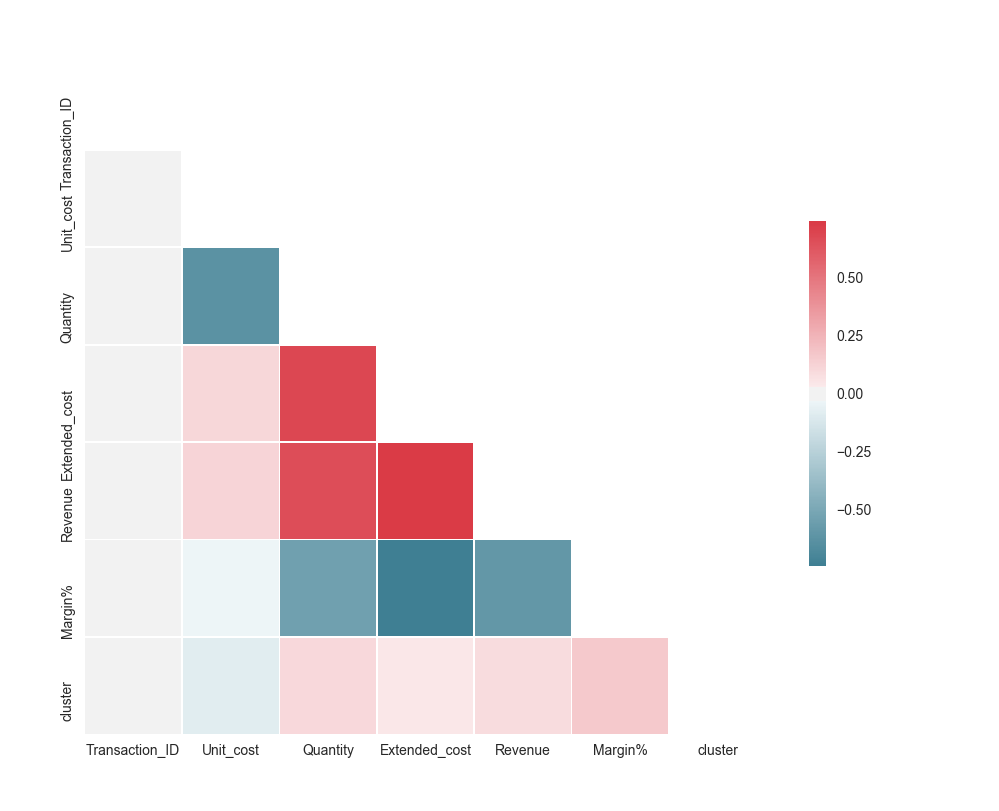
# Generate a custom diverging colormap

cmap = sns.diverging\_palette(220, 10, as\_cmap=True)

# Draw the heatmap with the mask and correct aspect ratio

sns.heatmap(correlations, mask=mask, cmap=cmap, vmax=.3, center=0,

square=True, linewidths=.5, cbar\_kws={"shrink": .5})



The predictor variables extended\_cost and revenue are highly(positively) correlated. Also we see that extended cost, quantity and extended\_cost, margin% are correlated and Unit\_cost and Quantity are negatively correlated. All these correlations makes sense as

Extended\_cost= Quantity\*Unit\_cost.

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

Considering step wise selection method, Random Forest classification using Extended\_cost and Revenue as predictor variables and 4 clusters based on Margin%.

#2.Data Preprocessing and Random Forest Regressions using Extended\_cost and Revenue

names1= list(data2.columns[3:5])

X\_Final1 = data2.iloc[:, 3:5]

y\_Final1 = data2.iloc[:, 6]

# Feature Scaling

from sklearn.preprocessing import StandardScaler

sc\_X\_Final1 = StandardScaler()

X\_Final1 = sc\_X\_Final1.fit\_transform(X\_Final1)

# Splitting the dataset into the Training set and Test set

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

X\_train11, X\_test11, y\_train11, y\_test11 = train\_test\_split(X\_Final1, y\_Final1, test\_size = 0.2, random\_state = 0)

# Fitting Random Forest Classification to the Training set

from sklearn.ensemble import RandomForestClassifier

classifier\_RF\_11 = RandomForestClassifier(n\_estimators = 100, criterion = 'entropy', random\_state = 0)

classifier\_RF\_11.fit(X\_train11, y\_train11)

# Predicting the Test set results

y\_pred\_RF11 = classifier\_RF\_11.predict(X\_test11)

# Making the Confusion Matrix

from sklearn.metrics import confusion\_matrix

from sklearn.metrics import accuracy\_score

cm\_RF11 = confusion\_matrix(y\_test11, y\_pred\_RF11)

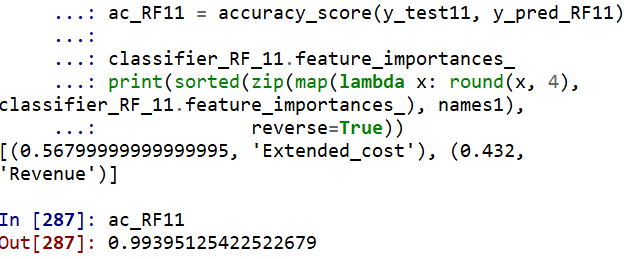
ac\_RF11 = accuracy\_score(y\_test11, y\_pred\_RF11)

classifier\_RF\_11.feature\_importances\_

print(sorted(zip(map(lambda x: round(x, 4), classifier\_RF\_11.feature\_importances\_), names1),

reverse=True))

Accuracy of the model: 99.39%

The feature importance of variables is given below: [(0.56799999999999995, 'Extended\_cost'), (0.432, 'Revenue')]

It implies that 56.80% of the Margin segmentation is explained by Extended cost and the remaining 43.20% by revenue.

Also, here the percentage is almost 99.4%.

As, extended\_cost is the product of Quantity and Unit\_cost. The use of Quantity and Unit\_cost will result in high correlation.

**# Random Forest Regressions using Quantity, Extended\_cost and Revenue**

names2= list(data2.columns[2:5])

X\_Final2 = data2.iloc[:, 2:5]

y\_Final2 = data2.iloc[:, 6]

# Feature Scaling

from sklearn.preprocessing import StandardScaler

sc\_X\_Final2 = StandardScaler()

X\_Final2 = sc\_X\_Final2.fit\_transform(X\_Final2)

# Splitting the dataset into the Training set and Test set

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

X\_train12, X\_test12, y\_train12, y\_test12 = train\_test\_split(X\_Final2, y\_Final2, test\_size = 0.2, random\_state = 0)

# Fitting Random Forest Classification to the Training set

from sklearn.ensemble import RandomForestClassifier

classifier\_RF\_12 = RandomForestClassifier(n\_estimators = 100, criterion = 'entropy', random\_state = 0)

classifier\_RF\_12.fit(X\_train12, y\_train12)

# Predicting the Test set results

y\_pred\_RF12 = classifier\_RF\_12.predict(X\_test12)

# Making the Confusion Matrix

from sklearn.metrics import confusion\_matrix

from sklearn.metrics import accuracy\_score

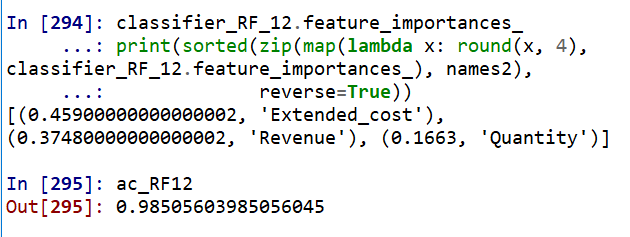
cm\_RF12 = confusion\_matrix(y\_test12, y\_pred\_RF12)

ac\_RF12 = accuracy\_score(y\_test12, y\_pred\_RF12)

classifier\_RF\_12.feature\_importances\_

print(sorted(zip(map(lambda x: round(x, 4), classifier\_RF\_12.feature\_importances\_), names2),

reverse=True))

The accuracy of the new classifier is 98.5%, which less than 99.4% of the previous classifier because of the correlation between the Quantity is correlated to both Revenue and extended cost.

**#Random Forest Regressions using Unit\_cost, Quantity, Extended\_cost and Revenue**

names3= list(data2.columns[1:5])

X\_Final3 = data2.iloc[:, 1:5]

y\_Final3 = data2.iloc[:, 6]

# Feature Scaling

from sklearn.preprocessing import StandardScaler

sc\_X\_Final3 = StandardScaler()

X\_Final3 = sc\_X\_Final2.fit\_transform(X\_Final3)

# Splitting the dataset into the Training set and Test set

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

X\_train13, X\_test13, y\_train13, y\_test13 = train\_test\_split(X\_Final3, y\_Final3, test\_size = 0.2, random\_state = 0)

# Fitting Random Forest Classification to the Training set

from sklearn.ensemble import RandomForestClassifier

classifier\_RF\_13 = RandomForestClassifier(n\_estimators = 100, criterion = 'entropy', random\_state = 0)

classifier\_RF\_13.fit(X\_train13, y\_train13)

# Predicting the Test set results

y\_pred\_RF13 = classifier\_RF\_13.predict(X\_test13)

# Making the Confusion Matrix

from sklearn.metrics import confusion\_matrix

from sklearn.metrics import accuracy\_score

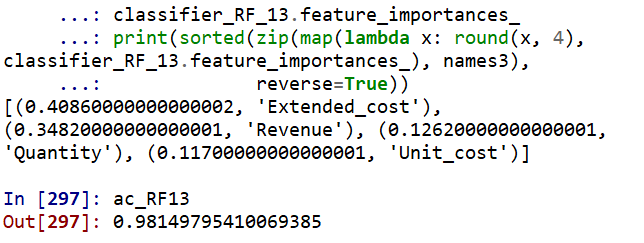
cm\_RF13 = confusion\_matrix(y\_test13, y\_pred\_RF13)

ac\_RF13 = accuracy\_score(y\_test13, y\_pred\_RF13)

classifier\_RF\_13.feature\_importances\_

print(sorted(zip(map(lambda x: round(x, 4), classifier\_RF\_13.feature\_importances\_), names3),

reverse=True))

The accuracy of the new classifier is still reduced because the Unit\_cost variable is also highly correlated to revenue and extended cost.

Therefore, the variables extended\_cost and Revenue are the best variables classifying the Margin% by 56.8 % and 43.2% respectively.