ASSIGNMENT 4 REPORT

Name: Sushant Kumbhar CWID:10453187 Email:skumbha2@stevens.edu

Result: Refer APPENDIX I for code

A 2D array is first constructed in x and 1 D array is conmstructed in y and a x_new is constructed as new 2D array and based on that, the coefficients are predicted.

```
m=[0,0,0,0,0], c=0, epochs=500, L=0.001
my m and c:  ([12.558376473022408, 13.07659427855137, 19.90471275194134, 15.621965523751422, 17.155649326070108], 12
1.80594595902986)
my prediction:  [[127.20604784242948, 128.7130530191922, 130.09447443122465, 124.31762125363434, 124.19203748890412],
[133.7056467525116, 131.7441576107289, 125.9904561281663, 126.25198801373732, 133.18258298136953], [125.7868885094181
2, 132.7535379725976, 134.74400924779172, 130.56401956988404, 138.12781041562175], [136.02193458564366, 131.804003894
23076, 136.17815424088116, 132.2726628599433, 123.83680147711755], [129.8691011422828, 125.06551933098318, 126.952640
75685088, 131.41310958162913, 130.3837706220649]]
```

Question 2: Refer APPENDIX II for code

This is the head of dataframe res_purchase

```
has_raised = await self.run_ast_nodes(code_ast.body, cell_name,
```

Out[7]:

	Year-Month	Agency Number	Agency Name	Cardholder Last Name	Cardholder First Initial	Description	Amount	Vendor	Transaction Date	Posted Date	Merchant Category Code (MCC)
0	201307	1000	OKLAHOMA STATE UNIVERSITY	Mason	C	GENERAL PURCHASE	890.00	NACAS	7/30/2013 0:00	7/31/2013 0:00	CHARITABLE AND SOCIAL SERVICE ORGANIZATIONS
1	201307	1000	OKLAHOMA STATE UNIVERSITY	Mason	C	ROOM CHARGES	368.96	SHERATON HOTEL	7/30/2013 0:00	7/31/2013 0:00	SHERATON
2	201307	1000	OKLAHOMA STATE UNIVERSITY	Massey	J	GENERAL PURCHASE	165.82	SEARS.COM 9300	7/29/2013 0:00	7/31/2013 0:00	DIRCT MARKETING/DIRCT MARKETERS--NOT ELSEWHERE...
3	201307	1000	OKLAHOMA STATE UNIVERSITY	Massey	T	GENERAL PURCHASE	96.39	WAL-MART #0137	7/30/2013 0:00	7/31/2013 0:00	GROCERY STORES,AND SUPERMARKETS
4	201307	1000	OKLAHOMA STATE UNIVERSITY	Mauro-Herrera	M	HAMMERMILL COPY PLUS COPY EA	125.96	STAPLES DIRECT	7/30/2013 0:00	7/31/2013 0:00	STATIONERY, OFFICE SUPPLIES, PRINTING AND WRIT...

This is the result of question 2 where I used . sum function to get the sum of columns. For ww_frainger column, the data type was object son I converted it to float and then calculated sum. Similar method is followed for grocery_store column.

[8]:

```
#Question 2.1
df["Amount"].sum()
```

t[8]: 188040606.2299999

[12]:

```
#Question 2.2
ww_grainger = df.loc[df['Vendor'].str.contains("WW GRAINGER", case=False)]
ww_grainger.Amount = ww_grainger.Amount.astype(float, copy = False)
ww_grainger["Amount"].sum()
```

:[12]: 5089340.5600000005

[10]:

```
#Question 2.3
wm_supercenter = df.loc[df['Vendor'].str.contains("WM SUPERCENTER", case = False)]
wm_supercenter.Amount = wm_supercenter.Amount.astype(float, copy = False)
wm_supercenter["Amount"].sum()
```

:[10]: 157457.46

Question 3: Refer APPENDIX III for code

Results

Figure below shows head of dataframe BalanceSheet.

[13]:

	Global Company Key	Data Date	Fiscal Year	Fiscal Quarter	Fiscal Year-end Month	Industry Format	Level of Consolidation - Company Interim Descriptor	Population Source	Data Format	Ticker Symbol	CUSIP	Company Name	ISO Currency Code	Calendar Data Year and Quarter	Fiscal Data Year and Quarter	Acc Ch Cur
0	1380	20100331	2010	1	12	INDL	C	D	STD	HES	42809H107	HESS CORP	USD	2010Q1	2010Q1	
1	1380	20100630	2010	2	12	INDL	C	D	STD	HES	42809H107	HESS CORP	USD	2010Q2	2010Q2	
2	1380	20100930	2010	3	12	INDL	C	D	STD	HES	42809H107	HESS CORP	USD	2010Q3	2010Q3	
3	1380	20101231	2010	4	12	INDL	C	D	STD	HES	42809H107	HESS CORP	USD	2010Q4	2010Q4	
4	1380	20110331	2011	1	12	INDL	C	D	STD	HES	42809H107	HESS CORP	USD	2011Q1	2011Q1	

To drop values having less than 70% NA values I used dropna method with threshold = 70% which means 70% of rows should be non NA. Only 175 columns are left.

[15]:

```
# Question 3.2
BalanceSheet_sixty = BalanceSheet.shape[0]*0.7 #Getting 70% values of the dataframe to keep as 30% NA values will l
BalanceSheet = BalanceSheet.dropna(thresh = BalanceSheet_sixty, axis = 1) #Using drop NA with tresh to remove 30%N
BalanceSheet.shape
```

[15]: (844, 175)

I calculated mean of 0 in the column and used drop method to drop those columns. After this operation 163 columns are left.

```
#Droping columns with more than 90% o values
BalanceSheet.drop(columns=BalanceSheet.columns[((BalanceSheet==0).mean()>0.9)],axis=1, inplace = True)
BalanceSheet.shape
```

]: (844, 163)

After that I used fillna method to fill the missing numerical data with the mean of the column

Then I used column indexing on find columns between “Accounting Changes - Cumulative Effect” to “Selling, General and Administrative Expenses columns” and found out 15:158 index were present to which I could apply lambda function.

Then I used corr command to find correlation between following columns.

	Current Assets - Other - Total	Current Assets - Total	Other Long-term Assets	Assets Netting & Other Adjustments
Current Assets - Other - Total	1.000000	0.790047	0.629802	0.042504
Current Assets - Total	0.790047	1.000000	0.665006	-0.072010
Other Long-term Assets	0.629802	0.665006	1.000000	-0.017979
Assets Netting & Other Adjustments	0.042504	-0.072010	-0.017979	1.000000

Then I used pd.merge function to merge two dataframes Rating and BalanceSheet on the columns 'Global Company Key', and 'Data Date' which changed the dataframe size and the columns now became 168. Stored t=his dfataframe into Matched

```
#using pd.merge to merge two dataframe on 'Data Date' column using inner merge
Matched = pd.merge(BalanceSheet, Ratings, how = 'inner', on = ['Global Company Key', 'Data Date']
Matched.shape
```

```
(822, 168)
```

Then I created dictionary with keys as "S&P Domestic Long Term Issuer Credit Rating", "Rate" column's values and values of the keys as rating and added it to new column "Rate".

```
Matched[["S&P Domestic Long Term Issuer Credit Rating", "Rate"]].
```

```
S&P Domestic Long Term Issuer Credit Rating  Rate
BBB                                          8       202
BB+                                          10      105
BBB-                                         9        96
BBB+                                         7        84
A                                            5        84
A-                                           6        61
BB                                           11       55
AAA                                          0        25
AA                                           2        24
A+                                           4        24
AA-                                          3        17
AA+                                          1         3
dtype: int64
```

Then I applied coefficients Linear Regression class created in Question 1 on 10 variables of the Matched dataframe and found coefficients of m and c. The following is the result I got.

```
print("coefficients of m and c: ",Linear_model.gradient_descent())
```

```
I use m=[0,0,0,0,0,0,0,0,0,0], c=0, epochs=500, L=0.001
coefficients of m and c:  ([0.5710729392318282, -0.06203801847632594, -0.09314771058985188, -0.01588549231470155, -0.08431471799470831, 0.5600276465088622, 0.513013301237527, 0.4002163956491606, 0.6580488217272298, -0.00735392891251687], 7.286377945451142)
```

CODES:

Importing all the required libraries:

```
# Importing the libraries
import numpy as np
from pandas import DataFrame
import pandas as pd
import warnings
pd.set_option('display.max_rows', None)
pd.set_option('display.max_columns', None)
```

APENDIX I:

```
class Linear_regression:

  def __init__(self, x, y, m, c, epochs, L):
    self.x,self.y,self.m,self.c,self.epochs,self.L = x,y,m,c,epochs,L #Initializing variables
x,y,m,c,epochs,L



  def gradient_descent(self) :
    for i in range(0,self.epochs):
      for i in range(0, len(self.m)):
        Dm = []
        Dc = []
        for xi, yi in zip(self.x,self.y):
          yi_pred = (xi[i] * self.m[i]) + self.c
          Dm.append((xi[i] * (yi_pred - yi)))
          Dc.append((yi_pred - yi))
        dm = sum(Dm) / len(Dm)
        dc = sum(Dc) / len(Dc)
        self.m[i] = self.m[i] - (self.L * dm)
        self.c = self.c - (self.L * dc)
    return self.m, self.c

  def predict(self,new_x):
    predicted_list = []
    for i in range(0, len(self.m)):
      a = []
      for p in new_x[i]:
        a.append((self.m[i] * p) + self.c)
      predicted_list.append(a)
    return predicted_list

if __name__ == "__main__":
```

```
#Constructing 2D array x
x = [[0.22, 0.46, 0.09, 0.27, 0.21],
    [0.32, 0.56, 0.89, 0.33, 0.77],
    [0.78, 0.99, 0.14, 0.88, 0.90],
    [0.07, 0.08, 0.09, 0.10, 0.11],
    [0.76, 0.22, 0.77, 0.93, 0.67],
    [0.49, 0.55, 0.61, 0.89, 0.40]]

# Constructing 1D array y
y = [55.85, 255.72, 110.37, 80.44, 150.55, 180.4, 131.62]

x_new = [[0.43, 0.55, 0.66, 0.20, 0.19],
        [0.91, 0.76, 0.32, 0.34, 0.87],
        [0.20, 0.55, 0.65, 0.44, 0.82],
        [0.91, 0.64, 0.92, 0.67, 0.13],
        [0.47, 0.19, 0.30, 0.56, 0.50],
        [0.21, 0.88, 0.75, 0.44, 0.53]]


Linear_model = Linear_regression(x,y,[0,0,0,0,0],0,500,0.001)
print("m=[0,0,0,0,0], c=0, epochs=500, L=0.001")
print("my m and c: ",Linear_model.gradient_descent())
print("my prediction: ", Linear_model.predict(x_new))
```

APPENDIX II:

```
#Question 2.1
df["Amount"].sum()

#Question 2.2
ww_grainger = df.loc[df['Vendor'].str.contains("WW GRAINGER", case=False)]
ww_grainger.Amount = ww_grainger.Amount.astype(float, copy = False)
ww_grainger["Amount"].sum()

#Question 2.3
wm_supercenter = df.loc[df['Vendor'].str.contains("WM SUPERCENTER", case = False)]
wm_supercenter.Amount = wm_supercenter.Amount.astype(float, copy = False)
wm_supercenter["Amount"].sum()

#Question 2.4
grocery_stores = df.loc[df['Merchant Category Code (MCC)'].str.contains("GROCERY STORES",
case = False)]
grocery_stores.Amount = grocery_stores.Amount.astype(float, copy = False)
grocery_stores["Amount"].sum()
```

APPENDIX III:

```
# Question 3.1
BalanceSheet = pd.read_excel(r'/Users/sushantkumbhar/Documents/Stevens Academics/Semester 4/FE 520 A - Intro to Python/Homeworks/Homework 4/Homework4_Dataset/Energy.xlsx')
Ratings = pd.read_excel(r'/Users/sushantkumbhar/Documents/Stevens Academics/Semester 4/FE 520 A - Intro to Python/Homeworks/Homework 4/Homework4_Dataset/EnergyRating.xlsx')

BalanceSheet.head()

BalanceSheet.iloc[:,373:376].head()

# Question 3.2
BalanceSheet_sixty = BalanceSheet.shape[0]*0.7 #Getting 70% values of the dataframe to keep as 30% NA values will be deducted
BalanceSheet = BalanceSheet.dropna(thresh = BalanceSheet_sixty, axis = 1) #Using drop NA with tresh to remove 30%NA columns
BalanceSheet.shape

# Question 3.3
#Droping columns with more than 90% o values
BalanceSheet.drop(columns=BalanceSheet.columns[((BalanceSheet==0).mean()>0.9)],axis=1, inplace = True)
BalanceSheet.shape


# Question 3.4
BalanceSheet = BalanceSheet.fillna(BalanceSheet.mean()) #Filling na values with mean
BalanceSheet.shape

#Question 3.5

BalanceSheet.iloc[:,15:158] = BalanceSheet.iloc[:,15:158].apply(lambda x: (x-x.min(axis = 0))/(x.max(axis = 0) - x.min(axis = 0)))
#BalanceSheet.iloc[:,15:99]



#Question 3.6
# Building correlation matrix for the following three columns
```

```
BalanceSheet[['Current Assets - Other - Total','Current Assets - Total', 'Other Long-term Assets',
'Assets Netting & Other Adjustments']].corr()


# Question 3.7
#Using pd.merge to merge two dataframe on 'Data Date' column using inner merge
Matched = pd.merge(BalanceSheet, Ratings, how = 'inner', on = ['Global Company Key', 'Data
Date'])
Matched.shape

#Matched["S&P Domestic Short Term Issuer Credit Rating"].isnull().value_counts()

#Matched[Matched.iloc[:, 1:2]==np.nan].head(5)

#Question 3.8
#Creating dictionary to map Rating to 'S&P Domestic Long Term Issuer Credit Rating' column
R = {'AAA' : 0,'AA+' : 1,'AA' : 2,'AA-' : 3,'A+' : 4,'A' : 5,'A-' : 6,
      'BBB+' : 7, 'BBB' : 8, 'BBB-' : 9, 'BB+' : 10, 'BB' : 11,
       np.nan : 12}

#Creating new column by mapping rating to the 'S&P Domestic Long Term Issuer Credit Rating'
column
Matched['Rate'] = Matched['S&P Domestic Long Term Issuer Credit Rating'].map(R)
Matched[["S&P Domestic Long Term Issuer Credit Rating", "Rate"]].value_counts()

#Question 3.9
x_rating = Matched.loc[:,'Accumulated Other Comprehensive Income (Loss)': 'Assets Level2
(Observable)'].values.tolist()
y_rating = list(Matched['Rate'])

m_rating = [0,0,0,0,0,0,0,0,0,0]


Linear_model = Linear_regression(x_rating,y_rating,m_rating,0,500,0.001)
print("I use m=[0,0,0,0,0,0,0,0,0,0], c=0, epochs=500, L=0.001")
print("The coefficients of m and c are: ",Linear_model.gradient_descent())
```