Table of Contents

Question 1

```
# 1. Linear Regression
class Linear_regression:
    def _init_(self, x, y, m, c, epochs, L):
        self.x = x
        self.y = y
        self.m = m
        self.c = c
        self.epochs = epochs
        self.L = L
        self.Dm = []
        self.Dc = []
        self.z = len(x)

    def gradient_descent(self):
        for self.z in range(self.epochs):
            for i in range(self.z):
                ypred = (self.x[i][0]*self.m)+self.c
                dm = self.x[i][0]*(ypred-self.y[i])
                self.Dm.extend([dm])
                dc = ypred-self.y[i]
                self.Dc.extend([dc])
            sum = 0
            n = 0
            for i in self.Dm:
                sum = sum+i
                n = n+1
                dm = sum/n
            sum = 0
            n = 0

            for i in self.Dc:
                sum = sum+i
                n = n+1
                dc = sum/n
            self.m = self.m-self.L*dm
            self.c = self.c-self.L*dc
        return(self.m, self.c)

    def predict(self, x_new):
        answer = []
        for i in range(len(x_new)):
            answer.append(self.m*x_new[i] + self.c)
        return (answer)
```

Question 2

```
# 2. Credit Transaction Data
file = open('res_purchase_2014.csv')
type(file)
csvreader = csv.reader(file)

header = []
header = next(csvreader)

amountIndex = header.index('Amount')
vendorIndex = header.index('Vendor')
merchantIndex = header.index('Merchant Category Code (MCC)')

cleanNumbers = re.compile(r'[^\d.()-]+')

amountSum = []
wwGraingerSum = []
wmSupercenterSum = []
groceryStoreSum = []
for row in csvreader:
    cleanedNumber = cleanNumbers.sub('', row[amountIndex])
    if (cleanedNumber[0] == "(" and cleanedNumber[-1] == ")"):
        cleanedNumber = cleanedNumber.replace("(", "-").replace(")", "")
    amountSum.append(float(cleanedNumber))

    if (row[vendorIndex] == "WW GRAINGER"):
        wwGraingerSum.append(float(cleanedNumber))

    if (row[vendorIndex] == "WM SUPERCENTER"):
        wmSupercenterSum.append(float(cleanedNumber))

    if ("GROCERY STORES" in row[merchantIndex]):
        groceryStoreSum.append(float(cleanedNumber))

print("1. Total amount spending captured in the dataset is: ${}"
.format(sum(amountSum)))
print("2. Total amount spending at WW GRAINGER is: ${}" .format(sum(wwGraingerSum)))
print("3. Total amount spending at WM SUPERCENTER is: ${}" .format(
    sum(wmSupercenterSum)))
print("4. Total amount spending at GROCERY STORES is: ${}" .format(
    sum(groceryStoreSum)))
```

In this question, we first read the CSV file and create variables to store indexes of Column Names we are going to work with. After that, we loop through the rows to clean the data and run a few operations to gain information. Later this information is displayed in formatted output.

Question 3

```
# 3.1 Read 'Energy.xlsx' and 'EnergyRating.xlsx' as BalanceSheet and
Ratings(dataframe).
BalanceSheet = pd.read_excel('Energy.xlsx')
Ratings = pd.read_excel('EnergyRating.xlsx')
```

We first read the xlsx file and load it into the panda data frame.

```
for (columnName, columnData) in BalanceSheet.iteritems():
    columnLength = BalanceSheet[BalanceSheet.columns[index]].count()
    totalMissingValues = BalanceSheet[BalanceSheet.columns[index]].isnull(
    ).sum()
    if (totalMissingValues > (0.3*columnLength)):
        deleteMissingValue.append(columnName)

    toalZeroValues = (BalanceSheet[columnName] == 0).sum()
    if (toalZeroValues > (0.9*columnLength)):
        deleteZeroValue.append(columnName)

    index += 1

# 3.2 For BalanceSheet, drop the column if more than 30% value in this colnmn is
missing value, see how many features are remaining.
BalanceSheet.drop(deleteMissingValue, axis=1, inplace=True)
print("Number of features remaining after dropping columns more than 30% missing
value: {}" .format(BalanceSheet.shape[1]))

# 3.3 For BalanceSheet, drop the column if more than 90% value in this colnmn is 0,
see how many features are remaining.
deleteZeroValue = list(set(deleteZeroValue) - set(deleteMissingValue))
BalanceSheet.drop(deleteZeroValue, axis=1, inplace=True)
print("Number of features remaining after dropping columns more than 90% zero value:
{}" .format(BalanceSheet.shape[1]))
```

After that, we loop through the rows to remove columns which have more than 30% missing data and more than 90% data is 0.

```
# 3.4 For BalanceSheet, replace all None or NaN with average value of each column.
for (columnName, columnData) in BalanceSheet.iteritems():
    if(is_numeric_dtype(BalanceSheet[columnName])):
        BalanceSheet[columnName] = BalanceSheet[columnName].fillna(
            BalanceSheet[columnName].mean())
```

We use fillna function to replace all none/NaN values to the mean of that column.

```
# 3.5 For BalanceSheet, Normalize the table
columnNamesNormalize = orignalColumnNames[orignalColumnNames.index(
    'Accounting Changes - Cumulative Effect'):(orignalColumnNames.index('Selling,
General and Administrative Expenses'))+1]
modifiedColumnNames = list(BalanceSheet.columns.values)

for columnName in columnNamesNormalize:
    if(columnName in modifiedColumnNames):
        BalanceSheet[columnName] = (BalanceSheet[columnName]-
BalanceSheet[columnName].min()) / \
            (BalanceSheet[columnName].max()-BalanceSheet[columnName].min())
```

We loop through to normalize the df for the sliced df.

```
# 3.6 Calculate the correlation matrix for variables
correlationMatrixColumns = ["Current Assets - Other - Total", "Current Assets -
Total",
                            "Other Long-term Assets", "Assets Netting & Other
Adjustments"]
modifiedColumnNames = list(BalanceSheet.columns.values)
correlationMatrixColumns = list(
    set(correlationMatrixColumns) & set(modifiedColumnNames))
correlationMatrix = BalanceSheet[correlationMatrixColumns].corr()
```

We compare current columns with original columns and create a correlation matrix for the specified columns.

```
# 3.7 Merge (inner) Ratings and BalanceSheet based on 'datadate' and 'Global Company
Key', and name merged dataset 'Matched'.
Matched = Ratings.merge(
    BalanceSheet, on=["Data Date", "Global Company Key"], how='inner')
```

We merge the Ratings df to BalanceSheet df based on the given columns.

```
# 3.8 Mapping
Rate = {"AAA": 0,       "AA+": 1,       "AA": 2,        "AA-": 3,       "A+": 4,
        "A": 5,        "A-": 6,        "BBB+": 7,       "BBB": 8,       "BBB-": 9,
        "BB+": 10,       "BB": 11,       "others": 12
        }
class MyDict(dict):
    def __missing__(self, key):
        return 12
Matched['Rate'] = Matched['S&P Domestic Long Term Issuer Credit Rating'].map(
    MyDict(Rate))
```

We create a dict with the given ratings and then map it to the df on Rating