Yanzichu 300476924

Assignment 4

Part1

1. I used 309 as Random seed and 0.3 as test split size.
2. Load dataset : by using read_csv( ) to read diamonds.csv into memory as a DataFrame
3. Drop the first column : I print first 10 rows for checking the attributes,and from column 2 to column 10 record features of diamonds , and column 11 is the price of diamonds.First column is index so it can be dropped.

```
   Unnamed: 0  carat        cut color clarity  depth  table     x     y     z  \
0           1   0.23      Ideal     E     SI2   61.5   55.0  3.95  3.98  2.43
1           2   0.21    Premium     E     SI1   59.8   61.0  3.89  3.84  2.31
2           3   0.23       Good     E     VS1   56.9   65.0  4.05  4.07  2.31
3           4   0.29    Premium     I     VS2   62.4   58.0  4.20  4.23  2.63
4           5   0.31       Good     J     SI2   63.3   58.0  4.34  4.35  2.75
5           6   0.24  Very Good     J    VVS2   62.8   57.0  3.94  3.96  2.48
6           7   0.24  Very Good     I    VVS1   62.3   57.0  3.95  3.98  2.47
7           8   0.26  Very Good     H     SI1   61.9   55.0  4.07  4.11  2.53
8           9   0.22       Fair     E     VS2   65.1   61.0  3.87  3.78  2.49
9          10   0.23  Very Good     H     VS1   59.4   61.0  4.00  4.05  2.39

   price
0    326
1    326
2    327
3    334
4    335
5    336
6    336
7    337
8    337
9    338
```

After dropping:

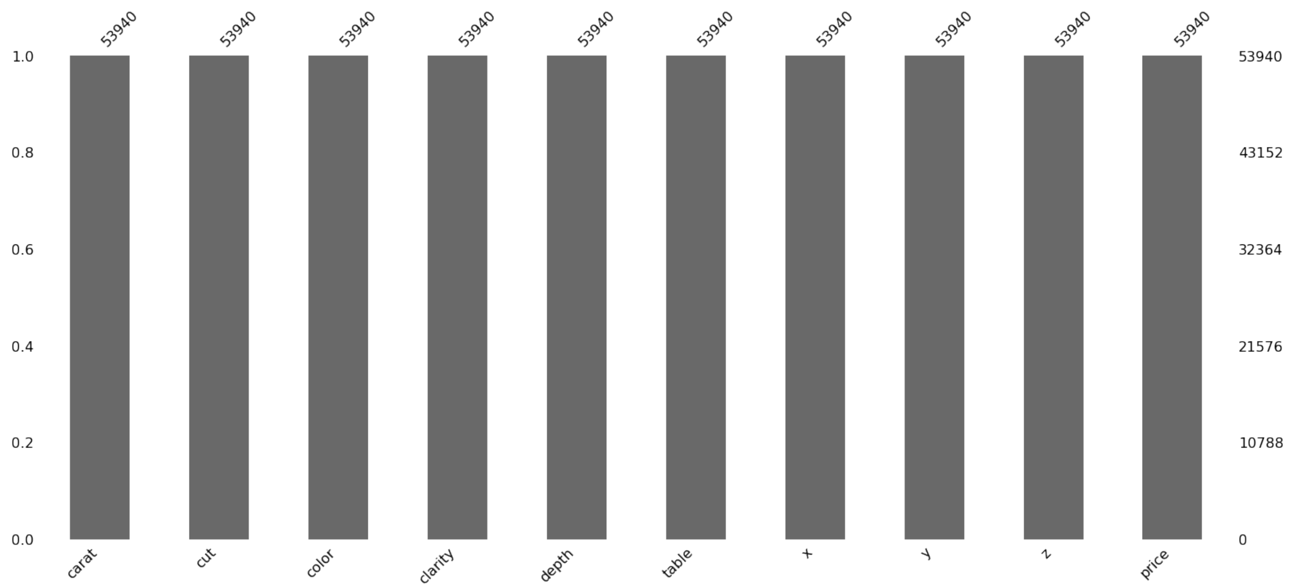
```
   carat      cut color clarity  depth  table     x     y     z  price
0   0.23    Ideal     E     SI2   61.5   55.0  3.95  3.98  2.43    326
1   0.21  Premium     E     SI1   59.8   61.0  3.89  3.84  2.31    326
2   0.23     Good     E     VS1   56.9   65.0  4.05  4.07  2.31    327
3   0.29  Premium     I     VS2   62.4   58.0  4.20  4.23  2.63    334
4   0.31     Good     J     SI2   63.3   58.0  4.34  4.35  2.75    335
```

4. Check missing value: there was no missing data.

```
print(df.isnull().sum()) # Check missing values
```

```
carat      0
cut        0
color      0
clarity    0
depth      0
table      0
x          0
y          0
z          0
price      0
dtype: int64
```

Visualization the missing value:



5.Check invalid value:
x, y, z represent the volume of a diamond so any of them can not be 0.
And the max value of y and z exceed mean of them too much , and those value are outliers.

In [6]:
```python
# here we can see the summry of the x, y and z min values are 0 which is impossible
print(df.describe())
```

```
              carat         depth         table             x             y  \
count  53940.000000  53940.000000  53940.000000  53940.000000  53940.000000
mean       0.797940     61.749405     57.457184      5.731157      5.734526
std        0.474011      1.432621      2.234491      1.121761      1.142135
min        0.200000     43.000000     43.000000      0.000000      0.000000
25%        0.400000     61.000000     56.000000      4.710000      4.720000
50%        0.700000     61.800000     57.000000      5.700000      5.710000
75%        1.040000     62.500000     59.000000      6.540000      6.540000
max        5.010000     79.000000     95.000000     10.740000     58.900000

                  z         price
count  53940.000000  53940.000000
mean       3.538734   3932.799722
std        0.705699   3989.439738
min        0.000000    326.000000
25%        2.910000    950.000000
50%        3.530000   2401.000000
75%        4.040000   5324.250000
max       31.800000  18823.000000
```

6. Data preprocessing(Cleaning):

In [7]:
```
# Looking to see how many unreasonable values in dataset
print(df.loc[(df['x']==0) | (df['y']==0) | (df['z']==0)])
print(df.loc[(df['x']>20) | (df['y']>20) | (df['z']>20)])
```

```
       carat        cut color clarity  depth  table     x     y    z  price
2207    1.00    Premium     G     SI2   59.1   59.0  6.55  6.48  0.0   3142
2314    1.01    Premium     H      I1   58.1   59.0  6.66  6.60  0.0   3167
4791    1.10    Premium     G     SI2   63.0   59.0  6.50  6.47  0.0   3696
5471    1.01    Premium     F     SI2   59.2   58.0  6.50  6.47  0.0   3837
10167   1.50       Good     G      I1   64.0   61.0  7.15  7.04  0.0   4731
11182   1.07      Ideal     F     SI2   61.6   56.0  0.00  6.62  0.0   4954
11963   1.00  Very Good     H     VS2   63.3   53.0  0.00  0.00  0.0   5139
13601   1.15      Ideal     G     VS2   59.2   56.0  6.88  6.83  0.0   5564
15951   1.14       Fair     G     VS1   57.5   67.0  0.00  0.00  0.0   6381
24394   2.18    Premium     H     SI2   59.4   61.0  8.49  8.45  0.0  12631
24520   1.56      Ideal     G     VS2   62.2   54.0  0.00  0.00  0.0  12800
26123   2.25    Premium     I     SI1   61.3   58.0  8.52  8.42  0.0  15397
26243   1.20    Premium     D    VVS1   62.1   59.0  0.00  0.00  0.0  15686
27112   2.20    Premium     H     SI1   61.2   59.0  8.42  8.37  0.0  17265
27429   2.25    Premium     H     SI2   62.8   59.0  0.00  0.00  0.0  18034
27503   2.02    Premium     H     VS2   62.7   53.0  8.02  7.95  0.0  18207
27739   2.80       Good     G     SI2   63.8   58.0  8.90  8.85  0.0  18788
49556   0.71       Good     F     SI2   64.1   60.0  0.00  0.00  0.0   2130
49557   0.71       Good     F     SI2   64.1   60.0  0.00  0.00  0.0   2130
51506   1.12    Premium     G      I1   60.4   59.0  6.71  6.67  0.0   2383
       carat        cut color clarity  depth  table     x      y      z  price
24067   2.00    Premium     H     SI2   58.9   57.0  8.09  58.90   8.06  12210
48410   0.51  Very Good     E     VS1   61.8   54.7  5.12   5.15  31.80   1970
49189   0.51      Ideal     E     VS1   61.8   55.0  5.15  31.80   5.12   2075
```

In [8]:
```
print(len(df.loc[(df['x']==0) | (df['y']==0) | (df['z']==0)]))
print(len(df.loc[(df['x'] >20) | (df['y'] >20) | (df['z']>20)]))
```

```
20
3
```

I dropped data x, y ,z = 0 and x , y ,z > 20.

In [9]:
```
# I dropped them as they don't make sense-------Outliers && Zero
df = df[(df[['x', 'y', 'z']] != 0).all(axis=1)]
df = df[(df[['x', 'y', 'z']] < 20).all(axis=1)]
#Check wether they has been removed
print(len(df.loc[(df['x']==0) | (df['y']==0) | (df['z']==0)]))
print(len(df.loc[(df['x'] >20) | (df['y'] >20) | (df['z']>20)]))
```

```
0
0
```

7. Data preprocessing (Convert):

Because we wanna use regression so the attribute like cut ,color and clarity should transfer into numeric value:

Cut					
Categorical Raw Data	Ideal	Premium	Very Good	Good	Fair
After Quantifying	100	90	80	70	60

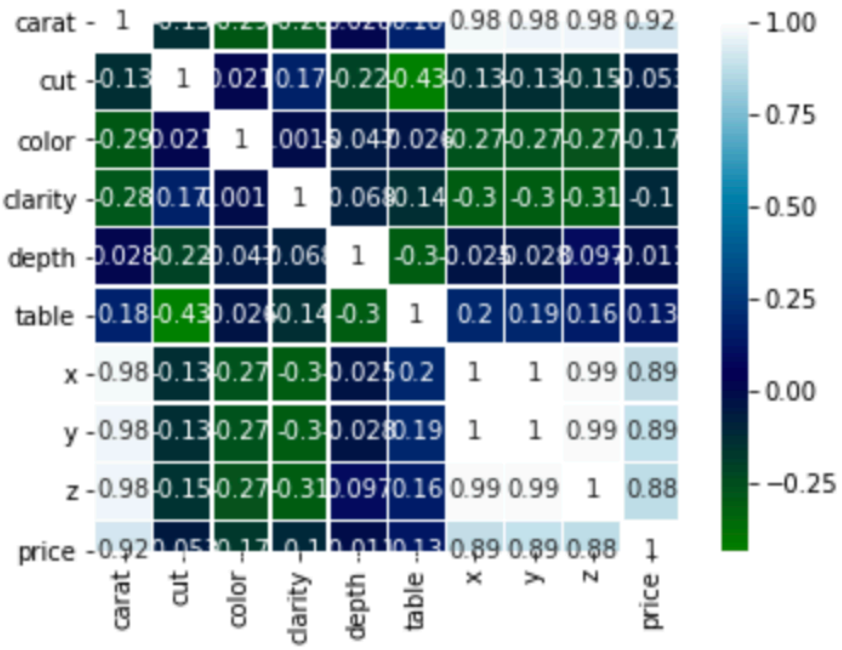
Colour							
Categorical Raw Data	D	E	F	G	H	I	J
After Quantifying	100	90	80	70	60	50	40

Clarity								
Categorical Raw Data	IF	VVS1	VVS2	VS1	VS2	SI1	SI2	I1
After Quantifying	100	90	80	70	60	50	40	30

The converted values in table above are follow the diamonds valuation chart.

8. Data preprocessing (Split):

The data set was split into 2 DataFrames and 2 Series, Xs_train_set, Xs_test_set, y_train_set, y_test_set, with the shape of (37758, 9), (16182, 9), (37758, 1) and (16182, 1), respectively. The test_size was set in the beginning of the program: 0.3.



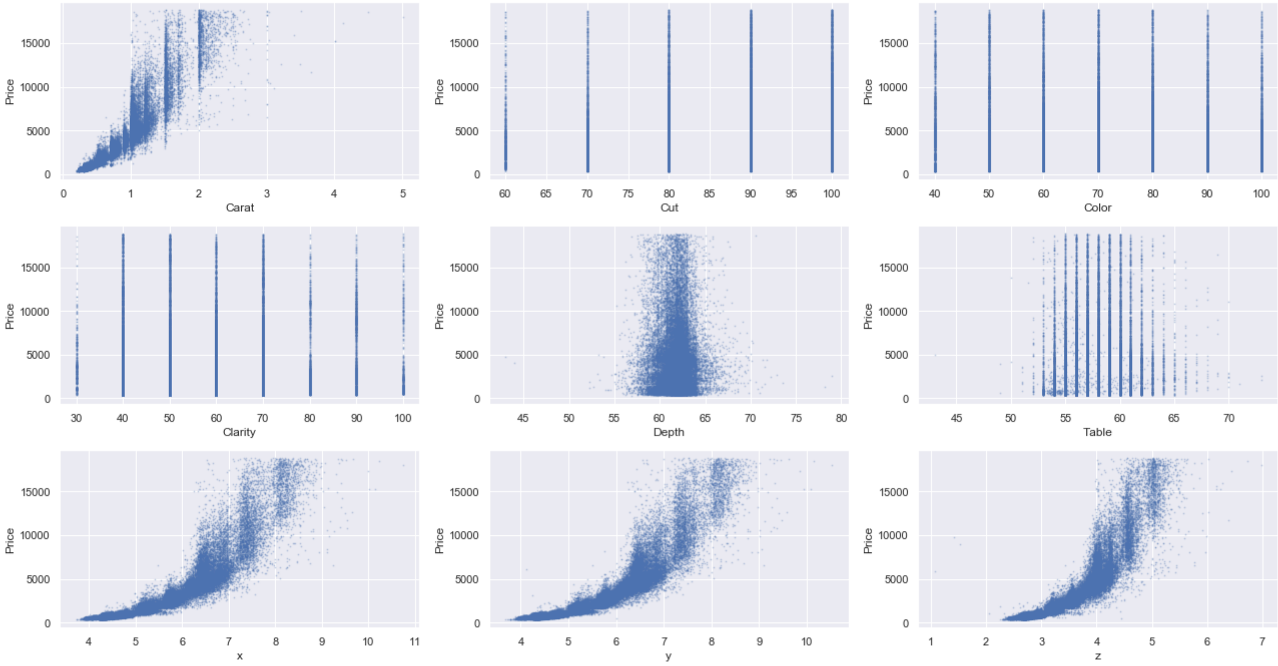
9. Data understanding:

	Carat	Cut	Colour	Clarity	Depth	Table	X	Y	Z	Price
Price	0.92	-0.051	-0.171	-0.144	-0.013	0.127	0.886	0.888	0.881	1

As the table shows, Carat and volume of diamond have big relation with price.

Yanzichu 300476924

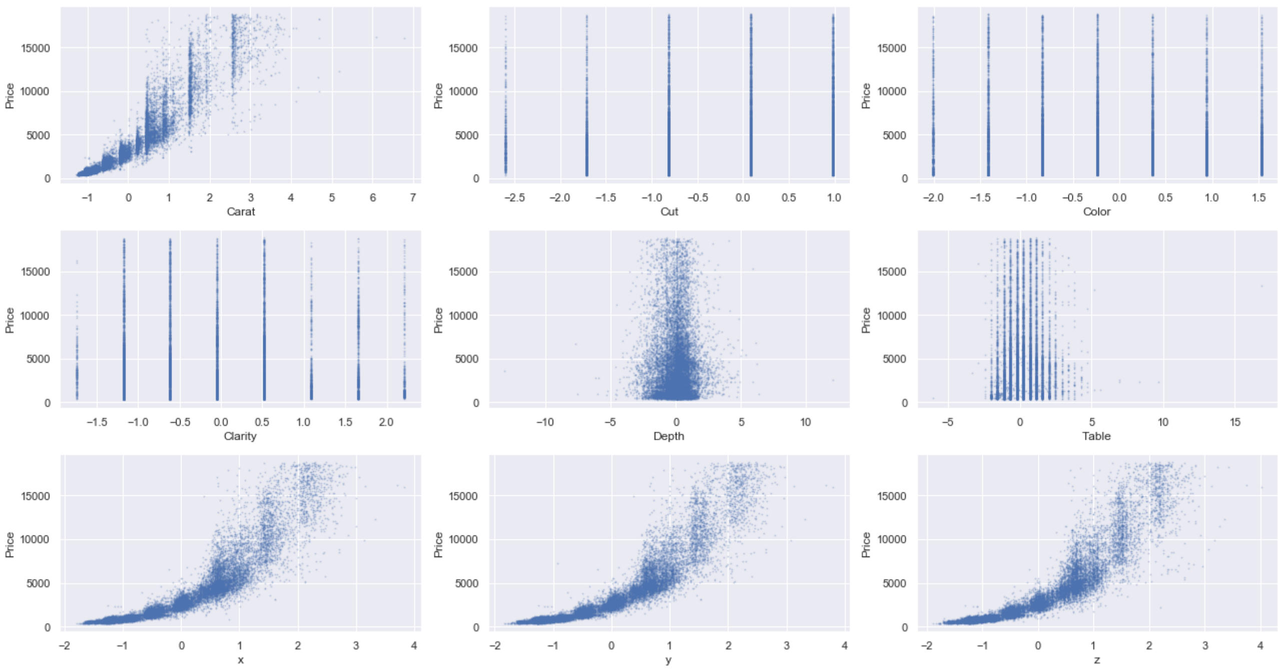
The distribution of 9 attributes :
After cleaning 23 unreasonable values but before Standardization



Standardization:
The Xs_train_set and Xs_test_set were standardized by the mean and standard deviation values of

```
#Standardize
Xs_train_set_mean = X_train.mean()
Xs_train_set_std  = X_train.std()
Xs_train_set      = (X_train - Xs_train_set_mean) / Xs_train_set_std
Xs_test_set       = (X_test  - Xs_train_set_mean) / Xs_train_set_std
```

Xs_train_set.
After Standardization:



10. Modelling : (default value)

	R^2	Rank	RMSE	Rank	MAE	Rank	MSE	Rank	Execution Time(s)	Rank
SVR	0.49	10	2891.84	10	1401.67	10	8362721.05	10	68.106	10
LinearSVR	0.84	9	1626.45	9	887.42	9	2645340.17	9	0.043	3
Ridge Regression	0.90	7	1275.89	7	844.66	8	1627900.27	8	0.010	1
LinearRegression	0.90	8	1275.83	6	844.64	7	1627736.29	7	0.018	2
SGDRegressor	0.90	6	1280.02	8	848.87	6	1638452.69	6	0.143	4
MLPRegressor	0.94	4	987.00	5	576.80	5	974164.58	4	19.245	9
DecisionTreeRegressor	0.96	2	756.06	3	366.81	2	571625.25	3	0.281	5
KNeighborsRegressor	0.96	5	796.47	4	410.80	4	634367.57	5	0.790	6
GradientBoostingRegressor	0.97	3	661.92	2	367.40	3	438135.67	2	1.632	8
RandomForestRegressor	0.98	1	574.45	1	288.75	1	329990.60	1	1.627	7

According to the table above, we can see the performances of RandomForestRegressor, KNeighborsRegressor and GradientBoostingRegressor are Top 3.And the worst case is SVR. However, If we compare time , the Top 3 good performance cases spent longer time than other algorithms.

I tuned SVR ,linear SVR and Multi- layer Perceptron Regression these three outcomes become better , but other algorithms' result did not change.And at the same time ,the execution time reduced a lot after optimizing the parameters.

Algoritm	Parameter I tuned	R square before tuning	R square after tuning
SVR	Kernel = ‚linear' C = 500.0 (the larger the better)	0.49	0.95
Linear SVR	C = 5.0 loss = 'squared_epsilon_insensitive' dual = True	0.84	0.90
MLP	activation = relu solver = lbfgs learning_rate = adaptive	0.94	0.96

Part2:

1. Firstly, we need to read adult.data and adult.test into memory. After I print out the first 5 rows of training set and test set I find there are no attribute name of original data, so I add them after I searched online:
   columns = ['Age','Workclass','fnlgwt','Education','Education Num','Marital Status','Occupation','Relationship','Race','Sex','Capital Gain','Capital Loss', ‘Hours/Week’,'Country','Above/Below 50K’]

```
print(train.head(3))#check whether attribute names are added
print("=========================================")
print(test.head(3))
```

```
   Age          Workclass  fnlgwt  Education  Education Num  \
0   39          State-gov   77516  Bachelors             13
1   50   Self-emp-not-inc   83311  Bachelors             13
2   38            Private  215646    HS-grad              9

        Marital Status         Occupation   Relationship   Race   Sex  \
0        Never-married       Adm-clerical  Not-in-family  White  Male
1   Married-civ-spouse    Exec-managerial        Husband  White  Male
2             Divorced  Handlers-cleaners  Not-in-family  White  Male

   Capital Gain  Capital Loss  Hours/Week        Country Above/Below 50K
0          2174             0          40  United-States           <=50K
1             0             0          13  United-States           <=50K
2             0             0          40  United-States           <=50K
=========================================
   Age  Workclass  fnlgwt     Education  Education Num      Marital Status  \
1   38    Private   89814       HS-grad              9  Married-civ-spouse
2   28  Local-gov  336951    Assoc-acdm             12  Married-civ-spouse
3   44    Private  160323  Some-college             10  Married-civ-spouse

          Occupation Relationship   Race   Sex  Capital Gain  Capital Loss  \
1    Farming-fishing      Husband  White  Male             0             0
2    Protective-serv      Husband  White  Male             0             0
3  Machine-op-inspct      Husband  Black  Male          7688             0

   Hours/Week        Country Above/Below 50K
1          50  United-States          <=50K.
2          40  United-States           >50K.
3          40  United-States           >50K.
```

```
print(train.isnull().sum()) # Check missing values
print("========================================")
print(test.isnull().sum()) # Check missing values
```

```
Age                0
Workclass          0
fnlgwt             0
Education          0
Education Num      0
Marital Status     0
Occupation         0
Relationship       0
Race               0
Sex                0
Capital Gain       0
Capital Loss       0
Hours/Week         0
Country            0
Above/Below 50K    0
dtype: int64
========================================
Age                0
Workclass          0
fnlgwt             0
Education          0
Education Num      0
Marital Status     0
Occupation         0
Relationship       0
Race               0
Sex                0
Capital Gain       0
Capital Loss       0
Hours/Week         0
Country            0
Above/Below 50K    0
dtype: int64
```

2. Missing value: There are no missing value in training set and test set.

3. I used train.describe() and test.describe() to see whether there are outliers and unreasonable values. And as we can see , these numeric attributes are good.

```
print(train.head(3))#check whether attribute names are added
print("==========================================")
print(test.head(3))
```

```
   Age         Workclass  fnlgwt  Education  Education Num  \
0   39         State-gov   77516  Bachelors             13
1   50  Self-emp-not-inc   83311  Bachelors             13
2   38           Private  215646    HS-grad              9

       Marital Status         Occupation   Relationship   Race   Sex  \
0       Never-married       Adm-clerical  Not-in-family  White  Male
1  Married-civ-spouse    Exec-managerial        Husband  White  Male
2            Divorced  Handlers-cleaners  Not-in-family  White  Male

   Capital Gain  Capital Loss  Hours/Week        Country Above/Below 50K
0          2174             0          40  United-States           <=50K
1             0             0          13  United-States           <=50K
2             0             0          40  United-States           <=50K
==========================================
   Age  Workclass  fnlgwt     Education  Education Num      Marital Status  \
1   38    Private   89814       HS-grad              9  Married-civ-spouse
2   28  Local-gov  336951    Assoc-acdm             12  Married-civ-spouse
3   44    Private  160323  Some-college             10  Married-civ-spouse

          Occupation Relationship   Race   Sex  Capital Gain  Capital Loss  \
1    Farming-fishing      Husband  White  Male             0             0
2    Protective-serv      Husband  White  Male             0             0
3  Machine-op-inspct      Husband  Black  Male          7688             0

   Hours/Week        Country Above/Below 50K
1          50  United-States          <=50K.
2          40  United-States           >50K.
3          40  United-States           >50K.
```

4. I dropped final weight and education these two columns because final weight represent the population of target people, however , what we want is to classify a person whether can earn above 50k or not.So final weight this attribute actually have some bad influence to the final result. The reason why I removed education is because education num represent a numeric value of educational level ,the bigger number is ,the higher education level it represent.It is duplicate to keep two same attributes. And numeric value is better than categorical values.

5. I convert Above/Below 50K this string variable to a binary variable (0 or 1) which represent above(1) and below(0) by using LabelEncoder() method in sklearn. And dummy variables were created for all the categorical data in both train set and test set.

```
   Above/Below 50K  Workclass_Federal-gov  Workclass_Local-gov  \
0                0                      0                    0
1                0                      0                    0
```

6. Then I checked wether training set and test set have same shape after making dummy variables.

```
# seeing if the datasets are balanced
print(train['Above/Below 50K'].value_counts()[0]/train.shape[0])
print(train['Above/Below 50K'].value_counts()[1]/train.shape[0])
```

```
0.7510775147536636
0.24892248524633645
```

```
# they aren't balanced

# seeing if they have the same number of columns
print(test.shape)
print(train.shape)
# they don't so need to find out what that is
```

```
(15059, 87)
(30162, 88)
```

```
# checking to see which column is missing
missing_cols = set(train) - set(test)#compare train and test set
print(missing_cols)
```

```
{'Country_Holand-Netherlands'}
```

```
# Adding in a column that was missing from the test set filled with 0's
test['Country_Holand-Netherlands'] = pd.Series(0, index = test.index)
```

```
#check whether it was inserted
print(test.shape)
print(train.shape)
```

```
(15059, 88)
(30162, 88)
```

Finally I finish the data preprocessing.

7.Modeling:

Using default setting for each classification algorithm, the accuracy, precision, recall rate, F1 score, and AUC, together with the rankings, are summarized below:

	Acc	R	Prec	R	Rec	R	F1	R	AUC	R
GaussianNaiveBayes	0.80	9	0.57	9	0.79	1	0.66	4	0.80	1
AdaBoostClassifier	0.85	3	0.74	6	0.63	3	0.68	3	0.78	4
KNeighborsClassifier	0.85	5	0.73	7	0.64	2	0.68	2	0.78	3
GradientBoostingClassifier	0.86	1	0.78	2	0.62	4	0.69	1	0.78	2
LogisticRegression	0.85	2	0.74	5	0.58	6	0.65	6	0.76	6
DecisionTreeClassifier	0.84	7	0.70	8	0.60	5	0.65	5	0.76	5
RandomForestClassifier	0.85	4	0.75	4	0.57	7	0.65	7	0.75	7
LinearDiscriminantAnalysis	0.84	6	0.76	3	0.48	8	0.59	8	0.72	8
MultilayerPerceptronClassifier	0.82	8	0.83	1	0.32	9	0.46	9	0.65	9
SVMClassifier	0.75	10	0.46	10	0.06	10	0.10	10	0.52	10

Comparing to accuracy, AUC seems better to evaluate the performance of a classification. The higher AUC the better performance. And F1 can be check at same time which can make our judgment more reliable because it can prevent overestimate superficial high value in AUC.

And F1 value is influenced by precision and recall value. There are two cases which will make F1 value similar ,high value of precision and low value of recall, high value of recall and low value of precision. So if we want to use F1 to evaluate the performance it is not easy because we need to check precision and recall values after checking F1.

So based on AUC and F1, we can get performance of Adaptive Boosting Classifier and Gradient Boosting Classifier are good.And yes, both of these two algorithms belong to Boosting algorithm. Boosting algorithm combines some average-performed models to get a better-performed model.

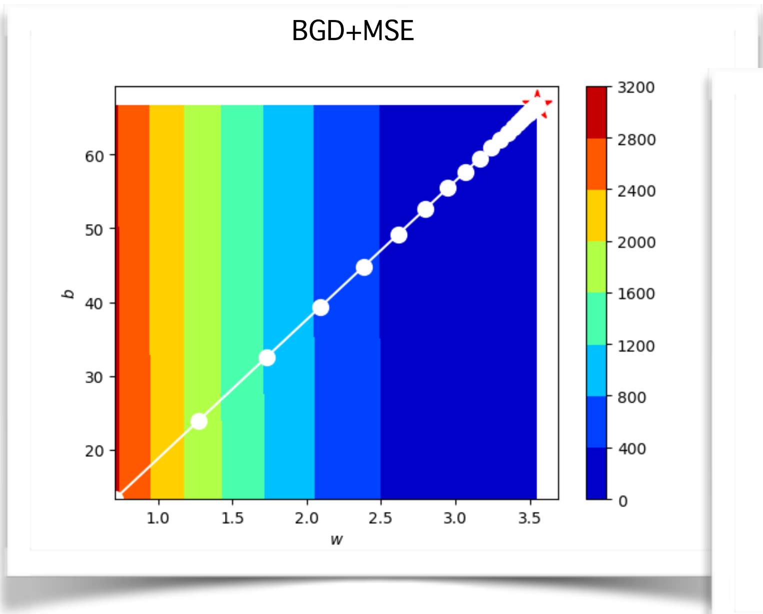
Gradient boosting generates learners during the learning process. It build first learner to predict the values/labels of samples, and calculate the loss (the difference between the outcome of the first learner and the real value). It will build a second learner to predict the loss after the first step. The step continues to learn the third, forth… until certain threshold.

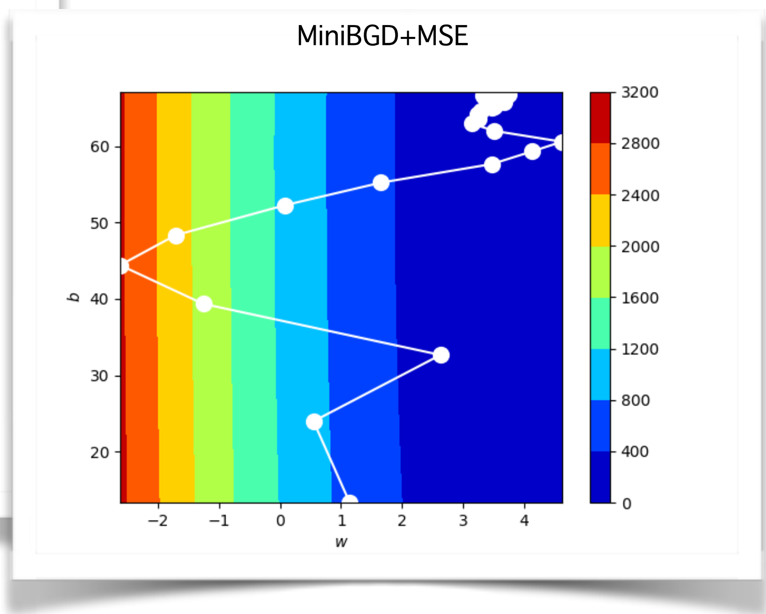
Adaptive boosting requires users specify a set of weak learners (alternatively, it will randomly generate a set of weak learner before the real learning process). It will learn the weights of how to add these learners to be a strong learner. The weight of each learner is learned by whether it predicts a sample correctly or not. If a learner is mis-predict a sample, the weight of the learner is reduced a bit. It will repeat such process until converge.

Part3:

1. Plot the paths of gradient descent of BGD+MSE and MiniBatchBGD+MSE, then discuss their differences and justify why.

a) Gradient Descent paths of BGD+MSE and MiniBatchBGD+MSE





BGD(Left hand side pic) uses all the training data in every loop of updating the weights to minimize the loss function. The loss moves toward the minimum directly since it considers all the data in the training set. The problem is the updating speed will be very slow if the dataset is huge.

SGD considers only one more new data point in the weights updating process. This can speed up the training very much. However , because of this, the noise cannot be filtered properly, which makes the path can not move toward the minimum for every time. However, the general direction points toward the minimum. The high speed is the biggest advantage.

MiniBGD(Right hand side pic) combines the advantages, but also compromises the shortages from both BGD and SGD. So, it's faster than BGD, and in the figure on the right-hand side, the path does not go toward the up-right corner directly.

b) Result of the four learnt models over the MSE, R-Squared, and MAE performance metrics on the test set.

Model	MSE	R square	MAE
BGD+MSE	2.42	0.84	1.28
MGBD+MSE	2.46	0.83	1.29
PSO+MSE	2.41	0.84	1.28
PSO+MAE	2.43	0.84	1.28

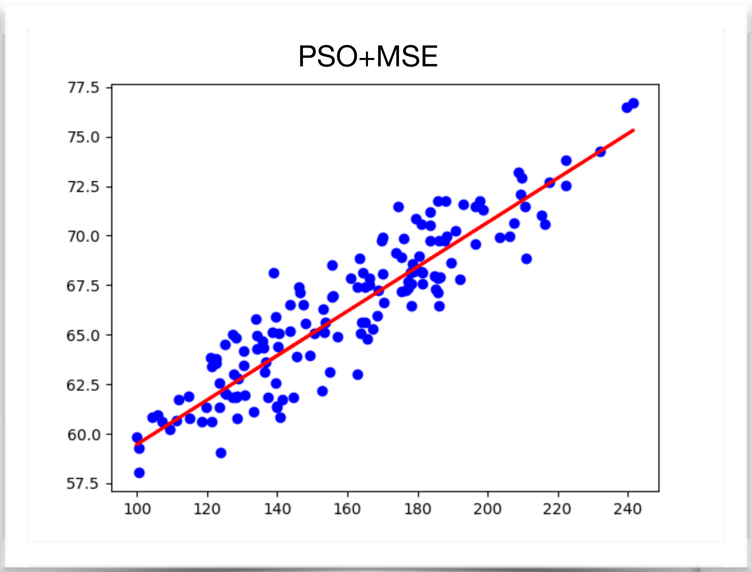
The differences in the metrics between the models are negligible and they converge to approximately the same point.

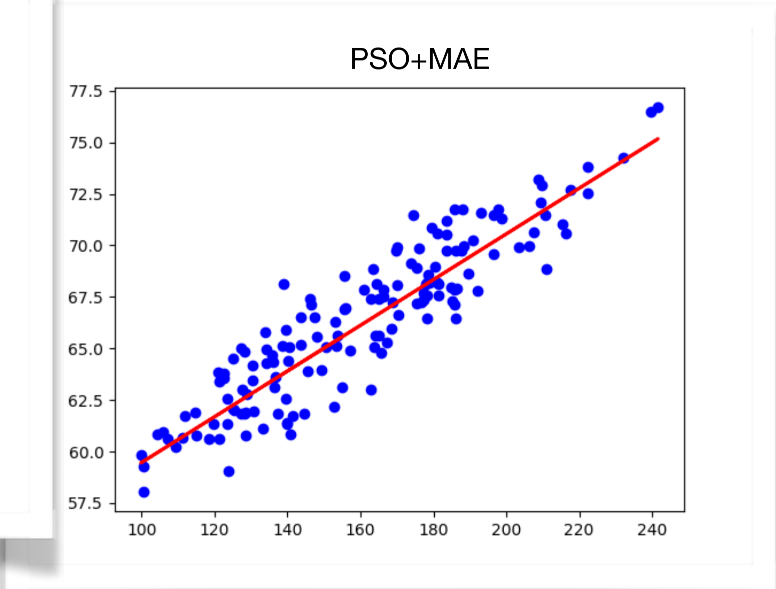
For the MSE metric, PSO + MSE yields the best result.

For the MAE metric all models, apart from MiniBatchBGD + MSE, yield the same result with MiniBatchBGD + MSE only differing by 0.01.

For the R-Squared metric, MiniBatchBGD + MSE yields the best result with the other three only 0.01 less.

c) Scatter plots with regression line learnt by PSO+MSE and PSO+MAE in test set





d) Computation time comparison among the 4 models

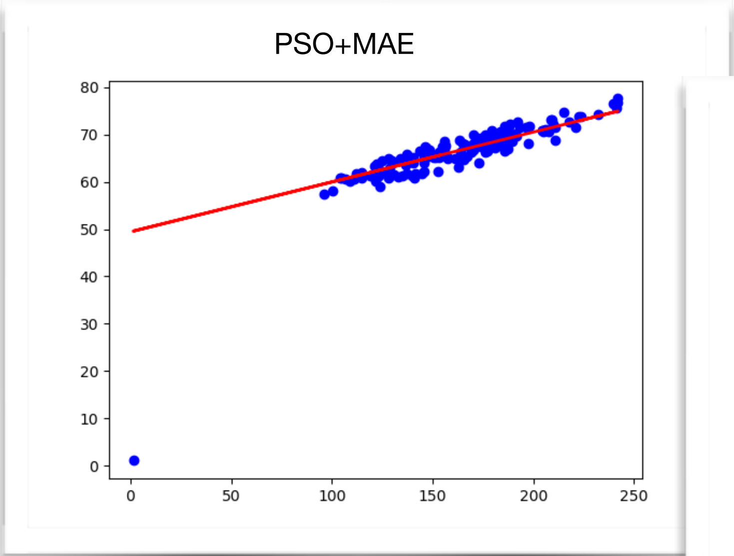
Models	Execution time (seconds)
BGD+MSE	0.009
MBGD+MSE	0.120
PSO+MSE	0.422

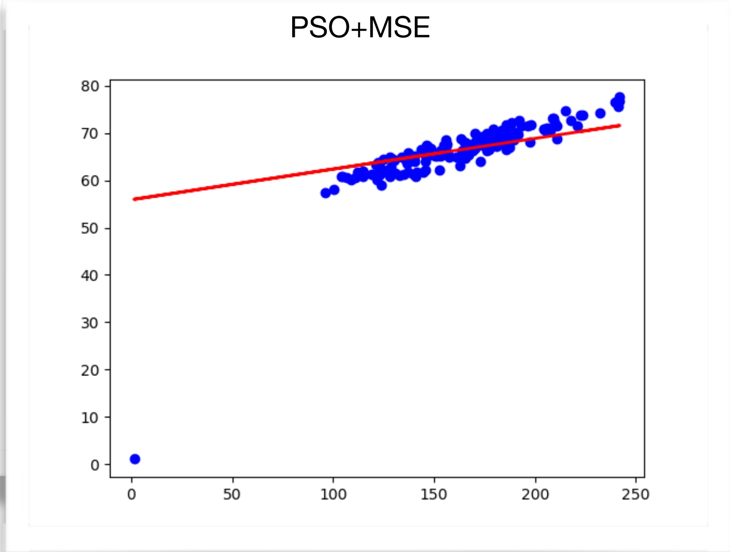
Among 3 models, BGD+MSE is the fastest, because the data set is not very large, and the array multiplication function in Numpy library is optimised. The processing as a whole array in BGD is faster than separating it into batches, calculating and saving for each batch. However, it is believed that, when calculating a much larger data set, the BGD method can be slower than MiniBGD, and even may not be fit into the memory of a computer.

For PSO algorithm , because of the complicated calculation is slower than subtraction PSO is slower than the other two optimizer.

2. On the dataset with outliers, PSO+MSE and PSO+MAE methods were implemented.

a) Scatter plots with regression line learnt by PSO+MSE and PSO+MAE in test set





b) Sensitivity of PSO+MES and PSE+MAE

In 3.1.c), the dataset has no outliers in the graph but we can see apparently there is an outlier in both of these plots.We can get from the graphs that PSO+MAE is less sensitive to outliers than PSO+MSE. The regression line in the PSO + MAE graph fit the data better than PSO+MSE due to the different calculation of error. Because MAE does not square the errors in the calculation but MSE does.

Since MSE squares the error (y - y_predicted = e), the value of error (e) increases a lot if e > 1. If we have an outlier in our data, the value of e will be high and e^2 will be >> |e|. This will make the model with MSE loss give more weight to outliers than a model with MAE loss.

c) Discuss whether we can use gradient descent or mini-batch gradient descent to optimise MAE? and explain why.

If the MSE cost function is plotted with the theta, the MSE cost function is parabolic, which allows the smaller step size to be automatically taken as the convergence process approaches the optimization point (because the slope also becomes smaller). This means that even with a fixed learning rate, the learning rate can be adjusted by itself, resulting in accurate modeling results. However, if MAE is used, the cost function is linear, which means that the adjustments for each convergence cycle are the same. Even if a small or varying learning rate can be used in the MAE, the convergence can be too fast and not as accurate as the MSE in the final phase of convergence.