The 'loc[ ]' & 'iloc[ ]' Usages, Errors, Solve Errors in pandas

M.Sc COMPUTER SCIENCE

Date : 31/03/2023

Name : Swetha Balaji Singh

Matriculation No : 92129337

Course : Python with Programming

Course Code : DLMDSPWP01

Tutor’s Name : Sanja Stajner

Table of Content

List of Figures

1. ABSTRACT

The research paper focuses on the use of the 'loc[ ]' and 'iloc[ ]' indexing attributes in the Python programming language. ‘'loc[ ]” and ‘'iloc[ ]” is the key features in pandas. Pandas is an open-source library used for data analysis. In pandas ‘'loc[ ]” and ‘'iloc[ ]” allows to manipulate and select data in multiple ways to retrieve or view data.

The research paper explores the difference between 'loc[ ]' and 'iloc[ ]', how to use them for slicing and selection. Both ’'loc[ ]” and ‘'iloc[ ]” is used to access the data by rows and columns. ‘'loc[ ]” is used to access data using label-based indexing or boolean arrays, while ‘'iloc[ ]” is used for integer based indexing or numerical based indexing. ‘'loc[ ]” and ‘'iloc[ ]” is used for slicing and sub-setting with multiple conditions, and indexing with multiple columns.

Research includes discussion on the importance of understanding the difference between 'loc[ ]' and 'iloc[ ]' in order to effectively manipulate and analyze data using the Pandas library in Python.

2. Introduction:

Pandas is a strong tool for data analysis and manipulation library in Python, which are extremely important in many sectors today. The *'loc[ ]'* and *'iloc[ ]'* indexers, which give an adaptable simple method to select and modify data, are one of Pandas' most important features. The purpose of this research paper is to give an in-depth description of *'loc[ ]'* and *'iloc[ ]'* in Pandas, as well as how to efficiently use them for data analysis. Among its many functionalities, Pandas allows *'loc[ ]'* and *'iloc[ ]'* for slicing data from dataframe, selecting data from data frame, manipulating data, data analysis, to create pivot table, merge dataframes, selecting data using conditional functions. It is most important for data scientist and data analyst to understand *'loc[ ]'* and *'iloc[ ]'* functions how it works and how to use *'loc[ ]'* and *'iloc[ ]'* functions for data manipulation and retrieve.

In pandas before introducing *'loc[ ]'* and *'iloc[ ]'* for indexing the data *ix* method was used for indexing, *ix* is the hybrid of *'loc[ ]'* and *'iloc[ ]'. 'loc[ ]'* and *'iloc[ ]'* was introduced in pandas version 0.11 in 2013.

In pandas DataFrame .'loc[ ]' and .'iloc[ ]' are python attributes of the dataframe object that use square bracket accessors to flexibly specify rows and columns, enabling the combination of projection and selection in a single operation. We always use square bracket accessors with either .'loc[ ]' and .'iloc[ ]' attributes. Inside the square brackets, we specify a first component that determines the desired rows, then a row-comma-column notation familiar from algebra when subscripting two-dimensional variables(David White, 2020 p.193).

"loc[ ]" is a pandas dataframe attribute that allows for label-based indexing and selection of data. *'loc[ ]'* is dataframe and series method. This means that indexing and selection of data can be performed based on the labels or names or integer of the rows and columns in the dataframe. To retrieve a data from a table we use syntax *'df.loc['row no','column name']'*

Example : list = df.loc['Row3','Student_name]
print(list)

The output of the dataframe.'loc[ ]' will return the current desired position value in the dataframe.

"iloc[ ]" is a Pandas dataframe attribute that allows for integer-based indexing and selection of data. This means that indexing and selection of data can be performed based on the integer position of the rows and columns in the dataframe, instead of their labels or names. To retrieve a data from a table we use syntax *'df.loc['row no','column no']*

Example : list = df.iloc[0:3]
print(list)

The output of the dataframe.'loc[ ]' will return the current desired position value in the dataframe.

In this research paper it explains about *'loc[ ]'* and *'iloc[ ]'*

Uses of *'loc[ ]'* and *'iloc[ ]'* in pandas
Differences in *'loc[ ]'* and *'iloc[ ]'* in pandas
Errors and error management
Avoid Errors in *'loc[ ]'* and *'iloc[ ]'* in pandas
Alternatives in *'loc[ ]'* and *'iloc[ ]'* in pandas
Conclusions

3. Uses of *'loc[ ]'* and *'iloc[ ]'* in pandas

To select or slice the dataframes using rows and columns, we utilize 'loc[ ]' and 'iloc[ ]' to choose a certain row and column. We can select multiple rows and columns at a time for slicing and selecting rows and columns in *'loc[ ]'* and *'iloc[ ]'*. By using this methods it is fast, quick, easy to read and interchange, merge data. Using of *'loc[ ]'* and *'iloc[ ]'* examples:

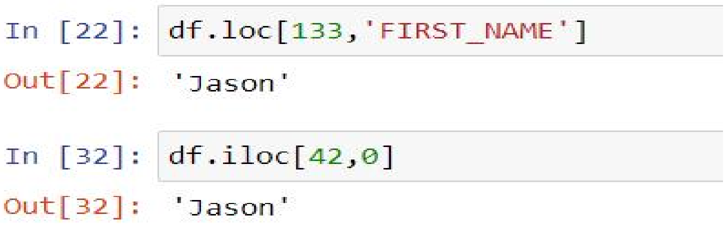
3.1 Select a Single Value :

To Select single data value from a dataframe in both *'loc[ ]'* and *'iloc[ ]'* syntax :

loc[ ] : loc[row_label, column_label]

iloc[ ] : iloc[row_no, column_no]

Example :



```
In [22]: df.loc[133,'FIRST_NAME']
Out[22]: 'Jason'

In [32]: df.iloc[42,0]
Out[32]: 'Jason'
```

Fig.1(Own Sources)

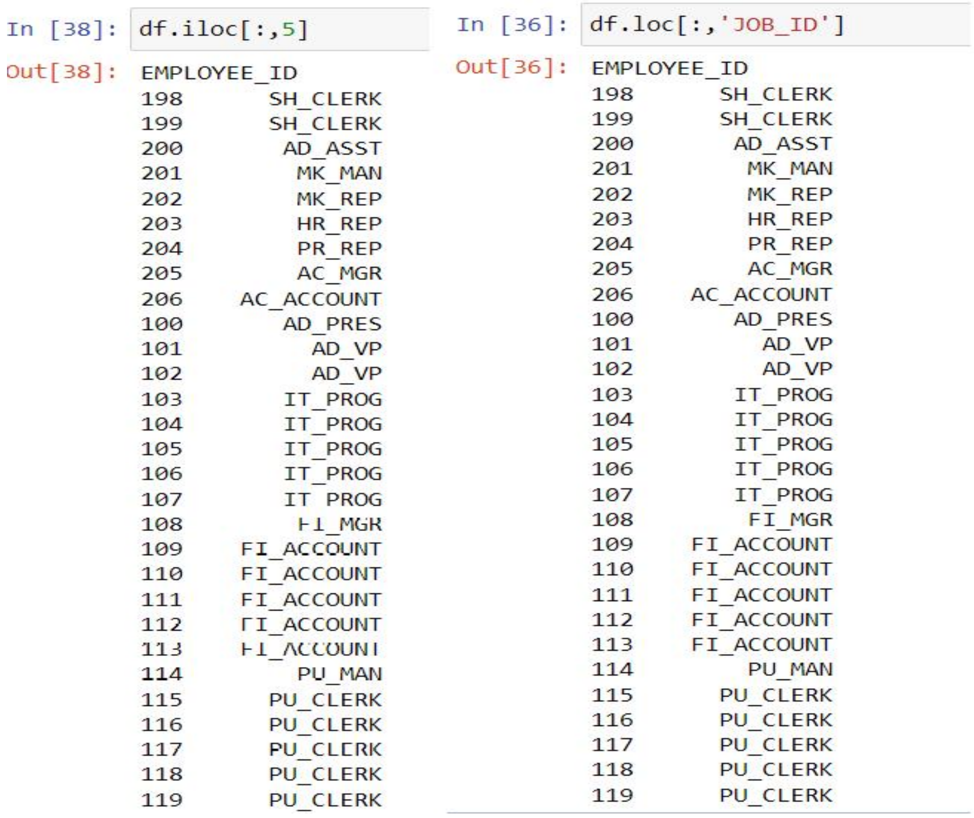
3.2 Select a single row:

To select single row from dataframe in both *'loc[ ]'* and *'iloc[ ]'* syntax :

loc[ ] : loc[:,row_label]

iloc[ ] : iloc[:,row_no]

Example



```
In [38]: df.iloc[:,5]
Out[38]: EMPLOYEE_ID
         198        SH_CLERK
         199        SH_CLERK
         200         AD_ASST
         201          MK_MAN
         202          MK_REP
         203          HR_REP
         204          PR_REP
         205          AC_MGR
         206      AC_ACCOUNT
         100         AD_PRES
         101           AD_VP
         102           AD_VP
         103         IT_PROG
         104         IT_PROG
         105         IT_PROG
         106         IT_PROG
         107         IT_PROG
         108          FI_MGR
         109      FI_ACCOUNT
         110      FI_ACCOUNT
         111      FI_ACCOUNT
         112      FI_ACCOUNT
         113      FI_ACCOUNT
         114          PU_MAN
         115        PU_CLERK
         116        PU_CLERK
         117        PU_CLERK
         118        PU_CLERK
         119        PU_CLERK
```

```
In [36]: df.loc[:,'JOB_ID']
Out[36]: EMPLOYEE_ID
         198        SH_CLERK
         199        SH_CLERK
         200         AD_ASST
         201          MK_MAN
         202          MK_REP
         203          HR_REP
         204          PR_REP
         205          AC_MGR
         206      AC_ACCOUNT
         100         AD_PRES
         101           AD_VP
         102           AD_VP
         103         IT_PROG
         104         IT_PROG
         105         IT_PROG
         106         IT_PROG
         107         IT_PROG
         108          FI_MGR
         109      FI_ACCOUNT
         110      FI_ACCOUNT
         111      FI_ACCOUNT
         112      FI_ACCOUNT
         113      FI_ACCOUNT
         114          PU_MAN
         115        PU_CLERK
         116        PU_CLERK
         117        PU_CLERK
         118        PU_CLERK
         119        PU_CLERK
```

Fig.2(OwnSource)

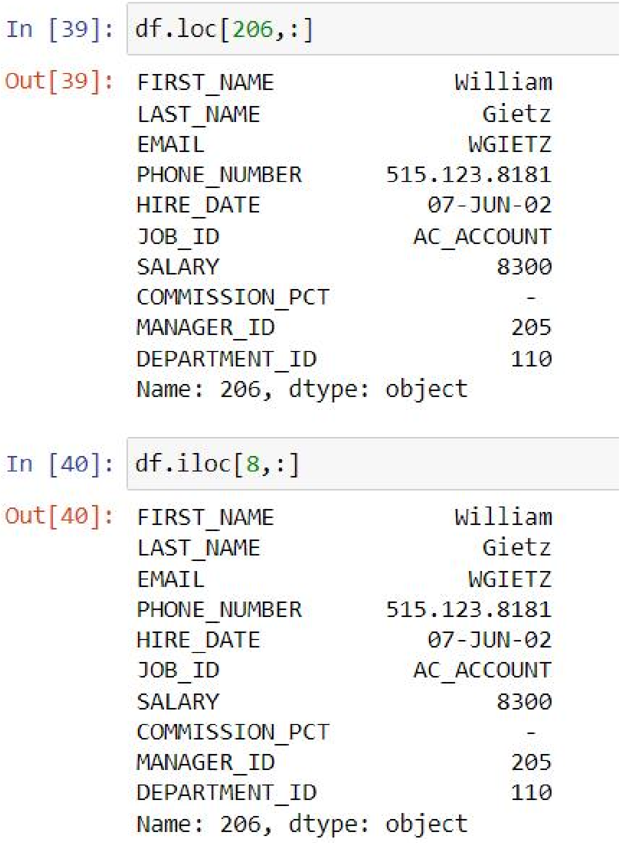
3.3 Select a single column:

To select single column from dataframe in both *'loc[ ]'* and *'iloc[ ]'* syntax :

loc[ ] : loc[row_label,:]

iloc[ ] : iloc[row_no,:]

Example :



```
In [39]:  df.loc[206,:]

Out[39]:  FIRST_NAME              William
          LAST_NAME                 Gietz
          EMAIL                    WGIETZ
          PHONE_NUMBER       515.123.8181
          HIRE_DATE             07-JUN-02
          JOB_ID               AC_ACCOUNT
          SALARY                     8300
          COMMISSION_PCT                -
          MANAGER_ID                  205
          DEPARTMENT_ID               110
          Name: 206, dtype: object

In [40]:  df.iloc[8,:]

Out[40]:  FIRST_NAME              William
          LAST_NAME                 Gietz
          EMAIL                    WGIETZ
          PHONE_NUMBER       515.123.8181
          HIRE_DATE             07-JUN-02
          JOB_ID               AC_ACCOUNT
          SALARY                     8300
          COMMISSION_PCT                -
          MANAGER_ID                  205
          DEPARTMENT_ID               110
          Name: 206, dtype: object
```

Fig.3(Own Source)

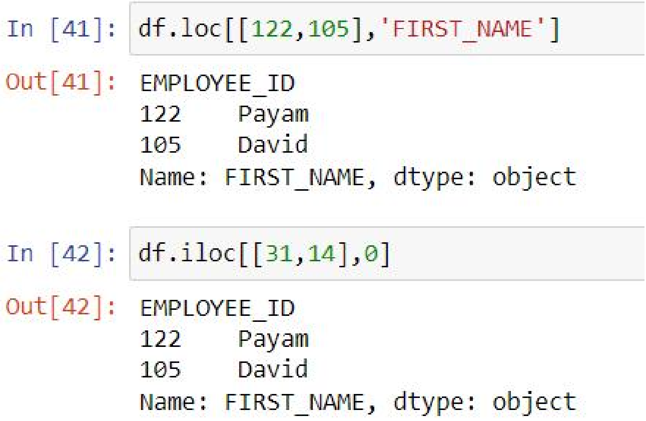
3.4 Select a multiple rows and columns:

To select multiple rows and columns from dataframe in both *'loc[ ]'* and *'iloc[ ]'* syntax :

loc[ ] : loc[row_labels, column_labels]

iloc[ ] : ilo[row_indices, column_indices]

Example :



```
In [41]:  df.loc[[122,105],'FIRST_NAME']

Out[41]:  EMPLOYEE_ID
          122    Payam
          105    David
          Name: FIRST_NAME, dtype: object

In [42]:  df.iloc[[31,14],0]

Out[42]:  EMPLOYEE_ID
          122    Payam
          105    David
          Name: FIRST_NAME, dtype: object
```

Fig.4(Own Source)

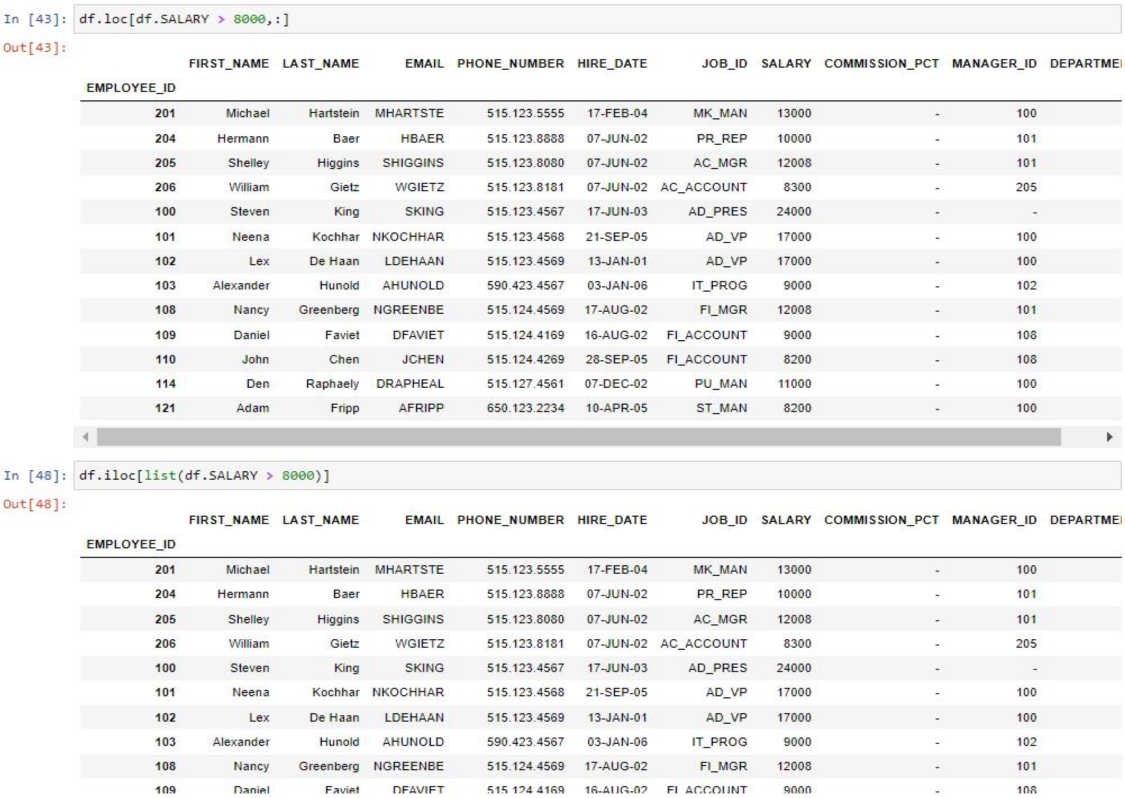
3.5 Using single Conditional Statements:

To select multiple rows and columns from dataframe using single conditional statement in both *'loc[ ]'* and *'iloc[ ]'* syntax :

loc[ ] : loc[boolean_array, :]

iloc[ ] : iloc[list(boolean_array)]

Example :



```
In [43]: df.loc[df.SALARY > 8000,:]
Out[43]:
```

	FIRST_NAME	LAST_NAME	EMAIL	PHONE_NUMBER	HIRE_DATE	JOB_ID	SALARY	COMMISSION_PCT	MANAGER_ID	DEPARTMEI
EMPLOYEE_ID										
201	Michael	Hartstein	MHARTSTE	515.123.5555	17-FEB-04	MK_MAN	13000	-	100	
204	Hermann	Baer	HBAER	515.123.8888	07-JUN-02	PR_REP	10000	-	101	
205	Shelley	Higgins	SHIGGINS	515.123.8080	07-JUN-02	AC_MGR	12008	-	101	
206	William	Gietz	WGIETZ	515.123.8181	07-JUN-02	AC_ACCOUNT	8300	-	205	
100	Steven	King	SKING	515.123.4567	17-JUN-03	AD_PRES	24000	-	-	
101	Neena	Kochhar	NKOCHHAR	515.123.4568	21-SEP-05	AD_VP	17000	-	100	
102	Lex	De Haan	LDEHAAN	515.123.4569	13-JAN-01	AD_VP	17000	-	100	
103	Alexander	Hunold	AHUNOLD	590.423.4567	03-JAN-06	IT_PROG	9000	-	102	
108	Nancy	Greenberg	NGREENBE	515.124.4569	17-AUG-02	FI_MGR	12008	-	101	
109	Daniel	Faviet	DFAVIET	515.124.4169	16-AUG-02	FI_ACCOUNT	9000	-	108	
110	John	Chen	JCHEN	515.124.4269	28-SEP-05	FI_ACCOUNT	8200	-	108	
114	Den	Raphaely	DRAPHEAL	515.127.4561	07-DEC-02	PU_MAN	11000	-	100	
121	Adam	Fripp	AFRIPP	650.123.2234	10-APR-05	ST_MAN	8200	-	100	

```
In [48]: df.iloc[list(df.SALARY > 8000)]
Out[48]:
```

	FIRST_NAME	LAST_NAME	EMAIL	PHONE_NUMBER	HIRE_DATE	JOB_ID	SALARY	COMMISSION_PCT	MANAGER_ID	DEPARTMEI
EMPLOYEE_ID										
201	Michael	Hartstein	MHARTSTE	515.123.5555	17-FEB-04	MK_MAN	13000	-	100	
204	Hermann	Baer	HBAER	515.123.8888	07-JUN-02	PR_REP	10000	-	101	
205	Shelley	Higgins	SHIGGINS	515.123.8080	07-JUN-02	AC_MGR	12008	-	101	
206	William	Gietz	WGIETZ	515.123.8181	07-JUN-02	AC_ACCOUNT	8300	-	205	
100	Steven	King	SKING	515.123.4567	17-JUN-03	AD_PRES	24000	-	-	
101	Neena	Kochhar	NKOCHHAR	515.123.4568	21-SEP-05	AD_VP	17000	-	100	
102	Lex	De Haan	LDEHAAN	515.123.4569	13-JAN-01	AD_VP	17000	-	100	
103	Alexander	Hunold	AHUNOLD	590.423.4567	03-JAN-06	IT_PROG	9000	-	102	
108	Nancy	Greenberg	NGREENBE	515.124.4569	17-AUG-02	FI_MGR	12008	-	101	
109	Daniel	Faviet	DFAVIET	515.124.4169	16-AUG-02	FI_ACCOUNT	9000	-	108	

Fig.5(Own source)

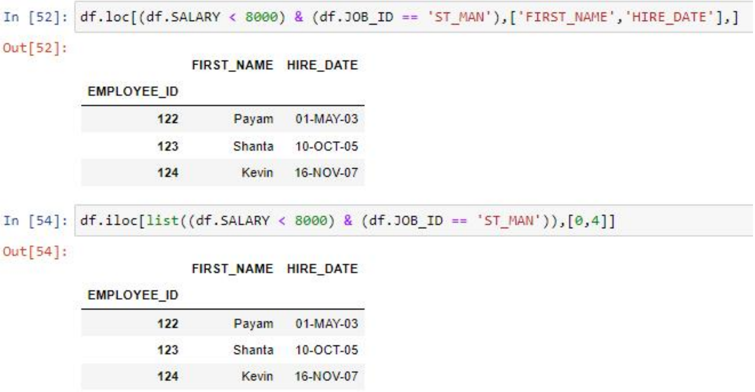
3.6 Using multiple conditional statements:

To select multiple rows and columns from dataframe using multiple conditional statement in both *'loc[ ]'* and *'iloc[ ]'* syntax :

loc[ ] : loc[(boolean_array) & (boolean_array) :]

iloc[ ] : iloc[list((boolean_array)&(boolean_array),:]

Example :



```
In [52]: df.loc[(df.SALARY < 8000) & (df.JOB_ID == 'ST_MAN'),['FIRST_NAME','HIRE_DATE'],]
Out[52]:
```

	FIRST_NAME	HIRE_DATE
EMPLOYEE_ID		
122	Payam	01-MAY-03
123	Shanta	10-OCT-05
124	Kevin	16-NOV-07

```
In [54]: df.iloc[list((df.SALARY < 8000) & (df.JOB_ID == 'ST_MAN')),[0,4]]
Out[54]:
```

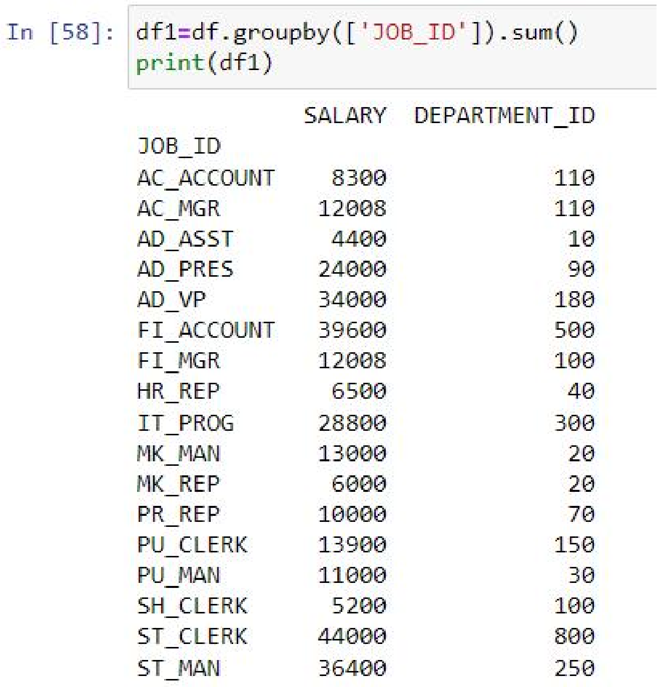
	FIRST_NAME	HIRE_DATE
EMPLOYEE_ID		
122	Payam	01-MAY-03
123	Shanta	10-OCT-05
124	Kevin	16-NOV-07

Fig.6(Own Source)

3.7 *groupby()* method in *'loc[ ]'* and *'iloc[ ]'*

In Python, *'loc[ ]'* and *'iloc[ ]'* are used in groupby functions to group data in a specific format based on conditions. It is also known as the itertools module since it organises data in a recursive manner depending on the key function.Methods such as 'groupby' can be applied to single or many columns in a dataframe.

Example:



```
In [58]: df1=df.groupby(['JOB_ID']).sum()
         print(df1)
```

```
            SALARY  DEPARTMENT_ID
JOB_ID
AC_ACCOUNT    8300            110
AC_MGR       12008            110
AD_ASST       4400             10
AD_PRES      24000             90
AD_VP        34000            180
FI_ACCOUNT   39600            500
FI_MGR       12008            100
HR_REP        6500             40
IT_PROG      28800            300
MK_MAN       13000             20
MK_REP        6000             20
PR_REP       10000             70
PU_CLERK     13900            150
PU_MAN       11000             30
SH_CLERK      5200            100
ST_CLERK     44000            800
ST_MAN       36400            250
```

Fig.7(Own Source)

4.Differences between *'loc[ ]'* and *'iloc[ ]'*

The difference between 'loc[ ]' and 'iloc[ ]' is used to extract the data in rows or columns from a dataframe. The differences of 'loc[ ]' and 'iloc[ ]' are :

	loc[ ]	iloc[ ]
Indexing	Label Based Indexing	Integer Based indexing
Datatypes of index	Integers,strings,timestamps	Integers
Slicing	Can be slice both rows and columns	Can be slice both rows and columns by integer positions
Arguments	Rows and columns labels of 2 arguments	Rows and columns indices of 2 arguments
Inclusive	It includes both start and end value of range	It exclude only end value.
Boolean Indexing	Uses boolean indexing based on conditions	It doesn't use boolean indexing
Element	Last element of the table	Does not include the last element

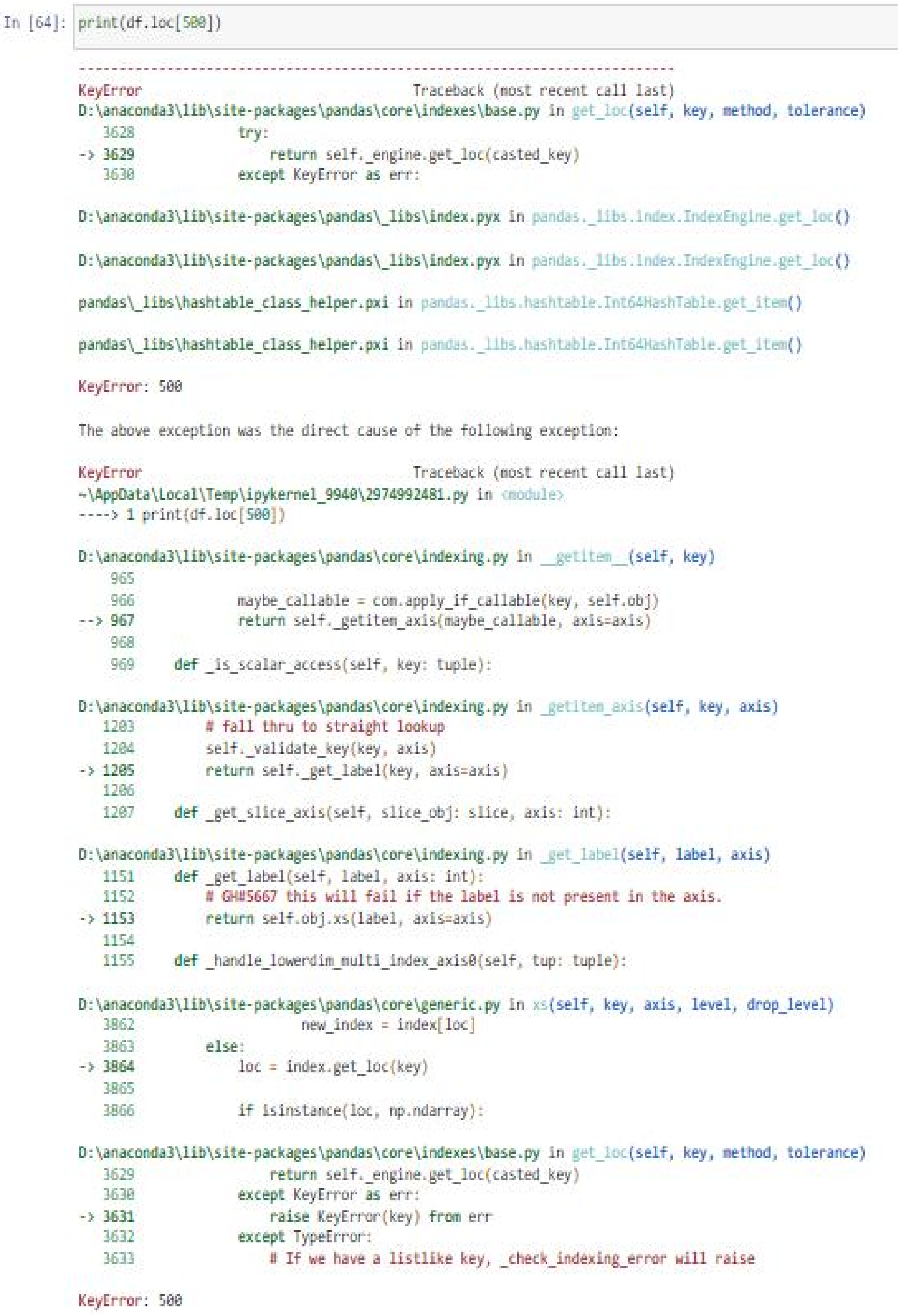
5. Errors in 'loc[ ]' and 'iloc[ ]' [ ]

In python when we use 'loc[ ]' and 'iloc[ ]' some common error occurs, they are :

5.1 Key Error :

When we access the row and column label which is not existing in the dataframe.

Example :



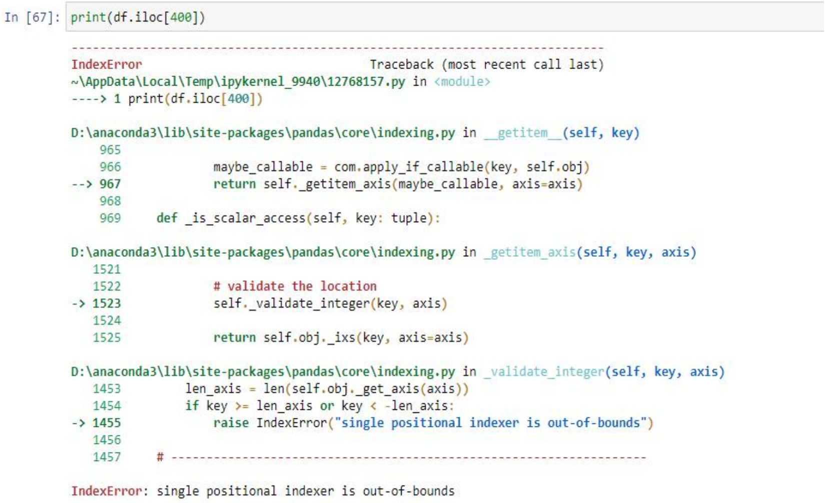
```
In [64]: print(df.loc[500])

---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
D:\anaconda3\lib\site-packages\pandas\core\indexes\base.py in get_loc(self, key, method, tolerance)
   3628             try:
-> 3629                 return self._engine.get_loc(casted_key)
   3630             except KeyError as err:

D:\anaconda3\lib\site-packages\pandas\_libs\index.pyx in pandas._libs.index.IndexEngine.get_loc()

D:\anaconda3\lib\site-packages\pandas\_libs\index.pyx in pandas._libs.index.IndexEngine.get_loc()

pandas\_libs\hashtable_class_helper.pxi in pandas._libs.hashtable.Int64HashTable.get_item()

pandas\_libs\hashtable_class_helper.pxi in pandas._libs.hashtable.Int64HashTable.get_item()

KeyError: 500

The above exception was the direct cause of the following exception:

KeyError                                  Traceback (most recent call last)
~\AppData\Local\Temp\ipykernel_9940\2974992481.py in <module>
----> 1 print(df.loc[500])

D:\anaconda3\lib\site-packages\pandas\core\indexing.py in __getitem__(self, key)
    965
    966             maybe_callable = com.apply_if_callable(key, self.obj)
--> 967             return self._getitem_axis(maybe_callable, axis=axis)
    968
    969     def _is_scalar_access(self, key: tuple):

D:\anaconda3\lib\site-packages\pandas\core\indexing.py in _getitem_axis(self, key, axis)
   1203         # fall thru to straight lookup
   1204         self._validate_key(key, axis)
-> 1205         return self._get_label(key, axis=axis)
   1206
   1207     def _get_slice_axis(self, slice_obj: slice, axis: int):

D:\anaconda3\lib\site-packages\pandas\core\indexing.py in _get_label(self, label, axis)
   1151     def _get_label(self, label, axis: int):
   1152         # GH#5667 this will fail if the label is not present in the axis.
-> 1153         return self.obj.xs(label, axis=axis)
   1154
   1155     def _handle_lowerdim_multi_index_axis0(self, tup: tuple):

D:\anaconda3\lib\site-packages\pandas\core\generic.py in xs(self, key, axis, level, drop_level)
   3862                 new_index = index[loc]
   3863         else:
-> 3864             loc = index.get_loc(key)
   3865
   3866             if isinstance(loc, np.ndarray):

D:\anaconda3\lib\site-packages\pandas\core\indexes\base.py in get_loc(self, key, method, tolerance)
   3629                 return self._engine.get_loc(casted_key)
   3630             except KeyError as err:
-> 3631                 raise KeyError(key) from err
   3632             except TypeError:
   3633                 # If we have a listlike key, _check_indexing_error will raise

KeyError: 500
```

Fig.8(Own Source)

5.2 Index Error:

When we access the row and column indexing in integer which is not existing in the dataframe.

Example:



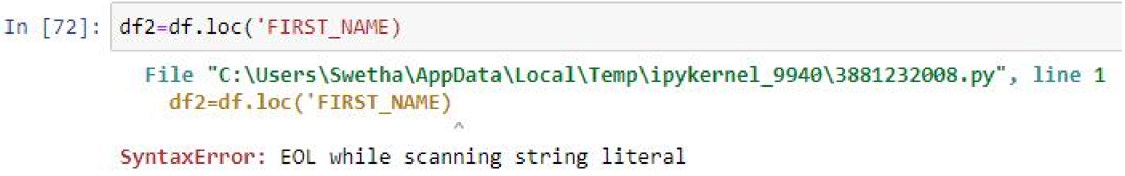
```
In [67]: print(df.iloc[400])

---------------------------------------------------------------------------
IndexError                                Traceback (most recent call last)
~\AppData\Local\Temp\ipykernel_9940\12768157.py in <module>
----> 1 print(df.iloc[400])

D:\anaconda3\lib\site-packages\pandas\core\indexing.py in __getitem__(self, key)
    965
    966             maybe_callable = com.apply_if_callable(key, self.obj)
--> 967             return self._getitem_axis(maybe_callable, axis=axis)
    968
    969     def _is_scalar_access(self, key: tuple):

D:\anaconda3\lib\site-packages\pandas\core\indexing.py in _getitem_axis(self, key, axis)
   1521
   1522             # validate the location
-> 1523             self._validate_integer(key, axis)
   1524
   1525             return self.obj._ixs(key, axis=axis)

D:\anaconda3\lib\site-packages\pandas\core\indexing.py in _validate_integer(self, key, axis)
   1453         len_axis = len(self.obj._get_axis(axis))
   1454         if key >= len_axis or key < -len_axis:
-> 1455             raise IndexError("single positional indexer is out-of-bounds")
   1456
   1457     # -------------------------------------------------------------------

IndexError: single positional indexer is out-of-bounds
```

Fig.9(Own Source)

5.3 Syntax Error:

This issue arises when the syntax is incorrect, for example, instead of rows, we mention columns, or in 'iloc[ ]', we mention strings instead of integers, or we forget to declare some parameters, and so on.

Example :



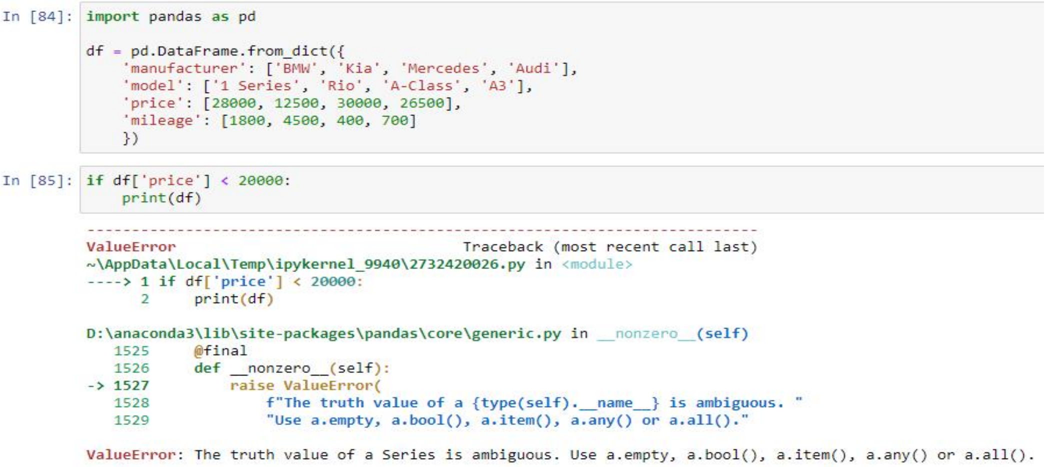
```
In [72]: df2=df.loc('FIRST_NAME)

  File "C:\Users\Swetha\AppData\Local\Temp\ipykernel_9940\3881232008.py", line 1
    df2=df.loc('FIRST_NAME)
                           ^
SyntaxError: EOL while scanning string literal
```

Fig.10(Own Source)

5.4 Ambiguity Error:

This error occur when the dataframe labels or index are not unique.

Example:



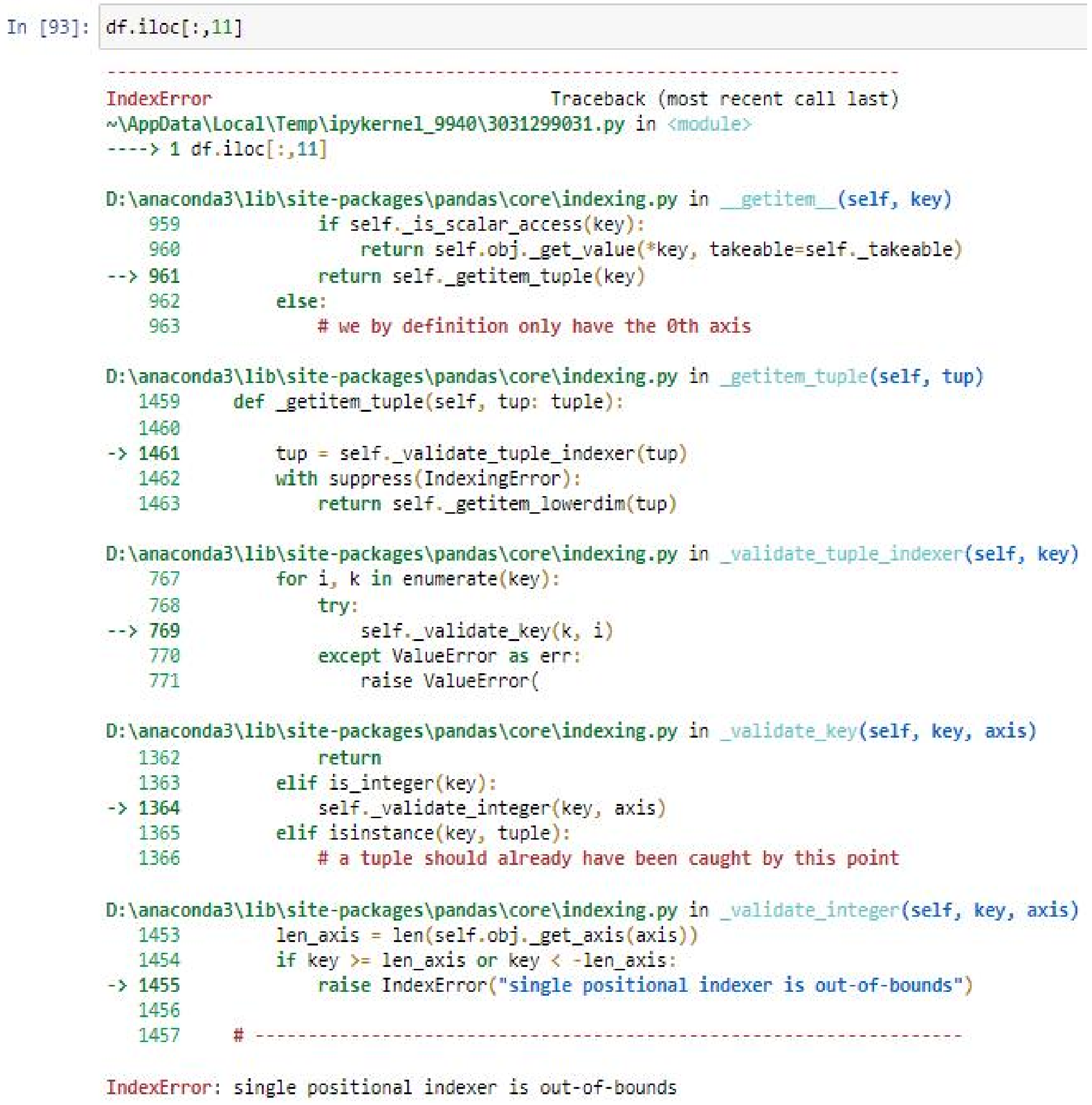
```
In [84]: import pandas as pd

         df = pd.DataFrame.from_dict({
             'manufacturer': ['BMW', 'Kia', 'Mercedes', 'Audi'],
             'model': ['1 Series', 'Rio', 'A-Class', 'A3'],
             'price': [28000, 12500, 30000, 26500],
             'mileage': [1800, 4500, 400, 700]
             })

In [85]: if df['price'] < 20000:
             print(df)

---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
~\AppData\Local\Temp\ipykernel_9940\2732420026.py in <module>
----> 1 if df['price'] < 20000:
      2     print(df)

D:\anaconda3\lib\site-packages\pandas\core\generic.py in __nonzero__(self)
   1525     @final
   1526     def __nonzero__(self):
-> 1527         raise ValueError(
   1528             f"The truth value of a {type(self).__name__} is ambiguous. "
   1529             "Use a.empty, a.bool(), a.item(), a.any() or a.all()."

ValueError: The truth value of a Series is ambiguous. Use a.empty, a.bool(), a.item(), a.any() or a.all().
```

Fig.11(Own Source)

5.5 Out-of-Bound:

This error occur when the row and columns of index is not define in dataframe range.

Example :



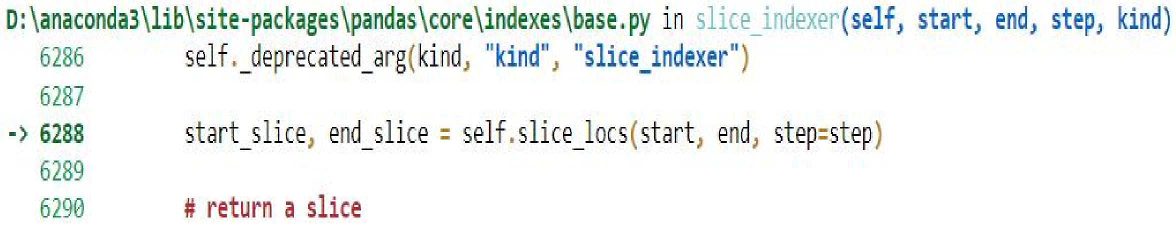
```
In [93]: df.iloc[:,11]

---------------------------------------------------------------------------
IndexError                                Traceback (most recent call last)
~\AppData\Local\Temp\ipykernel_9940\3031299031.py in <module>
----> 1 df.iloc[:,11]

D:\anaconda3\lib\site-packages\pandas\core\indexing.py in __getitem__(self, key)
    959             if self._is_scalar_access(key):
    960                 return self.obj._get_value(*key, takeable=self._takeable)
--> 961             return self._getitem_tuple(key)
    962         else:
    963             # we by definition only have the 0th axis

D:\anaconda3\lib\site-packages\pandas\core\indexing.py in _getitem_tuple(self, tup)
   1459     def _getitem_tuple(self, tup: tuple):
   1460
-> 1461         tup = self._validate_tuple_indexer(tup)
   1462         with suppress(IndexingError):
   1463             return self._getitem_lowerdim(tup)

D:\anaconda3\lib\site-packages\pandas\core\indexing.py in _validate_tuple_indexer(self, key)
    767         for i, k in enumerate(key):
    768             try:
--> 769                 self._validate_key(k, i)
    770             except ValueError as err:
    771                 raise ValueError(

D:\anaconda3\lib\site-packages\pandas\core\indexing.py in _validate_key(self, key, axis)
   1362             return
   1363         elif is_integer(key):
-> 1364             self._validate_integer(key, axis)
   1365         elif isinstance(key, tuple):
   1366             # a tuple should already have been caught by this point

D:\anaconda3\lib\site-packages\pandas\core\indexing.py in _validate_integer(self, key, axis)
   1453         len_axis = len(self.obj._get_axis(axis))
   1454         if key >= len_axis or key < -len_axis:
-> 1455             raise IndexError("single positional indexer is out-of-bounds")
   1456
   1457     # -------------------------------------------------------------------

IndexError: single positional indexer is out-of-bounds
```

Fig.12(Own Source)

5.6 Slicing Syntax Error

This error occur when the slicing syntax is incorrect or incomplete.

Example:



```
D:\anaconda3\lib\site-packages\pandas\core\indexes\base.py in slice_indexer(self, start, end, step, kind)
   6286         self._deprecated_arg(kind, "kind", "slice_indexer")
   6287
-> 6288         start_slice, end_slice = self.slice_locs(start, end, step=step)
   6289
   6290         # return a slice
```

Fig.13(Own Source)

6. Avoid errors in 'loc[ ]' and 'iloc[ ]'

'loc[ ]' and 'iloc[ ]' are used to analyse data in a certain format as specified by the user. While utilising 'loc[ ]' and 'iloc[ ]', problems arise during programme execution. To avoid this error, we need follow some guidelines.

6.1 Use of lables :

When using ''loc[ ]' ', specify the precise label name from the dataframe for slicing, selecting, or sorting the data.

6.2 Use of integer position:

When using ''iloc[ ]'', specify the precise integer position from the dataframe for slicing, selecting or sorting the data. For integer position it starts from '0' value.

6.3 Mention the correct end value :

While mentioning the end value for slicing for both 'loc[ ]' and 'iloc[ ]' user should keep in mind that in ''loc[ ]'' end value is included and for ''iloc[ ]'' end value is not mentioned

6.4 Index in Dataframe :

While creating a file the user should set the index properly so that when ''loc[ ]'' and ''iloc[ ]'' is used the position should be correct.

6.5 Null Values:

Labels and indexes in dataframe files should not be null, labels should have a unique name, and users should be able to access the label within the dataframe.

6.6 Syntax Error :

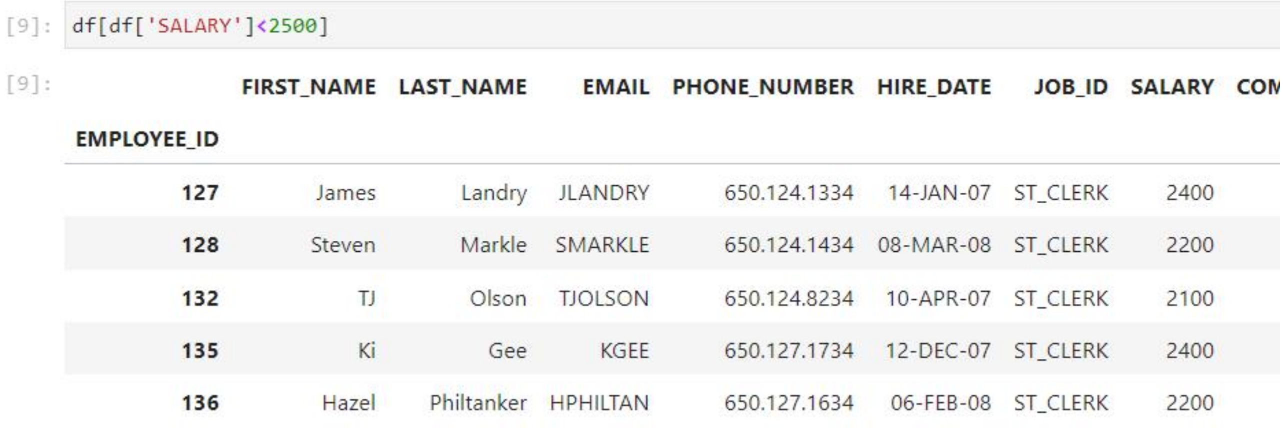
Both ''loc[ ]'' and ''iloc[ ]'' have a different syntax, while using syntax user should give the correct syntax for ''loc[ ]'' in label-based and ''iloc[ ]'' in integer-based

7. Alternative for 'loc[ ]' and 'iloc[ ]'

There is an alternative for 'loc[ ]' and 'iloc[ ]' user can use the boolean indexing and integer indexing to select the specific row and column of a dataframe.

7.1 Boolean indexing:

Example: df[df[‘SALARY’]<2500]



```
[9]: df[df['SALARY']<2500]
```

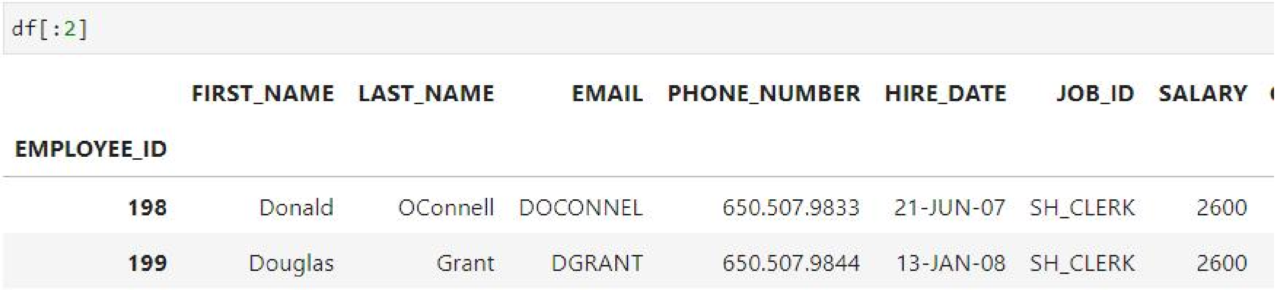
[9]:

	FIRST_NAME	LAST_NAME	EMAIL	PHONE_NUMBER	HIRE_DATE	JOB_ID	SALARY	COM
EMPLOYEE_ID								
127	James	Landry	JLANDRY	650.124.1334	14-JAN-07	ST_CLERK	2400	
128	Steven	Markle	SMARKLE	650.124.1434	08-MAR-08	ST_CLERK	2200	
132	TJ	Olson	TJOLSON	650.124.8234	10-APR-07	ST_CLERK	2100	
135	Ki	Gee	KGEE	650.127.1734	12-DEC-07	ST_CLERK	2400	
136	Hazel	Philtanker	HPHILTAN	650.127.1634	06-FEB-08	ST_CLERK	2200	

Fig.14

7.2 Integer Indexing:

Example: df[:2]



```
df[:2]
```

	FIRST_NAME	LAST_NAME	EMAIL	PHONE_NUMBER	HIRE_DATE	JOB_ID	SALARY
EMPLOYEE_ID							
198	Donald	OConnell	DOCONNEL	650.507.9833	21-JUN-07	SH_CLERK	2600
199	Douglas	Grant	DGRANT	650.507.9844	13-JAN-08	SH_CLERK	2600

Fig.15

8.Conclusion

This assignment introduces the 'loc[ ]' and 'iloc[ ]' [] methods in Python, which are two fundamental pandas properties that enable for label-based and integer-based indexing and data selection from a pandas dataframe, respectively. Also it focuses on 'loc[ ]' [] and 'iloc[ ]' [] in pandas and covers their introduction, usages, differences, slice and select data, mistakes, and how to utilize in pandas, for all topics it is explained with examples for usages, errors, slice and select data so that it can be understand easily. 'loc[ ]' and 'iloc[ ]' are used for data manipulating, sorting, selecting, filtering, merging multiple dataframes, slicing in data analysis, data science, algorithm, machine learning, finance etc. Also it explore how to use 'loc[ ]' [] and 'iloc[ ]' [] with other pandas methods like ‘groupby’ , ‘pivot table’ , ‘filtering’ , ‘selecting’ and ‘sorting’ to manipulate and analyse the data in a complex manner. 'loc[ ]' and 'iloc[ ]' method is a easy to use, fast, and quick to use.

References

Thomas Bressoud, David White(2020) Introduction to Data Systems: Building from Python - Page 193- *https://www.google.co.in/books/edition/Introduction_to_Data_Systems/4P0MEAAAQBAJ?hl=en&gbpv=0#:~:text=Got%20it ,Introduction%20to%20Data%20Systems,Building%20from%20Python,-By%C2%A0Thomas*

Alan Bernardo Palacio · 2021-Distributed Data Systems with Azure Databricks - Page 218 *https://books.google.co.in/books?id=RCAwEAAAQBAJ&pg=PA218&dq=how+to+avoid+loc+and+iloc+errors&hl=en&newbks=1&newbks_redir=1&sa=X&ved=2ahUKEwiJiujXmfz9AhUcT2wGHeYADiA4ChDoAXoECAsQAg*

Blaine Bateman, Saikat Basak, Thomas V. Joseph · 2022- The Pandas Workshop: A comprehensive guide to using Python-Page 244 *https://books.google.co.in/books?id=ezByEAAAQBAJ&pg=PA244&dq=alternative+of+loc+and+iloc&hl=en&newbks=1&newbks_redir=1&sa=X&ved=2ahUKEwjDwqmtm_z9AhWzTmwGHTLbBrcQ6AF6BAgGEAI*

Wes McKinney(2017) - Python for Data Analysis *https://books.google.co.in/books?id=UiM3DwAAQBAJ&printsec=frontcover&dq=loc[ ]+and+iloc[ ]+usages+in+which+programming+language&hl=en&newbks=1&newbks_redir=1&sa=X&ved=2ahUKEwiMn8DL3fn9AhVlV2wGHX5vBskQ6AF6BAgGEAI*

Jake VanderPlas - 2016 - Python Data Science Handbook : Essential Tools for working *https://books.google.co.in/books?id=6omNDQAAQBAJ&printsec=frontcover&dq=difference+between+loc[ ]+and+iloc[ ]&hl=en&newbks=1&newbks_redir=1&sa=X&ved=2ahUKEwiBpqXHzvn9AhUkV2wGHfghA88Q6AF6BAgFEAI*

https://www.geeksforgeeks.org/python-extracting-rows-using-pandas-iloc[ ]/

https://towardsdatascience.com/how-to-use-loc[ ]-and-iloc[ ]-for-selecting-data-in-pandas bd09cb4c3d79

https://www.statology.org/pandas-loc[ ]-vs-iloc[ ]/

https://www.analyticsvidhya.com/blog/2020/02/loc[ ]-iloc[ ]-pandas/

https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.loc[ ].html

https://www.w3schools.com/python/pandas/ref_df_loc[ ].asp

https://www.geeksforgeeks.org/how-to-combine-two-dataframe-in-python-pandas/

https://thispointer.com/how-to-drop-index-column-of-a-pandas-dataframe/

https://www.youtube.com/watch?v=0MvYbrKMCKA

https://www.youtube.com/watch?v=Zs_C41bSw3o

https://www.youtube.com/watch?v=bmx1XWFHWDg

https://www.youtube.com/watch?v=QrYPcO9VQDQ

https://www.datacamp.com/tutorial/python-select-columns https://www.geeksforgeeks.org/how-to-drop-one-or-multiple-columns-in-pandas-dataframe/

https://www.google.co.in/books/edition/Introduction_to_Data_Systems/4P0MEAAAQBAJ?hl=en&gbpv=1&dq=difference+between+loc[ ]+and+iloc[ ]+in+pandas+with+example&pg=PA193&printsec=frontcover

https://www.google.co.in/books/edition/Oswaal_CBSE_Chapterwise_Topicwise_Questi/fsCoEAAAQBAJ?hl=en&gbpv=1&dq=loc[ ]+and+iloc[ ]+in+pandas&pg=RA1-PA14&printsec=frontcover

https://vitalflux.com/pandas-dataframe-loc[ ]-iloc[ ]-brackets-examples/

The Application used in this project:-

- Anaconda - Jupyter Notebook
- Mysql - sqlalchemy
- Bokeh for Visualization
- SQLAlchemy
- Libraries(Pandas,numpy,bokeh

All the files for Mysql are uploaded from Jupyter Notebook and visualized in both Jupyter Notebook & Mysql.

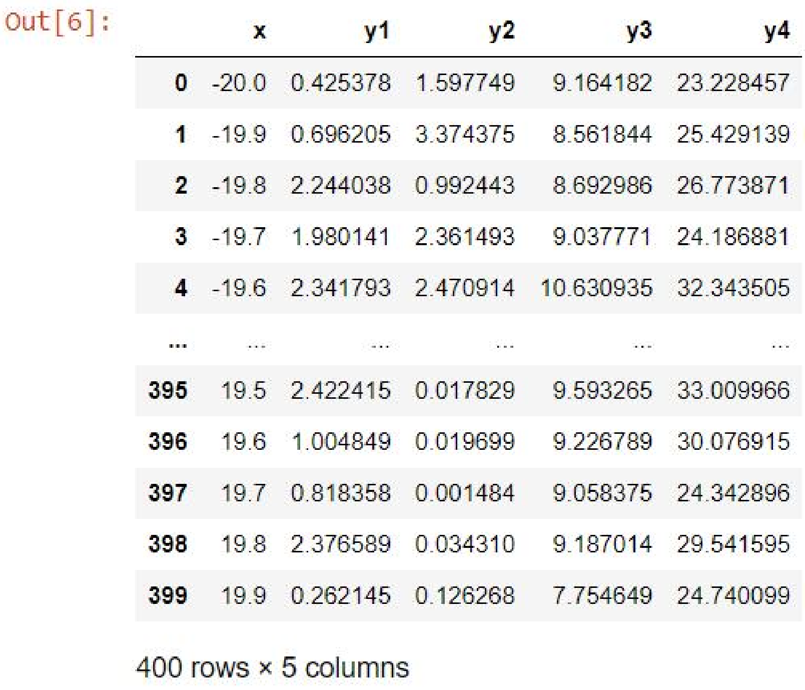
Training Data_set(1x, 4y)

Step 1:

Y values squared

```
import numpy as np
import pandas as pd
'''Training data set imported '''
trn_set=pd.read_csv('C:/Users/Swetha/IU/train.csv')
'''Printing x Values as it is'''
trn_cal=trn_set.'iloc[ ]'[0:, :1]
'''Calculating y values in squares'''
trn_cal1=trn_set.'iloc[ ]'[0:, 1:5]**2
'''concatenating both x values and squared y values'''
trn_squ=pd.concat([trn_cal,trn_cal1],axis=1,join='inner')
trn_squ
```

OUTPUT:



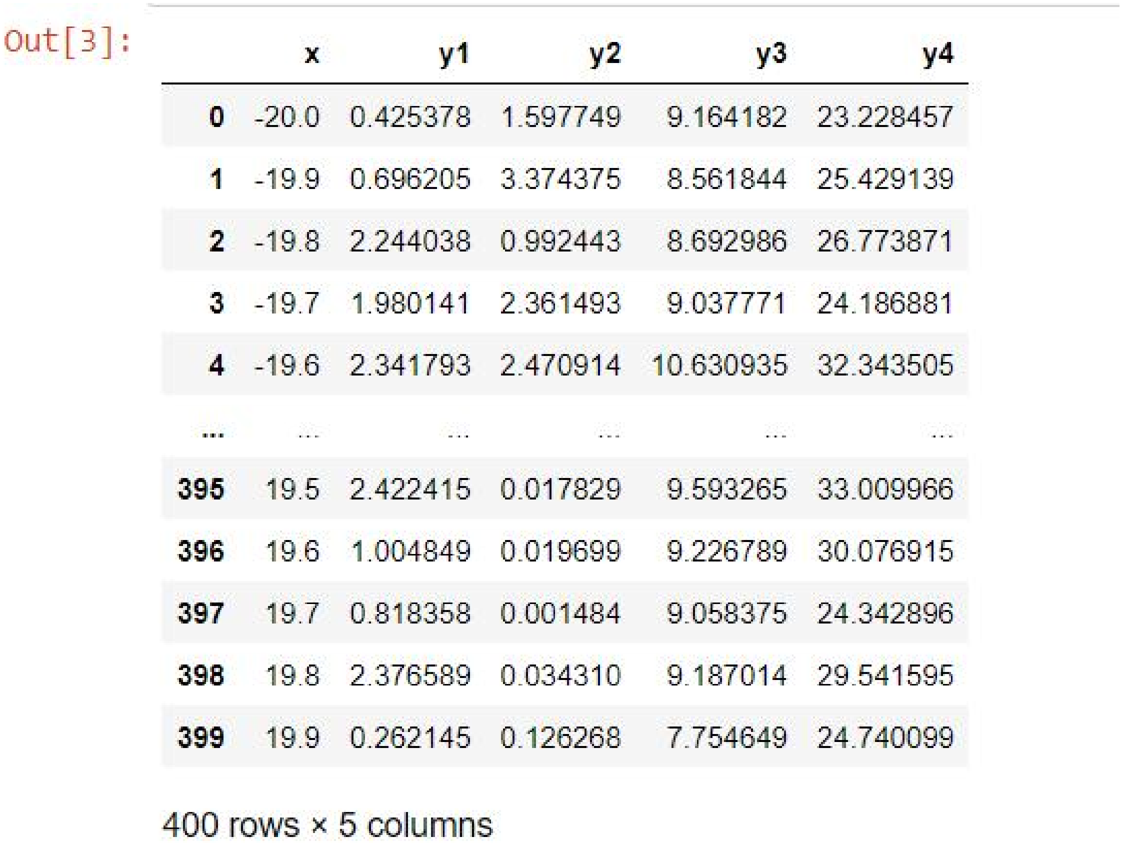
```
Out[6]:
           x        y1        y2         y3         y4
   0   -20.0  0.425378  1.597749   9.164182  23.228457
   1   -19.9  0.696205  3.374375   8.561844  25.429139
   2   -19.8  2.244038  0.992443   8.692986  26.773871
   3   -19.7  1.980141  2.361493   9.037771  24.186881
   4   -19.6  2.341793  2.470914  10.630935  32.343505
 ...     ...       ...       ...        ...        ...
 395    19.5  2.422415  0.017829   9.593265  33.009966
 396    19.6  1.004849  0.019699   9.226789  30.076915
 397    19.7  0.818358  0.001484   9.058375  24.342896
 398    19.8  2.376589  0.034310   9.187014  29.541595
 399    19.9  0.262145  0.126268   7.754649  24.740099

400 rows × 5 columns
```

Step 2:

Creating the Dataframe to Save the CSV File in Jupyter

```
df=pd.DataFrame(trn_squ)
'''dropping the index column and save to csv file'''
df.to_csv('C:/Users/Swetha/IU/tr_update.csv', index=False)
df
```

OUTPUT:



Out[3]:

	x	y1	y2	y3	y4
0	-20.0	0.425378	1.597749	9.164182	23.228457
1	-19.9	0.696205	3.374375	8.561844	25.429139
2	-19.8	2.244038	0.992443	8.692986	26.773871
3	-19.7	1.980141	2.361493	9.037771	24.186881
4	-19.6	2.341793	2.470914	10.630935	32.343505
...	...	...	...	...	...
395	19.5	2.422415	0.017829	9.593265	33.009966
396	19.6	1.004849	0.019699	9.226789	30.076915
397	19.7	0.818358	0.001484	9.058375	24.342896
398	19.8	2.376589	0.034310	9.187014	29.541595
399	19.9	0.262145	0.126268	7.754649	24.740099

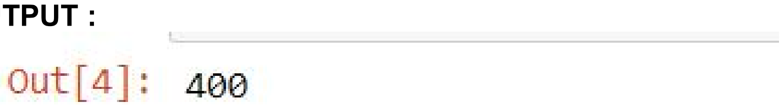
400 rows × 5 columns

Step 3:

Using sqlalchemy creating a DB file

```
'''Importing the modules'''
from sqlalchemy import create_engine as ce
import pandas as pd
'''Import data from CSV file'''
data=pd.read_csv('C:/Users/Swetha/IU/tr_update.csv')
'''Creating a DB file'''
file_db = ce('sqlite:///C:/Users/Swetha/assignment_dataset/tr_set.db')
'''Loading the CSV file into db file'''
data.to_sql('tr_update',file_db)
```

OUTPUT :



```
Out[4]: 400
```

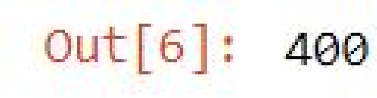
Step 4:

```
'''connecting to Mysql'''
mysql_engine = ce("mysql://root:mysql24S*@localhost:3306/trainsetdb")
```

Step 5:

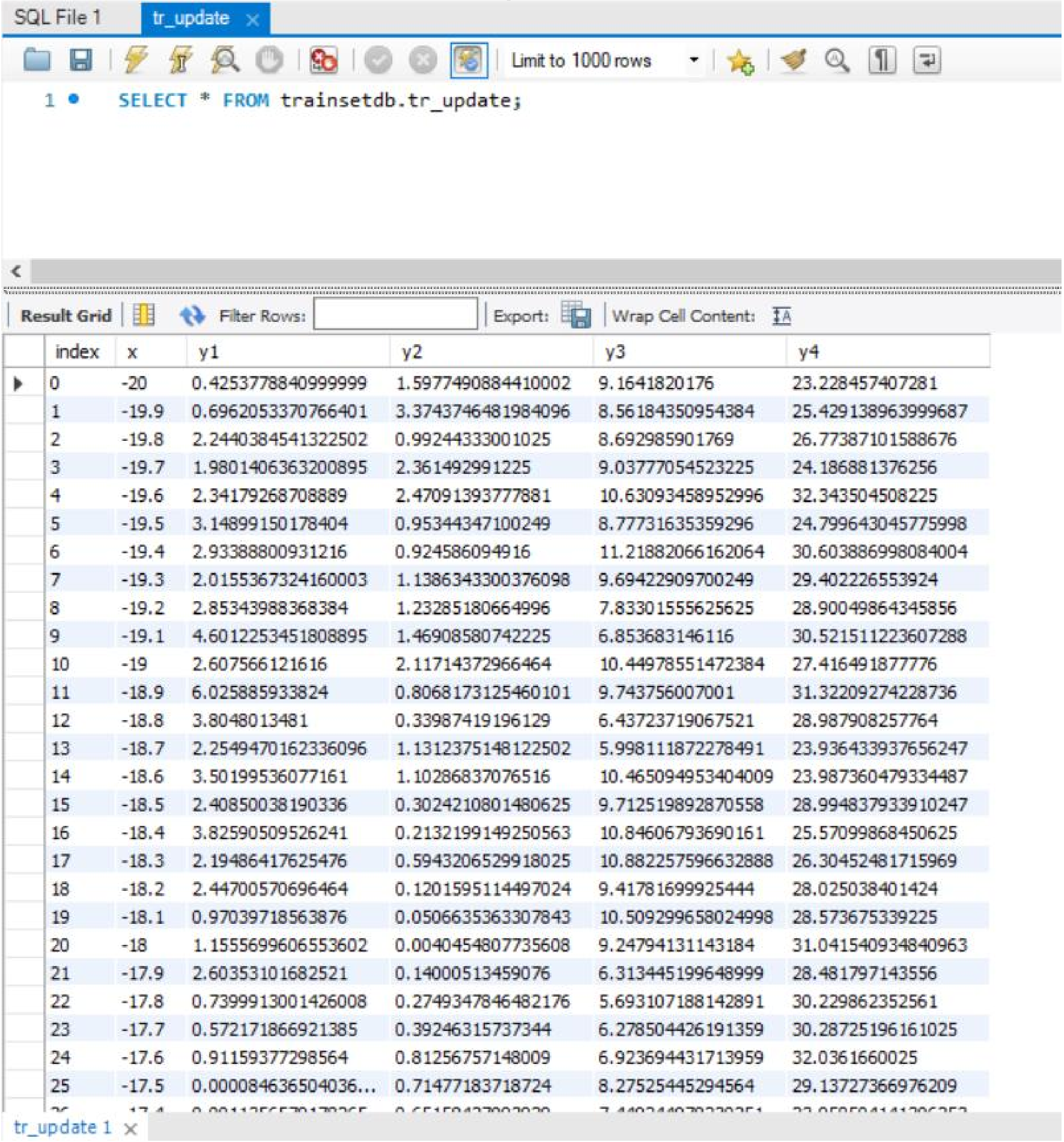
```
'''Loading CSV to Mysql'''
data.to_sql('tr_update',mysql_engine)
```

OUTPUT :



```
Out[6]: 400
```

Visualization in MYSQL - training data_set ‘Y’ Squared



```
SELECT * FROM trainsetdb.tr_update;
```

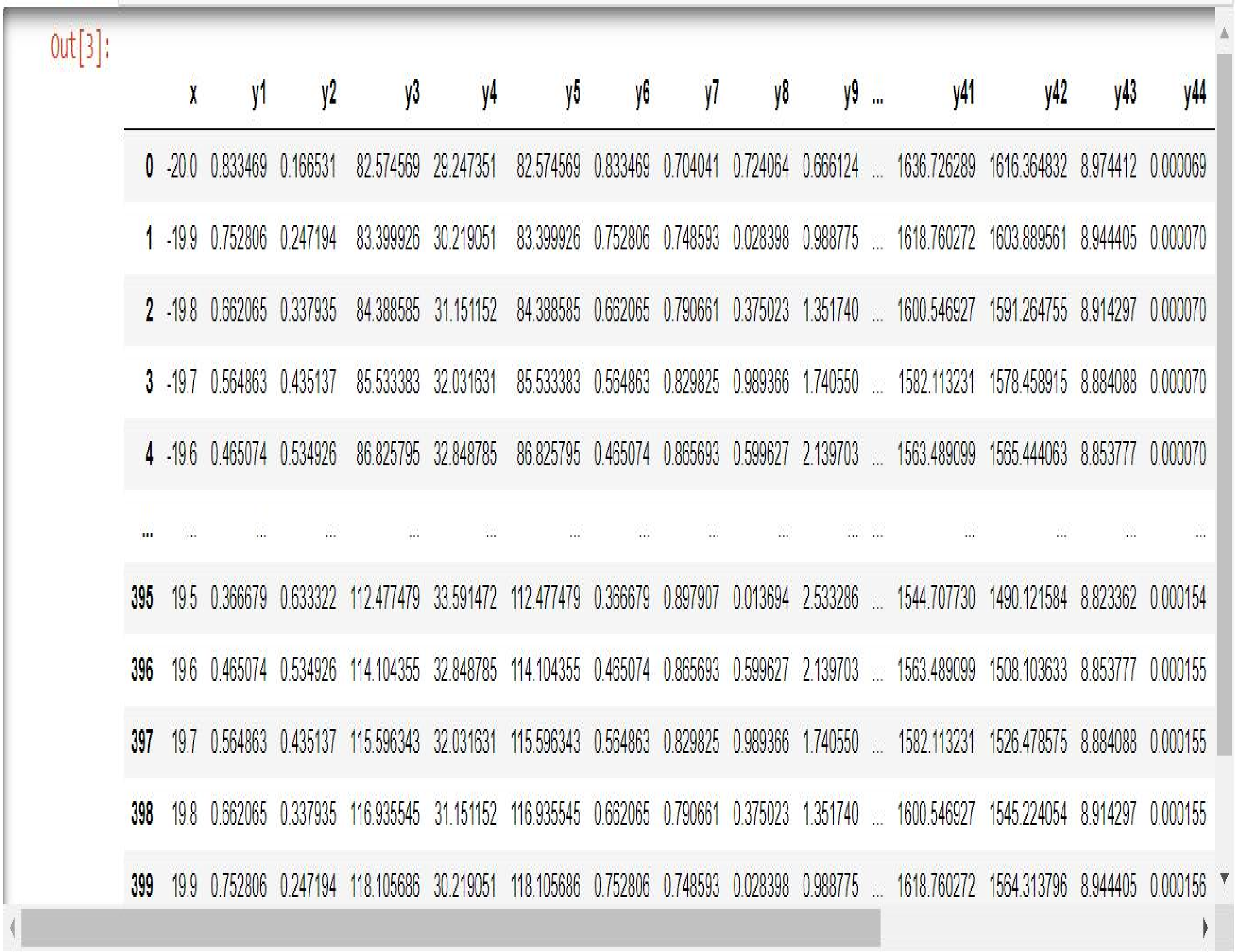
index	x	y1	y2	y3	y4
0	-20	0.4253778840999999	1.5977490884410002	9.1641820176	23.228457407281
1	-19.9	0.6962053370766401	3.3743746481984096	8.56184350954384	25.429138963999687
2	-19.8	2.2440384541322502	0.99244333001025	8.692985901769	26.7738710158676
3	-19.7	1.9801406363200895	2.361492991225	9.03777054523225	24.186881376256
4	-19.6	2.34179268708889	2.47091393777881	10.63093458952996	32.343504508225
5	-19.5	3.14899150178404	0.95344347100249	8.77731635359296	24.799643045775998
6	-19.4	2.93388800931216	0.924586094916	11.21882066162064	30.603886998084004
7	-19.3	2.0155367324160003	1.1386343300376098	9.69422909700249	29.402226553924
8	-19.2	2.85343988368384	1.23285180664996	7.8330155562 5625	28.90049864345856
9	-19.1	4.6012253451808895	1.46908580742225	6.853683146116	30.521511223607288
10	-19	2.607566121616	2.11714372966464	10.44978551472384	27.41649187776
11	-18.9	6.025885933824	0.8068173125460101	9.743756007001	31.32209274228736
12	-18.8	3.8048013481	0.33987419196129	6.43723719067521	28.987908257764
13	-18.7	2.2549470162336096	1.1312375148122502	5.998111872278491	23.936433937656247
14	-18.6	3.50199536077161	1.10286837076516	10.465094953404009	23.987360479334487
15	-18.5	2.40850038190336	0.3024210801480625	9.712519892870558	28.994837933910247
16	-18.4	3.82590509526241	0.2132199149250563	10.84606793690161	25.57099868450625
17	-18.3	2.19486417625476	0.5943206529918025	10.882257596632888	26.30452481715969
18	-18.2	2.44700570696464	0.1201595114497024	9.41781699925444	28.02503840142 4
19	-18.1	0.97039718563876	0.0506635363307843	10.509299658024998	28.573675339225
20	-18	1.1555699606553602	0.0040454807735608	9.24794131143184	31.041540934840963
21	-17.9	2.60353101682521	0.14000513459076	6.313445199648999	28.481797143556
22	-17.8	0.7399913001426008	0.2749347846482176	5.693107188142891	30.229862352561
23	-17.7	0.57217186692138 5	0.39246315737344	6.278504426191359	30.2872519616102 5
24	-17.6	0.91159377298564	0.81256757148009	6.923694431713959	32.0361660025
25	-17.5	0.00008463650403 6...	0.71477183718724	8.27525445294564	29.13727366976209

Ideal Data_set(1x, 50y)
Step 1:

Y values squared

```
import numpy as np
import pandas as pd
'''Training data set imported '''
ideal_set=pd.read_csv('C:/Users/Swetha/IU/ideal.csv')
'''Printing x Values as it is'''
ideal_cal=ideal_set.'iloc[ ]'[0:, :1]
'''Calculating y values in squares'''
ideal_cal1=ideal_set.'iloc[ ]'[0:, 1:51]**2
'''concatenating both x values and squared y values'''
ideal_squ=pd.concat([ideal_cal,ideal_cal1],axis=1,join='inner')
ideal_squ
```

OUTPUT :



Out[3]:

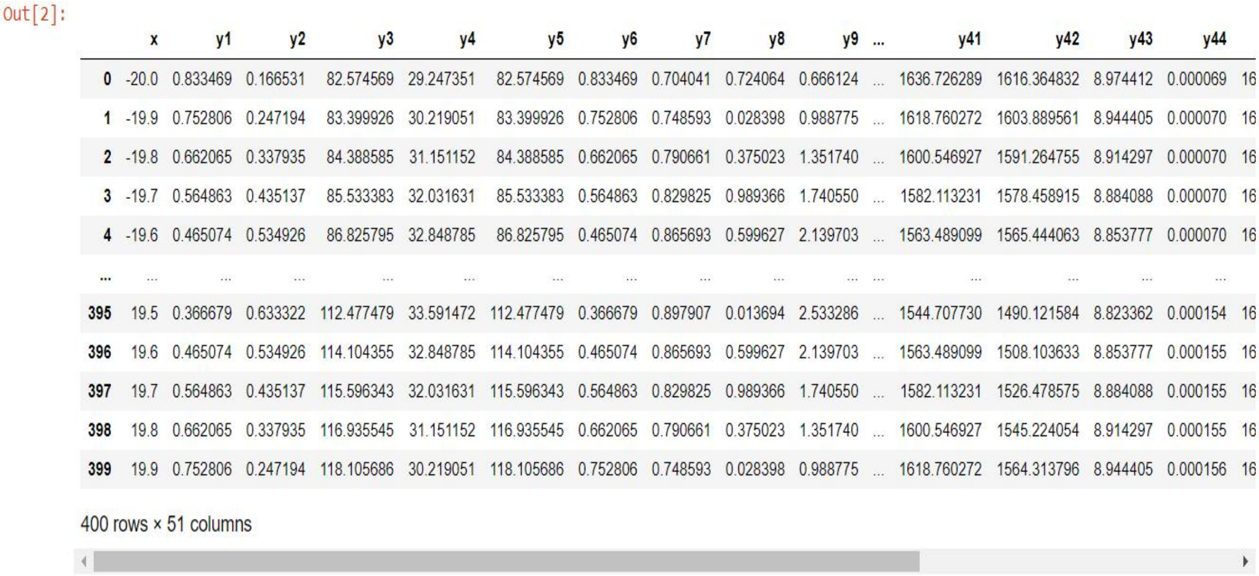
	x	y1	y2	y3	y4	y5	y6	y7	y8	y9	...	y41	y42	y43	y44
0	-20.0	0.833469	0.166531	82.574569	29.247351	82.574569	0.833469	0.704041	0.724064	0.666124	...	1636.726289	1616.364832	8.974412	0.000069
1	-19.9	0.752806	0.247194	83.399926	30.219051	83.399926	0.752806	0.748593	0.028398	0.988775	...	1618.760272	1603.889561	8.944405	0.000070
2	-19.8	0.662065	0.337935	84.388585	31.151152	84.388585	0.662065	0.790661	0.375023	1.351740	...	1600.546927	1591.264755	8.914297	0.000070
3	-19.7	0.564863	0.435137	85.533383	32.031631	85.533383	0.564863	0.829825	0.989366	1.740550	...	1582.113231	1578.458915	8.884088	0.000070
4	-19.6	0.465074	0.534926	86.825795	32.848785	86.825795	0.465074	0.865693	0.599627	2.139703	...	1563.489099	1565.444063	8.853777	0.000070
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
395	19.5	0.366679	0.633322	112.477479	33.591472	112.477479	0.366679	0.897907	0.013694	2.533286	...	1544.707730	1490.121584	8.823362	0.000154
396	19.6	0.465074	0.534926	114.104355	32.848785	114.104355	0.465074	0.865693	0.599627	2.139703	...	1563.489099	1508.103633	8.853777	0.000155
397	19.7	0.564863	0.435137	115.596343	32.031631	115.596343	0.564863	0.829825	0.989366	1.740550	...	1582.113231	1526.478575	8.884088	0.000155
398	19.8	0.662065	0.337935	116.935545	31.151152	116.935545	0.662065	0.790661	0.375023	1.351740	...	1600.546927	1545.224054	8.914297	0.000155
399	19.9	0.752806	0.247194	118.105686	30.219051	118.105686	0.752806	0.748593	0.028398	0.988775	...	1618.760272	1564.313796	8.944405	0.000156

Step 2:

Creating the Dataframe to Save the CSV File in Jupyter

```
df=pd.DataFrame(ideal_squ)
df.to_csv('C:/Users/Swetha/IU/ideal_update.csv', index=False)
df
```

OUTPUT :



Out[2]:

	x	y1	y2	y3	y4	y5	y6	y7	y8	y9	...	y41	y42	y43	y44	
0	-20.0	0.833469	0.166531	82.574569	29.247351	82.574569	0.833469	0.704041	0.724064	0.666124	...	1636.726289	1616.364832	8.974412	0.000069	16
1	-19.9	0.752806	0.247194	83.399926	30.219051	83.399926	0.752806	0.748593	0.028398	0.988775	...	1618.760272	1603.889561	8.944405	0.000070	16
2	-19.8	0.662065	0.337935	84.388585	31.151152	84.388585	0.662065	0.790661	0.375023	1.351740	...	1600.546927	1591.264755	8.914297	0.000070	16
3	-19.7	0.564863	0.435137	85.533383	32.031631	85.533383	0.564863	0.829825	0.989366	1.740550	...	1582.113231	1578.458915	8.884088	0.000070	16
4	-19.6	0.465074	0.534926	86.825795	32.848785	86.825795	0.465074	0.865693	0.599627	2.139703	...	1563.489099	1565.444063	8.853777	0.000070	16
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
395	19.5	0.366679	0.633322	112.477479	33.591472	112.477479	0.366679	0.897907	0.013694	2.533286	...	1544.707730	1490.121584	8.823362	0.000154	16
396	19.6	0.465074	0.534926	114.104355	32.848785	114.104355	0.465074	0.865693	0.599627	2.139703	...	1563.489099	1508.103633	8.853777	0.000155	16
397	19.7	0.564863	0.435137	115.596343	32.031631	115.596343	0.564863	0.829825	0.989366	1.740550	...	1582.113231	1526.478575	8.884088	0.000155	16
398	19.8	0.662065	0.337935	116.935545	31.151152	116.935545	0.662065	0.790661	0.375023	1.351740	...	1600.546927	1545.224054	8.914297	0.000155	16
399	19.9	0.752806	0.247194	118.105686	30.219051	118.105686	0.752806	0.748593	0.028398	0.988775	...	1618.760272	1564.313796	8.944405	0.000156	16

400 rows × 51 columns

Step 3:

Using sqlalchemy creating a DB file

```
'''Importing the modules'''
from sqlalchemy import create_engine as ce
import pandas as pd
'''Import data from CSV file'''
data=pd.read_csv('C:/Users/Swetha/IU/ideal_update.csv')
'''Creating a DB file'''
file_db = ce('sqlite:///C:/Users/Swetha/assignment_dataset/ideal_set.db')
'''Loading the CSV file into db file'''
data.to_sql('ideal_update',file_db)
```

OUTPUT :

```
Out[3]: 400
```

Step 4:

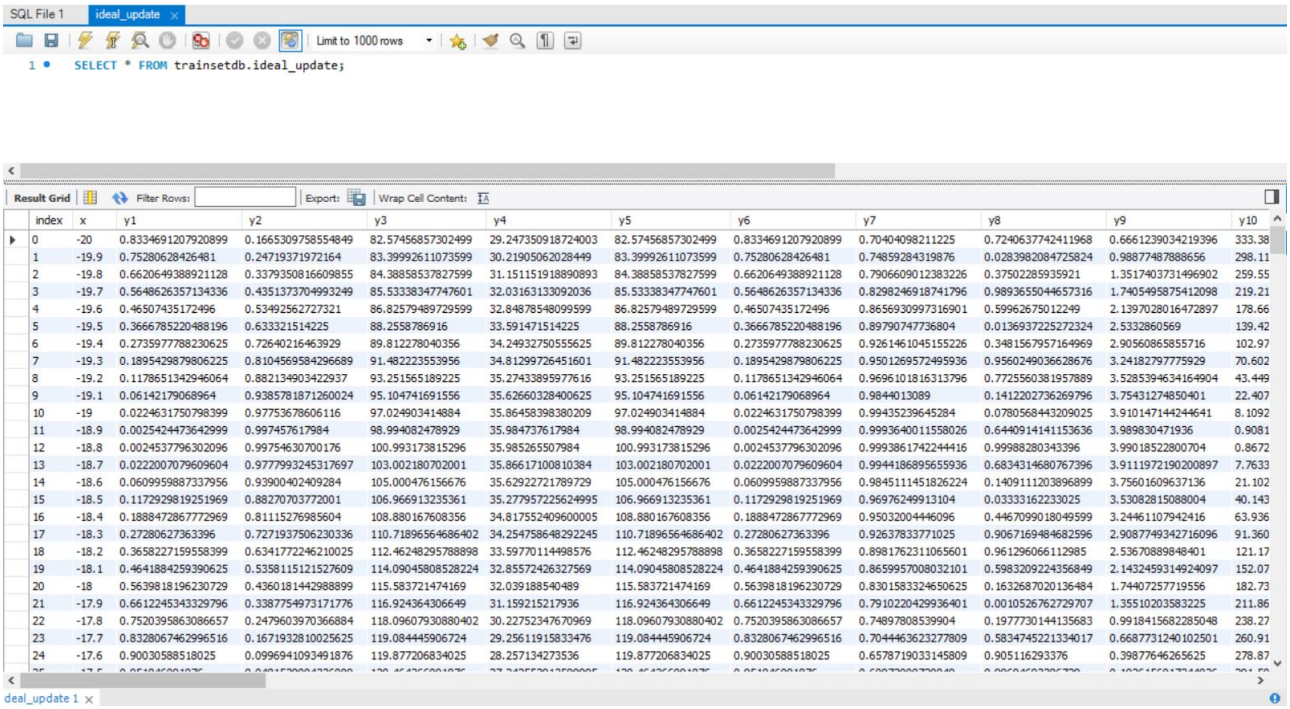
```
'''connecting to Mysql'''
mysql_engine = ce("mysql://root:mysql24S*@localhost:3306/trainsetdb")
```

Step 5:

```
'''Loading CSV to Mysql'''
data.to_sql('ideal_update',mysql_engine)
```

```
Out[3]: 400
```

Visualization in MYSQL - ideal data_set ‘Y’ Squared



```
SELECT * FROM trainsetdb.ideal_update;
```

index	x	y1	y2	y3	y4	y5	y6	y7	y8	y9	y10
0	-20	0.833469120792089 9	0.166530975855484 9	82.57456857302499	29.247350918724003	82.57456857302499	0.8334691207920899	0.70404098211225	0.7240637742411968	0.6661239034219396	333.38
1	-19.9	0.75280628426481	0.24719371972164	83.39992611073599	30.21905062028449	83.39992611073599	0.75280628426481	0.74859284319876	0.0283982084725824	0.98877487888656	298.11
2	-19.8	0.6620649388921128	0.3379350816609855	84.38858537827599	31.151151918890893	84.38858537827599	0.6620649388921128	0.7906609012383226	0.37502285935921	1.3517403731496902	259.55
3	-19.7	0.564862635713433 6	0.4351373704993249	85.53338347747601	32.03163133092036	85.53338347747601	0.5648626357134336	0.8298246918741796	0.9893655044657316	1.7405495875412098	219.21
4	-19.6	0.46507435172496	0.53492562727321	86.82579489729599	32.84878548099599	86.82579489729599	0.46507435172496	0.8656930997316901	0.59962675012249	2.1397028016472897	178.66
5	-19.5	0.3666785220488196	0.633321514225	88.2558786916	33.591471514225	88.2558786916	0.3666785220488196	0.89790747736804	0.0136937225272324	2.5332860569	139.42
6	-19.4	0.2735977788230625	0.72640216463929	89.812278040356	34.2493275055625	89.812278040356	0.2735977788230625	0.926146104515522 6	0.3481567957164969	2.90560865855716	102.97
7	-19.3	0.1895429879806225	0.8104569584296689	91.482223553956	34.81299726451601	91.482223553956	0.1895429879806225	0.9501269572495936	0.9560249036628676	3.24182797775929	70.602
8	-19.2	0.1178651342946064	0.882134903422937	93.251565189225	35.27433895977616	93.251565189225	0.1178651342946064	0.9696101816313796	0.7725560381957889	3.528539463416490 4	43.449
9	-19.1	0.06142179068964	0.9385781871260024	95.104741691556	35.62660328400625	95.104741691556	0.06142179068964	0.9844013089	0.1412202736269796	3.75431274850401	22.407
10	-19	0.0224631750798399	0.97753678606116	97.024903414884	35.86458398380209	97.024903414884	0.0224631750798399	0.99435239645284	0.0780568443209025	3.91014714424464 1	8.1092
11	-18.9	0.0025424473642999	0.997457617984	98.994082478929	35.984737617984	98.994082478929	0.0025424473642999	0.9993640011558026	0.6440914141153636	3.989830471936	0.9081
12	-18.8	0.0024537796302096	0.99754630700176	100.993173815296	35.985265507984	100.993173815296	0.0024537796302096	0.9993861742244416	0.99988280343396	3.99018522800704	0.8672
13	-18.7	0.0222007079609604	0.9777993245317697	103.002180702001	35.86617100810384	103.002180702001	0.0222007079609604	0.9944186895655936	0.6834314680767396	3.9111972190200897	7.7633
14	-18.6	0.0609959887337956	0.93900402409284	105.000476156676	35.6292272178972 9	105.000476156676	0.0609959887337956	0.9845111451826224	0.1409111203896899	3.7560160963713 6	21.102
15	-18.5	0.1172929819251969	0.88270703772001	106.966913235361	35.27957225624995	106.966913235361	0.1172929819251969	0.96976249913104	0.0333316223302 5	3.5308281508800 4	40.143
16	-18.4	0.1888472867772969	0.81115276985604	108.880167608356	34.817552409600005	108.880167608356	0.1888472867772969	0.95032004446096	0.4467099018049599	3.24461107942416	63.936
17	-18.3	0.27280627363396	0.7271937506230336	110.71896564686402	34.254758648292245	110.71896564686402	0.27280627363396	0.92637833771025	0.9067169484682596	2.9087749342716096	91.360
18	-18.2	0.3658227159558399	0.6341772246210025	112.46248295788898	33.59770114498576	112.46248295788898	0.3658227159558399	0.8981762311065601	0.96129606611298 5	2.53670889848401	121.17
19	-18.1	0.4641884259390625	0.5358115121527609	114.09045808528224	32.8557242632756 9	114.09045808528224	0.4641884259390625	0.8659957008032101	0.5983209224356849	2.1432459314924097	152.07
20	-18	0.5639818196230729	0.4360181442988899	115.583721474169	32.039188540489	115.583721474169	0.5639818196230729	0.8301583324650625	0.1632687020136484	1.7440725771955 6	182.73
21	-17.9	0.6612245343329796	0.3387754973171776	116.924364306649	31.159215217936	116.924364306649	0.6612245343329796	0.7910220429936401	0.0010526762729707	1.3551020358322 5	211.86
22	-17.8	0.7520395863086657	0.2479603970366884	118.09607930880402	30.2275234767096 9	118.09607930880402	0.7520395863086657	0.74897808539904	0.1977730144135683	0.9918415682285048	238.27
23	-17.7	0.8328067462996516	0.1671932810025625	119.084445906724	29.2561191583347 6	119.084445906724	0.8328067462996516	0.7044463623277809	0.5834745221334017	0.6687731240102501	260.91
24	-17.6	0.90030588518025	0.0996941093491876	119.877206834025	28.257134273536	119.877206834025	0.90030588518025	0.6578719033145809	0.905116293376	0.39877646265625	278.87

To find the least 4 minimum ideal data_set

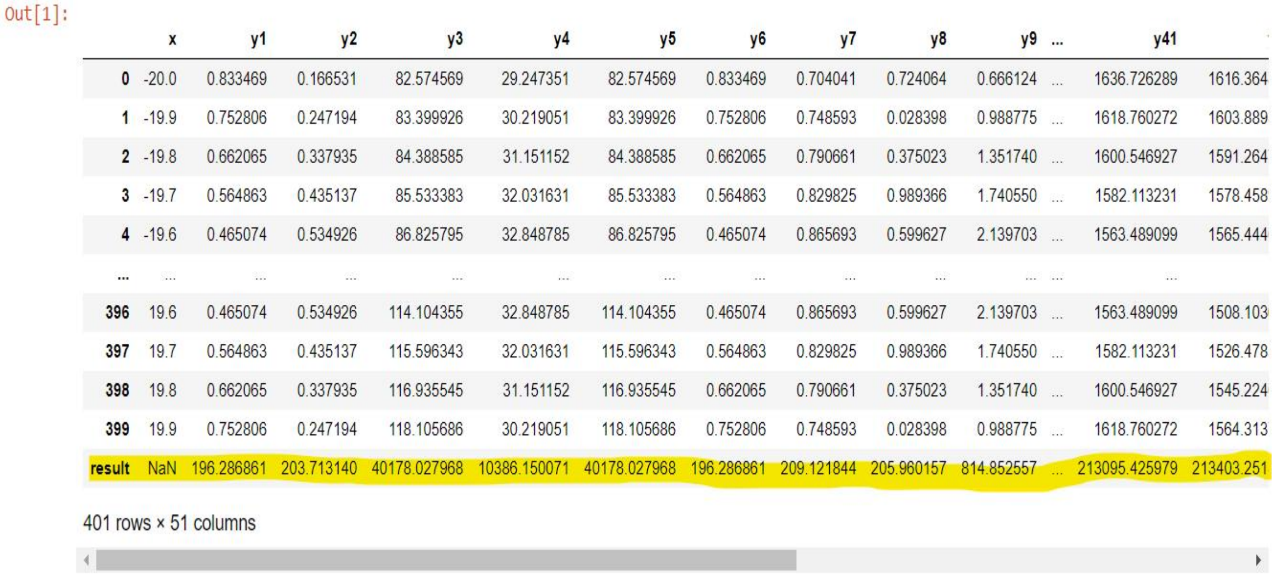
Sum of **'y'** values in columns to find the least minimum 4 data_set

'Y' values sum in "result" row

Step 1:

```
'''IMPORTING SQUARED IDEAL DATA AND SUM OF ALL DEVIATION'''
import pandas as pd
import numpy as np
id_min=pd.read_csv('C:/Users/Swetha/IU/ideal_update.csv')
df=pd.DataFrame(id_min)
dflist=['y1','y2','y3','y4','y5','y6','y7','y8','y9','y10',
       'y11','y12','y13','y14','y15','y16','y17','y18','y19','y20',
       'y21','y22','y23','y24','y25','y26','y27','y28','y29','y30',
       'y31','y32','y33','y34','y35','y36','y37','y38','y39','y40',
       'y41','y42','y43','y44','y45','y46','y47','y48','y49','y50']
df.'loc[ ]'['result'] = df[dflist].sum(axis=0)
df
```

OUTPUT :



Out[1]:

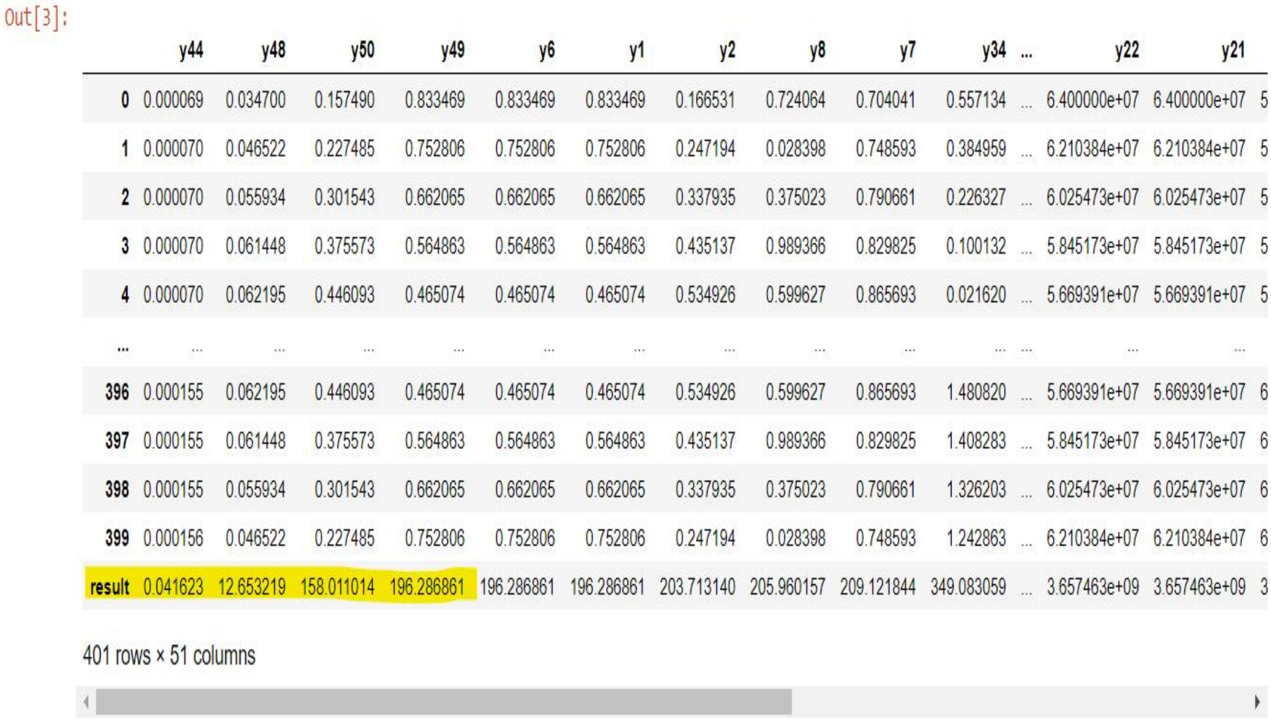
	x	y1	y2	y3	y4	y5	y6	y7	y8	y9	...	y41	
0	-20.0	0.833469	0.166531	82.574569	29.247351	82.574569	0.833469	0.704041	0.724064	0.666124	...	1636.726289	1616.364
1	-19.9	0.752806	0.247194	83.399926	30.219051	83.399926	0.752806	0.748593	0.028398	0.988775	...	1618.760272	1603.889
2	-19.8	0.662065	0.337935	84.388585	31.151152	84.388585	0.662065	0.790661	0.375023	1.351740	...	1600.546927	1591.264
3	-19.7	0.564863	0.435137	85.533383	32.031631	85.533383	0.564863	0.829825	0.989366	1.740550	...	1582.113231	1578.458
4	-19.6	0.465074	0.534926	86.825795	32.848785	86.825795	0.465074	0.865693	0.599627	2.139703	...	1563.489099	1565.444
...	...	...	...	...	...	...	...	...	...	...	...	...	
396	19.6	0.465074	0.534926	114.104355	32.848785	114.104355	0.465074	0.865693	0.599627	2.139703	...	1563.489099	1508.103
397	19.7	0.564863	0.435137	115.596343	32.031631	115.596343	0.564863	0.829825	0.989366	1.740550	...	1582.113231	1526.478
398	19.8	0.662065	0.337935	116.935545	31.151152	116.935545	0.662065	0.790661	0.375023	1.351740	...	1600.546927	1545.224
399	19.9	0.752806	0.247194	118.105686	30.219051	118.105686	0.752806	0.748593	0.028398	0.988775	...	1618.760272	1564.313
result	NaN	196.286861	203.713140	40178.027968	10386.150071	40178.027968	196.286861	209.121844	205.960157	814.852557	...	213095.425979	213403.251

401 rows × 51 columns

Step 2:
Sorting by Result Rows to find the least minimum values

```
'''SORTING THE SUMED ROWS '''
sum_of_col=pd.DataFrame(df)
df2=sum_of_col.sort_values(by = ["result"], axis=1, ascending=True)
df2
```

OUTPUT :



Out[3]:

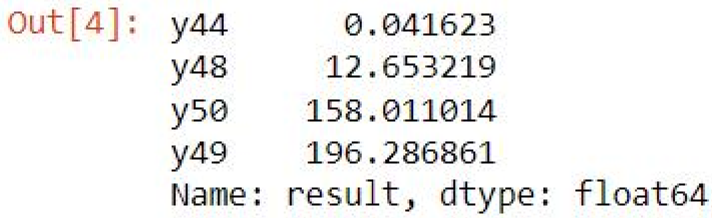
	y44	y48	y50	y49	y6	y1	y2	y8	y7	y34	...	y22	y21	
0	0.000069	0.034700	0.157490	0.833469	0.833469	0.833469	0.166531	0.724064	0.704041	0.557134	...	6.400000e+07	6.400000e+07	5
1	0.000070	0.046522	0.227485	0.752806	0.752806	0.752806	0.247194	0.028398	0.748593	0.384959	...	6.210384e+07	6.210384e+07	5
2	0.000070	0.055934	0.301543	0.662065	0.662065	0.662065	0.337935	0.375023	0.790661	0.226327	...	6.025473e+07	6.025473e+07	5
3	0.000070	0.061448	0.375573	0.564863	0.564863	0.564863	0.435137	0.989366	0.829825	0.100132	...	5.845173e+07	5.845173e+07	5
4	0.000070	0.062195	0.446093	0.465074	0.465074	0.465074	0.534926	0.599627	0.865693	0.021620	...	5.669391e+07	5.669391e+07	5
...	...	...	...	...	...	...	...	...	...	...	...	...	...	
396	0.000155	0.062195	0.446093	0.465074	0.465074	0.465074	0.534926	0.599627	0.865693	1.480820	...	5.669391e+07	5.669391e+07	6
397	0.000155	0.061448	0.375573	0.564863	0.564863	0.564863	0.435137	0.989366	0.829825	1.408283	...	5.845173e+07	5.845173e+07	6
398	0.000155	0.055934	0.301543	0.662065	0.662065	0.662065	0.337935	0.375023	0.790661	1.326203	...	6.025473e+07	6.025473e+07	6
399	0.000156	0.046522	0.227485	0.752806	0.752806	0.752806	0.247194	0.028398	0.748593	1.242863	...	6.210384e+07	6.210384e+07	6
result	0.041623	12.653219	158.011014	196.286861	196.286861	196.286861	203.713140	205.960157	209.121844	349.083059	...	3.657463e+09	3.657463e+09	3

401 rows × 51 columns

Step 3:

```
'''MINIMUM OF 4 IDEAL DATA SET'''
df3=df2.'loc[ ]'["result"]
df4=df3.head(4)
df4
```

OUTPUT :

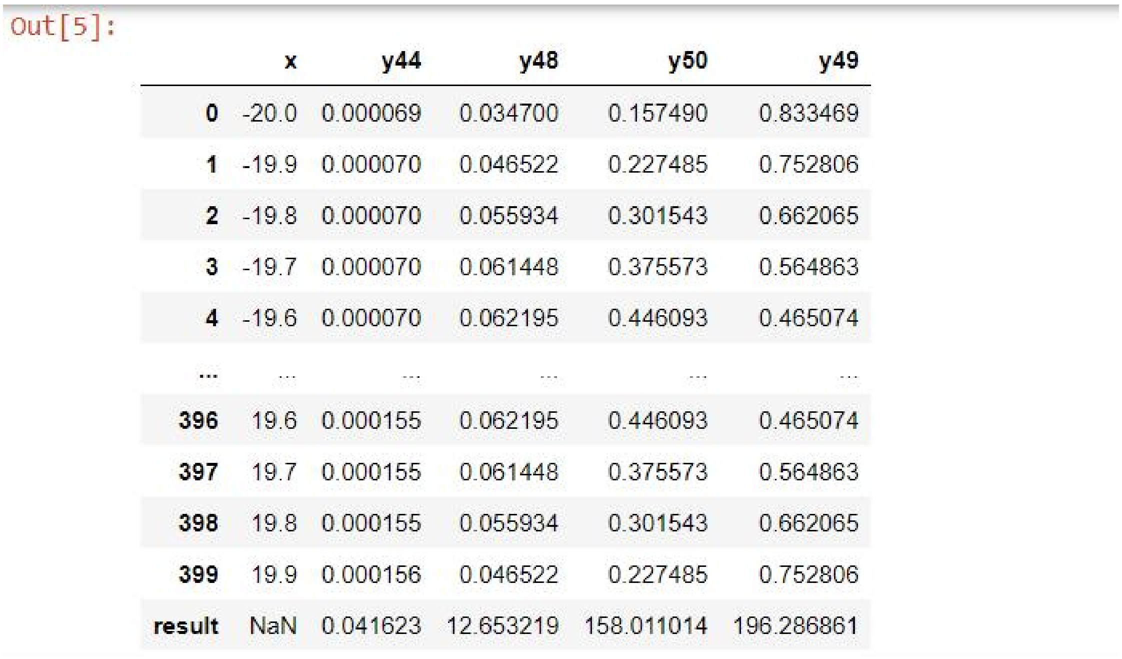


```
Out[4]: y44      0.041623
        y48     12.653219
        y50    158.011014
        y49    196.286861
        Name: result, dtype: float64
```

Step 4:
Visualization of 4 minimum least column of ideal set

```
df6=df[['x','y44','y48','y50','y49']]
df6
```

OUTPUT :



Out[5]:

	x	y44	y48	y50	y49
0	-20.0	0.000069	0.034700	0.157490	0.833469
1	-19.9	0.000070	0.046522	0.227485	0.752806
2	-19.8	0.000070	0.055934	0.301543	0.662065
3	-19.7	0.000070	0.061448	0.375573	0.564863
4	-19.6	0.000070	0.062195	0.446093	0.465074
...	...	...	...	...	...
396	19.6	0.000155	0.062195	0.446093	0.465074
397	19.7	0.000155	0.061448	0.375573	0.564863
398	19.8	0.000155	0.055934	0.301543	0.662065
399	19.9	0.000156	0.046522	0.227485	0.752806
result	NaN	0.041623	12.653219	158.011014	196.286861

Step 5:
Creating the Dataframe to Save the CSV File in Jupyter

```
df5=pd.DataFrame(df6)
'''dropping the index column and save to csv file'''
df5.to_csv('C:/Users/Swetha/IU/ideal_mininum_set_data.csv', index=False)
```

Step 6:

Using sqlalchemy creating a DB file

```
'''Importing the modules'''
from sqlalchemy import create_engine as ce
import pandas as pd
'''Import data from CSV file'''
data=pd.read_csv('C:/Users/Swetha/IU/ideal_mininum_set_data.csv')
'''Creating a DB file'''
file_db = ce('sqlite:///C:/Users/Swetha/assignment_dataset/ideal_min.db')
'''Loading the CSV file into db file'''
data.to_sql('ideal_mininum_set_data',file_db)
```

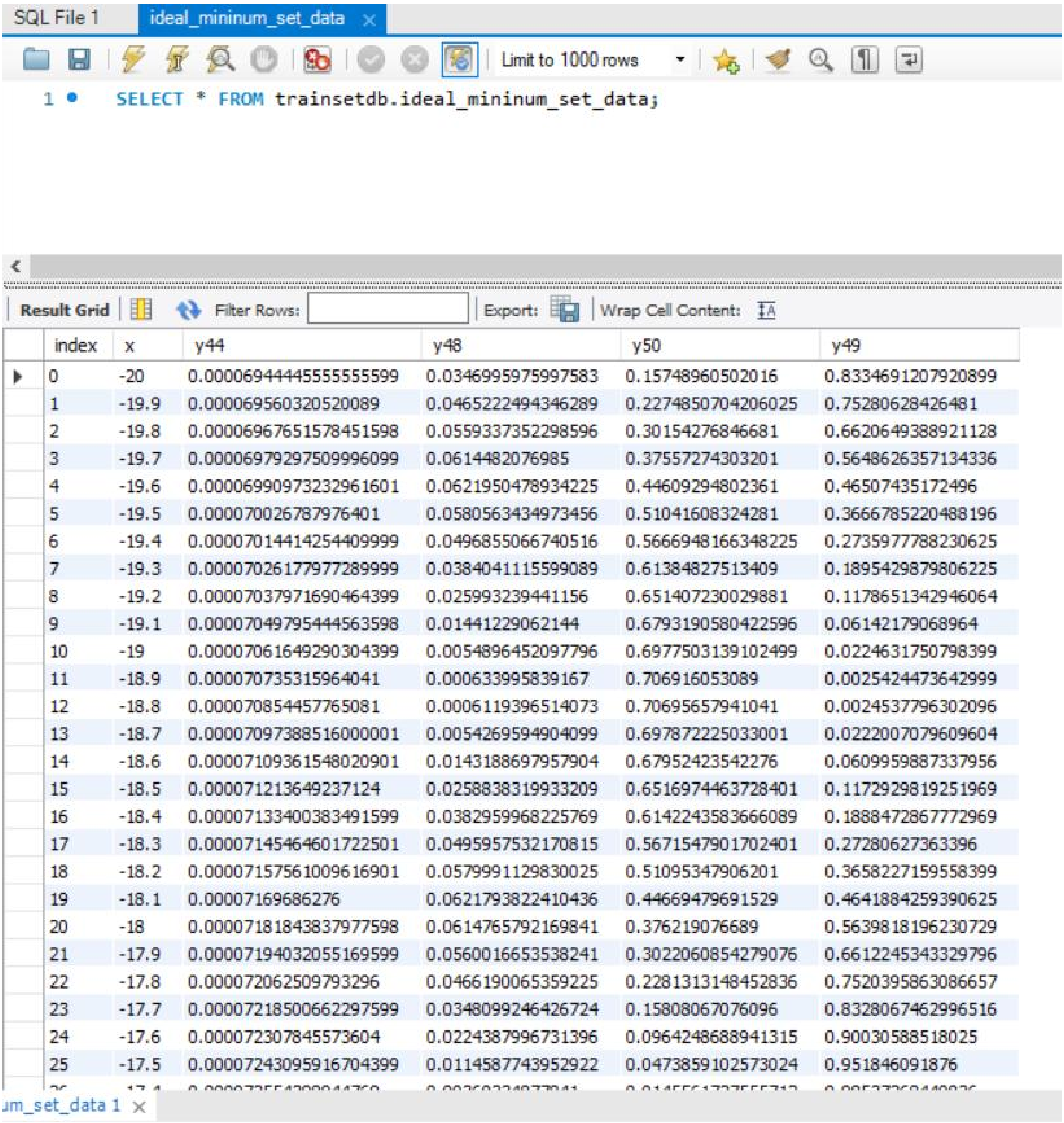
Step 7:

```
'''connecting to Mysql'''
mysql_engine = ce("mysql://root:mysql24S*@localhost:3306/trainsetdb")
```

Step 8:

```
'''Loading CSV to Mysql'''
data.to_sql('ideal_mininum_set_data',mysql_engine)
```

Visualization in MYSQL of ideal data_set - least of 4 minimum data_set



```
1 ●  SELECT * FROM trainsetdb.ideal_mininum_set_data;
```

index	x	y44	y48	y50	y49
0	-20	0.0000694444555555599	0.0346995975997583	0.15748960502016	0.8334691207920899
1	-19.9	0.000069560320520089	0.0465222494346289	0.2274850704206025	0.75280628426481
2	-19.8	0.00006967651578451598	0.0559337352298596	0.30154276846681	0.6620649388921128
3	-19.7	0.00006979297509996099	0.0614482076985	0.37557274303201	0.5648626357134336
4	-19.6	0.00006990973232961601	0.0621950478934225	0.44609294802361	0.46507435172496
5	-19.5	0.000070026787976401	0.0580563434973456	0.51041608324281	0.3666785220488196
6	-19.4	0.00007014414254409999	0.0496855066740516	0.5666948166348225	0.2735977788230625
7	-19.3	0.00007026177977289999	0.0384041115599089	0.61384827513409	0.1895429879806225
8	-19.2	0.00007037971690464399	0.025993239441156	0.651407230029881	0.1178651342946064
9	-19.1	0.00007049795444563598	0.01441229062144	0.6793190580422596	0.06142179068964
10	-19	0.00007061649290304399	0.0054896452097796	0.6977503139102499	0.0224631750798399
11	-18.9	0.000070735315964041	0.000633995839167	0.706916053089	0.0025424473642999
12	-18.8	0.000070854457765081	0.0006119396514073	0.70695657941041	0.0024537796302096
13	-18.7	0.00007097388516000001	0.0054269594904099	0.697872225033001	0.0222007079609604
14	-18.6	0.00007109361548020901	0.0143188697957904	0.67952423542276	0.0609959887337956
15	-18.5	0.000071213649237124	0.0258838319933209	0.6516974463728401	0.1172929819251969
16	-18.4	0.00007133400383491599	0.0382959968225769	0.6142243583666089	0.1888472867772969
17	-18.3	0.00007145464601722501	0.0495957532170815	0.5671547901702401	0.27280627363396
18	-18.2	0.00007157561009616901	0.0579991129830025	0.51095347906201	0.3658227159558399
19	-18.1	0.00007169686276	0.0621793822410436	0.44669479691529	0.4641884259390625
20	-18	0.00007181843837977598	0.0614765792169841	0.376219076689	0.5639818196230729
21	-17.9	0.00007194032055169599	0.0560016653538241	0.3022060854279076	0.6612245343329796
22	-17.8	0.000072062509793296	0.0466190065359225	0.2281313148452836	0.7520395863086657
23	-17.7	0.00007218500662297599	0.0348099246426724	0.15808067076096	0.8328067462996516
24	-17.6	0.000072307845573604	0.0224387996731396	0.0964248688941315	0.90030588518025
25	-17.5	0.00007243095916704399	0.0114587743952922	0.0473859102573024	0.951846091876

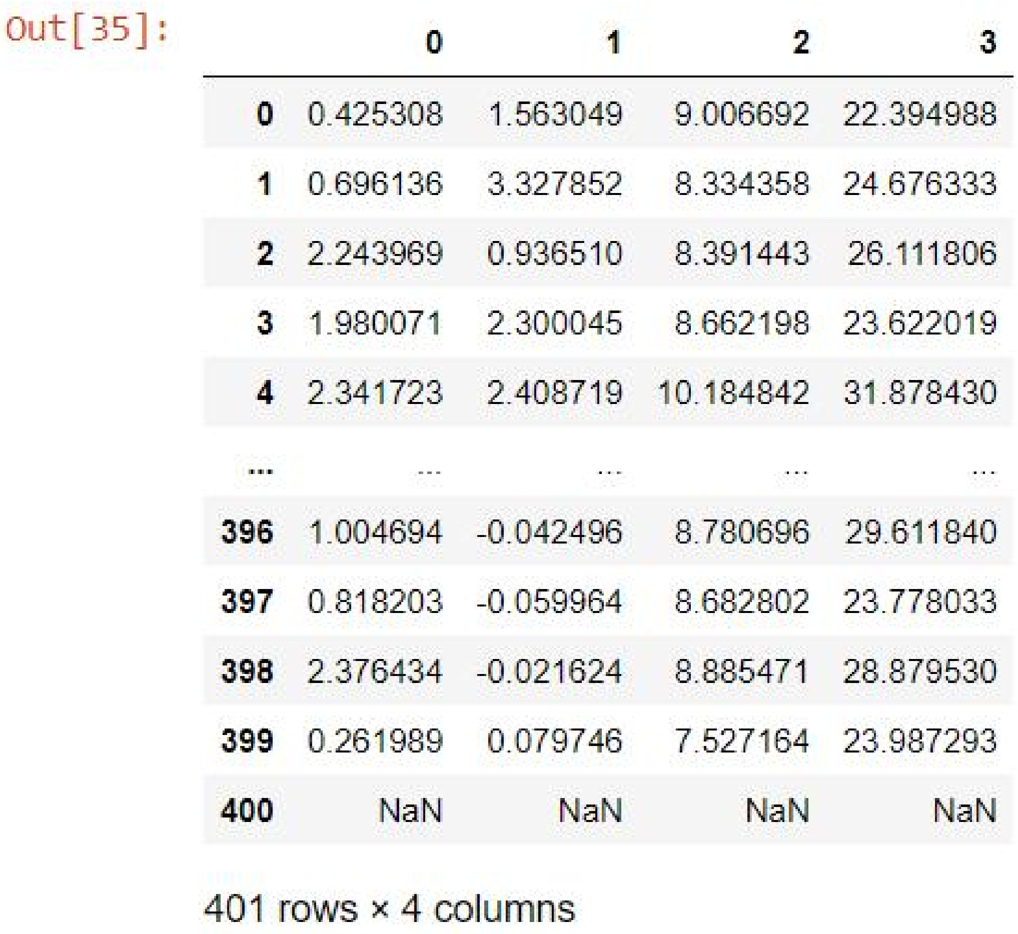
Finding the difference between Training Data_set & Ideal 4 minimum Data_set

Training Data Set - Ideal Minimum 4 Data Set

Step 1:

```
import pandas as pd
import numpy as np
df=pd.read_csv('C:/Users/Swetha/IU/tr_update.csv')
df1=pd.read_csv('C:/Users/Swetha/IU/ideal_minimum_set_data.csv')
dif2=df['y1']-df1['y44']
dif3=df['y2']-df1['y48']
dif4=df['y3']-df1['y50']
dif5=df['y4']-df1['y49']
dif6=pd.concat([dif2,dif3,dif4,dif5],axis=1,join='inner')
dif6
```

OUTPUT :



Out[35]:

	0	1	2	3
0	0.425308	1.563049	9.006692	22.394988
1	0.696136	3.327852	8.334358	24.676333
2	2.243969	0.936510	8.391443	26.111806
3	1.980071	2.300045	8.662198	23.622019
4	2.341723	2.408719	10.184842	31.878430
...	...	...	...	...
396	1.004694	-0.042496	8.780696	29.611840
397	0.818203	-0.059964	8.682802	23.778033
398	2.376434	-0.021624	8.885471	28.879530
399	0.261989	0.079746	7.527164	23.987293
400	NaN	NaN	NaN	NaN

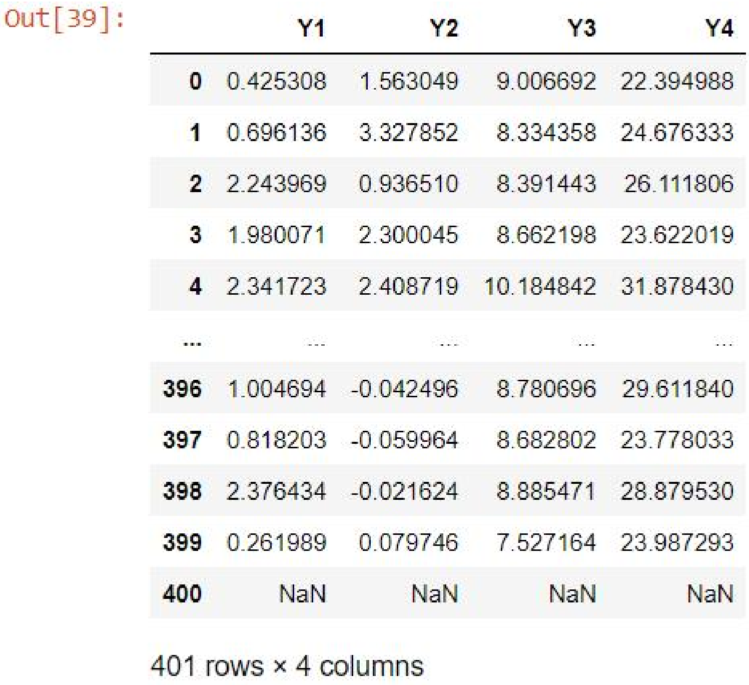
401 rows × 4 columns

Step 2:

Renaming the table Labels

```
dif7=pd.read_csv('C:/Users/Swetha/IU/tr_set_&_ideal_set_cal.csv')
dif8=pd.DataFrame(dif7)
least_squ=dif8.copy()
least_squ[["Y1","Y2","Y3","Y4"]]=dif8[["0","1","2","3"]]
dif9=least_squ.drop(['0','1','2','3'],axis=1)
dif9
```

OUTPUT :



Out[39]:

	Y1	Y2	Y3	Y4
0	0.425308	1.563049	9.006692	22.394988
1	0.696136	3.327852	8.334358	24.676333
2	2.243969	0.936510	8.391443	26.111806
3	1.980071	2.300045	8.662198	23.622019
4	2.341723	2.408719	10.184842	31.878430
...	...	...	...	...
396	1.004694	-0.042496	8.780696	29.611840
397	0.818203	-0.059964	8.682802	23.778033
398	2.376434	-0.021624	8.885471	28.879530
399	0.261989	0.079746	7.527164	23.987293
400	NaN	NaN	NaN	NaN

401 rows × 4 columns

Step 3:

Creating the Dataframe to Save the CSV File in Jupyter

```
comb_set=pd.DataFrame(dif9)
'''dropping the index column and save to csv file'''
comb_set.to_csv('C:/Users/Swetha/IU/tr_set_&_ideal_set_cal.csv', index=False)
comb_set
```

Step 4:

```
'''Importing the modules'''
from sqlalchemy import create_engine as ce
import pandas as pd
'''Import data from CSV file'''
data=pd.read_csv('C:/Users/Swetha/IU/tr_set_&_ideal_set_cal.csv')
'''Creating a DB file'''
file_db = ce('sqlite:///C:/Users/Swetha/assignment_dataset/tr_set_&_ideal_set.db')
'''Loading the CSV file into db file'''
data.to_sql('tr_set_&_ideal_set_cal',file_db)
```

OUTPUT :

```
Out[41]: 401
```

Step 5:

```
'''connecting to Mysql'''
mysql_engine = ce("mysql://root:mysql24S*@localhost:3306/trainsetdb")
```

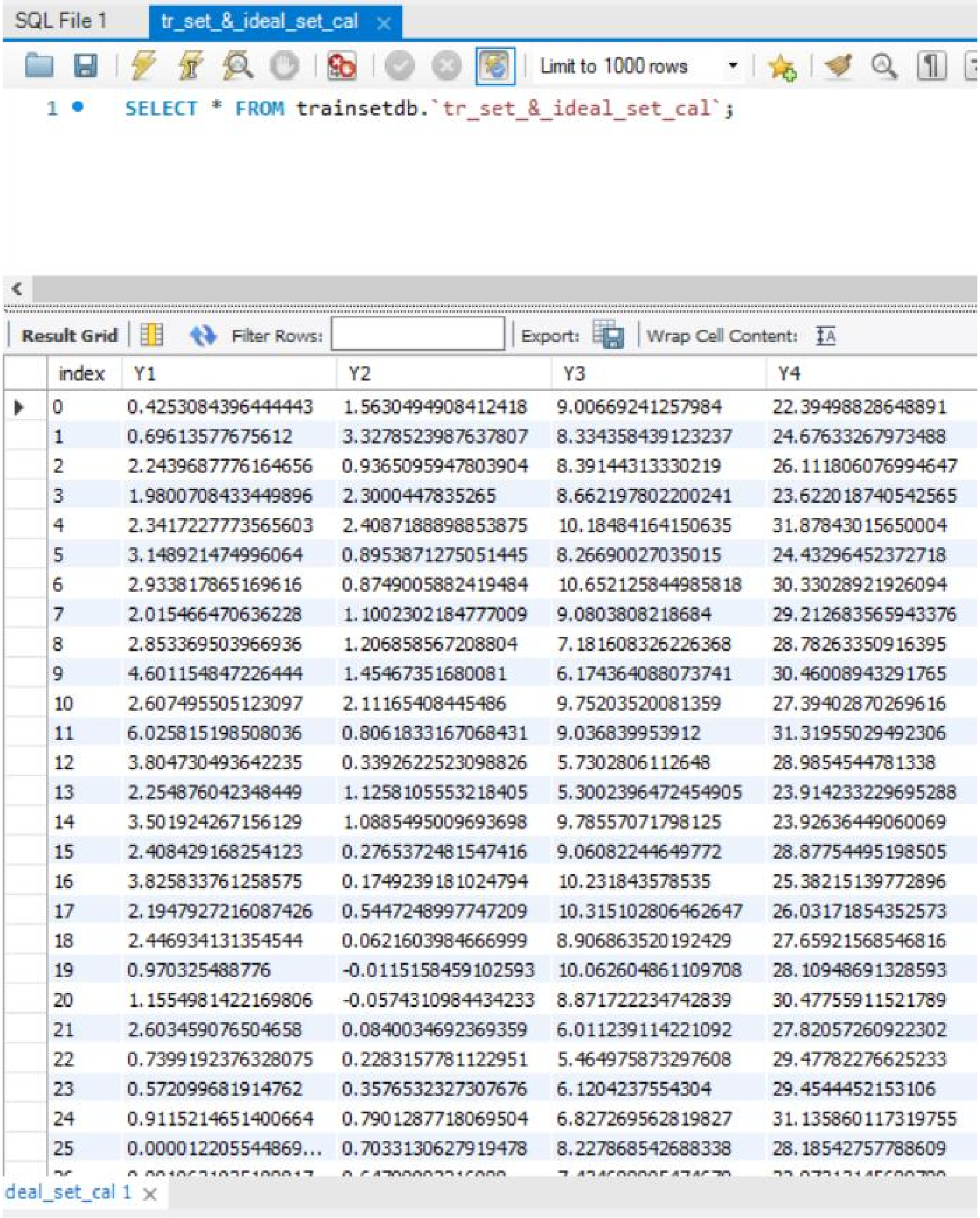
Step 6:

```
'''Loading CSV to Mysql'''
data.to_sql('tr_set_&_ideal_set_cal',mysql_engine)
```

OUTPUT :

```
Out[43]:  401
```

Visualization in MYSQL of training data_set & ideal 4 minimum data_set



```
1   SELECT * FROM trainsetdb.`tr_set_&_ideal_set_cal`;
```

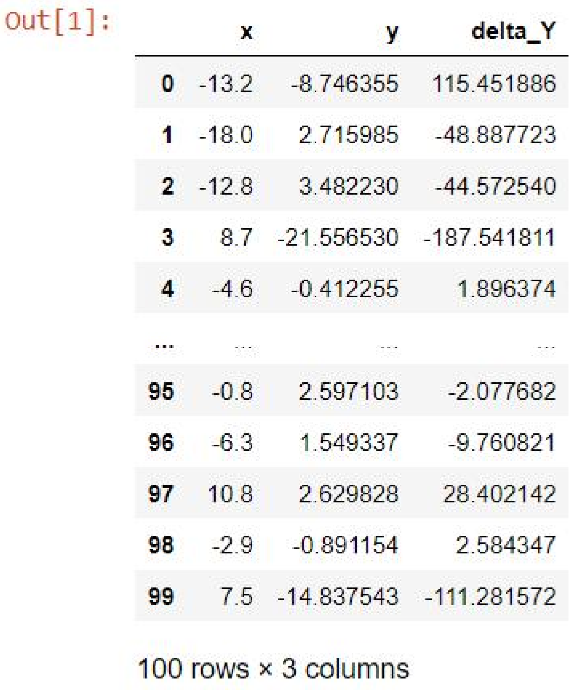
index	Y1	Y2	Y3	Y4
0	0.4253084396444443	1.5630494908412418	9.00669241257984	22.39498828648891
1	0.69613577675612	3.3278523987637807	8.334358439123237	24.67633267973488
2	2.2439687776164656	0.9365095947803904	8.39144313330219	26.111806076994647
3	1.9800708433449896	2.3000447835265	8.662197802200241	23.622018740542565
4	2.3417227773565603	2.4087188898853875	10.18484164150635	31.87843015650004
5	3.148921474996064	0.8953871275051445	8.26690027035015	24.43296452372718
6	2.933817865169616	0.8749005882419484	10.652125844985818	30.33028921926094
7	2.015466470636228	1.1002302184777009	9.0803808218684	29.212683565943376
8	2.853369503966936	1.206858567208804	7.181608326226368	28.78263350916395
9	4.601154847226444	1.45467351680081	6.174364088073741	30.46008943291765
10	2.607495505123097	2.11165408445486	9.75203520081359	27.39402870269616
11	6.025815198508036	0.8061833167068431	9.036839953912	31.31955029492306
12	3.804730493642235	0.3392622523098826	5.7302806112648	28.9854544781338
13	2.254876042348449	1.1258105553218405	5.3002396472454905	23.914233229695288
14	3.501924267156129	1.0885495009693698	9.78557071798125	23.92636449060069
15	2.408429168254123	0.2765372481547416	9.06082244649772	28.87754495198505
16	3.825833761258575	0.1749239181024794	10.231843578535	25.38215139772896
17	2.1947927216087426	0.5447248997747209	10.315102806462647	26.03171854352573
18	2.446934131354544	0.0621603984666999	8.906863520192429	27.65921568546816
19	0.970325488776	-0.0115158459102593	10.062604861109708	28.10948691328593
20	1.1554981422169806	-0.0574310984434233	8.871722234742839	30.47755911521789
21	2.603459076504658	0.0840034692369359	6.011239114221092	27.82057260922302
22	0.7399192376328075	0.2283157781122951	5.464975873297608	29.47782276625233
23	0.572099681914762	0.3576532327307676	6.1204237554304	29.4544452153106
24	0.9115214651400664	0.7901287718069504	6.827269562819827	31.135860117319755
25	0.000012205544869...	0.7033130627919478	8.227868542688338	28.18542757788609
26	[illegible]	[illegible]	[illegible]	[illegible]

Test Data_set
‘X’ values * ‘Y’ values to find delta functions

Step 1:

```
import pandas as pd
import numpy as np
test_file=pd.read_csv('C:/Users/Swetha/IU/test.csv')
test_file["delta_Y"]=test_file["x"] * test_file["y"]
test_file
```

OUTPUT :



Out[1]:

	x	y	delta_Y
0	-13.2	-8.746355	115.451886
1	-18.0	2.715985	-48.887723
2	-12.8	3.482230	-44.572540
3	8.7	-21.556530	-187.541811
4	-4.6	-0.412255	1.896374
...	...	...	...
95	-0.8	2.597103	-2.077682
96	-6.3	1.549337	-9.760821
97	10.8	2.629828	28.402142
98	-2.9	-0.891154	2.584347
99	7.5	-14.837543	-111.281572

100 rows × 3 columns

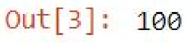
Step 2:
Creating the Dataframe to Save the CSV File in Jupyter

```
df=pd.DataFrame(test_file)
'''dropping the index column and save to csv file'''
df.to_csv('C:/Users/Swetha/IU/test_cal.csv', index=False)
df
```

Step 3:

```
'''Importing the modules'''
from sqlalchemy import create_engine as ce
import pandas as pd
'''Import data from CSV file'''
data=pd.read_csv('C:/Users/Swetha/IU/test_cal.csv')
'''Creating a DB file'''
file_db = ce('sqlite:///C:/Users/Swetha/assignment_dataset/test_cal.db')
'''Loading the CSV file into db file'''
data.to_sql('test_cal',file_db)
```

OUTPUT :



```
Out[3]: 100
```

Step 4:

```
'''connecting to Mysql'''
mysql_engine = ce("mysql://root:mysql24S*@localhost:3306/trainsetdb")
```

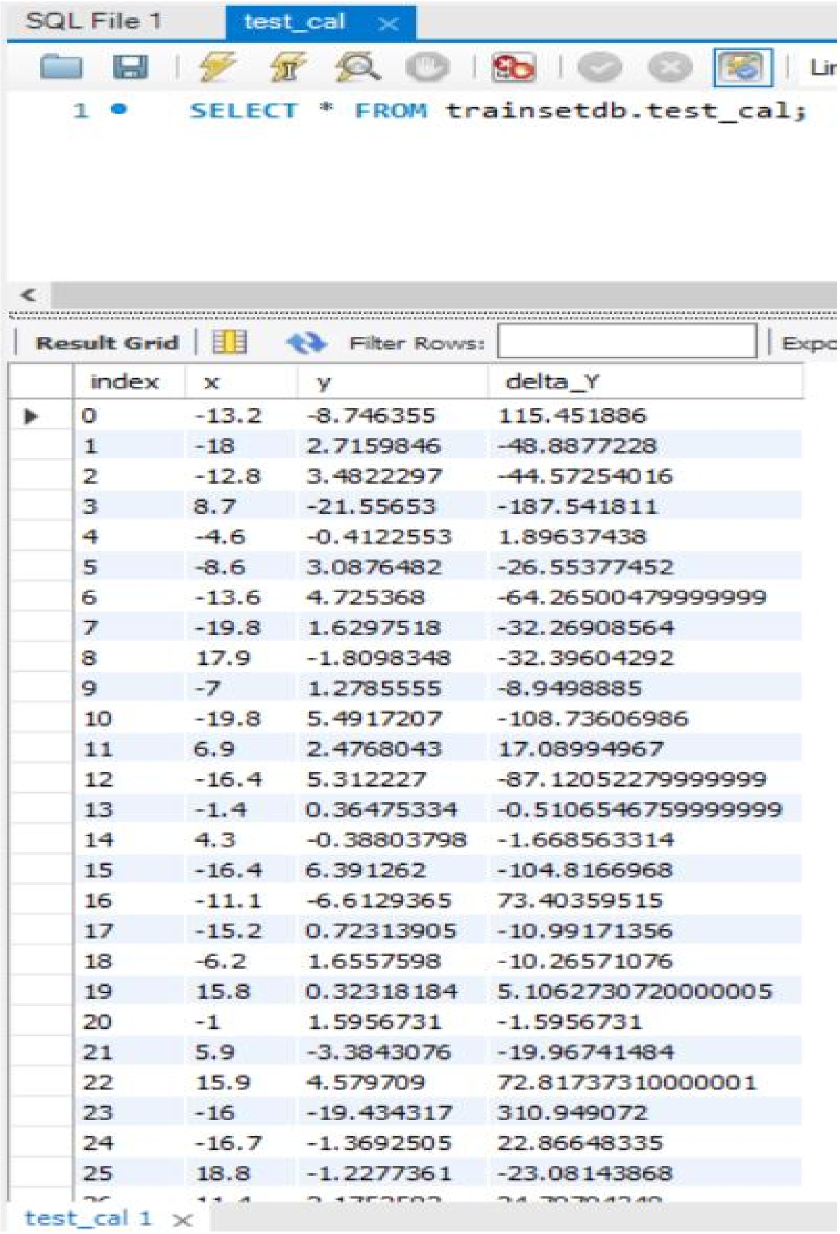
Step 5:

```
'''Loading CSV to Mysql'''
data.to_sql('test_cal',mysql_engine)
```

OUTPUT :

```
Out[5]:  100
```

Visualization in MYSQL of test_data_set to find the delta_set



```
1    SELECT * FROM trainsetdb.test_cal;
```

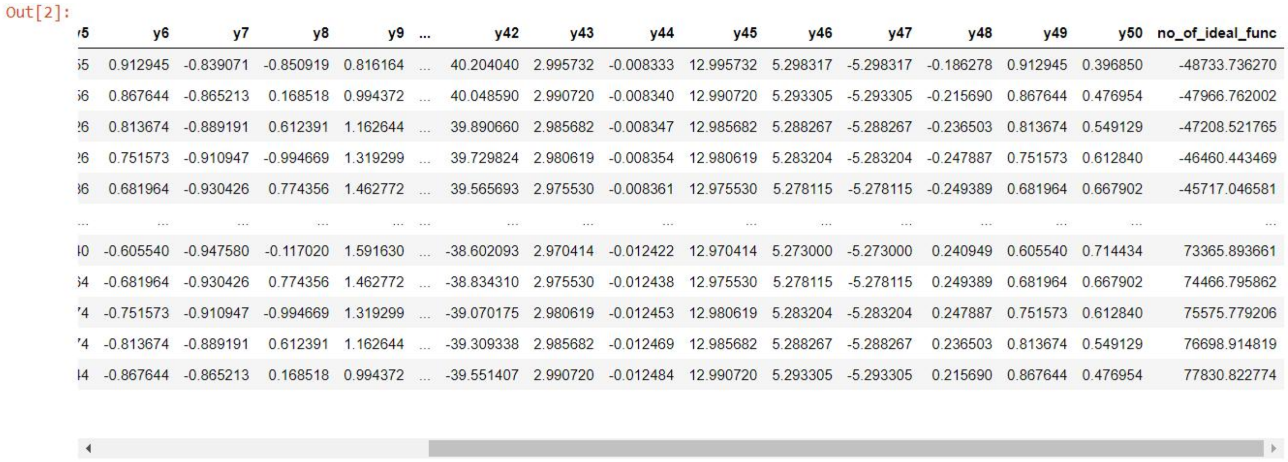
index	x	y	delta_Y
0	-13.2	-8.746355	115.451886
1	-18	2.7159846	-48.8877228
2	-12.8	3.4822297	-44.57254016
3	8.7	-21.55653	-187.541811
4	-4.6	-0.4122553	1.89637438
5	-8.6	3.0876482	-26.55377452
6	-13.6	4.725368	-64.26500479999999
7	-19.8	1.6297518	-32.26908564
8	17.9	-1.8098348	-32.39604292
9	-7	1.2785555	-8.9498885
10	-19.8	5.4917207	-108.73606986
11	6.9	2.4768043	17.08994967
12	-16.4	5.312227	-87.12052279999999
13	-1.4	0.36475334	-0.5106546759999999
14	4.3	-0.38803798	-1.668563314
15	-16.4	6.391262	-104.8166968
16	-11.1	-6.6129365	73.40359515
17	-15.2	0.72313905	-10.99171356
18	-6.2	1.6557598	-10.26571076
19	15.8	0.32318184	5.106273072000005
20	-1	1.5956731	-1.5956731
21	5.9	-3.3843076	-19.96741484
22	15.9	4.579709	72.8173731000001
23	-16	-19.434317	310.949072
24	-16.7	-1.3692505	22.86648335
25	18.8	-1.2277361	-23.08143868

Sum of ideal data_set rows for Table 3

Step 1:

```
import pandas as pd
import numpy as np
id_min=pd.read_csv('C:/Users/Swetha/IU/ideal.csv')
df=pd.DataFrame(id_min)
df['no_of_ideal_func'] = df.sum(axis=1)
df
```

OUTPUT :



Out[2]:

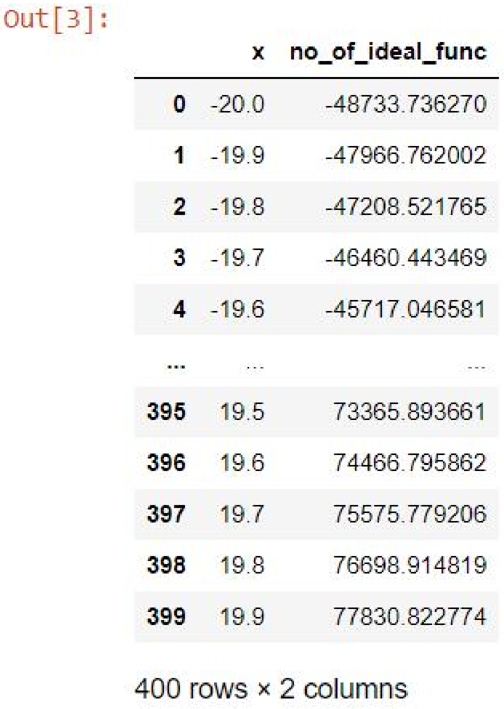
5	y6	y7	y8	y9	...	y42	y43	y44	y45	y46	y47	y48	y49	y50	no_of_ideal_func
5	0.912945	-0.839071	-0.850919	0.816164	...	40.204040	2.995732	-0.008333	12.995732	5.298317	-5.298317	-0.186278	0.912945	0.396850	-48733.736270
6	0.867644	-0.865213	0.168518	0.994372	...	40.048590	2.990720	-0.008340	12.990720	5.293305	-5.293305	-0.215690	0.867644	0.476954	-47966.762002
26	0.813674	-0.889191	0.612391	1.162644	...	39.890660	2.985682	-0.008347	12.985682	5.288267	-5.288267	-0.236503	0.813674	0.549129	-47208.521765
26	0.751573	-0.910947	-0.994669	1.319299	...	39.729824	2.980619	-0.008354	12.980619	5.283204	-5.283204	-0.247887	0.751573	0.612840	-46460.443469
36	0.681964	-0.930426	0.774356	1.462772	...	39.565693	2.975530	-0.008361	12.975530	5.278115	-5.278115	-0.249389	0.681964	0.667902	-45717.046581
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
40	-0.605540	-0.947580	-0.117020	1.591630	...	-38.602093	2.970414	-0.012422	12.970414	5.273000	-5.273000	0.240949	0.605540	0.714434	73365.893661
54	-0.681964	-0.930426	0.774356	1.462772	...	-38.834310	2.975530	-0.012438	12.975530	5.278115	-5.278115	0.249389	0.681964	0.667902	74466.795862
74	-0.751573	-0.910947	-0.994669	1.319299	...	-39.070175	2.980619	-0.012453	12.980619	5.283204	-5.283204	0.247887	0.751573	0.612840	75575.779206
74	-0.813674	-0.889191	0.612391	1.162644	...	-39.309338	2.985682	-0.012469	12.985682	5.288267	-5.288267	0.236503	0.813674	0.549129	76698.914819
44	-0.867644	-0.865213	0.168518	0.994372	...	-39.551407	2.990720	-0.012484	12.990720	5.293305	-5.293305	0.215690	0.867644	0.476954	77830.822774

Step 2:

Slicing table to sum_no_of_rows

```
df6=df[['x','no_of_ideal_func']]
df6
```

OUTPUT :



Out[3]:

	x	no_of_ideal_func
0	-20.0	-48733.736270
1	-19.9	-47966.762002
2	-19.8	-47208.521765
3	-19.7	-46460.443469
4	-19.6	-45717.046581
...	...	...
395	19.5	73365.893661
396	19.6	74466.795862
397	19.7	75575.779206
398	19.8	76698.914819
399	19.9	77830.822774

400 rows × 2 columns

Step 3:
Creating the Dataframe to Save the CSV File in Jupyter

```
df5=pd.DataFrame(df6)
'''dropping the index column and save to csv file'''
df5.to_csv('C:/Users/Swetha/IU/no_of_ideal_func.csv', index=False)
```

Step 4:

```
'''Importing the modules'''
from sqlalchemy import create_engine as ce
import pandas as pd
'''Import data from CSV file'''
data=pd.read_csv('C:/Users/Swetha/IU/no_of_ideal_func.csv')
'''Creating a DB file'''
file_db = ce('sqlite:///C:/Users/Swetha/assignment_dataset/no_of_ideal_func.db')
'''Loading the CSV file into db file'''
data.to_sql('no_of_ideal_func',file_db)
```

OUTPUT :

```
Out[5]: 400
```

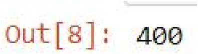
Step 5:

```
'''connecting to Mysql'''
mysql_engine = ce("mysql://root:mysql24S*@localhost:3306/trainsetdb")
```

Step 6:

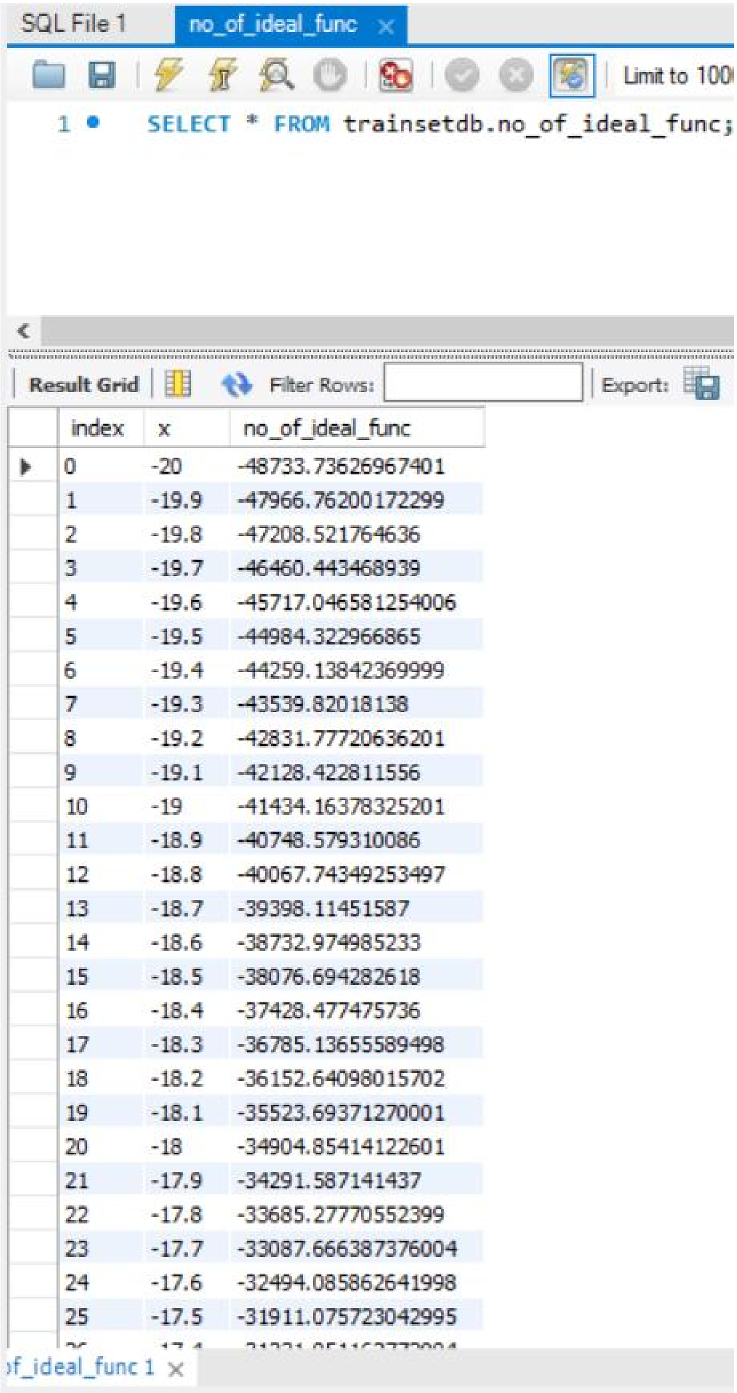
```
'''Loading CSV to Mysql'''
data.to_sql('no_of_ideal_func',mysql_engine)
```

OUTPUT :



```
Out[8]: 400
```

Visualization in MYSQL of no_of_ideal_function



```sql
SELECT * FROM trainsetdb.no_of_ideal_func;
```

index	x	no_of_ideal_func
0	-20	-48733.73626967401
1	-19.9	-47966.76200172299
2	-19.8	-47208.521764636
3	-19.7	-46460.443468939
4	-19.6	-45717.046581254006
5	-19.5	-44984.322966865
6	-19.4	-44259.13842369999
7	-19.3	-43539.82018138
8	-19.2	-42831.77720636201
9	-19.1	-42128.422811556
10	-19	-41434.16378325201
11	-18.9	-40748.579310086
12	-18.8	-40067.74349253497
13	-18.7	-39398.11451587
14	-18.6	-38732.974985233
15	-18.5	-38076.694282618
16	-18.4	-37428.477475736
17	-18.3	-36785.13655589498
18	-18.2	-36152.64098015702
19	-18.1	-35523.69371270001
20	-18	-34904.85414122601
21	-17.9	-34291.587141437
22	-17.8	-33685.27770552399
23	-17.7	-33087.666387376004
24	-17.6	-32494.085862641998
25	-17.5	-31911.075723042995

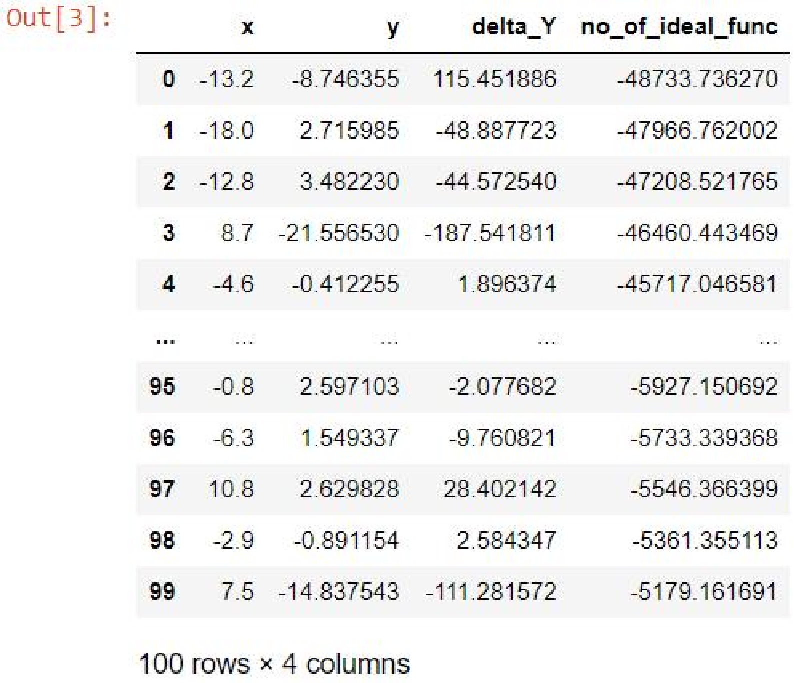
Table 3
Test Data Set & Delta Set

Step 1:

Combining the test calculate file & no of ideal function

```
import pandas as pd
import numpy as np
file1=pd.read_csv('C:/Users/Swetha/IU/test_cal.csv')
file2=pd.read_csv('C:/Users/Swetha/IU/no_of_ideal_func.csv')
df1=pd.DataFrame(file1)
df2=pd.DataFrame(file2)
dif3=pd.concat([df1[['x','y','delta_Y']],df2[['no_of_ideal_func']]],axis=1,join='inner')
dif3
```

OUTPUT :



Out[3]:

	x	y	delta_Y	no_of_ideal_func
0	-13.2	-8.746355	115.451886	-48733.736270
1	-18.0	2.715985	-48.887723	-47966.762002
2	-12.8	3.482230	-44.572540	-47208.521765
3	8.7	-21.556530	-187.541811	-46460.443469
4	-4.6	-0.412255	1.896374	-45717.046581
...	...	...	...	...
95	-0.8	2.597103	-2.077682	-5927.150692
96	-6.3	1.549337	-9.760821	-5733.339368
97	10.8	2.629828	28.402142	-5546.366399
98	-2.9	-0.891154	2.584347	-5361.355113
99	7.5	-14.837543	-111.281572	-5179.161691

100 rows × 4 columns

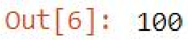
Step 2:
Creating the Dataframe to Save the CSV File in Jupyter

```
df5=pd.DataFrame(dif3)
'''dropping the index column and save to csv file'''
df5.to_csv('C:/Users/Swetha/IU/delta_file.csv', index=False)
```

Step 3:

```
'''Importing the modules'''
from sqlalchemy import create_engine as ce
import pandas as pd
'''Import data from CSV file'''
data=pd.read_csv('C:/Users/Swetha/IU/delta_file.csv')
'''Creating a DB file'''
file_db = ce('sqlite:///C:/Users/Swetha/assignment_dataset/delta_file.db')
'''Loading the CSV file into db file'''
data.to_sql('delta_file',file_db)
```

OUTPUT :



```
Out[6]: 100
```

Step 4:

'''connecting to Mysql'''
mysql_engine = ce("mysql://root:mysql24S@localhost:3306/trainsetdb")*

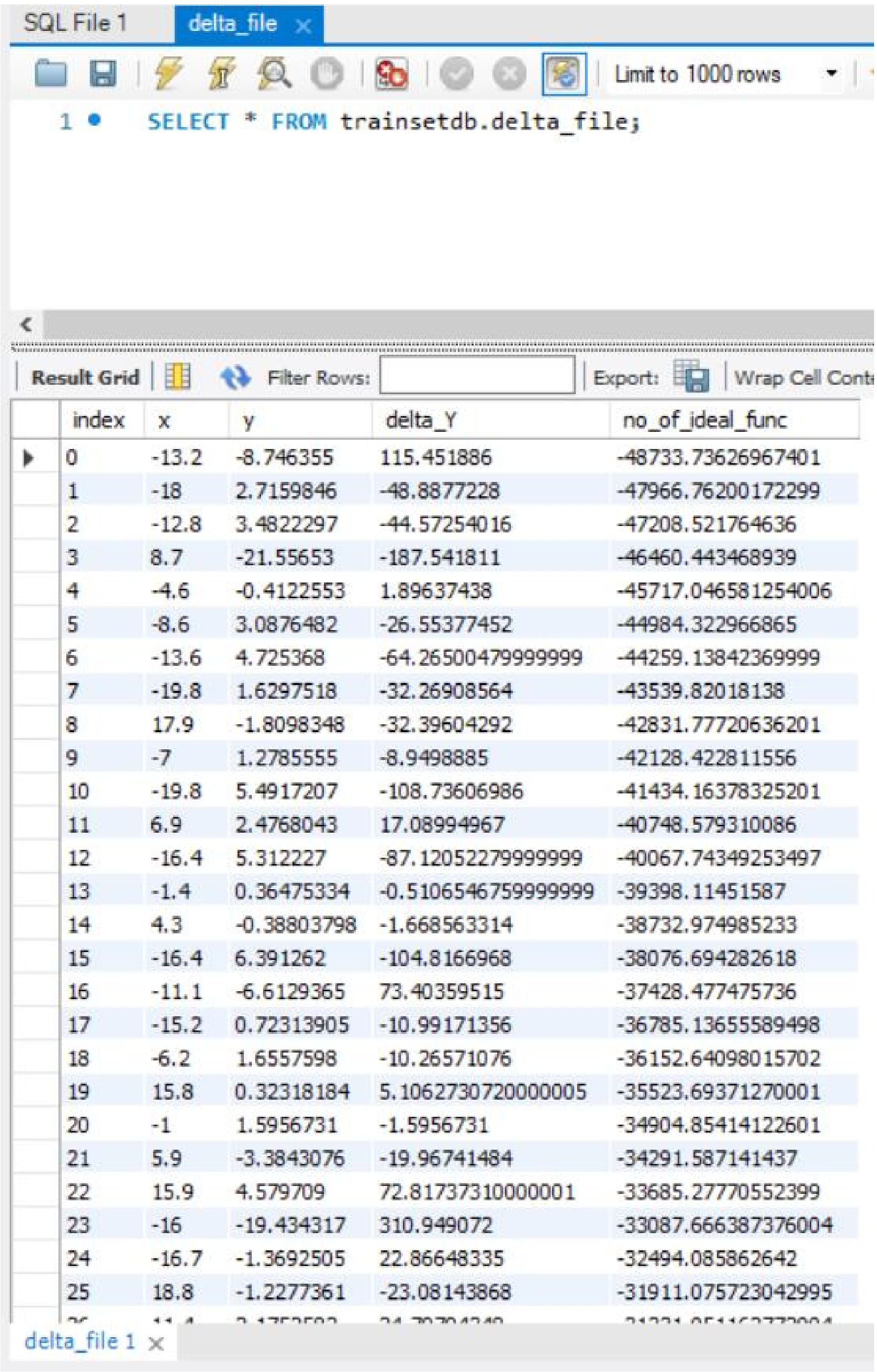
Step 5:

'''Loading CSV to Mysql'''
data.to_sql('delta_file',mysql_engine)

OUTPUT :

```
Out[8]: 100
```

Visualization in MYSQL of Delta File



```
SELECT * FROM trainsetdb.delta_file;
```

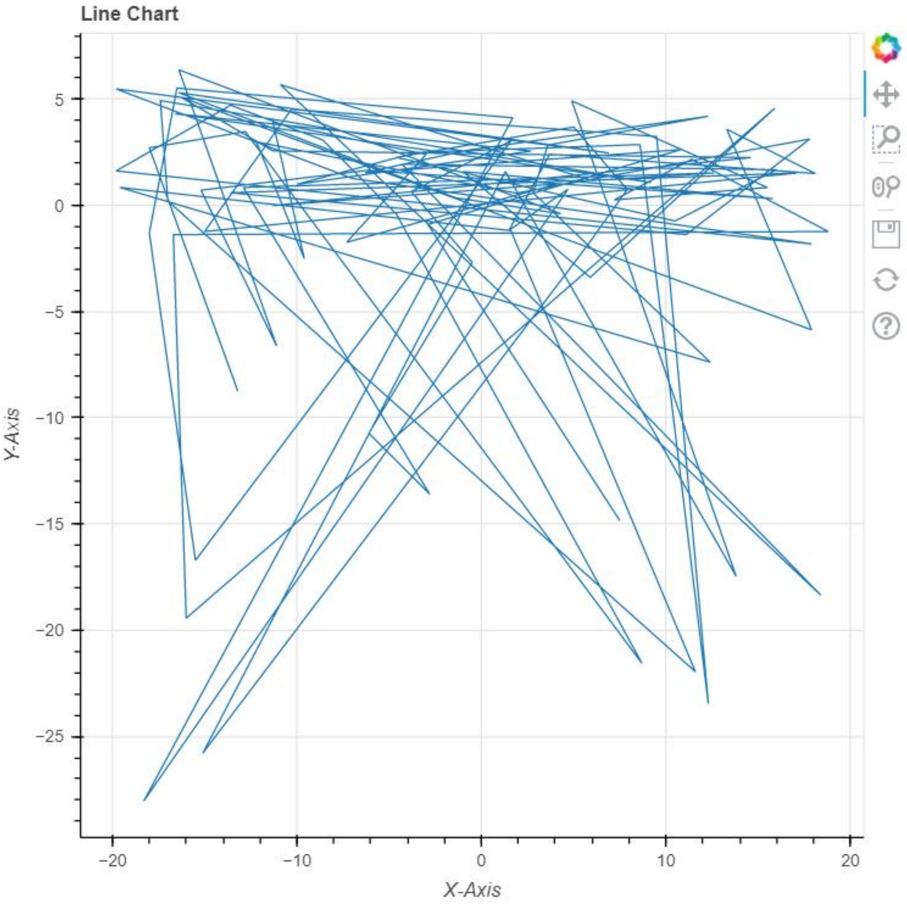
index	x	y	delta_Y	no_of_ideal_func
0	-13.2	-8.746355	115.451886	-48733.73626967401
1	-18	2.7159846	-48.8877228	-47966.76200172299
2	-12.8	3.4822297	-44.57254016	-47208.521764636
3	8.7	-21.55653	-187.541811	-46460.443468939
4	-4.6	-0.4122553	1.89637438	-45717.046581254006
5	-8.6	3.0876482	-26.55377452	-44984.322966865
6	-13.6	4.725368	-64.26500479999999	-44259.13842369999
7	-19.8	1.6297518	-32.26908564	-43539.82018138
8	17.9	-1.8098348	-32.39604292	-42831.77720636201
9	-7	1.2785555	-8.9498885	-42128.422811556
10	-19.8	5.4917207	-108.73606986	-41434.16378325201
11	6.9	2.4768043	17.08994967	-40748.579310086
12	-16.4	5.312227	-87.12052279999999	-40067.74349253497
13	-1.4	0.36475334	-0.5106546759999999	-39398.11451587
14	4.3	-0.38803798	-1.668563314	-38732.974985233
15	-16.4	6.391262	-104.8166968	-38076.694282618
16	-11.1	-6.6129365	73.40359515	-37428.477475736
17	-15.2	0.72313905	-10.99171356	-36785.13655589498
18	-6.2	1.6557598	-10.26571076	-36152.64098015702
19	15.8	0.32318184	5.1062730720000005	-35523.69371270001
20	-1	1.5956731	-1.5956731	-34904.85414122601
21	5.9	-3.3843076	-19.96741484	-34291.587141437
22	15.9	4.579709	72.81737310000001	-33685.27770552399
23	-16	-19.434317	310.949072	-33087.666387376004
24	-16.7	-1.3692505	22.86648335	-32494.085862642
25	18.8	-1.2277361	-23.08143868	-31911.075723042995

BOKEH VISUALIZATION

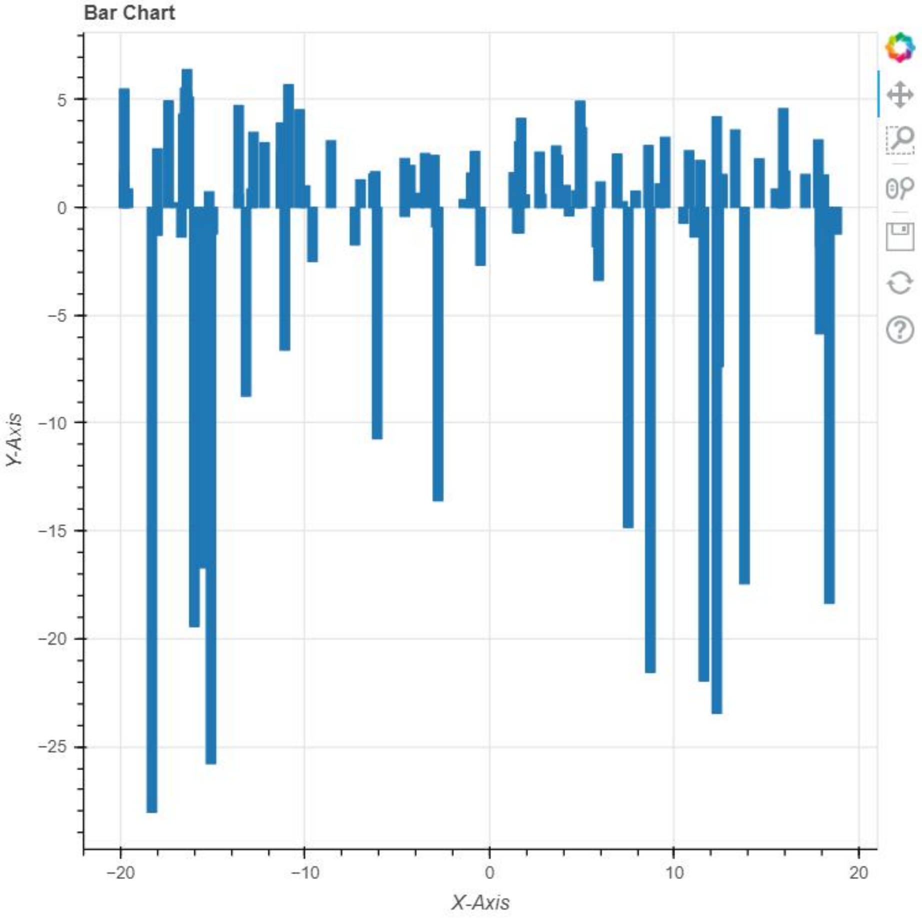
Test Data Set

```
from bokeh.plotting import figure, show
from bokeh.models import ColumnDataSource
import pandas as pd
df = pd.read_csv("test_cal.csv")
source = ColumnDataSource(df)
# Line Chart
p = figure(title="Line Chart", x_axis_label='X-Axis', y_axis_label='Y-Axis',)
p.line(x='x', y='y', source=source)
show(p)

# Bar Chart
source1 = ColumnDataSource(df)
p = figure(title="Bar Chart", x_axis_label='X-Axis', y_axis_label='Y-Axis')
p.vbar(x='x', top='y', source=source1, width=0.5)
show(p)

# Scatter Chart
source2 = ColumnDataSource(df)
p = figure(title="Scatter Chart", x_axis_label='X-Axis', y_axis_label='Y-Axis')
p.scatter(x='x', y='y', source=source2)
show(p)
```

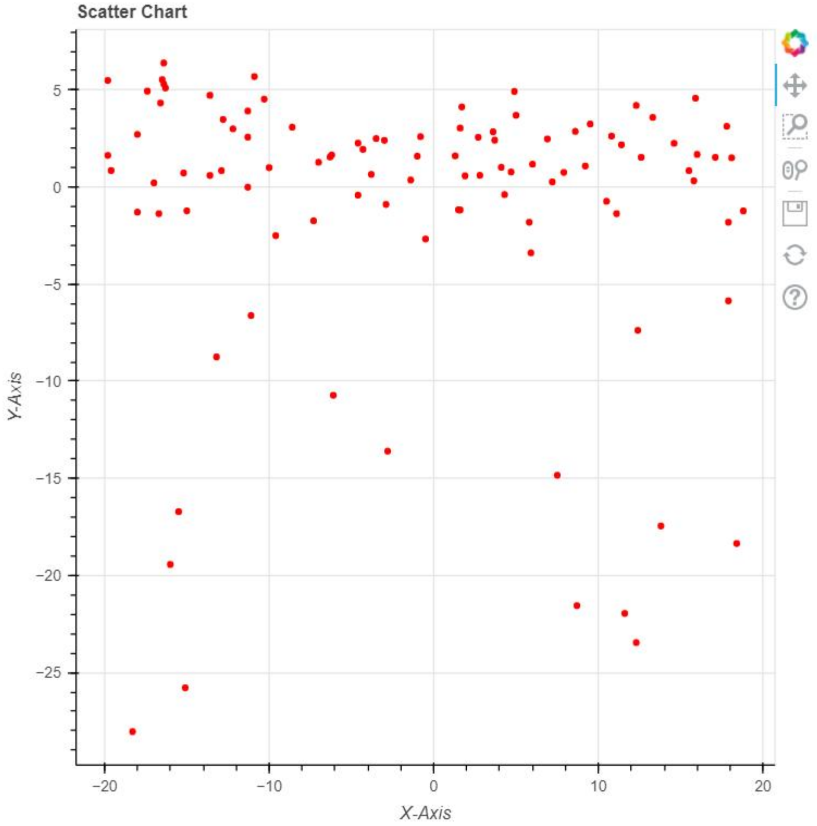
Line Chart



Bar Graph



Scatter Graph



How to run the project

This assignment application was developed in windows 10 with Python 3.11 installed. Below are the steps to run the project and applications and libraries installed are : -

PyMySQL => 1.0.2
pandas => 1.4.4
pandas-bokeh => 0.5.5
bokeh => 2.4.3
mysql => 0.0.3
SQLAlchemy => 1.4.39
numpy => 1.21.5
importlib-metadata => 4.11.3
jupyterlab =>3.4.4
jupyter => 1.0.0
anaconda-navigator =>2.3.1

1) Open Anaconda-navigator, Select JuypterLab it will open the local host on the computer.
2) Create a “Project” folder add training data_set csv file “train.csv”, test data_set csv file “test.csv”, ideal data_set csv file “ideal.csv”.
3) Square all ‘Y’ data_set in “train.csv”
4) Square all ‘Y’ data_set in “ideal.csv”, sum all rows and then sort the least 4 minimum data_set and save in a file db extension and save to MySQL
5) Find the difference between ideal data_set & least 4 minimum data_set, We get the least minimum data_set from both training data_set csv file “train.csv”and ideal data_set csv file “ideal.csv”.
6) To find the delta data_set, we import test.csv file in pandas dataframe and multiply ‘X’ and ‘Y’ values to get delta_y.
7) For Table 3 we calculate each row value in from ideal data_set ‘no_of_delta_functions’. And concatenate the test.csv columns ‘X’ and ‘Y’ and ‘delta_Y’ and ‘no_of_delta_functions’

Additional Task - Version Control System Github

To use Github to push code and check it:

- Create a Github account
- Create a repository in Github to store your code
- Clone the repository to your local machine using Git
- Write your code and make changes to the local repository
- Stage the changes using *"git add"* command
- Commit the changes with a message describing what was done using *"git commit"* command
- Push the changes to the remote repository using *"git push"* command
- Check the code in the Github repository to verify the changes were successfully pushed

We can use Github Desktop application for Windows or Mac to perform these steps in a graphical interface.

To clone a branch from a remote repository and develop it on local PC using Git:

1. Clone the repository:
   git clone https://github.com/[username]/[repository].git
2. Checkout the develop branch:
   git checkout develop
3. Create a new branch for your new feature:
   git checkout -b new-feature
4. Write code and make changes to the local repository
5. Stage the changes:
   git add [files]
6. Commit the changes with a message:
   git commit -m "Added new function"
7. Push the changes to the remote repository:
   git push origin new-feature
8. Go to the Github repository and create a pull request from the new-feature branch to the develop branch
9. Review the changes and merge the pull request.
   git pull origin develop