Table of Contents

SECTION 1

Introduction

This journal elaborates on a subsection of the assignment that requires selection of 4 ideal function amongst 50 using a training data set. The selection of theses functions are required to correctly map the test data points. These functions provide the best fit for the training data and thus used to calculate the respective deviation for the test data points by following the criteria that the deviation of the mapped points should be less than the square root of the maximum deviation of the regression obtained from the training data set. Discussed in the following sections are the steps taken to obtain the 4 functions. Below is the summary for each of the listed section:

- Data loading stage- The datasets are provided in csv format, which is loaded in-memory by a program module named csvloader.
- Data preparation stage-The data cleansing process is described, which means duplications, NA values and outliers are screened using a csvcleaner module. This step is a preparatory step which gives the final training data set for the calculation.
- Data processing stage- Best fit calculations are explained and the statistical method of least squared error is also outlined. I then discuss the four selected ideal functions and elaborate on some of the properties of these function and why they are the best fit.
- Data visualization stage- The plots of the selected ideal functions against the corresponding matching training data set function are listed. I also highlight the x value for which the deviation between training and ideal function is maximum along with the calculated value.

I conclude by discussing the results of the assignment and establish the correlation of the test data points against the qualifying ideal functions. The least squarees method to compute the ideal function is also elaborated after the Visualizations are presented.

Section 2

Loading Data

The csvloader module supports loading information from the provided csv files. The **load_data** function requires a path to the csv file containing the data and a label which is used to print the file information. I store the file path in unique variable and then call the **print_csv_info** method that loads the csv data to memory and prints information about the file such as-

- size
- shape
- dimensions
- data type

Printing this information, provides us the basic understanding of our data sets. Note the output below the code, we can see that:

- There are 50 ideal functions with 400 rows
- There are 4 training sets with 400 rows
- The data set is two dimensional
- All entries are of type float64

The first row is reserved for the *x* value. The code below uses the csvloader module to print information for the three datasets, it declares the path of the specified files and passes this information to the loader. The output is presented in the footnotes of the code.

In [6]:

```
from core.loaders.csvloader import csvloader;

path_test_data = './data/test.csv'
path_train_data = './data/train.csv'
path_ideal_data = './data/ideal.csv'

def load_data(path, label):
    data_importer = csvloader.CSVLoader(path, label)
    data_importer.read_csv()
    return data_importer

def print_csv_info(path, label):
    loader = load_data(path, label)
    loader.print_csv_info()

print_csv_info(path_test_data, 'test.csv')
print_csv_info(path_train_data, 'train.csv')
print_csv_info(path_ideal_data, 'ideal.csv')
```

```
file information for test.csv: {"size": 200, "shape": [100, 2], "dimensions": 2, "type":
"float64"}
file information for train.csv: {"size": 2000, "shape": [400, 5], "dimensions": 2, "type":
"float64"}
file information for ideal.csv: {"size": 20400, "shape": [400, 51], "dimensions": 2, "typ
e": "float64"}
```

The cvs loader module is listed in ***Appendix A***

Section 3

Cleaning Data

The csvcleanser module cleans and prepares the data for analysis. It removes any NA entries and prepares a sorted data set by the first column. Values that are 2* standard deviations from the mean are considered outliers.

The code below uses the csvcleanser module to clean the three datasets and it print various information such as num of duplicates. This information is listed in the footnote of the program below.

It is evident that the provided datasets do not have any duplicate or NA values. Additionally as part of data-preparation process the module also sorts the datasets.

In [7]:

```
from core.cleanser.csvcleanser import csvcleanser;

def print_cleansing_info(path, label):
    loader = load_data(path, label)
    data_cleanser = csvcleanser.CSVCleanser(loader.csv_data, loader.label)
    data_cleanser.print_cleansing_info()

print_cleansing_info(path_test_data, 'test.csv')
print_cleansing_info(path_train_data, 'train.csv')
print_cleansing_info(path_ideal_data, 'ideal.csv')
```

```
cleaning file information for test.csv: {"columns_with_na": [], "num_columns": 2, "sorted_
by_index": 0, "removed_duplicated": 0}
total outliers for test.csv: 11


cleaning file information for train.csv: {"columns_with_na": [], "num_columns": 5, "sorted
_by_index": 0, "removed_duplicated": 0}
total outliers for train.csv: 84


cleaning file information for ideal.csv: {"columns_with_na": [], "num_columns": 51, "sorte
d_by_index": 0, "removed_duplicated": 0}
total outliers for ideal.csv: 502
```

*The csvcleaner module is presented in **Appendix B***

Section 4

Best Fit Calculations

After loading and cleaning the training data set, the least squared error method defined in out core.stats module is used to find the best fit for the 4 y columns in the training data against the 50 ideal functions provided. The least square method is the process of finding the best-fitting curve for a set of data points by reducing the sum of the squares of the offsets of the points from the curve.

The Stats module is listed in ***Appendix C***. This module contains the least squares method that returns a key-value pair dictionary, where the key is the column of the training function in the training data set and the value is index of the ideal function in the ideal data set.

Below table lists the selected ideal function against the corresponding training function.

Training Function	Ideal Function
1	10
2	26
3	8
4	25

In [8]:

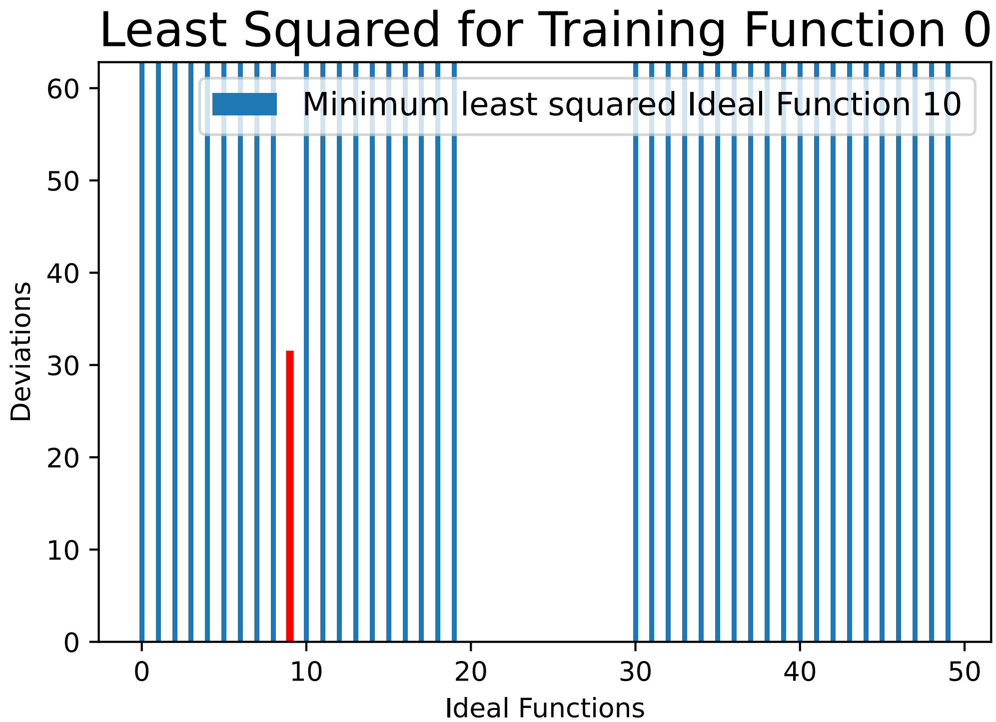
```
from core.stats import stats
import matplotlib.pyplot as plt
import json

stat = stats.Stat()

train_data_importer = load_data(path_train_data, 'train.csv')
train_data_cleanser = csvcleanser.CSVCleanser(train_data_importer.csv_data, train_data_imp
#remove outliers
# train_data_cleanser.remove_all_outliers()
trainData = train_data_cleanser.df

ideal_data_importer = load_data(path_ideal_data, 'ideal.csv')
ideal_data_cleanser = csvcleanser.CSVCleanser(ideal_data_importer.csv_data, ideal_data_imp
#remove outliers
# ideal_data_cleanser.remove_all_outliers()
idealData = ideal_data_cleanser.df

best_fit = stat.leastSquare(trainData[0].to_numpy(), trainData, idealData)
```

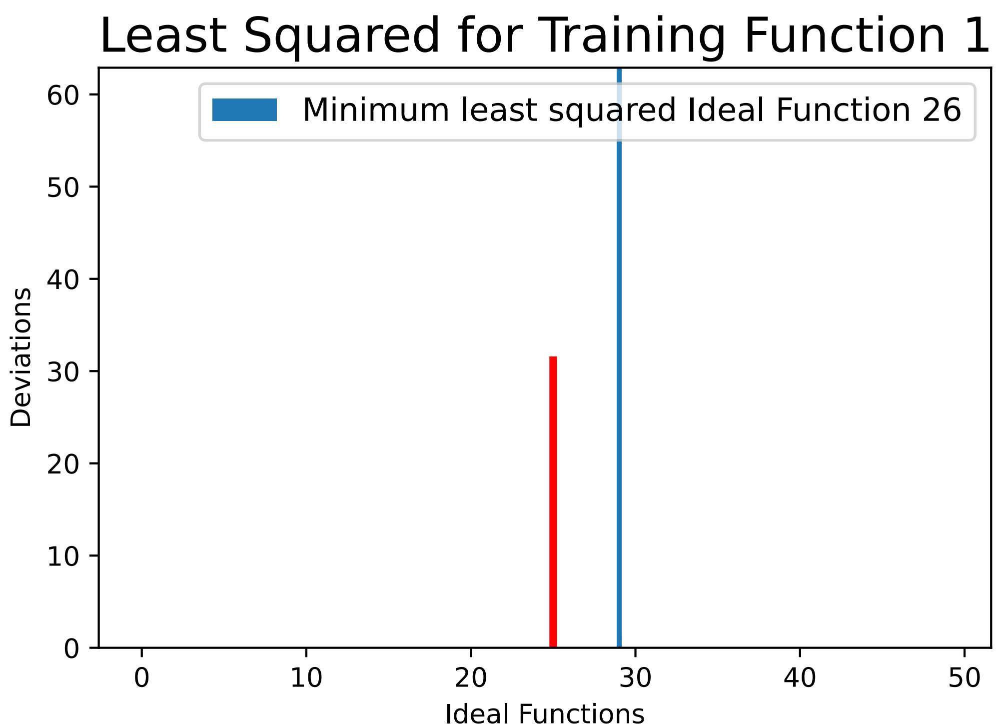
Least Square:

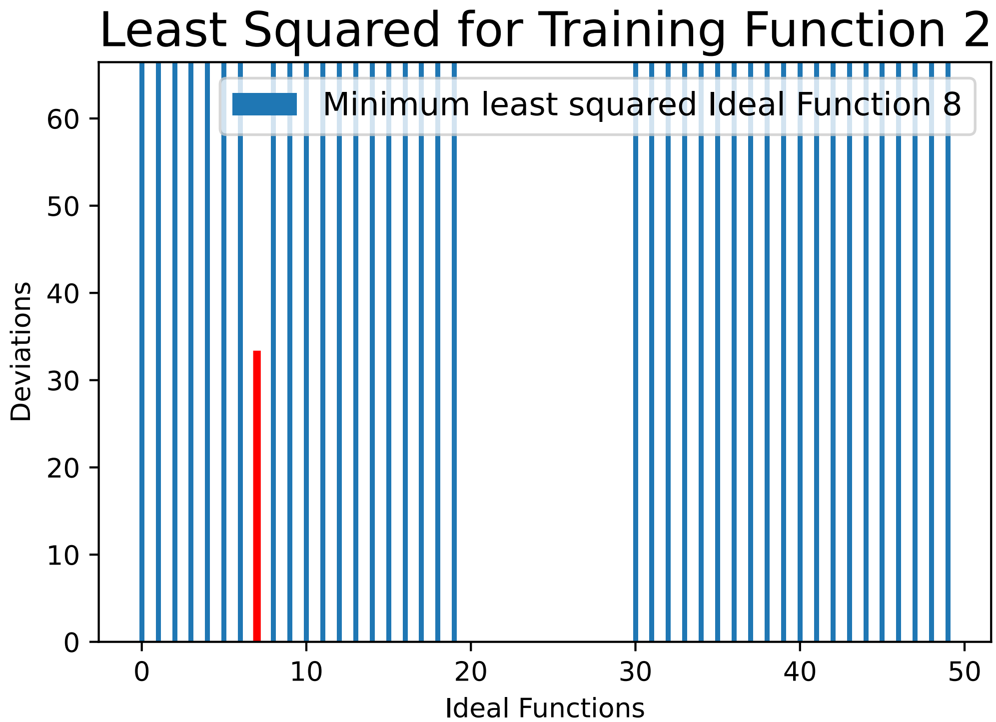
The method of least squares is a standard approach in regression analysis to approximate the solution of best fit in systems by minimizing the sum of the squares of the residuals made in the results of every single equation. Line of best fit refers to a line through a scatter plot of data points that best expresses the relationship between those points. Statisticians typically use the least squares method to arrive at the geometric equation for the line, either though manual calculations or regression analysis software. In the assignment we use the least square method to determine the ideal function instead of a regression line that best fits the training data set. The given data points are to be minimized by the method of reducing residuals or offsets of each point from the line. I do this by calculating the deviation of each training data point against the corresponding mapping in the ideal data set. Then instead of caluclating the sum we use squared values of these deviations and conclude the most fitting function which has the least squared deviation. In the blots we range limit the y-values of deviation to 2 * (lowest deviation value)

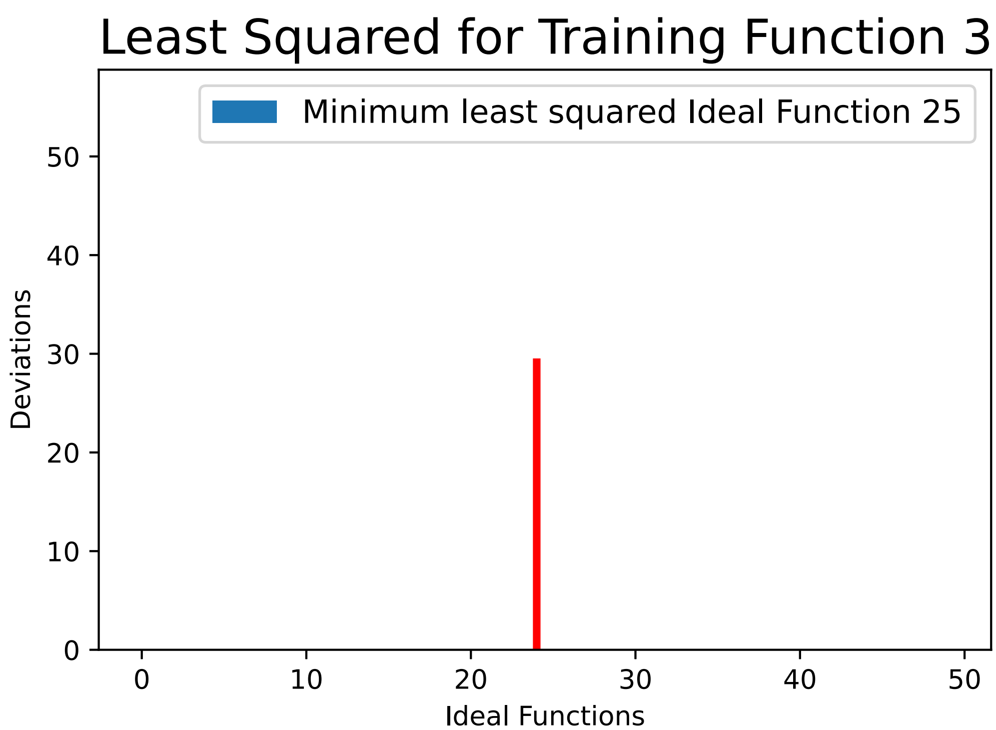
In [9]:

```
import math;
import numpy as np;
def leastSquare(x, fn1, fn2):
        """
            loops over the overlapping rows and calculates the least squared deviation bet
            this is used by the train and ideal functions to find the minimum squared devi
        """
        #filter values of x in 2 that are not in 1
        fn2 = fn2[fn2[0].isin(x)]
        info = {}
        deviation_info = []
        for i in range(len(fn1.columns)):
            if(i == 0): continue
            minSum = math.inf
            deviations = []
            for j in range(len(fn2.columns)):
                sum = 0
                if j == 0: continue
                y1 = fn1[i].to_numpy()
                y2 = fn2[j].to_numpy()
                for k in range(len(y1)):
                    if(k < len(y2)):
                        sum += (y1[k]-y2[k]) * (y1[k]-y2[k])
                deviations.append(sum)
                if (sum < minSum):
                    minSum = sum
                    info[i] = j
            deviation_info.append(deviations)
        info_json = json.dumps(info)

        for i in range(len(deviation_info)):
            deviations = deviation_info[i]
            low = min(deviations)
            low_index = deviations.index(low)
            plt.ylim([0, low * 2])
            barchart = plt.bar(range(len(deviations)), deviations, 0.3)
            barchart[low_index].set_color('r')
            plt.xlabel("Ideal Functions")
            plt.ylabel("Deviations")
            plt.legend(labels=['Minimum least squared Ideal Function {}'.format(low_index
            plt.title('Least Squared for Training Function {}'.format(i), size=18, loc='le
            plt.show()

leastSquare(trainData[0].to_numpy(), trainData, idealData)
```


Least Squared for Training Function 0
Minimum least squared Ideal Function 10
Deviations
Ideal Functions
0
10
20
30
40
50
60


Least Squared for Training Function 1
Minimum least squared Ideal Function 26
Deviations
Ideal Functions
0
10
20
30
40
50
60





In [ ]:

```
There is a print error, due to conversion of pdf. Diagram above
(Least squared for Training Function3) is not rendering correctly.
```

Selected Ideal Functions

Training Function	Ideal Function
1	10
2	26
3	8
4	25

Section 5

Plot of train functions against the selected ideal functions

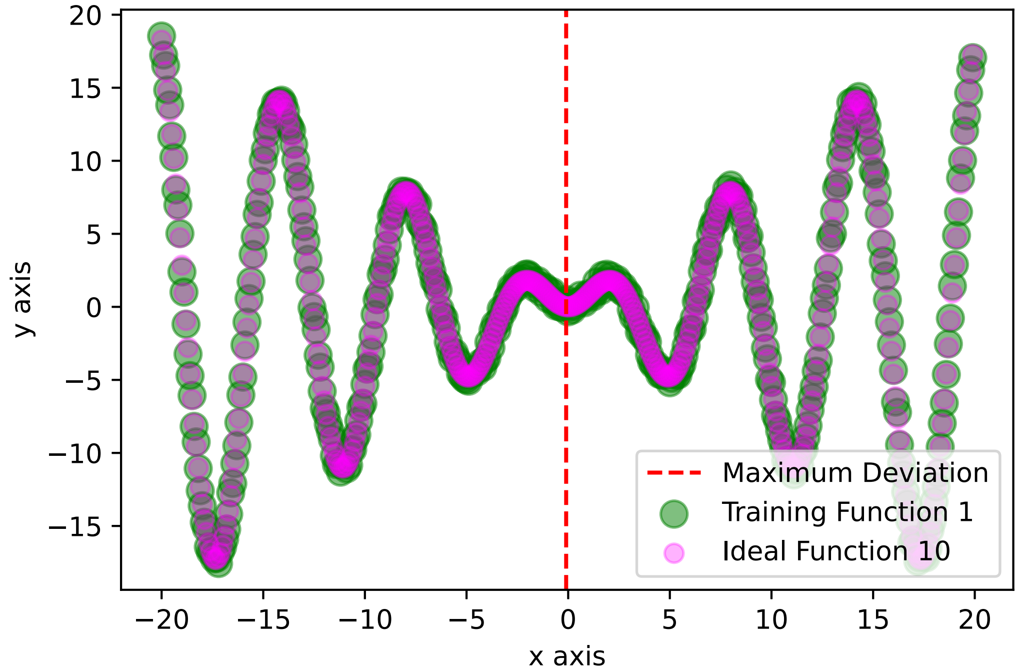
The below program loops over all the key-value pairs selected above and plots the training function and the corresponding ideal function. For each plot the x point for which the maximum deviation is recorded is marked in a dashed red line and the corresponding deviation value is also recorded.

Visualizations suggest that the ideal functions closely model the training function. The value of maximum deviation for all 4 functions is less than **0.5**, which further indicates that the chosed ideal functions are the best fit for the training functions.

In [10]:

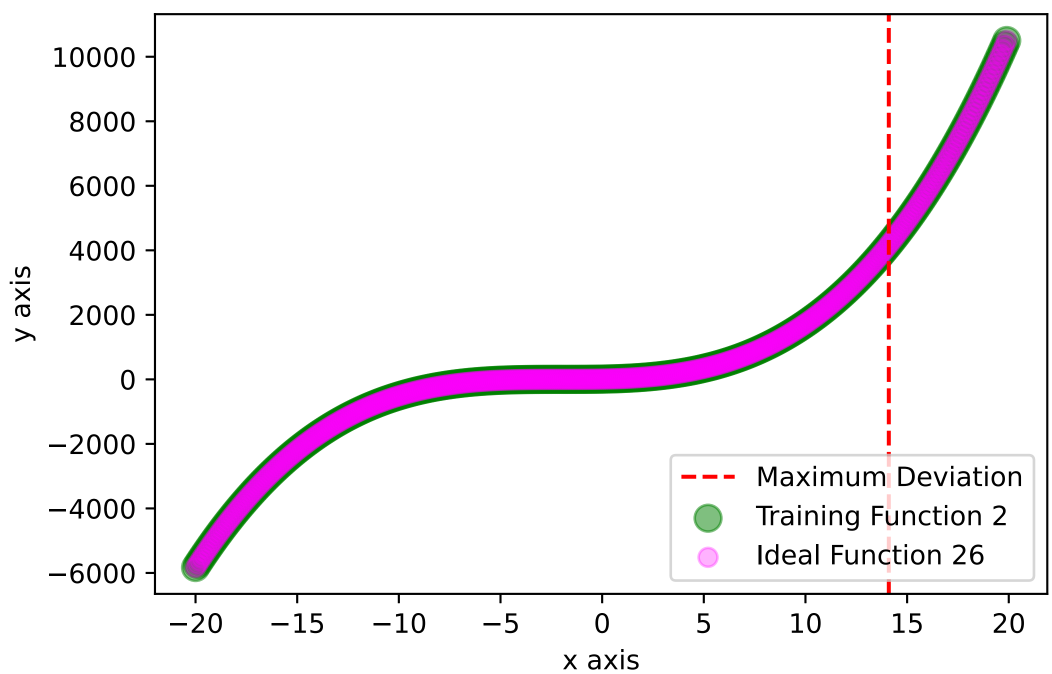
```
import numpy as np;
fit = json.loads(best_fit)
for key in fit:
    yTrain = trainData[int(key)]
    yIdeal = idealData[fit[key]]

    ind_max = np.argmax((yTrain-yIdeal)**2)
    x_max = trainData[0][ind_max]
    y_max = yTrain[ind_max]
    deviation_max = abs(yTrain[ind_max] - yIdeal[ind_max]);

    plt.scatter(y = yTrain, x = trainData[0], alpha = 0.5, color='green', s=100)
    plt.scatter(y = yIdeal, x = idealData[0], color = 'magenta', alpha = 0.3, s=50)
    plt.axvline(x_max, color="red", linestyle="dashed", alpha=1)

    plt.xlabel("x axis")
    plt.ylabel("y axis")
    plt.figtext(0.4, -0.05, 'Maxium Deviaiton: ' + str(round(abs(yTrain[ind_max]-yIdeal[ir
    plt.figtext(0.4, -0.1, 'x at Maxium Deviaiton: ' + str(x_max), size="large", color="re
    plt.legend(labels=['Maximum Deviation', 'Training Function {}'.format(key), 'Ideal Fur
    plt.title('Scatter Plot {}'.format(key), size=18, loc='left')
    plt.show()
```

Scatter Plot 1

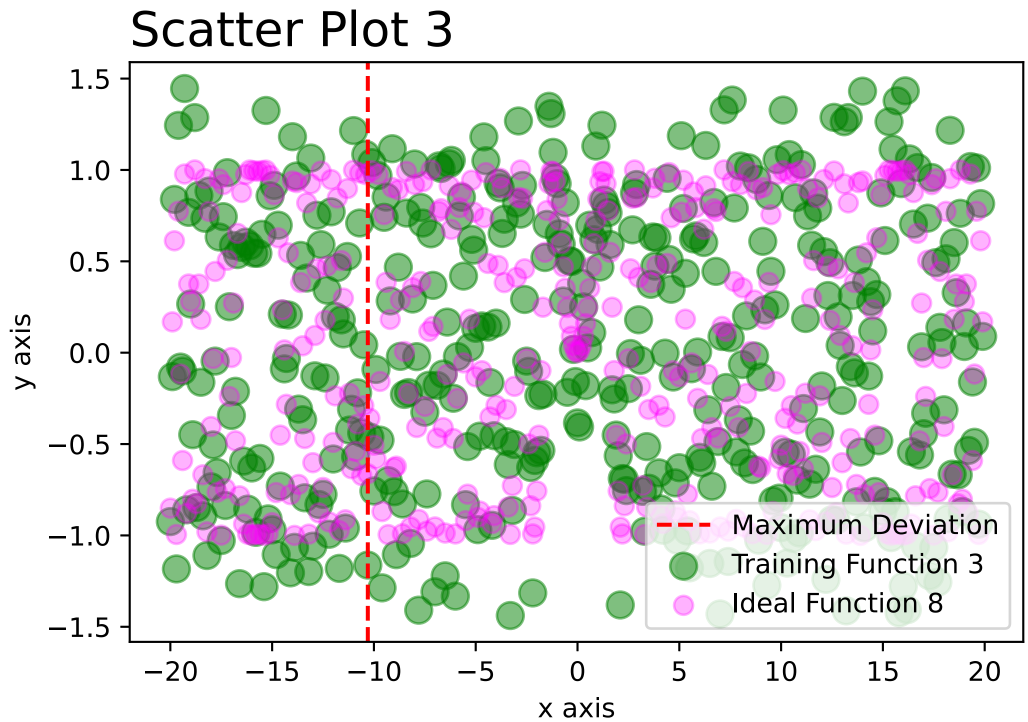


Maxium Deviaiton: 0.4986
x at Maxium Deviaiton: -0.1

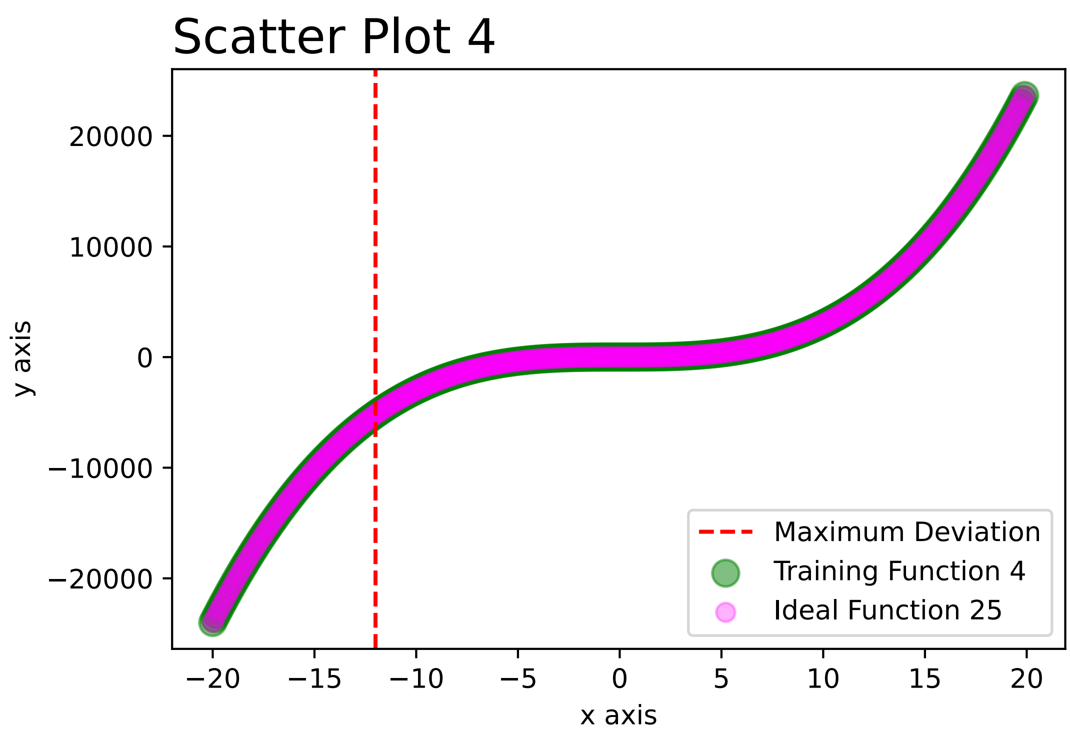
Scatter Plot 2



Maxium Deviaiton: 0.499
x at Maxium Deviaiton: 14.1



Maxium Deviaiton: 0.4999
x at Maxium Deviaiton: -10.3



Maxium Deviaiton: 0.497
x at Maxium Deviaiton: -12.0

*The stats module is presented in **Appendix C***

Section 6

CONCLUSION

As per the analysis of test data, several points map to another point in ideal function. Some points map to multiple ideal functions following the criterion of deviation specified in the assignment. This code snippet calculates the deviation between the test data points against corresponding point in the ideal function and this calculation is performed for each of the selected ideal functions. ``` df['deviation'] = abs(df[2]-df[1]) ``` The criterion of selections requires comparison of the above calculated value against the maximum deviation observed in Section 5. The below table lists all such points with their deviation.

Ideal Function 8

Mapping criterion: deviation < **0.7218556** (sqrt(2) * 0.4999)

FIELD1	x	test-y	ideal-8_y	delta-y
11	-12.9	-0.4529254	0.09427045	0.54719585
14	-11.7	-0.9947322	-0.9735021	0.021230099999999992
21	-8.3	0.31644812	-0.22314377	0.53959189
24	-6.6	-0.2251311	-0.4098569	0.18472580000000002
30	-4.6	0.62587744	0.73870605	0.11282861
37	-2.8	0.7222169	0.99990225	0.27768534999999994
41	-1.5	0.29984066	0.7780732	0.47823254000000004
44	-0.9	0.9397264	0.72428715	0.21543924999999997
46	-0.4	-0.07180414	0.15931821	0.23112234999999998
47	-0.1	0.26104912	0.009999833	0.251049287
51	1.4	0.31926435	0.92521155	0.6059472
66	10.4	0.28039157	0.974806	0.6944144299999999
70	12.2	-1.0703908	-0.9265537	0.14383709999999994
75	15.2	-0.70067745	-0.9911765	0.29049905

Ideal Function 10

Mapping criterion: deviation < **0.717984** (sqrt(2) * 0.4986)

FIELD1	x	test-y	ideal-10_y	delta-y(deviation)
3	-17.5	-17.11462	-17.073456	0.041163999999998424
7	-16.0	-4.8444533	-4.606453	0.2380002999999995
13	-12.5	-1.4725168	-0.8290237	0.6434930999999999
17	-9.7	-2.7348197	-2.6360781	0.09874159999999987
18	-9.0	4.3709817	3.7090664	0.6619152999999995
28	-5.0	-4.8346496	-4.7946215	0.04002809999999979
37	-2.8	0.7222169	0.9379668	0.21574989999999994
44	-0.9	0.9397264	0.7049942	0.23473219999999995
46	-0.4	-0.07180414	0.15576734	0.22757148
47	-0.1	0.26104912	0.009983341	0.25106577900000004
54	3.6	-2.097083	-1.5930736	0.5040093999999999
65	9.1	3.0492904	2.903795	0.14549539999999972
68	11.0	-10.541548	-10.999892	0.45834399999999853
77	15.2	7.320115	7.39326	0.07314499999999935

Ideal Function 25

Mapping criterion: deviation < **0.71568** (sqrt(2) * 0.497)

FIELD1	x	test-y	ideal-25_y	delta-y
8	-15.5	-11166.14	-11166.625	0.4850000000005821
33	-4.1	-201.09651	-201.763	0.6664900000000102
36	-2.9	-68.749565	-68.167	0.5825650000000024
42	-1.0	2.1515043	2.0	0.15150430000000004
50	1.1	9.401569	8.993	0.40856899999999996
60	6.7	906.75336	907.289	0.5356399999999439
74	15.0	10130.192	10130.0	0.19199999999909778
82	17.3	15538.486	15538.151	0.3350000000009459

Ideal Function 26

Mapping criterion: deviation < **0.7218556** (sqrt(2) * 0.4999)

FIELD1	x	test-y	ideal-26_y	delta-y.
9	-14.9	-2146.1052	-2146.689	0.5837999999998829
25	-5.8	-55.442513	-54.872	0.5705129999999983
32	-4.3	-11.712081	-12.167	0.4549190000000003
35	-3.5	-2.7388372	-3.375	0.6361628000000001
38	-2.6	-0.84336996	-0.216	0.62736996
40	-1.6	-0.22908556	0.064	0.29308555999999997
41	-1.5	0.29984066	0.125	0.17484065999999998
44	-0.9	0.9397264	1.331	0.3912736
52	1.8	54.41447	54.872	0.45752999999999844
58	5.6	439.5265	438.976	0.5504999999999995
62	7.3	804.66425	804.357	0.3072500000000673
63	7.8	941.35315	941.192	0.16115000000002055
67	10.7	2047.6775	2048.383	0.7054999999998017
84	18.0	8000.475	8000.0	0.4750000000003638

RESULTS

Each x point in the test data maps to mutliple points int the ideal function as listed below.

Index	x	count	Ideal Function
5	-17.5	1.0	10
10	-16.0	1.0	10
16	-12.5	1.0	10
21	-9.7	1.0	10
23	-9.0	1.0	10
34	-5.0	1.0	10
43	-2.8	1.0	10
51	-0.9	1.0	10
53	-0.4	1.0	10
54	-0.1	1.0	10
61	3.6	1.0	10
73	9.1	1.0	10
76	11.0	1.0	10
86	15.2	1.0	10
12	-14.9	1.0	26
31	-5.8	1.0	26
37	-4.3	1.0	26
41	-3.5	1.0	26
44	-2.6	1.0	26
47	-1.6	1.0	26
48	-1.5	1.0	26
51	-0.9	1.0	26
59	1.8	1.0	26
66	5.6	1.0	26
70	7.3	1.0	26
71	7.8	1.0	26
75	10.7	1.0	26
93	18.0	1.0	26
14	-12.9	1.0	8
17	-11.7	1.0	8
27	-8.3	1.0	8
30	-6.6	1.0	8
36	-4.6	1.0	8
43	-2.8	1.0	8

Index	x	count	Ideal Function
48	-1.5	1.0	8
51	-0.9	1.0	8
53	-0.4	1.0	8
54	-0.1	1.0	8
58	1.4	1.0	8
74	10.4	1.0	8
78	12.2	1.0	8
84	15.2	1.0	8
11	-15.5	1.0	25
39	-4.1	1.0	25
42	-2.9	1.0	25
50	-1.0	1.0	25
57	1.1	1.0	25
68	6.7	1.0	25
83	15.0	1.0	25
91	17.3	1.0	25
5	-17.5	1.0	10
10	-16.0	1.0	10
16	-12.5	1.0	10
21	-9.7	1.0	10
23	-9.0	1.0	10
34	-5.0	1.0	10
43	-2.8	1.0	10
51	-0.9	1.0	10
53	-0.4	1.0	10
54	-0.1	1.0	10
61	3.6	1.0	10
73	9.1	1.0	10
76	11.0	1.0	10
86	15.2	1.0	10
12	-14.9	1.0	26
31	-5.8	1.0	26
37	-4.3	1.0	26
41	-3.5	1.0	26
44	-2.6	1.0	26
47	-1.6	1.0	26
48	-1.5	1.0	26
51	-0.9	1.0	26

Index	x	count	Ideal Function
59	1.8	1.0	26
66	5.6	1.0	26
70	7.3	1.0	26
71	7.8	1.0	26
75	10.7	1.0	26
93	18.0	1.0	26
14	-12.9	1.0	8
17	-11.7	1.0	8
27	-8.3	1.0	8
30	-6.6	1.0	8
36	-4.6	1.0	8
43	-2.8	1.0	8
48	-1.5	1.0	8
51	-0.9	1.0	8
53	-0.4	1.0	8
54	-0.1	1.0	8
58	1.4	1.0	8
74	10.4	1.0	8
78	12.2	1.0	8
84	15.2	1.0	8
11	-15.5	1.0	25
39	-4.1	1.0	25
42	-2.9	1.0	25
50	-1.0	1.0	25
57	1.1	1.0	25
68	6.7	1.0	25
83	15.0	1.0	25
91	17.3	1.0	25

Summary of total matched points:

x	Total Matches
-17.5	1
-16	1
-12.5	1
-9.7	1
-9	1
-5	1
-2.8	1
-0.9	1
-0.4	1
-0.1	1
3.6	1
9.1	1
11	1
15.2	1
-14.9	1
-5.8	1
-4.3	1
-3.5	1
-2.6	1
-1.6	1
-1.5	1
-0.9	2
1.8	1
5.6	1
7.3	1
7.8	1
10.7	1
18	1
-12.9	1
-11.7	1
-8.3	1
-6.6	1
-4.6	1
-2.8	2
-1.5	2

x	Total Matches
-0.9	3
-0.4	2
-0.1	2
1.4	1
10.4	1
12.2	1
15.2	2
-15.5	1
-4.1	1
-2.9	1
-1	1
1.1	1
6.7	1
15	1
17.3	1

APPENDIX

Section A

CSV Loader Module

```
from numpy import genfromtxt
import logging
import traceback
import json

class CSVLoader:
    """
        Modules loads data from csv file in a pandas dataframe
        the csv_info method can be used to generate information about the
data
    """
    def __init__(self, path, label):
        self.csv_file_path = path
        self.label = label

    def read_csv(self):
        data = genfromtxt(self.csv_file_path, delimiter=',',
skip_header=True, converters={0: lambda num: float(num)})
        self.csv_data = data
        self.csv_data_info = self.csv_info(data)

    def csv_info(self, data):
        info = {}
        try:
            info['size'] = data.size
            info['shape'] = data.shape
            info['dimensions'] = data.ndim
            info['type'] = str(data.dtype)
            info_json = json.dumps(info)
        except Exception:
            logging.exception('unable to read data from CSV file: %s',
data)
            traceback.print_exc()
            raise
        return info_json

    def print_csv_info(self):
        print('file information for {}: {}'.format(self.label,
self.csv_data_info))
```

Section B

CSV Cleanser Module

```
import logging
import traceback
import json
import pandas as pd
from numpy import mean
from numpy import std

class CSVCleanser:
    """
        This module provides functionality to clean the csv data loaded
using the loader module
    """
    def __init__(self, ndarray, label):
        self.data = ndarray
        self.label = label
        self.df = pd.DataFrame(self.data)


    def __clean_info(self):
        """
            provides information about the data after
                removing duplicates
                remonin na
                sroting by first column
        """
        info = {}
        try:
            df = self.df
            df.sort_values(0, inplace=True)
            len_before_removing_duplicates = len(df)
            df.drop_duplicates(inplace=True)
            len_after_removing_duplicates = len(df)
            info['columns_with_na'] = df.columns[df.isna().any()].tolist()
            info['num_columns'] = len(df.columns)
            info['sorted_by_index'] = 0
            info['removed_duplicated'] = len_before_removing_duplicates -
len_after_removing_duplicates
            info_json = json.dumps(info)
        except Exception:
            logging.exception('unable to read cleansing info data of %s
from data frame: %s', self.label)
            traceback.print_exc()
            raise
        return info_json

    def __outlier_info(self, col):
        """
            private function
            finds outliers using mean and deviation
        """
        df = pd.DataFrame(self.data)
        data = df[col].to_numpy()
```

```
        # calculate summary statistics
        # value which are more than 2 standard deviations difference are
considered outliers.
        data_mean, data_std = mean(data), std(data)
        # identify outliers
        cut_off = data_std * 2
        lower, upper = data_mean - cut_off, data_mean + cut_off

        outliers = [x for x in data if x < lower or x > upper]
        return outliers

    def __filter_rows_by_values(self, df, col, values):
        return df[~df[col].isin(values)]

    def removeOutliers(self, col):
        """
            removes outliers using the __filter_rows_by_values private
function utility
        """
        self.df.sort_values(0, inplace=True)
        self.df.drop_duplicates(inplace=True)
        outliers = self.__outlier_info(col)
        self.df = self.__filter_rows_by_values(self.df, col, outliers)

    def print_cleansing_info(self):
        print('cleaning file information for {}: {}'.format(self.label,
self.__clean_info()))
```

Section C

Stat Module (Statistical)

```
import logging
import json
import pandas as pd
import math
import numpy as np

class Stat:
    def __init__(self):
        logging.info('stats package initialized')

    def leastSquare(self, x, fn1, fn2):
        """
            loops over the overllaping rows
            and caluclates the least squared deviation
            between two functions

            this is used by the train and ideal functions
            to find the minimum squared deviation
        """
        #filter values of x in 2 that are not in 1
        fn2 = fn2[fn2[0].isin(x)]
        info = {}
        for i in range(len(fn1.columns)):
            if(i == 0): continue
            minSum = math.inf
            for j in range(len(fn2.columns)):
                sum = 0
                if j == 0: continue
                y1 = fn1[i].to_numpy()
                y2 = fn2[j].to_numpy()
                for k in range(len(y1)):
                    sum += (y1[k]-y2[k]) * (y1[k]-y2[k])
                if (sum < minSum):
                    minSum = sum
                    info[i] = j

        info_json = json.dumps(info)
        return info_json

    def maximumDeviationOfRegression(self, df1, df2, df1_column,
df2_column):
        """
            calculates the maximum deviation for two data frames
            accross the two provided columns
        """
        df = df1.merge(df2, on=0, how='left')
        df.dropna(inplace=True)
        return np.max(np.abs(df[df1_column] - df[df2_column]))

    def differentialDeviation(self, df1, df2):
        """
            generates the difference in y values for two data frames
            and loads them into a new column called 'deviation'
```

```
"""
df2 = df2[df2[0].isin(df1[0])]
df = df1.merge(df2, on=0, how='left')
df.columns = [0, 1, 2]
df['deviation'] = abs(df[2]-df[1])
return df
```

Section D

Main Program

```
from core.loaders.csvloader import csvloader;
from core.cleanser.csvcleanser import csvcleanser;
from core.stats import stats;
import json
import math
from functools import reduce
import pandas as pd
import sqlalchemy


path_test_data = './data/test.csv'
path_train_data = './data/train.csv'
path_ideal_data = './data/ideal.csv'

def testDataAnalysis(deviations, max_deviation, key):
    """
        this function uses the criterion for selection of an ideal function
        and drops all the deviations are more the sqrt(2) of the caluclated
        deviation of regression found by using the train functions
    """
    df = deviations
    df = df.drop(df[df['deviation'] >  max_deviation * math.sqrt(2)].index)
    df.columns = ['x', 'test-y', key, 'delta-y_'+key]
    return df

def main():
    stat = stats.Stat()


    # load train data set
    train_data_importer = csvloader.CSVLoader(path_train_data, 'train.csv')
    train_data_importer.read_csv()
    train_data_importer.print_csv_info()

    #clean train data set
    train_data_cleanser =
csvcleanser.CSVCleanser(train_data_importer.csv_data,
train_data_importer.label)
    train_data_cleanser.print_cleansing_info()
    train_data_cleanser.remove_all_outliers()
    trainData = train_data_cleanser.df

    #load ideal data set
    ideal_data_importer = csvloader.CSVLoader(path_ideal_data, 'ideal.csv')
    ideal_data_importer.read_csv()
    ideal_data_importer.print_csv_info()

    #clean ideal data set
    ideal_data_cleanser =
csvcleanser.CSVCleanser(ideal_data_importer.csv_data,
ideal_data_importer.label)
    ideal_data_cleanser.print_cleansing_info()
    ideal_data_importer.remove_all_outliers()
```

```
    #generate best fit
    idealData = ideal_data_cleanser.df
    best_fit = stat.leastSquare(trainData[0].to_numpy(), trainData,
idealData)

    #calculate maximum deviation of regression
    fit = json.loads(best_fit)
    max_deviation = {}
    for key in fit:
        train_modified = trainData[[0,int(key)]].copy()
        ideal_modified = idealData[[0, fit[key]]].copy()
        max_deviation[key] =
stat.maximumDeviationOfRegression(train_modified, ideal_modified, int(key),
fit[key])

    # load test data and clean
    test_data_importer = csvloader.CSVLoader(path_test_data, 'test.csv')
    test_data_importer.read_csv()
    test_data_importer.print_csv_info()

    test_data_cleanser =
csvcleanser.CSVCleanser(test_data_importer.csv_data,
test_data_importer.label)
    test_data_cleanser.print_cleansing_info()
    test_data_cleanser.remove_all_outliers()

    testData = test_data_cleanser.df

    """
        Calculate the differential for each point in the test data
        and store the result in a csv per ideal function
        output director has 4 files each correspoding to 1 ideal function
    """
    data_frames = []
    for i in range(len(testData.columns)):
        for key in fit:
            ideal_modified = idealData[[0, fit[key]]].copy()
            deviations = stat.differentialDeviation(testData,
ideal_modified)
            matchDf = testDataAnalysis(deviations, max_deviation[key],
'ideal-'+str(fit[key])+'_y')
            matchDf.to_csv('data/csv/output/' + 'ideal-'+str(fit[key]) +
'.csv')
            data_frames.append(matchDf)

    """
        The merged data frames combines the y devations calulations
        across all the ideal functions
        output.cs contains the total information for
        all test data points accross the 4 selected ideal functions
    """
    df_merged = pd.concat(data_frames)
    df_merged_temp = df_merged.drop(['x','test-y'],axis=1)
    count = df_merged_temp.loc[:].count(axis=1)
    df_merged['count'] = count
    df_merged.to_csv('data/csv/output/output.csv')
```

```
    # Create the engine to connect to the PostgreSQL database
    engine =
sqlalchemy.create_engine('postgresql://sh.kumar:password@localhost:5432/sqlassignm


    data8 = pd.read_csv('data/csv/output/ideal-8.csv')
    data10 = pd.read_csv('data/csv/output/ideal-10.csv')
    data25 = pd.read_csv('data/csv/output/ideal-25.csv')
    data26 = pd.read_csv('data/csv/output/ideal-26.csv')

    # Write data into the table in PostgreSQL database
    data8.to_sql('ideal-8',engine)
    data10.to_sql('idea-10',engine)
    data25.to_sql('ideal-25',engine)
    data26.to_sql('ideal-26',engine)


if __name__ == '__main__':
    main()
```

Section E

Set up

```
1. Clone the develop branch.
https://github.com/ivegotwings/IUBH_PYTHON_ASSIGNMENT/tree/develop
2. Install the required packages
    pandas
    numpy
    matplotlib
3. Git commands-
    1. git co origin/develop
    2. git co -b temp-written-assignment
    3. add files to commit  git add.
    4. commit using- git commit -m "commit message"
    5. push using- git push temp-written-assignment
```

conda version : 4.10.3 Use miniconda to set up an environment

Or you can install these packages

packages in environment at /Users/sh.kumar/opt/miniconda3:

```
# packages in environment at /Users/sh.kumar/opt/miniconda3:
#
# Name                    Version                   Build  Channel
appdirs                   1.4.4                    pypi_0    pypi
appnope                   0.1.2           py39hecd8cb5_1001
attrs                     21.2.0                   pypi_0    pypi
backcall                  0.2.0              pyhd3eb1b0_0
blas                      1.0                         mkl
bleach                    4.1.0                    pypi_0    pypi
bottleneck                1.3.2            py39he3068b8_1
brotlipy                  0.7.0           py39h9ed2024_1003
ca-certificates           2021.10.26           hecd8cb5_2
certifi                   2021.10.8        py39hecd8cb5_0
cffi                      1.14.6           py39h2125817_0
charset-normalizer        2.0.4              pyhd3eb1b0_0
conda                     4.10.3           py39hecd8cb5_0
conda-package-handling    1.7.3            py39h9ed2024_1
cryptography              3.4.7            py39h2fd3fbb_0
cycler                    0.10.0                     py_2    conda-forge
debugpy                   1.4.1            py39h23ab428_0
decorator                 5.1.0              pyhd3eb1b0_0
defusedxml                0.7.1                    pypi_0    pypi
entrypoints               0.3              py39hecd8cb5_0
freetype                  2.10.4               h4cff582_1    conda-forge
greenlet                  1.1.1            py39h23ab428_0
idna                      3.2                pyhd3eb1b0_0
importlib-metadata        4.8.1                    pypi_0    pypi
intel-openmp              2021.3.0          hecd8cb5_3375
ipykernel                 6.4.1                    pypi_0    pypi
```

```
ipython                   7.27.0           py39h01d92e1_0
ipython-genutils          0.2.0                    pypi_0    pypi
jbig                      2.1               h0d85af4_2003    conda-forge
jedi                      0.18.0           py39hecd8cb5_1
jinja2                    3.0.2                    pypi_0    pypi
jpeg                      9d                 hbcb3906_0    conda-forge
jsonschema                4.1.0                    pypi_0    pypi
jupyter_client            7.0.1              pyhd3eb1b0_0
jupyter_core              4.7.1            py39hecd8cb5_0
jupyterlab-pygments       0.1.2                    pypi_0    pypi
kiwisolver                1.3.2            py39hf018cea_0    conda-forge
krb5                      1.19.2               hcd88c3b_0
lcms2                     2.12                 h577c468_0    conda-forge
lerc                      2.2.1                h046ec9c_0    conda-forge
libcxx                    12.0.0               h2f01273_0
libdeflate                1.7                  h35c211d_5    conda-forge
libedit                   3.1.20210714         h9ed2024_0
libffi                    3.3                  hb1e8313_2
libpng                    1.6.37               h7cec526_2    conda-forge
libpq                     12.2                 h1b4eb34_1
libsodium                 1.0.18               h1de35cc_0
libtiff                   4.3.0                h1167814_1    conda-forge
libwebp-base              1.2.1                h0d85af4_0    conda-forge
lz4-c                     1.9.3                he49afe7_1    conda-forge
markupsafe                2.0.1                    pypi_0    pypi
matplotlib                3.4.3            py39h6e9494a_1    conda-forge
matplotlib-base           3.4.3            py39hb07454d_1    conda-forge
matplotlib-inline         0.1.2              pyhd3eb1b0_2
mistune                   0.8.4                    pypi_0    pypi
mkl                       2021.3.0           hecd8cb5_517
mkl-service               2.4.0            py39h9ed2024_0
mkl_fft                   1.3.0            py39h4a7008c_2
mkl_random                1.2.2            py39hb2f4e1b_0
nbclient                  0.5.4                    pypi_0    pypi
nbconvert                 6.2.0                    pypi_0    pypi
nbformat                  5.1.3                    pypi_0    pypi
ncurses                   6.2                  h0a44026_1
nest-asyncio              1.5.1              pyhd3eb1b0_0
notebook-as-pdf           0.5.0                    pypi_0    pypi
numexpr                   2.7.3            py39h5873af2_1
numpy                     1.20.3           py39h4b4dc7a_0
numpy-base                1.20.3           py39he0bd621_0
olefile                   0.46               pyh9f0ad1d_1    conda-forge
openjpeg                  2.4.0                h6e7aa92_1    conda-forge
openssl                   1.1.1l               h9ed2024_0
packaging                 21.0                     pypi_0    pypi
pandas                    1.3.3            py39h5008ddb_0
pandocfilters             1.5.0                    pypi_0    pypi
parso                     0.8.2              pyhd3eb1b0_0
pexpect                   4.8.0              pyhd3eb1b0_3
pickleshare               0.7.5           pyhd3eb1b0_1003
pillow                    8.3.2            py39he9bb72f_0    conda-forge
pip                       21.2.4           py37hecd8cb5_0
prompt-toolkit            3.0.17             pyhca03da5_0
psycopg2                  2.8.6            py39hbcfaee0_1
ptyprocess                0.7.0              pyhd3eb1b0_2
pycosat                   0.6.3            py39h9ed2024_0
pycparser                 2.20                       py_2
```

```
pyee                      8.2.2                    pypi_0    pypi
pygments                  2.10.0             pyhd3eb1b0_0
pyopenssl                 20.0.1             pyhd3eb1b0_1
pyparsing                 2.4.7              pyh9f0ad1d_0    conda-forge
pypdf2                    1.26.0                   pypi_0    pypi
pyppeteer                 0.2.6                    pypi_0    pypi
pyrsistent                0.18.0                   pypi_0    pypi
pysocks                   1.7.1            py39hecd8cb5_0
python                    3.9.7                h88f2d9e_1
python-dateutil           2.8.2              pyhd3eb1b0_0
python.app                3                py39h9ed2024_0
python_abi                3.9                     2_cp39    conda-forge
pytz                      2021.1             pyhd3eb1b0_0
pyzmq                     22.3.0                   pypi_0    pypi
readline                  8.1                  h9ed2024_0
requests                  2.26.0             pyhd3eb1b0_0
ruamel_yaml               0.15.100         py39h9ed2024_0
setuptools                58.0.4           py39hecd8cb5_0
six                       1.16.0             pyhd3eb1b0_0
sqlalchemy                1.4.22           py39h9ed2024_0
sqlite                    3.36.0               hce871da_0
testpath                  0.5.0                    pypi_0    pypi
tk                        8.6.11               h7bc2e8c_0
tornado                   6.1              py39h9ed2024_0
tqdm                      4.62.2             pyhd3eb1b0_1
traitlets                 5.1.0              pyhd3eb1b0_0
tzdata                    2021a                h5d7bf9c_0
urllib3                   1.26.6             pyhd3eb1b0_1
wcwidth                   0.2.5              pyhd3eb1b0_0
webencodings              0.5.1                    pypi_0    pypi
websockets                9.1                      pypi_0    pypi
wheel                     0.37.0             pyhd3eb1b0_1
xz                        5.2.5                h1de35cc_0
yaml                      0.2.5                haf1e3a3_0
zeromq                    4.3.4                h23ab428_0
zipp                      3.6.0                    pypi_0    pypi
zlib                      1.2.11               h1de35cc_3
zstd                      1.5.0                h582d3a0_0    conda-forge
```