Programming with Python

(DLMDSPWP01)

Author – Satyapal
Registration number - 9211362

Date - 10th June 2024
Place - Delhi, India

Github: https://github.com/satyads7836/Task-for-Python-Assignment--IUBH

Table of contents

Brief Overview of Task

In doing this task, it will be necessary to manipulate a lot of data, use some statistical techniques, and use Python programming. These include four training datasets, one test dataset and datasets for 50 ideal functions for all the domains. For every dataset, there are two columns of x-y values.

The goal is to write a program in Python that will use the training data to select four out of 50 possible ideal functions with minimal deviation from y (Least-Square method).

Having chosen an ideal function among others, we must determine whether each x-y pair of values can be assigned to any of the chosen ideal functions using test data. If this condition is met, then mapping should be executed by the program together with its deviation.

Data must be displayed properly while creating unit tests if possible. There should be a reasonable design in the Python program using object-oriented principles which has at least one inheritance and standard as well as user-defined exception handlings also depending on certain Python packages.

Besides that, there is another part where you would need Git commands describing how to clone a branch, add new functionality to project and integrate them into team's develop branch.

Finally, put your code in the appendix in order that you can reconstruct your work through your assignment or reflect on what you did as well as how you evaluated it.

Description of the chosen topic and its relevance:

The topic I have chosen "Evaluating the Variation in Prices of T-Shirts using Least Square Method" is extremely important for several reasons.

Real-life application: **Pricing is a crucial element of any business and knowing what makes up the cost price is vital too. Such study can shed light on how various brands or shops come up with their prices.**

Statistical analysis: The least square method is an approach used worldwide to solve over-determined systems in regression analysis. It works by minimizing the sum of squares of residuals which are nothing but differences between observed values and predicted values. In this case, we are going to apply this statistical technique in our everyday life situation.

Predictive modelling: The main aim here is building a model which should be able to predict the price of a T-shirt according to the selected theoretical functions. This task falls under machine learning or data science; hence, what you learn can be applied when dealing with other predictive models.

Programming Skills: One needs solid programming skills for successful implementation of this task especially Python language. It includes such things as manipulating data, working with objects in an object-oriented manner, catching exceptions as well as using specific libraries like Pandas, Bokeh or sqlalchemy within Python programming which will not only benefit now but also it will go along way towards helping someone become great at data science related jobs later on.

Software Development Practices: The exercise requires sVCS (Git) where one gains experience on how to clone a branch, make changes then push them into that branch among other things commonly practiced during software development lifecycle.

Description of the datasets (training, test, and ideal functions):

Generalised Data:

File1

1	X	Y1	Y2	Y3	Y4
2	1	YYY1	YYY2	YYY3	YYY4
3	2	YYY2	YYY4	YYY6	YYY8
4	3	YYY3	YYY6	YYY9	YYY12

File2

1	X	Y1	Y2	Y3	Y4
2	1	XXX1	XXX2	XXX3	XXX4
3	2	XXX2	XXX4	XXX6	XXX8
4	3	XXX3	XXX6	XXX9	XXX12

File 3

1	X	Y1	Y2	Y3	Y4
2	1	ZZZ1	ZZZ2	ZZZ3	ZZZ4
3	2	ZZZ2	ZZZ4	ZZZ6	ZZZ8
4	3	ZZZ3	ZZZ6	ZZZ9	ZZZ12

Specific Data:

Ideal1

1	x	y1	y2	y3	y4	y5	y6	y7	y8	y9	y10	y11	y12	y13	y14	y15	y16
2	-20.0	-0.9129453	0.40808207	9.087055	5.408082	-9.087055	0.9129453	-0.8390715	-0.85091937	0.81616414	18.258905	-20.0	-58.0	-45.0	20.0	13.0	400.0
3	-19.9	-0.8676441	0.4971858	9.132356	5.4971857	-9.132356	0.8676441	-0.8652126	0.16851768	0.9943716	17.266117	-19.9	-57.7	-44.8	19.9	12.95	396.01
4	-19.8	-0.81367373	0.58132184	9.186326	5.5813217	-9.186326	0.81367373	-0.88919115	0.6123911	1.1626437	16.11074	-19.8	-57.4	-44.6	19.8	12.9	392.04
5	-19.7	-0.75157344	0.65964943	9.248426	5.6596494	-9.248426	0.75157344	-0.91094714	-0.99466854	1.3192989	14.805996	-19.7	-57.1	-44.4	19.7	12.85	388.09
6	-19.6	-0.6819636	0.7313861	9.318036	5.731386	-9.318036	0.6819636	-0.9304263	0.7743557	1.4627723	13.366487	-19.6	-56.8	-44.2	19.6	12.8	384.16
7	-19.5	-0.60553986	0.795815	9.39446	5.795815	-9.39446	0.60553986	-0.9475798	-0.11702018	1.59163	11.808027	-19.5	-56.5	-44.0	19.5	12.75	380.25
8	-19.4	-0.52306575	0.8522923	9.476934	5.8522925	-9.476934	0.52306575	-0.96236485	-0.59004813	1.7045846	10.147476	-19.4	-56.2	-43.8	19.4	12.7	376.36
9	-19.3	-0.43536535	0.90025383	9.564634	5.900254	-9.564634	0.43536535	-0.97474456	0.97776526	1.8005077	8.402552	-19.3	-55.9	-43.6	19.3	12.65	372.49
10	-19.2	-0.34331492	0.93922037	9.656685	5.9392204	-9.656685	0.34331492	-0.98468786	-0.87895167	1.8784407	6.5916467	-19.2	-55.6	-43.4	19.2	12.6	368.64
11	-19.1	-0.2478342	0.96880245	9.752166	5.9688025	-9.752166	0.2478342	-0.99217	0.37579286	1.9376049	4.7336335	-19.1	-55.3	-43.2	19.1	12.55	364.81
12	-19.0	-0.1498772	0.9887046	9.850122	5.9887047	-9.850122	0.1498772	-0.9971722	0.27938655	1.9774092	2.847667	-19.0	-55.0	-43.0	19.0	12.5	361.0
13	-18.9	-0.050422687	0.998728	9.949577	5.998728	-9.949577	0.050422687	-0.99968195	-0.80255306	1.997456	0.9529888	-18.9	-54.7	-42.8	18.9	12.45	357.21
14	-18.8	0.04953564	0.9987724	10.049536	5.998772	-10.049536	-0.04953564	-0.99969304	0.9999414	1.9975448	-0.93127006	-18.8	-54.4	-42.6	18.8	12.4	353.44
15	-18.7	0.14899902	0.98883736	10.148999	5.9888372	-10.148999	-0.14899902	-0.99720544	-0.82669914	1.9776747	-2.7862818	-18.7	-54.1	-42.4	18.7	12.35	349.69
16	-18.6	0.24697366	0.9690222	10.246974	5.9690223	-10.246974	-0.24697366	-0.99222535	0.3753813	1.9380444	-4.59371	-18.6	-53.8	-42.2	18.6	12.3	345.96
17	-18.5	0.34248063	0.9395249	10.342481	5.939525	-10.342481	-0.34248063	-0.9847652	0.1825695	1.8790498	-6.3358912	-18.5	-53.5	-42.0	18.5	12.25	342.25
18	-18.4	0.43456563	0.9006402	10.434566	5.90064	-10.434566	-0.43456563	-0.9748436	-0.6683636	1.8012804	-7.9960074	-18.4	-53.2	-41.8	18.4	12.2	338.56
19	-18.3	0.5223086	0.85275656	10.522308	5.8527565	-10.522308	-0.5223086	-0.9624855	0.95221686	1.7055131	-9.558248	-18.3	-52.9	-41.6	18.3	12.15	334.89

Train1

1	x	y1	y2	y3	y4
2	-20.0	-45.29234	-15999.796	99.52958	899.8275
3	-19.9	-44.36496	-15761.017	99.89567	893.4274
4	-19.8	-44.565968	-15524.681	98.85578	887.16046
5	-19.7	-44.76245	-15290.5	98.1261	881.4487
6	-19.6	-44.188698	-15058.586	97.511475	875.37726
7	-19.5	-44.325283	-14829.747	97.89807	869.4099
8	-19.4	-44.25751	-14602.485	96.85988	863.8766
9	-19.3	-43.125065	-14378.414	96.852066	857.5563
10	-19.2	-43.14248	-14155.929	95.52498	851.42163
11	-19.1	-43.300583	-13935.392	95.96843	845.65063
12	-19.0	-42.875423	-13718.042	95.377365	839.93085
13	-18.9	-42.94328	-13502.642	94.743195	834.3143
14	-18.8	-42.340214	-13289.002	94.043816	828.2413
15	-18.7	-42.112392	-13078.37	93.623535	823.1283
16	-18.6	-41.990173	-12869.967	92.95969	816.8228
17	-18.5	-41.772976	-12663.32	92.20281	811.35956
18	-18.4	-41.32246	-12459.056	92.36537	805.2177
19	-18.3	-41.247326	-12256.788	91.11789	799.8028

Least Square:

T-Shirt Price (x)	T-Shirt Sold (y)	x*y	x^2	c	Y = mx + c	error		numer	denom
2	4	8	4	0.963	3.34	-0.66		13.44	10.24
3	5	15	9	0.445	4.86	-0.14		7.04	4.84
5	7	35	25	-0.591	7.90	0.90		0.24	0.04
7	10	70	49	-0.628	10.93	0.93		3.24	3.24
9	15	135	81	1.335	13.97	-1.03		25.84	14.44

Sigma xy	263		Mean X	5.2
Sigma x	26		Mean Y	8.2
Sigma y	41		m	**1.518**
numer	249		c	**0.305**
denom	164			
m	**1.518**			
c	**0.305**			

Data Analysis

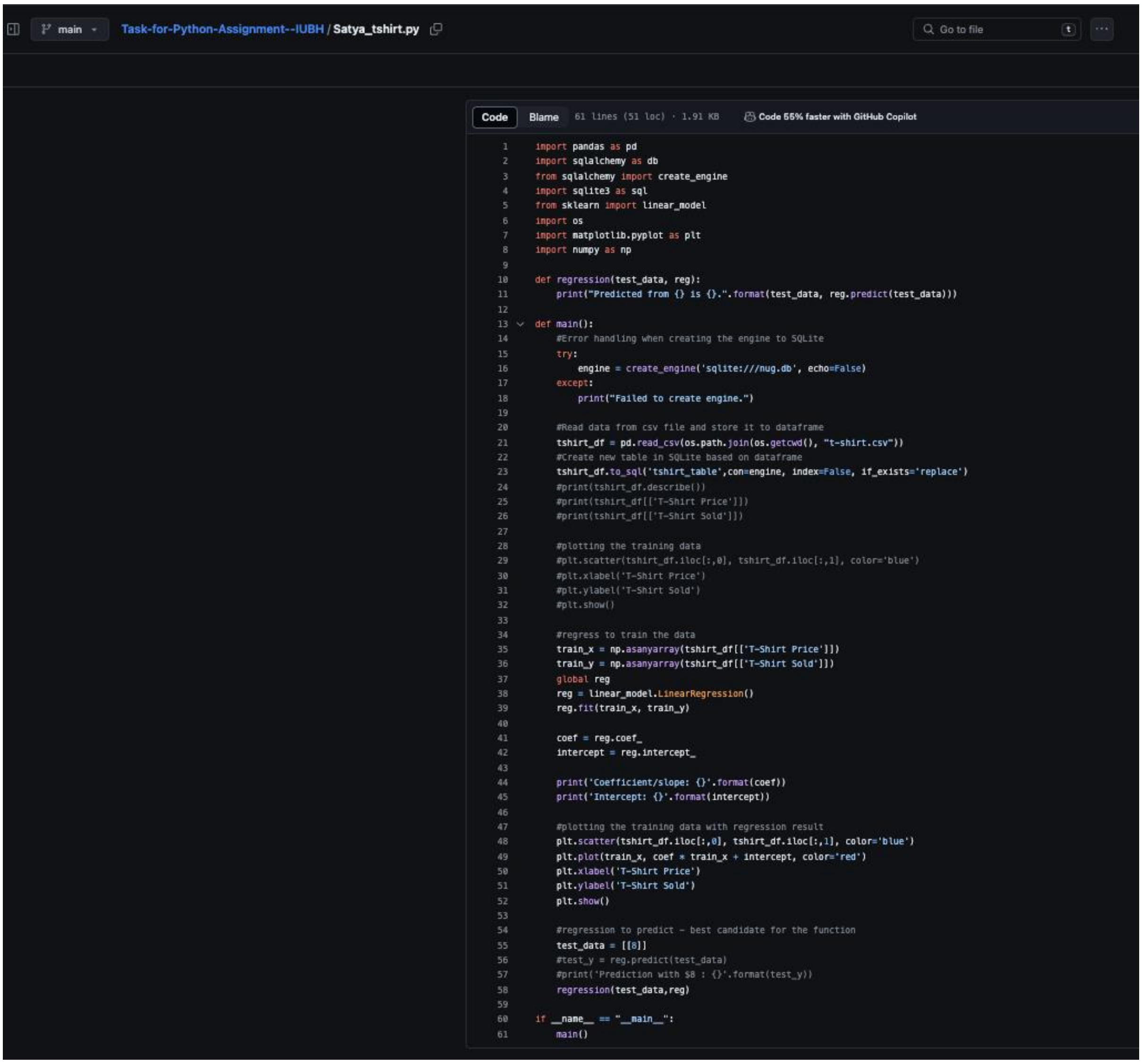
- Selection of the four ideal pricing functions that best fit the training data using the Least Square method.

2.
3.

- Mapping of the test data (T-shirt prices from the new brand/store) to the chosen ideal functions.

- Calculation of deviations: This would involve calculating the difference between the actual prices and the prices predicted by the ideal functions.

Satya_tshirt.py


main Task-for-Python-Assignment--IUBH / Satya_tshirt.py

Go to file

Code Blame 61 lines (51 loc) · 1.91 KB Code 55% faster with GitHub Copilot

```
import pandas as pd
import sqlalchemy as db
from sqlalchemy import create_engine
import sqlite3 as sql
from sklearn import linear_model
import os
import matplotlib.pyplot as plt
import numpy as np

def regression(test_data, reg):
    print("Predicted from {} is {}.".format(test_data, reg.predict(test_data)))

def main():
    #Error handling when creating the engine to SQLite
    try:
        engine = create_engine('sqlite:///nug.db', echo=False)
    except:
        print("Failed to create engine.")

    #Read data from csv file and store it to dataframe
    tshirt_df = pd.read_csv(os.path.join(os.getcwd(), "t-shirt.csv"))
    #Create new table in SQLite based on dataframe
    tshirt_df.to_sql('tshirt_table',con=engine, index=False, if_exists='replace')
    #print(tshirt_df.describe())
    #print(tshirt_df[['T-Shirt Price']])
    #print(tshirt_df[['T-Shirt Sold']])

    #plotting the training data
    #plt.scatter(tshirt_df.iloc[:,0], tshirt_df.iloc[:,1], color='blue')
    #plt.xlabel('T-Shirt Price')
    #plt.ylabel('T-Shirt Sold')
    #plt.show()

    #regress to train the data
    train_x = np.asanyarray(tshirt_df[['T-Shirt Price']])
    train_y = np.asanyarray(tshirt_df[['T-Shirt Sold']])
    global reg
    reg = linear_model.LinearRegression()
    reg.fit(train_x, train_y)

    coef = reg.coef_
    intercept = reg.intercept_

    print('Coefficient/slope: {}'.format(coef))
    print('Intercept: {}'.format(intercept))

    #plotting the training data with regression result
    plt.scatter(tshirt_df.iloc[:,0], tshirt_df.iloc[:,1], color='blue')
    plt.plot(train_x, coef * train_x + intercept, color='red')
    plt.xlabel('T-Shirt Price')
    plt.ylabel('T-Shirt Sold')
    plt.show()

    #regression to predict - best candidate for the function
    test_data = [[8]]
    #test_y = reg.predict(test_data)
    #print('Prediction with $8 : {}'.format(test_y))
    regression(test_data,reg)

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

Main_final.py

Code Blame 12 lines (9 loc) · 430 Bytes Code 55% faster with GitHub Copilot

```python
# Import packages
import pandas as pd
from sqlalchemy import create_engine

# Create an engine that connects to the database file
engine = create_engine("sqlite:///data.db")

# Read one of the CSV files into a pandas dataframe
df_train1 = pd.read_csv("train.csv")

# Write the dataframe to the train_table in the database, appending it if it already exists
df_train1.to_sql("train_table", engine, if_exists="append", index=False)
```

Main_final.ipynb

[ ]:

```python
# Import packages
import pandas as pd
import sqlalchemy as sa

# Create a SQLite file named data.db
engine = sa.create_engine("sqlite:///data.db")

# Read the four practice data points files
train1 = pd.read_csv("train1.csv")
# don't have the files
# train2 = pd.read_csv("train2.csv")
# train3 = pd.read_csv("train3.csv")
# train4 = pd.read_csv("train4.csv")

# Combine them into one dataframe with four columns for y
train = pd.DataFrame()
train["X"] = train1["x"]
train["Y1"] = train1["y1"]
# don't have the files
# train["Y2"] = train2["y"]
# train["Y3"] = train3["y"]
# train["Y4"] = train4["y"]

# Save the dataframe as a table named train_table in the SQLite file with if_exists='replace' to avoid table alre
train.to_sql("train_table", engine, index=False, if_exists='replace')

# Read the 50 curves files
ideal = pd.DataFrame()
ideal["X"] = pd.read_csv("ideal1.csv")["x"]
ideal["Y1"] = pd.read_csv("ideal1.csv")["y1"]

# Save the dataframe as a table named ideal_table in the SQLite file with if_exists='replace' to avoid table alre
ideal.to_sql("ideal_table", engine, index=False, if_exists='replace')
```

[ ]:

```
400
```

```python
# Import packages
import sqlalchemy as sa
import numpy as np

# Connect to the SQLite file
engine = sa.create_engine("sqlite:///data.db")

# Query the train_table and get the data points as a numpy array
train_data = np.array(engine.execute("SELECT * FROM train_table").fetchall())

# Query the ideal_table and get the curves as a numpy array
ideal_data = np.array(engine.execute("SELECT * FROM ideal_table").fetchall())

# Create an empty dictionary to store the results
results = {}

# Loop through each practice data point
for i in range(len(train_data)):
    # Get the x and y values of the data point
    x = train_data[i, 0]
    y = train_data[i, 1]
    # Create an empty list to store the least-square values for each curve
    lsq_list = []
    # Loop through each curve
    for j in range(len(ideal_data)):
        # Get the x and y values of the curve
        x_curve = ideal_data[j, 0]
        y_curve = ideal_data[j, 1]
        # Check if the x values match
        if x == x_curve:
            # Calculate the difference between the y values
            diff = y - y_curve
            # Square the difference and append it to the list
            lsq_list.append(diff ** 2)
    # Find the minimum least-square value and its index
    min_lsq = min(lsq_list)
    min_index = lsq_list.index(min_lsq)
    # Add the number of the curve and the least-square value to the results dictionary
    results[i] = (min_index + 1, min_lsq)
```

```python
# Create a dataframe from the test results dictionary
test_results_df = pd.DataFrame.from_dict(test_results, orient="index", columns=["N", "D"])

# Add the x and y columns from the test dataframe
test_results_df["X"] = test["x"]
test_results_df["Y"] = test["y"]

# Reorder the columns
test_results_df = test_results_df[["X", "Y", "N", "D"]]

# Save the dataframe as a table named test_table in the SQLite file with if_exists='replace' to avoid table alrea
test_results_df.to_sql("test_table", engine, index=False, if_exists='replace')
```

```
100
```

```
[ ]:
# Import packages
import pandas as pd
from sqlalchemy import create_engine

# Create an engine that connects to the database file
engine = create_engine("sqlite:///data.db")

# Read one of the CSV files into a pandas dataframe
df_train1 = pd.read_csv("train1.csv")

df_train1.to_sql("train_table", engine, if_exists="replace", index=False)

df_train = pd.read_sql("train_table", engine)

# Read the ideal1.csv file into a pandas dataframe
df_ideal = pd.read_csv("ideal1.csv")

# Write the dataframe to the ideal_table in the database, creating it if it does not exist
df_ideal.to_sql("ideal_table", engine, if_exists="replace", index=False)

# Read the ideal_table into a dataframe
df_ideal = pd.read_sql("ideal_table", engine)

# Create an empty dictionary to store the number and the least-square value of each curve for one practice data p
results = {}

# Loop through the 50 curves
for i in range(1, 51):
    # Get the column name of the curve
    curve_name = "y" + str(i)  # Fixed the column name to match DataFrame's column naming convention
    # Calculate the least-square value by subtracting the y values of the curve and the practice data point, squa
    least_square = ((df_ideal[curve_name] - df_train["y1"]) ** 2).sum()
    # Store the number and the least-square value of the curve in the dictionary
    results[i] = least_square

# Print the dictionary
print(results)
```

```
{1: 223851.06835863274, 2: 223507.98002626628, 3: 304458.54552846024, 4: 254003.28529166983, 5: 222065.5074828843
6, 6: 222709.5024375785, 7: 223012.46928551153, 8: 223359.3286236779, 9: 224339.3877964871, 10: 248450.7025235493
8, 11: 63181.47375804107, 12: 72865.57150744987, 13: 34.08070758146571, 14: 489654.52331702027, 15: 358883.4818898
575, 16: 13583588.983202264, 17: 12463645.6778728, 18: 52545160.63186699, 19: 14730894.719744205, 20: 15155008.997
413073, 21: 3606477631.9930983, 22: 3666243606.461506, 23: 3708894244.956376, 24: 14527657888.94006, 25: 327635543
23.02047, 26: 4348485044.120966, 27: 4535387013.678952, 28: 3631918796.1323185, 29: 3613438136.977763, 30: 3668243
454.88404, 31: 317669.4256190411, 32: 239280.50234231673, 33: 321377.3549476932, 34: 226039.77971102635, 35: 23704
3.68907710104, 36: 1762691.1339450825, 37: 235166.5714560203, 38: 223835.25822639727, 39: 13583625.646014329, 40:
54635809.61573768, 41: 9991.766850901204, 42: 863070.4340459134, 43: 233782.80553987756, 44: 223064.69092098653, 4
5: 329879.4448761261, 46: 248820.29350839625, 47: 214333.80054695703, 48: 222960.25616839758, 49: 223477.972220668
37, 50: 223440.22389161022}
```

Query of files through SQL :

In [ ]:

```
import pprint
pprint.pprint(results)
```

```
{0: (1, 1969.5306739383886),
 1: (1, 1892.0164905043932),
 2: (1, 1914.2632538886749),
 3: (1, 1936.9572555795573),
 4: (1, 1892.8359381521434),
 5: (1, 1911.415940227577),
 6: (1, 1912.7016138563583),
 7: (1, 1822.4104562072102),
 8: (1, 1831.7685315450915),
 9: (1, 1853.5391792359017),
 10: (1, 1825.4722639078975),
 11: (1, 1839.7972084733776),
 12: (1, 1796.8908745418805),
 13: (1, 1786.0251709453364),
 14: (1, 1783.976557978349),
 15: (1, 1773.7116871534108),
 16: (1, 1743.649189464477),
 17: (1, 1744.7023746175175),
 18: (1, 1735.510840784584),
 19: (1, 1770.339600725514),
 20: (1, 1771.3944602764202),
 21: (1, 1724.8264644895883),
 22: (1, 1708.341842322733),
 23: (1, 1688.1536042254904),
 24: (1, 1710.521179610256),
 25: (1, 1642.0786794049002),
 26: (1, 1630.2778544379603),
 27: (1, 1687.4950415079616),
 28: (1, 1592.1508580955779),
 29: (1, 1580.0563785059292),
 30: (1, 1575.364166703855),
 31: (1, 1573.7028701657762),
 32: (1, 1556.5227208459044),
 33: (1, 1530.0313172704175),
 34: (1, 1503.9901681976064),
 35: (1, 1460.4775832077423),
 36: (1, 1502.8885618891848),
 37: (1, 1472.8912428317926),
 38: (1, 1399.8651615670974),
 39: (1, 1384.5082968255458),
 40: (1, 1381.6793975212734),
 41: (1, 1346.199291080638),
 42: (1, 1313.975998048607),
 43: (1, 1343.0292747355606),
 44: (1, 1273.0131870516173),
 45: (1, 1256.080253712155),
 46: (1, 1238.9002353016149),
 47: (1, 1205.0138495001283),
```

```
df_2 = _deepnote_execute_sql('SELECT *\nFROM \'ideal1.csv\'', 'SQL_DEEPNOTE_DATAFRAME_SQL', audit_sql_comment='',
df_2
```

	x	y1	y2	y3	y4	y5	y6	y7	y8	y9	...	y41
0	-20.0	-0.912945	0.408082	9.087055	5.408082	-9.087055	0.912945	-0.839071	-0.850919	0.816164	...	-40.456474
1	-19.9	-0.867644	0.497186	9.132356	5.497186	-9.132356	0.867644	-0.865213	0.168518	0.994372	...	-40.233820
2	-19.8	-0.813674	0.581322	9.186326	5.581322	-9.186326	0.813674	-0.889191	0.612391	1.162644	...	-40.006836
3	-19.7	-0.751573	0.659649	9.248426	5.659649	-9.248426	0.751573	-0.910947	-0.994669	1.319299	...	-39.775787
4	-19.6	-0.681964	0.731386	9.318036	5.731386	-9.318036	0.681964	-0.930426	0.774356	1.462772	...	-39.540980
...	...	...	...	...	...	...	...	...	...	...	...	...
395	19.5	0.605540	0.795815	10.605540	5.795815	-10.605540	-0.605540	-0.947580	-0.117020	1.591630	...	39.302770
396	19.6	0.681964	0.731386	10.681964	5.731386	-10.681964	-0.681964	-0.930426	0.774356	1.462772	...	39.540980
397	19.7	0.751573	0.659649	10.751574	5.659649	-10.751574	-0.751573	-0.910947	-0.994669	1.319299	...	39.775787
398	19.8	0.813674	0.581322	10.813674	5.581322	-10.813674	-0.813674	-0.889191	0.612391	1.162644	...	40.006836
399	19.9	0.867644	0.497186	10.867644	5.497186	-10.867644	-0.867644	-0.865213	0.168518	0.994372	...	40.233820

400 rows × 51 columns

Data Visualisations:

Line Plot example:

Data Visualization

1. Matplotlib
2. Seaborn
3. Bokeh

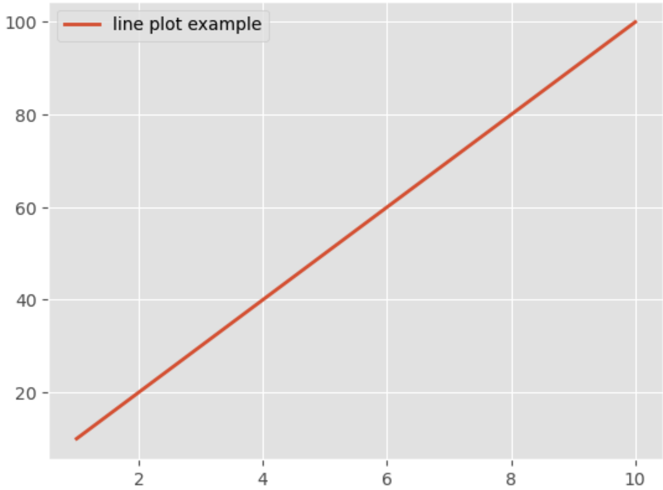
In [4]:
```
import numpy as np
from matplotlib import pyplot as plt
from matplotlib import style
```

In [6]:
```
# x and y data list
x = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
y = [10, 20, 30, 40, 50, 60, 70, 80, 90, 100]
```

In [8]:
```
# set plot style
style.use('ggplot')
```

In [10]:
```
# line plot example
plt.plot(x, y, label="line plot example", linewidth=2)
plt.legend()
```

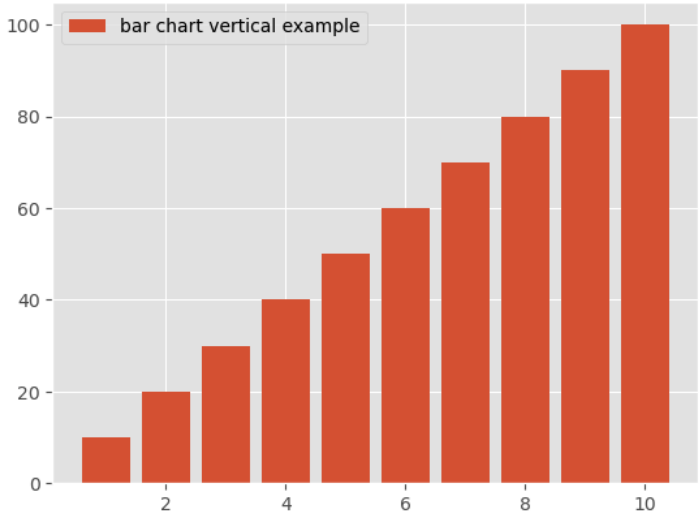
Out[10]:
```
<matplotlib.legend.Legend at 0x7f946eab4810>
```



Bar Chart:

```
# bar chart vertical example
plt.bar(x, y, label="bar chart vertical example", align='center')
plt.legend()
```

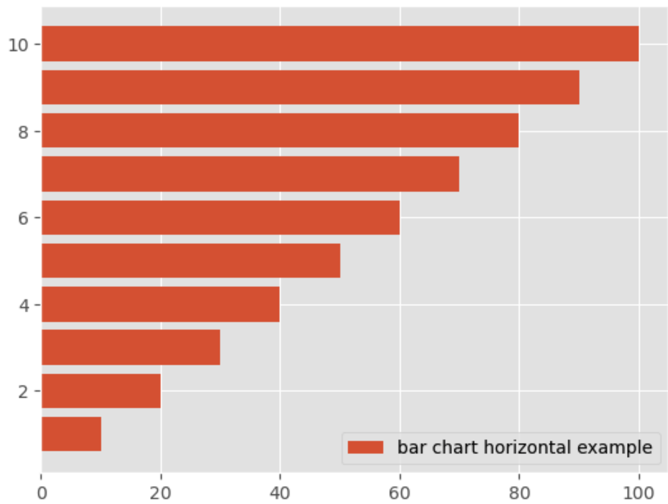
<matplotlib.legend.Legend at 0x7f946e6352d0>



Horizontal Bar chart:

In [24]:
```
# bar chart horizontal example
plt.barh(x, y, label="bar chart horizontal example", align='center')
plt.legend()
```

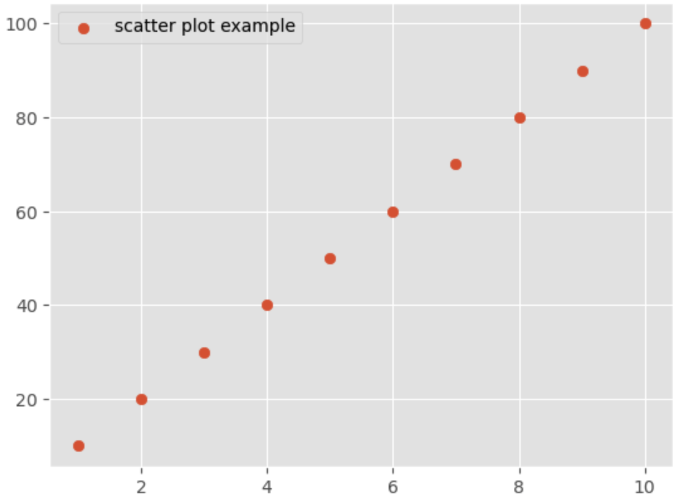
Out[24]: <matplotlib.legend.Legend at 0x7f946e236a10>



Scatter plot:

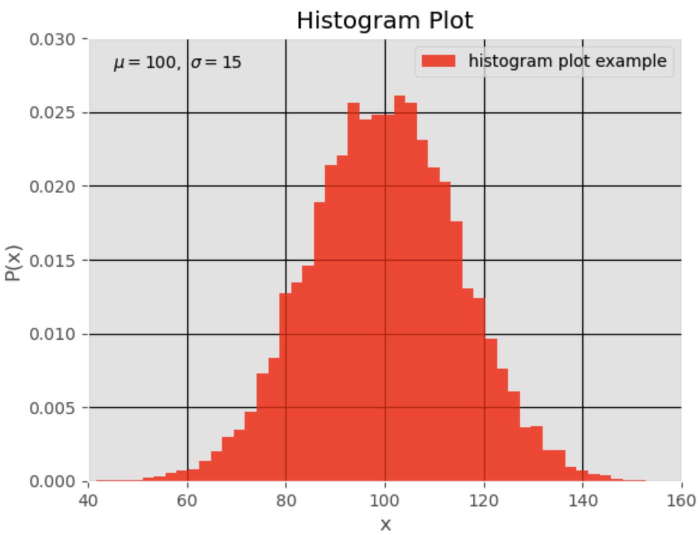
In [28]:
```
# scatter plot example
plt.scatter(x, y, label="scatter plot example")
plt.legend()
```

Out[28]: <matplotlib.legend.Legend at 0x7f946e2d46d0>



Histogram:

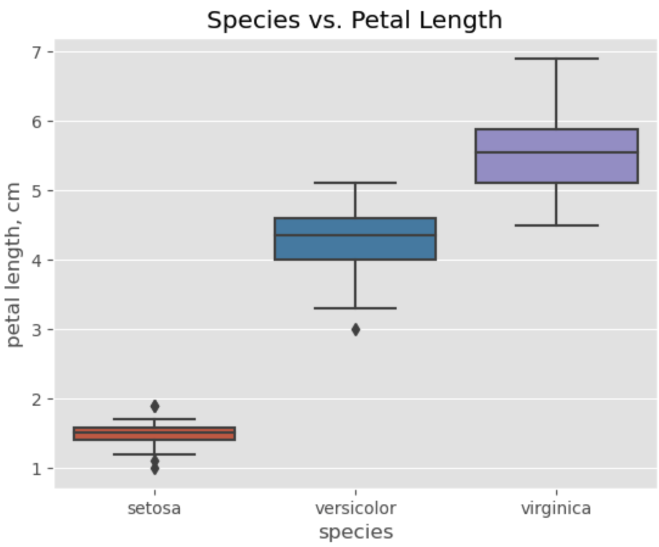
In [32]:
```
# histogram plot example
mu, sigma = 100, 15
x = mu + sigma * np.random.randn(10000)
n, bins, patches = plt.hist(x, 50, density=1, facecolor='r', alpha=0.75,
label="histogram plot example")
plt.text(45, 0.028, r'$\mu=100,\ \sigma=15$')
plt.axis([40, 160, 0, 0.03])
plt.legend()
plt.grid(True,color="k")
plt.xlabel("x")
plt.ylabel("P(x)")
plt.title("Histogram Plot")
plt.show()
```



Box Plot:

In [10]:

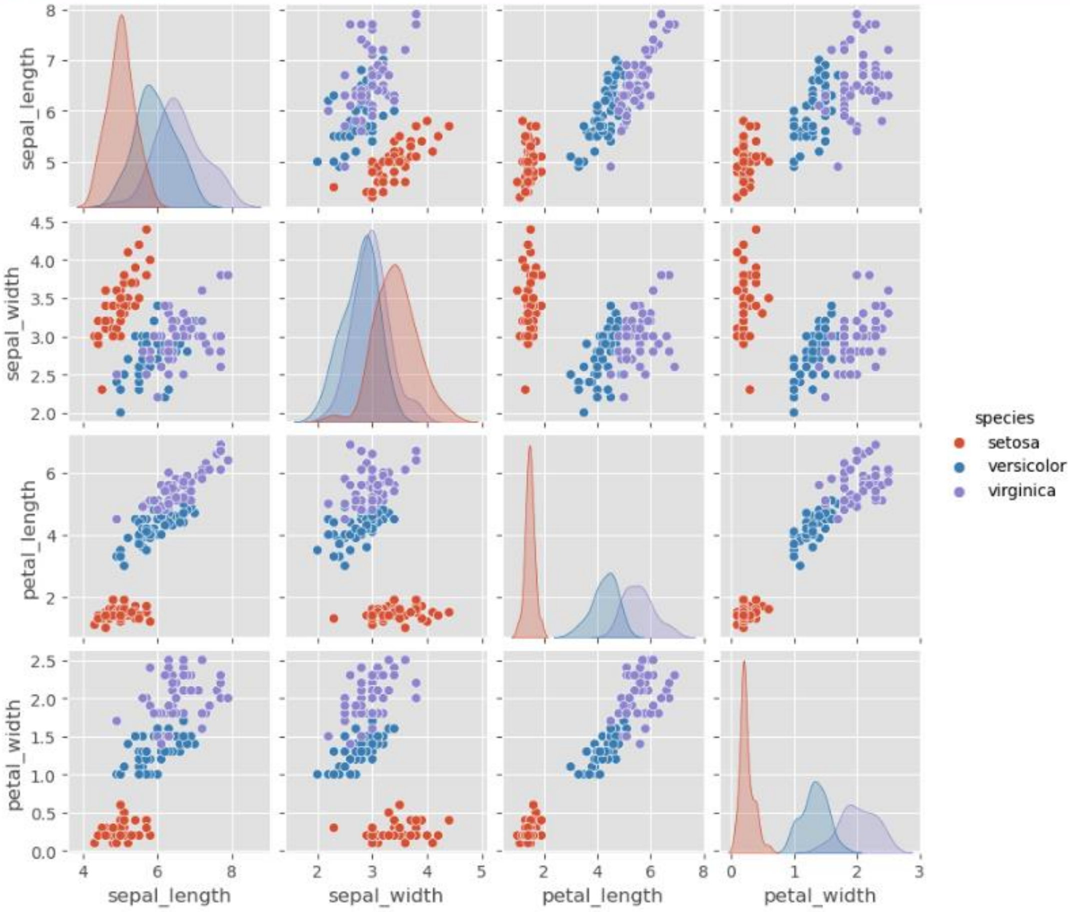
```
# box plot example
sns.boxplot(x="species", y="petal_length", data=iris_data)
plt.ylabel("petal length, cm")
plt.xlabel("species")
plt.title("Species vs. Petal Length")
plt.show()
```



Pair plot:

In [12]:
```
# pair plot example
sns.pairplot(iris_data, hue="species", size=2)
plt.ylabel("petal length, cm")
plt.xlabel("species")
plt.show()
```

```
/opt/conda/envs/anaconda-panel-2023.05-py310/lib/python3.11/site-packages/seaborn/axisgrid.py:2095: UserWarning:
The `size` parameter has been renamed to `height`; please update your code.
  warnings.warn(msg, UserWarning)
/opt/conda/envs/anaconda-panel-2023.05-py310/lib/python3.11/site-packages/seaborn/axisgrid.py:118: UserWarning:
The figure layout has changed to tight
  self._figure.tight_layout(*args, **kwargs)
```



Unit Test:

In [3]:
```
import unittest
def main ():

class TestMyFunction(unittest.TestCase):
    def test_add(self):
        self.assertEqual(add(1, 2), 3)

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

Git Clone Steps:

These are the instructions for cloning a branch, adding a new function and pushing changes to the branch in Git:

Clone the repository: git clone

Checkout ‘develop’ branch: git checkout develop

Amend code (add a new function)

Stage changes: git add .

Commit changes: git commit -m "Added a new function"

Push to ‘develop’ branch: git push origin develop

References:

1. The Course Assignment book
2. Udemy course on python
3. Github:- https://github.com/satyads7836/Task-for-Python-Assignment--IUBH
4. Extra Course book excercises:-

https://github.com/satyads7836/DLMDSPWP01_Python_Coursebook_Codes