In [ ]:
```
!jupyter nbconvert --to html /content/TEAM6_RDD.ipynb
```

In [ ]:
```
from google.colab import drive
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.m
ount("/content/drive", force_remount=True).
```

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

In [ ]:
```
# Câu 1
import numpy as np

# Tạo hai ma trận
matrix_a = np.array([[1, 2, 3],
                     [4, 5, 6],
                     [7, 8, 9]])

matrix_b = np.array([[9, 8, 7],
                     [6, 5, 4],
                     [3, 2, 1]])

# Phép cộng hai ma trận
result_addition = matrix_a + matrix_b

# Phép trừ hai ma trận
result_subtraction = matrix_a - matrix_b

# In kết quả
print("Ma trận A:")
print(matrix_a)

print("\nMa trận B:")
print(matrix_b)

print("\nPhép cộng hai ma trận:")
print(result_addition)

print("\nPhép trừ hai ma trận:")
print(result_subtraction)
```

```
Ma trận A:
[[1 2 3]
 [4 5 6]
 [7 8 9]]

Ma trận B:
[[9 8 7]
 [6 5 4]
 [3 2 1]]

Phép cộng hai ma trận:
[[10 10 10]
 [10 10 10]
 [10 10 10]]

Phép trừ hai ma trận:
[[-8 -6 -4]
 [-2  0  2]
 [ 4  6  8]]
```

In [ ]:
```
data_link="/content/drive/MyDrive/HMTK_O13_Thực hành/ie105/dataset.csv"
```

In [ ]:
```
# Câu 2.1
df = pd.read_csv(data_link)
df
```

Out[ ]:

	Unnamed: 0	X	Y	Z
0	1	78	84	86
1	2	85	94	97
2	3	80	83	73
3	4	96	94	96
4	5	86	86	83

In [ ]:
```
# Câu 2.2
df = df.rename(columns={"Unnamed: 0" : "id"})
df
```

Out[ ]:

	id	X	Y	Z
0	1	78	84	86
1	2	85	94	97
2	3	80	83	73
3	4	96	94	96
4	5	86	86	83

In [ ]:
```
# Câu 2.3
df_Sort = df.sort_values(by=['X', 'Y', 'Z'], ascending=False)
df_Sort
```

Out[ ]:

	id	X	Y	Z
3	4	96	94	96
4	5	86	86	83
1	2	85	94	97
2	3	80	83	73
0	1	78	84	86

In [ ]:
```
# Câu 2.4
print(df_Sort.loc[:, 'X'])
```

```
3    96
4    86
1    85
2    80
0    78
Name: X, dtype: int64
```

In [ ]:
```
# Câu 2.5
print(df.iloc[[0, 1]])
```

```
   id   X   Y   Z
0   1  78  84  86
1   2  85  94  97
```

In [ ]:
```
# Câu 2.6
row = df.loc[df['X'] == 85]
print(row)
```

```
   id   X   Y   Z
1   2  85  94  97
```

In [ ]:
```
# Câu 2.7
import pandas as pd
import numpy as np

# Thêm giá trị NaN vào dataset
data = {'id': [1, 2, 3, 4, 5],
        'X': [78, 85, 80, np.nan, 96],
        'Y': [84, 94, 83, 94, 86],
        'Z': [86, np.nan, 73, 96, 97]}

df = pd.DataFrame(data)
df
# Thay thế giá trị NaN thành 0
df.fillna(0, inplace=True)

print(df)
```

```
   id     X   Y     Z
0   1  78.0  84  86.0
1   2  85.0  94   0.0
2   3  80.0  83  73.0
3   4   0.0  94  96.0
4   5  96.0  86  97.0
```

In [ ]:
```
df
```

Out[ ]:

	id	X	Y	Z
0	1	78.0	84	86.0
1	2	85.0	94	0.0
2	3	80.0	83	73.0
3	4	0.0	94	96.0
4	5	96.0	86	97.0

In [ ]:
```
# Câu 2.8

dm = df['Z']

for i in range(len(dm)):
  if (dm.iloc[i]> 90): dm.iloc[i] = True
  else: dm.iloc[i] = False

df
```

```
<ipython-input-13-a0213edd7a4a>:7: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/us
er_guide/indexing.html#returning-a-view-versus-a-copy
  else: dm.iloc[i] = False
```

Out[ ]:

	id	X	Y	Z
0	1	78.0	84	False
1	2	85.0	94	False
2	3	80.0	83	False
3	4	0.0	94	True
4	5	96.0	86	True

In [ ]:
```
# Câu 2.9
import pandas as pd

data = {'X': [78, 85, 80, 96, 86, 86],
        'Y': [84, 94, 83, 94, 86, 86],
        'Z': [86, 97, 73, 96, 83, 83]}



# create 2 dataframes
d1 = df[['X', 'Y']]
d2 = df[['Z']]

# concatenate the 2 dataframes
d3 = pd.concat([d1, d2], axis=1)
```

```
print(d3)
```

```
      X   Y      Z
0  78.0  84  False
1  85.0  94  False
2  80.0  83  False
3   0.0  94   True
4  96.0  86   True
```

In [ ]:
```
import pandas as pd

data = {'X': [78, 85, 80, 96, 86, 86],
        'Y': [84, 94, 83, 94, 86, 86],
        'Z': [86, 97, 73, 96, 83, 83]}
data
```

Out[ ]:
```
{'X': [78, 85, 80, 96, 86, 86],
 'Y': [84, 94, 83, 94, 86, 86],
 'Z': [86, 97, 73, 96, 83, 83]}
```

In [ ]:
```
# Câu 2.10
data = {'X': [78, 85, 80, 96, 86, 86],
        'Y': [84, 94, 83, 94, 86, 86],
        'Z': [86, 97, 73, 96, 83, 83]}
df = pd.DataFrame(data)
df
```

Out[ ]:

	X	Y	Z
0	78	84	86
1	85	94	97
2	80	83	73
3	96	94	96
4	86	86	83
5	86	86	83

In [ ]:
```
# create d1 dataframes
d1 = df[['X', 'Y']]
d1
```

Out[ ]:

	X	Y
0	78	84
1	85	94
2	80	83
3	96	94
4	86	86
5	86	86

In [ ]:
```
# create d2 dataframe
d2 = df[['Z']]
d2
```

Out[ ]:

	Z
0	86
1	97
2	73
3	96
4	83
5	83

In [ ]:
```
# concatenate the 2 dataframes
d3 = pd.concat([d1, d2], axis=1)
print(d3)
```

```
    X   Y   Z
0  78  84  86
1  85  94  97
2  80  83  73
3  96  94  96
4  86  86  83
5  86  86  83
```

In [ ]:
```
# Câu 2.11
import pandas as pd
import statistics

# Dataframe
data = {'X': [78, 85, 80, 96, 86, 86], 'Y': [84, 94, 83, 94, 86, 86], 'Z': [86, 97, 73
dataset = pd.DataFrame(data)

# Results
print("Data Count: ", dataset.count())
print("Data Mean: ", dataset.mean())
print("Data Median: ", dataset.median())
print("Data Mode: ", dataset.mode())
print("Data Variance: ", dataset.var())
print("Data Standard Deviation: ", dataset.std())
```

```
Data Count:  X    6
Y    6
Z    6
dtype: int64
Data Mean:  X    85.166667
Y    87.833333
Z    86.333333
dtype: float64
Data Median:  X    85.5
Y    86.0
Z    84.5
dtype: float64
Data Mode:        X   Y     Z
0  86.0  86  83.0
1   NaN  94   NaN
Data Variance:  X    39.366667
Y    24.166667
Z    81.466667
dtype: float64
Data Standard Deviation:  X    6.274286
Y    4.915960
Z    9.025889
dtype: float64
```

Câu 3:

In [ ]:
```
# Câu 3
import numpy as np
from sklearn.linear_model import LinearRegression
# X is a matrix that represents the training dataset
# y is a vector of weights, to be associated with input dataset
X = np.array([[3], [5], [7], [9], [11]]).reshape(-1, 1)
y = [8.0, 9.1, 10.3, 11.4, 12.6]
lreg_model = LinearRegression()
lreg_model.fit(X, y)
# New data (unseen before)
new_data = np.array([[13]])
print('Model Prediction for new data: $%.2f' %
lreg_model.predict(new_data)[0] )
```

```
Model Prediction for new data: $13.73
```

Sử dụng TensorFlow

In [ ]:
```
import numpy as np
import tensorflow as tf

# X is a matrix that represents the training dataset
# y is a vector of weights, to be associated with input dataset
X = np.array([[3], [5], [7], [9], [11]]).reshape(-1, 1)
y = np.array([8.0, 9.1, 10.3, 11.4, 12.6])  # Convert y to a NumPy array

# Build the model
model_tf = tf.keras.Sequential([
    tf.keras.layers.Input(shape=(1,)),
    tf.keras.layers.Dense(units=1)
```

```
])

# Compile the model
model_tf.compile(optimizer='sgd', loss='mean_squared_error')

# Train the model
model_tf.fit(X, y, epochs=1000, verbose=0)

# New data (unseen before)
new_data = np.array([[13]])

# Make predictions
prediction_tf = model_tf.predict(new_data)
print('TensorFlow Model Prediction for new data: $%.2f' % prediction_tf[0, 0])
```

```
1/1 [==============================] - 0s 92ms/step
TensorFlow Model Prediction for new data: $13.96
```

Sử dụng Keras:

In [ ]:

```
import numpy as np
from keras.models import Sequential
from keras.layers import Dense

# X is a matrix that represents the training dataset
# y is a vector of weights, to be associated with input dataset
X = np.array([[3], [5], [7], [9], [11]])
y = np.array([8.0, 9.1, 10.3, 11.4, 12.6])  # Convert y to a NumPy array

# Build the model
model_keras = Sequential()
model_keras.add(Dense(units=1, input_dim=1))

# Compile the model
model_keras.compile(optimizer='sgd', loss='mean_squared_error')

# Train the model
model_keras.fit(X, y, epochs=1000, verbose=0)

# New data (unseen before)
new_data = np.array([[13]])

# Make predictions
prediction_keras = model_keras.predict(new_data)
print('Keras Model Prediction for new data: $%.2f' % prediction_keras[0, 0])
```

```
1/1 [==============================] - 0s 41ms/step
Keras Model Prediction for new data: $13.96
```

Sử dụng PyTorch:

In [ ]:

```
import torch
import torch.nn as nn
import torch.optim as optim

# Convert data to PyTorch tensors
```

```
X_tensor = torch.Tensor(X)
y_tensor = torch.Tensor(y).view(-1, 1)
new_data_tensor = torch.Tensor(new_data)

# Build the model
model_pytorch = nn.Linear(1, 1)

# Define the loss function and optimizer
criterion = nn.MSELoss()
optimizer = optim.SGD(model_pytorch.parameters(), lr=0.01)

# Train the model
for epoch in range(1000):
    # Forward pass
    outputs = model_pytorch(X_tensor)
    loss = criterion(outputs, y_tensor)

    # Backward pass and optimization
    optimizer.zero_grad()
    loss.backward()
    optimizer.step()

# Make predictions
prediction_pytorch = model_pytorch(new_data_tensor)
print('PyTorch Model Prediction for new data: $%.2f' % prediction_pytorch.item())
```

```
PyTorch Model Prediction for new data: $13.98
```

Thực Hiện Linear Regression với 4 Thư Viện: Scikit-learn, TensorFlow, Keras, và PyTorch

1. Scikit-learn: Scikit-learn cung cấp một giao diện đơn giản và hiệu quả để thực hiện Linear Regression và nhiều mô hình học máy khác. Dành cho các tác vụ đơn giản, Scikit-learn là sự lựa chọn phổ biến do dễ sử dụng và tài liệu hướng dẫn phong phú.

2. TensorFlow: TensorFlow là một thư viện mạnh mẽ chủ yếu được sử dụng cho deep learning, nhưng cũng cung cấp các công cụ linh hoạt cho học máy cơ bản. Việc sử dụng TensorFlow cho Linear Regression có thể cảm thấy mạnh mẽ hơn nhưng có thể phức tạp hơn so với Scikit-learn cho các tác vụ đơn giản.

3. Keras: Keras là một giao diện cao cấp giúp xây dựng mô hình trên nền tảng TensorFlow (hoặc Theano). Đối với Linear Regression, việc sử dụng Keras có thể cảm thấy quá nặng nề, nhưng nó trở nên hữu ích khi bạn muốn mở rộng sang các mô hình phức tạp hơn.

4. PyTorch: PyTorch được biết đến với cú pháp linh hoạt và dễ hiểu. Việc sử dụng PyTorch cho Linear Regression có thể là một sự lựa chọn tốt, đặc biệt là nếu bạn quan tâm đến tính toán gradient và muốn sử dụng các tính năng của PyTorch trong tương lai.

Cảm Nhận Chung:

Scikit-learn: Lựa chọn tốt cho các tác vụ đơn giản, dễ sử dụng và nhanh chóng.

TensorFlow: Phù hợp cho các dự án lớn, độ linh hoạt và hiệu suất cao, nhưng có thể quá mạnh mẽ cho các nhiệm vụ nhỏ.

Keras: Thuận lợi cho việc xây dựng các mô hình nhanh chóng trên nền tảng TensorFlow, đặc biệt là khi mở rộng sang deep learning.

PyTorch: Giao diện linh hoạt, thích hợp cho người mới bắt đầu và có thể mở rộng được cho các mô hình phức tạp hơn.

Tổng Kết:

Lựa chọn giữa các thư viện phụ thuộc vào mức độ phức tạp của công việc, sự quen thuộc cá nhân, và mục tiêu của dự án. Scikit-learn là lựa chọn an toàn cho các tác vụ đơn giản, trong khi TensorFlow, Keras và PyTorch cung cấp linh hoạt và hiệu suất cho các dự án phức tạp hơn.

Câu 4:

In [ ]:
```
# Câu 4: Sinh viên hoàn thành code phát hiện spam với SVMs và Linear regression
import pandas as pd
import numpy as np
df = pd.read_csv("/content/drive/MyDrive/HMTK_013_Thực hành/ie105/Datasets/Datasets/sm
y = df.iloc[:, 0].values
y = np.where(y == 'spam', -1, 1)
X = df.iloc[:, [1, 2]].values

from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(
 X, y, test_size=0.3, random_state=0)
from sklearn.linear_model import Perceptron
p = Perceptron(max_iter=40, eta0=0.1, random_state=0)
p.fit(X_train, y_train)
y_pred = p.predict(X_test)
from sklearn.metrics import accuracy_score
print('Misclassified samples: %d' % (y_test != y_pred).sum())
print('Accuracy: %.2f' % accuracy_score(y_test, y_pred))
```

```
Misclassified samples: 7
Accuracy: 0.77
```

SVMs

In [ ]:
```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score

# Đọc dữ liệu từ file CSV
```

```
df = pd.read_csv("/content/drive/MyDrive/HMTK_013_Thực hành/ie105/Datasets/Datasets/sm
y = df.iloc[:, 0].values
y = np.where(y == 'spam', -1, 1)
X = df.iloc[:, [1, 2]].values

# Chia dữ liệu thành tập huấn luyện và tập kiểm thử
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=

# Sử dụng SVM với kernel tuyến tính
svm = SVC(kernel='linear', C=1.0, random_state=0)
svm.fit(X_train, y_train)

# Dự đoán và đánh giá mô hình
y_pred = svm.predict(X_test)
print('Misclassified samples: %d' % (y_test != y_pred).sum())
print('Accuracy: %.2f' % accuracy_score(y_test, y_pred))
```

```
Misclassified samples: 7
Accuracy: 0.84
```

Câu 5:

Phương thức genfromtxt() trong thư viện NumPy là một công cụ mạnh mẽ để đọc và chuyển đổi dữ liệu từ các tệp văn bản thành các mảng NumPy. Dưới đây là một số chức năng chính và tính năng chuyên sâu của phương thức này, điều này sẽ giúp bạn nắm bắt và xử lý dữ liệu hiệu quả hơn:

Đọc Dữ Liệu Từ Tệp Văn Bản: genfromtxt() hỗ trợ nhiều định dạng tệp văn bản như CSV, TSV, và các định dạng tùy chỉnh khác, giúp bạn đọc và chuyển đổi dữ liệu một cách linh hoạt.

Xử Lý Dữ Liệu Thiếu: Nó cung cấp cơ chế mạnh mẽ để xử lý giá trị thiếu, bằng cách cho phép bạn đặt giá trị mặc định hoặc xác định cách xử lý dữ liệu thiếu.

Kiểm Soát Cột Đọc: Bạn có khả năng chỉ định chính xác các cột bạn quan tâm từ tệp văn bản, điều này làm giảm bộ nhớ cần thiết và tăng hiệu suất khi bạn chỉ cần một số cột cụ thể.

Kiểm Soát Kiểu Dữ Liệu: Phương thức này tự động suy luận kiểu dữ liệu của mỗi cột, nhưng bạn cũng có thể xác định kiểu dữ liệu mong muốn của từng cột hoặc toàn bộ mảng.

Chỉ Định Dòng Bắt Đầu và Kết Thúc: Bạn có thể chọn đọc chỉ một phần của tệp bằng cách chỉ định dòng bắt đầu và kết thúc, giảm thiểu thời gian đọc dữ liệu khi chỉ quan tâm đến một phần nhỏ.

Chấp Nhận Đối Số Delimiter Tùy Chọn: Nếu dữ liệu của bạn không phải là CSV và sử dụng một loại delimiter khác, bạn có thể chỉ định delimiter đó để phù hợp với định dạng của tệp văn bản.

Phương thức genfromtxt() mang lại sự linh hoạt và khả năng tùy chỉnh cao, làm cho quá trình xử lý dữ liệu từ các nguồn văn bản trở nên thuận tiện và hiệu quả.

Câu 6:

In [ ]:
```
import pandas as pd
import numpy as np
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score
from sklearn.model_selection import train_test_split

# Đọc dữ liệu từ file CSV
phishing_dataset = np.genfromtxt("/content/drive/MyDrive/HMTK_O13_Thực hành/ie105/Data
samples = phishing_dataset[:, :-1]
targets = phishing_dataset[:, -1]

# Chia dữ liệu thành tập huấn luyện và tập kiểm thử
training_samples, testing_samples, training_targets, testing_targets = train_test_spli
    samples, targets, test_size=0.2, random_state=0)

# Sử dụng Logistic Regression
log_classifier = LogisticRegression()
log_classifier.fit(training_samples, training_targets)

# Dự đoán và đánh giá mô hình
predictions_log = log_classifier.predict(testing_samples)

accuracy_log = accuracy_score(testing_targets, predictions_log)
precision_log = precision_score(testing_targets, predictions_log)
recall_log = recall_score(testing_targets, predictions_log)
f1_log = f1_score(testing_targets, predictions_log)

print("Logistic Regression Metrics:")
print("Accuracy: {:.2f}".format(accuracy_log * 100))
print("Precision: {:.2f}".format(precision_log))
print("Recall: {:.2f}".format(recall_log))
print("F1 Score: {:.2f}".format(f1_log))
```

```
Logistic Regression Metrics:
Accuracy: 91.68
Precision: 0.91
Recall: 0.94
F1 Score: 0.92
```

In [ ]:
```
import pandas as pd
import numpy as np
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score
from sklearn.model_selection import train_test_split
from sklearn.metrics import precision_score, recall_score, f1_score


# Đọc dữ liệu từ file CSV
phishing_dataset = np.genfromtxt("/content/drive/MyDrive/HMTK_O13_Thực hành/ie105/Data
samples = phishing_dataset[:, :-1]
targets = phishing_dataset[:, -1]

# Chia dữ liệu thành tập huấn luyện và tập kiểm thử
training_samples, testing_samples, training_targets, testing_targets = train_test_spli
    samples, targets, test_size=0.2, random_state=0)

# Sử dụng Decision Tree
tree_classifier = DecisionTreeClassifier()
tree_classifier.fit(training_samples, training_targets)
```

```
# Dự đoán và đánh giá mô hình
predictions_tree = tree_classifier.predict(testing_samples)
accuracy_tree = 100.0 * accuracy_score(testing_targets, predictions_tree)
predictions_tree = tree_classifier.predict(testing_samples)

accuracy_tree = accuracy_score(testing_targets, predictions_tree)
precision_tree = precision_score(testing_targets, predictions_tree)
recall_tree = recall_score(testing_targets, predictions_tree)
f1_tree = f1_score(testing_targets, predictions_tree)
print("Decision Tree accuracy: " + str(accuracy_tree))

print("Decision Tree Metrics:")
print("Accuracy: {:.2f}".format(accuracy_tree * 100))
print("Precision: {:.2f}".format(precision_tree))
print("Recall: {:.2f}".format(recall_tree))
print("F1 Score: {:.2f}".format(f1_tree))
```

```
Decision Tree accuracy: 0.9624604251469923
Decision Tree Metrics:
Accuracy: 96.25
Precision: 0.96
Recall: 0.97
F1 Score: 0.97
```