Technological Institute of the Philippines	Quezon City - Computer Engineering
Course Code:	CPE 019
Code Title:	Emerging Technologies in CpE 2
2nd Semester	AY 2023-2024

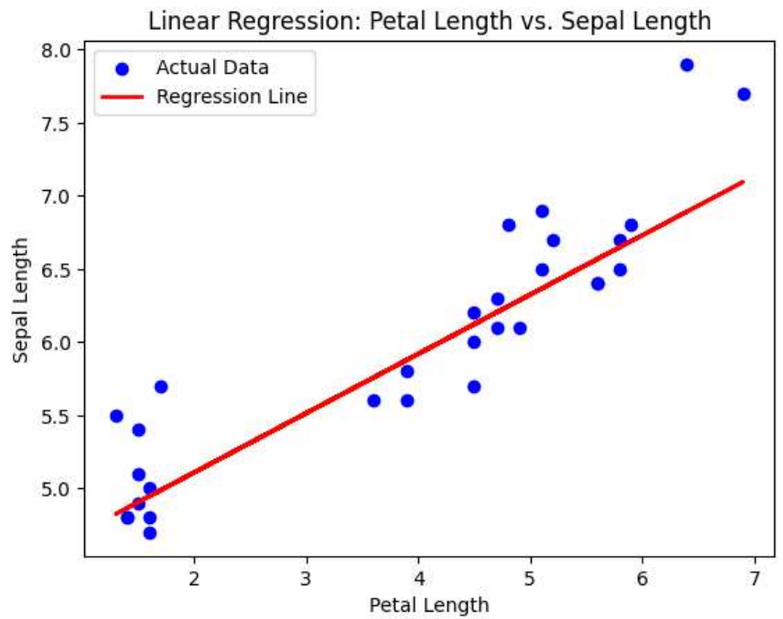
Activity	Prelim Project
Group Number	Group 14 (Beltran & Buan)
Section	CPE32S3
Date Performed:	01/03/24
Date Submitted:	05/03/24
Instructor:	Engr. Richard Roman

Linear Regression

Singular Linear Regression

```
from sklearn import datasets
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
import numpy as np


#loading the IRIS dataset
iris = datasets.load_iris()

#preparing the dataset
X = iris.data[:, 2:3]
y = iris.data[:, 0]

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)


#initializing the linear regression model and then training it
model = LinearRegression()
model.fit(X_train, y_train)

#prediction for the test set
y_pred = model.predict(X_test)
```

```
import matplotlib.pyplot as plt

# Scatter plot of actual data points
plt.scatter(X_test, y_test, color='blue', label='Actual Data')

# Plotting the regression line
plt.plot(X_test, y_pred, color='red', linewidth=2, label='Regression Line')

# Adding labels and title
plt.xlabel('Petal Length')
plt.ylabel('Sepal Length')
plt.title('Linear Regression: Petal Length vs. Sepal Length')
plt.legend()

# Display the plot
plt.show()
```




```
#printing the coefficients to see the linear relationship between petal and sepal length
print("Coefficients:", model.coef_)

#printing the mean squared error
mse = np.mean((y_pred - y_test) ** 2)
print("Mean Squared Error:", mse)
```

```
Coefficients: [0.40532217]
Mean Squared Error: 0.129093146356764
```

Evaluation Report:
By looking at our results, we can see that we got a coefficient value of 0.4053. This value represents the connection between our two variables which are petal and sepal length. It means that everytime the petal length increases, we can anticipate the sepal length to also increase by 0.4053.

As for our mean squared error, we got a value of 0.1291. This means that the model we trained is very close to the actual values meaning that our model is fairly accurate.

Multiple Linear Regression

```
#importing libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import r2_score
```

```
#loading the IRIS dataset
iris = pd.read_csv('/content/iris.csv')
iris.head()
```

	sepal.length	sepal.width	petal.length	petal.width	variety
0	5.1	3.5	1.4	0.2	Setosa
1	4.9	3.0	1.4	0.2	Setosa
2	4.7	3.2	1.3	0.2	Setosa
3	4.6	3.1	1.5	0.2	Setosa
4	5.0	3.6	1.4	0.2	Setosa

```
# converting the values of the variety column into numerical values
iris['variety'] = iris['variety'].astype('category')
iris['variety'] = iris['variety'].cat.codes

# checking if the values are numerical values
iris.head()
```

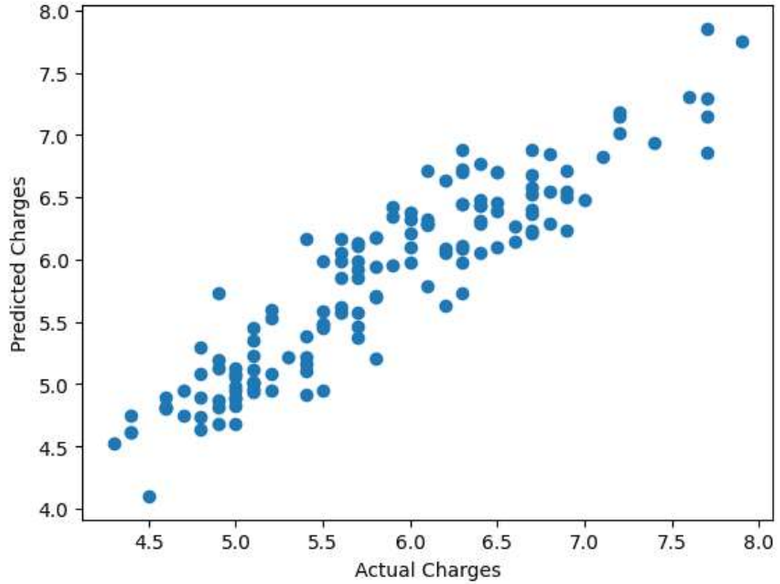
	sepal.length	sepal.width	petal.length	petal.width	variety
0	5.1	3.5	1.4	0.2	0
1	4.9	3.0	1.4	0.2	0
2	4.7	3.2	1.3	0.2	0
3	4.6	3.1	1.5	0.2	0
4	5.0	3.6	1.4	0.2	0

```
# preparing the data for training
X = iris.drop(columns = ['variety', 'sepal.length'])
y = iris['sepal.length']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.03, random_state = 0)


# creating a linear regression object and fitting the model with the training set
Lreg = LinearRegression()
Lreg.fit(X_train, y_train)
```

▾ LinearRegression

LinearRegression()

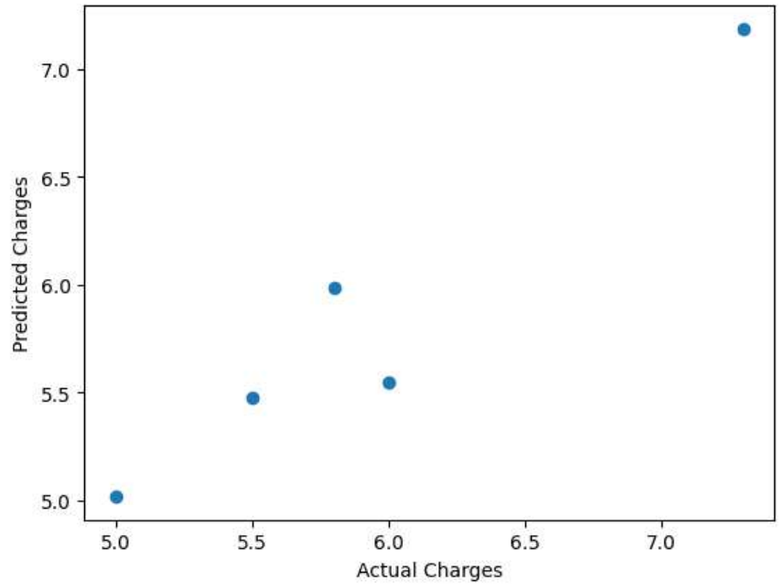
```
# making predictions using the training data
y_pred_train = Lreg.predict(X_train)

# visualizing the result
plt.scatter(y_train, y_pred_train)
plt.xlabel("Actual Charges")
plt.ylabel("Predicted Charges")
plt.show()
```




```
# showing the coefficients
print("Coefficients: ", Lreg.coef_)

# showing the R2-score of the model using the training data
r2_train = r2_score(y_train, y_pred_train)
print("R2 Score (Training): ", r2_train)
```

```
Coefficients:  [ 0.66050041  0.69113178 -0.51143859]
R2 Score (Training):  0.8567645287402356
```

```
# making predictions using the testing data
y_pred_test = Lreg.predict(X_test)

# visualizing the data
plt.scatter(y_test, y_pred_test)
plt.xlabel("Actual Charges")
plt.ylabel("Predicted Charges")
plt.show()
```




```
# showing the coefficients
print("Coefficients: ", Lreg.coef_)

# showing the R2-score of the model using the testing data
r2_test = r2_score(y_test, y_pred_test)
print("R2 Score (Testing): ", r2_test)
```

```
Coefficients:  [ 0.66050041  0.69113178 -0.51143859]
R2 Score (Testing):  0.9131919415587352
```

```
# Calculate adjusted R^2
n = len(y_train)
p = X_train.shape[1]
adjusted_r2_train = 1 - (1 - r2_train) * (n - 1) / (n - p - 1)
print("Adjusted R2 Score (Training):", adjusted_r2_train)

# Calculate adjusted R^2 for testing data
n = len(y_test)
p = X_test.shape[1]
adjusted_r2_test = 1 - (1 - r2_test) * (n - 1) / (n - p - 1)
print("Adjusted R2 Score (Testing):", adjusted_r2_test)
```

```
Adjusted R2 Score (Training): 0.8537169655219428
Adjusted R2 Score (Testing): 0.6527677662349407
```

Evaluation Report:

For our testing and training sets we got the coefficients: Sepal Length = 0.6605 Petal Length = 0.6911 Petal Width = -0.5114

Because the coefficients we got are the same for the two sets, it means that the relationship between the models independent and dependent variable is consistent.

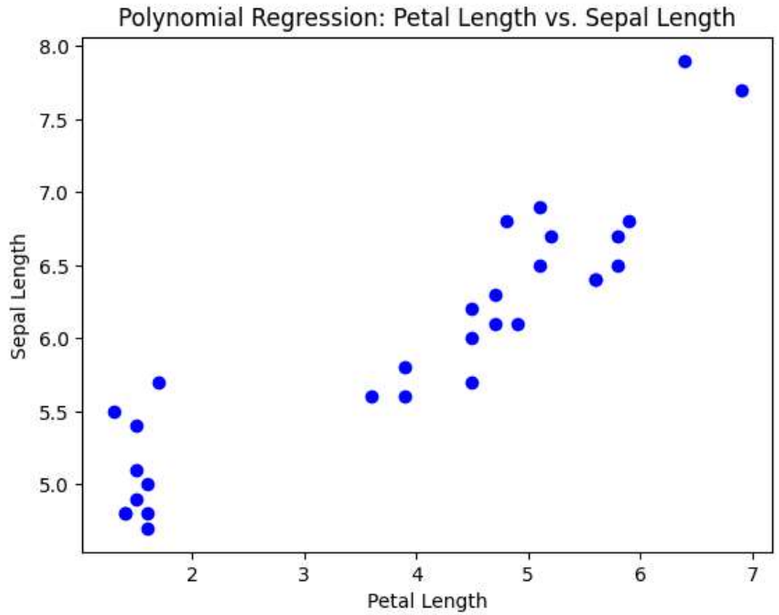
For the Predicted R2 Scores, the training set got a Predicted R2 score of 0.857 (85.7%) while the testing set got an R2 score of 0.913 (91.3%).

By analyzing the R2 scores that we obtained, we can determine that our model fits our testing data.

Polynomial Linear Regression

Since Polynomial Linear Regression is built on Linear Regression, we are going to use the previous linear regression model for Polynomial Linear Regression by transforming the input features into polynomial features and fitting the model accordingly.

```
#importing the polynomial features
from sklearn.preprocessing import PolynomialFeatures

#defining the degree of the polynomial
poly_degree = 2
poly = PolynomialFeatures(degree = poly_degree)

#transforming the input features of both training and test dataset
X_poly_train = poly.fit_transform(X_train)
X_poly_test = poly.transform(X_test)

#initializing the linear regression model and then training it
model2 = LinearRegression()
model2.fit(X_poly_train, y_train)

#prediction for the test set
y_pred_poly = model2.predict(X_poly_test)


# Visualising the Polynomial Regression results
plt.scatter(X_test, y_test, color='blue')


plt.title('Polynomial Regression: Petal Length vs. Sepal Length')
plt.xlabel('Petal Length')
plt.ylabel('Sepal Length')

plt.show()
```




```
mse_poly = np.mean((y_pred_poly - y_test) ** 2)
print(f"Polynomial Regression (Degree {poly_degree}) Mean Squared Error:", mse_poly)
```

```
Polynomial Regression (Degree 2) Mean Squared Error: 0.09749152969192712
```

Evaluation Report: By looking at our results, the model's mean squared error of 0.0974915 indicates that it performs very well. The polynomial regression of degree 2 means that it tries to find a curved relationship between two features which is the petal length and sepal length.

Logistic Regression

```
#loading all the needed libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sb
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import confusion_matrix


#loading the IRIS dataset and converting it into a dataframe for the logistic regression algorithm
```

```
#loading the IRIS dataset and converting it into a dataframe for the logistic regression algorithm
iris = pd.read_csv('/content/iris.csv')

# converting the values of the variety column into numerical values
iris['variety'] = iris['variety'].astype('category')
iris['variety'] = iris['variety'].cat.codes

#displaying the first 5 rows of the IRIS dataset
iris.head(5)
```

	sepal.length	sepal.width	petal.length	petal.width	variety
0	5.1	3.5	1.4	0.2	0
1	4.9	3.0	1.4	0.2	0
2	4.7	3.2	1.3	0.2	0
3	4.6	3.1	1.5	0.2	0
4	5.0	3.6	1.4	0.2	0

Next steps: View recommended plots

```
#preparing the data for training
X = iris.drop('variety', axis = 1)
y = iris['variety']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.5, random_state=42)


#initializaing the logistic regression algorithm and then training it
Lreg = LogisticRegression(max_iter = 500)

Lreg.fit(X_train, y_train)
```

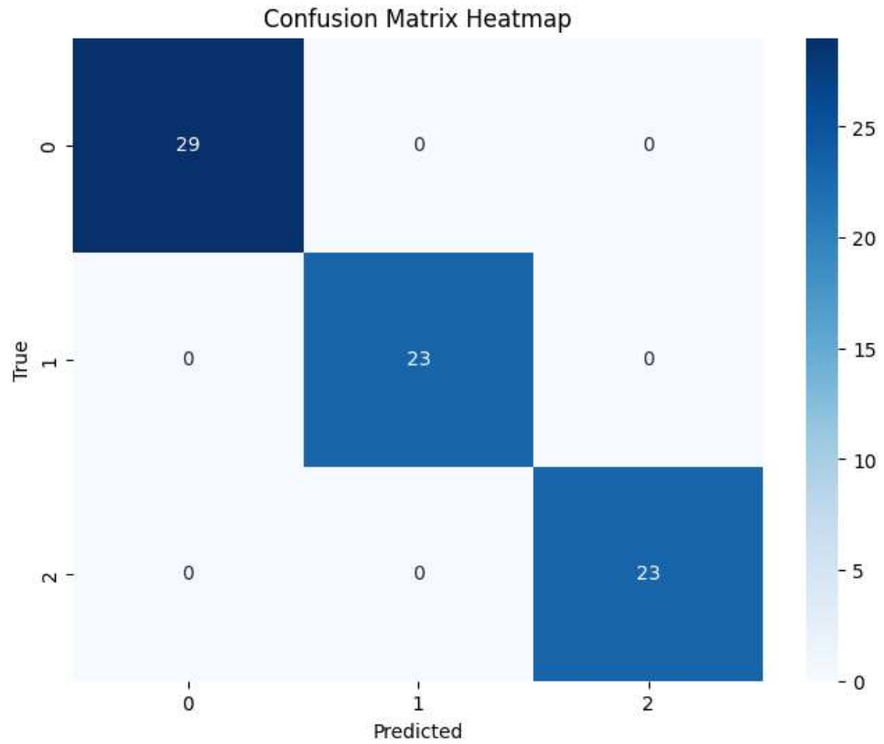
```
LogisticRegression(max_iter=500)
```

```
#making predictions on the test set
y_pred = Lreg.predict(X_test)


# displaying the confusion matrix
c_matrix = confusion_matrix(y_test, y_pred)
print(c_matrix)
```

```
[[29  0  0]
 [ 0 23  0]
```

```
      [ 0  0 23]]
```

```
# visualizing the confusion matrix
plt.figure(figsize=(8, 6))
sb.heatmap(c_matrix, annot=True, fmt="d", cmap="Blues")
plt.xlabel('Predicted')
plt.ylabel('True')
plt.title('Confusion Matrix Heatmap')
plt.show()
```



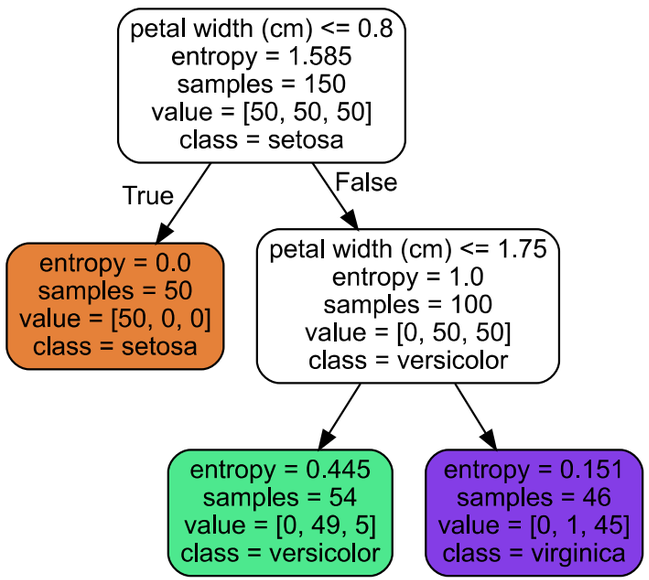
Evaluation Report:

In order to test the accuracy of our model, we used a confusion matrix. By looking at our confusion matrix, we can see that it identified 29 instances of class 1 and did not identify any as class 2 or 3. Similarly, the two succeeding rows did not misidentify any classes. This tells us that our model is performing very well because we did not get any misclassifications.

```
#loading all the needed libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sb
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import confusion_matrix

#loading the IRIS dataset and converting it into a dataframe for the logistic regression algorithm
iris = pd.read_csv('/content/iris.csv')

# converting the values of the variety column into numerical values
iris['variety'] = iris['variety'].astype('category')
iris['variety'] = iris['variety'].cat.codes

#displaying the first 5 rows of the IRIS dataset
iris.head(5)

#preparing the data for training
X = iris.drop('variety', axis = 1)
y = iris['variety']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.5, random_state=42)

#initializaing the logistic regression algorithm and then training it
Lreg = LogisticRegression(max_iter = 500)

Lreg.fit(X_train, y_train)

#making predictions on the test set
y_pred = Lreg.predict(X_test)

# displaying the confusion matrix
c_matrix = confusion_matrix(y_test, y_pred)
print(c_matrix)
```

Decision Tree

```
#Loading all the necessary libraries
from sklearn.datasets import load_iris
from sklearn.tree import DecisionTreeClassifier
from sklearn.tree import export_graphviz
from graphviz import Source

#Loading the iris dataset
iris = load_iris()
X = iris.data[:, 2:]
y = iris.target

#DecisionTreeClassifier
tree_clf = DecisionTreeClassifier(criterion='entropy', max_depth=2)
tree_clf.fit(X, y)

# Plotting the decision tree graph
export_graphviz( tree_clf, out_file="iris_tree.dot", feature_names=iris.feature_names[2:], class_names=iris.target_names, rounded=True, filled=True)

with open("iris_tree.dot") as f:
    dot_graph = f.read()

Source(dot_graph)
```



Evaluation Report: The Decision Tree generated a graph that shows nodes representing a decision based on the characteristics of the flowers, specifically their petal length and width. Consequently, these decisions have lead to different classes of iris flowers.

Random Forest

```
#importing necessary libraries and dataset
import pandas as pd
import numpy as np
from sklearn.datasets import load_iris
from sklearn.ensemble import RandomForestClassifier

np.random.seed(0)


#loading the dataset

iris = load_iris()

iris_df = pd.DataFrame(iris.data, columns = iris.feature_names)


#Adding a new column for species name
iris_df['species'] = pd.Categorical.from_codes(iris.target, iris.target_names)
iris_df.head()
```

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	species
0	5.1	3.5	1.4	0.2	setosa
1	4.9	3.0	1.4	0.2	setosa
2	4.7	3.2	1.3	0.2	setosa
3	4.6	3.1	1.5	0.2	setosa
4	5.0	3.6	1.4	0.2	setosa

```
#Creating a test and train data
iris_df['is_trained'] = np.random.uniform(0, 1, len(iris_df)) <= 0.75
iris_df.head()
```

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	species	is_trained
0	5.1	3.5	1.4	0.2	setosa	True
1	4.9	3.0	1.4	0.2	setosa	True
2	4.7	3.2	1.3	0.2	setosa	True
3	4.6	3.1	1.5	0.2	setosa	True
4	5.0	3.6	1.4	0.2	setosa	True

```
#Creating dataframes with test rows and training rows
train, test = iris_df[iris_df['is_trained'] ==True], iris_df[iris_df['is_trained'] ==False]

print('Number of values in the training data: ', len(train))
print('Number of values in the test data: ', len(test))
```

```
Number of values in the training data:  118
Number of values in the test data:  32
```

```
#Creating a list of the features column's names
features = iris_df.columns[:4]

#Converting the species name into digits
y = pd.factorize(train['species'])[0]

y
```

```
array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2])
```

```
#Creating the Random Forest Classifier
clf = RandomForestClassifier(n_jobs = 2, random_state = 0)

#Training the classifier
clf.fit(train[features], y)
```

```
RandomForestClassifier
```