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

Prelim Exam	**Group 3**
Name	Albo, Russel Mesa, Eric Railey
Section	CPE32S9
Date Performed:	March 5, 2024
Date Submitted:	March 5, 2024
Instructor:	Engr. Roman M. Richard

- Choose any dataset applicable for classification and/or prediction analysis problems.
- Show the application of the following algorithms:
- Linear Regression

> Singular LR
> Multiple LR
> Polynomial LR

- Logistic Regression
- Decision Tree
- Random Forest
- Provide Evaluation reports for all models

```
import numpy as np
import pandas as pd
import seaborn as sns
import os
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error, r2_score
from sklearn.preprocessing import PolynomialFeatures
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
from sklearn.compose import ColumnTransformer
from sklearn.ensemble import RandomForestClassifier
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import MinMaxScaler
from sklearn.tree  import DecisionTreeClassifier
from sklearn.tree import plot_tree

path = "/content/data.csv"
laptop = pd.read_csv(path)
laptop.head()
```

	Unnamed: 0.1	Unnamed: 0	brand	name	price	spec_rating	processor	CPU	Ram
0	0	0	HP	Victus 15-fb0157AX Gaming Laptop	49900	73.000000	5th Gen AMD Ryzen 5 5600H	Hexa Core, 12 Threads	8GB
1	1	1	HP	15s-fq5007TU Laptop	39900	60.000000	12th Gen Intel Core i3 1215U	Hexa Core (2P + 4E), 8 Threads	8GB
2	2	2	Acer	One 14 Z8-415 Laptop	26990	69.323529	11th Gen Intel Core i3 1115G4	Dual Core, 4 Threads	8GB
3	3	3	Lenovo	Yoga Slim 6 14IAP8 82WU0095IN Laptop	59729	66.000000	12th Gen Intel Core i5 1240P	12 Cores (4P + 8E), 16 Threads	16GB

```
data = pd.DataFrame(laptop)


#This code is used for removing the two unnamed columns
data = laptop.drop(laptop.columns[:2], axis=1)


data.head()
```

	brand	name	price	spec_rating	processor	CPU	Ram	Ram_type	ROM	R
0	HP	Victus 15-fb0157AX Gaming Laptop	49900	73.000000	5th Gen AMD Ryzen 5 5600H	Hexa Core, 12 Threads	8GB	DDR4	512GB	
1	HP	15s-fq5007TU Laptop	39900	60.000000	12th Gen Intel Core i3 1215U	Hexa Core (2P + 4E), 8 Threads	8GB	DDR4	512GB	
2	Acer	One 14 Z8-415 Laptop	26990	69.323529	11th Gen Intel Core i3 1115G4	Dual Core, 4 Threads	8GB	DDR4	512GB	

Singular Linear Regression

```
data['Ram'] = data.Ram.str.replace('GB', '')
data['ROM'] = data.ROM.str.replace('GB', '')
data['ROM'] = data.ROM.str.replace('TB', '000')
data.rename(columns = {'ROM':'ROM_GB'})
data['ROM'] = data['ROM'].astype(int)
data['Ram'] = data['Ram'].astype(int)


dummies = pd.get_dummies(data)
dummies
```

	price	spec_rating	Ram	ROM	display_size	resolution_width	resolution_height
0	49900	73.000000	8	512	15.6	1920.0	1080.0
1	39900	60.000000	8	512	15.6	1920.0	1080.0
2	26990	69.323529	8	512	14.0	1920.0	1080.0
3	59729	66.000000	16	512	14.0	2240.0	1400.0
4	69990	69.323529	8	256	13.3	2560.0	1600.0
...	...	...	...	...	...	...	...
888	44990	69.323529	8	512	15.6	1920.0	1080.0
889	110000	71.000000	16	1000	15.6	2560.0	1440.0
890	189990	89.000000	32	1000	14.0	2560.0	1600.0
891	129990	73.000000	16	512	15.6	1920.0	1080.0
892	131990	84.000000	16	1000	15.6	1920.0	1080.0

893 rows × 1228 columns

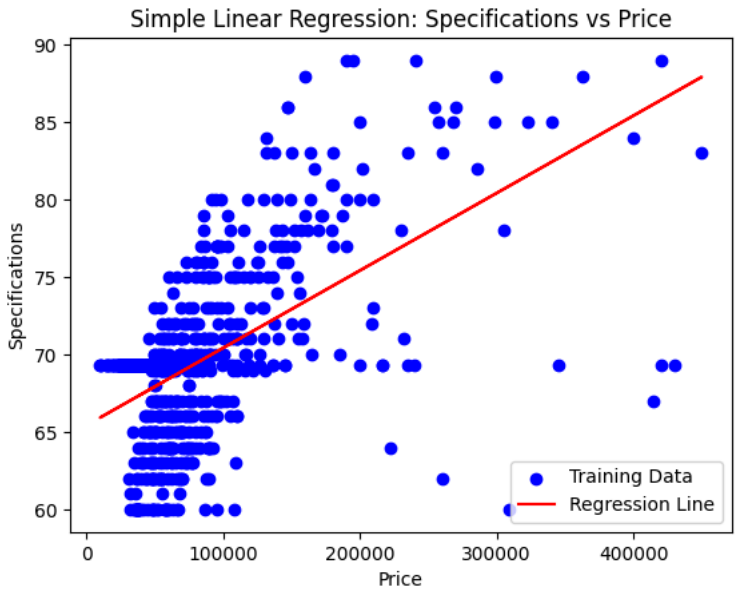
```
x = data[['price']]
y = data['spec_rating']


x_train, x_test, y_train, y_test = train_test_split(x,y, test_size= .2, random_state=42)


model = LinearRegression()
model.fit(x_train, y_train)
```

```
▾ LinearRegression
LinearRegression()
```

```
train_score = model.score(x_train, y_train)
train_score
```

```
0.3139568138871206
```

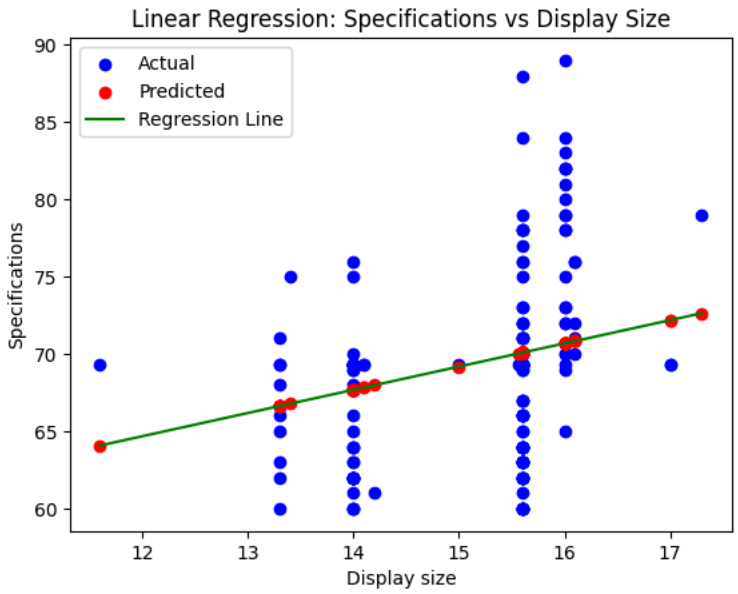
```
y_pred = model.predict(x_test)
plt.scatter(x_train, y_train, color='blue', label='Training Data')
plt.plot(x_train, model.predict(x_train), color='red', label='Regression Line')
plt.xlabel('Price')
plt.ylabel('Specifications')
plt.title('Simple Linear Regression: Specifications vs Price')
plt.legend()
plt.show()
```




```
mse = mean_squared_error(y_test, y_pred)
r_squared = r2_score(y_test, y_pred)

print("Mean Squared Error:", mse)
print("R-squared:", r_squared)
```

```
Mean Squared Error: 25.18826529745557
R-squared: 0.24042520626648756
```

```
X = data[['display_size'] ]
y = data['spec_rating']

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

model = LinearRegression()
model.fit(X_train, y_train)
predictions = model.predict(X_test)

plt.scatter(X_test['display_size'], y_test, color='blue', label='Actual')
plt.scatter(X_test['display_size'], predictions, color='red', label='Predicted')

x_values = np.linspace(X_test['display_size'].min(), X_test['display_size'].max(), 100)
y_values = model.coef_[0] * x_values + model.intercept_
plt.plot(x_values, y_values, color='green', label='Regression Line')

plt.xlabel('Display size')
plt.ylabel('Specifications')
plt.title('Linear Regression: Specifications vs Display Size')
plt.legend()
plt.show()
```




```
mse = mean_squared_error(y_test, y_pred)
r_squared = r2_score(y_test, y_pred)

print("Mean Squared Error:", mse)
print("R-squared:", r_squared)
```

```
Mean Squared Error: 25.18826529745557
R-squared: 0.24042520626648756
```

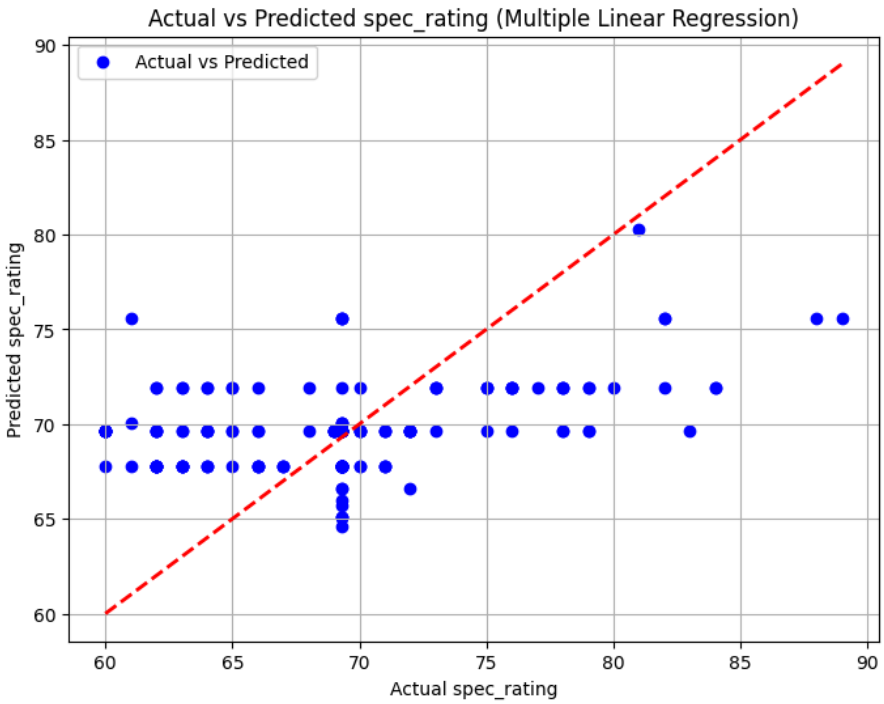
Multiple Regression

```
# Multipe Linear Regression
X = data[['Ram', 'ROM']]
y = data['spec_rating']

# Splitting the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Create a linear regression model
model = LinearRegression()

# Fit the model to the training data
model.fit(X_train, y_train)

# Make predictions
y_pred = model.predict(X_test)

# Evaluate the model
from sklearn.metrics import mean_squared_error, r2_score

mse = mean_squared_error(y_test, y_pred)
r_squared = r2_score(y_test, y_pred)

print("Mean Squared Error:", mse)
print("R-squared:", r_squared)

# Print the coefficients
coefficients = pd.DataFrame({'feature': X.columns, 'coefficient': model.coef_})
print("\nCoefficients:")
print(coefficients)

# Print the intercept
print("\nIntercept:", model.intercept_)
```

```
Mean Squared Error: 28.058473504838744
R-squared: 0.15387149638018027

Coefficients:
```

```
  feature  coefficient
0     Ram     0.230397
1     ROM     0.004666

Intercept: 63.55782157424087
```

```
plt.figure(figsize=(8, 6))
plt.scatter(y_test, y_pred, color='blue', label='Actual vs Predicted')
plt.plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max()], color='red', linestyle='--', lw=2)
plt.xlabel('Actual spec_rating')
plt.ylabel('Predicted spec_rating')
plt.title('Actual vs Predicted spec_rating (Multiple Linear Regression)')
plt.legend()
plt.grid(True)
plt.show()
```



Polynomial Regression

```
X = data[['spec_rating']]
y = data['price']
degree = 2
```

```
poly_features = PolynomialFeatures(degree=degree)
X_poly = poly_features.fit_transform(X)
```

```
model = LinearRegression()
model.fit(X_poly, y)
```

```
LinearRegression()
```

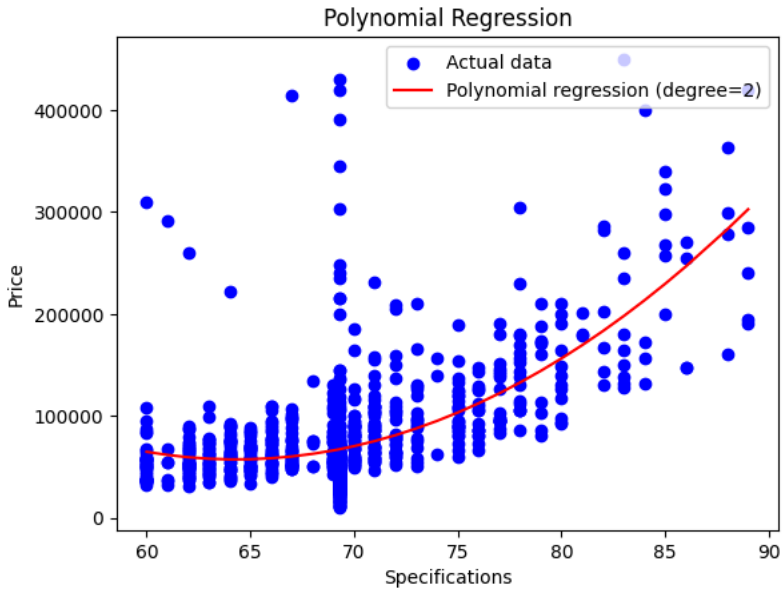
```
y_pred = model.predict(X_poly)
```

```
mse = mean_squared_error(y, y_pred)
print("Mean Squared Error:", mse)
print('R-squared:', r2_score(y, y_pred))
```

```
Mean Squared Error: 2194246611.2234645
R-squared: 0.40731746731895446
```

```
plt.scatter(X, y, color='blue', label='Actual data')
X_sorted, y_pred_sorted = zip(*sorted(zip(X.values, y_pred)))
plt.plot(X_sorted, y_pred_sorted, color='red', label='Polynomial regression (degree=' + str(degree) + ')')
plt.xlabel('Specifications')
plt.ylabel('Price')
plt.title('Polynomial Regression')
plt.legend()
```

```
<matplotlib.legend.Legend at 0x7c4787d6b8b0>
```



Logistic Regression

```
data['spec_rating_class'] = pd.cut(data['spec_rating'], bins=3, labels=[1, 2, 3])

# Selecting features and target variable
X = data[['Ram', 'ROM',  'resolution_width', 'resolution_height']]
y = data['spec_rating_class']

# Splitting the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Scaling the features
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Creating a logistic regression model
model = LogisticRegression()

# Fitting the model to the scaled training data
model.fit(X_train_scaled, y_train)

# Making predictions on the scaled testing data
y_pred = model.predict(X_test_scaled)

# Evaluating the model
print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))
print("\nClassification Report:")
print(classification_report(y_test, y_pred))
```

```
Confusion Matrix:
[[121   3   3]
 [ 36   6   0]
 [  2   5   3]]

Classification Report:
              precision    recall  f1-score   support

           1       0.76      0.95      0.85       127
           2       0.43      0.14      0.21        42
```

```
              3       0.50      0.30      0.37        10

       accuracy                           0.73       179
      macro avg       0.56      0.47      0.48       179
   weighted avg       0.67      0.73      0.67       179
```

```
model_LR = LogisticRegression(C=10)
model_LR.fit(X_train,y_train)
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_logistic.py:458: Converger
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
```

```
LogisticRegression
LogisticRegression(C=10)
```

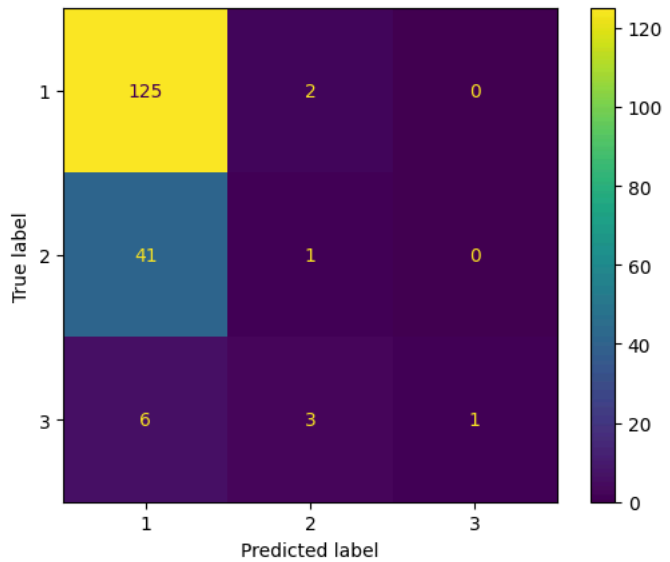
```
score_LR_train = model_LR.score(X_train,y_train)
print(f"Train accuracy: {score_LR_train}")
```

```
Train accuracy: 0.6848739495798319
```

```
score_LR_test = model_LR.score(X_test,y_test)
print(f"Test accuracy: {score_LR_test}")
```

```
Test accuracy: 0.7094972067039106
```

```
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay

ConfusionMatrixDisplay.from_estimator(model_LR,X_test,y_test);
```



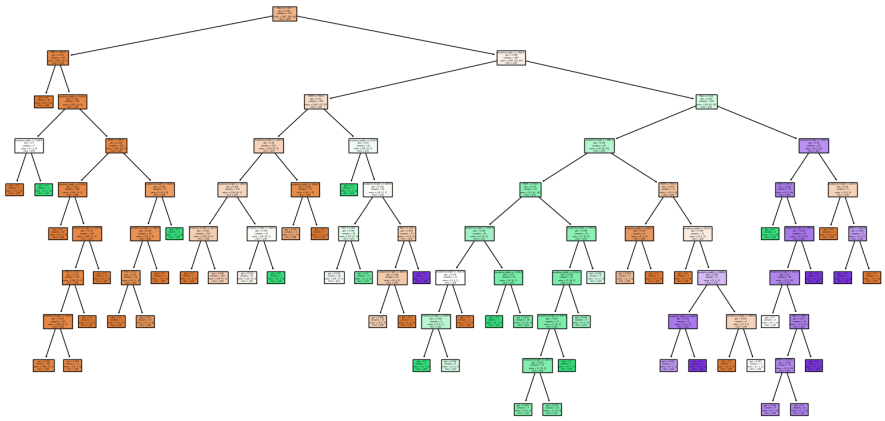
Decision Tree

```
classifier = DecisionTreeClassifier(criterion = 'entropy', random_state = 0)
classifier.fit(X_train, y_train)
```

```
DecisionTreeClassifier
DecisionTreeClassifier(criterion='entropy', random_state=0)
```

```
# Split data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Create and train the model
model = DecisionTreeClassifier()
model.fit(X_train, y_train)

# Make predictions
predictions = model.predict(X_test)

# Evaluate the model
accuracy = accuracy_score(y_test, predictions)
print("Accuracy:", accuracy)

# Print classification report
print("Classification Report:")
print(classification_report(y_test, predictions))

# Print confusion matrix
print("Confusion Matrix:")
print(confusion_matrix(y_test, predictions))

# Plot decision tree
plt.figure(figsize=(20,10))
plot_tree(model, filled=True, feature_names=X.columns, class_names=True)
plt.show()
```

```
Accuracy: 0.7150837988826816
Classification Report:
              precision    recall  f1-score   support

           1       0.78      0.87      0.83       127
           2       0.45      0.31      0.37        42
           3       0.50      0.40      0.44        10

    accuracy                           0.72       179
   macro avg       0.58      0.53      0.55       179
weighted avg       0.69      0.72      0.70       179

Confusion Matrix:
[[111  12   4]
 [ 29  13   0]
 [  2   4   4]]
```



```
#create the array for the target values
y_target = data['price'].values



columns = ['ROM', 'Ram', 'resolution_width','resolution_height']
#create the variable to hold the features that the classifier will use
X_input = data[list(columns)].values


#code cell 10
#import the tree module from the sklearn library
from sklearn import tree
#create clf_train as a decision tree classifier object
clf_train = tree.DecisionTreeClassifier(criterion="entropy", max_depth=3)
#train the model using the fit() method of the decision tree object.
#Supply the method with the input variable X_input and the target variable y_target
clf_train = clf_train.fit(X_input, y_target)
```

```
clf_train.score(X_input,y_target)
```

```
0.04143337066069429
```

```
from sklearn.tree import export_graphviz
import graphviz
#from sklearn.externals.six import StringIO
from io import StringIO
```

```
with open("/content/laptop.dot", 'w') as f:
  f = tree.export_graphviz(clf_train, out_file=f, feature_names=columns)
```

```
!apt-get -qq install -y graphviz && pip install pydot
```

```
Requirement already satisfied: pydot in /usr/local/lib/python3.10/dist-packages (1.4.2)
Requirement already satisfied: pyparsing>=2.1.4 in /usr/local/lib/python3.10/dist-packages (from pydot) (3.1.1)
```

```
#code cell 13
#run the Graphviz dot command to convert the .dot file to .png
!dot -Tpng laptop.dot -o laptop.png
```

```
#import the Image module from the Ipython.display libary
from IPython.display import Image
#display the decison tree graphic
Image('/content/laptop.png')
```