JULIE PHAM - 121540249 - Final Exam

------- Question 1 ------------

In [209…

```
# 1.1 Import the mtcars dataset and check data types
import pandas as pd
import numpy as np

# Load dataset
df = pd.read_csv("mtcars.csv")
df.head()
```

Out[209…

	model	mpg	cyl	disp	hp	drat	wt	qsec	vs	am	gear	carb
0	Mazda RX4	21.0	6	160.0	110	3.90	2.620	16.46	0	1	4	4
1	Mazda RX4 Wag	21.0	6	160.0	110	3.90	2.875	17.02	0	1	4	4
2	Datsun 710	22.8	4	108.0	93	3.85	2.320	18.61	1	1	4	1
3	Hornet 4 Drive	21.4	6	258.0	110	3.08	3.215	19.44	1	0	3	1
4	Hornet Sportabout	18.7	8	360.0	175	3.15	3.440	17.02	0	0	3	2

In [210…

```
print("Data Types")
df.dtypes
```

```
Data Types
```

Out[210…

```
model     object
mpg      float64
cyl        int64
disp     float64
hp         int64
drat     float64
wt       float64
qsec     float64
vs         int64
am         int64
gear       int64
carb       int64
dtype: object
```

In [211…

```
# 1.2. Are there any missing values in the dataset? --> NO
print("Missing Values")
df.isnull().sum()
```

```
Missing Values
```

Out[211…

```
model    0
mpg      0
cyl      0
disp     0
hp       0
drat     0
wt       0
qsec     0
vs       0
am       0
gear     0
carb     0
dtype: int64
```

In [212…

```
# 1.3. Print the descriptive statistics of the mtcars data to understand the data a little better (min, max, mean, median, 1st and
print(df.describe().T)
```

```
       count        mean         std     min        25%      50%      75%  \
mpg     32.0   20.090625    6.026948  10.400   15.42500   19.200   22.80
cyl     32.0    6.187500    1.785922   4.000    4.00000    6.000    8.00
disp    32.0  230.721875  123.938694  71.100  120.82500  196.300  326.00
hp      32.0  146.687500   68.562868  52.000   96.50000  123.000  180.00
drat    32.0    3.596563    0.534679   2.760    3.08000    3.695    3.92
wt      32.0    3.217250    0.978457   1.513    2.58125    3.325    3.61
qsec    32.0   17.848750    1.786943  14.500   16.89250   17.710   18.90
vs      32.0    0.437500    0.504016   0.000    0.00000    0.000    1.00
am      32.0    0.406250    0.498991   0.000    0.00000    0.000    1.00
gear    32.0    3.687500    0.737804   3.000    3.00000    4.000    4.00
carb    32.0    2.812500    1.615200   1.000    2.00000    2.000    4.00

          max
mpg    33.900
cyl     8.000
disp  472.000
hp    335.000
drat    4.930
wt      5.424
qsec   22.900
vs      1.000
am      1.000
gear    5.000
carb    8.000
```

In [213…

```
# 1.4. Splitting the Data-Set into Independent and Dependent Features
X = df.drop(columns=["am"]) # Independent variables
y = df["am"] # Dependent variable Transmission(am))
```

In [214…

```
# Count the occurrences ---> not imbalance
counts = y.value_counts()
# Calculate percentages
percentages = counts / len(y) * 100
print("Counts:\n", counts)
print("\nPercentages:\n", percentages)
```

```
Counts:
 am
0    19
1    13
Name: count, dtype: int64

Percentages:
 am
0    59.375
1    40.625
Name: count, dtype: float64
```

In [215…

```
# 1.5. Convert categorical variable into numeric Using one hot encoding method
from sklearn.preprocessing import OneHotEncoder, StandardScaler
categorical_cols = X.select_dtypes(include=['object']).columns
if len(categorical_cols) > 0:
    encoder = OneHotEncoder(drop='first', sparse_output=False)
    encoded_df = pd.DataFrame(
        encoder.fit_transform(X[categorical_cols]),
        columns=encoder.get_feature_names_out(categorical_cols),
        index=X.index
    )
    X = pd.concat([X.drop(columns=categorical_cols), encoded_df], axis=1)
X.head()
```

Out[215…

	mpg	cyl	disp	hp	drat	wt	qsec	vs	gear	carb	…	model_Merc 280C	model_Merc 450SE	model_Merc 450SL	model_Merc 450SLC	model_Pontiac Firebird	model_F
0	21.0	6	160.0	110	3.90	2.620	16.46	0	4	4	…	0.0	0.0	0.0	0.0	0.0	
1	21.0	6	160.0	110	3.90	2.875	17.02	0	4	4	…	0.0	0.0	0.0	0.0	0.0	
2	22.8	4	108.0	93	3.85	2.320	18.61	1	4	1	…	0.0	0.0	0.0	0.0	0.0	
3	21.4	6	258.0	110	3.08	3.215	19.44	1	3	1	…	0.0	0.0	0.0	0.0	0.0	
4	18.7	8	360.0	175	3.15	3.440	17.02	0	3	2	…	0.0	0.0	0.0	0.0	0.0	

5 rows × 41 columns

In [216…
```
# 1.6. Normalize the data set.
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
```

In [217…
```
# 1.7. Divide the dataset to training and test sets.
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(
    X_scaled, y, test_size=0.2, random_state=42
)
```

In [218…
```
# 1.8. Use the decision tree, KNN to predict the test set out values.
from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier
```

In [219…
```
# Decision Tree
dt_model = DecisionTreeClassifier(random_state=42)
dt_model.fit(X_train, y_train)
y_pred_dt = dt_model.predict(X_test)
```

In [220…
```
# KNN
knn_model = KNeighborsClassifier(n_neighbors=5)
knn_model.fit(X_train, y_train)
y_pred_knn = knn_model.predict(X_test)
```

In [221…
```
# 1.9. Display the confusion matrix to evaluate the model performance.
from sklearn.metrics import confusion_matrix, accuracy_score
import seaborn as sns
import matplotlib.pyplot as plt
```

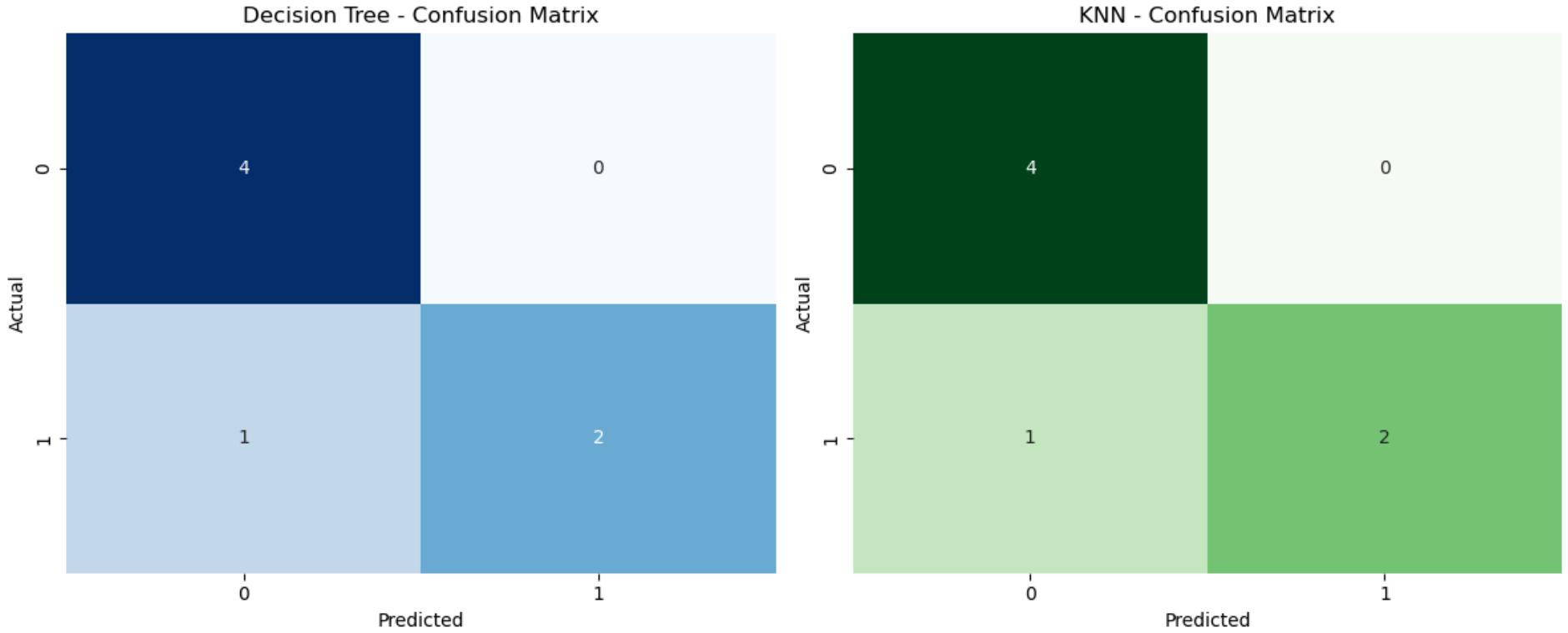
```
cm_dt = confusion_matrix(y_test, y_pred_dt)
cm_knn = confusion_matrix(y_test, y_pred_knn)

plt.figure(figsize=(12, 5))

# Decision Tree heatmap
plt.subplot(1, 2, 1)
sns.heatmap(cm_dt, annot=True, fmt="d", cmap="Blues", cbar=False)
plt.title("Decision Tree - Confusion Matrix")
plt.xlabel("Predicted")
plt.ylabel("Actual")

# KNN heatmap
plt.subplot(1, 2, 2)
sns.heatmap(cm_knn, annot=True, fmt="d", cmap="Greens", cbar=False)
plt.title("KNN - Confusion Matrix")
plt.xlabel("Predicted")
plt.ylabel("Actual")

plt.tight_layout()
plt.show()
```



In [222…

```
print("Confusion Matrix - Decision Tree")
print(cm_dt)
```

```
print("Confusion Matrix - KNN")
print(cm_knn)
```

```
Confusion Matrix - Decision Tree
[[4 0]
 [1 2]]
Confusion Matrix - KNN
[[4 0]
 [1 2]]
```

Both Decision Tree and KNN produced the exact same confusion matrix, meaning they made the same predictions on this dataset. The models are very good at predicting class 0 (automatic), but slightly weaker at predicting class 1 (manual).

In [224…

```
#1.10.Evaluate the model performance by computing Accuracy.
print("Accuracy Scores")
print(f"Decision Tree Accuracy: {accuracy_score(y_test, y_pred_dt):.2f}")
print(f"KNN Accuracy: {accuracy_score(y_test, y_pred_knn):.2f}")
```

```
Accuracy Scores
Decision Tree Accuracy: 0.86
KNN Accuracy: 0.86
```

In this case, both the Decision Tree and KNN models yielded the exact same performance metrics on this dataset. The result show that they have similar predictive power in this specific scenario (classify transmission is automatic or manual).

-------- Question 2 --------

In [227…

```
# 2.1. Import the Iris dataset
import pandas as pd

# Load dataset
df1 = pd.read_csv("iris_csv.csv")
df1.head()
```

Out[227…

	sepallength	sepalwidth	petallength	petalwidth	class
0	5.1	3.5	1.4	0.2	Iris-setosa
1	4.9	3.0	1.4	0.2	Iris-setosa
2	4.7	3.2	1.3	0.2	Iris-setosa
3	4.6	3.1	1.5	0.2	Iris-setosa
4	5.0	3.6	1.4	0.2	Iris-setosa

In [228…

```
#2.2. Remove the Species column from the iris dataset?
X = df1.drop(columns=["class"]) # Keep features only
y_true = df1["class"] # Keep the actual species for comparison
X.head()
```

Out[228…

	sepallength	sepalwidth	petallength	petalwidth
0	5.1	3.5	1.4	0.2
1	4.9	3.0	1.4	0.2
2	4.7	3.2	1.3	0.2
3	4.6	3.1	1.5	0.2
4	5.0	3.6	1.4	0.2

In [229…

```
#2.3. Create a model with three clusters that can cluster the observations in the data
from sklearn.cluster import KMeans
kmeans = KMeans(n_clusters=3, random_state=42, n_init=10)
kmeans.fit(X)
clusters = kmeans.labels_
```

```
C:\Users\ADMIN\anaconda3\Lib\site-packages\sklearn\cluster\_kmeans.py:1419: UserWarning: KMeans is known to have a memory leak on W
indows with MKL, when there are less chunks than available threads. You can avoid it by setting the environment variable OMP_NUM_TH
READS=1.
  warnings.warn(
```

In [230…

```
# 2.4 Compare clusters with actual species
comparison_df = pd.DataFrame({"Actual": y_true, "Cluster": clusters})
print("First 10 cluster assignments")
print(comparison_df.head(10))
```

```
First 10 cluster assignments
        Actual  Cluster
0  Iris-setosa        1
1  Iris-setosa        1
2  Iris-setosa        1
3  Iris-setosa        1
4  Iris-setosa        1
5  Iris-setosa        1
6  Iris-setosa        1
7  Iris-setosa        1
8  Iris-setosa        1
9  Iris-setosa        1
```

In [231…

```python
# Map cluster numbers to species using value_counts
mapping = {}
for cluster in range(3):
    mask = clusters == cluster
    mapping[cluster] = y_true[mask].value_counts().idxmax()
# finds the species with the highest count in that cluster.
# Predicted species based on mapping
predicted_species = [mapping[cluster] for cluster in clusters]

# Count mistakes
mistakes = np.sum(predicted_species != y_true)
print(f"Number of mistakes made by the model: {mistakes} out of {len(y_true)}")
```

```
Number of mistakes made by the model: 16 out of 150
```

In [232…

```python
16/150*100
```

Out[232…

```
10.666666666666668
```

16 mistakes out of 150 samples, which is roughly 10.7% error. That is pretty good for an unsupervised method like KMeans.

In [234…

```python
# Create comparison DataFrame
comparison_df = pd.DataFrame({
    "Actual Species": y_true,
    "Predicted Species": predicted_species
})
# Filter only incorrect predictions
incorrect_df = comparison_df[comparison_df["Actual Species"] != comparison_df["Predicted Species"]]
print("Incorrect Predictions")
incorrect_df
```

```
Incorrect Predictions
```

Out[234...

	Actual Species	Predicted Species
52	Iris-versicolor	Iris-virginica
77	Iris-versicolor	Iris-virginica
101	Iris-virginica	Iris-versicolor
106	Iris-virginica	Iris-versicolor
113	Iris-virginica	Iris-versicolor
114	Iris-virginica	Iris-versicolor
119	Iris-virginica	Iris-versicolor
121	Iris-virginica	Iris-versicolor
123	Iris-virginica	Iris-versicolor
126	Iris-virginica	Iris-versicolor
127	Iris-virginica	Iris-versicolor
133	Iris-virginica	Iris-versicolor
138	Iris-virginica	Iris-versicolor
142	Iris-virginica	Iris-versicolor
146	Iris-virginica	Iris-versicolor
149	Iris-virginica	Iris-versicolor

Most errors are between Versicolor and Virginica

Cluster assignments for these two species overlap a lot.

KMeans groups by feature similarity, and in the iris dataset, Versicolor and Virginica features are closer to each other compared to Setosa.

Setosa is never misclassified, which is expected because Setosa is well-separated in feature space (it's easily clustered).

In [265...

```
from sklearn.metrics import classification_report

# Generate classification report
report = classification_report(y_true, predicted_species)
print("Classification Report")
print(report)
```

```
Classification Report
                precision    recall  f1-score   support

    Iris-setosa       1.00      1.00      1.00        50
Iris-versicolor       0.77      0.96      0.86        50
 Iris-virginica       0.95      0.72      0.82        50

       accuracy                           0.89       150
      macro avg       0.91      0.89      0.89       150
   weighted avg       0.91      0.89      0.89       150
```

Accuracy = 0.89 (89%) --> Good

Setosa: Perfectly separated; Setosa is easily clustered.

Versicolor: Most Versicolor points were correctly identified; a few were misclassified as Virginica (lower precision).

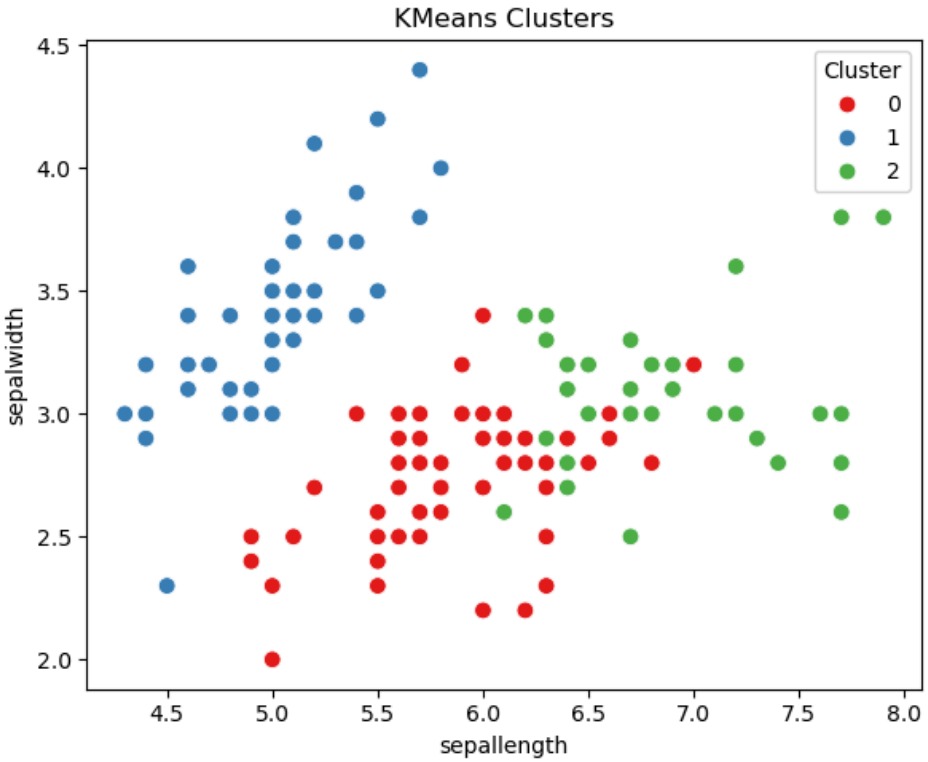
Virginica: Many Virginica points were misclassified as Versicolor (lower recall).

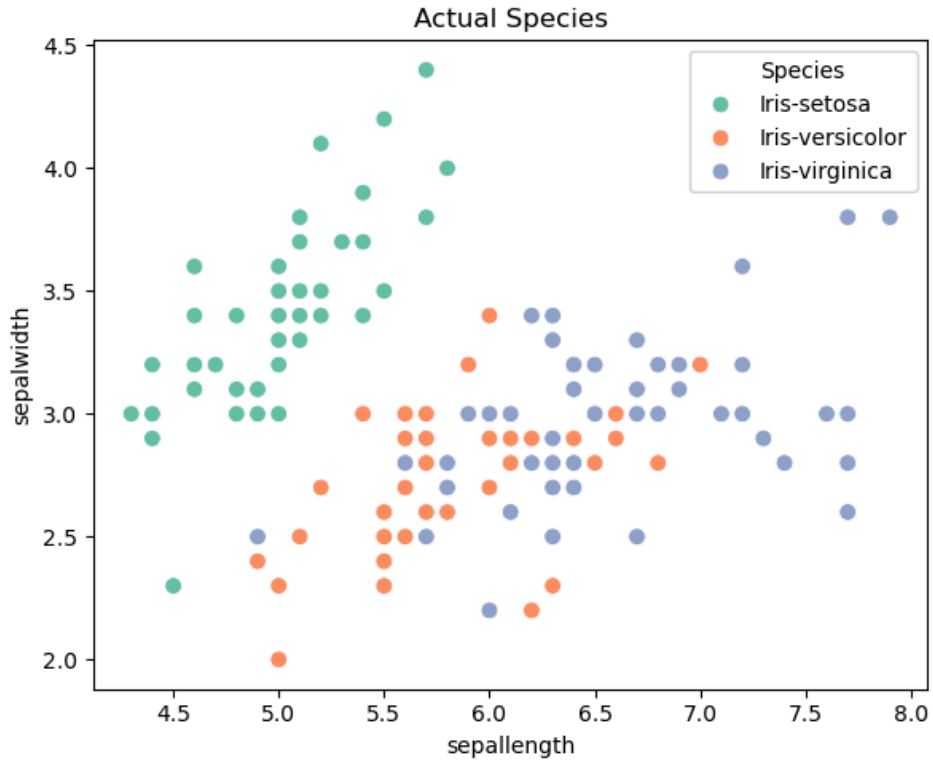
In [236…

```python
# Visualization: Clusters vs Actual Species
plt.figure(figsize=(12, 5))

# Left: KMeans clusters
plt.subplot(1, 2, 1)
sns.scatterplot(x=X.iloc[:, 0], y=X.iloc[:, 1], hue=clusters, palette="Set1", s=60)
plt.title("KMeans Clusters")
plt.xlabel(X.columns[0])
plt.ylabel(X.columns[1])
plt.legend(title="Cluster", loc="best")
# Right: True species
plt.subplot(1, 2, 2)
sns.scatterplot(x=X.iloc[:, 0], y=X.iloc[:, 1], hue=y_true, palette="Set2", s=60)
plt.title("Actual Species")
plt.xlabel(X.columns[0])
plt.ylabel(X.columns[1])
plt.legend(title="Species", loc="best")

plt.tight_layout()
plt.show()
```





Setosa points are isolated, while Versicolor and Virginica points are partially overlapping, explaining these 16 errors.