Bismillahhi Rehmani Rahim

Supervised Learning

*Supervised learning is a type of **Machine learning** where an algorithm is trained on a **labeled dataset**. The goal is for the algorithm to learn the **mapping** from inputs to outputs so it can predict the output for new, unseen data* **bold text**

Process of Supervised Learning

1. **Data Collection**: Gather a dataset containing input-output pairs.
2. **Data Preprocessing**: Clean the data, handle missing values, normalize features, and split the data into training and testing sets.
3. **Model Selection**: Choose an appropriate model based on the problem type (**Regression or Classification**) and dataset characteristics.
4. **Training**: Use the training data to let the model learn the mapping from inputs to outputs by minimizing the loss function.
5. **Evaluation**: Assess the model's performance on the testing set using appropriate metrics (e.g., **Accuracy, Precision, Recall for Classification tasks, or Mean Squared Error for Regression tasks**).
6. **Hyperparameter Tuning**: Adjust hyperparameters (like Learning Rate, Regularization Parameters) to improve model performance.
7. **Deployment**: Use the trained model to make predictions on new, unseen data.

Important Formulas

```
Image(filename='/content/drive/MyDrive/Screenshot (9).png')
```

1. **Accuracy:**

$$\text{Accuracy} = \frac{\text{Number of correct predictions}}{\text{Total number of predictions}}$$

```
Image(filename='/content/drive/MyDrive/Screenshot (10).png')
```

	Predicted Positive	Predicted Negative
Actual Positive	True Positive (TP)	False Negative (FN)
Actual Negative	False Positive (FP)	True Negative (TN)

1. Accuracy:

```
When the dataset is balanced and you want a general measure of model performance.
```

2. Precision:

```
When the cost of false positives is high.
```

3. Recall:

```
When the cost of false negatives is high.
```

4. F1 Score:

```
When you need a balance between precision and recall, particularly with imbalanced
```

5. Mean Squared Error (MSE):

```
When you want to penalize larger errors more significantly.
```

6. Root Mean Squared Error (RMSE):

```
When you need an error metric that is in the same units as the target variable.
```

7. Mean Absolute Error (MAE):

```
When you want an error metric that treats all errors equally.
```

8. R2 Score:

```
When you need to measure the proportion of variance explained by the model.
```

Classification or Regression

We'll use the Iris dataset, a classic dataset in machine learning, to classify iris flowers into three species based on the length and width of their sepals and petals

Import Libraries

```
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, classification_report, mean_squared_error, confu
import matplotlib.pyplot as plt
import joblib
from IPython.display import Image
import pandas as pd
import numpy as np
import seaborn as sns
```

Load the Dataset

Reading From File

```
path= '/content/drive/MyDrive/iris_data.csv'
data= pd.read_csv(path)
```

Data Processing

View Data

```
data.head()
```

	sepal_length	sepal_width	petal_length	petal_width	species
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: Generate code with data View recommended plots

```
data.tail()
```

	sepal_length	sepal_width	petal_length	petal_width	species
145	6.7	3.0	5.2	2.3	2
146	6.3	2.5	5.0	1.9	2
147	6.5	3.0	5.2	2.0	2
148	6.2	3.4	5.4	2.3	2
149	5.9	3.0	5.1	1.8	2

```
data.describe()
```

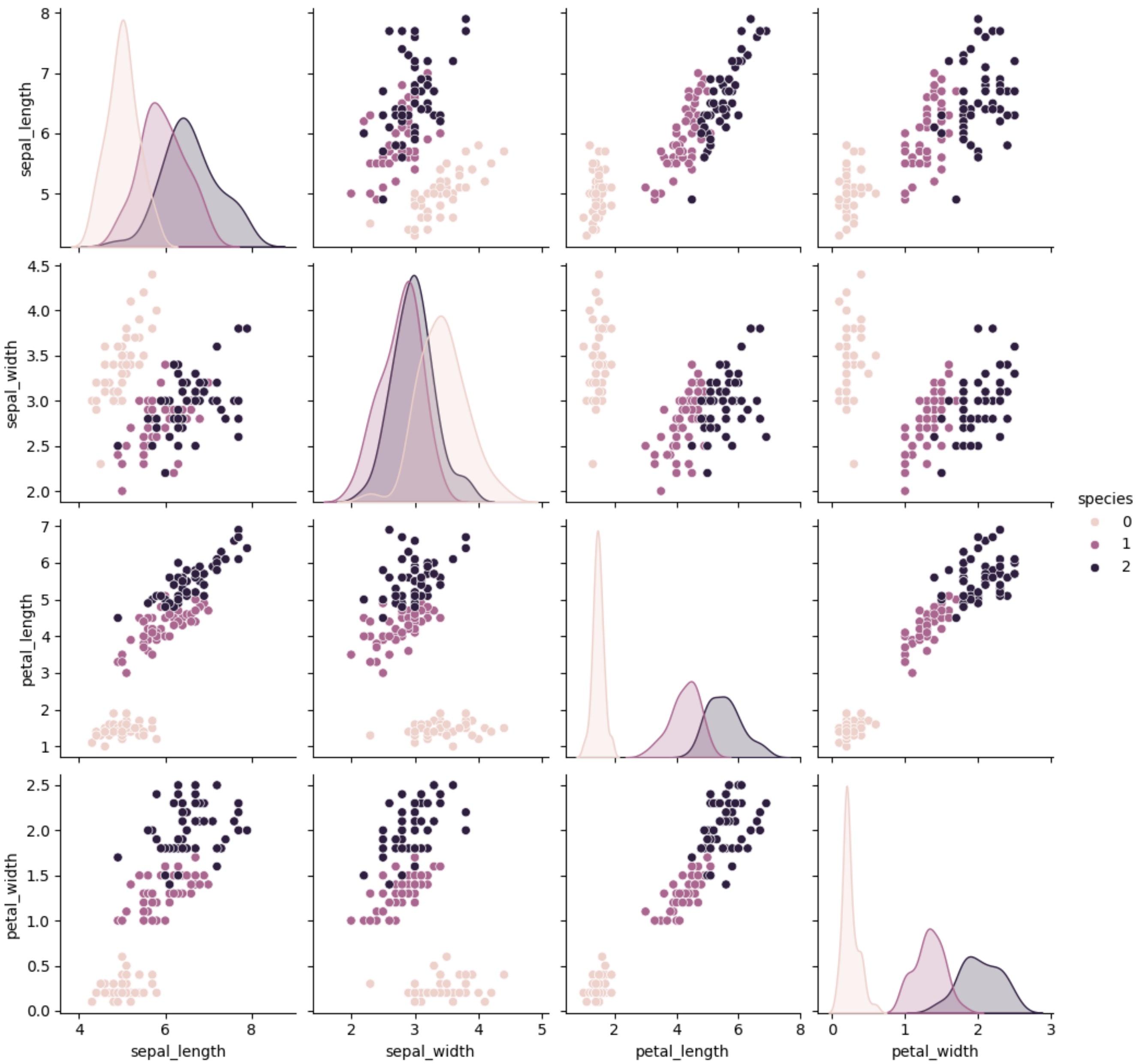
	sepal_length	sepal_width	petal_length	petal_width	species
count	150.000000	150.000000	150.000000	150.000000	150.000000
mean	5.843333	3.057333	3.758000	1.199333	1.000000
std	0.828066	0.435866	1.765298	0.762238	0.819232
min	4.300000	2.000000	1.000000	0.100000	0.000000
25%	5.100000	2.800000	1.600000	0.300000	0.000000
50%	5.800000	3.000000	4.350000	1.300000	1.000000
75%	6.400000	3.300000	5.100000	1.800000	2.000000
max	7.900000	4.400000	6.900000	2.500000	2.000000

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 150 entries, 0 to 149
Data columns (total 5 columns):
```

```
 #   Column        Non-Null Count  Dtype  
---  ------        --------------  -----  
 0   sepal_length  150 non-null    float64
 1   sepal_width   150 non-null    float64
 2   petal_length  150 non-null    float64
 3   petal_width   150 non-null    float64
 4   species       150 non-null    int64  
dtypes: float64(4), int64(1)
memory usage: 6.0 KB
```

```
sns.pairplot(data, hue="species")
plt.show()
```


sepal_length
sepal_width
petal_length
petal_width
species
0
1
2

Training Model

Selecting Features and Labels

```
X = data[["petal_length", "petal_width", "sepal_length", "sepal_width"]]
y = data["species"]
```

*Spliting Data in Two Sets {**Training Set and Testing Set**}*

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

Selecting a Model

```
model = RandomForestClassifier(n_estimators=100, max_depth=5, random_state=42)
```

Train on Data and Labels

```
model.fit(X_train, y_train)
```

```
RandomForestClassifier
RandomForestClassifier(max_depth=5, random_state=42)
```

Evaluation

Prediction

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

```
df_y_test=pd.DataFrame(y_test)
```

```
df = df_y_test  # your DataFrame
df = df.reset_index(drop=True)
col ='species'  # index of the column
```

Printing Actual and Predicted Classes

```
for i in y_pred:
  print("Predicted Class: ", y_pred[i], " and Actual Class: ", df.loc[(i,col)])
```

```
Predicted Class:  0  and Actual Class:  0
Predicted Class:  1  and Actual Class:  1
Predicted Class:  2  and Actual Class:  2
Predicted Class:  0  and Actual Class:  0
Predicted Class:  0  and Actual Class:  0
Predicted Class:  1  and Actual Class:  1
Predicted Class:  0  and Actual Class:  0
Predicted Class:  2  and Actual Class:  2
Predicted Class:  0  and Actual Class:  0
Predicted Class:  0  and Actual Class:  0
Predicted Class:  2  and Actual Class:  2
Predicted Class:  1  and Actual Class:  1
Predicted Class:  1  and Actual Class:  1
Predicted Class:  1  and Actual Class:  1
Predicted Class:  1  and Actual Class:  1
Predicted Class:  0  and Actual Class:  0
Predicted Class:  2  and Actual Class:  2
Predicted Class:  0  and Actual Class:  0
Predicted Class:  0  and Actual Class:  0
Predicted Class:  2  and Actual Class:  2
Predicted Class:  1  and Actual Class:  1
Predicted Class:  2  and Actual Class:  2
Predicted Class:  1  and Actual Class:  1
Predicted Class:  2  and Actual Class:  2
Predicted Class:  2  and Actual Class:  2
Predicted Class:  2  and Actual Class:  2
Predicted Class:  2  and Actual Class:  2
Predicted Class:  2  and Actual Class:  2
Predicted Class:  1  and Actual Class:  1
Predicted Class:  1  and Actual Class:  1
```

Accuracy

```
accuracy = accuracy_score(y_test, y_pred)

print("Accuracy:", accuracy*100,"%")
```

```
Accuracy: 100.0 %
```

Mean Squared Error

```
mse = mean_squared_error(y_test, y_pred)
print("Mean Squared Error:", mse)
```

```
Mean Squared Error: 0.0
```

R2 Score

```
r2 = r2_score(y_test, y_pred)
print("R2 Score:", mse)
```

```
R2 Score: 0.0
```

Report

```
print(classification_report(y_test, y_pred))
```

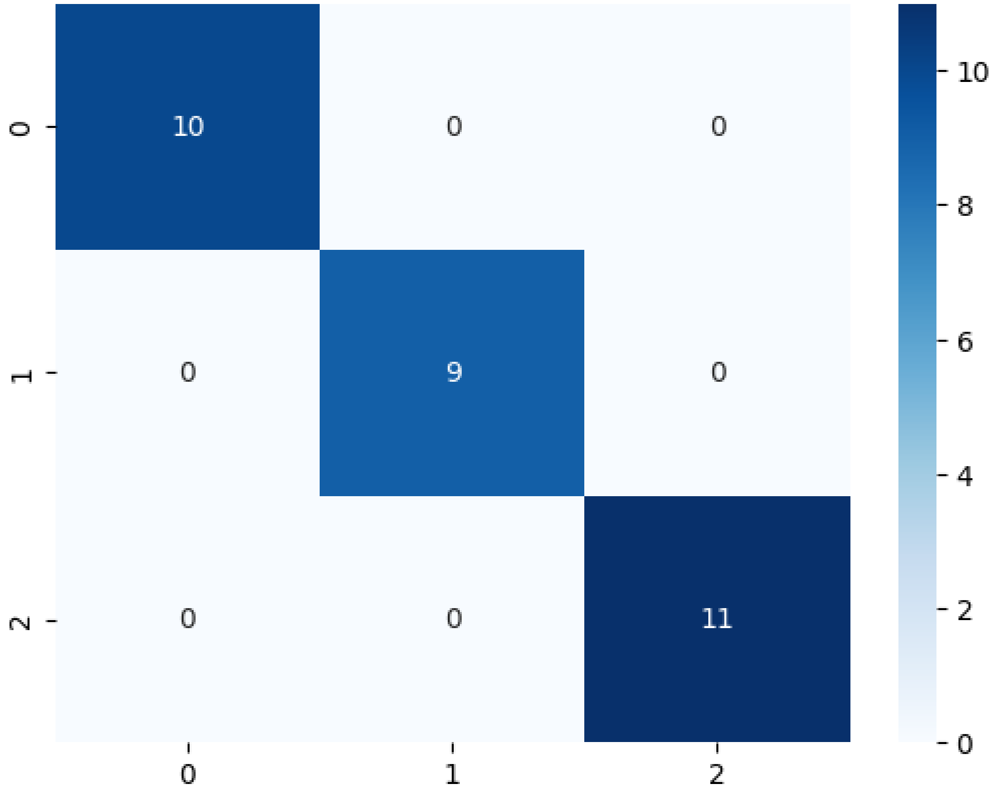
```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        10
           1       1.00      1.00      1.00         9
           2       1.00      1.00      1.00        11

    accuracy                           1.00        30
   macro avg       1.00      1.00      1.00        30
weighted avg       1.00      1.00      1.00        30
```

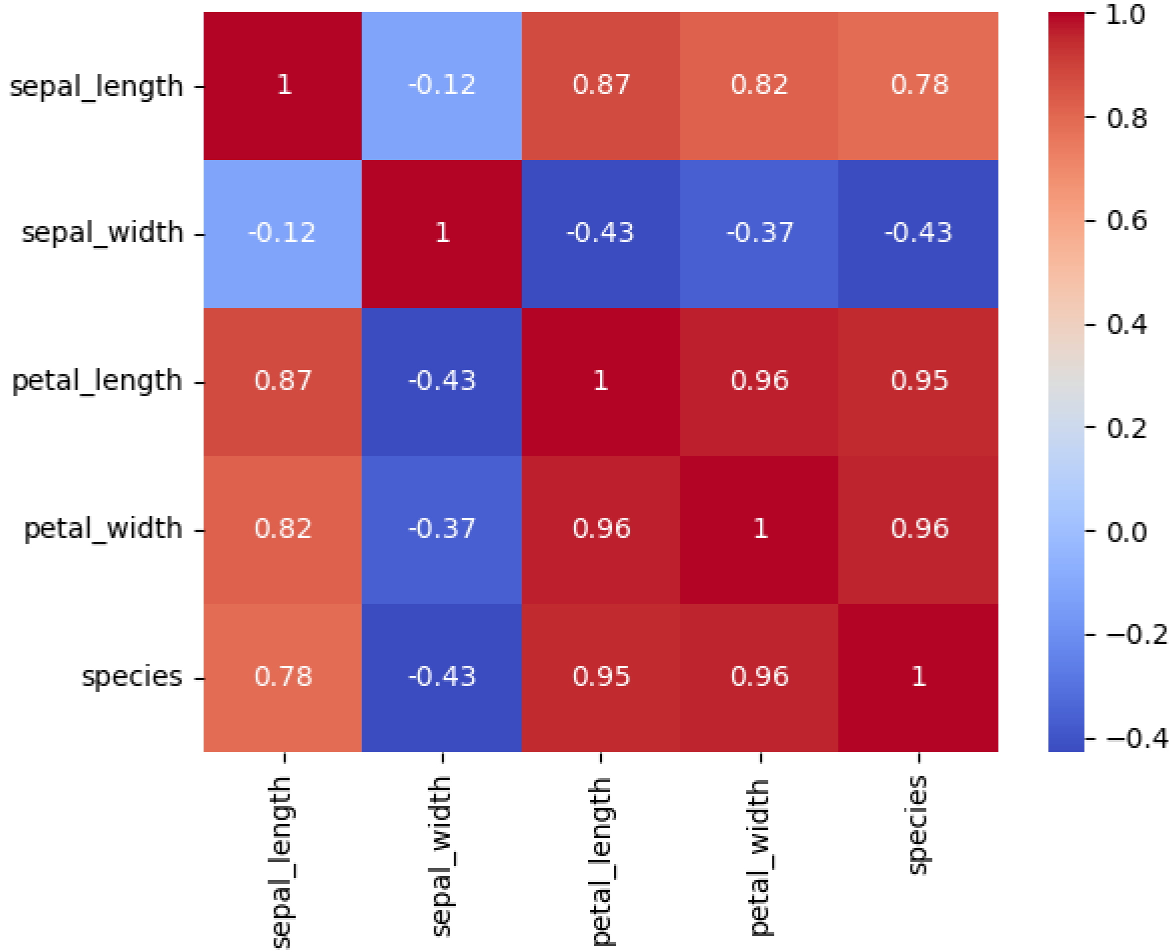
Plot the Confusion Matrix

```
cm = confusion_matrix(y_test, y_pred)
sns.heatmap(cm, annot=True, fmt="d", cmap="Blues")
plt.show()
```



Plot the Correlation Matrix

```
corr = data.corr()
sns.heatmap(corr, annot=True, cmap="coolwarm")
plt.show()
```



Hyperparameter Tuning

```
from sklearn.model_selection import GridSearchCV

# Define the hyperparameters and their values for tuning
param_grid = {
    'n_estimators': [50, 100, 200],
    'max_depth': [None, 10, 20, 30],
    'min_samples_split': [2, 5, 10]
}

# Initialize the model
model_iris = RandomForestClassifier(random_state=42)

# Initialize Grid Search
grid_search = GridSearchCV(estimator=model_iris, param_grid=param_grid, cv=5, n_jobs=-1, v

# Perform Grid Search
grid_search.fit(X_train, y_train)

# Get the best parameters and the best model
```

```
best_params = grid_search.best_params_
best_model = grid_search.best_estimator_

print(f"Best Parameters: {best_params}")

# Evaluate the tuned model
y_pred_iris_tuned = best_model.predict(X_test)
tuned_accuracy = accuracy_score(y_test, y_pred_iris_tuned)
```

```
Fitting 5 folds for each of 36 candidates, totalling 180 fits
Best Parameters: {'max_depth': None, 'min_samples_split': 5, 'n_estimators': 50}
Tuned Model Accuracy: 1.0
```

Deployment

Save Model

```
joblib.dump(model, "iris_model.joblib")
```

```
['iris_model.joblib']
```

Load A Safed Model

```
load_model= joblib.load('/content/drive/MyDrive/iris_model.joblib')
```

Mapper Function For Class

```
def mapp(cla):
  if cla == 0:
    clas= 'Iris Setosa'
    return clas
  elif cla ==1:
    clas= 'Iris Versicolor'
    return clas
  elif cla ==2:
    clas= 'Iris Virginica'
    return clas
  else:
    return None
```

Start coding or generate with AI.

Input and Prediction on User Data

```
def predict_class():
  petal_length = float(input("Enter petal length: "))
  petal_width = float(input("Enter petal width: "))
  sepal_length = float(input("Enter sepal length: "))
  sepal_width = float(input("Enter sepal width: "))

  new_data = np.array([[petal_length, petal_width, sepal_length, sepal_width]])

  predicted_class = ir.predict(new_data)[0]

  print(f"Predicted class: {mapp(predicted_class)}")
```

Main Loop

```
def main():
  load_model= joblib.load('/content/drive/MyDrive/iris_model.joblib')
  ir=load_model
  a=3
  while a>=0:
    predict_class()
    a=a-1

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

```
Enter petal length: 5
Enter petal width: 2
Enter sepal length: 5
Enter sepal width: 2
Predicted class: Iris Virginica
Enter petal length: 7
Enter petal width: 1
Enter sepal length: 6
Enter sepal width: 2
Predicted class: Iris Virginica
Enter petal length: 5
Enter petal width: 5
Enter sepal length: 5
Enter sepal width: 5
Predicted class: Iris Virginica
Enter petal length: 0
Enter petal width: 0
Enter sepal length: 0
Enter sepal width: 0
Predicted class: Iris Setosa
```

UnSupervised Learning

Unsupervised learning is a type of machine learning where the algorithm is trained on data that has not been labeled, classified, or categorized. Instead, the algorithm tries to learn the patterns and structure from the input data without explicit instructions on what to predict. The goal of unsupervised learning is to find hidden patterns or intrinsic structures in the data.

```
#!pip install scikit-learn==1.1.3
```

```
import numpy as np
import pandas as pd
from sklearn.datasets import load_boston
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Conv1D, Flatten, Dropout
from tensorflow.keras.optimizers import Adam


# Load the dataset
boston = load_boston()
X = boston.data
y = boston.target


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


# Standardize the features
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)


# Reshape the data to fit into the CNN
X_train = X_train.reshape((X_train.shape[0], X_train.shape[1], 1))
X_test = X_test.reshape((X_test.shape[0], X_test.shape[1], 1))


# Build the CNN model
model = Sequential()

model.add(Conv1D(filters=64, kernel_size=2, activation='relu', input_shape=(X_train.shape[
model.add(Dropout(0.5))

model.add(Conv1D(filters=32, kernel_size=2, activation='relu'))
```

```
model.add(Dropout(0.5))

model.add(Flatten())

model.add(Dense(64, activation='relu'))
model.add(Dropout(0.5))



# prompt: loss function cross intopy

model.compile(loss='mse', optimizer=Adam(learning_rate=0.01))



# Train the model
model.fit(X_train, y_train, epochs=100, batch_size=32, validation_split=0.2)
```

```
Epoch 58/100
11/11 [==============================] - 0s 8ms/step - loss: 44.3372 - val_loss: 25.2
Epoch 59/100
11/11 [==============================] - 0s 9ms/step - loss: 50.8519 - val_loss: 26.4
Epoch 60/100
11/11 [==============================] - 0s 8ms/step - loss: 44.7474 - val_loss: 33.0
Epoch 61/100
11/11 [==============================] - 0s 7ms/step - loss: 43.8334 - val_loss: 17.8
Epoch 62/100
11/11 [==============================] - 0s 9ms/step - loss: 42.3233 - val_loss: 20.0
Epoch 63/100
11/11 [==============================] - 0s 10ms/step - loss: 43.7297 - val_loss: 18.
Epoch 64/100
11/11 [==============================] - 0s 8ms/step - loss: 36.0936 - val_loss: 21.3
Epoch 65/100
```