Experiment no 1

The Goodness of Fit test is a statistical analysis used to determine if a sample of data matches a specific distribution. In simple terms, it checks how well the observed data aligns with a hypothesized probability distribution, like the normal distribution, binomial distribution, or others.

There are several types of Goodness of Fit tests, but two common ones are:

- Chi-square Goodness of Fit Test: This test is commonly used for categorical data and compares observed frequencies to expected frequencies.
- Kolmogorov-Smirnov (K-S) Test: This test is used for continuous data and measures the maximum difference between the cumulative distribution of the observed data and the hypothesized distribution.

Chi-Square Goodness of Fit Test

- The Chi-square goodness of fit test is a statistical hypothesis test used to determine whether a variable is likely to come from a specified distribution or not.
- It is often used to evaluate whether sample data is representative of the full population.
- Chi-Square goodness of fit test is a non-parametric test that is used to find out how the observed value of a given phenomena is significantly different from the expected value.
- In Chi-Square goodness of fit test, the term goodness of fit is used to compare the observed sample distribution with the expected probability distribution.
- Chi-Square goodness of fit test determines how well theoretical distribution (such as normal, binomial, or Poisson) fits the empirical distribution.
- In Chi-Square goodness of fit test, sample data is divided into intervals. Then the numbers of points that fall into the interval are compared, with the expected numbers of points in each interval.

Hypothesis

- Null Hypothesis: In Chi-Square goodness of fit test, the null hypothesis assumes that *there is no significant difference between the observed and the expected value*.
- Alternate Hypothesis: In Chi-Square goodness of fit test, the alternative hypothesis assumes that *there is a significant difference between the observed and the expected value*.

The test statistic for the chi-square (X2) goodness of fit test is Pearson's chi-square:

$$\chi^2 = \sum \frac{(O - E)^2}{E}$$

χ^2 = the test statistic $\sum$ = the sum of

χ^2 = the test statistic $\sum$ the sum of

O = Observed frequencies E = Expected frequencies

Formula Explanation

Hypothesis Testing

- If the calculated value of Chi-Square Goodness of Fit is greater than the table value, we reject the null hypothesis and conclude that there is a significant difference between the observed and the expected frequency.
- If the calculated value of Chi-Square Goodness of Fit is less than the table value, we accept the null hypothesis and conclude that there is no significant difference between the observed and the expected frequency.

Hardcode function of chi square

```
def chi_square_hardcore(observed_data , expected_data):
  chi_square = 0;
  n = len(observed_data)
  for i in range(n):
    chi_square += ((observed_data[i]-expected_data[i])**2)/expected_data[i]

  return chi_square
```

Using builtin libraries

```
from scipy import stats
def chi_square_library(observed_data , expected_data):
  chi_square_test_statistic, p_value = stats.chisquare(observed_data , expected_data)

  return chi_square_test_statistic , p_value
```

```
observed_data = [8, 6, 10, 7, 8, 11, 9]
expected_data = [9, 8, 11, 8, 10, 7, 6]

chi1 = chi_square_hardcore(observed_data , expected_data)
chi2, p_value = chi_square_library(observed_data , expected_data)

print('Hardcoded value :', chi1)
print('Library value : ' ,chi2)
print('P value : ', p_value)
```

```
Hardcoded value : 5.0127344877344875
Library value :  5.0127344877344875
P value :  0.542180861413329
```

```
if(chi1< p_value):
```

```
if(chi1< p_value):
  print('We accpet the null hypothesis, No significant difference between observed and expected')
else:
  print('We reject the null hypotheses , There is difference between observed and expected')
```

```
We reject the null hypotheses , There is difference between observed and expected
```

```
if(chi2< p_value):
  print('We accpet the null hypothesis, No significant difference between observed and expected')
else:
  print('We reject the null hypotheses , There is difference between observed and expected')
```

```
We reject the null hypotheses , There is difference between observed and expected
```

KS Test

- KS stands for "Kolmogorov-Smirnov"
- This test is used in situations where a comparison has to be made between an observed sample distribution and theoretical distribution.
- The Kolmogorov-Smirnov Goodness of Fit Test (K-S test) compares your data with a known distribution and lets you know if they have the same distribution.
- As per Wikipedia, "The Kolmogorov–Smirnov statistic quantifies a distance between the empirical distribution function of the sample and the cumulative distribution function of the reference distribution, or between the empirical distribution functions of two samples."
- In probability theory and statistics, the cumulative distribution function (CDF) of a real-valued random variable , or just distribution function of , evaluated at , is the probability that will take a value less than or equal to.
- An empirical cumulative distribution function (also called the empirical distribution function, ECDF, or just EDF) and a cumulative distribution function are basically the same thing: they are both probability models for data. However, while a CDF is a hypothetical model of a distribution, the ECDF models empirical (i.e. observed) data.

KS One Sample Test: This test is used as a test of goodness of fit and is ideal when the size of the sample is small. It compares the cumulative distribution function for a variable with a specified distribution. ***The null hypothesis assumes no difference between the observed and theoretical distribution*** and the value of test statistic 'D' is calculated as:

Formula

D = Maximum|Fo(X)−Fr(X)| Where,

- Fo(X) = Observed cumulative frequency distribution of a random sample of n observations.
- Fo(X)=kn = (No.of observations ≤ X)/(Total no.of observations).
- Fr(X) = The theoretical frequency distribution.

The critical value of D is found from the K-S table values for one sample test.

Acceptance Criteria: If calculated value is less than critical value accept null hypothesis.

Rejection Criteria: If calculated value is greater than table value reject null hypothesis.

KS Two Sample Test: When instead of one, there are two independent samples then K-S two sample test can be used to test the agreement between two cumulative distributions. The null hypothesis states that there is no difference between the two distributions. The D-statistic is calculated in the same manner as the K-S One Sample Test.

Formula $D = \text{Maximum}|Fn1(X) - Fn2(X)|$ Where,

- n1 = Observations from first sample.
- n2 = Observations from second sample.

It has been seen that when the cumulative distributions show large maximum deviation |D| it is indicating towards a difference between the two sample distributions.

The critical value of D for samples where n1=n2 and is ≤ 40, the K-S table for two sample case is used. When n1 and/or n2 > 40 then the K-S table for large samples of two sample test should be used. The null hypothesis is accepted if the calculated value is less than the table value and vice-versa.

General Steps: The general steps to run the test are:

1. Create an EDF for your sample data (see Empirical Distribution Function for steps),
2. Specify a parent distribution (i.e. one that you want to compare your EDF to)
3. Graph the two distributions together.
4. Measure the greatest vertical distance between the two graphs.
5. Calculate the test statistic.
6. Find the critical value in the KS table.
7. Compare to the critical value.

Advantages and Disadvantages

Advantages include:

- The test is distribution free. That means you don't have to know the underlying population distribution for your data before running this test.
- The D statistic (not to be confused with Cohen's D) used for the test is easy to calculate.
- It can be used as a goodness of fit test following regression analysis.
- There are no restrictions on sample size; Small samples are acceptable.
- Tables are readily available.

Although the K-S test has many advantages, it also has a few limitations:

- In order for the test to work, you must specify the location, scale, and shape parameters. If these parameters are estimated from the data, it invalidates the test. If you don't know these parameters, you may want to run a less formal test (like the one outlined in the empirical distribution function article).
- It generally can't be used for discrete distributions, especially if you are using software (most software packages don't have the necessary extensions for discrete K-S Test and the manual calculations are convoluted).

- Sensitivity is higher at the center of the distribution and lower at the tails.

```
def ks_hardcode(observed_data, expected_data):
  fo = []
  ft = []
  d = []
  n = len(observed_data)
  s = sum(observed_data)
  fo.append(observed_data[0]/s)
  ft.append(expected_data[0]/s)
  for i in range(1, n):
    fo.append(observed_data[i]/s + fo[i-1])
    ft.append(expected_data[i]/s + ft[i-1])

  for i in range(n):
    if fo[i] > ft[i]:
      d.append(fo[i] - ft[i])
    else:
      d.append(ft[i] - fo[i])

  d0 = 1.36/(s**0.5)
  return max(d), d0
```

```
from scipy import stats
import matplotlib.pyplot as plt

def ks_library(observed_data, distribution):
  ks2, p = stats.kstest(observed_data, distribution)
  return ks2, p
```

```
observed_data = [5, 9, 11, 16, 19]
expected_data = [12, 12, 12, 12, 12]

ks1, d0 = ks_hardcode(observed_data, expected_data)
print(ks1)
print(d0)

if (ks1 > d0):
  print('Reject the Null Hypothesis')
else:
  print('Accept the Null Hypothesis')

ks2, p = ks_library(observed_data, 'uniform')
print(ks2)
print(p)

if (ks2 > p):
  print('Reject the Null Hypothesis')
else:
  print('Accept the Null Hypothesis')
```

```
0.18333333333333346
0.17557524502806957
Reject the Null Hypothesis
```

```
1.0
0.0
Reject the Null Hypothesis
```

Experiment no. 2- Ensemble learning

Dataset link - https://www.kaggle.com/datasets/spscientist/telecom-data?resource=download

```
import pandas as pd
import time
import numpy as np
df=pd.read_csv('/content/telecom_churn.csv')
df

x = df[['international plan', 'voice mail plan', 'total day charge', 'total eve charge', 'total night
x['international plan'] = x['international plan'].map({'yes': 1, 'no': 0})
x['voice mail plan'] = x['voice mail plan'].map({'yes': 1, 'no': 0})
y = df['churn']
```

```
<ipython-input-32-611c69a08fbc>:8: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/ind
  x['international plan'] = x['international plan'].map({'yes': 1, 'no': 0})
<ipython-input-32-611c69a08fbc>:9: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/ind
  x['voice mail plan'] = x['voice mail plan'].map({'yes': 1, 'no': 0})
```

```
#Central Model using SVM
from sklearn.svm import SVC
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score

# Split the dataset into train and test sets
X_train, X_test, y_train, y_test = train_test_split(x, y, test_size=0.2, random_state=42)

# Train the SVM model
central_model = SVC(random_state=42)
central_model.fit(X_train, y_train)

# Predict and evaluate the centralized model
y_pred_central = central_model.predict(X_test)
central_model_accuracy = accuracy_score(y_test, y_pred_central)
print(f'Central Model Accuracy: {central_model_accuracy}')

#Dividing Dataset
```

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier

X_part1, X_temp, y_part1, y_temp = train_test_split(X_train, y_train, test_size=0.67, random_state=42)
X_part2, X_part3, y_part2, y_part3 = train_test_split(X_temp, y_temp, test_size=0.5, random_state=42)

model1 = DecisionTreeClassifier(random_state=42)
model1.fit(X_part1, y_part1)

model2 = RandomForestClassifier(random_state=42)
model2.fit(X_part2, y_part2)

model3 = LogisticRegression(random_state=42, max_iter=1000)
model3.fit(X_part3, y_part3)
```

```
Central Model Accuracy: 0.8665667166416792
```

LogisticRegression

```
LogisticRegression(max_iter=1000, random_state=42)
```

```
#Ensembled Dataset
from sklearn.ensemble import VotingClassifier

# Voting Classifier
ensemble_model = VotingClassifier(estimators=[
    ('dt', model1),
    ('rf', model2),
    ('lr', model3)
], voting='hard')

ensemble_model.fit(X_train, y_train)

y_pred_ensemble = ensemble_model.predict(X_test)
ensemble_model_accuracy = accuracy_score(y_test, y_pred_ensemble)
print(f'Ensemble Model Accuracy: {ensemble_model_accuracy}')

#Compare
print(f'Central Model Accuracy: {central_model_accuracy}')
print(f'Ensemble Model Accuracy: {ensemble_model_accuracy}')
```

```
Ensemble Model Accuracy: 0.9145427286356822
Central Model Accuracy: 0.8665667166416792
Ensemble Model Accuracy: 0.9145427286356822
```

Confusion matrix

```
from sklearn.metrics import confusion_matrix

# Confusion matrix for the centralized model
conf_matrix_central = confusion_matrix(y_test, y_pred_central)
```

```
print("Central Model Confusion Matrix:")
print(conf_matrix_central)

# Confusion matrix for the ensemble model
conf_matrix_ensemble = confusion_matrix(y_test, y_pred_ensemble)
print("Ensemble Model Confusion Matrix:")
print(conf_matrix_ensemble)
```

```
Central Model Confusion Matrix:
[[566   0]
 [ 89  12]]
Ensemble Model Confusion Matrix:
[[560   6]
 [ 51  50]]
```

Experiment no. 3 - Incremental learning

```
import pandas as pd
from sklearn.linear_model import SGDClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, confusion_matrix

# Load dataset
df = pd.read_csv('/content/telecom_churn.csv')

# Feature selection and encoding
x = df[['international plan', 'voice mail plan', 'total day charge', 'total eve charge', 'total night
x['international plan'] = x['international plan'].map({'yes': 1, 'no': 0})
x['voice mail plan'] = x['voice mail plan'].map({'yes': 1, 'no': 0})
y = df['churn']

# Split the dataset 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)
```

```
<ipython-input-38-ade6bd758a40>:11: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/ind
  x['international plan'] = x['international plan'].map({'yes': 1, 'no': 0})
<ipython-input-38-ade6bd758a40>:12: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/ind
  x['voice mail plan'] = x['voice mail plan'].map({'yes': 1, 'no': 0})
```

```
# Initialize the SGDClassifier (supports incremental learning)
incremental_model = SGDClassifier(random_state=42)
```

```
# Split the training data into smaller chunks for incremental learning
batch_size = 100  # Define a batch size for incremental updates
num_batches = len(X_train) // batch_size

# Train incrementally on each batch
for i in range(0, len(X_train), batch_size):
    end = i + batch_size if (i + batch_size) < len(X_train) else len(X_train)
    X_batch = X_train[i:end]
    y_batch = y_train[i:end]
    incremental_model.partial_fit(X_batch, y_batch, classes=[0, 1])
```

```
# Evaluate on test data
y_pred_incremental = incremental_model.predict(X_test)
incremental_model_accuracy = accuracy_score(y_test, y_pred_incremental)
print(f'Incremental Model Accuracy: {incremental_model_accuracy}')
```

```
Incremental Model Accuracy: 0.848575712143928
```

```
# Confusion Matrix
conf_matrix_incremental = confusion_matrix(y_test, y_pred_incremental)
print("Incremental Model Confusion Matrix:")
print(conf_matrix_incremental)
```

```
Incremental Model Confusion Matrix:
[[566   0]
 [101   0]]
```

Experiment Neural Network for predictions

```
import tensorflow as tf
from tensorflow import keras
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler

# Load the Iris dataset
iris = load_iris()
X = iris.data
y = iris.target

# 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)

# Scale the data
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

```
# Convert labels to one-hot encoding
y_train = keras.utils.to_categorical(y_train, num_classes=3)
y_test = keras.utils.to_categorical(y_test, num_classes=3)

# Define the neural network model
model = keras.Sequential([
    keras.layers.Dense(10, activation='relu', input_shape=(4,)),
    keras.layers.Dense(3, activation='softmax')
])

# Compile the model
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])

# Train the model
model.fit(X_train, y_train, epochs=50, batch_size=10, verbose=1)

# Evaluate the model
loss, accuracy = model.evaluate(X_test, y_test, verbose=0)
print('Test loss:', loss)
print('Test accuracy:', accuracy)

# Make predictions
y_pred = model.predict(X_test)
y_pred_classes = y_pred.argmax(axis=1)
y_true_classes = y_test.argmax(axis=1)

# Analyze performance (e.g., confusion matrix, classification report)
from sklearn.metrics import classification_report, confusion_matrix
print(classification_report(y_true_classes, y_pred_classes))
print(confusion_matrix(y_true_classes, y_pred_classes))
```

```
/usr/local/lib/python3.10/dist-packages/keras/src/layers/core/dense.py:87: UserWarning: Do not pas
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
Epoch 1/50
12/12 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.4914 - loss: 0.9309
Epoch 2/50
12/12 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.6188 - loss: 0.8714
Epoch 3/50
12/12 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.5474 - loss: 0.8829
Epoch 4/50
12/12 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.5782 - loss: 0.8127
Epoch 5/50
12/12 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.7057 - loss: 0.7146
Epoch 6/50
12/12 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.6838 - loss: 0.7483
Epoch 7/50
12/12 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.6816 - loss: 0.7334
Epoch 8/50
12/12 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.7240 - loss: 0.6373
Epoch 9/50
12/12 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.6885 - loss: 0.6451
Epoch 10/50
12/12 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.7222 - loss: 0.6024
Epoch 11/50
12/12 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.7636 - loss: 0.6196
Epoch 12/50
12/12 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.8244 - loss: 0.5638
```

```
Epoch 13/50
12/12 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.7160 - loss: 0.5861
Epoch 14/50
12/12 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.8352 - loss: 0.5303
Epoch 15/50
12/12 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.7838 - loss: 0.5627
Epoch 16/50
12/12 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.8405 - loss: 0.4835
Epoch 17/50
12/12 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.8004 - loss: 0.5198
Epoch 18/50
12/12 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.7512 - loss: 0.5772
Epoch 19/50
12/12 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.8427 - loss: 0.4689
Epoch 20/50
12/12 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.8036 - loss: 0.5110
Epoch 21/50
12/12 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.8546 - loss: 0.4481
Epoch 22/50
12/12 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.8346 - loss: 0.4467
Epoch 23/50
12/12 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.7972 - loss: 0.4764
Epoch 24/50
12/12 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.7976 - loss: 0.5110
Epoch 25/50
12/12 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.8581 - loss: 0.4025
Epoch 26/50
12/12 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.8013 - loss: 0.4948
Epoch 27/50
12/12 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.8474 - loss: 0.4197
Epoch 28/50
12/12 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.8064 - loss: 0.4330
```

Neural network for classification

```
import tensorflow as tf
from tensorflow import keras
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import classification_report, confusion_matrix

# Load the Iris dataset
iris = load_iris()
X = iris.data
y = iris.target

# 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)

# Scale the data
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# Convert labels to one-hot encoding
y_train = keras.utils.to_categorical(y_train, num_classes=3)
```

```
y_test = keras.utils.to_categorical(y_test, num_classes=3)

# Define the neural network model for classification
model = keras.Sequential([
    keras.layers.Dense(10, activation='relu', input_shape=(4,)),
    keras.layers.Dense(3, activation='softmax')  # 3 output units for 3 classes
])

# Compile the model
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])

# Train the model
model.fit(X_train, y_train, epochs=50, batch_size=10, verbose=1)

# Evaluate the model
loss, accuracy = model.evaluate(X_test, y_test, verbose=0)
print('Test loss:', loss)
print('Test accuracy:', accuracy)

# Make predictions
y_pred = model.predict(X_test)
y_pred_classes = y_pred.argmax(axis=1)  # Predicted class labels
y_true_classes = y_test.argmax(axis=1)  # True class labels

# Analyze performance using confusion matrix and classification report
print("Classification Report:\n", classification_report(y_true_classes, y_pred_classes))
print("Confusion Matrix:\n", confusion_matrix(y_true_classes, y_pred_classes))
```

```
/usr/local/lib/python3.10/dist-packages/keras/src/layers/core/dense.py:87: UserWarning: Do not pas
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
Epoch 1/50
12/12 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.4721 - loss: 1.0355
Epoch 2/50
12/12 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.4959 - loss: 0.9800
Epoch 3/50
12/12 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.5237 - loss: 0.9557
Epoch 4/50
12/12 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.5919 - loss: 0.8798
Epoch 5/50
12/12 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.6475 - loss: 0.8391
Epoch 6/50
12/12 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.6756 - loss: 0.8006
Epoch 7/50
12/12 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.6896 - loss: 0.7370
Epoch 8/50
12/12 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.7128 - loss: 0.7156
Epoch 9/50
12/12 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.7801 - loss: 0.6840
Epoch 10/50
12/12 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.8097 - loss: 0.6305
Epoch 11/50
12/12 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.7877 - loss: 0.6250
Epoch 12/50
12/12 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.7268 - loss: 0.6269
Epoch 13/50
12/12 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.8134 - loss: 0.5728
Epoch 14/50
```

```
12/12 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.8138 - loss: 0.5293
Epoch 15/50
12/12 ━━━━━━━━━━━━━━━━━━━━ 0s 13ms/step - accuracy: 0.8755 - loss: 0.4955
Epoch 16/50
12/12 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.8115 - loss: 0.5262
Epoch 17/50
12/12 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.7947 - loss: 0.5642
Epoch 18/50
12/12 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.8136 - loss: 0.4816
Epoch 19/50
12/12 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 0.8514 - loss: 0.4776
Epoch 20/50
12/12 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.8814 - loss: 0.4004
Epoch 21/50
12/12 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.8646 - loss: 0.4482
Epoch 22/50
12/12 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.8469 - loss: 0.4201
Epoch 23/50
12/12 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.8737 - loss: 0.3932
Epoch 24/50
12/12 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.8640 - loss: 0.3984
Epoch 25/50
12/12 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.8786 - loss: 0.3608
Epoch 26/50
12/12 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.8405 - loss: 0.3914
Epoch 27/50
12/12 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.8934 - loss: 0.3545
Epoch 28/50
12/12 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.8550 - loss: 0.3650
```

Start coding or generate with AI.