**University of Central Missouri**

**Department of Computer Science & Cybersecurity**

**CS4700 Artificial Intelligence**

**Spring 2025**

**Bonus Assignment**

**Student name:**

**Evan Finger**

**Submission Requirements:**

* Once finished your assignment push your source code to your repo (GitHub) and explain the work through the ReadMe file properly. Make sure you add your student info in the ReadMe file.
* Submit your GitHub link on the BB.
* Comment your code appropriately ***IMPORTANT.***

**Exercise 1: House Price Prediction with Linear Regression**

Fill in the blanks (denoted by \_\_\_\_\_) to complete the following code that loads the California Housing dataset, trains a linear regression model, and evaluates its performance

*import numpy as np*

*from sklearn.datasets import fetch\_california\_housing*

*from sklearn.model\_selection import train\_test\_split*

*from sklearn.linear\_model import LinearRegression*

*from sklearn.metrics import mean\_squared\_error, r2\_score*

*# Load dataset*

*housing = fetch\_california\_housing()*

*X = housing.data*

*y = housing.target*

*# Split data into training and testing sets (80/20 split)*

*X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=\_\_\_\_\_, random\_state=42)*

*# Build and train the model*

*model = LinearRegression()*

*model.\_\_\_\_\_(X\_train, y\_train)*

*# Make predictions on the test set*

*y\_pred = model.\_\_\_\_\_(X\_test)*

*# Evaluate the model*

*mse = mean\_squared\_error(y\_test, y\_pred)*

*r2 = r2\_score(y\_test, y\_pred)*

*print("Mean Squared Error:", mse)*

*print("R^2 Score:", r2)*

**Answer:**

**Fit, and predict**

Mean Squared Error: 0.5558915986952425

R^2 Score: 0.5757877060324521

**Exercise 2: Unsupervised Learning with K-Means Clustering**

Fill in the blanks to complete the K-means clustering example that generates synthetic data, clusters it, and visualizes the clusters.

*import numpy as np*

*import matplotlib.pyplot as plt*

*from sklearn.cluster import KMeans*

*from sklearn.datasets import make\_blobs*

*# Generate synthetic data for clustering*

*X, y = make\_blobs(n\_samples=300, centers=4, random\_state=42)*

*# Initialize K-Means with 4 clusters*

*kmeans = KMeans(n\_clusters=\_\_\_\_\_, random\_state=42)*

*kmeans.\_\_\_\_\_(X)*

*# Get cluster labels and centroids*

*labels = kmeans.labels\_*

*centroids = kmeans.cluster\_centers\_*

*plt.scatter(X[:, 0], X[:, 1], c=labels, cmap='viridis', marker='o')*

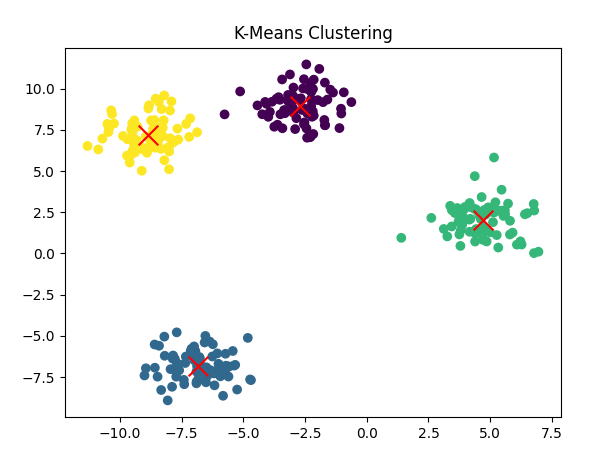
*plt.scatter(centroids[:, 0], centroids[:, 1], marker='x', color='red', s=200)*

*plt.title("K-Means Clustering")*

*plt.show()*

**Answer:**

**4, and fit**

****

**Exercise 3: Simple Neural Network for Regression using Keras**

Fill in the blanks to create a simple neural network model for regression. This model uses two hidden layers and a linear output layer.

*import numpy as np*

*from tensorflow.keras.models import Sequential*

*from tensorflow.keras.layers import Dense*

*from tensorflow.keras.optimizers import Adam*

*# Assume X\_train is already defined and has shape (num\_samples, num\_features)*

*num\_features = X\_train.shape[1]*

*# Create a simple neural network model for regression*

*model = Sequential([*

*Dense(64, activation='relu', input\_shape=(\_\_\_\_\_,)), # Input layer*

*Dense(64, activation='relu'),*

*Dense(1, activation='linear') # Output layer for regression*

*])*

*model.compile(optimizer=Adam(learning\_rate=\_\_\_\_\_), loss='mse')*

*# Train the model*

*model.fit(X\_train, y\_train, epochs=50, batch\_size=32, verbose=1)*

**Answer:**

**Num\_features, and 0.001**

On Epoch 50/50

**516/516** ━━━━━━━━━━━━━━━━━━━━ **1s** 2ms/step - loss: 1.9327

**Exercise 4: Updating the State in the GridWorld Environment**

Fill in the blank in the step method to update the row when the action is “up” (action 0).

*def step(self, action):*

*# Actions: 0=up, 1=right, 2=down, 3=left*

*r, c = self.state*

*if action == 0: # up*

*r = max(r - \_\_\_\_\_, 0)*

*# (Other actions omitted for brevity)*

*self.state = (r, c)*

*# Terminal state check*

*if self.state in self.terminal:*

*return self.state, 1, True*

*else:*

*return self.state, 0, False*

**Answer:**

**1**