TASK 2: CAR PRICE PREDICTION WITH ML

- **Program:**

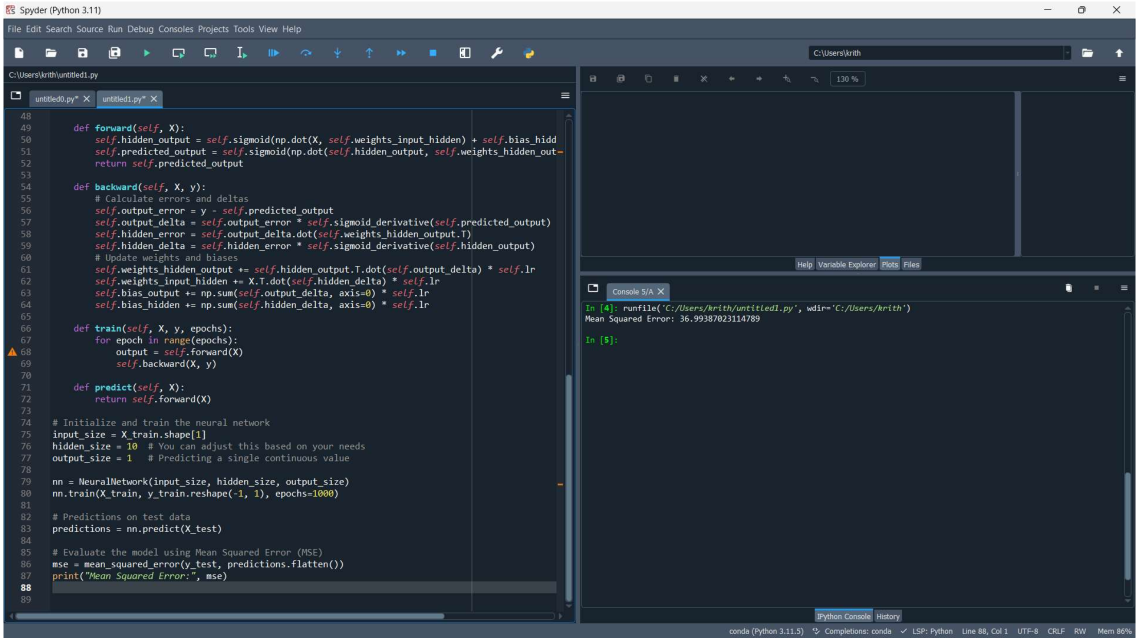
```
import numpy as np

import pandas as pd

from sklearn.model_selection import train_test_split

from sklearn.preprocessing import StandardScaler

from sklearn.metrics import mean_squared_error

from sklearn.metrics import accuracy_score

# Load the dataset

data = pd.read_csv('C:\\Users\\krith\\Downloads\\Aaron\\cardata.csv')  # Replace with the correct file path


# Preprocess the data

# Drop the Car_Name column as it might not contribute significantly to predicting the price

data = data.drop('Car_Name', axis=1)


# One-hot encode categorical variables

categorical_cols = ['Fuel_Type', 'Selling_type', 'Transmission']

data = pd.get_dummies(data, columns=categorical_cols, drop_first=True)


# Separate features and target variable

X = data.drop('Selling_Price', axis=1).values

y = data['Selling_Price'].values


# Standardize the features

scaler = StandardScaler()
```

```
X = scaler.fit_transform(X)


# Split 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)


# Define the Neural Network class
class NeuralNetwork:
    def __init__(self, input_size, hidden_size, output_size):
        self.input_size = input_size
        self.hidden_size = hidden_size
        self.output_size = output_size
        # Initialize weights and biases
        self.weights_input_hidden = np.random.randn(self.input_size,
self.hidden_size)
        self.weights_hidden_output = np.random.randn(self.hidden_size,
self.output_size)
        self.bias_hidden = np.zeros((1, self.hidden_size))
        self.bias_output = np.zeros((1, self.output_size))
        # Learning rate
        self.lr = 0.01


    def sigmoid(self, x):
        return 1 / (1 + np.exp(-x))


    def sigmoid_derivative(self, x):
        return x * (1 - x)
```

```
    def forward(self, X):
        self.hidden_output = self.sigmoid(np.dot(X, self.weights_input_hidden) + self.bias_hidden)
        self.predicted_output = self.sigmoid(np.dot(self.hidden_output, self.weights_hidden_output) + self.bias_output)
        return self.predicted_output

    def backward(self, X, y):
        # Calculate errors and deltas
        self.output_error = y - self.predicted_output
        self.output_delta = self.output_error * self.sigmoid_derivative(self.predicted_output)
        self.hidden_error = self.output_delta.dot(self.weights_hidden_output.T)
        self.hidden_delta = self.hidden_error * self.sigmoid_derivative(self.hidden_output)
        # Update weights and biases
        self.weights_hidden_output += self.hidden_output.T.dot(self.output_delta) * self.lr
        self.weights_input_hidden += X.T.dot(self.hidden_delta) * self.lr
        self.bias_output += np.sum(self.output_delta, axis=0) * self.lr
        self.bias_hidden += np.sum(self.hidden_delta, axis=0) * self.lr

    def train(self, X, y, epochs):
        for epoch in range(epochs):
            output = self.forward(X)
            self.backward(X, y)
```

```python
    def predict(self, X):
        return self.forward(X)

# Initialize and train the neural network
input_size = X_train.shape[1]
hidden_size = 10  # You can adjust this based on your needs
output_size = 1   # Predicting a single continuous value

nn = NeuralNetwork(input_size, hidden_size, output_size)
nn.train(X_train, y_train.reshape(-1, 1), epochs=1000)

# Predictions on test data
predictions = nn.predict(X_test)

# Evaluate the model using Mean Squared Error (MSE)
mse = mean_squared_error(y_test, predictions.flatten())
print("Mean Squared Error:", mse)
```

- **Output:**


Spyder (Python 3.11)
File Edit Search Source Run Debug Consoles Projects Tools View Help
C:\Users\krith
C:\Users\krith\untitled1.py
untitled0.py*
untitled1.py*
def forward(self, X):
self.hidden_output = self.sigmoid(np.dot(X, self.weights_input_hidden) + self.bias_hidd
self.predicted_output = self.sigmoid(np.dot(self.hidden_output, self.weights_hidden_out
return self.predicted_output
def backward(self, X, y):
# Calculate errors and deltas
self.output_error = y - self.predicted_output
self.output_delta = self.output_error * self.sigmoid_derivative(self.predicted_output)
self.hidden_error = self.output_delta.dot(self.weights_hidden_output.T)
self.hidden_delta = self.hidden_error * self.sigmoid_derivative(self.hidden_output)
# Update weights and biases
self.weights_hidden_output += self.hidden_output.T.dot(self.output_delta) * self.lr
self.weights_input_hidden += X.T.dot(self.hidden_delta) * self.lr
self.bias_output += np.sum(self.output_delta, axis=0) * self.lr
self.bias_hidden += np.sum(self.hidden_delta, axis=0) * self.lr
def train(self, X, y, epochs):
for epoch in range(epochs):
output = self.forward(X)
self.backward(X, y)
def predict(self, X):
return self.forward(X)
# Initialize and train the neural network
input_size = X_train.shape[1]
hidden_size = 10 # You can adjust this based on your needs
output_size = 1   # Predicting a single continuous value
nn = NeuralNetwork(input_size, hidden_size, output_size)
nn.train(X_train, y_train.reshape(-1, 1), epochs=1000)
# Predictions on test data
predictions = nn.predict(X_test)
# Evaluate the model using Mean Squared Error (MSE)
mse = mean_squared_error(y_test, predictions.flatten())
print("Mean Squared Error:", mse)
130 %
Help Variable Explorer Plots Files
Console 5/A
In [4]: runfile('C:/Users/krith/untitled1.py', wdir='C:/Users/krith')
Mean Squared Error: 36.99387023114789
In [5]:
IPython Console History
conda (Python 3.11.5) Completions: conda LSP: Python Line 88, Col 1 UTF-8 CRLF RW Mem 86%