Gradient Descent and Data Visualization

ROHIT RAGHUWANSHI

February 5, 2025

1 Python Code

```
!pip install openpyxl
!pip install missingno
!pip install seaborn

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

# Load dataset
dataset_url =
↪ "https://docs.google.com/spreadsheets/d/1-vUIn3tkLMALZ5dZVldliMO--88tQ0zQqqz5c5a8pNQ/export?format=csv"
df = pd.read_csv(dataset_url)

# --- Convert specific columns to numeric ---
numeric_columns = ['Time to Reach (hr)', 'Distance (km)']

for col in numeric_columns:
    if col in df.columns:
        df[col] = pd.to_numeric(df[col], errors='coerce')
    else:
        print(f"Warning: Column '{col}' not found in DataFrame. Skipping conversion.")

# Preprocess Data: Fill missing values with mean
numeric_df = df[numeric_columns]
df[numeric_columns] = numeric_df.fillna(numeric_df.mean())

# Extract features
X = df['Time to Reach (hr)'].values.reshape(-1, 1)
y = df['Distance (km)'].values.reshape(-1, 1)

# Normalize Data
X = (X - np.mean(X)) / np.std(X)
y = (y - np.mean(y)) / np.std(y)

# Initialize parameters
W = np.array([0.35])  # Initial weight
b = 0  # Initial bias
learning_rates = [0.01, 0.05, 0.1]  # Different learning rates
epochs = 100
batch_size = 8

# Function to calculate gradient and loss
def calculate_gradient_loss(X_batch, y_batch, W, b):
    y_pred = W * X_batch + b
    dW = -2 * np.mean((y_batch - y_pred) * X_batch)
    db = -2 * np.mean(y_batch - y_pred)
    loss = np.mean((y_batch - y_pred) ** 2)
    return dW, db, loss
```

```python
# Function to perform Gradient Descent
def gradient_descent(X, y, W, b, learning_rate, epochs, batch_size):
    m = len(X)
    losses = []
    gradient_norms = []

    for epoch in range(epochs):
        indices = np.random.permutation(m)
        X_shuffled = X[indices]
        y_shuffled = y[indices]

        for i in range(0, m, batch_size):
            X_batch = X_shuffled[i:i + batch_size]
            y_batch = y_shuffled[i:i + batch_size]

            dW, db, loss = calculate_gradient_loss(X_batch, y_batch, W, b)

            W -= learning_rate * dW
            b -= learning_rate * db

        loss = np.mean((y - (W * X + b)) ** 2)
        losses.append(loss)
        gradient_norms.append(np.sqrt(dW**2 + db**2))  # Gradient norm

        if epoch % 100 == 0:
            print(f"Epoch {epoch}: Loss = {loss:.4f}, W = {W[0]:.4f}, b = {b:.4f}")

    return W, b, losses, gradient_norms

# Run Gradient Descent for different learning rates
for learning_rate in learning_rates:
    print(f"\nRunning Gradient Descent with learning rate: {learning_rate}")
    W_final, b_final, losses, gradient_norms = gradient_descent(X, y, W, b, learning_rate,
    ↪ epochs, batch_size)

    # Plot Loss and Gradient Norm
    plt.figure(figsize=(12, 5))
    plt.subplot(1, 2, 1)
    plt.plot(range(epochs), losses)
    plt.xlabel('Epochs')
    plt.ylabel('Loss')
    plt.title(f'Loss vs Epochs (Learning Rate: {learning_rate})')

    plt.subplot(1, 2, 2)
    plt.plot(range(epochs), gradient_norms)
    plt.xlabel('Epochs')
    plt.ylabel('Gradient Norm')
    plt.title(f'Gradient Norm vs Epochs (Learning Rate: {learning_rate})')

    plt.tight_layout()
    plt.show()

# Scatter plot with regression line
plt.figure(figsize=(8, 6))
sns.regplot(x='Time to Reach (hr)', y='Distance (km)', data=df, scatter_kws={'alpha': 0.6},
↪ line_kws={'color': 'red'})
plt.title('Scatter Plot with Regression Line')
plt.xlabel('Time to Reach (hr)')
plt.ylabel('Distance (km)')
plt.show()

# Distribution of Time to Reach (hr)
```

```
plt.figure(figsize=(8, 6))
sns.histplot(df['Time to Reach (hr)'], kde=True)
plt.title('Distribution of Time to Reach (hr)')
plt.xlabel('Time to Reach (hr)')
plt.ylabel('Frequency')
plt.show()

# Distribution of Distance (km)
plt.figure(figsize=(8, 6))
sns.histplot(df['Distance (km)'], kde=True)
plt.title('Distribution of Distance (km)')
plt.xlabel('Distance (km)')
plt.ylabel('Frequency')
plt.show()

# Box plot of Time to Reach (hr) by Location Name (if applicable)
if 'Location Name' in df.columns:
    plt.figure(figsize=(12, 6))
    sns.boxplot(x='Location Name', y='Time to Reach (hr)', data=df)
    plt.title('Box Plot of Time to Reach (hr) by Location Name')
    plt.xticks(rotation=45, ha='right')
    plt.show()

# Box plot of Distance (km) by Location Name (if applicable)
if 'Location Name' in df.columns:
    plt.figure(figsize=(12, 6))
    sns.boxplot(x='Location Name', y='Distance (km)', data=df)
    plt.title('Box Plot of Distance (km) by Location Name')
    plt.xticks(rotation=45, ha='right')
    plt.show()
```