CS584: PROJECT 1

TITLE: VIDEO GAMES SALES PREDICTION MEMBERS:

1. RITHIKA KAVITHA SURESH - A20564346
2. EKTA SHUKLA - A20567127
3. ROGER KEWIN SAMSON - A20563057
4. JUDE ROSUN - A20564339

1. Importing Libraries and Loading Data for video games sales - vgsales

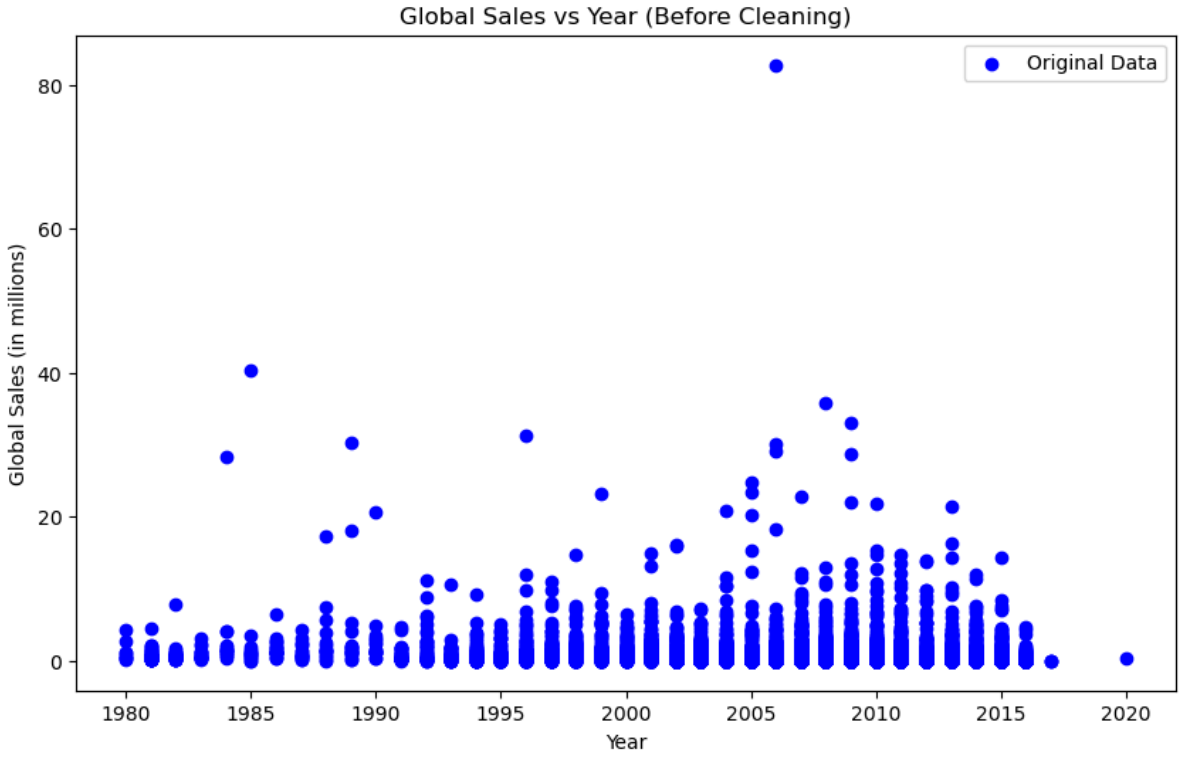
In [15]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split

# Load the vgsales.csv dataset
file_path = 'vgsales.csv'
df = pd.read_csv(file_path)

df.head()
```

Out[15]:

	Rank	Name	Platform	Year	Genre	Publisher	NA_Sales	EU_Sales	JP_Sales	Other
0	1	Wii Sports	Wii	2006.0	Sports	Nintendo	41.49	29.02	3.77	
1	2	Super Mario Bros.	NES	1985.0	Platform	Nintendo	29.08	3.58	6.81	
2	3	Mario Kart Wii	Wii	2008.0	Racing	Nintendo	15.85	12.88	3.79	
3	4	Wii Sports Resort	Wii	2009.0	Sports	Nintendo	15.75	11.01	3.28	
4	5	Pokemon Red/Pokemon Blue	GB	1996.0	Role-Playing	Nintendo	11.27	8.89	10.22	

2. Data visualization (scatter plot) - Global sales vs year(before cleaning)

In [16]:
```
# Scatter plot of original data (before cleaning)
plt.figure(figsize=(10, 6))
plt.scatter(df['Year'], df['Global_Sales'], color='blue', label='Original Data')
plt.title('Global Sales vs Year (Before Cleaning)')
plt.xlabel('Year')
plt.ylabel('Global Sales (in millions)')
plt.legend()
plt.show()
```



Data cleaning and processing

- Filling missing value

In [17]:
```
# Fill missing Year values with the median year
median_year = df['Year'].median()
df['Year'].fillna(median_year, inplace=True)

# Fill missing Publisher values with 'Unknown'
df['Publisher'].fillna('Unknown', inplace=True)

# Handle missing sales values
df['NA_Sales'].fillna(0, inplace=True)
df['EU_Sales'].fillna(0, inplace=True)
df['JP_Sales'].fillna(0, inplace=True)
df['Other_Sales'].fillna(0, inplace=True)


df.dropna(subset=['Global_Sales'], inplace=True)

df['Year'] = df['Year'].astype(int)

df.isnull().sum()
```

Out[17]:
```
Rank            0
Name            0
Platform        0
Year            0
Genre           0
Publisher       0
NA_Sales        0
EU_Sales        0
JP_Sales        0
Other_Sales     0
Global_Sales    0
dtype: int64
```

Feature and Splitting the dataset for Training and

In [18]:
```
X = df[['Year', 'NA_Sales', 'EU_Sales', 'JP_Sales', 'Other_Sales']].values
y = df['Global_Sales'].values

# Splitting the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_sta
```

Train-test and standardization

In [19]:
```
# Standardizing the data
def standardize(X):
    mean = np.mean(X, axis=0)
    std = np.std(X, axis=0)
    return (X - mean) / std, mean, std
X_train_scaled, mean, std = standardize(X_train)
X_test_scaled = (X_test - mean) / std
```

Elasticnet model implementation

In [20]:
```
class ElasticNetRegressionUpdated:
    def __init__(self, l1_ratio=0.1, alpha=0.01, max_iter=1000, tol=1e-4, learning_
        self.l1_ratio = l1_ratio
        self.alpha = alpha
        self.max_iter = max_iter  # Maximum iterations for gradient descent
        self.tol = tol
        self.learning_rate = learning_rate

    def fit(self, X, y):
        m, n = X.shape
        self.coef_ = np.zeros(n)
        self.intercept_ = 0

        for _ in range(self.max_iter):
            y_pred = np.dot(X, self.coef_) + self.intercept_
            residuals = y_pred - y

            gradient_w = (2/m) * np.dot(X.T, residuals) + self.alpha * ((1 - self.l
            gradient_b = (2/m) * np.sum(residuals)

            self.coef_ -= self.learning_rate * gradient_w
            self.intercept_ -= self.learning_rate * gradient_b

            if np.linalg.norm(gradient_w) < self.tol:
                break

    def predict(self, X):
        return np.dot(X, self.coef_) + self.intercept_
```

ElasticNet Regression Model Initialization and Training

In [21]:
```
# Initialize and train the updated ElasticNet regression model
elastic_net_model_updated = ElasticNetRegressionUpdated(l1_ratio=0.1, alpha=0.01, m
elastic_net_model_updated.fit(X_train_scaled, y_train)
```

Predictions and Evaluation of Updated ElasticNet Model

In [22]:
```
y_pred_updated = elastic_net_model_updated.predict(X_test_scaled)
predicted_df_updated = pd.DataFrame({'Actual': y_test, 'Predicted': y_pred_updated}
predicted_df_updated.to_csv('predicted_vgsales_updated.csv', index=False)
predicted_df_updated.head()
```

Out[22]:

	Actual	Predicted
0	1.25	1.252061
1	0.34	0.351645
2	0.27	0.282011
3	0.14	0.142005
4	0.31	0.313169

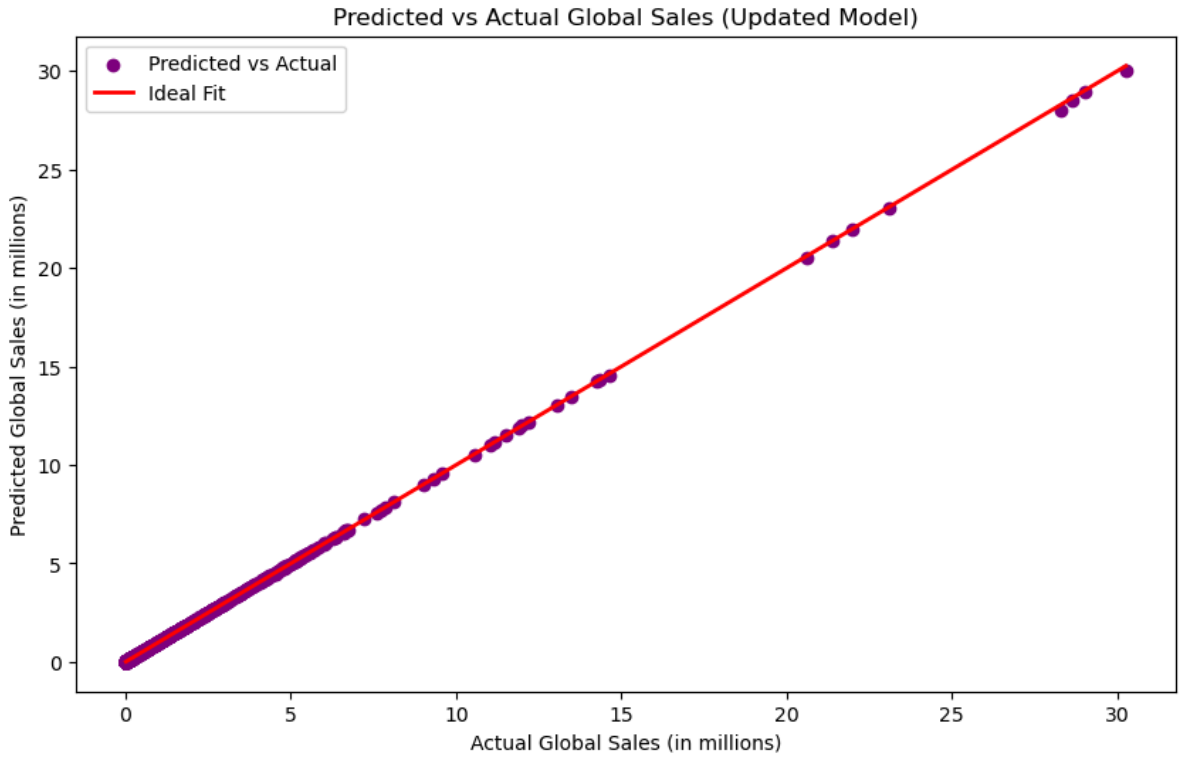
Model evaluation

In [23]:
```
# Calculating Mean Squared Error (MSE) and R-squared (R2)
mse_updated = np.mean((y_pred_updated - y_test) ** 2)
r_squared_updated = 1 - (np.sum((y_test - y_pred_updated) ** 2) / np.sum((y_test -

print(f'Mean Squared Error: {mse_updated}')
print(f'R-squared: {r_squared_updated}')
```

```
Mean Squared Error: 7.482819642676165e-05
R-squared: 0.9999646605217778
```

Data Visualizations

In [24]:
```
# Scatter plot of predicted vs actual Global Sales
plt.figure(figsize=(10, 6))
plt.scatter(y_test, y_pred_updated, color='purple', label='Predicted vs Actual')
plt.plot([min(y_test), max(y_test)], [min(y_test), max(y_test)], color='red', lw=2,
plt.title('Predicted vs Actual Global Sales (Updated Model)')
plt.xlabel('Actual Global Sales (in millions)')
plt.ylabel('Predicted Global Sales (in millions)')
plt.legend()
plt.show()
```



In [25]:
```
predicted_df_updated = pd.DataFrame({'Actual': y_test, 'Predicted': y_pred_updated}
predicted_df_updated.to_csv('predicted_vgsales_updated.csv', index=False)
predicted_df_updated.head()
```

Out[25]:

	Actual	Predicted
0	1.25	1.252061
1	0.34	0.351645
2	0.27	0.282011
3	0.14	0.142005
4	0.31	0.313169

In [ ]: