CCP Report

Data Mining

Title: Video Game Data Analysis

Presented by:
Nameer Iqbal (02-136212-016)
Ammar Khalid (02-136212-025)

Acknowledgement:

We would like to express my sincere gratitude to my teacher, Miss Sadia Nazim, for her unwavering support and guidance throughout the course. Her commitment to fostering a positive and enriching learning environment has been instrumental in shaping my understanding of data mining and analysis. Her expertise, encouragement, and dedication have inspired me to explore and apply various techniques, contributing significantly to my academic and professional growth. We am truly thankful for the valuable insights, constructive feedback, and the genuine passion they bring to the field of education. Her impact will undoubtedly resonate in our academic journey and future endeavors.

1. Introduction:

This report delves into the analysis of a video game sales dataset using various data mining techniques. The dataset encompasses information on video game sales in different regions, such as North America (NA_sales), Japan (JP_sales), Europe (EU_sales), and global sales (Global_sales). The primary objective is to derive meaningful insights from the data, involving data preprocessing and the application of data mining techniques for in-depth analysis.

2. Data Overview:

a) Dataset Loading and Inspection (EDA):

The dataset was first uploaded to google colab using the Upload() function and then it was loaded using the Pandas library in Python. EDA was performed by an initial inspection of the dataset that provides a snapshot of the first few rows, data types, and summary statistics.

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from google.colab import files
uploaded=files.upload()
```

```
Choose files  No file chosen        Upload widget is only available when the cell has been executed
Saving vgogte.csv to vgogte.csv
```

```
[ ] import io
    file_name = "vgogte.csv"
    df = pd.read_csv(io.StringIO(uploaded[file_name].decode('utf-8')))
    print(df.head())
```

```
                         Name Platform Year_of_Release         Genre  \
0                'Wii Sports'      Wii            2006        Sports
1           'Super Mario Bros.'      NES            1985      Platform
2              'Mario Kart Wii'      Wii            2008        Racing
3           'Wii Sports Resort'      Wii            2009        Sports
4  'Pokemon Red/Pokemon Blue'       GB            1996  Role-Playing

  Publisher NA_Sales EU_Sales  JP_Sales  Global_Sales
0  Nintendo    41.36    28.96      3.77         82.53
1  Nintendo    29.08     3.58      6.81         40.24
2  Nintendo    15.68    12.76      3.79         35.52
3  Nintendo    15.61    10.93      3.28         32.77
4  Nintendo    11.27     8.89     10.22         31.37
```

```
[ ] df.head()
```

	Name	Platform	Year_of_Release	Genre	Publisher	NA_Sales	EU_Sales	JP_Sales	Other_Sales	Global_Sales	Unnamed: 10	Unnamed: 11	Unnamed: 12	Unnamed: 13	Unnamed: 14	Unnamed: 15	Unnamed: 16	Unnamed: 17	Unnamed: 18
0	'Wii Sports'	Wii	2006	Sports	Nintendo	41.36	28.96	3.77	8.45	82.53	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
1	'Super Mario Bros.'	NES	1985	Platform	Nintendo	29.08	3.58	6.81	0.77	40.24	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2	'Mario Kart Wii'	Wii	2008	Racing	Nintendo	15.68	12.76	3.79	3.29	35.52	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
3	'Wii Sports Resort'	Wii	2009	Sports	Nintendo	15.61	10.93	3.28	2.95	32.77	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
4	'Pokemon Red/Pokemon Blue'	GB	1996	Role-Playing	Nintendo	11.27	8.89	10.22	1	31.37	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN

```
df.info()
df.describe()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 16719 entries, 0 to 16718
Data columns (total 19 columns):
 #   Column           Non-Null Count  Dtype
---  ------           --------------  -----
 0   Name             16719 non-null  object
 1   Platform         16719 non-null  object
 2   Year_of_Release  16452 non-null  object
 3   Genre            16717 non-null  object
 4   Publisher        16666 non-null  object
 5   NA_Sales         16718 non-null  object
 6   EU_Sales         16719 non-null  object
 7   JP_Sales         16719 non-null  object
 8   Other_Sales      16719 non-null  object
 9   Global_Sales     16719 non-null  object
 10  Unnamed: 10      0 non-null      float64
 11  Unnamed: 11      0 non-null      float64
 12  Unnamed: 12      0 non-null      float64
 13  Unnamed: 13      0 non-null      float64
 14  Unnamed: 14      0 non-null      float64
 15  Unnamed: 15      0 non-null      float64
 16  Unnamed: 16      644 non-null    object
 17  Unnamed: 17      51 non-null     object
 18  Unnamed: 18      7 non-null      object
dtypes: float64(6), object(13)
memory usage: 2.4+ MB
```

```
[ ] df.isnull().sum()
```

```
Name                2
Platform            1
Year_of_Release   267
Genre               2
Publisher          53
NA_Sales           25
EU_Sales           26
JP_Sales           16
Global_Sales       16
dtype: int64
```

3. Data Cleaning:

To ensure data quality, several cleaning steps were performed:

a) Removal of unnamed columns.

There were some unnamed columns present in the dataset as shown in the above figures.

```
[ ]

# Identify columns with 'Unnamed' in the column name
unwanted_columns = [col for col in df.columns if 'Unnamed' in col]

# Drop the unwanted columns
df = df.drop(columns=unwanted_columns, axis=1)

# Now df contains the original dataset without the 'Unnamed' columns
df.head()
```

	Name	Platform	Year_of_Release	Genre	Publisher	NA_Sales	EU_Sales	JP_Sales	Global_Sales
0	'Wii Sports'	Wii	2006	Sports	Nintendo	41.36	28.96	3.77	82.53
1	'Super Mario Bros.'	NES	1985	Platform	Nintendo	29.08	3.58	6.81	40.24
2	'Mario Kart Wii'	Wii	2008	Racing	Nintendo	15.68	12.76	3.79	35.52
3	'Wii Sports Resort'	Wii	2009	Sports	Nintendo	15.61	10.93	3.28	32.77
4	'Pokemon Red/Pokemon Blue'	GB	1996	Role-Playing	Nintendo	11.27	8.89	10.22	31.37

b) Conversion of relevant columns ('NA_Sales', 'EU_Sales', 'JP_Sales') to numeric format (float) & Removal of rows with missing values (NaN).

All columns were having the object datatype. In order to perform further functions on numeric datas, the datatypes were changed to numeric. In addition to that there were some string values present in the numeric columns that were also dropped as incorrect data.

```
# Convert columns 'A' and 'B' to numeric, coercing errors to NaN
df['NA_Sales'] = pd.to_numeric(df['NA_Sales'], errors='coerce')
df['EU_Sales'] = pd.to_numeric(df['EU_Sales'], errors='coerce')
df['JP_Sales'] = pd.to_numeric(df['JP_Sales'], errors='coerce')

# Drop rows with NaN values (non-convertible strings)
df = df.dropna()

# Convert the columns to float
df['NA_Sales'] = df['NA_Sales'].astype(float)
df['EU_Sales'] = df['EU_Sales'].astype(float)
df['JP_Sales'] = df['JP_Sales'].astype(float)
# Check the data types after conversion
print(df.dtypes)
print(df)
```

```
Name               object
Platform           object
Year_of_Release    object
Genre              object
Publisher          object
NA_Sales          float64
EU_Sales          float64
JP_Sales          float64
Global_Sales      float64
dtype: object
```

c) Imputation of missing values using the Z-score method.

The missing values in the dataset were replaced using the Z-score method.

```
[ ] df.replace('', np.nan, inplace=True)

    # Function to replace missing values using z-score method
    def replace_missing_with_zscore(df, column):
        mean = df[column].mean()
        std_dev = df[column].std()
        missing_values = df[column].isnull()

        # Calculate z-scores for non-missing values
        z_scores = (df[column] - mean) / std_dev

        # Replace missing values with mean of non-missing values
        df.loc[missing_values, column] = mean

    # Replace missing values in each column with z-score method
    for column in df.columns:
        if df[column].dtype != 'object':  # Process only numerical columns
            replace_missing_with_zscore(df, column)
    df.isnull().sum()
```

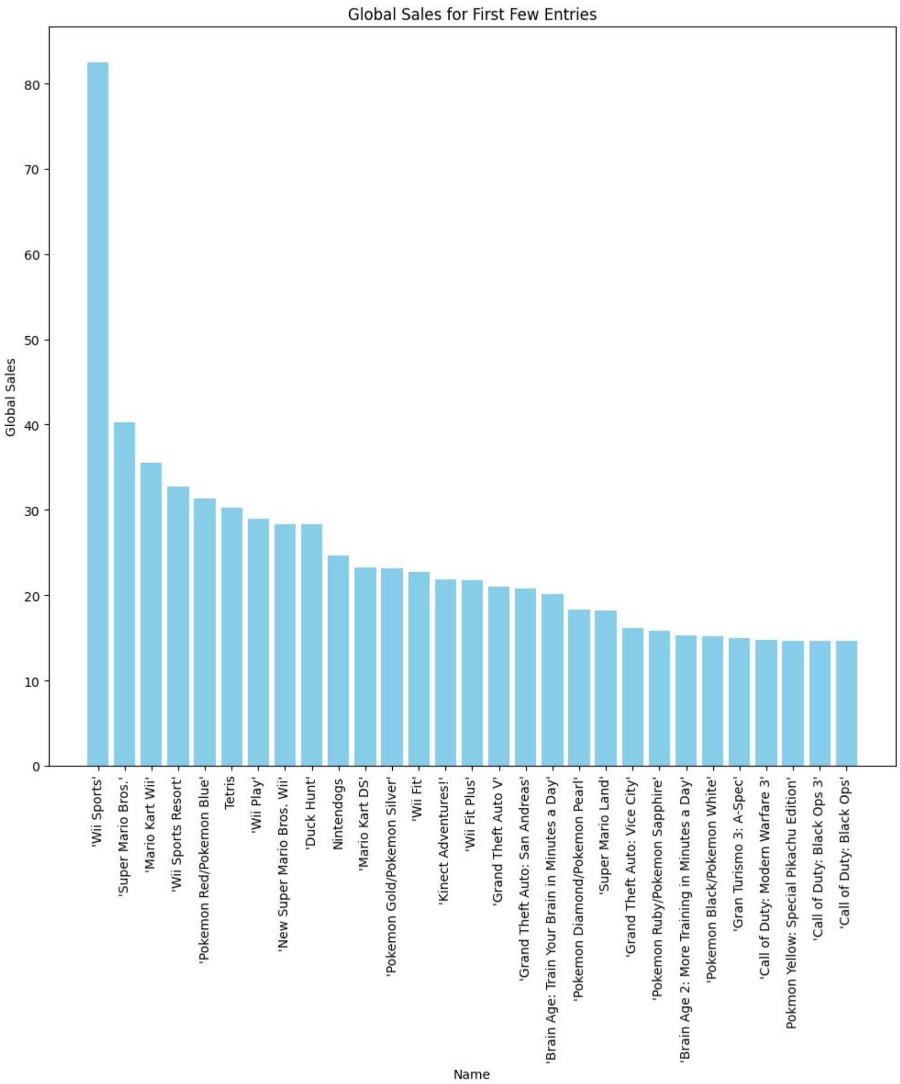
```
Name               0
Platform           0
Year_of_Release    0
Genre              0
Publisher          0
NA_Sales           0
EU_Sales           0
JP_Sales           0
Global_Sales       0
dtype: int64
```

4. Data Visualization:

a) Bar Plot for Global Sales of Video Games:

The following bar graph shows the name of the video game and their corresponding sales across the globe. First 30 entries were chosen to visualize the data more accurately.

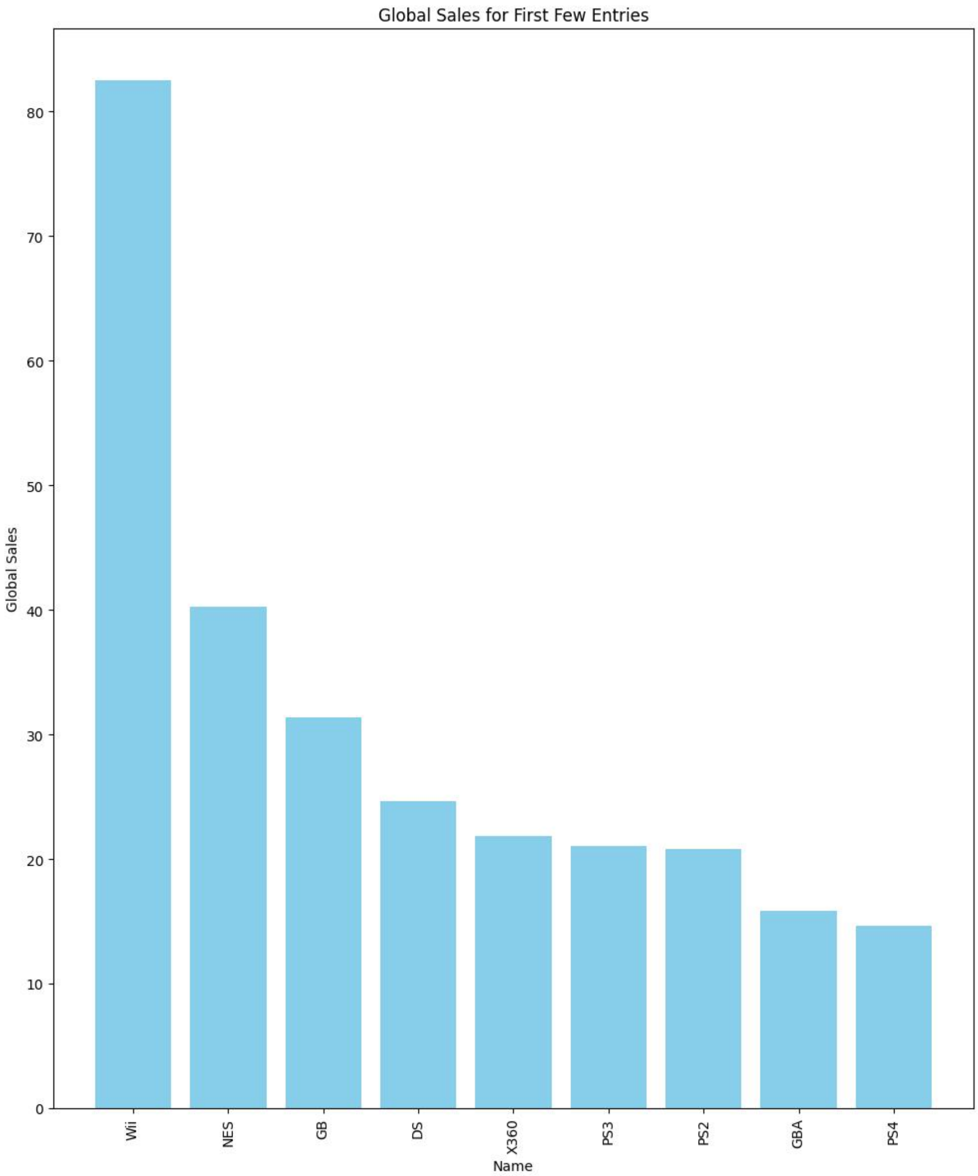
```
num_entries_to_plot = 30
subset_df = df.head(num_entries_to_plot)

# Extracting the 'Names' and 'Global_sales' columns from the subset
names = subset_df['Name']
global_sales = subset_df['Global_Sales']

# Creating a bar plot
plt.figure(figsize=(10, 12))
plt.bar(names, global_sales, color='skyblue')
plt.xlabel('Name')
plt.ylabel('Global Sales')
plt.title('Global Sales for First Few Entries')
plt.xticks(rotation=90)  # Rotate x-axis labels for better readability if needed
plt.tight_layout()

plt.show()
```



b) Platform and global sales:

The following bar graph visualizes the relation between the platforms the games were released on and their corresponding global sales. First 30 entries were chosen to visualize the data more accurately

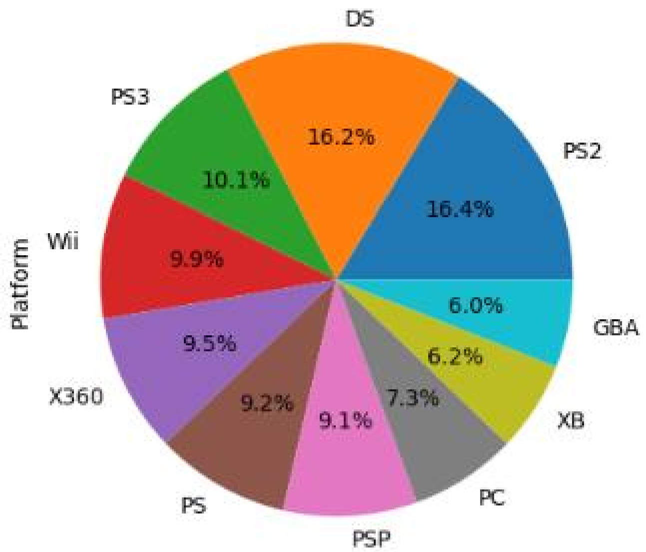
```
num_entries_to_plot = 30
subset_df = df.head(num_entries_to_plot)

# Extracting the 'Names' and 'Global_sales' columns from the subset
names = subset_df['Platform']
global_sales = subset_df['Global_Sales']

# Creating a bar plot
plt.figure(figsize=(10, 12))
plt.bar(names, global_sales, color='skyblue')
plt.xlabel('Name')
plt.ylabel('Global Sales')
plt.title('Global Sales for First Few Entries')
plt.xticks(rotation=90)  # Rotate x-axis labels for better readability if needed
plt.tight_layout()

plt.show()
```



c) Pie Chart :

The following pie chart shows the top 10 platforms upon which the most games were released.

```
df.Platform.value_counts().head(10).plot.pie(autopct='%1.1f%%')
plt.show()
```



(PTO)

5. Outlier Detection:

Outliers in sales data for North America (NA_Sales), Europe (EU_Sales), and Japan (JP_Sales) were detected using the Interquartile Range (IQR) method.

a) North America Sales:

```
# Calculate Q1 (25th percentile)
Q1 = df['NA_Sales'].quantile(0.25)

# Calculate Q3 (75th percentile)
Q3 = df['NA_Sales'].quantile(0.75)

# Calculate IQR (Interquartile Range)
IQR = Q3 - Q1

# Calculate lower and upper bounds for outliers
lower_bound = Q1 - 1.5 * IQR
upper_bound = Q3 + 1.5 * IQR

# Detect outliers
outliers = df[(df['NA_Sales'] < lower_bound) | (df['NA_Sales'] > upper_bound)]

print("Lower bound for outliers:", lower_bound)
print("Upper bound for outliers:", upper_bound)
print("Outliers:")
print(outliers)
```

```
Lower bound for outliers: -0.36
Upper bound for outliers: 0.6
Outliers:

            Genre             Publisher  NA_Sales  EU_Sales  JP_Sales  \
0          Sports              Nintendo     41.36     28.96      3.77
1        Platform              Nintendo     29.08      3.58      6.81
2          Racing              Nintendo     15.68     12.76      3.79
3          Sports              Nintendo     15.61     10.93      3.28
4    Role-Playing              Nintendo     11.27      8.89     10.22
...           ...                   ...       ...       ...       ...
2966       Sports     'Electronic Arts'      0.62      0.04      0.00
2970       Sports     'Electronic Arts'      0.63      0.02      0.00
2986       Racing                   THQ      0.63      0.00      0.00
3015       Action  'Mattel Interactive'      0.63      0.03      0.00
3061       Action            Activision      0.62      0.04      0.00

      Global_Sales
0            82.53
1            40.24
2            35.52
3            32.77
4            31.37
...            ...
2966          0.68
2970          0.68
2986          0.68
3015          0.67
3061          0.66

[1643 rows x 9 columns]
```

b) Europe Sales:

```
[ ]
# Calculate Q1 (25th percentile)
Q1 = df['EU_Sales'].quantile(0.25)

# Calculate Q3 (75th percentile)
Q3 = df['EU_Sales'].quantile(0.75)

# Calculate IQR (Interquartile Range)
IQR = Q3 - Q1

# Calculate lower and upper bounds for outliers
lower_bound = Q1 - 1.5 * IQR
upper_bound = Q3 + 1.5 * IQR

# Detect outliers
outliers = df[(df['EU_Sales'] < lower_bound) | (df['EU_Sales'] > upper_bound)]

print("Lower bound for outliers:", lower_bound)
print("Upper bound for outliers:", upper_bound)
print("Outliers:")
print(outliers)
```

```
Lower bound for outliers: -0.165
Upper bound for outliers: 0.275
Outliers:

             Genre                       Publisher  NA_Sales  EU_Sales  \
0           Sports                        Nintendo     41.36     28.96
1         Platform                        Nintendo     29.08      3.58
2           Racing                        Nintendo     15.68     12.76
3           Sports                        Nintendo     15.61     10.93
4     Role-Playing                        Nintendo     11.27      8.89
...            ...                             ...       ...       ...
5483          Misc                  'Nordic Games'      0.00      0.29
5487          Misc                   'Deep Silver'      0.00      0.28
5568        Action             'Eidos Interactive'      0.00      0.28
5726        Sports                         Ubisoft      0.00      0.29
5760          Misc  'Sony Computer Entertainment'      0.00      0.28

      JP_Sales  Global_Sales
0         3.77         82.53
1         6.81         40.24
2         3.79         35.52
3         3.28         32.77
4        10.22         31.37
...        ...           ...
5483      0.00          0.33
5487      0.00          0.33
5568      0.00          0.32
5726      0.00          0.31
5760      0.00          0.31

[2011 rows x 9 columns]
```

c) Japan Sales:

```
[ ]
    # Calculate Q1 (25th percentile)
    Q1 = df['JP_Sales'].quantile(0.25)

    # Calculate Q3 (75th percentile)
    Q3 = df['JP_Sales'].quantile(0.75)

    # Calculate IQR (Interquartile Range)
    IQR = Q3 - Q1

    # Calculate lower and upper bounds for outliers
    lower_bound = Q1 - 1.5 * IQR
    upper_bound = Q3 + 1.5 * IQR

    # Detect outliers
    outliers = df[(df['JP_Sales'] < lower_bound) | (df['JP_Sales'] > upper_bound)]

    print("Lower bound for outliers:", lower_bound)
    print("Upper bound for outliers:", upper_bound)
    print("Outliers:")
    print(outliers)
```

```
Lower bound for outliers: -0.06
Upper bound for outliers: 0.1
Outliers:

       Year_of_Release         Genre                        Publisher  NA_Sales  \
0                 2006        Sports                         Nintendo     41.36
1                 1985      Platform                         Nintendo     29.08
2                 2008        Racing                         Nintendo     15.68
3                 2009        Sports                         Nintendo     15.61
4                 1996  Role-Playing                         Nintendo     11.27
...                ...           ...                              ...       ...
10349             2012        Action             'Namco Bandai Games'      0.00
10352             1995  Role-Playing             'Namco Bandai Games'      0.00
10353             2011        Action  'Konami Digital Entertainment'      0.00
10356             2016        Action                            PQube      0.00
10357             2010      Fighting  'Konami Digital Entertainment'      0.00

       EU_Sales  JP_Sales  Global_Sales
0         28.96      3.77         82.53
1          3.58      6.81         40.24
2         12.76      3.79         35.52
3         10.93      3.28         32.77
4          8.89     10.22         31.37
...         ...       ...           ...
10349      0.00      0.11          0.11
10352      0.00      0.11          0.11
10353      0.00      0.11          0.11
10356      0.00      0.11          0.11
10357      0.00      0.11          0.11

[2391 rows x 9 columns]
```

6. Predictive Modeling:

Support Vector Regressor (SVR) was used to predict global video game sales based on features such as gaming platform, sales in Europe, Japan, and North America. Categorical encoding is applied to the 'Platform' column, and the dataset is split into training and testing sets. The SVR model with a linear kernel is then initialized, trained, and used to make predictions on the test set. Model performance is evaluated using metrics such as Maean Squared Error (MSE) and R-squared Score (R2), providing insights into the model's predictive accuracy.

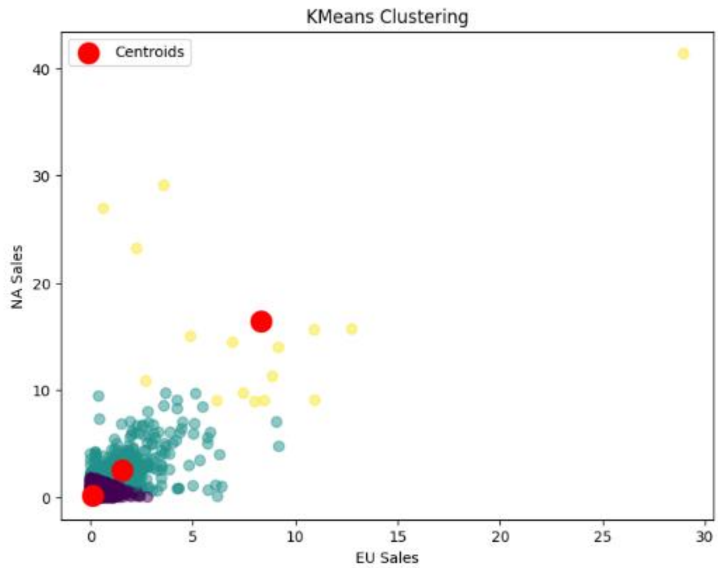
```
label_encoder = LabelEncoder()
df['Platform'] = label_encoder.fit_transform(df['Platform'])

# Define features and target variable
features = ['Platform', 'EU_Sales', 'JP_Sales', 'NA_Sales']
target = 'Global_Sales'

X = df[features]
y = df[target]

# Split the 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)

# Initialize and train the SVR model
svr = SVR(kernel='linear')  # You can try different kernels: 'linear', 'rbf', 'poly', etc.
svr.fit(X_train, y_train)

# Predict using the trained model
y_pred = svr.predict(X_test)

# Evaluate the model
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

print(f"Mean Squared Error: {mse:.2f}")
print(f"R-squared Score: {r2:.2f}")
#Lower values of MSE indicate a better fit of the model to the data. A value of zero means
#
```

```
Mean Squared Error: 0.01
R-squared Score: 1.00
```

7. Clustering Analysis:

KMeans clustering was used on video game sales data, specifically focusing on the 'EU_Sales,' 'JP_Sales,' and 'NA_Sales' features. It selects three clusters and utilizes the KMeans algorithm to assign cluster labels and identify centroids. These labels are incorporated into the dataset, and the resulting clusters are visualized in a scatter plot, with centroids highlighted in red. The printed cluster centers showcase the average sales values for each cluster in the specified dimensions. Overall, this analysis aims to uncover inherent patterns and groupings in video game sales data based on the selected sales features.

```python
# Considering 'EU_Sales', 'JP_Sales', 'NA_Sales' for clustering
features = ['EU_Sales', 'JP_Sales', 'NA_Sales']
X = df[features]

# Choosing the number of clusters
num_clusters = 3

# Initialize KMeans with the number of clusters
kmeans = KMeans(n_clusters=num_clusters, random_state=42)

# Fit KMeans to the data
kmeans.fit(X)

# Getting cluster labels and centroids
cluster_labels = kmeans.labels_
centroids = kmeans.cluster_centers_

# Adding cluster labels to the dataset
df['Cluster'] = cluster_labels

# Visualizing the clusters
plt.figure(figsize=(8, 6))
plt.scatter(X['EU_Sales'], X['NA_Sales'], c=cluster_labels, cmap='viridis', s=50, alpha=0.5)
plt.scatter(centroids[:, 0], centroids[:, 2], marker='o', c='red', s=200, label='Centroids')
plt.xlabel('EU Sales')
plt.ylabel('NA Sales')
plt.title('KMeans Clustering')
plt.legend()
plt.show()

# Print cluster centers
print("Cluster Centers:")
for i, centroid in enumerate(centroids):
    print(f"Cluster {i+1}: {centroid}")
```




```
Cluster Centers:
Cluster 1: [0.08978427 0.06041869 0.16835206]
Cluster 2: [1.54281955 0.47464286 2.54898496]
Cluster 3: [ 8.303125  3.988125 16.440625]
```

Conclusion

In conclusion, this report has provided an in-depth analysis of the video game sales dataset through data cleaning, visualization, outlier detection, predictive modeling, and clustering. The insights gained from this analysis can inform decision-making in the gaming industry, with the potential for further exploration based on specific research questions and business objectives.