University of Hafr Al-Batin

College of Computer Science and Engineering (CCSE)

DSC307-Project Data Science in Research

Analyzing Video Game Sales Data

Group Members:

#	Name	ID
1	Fadiyah alanazi	2201001182
2	Ghazlan alanazi	2201000995
3	Elham khatim	2201001444
4	Ebtisam falih	2201002692
	Reem alqahtani	2201003207
	Latifah alanazi	2201002712

Introduction

1.1 Background

The video game industry has seen remarkable growth over the years, becoming a multi-billion dollar global market. The availability of data on video game sales provides a unique opportunity to gain insights into player preferences, market trends, and the evolution of the industry
The world of video games is a vast and dynamic industry that has captivated millions of players and generated billions of dollars in revenue over the years. With the advancement of technology and the emergence of various gaming platforms, understanding the trends and patterns within the video game market has become increasingly important.

In this project, we delve into the fascinating realm of video game sales data. We utilize the "Video Game Sales" dataset obtained from Kaggle, which contains a wealth of information about video game titles, their platforms, genres, release years, and sales figures in different regions. By analyzing this dataset, we aim to gain valuable insights into the world of video games, uncover trends,

1.2 Project Objectives

The main objectives of this project are as follows:

- Visualize key trends and patterns in video game sales data.
- Extract insights into the most popular platforms, genres, and sales trends.

Throughout this project, we will employ various data analysis and visualization techniques to extract meaningful information from the dataset. We will also perform data cleaning and preprocessing to ensure the quality and reliability of our analyses.

1.3 Dataset Overview

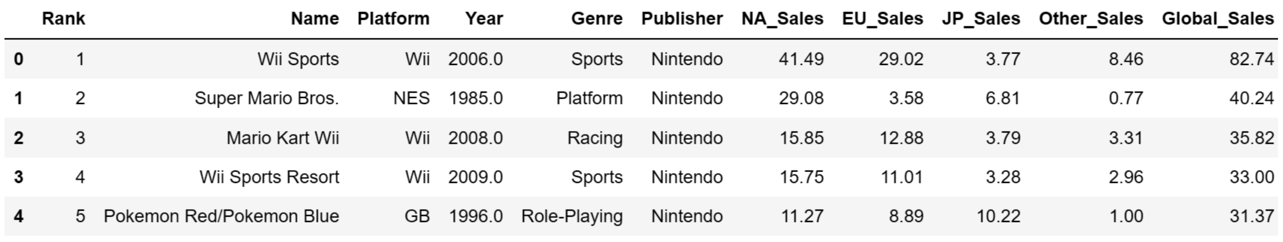
- Source: Kaggle's "Video Game Sales" dataset
- URL:(https://www.kaggle.com/datasets/gregorut/videogamesales)
- Number of Rows: [16598]
- Number of Columns: [11]

This project serves as a practical exercise in data exploration, data cleaning, and data visualization using Python and popular libraries such as Pandas, Matplotlib, and Seaborn. It provides an opportunity to gain insights into the video game industry and enhance our skills in data analysis.

So, let's embark on this journey to uncover the secrets hidden within the world of video game sales data!

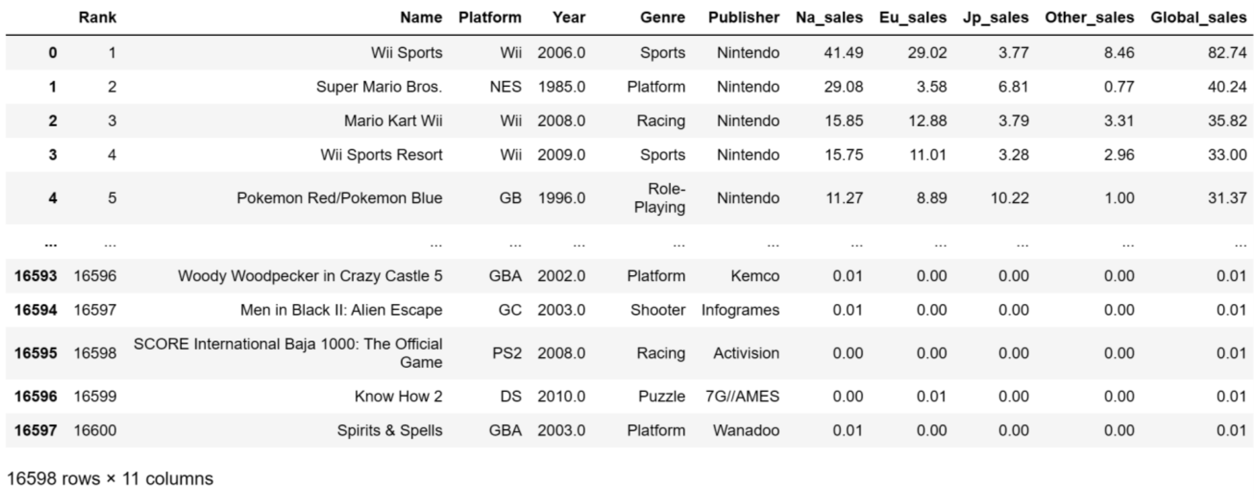
2.Data Exploration

First of all, let’s get familiarized with our data content. So, We displayed the first top records of our dataset.



	Rank	Name	Platform	Year	Genre	Publisher	NA_Sales	EU_Sales	JP_Sales	Other_Sales	Global_Sales
0	1	Wii Sports	Wii	2006.0	Sports	Nintendo	41.49	29.02	3.77	8.46	82.74
1	2	Super Mario Bros.	NES	1985.0	Platform	Nintendo	29.08	3.58	6.81	0.77	40.24
2	3	Mario Kart Wii	Wii	2008.0	Racing	Nintendo	15.85	12.88	3.79	3.31	35.82
3	4	Wii Sports Resort	Wii	2009.0	Sports	Nintendo	15.75	11.01	3.28	2.96	33.00
4	5	Pokemon Red/Pokemon Blue	GB	1996.0	Role-Playing	Nintendo	11.27	8.89	10.22	1.00	31.37

As shown below, the number of records is 16598 and the number of columns is 11.



	Rank	Name	Platform	Year	Genre	Publisher	Na_sales	Eu_sales	Jp_sales	Other_sales	Global_sales
0	1	Wii Sports	Wii	2006.0	Sports	Nintendo	41.49	29.02	3.77	8.46	82.74
1	2	Super Mario Bros.	NES	1985.0	Platform	Nintendo	29.08	3.58	6.81	0.77	40.24
2	3	Mario Kart Wii	Wii	2008.0	Racing	Nintendo	15.85	12.88	3.79	3.31	35.82
3	4	Wii Sports Resort	Wii	2009.0	Sports	Nintendo	15.75	11.01	3.28	2.96	33.00
4	5	Pokemon Red/Pokemon Blue	GB	1996.0	Role-Playing	Nintendo	11.27	8.89	10.22	1.00	31.37
...	...	...	...	...	...	...	...	...	...	...	...
16593	16596	Woody Woodpecker in Crazy Castle 5	GBA	2002.0	Platform	Kemco	0.01	0.00	0.00	0.00	0.01
16594	16597	Men in Black II: Alien Escape	GC	2003.0	Shooter	Infogrames	0.01	0.00	0.00	0.00	0.01
16595	16598	SCORE International Baja 1000: The Official Game	PS2	2008.0	Racing	Activision	0.00	0.00	0.00	0.00	0.01
16596	16599	Know How 2	DS	2010.0	Puzzle	7G//AMES	0.00	0.01	0.00	0.00	0.01
16597	16600	Spirits & Spells	GBA	2003.0	Platform	Wanadoo	0.01	0.00	0.00	0.00	0.01

16598 rows × 11 columns

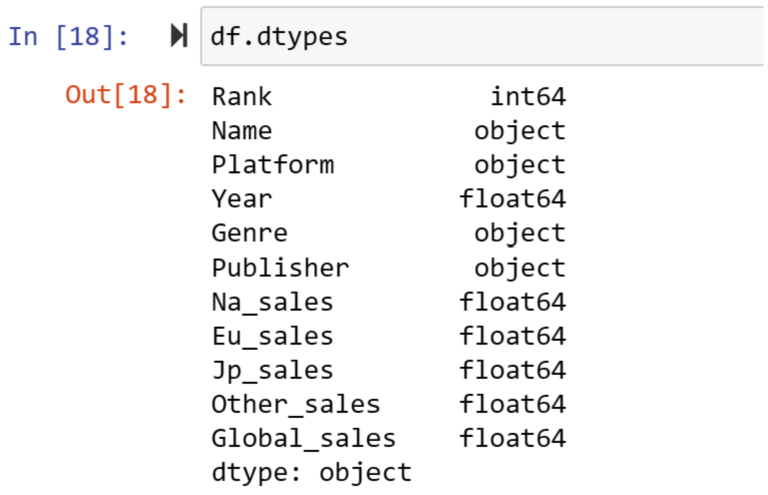
Now, let’s know more about the columns and their data types in addition to viewing statistical factors about each column.

The columns’ data types seem to be logical and we have some erorr

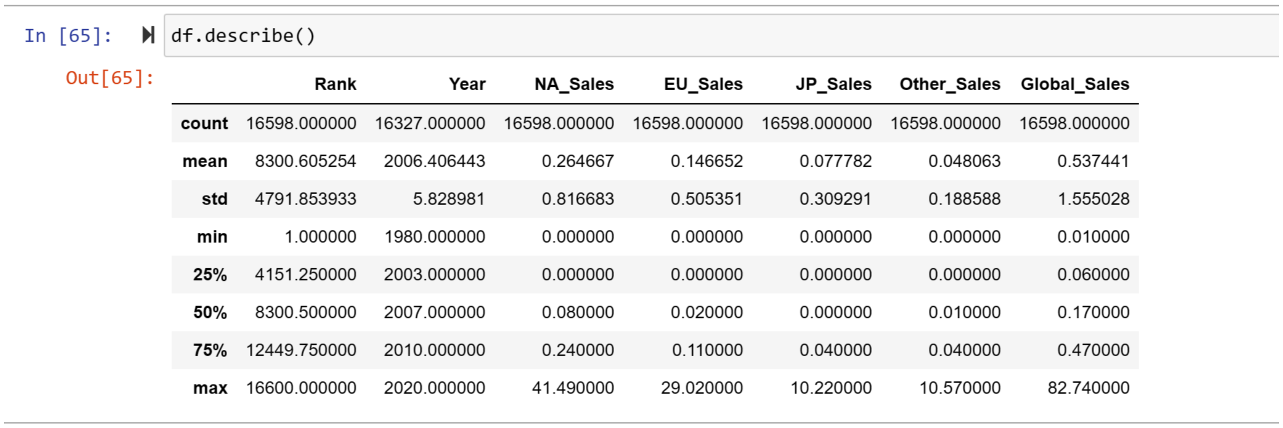
The describe method is used to view basic information about each column. It shows the following:

- The values count
- The values’ average

- The values’ standard deviation
- The minimum value
- The maximum value



```
In [18]:  df.dtypes

Out[18]:  Rank              int64
          Name             object
          Platform         object
          Year            float64
          Genre            object
          Publisher        object
          Na_sales        float64
          Eu_sales        float64
          Jp_sales        float64
          Other_sales     float64
          Global_sales    float64
          dtype: object
```



```
In [65]:  df.describe()

Out[65]:
```

	Rank	Year	NA_Sales	EU_Sales	JP_Sales	Other_Sales	Global_Sales
count	16598.000000	16327.000000	16598.000000	16598.000000	16598.000000	16598.000000	16598.000000
mean	8300.605254	2006.406443	0.264667	0.146652	0.077782	0.048063	0.537441
std	4791.853933	5.828981	0.816683	0.505351	0.309291	0.188588	1.555028
min	1.000000	1980.000000	0.000000	0.000000	0.000000	0.000000	0.010000
25%	4151.250000	2003.000000	0.000000	0.000000	0.000000	0.000000	0.060000
50%	8300.500000	2007.000000	0.080000	0.020000	0.000000	0.010000	0.170000
75%	12449.750000	2010.000000	0.240000	0.110000	0.040000	0.040000	0.470000
max	16600.000000	2020.000000	41.490000	29.020000	10.220000	10.570000	82.740000

3.Data Cleaning & Preprocessing

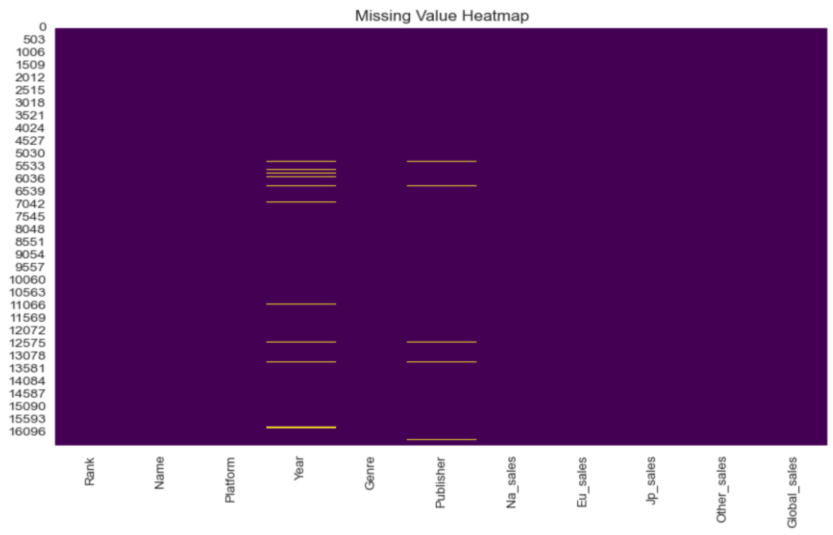
3.1Check the missing values in each column

We checked the missing values in each column in 2 different ways by counting the missing values and plotting them.

our dataset have some erorr and we can handle with erorrs.

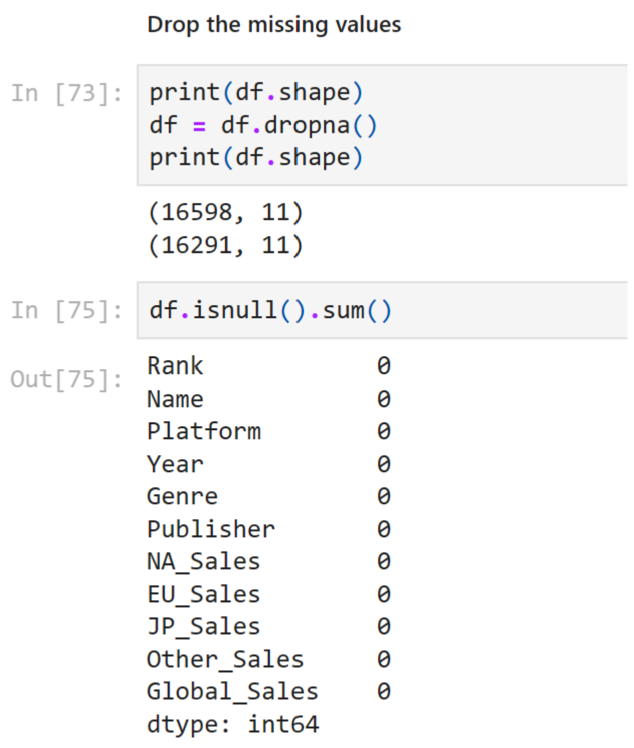
```
In [72]:  # Data Cleaning
          # Handle missing values
          df.isnull().sum()

Out[72]:  Rank              0
          Name              0
          Platform          0
          Year            271
          Genre             0
          Publisher        58
          NA_Sales          0
          EU_Sales          0
          JP_Sales          0
          Other_Sales       0
          Global_Sales      0
          dtype: int64
```

```
In [19]: plt.figure(figsize=(10, 6))
         sns.heatmap(df.isnull(), cmap='viridis', cbar=False)
         plt.title('Missing Value Heatmap')
         plt.show()
```



3.2Data cleaning:

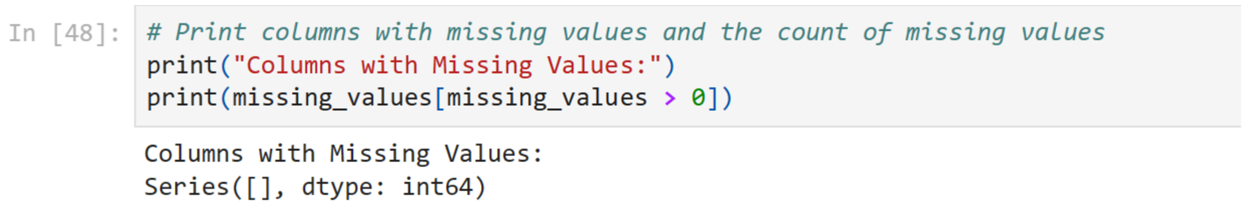
- Dropping rows with missing values using df.dropna(). And check after dropping:



Drop the missing values

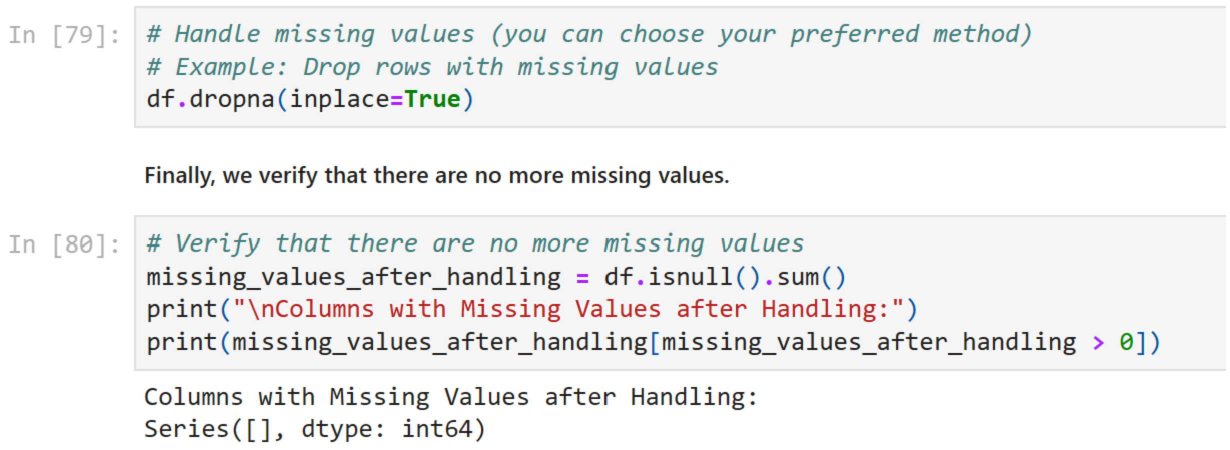
```
In [73]: print(df.shape)
         df = df.dropna()
         print(df.shape)

(16598, 11)
(16291, 11)

In [75]: df.isnull().sum()

Out[75]: 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
```

- Print clumnus with missing values and cunting of missing values:



```
In [48]: # Print columns with missing values and the count of missing values
         print("Columns with Missing Values:")
         print(missing_values[missing_values > 0])

Columns with Missing Values:
Series([], dtype: int64)
```

- We choose to handle missing values by dropping rows with missing values using data.dropna(inplace=True) and we verify that there are no more missing value.

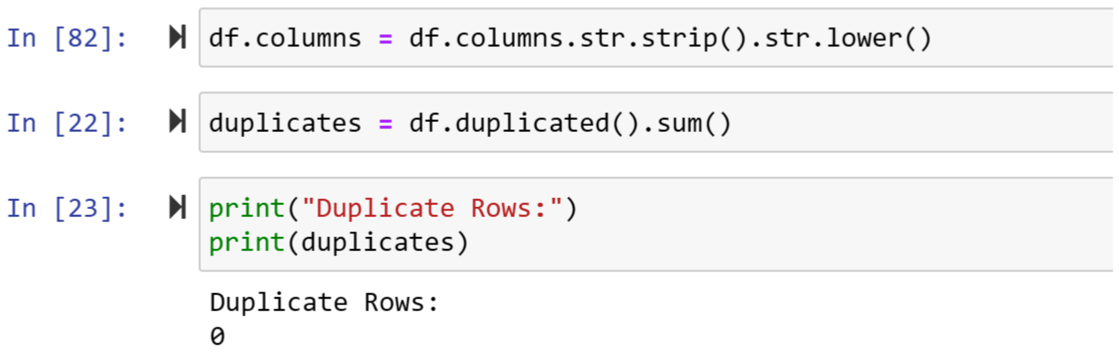


```
In [79]: # Handle missing values (you can choose your preferred method)
         # Example: Drop rows with missing values
         df.dropna(inplace=True)
```

Finally, we verify that there are no more missing values.

```
In [80]: # Verify that there are no more missing values
         missing_values_after_handling = df.isnull().sum()
         print("\nColumns with Missing Values after Handling:")
         print(missing_values_after_handling[missing_values_after_handling > 0])

         Columns with Missing Values after Handling:
         Series([], dtype: int64)
```

3.3Check the duplicates



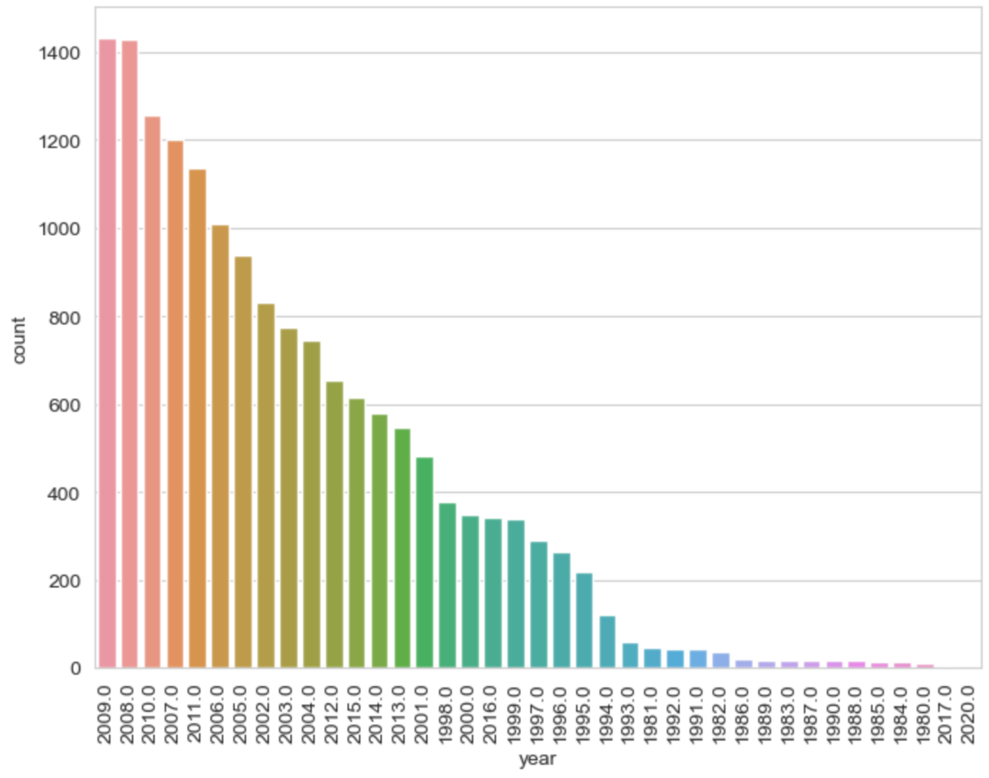
```
In [82]:  df.columns = df.columns.str.strip().str.lower()

In [22]:  duplicates = df.duplicated().sum()

In [23]:  print("Duplicate Rows:")
          print(duplicates)

          Duplicate Rows:
          0
```

As shown above, we don’t have duplicates .

4. Data Visualization

A bar chart showing global sales by platform

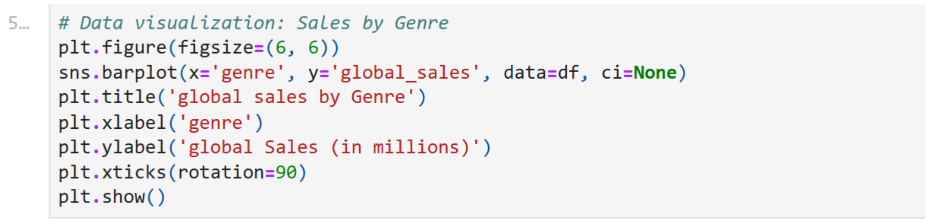
```
plt.figure(figsize=(8, 6))
sns.countplot(x="year", data=df, order = df.groupby(by=['year'])['name'].count().sort_values(ascending=False).index)
plt.xticks(rotation=90)
plt.show()
```



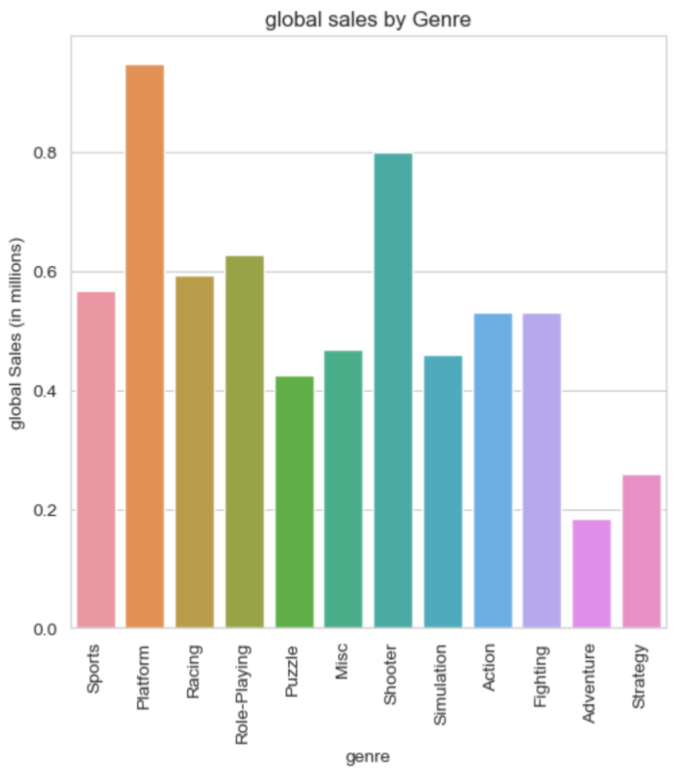
This line displays the plot on the screen.

The resulting figure is a bar plot (countplot) showing the number of video game releases per year. The years are displayed on the x-axis, and the count of game releases is shown on the y-axis. The bars are ordered in descending order of game releases, which means the year with the most game releases will appear first on the left. Rotating the x-axis labels by 90 degrees helps prevent label overlap and makes it easier to read the years.

A bar chart showing global sales by genre.

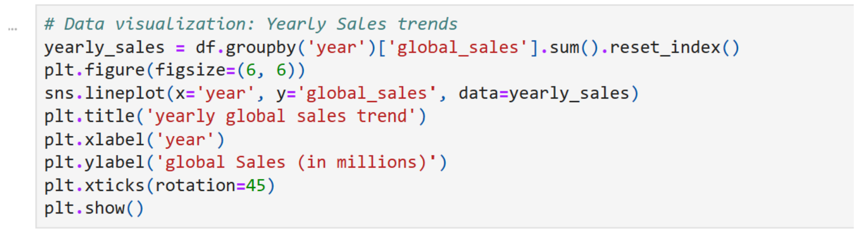


```
# Data visualization: Sales by Genre
plt.figure(figsize=(6, 6))
sns.barplot(x='genre', y='global_sales', data=df, ci=None)
plt.title('global sales by Genre')
plt.xlabel('genre')
plt.ylabel('global Sales (in millions)')
plt.xticks(rotation=90)
plt.show()
```

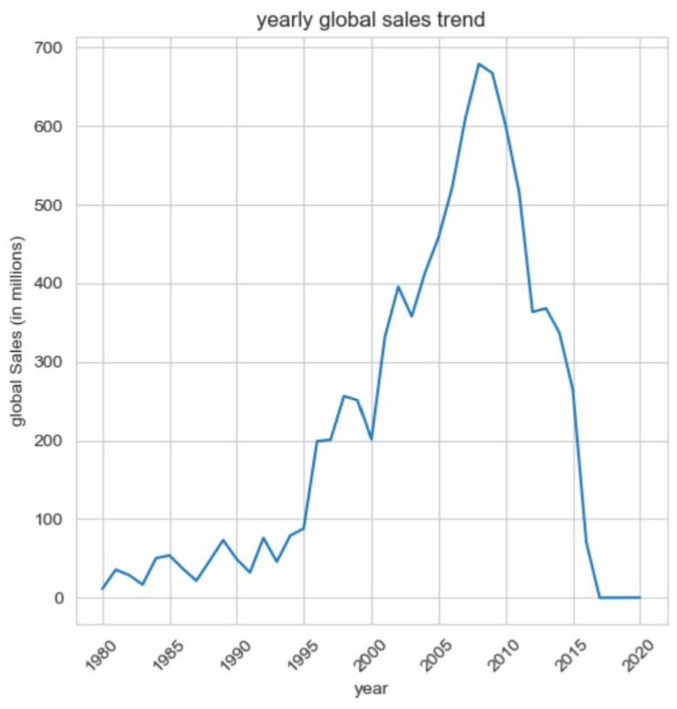


This line displays the plot on the screen.
The resulting figure is a bar chart that shows global sales by genre. Each genre is represented on the x-axis, and the global sales (in millions) are displayed on the y-axis. The bars represent the sales for each genre. The title, axis labels, and rotated x-axis labels make the chart more informative and readable.

4.1 A line plot showing yearly global sales trends.



```
# Data visualization: Yearly Sales trends
yearly_sales = df.groupby('year')['global_sales'].sum().reset_index()
plt.figure(figsize=(6, 6))
sns.lineplot(x='year', y='global_sales', data=yearly_sales)
plt.title('yearly global sales trend')
plt.xlabel('year')
plt.ylabel('global Sales (in millions)')
plt.xticks(rotation=45)
plt.show()
```



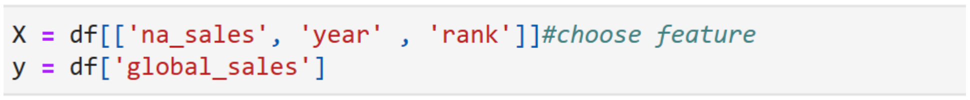
This line displays the plot on the screen.

The resulting figure is a line plot that shows the yearly global sales trend. Each point on the line represents the total global sales for a specific year, and the line connects these points to illustrate the trend over time. The title, axis labels, and rotated x-axis labels make the chart more informative and readable. This type of plot is useful for visualizing how global sales have changed over the years.

5. ML algorithm implementation

Implementing a machine learning algorithm typically involves several steps, including data preprocessing, model selection, training, evaluation, and prediction. Below, I'll provide a simplified example of implementing a machine learning algorithm using Python's scikit-learn library. We'll use a basic algorithm like random forest Regression for illustration.

- chooce feature



```
X = df[['na_sales', 'year' , 'rank']]#choose feature
y = df['global_sales']
```

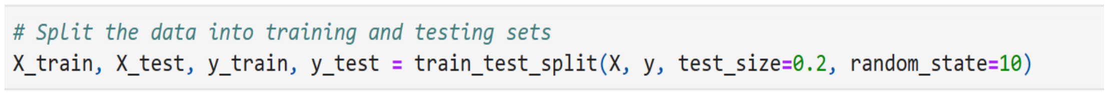
This line selects a subset of features from your dataset and assigns them to the variable X. The selected features are:

These features will be used as the input or independent variables for your machine learning model. They are the variables you believe are relevant for predicting the 'global_sales' target variable.

y = df['global_sales']: This line selects the target variable, which is 'global_sales', from your dataset and assigns it to the variable y. The 'global_sales' variable is what you want your machine learning model to predict. It represents the global sales of video games.

By selecting these specific features (input variables) and the 'global_sales' variable (target variable), you are preparing your data for a regression task. You can use this data to train a regression model to predict global sales based on the chosen features. The machine learning model will learn the relationships between the features and the target variable, allowing you to make predictions about global sales for new data points.

- Split the data into training and testing sets



```python
# 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=10)
```

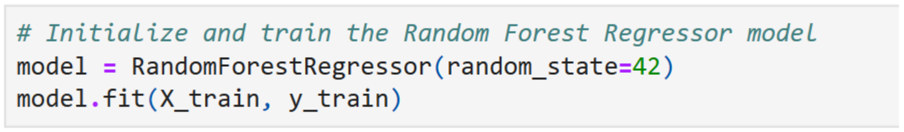
X_train, X_test, y_train, and y_test: These are variables that will store the training and testing data. Specifically:

test_size=0.2: This parameter specifies that 20% of the data will be reserved for testing, while the remaining 80% will be used for training. Adjusting this value allows you to control the size of your testing set.

random_state=10: This is an optional parameter that sets a random seed for the data splitting process. It ensures that the data split is reproducible. If you use the same random state value in the future, you will get the same split. This is useful for debugging and ensuring consistent results.

After running this code, you'll have four sets of data:

- nitializing and training a Random Forest Regressor mode



```python
# Initialize and train the Random Forest Regressor model
model = RandomForestRegressor(random_state=42)
model.fit(X_train, y_train)
```

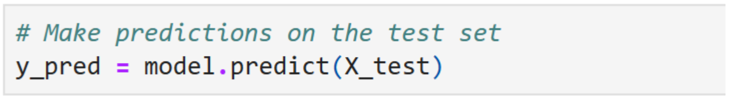
initializing and training a Random Forest Regressor model using the scikit-learn library.

The model you've created is now an instance of the Random Forest Regressor class with its specific configuration.

After initializing the model, you'll need to train it using your training data.

During the training process, the Random Forest Regressor model uses an ensemble of decision trees to learn the relationships between the features and the target variable (Global_Sales). The random forest ensemble method combines the predictions of multiple decision trees to make more accurate predictions.

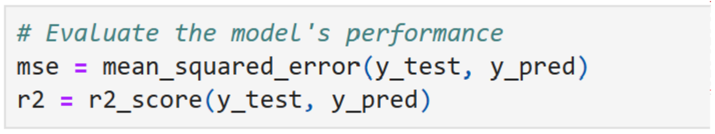
Once the model is trained, it's ready to make predictions on new data. In your case, it will predict global sales based on the features you provided during training.



```
# Make predictions on the test set
y_pred = model.predict(X_test)
```

After running this line of code, y_pred will contain the predicted global sales values for the test set, which you can then compare to the actual (true) global sales values from the test set to assess the model's accuracy and performance.

You can use various evaluation metrics, such as mean squared error (MSE), R-squared (R2), or others, to quantify how well the model's predictions align with the actual values and to gauge the model's predictive accuracy. These metrics will help you assess the model's performance and determine if it's suitable for your regression task.



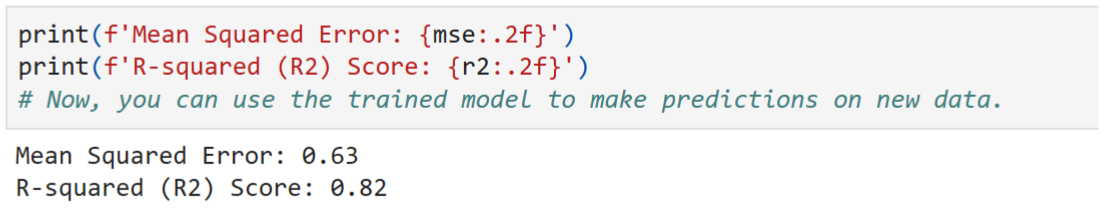
```
# Evaluate the model's performance
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)
```

After running these lines of code, you will have two metrics that provide information about your model's performance:

The MSE indicates how close the predicted values are to the actual values. Lower MSE values are desirable, as they suggest that the model's predictions are closer to the true values.

The R2 score provides an estimate of how well the model explains the variance in the target variable. An R2 score close to 1 suggests that the model is a good fit for the data, while a score closer to 0 indicates a weaker fit.

By comparing these metrics to baseline or other models, you can assess how well your Random Forest Regressor model performs and determine whether it meets your predictive needs.



```python
print(f'Mean Squared Error: {mse:.2f}')
print(f'R-squared (R2) Score: {r2:.2f}')
# Now, you can use the trained model to make predictions on new data.
```

```
Mean Squared Error: 0.63
R-squared (R2) Score: 0.82
```

The output of these lines will display the calculated MSE and R2 score in a more human-readable format, making it easier to understand the model's performance. The output might look something like this:

The printed values provide insights into how well the Random Forest Regressor model is performing for your regression task. A lower MSE and a higher R2 score typically indicate better model performance. With these metrics, you can assess the model's predictive accuracy and determine its suitability for making predictions on new data or for the specific task at hand.

- conclusion

In this conversation, you learned how to build and use a Random Forest Regressor model for predicting video game global sales based on selected features. Here's a summary of the key steps and concepts covered:

1. **Data Loading:** You loaded your dataset from a CSV file using the pandas library.

2. **Feature Selection:** You selected specific features from the dataset to use as input variables for your model and chose 'Global_Sales' as the target variable to predict.

3. **Data Splitting:** You divided your dataset into training and testing sets to train the model on one portion of the data and evaluate its performance on another.

4. **Model Training:** You initialized and trained a Random Forest Regressor model using the training data, allowing it to learn the relationships between the selected features and the target variable.

5. **Model Prediction:** You used the trained model to make predictions on the testing data, resulting in a set of predicted values for 'Global_Sales.'

6. **Model Evaluation:** You evaluated the model's performance using metrics such as Mean Squared Error (MSE) and R-squared (R2) to assess how well the model's predictions aligned with the actual values.

7. **Using the Model for Predictions:** To use the model for making predictions on new data, you prepared the new data in the same format as the training data, used the `model.predict()` method, and stored or used the predictions as needed for your specific application.

The steps you followed represent a typical workflow for building and using a machine learning model for regression tasks. By selecting relevant features, training the model, and evaluating its performance, you can gain valuable insights and make predictions for various applications.