Name: Chibuonu Ezennabike

Student Number: 202315974

In [88]:
```
# import the libraries
import csv
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
import warnings
warnings.simplefilter(action='ignore', category=FutureWarning)
```

In [6]:
```
Video_Games_dataset = pd.read_csv("Video_Games.csv")
Video_Games_dataset
```

4	Red/Pokemon Blue	GB	1996.0	Role-Playing	Nintendo	11.27	8.89
...	...	...	...	...	...	...	...
16714	Samurai Warriors: Sanada Maru	PS3	2016.0	Action	Tecmo Koei	0.00	0.00
16715	LMA Manager 2007	X360	2006.0	Sports	Codemasters	0.00	0.01
16716	Haitaka no Psychedelica	PSV	2016.0	Adventure	Idea Factory	0.00	0.00
16717	Spirits & Spells	GBA	2003.0	Platform	Wanadoo	0.01	0.00
16718	Winning Post 8 2016	PSV	2016.0	Simulation	Tecmo Koei	0.00	0.00

16719 rows × 16 columns

In [7]:
```
# print the first 5 rows of the data
Video_Games_dataset.head(5)
```

Out[7]:

	Name	Platform	Year_of_Release	Genre	Publisher	NA_Sales	EU_Sales	JP_Sales	Otl
0	Wii Sports	Wii	2006.0	Sports	Nintendo	41.36	28.96	3.77	
1	Super Mario Bros.	NES	1985.0	Platform	Nintendo	29.08	3.58	6.81	
2	Mario Kart Wii	Wii	2008.0	Racing	Nintendo	15.68	12.76	3.79	
3	Wii Sports Resort	Wii	2009.0	Sports	Nintendo	15.61	10.93	3.28	
4	Pokemon Red/Pokemon Blue	GB	1996.0	Role-Playing	Nintendo	11.27	8.89	10.22	

In [8]:
```
# check for missing values
Video_Games_dataset.isna().sum()
```

Out[8]:
```
Name                 2
Platform             0
Year_of_Release    269
Genre                2
Publisher           54
NA_Sales             0
EU_Sales             0
JP_Sales             0
Other_Sales          0
Global_Sales         0
Critic_Score      8582
Critic_Count      8582
User_Score        6704
User_Count        9129
Developer         6623
Rating            6769
dtype: int64
```

DATA CLEANING and PREPROCESSING

In [9]:
```
# Check the data types
Video_Games_dataset.info()
```

```
 #   Column           Non-Null Count  Dtype
---  ------           --------------  -----
 0   Name             16717 non-null  object
 1   Platform         16719 non-null  object
 2   Year_of_Release  16450 non-null  float64
 3   Genre            16717 non-null  object
 4   Publisher        16665 non-null  object
 5   NA_Sales         16719 non-null  float64
 6   EU_Sales         16719 non-null  float64
 7   JP_Sales         16719 non-null  float64
 8   Other_Sales      16719 non-null  float64
 9   Global_Sales     16719 non-null  float64
 10  Critic_Score     8137 non-null   float64
 11  Critic_Count     8137 non-null   float64
 12  User_Score       10015 non-null  object
 13  User_Count       7590 non-null   float64
 14  Developer        10096 non-null  object
 15  Rating           9950 non-null   object
dtypes: float64(9), object(7)
memory usage: 2.0+ MB
```

In [10]:
```
Video_Games_dataset.describe()
```

Out[10]:

	Year_of_Release	NA_Sales	EU_Sales	JP_Sales	Other_Sales	Global_Sales	Cri
count	16450.000000	16719.000000	16719.000000	16719.000000	16719.000000	16719.000000	81
mean	2006.487356	0.263330	0.145025	0.077602	0.047332	0.533543	
std	5.878995	0.813514	0.503283	0.308818	0.186710	1.547935	
min	1980.000000	0.000000	0.000000	0.000000	0.000000	0.010000	
25%	2003.000000	0.000000	0.000000	0.000000	0.000000	0.060000	
50%	2007.000000	0.080000	0.020000	0.000000	0.010000	0.170000	
75%	2010.000000	0.240000	0.110000	0.040000	0.030000	0.470000	
max	2020.000000	41.360000	28.960000	10.220000	10.570000	82.530000	

Name Cleaning

In [11]:
```
#Replacing the missing names with Anonymous
Video_Games_dataset.loc[Video_Games_dataset["Name"].isna()]
```

Out[11]:

	Name	Platform	Year_of_Release	Genre	Publisher	NA_Sales	EU_Sales	JP_Sales	Oth
659	NaN	GEN	1993.0	NaN	Acclaim Entertainment	1.78	0.53	0.00	
14246	NaN	GEN	1993.0	NaN	Acclaim Entertainment	0.00	0.00	0.03	

In [12]:
```
Video_Games_dataset["Name"].replace(np.nan, "Anonymous" , inplace=True)
```

In [13]:
```
Video_Games_dataset.loc[Video_Games_dataset["Name"].isna()]
```

Out[13]:

	Name	Platform	Year_of_Release	Genre	Publisher	NA_Sales	EU_Sales	JP_Sales	Other_Sales

Year of Release Cleaning

In [14]:
```
Video_Games_dataset.loc[Video_Games_dataset["Year_of_Release"].isna()]
```

Out[14]:

	Name	Platform	Year_of_Release	Genre	Publisher	NA_Sales	EU_Sales	JP_
183	Madden NFL 2004	PS2	NaN	Sports	Electronic Arts	4.26	0.26	
377	FIFA Soccer 2004	PS2	NaN	Sports	Electronic Arts	0.59	2.36	
456	LEGO Batman: The Videogame	Wii	NaN	Action	Warner Bros. Interactive Entertainment	1.80	0.97	
475	wwe Smackdown vs. Raw 2006	PS2	NaN	Fighting	NaN	1.57	1.02	
609	Space Invaders	2600	NaN	Shooter	Atari	2.36	0.14	
...	...	...	...	...	...	...	...	
16376	PDC World Championship Darts 2008	PSP	NaN	Sports	Oxygen Interactive	0.01	0.00	
16409	Freaky Flyers	GC	NaN	Racing	Unknown	0.01	0.00	
16452	Inversion	PC	NaN	Shooter	Namco Bandai Games	0.01	0.00	
16462	Hakuouki: Shinsengumi Kitan	PS3	NaN	Adventure	Unknown	0.01	0.00	
16526	Virtua Quest	GC	NaN	Role-Playing	Unknown	0.01	0.00	

269 rows × 16 columns

In [15]:
```
# Using the highest occurring year to replace the missing years
Video_Games_dataset["Year_of_Release"].replace(np.nan, "2020" , inplace=True)
```

In [16]:
```
Video_Games_dataset.loc[Video_Games_dataset["Year_of_Release"].isna()]
```

Out[16]:

	Name	Platform	Year_of_Release	Genre	Publisher	NA_Sales	EU_Sales	JP_Sales	Other_Sales

Publish Cleaning

In [17]:
```
Video_Games_dataset.loc[Video_Games_dataset["Publisher"].isna()]
```

6626	Network Collection: Game Boy Advance V...	GBA	2004.0	Misc	NaN	0.18	0.07
6831	All Grown Up!: Game Boy Advance Video Volume 1	GBA	2004.0	Misc	NaN	0.17	0.06
7189	Nicktoons Collection: Game Boy Advance Video V...	GBA	2004.0	Misc	NaN	0.16	0.06
7333	Yu Yu Hakusho: Dark Tournament	PS2	2020	Fighting	NaN	0.10	0.08
	SpongeBob						

In [18]:
```
Video_Games_dataset["Publisher"].replace(np.nan, "Unknown" , inplace=True)
```

In [19]:
```
Video_Games_dataset.loc[Video_Games_dataset["Publisher"].isna()]
```

Out[19]:

	Name	Platform	Year_of_Release	Genre	Publisher	NA_Sales	EU_Sales	JP_Sales	Other_Sales

Critic Score Cleaning

In [20]:
```
Video_Games_dataset["Critic_Score"].describe()
```

Out[20]:
```
count    8137.000000
mean       68.967679
std        13.938165
min        13.000000
25%        60.000000
50%        71.000000
75%        79.000000
max        98.000000
Name: Critic_Score, dtype: float64
```

In [21]:
```
Video_Games_dataset.loc[Video_Games_dataset["Critic_Score"].isna()]
```

Out[21]:

	Name	Platform	Year_of_Release	Genre	Publisher	NA_Sales	EU_Sales	JP_S
1	Super Mario Bros.	NES	1985.0	Platform	Nintendo	29.08	3.58	
4	Pokemon Red/Pokemon Blue	GB	1996.0	Role-Playing	Nintendo	11.27	8.89	
5	Tetris	GB	1989.0	Puzzle	Nintendo	23.20	2.26	
9	Duck Hunt	NES	1984.0	Shooter	Nintendo	26.93	0.63	
10	Nintendogs	DS	2005.0	Simulation	Nintendo	9.05	10.95	
...	...	...	...	...	...	...	...	
16714	Samurai Warriors: Sanada Maru	PS3	2016.0	Action	Tecmo Koei	0.00	0.00	
16715	LMA Manager 2007	X360	2006.0	Sports	Codemasters	0.00	0.01	
16716	Haitaka no Psychedelica	PSV	2016.0	Adventure	Idea Factory	0.00	0.00	
16717	Spirits & Spells	GBA	2003.0	Platform	Wanadoo	0.01	0.00	
16718	Winning Post 8 2016	PSV	2016.0	Simulation	Tecmo Koei	0.00	0.00	

8582 rows × 16 columns

In [22]:
```
# Replacing the Nan values with 0 because there are no zeros in the raw data
Video_Games_dataset["Critic_Score"].replace(np.nan, "0" , inplace=True)
```

In [23]:
```
Video_Games_dataset.loc[Video_Games_dataset["Critic_Score"].isna()]
```

Out[23]:

	Name	Platform	Year_of_Release	Genre	Publisher	NA_Sales	EU_Sales	JP_Sales	Other_Sales

In [24]:
```
Video_Games_dataset["Critic_Score"] = Video_Games_dataset["Critic_Score"].astype
```

Critic Count Cleaning

In [25]:
```
Video_Games_dataset["Critic_Count"].describe()
```

Out[25]:
```
count    8137.000000
mean       26.360821
std        18.980495
min         3.000000
25%        12.000000
50%        21.000000
75%        36.000000
max       113.000000
Name: Critic_Count, dtype: float64
```

In [26]:
```
Video_Games_dataset.loc[Video_Games_dataset["Critic_Count"].isna()]
```

Out[26]:

	Name	Platform	Year_of_Release	Genre	Publisher	NA_Sales	EU_Sales	JP_S
1	Super Mario Bros.	NES	1985.0	Platform	Nintendo	29.08	3.58	
4	Pokemon Red/Pokemon Blue	GB	1996.0	Role-Playing	Nintendo	11.27	8.89	
5	Tetris	GB	1989.0	Puzzle	Nintendo	23.20	2.26	
9	Duck Hunt	NES	1984.0	Shooter	Nintendo	26.93	0.63	
10	Nintendogs	DS	2005.0	Simulation	Nintendo	9.05	10.95	
...	...	...	...	...	...	...	...	
16714	Samurai Warriors: Sanada Maru	PS3	2016.0	Action	Tecmo Koei	0.00	0.00	
16715	LMA Manager 2007	X360	2006.0	Sports	Codemasters	0.00	0.01	
16716	Haitaka no Psychedelica	PSV	2016.0	Adventure	Idea Factory	0.00	0.00	
16717	Spirits & Spells	GBA	2003.0	Platform	Wanadoo	0.01	0.00	
16718	Winning Post 8 2016	PSV	2016.0	Simulation	Tecmo Koei	0.00	0.00	

8582 rows × 16 columns

In [27]:
```
# Replacing the missing values with 0 becuase there is no zero in the raw data
Video_Games_dataset["Critic_Count"].replace(np.nan, "0" , inplace=True)
```

In [28]:
```
Video_Games_dataset.loc[Video_Games_dataset["Critic_Count"].isna()]
```

Out[28]:

Name	Platform	Year_of_Release	Genre	Publisher	NA_Sales	EU_Sales	JP_Sales	Other_Sales

User Score Cleaning

In [29]:
```
Video_Games_dataset["User_Score"].describe()
```

Out[29]:
```
count     10015
unique       96
top         tbd
freq        2425
Name: User_Score, dtype: object
```

In [30]:
```
Video_Games_dataset.loc[(Video_Games_dataset["User_Score"] == "tbd")]
```

Out[30]:

	Name	Platform	Year_of_Release	Genre	Publisher	NA_Sales	EU_Sales	JP_Sa
119	Zumba Fitness	Wii	2010.0	Sports	505 Games	3.45	2.59	
301	Namco Museum: 50th Anniversary	PS2	2005.0	Misc	Namco Bandai Games	2.08	1.35	
520	Zumba Fitness 2	Wii	2011.0	Sports	Majesco Entertainment	1.51	1.03	
645	uDraw Studio	Wii	2010.0	Misc	THQ	1.65	0.57	
657	Frogger's Adventures: Temple of the Frog	GBA	2020	Adventure	Konami Digital Entertainment	2.15	0.18	
...	...	...	...	...	...	...	...	
16699	Planet Monsters	GBA	2001.0	Action	Titus	0.01	0.00	
16701	Bust-A-Move 3000	GC	2003.0	Puzzle	Ubisoft	0.01	0.00	
16702	Mega Brain Boost	DS	2008.0	Puzzle	Majesco Entertainment	0.01	0.00	
16708	Plushees	DS	2008.0	Simulation	Destineer	0.01	0.00	
16710	Men in Black II: Alien Escape	GC	2003.0	Shooter	Infogrames	0.01	0.00	

2425 rows × 16 columns

In [31]:
```
Video_Games_dataset["User_Score"].replace("tbd", np.nan, inplace=True)
```

In [32]:
```
Video_Games_dataset.loc[(Video_Games_dataset["User_Score"] == "tbd")]
```

Out[32]:

	Name	Platform	Year_of_Release	Genre	Publisher	NA_Sales	EU_Sales	JP_Sales	Other_Sales

In [33]:
```
Video_Games_dataset.loc[Video_Games_dataset["User_Score"].isna()]
```

Out[33]:

	Name	Platform	Year_of_Release	Genre	Publisher	NA_Sales	EU_Sales	JP_S
1	Super Mario Bros.	NES	1985.0	Platform	Nintendo	29.08	3.58	
4	Pokemon Red/Pokemon Blue	GB	1996.0	Role-Playing	Nintendo	11.27	8.89	
5	Tetris	GB	1989.0	Puzzle	Nintendo	23.20	2.26	
9	Duck Hunt	NES	1984.0	Shooter	Nintendo	26.93	0.63	
10	Nintendogs	DS	2005.0	Simulation	Nintendo	9.05	10.95	
...	...	...	...	...	...	...	...	
16714	Samurai Warriors: Sanada Maru	PS3	2016.0	Action	Tecmo Koei	0.00	0.00	
16715	LMA Manager 2007	X360	2006.0	Sports	Codemasters	0.00	0.01	
16716	Haitaka no Psychedelica	PSV	2016.0	Adventure	Idea Factory	0.00	0.00	
16717	Spirits & Spells	GBA	2003.0	Platform	Wanadoo	0.01	0.00	
16718	Winning Post 8 2016	PSV	2016.0	Simulation	Tecmo Koei	0.00	0.00	

9129 rows × 16 columns

In [34]:
```
Video_Games_dataset["User_Score"] = Video_Games_dataset["User_Score"].astype("fl
```

In [35]:
```
print(type(Video_Games_dataset["User_Score"]))
```

```
<class 'pandas.core.series.Series'>
```

In [36]:
```
Video_Games_dataset["User_Score"].describe()
```

Out[36]:
```
count    7590.000000
mean        7.125046
std         1.500006
min         0.000000
25%         6.400000
50%         7.500000
75%         8.200000
max         9.700000
Name: User_Score, dtype: float64
```

In [37]:
```
# using the median to replace the missing value.
Video_Games_dataset["User_Score"].replace(np.nan, "7.5",  inplace=True)
```

In [38]:
```
Video_Games_dataset.loc[Video_Games_dataset["User_Score"].isna()]
```

Out[38]:

	Name	Platform	Year_of_Release	Genre	Publisher	NA_Sales	EU_Sales	JP_Sales	Other_Sales

User Count Cleaning

In [39]:
```
Video_Games_dataset.loc[Video_Games_dataset["User_Count"].isna()]
```

Out[39]:

	Name	Platform	Year_of_Release	Genre	Publisher	NA_Sales	EU_Sales	JP_S
1	Super Mario Bros.	NES	1985.0	Platform	Nintendo	29.08	3.58	
4	Pokemon Red/Pokemon Blue	GB	1996.0	Role-Playing	Nintendo	11.27	8.89	
5	Tetris	GB	1989.0	Puzzle	Nintendo	23.20	2.26	
9	Duck Hunt	NES	1984.0	Shooter	Nintendo	26.93	0.63	
10	Nintendogs	DS	2005.0	Simulation	Nintendo	9.05	10.95	
...	...	...	...	...	...	...	...	
16714	Samurai Warriors: Sanada Maru	PS3	2016.0	Action	Tecmo Koei	0.00	0.00	
16715	LMA Manager 2007	X360	2006.0	Sports	Codemasters	0.00	0.01	
16716	Haitaka no Psychedelica	PSV	2016.0	Adventure	Idea Factory	0.00	0.00	
16717	Spirits & Spells	GBA	2003.0	Platform	Wanadoo	0.01	0.00	
16718	Winning Post 8 2016	PSV	2016.0	Simulation	Tecmo Koei	0.00	0.00	

9129 rows × 16 columns

In [40]:
```
Video_Games_dataset["User_Count"].replace(np.nan, "0" , inplace=True)
```

In [41]:
```
Video_Games_dataset.loc[Video_Games_dataset["User_Count"].isna()]
```

Out[41]:

Name	Platform	Year_of_Release	Genre	Publisher	NA_Sales	EU_Sales	JP_Sales	Other_Sales

Developer Cleaning

In [42]:
```
Video_Games_dataset.loc[Video_Games_dataset["Developer"].isna()]
```

Out[42]:

	Name	Platform	Year_of_Release	Genre	Publisher	NA_Sales	EU_Sales	JP_S
1	Super Mario Bros.	NES	1985.0	Platform	Nintendo	29.08	3.58	
4	Pokemon Red/Pokemon Blue	GB	1996.0	Role-Playing	Nintendo	11.27	8.89	
5	Tetris	GB	1989.0	Puzzle	Nintendo	23.20	2.26	
9	Duck Hunt	NES	1984.0	Shooter	Nintendo	26.93	0.63	
10	Nintendogs	DS	2005.0	Simulation	Nintendo	9.05	10.95	
...	...	...	...	...	...	...	...	
16714	Samurai Warriors: Sanada Maru	PS3	2016.0	Action	Tecmo Koei	0.00	0.00	
16715	LMA Manager 2007	X360	2006.0	Sports	Codemasters	0.00	0.01	
16716	Haitaka no Psychedelica	PSV	2016.0	Adventure	Idea Factory	0.00	0.00	
16717	Spirits & Spells	GBA	2003.0	Platform	Wanadoo	0.01	0.00	
16718	Winning Post 8 2016	PSV	2016.0	Simulation	Tecmo Koei	0.00	0.00	

6623 rows × 16 columns

In [43]:
```
Video_Games_dataset["Developer"].replace(np.nan, "Unknown" , inplace=True)
```

In [44]:
```
Video_Games_dataset.loc[Video_Games_dataset["Developer"].isna()]
```

Out[44]:

	Name	Platform	Year_of_Release	Genre	Publisher	NA_Sales	EU_Sales	JP_Sales	Other_Sales

Rating Cleaning

In [45]:
```
Video_Games_dataset.loc[Video_Games_dataset["Rating"].isna()]
```

Out[45]:

	Name	Platform	Year_of_Release	Genre	Publisher	NA_Sales	EU_Sales	JP_S
1	Super Mario Bros.	NES	1985.0	Platform	Nintendo	29.08	3.58	
4	Pokemon Red/Pokemon Blue	GB	1996.0	Role-Playing	Nintendo	11.27	8.89	
5	Tetris	GB	1989.0	Puzzle	Nintendo	23.20	2.26	
9	Duck Hunt	NES	1984.0	Shooter	Nintendo	26.93	0.63	
10	Nintendogs	DS	2005.0	Simulation	Nintendo	9.05	10.95	
...	...	...	...	...	...	...	...	
16714	Samurai Warriors: Sanada Maru	PS3	2016.0	Action	Tecmo Koei	0.00	0.00	
16715	LMA Manager 2007	X360	2006.0	Sports	Codemasters	0.00	0.01	
16716	Haitaka no Psychedelica	PSV	2016.0	Adventure	Idea Factory	0.00	0.00	
16717	Spirits & Spells	GBA	2003.0	Platform	Wanadoo	0.01	0.00	
16718	Winning Post 8 2016	PSV	2016.0	Simulation	Tecmo Koei	0.00	0.00	

6769 rows × 16 columns

In [46]:
```
Video_Games_dataset["Rating"].describe()
```

Out[46]:
```
count     9950
unique       8
top          E
freq      3991
Name: Rating, dtype: object
```

In [47]:
```
Video_Games_dataset["Rating"].replace(np.nan, "Unknown" , inplace=True)
```

In [48]:
```
Video_Games_dataset.loc[Video_Games_dataset["Rating"].isna()]
```

Out[48]:

Name	Platform	Year_of_Release	Genre	Publisher	NA_Sales	EU_Sales	JP_Sales	Other_Sales

In [49]:
```
Video_Games_dataset_object = Video_Games_dataset.select_dtypes(exclude = "object
```

In [50]:
```
Video_Games_dataset_object
```

Out[50]:

	NA_Sales	EU_Sales	JP_Sales	Other_Sales	Global_Sales	Critic_Score
0	41.36	28.96	3.77	8.45	82.53	76.0
1	29.08	3.58	6.81	0.77	40.24	0.0
2	15.68	12.76	3.79	3.29	35.52	82.0
3	15.61	10.93	3.28	2.95	32.77	80.0
4	11.27	8.89	10.22	1.00	31.37	0.0
...	...	...	...	...	...	...
16714	0.00	0.00	0.01	0.00	0.01	0.0
16715	0.00	0.01	0.00	0.00	0.01	0.0
16716	0.00	0.00	0.01	0.00	0.01	0.0
16717	0.01	0.00	0.00	0.00	0.01	0.0
16718	0.00	0.00	0.01	0.00	0.01	0.0

16719 rows × 6 columns

In [51]:
```
# changing columns from float to int
Video_Games_dataset['Year_of_Release'] = Video_Games_dataset['Year_of_Release'].
Video_Games_dataset['NA_Sales'] = Video_Games_dataset['NA_Sales'].astype(int)
Video_Games_dataset['EU_Sales'] = Video_Games_dataset['EU_Sales'].astype(int)
Video_Games_dataset['JP_Sales'] = Video_Games_dataset['JP_Sales'].astype(int)
Video_Games_dataset['Other_Sales'] = Video_Games_dataset['Other_Sales'].astype(i
Video_Games_dataset['Global_Sales'] = Video_Games_dataset['Global_Sales'].astype
Video_Games_dataset['Critic_Score'] = Video_Games_dataset['Critic_Score'].astype
Video_Games_dataset['Critic_Count'] = Video_Games_dataset['Critic_Count'].astype
Video_Games_dataset['User_Count'] = Video_Games_dataset['User_Count'].astype(int
```

In [52]:
```
Video_Games_dataset.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 16719 entries, 0 to 16718
Data columns (total 16 columns):
 #   Column           Non-Null Count  Dtype 
---  ------           --------------  ----- 
 0   Name             16719 non-null  object
 1   Platform         16719 non-null  object
 2   Year_of_Release  16719 non-null  int32 
 3   Genre            16717 non-null  object
 4   Publisher        16719 non-null  object
 5   NA_Sales         16719 non-null  int32 
 6   EU_Sales         16719 non-null  int32 
 7   JP_Sales         16719 non-null  int32 
 8   Other_Sales      16719 non-null  int32 
 9   Global_Sales     16719 non-null  int32 
 10  Critic_Score     16719 non-null  int32 
 11  Critic_Count     16719 non-null  int32 
 12  User_Score       16719 non-null  object
 13  User_Count       16719 non-null  int32 
 14  Developer        16719 non-null  object
 15  Rating           16719 non-null  object
dtypes: int32(9), object(7)
memory usage: 1.5+ MB
```

In [53]:
```
Video_Games_dataset.describe().transpose()
```

Out[53]:

	count	mean	std	min	25%	50%	75%	max
Year_of_Release	16719.0	2006.704767	6.074303	1980.0	2003.0	2008.0	2011.0	2020.0
NA_Sales	16719.0	0.106226	0.750617	0.0	0.0	0.0	0.0	41.0
EU_Sales	16719.0	0.050302	0.440142	0.0	0.0	0.0	0.0	28.0
JP_Sales	16719.0	0.024403	0.250379	0.0	0.0	0.0	0.0	10.0
Other_Sales	16719.0	0.006878	0.142653	0.0	0.0	0.0	0.0	10.0
Global_Sales	16719.0	0.292123	1.493682	0.0	0.0	0.0	0.0	82.0
Critic_Score	16719.0	33.566003	35.817714	0.0	0.0	0.0	70.0	98.0
Critic_Count	16719.0	12.829595	18.679793	0.0	0.0	0.0	21.0	113.0
User_Count	16719.0	73.648245	386.695153	0.0	0.0	0.0	20.0	10665.0

In [54]:
```
Video_Games_dataset.Platform.unique()
```

Out[54]:
```
array(['Wii', 'NES', 'GB', 'DS', 'X360', 'PS3', 'PS2', 'SNES', 'GBA',
       'PS4', '3DS', 'N64', 'PS', 'XB', 'PC', '2600', 'PSP', 'XOne',
       'WiiU', 'GC', 'GEN', 'DC', 'PSV', 'SAT', 'SCD', 'WS', 'NG', 'TG16',
       '3DO', 'GG', 'PCFX'], dtype=object)
```

To avoid over fitting I will use only Platform with 1000 values and above

In [55]:
```
Video_Games_dataset["Platform"].value_counts()
```

Out[55]:
```
PS2     2161
DS      2152
PS3     1331
Wii     1320
X360    1262
PSP     1209
PS      1197
PC       974
XB       824
GBA      822
GC       556
3DS      520
PSV      432
PS4      393
N64      319
XOne     247
SNES     239
SAT      173
WiiU     147
2600     133
NES       98
GB        98
DC        52
GEN       29
NG        12
SCD        6
WS         6
3DO        3
TG16       2
GG         1
PCFX       1
Name: Platform, dtype: int64
```

In [56]:
```
# Filter games platforms with counts >= 1000
Highest_Platform = Video_Games_dataset["Platform"].value_counts()[Video_Games_da

# Print the lists of the platforms
print(Highest_Platform)
```

```
['PS2', 'DS', 'PS3', 'Wii', 'X360', 'PSP', 'PS']
```

In [57]:
```
Lower_Platform = []
for items in Video_Games_dataset["Platform"]:
        if items in Highest_Platform:
            Lower_Platform.append(items)
else:
    Lower_Platform.append('lower_platforms')
```

In [58]:
```
#Rename platforms with counts below 1000 as 'lower_platforms'
Platform_lower = []
for items in range(Video_Games_dataset.shape[0]): #loop through Video_Games_data
    Platform_Number = Video_Games_dataset.Platform.values[items]
    flag=0
    for val in Highest_Platform:#loop through highest platforms list
        if val in Platform_Number and flag==0:
            Platform_lower.append(val)
            flag=1
    if flag==0:
        Platform_lower.append('low_platforms')
```

In [59]:
```
Video_Games_dataset['Best_Platform']=Platform_lower # create new column with the
Video_Games_dataset['Best_Platform'].value_counts()
```

Out[59]:
```
low_platforms    4595
DS               2672
PS2              2161
PS               2022
Wii              1467
PS3              1331
X360             1262
PSP              1209
Name: Best_Platform, dtype: int64
```

In [60]:
```
# check for duplicates
Video_Games_dataset.duplicated().sum()
```

Out[60]:
```
0
```

In [61]:
```
Video_Games_dataset.select_dtypes(object).describe()
```

Out[61]:

	Name	Platform	Genre	Publisher	User_Score	Developer	Rating	Best_Platform
count	16719	16719	16717	16719	16719	16719	16719	16719
unique	11563	31	12	581	96	1697	9	8
top	Need for Speed: Most Wanted	PS2	Action	Electronic Arts	7.5	Unknown	Unknown	low_platforms
freq	12	2161	3370	1356	9129	6623	6769	4595

In [62]:
```
Video_Games_dataset_object = Video_Games_dataset.select_dtypes(include = "object
Video_Games_dataset_object
```

Out[62]:

	Name	Platform	Genre	Publisher	User_Score	Developer	Rating	Best_Plat
0	Wii Sports	Wii	Sports	Nintendo	8.0	Nintendo	E	
1	Super Mario Bros.	NES	Platform	Nintendo	7.5	Unknown	Unknown	low_platf
2	Mario Kart Wii	Wii	Racing	Nintendo	8.3	Nintendo	E	
3	Wii Sports Resort	Wii	Sports	Nintendo	8.0	Nintendo	E	
4	Pokemon Red/Pokemon Blue	GB	Role-Playing	Nintendo	7.5	Unknown	Unknown	low_platf
...	...	...	...	...	...	...	...	
16714	Samurai Warriors: Sanada Maru	PS3	Action	Tecmo Koei	7.5	Unknown	Unknown	
16715	LMA Manager 2007	X360	Sports	Codemasters	7.5	Unknown	Unknown	
16716	Haitaka no Psychedelica	PSV	Adventure	Idea Factory	7.5	Unknown	Unknown	
16717	Spirits & Spells	GBA	Platform	Wanadoo	7.5	Unknown	Unknown	low_platf
16718	Winning Post 8 2016	PSV	Simulation	Tecmo Koei	7.5	Unknown	Unknown	

16719 rows × 8 columns

In [63]:
```
Video_Games_dataset_numeric = Video_Games_dataset.select_dtypes(exclude = "objec
Video_Games_dataset_numeric
```

Out[63]:

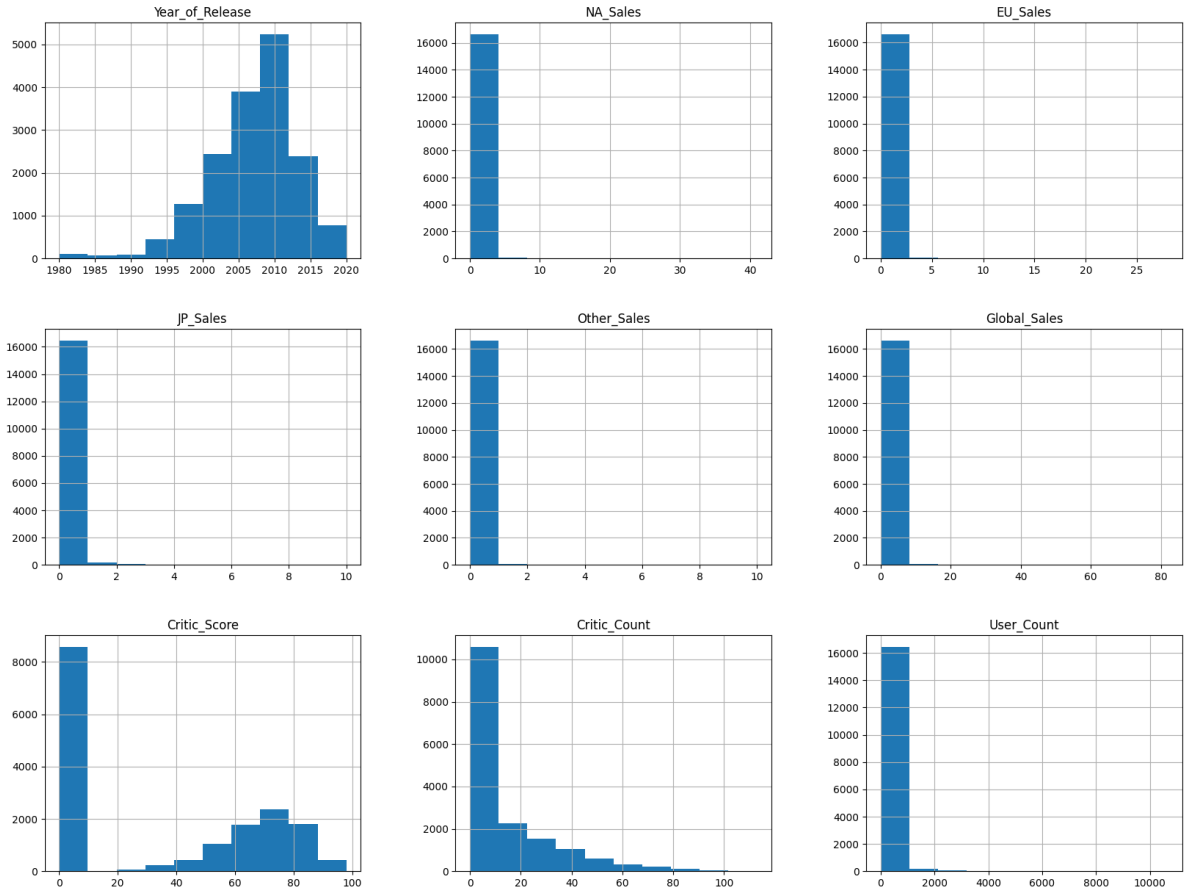
	Year_of_Release	NA_Sales	EU_Sales	JP_Sales	Other_Sales	Global_Sales	Critic_Score	C
0	2006	41	28	3	8	82	76	
1	1985	29	3	6	0	40	0	
2	2008	15	12	3	3	35	82	
3	2009	15	10	3	2	32	80	
4	1996	11	8	10	1	31	0	
...	...	...	...	...	...	...	...	...
16714	2016	0	0	0	0	0	0	
16715	2006	0	0	0	0	0	0	
16716	2016	0	0	0	0	0	0	
16717	2003	0	0	0	0	0	0	
16718	2016	0	0	0	0	0	0	

16719 rows × 9 columns

DATA VISUALIZATION

In [63]:
```
# display all the numerical datatypes
plt.figure(dpi=120)
Video_Games_dataset.hist(figsize=(20,15))
plt.show()
```

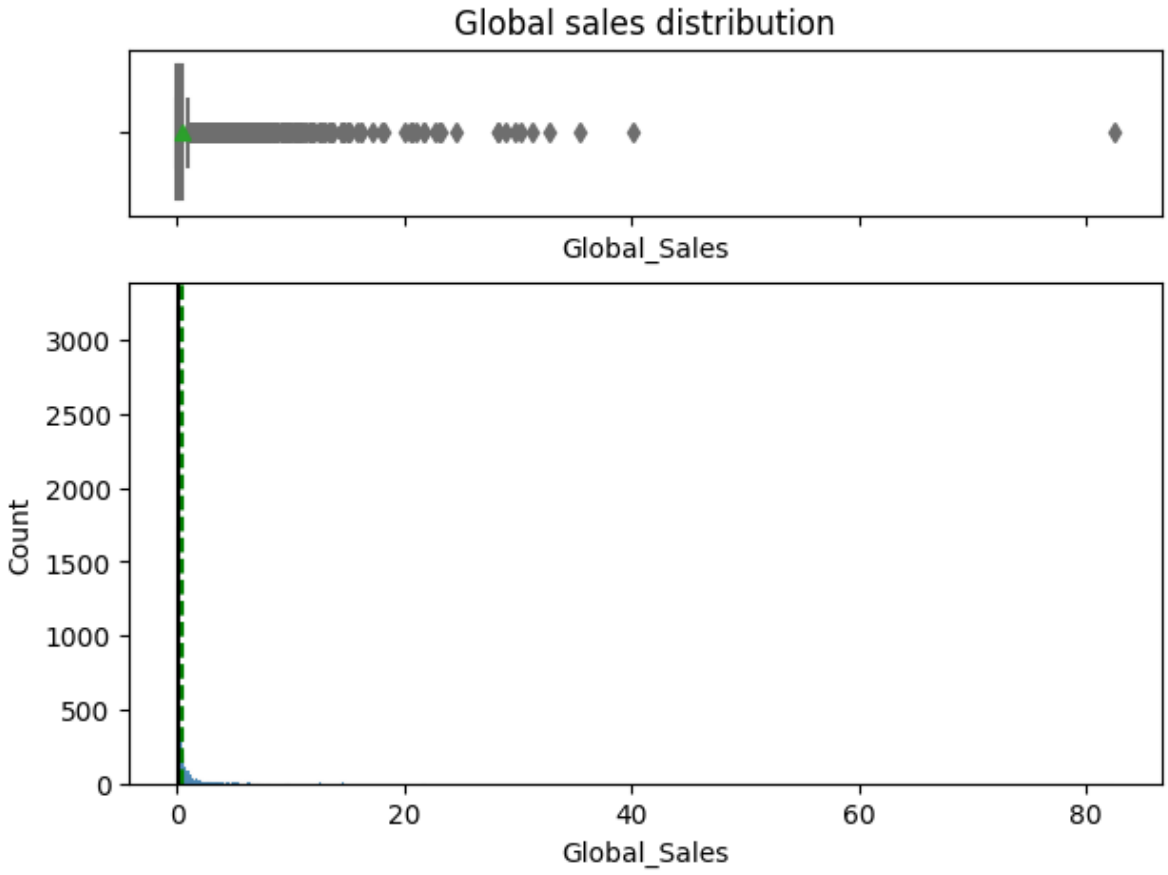
```
<Figure size 768x576 with 0 Axes>
```



Defining the function for creating histogram and boxplot

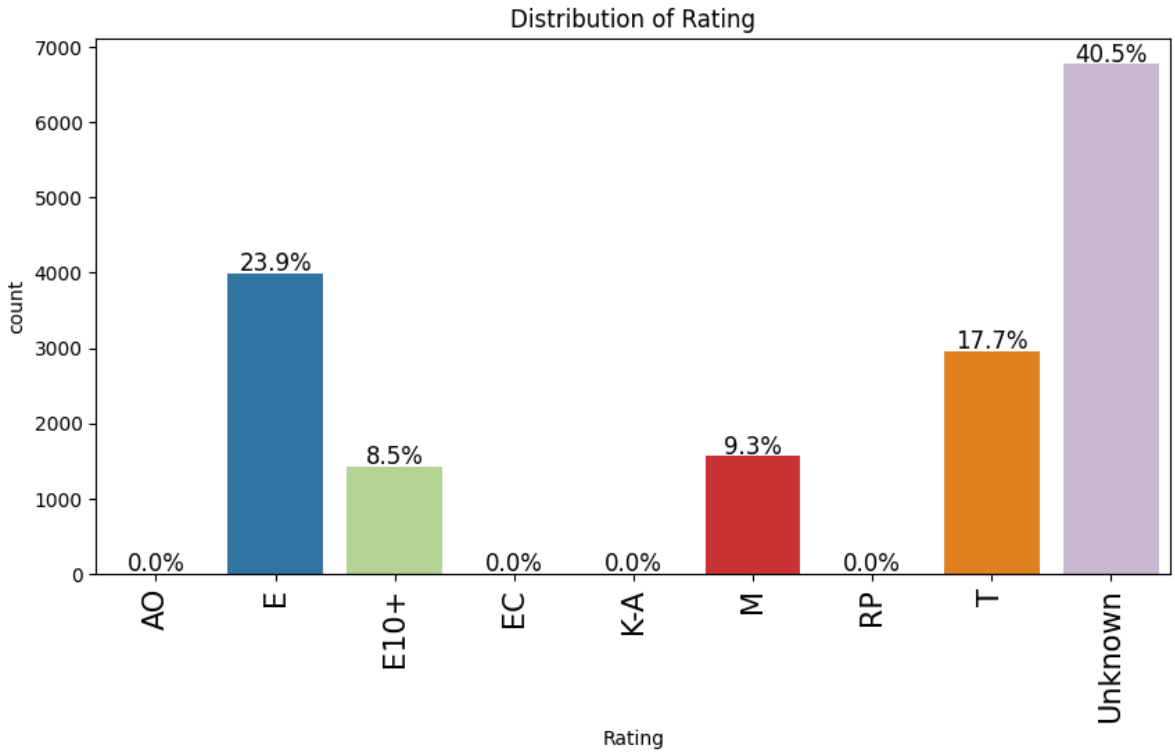
In [34]:
```
# create histogram and boxplot

def histogram_boxplot(data, feature, figsize=(7, 5), kde=False, bins=None, title
    """
    Boxplot and histogram

    data: dataframe
    feature: dataframe column
    figsize: size of figure (default (12,7))
    kde: show density curve (default False)
    bins: number of bins for histogram (default None)
    """
    f2, (ax_box2, ax_hist2) = plt.subplots(
        nrows=2,  #rows of subplot grid= 2
        sharex=True,
        gridspec_kw={"height_ratios": (0.25, 0.75)},
        figsize=figsize,
    )  # create the 2 subplots
    sns.boxplot(
        data=data, x=feature, ax=ax_box2, showmeans=True, color="violet"
    )
    sns.histplot(
        data=data, x=feature, kde=kde, ax=ax_hist2, bins=bins, palette="winter"
    ) if bins else sns.histplot(
        data=data, x=feature, kde=kde, ax=ax_hist2
    )  # For histogram
    ax_hist2.axvline(
        data[feature].mean(), color="green", linestyle="--"
    )  # Add mean
    ax_hist2.axvline(
        data[feature].median(), color="black", linestyle="-"
    )  # Add median
    if title:
      ax_box2.set_title(title)
    plt.show()
```

In [35]:
```
# check global sales distribution
histogram_boxplot(data = Video_Games_dataset, feature="Global_Sales", title="Glo
```

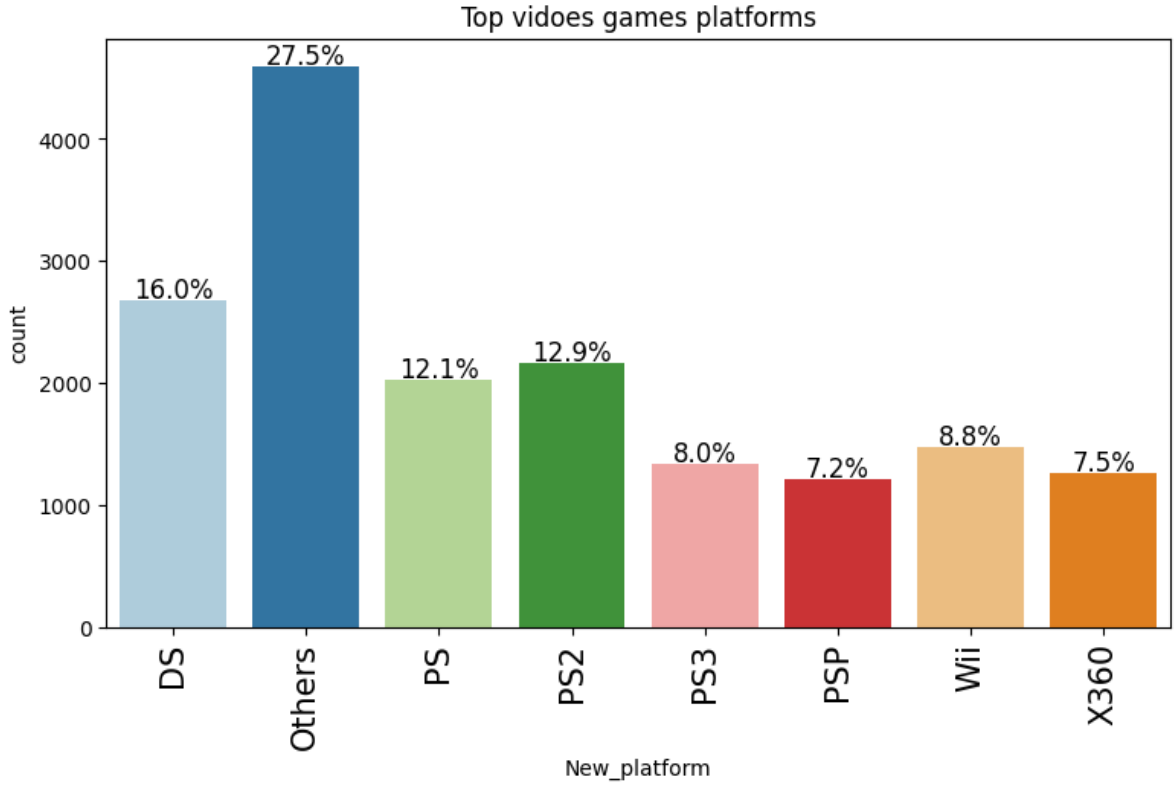


In [36]:
```
# create tag barplots

def tag_barplot(data, feature, perc=False, n=None, title = None):
    """
    Barplot with percentage at the top

    data: dataframe
    feature: dataframe column
    perc: whether to display percentages instead of count (default is False)
    n: displays the top n category levels (default is None, i.e., display all lev
    """

    total = len(data[feature])  # length of the column
    count = data[feature].nunique()
    if n is None:
        plt.figure(figsize=(count + 1, 5))
    else:
        plt.figure(figsize=(n + 1, 5))

    plt.xticks(rotation=90, fontsize=15)
    ax = sns.countplot(
        data=data,
        x=feature,
        palette="Paired",
        order=data[feature].value_counts().index[:n].sort_values(),
    )

    for p in ax.patches:
        if perc == True:
            label = "{:.1f}%".format(
                100 * p.get_height() / total
            )  # percentage of each class
        else:
            label = p.get_height()  # count of each level

        x = p.get_x() + p.get_width() / 2  # width of the plot
        y = p.get_height()  # height of the plot

        ax.annotate(
            label,
            (x, y),
            ha="center",
            va="center",
            size=12,
            xytext=(0, 5),
            textcoords="offset points",
        )  # annotate the percentage
    plt.title(title)
    plt.show()  # show the plot
```

In [81]:
```
tag_barplot(data = Video_Games_dataset, feature = "Rating", perc = True, title =
```



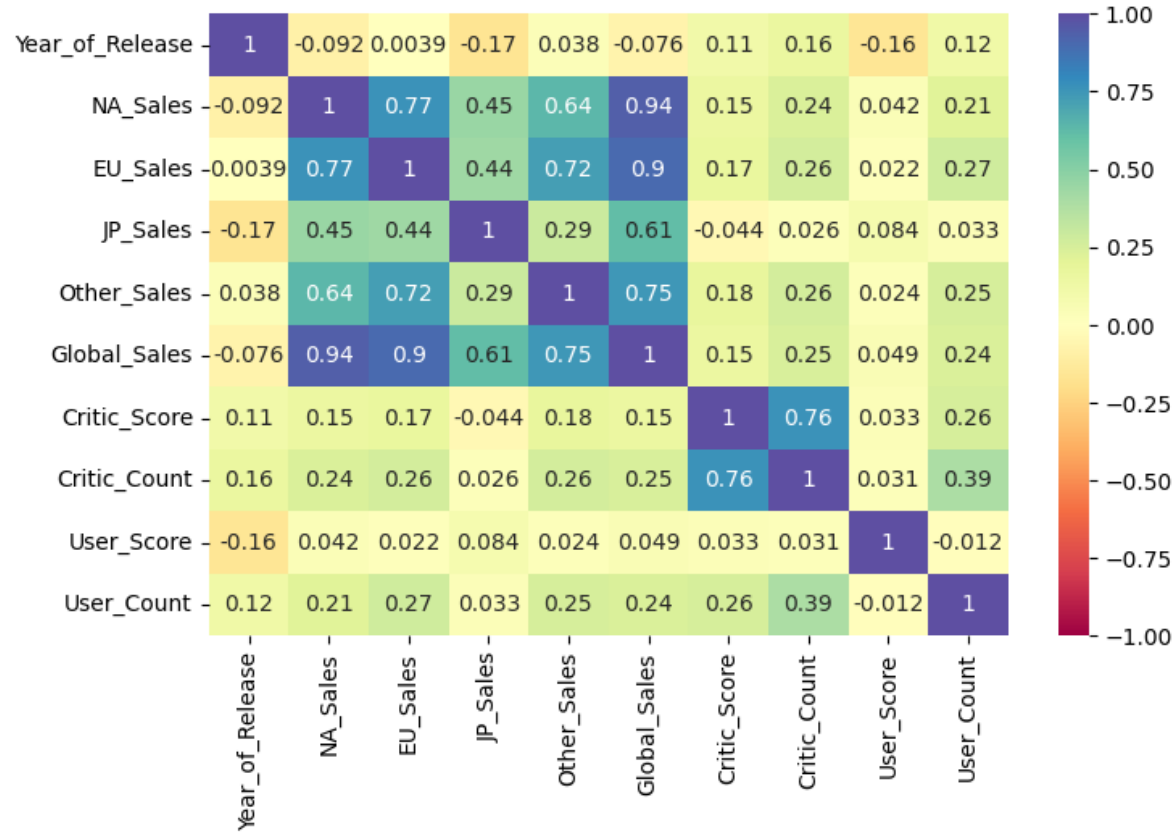
In [82]:
```
# check the new created platform
tag_barplot(data = Video_Games_dataset, feature = "New_platform", perc = True, t
```



In [37]:
```
# extract numerical features
numerical_data=Video_Games_dataset.select_dtypes(include ="numbers")
```

In [38]:
```
# heat map
map_correllation =numerical_data.correlation()
plt.figure(figsize = (8,5))
sns.heatmap(map_correllation,annot=True, vmin=-1, vmax =1,cmap="Spectral")
```

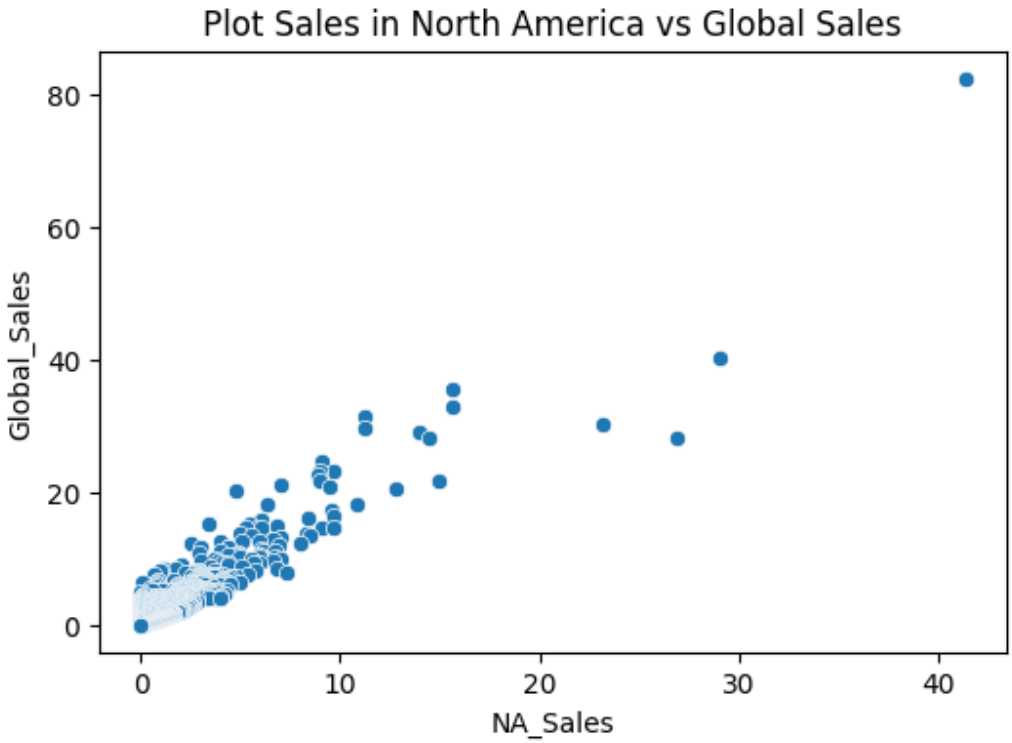
Out[38]: <Axes: >



In [85]:
```
# create scatterplot
def scatterplot_function(x,y, title,data = Video_Games_dataset):
  plt.figure(figsize = (6,4))
  sns.scatterplot(x=x,y=y, data = Video_Games_dataset)
  plt.title(title)
  plt.show()
```

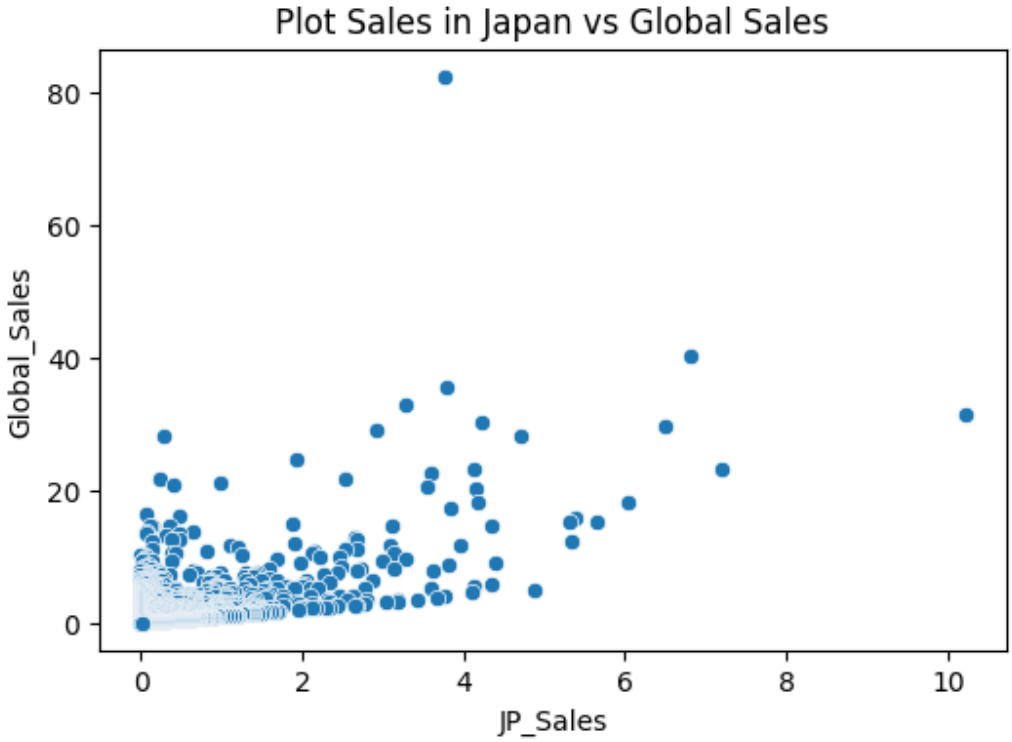
North America Sales vs Global Sales

In [86]:
```
scatterplot_function(x ="NA_Sales",y="Global_Sales",
                     title = "Plot Sales in North America vs Global Sales")
```



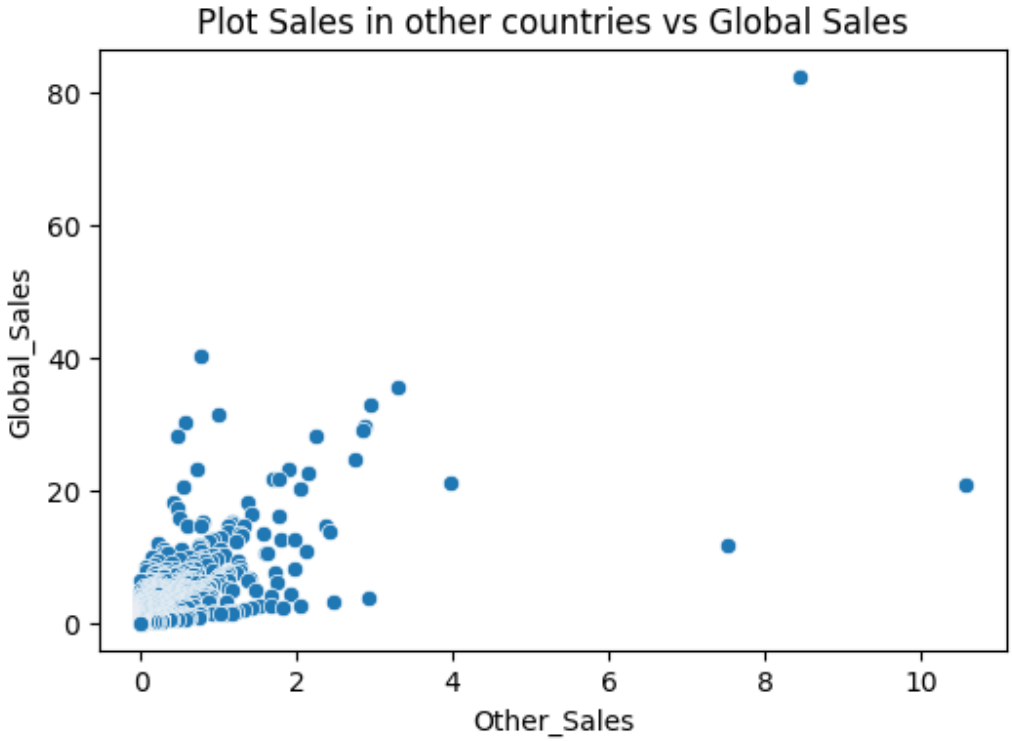
Japan Sales vs Global Sales

In [87]:
```
scatterplot_function(x ="JP_Sales",y="Global_Sales",
                     title = "Plot Sales in Japan vs Global Sales")
```



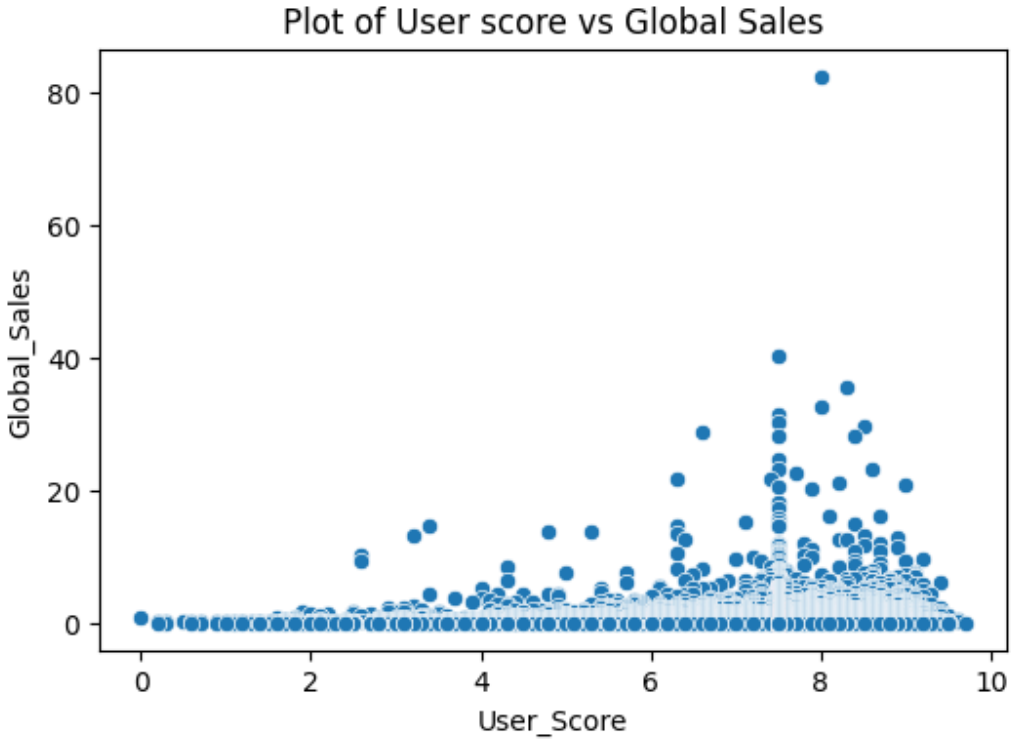
Other Region Sales vs Global Sales

In [88]:
```
scatterplot_function(x ="Other_Sales",y="Global_Sales",
                     title = "Plot Sales in other countries vs Global Sales")
```



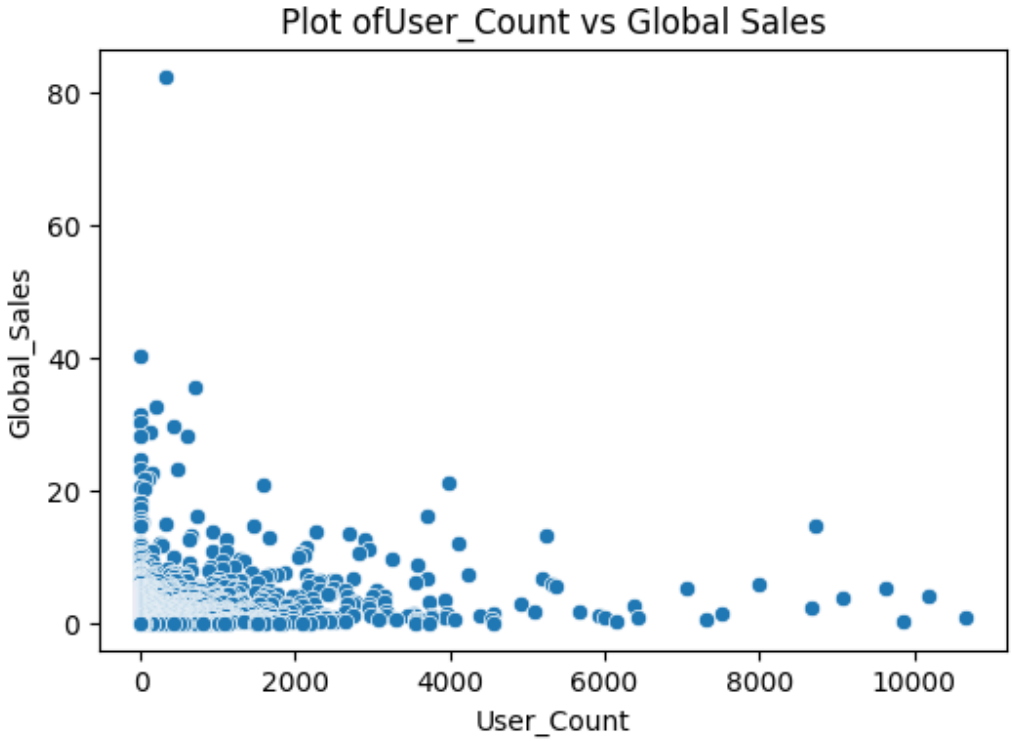
User_score vs Global Sales

In [89]:
```
scatterplot_function(x ="User_Score",y="Global_Sales",
                     title = "Plot of User score vs Global Sales")
```



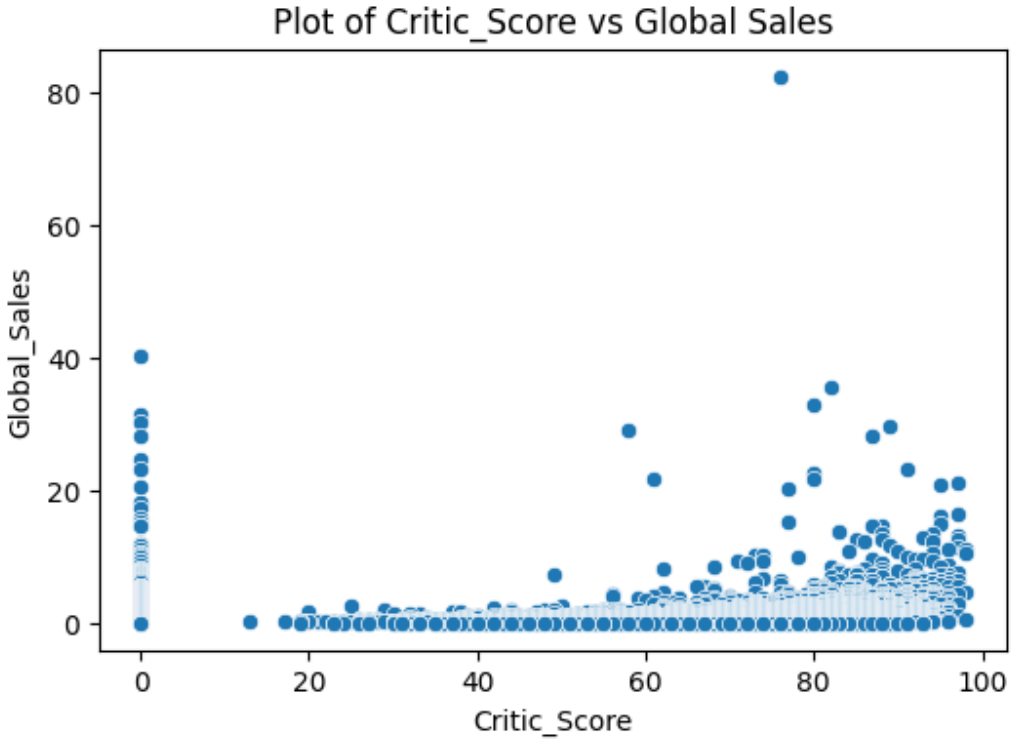
User Count vs Global Sales

In [90]:
```
scatterplot_function(x = "User_Count",y="Global_Sales",
                     title = "Plot ofUser_Count vs Global Sales")
```



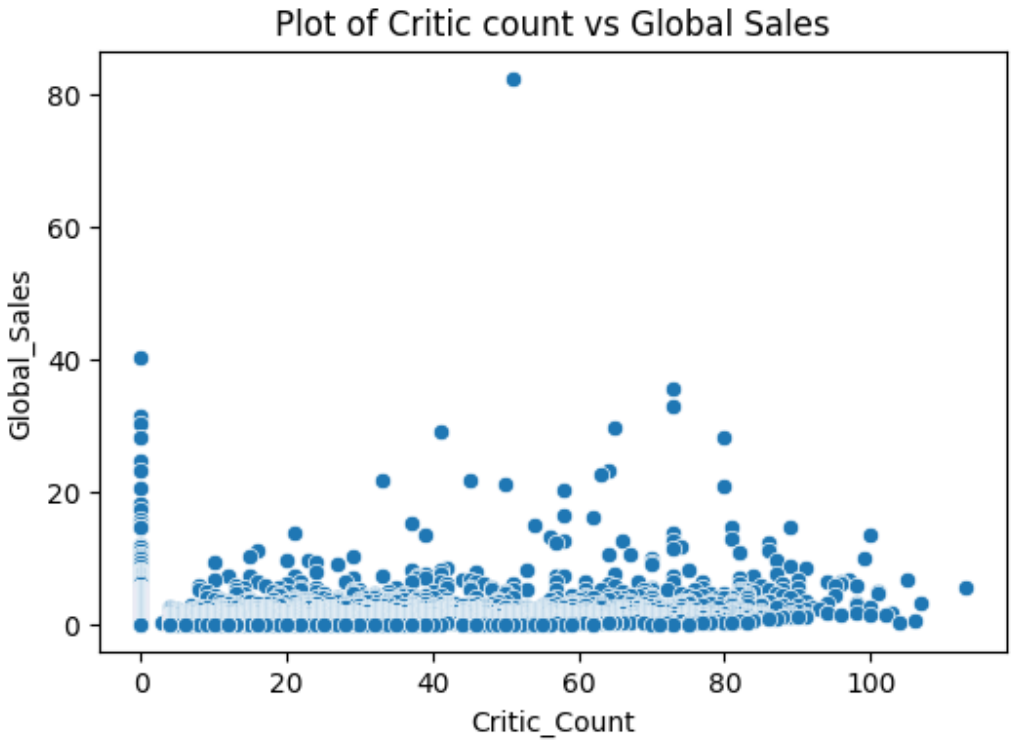
Critic Score vs Global Sales

In [91]:
```
scatterplot_function(x ="Critic_Score",y="Global_Sales",
                     title = "Plot of Critic_Score vs Global Sales")
```



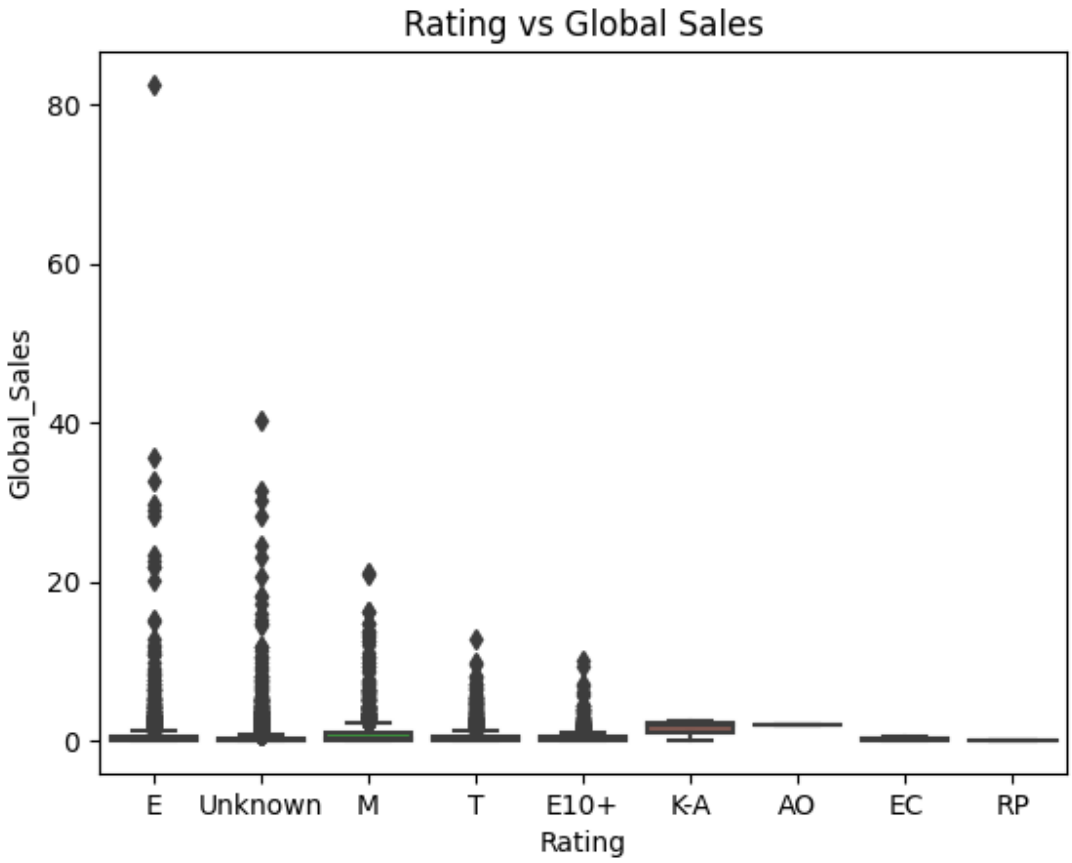
Critic Count vs Global Sales

In [92]:
```
scatterplot_function(x ="Critic_Count",y="Global_Sales",
                     title = "Plot of Critic count vs Global Sales")
```



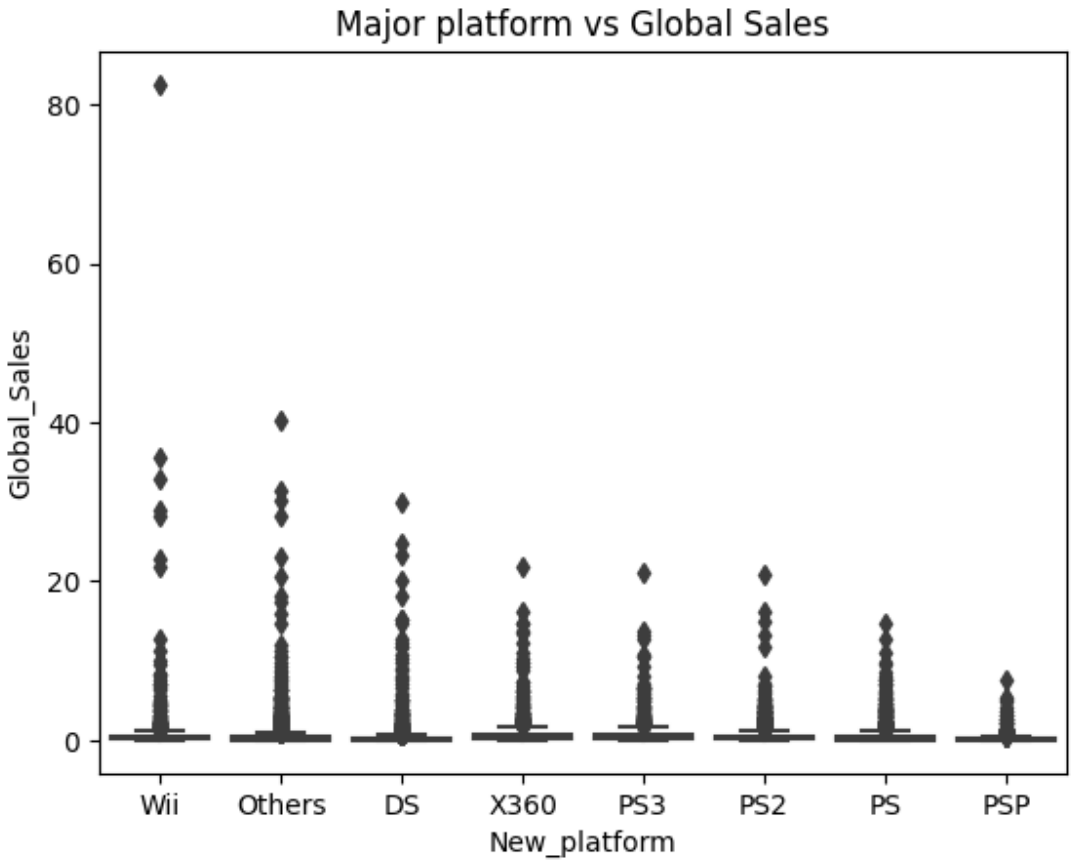
Rating vs Global Sales

In [93]:
```
sns.boxplot(x ="Rating", y= "Global_Sales", data = Video_Games_dataset);
plt.title("Rating vs Global Sales");
```

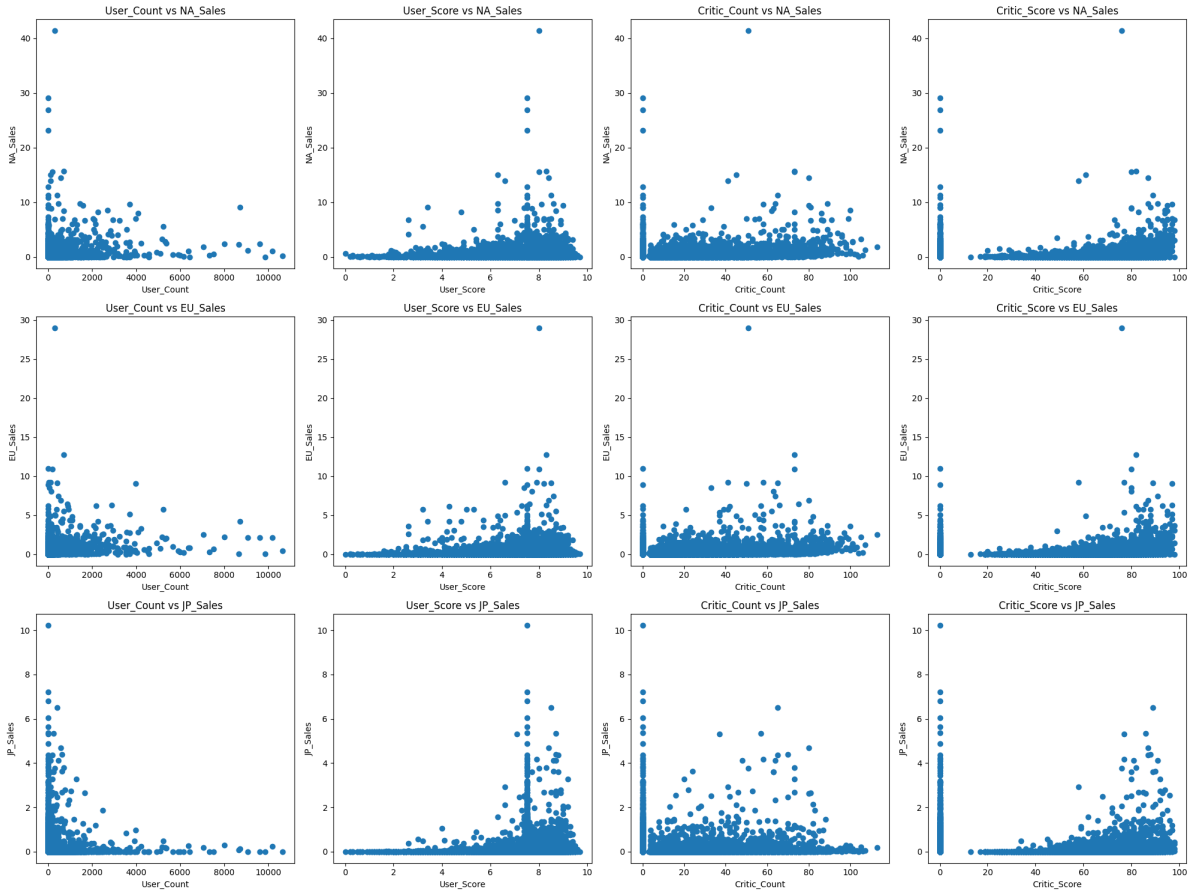


New_platform vs global sales

In [94]:
```
sns.boxplot(x ="New_platform", y= "Global_Sales", data = Video_Games_dataset);
plt.title("Major platform vs Global Sales");
```

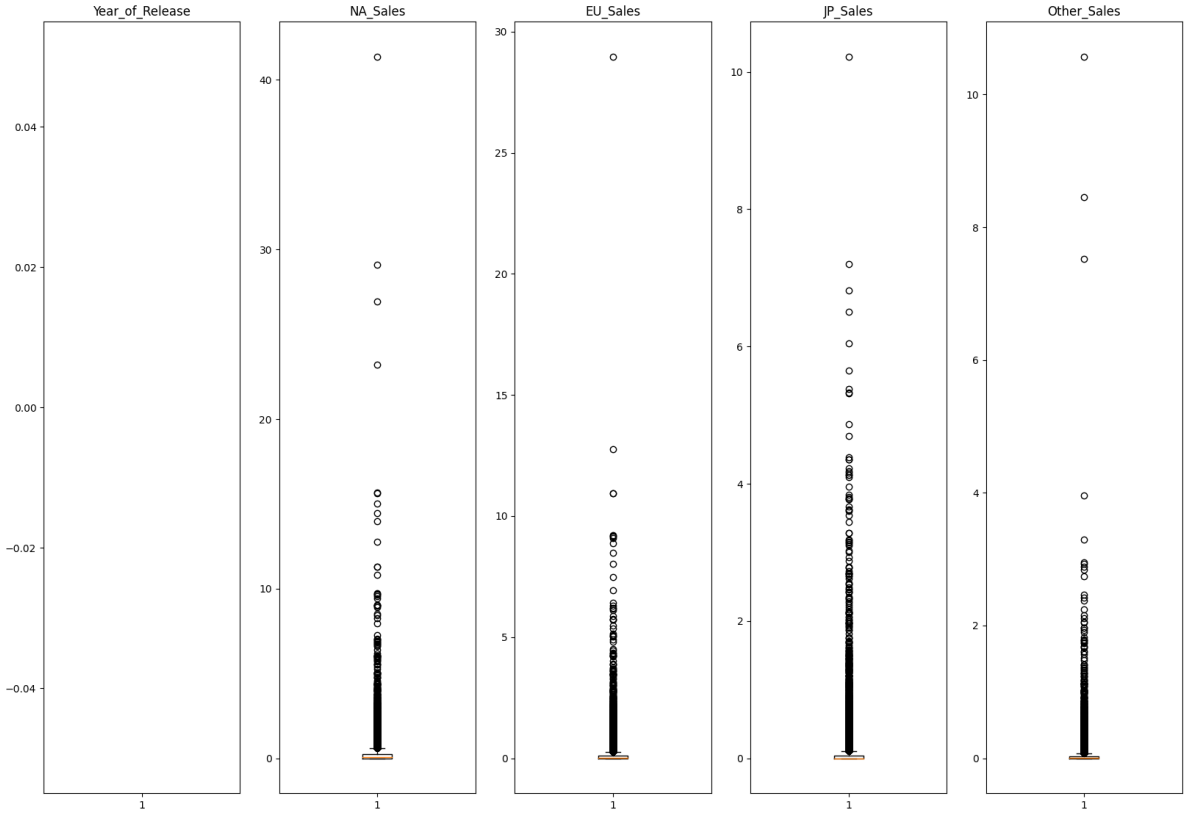


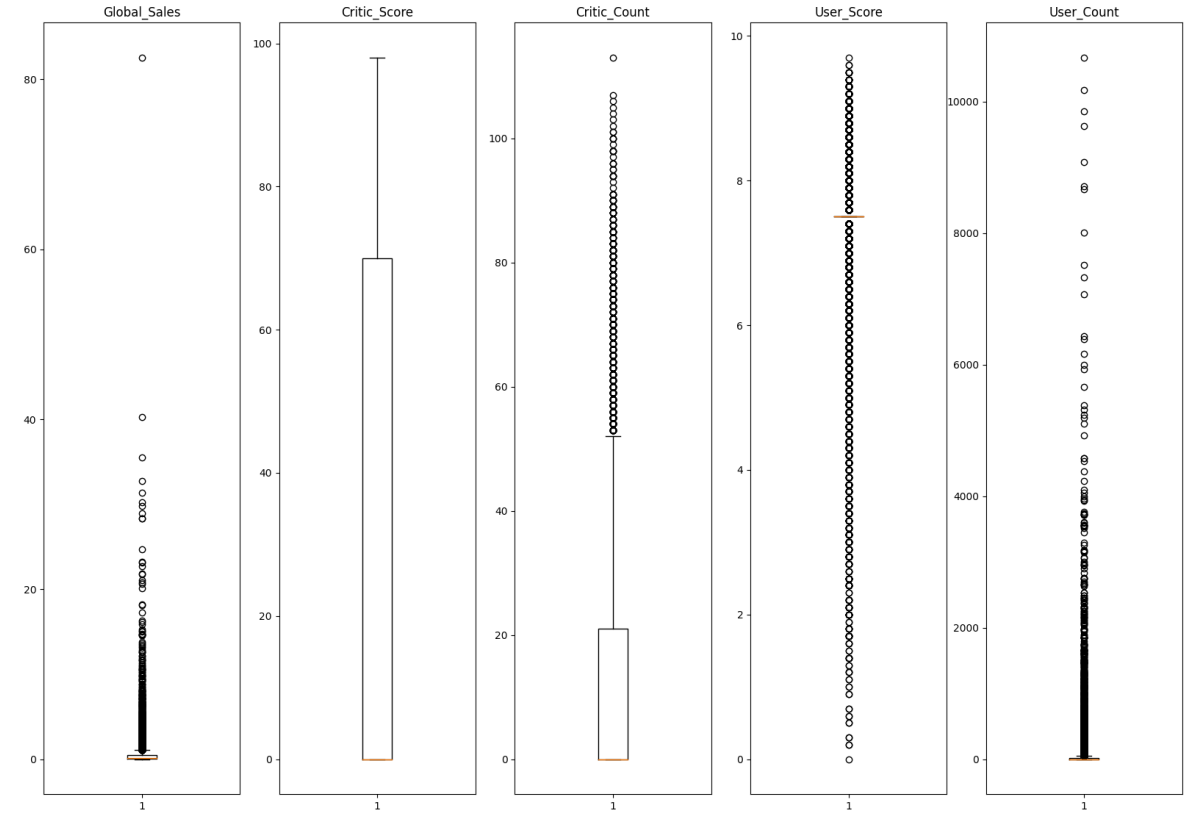
In [31]:
```
x_features = ["User_Count", "User_Score", "Critic_Count", "Critic_Score"]
y_features = ["NA_Sales", "EU_Sales", "JP_Sales"]

fig, axes = plt.subplots(nrows = len(y_features), ncols = len (x_features), figs

for i, y_features in enumerate(y_features):
    for o, x_feat in enumerate(x_features):
        axes[i, o].scatter(Video_Games_dataset[x_features],Video_Games_dataset[y
        axes[i, o].set_xlabel(x_features)
        axes[i, o].set_ylabel(y_features)
        axes[i, o].set_title(f'{x_features} vs {y_features}')
        axes[i, o].grid(False)
plt.savefig("scatter_plots_of_no_correlation.png")
plt.tight_layout()
plt.show()
```



Outlier Handling

In [39]:
```
# handle outlier
# Create figure of size of 20 by 30 inches
plt.figure(figsize=(20, 30))

# Loop through numerical variable
for item, variable in enumerate(numerical_data):

    # Create subplot
    plt.subplot(2, 5, item+1)

    # Create boxplot
    plt.boxplot(Video_Games_dataset[variable])

    # Set title of the current subplot
    plt.title(variable)

# Display the figure
plt.show()
```


Year_of_Release
NA_Sales
EU_Sales
JP_Sales
Other_Sales


Global_Sales
Critic_Score
Critic_Count
User_Score
User_Count

Treat Outlier

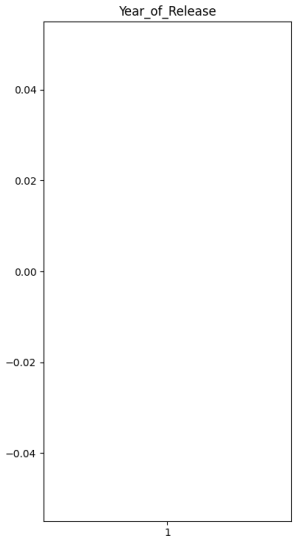
In [40]:
```
# drop Global sales
numerical_data.drop("Global_Sales", axis = 1,  inplace = True)

numerical_data.columns
```

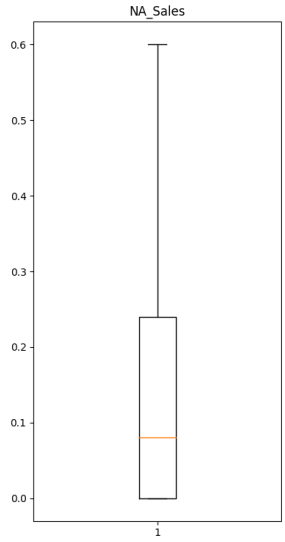
Out[40]:
```
Index(['Year_of_Release', 'NA_Sales', 'EU_Sales', 'JP_Sales', 'Other_Sales',
       'Critic_Score', 'Critic_Count', 'User_Score', 'User_Count'],
      dtype='object')
```

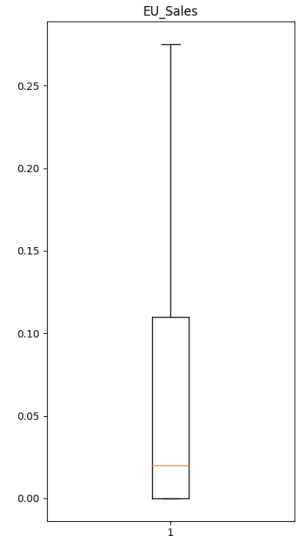
In [41]:
```
# Treatment of  outliers

def outliers_treatment(Video_Games_dataset, columns):
    """
    Treat outliers in a numerical column of a dataframe using the interquartile

    Parameters:
    Video_Games_dataset (pandas.DataFrame): The dataframe containing the column
    col (str): The name of the column to be treated.

    Returns:
    pandas.DataFrame: The modified dataframe with clipped values for the specifi
    """
    # Calculate the 25th and 75th quantiles of the column
    Q1 = Video_Games_dataset[columns].quantile(0.25)
    Q3 = Video_Games_dataset[columns].quantile(0.75)

    # Calculate the interquartile range (IQR)
    IQR = Q3 - Q1

    # Calculate the lower and upper whiskers for outlier detection
    lower = Q1 - (1.5 * IQR)
    upper= Q3 + (1.5 * IQR)

    # Clip the values in the column to be within the lower and upper whiskers
    Video_Games_dataset[columns] = np.clip(Video_Games_dataset[columns], lower,

    return Video_Games_dataset


# This function treats outliers in all numerical columns of a dataframe
def all_outliers_treatment(Video_Games_dataset, columns_list):
    """
    Treat outliers in all numerical columns of a dataframe using the interquarti

    Parameters:
    Video_Games_dataset (pandas.DataFrame): The dataframe containing the columns
    col_list (list of str): The list of column names to be treated.

    Returns:
    pandas.DataFrame: The modified dataframe with clipped values for all specifi
    """
    # Loop through the columns in the col_list and call the outliers_treatment f
    for col in columns_list:
        Video_Games_dataset = outliers_treatment(Video_Games_dataset, col)

    # Return the modified dataframe
    return Video_Games_dataset
```

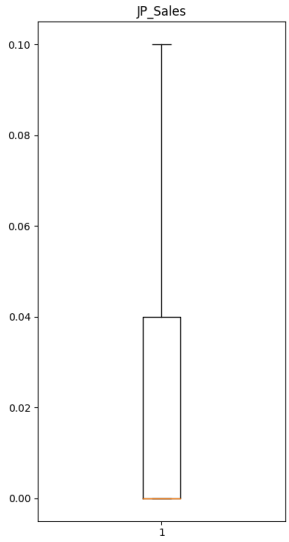
In [42]:
```
# Handling all the outliers
Video_Games_dataset = all_outliers_treatment(Video_Games_dataset,numerical_data)
```

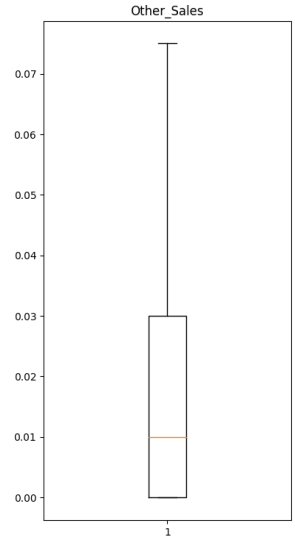
In [43]:
```
# Create a figure size of 20 by 30 inches
plt.figure(figsize=(20, 30))

# Loop through each numerical variable
for item, variable in enumerate(numerical_data):

    # Create a subplot
    plt.subplot(3, 4, i+1)

    # Create a boxplot of the current variable
    plt.boxplot(Video_Games_dataset[variable], whis=1.5)

    # Set the title of subplot
    plt.title(variable)

# Show the figure
plt.show()
```

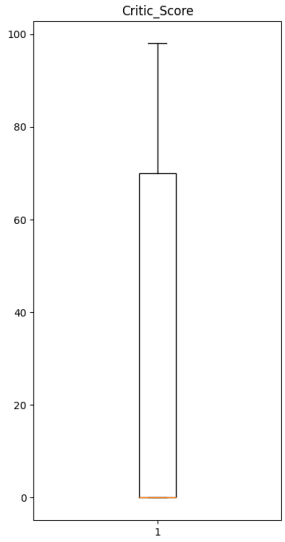

Year_of_Release
0.04
0.02
0.00
−0.02
−0.04
1

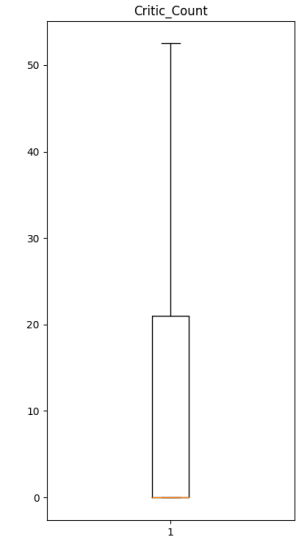

NA_Sales
0.6
0.5
0.4
0.3
0.2
0.1
0.0
1

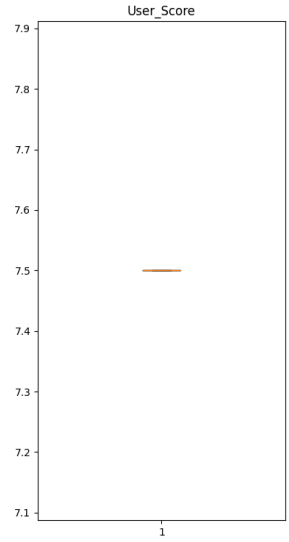

EU_Sales
0.25
0.20
0.15
0.10
0.05
0.00
1

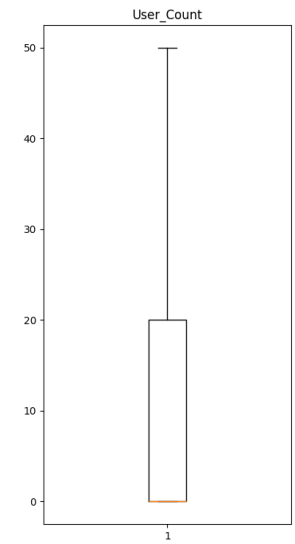

JP_Sales
0.10
0.08
0.06
0.04
0.02
0.00
1


Other_Sales
0.07
0.06
0.05
0.04
0.03
0.02
0.01
0.00
1


Critic_Score
100
80
60
40
20
0
1

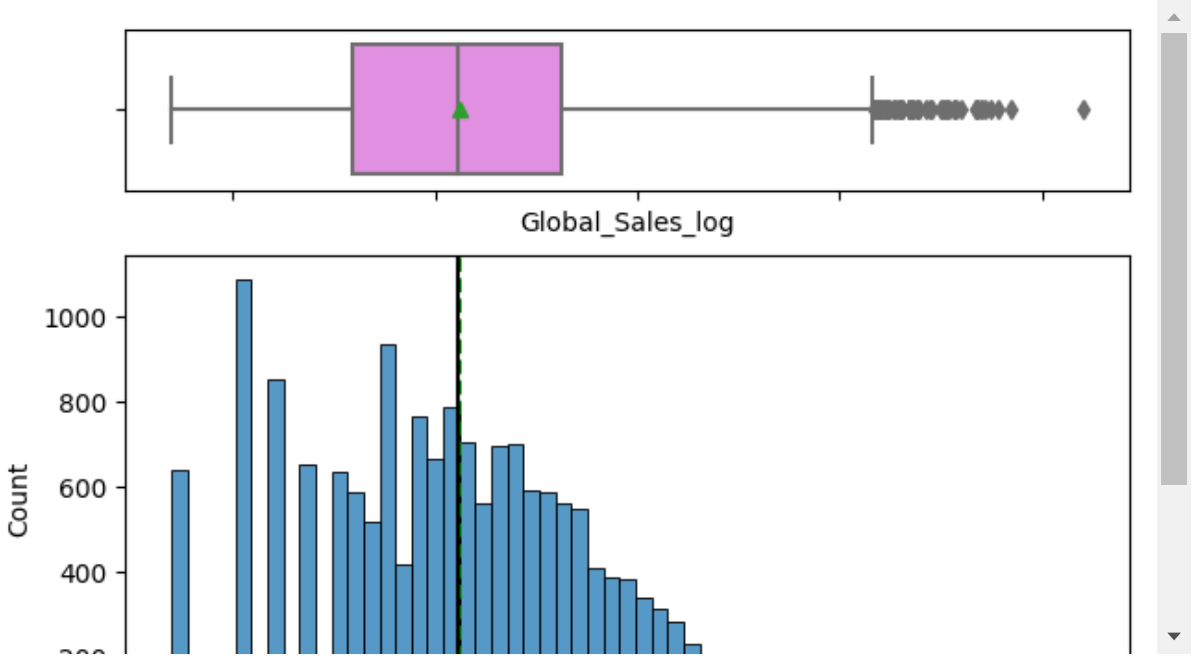

Critic_Count
50
40
30
20
10
0
1


User_Score
7.9
7.8
7.7
7.6
7.5
7.4
7.3
7.2
7.1
1


User_Count
50
40
30
20
10
0
1

Treating Global_Sales

In [44]:
```
# change the target using log transformation.
Video_Games_dataset["Global_Sales_Record"] = np.log(Video_Games_dataset["Global_

histogram_boxplot(data = Video_Games_dataset, feature="Global_Sales_Record")
```



MODELLING

In [45]:
```
# import required libraries
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LinearRegression
from sklearn.metrics import r2_score, mean_squared_error,mean_absolute_error
from sklearn.linear_model import Ridge
from sklearn.linear_model import Lasso
from sklearn.ensemble import RandomForestRegressor
from sklearn.neighbors import KNeighborsRegressor
from sklearn.ensemble import GradientBoostingRegressor
from sklearn.svm import SVR
```

selecting all numerical feature to predict sales

In [46]:
```
# Select features and target
X = Video_Games_dataset[[ "NA_Sales", "JP_Sales", "EU_Sales", "Other_Sales", "Cr
        "User_Score", "User_Count"]]

#y = Video_Games_dataset["Global_Sales"]
y = Video_Games_dataset["Global_Sales_Record"] + 1e-10
```

In [47]:
```
# split the data into training and testing
x_train, x_test, y_train, y_test = train_test_split(X,y,test_size = 0.2,random_s
```

In [2]:
```
#x_train.shape
```

In [3]:
```
#x_test.shape
```

In [49]:
```
# Normalize the data
scaler = StandardScaler() # convert data to have a mean


x_train_sd = scaler.fit_transform(x_train)
x_test_sd = scaler.transform(x_test)
```

Simple linear regression

In [50]:
```
#convert to pandas dataframe
x_train_sd = pd.DataFrame(x_train_sd, columns=x_train.columns)
x_test_sd = pd.DataFrame(x_test_sd, columns=x_test.columns)
```

Reshaping the attributes

In [51]:
```
# NA_Sales
x_train_sd_NA = x_train_sd['NA_Sales'].to_numpy().reshape(-1,1)
x_test_sd_NA = x_test_sd['NA_Sales'].to_numpy().reshape(-1,1)

# EU_Sales
x_train_sd_EU = x_train_sd['EU_Sales'].to_numpy().reshape(-1,1)
x_test_sd_EU = x_test_sd['EU_Sales'].to_numpy().reshape(-1,1)

# JP_Sales
x_train_sd_JP = x_train_sd['JP_Sales'].to_numpy().reshape(-1,1)
x_test_sd_JP = x_test_sd['JP_Sales'].to_numpy().reshape(-1,1)

# other_sales
x_train_sd_others = x_train_sd['Other_Sales'].to_numpy().reshape(-1,1)
x_test_sd_others = x_test_sd['Other_Sales'].to_numpy().reshape(-1,1)
```

Model Evaluation

In [52]:
```
# function to calculate adjusted R-squared

def adj_r2_score(predictors, targets, predictions):
    r2 = r2_score(targets, predictions)
    n = predictors.shape[0]
    k = predictors.shape[1]
    return 1 - ((1 - r2) * (n - 1) / (n - k - 1))


# function to calculate MAPE mean absolute percentage error
def mape_score(targets, predictions):
    return np.mean(np.abs(targets - predictions) / targets) * 100


# function to compute different metrics to check performance of a regression mod
def model_performance_regression(model, predictors, target):
    """
    Function to compute different metrics to check regression model performance

    model: regressor
    predictors: independent variables
    target: dependent variable
    """

    # predict using the independent variables
    pred = model.predict(predictors)

    r2 = r2_score(target, pred)  # to compute R-squared
    adjr2 = adj_r2_score(predictors, target, pred)  # to compute adjusted R-squa
    rmse = np.sqrt(mean_squared_error(target, pred))  # to compute RMSE
    mae = mean_absolute_error(target, pred)  # to compute MAE
    mape = mape_score(target, pred)  # to compute MAPE
    mse= mean_squared_error(target, pred) # to compute MSE
    # creating a dataframe of metrics
    Video_Games_dataset_perf = pd.DataFrame(
        {
            "MSE":mse,
            "RMSE": rmse,
            "MAE": mae,
            "R-squared": r2,
            "Adj. R-squared": adjr2,
            "MAPE": mape,
        },
        index=[0],
    )

    return Video_Games_dataset_performance
```

Training and evaluation the model

In [53]:
```
# train
line_reg_NA = LinearRegression()
line_reg_NA.fit(x_train_sd_NA, y_train)
#evaluate
line_reg_test_NA = model_performance_regression(line_reg_NA, x_test_sd_NA,y_test
print("Model performance of NA_sales")
print(line_reg_test_NA)
print()

#train model
line_reg_EU = LinearRegression()
line_reg_EU.fit(x_train_sd_EU, y_train)

#evaluate model
line_reg_test_EU = model_performance_regression(lin_reg_EU, x_test_sd_EU,y_test)
print("Model performance of EU_sales")
print(line_reg_test_EU)
print()

#train model JP_sales
line_reg_JP = LinearRegression()
line_reg_JP.fit(x_train_sd_JP, y_train)

#evaluate model
line_reg_test_JP = model_performance_regression(line_reg_JP, x_test_sd_JP,y_test
print("Model performance of JP_sales")
print(line_reg_test_JP)
print()

#train model JP_sales
line_reg_others = LinearRegression()
line_reg_others.fit(x_train_sd_others, y_train)

#evaluate model
line_reg_test_others = model_performance_regression(line_reg_others, x_test_sd_o
print("Model performance of others_sales")
print(line_reg_test_others)
print()
```

```
Model performance of NA_sales
        MSE      RMSE       MAE  R-squared  Adj. R-squared          MAPE
0  0.727802  0.853113  0.641517   0.658432         0.65833  4.858293e+08

Model performance of EU_sales
        MSE      RMSE       MAE  R-squared  Adj. R-squared          MAPE
0  0.966538  0.983127  0.760014    0.54639        0.546254  2.869063e+08

Model performance of JP_sales
        MSE      RMSE       MAE  R-squared  Adj. R-squared          MAPE
0  1.981565  1.407681  1.154542   0.070023        0.069745  8.856782e+08

Model performance of others_sales
        MSE      RMSE       MAE  R-squared  Adj. R-squared          MAPE
0  0.814165  0.902311  0.695988   0.617901        0.617786  1.983194e+08
```

Multiple linear regression

In [54]:
```
# Build linear regression model
mline_reg = LinearRegression()

# build model
mline_reg.fit(x_train_sd,y_train)
```

Out[54]:
```
LinearRegression()
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

Interpreting Model Performance

In [55]:
```
#train data set
line_reg_training_num = model_performance_regression(mline_reg,x_train_sd,y_trai
line_reg_training_num
print("Model performance of the numerical data on the training data")
print(line_reg_training_num )
print()

# test data set
line_reg_testing_num = model_performance_regression(mline_reg,x_test_sd,y_test)
line_reg_testing_num
print("Model performance of the numerical data on the testing data")
print(line_reg_testing_num)
```

```
Model performance of the numerical data on the training data
        MSE      RMSE       MAE  R-squared  Adj. R-squared          MAPE
0  0.421617  0.649321  0.514507   0.803648         0.80353  8.901474e+08

Model performance of the numerical data on the testing data
        MSE      RMSE       MAE  R-squared  Adj. R-squared          MAPE
0  0.418727  0.647091  0.517477   0.803485        0.803014  3.359908e+08
```

Determing the optimum aplha value for ridge regressors

In [56]:
```
#  import the ridge algorithm
from sklearn.linear_model import Ridge
```

In [57]:
```
# instantiate the model
ridge = Ridge(random_state=0)
#ridge = Ridge(random_state=0, alpha = 1.5)
# fit the model
ridge.fit(x_train_sd,y_train)
```

Out[57]:
```
Ridge(random_state=0)
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [58]:
```
# check ridge regression performance on trian data
ridge_train = model_performance_regression(ridge, x_train_sd,y_train)
ridge_train
```

Out[58]:

	MSE	RMSE	MAE	R-squared	Adj. R-squared	MAPE
0	0.421617	0.649321	0.514502	0.803648	0.80353	8.900502e+08

In [59]:
```
# check ridge regression performnace on testing data
ridge_test = model_performance_regression(ridge, x_test_sd,y_test)
ridge_test
```

Out[59]:

	MSE	RMSE	MAE	R-squared	Adj. R-squared	MAPE
0	0.418729	0.647093	0.517473	0.803484	0.803013	3.359346e+08

In [120]:
```
# check different alpha value on model

alpha_values = np.linspace(0.2, 10, num = 10)
for alpha in alpha_values:
  ridge = Ridge(random_state=0, alpha=alpha)
  ridge.fit(x_train_sd,y_train)
  ridge_train = model_performance_regression(ridge, x_train_sd,y_train)
  print(ridge_train)
  print(alpha)
```

```
        MSE      RMSE       MAE  R-squared  Adj. R-squared          MAPE
0  0.421617  0.649321  0.514506   0.803648         0.80353  8.901280e+08
0.2
        MSE      RMSE     MAE  R-squared  Adj. R-squared          MAPE
0  0.421617  0.649321  0.5145   0.803648         0.80353  8.900222e+08
1.288888888888889
        MSE      RMSE       MAE  R-squared  Adj. R-squared          MAPE
0  0.421617  0.649321  0.514495   0.803648         0.80353  8.899164e+08
2.377777777777778
        MSE      RMSE      MAE  R-squared  Adj. R-squared          MAPE
0  0.421617  0.649321  0.51449   0.803648         0.80353  8.898107e+08
3.4666666666666672
        MSE      RMSE       MAE  R-squared  Adj. R-squared          MAPE
0  0.421617  0.649321  0.514485   0.803648         0.80353  8.897050e+08
4.555555555555556
        MSE      RMSE       MAE  R-squared  Adj. R-squared          MAPE
0  0.421618  0.649321  0.514479   0.803648         0.80353  8.895993e+08
5.644444444444445
        MSE      RMSE       MAE  R-squared  Adj. R-squared          MAPE
0  0.421618  0.649321  0.514474   0.803648         0.80353  8.894937e+08
6.733333333333334
        MSE      RMSE       MAE  R-squared  Adj. R-squared          MAPE
0  0.421618  0.649321  0.514469   0.803648         0.80353  8.893881e+08
7.822222222222224
        MSE      RMSE       MAE  R-squared  Adj. R-squared          MAPE
0  0.421618  0.649321  0.514464   0.803648         0.80353  8.892826e+08
8.911111111111111
        MSE      RMSE       MAE  R-squared  Adj. R-squared          MAPE
0  0.421618  0.649321  0.514458   0.803647         0.80353  8.891770e+08
10.0
```

Determing the alpha for lasso regression

In [60]:
```
# instantiate the model without setting the alpha value
lasso = Lasso(random_state=0)
# fit the model
lasso.fit(x_train_sd,y_train)
```

Out[60]:
```
Lasso(random_state=0)
```
In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [61]:
```
# check performance
lasso_train = model_performance_regression(lasso, x_train_sd,y_train)
lasso_train
```

Out[61]:

	MSE	RMSE	MAE	R-squared	Adj. R-squared	MAPE
0	1.725439	1.31356	1.066118	0.196442	0.195961	1.938812e+09

In [62]:
```
# check performance
lasso_test = model_performance_regression(lasso, x_test_sd,y_test)
lasso_test
```

Out[62]:

	MSE	RMSE	MAE	R-squared	Adj. R-squared	MAPE
0	1.710238	1.307761	1.048251	0.197361	0.195436	8.883424e+08

In [63]:
```
# change alpha value for lasso
alpha_values = np.linspace(0.001, 0.2, num = 10)# start 0.2, stop 10, number of
for alpha in alpha_values:
  lasso = Lasso(random_state=0, alpha=alpha)
  lasso.fit(x_train_sd,y_train)
  lasso_train = model_performance_regression(lasso, x_train_sd,y_train)
  print(lasso_train)
  print(alpha)
```

```
        MSE      RMSE       MAE  R-squared  Adj. R-squared          MAPE
0  0.421631  0.649331  0.514424   0.803641        0.803524  8.872091e+08
0.001
        MSE      RMSE       MAE  R-squared  Adj. R-squared          MAPE
0  0.427989  0.654208  0.515207   0.800681        0.800561  8.220325e+08
0.023111111111111114
        MSE      RMSE       MAE  R-squared  Adj. R-squared          MAPE
0  0.432311  0.657503  0.513689   0.798668        0.798547  8.096296e+08
0.045222222222222226
        MSE      RMSE       MAE  R-squared  Adj. R-squared          MAPE
0  0.438955  0.662537  0.513857   0.795573        0.795451  7.960010e+08
0.06733333333333334
        MSE      RMSE       MAE  R-squared  Adj. R-squared          MAPE
0  0.446463  0.668179  0.514276   0.792077        0.791952  7.746786e+08
0.08944444444444445
        MSE      RMSE      MAE  R-squared  Adj. R-squared          MAPE
0  0.456089  0.675344  0.51585   0.787594        0.787467  7.533563e+08
0.11155555555555556
        MSE      RMSE       MAE  R-squared  Adj. R-squared          MAPE
0  0.467833  0.683983  0.518612   0.782125        0.781994  7.320339e+08
0.13366666666666668
        MSE      RMSE       MAE  R-squared  Adj. R-squared          MAPE
0  0.481694  0.694042  0.523098   0.775669        0.775535  7.159342e+08
0.1557777777777778
        MSE      RMSE      MAE  R-squared  Adj. R-squared          MAPE
0  0.497673  0.705459  0.52913   0.768228        0.768089  7.077241e+08
0.1778888888888889
        MSE      RMSE      MAE  R-squared  Adj. R-squared          MAPE
0  0.515769  0.718171  0.53647     0.7598        0.759656  7.062471e+08
0.2
```

Determining the best regressor

In [64]:
```
# instantiate all the regessor and fitting the model

# Random Forest Regression
rf = RandomForestRegressor(n_estimators=100, random_state=42)
rf.fit(x_train_sd, y_train)

# Gradient Boosting Regression
gb = GradientBoostingRegressor(n_estimators=100, random_state=42)
gb.fit(x_train_sd, y_train)

# Build the linear regression model
mlin_reg = LinearRegression()

# build model
mlin_reg.fit(x_train_sd,y_train)

lasso = Lasso(random_state=0, alpha=0.001)
lasso.fit(x_train_sd,y_train)

# Support Vector Regression
svr = SVR(kernel='linear')
svr.fit(x_train_sd, y_train)


# K Neighbors Regression
kn = KNeighborsRegressor(n_neighbors=5, weights='distance')
kn.fit(x_train_sd, y_train)
y_pred_kn = kn.predict(x_test_sd)
```

In [65]:
```
# testing the performance of all the regressors
lasso_test = model_performance_regression(lasso, x_test_sd,y_test)
print("Model performance of lasso regression")
print(lasso_test)
print()

# check performance of ridge regression on testing data
ridge_test = model_performance_regression(mlin_reg, x_test_sd,y_test)
print("Model performance of ridge regression")
print(ridge_test)
print()


gradient_test = model_performance_regression(gb, x_test_sd,y_test)
print("Model performance of Gradient Boosting regression")
print(gradient_test)
print()

random_test = model_performance_regression(rf, x_test_sd,y_test)
print("Model performance of RandomForest regression")
print(random_test)
print()

support_test = model_performance_regression(svr, x_test_sd,y_test)
print("Model performance of support vector regression")
print(support_test)
print()


linear_test = model_performance_regression(mlin_reg, x_test_sd,y_test)
print("Model performance of Linear regression")
print(linear_test)
print()
```

```
Model performance of lasso regression
        MSE      RMSE       MAE  R-squared  Adj. R-squared          MAPE
0  0.418786  0.647136  0.517342   0.803458        0.802987  3.338751e+08

Model performance of ridge regression
        MSE      RMSE       MAE  R-squared  Adj. R-squared          MAPE
0  0.418727  0.647091  0.517477   0.803485        0.803014  3.359908e+08

Model performance of Gradient Boosting regression
        MSE      RMSE       MAE  R-squared  Adj. R-squared          MAPE
0  0.087831  0.296362  0.164699    0.95878        0.958681  2.272803e+07

Model performance of RandomForest regression
        MSE      RMSE       MAE  R-squared  Adj. R-squared          MAPE
0  0.086145  0.293505  0.147731   0.959571        0.959474  4.609006e+07

Model performance of support vector regression
        MSE      RMSE       MAE  R-squared  Adj. R-squared          MAPE
0  0.438997  0.662569  0.502283   0.793972        0.793478  2.747299e+08

Model performance of Linear regression
        MSE      RMSE       MAE  R-squared  Adj. R-squared          MAPE
0  0.418727  0.647091  0.517477   0.803485        0.803014  3.359908e+08
```

Evaluating the model performance on training and test data of the the

In [66]:
```
# random forest training
random_train = model_performance_regression(rf, x_train_sd,y_train)
print("Model performance of RandomForest regression on train data")
print(random_train)
print()
# random forest train
random_test = model_performance_regression(rf, x_test_sd,y_test)
print("Model performance of RandomForest regression on test data")
print(random_test)
print()
print()

# gradian_bosting test
gradient_train = model_performance_regression(gb, x_train_sd,y_train)
print("Model performance of Gradient Boosting regression on train data")
print(gradient_train)
print()

gradient_test = model_performance_regression(gb, x_test_sd,y_test)
print("Model performance of Gradient Boosting regression on test data")
print(gradient_test)
print()
```

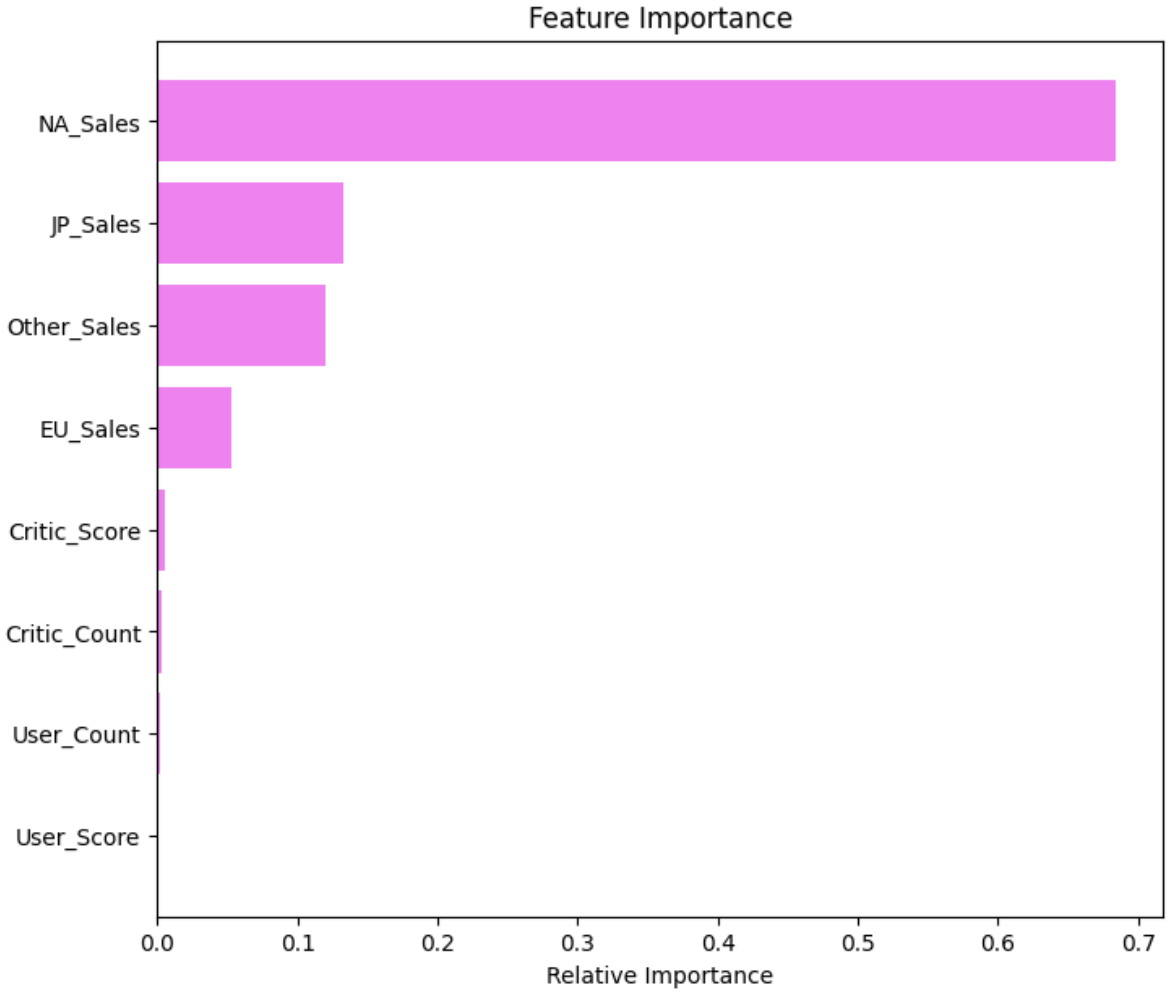
```
Model performance of RandomForest regression on train data
        MSE      RMSE       MAE  R-squared  Adj. R-squared          MAPE
0  0.049282  0.221995  0.092818   0.977049        0.977035  2.434744e+08

Model performance of RandomForest regression on test data
        MSE      RMSE       MAE  R-squared  Adj. R-squared          MAPE
0  0.086145  0.293505  0.147731   0.959571        0.959474  4.609006e+07


Model performance of Gradient Boosting regression on train data
        MSE      RMSE       MAE  R-squared  Adj. R-squared          MAPE
0  0.080265  0.283312  0.159722   0.962619        0.962597  4.150045e+08

Model performance of Gradient Boosting regression on test data
        MSE      RMSE       MAE  R-squared  Adj. R-squared          MAPE
0  0.087831  0.296362  0.164699    0.95878        0.958681  2.272803e+07
```

visualizing the the important feature by random forest model

In [67]:
```
feature_names = x_train.columns.to_list()
```

In [68]:
```
# Set features according to importance
import_feature = rf.feature_importances_

#import_feature
```

In [69]:
```
# visaulization
import_feature = rf.feature_importances_
indices=np.argsort(import_feature)# sorting the important features
plt.figure(figsize=(8,7))
plt.title("Feature Importance")
plt.barh(range(len(indices)),import_feature[indices], color ="violet", align="ce

plt.yticks(range(len(indices)), [feature_names[i] for i in indices])
plt.xlabel("Relative Importance")
plt.show()
```



K-fold cross validation to check model performance

- For K values from 5,10,15

In [71]:
```
from sklearn.metrics import r2_score, mean_squared_error, mean_absolute_error
from sklearn.model_selection import cross_val_score,KFold, cross_val_predict
import numpy as np
import pandas as pd

# random forest regressor
rf = RandomForestRegressor()

# Set k values
k_values = [5,10,15]

# DataFrame to store the evaluation metrics
Video_Games_dataset_eval = pd.DataFrame(columns=['k', 'MSE', 'RMSE', 'MAE', 'R-s

# Loop over k values
for k in k_values:
    # Set up cross-validation
    cv = KFold(n_splits=k, shuffle=True, random_state=42)

    # cross-validation on the training set using evaluation metrics
    y_preds = cross_val_predict(rf, X, y, cv=cv)
    scores_r2 = r2_score(y, y_preds)
    scores_rmse = mean_squared_error(y, y_preds, squared=False)
    scores_mae = mean_absolute_error(y, y_preds)
    scores_mse = mean_squared_error(y, y_preds)

    # Calculate adjusted R-squared
    adj_r2 = 1 - (1-scores_r2)*(len(y)-1)/(len(y)-X.shape[1]-1)

    # Calculate mean absolute percentage error (MAPE)
    mape = np.mean(np.abs((y - y_preds) / y)) * 100

    # Add evaluation metrics to the DataFrame
    Video_Games_dataset_eval = pd.concat([Video_Games_dataset_eval, pd.DataFrame
                                          'MSE': [scores_mse],
                                          'RMSE': [scores_rmse],
                                          'MAE': [scores_mae],
                                          'R-squared': [scores_r2],
                                          'Adj. R-squared': [adj_r2],
                                          'MAPE': [mape]})], ignore_index=

# Print the evaluation metrics DataFrame
print(Video_Games_dataset_eval)
```

	k	MSE	RMSE	MAE	R-squared	Adj. R-squared	MAPE
0	5	0.084146	0.290079	0.144999	0.960752	0.960734	4.177172e+08
1	10	0.083278	0.288579	0.144466	0.961157	0.961139	3.352200e+08
2	15	0.083039	0.288165	0.144125	0.961269	0.961250	3.595924e+08

b. What effect will the number of critics and users as well as their review scores have on the sales of Video games in North America, EU and Japan?

Effect "Critic_Score", "Critic_Count","User_Score", "User_Count" on the sales of Video games in North America

In [137]:
```
# Select features and target
x_NA = Video_Games_dataset[[ "Critic_Score", "Critic_Count","User_Score", "User_
y_NA = Video_Games_dataset["NA_Sales"] +1e-10
```

In [138]:
```
x_train, x_test, y_train, y_test = train_test_split(x_NA,y_NA,test_size = 0.2,ra
```

In [139]:
```
x_train_NA = scaler.fit_transform(x_train)
```

In [140]:
```
rf_top_NA = RandomForestRegressor(n_estimators=100, random_state=42)
rf_top_NA.fit(x_train_NA, y_train)

print("Effect of critics and users as well as their review scores on sales in no
# random forest training
random_train_NA = model_performance_regression(rf_top_NA, x_train_NA,y_train)
print("Model performance on train data")
print(f"{random_train_NA}\n")

# random forest train
random_test_NA = model_performance_regression(rf_top_NA, x_test_NA,y_test)
print("Model performance on test data")
print(random_test_NA)
```

```
Effect of critics and users as well as their review scores on sales in north am
erican

Model performance on train data
        MSE     RMSE       MAE  R-squared  Adj. R-squared          MAPE
0  0.021109  0.14529  0.103161    0.45768        0.457518  2.918651e+10

Model performance on test data
        MSE      RMSE       MAE  R-squared  Adj. R-squared          MAPE
0  0.033174  0.182138  0.136544   0.147373        0.146352  3.307079e+10
```

Effect "Critic_Score", "Critic_Count","User_Score", "User_Count" on the sales of Video games in EU

In [141]:
```
# Select features and target
x_EU = Video_Games_dataset[["Critic_Score", "Critic_Count","User_Score", "User_C
y_EU = Video_Games_dataset["EU_Sales"] +1e-10
```

In [142]:
```
x_train, x_test, y_train, y_test = train_test_split(x_EU,y_EU,test_size = 0.2,ra
```

In [143]:
```
x_train_EU = scaler.fit_transform(x_train)
x_test_EU = scaler.transform(x_test)
```

In [144]:
```
rf_top_EU = RandomForestRegressor(n_estimators=100, random_state=42)
rf_top_EU.fit(x_train_EU, y_train)

print("Effect of critics and users as well as their review scores on sales in EU
# random forest training
random_train_EU = model_performance_regression(rf_top_EU, x_train_EU,y_train)
print("Model performance on train data")
print(f"{random_train_EU}\n")

# random forest train
random_test_EU = model_performance_regression(rf_top_EU, x_test_EU,y_test)
print("Model performance on test data")
print(random_test_EU)
```

```
Effect of critics and users as well as their review scores on sales in EU

Model performance on train data
        MSE      RMSE       MAE  R-squared  Adj. R-squared          MAPE
0  0.004704  0.068583  0.048492   0.492346        0.492194  1.493621e+10

Model performance on test data
        MSE      RMSE       MAE  R-squared  Adj. R-squared          MAPE
0  0.007575  0.087034  0.065405   0.171973        0.170981  1.694820e+10
```

Effect "Critic_Score", "Critic_Count","User_Score", "User_Count" on the sales of Video games in JP

In [145]:
```
# Select features and target
x_JP = Video_Games_dataset[["Critic_Score", "Critic_Count","User_Score", "User_C
y_JP = Video_Games_dataset["JP_Sales"] +1e-10
```

In [146]:
```
x_train, x_test, y_train, y_test = train_test_split(x_JP,y_JP,test_size = 0.2,ra
```

In [147]:
```
x_train_JP = scaler.fit_transform(x_train)
x_test_JP = scaler.transform(x_test)
```

In [148]:
```
rf_top_JP = RandomForestRegressor(n_estimators=100, random_state=42)
rf_top_JP.fit(x_train_JP, y_train)

print("Effect of critics and users as well as their review scores on sales in Ja
# random forest training
random_train_JP = model_performance_regression(rf_top_JP, x_train_JP,y_train)
print("Model performance on train data")
print(f"{random_train_JP}\n")

# random forest train
random_test_JP = model_performance_regression(rf_top_JP, x_test_JP,y_test)
print("Model performance on test data")
print(random_test_JP)
```

```
Effect of critics and users as well as their review scores on sales in Japan

Model performance on train data
       MSE      RMSE       MAE  R-squared  Adj. R-squared          MAPE
0  0.00099  0.031463  0.023195   0.303745        0.303536  1.034565e+10

Model performance on test data
        MSE      RMSE       MAE  R-squared  Adj. R-squared          MAPE
0  0.001392  0.037316  0.028763   0.013192         0.01201  1.346726e+10
```

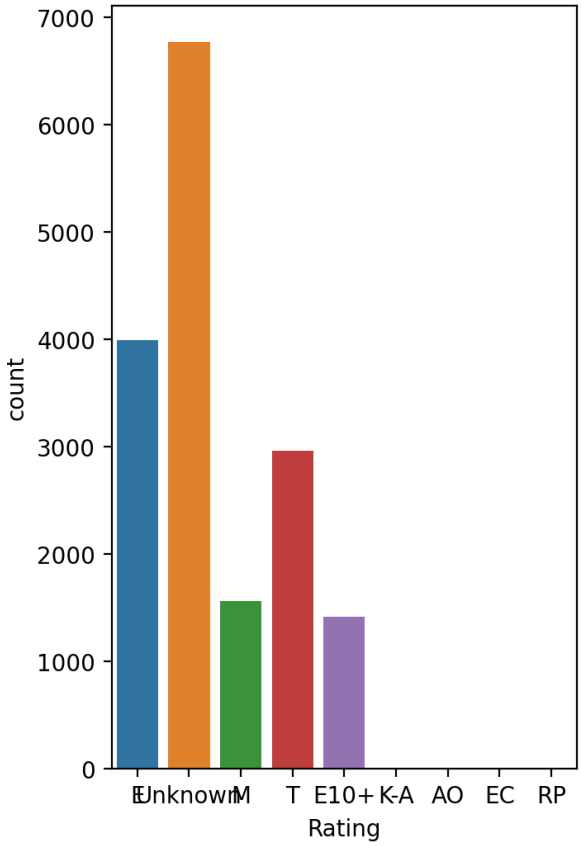
1(d) Use all the relevant categorical variables in the Video Game Dataset as the targetvariable at each instance and determine which of the variables performed best in classifying the dataset. Explain your findings.

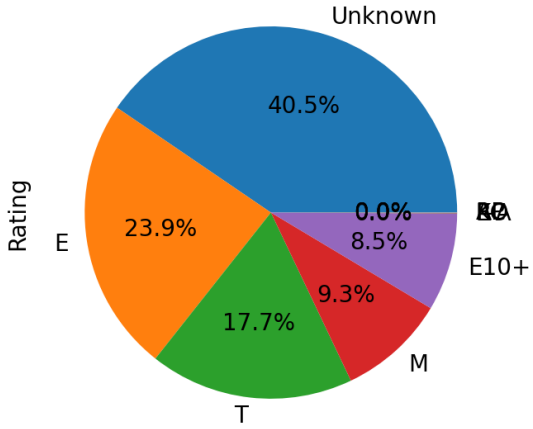
- A Classification problem

In [89]:
```
Video_Games_dataset["Rating"].value_counts()
```

Out[89]:
```
Unknown    6767
E          3991
T          2961
M          1563
E10+       1420
EC            8
K-A           3
RP            3
AO            1
Name: Rating, dtype: int64
```

In [155]:
```
# view rating in data set
fig, ax_position=plt.subplots(1,2,figsize=(8,6),dpi=200) # creates the framework
a = sns.countplot(x = 'Rating', data = Video_Games_dataset, ax=ax_position[0]) #
a = Video_Games_dataset['Rating'].value_counts().plot.pie(autopct="%1.1f%%", ax=
```





In [189]:
```
# subset the rating column
N_Video_Games_dataset=Video_Games_dataset[Video_Games_dataset["Rating"].isin(["E
```

In [190]:
```
X = N_Video_Games_dataset[["Genre", "NA_Sales", "JP_Sales", "EU_Sales", "Other_S
        "User_Score", "User_Count", "New_platform", "Global_Sales"]]
y = N_Video_Games_dataset['Rating']
```

In [191]:
```
# check changes
from typing import Counter
Counter(y)
```

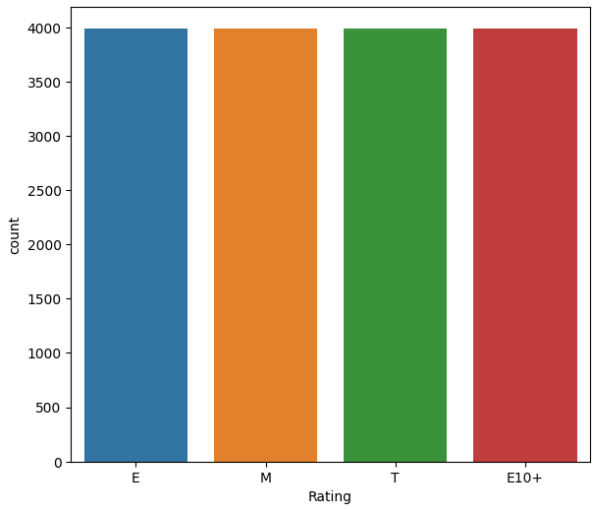
Out[191]:
```
Counter({'E': 3991, 'M': 1563, 'T': 2961, 'E10+': 1420})
```

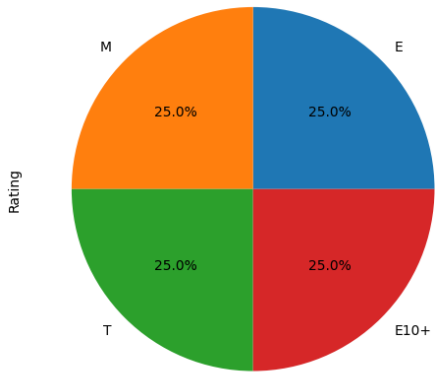
In [192]:
```
# onehot encoding  on selected features
x_new = pd.get_dummies(X, drop_first=True)
```

In [193]:
```
# import SMOTE to scale our data
from imblearn.over_sampling import SMOTE
sm = SMOTE(random_state=42, k_neighbors = 2)
```

In [194]:
```
# apply SMOTE to resample the dataset
X_res, y_res = sm.fit_resample(x_new, y) # The object is applied
X, y = X_res, y_res # reassigning the balanced dataset to X,y
```

In [143]:
```
# Plot of the dataset after applying smoot to know if the changes has been made
N_Video_Games_dataset = pd.concat([X,y], axis = 1)
fig, ax=plt.subplots(1,2,figsize=(15,6)) # creating the axis shell for subplot
a = sns.countplot(x='Rating',data=N_Video_Games_dataset, ax=ax[0]) # assigning e
a= N_Video_Games_dataset['Rating'].value_counts().plot.pie(autopct="%1.1f%%", ax
```





In [195]:
```
# encoding the target

from sklearn.preprocessing import LabelEncoder

label_e = LabelEncoder()
y = label_e.fit_transform(y)
```

In [196]:
```
#from typing import Counter
Counter(y)
```

Out[196]:
```
Counter({0: 3991, 2: 3991, 3: 3991, 1: 3991})
```

In [197]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_
```

In [198]:
```
# scaling our data
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

In [116]:
```
!pip install scikit-plot
```

```
Looking in indexes: https://pypi.org/simple, (https://pypi.org/simple,) http
s://us-python.pkg.dev/colab-wheels/public/simple/ (https://us-python.pkg.dev/co
lab-wheels/public/simple/)
Collecting scikit-plot
  Downloading scikit_plot-0.3.7-py3-none-any.whl (33 kB)
Requirement already satisfied: scipy>=0.9 in /usr/local/lib/python3.10/dist-pac
kages (from scikit-plot) (1.10.1)
Requirement already satisfied: scikit-learn>=0.18 in /usr/local/lib/python3.10/
dist-packages (from scikit-plot) (1.2.2)
Requirement already satisfied: matplotlib>=1.4.0 in /usr/local/lib/python3.10/d
ist-packages (from scikit-plot) (3.7.1)
Requirement already satisfied: joblib>=0.10 in /usr/local/lib/python3.10/dist-p
ackages (from scikit-plot) (1.2.0)
Requirement already satisfied: numpy>=1.20 in /usr/local/lib/python3.10/dist-pa
ckages (from matplotlib>=1.4.0->scikit-plot) (1.22.4)
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.10/dist-p
ackages (from matplotlib>=1.4.0->scikit-plot) (0.11.0)
Requirement already satisfied: python-dateutil>=2.7 in /usr/local/lib/python3.1
0/dist-packages (from matplotlib>=1.4.0->scikit-plot) (2.8.2)
Requirement already satisfied: pillow>=6.2.0 in /usr/local/lib/python3.10/dist-
packages (from matplotlib>=1.4.0->scikit-plot) (8.4.0)
Requirement already satisfied: pyparsing>=2.3.1 in /usr/local/lib/python3.10/di
st-packages (from matplotlib>=1.4.0->scikit-plot) (3.0.9)
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.10/dis
t-packages (from matplotlib>=1.4.0->scikit-plot) (23.1)
Requirement already satisfied: kiwisolver>=1.0.1 in /usr/local/lib/python3.10/d
ist-packages (from matplotlib>=1.4.0->scikit-plot) (1.4.4)
Requirement already satisfied: fonttools>=4.22.0 in /usr/local/lib/python3.10/d
ist-packages (from matplotlib>=1.4.0->scikit-plot) (4.39.3)
Requirement already satisfied: contourpy>=1.0.1 in /usr/local/lib/python3.10/di
st-packages (from matplotlib>=1.4.0->scikit-plot) (1.0.7)
Requirement already satisfied: threadpoolctl>=2.0.0 in /usr/local/lib/python3.1
0/dist-packages (from scikit-learn>=0.18->scikit-plot) (3.1.0)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.10/dist-packa
ges (from python-dateutil>=2.7->matplotlib>=1.4.0->scikit-plot) (1.16.0)
Installing collected packages: scikit-plot
Successfully installed scikit-plot-0.3.7
```

In [117]:
```
from scikitplot.metrics import plot_roc_curve
```

In [119]:
```
from sklearn.metrics import (accuracy_score, classification_report, confusion_ma
```

In [148]:
```
from sklearn.ensemble import RandomForestClassifier
# Initialize a decision tree classifier
rf_clf = RandomForestClassifier(random_state=0,)

# Fit the model on the training set
rf_clf.fit(X_train_scaled,y_train)
```

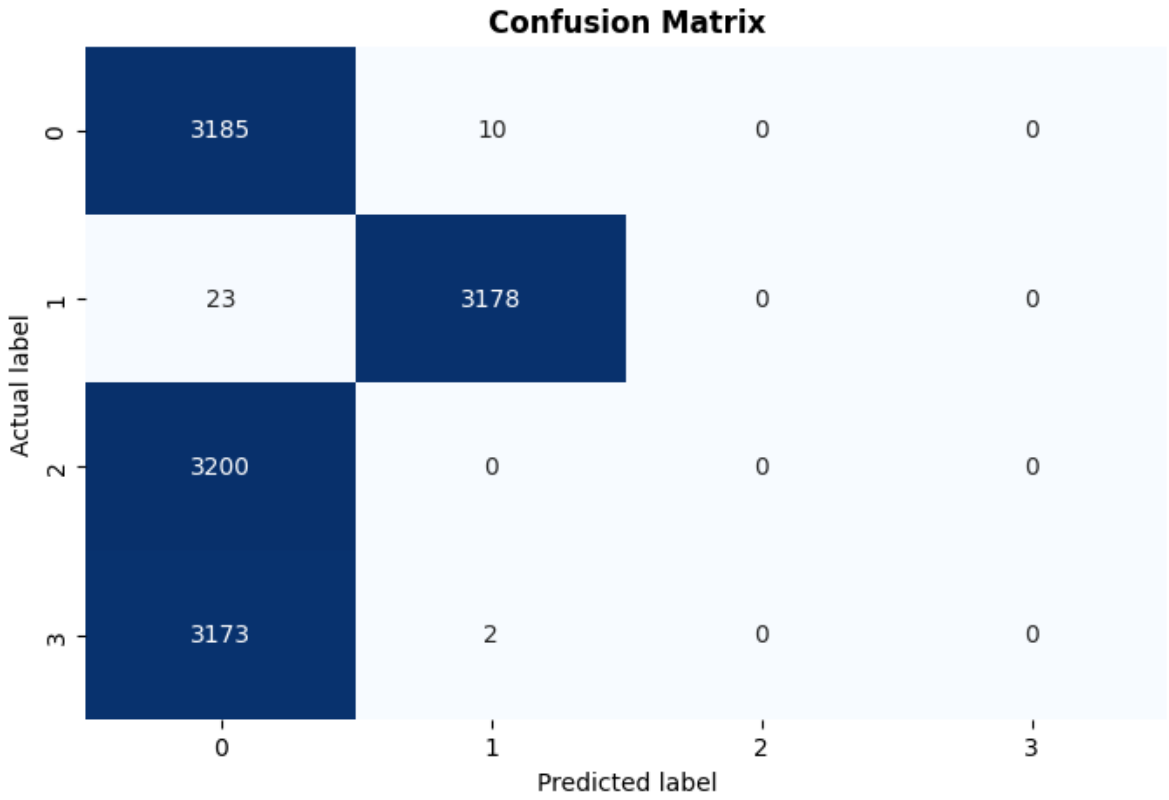
Out[148]:
```
RandomForestClassifier(random_state=0)
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [152]:

```
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_sc

def model_performance_classification(model, predictors, target, threshold=0.5):
    """
    Function to compute different metrics to check regression model performance

    model: classifier
    predictors: independent variables
    target: dependent variable
    """
    # predicting using the independent variables
    pred_proba = model.predict_proba(predictors)[:, 1]
    # convert the probability to class
    pred_class = np.round(pred_proba > threshold)

    # compute accuracy, recall, precision, and F1 score
    acc = accuracy_score(target, pred_class)
    recall = recall_score(target, pred_class, average="weighted")
    precision = precision_score(target, pred_class, average="weighted",zero_divi
    f1 = f1_score(target, pred_class, average="weighted")

    # create a dataframe of metrics
    Video_Games_dataset_perf = pd.DataFrame(
        {
            "Accuracy": acc,
            "Recall": recall,
            "Precision": precision,
            "F1-score": f1,
        },
        index=[0],
    )

    # create a confusion matrix
    conf = confusion_matrix(target, pred_class)

    # plot the confusion matrix with color-coded sections
    plt.figure(figsize=(8, 5))
    sns.heatmap(
        conf,
        annot=True,
        fmt="g",
        cmap=sns.color_palette("Blues", as_cmap=True),
        cbar=False,
    )
    plt.xlabel("Predicted label")
    plt.ylabel("Actual label")
    plt.title("Confusion Matrix", fontweight="bold")
    plt.show()

    return Video_Games_dataset_performance
```

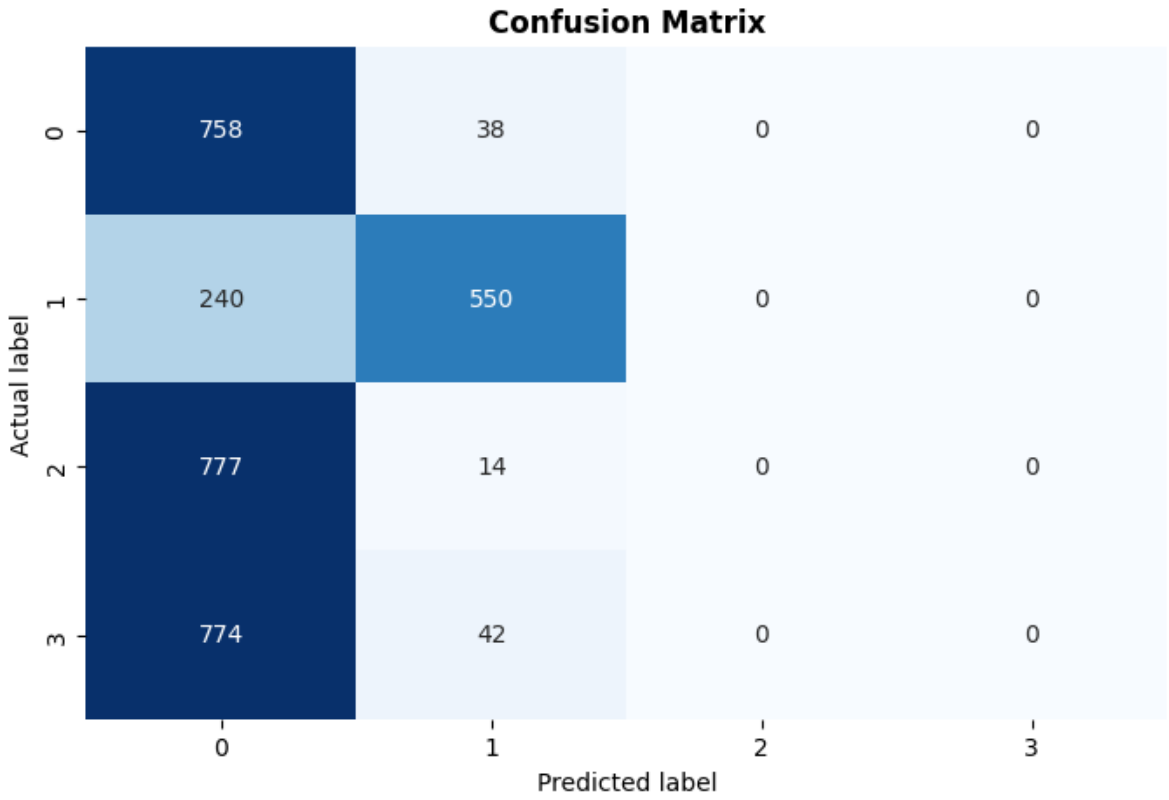
In [153]:
```
rf_train_cls = model_performance_classification(rf_clf,X_train_scaled,y_train)
rf_train_cls
```



Out[153]:

	Accuracy	Recall	Precision	F1-score
0	0.498238	0.498238	0.332869	0.374009

In [154]:
```
rf_train_cls_test = model_performance_classification(rf_clf,X_test_scaled,y_test
rf_train_cls_test
```



Out[154]:

	Accuracy	Recall	Precision	F1-score
0	0.409646	0.409646	0.285436	0.302773

In [200]:
```
# Initialize a decision tree classifier
rf_clf = RandomForestClassifier(random_state=0,)

# Fit the model on the training set
rf_clf.fit(X_train_scaled,y_train)

# Make predictions on the testing set
y_pred = rf_clf.predict(X_test_scaled)

# Calculate evaluation metrics
accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred, average = "weighted")
recall = recall_score(y_test, y_pred, average = "weighted")
f1 = f1_score(y_test, y_pred, average = "weighted")

# Create a confusion matrix
cm = confusion_matrix(y_test, y_pred)

# Print the evaluation metrics and confusion matrix
Video_Games_dataset_eval = pd.DataFrame({'Accuracy': [accuracy],
                         'Precision': [precision],
                         'Recall': [recall],
                         'F1-score': [f1]})

print("Evaluation Metrics:")
print(Video_Games_dataset_eval)
print()

# Print the classification report
print("Classification Report:")
print(classification_report(y_test, y_pred))

print("Confusion Matrix for the Testing Set:")
print(cm)
```

```
Evaluation Metrics:
   Accuracy  Precision    Recall  F1-score
0  0.740683   0.736322  0.740683  0.736341

Classification Report:
              precision    recall  f1-score   support

           0       0.76      0.79      0.77       796
           1       0.74      0.79      0.77       790
           2       0.78      0.84      0.81       791
           3       0.67      0.55      0.60       816

    accuracy                           0.74      3193
   macro avg       0.74      0.74      0.74      3193
weighted avg       0.74      0.74      0.74      3193

Confusion Matrix:
[[629  76  14  77]
 [ 72 627  29  62]
 [ 11  34 664  82]
 [119 107 145 445]]
```

to check if the model did not overfit

In [201]:
```
# Make predictions on the training set
y_train_pred = rf_clf.predict(X_train_scaled)

# Calculate evaluation metrics
train_accuracy = accuracy_score(y_train, y_train_pred)
train_precision = precision_score(y_train, y_train_pred, average="weighted")
train_recall = recall_score(y_train, y_train_pred, average="weighted")
train_f1 = f1_score(y_train, y_train_pred, average="weighted")

# Create a confusion matrix
train_cm = confusion_matrix(y_train, y_train_pred)

# Print the evaluation metrics and confusion matrix for the training set
Video_Games_dataset_train_eval = pd.DataFrame({'Accuracy': [train_accuracy],
                         'Precision': [train_precision],
                         'Recall': [train_recall],
                         'F1-score': [train_f1]})
print("Evaluation Metrics on the Training Set:")
print(Video_Games_dataset_train_eval)

print("Confusion Matrix for the Training Set:")
print(train_cm)
```

```
Evaluation Metrics on the Training Set:
   Accuracy  Precision   Recall  F1-score
0   0.99585   0.995858  0.99585  0.995852
Confusion Matrix for the Training Set:
[[3180   12    0    3]
 [  18 3180    0    3]
 [   0    0 3200    0]
 [  13    4    0 3158]]
```

training and testin together

In [202]:
```
# Make predictions on the training set
y_train_pred = rf_clf.predict(X_train_scaled)

# Calculate evaluation metrics
train_accuracy = accuracy_score(y_train, y_train_pred)
train_precision = precision_score(y_train, y_train_pred, average="weighted")
train_recall = recall_score(y_train, y_train_pred, average="weighted")
train_f1 = f1_score(y_train, y_train_pred, average="weighted")

# Create a confusion matrix
train_cm = confusion_matrix(y_train, y_train_pred)

# Print the evaluation metrics and confusion matrix for the training set
Video_Games_dataset_train_eval = pd.DataFrame({'Accuracy': [train_accuracy],
                         'Precision': [train_precision],
                         'Recall': [train_recall],
                         'F1-score': [train_f1]})
print("Evaluation Metrics on Rating Training Set:")
print(Video_Games_dataset_train_eval)

print("Confusion Matrix for Rating Training Set:")
print(train_cm)


# Make predictions on the testing set
y_pred = rf_clf.predict(X_test_scaled)

# Calculate evaluation metrics
accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred, average = "weighted")
recall = recall_score(y_test, y_pred, average = "weighted")
f1 = f1_score(y_test, y_pred, average = "weighted")

# Create a confusion matrix
cm = confusion_matrix(y_test, y_pred)

# Print the evaluation metrics and confusion matrix for the testing set
Video_Games_dataset_eval = pd.DataFrame({'Accuracy': [accuracy],
                         'Precision': [precision],
                         'Recall': [recall],
                         'F1-score': [f1]})
print("\nEvaluation Metrics on Rating Testing Set:")
print(Video_Games_dataset_eval)

print("Confusion Matrix for Rating Testing Set:")
print(cm)
```

```
Evaluation Metrics on the Training Set:
   Accuracy  Precision   Recall  F1-score
0   0.99585   0.995858  0.99585  0.995852
Confusion Matrix for the Training Set:
[[3180   12    0    3]
 [  18 3180    0    3]
 [   0    0 3200    0]
 [  13    4    0 3158]]

Evaluation Metrics on the Testing Set:
   Accuracy  Precision    Recall  F1-score
0  0.740683   0.736322  0.740683  0.736341
Confusion Matrix for the Testing Set:
[[629  76  14  77]
 [ 72 627  29  62]
 [ 11  34 664  82]
 [119 107 145 445]]
```

Genre as the target

In [224]:
```
Video_Games_dataset["Genre"].value_counts()
```

Out[224]:
```
Action          3370
Sports          2348
Misc            1750
Role-Playing    1500
Shooter         1323
Adventure       1303
Racing          1249
Platform         888
Simulation       874
Fighting         849
Strategy         683
Puzzle           580
Name: Genre, dtype: int64
```

In [225]:
```
# Select features and target
X_g =Video_Games_dataset[["Rating", "NA_Sales", "JP_Sales", "EU_Sales", "Other_S
        "User_Score", "User_Count", "New_platform", "Global_Sales"]]

y_g = Video_Games_dataset["Genre"]
```

In [226]:
```
# onehot encoding  on selected features
x_new_g = pd.get_dummies(X_g, drop_first=True)

#x_new_g
```

In [227]:
```
label_e = LabelEncoder()
y_new_g = label_e.fit_transform(y_g)
```

In [228]:
```
#from typing import Counter
Counter(y_new_g)
```

Out[228]:
```
Counter({10: 2348,
         4: 888,
         6: 1249,
         7: 1500,
         5: 580,
         3: 1750,
         8: 1323,
         9: 874,
         0: 3370,
         2: 849,
         1: 1303,
         11: 683})
```

In [229]:
```
# applying smooth
sm = SMOTE(random_state=42, k_neighbors = 2) #
X_res, y_res = sm.fit_resample(x_new_g, y_new_g) # The object is applied
X, y = X_res, y_res
```

In [230]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_
```

In [231]:
```
# scaling our data
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()
X_train_g = scaler.fit_transform(X_train)
X_test_g = scaler.transform(X_test)
```

In [185]:
```
# Initialize a decision tree classifier
rf_clf_g = RandomForestClassifier(random_state=0,)

# Fit the model on the training set
rf_clf_g.fit(X_train_g,y_train)
```

Out[185]:
```
RandomForestClassifier(random_state=0)
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

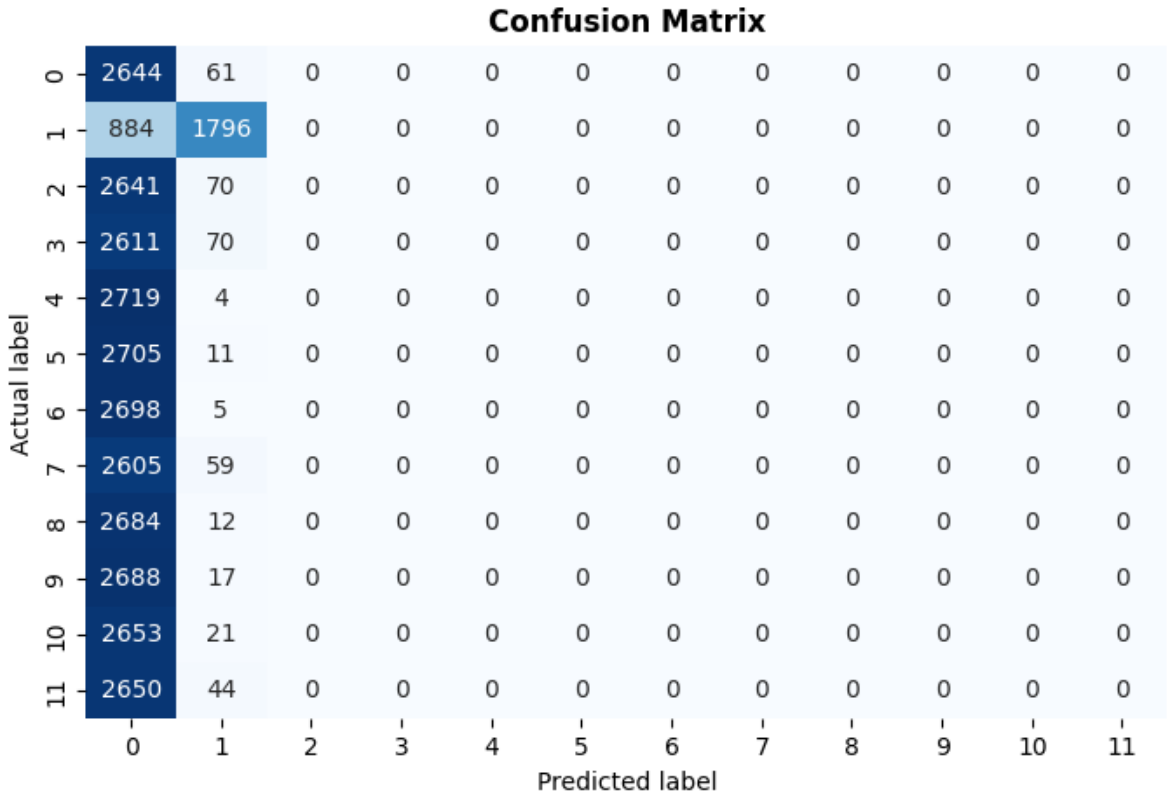
In [233]:
```
# Initialize a decision tree classifier
rf_clf_g = RandomForestClassifier(random_state=0,)

# Fit the model on the training set
rf_clf_g.fit(X_train_g,y_train)

# Make predictions on the training set
y_train_pred_g = rf_clf_g.predict(X_train_g)

# Calculate evaluation metrics
train_accuracy_g = accuracy_score(y_train, y_train_pred_g)
train_precision_g = precision_score(y_train, y_train_pred_g, average="weighted")
train_recall_g = recall_score(y_train, y_train_pred_g, average="weighted")
train_f1_g = f1_score(y_train, y_train_pred_g, average="weighted")

# Create a confusion matrix
train_cm_g = confusion_matrix(y_train, y_train_pred_g)

# Print the evaluation metrics and confusion matrix for the training set
Video_Games_dataset_train_eval_g = pd.DataFrame({'Accuracy': [train_accuracy_g],
                          'Precision': [train_precision_g],
                          'Recall': [train_recall_g],
                          'F1-score': [train_f1_g]})
print("Evaluation Metrics on Genre Training Set:")
print(f"{Video_Games_dataset_train_eval_g}\n")

# Classification report for training set
print("\nClassification Report for Genre Training Set:")
print(f"{classification_report(y_train, y_train_pred_g)}\n")

print("Confusion Matrix for Genre Training Set:")
print(train_cm_g)
print()

# Make predictions on the testing set
y_pred_g = rf_clf_g.predict(X_test_g)

# Calculate evaluation metrics
accuracy_g = accuracy_score(y_test, y_pred_g)
precision_g = precision_score(y_test, y_pred_g, average = "weighted")
recall_g = recall_score(y_test, y_pred_g, average = "weighted")
f1_g = f1_score(y_test, y_pred_g, average = "weighted")

# Create a confusion matrix
cm_g = confusion_matrix(y_test, y_pred_g)

# Print the evaluation metrics and confusion matrix for the testing set
Video_Games_dataset_eval_g = pd.DataFrame({'Accuracy': [accuracy_g],
                          'Precision': [precision_g],
                          'Recall': [recall_g],
                          'F1-score': [f1_g]})
print("\nEvaluation Metrics on Genre Testing Set:")
print(f"{Video_Games_dataset_eval_g}\n")

# Classification report for testing set
print("\nClassification Report for Genre Testing Set:")
print(f"{classification_report(y_test, y_pred_g)}\n")

print("Confusion Matrix for Genre Testing Set:")
```

```
print(cm_g)
```

```
Evaluation Metrics on Genre Training Set:
   Accuracy  Precision    Recall  F1-score
0  0.878369   0.885849  0.878369  0.879791


Classification Report for Genre Training Set:
              precision    recall  f1-score   support

           0       0.96      0.80      0.87      2705
           1       0.65      0.87      0.75      2680
           2       0.89      0.88      0.88      2711
           3       0.83      0.79      0.81      2681
           4       0.92      0.96      0.94      2723
           5       0.91      0.93      0.92      2716
           6       0.94      0.92      0.93      2703
           7       0.88      0.81      0.84      2664
           8       0.95      0.94      0.95      2696
           9       0.92      0.89      0.90      2705
          10       0.92      0.84      0.88      2674
          11       0.86      0.90      0.88      2694

    accuracy                           0.88     32352
   macro avg       0.89      0.88      0.88     32352
weighted avg       0.89      0.88      0.88     32352


Confusion Matrix for Genre Training Set:
[[2157  212   30   59   21   27   21   58   21   17   33   49]
 [   6 2331   39   97   32   18   18   51   14   10   17   47]
 [   4  153 2383   13   44   16   16   17   13    9    9   34]
 [  13  220   32 2112   25   54   14   44   13   64   44   46]
 [   4    6   12   18 2626    6   22    3    5    6   11    4]
 [   3   24    5   31   13 2525   16   22   11   35    6   25]
 [   9   29   38   15   24   21 2500    6    7    9   23   22]
 [  13  236   45   49   24   21    6 2168   20   12   12   58]
 [   6   25   31   14    7   23    8    5 2535    5    9   28]
 [   4  128   13   45   10   19    2   22    7 2394   20   41]
 [  20   72   39   56   26   38   38   51   16   33 2251   34]
 [   8  132   14   21    4   15    5   22    2   11   25 2435]]


Evaluation Metrics on Genre Testing Set:
   Accuracy  Precision    Recall  F1-score
0  0.549827   0.546553  0.549827  0.546669


Classification Report for Genre Testing Set:
              precision    recall  f1-score   support

           0       0.40      0.36      0.38       665
           1       0.45      0.54      0.49       690
           2       0.60      0.62      0.61       659
           3       0.44      0.37      0.40       689
           4       0.60      0.66      0.63       647
           5       0.66      0.72      0.69       654
           6       0.56      0.55      0.56       667
           7       0.54      0.48      0.51       706
           8       0.58      0.61      0.59       674
           9       0.57      0.56      0.57       665
          10       0.54      0.46      0.50       696
          11       0.62      0.67      0.65       676
```

```
              accuracy                           0.55      8088
             macro avg       0.55      0.55      0.55      8088
          weighted avg       0.55      0.55      0.55      8088


Confusion Matrix for Genre Testing Set:
[[239  53  33  45  25  20  41  58  73  20  39  19]
 [ 26 375  23  46  23  25  20  36  38  26  20  32]
 [ 20  53 411  18  23  14   6  28  28  14  14  30]
 [ 46  74  22 254  33  41  39  37  16  53  42  32]
 [ 24  13  20  26 428  21  32  13  18  12  28  12]
 [  9  23   8  26   9 474  17  12   6  39  10  21]
 [ 36  23  21  30  42  20 368  10  22  24  45  26]
 [ 45  80  58  22  24  19  12 340  40  23  15  28]
 [ 64  11  24  13  32  18  21  30 408  11  18  24]
 [ 19  47  26  34  23  35  20  12  17 373  24  35]
 [ 47  30  25  51  41  23  60  29  23  26 323  18]
 [ 16  50  18  16  11   9  16  23  12  33  18 454]]
```

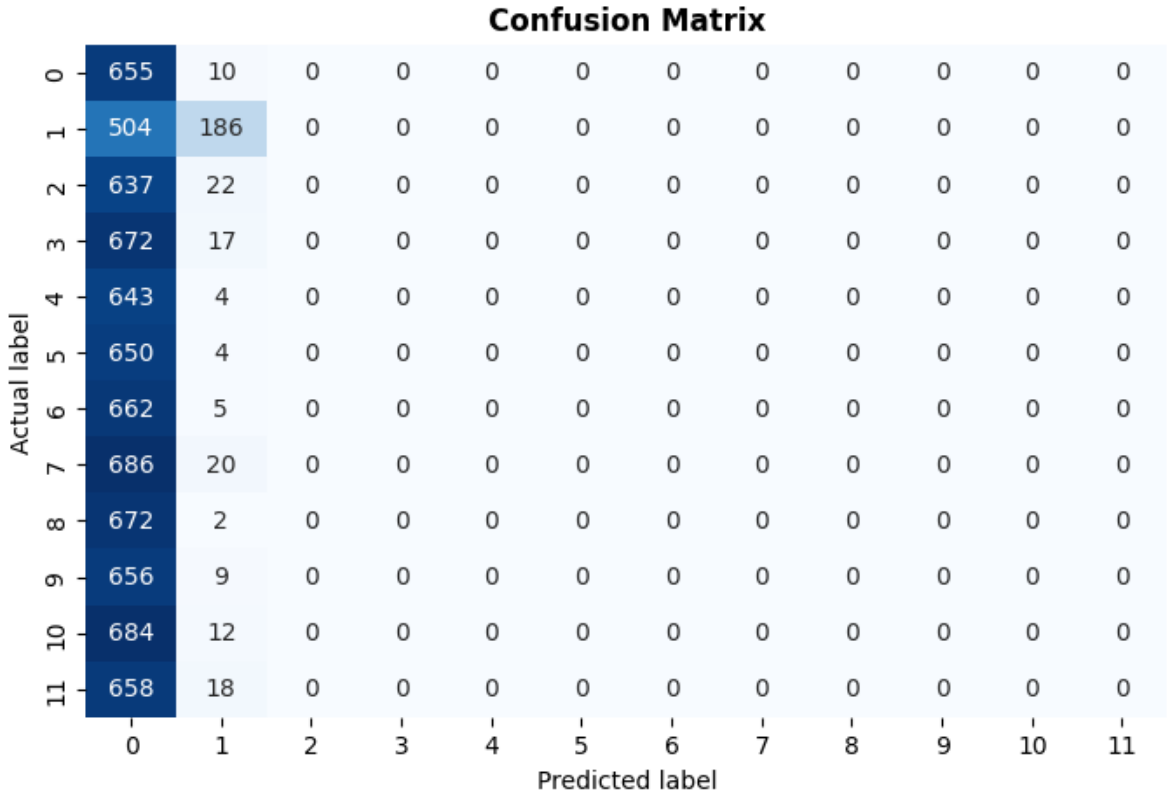
In [186]:
```
# evaluate the traing data
rf_train_g = model_performance_classification(rf_clf_g,X_train_g,y_train)
rf_train_g
```



Out[186]:

	Accuracy	Recall	Precision	F1-score
0	0.13724	0.13724	0.075886	0.074796

In [187]:
```
# evaluate the testing data
rf_test_g = model_performance_classification(rf_clf_g,X_test_g,y_test)
rf_test_g
```



Out[187]:

	Accuracy	Recall	Precision	F1-score
0	0.103981	0.103981	0.058276	0.044523

New plat form as the target

In [234]:
```
Video_Games_dataset["New_platform"].value_counts
```

Out[234]:
```
<bound method IndexOpsMixin.value_counts of 0          Wii
1       Others
2          Wii
3          Wii
4       Others
         ...
16712      PS3
16713     X360
16714       PS
16715   Others
16716       PS
Name: New_platform, Length: 16717, dtype: object>
```

In [235]:
```
# Select features and target
X_n =Video_Games_dataset[["Rating", "NA_Sales", "JP_Sales", "EU_Sales", "Other_S
        "User_Score", "User_Count", "Genre", "Global_Sales"]]

y_n = Video_Games_dataset["New_platform"]
```

In [236]:
```
# onehot encoding  on selected features
x_new_n = pd.get_dummies(X_n, drop_first=True)

#x_new_g
```

In [237]:
```
label_e = LabelEncoder()
y_new_n = label_e.fit_transform(y_n)
```

In [238]:
```
# applying smooth
sm = SMOTE(random_state=42, k_neighbors = 2) #
X_res, y_res = sm.fit_resample(x_new_n, y_new_n) # The object is applied
X, y = X_res, y_res
```

In [239]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_
```

In [240]:
```
# scaling our data
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()
X_train_n = scaler.fit_transform(X_train)
X_test_n = scaler.transform(X_test)
```

In [243]:
```
# Initialize a decision tree classifier
rf_clf_n = RandomForestClassifier(random_state=0,)

# Fit the model
rf_clf_n.fit(X_train_n,y_train)

# Make predictions
y_train_pred_n = rf_clf_n.predict(X_train_n)

# Calculate evaluation metrics
train_accuracy_n = accuracy_score(y_train, y_train_pred_n)
train_precision_n = precision_score(y_train, y_train_pred_n, average="weighted")
train_recall_n = recall_score(y_train, y_train_pred_n, average="weighted")
train_f1_n = f1_score(y_train, y_train_pred_n, average="weighted")

# Create a confusion matrix
train_cm_n = confusion_matrix(y_train, y_train_pred_n)

# Print the evaluation metrics and confusion matrix for the training set
Video_Games_dataset_train_eval_n = pd.DataFrame({'Accuracy': [train_accuracy_n],
                          'Precision': [train_precision_n],
                          'Recall': [train_recall_n],
                          'F1-score': [train_f1_n]})
print("Evaluation Metrics on Platform Training Set:")
print(f"{Video_Games_dataset_train_eval_n}\n")

# Classification report for training set
print("\nClassification Report for Platform Training Set:")
print(f"{classification_report(y_train, y_train_pred_n)}\n")

print("Confusion Matrix for Platform Training Set:")
print(train_cm_n)
print()

# Make predictions on the testing set
y_pred_n = rf_clf_n.predict(X_test_n)

# Calculate evaluation metrics
accuracy_n = accuracy_score(y_test, y_pred_n)
precision_n = precision_score(y_test, y_pred_n, average = "weighted")
recall_n = recall_score(y_test, y_pred_n, average = "weighted")
f1_n = f1_score(y_test, y_pred_n, average = "weighted")

# Create a confusion matrix
cm_n = confusion_matrix(y_test, y_pred_n)

# Print the evaluation metrics and confusion matrix for the testing set
Video_Games_dataset_eval_n = pd.DataFrame({'Accuracy': [accuracy_n],
                          'Precision': [precision_n],
                          'Recall': [recall_n],
                          'F1-score': [f1_n]})
print("\nEvaluation Metrics on Platform Testing Set:")
print(f"{Video_Games_dataset_eval_n}\n")

# Classification report for testing set
print("\nClassification Report for Platform Testing Set:")
print(f"{classification_report(y_test, y_pred_n)}\n")

print("Confusion Matrix for Platform Testing Set:")
```

```
print(cm_n)
```

```
Evaluation Metrics on Platform Training Set:
   Accuracy  Precision    Recall  F1-score
0  0.906243   0.917849  0.906243  0.908992


Classification Report for Platform Training Set:
              precision    recall  f1-score   support

           0       0.84      0.83      0.84      3677
           1       0.96      0.93      0.95      3658
           2       0.96      0.87      0.91      3667
           3       0.95      0.87      0.91      3706
           4       0.98      0.90      0.94      3692
           5       0.70      0.94      0.80      3679
           6       0.96      0.95      0.96      3687
           7       0.99      0.97      0.98      3629

    accuracy                           0.91     29395
   macro avg       0.92      0.91      0.91     29395
weighted avg       0.92      0.91      0.91     29395


Confusion Matrix for Platform Training Set:
[[3059   38   34   50   21  393   78    4]
 [  86 3390   18   38   10   88   18   10]
 [  88   13 3183   13   18  348    4    0]
 [ 109   30   10 3216    8  326    1    6]
 [  66   16   30   18 3317  233    5    7]
 [ 140   13   19   36   12 3447   12    0]
 [  83    8   12   16    5   42 3517    4]
 [  13    5    4    1    2   82   12 3510]]


Evaluation Metrics on Platform Testing Set:
   Accuracy  Precision    Recall  F1-score
0  0.708532   0.713388  0.708532  0.708671


Classification Report for Platform Testing Set:
              precision    recall  f1-score   support

           0       0.59      0.53      0.56       916
           1       0.78      0.79      0.79       935
           2       0.79      0.70      0.74       926
           3       0.72      0.70      0.71       887
           4       0.75      0.72      0.73       901
           5       0.58      0.76      0.66       914
           6       0.72      0.66      0.69       906
           7       0.78      0.79      0.78       964

    accuracy                           0.71      7349
   macro avg       0.71      0.71      0.71      7349
weighted avg       0.71      0.71      0.71      7349


Confusion Matrix for Platform Testing Set:
[[488  55  33  45  19 146 107  23]
 [ 38 739  36  25   9  41  17  30]
 [ 41  39 649  21  57  87  15  17]
 [ 32  37  23 623  42  98  13  19]
 [ 31  20  34  34 649  61  17  55]
 [ 64  15  15  51  39 695  23  12]
```

```
 [101  16  14  44  29  37 600  65]
 [ 27  21  19  26  27  39  41 764]]
```

In [247]:
```
print("\nModel performance using  Platform as target:")
print(f"{Video_Games_dataset_eval_n}\n")

print("\nModel performance using  Genre as target:")
print(f"{Video_Games_dataset_eval_g}\n")

print("\nModel performance using  Rating as target:")
print(Video_Games_dataset_eval)
```

```
Model performance using  Platform as target:
   Accuracy  Precision    Recall  F1-score
0  0.708532   0.713388  0.708532  0.708671


Model performance using  Genre as target:
   Accuracy  Precision    Recall  F1-score
0  0.549827   0.546553  0.549827  0.546669


Model performance using  Rating as target:
   Accuracy  Precision    Recall  F1-score
0  0.740683   0.736322  0.740683  0.736341
```

checking the performance of different regressors

In [273]:
```
# subsetting the rating column
N_Video_Games_dataset=Video_Games_dataset[Video_Games_dataset["Rating"].isin(["E
```

In [274]:
```
X = N_Video_Games_dataset[["Genre", "NA_Sales", "JP_Sales", "EU_Sales", "Other_S
        "User_Score", "User_Count", "New_platform", "Global_Sales"]]
y = N_Video_Games_dataset['Rating']
```

In [275]:
```
# onehot encoding  on selected features
x_new = pd.get_dummies(X, drop_first=True)
```

In [276]:
```
# encoding the target

from sklearn.preprocessing import LabelEncoder

label_e = LabelEncoder()
y = label_e.fit_transform(y)
```

In [277]:
```
# instantiting smote

sm = SMOTE(random_state=42, k_neighbors = 2)
```

In [278]:
```
# apply SMOTE to resample the dataset
X_res, y_res = sm.fit_resample(x_new, y) # The object is applied
X, y = X_res, y_res
```

In [279]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_
```

In [281]:
```
# scaling our data

scaler = MinMaxScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

In [285]:
```
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_sc
from sklearn.model_selection import train_test_split


# Initialize a dictionary to store the models and their evaluation metrics
models = {'Decision Tree': DecisionTreeClassifier(),
          'Random Forest': RandomForestClassifier(random_state=0),
          'KNeighbor':KNeighborsClassifier(),
          'Support vector':SVC(random_state=42),
          'Logistic Regression': LogisticRegression(max_iter=1000)}

# Initialize a DataFrame to store the evaluation metrics
Video_Games_dataset_eval = pd.DataFrame(columns=['Model', 'Accuracy', 'Precision

# Loop over the models
for name, model in models.items():
    # Fit the model on the training data
    model.fit(X_train_scaled, y_train)

    # Predict the target values on the testing data
    y_pred = model.predict(X_test_scaled)

    # Calculate the evaluation metrics
    acc = accuracy_score(y_test, y_pred)
    prec = precision_score(y_test, y_pred,  average = "weighted")
    rec = recall_score(y_test, y_pred, average = "weighted")
    f1 = f1_score(y_test, y_pred, average = "weighted")


   # Add evaluation metrics to the DataFrame
    Video_Games_dataset_eval = pd.concat([Video_Games_dataset_eval, pd.DataFrame
                               'Accuracy': [acc],
                               'Precision': [prec],
                               'Recall': [rec],
                               'F1-score': [f1]})], ignore_index=True)

    # print classication report  of each model
    print(f"Classfication report for {name}:\n{classification_report(y_test, y_p

    # Print the confusion matrix for each model
    print(f"Confusion matrix for {name}:\n{confusion_matrix(y_test, y_pred)}\n")

# Print the evaluation metrics DataFrame
print(Video_Games_dataset_eval)
```

```
Classfication report for Decision Tree:
              precision    recall  f1-score   support

           0       0.67      0.67      0.67       796
           1       0.67      0.67      0.67       790
           2       0.71      0.74      0.73       791
           3       0.52      0.50      0.51       816

    accuracy                           0.64      3193
   macro avg       0.64      0.65      0.64      3193
weighted avg       0.64      0.64      0.64      3193


Confusion matrix for Decision Tree:
[[531 105  33 127]
 [ 95 530  50 115]
 [ 22  46 589 134]
 [142 108 158 408]]

Classfication report for Random Forest:
              precision    recall  f1-score   support

           0       0.76      0.79      0.77       796
           1       0.74      0.79      0.77       790
           2       0.78      0.84      0.81       791
           3       0.67      0.55      0.60       816

    accuracy                           0.74      3193
   macro avg       0.74      0.74      0.74      3193
weighted avg       0.74      0.74      0.74      3193


Confusion matrix for Random Forest:
[[629  76  14  77]
 [ 72 627  29  62]
 [ 11  34 664  82]
 [119 107 145 445]]

Classfication report for KNeighbor:
              precision    recall  f1-score   support

           0       0.72      0.76      0.74       796
           1       0.64      0.73      0.68       790
           2       0.69      0.75      0.72       791
           3       0.60      0.44      0.51       816

    accuracy                           0.67      3193
   macro avg       0.66      0.67      0.66      3193
weighted avg       0.66      0.67      0.66      3193


Confusion matrix for KNeighbor:
[[601  99  17  79]
 [ 90 573  68  59]
 [ 15  81 590 105]
 [129 149 178 360]]

Classfication report for Support vector:
              precision    recall  f1-score   support

           0       0.70      0.79      0.74       796
           1       0.64      0.65      0.64       790
```

```
              2       0.66      0.76      0.71       791
              3       0.60      0.43      0.50       816

    accuracy                           0.66      3193
   macro avg       0.65      0.66      0.65      3193
weighted avg       0.65      0.66      0.65      3193


Confusion matrix for Support vector:
[[630  72  18  76]
 [126 510  92  62]
 [ 20  75 603  93]
 [126 144 196 350]]

Classfication report for Logistic Regression:
              precision    recall  f1-score   support

           0       0.69      0.78      0.73       796
           1       0.62      0.60      0.61       790
           2       0.66      0.72      0.69       791
           3       0.55      0.45      0.49       816

    accuracy                           0.63      3193
   macro avg       0.63      0.64      0.63      3193
weighted avg       0.63      0.63      0.63      3193


Confusion matrix for Logistic Regression:
[[618  74  20  84]
 [134 473  85  98]
 [ 13  91 571 116]
 [134 126 191 365]]

                 Model  Accuracy  Precision    Recall  F1-score
0        Decision Tree  0.644535   0.642556  0.644535  0.643387
1        Random Forest  0.740683   0.736322  0.740683  0.736341
2            KNeighbor  0.665205   0.660526  0.665205  0.658781
3       Support vector  0.655496   0.649937  0.655496  0.647406
4  Logistic Regression  0.634826   0.628396  0.634826  0.629172
```

kfold cross_validation

In [286]:
```
# Import the necessary libraries
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import cross_val_score, StratifiedKFold
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_sc
# Initialize a random forest classifier
rf = RandomForestClassifier()

# Set k values
k_values = [5, 10, 15]

# Initialize a DataFrame to store the evaluation metrics
Video_Games_dataset_eval = pd.DataFrame(columns=['k', 'Accuracy', 'Precision', '

# Loop over k values
for k in k_values:
    # Set up cross-validation
    cv = StratifiedKFold(n_splits=k, shuffle=True, random_state=42)

    # Perform cross-validation on the training set using evaluation metrics
    acc_scores = cross_val_score(rf, X, y, cv=cv, scoring='accuracy')
    prec_scores = cross_val_score(rf, X, y, cv=cv, scoring='precision_weighted')
    rec_scores = cross_val_score(rf, X, y, cv=cv, scoring='recall_weighted')
    f1_scores = cross_val_score(rf, X, y, cv=cv, scoring='f1_weighted')

    # Calculate the mean of the evaluation metrics
    acc_mean = np.mean(acc_scores)
    prec_mean = np.mean(prec_scores)
    rec_mean = np.mean(rec_scores)
    f1_mean = np.mean(f1_scores)

    # Add evaluation metrics to the DataFrame
    Video_Games_dataset_eval = pd.concat([Video_Games_dataset_eval, pd.DataFrame
                                          'Accuracy': [acc_mean],
                                          'Precision': [prec_mean],
                                          'Recall': [rec_mean],
                                          'F1-score': [f1_mean]})], ignore_

# Print the evaluation metrics DataFrame
print(Video_Games_dataset_eval)
```

```
    k  Accuracy  Precision    Recall  F1-score
0   5  0.726760   0.724216  0.726259  0.724146
1  10  0.733399   0.731287  0.734965  0.731039
2  15  0.737912   0.736076  0.741106  0.732439
```

In [287]:
```
# Define classifier
clf = DecisionTreeClassifier()

# Define cross-validation strategy
cv = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)

# Perform cross-validation
scores = cross_val_score(clf, X, y, cv=cv)

# Print mean and standard deviation of scores
print('Cross-validation scores: ', scores)
print('Mean score: ', scores.mean())
print('Standard deviation: ', scores.std())

# Fit classifier on entire dataset
clf.fit(X, y)

# Make predictions on testing data
y_pred = clf.predict(X)

# Print classification report
print('Classification Report:')
print(classification_report(y, y_pred))

# Print confusion matrix
print('Confusion Matrix:')
print(confusion_matrix(y, y_pred))
```

```
Cross-validation scores:  [0.64578766 0.63764485 0.64296899 0.63075478 0.631892
23]
Mean score:  0.6378097019875086
Standard deviation:  0.005917461407661162
Classification Report:
              precision    recall  f1-score   support

           0       0.98      1.00      0.99      3991
           1       1.00      0.99      0.99      3991
           2       1.00      1.00      1.00      3991
           3       1.00      0.99      1.00      3991

    accuracy                           1.00     15964
   macro avg       1.00      1.00      1.00     15964
weighted avg       1.00      1.00      1.00     15964

Confusion Matrix:
[[3985    5    0    1]
 [  37 3954    0    0]
 [   0    0 3991    0]
 [  25    9    0 3957]]
```

In [67]:
```
Video_Games_dataset.sample(5)
```

Out[67]:

	Name	Platform	Year_of_Release	Genre	Publisher	NA_Sales	EU_Sales	JP_Sa
14796	Negima!? Dream Tactic Yumemiru Otome Princess	PS2	2007	Strategy	Konami Digital Entertainment	0	0	
7263	Naruto Shippuden: Ultimate Ninja Storm Revolution	X360	2014	Fighting	Namco Bandai Games	0	0	
2284	AMF Bowling Pinbusters!	Wii	2007	Sports	Bethesda Softworks	0	0	
15188	Harukanaru Toki no Naka de 3 with Izayoiki Aiz...	PSP	2009	Adventure	Tecmo Koei	0	0	
10737	Natural Doctrine	PS4	2014	Role-Playing	Nippon Ichi Software	0	0	

In [69]:
```
# Drop irrelevant columns
Video_Games_dataset2 = Video_Games_dataset.drop(["Name", "Platform", "Year_of_Re
                "Publisher", "Developer", "Global_Sales"], axis = 1)
```

In [70]:
```
# select numeric variables
x = Video_Games_dataset2.select_dtypes(include="number")
x
```

Out[70]:

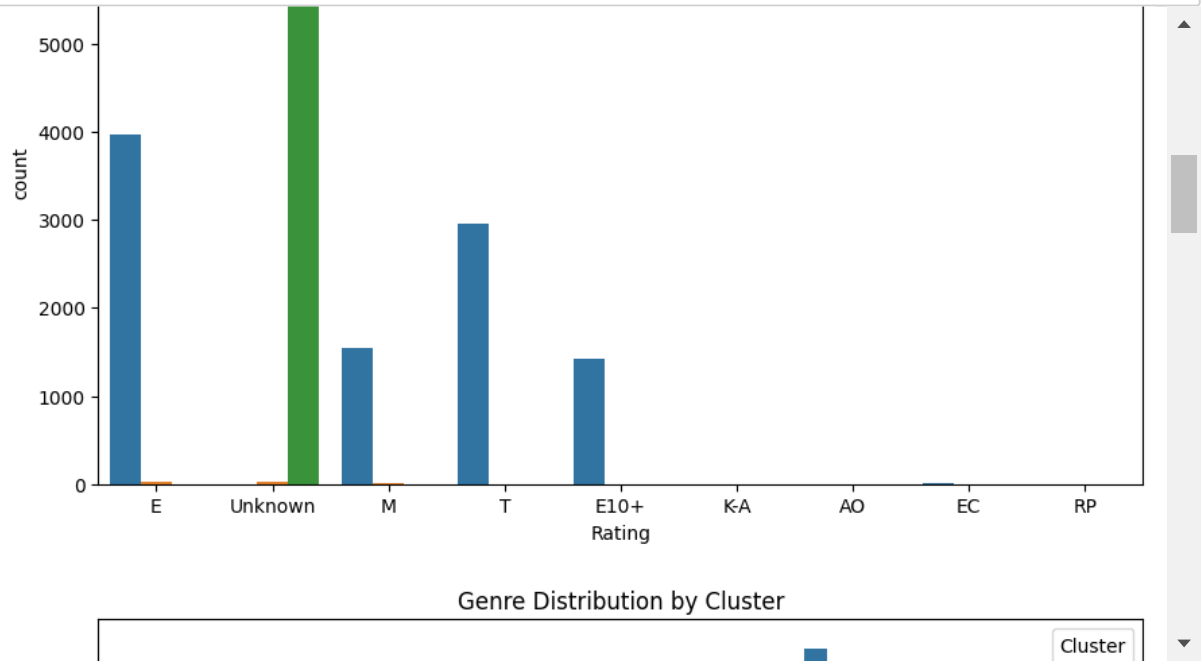
	NA_Sales	EU_Sales	JP_Sales	Other_Sales	Critic_Score	Critic_Count	User_Count
0	41	28	3	8	76	51	322
1	29	3	6	0	0	0	0
2	15	12	3	3	82	73	709
3	15	10	3	2	80	73	192
4	11	8	10	1	0	0	0
...	...	...	...	...	...	...	...
16714	0	0	0	0	0	0	0
16715	0	0	0	0	0	0	0
16716	0	0	0	0	0	0	0
16717	0	0	0	0	0	0	0
16718	0	0	0	0	0	0	0

16719 rows × 7 columns

In [87]:
```
# Clustering
preprocessor = make_column_transformer(
    (StandardScaler(), ['Global_Sales']),
    (OneHotEncoder(), ['Genre', 'Rating'])
)
X = preprocessor.fit_transform(Video_Games_dataset)

# Apply K-Means Clustering
kmeans = KMeans(n_clusters=3, random_state=0)  # Adjust n_clusters as needed
clusters = kmeans.fit_predict(X)

# Internal Evaluation: Silhouette Score
silhouette_avg = silhouette_score(X, clusters)
print(f"Silhouette Score: {silhouette_avg}")

# External Evaluation: Compare clusters with another categorical variable (e.g.,
# Assign the most frequent platform to each cluster
Video_Games_dataset['Cluster'] = clusters
most_common_platforms = Video_Games_dataset.groupby('Cluster')['Platform'].agg(l

# Print the most common platform in each cluster
print("Most Common Platform in each Cluster:")
print(most_common_platforms)
```

```
Silhouette Score: 0.18944097999107892
Most Common Platform in each Cluster:
Cluster
0    PS2
1    Wii
2     PS
Name: Platform, dtype: object
```

In [90]:
```
# Analyze distribution of 'Global_Sales' in each cluster
plt.figure(figsize=(10, 6))
sns.boxplot(x='Cluster', y='Global_Sales', data=Video_Games_dataset)
plt.title('Global Sales Distribution by Cluster')
plt.show()

# Analyze distribution of 'Rating' in each cluster
plt.figure(figsize=(10, 6))
sns.countplot(x='Rating', hue='Cluster', data=Video_Games_dataset)
plt.title('Rating Distribution by Cluster')
plt.show()

# Analyze distribution of 'Genre' in each cluster
plt.figure(figsize=(10, 6))
sns.countplot(x='Genre', hue='Cluster', data=Video_Games_dataset)
plt.xticks(rotation=45)
plt.title('Genre Distribution by Cluster')
plt.show()

# Summary statistics for each cluster
for i in range(kmeans.n_clusters):
    cluster_data = Video_Games_dataset[Video_Games_dataset['Cluster'] == i]
    print(f"Cluster {i} Summary Statistics:")
    print(cluster_data.describe(), "\n")

# Exploring most common values in each cluster
for feature in ['Rating', 'Genre']:
    print(f"Most common {feature} in each cluster:")
    most_common = Video_Games_dataset.groupby('Cluster')[feature].agg(lambda x:
    print(most_common, "\n")
```



In [ ]:

COMPONENT 3

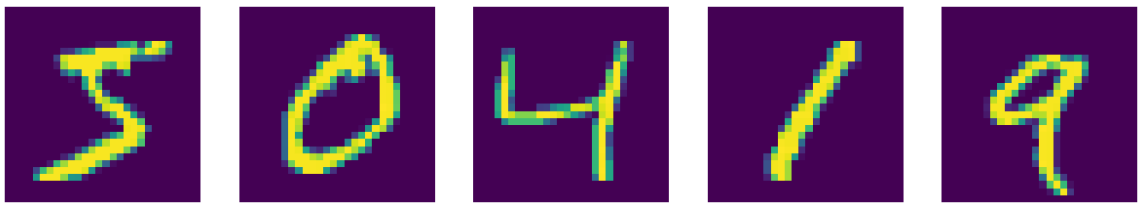
Name: Chibuonu Ezennabike

Student Number: 202315974

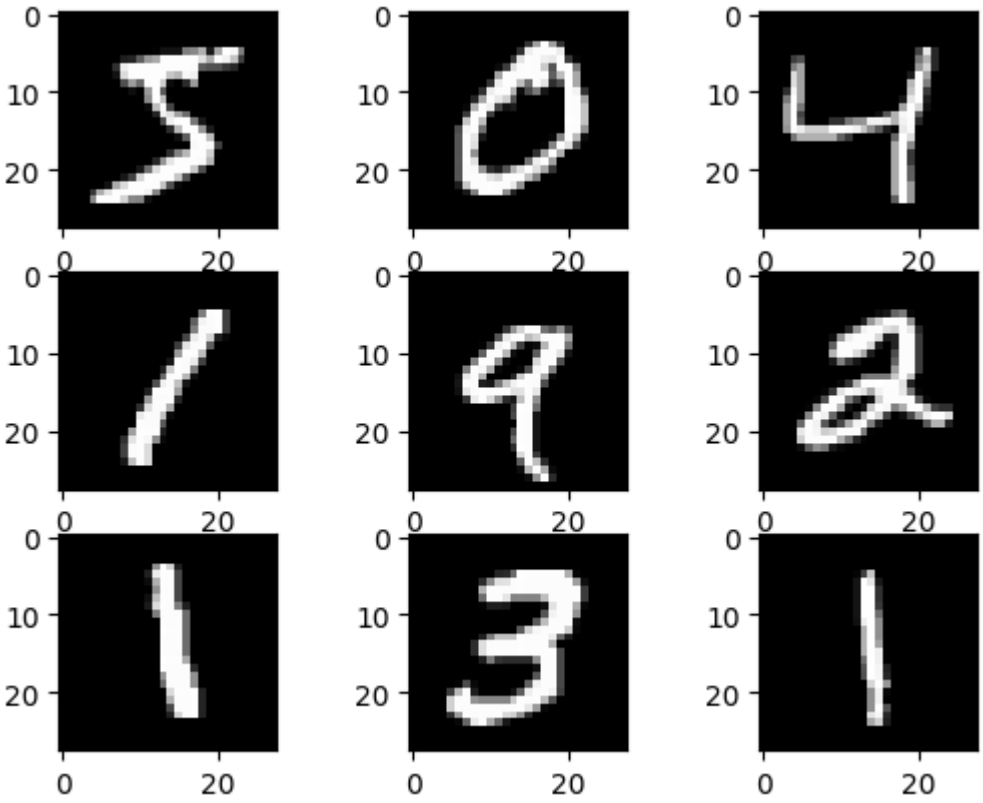
In [1]:
```
# import necessary libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.model_selection import KFold

#import tensorflow as tf
from tensorflow import keras
from tensorflow.keras.datasets import   mnist
from sklearn.metrics import accuracy_score
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dr
from tensorflow.keras.optimizers import SGD, Adam
from tensorflow.keras.regularizers import l2, l1
from tensorflow.keras.utils import to_categorical
from tensorflow.keras.callbacks import EarlyStopping
from tensorflow.keras.callbacks import LearningRateScheduler
from keras.api._v2.keras import callbacks
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix
```

In [2]:
```
# Load the MNIST dataset
(x_train, y_train), (x_test, y_test) = mnist.load_data()
```

In [3]:
```
# Plot first 5 images from dataset

fig, axs = plt.subplots(1, 5, figsize=(15, 3))
for item in range(5):
  axs[item].imshow(x_train[item])
  axs[item].axis('off')
plt.show()
```



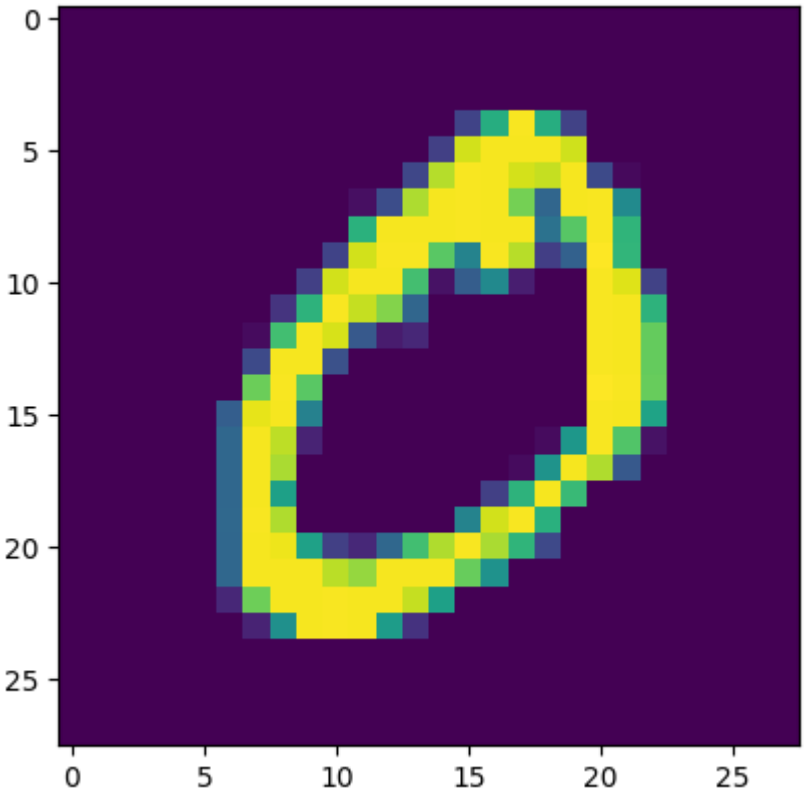
In [5]:
```
for item in range(9):
 # define subplot
 plt.subplot(330 + 1 + item)
 # plot raw pixel data
 plt.imshow(x_train[item], cmap=plt.get_cmap('gray'))
```



In [6]:
```
print(x_train[1].shape)
print(x_train.shape, y_train.shape)

print(x_train.shape , y_test.shape)

plt.imshow(x_train[1])
```

```
(28, 28)
(60000, 28, 28) (60000,)
(60000, 28, 28) (10000,)
```

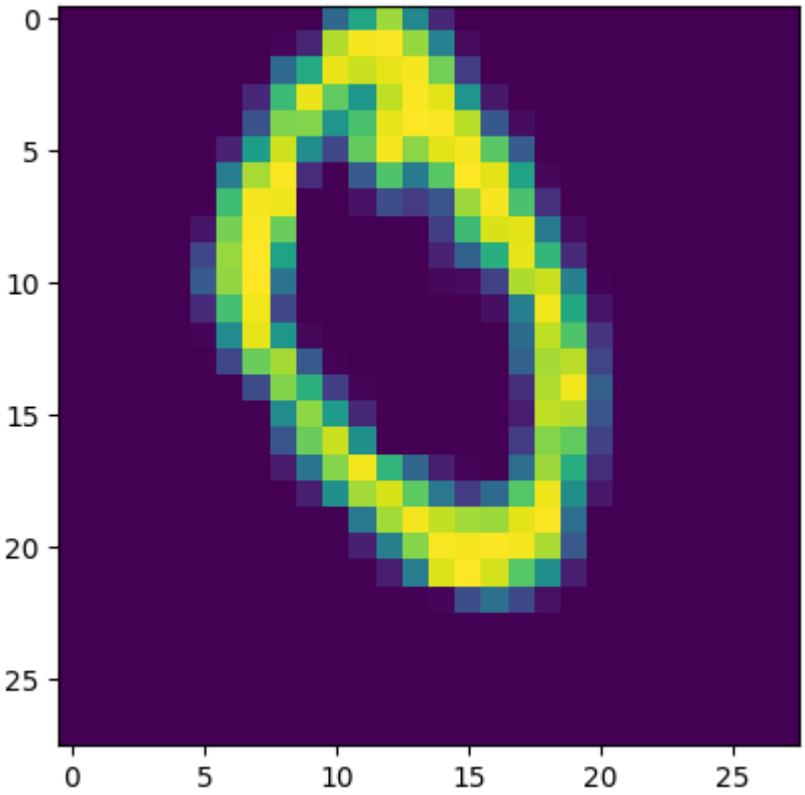
Out[6]: <matplotlib.image.AxesImage at 0x213ccadc310>



In [7]:
```
# Balance pixels values beween 0  and 1
x_train = x_train / 255.0
x_test = x_test / 255.0
```

In [8]:
```
# Data Augmentation
train_datagen = ImageDataGenerator(
  rotation_range=20, # rotate images 10 degrees
  width_shift_range=0.1, # shift images horizontal to 10% of the width
  height_shift_range=0.1, # shift images vertical to 10% of the height
  horizontal_flip=True, # flip images horizontal
  vertical_flip=False, # flip images vertical
  shear_range=0.10, # crops part of the image
  zoom_range=0.10 # #zooms the image by 10%
)



# Fitting changes to training dataset
x_train = x_train.reshape((x_train.shape + (1,)))
x_test = x_test.reshape((x_test.shape + (1,)))


# showing example
plt.imshow(train_datagen.random_transform(x_train[1]))
```

Out[8]:
```
<matplotlib.image.AxesImage at 0x213ccc650d0>
```



Modelling

COMPONENT 3A

In [9]:
```
# Define the model
model = Sequential()

# First Convolutional Layer
model.add(Conv2D(filters=32, kernel_size=(3,3), input_shape=(28,28,1), activ

# Second Convolutional Layer
model.add(Conv2D(filters=64, kernel_size=(3,3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2,2)))

# Third Convolutional Layer
model.add(Conv2D(filters=128, kernel_size=(3,3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2,2)))

# Flattening and Dense Layers
model.add(Flatten())
model.add(Dense(128, activation='relu'))
model.add(Dense(10, activation='softmax'))

# Print the model summary
model.summary()
```

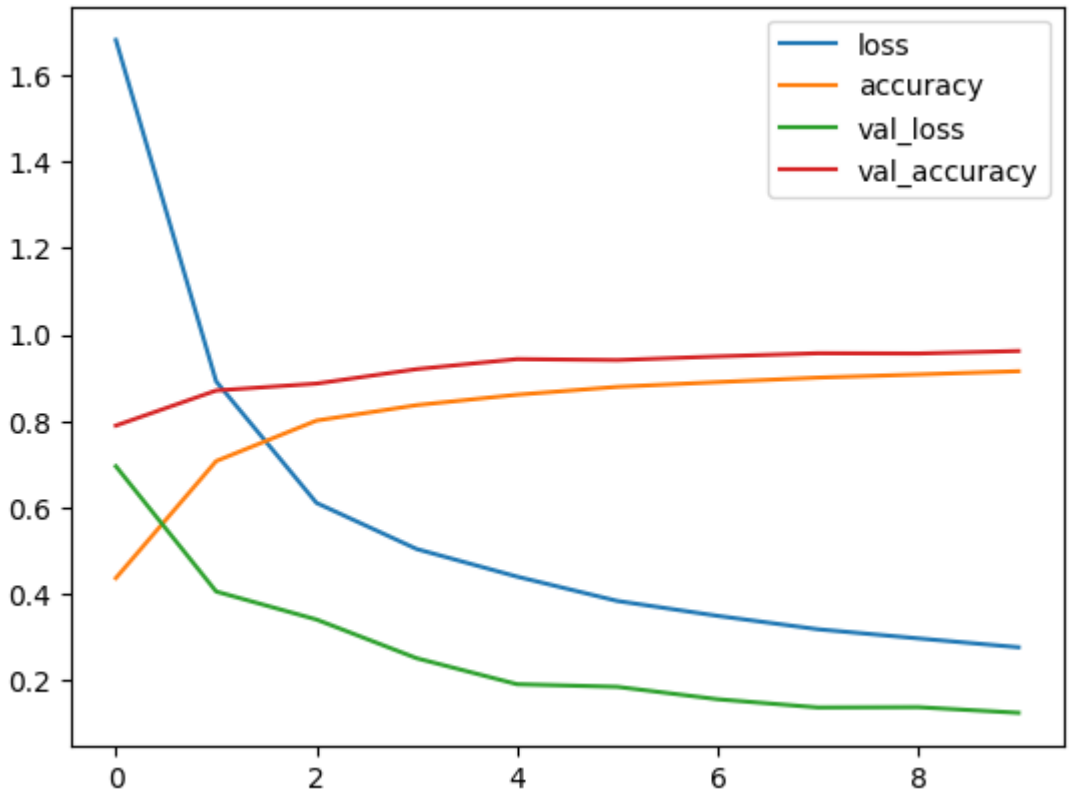
```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 28, 28, 32)        320

 conv2d_1 (Conv2D)           (None, 26, 26, 64)        18496

 max_pooling2d (MaxPooling2D  (None, 13, 13, 64)       0
 )

 conv2d_2 (Conv2D)           (None, 11, 11, 128)       73856

 max_pooling2d_1 (MaxPooling  (None, 5, 5, 128)        0
 2D)

 flatten (Flatten)           (None, 3200)              0

 dense (Dense)               (None, 128)               409728

 dense_1 (Dense)             (None, 10)                1290

=================================================================
Total params: 503,690
Trainable params: 503,690
Non-trainable params: 0
_________________________________________________________________
```

In [10]:
```
# Define the model
model = Sequential()
model.add(Conv2D(32, (3, 3), activation='relu', input_shape=(28, 28, 1))) #
model.add(MaxPooling2D((2, 2)))
model.add(Conv2D(64, (3, 3), activation='relu'))
model.add(MaxPooling2D((2, 2)))
model.add(Flatten())
model.add(Dense(64, activation='relu'))
model.add(Dense(10, activation='softmax'))  # 10 units for 10 classes

# Compile the model
model.compile(optimizer=SGD(learning_rate=0.001, momentum=0.9),
              loss='categorical_crossentropy',
              metrics=['accuracy'])

# Assume train_datagen, x_train, y_train, x_test, y_test are already defined
# Fit the model on the training data
history = model.fit(
    train_datagen.flow(x_train, keras.utils.to_categorical(y_train, num_clas
    epochs=10,
    validation_data=(x_test, keras.utils.to_categorical(y_test, num_classes=
)
```

```
Epoch 1/10
938/938 [==============================] - 29s 30ms/step - loss: 1.6817 -
accuracy: 0.4364 - val_loss: 0.6952 - val_accuracy: 0.7893
Epoch 2/10
938/938 [==============================] - 26s 28ms/step - loss: 0.8914 -
accuracy: 0.7071 - val_loss: 0.4052 - val_accuracy: 0.8705
Epoch 3/10
938/938 [==============================] - 27s 29ms/step - loss: 0.6103 -
accuracy: 0.8009 - val_loss: 0.3404 - val_accuracy: 0.8865
Epoch 4/10
938/938 [==============================] - 26s 28ms/step - loss: 0.5035 -
accuracy: 0.8367 - val_loss: 0.2507 - val_accuracy: 0.9200
Epoch 5/10
938/938 [==============================] - 27s 28ms/step - loss: 0.4397 -
accuracy: 0.8608 - val_loss: 0.1907 - val_accuracy: 0.9432
Epoch 6/10
938/938 [==============================] - 26s 28ms/step - loss: 0.3834 -
accuracy: 0.8790 - val_loss: 0.1846 - val_accuracy: 0.9409
Epoch 7/10
938/938 [==============================] - 26s 27ms/step - loss: 0.3489 -
accuracy: 0.8899 - val_loss: 0.1559 - val_accuracy: 0.9494
Epoch 8/10
938/938 [==============================] - 27s 29ms/step - loss: 0.3178 -
accuracy: 0.9003 - val_loss: 0.1368 - val_accuracy: 0.9567
Epoch 9/10
938/938 [==============================] - 25s 26ms/step - loss: 0.2967 -
accuracy: 0.9077 - val_loss: 0.1375 - val_accuracy: 0.9561
Epoch 10/10
938/938 [==============================] - 27s 28ms/step - loss: 0.2761 -
accuracy: 0.9151 - val_loss: 0.1245 - val_accuracy: 0.9620
```

In [11]:
```
pd.DataFrame(history.history).plot()
```

Out[11]: <Axes: >

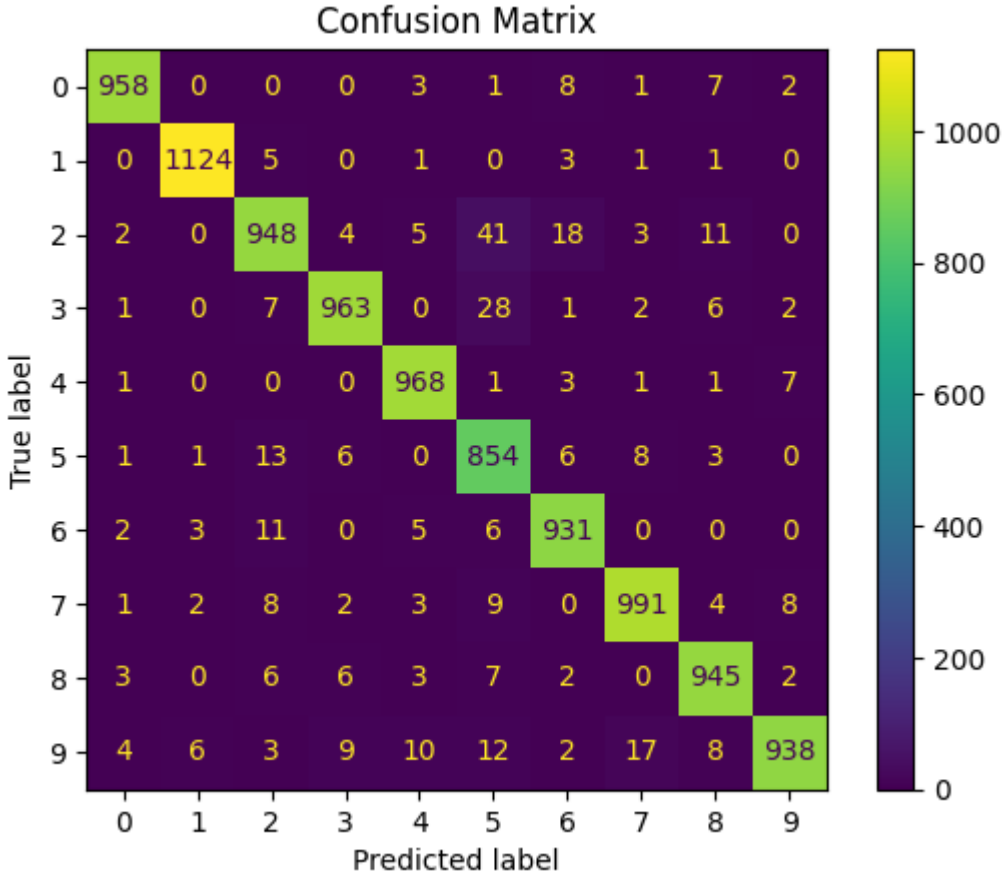


In [12]:
```
# Evaluation model perfomance
y_pred = np.argmax(model.predict(x_test), axis=-1)

print("\n\n" )
# Print report
print(classification_report(y_test, y_pred))

print("\n\n" )
# Print the confusion matrix
from sklearn.metrics import ConfusionMatrixDisplay
plt.figure(dpi=200, figsize=(10,15))
ConfusionMatrixDisplay.from_predictions(y_test,y_pred)
plt.title('Confusion Matrix')
plt.show()

test_loss, test_acc = model.evaluate(x_test, keras.utils.to_categorical(y_te
print("Training accuracy:", history.history['accuracy'][-1])
print("Validation accuracy:", history.history['val_accuracy'][-1])
print("Testing accuracy:", test_acc)
```

```
313/313 [==============================] - 1s 3ms/step
```

	precision	recall	f1-score	support
0	0.98	0.98	0.98	980
1	0.99	0.99	0.99	1135
2	0.95	0.92	0.93	1032
3	0.97	0.95	0.96	1010
4	0.97	0.99	0.98	982
5	0.89	0.96	0.92	892
6	0.96	0.97	0.96	958
7	0.97	0.96	0.97	1028
8	0.96	0.97	0.96	974
9	0.98	0.93	0.95	1009
accuracy			0.96	10000
macro avg	0.96	0.96	0.96	10000
weighted avg	0.96	0.96	0.96	10000

```
<Figure size 2000x3000 with 0 Axes>
```




```
313/313 [==============================] - 1s 3ms/step - loss: 0.1245 - ac
curacy: 0.9620
Training accuracy: 0.9151333570480347
Validation accuracy: 0.9620000123977661
Testing accuracy: 0.9620000123977661
```

Model with regularization

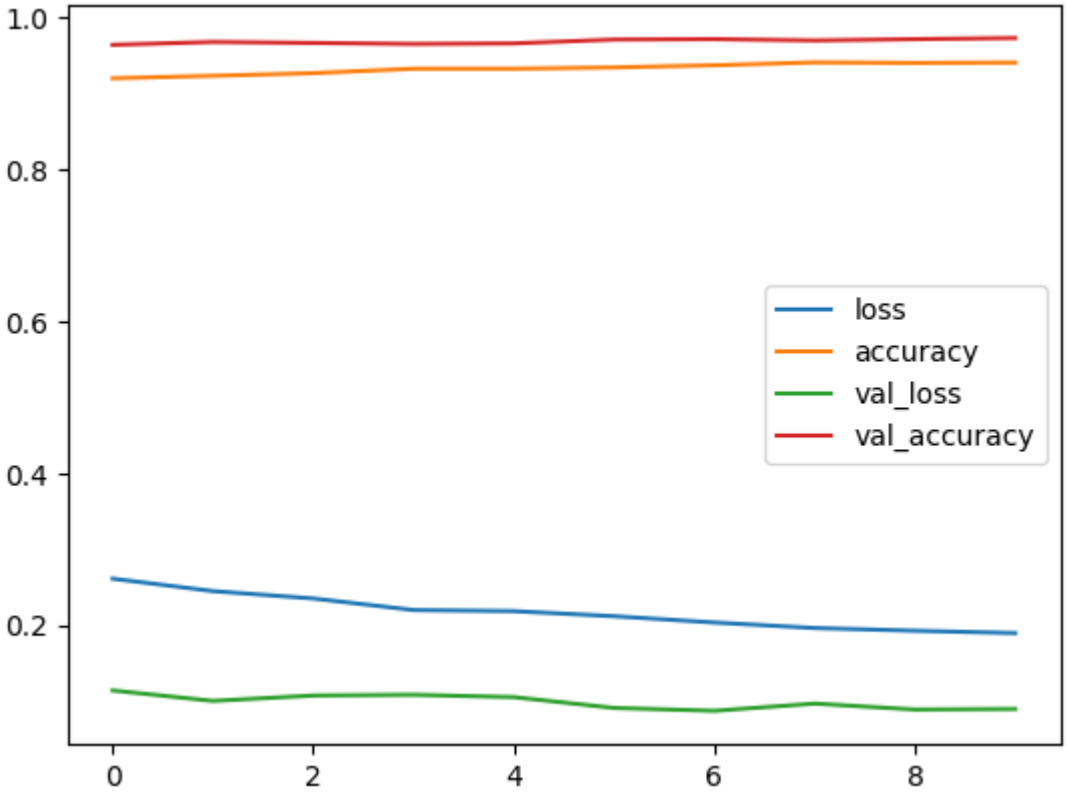
In [13]:
```
modelR = Sequential()
modelR.add(Conv2D(filters=32,kernel_size=(3,3),input_shape=(28,28,1), activa
#model.add(Conv2D(filters=32,kernel_size=(3,3),input_shape=(28,28,1),activat
modelR.add(MaxPooling2D(pool_size=(2,2)))
modelR.add(Conv2D(filters=64,kernel_size=(3,3),activation = 'relu'))
modelR.add(MaxPooling2D(pool_size=(2,2)))
modelR.add(Conv2D(filters=128,kernel_size=(3,3),activation = 'relu'))
modelR.add(MaxPooling2D(pool_size=(2,2)))
modelR.add(BatchNormalization())
modelR.add(Flatten())
modelR.add(Dense(100,activation = 'relu'))
modelR.add(Dropout(0.5))
modelR.add(Dense(10,activation = 'softmax'))


# printing the model summary
modelR.summary()
```

```
Model: "sequential_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_5 (Conv2D)           (None, 28, 28, 32)        320

 max_pooling2d_4 (MaxPooling  (None, 14, 14, 32)       0
 2D)

 conv2d_6 (Conv2D)           (None, 12, 12, 64)        18496

 max_pooling2d_5 (MaxPooling  (None, 6, 6, 64)         0
 2D)

 conv2d_7 (Conv2D)           (None, 4, 4, 128)         73856

 max_pooling2d_6 (MaxPooling  (None, 2, 2, 128)        0
 2D)

 batch_normalization (BatchN  (None, 2, 2, 128)        512
 ormalization)

 flatten_2 (Flatten)         (None, 512)               0

 dense_4 (Dense)             (None, 100)               51300

 dropout (Dropout)           (None, 100)               0

 dense_5 (Dense)             (None, 10)                1010

=================================================================
Total params: 145,494
Trainable params: 145,238
Non-trainable params: 256
_________________________________________________________________
```

In [14]:
```
modelR.compile(optimizer=SGD(learning_rate=0.001, momentum=0.9), loss='categ

history1=model.fit(train_datagen.flow(x_train, keras.utils.to_categorical(y_

pd.DataFrame(history1.history).plot()
```

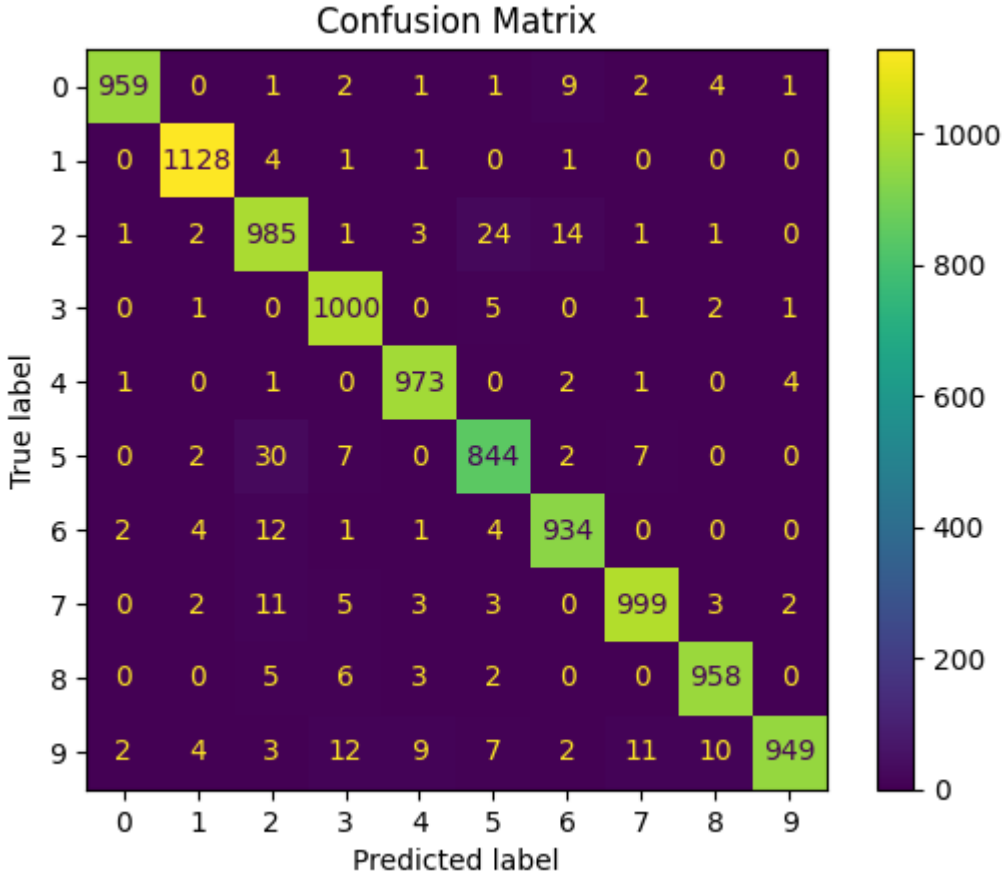
```
Epoch 1/10
938/938 [==============================] - 28s 30ms/step - loss: 0.2608 -
accuracy: 0.9199 - val_loss: 0.1136 - val_accuracy: 0.9639
Epoch 2/10
938/938 [==============================] - 30s 32ms/step - loss: 0.2444 -
accuracy: 0.9231 - val_loss: 0.0998 - val_accuracy: 0.9677
Epoch 3/10
938/938 [==============================] - 27s 29ms/step - loss: 0.2347 -
accuracy: 0.9266 - val_loss: 0.1069 - val_accuracy: 0.9664
Epoch 4/10
938/938 [==============================] - 25s 27ms/step - loss: 0.2195 -
accuracy: 0.9324 - val_loss: 0.1078 - val_accuracy: 0.9651
Epoch 5/10
938/938 [==============================] - 26s 28ms/step - loss: 0.2180 -
accuracy: 0.9324 - val_loss: 0.1048 - val_accuracy: 0.9658
Epoch 6/10
938/938 [==============================] - 29s 31ms/step - loss: 0.2113 -
accuracy: 0.9341 - val_loss: 0.0905 - val_accuracy: 0.9709
Epoch 7/10
938/938 [==============================] - 32s 34ms/step - loss: 0.2031 -
accuracy: 0.9370 - val_loss: 0.0867 - val_accuracy: 0.9714
Epoch 8/10
938/938 [==============================] - 36s 38ms/step - loss: 0.1958 -
accuracy: 0.9408 - val_loss: 0.0963 - val_accuracy: 0.9697
Epoch 9/10
938/938 [==============================] - 35s 37ms/step - loss: 0.1922 -
accuracy: 0.9401 - val_loss: 0.0884 - val_accuracy: 0.9714
Epoch 10/10
938/938 [==============================] - 32s 34ms/step - loss: 0.1891 -
accuracy: 0.9406 - val_loss: 0.0891 - val_accuracy: 0.9729
```

Out[14]:
```
<Axes: >
```


1.0
0.8
0.6
0.4
0.2
0
2
4
6
8
loss
accuracy
val_loss
val_accuracy

In [15]:
```
y_pred = np.argmax(model.predict(x_test), axis=-1)

print("\n\n" )
# Print report
print(classification_report(y_test, y_pred))

print("\n\n" )
# Print confusion matrix
from sklearn.metrics import ConfusionMatrixDisplay
plt.figure(dpi=200, figsize=(10,15))
ConfusionMatrixDisplay.from_predictions(y_test,y_pred)
plt.title('Confusion Matrix')
plt.show()

print("\n\n" )
test_loss, test_acc = model.evaluate(x_test, keras.utils.to_categorical(y_te
print("Training accuracy:", history1.history['accuracy'][-1])
print("Validation accuracy:", history1.history['val_accuracy'][-1])
print("Testing accuracy:", test_acc)
```

```
313/313 [==============================] - 1s 3ms/step


              precision    recall  f1-score   support

           0       0.99      0.98      0.99       980
           1       0.99      0.99      0.99      1135
           2       0.94      0.95      0.95      1032
           3       0.97      0.99      0.98      1010
           4       0.98      0.99      0.98       982
           5       0.95      0.95      0.95       892
           6       0.97      0.97      0.97       958
           7       0.98      0.97      0.97      1028
           8       0.98      0.98      0.98       974
           9       0.99      0.94      0.97      1009

    accuracy                           0.97     10000
   macro avg       0.97      0.97      0.97     10000
weighted avg       0.97      0.97      0.97     10000



<Figure size 2000x3000 with 0 Axes>
```




```
313/313 [==============================] - 1s 3ms/step - loss: 0.0891 - ac
curacy: 0.9729
Training accuracy: 0.9405666589736938
Validation accuracy: 0.9728999733924866
Testing accuracy: 0.9728999733924866
```

L1 Regularization Model

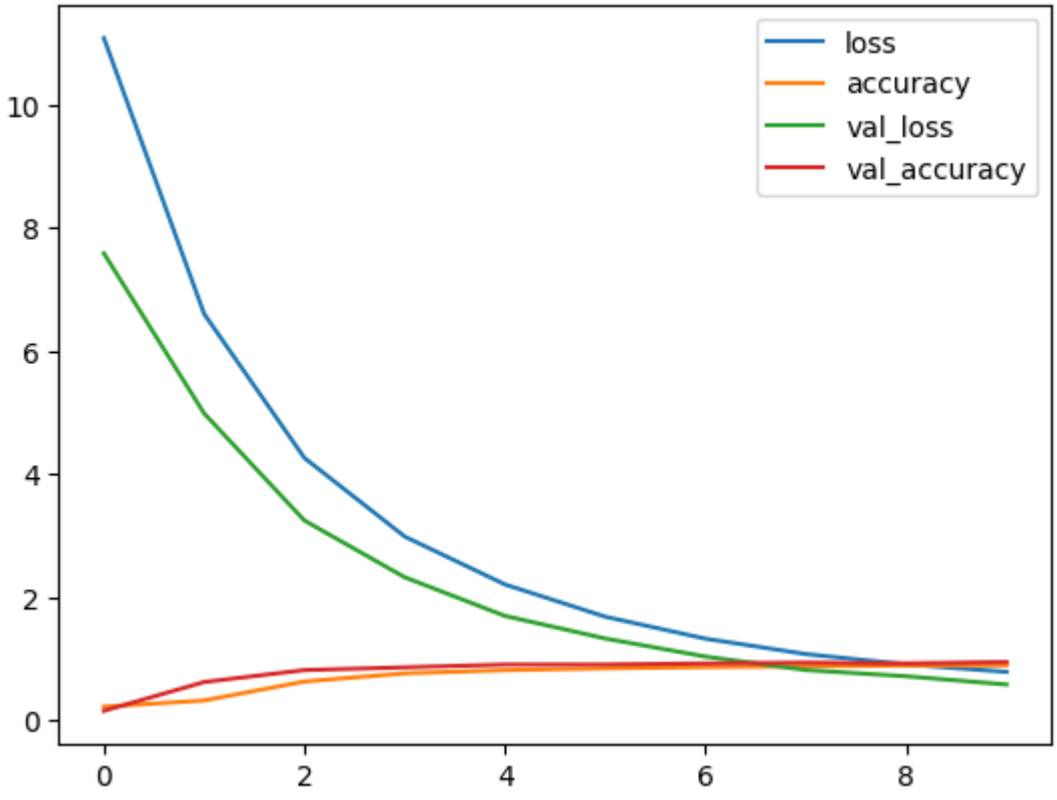
In [18]:
```
# cnn convolutional blocks
modelL1 = Sequential()
modelL1.add(Conv2D(filters=32,kernel_size=(3,3),input_shape=(28,28,1), activ
#model.add(Conv2D(filters=32,kernel_size=(3,3),input_shape=(28,28,1),activat
modelL1.add(MaxPooling2D(pool_size=(2,2)))

modelL1.add(Conv2D(filters=64,kernel_size=(3,3),activation = 'relu', kernel_
modelL1.add(MaxPooling2D(pool_size=(2,2)))

modelL1.add(Conv2D(filters=128,kernel_size=(3,3),activation = 'relu', kernel
modelL1.add(MaxPooling2D(pool_size=(2,2)))

modelL1.add(Flatten())
modelL1.add(Dense(1024,activation = 'relu', kernel_regularizer=l2(0.01)))

modelL1.add(Dense(10,activation = 'softmax'))


# print the model summary
modelL1.summary()
```

```
Model: "sequential_4"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_11 (Conv2D)          (None, 28, 28, 32)        320

 max_pooling2d_10 (MaxPoolin  (None, 14, 14, 32)       0
 g2D)

 conv2d_12 (Conv2D)          (None, 12, 12, 64)        18496

 max_pooling2d_11 (MaxPoolin  (None, 6, 6, 64)         0
 g2D)

 conv2d_13 (Conv2D)          (None, 4, 4, 128)         73856

 max_pooling2d_12 (MaxPoolin  (None, 2, 2, 128)        0
 g2D)

 flatten_4 (Flatten)         (None, 512)               0

 dense_8 (Dense)             (None, 1024)              525312

 dense_9 (Dense)             (None, 10)                10250

=================================================================
Total params: 628,234
Trainable params: 628,234
Non-trainable params: 0
_________________________________________________________________
```

In [19]:
```
# model with l1 regularization
modelL1.compile(optimizer=SGD(learning_rate=0.001, momentum=0.9), loss='cate

# train 10 epoch with batch size 64.
history1=modelL1.fit(train_datagen.flow(x_train, keras.utils.to_categorical(


pd.DataFrame(history1.history).plot()
```

```
Epoch 1/10
938/938 [==============================] - 47s 49ms/step - loss: 11.0814 -
accuracy: 0.2241 - val_loss: 7.5849 - val_accuracy: 0.1566
Epoch 2/10
938/938 [==============================] - 44s 47ms/step - loss: 6.6011 -
accuracy: 0.3244 - val_loss: 4.9853 - val_accuracy: 0.6245
Epoch 3/10
938/938 [==============================] - 45s 47ms/step - loss: 4.2620 -
accuracy: 0.6328 - val_loss: 3.2461 - val_accuracy: 0.8177
Epoch 4/10
938/938 [==============================] - 45s 48ms/step - loss: 2.9859 -
accuracy: 0.7653 - val_loss: 2.3251 - val_accuracy: 0.8630
Epoch 5/10
938/938 [==============================] - 43s 46ms/step - loss: 2.2080 -
accuracy: 0.8171 - val_loss: 1.6968 - val_accuracy: 0.9073
Epoch 6/10
938/938 [==============================] - 45s 48ms/step - loss: 1.6841 -
accuracy: 0.8462 - val_loss: 1.3304 - val_accuracy: 0.9078
Epoch 7/10
938/938 [==============================] - 44s 47ms/step - loss: 1.3274 -
accuracy: 0.8668 - val_loss: 1.0368 - val_accuracy: 0.9259
Epoch 8/10
938/938 [==============================] - 47s 50ms/step - loss: 1.0781 -
accuracy: 0.8813 - val_loss: 0.8189 - val_accuracy: 0.9390
Epoch 9/10
938/938 [==============================] - 46s 49ms/step - loss: 0.9097 -
accuracy: 0.8914 - val_loss: 0.7178 - val_accuracy: 0.9265
Epoch 10/10
938/938 [==============================] - 47s 50ms/step - loss: 0.7890 -
accuracy: 0.8962 - val_loss: 0.5839 - val_accuracy: 0.9485
```

Out[19]: <Axes: >

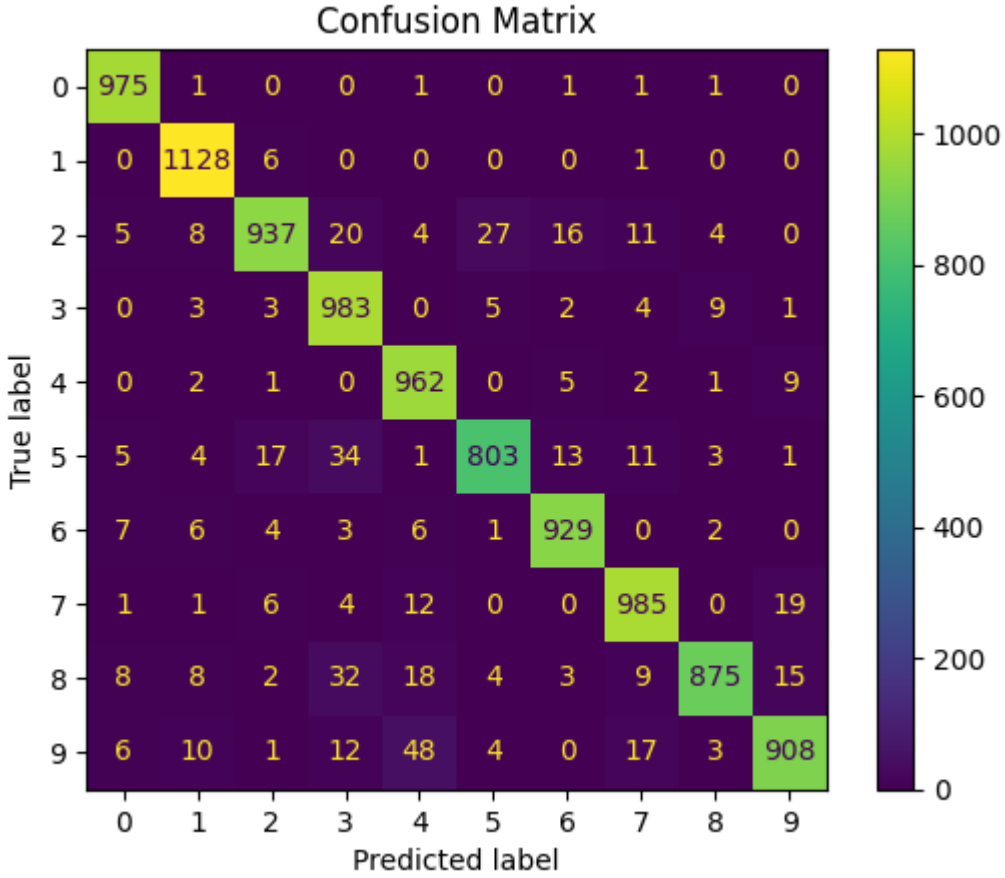

loss
accuracy
val_loss
val_accuracy
10
8
6
4
2
0
0
2
4
6
8

In [20]:
```
# model evaluation
y_pred = np.argmax(modelL1.predict(x_test), axis=-1)

print("\n\n" )
# Print the classification report
print(classification_report(y_test, y_pred))

print("\n\n" )
# Print the confusion matrix
from sklearn.metrics import ConfusionMatrixDisplay
plt.figure(dpi=200, figsize=(10,15))
ConfusionMatrixDisplay.from_predictions(y_test,y_pred)
plt.title('Confusion Matrix')
plt.show()

print("\n\n" )
test_loss, test_acc = modelL1.evaluate(x_test, keras.utils.to_categorical(y_
print("Training accuracy:", history1.history['accuracy'][-1])
print("Validation accuracy:", history1.history['val_accuracy'][-1])
print("Testing accuracy:", test_acc)
```

```
313/313 [==============================] - 2s 6ms/step
```

```
              precision    recall  f1-score   support

           0       0.97      0.99      0.98       980
           1       0.96      0.99      0.98      1135
           2       0.96      0.91      0.93      1032
           3       0.90      0.97      0.94      1010
           4       0.91      0.98      0.95       982
           5       0.95      0.90      0.93       892
           6       0.96      0.97      0.96       958
           7       0.95      0.96      0.95      1028
           8       0.97      0.90      0.93       974
           9       0.95      0.90      0.93      1009

    accuracy                           0.95     10000
   macro avg       0.95      0.95      0.95     10000
weighted avg       0.95      0.95      0.95     10000
```

```
<Figure size 2000x3000 with 0 Axes>
```




```
313/313 [==============================] - 2s 6ms/step - loss: 0.5839 - ac
curacy: 0.9485
Training accuracy: 0.8961833119392395
Validation accuracy: 0.9484999775886536
Testing accuracy: 0.9484999775886536
```

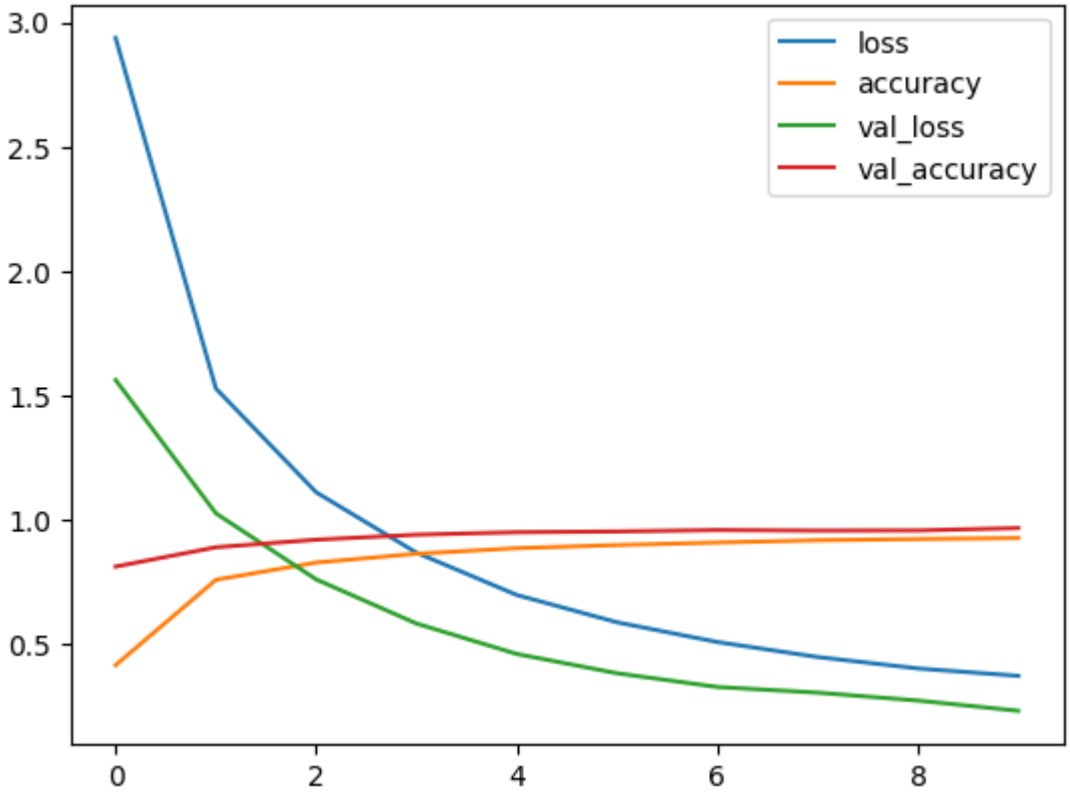
L2 Regularization Model

In [21]:
```
# CNN blocks
modelL2 = Sequential()
modelL2.add(Conv2D(filters=32,kernel_size=(3,3),input_shape=(28,28,1), activ
#model.add(Conv2D(filters=32,kernel_size=(3,3),input_shape=(28,28,1),activat
modelL2.add(MaxPooling2D(pool_size=(2,2)))

modelL2.add(Conv2D(filters=64,kernel_size=(3,3),activation = 'relu', kernel_
modelL2.add(MaxPooling2D(pool_size=(2,2)))

modelL2.add(Conv2D(filters=128,kernel_size=(3,3),activation = 'relu', kernel
modelL2.add(MaxPooling2D(pool_size=(2,2)))

modelL2.add(Flatten())
modelL2.add(Dense(128,activation = 'relu'))

modelL2.add(Dense(10,activation = 'softmax'))
```

In [22]:
```
modelL2.compile(optimizer=SGD(learning_rate=0.001, momentum=0.9), loss='cate

# 10 epochs and 64 batch size
history2=modelL2.fit(train_datagen.flow(x_train, keras.utils.to_categorical(


pd.DataFrame(history2.history).plot()
```

```
Epoch 1/10
938/938 [==============================] - 41s 43ms/step - loss: 2.9366 -
accuracy: 0.4182 - val_loss: 1.5634 - val_accuracy: 0.8140
Epoch 2/10
938/938 [==============================] - 39s 41ms/step - loss: 1.5289 -
accuracy: 0.7610 - val_loss: 1.0277 - val_accuracy: 0.8916
Epoch 3/10
938/938 [==============================] - 41s 43ms/step - loss: 1.1122 -
accuracy: 0.8302 - val_loss: 0.7625 - val_accuracy: 0.9220
Epoch 4/10
938/938 [==============================] - 42s 44ms/step - loss: 0.8697 -
accuracy: 0.8654 - val_loss: 0.5851 - val_accuracy: 0.9420
Epoch 5/10
938/938 [==============================] - 40s 43ms/step - loss: 0.6999 -
accuracy: 0.8879 - val_loss: 0.4631 - val_accuracy: 0.9520
Epoch 6/10
938/938 [==============================] - 40s 42ms/step - loss: 0.5901 -
accuracy: 0.9004 - val_loss: 0.3855 - val_accuracy: 0.9551
Epoch 7/10
938/938 [==============================] - 40s 42ms/step - loss: 0.5109 -
accuracy: 0.9105 - val_loss: 0.3305 - val_accuracy: 0.9608
Epoch 8/10
938/938 [==============================] - 42s 45ms/step - loss: 0.4507 -
accuracy: 0.9198 - val_loss: 0.3077 - val_accuracy: 0.9585
Epoch 9/10
938/938 [==============================] - 39s 42ms/step - loss: 0.4049 -
accuracy: 0.9244 - val_loss: 0.2760 - val_accuracy: 0.9592
Epoch 10/10
938/938 [==============================] - 39s 41ms/step - loss: 0.3751 -
accuracy: 0.9287 - val_loss: 0.2348 - val_accuracy: 0.9687
```

Out[22]:
```
<Axes: >
```

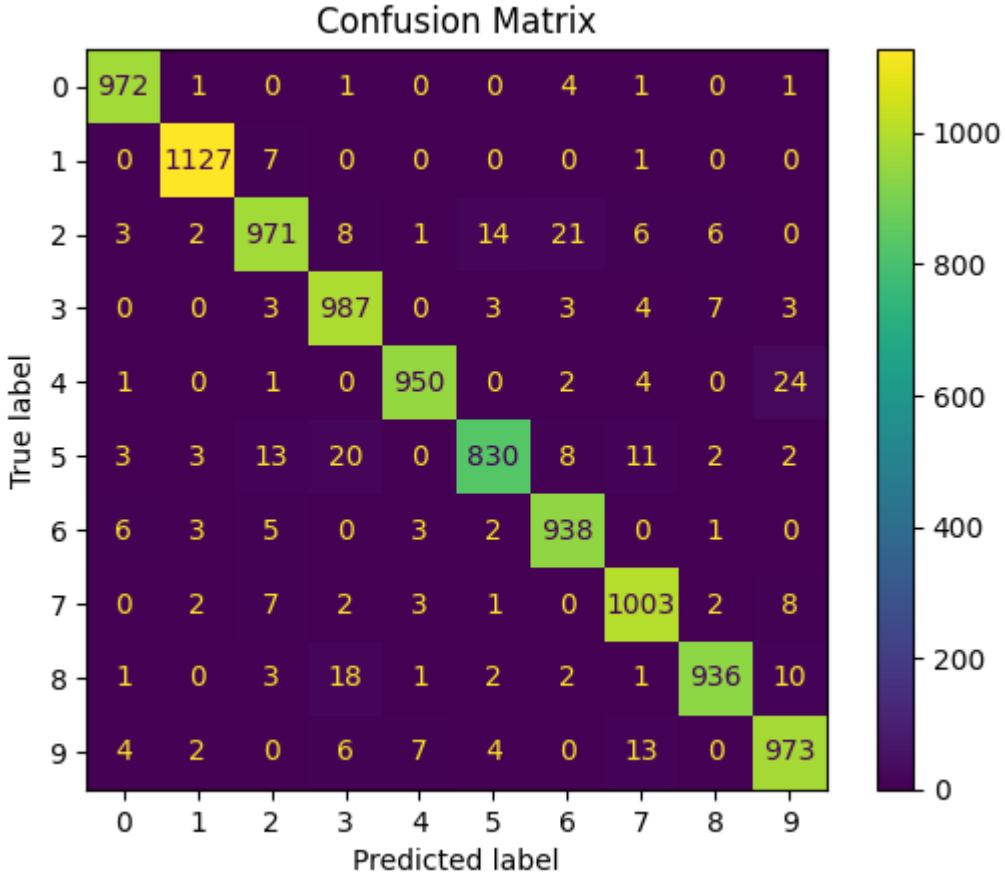

loss
accuracy
val_loss
val_accuracy
3.0
2.5
2.0
1.5
1.0
0.5
0
2
4
6
8

In [24]:
```
# model evalution
y_pred = np.argmax(modelL2.predict(x_test), axis=-1)

print("\n\n" )
# Print the classification report
print(classification_report(y_test, y_pred))

print("\n\n" )
# Print the confusion matrix
from sklearn.metrics import ConfusionMatrixDisplay
plt.figure(dpi=200, figsize=(10,15))
ConfusionMatrixDisplay.from_predictions(y_test,y_pred)
plt.title('Confusion Matrix')
plt.show()


print("\n\n" )
test_loss, test_acc = modelL2.evaluate(x_test, keras.utils.to_categorical(y_
print("Training accuracy:", history2.history['accuracy'][-1])
print("Validation accuracy:", history2.history['val_accuracy'][-1])
print("Testing accuracy:", test_acc)
```

```
313/313 [==============================] - 1s 4ms/step
```

```
              precision    recall  f1-score   support

           0       0.98      0.99      0.99       980
           1       0.99      0.99      0.99      1135
           2       0.96      0.94      0.95      1032
           3       0.95      0.98      0.96      1010
           4       0.98      0.97      0.98       982
           5       0.97      0.93      0.95       892
           6       0.96      0.98      0.97       958
           7       0.96      0.98      0.97      1028
           8       0.98      0.96      0.97       974
           9       0.95      0.96      0.96      1009

    accuracy                           0.97     10000
   macro avg       0.97      0.97      0.97     10000
weighted avg       0.97      0.97      0.97     10000
```

```
<Figure size 2000x3000 with 0 Axes>
```




```
313/313 [==============================] - 2s 5ms/step - loss: 0.2348 - ac
curacy: 0.9687
Training accuracy: 0.9286500215530396
Validation accuracy: 0.9686999917030334
Testing accuracy: 0.9686999917030334
```

Trying early stop regularization model

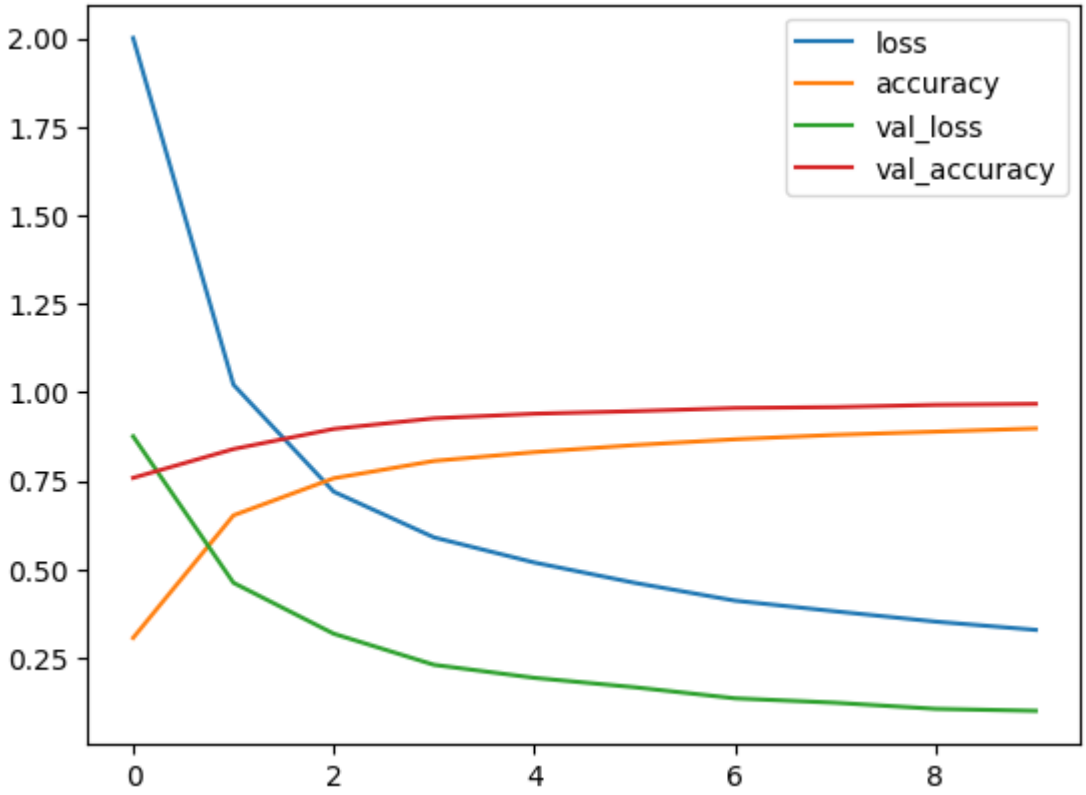
In [25]:
```
# CNN convolutional blocks
modelES = Sequential()
modelES.add(Conv2D(filters=32,kernel_size=(3,3),input_shape=(28,28,1), activ
#model.add(Conv2D(filters=32,kernel_size=(3,3),input_shape=(28,28,1),activat
modelES.add(MaxPooling2D(pool_size=(2,2)))

modelES.add(Conv2D(filters=64,kernel_size=(3,3),activation = 'relu'))
modelES.add(MaxPooling2D(pool_size=(2,2)))

modelES.add(Conv2D(filters=128,kernel_size=(3,3),activation = 'relu'))
modelES.add(MaxPooling2D(pool_size=(2,2)))

modelES.add(Flatten())
modelES.add(Dense(128,activation = 'relu'))
modelES.add(Dropout(0.5))
modelES.add(Dense(10,activation = 'softmax'))

# print the model summary
modelES.summary()
```

```
Model: "sequential_6"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_17 (Conv2D)          (None, 28, 28, 32)        320

 max_pooling2d_16 (MaxPoolin  (None, 14, 14, 32)       0
 g2D)

 conv2d_18 (Conv2D)          (None, 12, 12, 64)        18496

 max_pooling2d_17 (MaxPoolin  (None, 6, 6, 64)         0
 g2D)

 conv2d_19 (Conv2D)          (None, 4, 4, 128)         73856

 max_pooling2d_18 (MaxPoolin  (None, 2, 2, 128)        0
 g2D)

 flatten_6 (Flatten)         (None, 512)               0

 dense_12 (Dense)            (None, 128)               65664

 dropout_1 (Dropout)         (None, 128)               0

 dense_13 (Dense)            (None, 10)                1290

=================================================================
Total params: 159,626
Trainable params: 159,626
Non-trainable params: 0
_________________________________________________________________
```

In [26]:
```
modelES.compile(optimizer=SGD(learning_rate=0.001, momentum=0.9), loss='cate

history3=modelES.fit(train_datagen.flow(x_train, keras.utils.to_categorical(


pd.DataFrame(history3.history).plot()
```

```
Epoch 1/10
938/938 [==============================] - 38s 40ms/step - loss: 2.0003 -
accuracy: 0.3067 - val_loss: 0.8757 - val_accuracy: 0.7585
Epoch 2/10
938/938 [==============================] - 39s 42ms/step - loss: 1.0215 -
accuracy: 0.6525 - val_loss: 0.4622 - val_accuracy: 0.8398
Epoch 3/10
938/938 [==============================] - 38s 40ms/step - loss: 0.7194 -
accuracy: 0.7573 - val_loss: 0.3184 - val_accuracy: 0.8966
Epoch 4/10
938/938 [==============================] - 36s 39ms/step - loss: 0.5903 -
accuracy: 0.8062 - val_loss: 0.2306 - val_accuracy: 0.9266
Epoch 5/10
938/938 [==============================] - 38s 41ms/step - loss: 0.5194 -
accuracy: 0.8311 - val_loss: 0.1938 - val_accuracy: 0.9395
Epoch 6/10
938/938 [==============================] - 39s 42ms/step - loss: 0.4621 -
accuracy: 0.8510 - val_loss: 0.1673 - val_accuracy: 0.9468
Epoch 7/10
938/938 [==============================] - 45s 48ms/step - loss: 0.4118 -
accuracy: 0.8673 - val_loss: 0.1360 - val_accuracy: 0.9556
Epoch 8/10
938/938 [==============================] - 49s 53ms/step - loss: 0.3816 -
accuracy: 0.8799 - val_loss: 0.1236 - val_accuracy: 0.9582
Epoch 9/10
938/938 [==============================] - 43s 46ms/step - loss: 0.3525 -
accuracy: 0.8888 - val_loss: 0.1063 - val_accuracy: 0.9646
Epoch 10/10
938/938 [==============================] - 37s 39ms/step - loss: 0.3292 -
accuracy: 0.8979 - val_loss: 0.1007 - val_accuracy: 0.9673
```

Out[26]: <Axes: >



In [27]:
```
print("\n\n" )
test_loss, test_acc = modelES.evaluate(x_test, keras.utils.to_categorical(y_
print("Training accuracy:", history3.history['accuracy'][-1])
print("Validation accuracy:", history3.history['val_accuracy'][-1])
print("Testing accuracy:", test_acc)
```

```
313/313 [==============================] - 2s 6ms/step - loss: 0.1007 - ac
curacy: 0.9673
Training accuracy: 0.8978999853134155
Validation accuracy: 0.9672999978065491
Testing accuracy: 0.9672999978065491
```

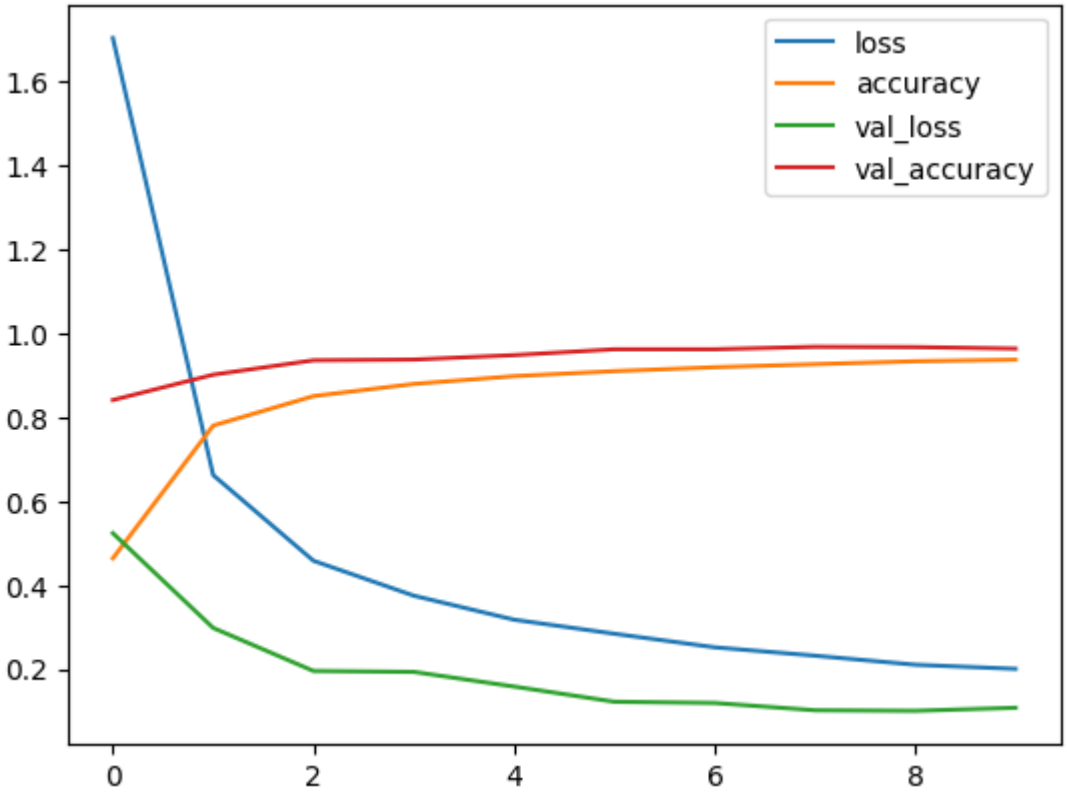
Batch Normalization Model

In [28]:
```
# Model with Batch Normalization
modelBN = Sequential()
modelBN.add(Conv2D(filters=32,kernel_size=(3,3),input_shape=(28,28,1), activ
modelBN.add(MaxPooling2D(pool_size=(2,2)))
model.add(BatchNormalization())
modelBN.add(Conv2D(filters=64,kernel_size=(3,3),activation = 'relu'))
modelBN.add(MaxPooling2D(pool_size=(2,2)))
model.add(BatchNormalization())
modelBN.add(Conv2D(filters=128,kernel_size=(3,3),activation = 'relu'))
modelBN.add(MaxPooling2D(pool_size=(2,2)))
model.add(BatchNormalization())

modelBN.add(Flatten())
modelBN.add(Dense(128,activation = 'relu'))
model.add(BatchNormalization())
modelBN.add(Dense(10,activation = 'softmax'))
```

In [29]:
```
# modelBN
modelBN.compile(optimizer=SGD(learning_rate=0.001, momentum=0.9), loss='cate

# 10 epochs and 64 batch size
historyBN=modelBN.fit(train_datagen.flow(x_train, keras.utils.to_categorical


pd.DataFrame(historyBN.history).plot()
```

```
Epoch 1/10
938/938 [==============================] - 40s 42ms/step - loss: 1.7038 -
accuracy: 0.4641 - val_loss: 0.5237 - val_accuracy: 0.8411
Epoch 2/10
938/938 [==============================] - 41s 44ms/step - loss: 0.6622 -
accuracy: 0.7802 - val_loss: 0.2981 - val_accuracy: 0.9015
Epoch 3/10
938/938 [==============================] - 39s 41ms/step - loss: 0.4581 -
accuracy: 0.8505 - val_loss: 0.1955 - val_accuracy: 0.9358
Epoch 4/10
938/938 [==============================] - 37s 40ms/step - loss: 0.3749 -
accuracy: 0.8795 - val_loss: 0.1933 - val_accuracy: 0.9375
Epoch 5/10
938/938 [==============================] - 39s 42ms/step - loss: 0.3177 -
accuracy: 0.8981 - val_loss: 0.1584 - val_accuracy: 0.9481
Epoch 6/10
938/938 [==============================] - 39s 41ms/step - loss: 0.2842 -
accuracy: 0.9100 - val_loss: 0.1221 - val_accuracy: 0.9621
Epoch 7/10
938/938 [==============================] - 42s 45ms/step - loss: 0.2519 -
accuracy: 0.9193 - val_loss: 0.1192 - val_accuracy: 0.9621
Epoch 8/10
938/938 [==============================] - 40s 43ms/step - loss: 0.2319 -
accuracy: 0.9266 - val_loss: 0.1019 - val_accuracy: 0.9680
Epoch 9/10
938/938 [==============================] - 40s 43ms/step - loss: 0.2103 -
accuracy: 0.9335 - val_loss: 0.1005 - val_accuracy: 0.9673
Epoch 10/10
938/938 [==============================] - 40s 43ms/step - loss: 0.2005 -
accuracy: 0.9373 - val_loss: 0.1078 - val_accuracy: 0.9637
```

Out[29]:
```
<Axes: >
```

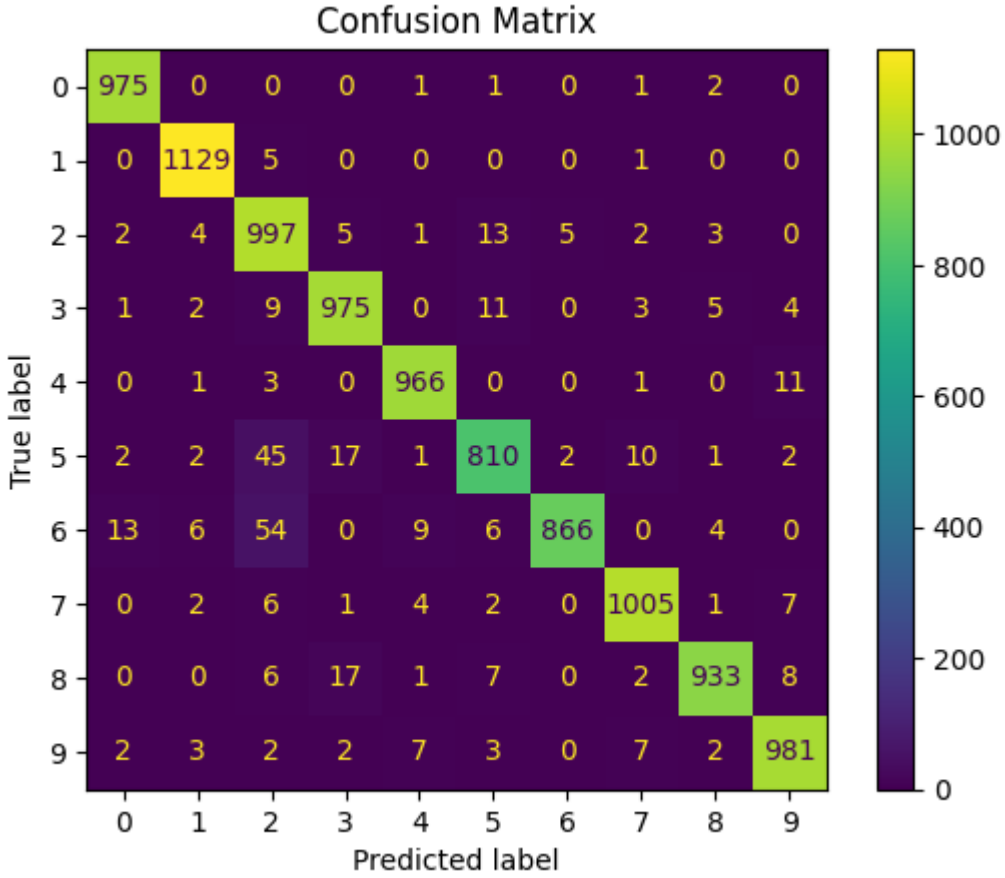

loss
accuracy
val_loss
val_accuracy
1.6
1.4
1.2
1.0
0.8
0.6
0.4
0.2
0
2
4
6
8

In [30]:
```
# evaluate modelBN
y_pred = np.argmax(modelBN.predict(x_test), axis=-1)

print("\n\n" )
# Print the classification report
print(classification_report(y_test, y_pred))

print("\n\n" )
# Print the confusion matrix
from sklearn.metrics import ConfusionMatrixDisplay
plt.figure(dpi=200, figsize=(10,15))
ConfusionMatrixDisplay.from_predictions(y_test,y_pred)
plt.title('Confusion Matrix')
plt.show()


print("\n\n" )
test_loss, test_acc = modelBN.evaluate(x_test, keras.utils.to_categorical(y_
print("Training accuracy:", historyBN.history['accuracy'][-1])
print("Validation accuracy:", historyBN.history['val_accuracy'][-1])
print("Testing accuracy:", test_acc)
```

```
313/313 [==============================] - 1s 4ms/step
```

	precision	recall	f1-score	support
0	0.98	0.99	0.99	980
1	0.98	0.99	0.99	1135
2	0.88	0.97	0.92	1032
3	0.96	0.97	0.96	1010
4	0.98	0.98	0.98	982
5	0.95	0.91	0.93	892
6	0.99	0.90	0.95	958
7	0.97	0.98	0.98	1028
8	0.98	0.96	0.97	974
9	0.97	0.97	0.97	1009
accuracy			0.96	10000
macro avg	0.96	0.96	0.96	10000
weighted avg	0.96	0.96	0.96	10000

```
<Figure size 2000x3000 with 0 Axes>
```




```
313/313 [==============================] - 1s 5ms/step - loss: 0.1078 - ac
curacy: 0.9637
Training accuracy: 0.9372833371162415
Validation accuracy: 0.963699996471405
Testing accuracy: 0.963699996471405
```

COMPONENT 3B

In [31]:
```
model3B = Sequential()
model3B.add(Conv2D(filters=32,kernel_size=(3,3),input_shape=(28,28,1), activ
#model.add(Conv2D(filters=32,kernel_size=(3,3),input_shape=(28,28,1),activat
model3B.add(MaxPooling2D(pool_size=(2,2)))

model3B.add(Conv2D(filters=64,kernel_size=(3,3),activation = 'relu'))
model3B.add(MaxPooling2D(pool_size=(2,2)))

# model.add(Conv2D(filters=128,kernel_size=(3,3),activation = 'relu', paddir
# model.add(MaxPooling2D(pool_size=(2,2)))

# model.add(Conv2D(filters=256,kernel_size=(3,3),activation = 'relu', paddir
# model.add(MaxPooling2D(pool_size=(2,2)))

model3B.add(Flatten())
model3B.add(Dense(128,activation = 'relu'))
model3B.add(Dropout(0.5))
model3B.add(Dense(10,activation = 'softmax'))

# printing the model summary
model3B.summary()
```

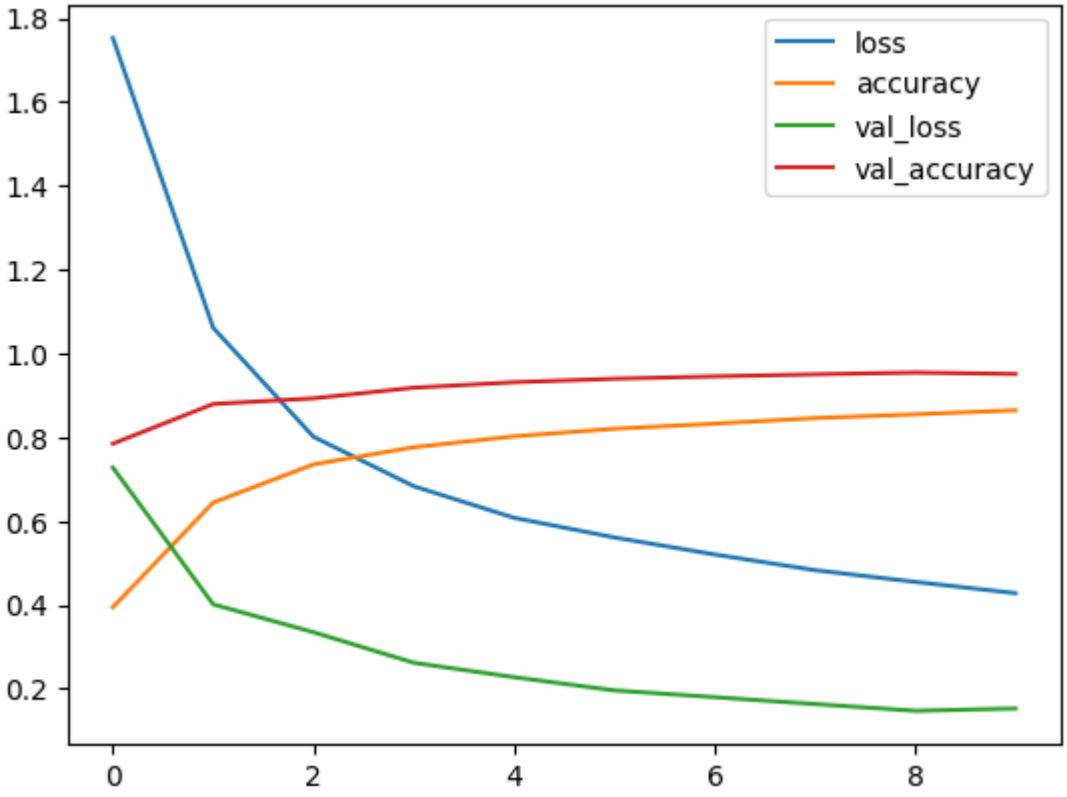
```
Model: "sequential_8"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_23 (Conv2D)          (None, 28, 28, 32)        320

 max_pooling2d_22 (MaxPoolin  (None, 14, 14, 32)       0
 g2D)

 conv2d_24 (Conv2D)          (None, 12, 12, 64)        18496

 max_pooling2d_23 (MaxPoolin  (None, 6, 6, 64)         0
 g2D)

 flatten_8 (Flatten)         (None, 2304)              0

 dense_16 (Dense)            (None, 128)               295040

 dropout_2 (Dropout)         (None, 128)               0

 dense_17 (Dense)            (None, 10)                1290

=================================================================
Total params: 315,146
Trainable params: 315,146
Non-trainable params: 0
_________________________________________________________________
```

In [32]:

```
model3B.compile(optimizer=SGD(learning_rate=0.001, momentum=0.9), loss='cate

# 10 Epochs and batch size 64
history3B=model3B.fit(train_datagen.flow(x_train, keras.utils.to_categorical


pd.DataFrame(history3B.history).plot()
```

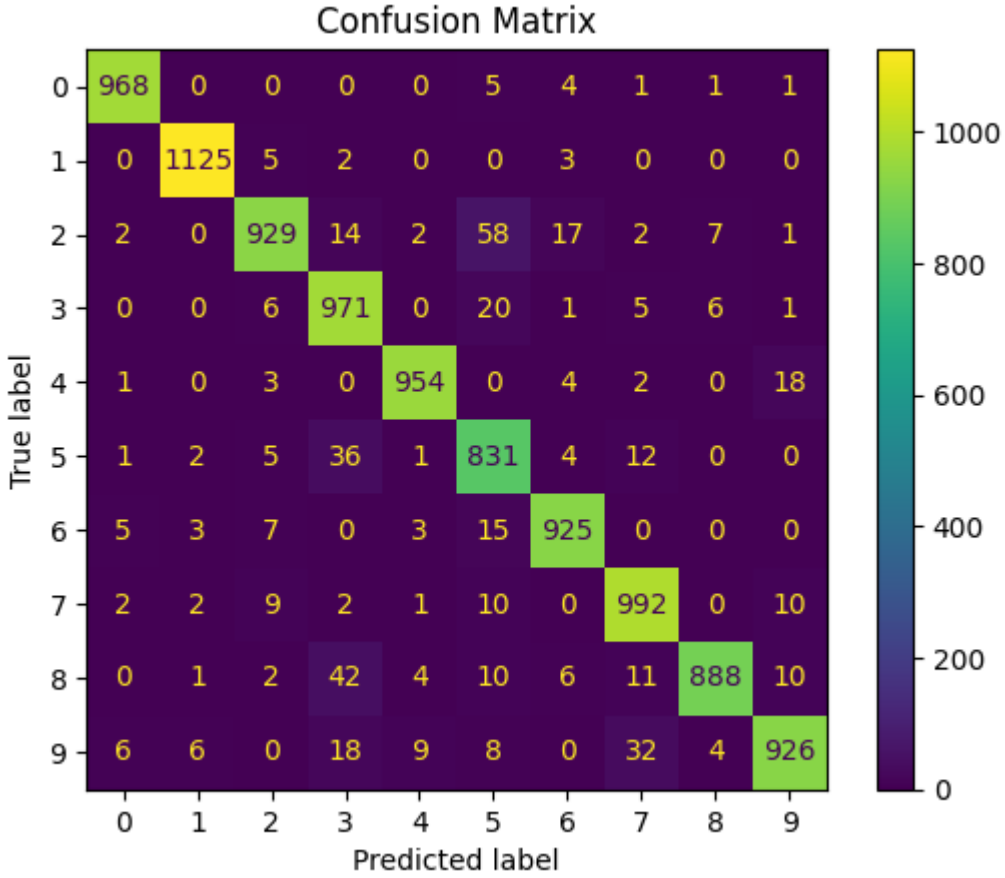
```
Epoch 1/10
938/938 [==============================] - 38s 40ms/step - loss: 1.7528 -
accuracy: 0.3944 - val_loss: 0.7275 - val_accuracy: 0.7845
Epoch 2/10
938/938 [==============================] - 35s 37ms/step - loss: 1.0612 -
accuracy: 0.6441 - val_loss: 0.4011 - val_accuracy: 0.8793
Epoch 3/10
938/938 [==============================] - 40s 43ms/step - loss: 0.8011 -
accuracy: 0.7352 - val_loss: 0.3346 - val_accuracy: 0.8928
Epoch 4/10
938/938 [==============================] - 36s 39ms/step - loss: 0.6831 -
accuracy: 0.7761 - val_loss: 0.2617 - val_accuracy: 0.9182
Epoch 5/10
938/938 [==============================] - 36s 39ms/step - loss: 0.6077 -
accuracy: 0.8021 - val_loss: 0.2273 - val_accuracy: 0.9312
Epoch 6/10
938/938 [==============================] - 36s 39ms/step - loss: 0.5608 -
accuracy: 0.8202 - val_loss: 0.1958 - val_accuracy: 0.9396
Epoch 7/10
938/938 [==============================] - 38s 40ms/step - loss: 0.5202 -
accuracy: 0.8324 - val_loss: 0.1796 - val_accuracy: 0.9452
Epoch 8/10
938/938 [==============================] - 37s 39ms/step - loss: 0.4827 -
accuracy: 0.8457 - val_loss: 0.1632 - val_accuracy: 0.9501
Epoch 9/10
938/938 [==============================] - 36s 39ms/step - loss: 0.4547 -
accuracy: 0.8547 - val_loss: 0.1470 - val_accuracy: 0.9544
Epoch 10/10
938/938 [==============================] - 39s 42ms/step - loss: 0.4277 -
accuracy: 0.8644 - val_loss: 0.1525 - val_accuracy: 0.9509
```

Out[32]: <Axes: >


loss
accuracy
val_loss
val_accuracy
1.8
1.6
1.4
1.2
1.0
0.8
0.6
0.4
0.2
0
2
4
6
8

In [33]:
```
y_pred = np.argmax(model3B.predict(x_test), axis=-1)

print("\n\n" )
# Print the classification report
print(classification_report(y_test, y_pred))

print("\n\n" )
# Print the confusion matrix
from sklearn.metrics import ConfusionMatrixDisplay
plt.figure(dpi=200, figsize=(10,15))
ConfusionMatrixDisplay.from_predictions(y_test,y_pred)
plt.title('Confusion Matrix')
plt.show()


print("\n\n" )
test_loss, test_acc = model3B.evaluate(x_test, keras.utils.to_categorical(y_
print("Training accuracy:", history3B.history['accuracy'][-1])
print("Validation accuracy:", history3B.history['val_accuracy'][-1])
print("Testing accuracy:", test_acc)
```

```
313/313 [==============================] - 2s 5ms/step


              precision    recall  f1-score   support

           0       0.98      0.99      0.99       980
           1       0.99      0.99      0.99      1135
           2       0.96      0.90      0.93      1032
           3       0.89      0.96      0.93      1010
           4       0.98      0.97      0.98       982
           5       0.87      0.93      0.90       892
           6       0.96      0.97      0.96       958
           7       0.94      0.96      0.95      1028
           8       0.98      0.91      0.94       974
           9       0.96      0.92      0.94      1009

    accuracy                           0.95     10000
   macro avg       0.95      0.95      0.95     10000
weighted avg       0.95      0.95      0.95     10000



<Figure size 2000x3000 with 0 Axes>
```



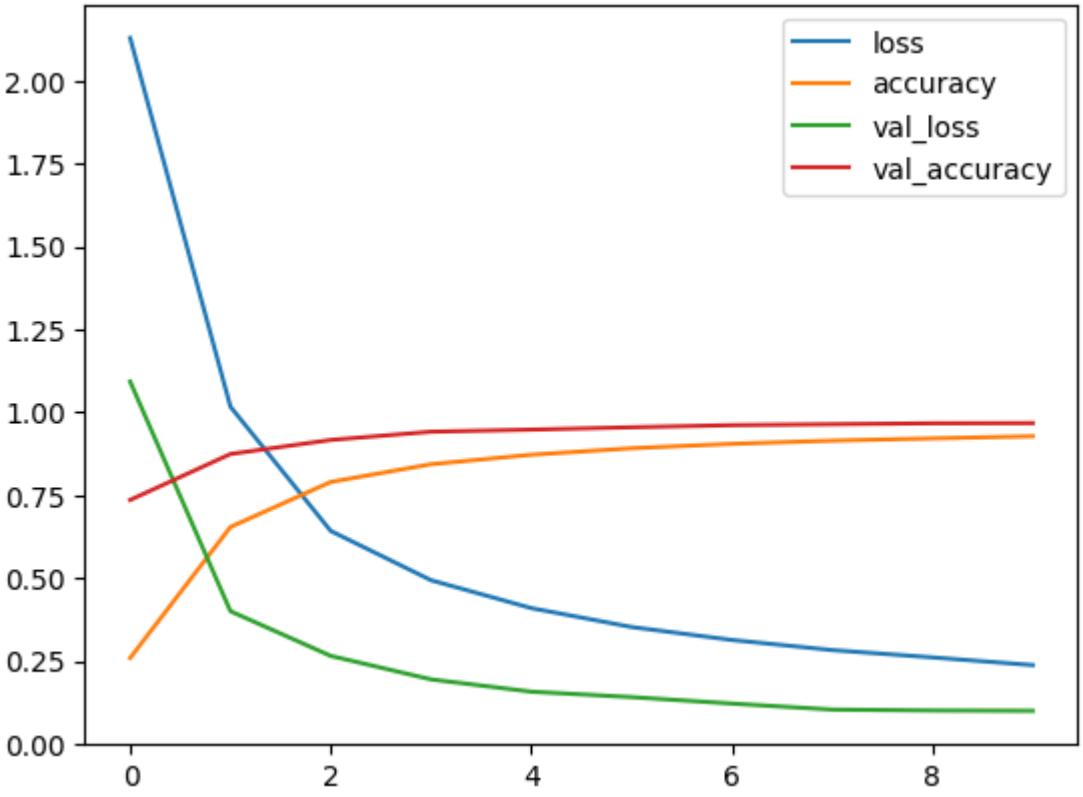

```
313/313 [==============================] - 2s 5ms/step - loss: 0.1525 - ac
curacy: 0.9509
Training accuracy: 0.8643666505813599
Validation accuracy: 0.9509000182151794
Testing accuracy: 0.9509000182151794
```

In [34]:
```
# 4 BLOCK CNN MODEL
model4B = Sequential()
model4B.add(Conv2D(filters=32,kernel_size=(3,3),input_shape=(28,28,1), activ
#model.add(Conv2D(filters=32,kernel_size=(3,3),input_shape=(28,28,1),activat
model4B.add(MaxPooling2D(pool_size=(2,2)))

model4B.add(Conv2D(filters=64,kernel_size=(3,3),activation = 'relu', padding
model4B.add(MaxPooling2D(pool_size=(2,2)))

model4B.add(Conv2D(filters=128,kernel_size=(3,3),activation = 'relu', paddir
model4B.add(MaxPooling2D(pool_size=(2,2)))

model4B.add(Conv2D(filters=256,kernel_size=(3,3),activation = 'relu', paddir
model4B.add(MaxPooling2D(pool_size=(2,2)))

model4B.add(Flatten())
model4B.add(Dense(128,activation = 'relu'))
model4B.add(Dropout(0.5))
model4B.add(Dense(10,activation = 'softmax'))
```

In [35]:
```
model4B.compile(optimizer=SGD(learning_rate=0.001, momentum=0.9), loss='cate

# 10 epochs and batch size 64
history4B=model4B.fit(train_datagen.flow(x_train, keras.utils.to_categorical


pd.DataFrame(history4B.history).plot()
```

```
Epoch 1/10
938/938 [==============================] - 67s 71ms/step - loss: 2.1284 -
accuracy: 0.2588 - val_loss: 1.0922 - val_accuracy: 0.7353
Epoch 2/10
938/938 [==============================] - 61s 65ms/step - loss: 1.0166 -
accuracy: 0.6539 - val_loss: 0.3999 - val_accuracy: 0.8739
Epoch 3/10
938/938 [==============================] - 61s 65ms/step - loss: 0.6419 -
accuracy: 0.7896 - val_loss: 0.2651 - val_accuracy: 0.9163
Epoch 4/10
938/938 [==============================] - 69s 74ms/step - loss: 0.4933 -
accuracy: 0.8431 - val_loss: 0.1943 - val_accuracy: 0.9411
Epoch 5/10
938/938 [==============================] - 66s 70ms/step - loss: 0.4092 -
accuracy: 0.8717 - val_loss: 0.1572 - val_accuracy: 0.9474
Epoch 6/10
938/938 [==============================] - 60s 64ms/step - loss: 0.3521 -
accuracy: 0.8913 - val_loss: 0.1408 - val_accuracy: 0.9545
Epoch 7/10
938/938 [==============================] - 63s 67ms/step - loss: 0.3131 -
accuracy: 0.9046 - val_loss: 0.1214 - val_accuracy: 0.9609
Epoch 8/10
938/938 [==============================] - 58s 62ms/step - loss: 0.2826 -
accuracy: 0.9142 - val_loss: 0.1032 - val_accuracy: 0.9638
Epoch 9/10
938/938 [==============================] - 61s 65ms/step - loss: 0.2605 -
accuracy: 0.9209 - val_loss: 0.1005 - val_accuracy: 0.9665
Epoch 10/10
938/938 [==============================] - 60s 64ms/step - loss: 0.2373 -
accuracy: 0.9275 - val_loss: 0.0995 - val_accuracy: 0.9671
```

Out[35]: <Axes: >

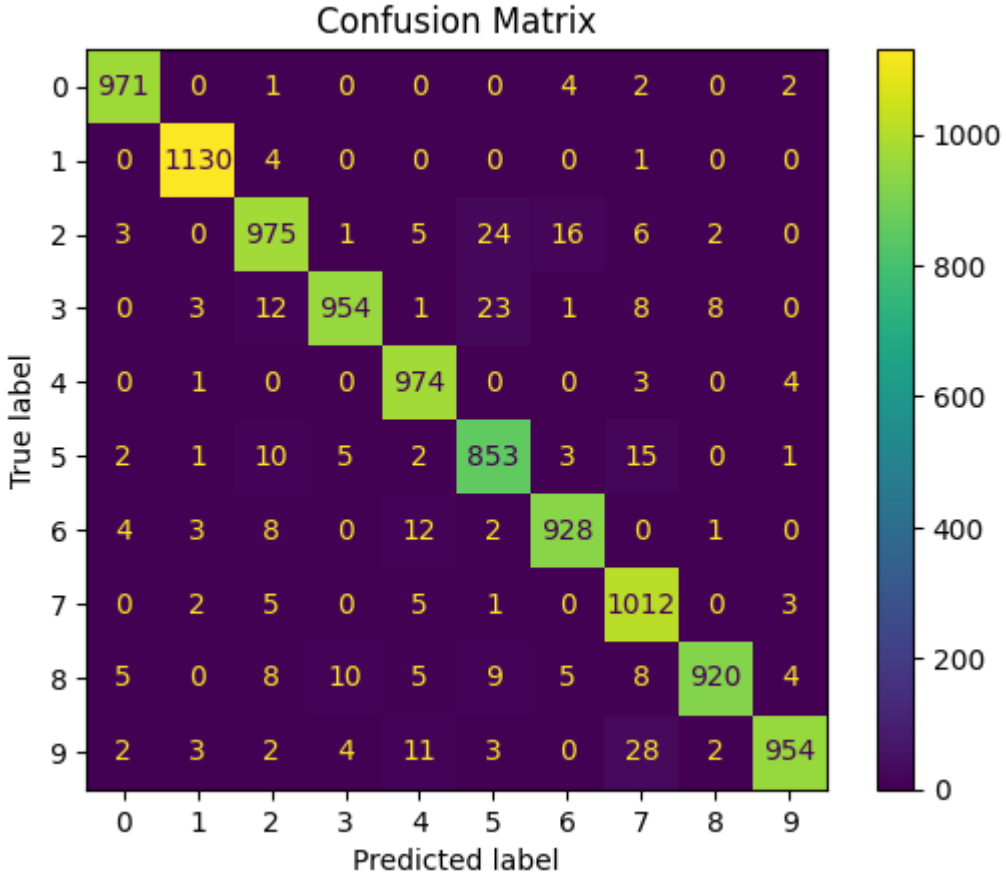

loss
accuracy
val_loss
val_accuracy
2.00
1.75
1.50
1.25
1.00
0.75
0.50
0.25
0.00
0
2
4
6
8

In [36]:
```
# evaluate model
y_pred = np.argmax(model4B.predict(x_test), axis=-1)

print("\n\n" )
# Print the classification report
print(classification_report(y_test, y_pred))

print("\n\n" )
# Print the confusion matrix
from sklearn.metrics import ConfusionMatrixDisplay
plt.figure(dpi=200, figsize=(10,15))
ConfusionMatrixDisplay.from_predictions(y_test,y_pred)
plt.title('Confusion Matrix')
plt.show()


print("\n\n" )
test_loss, test_acc = model4B.evaluate(x_test, keras.utils.to_categorical(y_
print("Training accuracy:", history4B.history['accuracy'][-1])
print("Validation accuracy:", history4B.history['val_accuracy'][-1])
print("Testing accuracy:", test_acc)
```

```
313/313 [==============================] - 2s 6ms/step
```

```
              precision    recall  f1-score   support

           0       0.98      0.99      0.99       980
           1       0.99      1.00      0.99      1135
           2       0.95      0.94      0.95      1032
           3       0.98      0.94      0.96      1010
           4       0.96      0.99      0.98       982
           5       0.93      0.96      0.94       892
           6       0.97      0.97      0.97       958
           7       0.93      0.98      0.96      1028
           8       0.99      0.94      0.96       974
           9       0.99      0.95      0.97      1009

    accuracy                           0.97     10000
   macro avg       0.97      0.97      0.97     10000
weighted avg       0.97      0.97      0.97     10000
```

```
<Figure size 2000x3000 with 0 Axes>
```




```
313/313 [==============================] - 2s 7ms/step - loss: 0.0995 - ac
curacy: 0.9671
Training accuracy: 0.9275166392326355
Validation accuracy: 0.9671000242233276
Testing accuracy: 0.9671000242233276
```

COMPONENT 3C

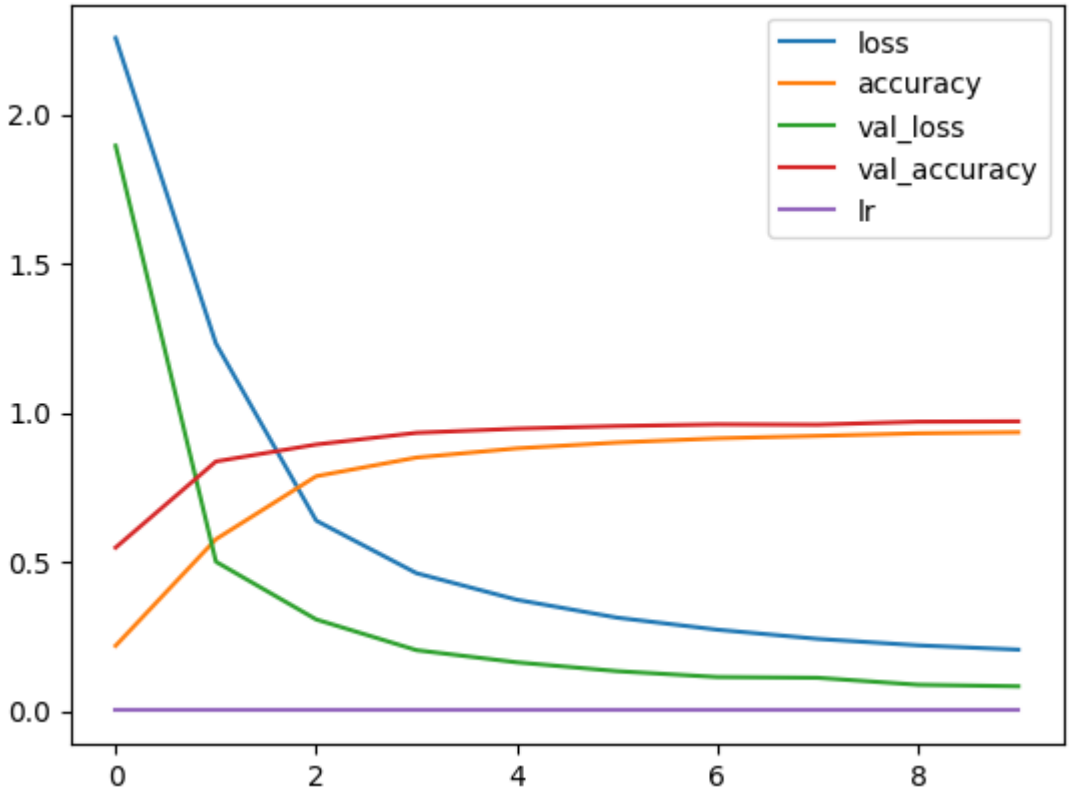
In [37]:
```
# model with learning rate
model3C = Sequential()
model3C.add(Conv2D(filters=32,kernel_size=(3,3),input_shape=(28,28,1), activ
#model.add(Conv2D(filters=32,kernel_size=(3,3),input_shape=(28,28,1),activat
model3C.add(MaxPooling2D(pool_size=(2,2)))

model3C.add(Conv2D(filters=64,kernel_size=(3,3),activation = 'relu', padding
model3C.add(MaxPooling2D(pool_size=(2,2)))

model3C.add(Conv2D(filters=128,kernel_size=(3,3),activation = 'relu', paddir
model3C.add(MaxPooling2D(pool_size=(2,2)))

model3C.add(Conv2D(filters=256,kernel_size=(3,3),activation = 'relu', paddir
model3C.add(MaxPooling2D(pool_size=(2,2)))

model3C.add(Flatten())
model3C.add(Dense(512,activation = 'relu'))
model3C.add(Dropout(0.5))
model3C.add(Dense(10,activation = 'softmax'))
```

In [39]:
```
from keras.callbacks import Callback
from keras import backend as K

# Learning Rate Adjustment
class CustomLearningRateScheduler(Callback):
    def __init__(self, schedule):
        super(CustomLearningRateScheduler, self).__init__()
        self.schedule = schedule

    def on_epoch_begin(self, epoch, logs=None):
        if not hasattr(self.model.optimizer, "lr"):
            raise ValueError('Optimizer must have a "lr" attribute.')
        # learning rate from model's optimizer
        lr = float(K.get_value(self.model.optimizer.learning_rate))
        # Calling scheduled learning rate
        scheduled_lr = self.schedule(epoch, lr)
        # learning rate in optimizer
        K.set_value(self.model.optimizer.learning_rate, scheduled_lr)
        print(f"Epoch {epoch+1}: Learning rate is {scheduled_lr}.")

# Defining the learning rate schedule function
def lr_schedule(epoch, lr):
    if epoch > 10:
        lr = 0.0005
    if epoch > 20:
        lr = 0.0001
    return lr
```

In [40]:
```
# evaluate model3C
model3C.compile(optimizer=SGD(learning_rate=0.001, momentum=0.9), loss='cate
lr_scheduler = LearningRateScheduler(lr_schedule)
# 10 epochs and 64 batch size
history3C=model3C.fit(train_datagen.flow(x_train, keras.utils.to_categorical


pd.DataFrame(history3C.history).plot()
```

```
Epoch 1/10
938/938 [==============================] - 61s 65ms/step - loss: 2.2556 -
accuracy: 0.2191 - val_loss: 1.8946 - val_accuracy: 0.5481 - lr: 0.0010
Epoch 2/10
938/938 [==============================] - 63s 67ms/step - loss: 1.2319 -
accuracy: 0.5760 - val_loss: 0.5006 - val_accuracy: 0.8366 - lr: 0.0010
Epoch 3/10
938/938 [==============================] - 69s 73ms/step - loss: 0.6383 -
accuracy: 0.7872 - val_loss: 0.3071 - val_accuracy: 0.8935 - lr: 0.0010
Epoch 4/10
938/938 [==============================] - 63s 67ms/step - loss: 0.4621 -
accuracy: 0.8498 - val_loss: 0.2042 - val_accuracy: 0.9326 - lr: 0.0010
Epoch 5/10
938/938 [==============================] - 66s 70ms/step - loss: 0.3732 -
accuracy: 0.8805 - val_loss: 0.1635 - val_accuracy: 0.9465 - lr: 0.0010
Epoch 6/10
938/938 [==============================] - 64s 68ms/step - loss: 0.3132 -
accuracy: 0.9001 - val_loss: 0.1339 - val_accuracy: 0.9552 - lr: 0.0010
Epoch 7/10
938/938 [==============================] - 62s 67ms/step - loss: 0.2728 -
accuracy: 0.9140 - val_loss: 0.1142 - val_accuracy: 0.9609 - lr: 0.0010
Epoch 8/10
938/938 [==============================] - 61s 65ms/step - loss: 0.2421 -
accuracy: 0.9223 - val_loss: 0.1117 - val_accuracy: 0.9599 - lr: 0.0010
Epoch 9/10
938/938 [==============================] - 62s 66ms/step - loss: 0.2204 -
accuracy: 0.9309 - val_loss: 0.0883 - val_accuracy: 0.9698 - lr: 0.0010
Epoch 10/10
938/938 [==============================] - 61s 65ms/step - loss: 0.2060 -
accuracy: 0.9344 - val_loss: 0.0831 - val_accuracy: 0.9710 - lr: 0.0010
```

Out[40]: <Axes: >

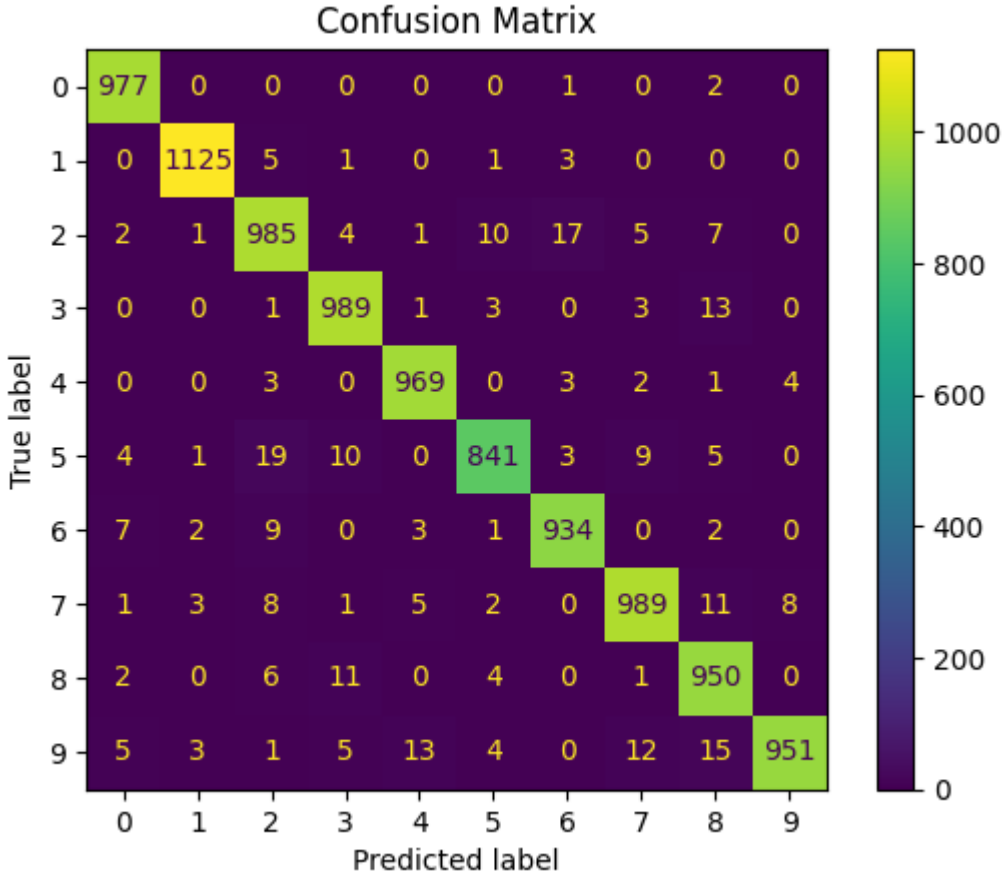

loss
accuracy
val_loss
val_accuracy
lr
2.0
1.5
1.0
0.5
0.0
0
2
4
6
8

In [41]:
```
# predict test data
y_pred = np.argmax(model3C.predict(x_test), axis=-1)

print("\n\n" )
# Print classification report
print(classification_report(y_test, y_pred))

print("\n\n" )
# Print the confusion matrix
from sklearn.metrics import ConfusionMatrixDisplay
plt.figure(dpi=200, figsize=(10,15))
ConfusionMatrixDisplay.from_predictions(y_test,y_pred)
plt.title('Confusion Matrix')
plt.show()


print("\n\n" )
test_loss, test_acc = model3C.evaluate(x_test, keras.utils.to_categorical(y_
print("Training accuracy:", history3C.history['accuracy'][-1])
print("Validation accuracy:", history3C.history['val_accuracy'][-1])
print("Testing accuracy:", test_acc)
```

```
313/313 [==============================] - 2s 5ms/step
```

```
              precision    recall  f1-score   support

           0       0.98      1.00      0.99       980
           1       0.99      0.99      0.99      1135
           2       0.95      0.95      0.95      1032
           3       0.97      0.98      0.97      1010
           4       0.98      0.99      0.98       982
           5       0.97      0.94      0.96       892
           6       0.97      0.97      0.97       958
           7       0.97      0.96      0.97      1028
           8       0.94      0.98      0.96       974
           9       0.99      0.94      0.96      1009

    accuracy                           0.97     10000
   macro avg       0.97      0.97      0.97     10000
weighted avg       0.97      0.97      0.97     10000
```

```
<Figure size 2000x3000 with 0 Axes>
```



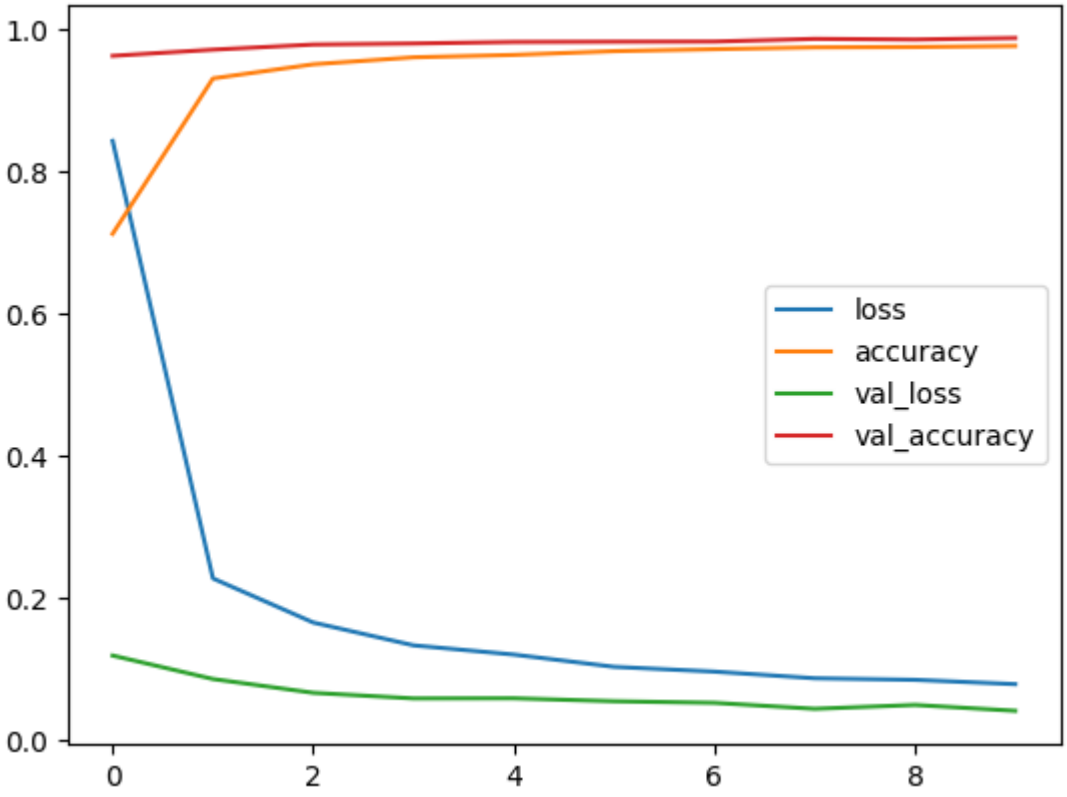

```
313/313 [==============================] - 2s 7ms/step - loss: 0.0831 - ac
curacy: 0.9710
Training accuracy: 0.9344000220298767
Validation accuracy: 0.9710000157356262
Testing accuracy: 0.9710000157356262
```

In [42]:
```
# model without learning rate
model3C2 = Sequential()
model3C2.add(Conv2D(filters=32,kernel_size=(3,3),input_shape=(28,28,1), acti
#model.add(Conv2D(filters=32,kernel_size=(3,3),input_shape=(28,28,1),activat
model3C2.add(MaxPooling2D(pool_size=(2,2)))

model3C2.add(Conv2D(filters=64,kernel_size=(3,3),activation = 'relu', paddir
model3C2.add(MaxPooling2D(pool_size=(2,2)))

model3C2.add(Conv2D(filters=128,kernel_size=(3,3),activation = 'relu', paddi
model3C2.add(MaxPooling2D(pool_size=(2,2)))

model3C2.add(Conv2D(filters=256,kernel_size=(3,3),activation = 'relu', paddi
model3C2.add(MaxPooling2D(pool_size=(2,2)))

model3C2.add(Flatten())
model3C2.add(Dense(512,activation = 'relu'))
model3C2.add(Dropout(0.5))
model3C2.add(Dense(10,activation = 'softmax'))
```

In [43]:
```
# Compile model3C2 without specifying learning rate
model3C2.compile(optimizer=SGD(momentum=0.9), loss='categorical_crossentropy

# Train the model for 10 epochs with a batch size of 64
history3C2 = model3C2.fit(train_datagen.flow(x_train, keras.utils.to_categor
                          epochs=10,
                          validation_data=(x_test, keras.utils.to_categorica
                         )

# Plot the training history
pd.DataFrame(history3C2.history).plot()
```

```
Epoch 1/10
938/938 [==============================] - 68s 72ms/step - loss: 0.8418 -
accuracy: 0.7111 - val_loss: 0.1180 - val_accuracy: 0.9614
Epoch 2/10
938/938 [==============================] - 61s 65ms/step - loss: 0.2265 -
accuracy: 0.9297 - val_loss: 0.0850 - val_accuracy: 0.9701
Epoch 3/10
938/938 [==============================] - 59s 63ms/step - loss: 0.1644 -
accuracy: 0.9495 - val_loss: 0.0655 - val_accuracy: 0.9773
Epoch 4/10
938/938 [==============================] - 66s 71ms/step - loss: 0.1323 -
accuracy: 0.9593 - val_loss: 0.0577 - val_accuracy: 0.9788
Epoch 5/10
938/938 [==============================] - 66s 70ms/step - loss: 0.1193 -
accuracy: 0.9627 - val_loss: 0.0579 - val_accuracy: 0.9809
Epoch 6/10
938/938 [==============================] - 62s 66ms/step - loss: 0.1021 -
accuracy: 0.9680 - val_loss: 0.0537 - val_accuracy: 0.9813
Epoch 7/10
938/938 [==============================] - 57s 61ms/step - loss: 0.0954 -
accuracy: 0.9708 - val_loss: 0.0515 - val_accuracy: 0.9815
Epoch 8/10
938/938 [==============================] - 58s 62ms/step - loss: 0.0860 -
accuracy: 0.9733 - val_loss: 0.0430 - val_accuracy: 0.9853
Epoch 9/10
938/938 [==============================] - 57s 61ms/step - loss: 0.0838 -
accuracy: 0.9738 - val_loss: 0.0485 - val_accuracy: 0.9844
Epoch 10/10
938/938 [==============================] - 62s 66ms/step - loss: 0.0777 -
accuracy: 0.9753 - val_loss: 0.0402 - val_accuracy: 0.9865
```

Out[43]: <Axes: >

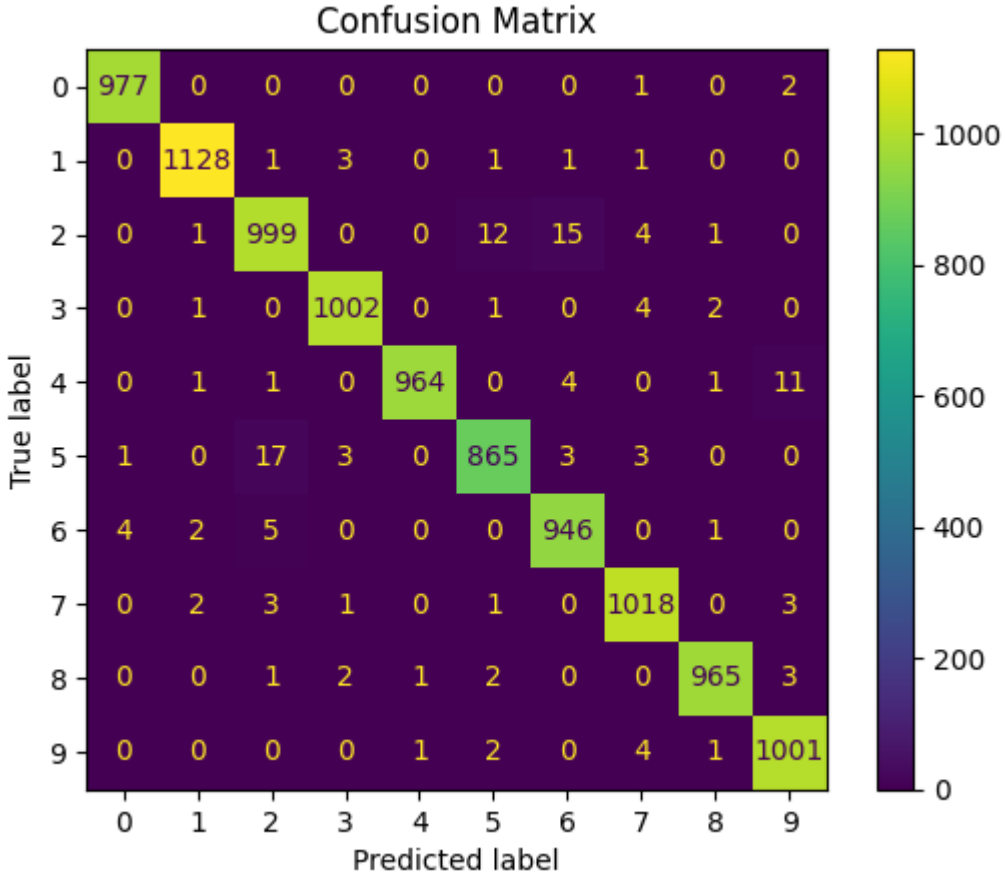

1.0
0.8
0.6
0.4
0.2
0.0
0
2
4
6
8
loss
accuracy
val_loss
val_accuracy

In [44]:
```
# predict test data
y_pred = np.argmax(model3C2.predict(x_test), axis=-1)

print("\n\n" )
# Print classification report
print(classification_report(y_test, y_pred))

print("\n\n" )
# Print the confusion matrix
from sklearn.metrics import ConfusionMatrixDisplay
plt.figure(dpi=200, figsize=(10,15))
ConfusionMatrixDisplay.from_predictions(y_test,y_pred)
plt.title('Confusion Matrix')
plt.show()


print("\n\n" )
test_loss, test_acc = model3C.evaluate(x_test, keras.utils.to_categorical(y_
print("Training accuracy:", history3C2.history['accuracy'][-1])
print("Validation accuracy:", history3C2.history['val_accuracy'][-1])
print("Testing accuracy:", test_acc)
```

```
313/313 [==============================] - 2s 6ms/step
```

```
              precision    recall  f1-score   support

           0       0.99      1.00      1.00       980
           1       0.99      0.99      0.99      1135
           2       0.97      0.97      0.97      1032
           3       0.99      0.99      0.99      1010
           4       1.00      0.98      0.99       982
           5       0.98      0.97      0.97       892
           6       0.98      0.99      0.98       958
           7       0.98      0.99      0.99      1028
           8       0.99      0.99      0.99       974
           9       0.98      0.99      0.99      1009

    accuracy                           0.99     10000
   macro avg       0.99      0.99      0.99     10000
weighted avg       0.99      0.99      0.99     10000
```

```
<Figure size 2000x3000 with 0 Axes>
```




```
313/313 [==============================] - 2s 6ms/step - loss: 0.0831 - ac
curacy: 0.9710
Training accuracy: 0.9752833247184753
Validation accuracy: 0.9865000247955322
Testing accuracy: 0.9710000157356262
```

In [ ]: