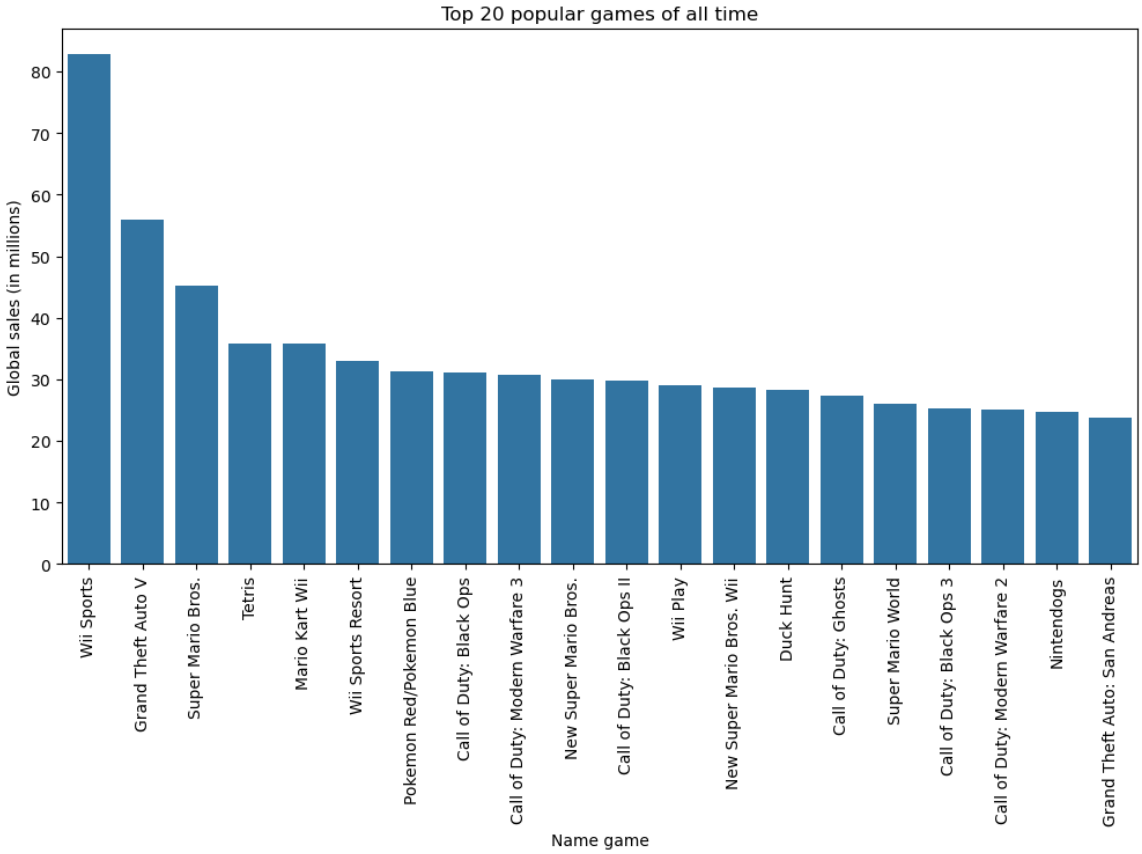
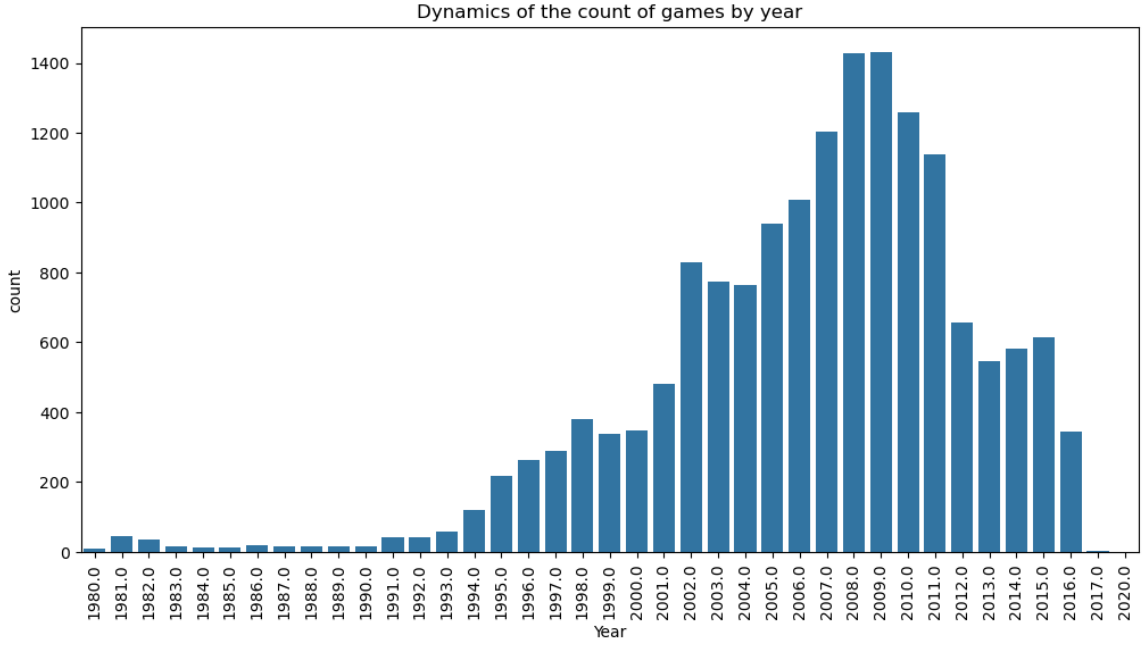
In [27]:
```
import numpy as np
import pandas as pd
from scipy import stats
import matplotlib.pyplot as plt

from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split

import tensorflow as tf

from tensorflow_addons.metrics import RSquare
import seaborn as sns
import boto3
from sagemaker import get_execution_role
```

In [8]:
```
# Get the execution role
role = get_execution_role()

# Specify the S3 bucket name and the key (path) to your CSV file
bucket = 'sagemaker-us-east-2-905418369030'
key = 'vgsales.csv'

# Create an S3 client
s3 = boto3.client('s3')

# Download the dataset from S3 to the notebook instance
s3.download_file(bucket, key, 'vgsales.csv')


data = pd.read_csv('vgsales.csv', index_col='Rank')
df=data
```

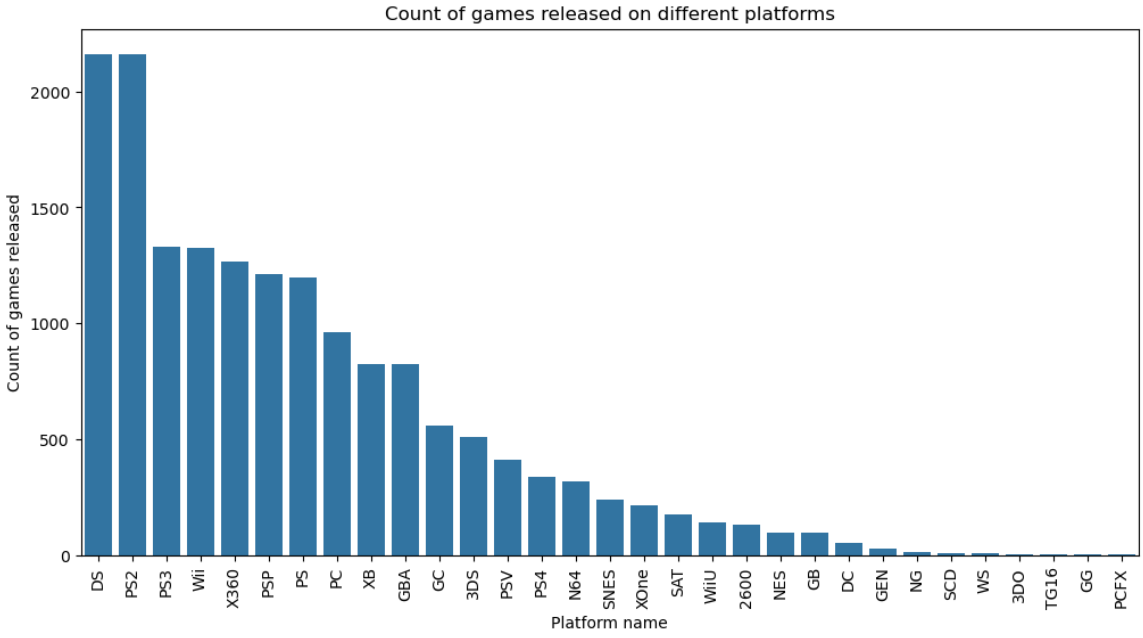
In [13]:
```
top_games = df.groupby('Name')['Global_Sales'].sum().sort_values(ascending = Fal
plt.figure(figsize = (12, 6))
sns.barplot(data = top_games, x = 'Name', y = 'Global_Sales').set(title = 'Top 2
plt.ylabel('Global sales (in millions)')
plt.xlabel('Name game')
plt.xticks(rotation = 90)
plt.show()
```



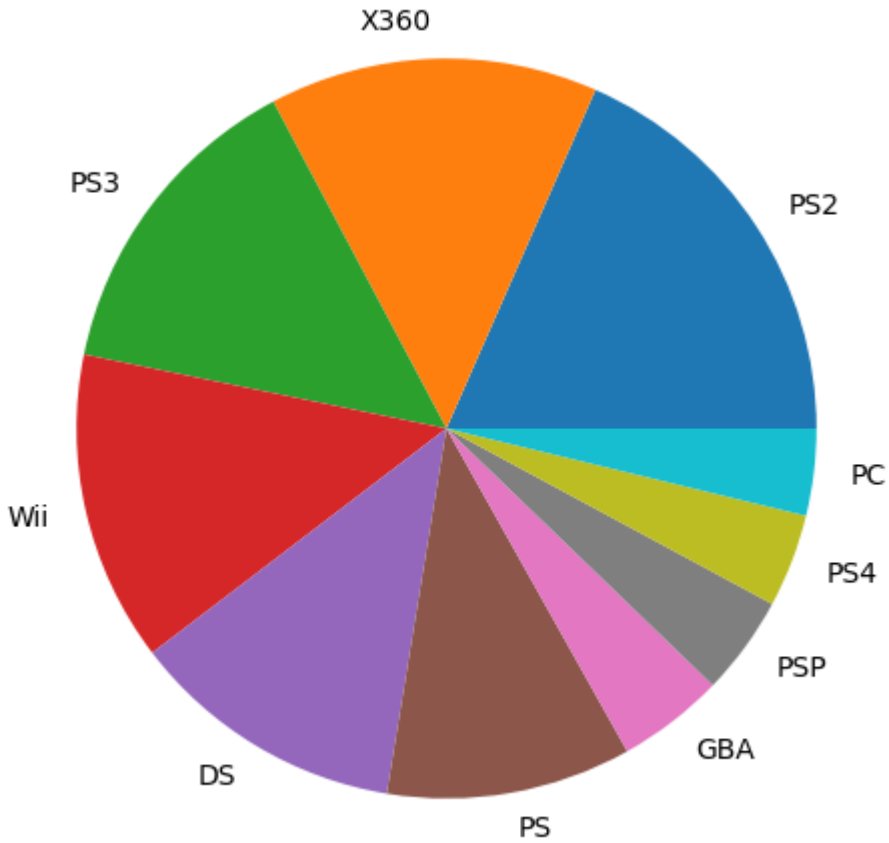
In [14]:
```
year_games = df['Year'].value_counts().reset_index().sort_values(by = 'count', a
plt.figure(figsize = (12, 6))
sns.barplot(data = year_games, x = 'Year', y = 'count').set(title = 'Dynamics of
plt.xticks(rotation = 90)
plt.show()
```



In [15]:
```
game_on_platform = df['Platform'].value_counts().reset_index()
plt.figure(figsize = (12, 6))
sns.barplot(data = game_on_platform, x = 'Platform', y = 'count').set(title = 'C
plt.ylabel('Count of games released')
plt.xlabel('Platform name')
plt.xticks(rotation = 90)
plt.show()
```

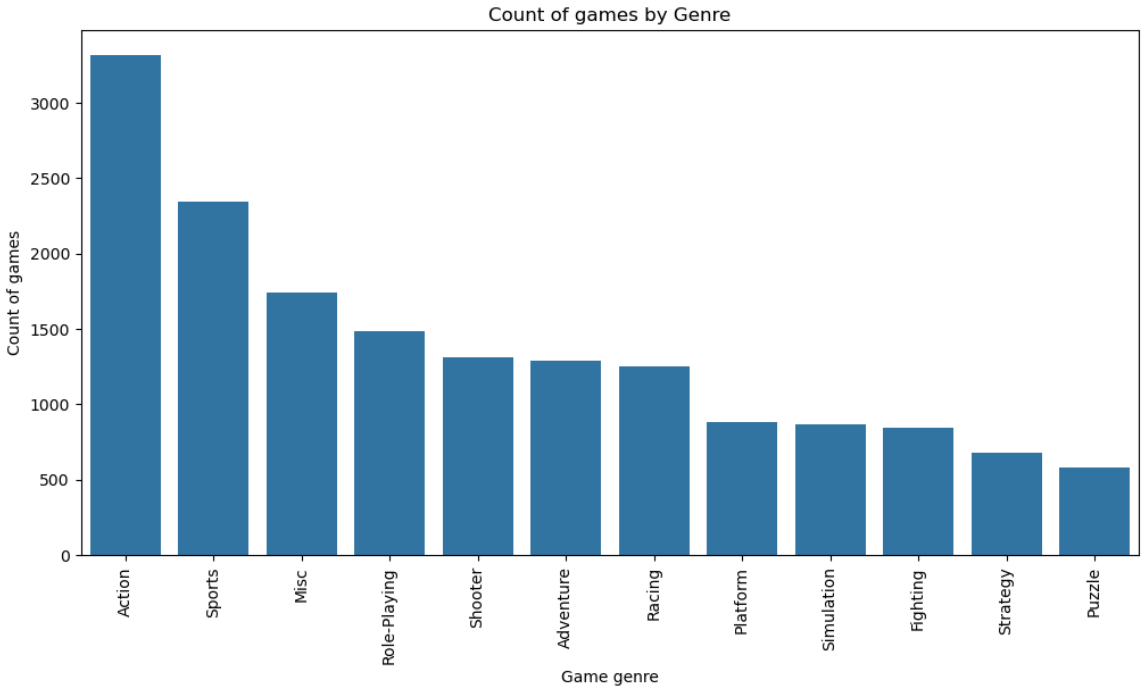


In [19]:
```
ranking_platforms = df.groupby('Platform')['Global_Sales'].sum().reset_index().s
plt.figure(figsize = (12, 6))
plt.pie(ranking_platforms['Global_Sales'].head(10), labels = ranking_platforms['
plt.show()
```



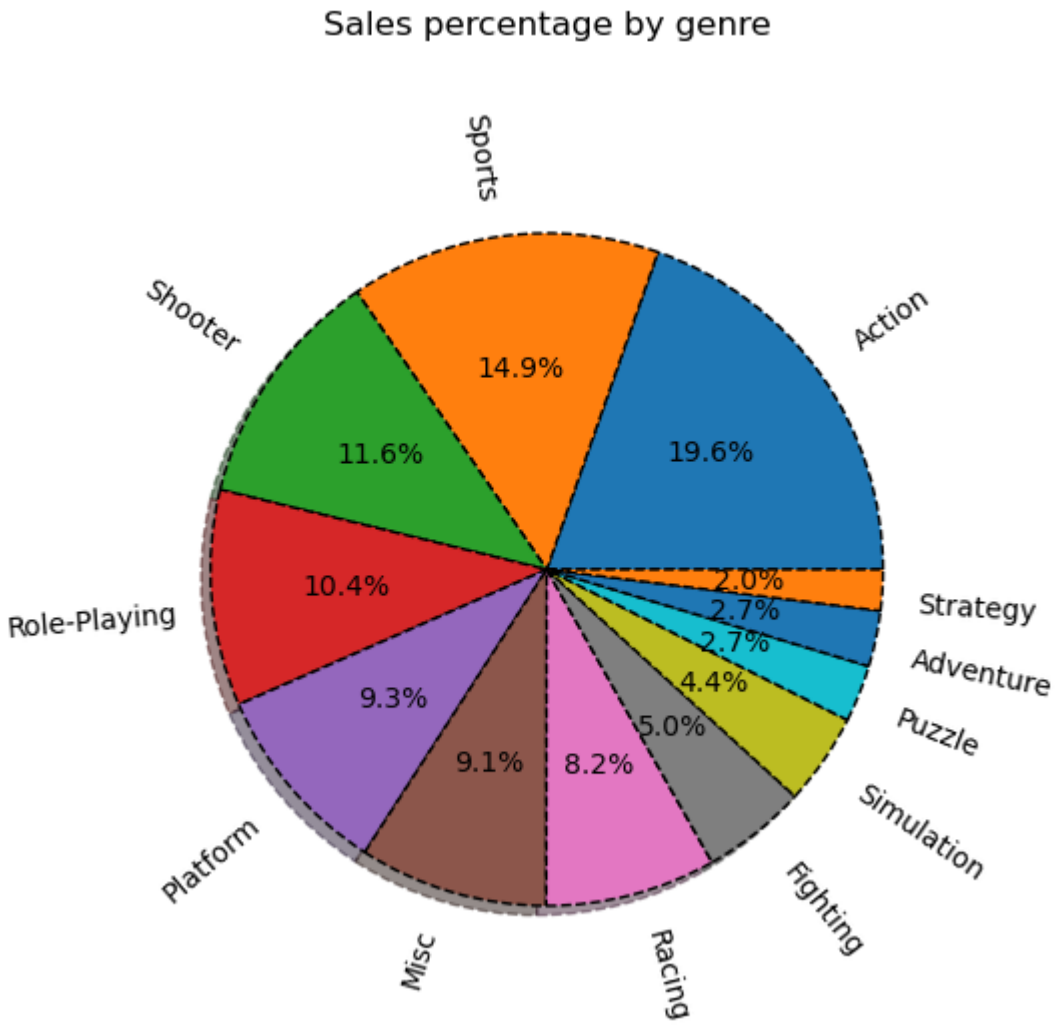
In [20]:
```
count_genre = df['Genre'].value_counts().reset_index()
plt.figure(figsize = (12, 6))
sns.barplot(data = count_genre, x = 'Genre', y = 'count').set(title = 'Count of
plt.ylabel('Count of games')
plt.xlabel('Game genre')
```

```
plt.xticks(rotation = 90)
plt.show()
```

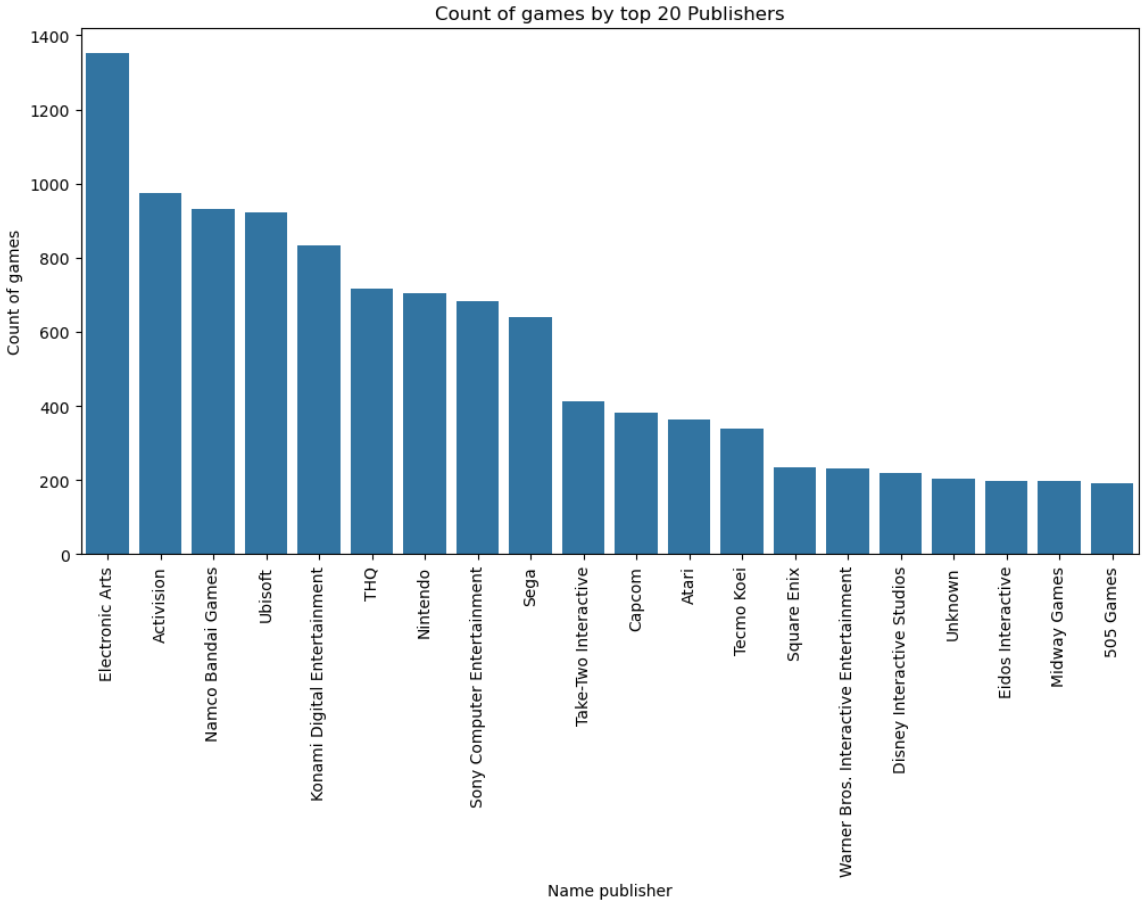


In [21]:
```
genres_sales = df.groupby('Genre')['Global_Sales'].sum().reset_index().sort_valu
genres_sales['percentage_of_total_sales'] = round(genres_sales['Global_Sales'] /
plt.figure(figsize = (15, 10))

fig, ax = plt.subplots()
ax.pie(genres_sales['Global_Sales'], labels=genres_sales['Genre'], autopct='%1.1
ax.axis("equal")
plt.title('Sales percentage by genre', x= 0.5 , y= 1.2 )
plt.show()
```

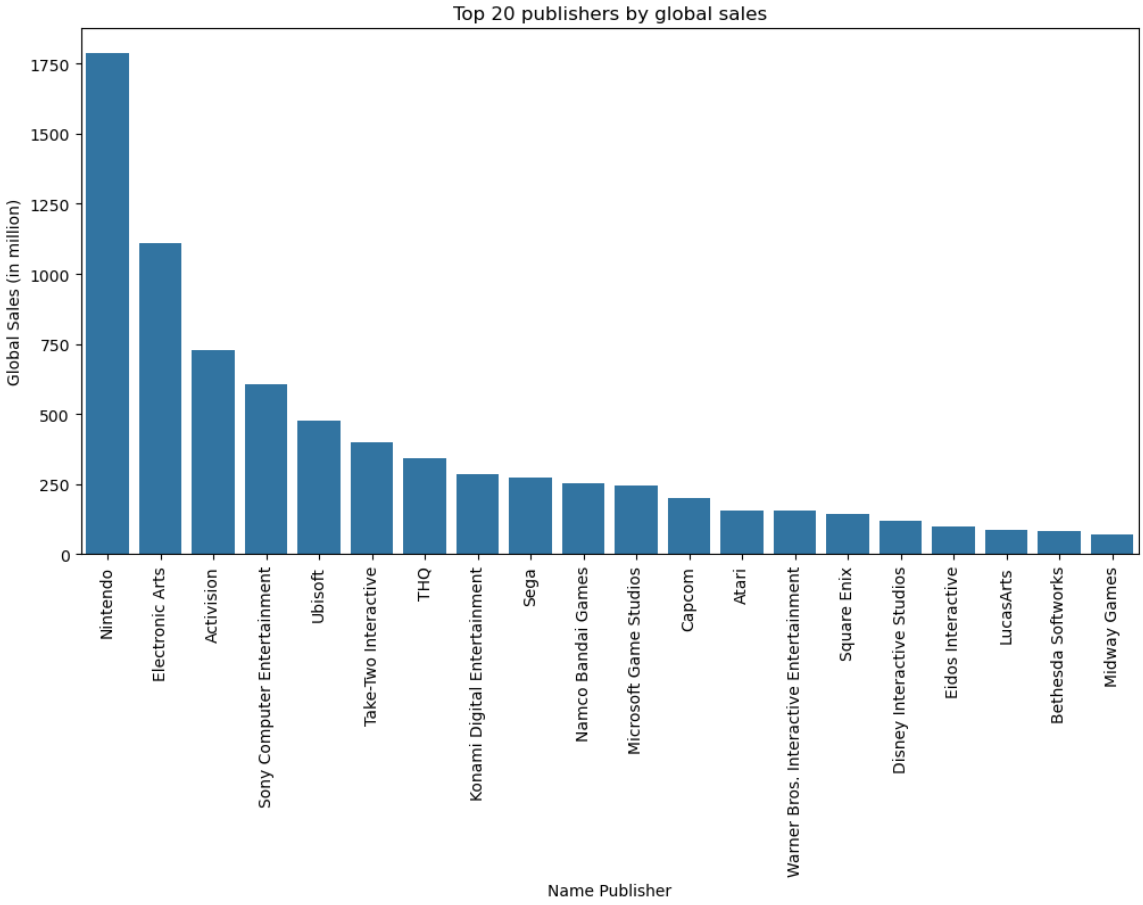
```
<Figure size 1500x1000 with 0 Axes>
```



In [22]:
```
count_publisher = df['Publisher'].value_counts().reset_index().head(20)
plt.figure(figsize = (12, 6))
sns.barplot(data = count_publisher, x = 'Publisher', y = 'count').set(title = 'C
plt.ylabel('Count of games')
plt.xlabel('Name publisher')
plt.xticks(rotation = 90)
plt.show()
```

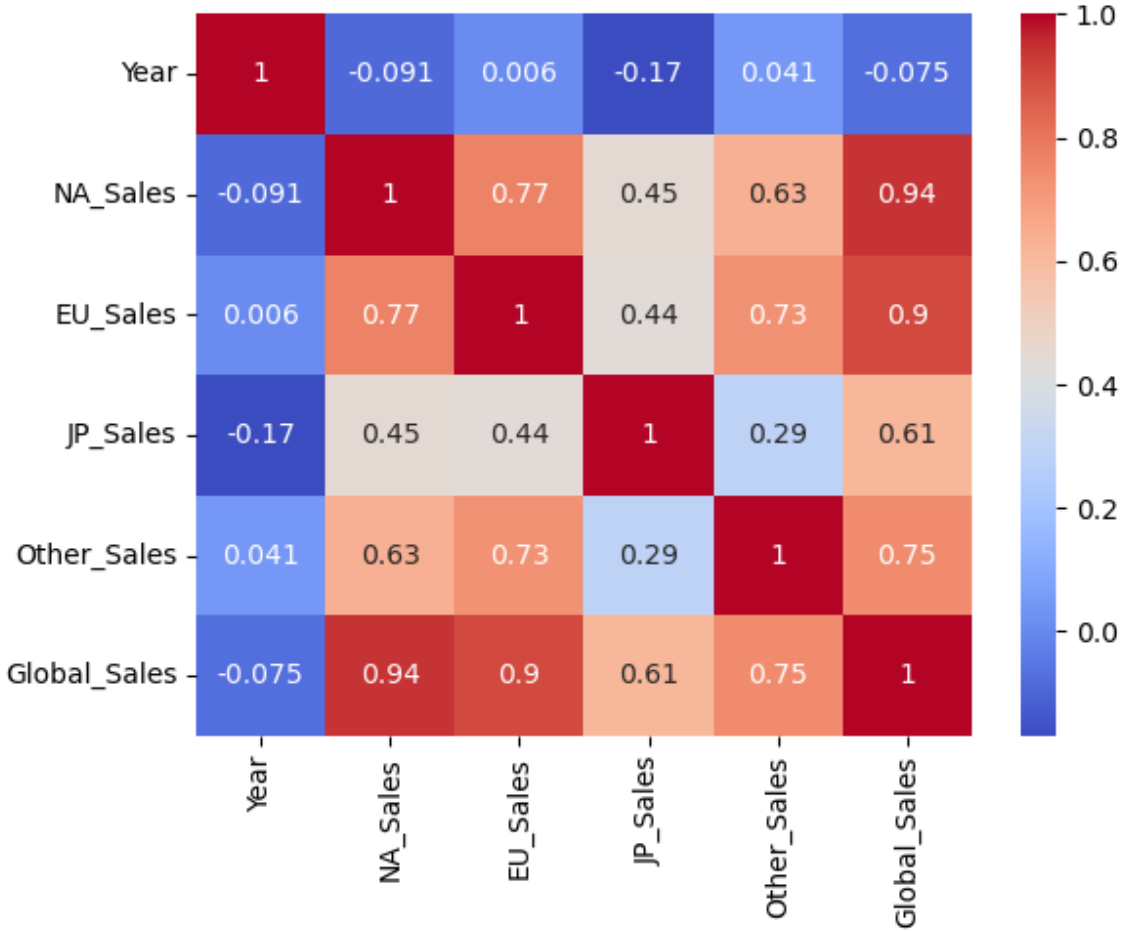


In [23]:
```
ranking_publisher = df.groupby('Publisher')['Global_Sales'].sum().reset_index().
plt.figure(figsize = (12, 6))
sns.barplot(data = ranking_publisher, x = 'Publisher', y = 'Global_Sales').set(t
plt.ylabel('Global Sales (in million)')
plt.xlabel('Name Publisher')
plt.xticks(rotation = 90)
plt.show()
```



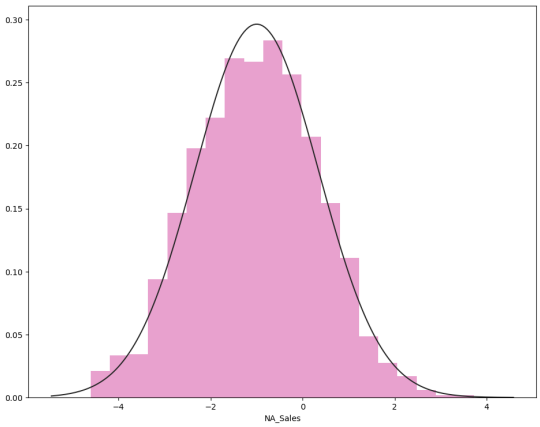
In [24]:
```
df_sales = df[['Year','NA_Sales', 'EU_Sales', 'JP_Sales', 'Other_Sales', 'Global
df_sales
sns.heatmap(df_sales.corr(), annot = True, cmap= 'coolwarm')
```

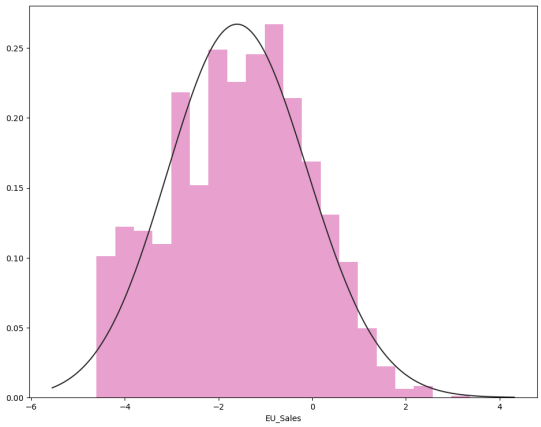
Out[24]: <Axes: >

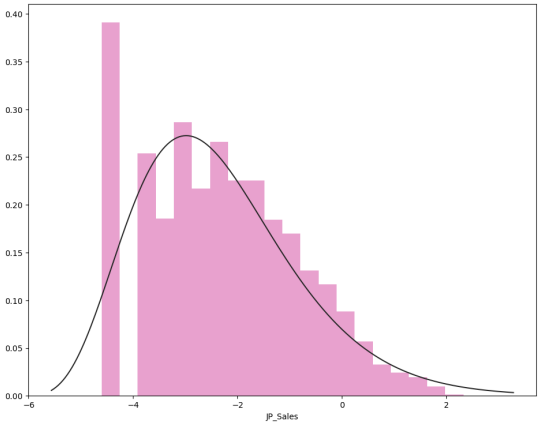


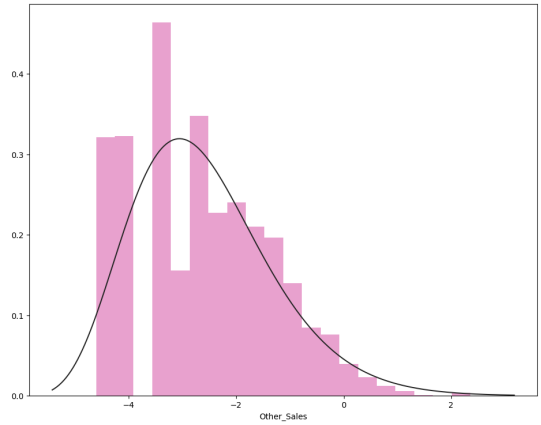
In [28]:
```
data_hist_sales = df.copy()
data_hist_sales = data_hist_sales[data_hist_sales.NA_Sales != 0]
data_hist_sales = data_hist_sales[data_hist_sales.EU_Sales != 0]
data_hist_sales = data_hist_sales[data_hist_sales.Other_Sales != 0]
data_hist_sales = data_hist_sales[data_hist_sales.JP_Sales != 0]
data_hist_sales = data_hist_sales[data_hist_sales.Global_Sales != 0]

plt.figure(figsize=(25,30))
sales_columns = ['NA_Sales', 'EU_Sales', 'JP_Sales', 'Other_Sales', 'Global_Sale
for i, column in enumerate(sales_columns):
    plt.subplot(3, 2, i + 1)
    sns.distplot(np.log(data_hist_sales[column]), bins=20, kde=False, fit = stat
```

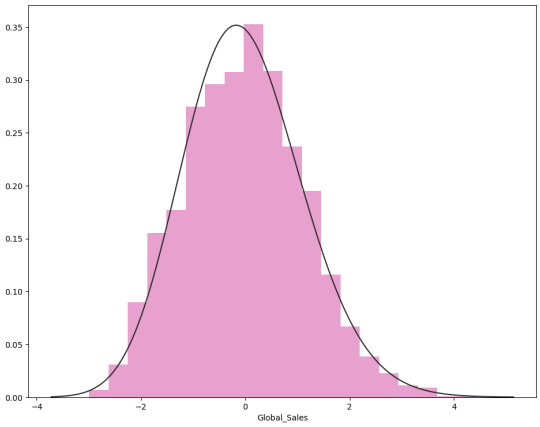
```
/tmp/ipykernel_5374/3986433281.py:12: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(np.log(data_hist_sales[column]), bins=20, kde=False, fit = stat
s.gamma, color = 'mediumvioletred')
```









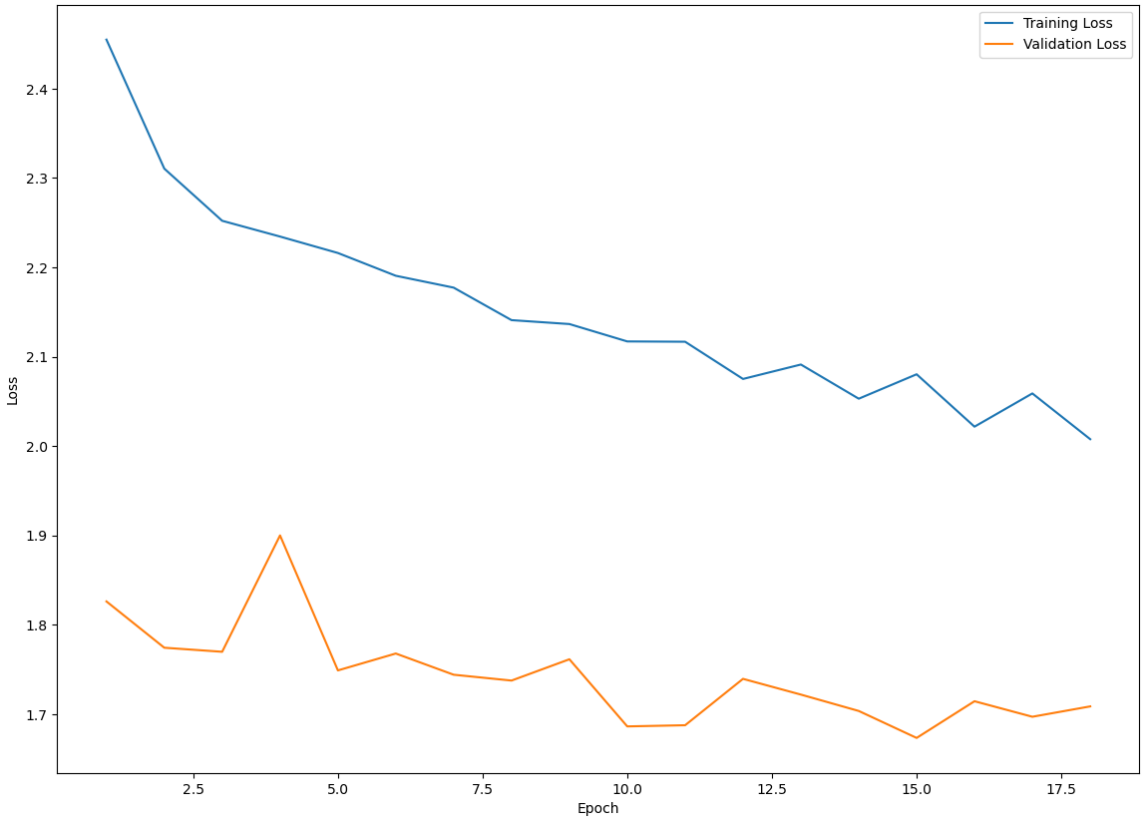


In [3]:
```
columns_to_drop = ['Name', 'NA_Sales', 'EU_Sales', 'JP_Sales', 'Other_Sales']

data.drop(columns_to_drop, axis=1, inplace=True)
data['Year'] = data['Year'].fillna(data['Year'].mean())
data = data.dropna(axis=0)
counts = data['Publisher'].value_counts()

data['Publisher'] = data['Publisher'].apply(lambda x: 'Small Publisher' if count
onehot_columns = ['Platform', 'Genre', 'Publisher']
```

In [4]:
```
def onehot_encode(data, columns):
    for column in columns:
        dummies = pd.get_dummies(data[column])
        data = pd.concat([data, dummies], axis=1)
        data.drop(column, axis=1, inplace=True)
    return data
data = onehot_encode(data, onehot_columns)
```

In [5]:
```
y = data['Global_Sales']
X = data.drop('Global_Sales', axis=1)
scaler = StandardScaler()

X = scaler.fit_transform(X)
X_train, X_test, y_train, y_test = train_test_split(X, y, train_size=0.8)
inputs = tf.keras.Input(shape=(91,))
x = tf.keras.layers.Dense(128, activation='relu')(inputs)
x = tf.keras.layers.Dense(128, activation='relu')(x)
outputs = tf.keras.layers.Dense(1)(x)

model = tf.keras.Model(inputs=inputs, outputs=outputs)


optimizer = tf.keras.optimizers.RMSprop(0.001)

model.compile(
    optimizer=optimizer,
    loss='mse'
)


batch_size = 64
epochs = 18

history = model.fit(
    X_train,
    y_train,
    validation_split=0.2,
    batch_size=batch_size,
    epochs=epochs,
    verbose=0
)
plt.figure(figsize=(14, 10))

epochs_range = range(1, epochs + 1)
train_loss = history.history['loss']
val_loss = history.history['val_loss']

plt.plot(epochs_range, train_loss, label="Training Loss")
plt.plot(epochs_range, val_loss, label="Validation Loss")

plt.xlabel("Epoch")
plt.ylabel("Loss")
plt.legend()

plt.show()
```

```
2024-02-12 20:28:42.933511: E external/local_xla/xla/stream_executor/cuda/cuda_
driver.cc:274] failed call to cuInit: CUDA_ERROR_NO_DEVICE: no CUDA-capable dev
ice is detected
```



In [39]:
```
y_test
```

Out[39]:
```
Rank
3341     0.60
16176    0.01
12303    0.06
4426     0.44
1243     1.51
         ...
5805     0.31
5074     0.38
14130    0.03
16126    0.01
9768     0.12
Name: Global_Sales, Length: 3308, dtype: float64
```

In [37]:
```
y_train
```

Out[37]:
```
Rank
15075    0.02
3397     0.59
811      2.07
757      2.16
12726    0.06
         ...
14525    0.03
12617    0.06
7150     0.22
622      2.48
13756    0.04
Name: Global_Sales, Length: 13232, dtype: float64
```

In [ ]: