Tech Presentation

Mina Sung

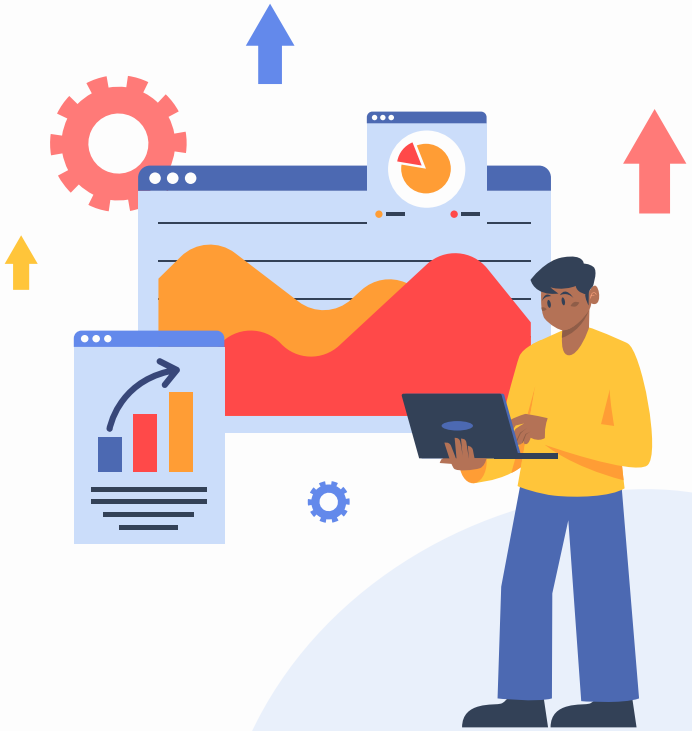
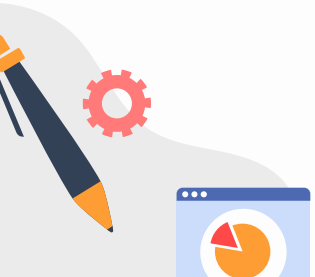


Table of contents



01. Data analysis

Video Game Data

Rank, Name, Platform, Year, Genre, Publisher, NA_Sales, EU Sales, JP_Sales, Other_Sales

# Rank	A Name	A Platform	A Year	A Genre	A Publisher	# NA_Sales	# EU_Sales	# JP_Sales	# Other_Sales
1	Wii Sports	Wii	2006	Sports	Nintendo	41.49	29.02	3.77	8.46
2	Super Mario Bros.	NES	1985	Platform	Nintendo	29.08	3.58	6.81	0.77
3	Mario Kart Wii	Wii	2008	Racing	Nintendo	15.85	12.88	3.79	3.31
4	Wii Sports Resort	Wii	2009	Sports	Nintendo	15.75	11.01	3.28	2.96

01. Data analysis

```
import matplotlib.pyplot as plt
import seaborn as sns
import pandas as pd
from google.colab import drive
```

```
[ ] drive.mount('/content/drive')
    file_path= '/content/drive/MyDrive/vgsales.csv'
    df= pd.read_csv(file_path)
    print(df.head())
```

1. Import Libraries & Mount Google Drive and Load CSV

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

```
[ ] df.fillna("Unknown", inplace=True)
```

2. Check for Missing Values & Fill Missing Values with "Unknown"

01. Data analysis

```
[ ] popular_globalsales = df[df['Global_Sales'] > 30]
    print(popular_globalsales)
```

```
    Rank                     Name Platform    Year         Genre Publisher  \
0     1                Wii Sports      Wii  2006.0        Sports  Nintendo
1     2          Super Mario Bros.      NES  1985.0      Platform  Nintendo
2     3             Mario Kart Wii      Wii  2008.0        Racing  Nintendo
3     4          Wii Sports Resort      Wii  2009.0        Sports  Nintendo
4     5  Pokemon Red/Pokemon Blue       GB  1996.0  Role-Playing  Nintendo
5     6                    Tetris       GB  1989.0        Puzzle  Nintendo
6     7     New Super Mario Bros.       DS  2006.0      Platform  Nintendo

   NA_Sales  EU_Sales  JP_Sales  Other_Sales  Global_Sales
0     41.49     29.02      3.77         8.46         82.74
1     29.08      3.58      6.81         0.77         40.24
2     15.85     12.88      3.79         3.31         35.82
3     15.75     11.01      3.28         2.96         33.00
4     11.27      8.89     10.22         1.00         31.37
5     23.20      2.26      4.22         0.58         30.26
6     11.38      9.23      6.50         2.90         30.01
```

Which games have global sales of over 30 million?

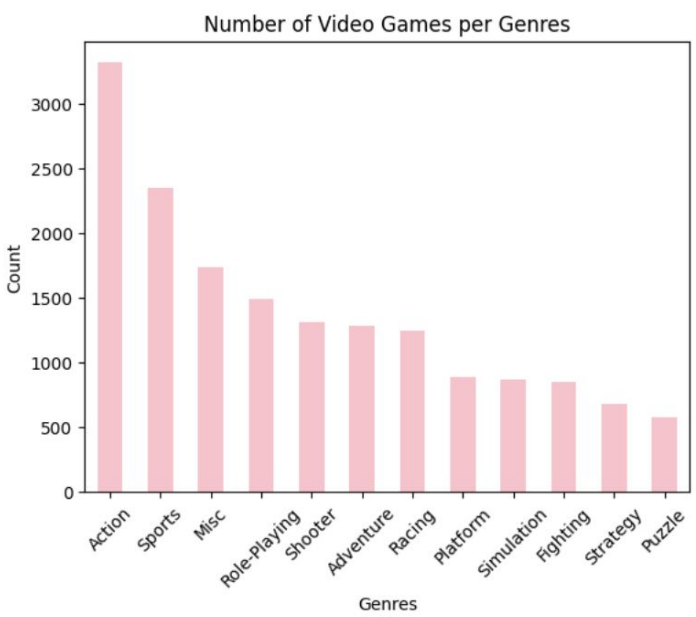
```
[ ] max_globalsales = df[df['Global_Sales'] == df['Global_Sales'].max()]
    print(max_globalsales)
```

```
   Rank       Name Platform    Year   Genre Publisher  NA_Sales  EU_Sales  \
0     1  Wii Sports      Wii  2006.0  Sports  Nintendo     41.49     29.02

   JP_Sales  Other_Sales  Global_Sales
0      3.77         8.46         82.74
```

Which game has the highest global sales value?

01. Data analysis: Bar Graph

```
[ ] df["Genre"].value_counts().plot(kind="bar", color="pink")
    plt.title("Number of Video Games per Genres")
    plt.xlabel("Genres")
    plt.ylabel("Count")
    plt.xticks(rotation=45)

    plt.show()
```

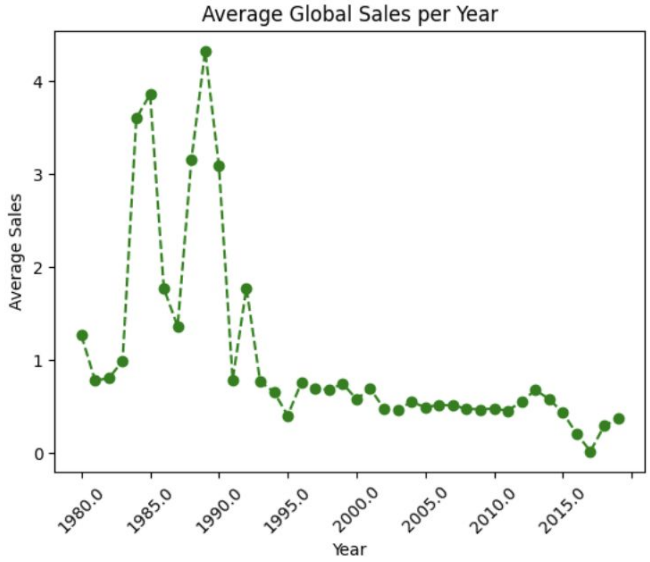


Conclusion

- Action is the most popular genre, with the highest number of video game counts.

01. Data analysis: Line Graph

```
average_sales_by_year = df.groupby('Year')['Global_Sales'].mean()
average_sales_by_year.plot(kind="line", marker="o", color="green", linestyle="--")
plt.title("Average Global Sales per Year")
plt.xlabel("Year")
plt.ylabel("Average Sales")
plt.xticks(rotation=45)
plt.show()
```

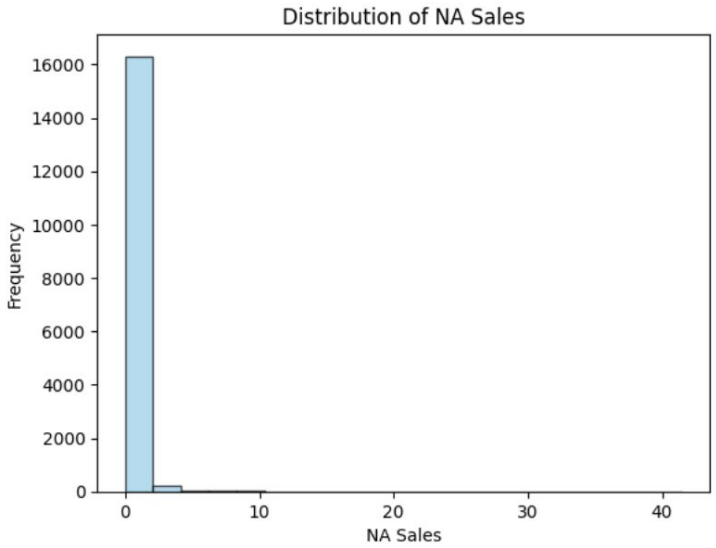


Conclusion

- The late 1980s and early 1990s experienced the highest average global sales.
- After the peak, the average sales per year decrease sharply.

01. Data analysis: Histogram

```
[ ] plt.hist(df["NA_Sales"], bins=20, color="skyblue", edgecolor="black", alpha= 0.7)
    plt.title("Distribution of NA Sales")
    plt.xlabel("NA Sales")
    plt.ylabel("Frequency")
    plt.show()
```



Conclusion

- Most video games in North America had very low sales. Only a few games sold in large numbers.
- The sales distribution is skewed to the left. This means popular games are rare, while low-selling games are common.

02. CNN Model Training

Sign Language Data

02. CNN Model Training

```python
#importing libraries
import tensorflow as tf
import matplotlib.pyplot as plt

from tensorflow import keras
from google.colab import drive
from tensorflow.keras.datasets import mnist
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Flatten, Conv2D, MaxPooling2D, Dropout
import pandas as pd
import numpy as np
```

1. Import Libraries

```python
# Load CSVs
train_data = pd.read_csv('sign_mnist_train.csv')
test_data = pd.read_csv('sign_mnist_test.csv')

# Extract labels and features
y_train = train_data['label'].values
y_test = test_data['label'].values

# Convert to NumPy arrays, normalize, and reshape
x_train = train_data.drop('label', axis=1).values.astype('float32') / 255.0
x_test = test_data.drop('label', axis=1).values.astype('float32') / 255.0

#Reshape
x_train = x_train.reshape(-1, 28, 28, 1)
x_test = x_test.reshape(-1, 28, 28, 1)
```

2.

- Load CSVs
- Extract Labels (y) and Features (x)
- Convert Features to NumPy Arrays and Normalize
- Reshape the Input to (28, 28, 1)

02. CNN Model Training

```python
#Create our model
model = Sequential([
    Conv2D(64, (3, 3), activation='relu', input_shape=(28, 28, 1)),
    MaxPooling2D((2, 2)),
    Conv2D(64, (3, 3), activation='relu'),
    MaxPooling2D((2, 2)),

    Flatten(),
    Dense(128, activation='relu'),
    Dense(16, activation='sigmoid'),
    Dropout(0.5),
    Dense(25, activation='softmax')
])
```

- **Conv2D:** extract features
- **MaxPooling2D:** reduces the dimensionality of the data
- **Flatten:** Converts 3D features to 1D
- **DropOut:** Drops neurons to reduce overfitting

02. CNN Model Training

```
#Complie the model
model.compile(optimizer='adam', #Adam, SGD are two optimizer
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])
#Train the model
history = model.fit(x_train, y_train, epochs=20, validation_data=(x_test, y_test))
```

```
Epoch 1/20
858/858 ━━━━━━━━━━━━━━━━━━━━ 45s 50ms/step - accuracy: 0.0673 - loss: 3.1302 - val_accuracy: 0.4099 - val_loss: 2.1980
Epoch 2/20
858/858 ━━━━━━━━━━━━━━━━━━━━ 80s 48ms/step - accuracy: 0.3516 - loss: 2.0730 - val_accuracy: 0.6870 - val_loss: 1.3986
Epoch 3/20
858/858 ━━━━━━━━━━━━━━━━━━━━ 83s 50ms/step - accuracy: 0.5289 - loss: 1.4832 - val_accuracy: 0.7567 - val_loss: 1.0145
Epoch 4/20
858/858 ━━━━━━━━━━━━━━━━━━━━ 80s 48ms/step - accuracy: 0.6109 - loss: 1.1800 - val_accuracy: 0.8066 - val_loss: 0.7934
Epoch 5/20
858/858 ━━━━━━━━━━━━━━━━━━━━ 84s 50ms/step - accuracy: 0.6450 - loss: 1.0439 - val_accuracy: 0.8327 - val_loss: 0.7005
Epoch 6/20
858/858 ━━━━━━━━━━━━━━━━━━━━ 85s 54ms/step - accuracy: 0.6611 - loss: 0.9569 - val_accuracy: 0.8419 - val_loss: 0.6247
Epoch 7/20
858/858 ━━━━━━━━━━━━━━━━━━━━ 79s 51ms/step - accuracy: 0.6573 - loss: 0.9260 - val_accuracy: 0.8597 - val_loss: 0.5741
Epoch 8/20
858/858 ━━━━━━━━━━━━━━━━━━━━ 82s 51ms/step - accuracy: 0.6682 - loss: 0.8875 - val_accuracy: 0.8696 - val_loss: 0.5564
Epoch 9/20
858/858 ━━━━━━━━━━━━━━━━━━━━ 80s 49ms/step - accuracy: 0.6690 - loss: 0.8738 - val_accuracy: 0.8858 - val_loss: 0.5142
Epoch 10/20
858/858 ━━━━━━━━━━━━━━━━━━━━ 83s 50ms/step - accuracy: 0.6752 - loss: 0.8492 - val_accuracy: 0.8885 - val_loss: 0.5365
Epoch 11/20
858/858 ━━━━━━━━━━━━━━━━━━━━ 45s 53ms/step - accuracy: 0.6789 - loss: 0.8338 - val_accuracy: 0.8857 - val_loss: 0.4954
Epoch 12/20
858/858 ━━━━━━━━━━━━━━━━━━━━ 79s 49ms/step - accuracy: 0.6903 - loss: 0.8041 - val_accuracy: 0.8963 - val_loss: 0.4728
Epoch 13/20
858/858 ━━━━━━━━━━━━━━━━━━━━ 81s 48ms/step - accuracy: 0.6772 - loss: 0.8224 - val_accuracy: 0.8682 - val_loss: 0.5718
Epoch 14/20
858/858 ━━━━━━━━━━━━━━━━━━━━ 83s 49ms/step - accuracy: 0.6824 - loss: 0.8115 - val_accuracy: 0.8749 - val_loss: 0.5457
Epoch 15/20
858/858 ━━━━━━━━━━━━━━━━━━━━ 82s 50ms/step - accuracy: 0.6892 - loss: 0.7961 - val_accuracy: 0.8747 - val_loss: 0.5189
Epoch 16/20
858/858 ━━━━━━━━━━━━━━━━━━━━ 80s 48ms/step - accuracy: 0.6845 - loss: 0.8049 - val_accuracy: 0.8879 - val_loss: 0.4954
Epoch 17/20
858/858 ━━━━━━━━━━━━━━━━━━━━ 81s 47ms/step - accuracy: 0.6803 - loss: 0.8077 - val_accuracy: 0.8848 - val_loss: 0.5187
Epoch 18/20
858/858 ━━━━━━━━━━━━━━━━━━━━ 41s 48ms/step - accuracy: 0.6830 - loss: 0.8025 - val_accuracy: 0.9052 - val_loss: 0.4763
Epoch 19/20
858/858 ━━━━━━━━━━━━━━━━━━━━ 82s 48ms/step - accuracy: 0.6837 - loss: 0.7986 - val_accuracy: 0.8473 - val_loss: 0.7769
Epoch 20/20
858/858 ━━━━━━━━━━━━━━━━━━━━ 41s 47ms/step - accuracy: 0.6809 - loss: 0.8103 - val_accuracy: 0.8797 - val_loss: 0.5799
```

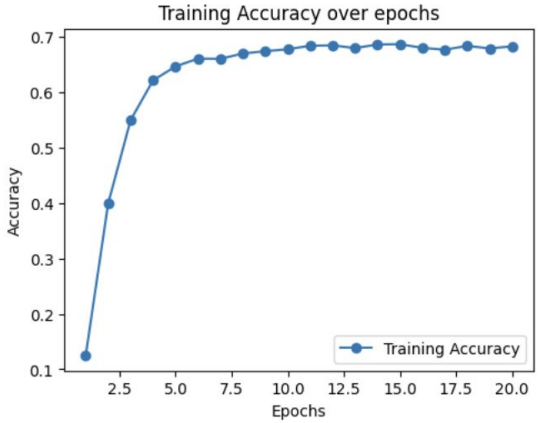
Epoch: one complete pass of the entire training dataset through the network during the learning process

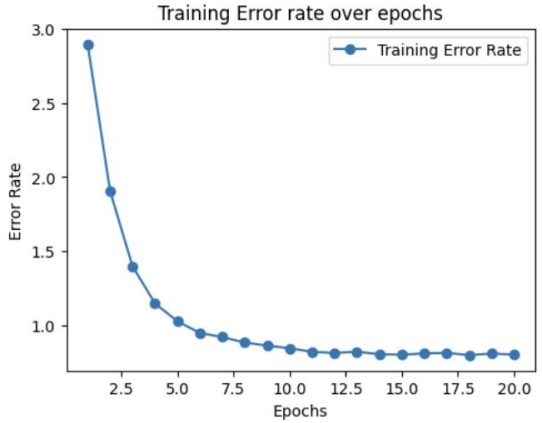
With increasing epochs, training and test accuracy increase, while training and test loss decrease. After epoch 12, accuracy and loss plateau and remain stable.

02. CNN Model Training

```python
#Evaluation
test_loss, test_accuracy = model.evaluate(x_test, y_test)
error_rate = 1 - test_accuracy
print(f'Test loss: {test_loss}')
print(f'Test accuracy: {test_accuracy}')
print(f'Error Rate: {error_rate}')

#Accuracy and Error rate visualization
epochs_range = range(1, 21)
plt.figure(figsize=(12, 4))

#Accuracy Visualization
plt.subplot(1, 2, 1)
plt.plot(epochs_range, history.history['accuracy'], label='Training Accuracy', marker ='o')
plt.title('Training Accuracy over epochs')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()

#Error Rate Visualization
plt.subplot(1, 2, 2)
plt.plot(epochs_range, history.history['loss'], label='Training Error Rate', marker='o')
plt.title('Training Error rate over epochs')
plt.xlabel('Epochs')
plt.ylabel('Error Rate')
plt.legend()

plt.show()
```

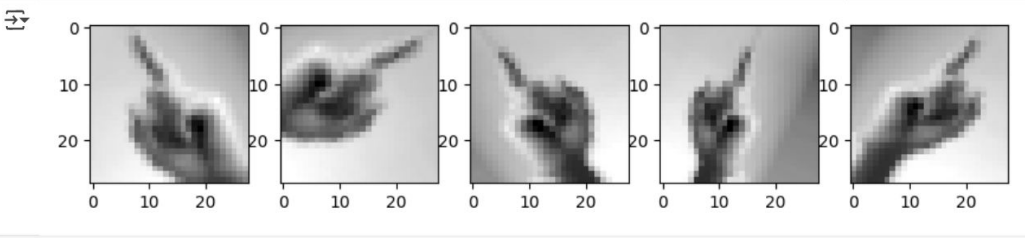
```
225/225 ━━━━━━━━━━━━━━━━━━━━ 4s 17ms/step - accuracy: 0.8783 - loss: 0.5863
Test loss: 0.5798971056938171
Test accuracy: 0.8796709179878235
Error Rate: 0.12032908201217651
```





02. CNN Model Training

```python
#Data augmentation to help underfitting
from tensorflow.keras.preprocessing.image import ImageDataGenerator

datagen = ImageDataGenerator(
    rotation_range=60,
    width_shift_range=0.1,
    height_shift_range=0.1,
    shear_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True,
)


#Apply to the images
Sample_img = x_train[0].reshape(1, 28, 28, 1)
Augmented_images = [datagen.random_transform(Sample_img[0]) for _ in range (5)]

#Plot augmented images
plt.figure(figsize=(10, 5))
for i in range(5):
    plt.subplot(1, 5, i+1)
    plt.imshow(Augmented_images[i].reshape(28, 28), cmap='gray')
```



- Data augmentation generates variations of a single image to improve model generalization which helps reduce underfitting.
- Apply random transformations such as rotation, shifting, zooming to an image from the dataset.

03. Reflection

Fun part

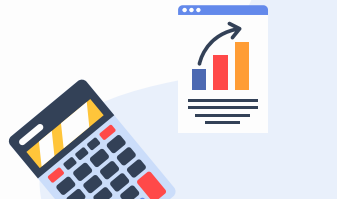
Data analysis

Challenging part

Understanding how CNN works

Improvement

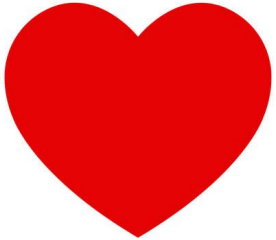
Increasing the number of layers in a CNN for deeper learning



03. Reflection

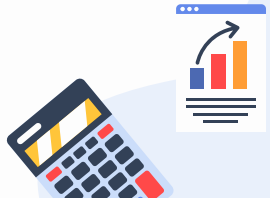
I could have a deeper understanding of...

- Different uses of dataset
- How machine learning actually works



I appreciated...

- Coach Ella's clear explanation
- Comfortable mood for asking questions



Thank You