# Project for Course 7: Historical Structures Classification

## By John Hamilton

(This is a Jupyter Notebook that has been converted to Word Doc.)

## PART 1 - Build deep learning model for Image Classification

### Import the required libraries

import matplotlib.pyplot as plt  
import numpy as np  
import tensorflow as tf  
  
from tensorflow import keras  
from tensorflow.keras import layers  
from tensorflow.keras.models import Sequential  
from keras.applications.nasnet import NASNetLarge  
from keras.callbacks import EarlyStopping  
from tensorflow.keras.preprocessing.image import ImageDataGenerator  
batch\_size = 512  
img\_height = 128  
img\_width = 128

### Make sure there are no invalid image files in the data that need to be deleted

from pathlib import Path  
import imghdr  
  
image\_extensions = [".jpg"]  
img\_type\_accepted\_by\_tf = ["bmp", "gif", "jpeg", "png"]  
  
def check\_image\_files(data\_dir):  
 for filepath in Path(data\_dir).rglob("\*"):  
 if filepath.suffix.lower() in image\_extensions:  
 img\_type = imghdr.what(filepath)  
 if img\_type is None:  
 print(f"{filepath} is not an image")  
 elif img\_type not in img\_type\_accepted\_by\_tf:  
 print(f"{filepath} is a {img\_type}, not accepted by TensorFlow")  
  
data\_dir = "./Stuctures\_Dataset"  
check\_image\_files(data\_dir)  
data\_dir = "./Dataset\_test\_original\_1478"  
check\_image\_files(data\_dir)

C:\Users\jbham\AppData\Local\Temp\ipykernel\_17548\2293567428.py:2: DeprecationWarning: 'imghdr' is deprecated and slated for removal in Python 3.13  
 import imghdr

### Check the integrity of the image files and report any bad files so we can delete them

from struct import unpack  
from tqdm import tqdm  
import os  
  
  
marker\_mapping = {  
 0xffd8: "Start of Image",  
 0xffe0: "Application Default Header",  
 0xffdb: "Quantization Table",  
 0xffc0: "Start of Frame",  
 0xffc4: "Define Huffman Table",  
 0xffda: "Start of Scan",  
 0xffd9: "End of Image"  
}  
  
  
class JPEG:  
 def \_\_init\_\_(self, image\_file):  
 with open(image\_file, 'rb') as f:  
 self.img\_data = f.read()  
   
 def decode(self):  
 data = self.img\_data  
 while(True):  
 marker, = unpack(">H", data[0:2])  
 # print(marker\_mapping.get(marker))  
 if marker == 0xffd8:  
 data = data[2:]  
 elif marker == 0xffd9:  
 return  
 elif marker == 0xffda:  
 data = data[-2:]  
 else:  
 lenchunk, = unpack(">H", data[2:4])  
 data = data[2+lenchunk:]   
 if len(data)==0:  
 break   
  
  
bads = []  
  
for filepath in Path(data\_dir).rglob("\*/\*"):  
 image = JPEG(filepath)   
 try:  
 image.decode()   
 except:  
 bads.append(filepath)  
  
print(bads)

[]

### Take a look at some of the images

import os  
import matplotlib.image as mpimg  
  
def display\_image\_files(data\_dir):  
 for dir in Path(data\_dir).iterdir():  
 print (f'Class: {dir}')  
 fig = plt.figure(figsize=(10, 10))   
 count = 0  
 for filename in os.listdir(dir):  
 count += 1  
 if (count <= 3):  
 image = mpimg.imread(f'{dir}\\{filename}')  
 fig.add\_subplot(1, 3, count)  
 plt.imshow(image)  
 plt.axis('off')   
 plt.show()  
  
display\_image\_files('Stuctures\_Dataset')   
display\_image\_files('Dataset\_test\_original\_1478')

Class: Stuctures\_Dataset\altar



Class: Stuctures\_Dataset\apse



Class: Stuctures\_Dataset\bell\_tower



Class: Stuctures\_Dataset\column



Class: Stuctures\_Dataset\dome(inner)



Class: Stuctures\_Dataset\dome(outer)



Class: Stuctures\_Dataset\flying\_buttress



Class: Stuctures\_Dataset\gargoyle



Class: Stuctures\_Dataset\portal



Class: Stuctures\_Dataset\stained\_glass



Class: Stuctures\_Dataset\vault



Class: Dataset\_test\_original\_1478\altar



Class: Dataset\_test\_original\_1478\apse



Class: Dataset\_test\_original\_1478\bell\_tower



Class: Dataset\_test\_original\_1478\column



Class: Dataset\_test\_original\_1478\dome(inner)



Class: Dataset\_test\_original\_1478\dome(outer)



Class: Dataset\_test\_original\_1478\flying\_buttress



Class: Dataset\_test\_original\_1478\gargoyle



Class: Dataset\_test\_original\_1478\portal

<Figure size 1000x1000 with 0 Axes>

Class: Dataset\_test\_original\_1478\stained\_glass



Class: Dataset\_test\_original\_1478\vault



### Load classified images from disk into Train and Test datasets

train\_ds = tf.keras.utils.image\_dataset\_from\_directory(  
 'Stuctures\_dataset',  
 shuffle=True,  
 seed=42,  
 image\_size=(img\_height, img\_width),  
 batch\_size=batch\_size  
 )  
  
test\_ds = tf.keras.utils.image\_dataset\_from\_directory(  
 'Dataset\_test\_original\_1478',  
 shuffle=True,  
 seed=42,  
 image\_size=(img\_height, img\_width),  
 batch\_size=batch\_size  
 )  
  
class\_names = train\_ds.class\_names  
num\_classes = len(class\_names)

Found 9977 files belonging to 11 classes.  
Found 1439 files belonging to 11 classes.

### Adjust datasets for use in the CNN:

1. Apply normalization to images (x)
2. Change labels to categorical (y)

normalization\_layer = layers.Rescaling(1./255)  
  
train\_ds = train\_ds.map(lambda x, y: (normalization\_layer(x), keras.utils.to\_categorical(y, num\_classes=num\_classes)))  
test\_ds = test\_ds.map(lambda x, y: (normalization\_layer(x), keras.utils.to\_categorical(y, num\_classes=num\_classes)))

### Create the Convolutional Neural Network by specifying its layers

model = Sequential([  
 NASNetLarge(include\_top=False, input\_shape=(img\_height, img\_width, 3), weights='imagenet'),  
 layers.GlobalAveragePooling2D(),  
 layers.Dropout(0.1),  
 layers.Dense(256, activation='relu'),  
 layers.Dropout(0.1),  
 layers.Dense(128, activation='relu'),  
 layers.Dropout(0.1),  
 layers.Dense(num\_classes, activation='softmax')  
])

### Compile the CNN

model.compile(optimizer='adam',  
 loss='categorical\_crossentropy',  
 metrics=['accuracy','precision','recall'])

### Create Early Stopping callback to be passed into model.fit()

early\_stopping = EarlyStopping(monitor='val\_loss', patience=3, verbose=1)

### Train the model on the training dataset and test using the test dataset (without augmentation)

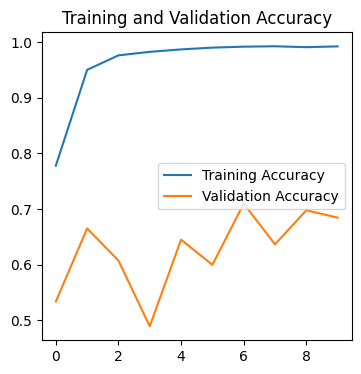
epochs=50  
history = model.fit(  
 train\_ds,  
 validation\_data=test\_ds,  
 epochs=epochs,  
 callbacks=[early\_stopping]   
)

Epoch 1/50  
20/20 ━━━━━━━━━━━━━━━━━━━━ 626s 27s/step - accuracy: 0.5992 - loss: 1.2894 - precision: 0.8813 - recall: 0.4594 - val\_accuracy: 0.5337 - val\_loss: 37.0114 - val\_precision: 0.5341 - val\_recall: 0.5337  
Epoch 2/50  
20/20 ━━━━━━━━━━━━━━━━━━━━ 557s 28s/step - accuracy: 0.9494 - loss: 0.2360 - precision: 0.9695 - recall: 0.9170 - val\_accuracy: 0.6650 - val\_loss: 22.2792 - val\_precision: 0.6660 - val\_recall: 0.6650  
Epoch 3/50  
20/20 ━━━━━━━━━━━━━━━━━━━━ 555s 28s/step - accuracy: 0.9756 - loss: 0.1024 - precision: 0.9833 - recall: 0.9684 - val\_accuracy: 0.6074 - val\_loss: 18.3247 - val\_precision: 0.6075 - val\_recall: 0.6067  
Epoch 4/50  
20/20 ━━━━━━━━━━━━━━━━━━━━ 563s 28s/step - accuracy: 0.9810 - loss: 0.0719 - precision: 0.9853 - recall: 0.9766 - val\_accuracy: 0.4892 - val\_loss: 64.3581 - val\_precision: 0.4892 - val\_recall: 0.4892  
Epoch 5/50  
20/20 ━━━━━━━━━━━━━━━━━━━━ 561s 28s/step - accuracy: 0.9868 - loss: 0.0652 - precision: 0.9896 - recall: 0.9822 - val\_accuracy: 0.6449 - val\_loss: 18.1541 - val\_precision: 0.6458 - val\_recall: 0.6449  
Epoch 6/50  
20/20 ━━━━━━━━━━━━━━━━━━━━ 556s 28s/step - accuracy: 0.9901 - loss: 0.0380 - precision: 0.9915 - recall: 0.9878 - val\_accuracy: 0.5997 - val\_loss: 37.5581 - val\_precision: 0.6001 - val\_recall: 0.5997  
Epoch 7/50  
20/20 ━━━━━━━━━━━━━━━━━━━━ 556s 28s/step - accuracy: 0.9929 - loss: 0.0340 - precision: 0.9945 - recall: 0.9913 - val\_accuracy: 0.7095 - val\_loss: 14.4961 - val\_precision: 0.7095 - val\_recall: 0.7095  
Epoch 8/50  
20/20 ━━━━━━━━━━━━━━━━━━━━ 555s 28s/step - accuracy: 0.9923 - loss: 0.0317 - precision: 0.9937 - recall: 0.9905 - val\_accuracy: 0.6366 - val\_loss: 22.4935 - val\_precision: 0.6374 - val\_recall: 0.6366  
Epoch 9/50  
20/20 ━━━━━━━━━━━━━━━━━━━━ 557s 28s/step - accuracy: 0.9925 - loss: 0.0294 - precision: 0.9938 - recall: 0.9911 - val\_accuracy: 0.6977 - val\_loss: 19.2085 - val\_precision: 0.6977 - val\_recall: 0.6977  
Epoch 10/50  
20/20 ━━━━━━━━━━━━━━━━━━━━ 556s 28s/step - accuracy: 0.9932 - loss: 0.0315 - precision: 0.9943 - recall: 0.9910 - val\_accuracy: 0.6845 - val\_loss: 26.5311 - val\_precision: 0.6855 - val\_recall: 0.6845  
Epoch 10: early stopping

### Plot Training vs Validation accuracy

epochs = early\_stopping.stopped\_epoch+1  
  
acc = history.history['accuracy']  
val\_acc = history.history['val\_accuracy']  
  
loss = history.history['loss']  
val\_loss = history.history['val\_loss']  
  
epochs\_range = range(epochs)  
  
plt.figure(figsize=(4, 4))  
plt.plot(epochs\_range, acc, label='Training Accuracy')  
plt.plot(epochs\_range, val\_acc, label='Validation Accuracy')  
plt.legend(loc='right')  
plt.title('Training and Validation Accuracy')

Text(0.5, 1.0, 'Training and Validation Accuracy')



### Apply augmentation and rescale the data

train\_aug = ImageDataGenerator (rescale = 1./255,shear\_range = 0.3,zoom\_range = 0.7,horizontal\_flip = True)  
test\_aug = ImageDataGenerator (rescale=1./255)  
  
train\_ds = train\_aug.flow\_from\_directory(  
 "Stuctures\_Dataset",   
 seed=42,  
 target\_size=(img\_height, img\_width),  
 batch\_size=batch\_size,  
 class\_mode='categorical',  
 shuffle=True  
)  
  
test\_ds = test\_aug.flow\_from\_directory(  
 "Dataset\_test\_original\_1478",  
 seed=42,  
 target\_size=(img\_height, img\_width),  
 batch\_size=batch\_size,  
 class\_mode='categorical',  
 shuffle=True  
)

Found 9977 images belonging to 11 classes.  
Found 1439 images belonging to 11 classes.

### Recreate the model and refit

augmented\_model = Sequential([  
 NASNetLarge(include\_top=False, input\_shape=(img\_height, img\_width, 3), weights='imagenet'),  
 layers.GlobalAveragePooling2D(),  
 layers.Dropout(0.1),  
 layers.Dense(256, activation='relu'),  
 layers.Dropout(0.1),  
 layers.Dense(128, activation='relu'),  
 layers.Dropout(0.1),  
 layers.Dense(num\_classes, activation='softmax')  
])  
  
augmented\_model.compile(optimizer='adam',  
 loss='categorical\_crossentropy',  
 metrics=['accuracy','precision','recall'])  
  
early\_stopping2 = EarlyStopping(monitor='val\_loss', patience=3, verbose=1)  
  
epochs=50  
history2 = augmented\_model.fit(  
 train\_ds,  
 validation\_data=test\_ds,  
 epochs=epochs,  
 callbacks=[early\_stopping2]   
)

Epoch 1/50

c:\Users\jbham\AppData\Local\Programs\Python\Python312\Lib\site-packages\keras\src\trainers\data\_adapters\py\_dataset\_adapter.py:120: UserWarning: Your `PyDataset` class should call `super().\_\_init\_\_(\*\*kwargs)` in its constructor. `\*\*kwargs` can include `workers`, `use\_multiprocessing`, `max\_queue\_size`. Do not pass these arguments to `fit()`, as they will be ignored.  
 self.\_warn\_if\_super\_not\_called()

20/20 ━━━━━━━━━━━━━━━━━━━━ 640s 28s/step - accuracy: 0.5768 - loss: 1.3723 - precision: 0.8657 - recall: 0.4276 - val\_accuracy: 0.4892 - val\_loss: 25.5842 - val\_precision: 0.4892 - val\_recall: 0.4892  
Epoch 2/50  
20/20 ━━━━━━━━━━━━━━━━━━━━ 586s 29s/step - accuracy: 0.8963 - loss: 0.3803 - precision: 0.9358 - recall: 0.8691 - val\_accuracy: 0.4190 - val\_loss: 40.8873 - val\_precision: 0.4189 - val\_recall: 0.4183  
Epoch 3/50  
20/20 ━━━━━━━━━━━━━━━━━━━━ 633s 32s/step - accuracy: 0.9312 - loss: 0.2549 - precision: 0.9560 - recall: 0.9070 - val\_accuracy: 0.6553 - val\_loss: 14.5492 - val\_precision: 0.6564 - val\_recall: 0.6546  
Epoch 4/50  
20/20 ━━━━━━━━━━━━━━━━━━━━ 586s 29s/step - accuracy: 0.9427 - loss: 0.2164 - precision: 0.9555 - recall: 0.9298 - val\_accuracy: 0.5254 - val\_loss: 37.4650 - val\_precision: 0.5265 - val\_recall: 0.5254  
Epoch 5/50  
20/20 ━━━━━━━━━━━━━━━━━━━━ 572s 28s/step - accuracy: 0.9538 - loss: 0.1685 - precision: 0.9644 - recall: 0.9393 - val\_accuracy: 0.6650 - val\_loss: 12.4769 - val\_precision: 0.6674 - val\_recall: 0.6650  
Epoch 6/50  
20/20 ━━━━━━━━━━━━━━━━━━━━ 571s 28s/step - accuracy: 0.9571 - loss: 0.1452 - precision: 0.9665 - recall: 0.9487 - val\_accuracy: 0.5129 - val\_loss: 32.6393 - val\_precision: 0.5146 - val\_recall: 0.5129  
Epoch 7/50  
20/20 ━━━━━━━━━━━━━━━━━━━━ 570s 28s/step - accuracy: 0.9560 - loss: 0.1535 - precision: 0.9676 - recall: 0.9466 - val\_accuracy: 0.6095 - val\_loss: 21.4846 - val\_precision: 0.6099 - val\_recall: 0.6095  
Epoch 8/50  
20/20 ━━━━━━━━━━━━━━━━━━━━ 570s 28s/step - accuracy: 0.9627 - loss: 0.1377 - precision: 0.9701 - recall: 0.9574 - val\_accuracy: 0.5830 - val\_loss: 26.7260 - val\_precision: 0.5830 - val\_recall: 0.5830  
Epoch 8: early stopping

### Plot Training vs Validation accuracy

epochs = early\_stopping2.stopped\_epoch+1  
  
acc = history2.history['accuracy']  
val\_acc = history2.history['val\_accuracy']  
  
loss = history2.history['loss']  
val\_loss = history2.history['val\_loss']  
  
epochs\_range = range(epochs)  
  
plt.figure(figsize=(4, 4))  
plt.plot(epochs\_range, acc, label='Training Accuracy')  
plt.plot(epochs\_range, val\_acc, label='Validation Accuracy')  
plt.legend(loc='right')  
plt.title('Training and Validation Accuracy')

Text(0.5, 1.0, 'Training and Validation Accuracy')



### Observations

1. On both unaugmented and augmented models, the accuracy improved overall but spent a lot of time moving up and down
2. Even though we added more images via augmentation, the augmented model did not do better than the unaugmented one

## PART 2 - Exploratory Data Analysis and Recommender Model

### First we load the users dataset and take a quick look at the head of the data

import numpy as np  
import pandas as pd  
  
df\_users = pd.read\_csv("user.csv")  
df\_users.head()

User\_Id Location Age  
0 1 Semarang, Jawa Tengah 20  
1 2 Bekasi, Jawa Barat 21  
2 3 Cirebon, Jawa Barat 23  
3 4 Bekasi, Jawa Barat 21  
4 5 Lampung, Sumatera Selatan 20

### Check the info for the dataset

df\_users.info()

<class 'pandas.core.frame.DataFrame'>  
RangeIndex: 300 entries, 0 to 299  
Data columns (total 3 columns):  
 # Column Non-Null Count Dtype   
--- ------ -------------- -----   
 0 User\_Id 300 non-null int64   
 1 Location 300 non-null object  
 2 Age 300 non-null int64   
dtypes: int64(2), object(1)  
memory usage: 7.2+ KB

### Convert column to category and check for zeros

df\_users['Location'] = pd.Categorical(df\_users['Location'])  
  
print('Zeros:\n', df\_users[df\_users==0].count(), sep='')

Zeros:  
User\_Id 0  
Location 0  
Age 0  
dtype: int64

### Now load the ratings dataset and take a quick look at the head of the data

df\_ratings = pd.read\_csv("tourism\_rating.csv")  
df\_ratings.head()

User\_Id Place\_Id Place\_Ratings  
0 1 179 3  
1 1 344 2  
2 1 5 5  
3 1 373 3  
4 1 101 4

### Check the info for the dataset

df\_ratings.info()

<class 'pandas.core.frame.DataFrame'>  
RangeIndex: 10000 entries, 0 to 9999  
Data columns (total 3 columns):  
 # Column Non-Null Count Dtype  
--- ------ -------------- -----  
 0 User\_Id 10000 non-null int64  
 1 Place\_Id 10000 non-null int64  
 2 Place\_Ratings 10000 non-null int64  
dtypes: int64(3)  
memory usage: 234.5 KB

### Check for zeros

print('Zeros:\n', df\_ratings[df\_ratings==0].count(), sep='')

Zeros:  
User\_Id 0  
Place\_Id 0  
Place\_Ratings 0  
dtype: int64

### Now load the locations dataset and look at the head

df\_locations = pd.read\_excel("tourism\_with\_id.xlsx")  
df\_locations.head()

Place\_Id Place\_Name \  
0 1 Monumen Nasional   
1 2 Kota Tua   
2 3 Dunia Fantasi   
3 4 Taman Mini Indonesia Indah (TMII)   
4 5 Atlantis Water Adventure   
  
 Description Category City \  
0 Monumen Nasional atau yang populer disingkat d... Budaya Jakarta   
1 Kota tua di Jakarta, yang juga bernama Kota Tu... Budaya Jakarta   
2 Dunia Fantasi atau disebut juga Dufan adalah t... Taman Hiburan Jakarta   
3 Taman Mini Indonesia Indah merupakan suatu kaw... Taman Hiburan Jakarta   
4 Atlantis Water Adventure atau dikenal dengan A... Taman Hiburan Jakarta   
  
 Price Rating Time\_Minutes \  
0 20000 4.6 15.0   
1 0 4.6 90.0   
2 270000 4.6 360.0   
3 10000 4.5 NaN   
4 94000 4.5 60.0   
  
 Coordinate Lat Long \  
0 {'lat': -6.1753924, 'lng': 106.8271528} -6.175392 106.827153   
1 {'lat': -6.137644799999999, 'lng': 106.8171245} -6.137645 106.817125   
2 {'lat': -6.125312399999999, 'lng': 106.8335377} -6.125312 106.833538   
3 {'lat': -6.302445899999999, 'lng': 106.8951559} -6.302446 106.895156   
4 {'lat': -6.12419, 'lng': 106.839134} -6.124190 106.839134   
  
 Unnamed: 11 Unnamed: 12   
0 NaN 1   
1 NaN 2   
2 NaN 3   
3 NaN 4   
4 NaN 5

### Check the info for the dataset

df\_locations.info()

<class 'pandas.core.frame.DataFrame'>  
RangeIndex: 437 entries, 0 to 436  
Data columns (total 13 columns):  
 # Column Non-Null Count Dtype   
--- ------ -------------- -----   
 0 Place\_Id 437 non-null int64   
 1 Place\_Name 437 non-null object   
 2 Description 437 non-null object   
 3 Category 437 non-null object   
 4 City 437 non-null object   
 5 Price 437 non-null int64   
 6 Rating 437 non-null float64  
 7 Time\_Minutes 205 non-null float64  
 8 Coordinate 437 non-null object   
 9 Lat 437 non-null float64  
 10 Long 437 non-null float64  
 11 Unnamed: 11 0 non-null float64  
 12 Unnamed: 12 437 non-null int64   
dtypes: float64(5), int64(3), object(5)  
memory usage: 44.5+ KB

### Clean up columns, replace NaN, and count zeros

df\_locations['Category'] = pd.Categorical(df\_locations['Category'])  
df\_locations['City'] = pd.Categorical(df\_locations['City'])  
df\_locations.drop(['Coordinate', 'Unnamed: 11', 'Unnamed: 12'], axis=1, inplace=True)  
df\_locations.fillna({'Time\_Minutes': 0}, inplace=True)  
  
print('Zeros:\n', df\_locations[df\_locations==0].count(), sep='')

Zeros:  
Place\_Id 0  
Place\_Name 0  
Description 0  
Category 0  
City 0  
Price 137  
Rating 0  
Time\_Minutes 232  
Lat 0  
Long 0  
dtype: int64

Zeros are OK for Price and Time\_Minutes

### Check location dataset column stats

df\_locations.describe().T

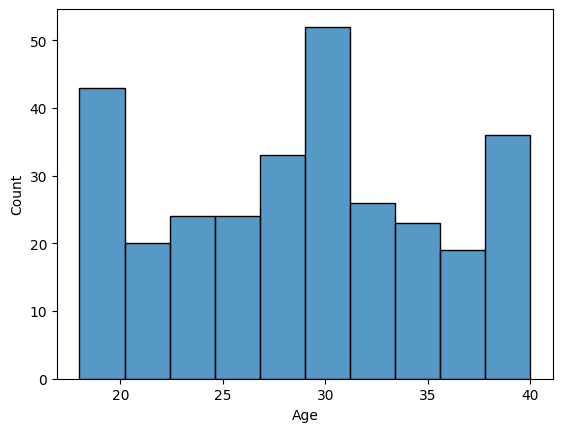
count mean std min 25% \  
Place\_Id 437.0 219.000000 126.295289 1.000000 110.000000   
Price 437.0 24652.173913 66446.374709 0.000000 0.000000   
Rating 437.0 4.442792 0.208587 3.400000 4.300000   
Time\_Minutes 437.0 38.752860 54.876745 0.000000 0.000000   
Lat 437.0 -7.095438 0.727241 -8.197894 -7.749590   
Long 437.0 109.160142 1.962848 103.931398 107.578369   
  
 50% 75% max   
Place\_Id 219.000000 328.000000 437.000000   
Price 5000.000000 20000.000000 900000.000000   
Rating 4.500000 4.600000 5.000000   
Time\_Minutes 0.000000 60.000000 360.000000   
Lat -7.020524 -6.829411 1.078880   
Long 110.237468 110.431869 112.821662

Everything looks good. Price and Time\_Minutes have zeros in the lower quartiles but that is ok I think.

### Analyze the age distribution of the users

import seaborn as sns  
  
sns.histplot(df\_users['Age'])

<Axes: xlabel='Age', ylabel='Count'>



Observations:

1. Most of the users are between 25 and 35
2. More users are 30 than any other age
3. However, there are surges at the youngest and oldest ages

### Where are most of these users coming from?

df\_users.Location.value\_counts()

Location  
Bekasi, Jawa Barat 39  
Semarang, Jawa Tengah 22  
Lampung, Sumatera Selatan 20  
Yogyakarta, DIY 20  
Bogor, Jawa Barat 17  
Cirebon, Jawa Barat 14  
Jakarta Selatan, DKI Jakarta 14  
Subang, Jawa Barat 14  
Depok, Jawa Barat 12  
Ponorogo, Jawa Timur 11  
Jakarta Pusat, DKI Jakarta 10  
Surabaya, Jawa Timur 10  
Jakarta Utara, DKI Jakarta 10  
Sragen, Jawa Tengah 9  
Serang, Banten 9  
Tanggerang, Banten 8  
Bandung, Jawa Barat 8  
Kota Gede, DIY 8  
Karawang, Jawa Barat 8  
Jakarta Timur, DKI Jakarta 6  
Jakarta Barat, DKI Jakarta 6  
Palembang, Sumatera Selatan 5  
Purwakarat, Jawa Barat 4  
Cilacap, Jawa Tengah 4  
Solo, Jawa Tengah 4  
Klaten, Jawa Tengah 4  
Madura, Jawa Timur 2  
Nganjuk, Jawa Timur 2  
Name: count, dtype: int64

### What are the different categories of tourist spots?

df\_locations.Category.value\_counts()

Category  
Taman Hiburan 135  
Budaya 117  
Cagar Alam 106  
Bahari 47  
Tempat Ibadah 17  
Pusat Perbelanjaan 15  
Name: count, dtype: int64

### What kind of tourism each location is most famous or suitable for?

display(df\_locations.groupby('City', observed=False)['Category'].agg(pd.Series.mode))

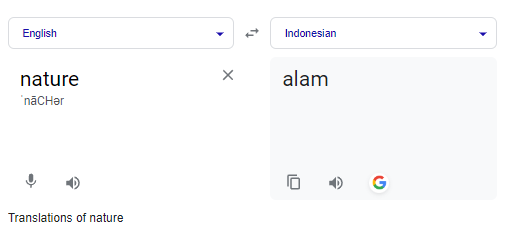
City  
Bandung Cagar Alam  
Jakarta Budaya  
Semarang Cagar Alam  
Surabaya ['Budaya', 'Taman Hiburan']  
Categories (6, obj...  
Yogyakarta Taman Hiburan  
Name: Category, dtype: object

Results:

* Looks like Bandung is mostly Cagar Alam.
* Looks like Jakarta is mostly Budaya.
* Looks like Semarang is mostly Cagar Alam.
* Looks like Surabaya is mostly Budaya and Taman Hiburan.
* Looks like Yogyakarta is mostly Taman Hiburan.

### Which city would be best for a nature enthusiast?

(Let's translate "nature" and search for that word)



df\_locations[df\_locations['Description'].str.contains(' alam ')]['City'].value\_counts()

City  
Yogyakarta 25  
Bandung 23  
Semarang 7  
Jakarta 5  
Surabaya 0  
Name: count, dtype: int64

### What spots are most loved by tourists?

df\_location\_ratings = pd.merge(df\_locations, df\_ratings, left\_on='Place\_Id', right\_on='Place\_Id', how='left')  
most\_loved\_places = df\_location\_ratings.groupby('Place\_Name')['Place\_Ratings'].sum().reset\_index()  
most\_loved\_places.sort\_values('Place\_Ratings', ascending=False)

Place\_Name Place\_Ratings  
86 Gereja Perawan Maria Tak Berdosa Surabaya 125  
158 Keraton Surabaya 123  
272 Pantai Parangtritis 123  
395 Taman Sungai Mudal 120  
261 Pantai Kesirat 115  
.. ... ...  
0 Air Mancur Menari 36  
237 Museum Ullen Sentalu 34  
354 Taman Barunawati 34  
144 Kauman Pakualaman Yogyakarta 33  
279 Pantai Sanglen 31  
  
[437 rows x 2 columns]

### Which category of places are users liking the most?

most\_loved\_categories = df\_location\_ratings.groupby('Category', observed=False)['Place\_Ratings'].sum().reset\_index()  
most\_loved\_categories.sort\_values('Place\_Ratings', ascending=False)

Category Place\_Ratings  
4 Taman Hiburan 9519  
1 Budaya 8142  
2 Cagar Alam 7440  
0 Bahari 3244  
5 Tempat Ibadah 1186  
3 Pusat Perbelanjaan 1134

### Build a recommender model to suggest other places given a current place name

1. Sets up tables using collaborative filtering algorithm
2. Figures out the place's Place\_Id
3. Passes the Place\_Id to the tables
4. Converts resulting Place\_Ids back to Place\_Names
5. Presents results

def find\_most\_similar\_places(place, num=5):  
 pivot = df\_ratings.pivot\_table(index='User\_Id', columns='Place\_Id', values='Place\_Ratings')  
 x = pivot.apply(lambda col: col.fillna(col.mean()), axis=0)  
 x\_corr = x.corr()  
 place\_id = df\_locations[df\_locations['Place\_Name']==place].iloc[0,0]  
 rec\_corr = x\_corr[place\_id]  
 rec\_corr = rec\_corr.sort\_values(ascending=False)  
 rec\_corr.dropna(inplace=True)  
 recommendations = []  
 for row in pd.DataFrame(rec\_corr).iloc[1:num+1,:].iterrows():  
 recommendations.append(df\_locations[df\_locations['Place\_Id']==row[0]].iloc[0,1])  
 display(recommendations)  
  
# Find the top 10 most similar places (based on ratings)  
find\_most\_similar\_places('Monumen Nasional', 10)

['Stone Garden Citatah',  
 'Air Mancur Menari',  
 'Situ Patenggang',  
 'Museum De Javasche Bank',  
 'Pantai Nguluran',  
 'Goa Pindul',  
 'Curug Aseupan',  
 'Happyfarm Ciwidey',  
 'Umbul Sidomukti',  
 'Taman Film']