Preperation

Most of the data we use for the machine learning is in google drive. This cell allows us to use the data on Google Drive

```
import os
from os.path import join
from google.colab import drive
drive.mount("/content/drive", force_remount=True)

# to be used a prefix for file I/O
path_prefix = "/content/drive/My Drive/Colab/DSA 210/Project/Data"
```

```
Mounted at /content/drive
```

This cell lists the files in a specified directory and creates two lists based on these filenames.

exceptions_2015 and exceptions_2014 created here will be used in data processing. The year is last year with data on the datalists.

```
worldbank_fname_list_csv = os.listdir(path_prefix)
worldbank_fname_list = [item[:-4] if item.endswith(".csv") else item for item in  os.listdir(path_prefix)]

exceptions_2015 = ["Renewable electricity output (% of total electricity output).csv",]
exceptions_2014 = ["Electric power consumption (kWh per capita).csv"]
len(os.listdir(path_prefix)), len(worldbank_fname_list_csv), len(worldbank_fname_list)
```

```
(55, 55, 55)
```

Check to see if there is any missing data

```
n = os.listdir(path_prefix)
for item in n:
  if item not in worldbank_fname_list_csv:
    print(item)
```

This code block is used to set up authentication for accessing datasets from the Kaggle platform within the Google Colab environment.

```
from google.colab import userdata
os.environ["KAGGLE_KEY"] = userdata.get('KAGGLE_KEY')
os.environ["KAGGLE_USERNAME"] = userdata.get('KAGGLE_USERNAME')
```

This code block imports various Python libraries and modules that are essential for data manipulation, analysis, visualization, and machine learning tasks.

```
### Basic libraries
import math
import datetime
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline

import pickle

# Data Preprocessing
from scipy.stats import chi2_contingency
from scipy import stats
from sklearn.preprocessing import LabelEncoder


# Model Training
from sklearn.utils import shuffle
from sklearn.model_selection import train_test_split, RandomizedSearchCV
from sklearn.metrics import mean_absolute_error, root_mean_squared_log_error
from sklearn.experimental import enable_halving_search_cv
from sklearn.model_selection import HalvingRandomSearchCV
from sklearn.model_selection import GridSearchCV
from sklearn.ensemble import RandomForestRegressor
from sklearn.linear_model import Ridge
from sklearn.svm import SVR
from sklearn.ensemble import GradientBoostingRegressor
from sklearn.linear_model import Lasso
```

```
from sklearn.linear_model import LinearRegression
import xgboost as xgb
import lightgbm as lgb
import re
from sklearn.tree import DecisionTreeRegressor
```

Gathering Data

These code blocks are used to download datasets from Kaggle and then extract its contents.

```
!kaggle datasets download -d mathurinache/world-happiness-report

!unzip "world-happiness-report.zip"
```

```
Dataset URL: https://www.kaggle.com/datasets/mathurinache/world-happiness-report
License(s): CC0-1.0
Downloading world-happiness-report.zip to /content
  0% 0.00/67.4k [00:00<?, ?B/s]
100% 67.4k/67.4k [00:00<00:00, 178MB/s]
Archive:  world-happiness-report.zip
  inflating: 2015.csv
  inflating: 2016.csv
  inflating: 2017.csv
  inflating: 2018.csv
  inflating: 2019.csv
  inflating: 2020.csv
  inflating: 2021.csv
  inflating: 2022.csv
```

```
!kaggle datasets download -d azathoth42/myanimelist

!unzip "myanimelist.zip"
```

```
Dataset URL: https://www.kaggle.com/datasets/azathoth42/myanimelist
License(s): DbCL-1.0
Downloading myanimelist.zip to /content
100% 1.91G/1.92G [00:14<00:00, 243MB/s]
100% 1.92G/1.92G [00:14<00:00, 147MB/s]
Archive:  myanimelist.zip
  inflating: AnimeList.csv
  inflating: UserAnimeList.csv
  inflating: UserList.csv
  inflating: anime_cleaned.csv
  inflating: anime_filtered.csv
  inflating: animelists_cleaned.csv
  inflating: animelists_filtered.csv
  inflating: users_cleaned.csv
  inflating: users_filtered.csv
```

Data Preprocessing

This code block focuses on loading and cleaning dataset of World Happiness Report, into a pandas DataFrame named df.

```
df = pd.read_csv('2020.csv')
df = df.rename(columns={"Country name": "Country"})
df.sort_values(by='Country', ascending=True, inplace=True)
df = df[df["Country"] != "Kosovo"].copy()
df = df.rename(columns={"Ladder score": "Happiness Score"})
df.reset_index(drop=True, inplace=True)
df['Country'] = df['Country'].str.lower()
df['Country'] = df['Country'].str.strip()
df.head()
```

	Country	Regional indicator	Happiness Score	Standard error of ladder score	upperwhisker	lowerwhisker	Logged GDP per capita	Social support	Healthy life expectancy	Freedom to make life choices	Generosity	P
0	afghanistan	South Asia	2.5669	0.031311	2.628270	2.505530	7.462861	0.470367	52.590000	0.396573	-0.096429	
1	albania	Central and Eastern Europe	4.8827	0.056116	4.992687	4.772713	9.417931	0.671070	68.708138	0.781994	-0.042309	
2	algeria	Middle East and North Africa	5.0051	0.044236	5.091802	4.918397	9.537965	0.803385	65.905174	0.466611	-0.121105	
3	argentina	Latin America and Caribbean	5.9747	0.053442	6.079446	5.869954	9.810955	0.900568	68.803802	0.831132	-0.194914	
4	armenia	Commonwealth of Independent States	4.6768	0.058595	4.791646	4.561953	9.100476	0.757479	66.750656	0.712018	-0.138780	

```
1 df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 152 entries, 0 to 151
Data columns (total 20 columns):
 #   Column                                    Non-Null Count  Dtype
---  ------                                    --------------  -----
 0   Country                                   152 non-null    object
 1   Regional indicator                        152 non-null    object
 2   Happiness Score                           152 non-null    float64
 3   Standard error of ladder score            152 non-null    float64
 4   upperwhisker                              152 non-null    float64
 5   lowerwhisker                              152 non-null    float64
 6   Logged GDP per capita                     152 non-null    float64
 7   Social support                            152 non-null    float64
 8   Healthy life expectancy                   152 non-null    float64
 9   Freedom to make life choices              152 non-null    float64
 10  Generosity                                152 non-null    float64
 11  Perceptions of corruption                 152 non-null    float64
 12  Ladder score in Dystopia                  152 non-null    float64
 13  Explained by: Log GDP per capita          152 non-null    float64
 14  Explained by: Social support              152 non-null    float64
 15  Explained by: Healthy life expectancy     152 non-null    float64
 16  Explained by: Freedom to make life choices  152 non-null  float64
 17  Explained by: Generosity                  152 non-null    float64
 18  Explained by: Perceptions of corruption   152 non-null    float64
 19  Dystopia + residual                       152 non-null    float64
dtypes: float64(18), object(2)
memory usage: 23.9+ KB
```

This code block iterates through a list of World Bank data filenames and merges the relevant data from each file into the main DataFrame, named df. The goal is to enrich the data with various country-level indicators from the World Bank datasets.

```
rows_to_skip = 4
for i in range (len(worldbank_fname_list_csv)):
  fname = worldbank_fname_list_csv[i]
  df_world_bank = pd.read_csv(join(path_prefix, fname), skiprows=rows_to_skip, sep=',', on_bad_lines='skip')

  if 'Country Name' in df_world_bank.columns:
    df_world_bank['Country Name'] = df_world_bank['Country Name'].str.lower().str.strip()
  else:
    print(f"Warning: 'Country Name' column not found in {fname}")
    continue

  df_tmp = df[['Country']].copy()
  if fname in exceptions_2015:
    df_tmp = df_tmp.merge(df_world_bank[['Country Name', '2015']], left_on='Country', right_on='Country Name', how='left')
    df[worldbank_fname_list[i]] = df_tmp['2015']
  elif fname in exceptions_2014:
    df_tmp = df_tmp.merge(df_world_bank[['Country Name', '2014']], left_on='Country', right_on='Country Name', how='left')
    df[worldbank_fname_list[i]] = df_tmp['2014']
  else:
    df_tmp = df_tmp.merge(df_world_bank[['Country Name', '2020']], left_on='Country', right_on='Country Name', how='left')
    df[worldbank_fname_list[i]] = df_tmp['2020']
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 152 entries, 0 to 151
Data columns (total 75 columns):
 #   Column                                                    Non-Null Count  Dtype
```

```
---  ------                                                                  --------------  -----
 0   Country                                                                 152 non-null    object
 1   Regional indicator                                                      152 non-null    object
 2   Happiness Score                                                         152 non-null    float64
 3   Standard error of ladder score                                          152 non-null    float64
 4   upperwhisker                                                            152 non-null    float64
 5   lowerwhisker                                                            152 non-null    float64
 6   Logged GDP per capita                                                   152 non-null    float64
 7   Social support                                                          152 non-null    float64
 8   Healthy life expectancy                                                 152 non-null    float64
 9   Freedom to make life choices                                            152 non-null    float64
 10  Generosity                                                              152 non-null    float64
 11  Perceptions of corruption                                               152 non-null    float64
 12  Ladder score in Dystopia                                                152 non-null    float64
 13  Explained by: Log GDP per capita                                        152 non-null    float64
 14  Explained by: Social support                                            152 non-null    float64
 15  Explained by: Healthy life expectancy                                   152 non-null    float64
 16  Explained by: Freedom to make life choices                              152 non-null    float64
 17  Explained by: Generosity                                                152 non-null    float64
 18  Explained by: Perceptions of corruption                                 152 non-null    float64
 19  Dystopia + residual                                                     152 non-null    float64
 20  High-technology exports (current US$)                                   117 non-null    float64
 21  Forest area (% of land area)                                            130 non-null    float64
 22  Government expenditure on education, total (% of GDP)                   118 non-null    float64
 23  GDP per capita growth (annual %)                                        129 non-null    float64
 24  Urban population (% of total population)                                130 non-null    float64
 25  Net migration                                                           130 non-null    float64
 26  GNI per capita, PPP (current international $)                           129 non-null    float64
 27  Exports of goods and services (% of GDP)                                123 non-null    float64
 28  Agriculture, forestry, and fishing, value added (% of GDP)              129 non-null    float64
 29  Literacy rate, adult total (% of people ages 15 and above)              28 non-null     float64
 30  Access to electricity (% of population)                                 130 non-null    float64
 31  Population growth (annual %)                                            130 non-null    float64
 32  Renewable electricity output (% of total electricity output)            130 non-null    float64
 33  Electric power consumption (kWh per capita)                             113 non-null    float64
 34  Central government debt, total (% of GDP)                               51 non-null     float64
 35  Renewable internal freshwater resources per capita (cubic meters)       129 non-null    float64
 36  Population density (people per sq. km of land area)                     130 non-null    float64
 37  Time required to get electricity (days)                                 0 non-null      float64
 38  Mobile cellular subscriptions (per 100 people)                          130 non-null    float64
 39  Fixed broadband subscriptions (per 100 people)                          128 non-null    float64
 40  Secure Internet servers (per 1 million people)                          130 non-null    float64
 41  Logistics performance index Overall (1=low to 5=high)                   0 non-null      float64
 42  Tax revenue (% of GDP)                                                  103 non-null    float64
 43  Mortality rate, under-5 (per 1,000 live births)                         130 non-null    float64
 44  Employment to population ratio, 15+, total (%) (modeled ILO estimate)   130 non-null    float64
 45  Death rate, crude (per 1,000 people)                                    130 non-null    float64
 46  Poverty headcount ratio at $2.15 a day (2017 PPP) (% of population)     55 non-null     float64
 47  PM2.5 air pollution, mean annual exposure (micrograms per cubic meter)  130 non-null    float64
 48  Agricultural land (% of land area)                                      130 non-null    float64
 49  Rural population (% of total population)                                130 non-null    float64
 50  Income share held by lowest 20%                                         55 non-null     float64
 51  Maternal mortality ratio (modeled estimate, per 100,000 live births)    130 non-null    float64
 52  CPIA public sector management and institutions cluster average (1=low to 6=high)  41 non-null  float64
```

```
1 df.head()
```

	Country	Regional indicator	Happiness Score	Standard error of ladder score	upperwhisker	lowerwhisker	Logged GDP per capita	Social support	Healthy life expectancy	Freedom to make life choices	...	Contrace prevale any meth of ma women 1
0	afghanistan	South Asia	2.5669	0.031311	2.628270	2.505530	7.462861	0.470367	52.590000	0.396573	...	
1	albania	Central and Eastern Europe	4.8827	0.056116	4.992687	4.772713	9.417931	0.671070	68.708138	0.781994	...	
2	algeria	Middle East and North Africa	5.0051	0.044236	5.091802	4.918397	9.537965	0.803385	65.905174	0.466611	...	
3	argentina	Latin America and Caribbean	5.9747	0.053442	6.079446	5.869954	9.810955	0.900568	68.803802	0.831132	...	
4	armenia	Commonwealth of Independent States	4.6768	0.058595	4.791646	4.561953	9.100476	0.757479	66.750656	0.712018	...	

5 rows × 75 columns

```
1 df_world_bank.head(5)
```

	Country Name	Country Code	Indicator Name	Indicator Code	1960	1961	1962	1963	1964	1965	...	2015	2016	
0	aruba	ABW	Fertility rate, total (births per woman)	SP.DYN.TFRT.IN	4.820000	4.655000	4.471000	4.271000	4.059000	3.842000	...	1.972000	1.953000	1.
1	africa eastern and southern	AFE	Fertility rate, total (births per woman)	SP.DYN.TFRT.IN	6.723226	6.742124	6.762343	6.778133	6.787821	6.799767	...	4.678619	4.616540	4.
2	afghanistan	AFG	Fertility rate, total (births per woman)	SP.DYN.TFRT.IN	7.282000	7.284000	7.292000	7.302000	7.304000	7.305000	...	5.405000	5.262000	5.
			Fertility											

This code block iterates through each column in the df DataFrame to identify and handle missing values (represented as NaN). Every column that have more than 50 missing column is dropped.

```
for col in df.columns:
    if df[col].isnull().any():
        nan_sum = df[col].isnull().sum()
        print(f"Column '{col}' has {nan_sum} NaN values.")
        if nan_sum > 50:
          df.drop(columns=[col], inplace=True)
          print(f"Column '{col}' has been dropped.")
```

```
Column 'High-technology exports (current US$)' has 35 NaN values.
Column 'Forest area (% of land area)' has 22 NaN values.
Column 'Government expenditure on education, total (% of GDP)' has 34 NaN values.
Column 'GDP per capita growth (annual %)' has 23 NaN values.
Column 'Urban population (% of total population)' has 22 NaN values.
Column 'Net migration' has 22 NaN values.
Column 'GNI per capita, PPP (current international $)' has 23 NaN values.
Column 'Exports of goods and services (% of GDP)' has 29 NaN values.
Column 'Agriculture, forestry, and fishing, value added (% of GDP)' has 23 NaN values.
Column 'Literacy rate, adult total (% of people ages 15 and above)' has 124 NaN values.
Column 'Literacy rate, adult total (% of people ages 15 and above)' has been dropped.
Column 'Access to electricity (% of population)' has 22 NaN values.
Column 'Population growth (annual %)' has 22 NaN values.
Column 'Renewable electricity output (% of total electricity output)' has 22 NaN values.
Column 'Electric power consumption (kWh per capita)' has 39 NaN values.
Column 'Central government debt, total (% of GDP)' has 101 NaN values.
Column 'Central government debt, total (% of GDP)' has been dropped.
Column 'Renewable internal freshwater resources per capita (cubic meters)' has 23 NaN values.
Column 'Population density (people per sq. km of land area)' has 22 NaN values.
Column 'Time required to get electricity (days)' has 152 NaN values.
Column 'Time required to get electricity (days)' has been dropped.
Column 'Mobile cellular subscriptions (per 100 people)' has 22 NaN values.
Column 'Fixed broadband subscriptions (per 100 people)' has 24 NaN values.
Column 'Secure Internet servers (per 1 million people)' has 22 NaN values.
Column 'Logistics performance index Overall (1=low to 5=high)' has 152 NaN values.
Column 'Logistics performance index Overall (1=low to 5=high)' has been dropped.
Column 'Tax revenue (% of GDP)' has 49 NaN values.
Column 'Mortality rate, under-5 (per 1,000 live births)' has 22 NaN values.
Column 'Employment to population ratio, 15+, total (%) (modeled ILO estimate)' has 22 NaN values.
Column 'Death rate, crude (per 1,000 people)' has 22 NaN values.
Column 'Poverty headcount ratio at $2.15 a day (2017 PPP) (% of population)' has 97 NaN values.
Column 'Poverty headcount ratio at $2.15 a day (2017 PPP) (% of population)' has been dropped.
Column 'PM2.5 air pollution, mean annual exposure (micrograms per cubic meter)' has 22 NaN values.
Column 'Agricultural land (% of land area)' has 22 NaN values.
Column 'Rural population (% of total population)' has 22 NaN values.
Column 'Income share held by lowest 20%' has 97 NaN values.
Column 'Income share held by lowest 20%' has been dropped.
Column 'Maternal mortality ratio (modeled estimate, per 100,000 live births)' has 22 NaN values.
Column 'CPIA public sector management and institutions cluster average (1=low to 6=high)' has 111 NaN values.
Column 'CPIA public sector management and institutions cluster average (1=low to 6=high)' has been dropped.
Column 'Population in urban agglomerations of more than 1 million (% of total population)' has 52 NaN values.
Column 'Population in urban agglomerations of more than 1 million (% of total population)' has been dropped.
Column 'School enrollment, primary and secondary (gross), gender parity index (GPI)' has 70 NaN values.
Column 'School enrollment, primary and secondary (gross), gender parity index (GPI)' has been dropped.
Column 'Labor force, female (% of total labor force)' has 22 NaN values.
Column 'Literacy rate, youth total (% of people ages 15-24)' has 121 NaN values.
Column 'Literacy rate, youth total (% of people ages 15-24)' has been dropped.
Column 'Population in the largest city (% of urban population)' has 33 NaN values.
Column 'New businesses registered (number)' has 50 NaN values.
Column 'Strength of legal rights index (0=weak to 12=strong)' has 152 NaN values.
Column 'Strength of legal rights index (0=weak to 12=strong)' has been dropped.
Column 'Firms with female participation in ownership (% of firms)' has 142 NaN values.
Column 'Firms with female participation in ownership (% of firms)' has been dropped.
Column 'Cause of death, by injury (% of total)' has 152 NaN values.
Column 'Cause of death, by injury (% of total)' has been dropped.
Column 'Income share held by highest 20%' has 97 NaN values.
```

```
Column 'Income share held by highest 20%' has been dropped.
Column 'Refugee population by country or territory of asylum' has 29 NaN values.
```

```
df.head(5)
```

	Country	Regional indicator	Happiness Score	Standard error of ladder score	upperwhisker	lowerwhisker	Logged GDP per capita	Social support	Healthy life expectancy	Freedom to make life choices	...	Refug populati by count territo of asyl
0	afghanistan	South Asia	2.5669	0.031311	2.628270	2.505530	7.462861	0.470367	52.590000	0.396573	...	72278
1	albania	Central and Eastern Europe	4.8827	0.056116	4.992687	4.772713	9.417931	0.671070	68.708138	0.781994	...	105
2	algeria	Middle East and North Africa	5.0051	0.044236	5.091802	4.918397	9.537965	0.803385	65.905174	0.466611	...	97651
3	argentina	Latin America and Caribbean	5.9747	0.053442	6.079446	5.869954	9.810955	0.900568	68.803802	0.831132	...	3965
4	armenia	Commonwealth of Independent States	4.6768	0.058595	4.791646	4.561953	9.100476	0.757479	66.750656	0.712018	...	107910

5 rows × 59 columns

```
country_list = df['Country'].tolist()
```

This code block focuses on loading user data from MAL user dataset and performing initial processing steps, specifically related to dates and calculating user age.

```
users = pd.read_csv('users_cleaned.csv')
users['join_date'] = pd.to_datetime(users['join_date'])
users['last_online'] = pd.to_datetime(users['last_online'])
users['birth_date'] = pd.to_datetime(users['birth_date'])
now = datetime.datetime.now()
users['age'] = (now - users['birth_date']).dt.days / 365.25
users['age'] = users['age'].astype(int)
users.head()
```

	username	user_id	user_watching	user_completed	user_onhold	user_dropped	user_plantowatch	user_days_spent_watching	gen
0	karthiga	2255153	3	49	1	0	0	55.091667	Fem
1	Damonashu	37326	45	195	27	25	59	82.574306	M
2	bskai	228342	25	414	2	5	11	159.483333	M
3	terune_uzumaki	327311	5	5	0	0	0	11.394444	Fem
4	Bas_G	5015094	35	114	6	20	175	30.458333	M

```
users.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 108711 entries, 0 to 108710
Data columns (total 18 columns):
 #   Column                   Non-Null Count   Dtype
---  ------                   --------------   -----
 0   username                 108710 non-null  object
 1   user_id                  108711 non-null  int64
 2   user_watching            108711 non-null  int64
 3   user_completed           108711 non-null  int64
 4   user_onhold              108711 non-null  int64
 5   user_dropped             108711 non-null  int64
 6   user_plantowatch         108711 non-null  int64
 7   user_days_spent_watching 108711 non-null  float64
 8   gender                   108711 non-null  object
 9   location                 108706 non-null  object
 10  birth_date               108711 non-null  datetime64[ns]
 11  access_rank              0 non-null       float64
 12  join_date                108711 non-null  datetime64[ns]
 13  last_online              108711 non-null  datetime64[ns]
 14  stats_mean_score         108711 non-null  float64
```

```
 15  stats_rewatched           108711 non-null  float64
 16  stats_episodes            108711 non-null  int64
 17  age                       108711 non-null  int64
dtypes: datetime64[ns](3), float64(4), int64(8), object(3)
memory usage: 14.9+ MB
```

This code block checks for and removes two columns, 'access_rank' and 'user_id', from the users DataFrame. These columns are not needed for subsequent analysis or modeling.

```
if 'access_rank' in users.columns:
    users = users.drop(columns=['access_rank'])
    print("Column 'access_rank' dropped successfully from users.")
else:
    print("Column 'access_rank' not found in users.")

if 'user_id' in users.columns:
    users = users.drop(columns=['user_id'])
    print("Column 'user_id' dropped successfully from users.")
else:
    print("Column 'user_id' not found in users.")
```

```
Column 'access_rank' dropped successfully from users.
Column 'user_id' dropped successfully from users.
```

```
us_states = [
    "Alabama", "Alaska", "Arizona", "Arkansas", "California", "Colorado",
    "Connecticut", "Delaware", "Florida", "Georgia", "Hawaii", "Idaho",
    "Illinois", "Indiana", "Iowa", "Kansas", "Kentucky", "Louisiana",
    "Maine", "Maryland", "Massachusetts", "Michigan", "Minnesota",
    "Mississippi", "Missouri", "Montana", "Nebraska", "Nevada",
    "New Hampshire", "New Jersey", "New Mexico", "New York",
    "North Carolina", "North Dakota", "Ohio", "Oklahoma", "Oregon",
    "Pennsylvania", "Rhode Island", "South Carolina", "South Dakota",
    "Tennessee", "Texas", "Utah", "Vermont", "Virginia", "Washington",
    "West Virginia", "Wisconsin", "Wyoming"
]
us_states_lower = [state.lower() for state in us_states]
```

Function to turn location data to country.

```
def find_country(input):
  input = str(input)
  if input == 'nan':
    return 'Other'
  else:
    for country in country_list:
      if country in input.lower():
        return country
    for state in us_states_lower:
      if state in input.lower():
        return 'united states'
    return 'Other'
```

This code block applies the above function to determine the country for each user based on their location information.

```
users['Country'] = users['location'].apply(find_country)
users.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 108711 entries, 0 to 108710
Data columns (total 17 columns):
 #   Column                    Non-Null Count   Dtype
---  ------                    --------------   -----
 0   username                  108710 non-null  object
 1   user_watching             108711 non-null  int64
 2   user_completed            108711 non-null  int64
 3   user_onhold               108711 non-null  int64
 4   user_dropped              108711 non-null  int64
 5   user_plantowatch          108711 non-null  int64
 6   user_days_spent_watching  108711 non-null  float64
 7   gender                    108711 non-null  object
 8   location                  108706 non-null  object
 9   birth_date                108711 non-null  datetime64[ns]
 10  join_date                 108711 non-null  datetime64[ns]
 11  last_online               108711 non-null  datetime64[ns]
 12  stats_mean_score          108711 non-null  float64
 13  stats_rewatched           108711 non-null  float64
 14  stats_episodes            108711 non-null  int64
 15  age                       108711 non-null  int64
```

```
 16  Country                   108711 non-null  object
dtypes: datetime64[ns](3), float64(3), int64(7), object(4)
memory usage: 14.1+ MB
```

This code block drops the now unnecessary location column.

```
if 'location' in users.columns:
    users = users.drop(columns=['location'])
    print("Column 'location' dropped successfully from users.")
else:
    print("Column 'location' not found in users.")
```

```
Column 'location' dropped successfully from users.
```

Country column is important to merge df to users. The rows with Other value in Country column can't be merged and thus will be dropped.

```
other_count = users[users['Country'] == 'Other'].shape[0]
print(f"Number of 'Other' values in the 'Country' column: {other_count}")
```

```
Number of 'Other' values in the 'Country' column: 40904
```

This code block performs several filtering and cleaning steps on the users DataFrame to prepare it for further analysis or merging.

```
users = users[users['Country'] != 'Other']
users = users[users['user_completed'] != 0]
users = users[users['stats_mean_score'] != 0]
users = users[users['gender'].isin(['Male', 'Female'])]
users.reset_index(drop=True, inplace=True)
users.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 65828 entries, 0 to 65827
Data columns (total 16 columns):
 #   Column                    Non-Null Count  Dtype
---  ------                    --------------  -----
 0   username                  65827 non-null  object
 1   user_watching             65828 non-null  int64
 2   user_completed            65828 non-null  int64
 3   user_onhold               65828 non-null  int64
 4   user_dropped              65828 non-null  int64
 5   user_plantowatch          65828 non-null  int64
 6   user_days_spent_watching  65828 non-null  float64
 7   gender                    65828 non-null  object
 8   birth_date                65828 non-null  datetime64[ns]
 9   join_date                 65828 non-null  datetime64[ns]
 10  last_online               65828 non-null  datetime64[ns]
 11  stats_mean_score          65828 non-null  float64
 12  stats_rewatched           65828 non-null  float64
 13  stats_episodes            65828 non-null  int64
 14  age                       65828 non-null  int64
 15  Country                   65828 non-null  object
dtypes: datetime64[ns](3), float64(3), int64(7), object(3)
memory usage: 8.0+ MB
```

This code block performs a merge operation between two pandas DataFrames: users and df.

```
user_country_stats = pd.merge(users, df, on='Country', how='inner')
user_country_stats.head()
```

	username	user_watching	user_completed	user_onhold	user_dropped	user_plantowatch	user_days_spent_watching	gender	birt
0	karthiga	3	49	1	0	0	55.091667	Female	199
1	Damonashu	45	195	27	25	59	82.574306	Male	199
2	bskai	25	414	2	5	11	159.483333	Male	199
3	terune_uzumaki	5	5	0	0	0	11.394444	Female	199
4	HimeAria	2	87	2	0	28	19.963194	Female	199

5 rows × 74 columns

This code block is designed to check each column in the user_country_stats DataFrame for missing values (represented as NaN). If a column contains missing values, the code calculates the total number of missing values in that column and prints this information. If the number of missing values is bigger than 5000, the column is dropped.

```
for col in user_country_stats.columns:
    if user_country_stats[col].isnull().any():
        nan_sum = user_country_stats[col].isnull().sum()
        print(f"Column '{col}' has {nan_sum} NaN values.")
        if nan_sum > 5000:
          user_country_stats.drop(columns=[col], inplace=True)
          print(f"Column '{col}' has been dropped.")
```

```
Column 'username' has 1 NaN values.
Column 'High-technology exports (current US$)' has 4748 NaN values.
Column 'Forest area (% of land area)' has 4350 NaN values.
Column 'Government expenditure on education, total (% of GDP)' has 5395 NaN values.
Column 'Government expenditure on education, total (% of GDP)' has been dropped.
Column 'GDP per capita growth (annual %)' has 4350 NaN values.
Column 'Urban population (% of total population)' has 4350 NaN values.
Column 'Net migration' has 4350 NaN values.
Column 'GNI per capita, PPP (current international $)' has 4350 NaN values.
Column 'Exports of goods and services (% of GDP)' has 4419 NaN values.
Column 'Agriculture, forestry, and fishing, value added (% of GDP)' has 4350 NaN values.
Column 'Access to electricity (% of population)' has 4350 NaN values.
Column 'Population growth (annual %)' has 4350 NaN values.
Column 'Renewable electricity output (% of total electricity output)' has 4350 NaN values.
Column 'Electric power consumption (kWh per capita)' has 4381 NaN values.
Column 'Renewable internal freshwater resources per capita (cubic meters)' has 4364 NaN values.
Column 'Population density (people per sq. km of land area)' has 4350 NaN values.
Column 'Mobile cellular subscriptions (per 100 people)' has 4350 NaN values.
Column 'Fixed broadband subscriptions (per 100 people)' has 4350 NaN values.
Column 'Secure Internet servers (per 1 million people)' has 4350 NaN values.
Column 'Tax revenue (% of GDP)' has 6695 NaN values.
Column 'Tax revenue (% of GDP)' has been dropped.
Column 'Mortality rate, under-5 (per 1,000 live births)' has 4350 NaN values.
Column 'Employment to population ratio, 15+, total (%) (modeled ILO estimate)' has 4350 NaN values.
Column 'Death rate, crude (per 1,000 people)' has 4350 NaN values.
Column 'PM2.5 air pollution, mean annual exposure (micrograms per cubic meter)' has 4350 NaN values.
Column 'Agricultural land (% of land area)' has 4350 NaN values.
Column 'Rural population (% of total population)' has 4350 NaN values.
Column 'Maternal mortality ratio (modeled estimate, per 100,000 live births)' has 4350 NaN values.
Column 'Labor force, female (% of total labor force)' has 4350 NaN values.
Column 'Population in the largest city (% of urban population)' has 4662 NaN values.
Column 'New businesses registered (number)' has 23923 NaN values.
Column 'New businesses registered (number)' has been dropped.
Column 'Refugee population by country or territory of asylum' has 5135 NaN values.
Column 'Refugee population by country or territory of asylum' has been dropped.
Column 'Life expectancy at birth, male (years)' has 4350 NaN values.
Column 'Life expectancy at birth, female (years)' has 4350 NaN values.
Column 'Medium and high-tech exports (% manufactured exports)' has 4391 NaN values.
Column 'Military expenditure (% of GDP)' has 4602 NaN values.
Column 'Inflation, consumer prices (annual %)' has 4903 NaN values.
Column 'Population, total' has 4350 NaN values.
Column 'GDP per capita (current US$)' has 4350 NaN values.
Column 'Unemployment, total (% of total labor force) (modeled ILO estimate)' has 4350 NaN values.
Column 'Fertility rate, total (births per woman)' has 4350 NaN values.
```

This code block drops the rows with NaN value in most of the columns.

```
if 'Inflation, consumer prices (annual %)' in user_country_stats.columns:
    user_country_stats.dropna(subset=['Inflation, consumer prices (annual %)'], inplace=True)
    print("Rows with missing values in 'Inflation, consumer prices (annual %)' dropped.")
else:
    print("'Inflation, consumer prices (annual %)' column not found in the DataFrame.")

user_country_stats.info()
```

```
Rows with missing values in 'Inflation, consumer prices (annual %)' dropped.
<class 'pandas.core.frame.DataFrame'>
Index: 60925 entries, 0 to 65827
Data columns (total 70 columns):
 #   Column                                                                Non-Null Count  Dtype
---  ------                                                                --------------  -----
 0   username                                                              60924 non-null  object
 1   user_watching                                                         60925 non-null  int64
 2   user_completed                                                        60925 non-null  int64
 3   user_onhold                                                           60925 non-null  int64
 4   user_dropped                                                          60925 non-null  int64
 5   user_plantowatch                                                      60925 non-null  int64
 6   user_days_spent_watching                                              60925 non-null  float64
 7   gender                                                                60925 non-null  object
 8   birth_date                                                            60925 non-null  datetime64[ns]
 9   join_date                                                             60925 non-null  datetime64[ns]
 10  last_online                                                           60925 non-null  datetime64[ns]
 11  stats_mean_score                                                      60925 non-null  float64
 12  stats_rewatched                                                       60925 non-null  float64
 13  stats_episodes                                                        60925 non-null  int64
 14  age                                                                   60925 non-null  int64
 15  Country                                                               60925 non-null  object
 16  Regional indicator                                                    60925 non-null  object
 17  Happiness Score                                                       60925 non-null  float64
 18  Standard error of ladder score                                        60925 non-null  float64
 19  upperwhisker                                                          60925 non-null  float64
 20  lowerwhisker                                                          60925 non-null  float64
 21  Logged GDP per capita                                                 60925 non-null  float64
 22  Social support                                                        60925 non-null  float64
 23  Healthy life expectancy                                               60925 non-null  float64
 24  Freedom to make life choices                                          60925 non-null  float64
 25  Generosity                                                            60925 non-null  float64
 26  Perceptions of corruption                                             60925 non-null  float64
 27  Ladder score in Dystopia                                              60925 non-null  float64
 28  Explained by: Log GDP per capita                                      60925 non-null  float64
 29  Explained by: Social support                                          60925 non-null  float64
 30  Explained by: Healthy life expectancy                                 60925 non-null  float64
 31  Explained by: Freedom to make life choices                            60925 non-null  float64
 32  Explained by: Generosity                                              60925 non-null  float64
 33  Explained by: Perceptions of corruption                               60925 non-null  float64
 34  Dystopia + residual                                                   60925 non-null  float64
 35  High-technology exports (current US$)                                 60527 non-null  float64
 36  Forest area (% of land area)                                          60925 non-null  float64
 37  GDP per capita growth (annual %)                                      60925 non-null  float64
 38  Urban population (% of total population)                              60925 non-null  float64
 39  Net migration                                                         60925 non-null  float64
 40  GNI per capita, PPP (current international $)                         60925 non-null  float64
 41  Exports of goods and services (% of GDP)                              60872 non-null  float64
 42  Agriculture, forestry, and fishing, value added (% of GDP)            60925 non-null  float64
 43  Access to electricity (% of population)                               60925 non-null  float64
 44  Population growth (annual %)                                          60925 non-null  float64
 45  Renewable electricity output (% of total electricity output)          60925 non-null  float64
 46  Electric power consumption (kWh per capita)                           60894 non-null  float64
 47  Renewable internal freshwater resources per capita (cubic meters)     60911 non-null  float64
 48  Population density (people per sq. km of land area)                   60925 non-null  float64
 49  Mobile cellular subscriptions (per 100 people)                        60925 non-null  float64
 50  Fixed broadband subscriptions (per 100 people)                        60925 non-null  float64
 51  Secure Internet servers (per 1 million people)                        60925 non-null  float64
```

This code block is used to identify and count missing values within each column of the user_country_stats DataFrame.

```
for col in user_country_stats.columns:
    if user_country_stats[col].isnull().any():
        nan_sum = user_country_stats[col].isnull().sum()
        print(f"Column '{col}' has {nan_sum} NaN values.")
```

```
Column 'username' has 1 NaN values.
Column 'High-technology exports (current US$)' has 398 NaN values.
Column 'Exports of goods and services (% of GDP)' has 53 NaN values.
Column 'Electric power consumption (kWh per capita)' has 31 NaN values.
Column 'Renewable internal freshwater resources per capita (cubic meters)' has 14 NaN values.
Column 'Population in the largest city (% of urban population)' has 312 NaN values.
Column 'Medium and high-tech exports (% manufactured exports)' has 41 NaN values.
Column 'Military expenditure (% of GDP)' has 252 NaN values.
```

This code block drops the column named username in the user_country_stats DataFrame.

```
if 'username' in user_country_stats.columns:
    user_country_stats.dropna(subset=['username'], inplace=True)
    print("Rows with missing values in 'username' dropped.")
else:
    print("'username' column not found in the DataFrame.")
```

```
Rows with missing values in 'username' dropped.
```

This code block focuses on addressing any remaining missing values within the numerical columns of the user_country_stats DataFrame.

```
numeric_columns = user_country_stats.select_dtypes(include=np.number).columns
user_country_stats[numeric_columns] = user_country_stats[numeric_columns].fillna(user_country_stats[numeric_columns].mean())
```

This code block is used to systematically check each column in the user_country_stats DataFrame for missing values.

```
for col in user_country_stats.columns:
    if user_country_stats[col].isnull().any():
        nan_sum = user_country_stats[col].isnull().sum()
        print(f"Column '{col}' has {nan_sum} NaN values.")
```

This code block is used to remove three columns from the user_country_stats DataFrame: 'upperwhisker', 'lowerwhisker', and 'Standard error of ladder score'. These columns are considered unnecessary for the subsequent analysis or machine learning tasks.

```
if 'upperwhisker' in user_country_stats.columns:
    user_country_stats = user_country_stats.drop(columns=['upperwhisker'])
    print("Column 'upperwhisker' dropped successfully from user_country_stats.")
else:
    print("Column 'upperwhisker' not found in user_country_stats.")

if 'lowerwhisker' in user_country_stats.columns:
    user_country_stats = user_country_stats.drop(columns=['lowerwhisker'])
    print("Column 'lowerwhisker' dropped successfully from user_country_stats.")
else:
    print("Column 'lowerwhisker' not found in user_country_stats.")

if 'Standard error of ladder score' in user_country_stats.columns:
    user_country_stats = user_country_stats.drop(columns=['Standard error of ladder score'])
    print("Column 'Standard error of ladder score' dropped successfully from user_country_stats.")
else:
    print("Column 'Standard error of ladder score' not found in user_country_stats.")
```

```
Column 'upperwhisker' dropped successfully from user_country_stats.
Column 'lowerwhisker' dropped successfully from user_country_stats.
Column 'Standard error of ladder score' dropped successfully from user_country_stats.
```

This code block focuses on converting categorical data in two columns of user_country_stats into a numerical format.

```
label_encoder = LabelEncoder()
user_country_stats['Regional indicator'] = label_encoder.fit_transform(user_country_stats['Regional indicator'])
user_country_stats['Country'] = label_encoder.fit_transform(user_country_stats['Country'])
```

This line of code converts the categorical 'gender' column into a numerical representation, creating a new column called gender_numeric.

```
user_country_stats['gender_numeric'] = user_country_stats['gender'].map({'Male': 0, 'Female': 1})
```

```
user_country_stats.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 60924 entries, 0 to 65827
Data columns (total 68 columns):
 #   Column                                                                Non-Null Count  Dtype
---  ------                                                                --------------  -----
 0   username                                                              60924 non-null  object
 1   user_watching                                                         60924 non-null  int64
 2   user_completed                                                        60924 non-null  int64
 3   user_onhold                                                           60924 non-null  int64
 4   user_dropped                                                          60924 non-null  int64
 5   user_plantowatch                                                      60924 non-null  int64
 6   user_days_spent_watching                                              60924 non-null  float64
 7   gender                                                                60924 non-null  object
 8   birth_date                                                            60924 non-null  datetime64[ns]
 9   join_date                                                             60924 non-null  datetime64[ns]
 10  last_online                                                           60924 non-null  datetime64[ns]
 11  stats_mean_score                                                      60924 non-null  float64
 12  stats_rewatched                                                       60924 non-null  float64
 13  stats_episodes                                                        60924 non-null  int64
```

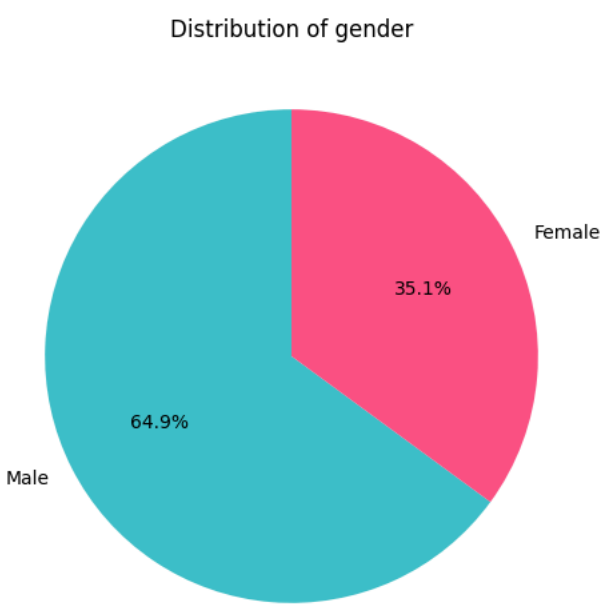
```
 14  age                                                                  60924 non-null  int64
 15  Country                                                              60924 non-null  int64
 16  Regional indicator                                                   60924 non-null  int64
 17  Happiness Score                                                      60924 non-null  float64
 18  Logged GDP per capita                                                60924 non-null  float64
 19  Social support                                                       60924 non-null  float64
 20  Healthy life expectancy                                              60924 non-null  float64
 21  Freedom to make life choices                                         60924 non-null  float64
 22  Generosity                                                           60924 non-null  float64
 23  Perceptions of corruption                                            60924 non-null  float64
 24  Ladder score in Dystopia                                             60924 non-null  float64
 25  Explained by: Log GDP per capita                                     60924 non-null  float64
 26  Explained by: Social support                                         60924 non-null  float64
 27  Explained by: Healthy life expectancy                                60924 non-null  float64
 28  Explained by: Freedom to make life choices                           60924 non-null  float64
 29  Explained by: Generosity                                             60924 non-null  float64
 30  Explained by: Perceptions of corruption                              60924 non-null  float64
 31  Dystopia + residual                                                  60924 non-null  float64
 32  High-technology exports (current US$)                                60924 non-null  float64
 33  Forest area (% of land area)                                         60924 non-null  float64
 34  GDP per capita growth (annual %)                                     60924 non-null  float64
 35  Urban population (% of total population)                             60924 non-null  float64
 36  Net migration                                                        60924 non-null  float64
 37  GNI per capita, PPP (current international $)                        60924 non-null  float64
 38  Exports of goods and services (% of GDP)                             60924 non-null  float64
 39  Agriculture, forestry, and fishing, value added (% of GDP)           60924 non-null  float64
 40  Access to electricity (% of population)                              60924 non-null  float64
 41  Population growth (annual %)                                         60924 non-null  float64
 42  Renewable electricity output (% of total electricity output)         60924 non-null  float64
 43  Electric power consumption (kWh per capita)                          60924 non-null  float64
 44  Renewable internal freshwater resources per capita (cubic meters)    60924 non-null  float64
 45  Population density (people per sq. km of land area)                  60924 non-null  float64
 46  Mobile cellular subscriptions (per 100 people)                       60924 non-null  float64
 47  Fixed broadband subscriptions (per 100 people)                       60924 non-null  float64
 48  Secure Internet servers (per 1 million people)                       60924 non-null  float64
 49  Mortality rate, under-5 (per 1,000 live births)                      60924 non-null  float64
 50  Employment to population ratio, 15+, total (%) (modeled ILO estimate)  60924 non-null  float64
 51  Death rate, crude (per 1,000 people)                                 60924 non-null  float64
 52  PM2.5 air pollution, mean annual exposure (micrograms per cubic meter)  60924 non-null  float64
```

Conduct Exploratory Data Analysis

```
numerical_features = ['age',
                      'stats_mean_score',
                      'Happiness Score',
                      'user_days_spent_watching',
                      'user_completed',
                      'GDP per capita (current US$)',
                      'Logged GDP per capita',
                      'Social support',
                      'Healthy life expectancy',
                      'Dystopia + residual',
                      'Freedom to make life choices',
                      'Generosity',
                      'Perceptions of corruption',
                      'user_watching',
                      'user_plantowatch']

interesting_pairs = [
    ('Happiness Score', 'Logged GDP per capita'),
    ('Happiness Score', 'Healthy life expectancy'),
    ('stats_mean_score', 'user_completed'),
    ('user_days_spent_watching', 'user_completed'),
]
```

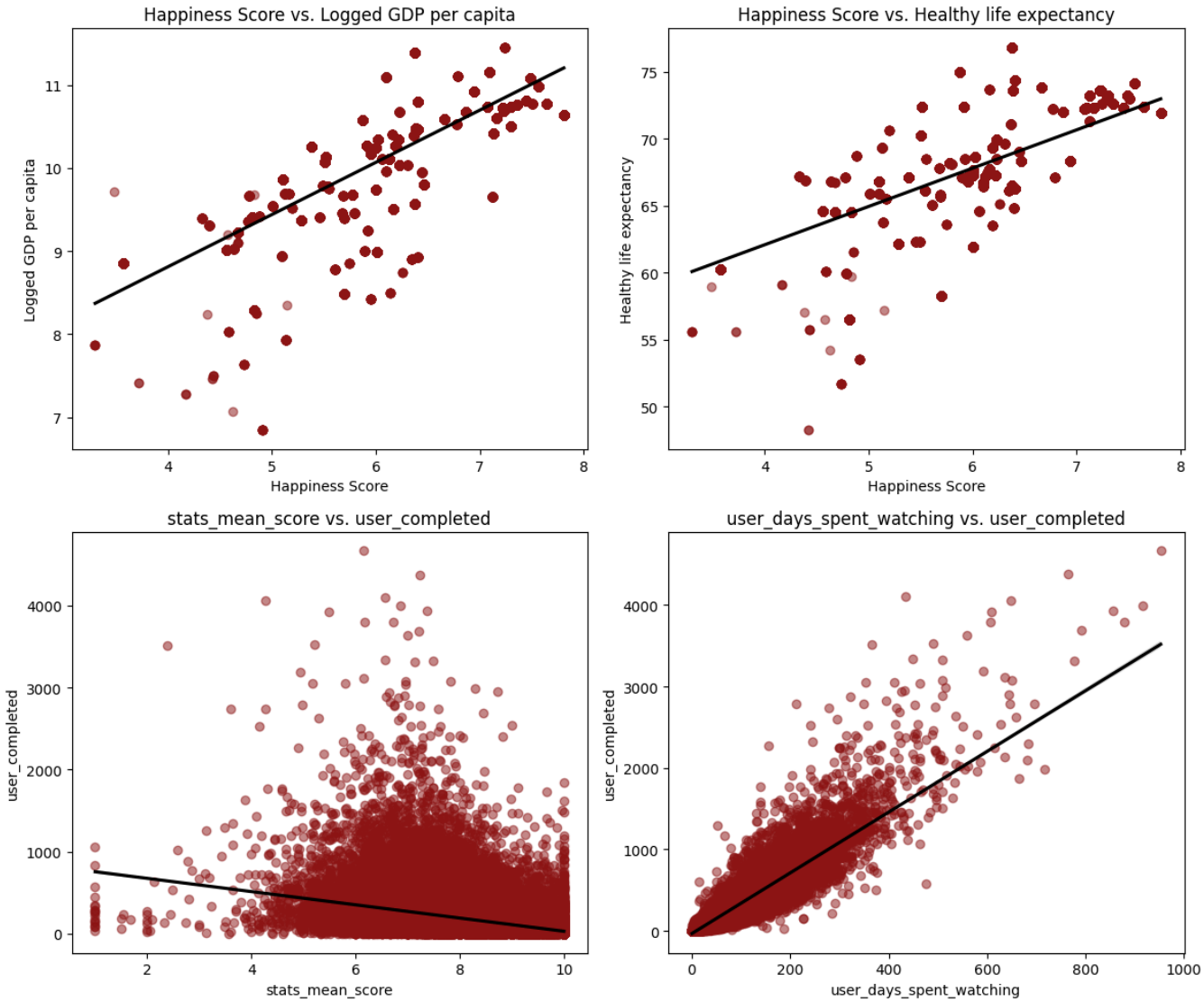
This code block focuses on visualizing the distribution of the gender column in the user_country_stats DataFrame using a pie chart.

```
categorical_feature = 'gender'
my_colors = ['#3FC1C9', '#FC5185']

gender_counts = user_country_stats[categorical_feature].value_counts()
plt.figure(figsize=(8, 6))
plt.pie(gender_counts, labels=gender_counts.index, autopct='%1.1f%%', startangle=90, colors=my_colors)
plt.title(f'Distribution of {categorical_feature}')
plt.show()
```



This code block focuses on visualizing the relationships between specific pairs of numerical features identified as "interesting" earlier in the notebook. It uses scatter plots with regression lines to show the general trend between the features.

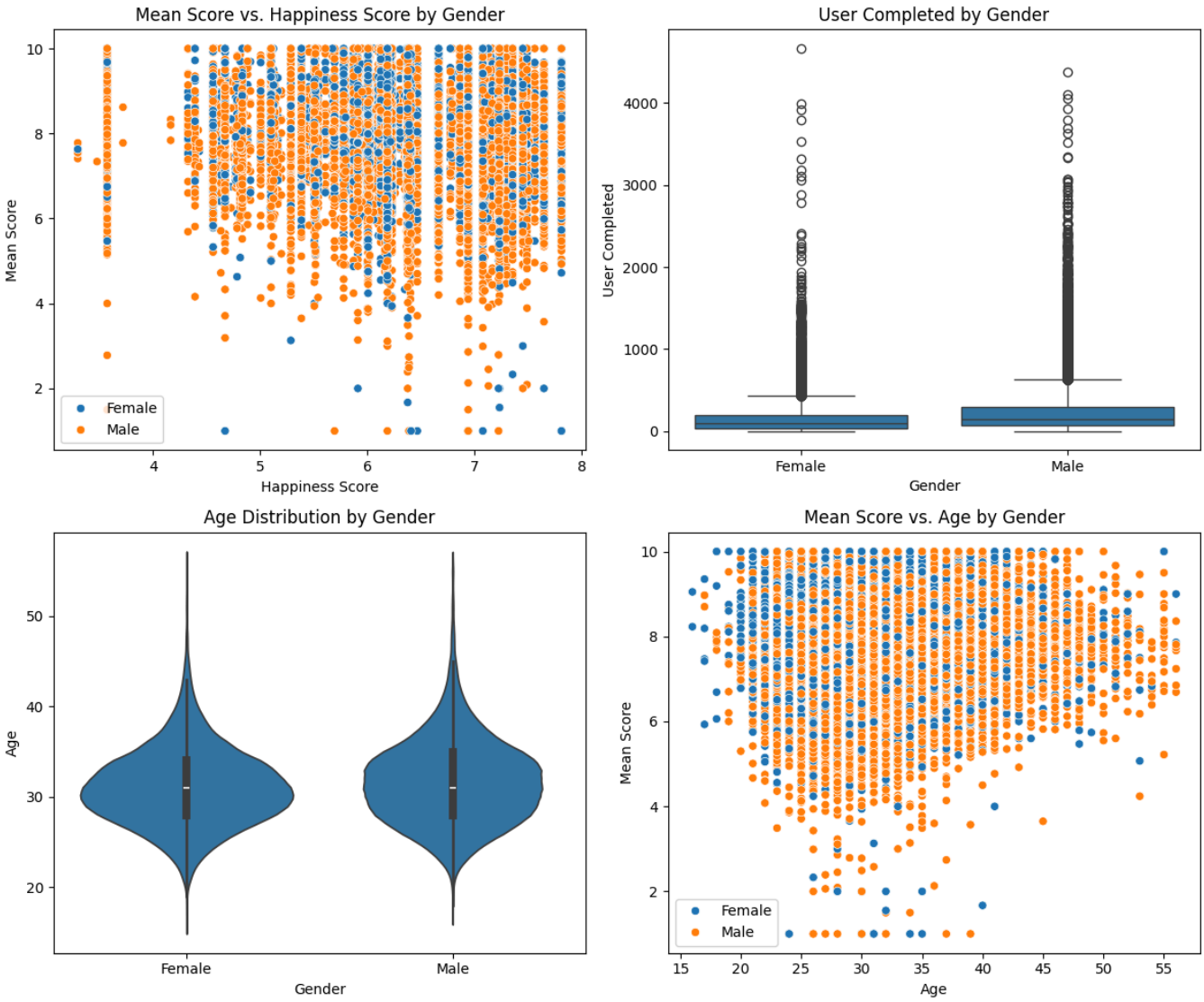
```
num_pairs = len(interesting_pairs)
num_cols = 2
num_rows = math.ceil(num_pairs / num_cols)

fig, axes = plt.subplots(num_rows, num_cols, figsize=(12, 10))
axes = axes.flatten()

for i, pair in enumerate(interesting_pairs):
    feature1, feature2 = pair
    sns.regplot(x=feature1, y=feature2, data=user_country_stats, scatter_kws={'alpha': 0.5, 'color': '#8E1616'}, line_kws={'color':
    axes[i].set_title(f'{feature1} vs. {feature2}')

for i in range(num_pairs, num_rows * num_cols):
    fig.delaxes(axes[i])

plt.tight_layout()
plt.show()
```



This code block uses the Seaborn and Matplotlib libraries to create a figure with multiple plots, specifically a 2x2 grid of subplots. Each subplot visualizes the relationship between different features from the user_country_stats DataFrame, differentiated by gender.

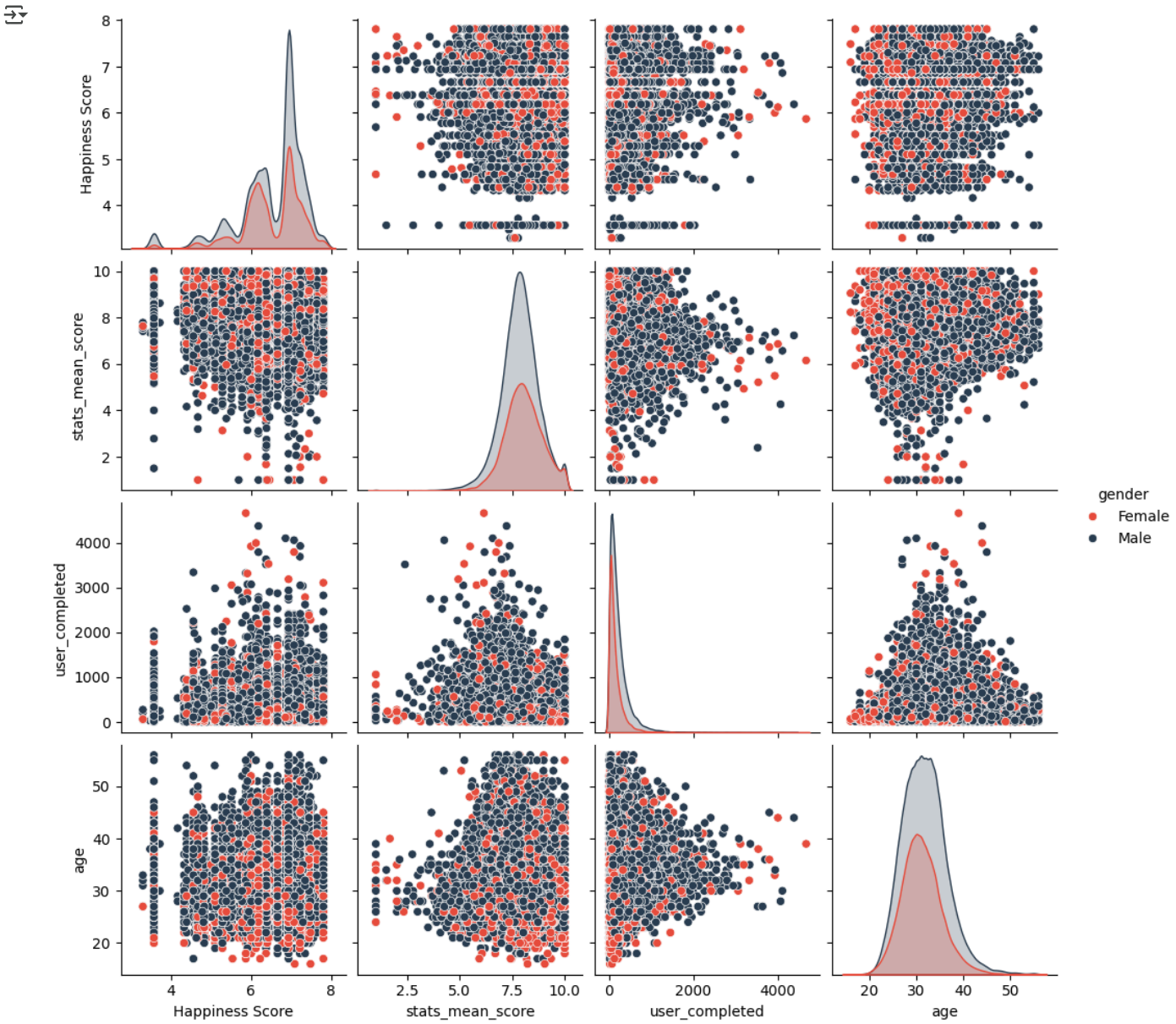
```
fig, axes = plt.subplots(2, 2, figsize=(12, 10))
sns.scatterplot(x='Happiness Score', y='stats_mean_score', hue='gender', data=user_country_stats, ax=axes[0, 0])
axes[0, 0].set_title('Mean Score vs. Happiness Score by Gender')
axes[0, 0].set_xlabel('Happiness Score')
axes[0, 0].set_ylabel('Mean Score')
axes[0, 0].legend()

sns.boxplot(x='gender', y='user_completed', data=user_country_stats, ax=axes[0, 1])
axes[0, 1].set_title('User Completed by Gender')
axes[0, 1].set_xlabel('Gender')
axes[0, 1].set_ylabel('User Completed')

sns.violinplot(x='gender', y='age', data=user_country_stats, ax=axes[1, 0])
axes[1, 0].set_title('Age Distribution by Gender')
axes[1, 0].set_xlabel('Gender')
axes[1, 0].set_ylabel('Age')

sns.scatterplot(x='age', y='stats_mean_score', hue='gender', data=user_country_stats, ax=axes[1, 1])
axes[1, 1].set_title('Mean Score vs. Age by Gender')
axes[1, 1].set_xlabel('Age')
axes[1, 1].set_ylabel('Mean Score')
axes[1, 1].legend()

```

```
plt.tight_layout()
plt.show();
```



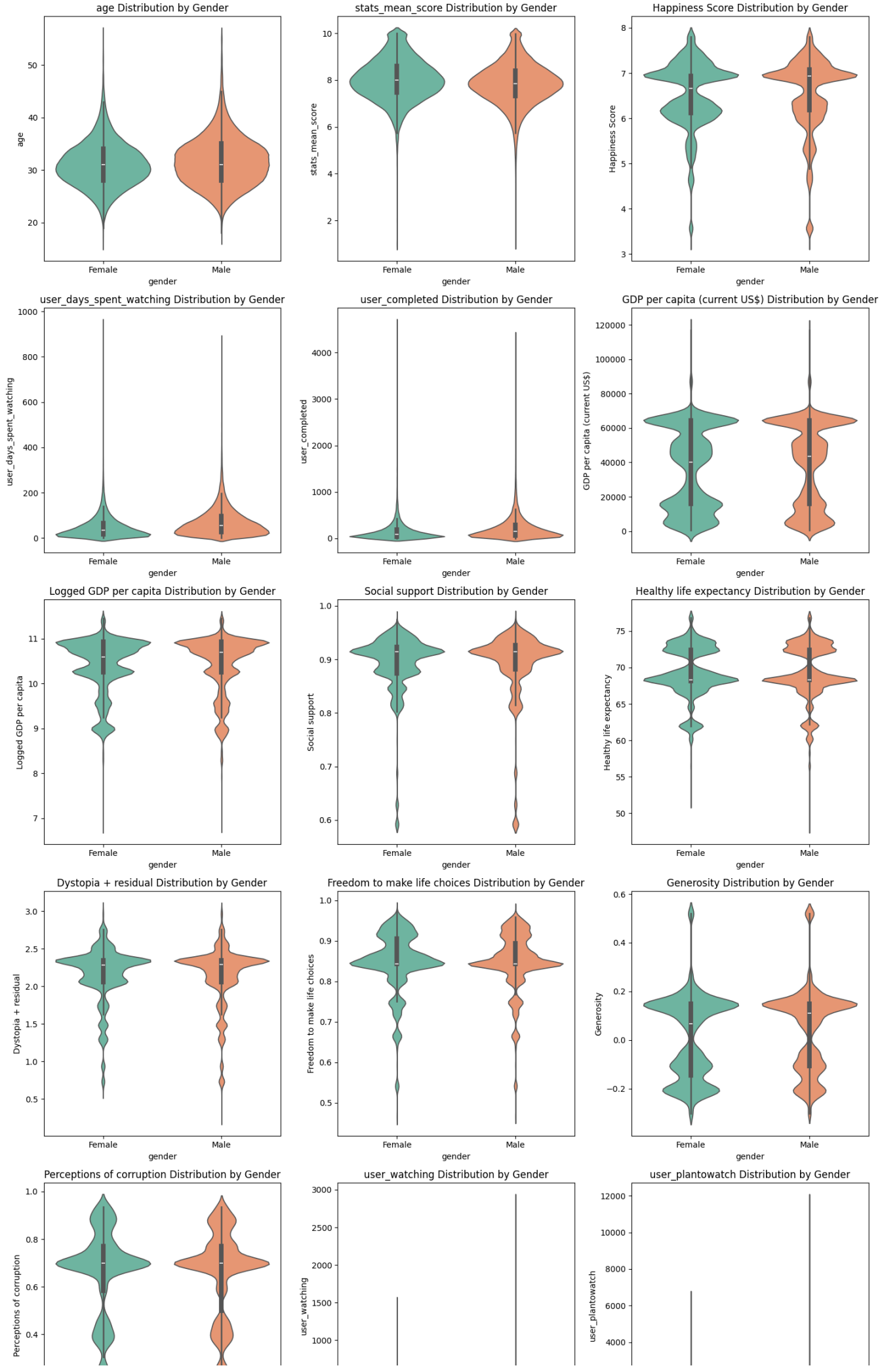
This code block generates a pair plot using the seaborn library. The resulting pair plot will show scatter plots for every combination of the specified numerical variables ('Happiness Score', 'stats_mean_score', 'user_completed', and 'age').

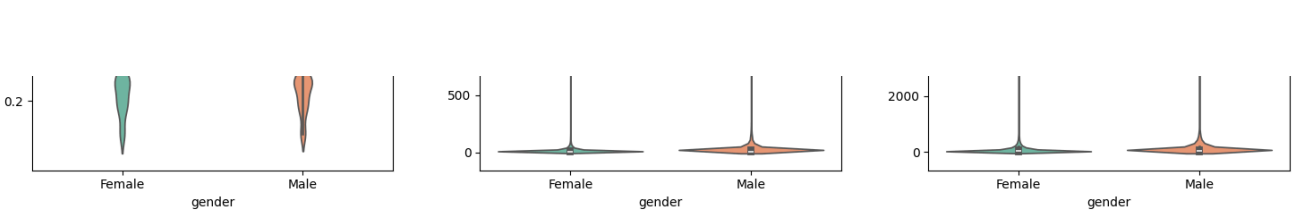
```
sns.pairplot(user_country_stats,
             vars=['Happiness Score', 'stats_mean_score', 'user_completed', 'age'],
             hue='gender',
             palette={'Male': '#2C3E50', 'Female': '#E74C3C'})
plt.show()
```



This code block generates a series of violin plots, one for each numerical feature in the numerical_features list. Each plot shows the distribution of that feature for both male and female users, allowing for a visual comparison of how these distributions differ by gender.

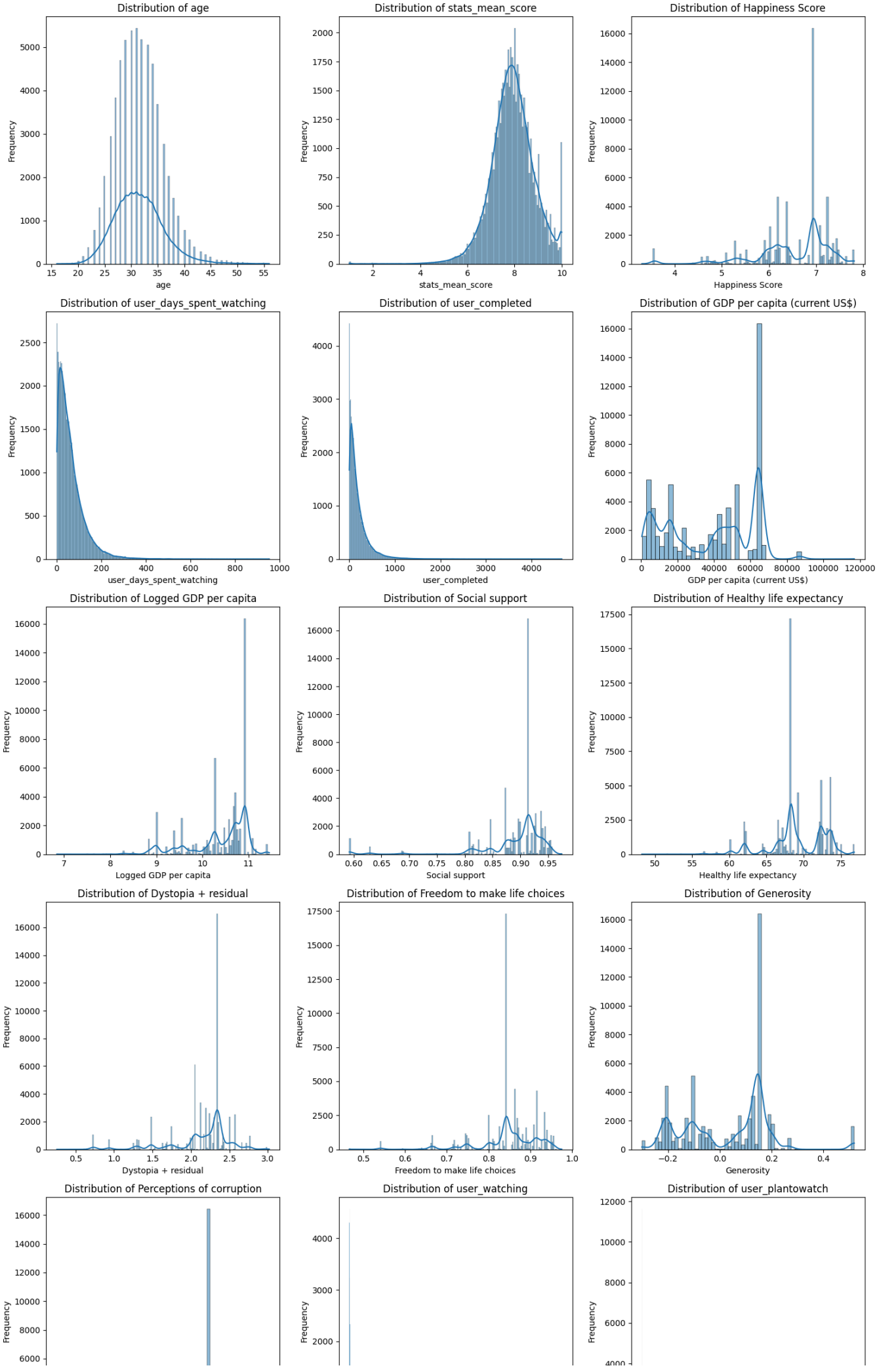
```
num_features = len(numerical_features)
num_cols = 3
num_rows = math.ceil(num_features / num_cols)

fig, axes = plt.subplots(num_rows, num_cols, figsize=(15, 5 * num_rows))

axes = axes.flatten()

for i, feature in enumerate(numerical_features):
    sns.violinplot(x='gender', y=feature, data=user_country_stats, ax=axes[i], hue='gender', palette="Set2", legend=False)
    axes[i].set_title(f'{feature} Distribution by Gender')

for i in range(num_features, num_rows * num_cols):
    fig.delaxes(axes[i])

plt.tight_layout()
plt.show()
```

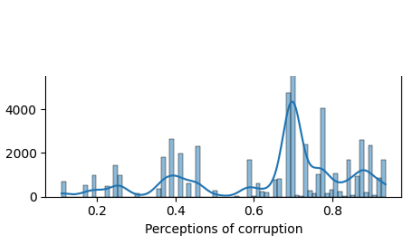

age Distribution by Gender
stats_mean_score Distribution by Gender
Happiness Score Distribution by Gender
user_days_spent_watching Distribution by Gender
user_completed Distribution by Gender
GDP per capita (current US$) Distribution by Gender
Logged GDP per capita Distribution by Gender
Social support Distribution by Gender
Healthy life expectancy Distribution by Gender
Dystopia + residual Distribution by Gender
Freedom to make life choices Distribution by Gender
Generosity Distribution by Gender
Perceptions of corruption Distribution by Gender
user_watching Distribution by Gender
user_plantowatch Distribution by Gender
gender
Female
Male

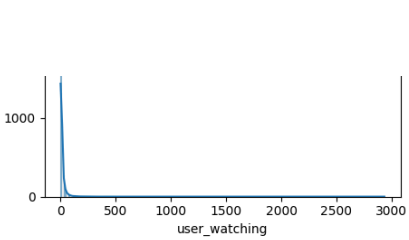

0.2
Female
Male
gender
500
0
Female
Male
gender
2000
0
Female
Male
gender

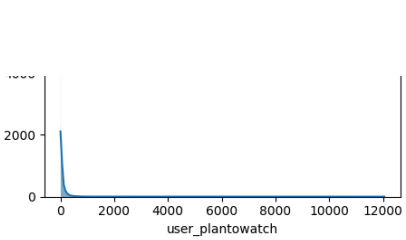
This code block focuses on visualizing the distribution of several numerical features from the user_country_stats DataFrame using histograms.

```
num_rows = 5
num_cols = 3

fig, axes = plt.subplots(num_rows, num_cols, figsize=(15, 25))

axes = axes.flatten()

for i, feature in enumerate(numerical_features):
    sns.histplot(user_country_stats[feature], kde=True, ax=axes[i])
    axes[i].set_title(f'Distribution of {feature}')
    axes[i].set_xlabel(feature)
    axes[i].set_ylabel('Frequency')

plt.tight_layout()

plt.show()
```


Distribution of age
Frequency
age
Distribution of stats_mean_score
Frequency
stats_mean_score
Distribution of Happiness Score
Frequency
Happiness Score
Distribution of user_days_spent_watching
Frequency
user_days_spent_watching
Distribution of user_completed
Frequency
user_completed
Distribution of GDP per capita (current US$)
Frequency
GDP per capita (current US$)
Distribution of Logged GDP per capita
Frequency
Logged GDP per capita
Distribution of Social support
Frequency
Social support
Distribution of Healthy life expectancy
Frequency
Healthy life expectancy
Distribution of Dystopia + residual
Frequency
Dystopia + residual
Distribution of Freedom to make life choices
Frequency
Freedom to make life choices
Distribution of Generosity
Frequency
Generosity
Distribution of Perceptions of corruption
Frequency
Distribution of user_watching
Frequency
Distribution of user_plantowatch
Frequency

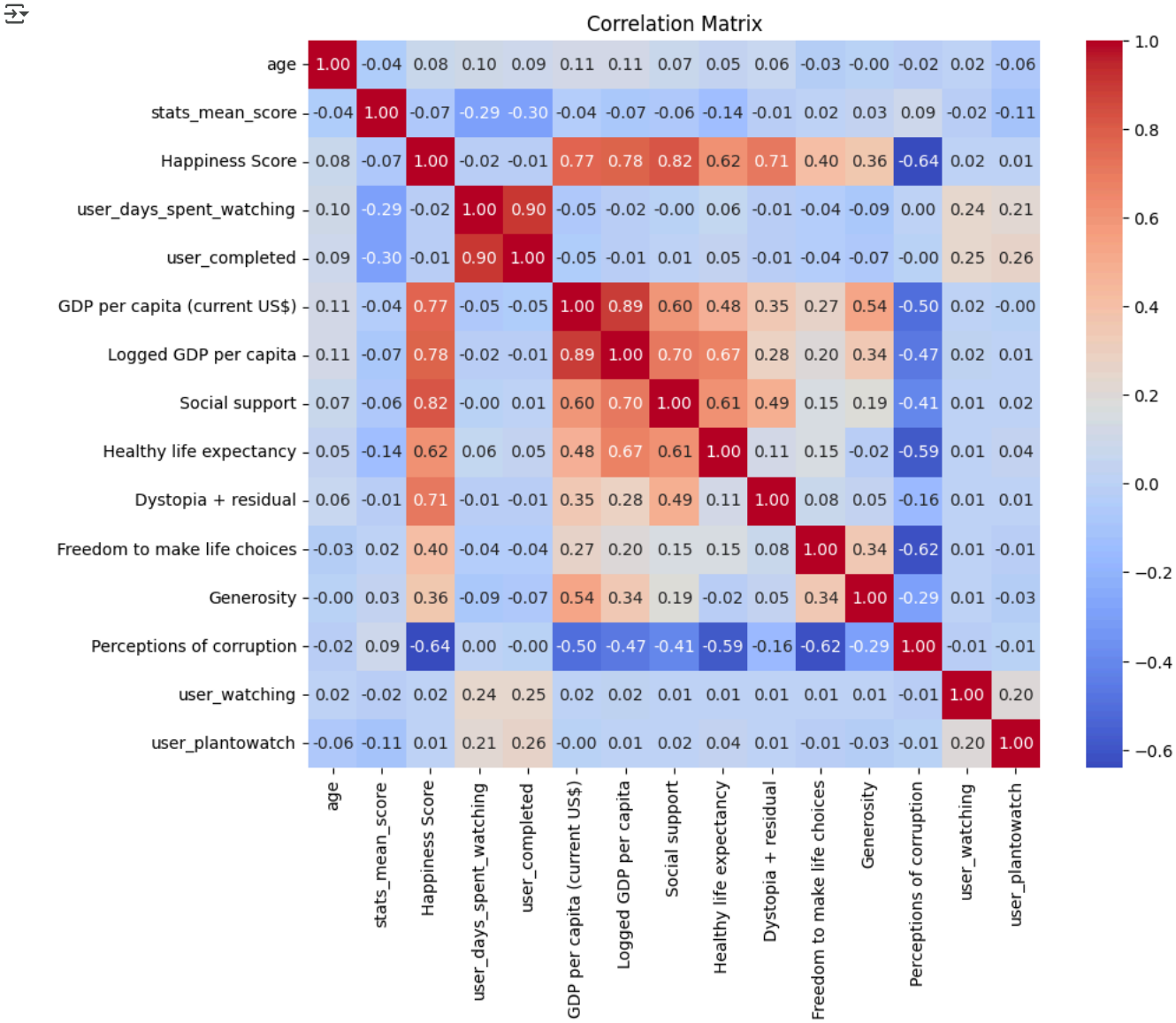

4000
2000
0
0.2
0.4
0.6
0.8
Perceptions of corruption


1000
0
0
500
1000
1500
2000
2500
3000
user_watching


2000
0
0
2000
4000
6000
8000
10000
12000
user_plantowatch

This code block calculates and visualizes the correlation matrix for the numerical features in the user_country_stats DataFrame.

```
correlation_matrix = user_country_stats[numerical_features].corr()

plt.figure(figsize=(10, 8))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', fmt=".2f")
plt.title('Correlation Matrix')
plt.show()
```



Hypothesis Tests

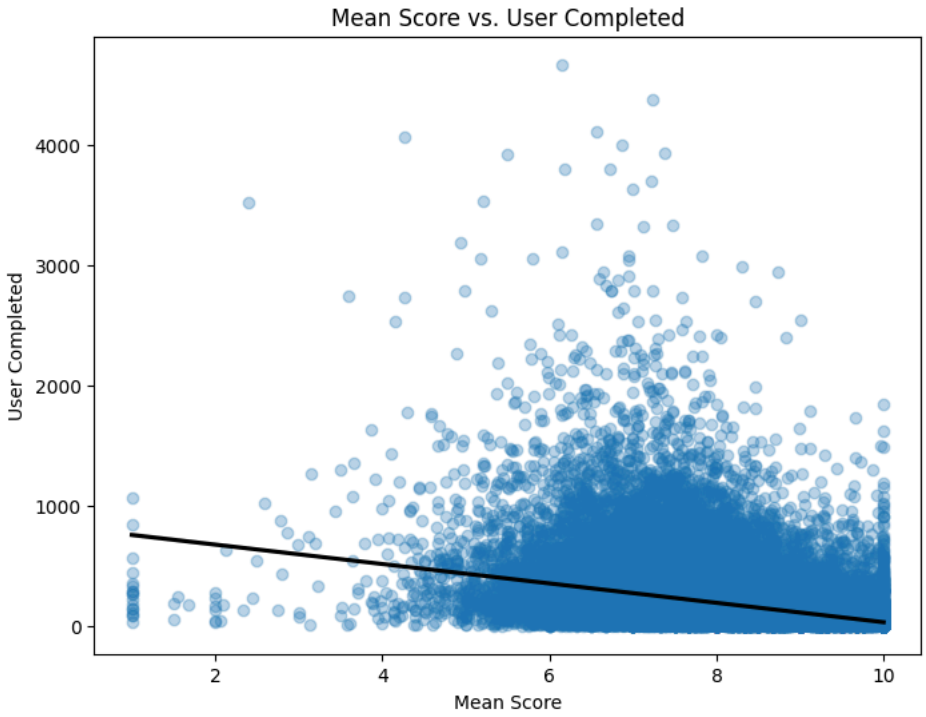
The threshold for the hypotheses is α = 0.05. If the p-value is less than or equal to α, we reject the null hypothesis and conclude that the variables are likely dependent. If the p-value is greater than α, we fail to reject the null hypothesis and conclude that the variables are likely independent.

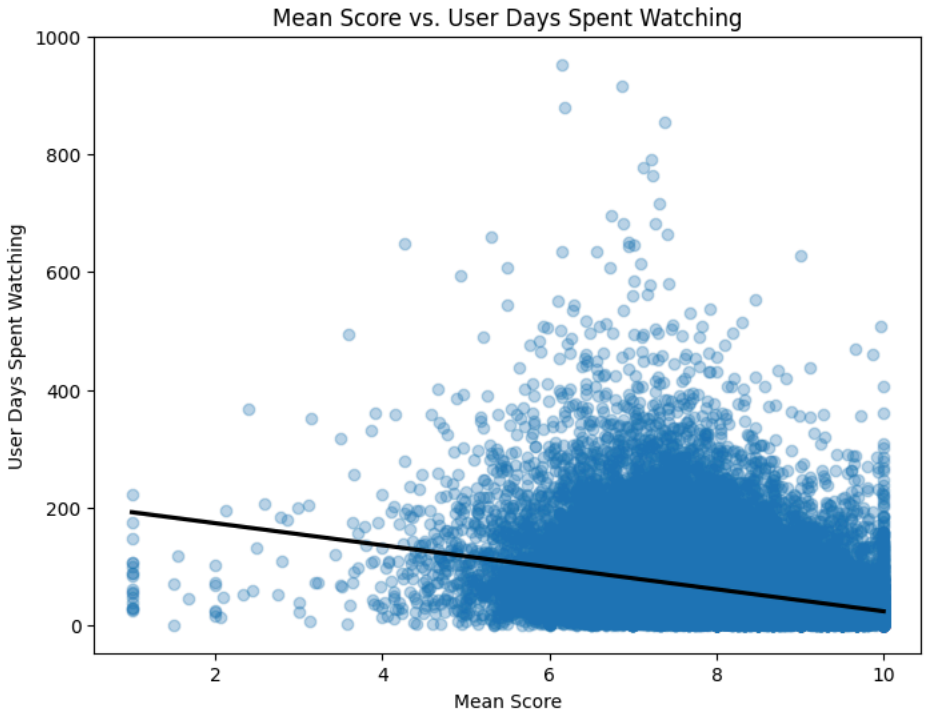
H1: Users with a higher 'Mean Score' tend to have higher engagement metrics, such as a larger number of 'User Completed' anime and more 'User Days Spent Watching', suggesting satisfaction leads to deeper engagement

```
correlation_with_completed = user_country_stats['stats_mean_score'].corr(user_country_stats['user_completed'])
correlation_with_days_spent = user_country_stats['stats_mean_score'].corr(user_country_stats['user_days_spent_watching'])

print(f"Correlation between Mean Score and User Completed: {correlation_with_completed}")
print(f"Correlation between Mean Score and User Days Spent Watching: {correlation_with_days_spent}")
```

```
Correlation between Mean Score and User Completed: -0.2983295936486229
Correlation between Mean Score and User Days Spent Watching: -0.28560726935590597
```

```
plt.figure(figsize=(8, 6))
sns.regplot(x='stats_mean_score', y='user_completed', data=user_country_stats, scatter_kws={'alpha':0.3}, line_kws={'color': 'black
plt.title('Mean Score vs. User Completed')
plt.xlabel('Mean Score')
plt.ylabel('User Completed')
plt.show()

plt.figure(figsize=(8, 6))
sns.regplot(x='stats_mean_score', y='user_days_spent_watching', data=user_country_stats, scatter_kws={'alpha':0.3}, line_kws={'colo
plt.title('Mean Score vs. User Days Spent Watching')
plt.xlabel('Mean Score')
plt.ylabel('User Days Spent Watching')
plt.show()
```






```
tmp = user_country_stats.sample(n=500, random_state=42).copy()

tmp['mean_score_category'] = pd.qcut(tmp['stats_mean_score'], 3, labels=['low', 'medium', 'high'])
tmp['completed_category'] = pd.qcut(tmp['user_completed'], 3, labels=['low', 'medium', 'high'])
tmp['days_spent_category'] = pd.qcut(tmp['user_days_spent_watching'], 3, labels=['low', 'medium', 'high'])

contingency_table_completed = pd.crosstab(tmp['mean_score_category'], tmp['completed_category'])

contingency_table_days_spent = pd.crosstab(tmp['mean_score_category'], tmp['days_spent_category'])
```

```
chi2_stat_completed, p_val_completed, dof_completed, expected_completed = chi2_contingency(contingency_table_completed)

chi2_stat_days_spent, p_val_days_spent, dof_days_spent, expected_days_spent = chi2_contingency(contingency_table_days_spent)

print(f"Chi-square test for Mean Score and User Completed:")
print(f"Chi-square statistic: {chi2_stat_completed}")
print(f"P-value: {p_val_completed}")

print(f"\nChi-square test for Mean Score and User Days Spent Watching:")
print(f"Chi-square statistic: {chi2_stat_days_spent}")
print(f"P-value: {p_val_days_spent}")
```

```
Chi-square test for Mean Score and User Completed:
Chi-square statistic: 53.40157794100588
P-value: 7.022420420867169e-11

Chi-square test for Mean Score and User Days Spent Watching:
Chi-square statistic: 39.017057813558736
P-value: 6.910159230681829e-08
```

Conclusion: We reject the null hypothesis and conclude that there is a significant association

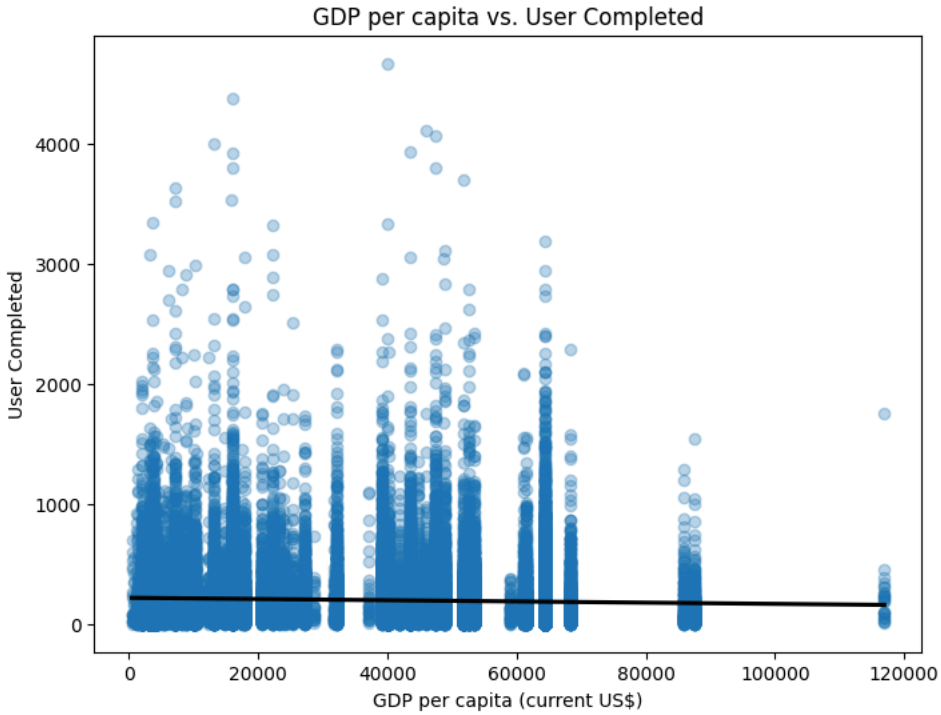
H2: Users from countries with higher 'GDP per capita (current US$)' tend to watch more anime ('User Completed', 'User Watched Episodes') and potentially have higher 'Mean Scores', possibly reflecting better access or more leisure time/income.

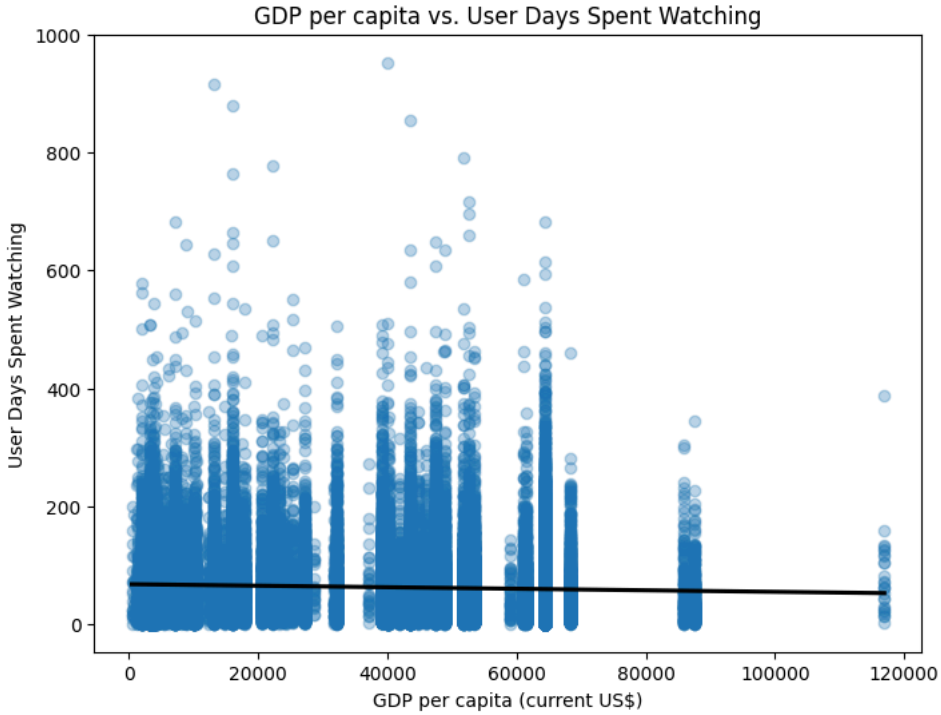
```
correlation_with_completed = user_country_stats['GDP per capita (current US$)'].corr(user_country_stats['user_completed'])
correlation_with_days_spent = user_country_stats['GDP per capita (current US$)'].corr(user_country_stats['user_days_spent_watching'])
correlation_with_mean_score = user_country_stats['GDP per capita (current US$)'].corr(user_country_stats['stats_mean_score'])

print(f"Correlation between GDP per capita and User Completed: {correlation_with_completed}")
print(f"Correlation between GDP per capita and User Days Spent Watching: {correlation_with_days_spent}")
print(f"Correlation between GDP per capita and Mean Score: {correlation_with_mean_score}")
```

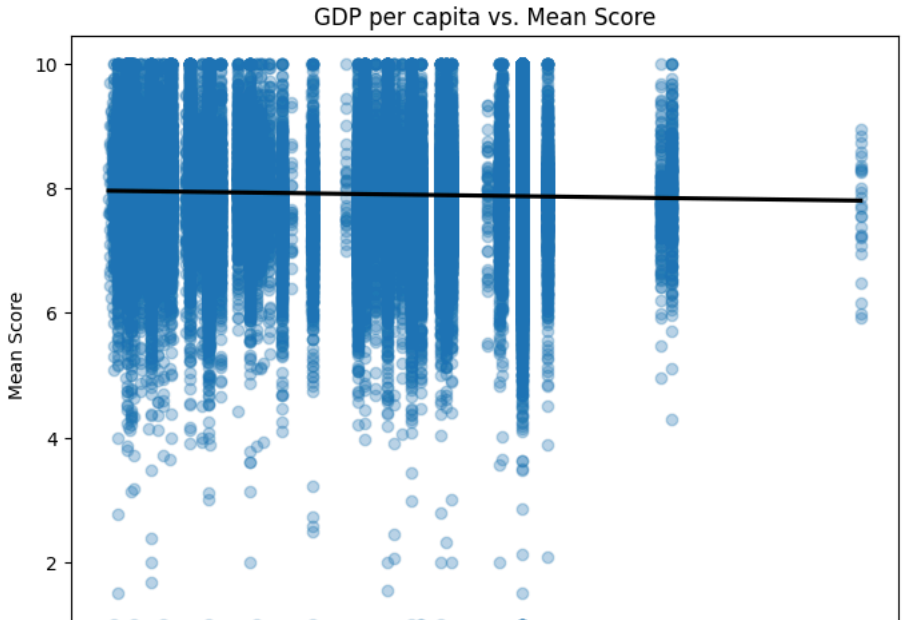
```
Correlation between GDP per capita and User Completed: -0.049227594442320624
Correlation between GDP per capita and User Days Spent Watching: -0.05220467832304506
Correlation between GDP per capita and Mean Score: -0.035823753105141014
```

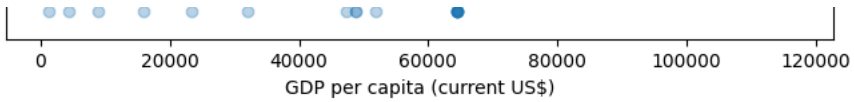
```
plt.figure(figsize=(8, 6))
sns.regplot(x='GDP per capita (current US$)', y='user_completed', data=user_country_stats, scatter_kws={'alpha':0.3}, line_kws={'co
plt.title('GDP per capita vs. User Completed')
plt.xlabel('GDP per capita (current US$)')
plt.ylabel('User Completed')
plt.show()

plt.figure(figsize=(8, 6))
sns.regplot(x='GDP per capita (current US$)', y='user_days_spent_watching', data=user_country_stats, scatter_kws={'alpha':0.3}, lin
plt.title('GDP per capita vs. User Days Spent Watching')
plt.xlabel('GDP per capita (current US$)')
plt.ylabel('User Days Spent Watching')
plt.show()

plt.figure(figsize=(8, 6))
sns.regplot(x='GDP per capita (current US$)', y='stats_mean_score', data=user_country_stats, scatter_kws={'alpha':0.3}, line_kws={'
plt.title('GDP per capita vs. Mean Score')
plt.xlabel('GDP per capita (current US$)')
plt.ylabel('Mean Score')
plt.show()
```

[4]


GDP per capita vs. User Completed
User Completed
GDP per capita (current US$)
0
1000
2000
3000
4000
0
20000
40000
60000
80000
100000
120000


GDP per capita vs. User Days Spent Watching
User Days Spent Watching
GDP per capita (current US$)
0
200
400
600
800
1000
0
20000
40000
60000
80000
100000
120000


GDP per capita vs. Mean Score
Mean Score
2
4
6
8
10




```
tmp = user_country_stats.sample(n=500, random_state=42).copy()

tmp['gdp_category'] = pd.qcut(tmp['GDP per capita (current US$)'], 3, labels=['low', 'medium', 'high'])

tmp['completed_category'] = pd.qcut(tmp['user_completed'], 3, labels=['low', 'medium', 'high'])

tmp['days_spent_category'] = pd.qcut(tmp['user_days_spent_watching'], 3, labels=['low', 'medium', 'high'])

tmp['mean_score_category'] = pd.qcut(tmp['stats_mean_score'], 3, labels=['low', 'medium', 'high'])

contingency_table_completed = pd.crosstab(tmp['gdp_category'], tmp['completed_category'])

contingency_table_days_spent = pd.crosstab(tmp['gdp_category'], tmp['days_spent_category'])

contingency_table_mean_score = pd.crosstab(tmp['gdp_category'], tmp['mean_score_category'])

chi2_stat_completed, p_val_completed, dof_completed, expected_completed = chi2_contingency(contingency_table_completed)

chi2_stat_days_spent, p_val_days_spent, dof_days_spent, expected_days_spent = chi2_contingency(contingency_table_days_spent)

chi2_stat_mean_score, p_val_mean_score, dof_mean_score, expected_mean_score = chi2_contingency(contingency_table_mean_score)

print(f"Chi-square test for GDP and User Completed:")
print(f"Chi-square statistic: {chi2_stat_completed}")
print(f"P-value: {p_val_completed}")

print(f"\nChi-square test for GDP and User Days Spent Watching:")
print(f"Chi-square statistic: {chi2_stat_days_spent}")
print(f"P-value: {p_val_days_spent}")

print(f"\nChi-square test for GDP and Mean Score:")
print(f"Chi-square statistic: {chi2_stat_mean_score}")
print(f"P-value: {p_val_mean_score}")
```

```
Chi-square test for GDP and User Completed:
Chi-square statistic: 5.383837618156164
P-value: 0.25013050340042614

Chi-square test for GDP and User Days Spent Watching:
Chi-square statistic: 3.1269498625861756
P-value: 0.5368099282784534

Chi-square test for GDP and Mean Score:
Chi-square statistic: 8.533135573192155
P-value: 0.07388917746679664
```

Conclusion: We fail to reject the null hypothesis and conclude that there is no significant association

H3: Higher 'Fixed broadband subscriptions', 'Mobile cellular subscriptions', or 'Secure Internet servers (per 1 million people)' correlate with higher 'User Completed' counts for users in that country.

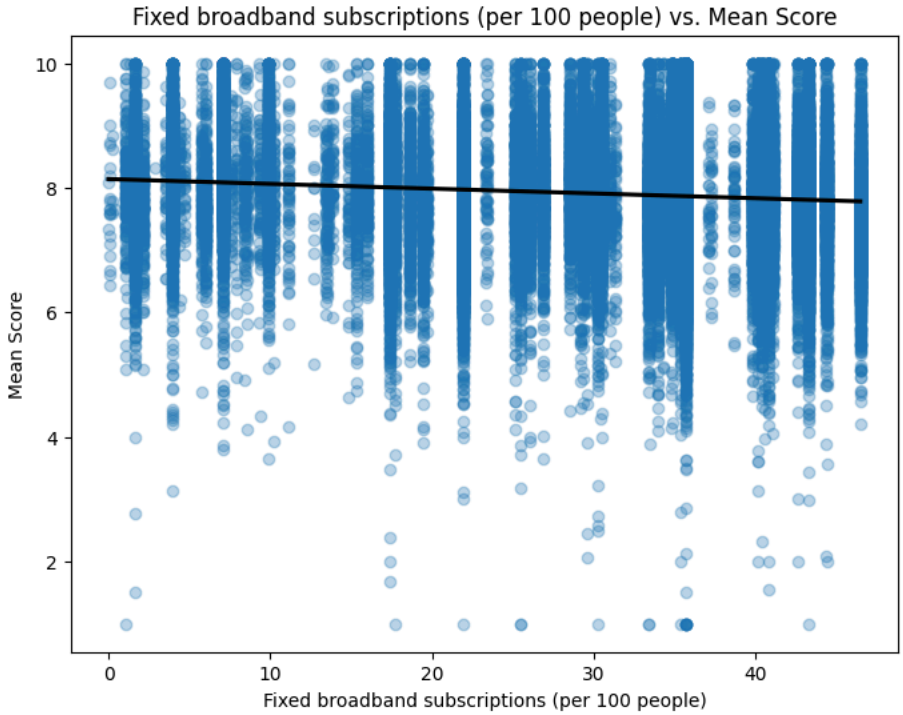
```
correlation_with_days_broadband = user_country_stats['Fixed broadband subscriptions (per 100 people)'].corr(user_country_stats['stat
correlation_with_mobile = user_country_stats['Mobile cellular subscriptions (per 100 people)'].corr(user_country_stats['stats_mean_s
correlation_with_server = user_country_stats['Secure Internet servers (per 1 million people)'].corr(user_country_stats['stats_mean_s

print(f"Correlation between Fixed broadband subscriptions and Mean Score: {correlation_with_days_broadband}")
print(f"Correlation between Mobile cellular subscriptions and Mean Score: {correlation_with_mobile}")
print(f"Correlation between Secure Internet servers and Mean Score: {correlation_with_server}")
```

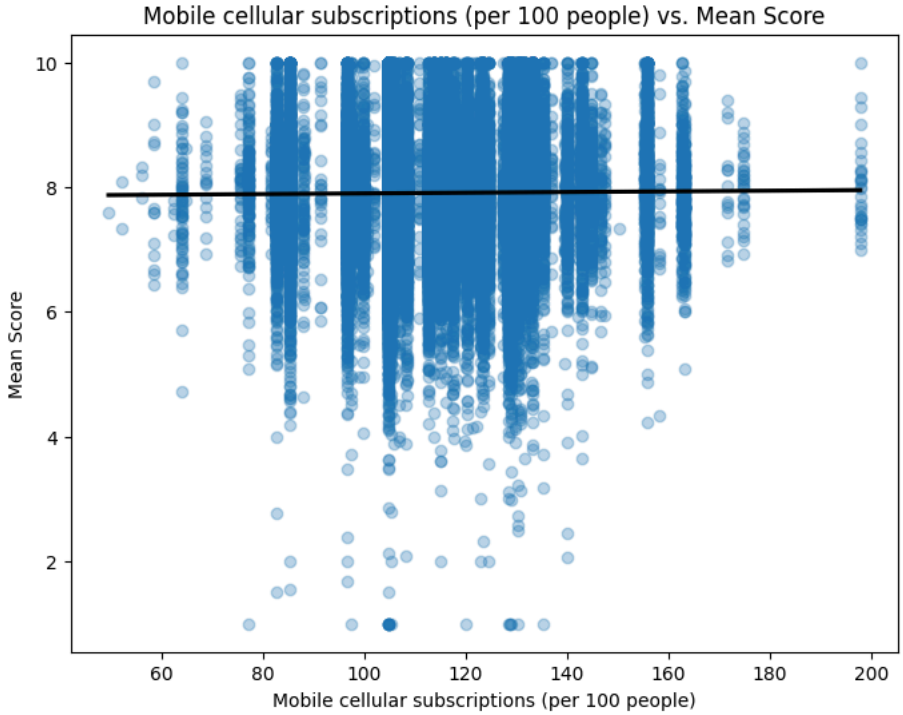
```
Correlation between Fixed broadband subscriptions and Mean Score: -0.10001294168824629
Correlation between Mobile cellular subscriptions and Mean Score: 0.009402702266438736
Correlation between Secure Internet servers and Mean Score: 0.005194191434505355
```

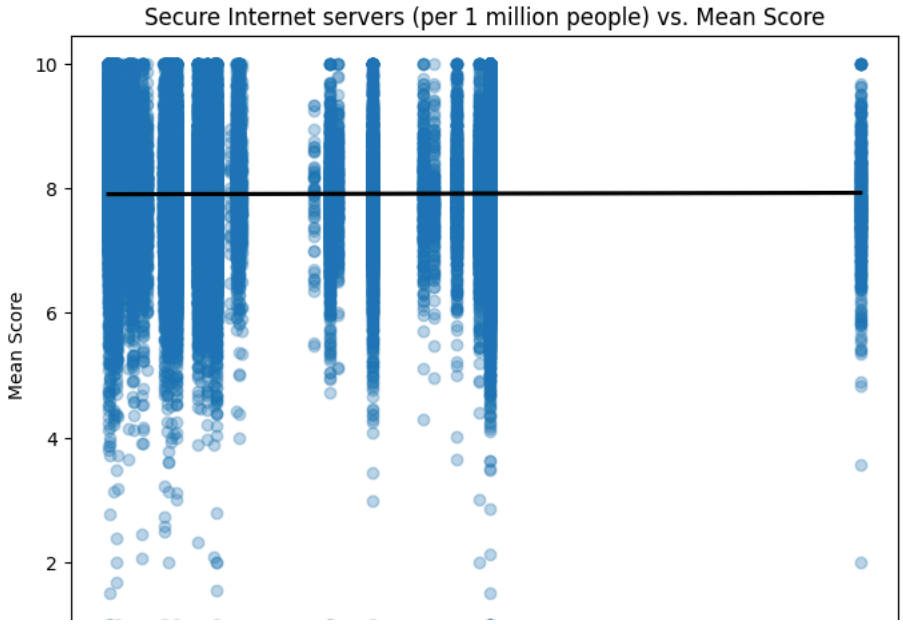
```
plt.figure(figsize=(8, 6))
sns.regplot(x='Fixed broadband subscriptions (per 100 people)', y='stats_mean_score', data=user_country_stats, scatter_kws={'alpha'
plt.title('Fixed broadband subscriptions (per 100 people) vs. Mean Score')
plt.xlabel('Fixed broadband subscriptions (per 100 people)')
plt.ylabel('Mean Score')
plt.show()

plt.figure(figsize=(8, 6))
sns.regplot(x='Mobile cellular subscriptions (per 100 people)', y='stats_mean_score', data=user_country_stats, scatter_kws={'alpha'
plt.title('Mobile cellular subscriptions (per 100 people) vs. Mean Score')
plt.xlabel('Mobile cellular subscriptions (per 100 people)')
```

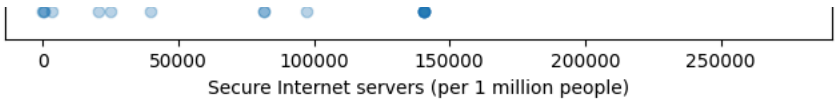
```
plt.ylabel('Mean Score')
plt.show()

plt.figure(figsize=(8, 6))
sns.regplot(x='Secure Internet servers (per 1 million people)', y='stats_mean_score', data=user_country_stats, scatter_kws={'alpha'
plt.title('Secure Internet servers (per 1 million people) vs. Mean Score')
plt.xlabel('Secure Internet servers (per 1 million people)')
plt.ylabel('Mean Score')
plt.show()
```

[4]


Fixed broadband subscriptions (per 100 people) vs. Mean Score
Mean Score
Fixed broadband subscriptions (per 100 people)
10
8
6
4
2
0
10
20
30
40


Mobile cellular subscriptions (per 100 people) vs. Mean Score
Mean Score
Mobile cellular subscriptions (per 100 people)
10
8
6
4
2
60
80
100
120
140
160
180
200


Secure Internet servers (per 1 million people) vs. Mean Score
Mean Score
10
8
6
4
2




```
tmp = user_country_stats.sample(n=500, random_state=42).copy()

tmp['broadband_category'] = pd.qcut(tmp['Fixed broadband subscriptions (per 100 people)'], 3, labels=['low', 'medium', 'high'])
tmp['mobile_category'] = pd.qcut(tmp['Mobile cellular subscriptions (per 100 people)'], 3, labels=['low', 'medium', 'high'])
tmp['server_category'] = pd.qcut(tmp['Secure Internet servers (per 1 million people)'], 3, labels=['low', 'medium', 'high'])
tmp['mean_score_category'] = pd.qcut(tmp['stats_mean_score'], 3, labels=['low', 'medium', 'high'])

contingency_table_broadband = pd.crosstab(tmp['broadband_category'], tmp['mean_score_category'])

contingency_table_mobile = pd.crosstab(tmp['mobile_category'], tmp['mean_score_category'])

contingency_table_server = pd.crosstab(tmp['server_category'], tmp['mean_score_category'])

chi2_stat_broadband, p_val_broadband, dof_broadband, expected_broadband = chi2_contingency(contingency_table_broadband)

chi2_stat_mobile, p_val_mobile, dof_mobile, expected_mobile = chi2_contingency(contingency_table_mobile)

chi2_stat_server, p_val_server, dof_server, expected_server = chi2_contingency(contingency_table_server)

print(f"Chi-square test for Fixed broadband subscriptions (per 100 people) and Mean Score:")
print(f"Chi-square statistic: {chi2_stat_broadband}")
print(f"P-value: {p_val_broadband}")

print(f"\nChi-square test for Mobile cellular subscriptions (per 100 people) and Mean Score:")
print(f"Chi-square statistic: {chi2_stat_mobile}")
print(f"P-value: {p_val_mobile}")

print(f"\nChi-square test for Secure Internet servers (per 1 million people) and Mean Score:")
print(f"Chi-square statistic: {chi2_stat_server}")
print(f"P-value: {p_val_server}")
```

```
Chi-square test for Fixed broadband subscriptions (per 100 people) and Mean Score:
Chi-square statistic: 9.162235472910053
P-value: 0.05716984705773421

Chi-square test for Mobile cellular subscriptions (per 100 people) and Mean Score:
Chi-square statistic: 3.653328246357328
P-value: 0.45495040686979304

Chi-square test for Secure Internet servers (per 1 million people) and Mean Score:
Chi-square statistic: 9.243525206249089
P-value: 0.055292535695738776
```

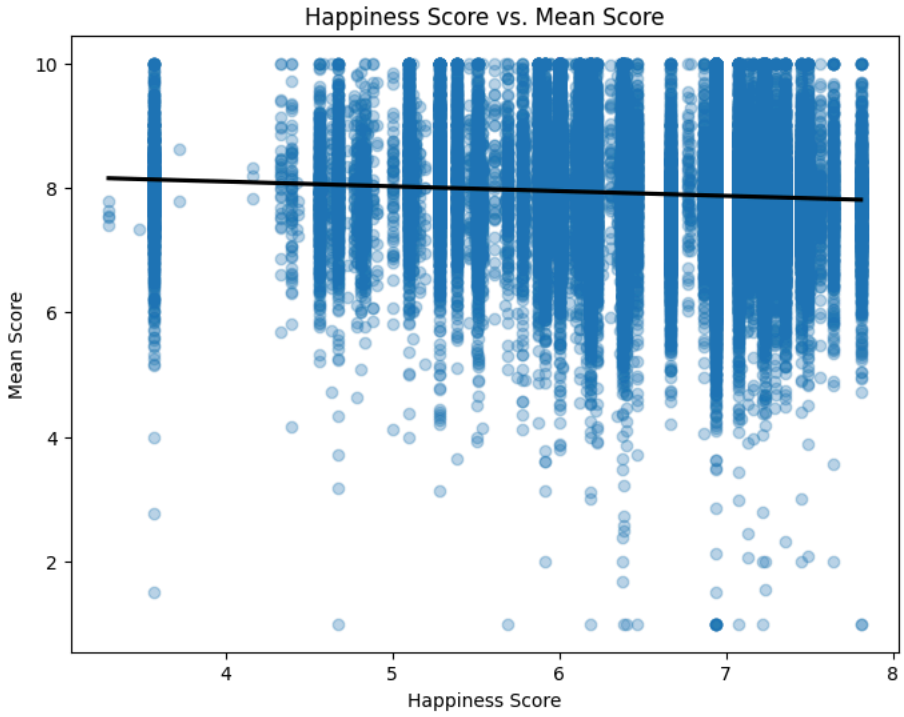
Conclusion: We fail to reject the null hypothesis and conclude that there is no significant association

H4: Higher national happiness correlates with higher average 'User Mean Score' from users in that country.

```
correlation = user_country_stats['Happiness Score'].corr(user_country_stats['stats_mean_score'])

print(f"Correlation between 'Happiness Score and Mean Score: {correlation}")
```

```
Correlation between 'Happiness Score and Mean Score: -0.06687589793370768
```

```
plt.figure(figsize=(8, 6))
sns.regplot(x='Happiness Score', y='stats_mean_score', data=user_country_stats, scatter_kws={'alpha':0.3}, line_kws={'color': 'blacl
plt.title('Happiness Score vs. Mean Score')
plt.xlabel('Happiness Score')
plt.ylabel('Mean Score')
plt.show()
```




```
tmp = user_country_stats.sample(n=500, random_state=42).copy()

tmp['happiness_category'] = pd.qcut(tmp['Happiness Score'], 3, labels=['low', 'medium', 'high'])
tmp['mean_score_category'] = pd.qcut(tmp['stats_mean_score'], 3, labels=['low', 'medium', 'high'])

contingency_table = pd.crosstab(tmp['happiness_category'], tmp['mean_score_category'])

chi2_stat, p_val, dof, expected = chi2_contingency(contingency_table)

print(f"Chi-square test for Happiness Score and Mean Score:")
print(f"Chi-square statistic: {chi2_stat}")
print(f"P-value: {p_val}")
```

```
Chi-square test for Happiness Score and Mean Score:
Chi-square statistic: 5.702573456003083
P-value: 0.22248859870279736
```

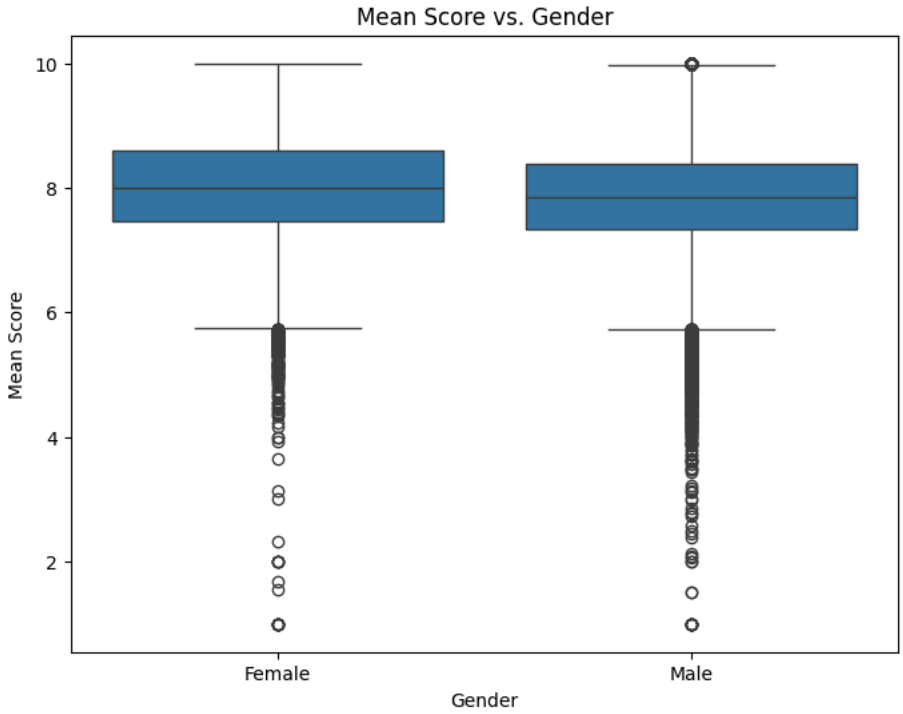
Conclusion: We fail to reject the null hypothesis and conclude that there is no significant association

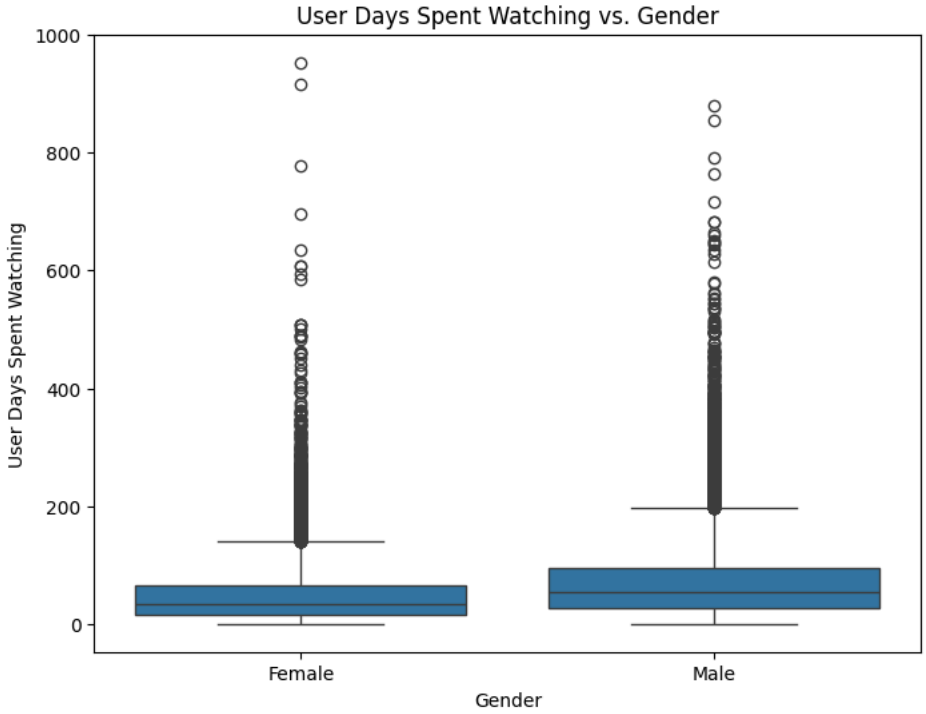
H5: Male and female users show differences in their anime watching habits, such as the total time spent watching anime or the average scores they give to anime.

```
correlation_with_days_spent_watching = user_country_stats['user_days_spent_watching'].corr(user_country_stats['gender_numeric'])
correlation_with_mean_score = user_country_stats['stats_mean_score'].corr(user_country_stats['gender_numeric'])

print(f"Correlation between Gender and User Days Spent Watching: {correlation_with_days_spent_watching}")
print(f"Correlation between Gender and Mean Score: {correlation_with_mean_score}")
```

```
Correlation between Gender and User Days Spent Watching: -0.17631070389647777
Correlation between Gender and Mean Score: 0.09952580830135989
```

```
plt.figure(figsize=(8, 6))
sns.boxplot(x='gender', y='stats_mean_score', data=user_country_stats)
plt.title('Mean Score vs. Gender')
plt.xlabel('Gender')
plt.ylabel('Mean Score')
plt.show()

plt.figure(figsize=(8, 6))
sns.boxplot(x='gender', y='user_days_spent_watching', data=user_country_stats)
plt.title('User Days Spent Watching vs. Gender')
plt.xlabel('Gender')
plt.ylabel('User Days Spent Watching')
plt.show()
```






```
tmp = user_country_stats.sample(n=500, random_state=42).copy()

tmp['mean_score_category'] = pd.qcut(tmp['stats_mean_score'], 3, labels=['low', 'medium', 'high'])
tmp['days_spent_watching_category'] = pd.qcut(tmp['user_days_spent_watching'], 3, labels=['low', 'medium', 'high'])

contingency_table_mean = pd.crosstab(tmp['gender'], tmp['mean_score_category'])
contingency_table_days_watching = pd.crosstab(tmp['gender'], tmp['days_spent_watching_category'])

chi2_stat_mean, p_val_mean, dof_mean, expected_mean = chi2_contingency(contingency_table_mean)

chi2_stat_days_watching, p_val_days_watching, dof_days_watching, expected_days_watching = chi2_contingency(contingency_table_days_wa

print(f"Chi-square test for Mean Score and Gender:")
print(f"Chi-square statistic: {chi2_stat_mean}")
print(f"P-value: {p_val_mean}")

print(f"\nChi-square test for User Days Spent Watching and Gender:")
print(f"Chi-square statistic: {chi2_stat_days_watching}")
print(f"P-value: {p_val_days_watching}")
```

```
Chi-square test for Mean Score and Gender:
Chi-square statistic: 0.8739665445468503
P-value: 0.6459822371344817
```

```
Chi-square test for User Days Spent Watching and Gender:
Chi-square statistic: 12.10587008445579
P-value: 0.002350951727874106
```

Conclusion: We fail to reject the null hypothesis for average score and conclude that there is no significant association between gender and the average score users give to anime. However, we reject the null hypothesis for user days spent watching and conclude that there is a significant association between gender and the total time spent watching anime.

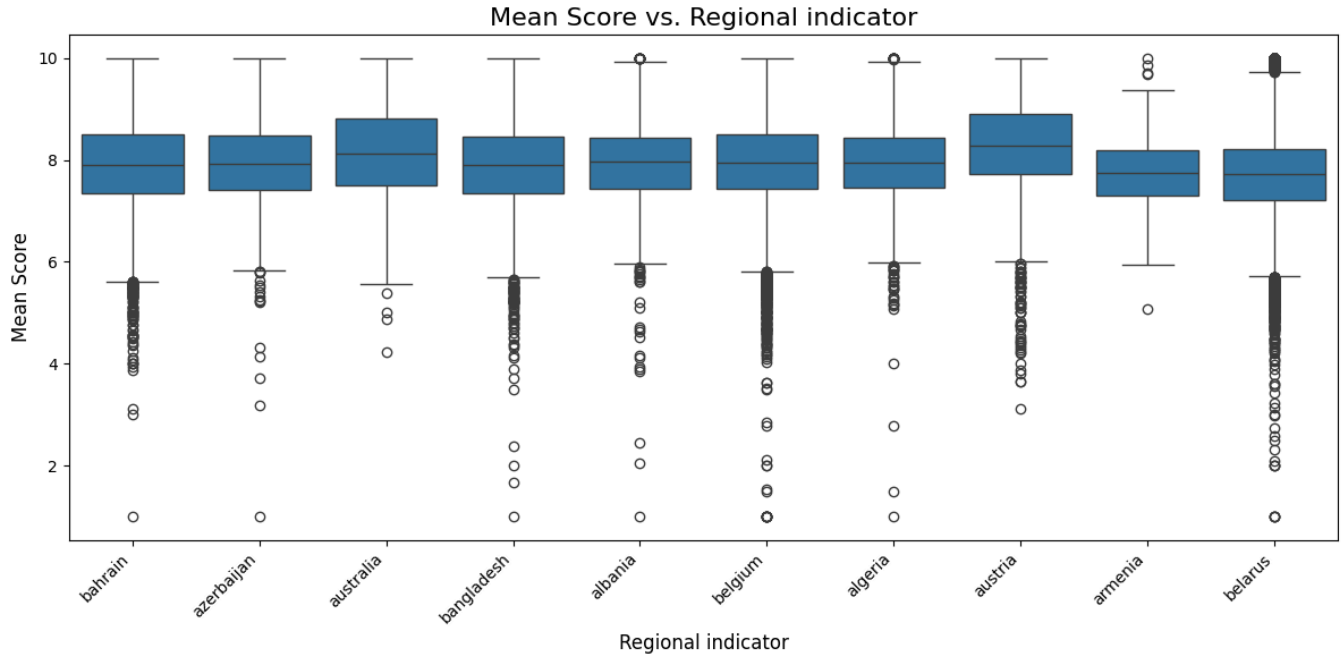
H6: Users from different regional indicators (e.g., Western Europe, Sub-Saharan Africa) exhibit distinct anime watching behaviors, such as mean score, watching time, or completed anime.

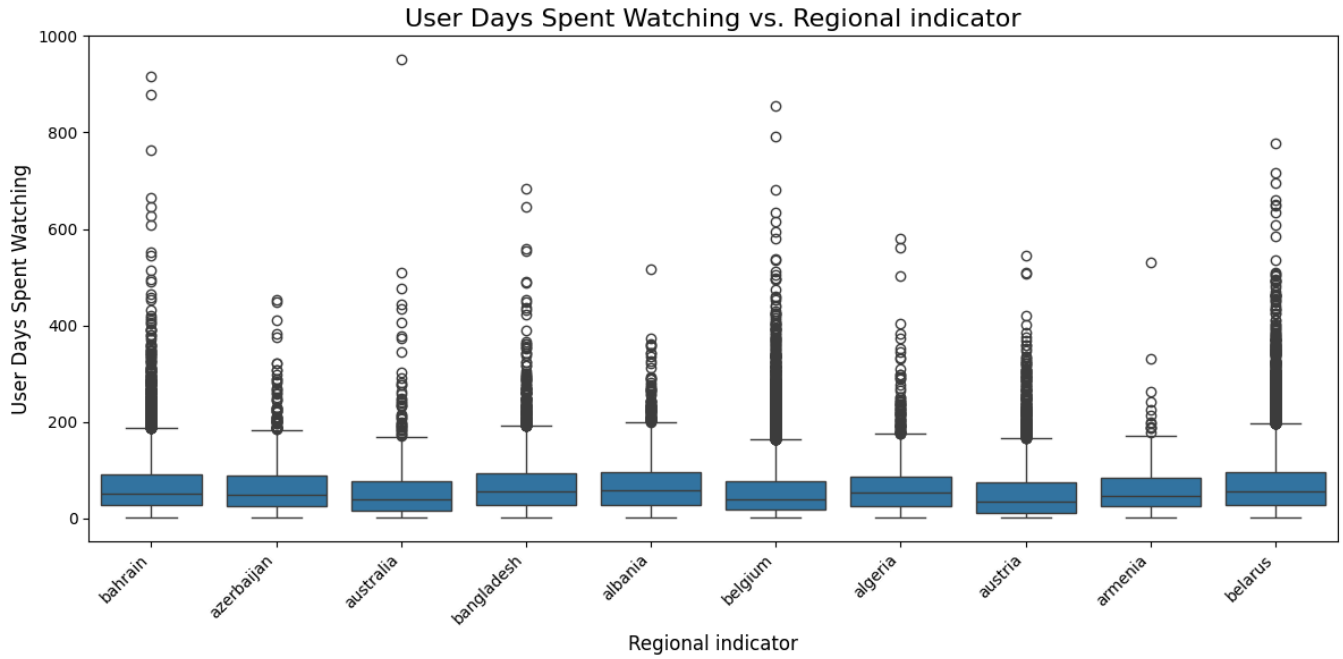
```
correlation_with_days_spent_watching = user_country_stats['user_days_spent_watching'].corr(user_country_stats['Regional indicator'])
correlation_with_mean_score = user_country_stats['stats_mean_score'].corr(user_country_stats['Regional indicator'])

print(f"Correlation between Regional indicator and User Days Spent Watching: {correlation_with_days_spent_watching}")
print(f"Correlation between Regional indicator and Mean Score: {correlation_with_mean_score}")
```

```
Correlation between Regional indicator and User Days Spent Watching: 0.011590359946842354
Correlation between Regional indicator and Mean Score: -0.0521957966286115
```

```
original_labels = label_encoder.inverse_transform(user_country_stats['Regional indicator'].unique())

plt.figure(figsize=(12, 6))
sns.boxplot(x='Regional indicator', y='stats_mean_score', data=user_country_stats)
plt.title('Mean Score vs. Regional indicator', fontsize=16)
plt.xlabel('Regional indicator', fontsize=12)
plt.ylabel('Mean Score', fontsize=12)
plt.xticks(ticks=np.arange(len(original_labels)), labels=original_labels, rotation=45, ha='right')
plt.tight_layout()
plt.show()

plt.figure(figsize=(12, 6))
sns.boxplot(x='Regional indicator', y='user_days_spent_watching', data=user_country_stats)
plt.title('User Days Spent Watching vs. Regional indicator', fontsize=16)
plt.xlabel('Regional indicator', fontsize=12)
plt.ylabel('User Days Spent Watching', fontsize=12)
plt.xticks(ticks=np.arange(len(original_labels)), labels=original_labels, rotation=45, ha='right')
plt.tight_layout()
plt.show()
```

[+]






```
tmp = user_country_stats.sample(n=500, random_state=42).copy()

tmp['mean_score_category'] = pd.qcut(tmp['stats_mean_score'], 3, labels=['low', 'medium', 'high'])
tmp['days_spent_watching_category'] = pd.qcut(tmp['user_days_spent_watching'], 3, labels=['low', 'medium', 'high'])

contingency_table_mean = pd.crosstab(tmp['Regional indicator'], tmp['mean_score_category'])
contingency_table_days_watching = pd.crosstab(tmp['Regional indicator'], tmp['days_spent_watching_category'])

chi2_stat_mean, p_val_mean, dof_mean, expected_mean = chi2_contingency(contingency_table_mean)

chi2_stat_days_watching, p_val_days_watching, dof_days_watching, expected_days_watching = chi2_contingency(contingency_table_days_wa

print(f"Chi-square test for Mean Score and Regional indicator:")
print(f"Chi-square statistic: {chi2_stat_mean}")
print(f"P-value: {p_val_mean}")

print(f"\nChi-square test for User Days Spent Watching and Regional indicator:")
```

```
print(f"Chi-square statistic: {chi2_stat_days_watching}")
print(f"P-value: {p_val_days_watching}")
```

```
Chi-square test for Mean Score and Regional indicator:
Chi-square statistic: 30.314558157556036
P-value: 0.03449826829611721

Chi-square test for User Days Spent Watching and Regional indicator:
Chi-square statistic: 39.6734092927294
P-value: 0.0023117573694563078
```

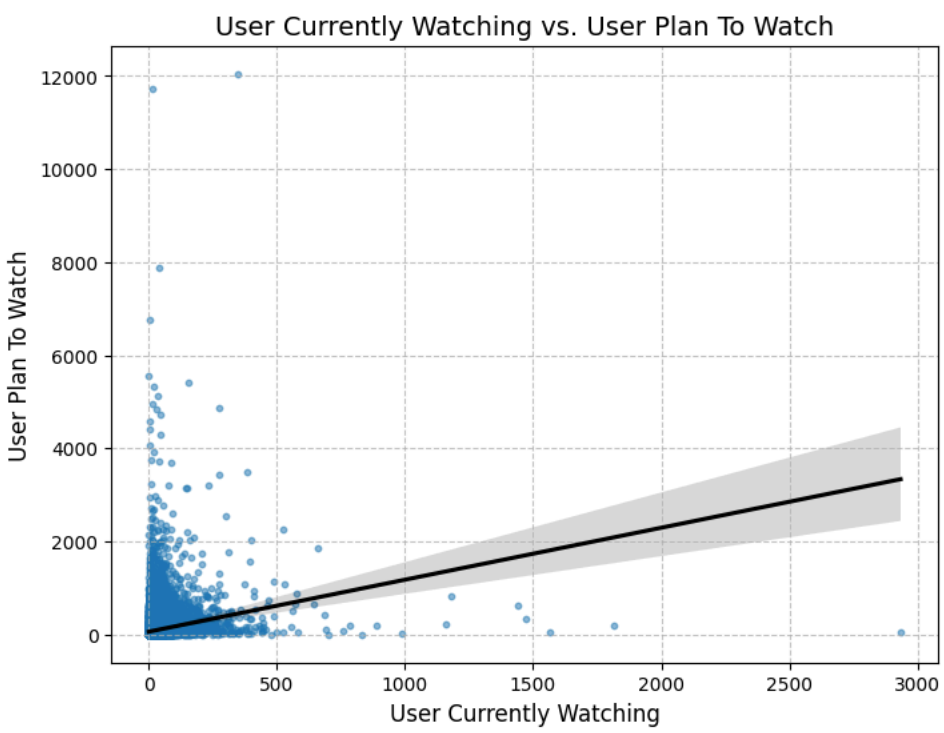
Conclusion: We reject the null hypothesis and conclude that there is a significant association

H7: There's a positive correlation between the number of anime a user is currently watching and the number they plan to watch, suggesting active viewers tend to plan future viewing.

```
correlation = user_country_stats['user_watching'].corr(user_country_stats['user_plantowatch'])

print(f"Correlation between User Watching and User Plan to Watch: {correlation}")
```

```
Correlation between User Watching and User Plan to Watch: 0.20253436731703003
```

```
plt.figure(figsize=(8, 6))
sns.regplot(x='user_watching', y='user_plantowatch', data=user_country_stats, scatter_kws={'alpha':0.5, 's': 10}, line_kws={'color'
plt.title('User Currently Watching vs. User Plan To Watch', fontsize=14)
plt.xlabel('User Currently Watching', fontsize=12)
plt.ylabel('User Plan To Watch', fontsize=12)
plt.grid(True, linestyle='--', alpha=0.7)
plt.show()
```




```
tmp = user_country_stats.sample(n=500, random_state=42).copy()

tmp['user_watching_category'] = pd.qcut(tmp['user_watching'], 3, labels=['low', 'medium', 'high'])
tmp['user_plantowatch_category'] = pd.qcut(tmp['user_plantowatch'], 3, labels=['low', 'medium', 'high'])

contingency_table = pd.crosstab(tmp['user_watching_category'], tmp['user_plantowatch_category'])

chi2_stat, p_val, dof, expected = chi2_contingency(contingency_table)

print(f"Chi-square test for User Currently Watching and User Plan To Watch:")
print(f"Chi-square statistic: {chi2_stat}")
print(f"P-value: {p_val}")
```

```
Chi-square test for User Currently Watching and User Plan To Watch:
Chi-square statistic: 76.58364062372213
P-value: 9.212465740454514e-16
```

Conclusion: We reject the null hypothesis and conclude that there is a significant association

Preperation For Machine Learning

```
1 user_country_stats.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 60924 entries, 0 to 65827
Data columns (total 68 columns):
 #   Column                                                    Non-Null Count  Dtype
---  ------                                                    --------------  -----
 0   username                                                  60924 non-null  object
 1   user_watching                                             60924 non-null  int64
 2   user_completed                                            60924 non-null  int64
 3   user_onhold                                               60924 non-null  int64
 4   user_dropped                                              60924 non-null  int64
 5   user_plantowatch                                          60924 non-null  int64
 6   user_days_spent_watching                                  60924 non-null  float64
 7   gender                                                    60924 non-null  object
 8   birth_date                                                60924 non-null  datetime64[ns]
 9   join_date                                                 60924 non-null  datetime64[ns]
 10  last_online                                               60924 non-null  datetime64[ns]
 11  stats_mean_score                                          60924 non-null  float64
 12  stats_rewatched                                           60924 non-null  float64
 13  stats_episodes                                            60924 non-null  int64
 14  age                                                       60924 non-null  int64
 15  Country                                                   60924 non-null  int64
 16  Regional indicator                                        60924 non-null  int64
 17  Happiness Score                                           60924 non-null  float64
 18  Logged GDP per capita                                     60924 non-null  float64
 19  Social support                                            60924 non-null  float64
 20  Healthy life expectancy                                   60924 non-null  float64
 21  Freedom to make life choices                              60924 non-null  float64
 22  Generosity                                                60924 non-null  float64
 23  Perceptions of corruption                                 60924 non-null  float64
 24  Ladder score in Dystopia                                  60924 non-null  float64
 25  Explained by: Log GDP per capita                          60924 non-null  float64
 26  Explained by: Social support                              60924 non-null  float64
 27  Explained by: Healthy life expectancy                     60924 non-null  float64
 28  Explained by: Freedom to make life choices                60924 non-null  float64
 29  Explained by: Generosity                                  60924 non-null  float64
 30  Explained by: Perceptions of corruption                   60924 non-null  float64
 31  Dystopia + residual                                       60924 non-null  float64
 32  High-technology exports (current US$)                     60924 non-null  float64
 33  Forest area (% of land area)                              60924 non-null  float64
 34  GDP per capita growth (annual %)                          60924 non-null  float64
 35  Urban population (% of total population)                  60924 non-null  float64
 36  Net migration                                             60924 non-null  float64
 37  GNI per capita, PPP (current international $)             60924 non-null  float64
 38  Exports of goods and services (% of GDP)                  60924 non-null  float64
 39  Agriculture, forestry, and fishing, value added (% of GDP)  60924 non-null  float64
 40  Access to electricity (% of population)                   60924 non-null  float64
 41  Population growth (annual %)                              60924 non-null  float64
 42  Renewable electricity output (% of total electricity output)  60924 non-null  float64
 43  Electric power consumption (kWh per capita)               60924 non-null  float64
 44  Renewable internal freshwater resources per capita (cubic meters)  60924 non-null  float64
 45  Population density (people per sq. km of land area)       60924 non-null  float64
 46  Mobile cellular subscriptions (per 100 people)            60924 non-null  float64
 47  Fixed broadband subscriptions (per 100 people)            60924 non-null  float64
 48  Secure Internet servers (per 1 million people)            60924 non-null  float64
 49  Mortality rate, under-5 (per 1,000 live births)           60924 non-null  float64
 50  Employment to population ratio, 15+, total (%) (modeled ILO estimate)  60924 non-null  float64
 51  Death rate, crude (per 1,000 people)                      60924 non-null  float64
 52  PM2.5 air pollution, mean annual exposure (micrograms per cubic meter)  60924 non-null  float64
```

This code block removes two columns, username and gender, from the user_country_stats DataFrame.

```
1 user_country_stats = user_country_stats.drop(labels="username", axis=1)
2 user_country_stats = user_country_stats.drop(labels="gender", axis=1)
```

This code block transforms the date columns in the user_country_stats DataFrame into a numerical representation.

```
1 user_country_stats['join_date'] = (user_country_stats['join_date'] - pd.Timestamp('1970-01-01')).dt.days
2 user_country_stats['last_online'] = (user_country_stats['last_online'] - pd.Timestamp('1970-01-01')).dt.days
3 user_country_stats['birth_date'] = (user_country_stats['birth_date'] - pd.Timestamp('1970-01-01')).dt.days
```

The purpose of this function is to evaluate the performance of a machine learning model on both training and testing datasets. It calculates several common evaluation metrics and returns them in a dictionary.

```
1 def show_scores(model,
2                 train_features,
3                 train_labels,
```

```
                test_features,
                test_labels):

    np.random.seed(42)
    train_preds = model.predict(X=train_features)
    test_preds = model.predict(X=test_features)

    train_preds_clipped = np.maximum(0, train_preds)
    test_preds_clipped = np.maximum(0, test_preds)

    scores = {"Training MAE": mean_absolute_error(y_true=train_labels,
                                                  y_pred=train_preds),
              "Test MAE": mean_absolute_error(y_true=test_labels,
                                              y_pred=test_preds),
              "Training RMSLE": root_mean_squared_log_error(y_true=train_labels,
                                                            y_pred=train_preds_clipped),
              "Test RMSLE": root_mean_squared_log_error(y_true=test_labels,
                                                        y_pred=test_preds_clipped),
              "Training R^2": model.score(X=train_features,
                                          y=train_labels),
              "Test R^2": model.score(X=test_features,
                                      y=test_labels)}
    return scores
```

Machine Learning Model (Mean Score)

This code block prepares the sample data for training a machine learning model to predict the 'stats_mean_score'.

```
sample = user_country_stats.sample(n=1000, random_state=42)

x_sample = sample.drop(labels="stats_mean_score", axis=1)
y_sample = sample["stats_mean_score"]
```

This code block splits the data into training and testing sets.

```
x_sample_train, x_sample_test, y_sample_train, y_sample_test = train_test_split(x_sample, y_sample, test_size=0.2, random_state=42)
```

This code block splits the main dataset into features and the target variable, and then further divides these into training and testing sets.

```
X = user_country_stats.drop('stats_mean_score', axis=1)
y = user_country_stats['stats_mean_score']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

This code block focuses on cleaning the column names of training and testing feature sets, X_train and X_test.

```
X_train.columns = [re.sub('[^A-Za-z0-9_]+', '_', col) for col in X_train.columns]
X_test.columns = [re.sub('[^A-Za-z0-9_]+', '_', col) for col in X_test.columns]
```

This code block trains a baseline machine learning model, specifically a Random Forest Regressor, on a sample of the sample data and then evaluates its performance.

```
np.random.seed(42)

base_model_sample = RandomForestRegressor(n_jobs=-1)

base_model_sample.fit(X=x_sample_train, y=y_sample_train)

base_model_sample_scores = show_scores(model=base_model_sample,
                                       train_features=x_sample_train,
                                       train_labels=y_sample_train,
                                       test_features=x_sample_test,
                                       test_labels=y_sample_test)

print(f"Model score on {len(sample)} samples:")
base_model_sample_scores
```

```
Model score on 1000 samples:
{'Training MAE': 0.24717012500000024,
 'Test MAE': 0.6344839999999998,
 'Training RMSLE': 0.038674498687953,
 'Test RMSLE': 0.10172722274203456,
```

```
    'Training R^2': 0.8779475785877052,
    'Test R^2': -0.0063945933095461704}
```

This code block tunes the hyperparameters of a Random Forest Regressor model using a technique called Randomized Search Cross-Validation.

```
np.random.seed(42)

rf_grid = {"n_estimators": np.arange(10, 200, 10),
           "max_depth": [None, 10, 20],
           "min_samples_split": np.arange(2, 10, 1),
           "min_samples_leaf": np.arange(1, 10, 1),
           "max_features": [0.5, 1.0, "sqrt"], }

sample_rs_model = RandomizedSearchCV(estimator=RandomForestRegressor(),
                                     param_distributions=rf_grid,
                                     n_iter=20,
                                     cv=3,
                                     verbose=3)

sample_rs_model.fit(X=x_sample_train,
                    y=y_sample_train)
```

```
Fitting 3 folds for each of 20 candidates, totalling 60 fits
[CV 1/3] END max_depth=10, max_features=sqrt, min_samples_leaf=3, min_samples_split=8, n_estimators=130;, score=0.145 total time=
[CV 2/3] END max_depth=10, max_features=sqrt, min_samples_leaf=3, min_samples_split=8, n_estimators=130;, score=0.110 total time=
[CV 3/3] END max_depth=10, max_features=sqrt, min_samples_leaf=3, min_samples_split=8, n_estimators=130;, score=0.158 total time=
[CV 1/3] END max_depth=None, max_features=0.5, min_samples_leaf=6, min_samples_split=7, n_estimators=60;, score=0.124 total time=
[CV 2/3] END max_depth=None, max_features=0.5, min_samples_leaf=6, min_samples_split=7, n_estimators=60;, score=0.130 total time=
[CV 3/3] END max_depth=None, max_features=0.5, min_samples_leaf=6, min_samples_split=7, n_estimators=60;, score=0.156 total time=
[CV 1/3] END max_depth=10, max_features=0.5, min_samples_leaf=9, min_samples_split=5, n_estimators=140;, score=0.128 total time=
[CV 2/3] END max_depth=10, max_features=0.5, min_samples_leaf=9, min_samples_split=5, n_estimators=140;, score=0.134 total time=
[CV 3/3] END max_depth=10, max_features=0.5, min_samples_leaf=9, min_samples_split=5, n_estimators=140;, score=0.172 total time=
[CV 1/3] END max_depth=10, max_features=0.5, min_samples_leaf=8, min_samples_split=3, n_estimators=50;, score=0.113 total time=
[CV 2/3] END max_depth=10, max_features=0.5, min_samples_leaf=8, min_samples_split=3, n_estimators=50;, score=0.127 total time=
[CV 3/3] END max_depth=10, max_features=0.5, min_samples_leaf=8, min_samples_split=3, n_estimators=50;, score=0.173 total time=
[CV 1/3] END max_depth=20, max_features=sqrt, min_samples_leaf=7, min_samples_split=7, n_estimators=140;, score=0.144 total time=
[CV 2/3] END max_depth=20, max_features=sqrt, min_samples_leaf=7, min_samples_split=7, n_estimators=140;, score=0.128 total time=
[CV 3/3] END max_depth=20, max_features=sqrt, min_samples_leaf=7, min_samples_split=7, n_estimators=140;, score=0.134 total time=
[CV 1/3] END max_depth=20, max_features=sqrt, min_samples_leaf=3, min_samples_split=3, n_estimators=180;, score=0.144 total time=
[CV 2/3] END max_depth=20, max_features=sqrt, min_samples_leaf=3, min_samples_split=3, n_estimators=180;, score=0.132 total time=
[CV 3/3] END max_depth=20, max_features=sqrt, min_samples_leaf=3, min_samples_split=3, n_estimators=180;, score=0.155 total time=
[CV 1/3] END max_depth=10, max_features=1.0, min_samples_leaf=2, min_samples_split=7, n_estimators=160;, score=0.139 total time=
[CV 2/3] END max_depth=10, max_features=1.0, min_samples_leaf=2, min_samples_split=7, n_estimators=160;, score=0.110 total time=
[CV 3/3] END max_depth=10, max_features=1.0, min_samples_leaf=2, min_samples_split=7, n_estimators=160;, score=0.178 total time=
[CV 1/3] END max_depth=10, max_features=1.0, min_samples_leaf=6, min_samples_split=3, n_estimators=150;, score=0.127 total time=
[CV 2/3] END max_depth=10, max_features=1.0, min_samples_leaf=6, min_samples_split=3, n_estimators=150;, score=0.131 total time=
[CV 3/3] END max_depth=10, max_features=1.0, min_samples_leaf=6, min_samples_split=3, n_estimators=150;, score=0.165 total time=
[CV 1/3] END max_depth=None, max_features=0.5, min_samples_leaf=4, min_samples_split=2, n_estimators=110;, score=0.127 total time=
[CV 2/3] END max_depth=None, max_features=0.5, min_samples_leaf=4, min_samples_split=2, n_estimators=110;, score=0.122 total time=
[CV 3/3] END max_depth=None, max_features=0.5, min_samples_leaf=4, min_samples_split=2, n_estimators=110;, score=0.160 total time=
[CV 1/3] END max_depth=10, max_features=0.5, min_samples_leaf=3, min_samples_split=2, n_estimators=190;, score=0.149 total time=
[CV 2/3] END max_depth=10, max_features=0.5, min_samples_leaf=3, min_samples_split=2, n_estimators=190;, score=0.125 total time=
[CV 3/3] END max_depth=10, max_features=0.5, min_samples_leaf=3, min_samples_split=2, n_estimators=190;, score=0.172 total time=
[CV 1/3] END max_depth=10, max_features=1.0, min_samples_leaf=1, min_samples_split=7, n_estimators=120;, score=0.152 total time=
[CV 2/3] END max_depth=10, max_features=1.0, min_samples_leaf=1, min_samples_split=7, n_estimators=120;, score=0.115 total time=
[CV 3/3] END max_depth=10, max_features=1.0, min_samples_leaf=1, min_samples_split=7, n_estimators=120;, score=0.184 total time=
[CV 1/3] END max_depth=20, max_features=sqrt, min_samples_leaf=5, min_samples_split=6, n_estimators=90;, score=0.161 total time=
[CV 2/3] END max_depth=20, max_features=sqrt, min_samples_leaf=5, min_samples_split=6, n_estimators=90;, score=0.122 total time=
[CV 3/3] END max_depth=20, max_features=sqrt, min_samples_leaf=5, min_samples_split=6, n_estimators=90;, score=0.152 total time=
[CV 1/3] END max_depth=20, max_features=sqrt, min_samples_leaf=3, min_samples_split=8, n_estimators=20;, score=0.118 total time=
[CV 2/3] END max_depth=20, max_features=sqrt, min_samples_leaf=3, min_samples_split=8, n_estimators=20;, score=0.120 total time=
[CV 3/3] END max_depth=20, max_features=sqrt, min_samples_leaf=3, min_samples_split=8, n_estimators=20;, score=0.135 total time=
[CV 1/3] END max_depth=20, max_features=sqrt, min_samples_leaf=2, min_samples_split=2, n_estimators=160;, score=0.135 total time=
[CV 2/3] END max_depth=20, max_features=sqrt, min_samples_leaf=2, min_samples_split=2, n_estimators=160;, score=0.108 total time=
[CV 3/3] END max_depth=20, max_features=sqrt, min_samples_leaf=2, min_samples_split=2, n_estimators=160;, score=0.154 total time=
[CV 1/3] END max_depth=20, max_features=0.5, min_samples_leaf=1, min_samples_split=8, n_estimators=10;, score=0.105 total time=
[CV 2/3] END max_depth=20, max_features=0.5, min_samples_leaf=1, min_samples_split=8, n_estimators=10;, score=0.111 total time=
[CV 3/3] END max_depth=20, max_features=0.5, min_samples_leaf=1, min_samples_split=8, n_estimators=10;, score=0.150 total time=
[CV 1/3] END max_depth=None, max_features=1.0, min_samples_leaf=3, min_samples_split=2, n_estimators=140;, score=0.145 total time=
[CV 2/3] END max_depth=None, max_features=1.0, min_samples_leaf=3, min_samples_split=2, n_estimators=140;, score=0.128 total time=
[CV 3/3] END max_depth=None, max_features=1.0, min_samples_leaf=3, min_samples_split=2, n_estimators=140;, score=0.157 total time=
[CV 1/3] END max_depth=None, max_features=0.5, min_samples_leaf=6, min_samples_split=2, n_estimators=100;, score=0.142 total time=
[CV 2/3] END max_depth=None, max_features=0.5, min_samples_leaf=6, min_samples_split=2, n_estimators=100;, score=0.133 total time=
[CV 3/3] END max_depth=None, max_features=0.5, min_samples_leaf=6, min_samples_split=2, n_estimators=100;, score=0.164 total time=
[CV 1/3] END max_depth=20, max_features=1.0, min_samples_leaf=7, min_samples_split=7, n_estimators=10;, score=0.100 total time=
[CV 2/3] END max_depth=20, max_features=1.0, min_samples_leaf=7, min_samples_split=7, n_estimators=10;, score=0.067 total time=
[CV 3/3] END max_depth=20, max_features=1.0, min_samples_leaf=7, min_samples_split=7, n_estimators=10;, score=0.164 total time=
[CV 1/3] END max_depth=10, max_features=sqrt, min_samples_leaf=1, min_samples_split=7, n_estimators=150;, score=0.163 total time=
[CV 2/3] END max_depth=10, max_features=sqrt, min_samples_leaf=1, min_samples_split=7, n_estimators=150;, score=0.147 total time=
[CV 3/3] END max_depth=10, max_features=sqrt, min_samples_leaf=1, min_samples_split=7, n_estimators=150;, score=0.158 total time=
[CV 1/3] END max_depth=None, max_features=1.0, min_samples_leaf=8, min_samples_split=2, n_estimators=20;, score=0.100 total time=
[CV 2/3] END max_depth=None, max_features=1.0, min_samples_leaf=8, min_samples_split=2, n_estimators=20;, score=0.116 total time=
[CV 3/3] END max_depth=None, max_features=1.0, min_samples_leaf=8, min_samples_split=2, n_estimators=20;, score=0.180 total time=
```

RandomizedSearchCV

best_estimator_: RandomForestRegressor

RandomForestRegressor

```
sample_rs_model.best_params_
```

```
{'n_estimators': np.int64(150),
 'min_samples_split': np.int64(7),
 'min_samples_leaf': np.int64(1),
 'max_features': 'sqrt',
 'max_depth': 10}
```

```
sample_rs_model_scores = show_scores(model=sample_rs_model,
                                     train_features=x_sample_train,
                                     train_labels=y_sample_train,
                                     test_features=x_sample_test,
                                     test_labels=y_sample_test)
sample_rs_model_scores
```

```
{'Training MAE': 0.4282388818111595,
 'Test MAE': 0.5979108009171518,
```

```
    'Training RMSLE': 0.06718667252319614,
    'Test RMSLE': 0.08965091362916926,
    'Training R^2': 0.626504320648437,
    'Test R^2': 0.175379145651261}
```

This code block performs a Grid Search to find the best hyperparameters for a RandomForestRegressor model using the sample training data.

```
np.random.seed(42)

rf_grid = {"n_estimators": np.arange(50, 150, 50),
           "max_depth": [None, 10, 20],
           "min_samples_split": np.arange(2, 8, 2),
           "min_samples_leaf": np.arange(2, 6, 2),
           "max_features": [0.5, "sqrt"],
           "oob_score": [True, False]
}

sample_gs_model = GridSearchCV(estimator=RandomForestRegressor(n_jobs=-1),
                               param_grid=rf_grid,
                               cv=3,
                               verbose=3,
                               n_jobs=-1)

sample_gs_model.fit(X=x_sample_train,
                    y=y_sample_train)
```

```
Fitting 3 folds for each of 144 candidates, totalling 432 fits
```

GridSearchCV

best_estimator_: RandomForestRegressor

RandomForestRegressor

```
sample_gs_model.best_params_
```

```
{'max_depth': 10,
 'max_features': 0.5,
 'min_samples_leaf': np.int64(4),
 'min_samples_split': np.int64(2),
 'n_estimators': np.int64(100),
 'oob_score': False}
```

```
sample_gs_model_scores = show_scores(model=sample_rs_model,
                                     train_features=x_sample_train,
                                     train_labels=y_sample_train,
                                     test_features=x_sample_test,
                                     test_labels=y_sample_test)
sample_gs_model_scores
```

```
{'Training MAE': 0.4282388818111595,
 'Test MAE': 0.5979108009171518,
 'Training RMSLE': 0.06718667252319614,
 'Test RMSLE': 0.08965091362916926,
 'Training R^2': 0.626504320648437,
 'Test R^2': 0.175379145651261}
```

This code block sets up and runs a HalvingRandomSearchCV for a RandomForestRegressor model. The goal is to find the best combination of hyperparameters for the model efficiently.

```
np.random.seed(42)

rf_grid = {"n_estimators": np.arange(10, 200, 10),
           "max_depth": [None, 10, 20],
           "min_samples_split": np.arange(2, 10, 1),
           "min_samples_leaf": np.arange(1, 10, 1),
           "max_features": [0.5, 1.0, "sqrt"],
           "oob_score": [True, False]}

sample_hrs_model = HalvingRandomSearchCV(estimator=RandomForestRegressor(n_jobs=-1),
                                         param_distributions=rf_grid,
                                         cv=3,
                                         verbose=3,
                                         n_jobs=-1)

```

```
16 sample_hrs_model.fit(X=x_sample_train,
```

```
n_iterations: 5
n_required_iterations: 5
n_possible_iterations: 5
min_resources_: 6
max_resources_: 800
aggressive_elimination: False
factor: 3
----------
iter: 0
n_candidates: 133
n_resources: 6
Fitting 3 folds for each of 133 candidates, totalling 399 fits
/usr/local/lib/python3.11/dist-packages/sklearn/model_selection/_search.py:1108: UserWarning: One or more of the test scores are r
 nan nan nan nan nan nan nan nan nan nan nan nan nan nan nan nan nan nan
 nan nan nan nan nan nan nan nan nan nan nan nan nan nan nan nan nan nan
 nan nan nan nan nan nan nan nan nan nan nan nan nan nan nan nan nan nan
 nan nan nan nan nan nan nan nan nan nan nan nan nan nan nan nan nan nan
 nan nan nan nan nan nan nan nan nan nan nan nan nan nan nan nan nan nan
 nan nan nan nan nan nan nan nan nan nan nan nan nan nan nan nan nan nan
 nan nan nan nan nan nan nan]
  warnings.warn(
----------
iter: 1
n_candidates: 45
n_resources: 18
Fitting 3 folds for each of 45 candidates, totalling 135 fits
/usr/local/lib/python3.11/dist-packages/sklearn/model_selection/_search.py:1108: UserWarning: One or more of the test scores are r
        nan         nan         nan         nan         nan         nan
        nan         nan         nan         nan         nan         nan
        nan         nan         nan         nan         nan         nan
        nan         nan         nan         nan         nan         nan
        nan         nan         nan         nan         nan         nan
        nan         nan         nan         nan         nan         nan
        nan         nan         nan         nan         nan         nan
        nan         nan         nan         nan         nan         nan
        nan         nan         nan         nan         nan         nan
        nan         nan         nan         nan         nan         nan
        nan         nan         nan         nan         nan         nan
        nan         nan         nan         nan         nan         nan
        nan         nan         nan         nan         nan         nan
        nan         nan         nan         nan         nan         nan
        nan         nan         nan         nan         nan         nan
        nan         nan         nan         nan         nan         nan
        nan         nan         nan         nan         nan         nan
        nan         nan         nan         nan         nan         nan
        nan         nan         nan         nan         nan         nan
        nan         nan         nan         nan         nan         nan
        nan         nan         nan         nan         nan         nan
        nan -0.20411    -0.09576914 -0.25931738 -0.5518143  -0.25208502
  0.03765969 -0.55932914 -0.24057681 -0.31721553 -0.28524958 -0.32162119
 -0.3379433  -0.24594931 -0.01075845 -0.33479507 -0.13554247 -0.22992682
 -0.27114003 -0.26282533 -0.28767893 -0.24955575 -0.2819273  -0.24981084
 -0.34204913 -0.29014534 -0.31538728 -0.32878974 -0.55474134 -0.35271075
 -0.26528275 -0.30300277 -0.40410833 -0.31572008 -0.55554751 -0.23699948
 -0.21844383 -0.21980486 -0.32614462 -0.50683992 -0.35419508 -0.32124427
 -0.31712303 -0.26388989 -0.33793436 -0.12823689]
  warnings.warn(
----------
iter: 2
n_candidates: 15
n_resources: 54
Fitting 3 folds for each of 15 candidates, totalling 45 fits
/usr/local/lib/python3.11/dist-packages/sklearn/model_selection/_search.py:1108: UserWarning: One or more of the test scores are r
        nan         nan         nan         nan         nan         nan
        nan         nan         nan         nan         nan         nan
        nan         nan         nan         nan         nan         nan
        nan         nan         nan         nan         nan         nan
        nan         nan         nan         nan         nan         nan
        nan         nan         nan         nan         nan         nan
        nan         nan         nan         nan         nan         nan
        nan         nan         nan         nan         nan         nan
        nan         nan         nan         nan         nan         nan
        nan         nan         nan         nan         nan         nan
        nan         nan         nan         nan         nan         nan
        nan         nan         nan         nan         nan         nan
        nan         nan         nan         nan         nan         nan
        nan         nan         nan         nan         nan         nan
        nan         nan         nan         nan         nan         nan
        nan         nan         nan         nan         nan         nan
        nan         nan         nan         nan         nan         nan
        nan         nan         nan         nan         nan         nan
        nan         nan         nan         nan         nan         nan
        nan         nan         nan         nan         nan         nan
        nan         nan         nan         nan         nan         nan
        nan -0.20411    -0.09576914 -0.25931738 -0.5518143  -0.25208502
  0.03765969 -0.55932914 -0.24057681 -0.31721553 -0.28524958 -0.32162119
 -0.3379433  -0.24594931 -0.01075845 -0.33479507 -0.13554247 -0.22992682
 -0.27114003 -0.26282533 -0.28767893 -0.24955575 -0.2819273  -0.24981084
 -0.34204913 -0.29014534 -0.31538728 -0.32878974 -0.55474134 -0.35271075
 -0.26528275 -0.30300277 -0.40410833 -0.31572008 -0.55554751 -0.23699948
```

```
 -0.21844383 -0.21980486 -0.32614462 -0.50683992 -0.35419508 -0.32124427
 -0.31712303 -0.26388989 -0.33793436 -0.12823689 -0.09790773 -0.00820409
  0.09628789 -0.03356175 -0.08596959 -0.08644856  0.01245921 -0.11252753
 -0.14438784 -0.10807946 -0.22608813 -0.42678851 -0.23527333 -0.212915
 -0.21507066]
  warnings.warn(
----------
iter: 3
n_candidates: 5
n_resources: 162
Fitting 3 folds for each of 5 candidates, totalling 15 fits
/usr/local/lib/python3.11/dist-packages/sklearn/model_selection/_search.py:1108: UserWarning: One or more of the test scores are r
         nan         nan         nan         nan         nan         nan
         nan         nan         nan         nan         nan         nan
         nan         nan         nan         nan         nan         nan
         nan         nan         nan         nan         nan         nan
         nan         nan         nan         nan         nan         nan
         nan         nan         nan         nan         nan         nan
         nan         nan         nan         nan         nan         nan
         nan         nan         nan         nan         nan         nan
         nan         nan         nan         nan         nan         nan
         nan         nan         nan         nan         nan         nan
         nan         nan         nan         nan         nan         nan
         nan         nan         nan         nan         nan         nan
         nan         nan         nan         nan         nan         nan
         nan         nan         nan         nan         nan         nan
         nan         nan         nan         nan         nan         nan
         nan         nan         nan         nan         nan         nan
         nan         nan         nan         nan         nan         nan
         nan         nan         nan         nan         nan         nan
         nan         nan         nan         nan         nan         nan
         nan         nan         nan         nan         nan         nan
         nan         nan         nan         nan         nan         nan
         nan -0.20411    -0.09576914 -0.25931738 -0.5518143  -0.25208502
  0.03765969 -0.55932914 -0.24057681 -0.31721553 -0.28524958 -0.32162119
 -0.3379433  -0.24594931 -0.01075845 -0.33479507 -0.13554247 -0.22992682
 -0.27114003 -0.26282533 -0.28767893 -0.24955575 -0.2819273  -0.24981084
 -0.34204913 -0.29014534 -0.31538728 -0.32878974 -0.55474134 -0.35271075
 -0.26528275 -0.30300277 -0.40410833 -0.31572008 -0.55554751 -0.23699948
 -0.21844383 -0.21980486 -0.32614462 -0.50683992 -0.35419508 -0.32124427
 -0.31712303 -0.26388989 -0.33793436 -0.12823689 -0.09790773 -0.00820409
  0.09628789 -0.03356175 -0.08596959 -0.08644856  0.01245921 -0.11252753
 -0.14438784 -0.10807946 -0.22608813 -0.42678851 -0.23527333 -0.212915
 -0.21507066  0.07723623  0.09534397  0.08889325  0.08708174  0.07908154]
  warnings.warn(
----------
iter: 4
n_candidates: 2
n_resources: 486
Fitting 3 folds for each of 2 candidates, totalling 6 fits
/usr/local/lib/python3.11/dist-packages/sklearn/model_selection/_search.py:1108: UserWarning: One or more of the test scores are r
         nan         nan         nan         nan         nan         nan
         nan         nan         nan         nan         nan         nan
         nan         nan         nan         nan         nan         nan
         nan         nan         nan         nan         nan         nan
         nan         nan         nan         nan         nan         nan
         nan         nan         nan         nan         nan         nan
         nan         nan         nan         nan         nan         nan
         nan         nan         nan         nan         nan         nan
         nan         nan         nan         nan         nan         nan
         nan         nan         nan         nan         nan         nan
         nan         nan         nan         nan         nan         nan
         nan         nan         nan         nan         nan         nan
         nan         nan         nan         nan         nan         nan
         nan         nan         nan         nan         nan         nan
         nan         nan         nan         nan         nan         nan
         nan         nan         nan         nan         nan         nan
         nan         nan         nan         nan         nan         nan
         nan         nan         nan         nan         nan         nan
         nan         nan         nan         nan         nan         nan
         nan         nan         nan         nan         nan         nan
         nan         nan         nan         nan         nan         nan
         nan -0.20411    -0.09576914 -0.25931738 -0.5518143  -0.25208502
  0.03765969 -0.55932914 -0.24057681 -0.31721553 -0.28524958 -0.32162119
 -0.3379433  -0.24594931 -0.01075845 -0.33479507 -0.13554247 -0.22992682
 -0.27114003 -0.26282533 -0.28767893 -0.24955575 -0.2819273  -0.24981084
 -0.34204913 -0.29014534 -0.31538728 -0.32878974 -0.55474134 -0.35271075
 -0.26528275 -0.30300277 -0.40410833 -0.31572008 -0.55554751 -0.23699948
 -0.21844383 -0.21980486 -0.32614462 -0.50683992 -0.35419508 -0.32124427
 -0.31712303 -0.26388989 -0.33793436 -0.12823689 -0.09790773 -0.00820409
  0.09628789 -0.03356175 -0.08596959 -0.08644856  0.01245921 -0.11252753
 -0.14438784 -0.10807946 -0.22608813 -0.42678851 -0.23527333 -0.212915
 -0.21507066  0.07723623  0.09534397  0.08889325  0.08708174  0.07908154
  0.09009783  0.08803733]
  warnings.warn(
```

▸ **HalvingRandomSearchCV**

▸ **best_estimator_: RandomForestRegressor**


RandomForestRegressor

```
sample_hrs_model.best_params_
```

```
{'oob_score': False,
 'n_estimators': np.int64(90),
 'min_samples_split': np.int64(9),
 'min_samples_leaf': np.int64(7),
 'max_features': 'sqrt',
 'max_depth': 10}
```

```
sample_hrs_model_scores = show_scores(model=sample_hrs_model,
                                      train_features=x_sample_train,
                                      train_labels=y_sample_train,
                                      test_features=x_sample_test,
                                      test_labels=y_sample_test)
sample_hrs_model_scores
```

```
{'Training MAE': 0.5158075035008145,
 'Test MAE': 0.5807193498483294,
 'Training RMSLE': 0.08305704803787987,
 'Test RMSLE': 0.08779701839845318,
 'Training R^2': 0.431613061749733,
 'Test R^2': 0.21029860671427314}
```

This code block trains a baseline machine learning model, specifically a Random Forest Regressor, on the full training dataset and then evaluates its performance.

```
base_model = RandomForestRegressor(n_jobs=-1)

base_model.fit(X=X_train, y=y_train)

base_model_scores = show_scores(model=base_model,
                                train_features=X_train,
                                train_labels=y_train,
                                test_features=X_test,
                                test_labels=y_test)
base_model_scores
```

```
{'Training MAE': 0.2205965304991896,
 'Test MAE': 0.5887088469429627,
 'Training RMSLE': 0.038302907056537425,
 'Test RMSLE': 0.09367731487555116,
 'Training R^2': 0.8946714395636954,
 'Test R^2': 0.2453869189700515}
```

This code block focuses on tuning the hyperparameters of a RandomForestRegressor model using Randomized Search Cross-Validation.

```
np.random.seed(42)

rf_grid = {"n_estimators": np.arange(10, 200, 10),
           "max_depth": [None, 10, 20],
           "min_samples_split": np.arange(2, 10, 1),
           "min_samples_leaf": np.arange(1, 10, 1),
           "max_features": [0.5, 1.0, "sqrt"], }

rs_model = RandomizedSearchCV(estimator=RandomForestRegressor(),
                              param_distributions=rf_grid,
                              n_iter=20,
                              cv=3,
                              verbose=3)

rs_model.fit(X_train, y_train)
```

```
Fitting 3 folds for each of 20 candidates, totalling 60 fits
[CV 1/3] END max_depth=10, max_features=sqrt, min_samples_leaf=3, min_samples_split=8, n_estimators=130;, score=0.246 total time=
[CV 2/3] END max_depth=10, max_features=sqrt, min_samples_leaf=3, min_samples_split=8, n_estimators=130;, score=0.247 total time=
[CV 3/3] END max_depth=10, max_features=sqrt, min_samples_leaf=3, min_samples_split=8, n_estimators=130;, score=0.235 total time=
[CV 1/3] END max_depth=None, max_features=0.5, min_samples_leaf=6, min_samples_split=7, n_estimators=60;, score=0.257 total time=
[CV 2/3] END max_depth=None, max_features=0.5, min_samples_leaf=6, min_samples_split=7, n_estimators=60;, score=0.261 total time=
[CV 3/3] END max_depth=None, max_features=0.5, min_samples_leaf=6, min_samples_split=7, n_estimators=60;, score=0.251 total time=
[CV 1/3] END max_depth=10, max_features=0.5, min_samples_leaf=9, min_samples_split=5, n_estimators=140;, score=0.262 total time=
[CV 2/3] END max_depth=10, max_features=0.5, min_samples_leaf=9, min_samples_split=5, n_estimators=140;, score=0.266 total time= 1
[CV 3/3] END max_depth=10, max_features=0.5, min_samples_leaf=9, min_samples_split=5, n_estimators=140;, score=0.257 total time= 1
[CV 1/3] END max_depth=10, max_features=0.5, min_samples_leaf=8, min_samples_split=3, n_estimators=50;, score=0.259 total time=  2
[CV 2/3] END max_depth=10, max_features=0.5, min_samples_leaf=8, min_samples_split=3, n_estimators=50;, score=0.264 total time=  2
[CV 3/3] END max_depth=10, max_features=0.5, min_samples_leaf=8, min_samples_split=3, n_estimators=50;, score=0.258 total time=  2
[CV 1/3] END max_depth=20, max_features=sqrt, min_samples_leaf=7, min_samples_split=7, n_estimators=140;, score=0.256 total time=
[CV 2/3] END max_depth=20, max_features=sqrt, min_samples_leaf=7, min_samples_split=7, n_estimators=140;, score=0.257 total time=
[CV 3/3] END max_depth=20, max_features=sqrt, min_samples_leaf=7, min_samples_split=7, n_estimators=140;, score=0.249 total time=
[CV 1/3] END max_depth=20, max_features=sqrt, min_samples_leaf=3, min_samples_split=3, n_estimators=180;, score=0.256 total time=
[CV 2/3] END max_depth=20, max_features=sqrt, min_samples_leaf=3, min_samples_split=3, n_estimators=180;, score=0.258 total time=
[CV 3/3] END max_depth=20, max_features=sqrt, min_samples_leaf=3, min_samples_split=3, n_estimators=180;, score=0.248 total time=
[CV 1/3] END max_depth=10, max_features=1.0, min_samples_leaf=2, min_samples_split=7, n_estimators=160;, score=0.259 total time= 2
[CV 2/3] END max_depth=10, max_features=1.0, min_samples_leaf=2, min_samples_split=7, n_estimators=160;, score=0.261 total time= 2
[CV 3/3] END max_depth=10, max_features=1.0, min_samples_leaf=2, min_samples_split=7, n_estimators=160;, score=0.253 total time= 2
[CV 1/3] END max_depth=10, max_features=1.0, min_samples_leaf=6, min_samples_split=3, n_estimators=150;, score=0.259 total time= 2
[CV 2/3] END max_depth=10, max_features=1.0, min_samples_leaf=6, min_samples_split=3, n_estimators=150;, score=0.262 total time= 2
[CV 3/3] END max_depth=10, max_features=1.0, min_samples_leaf=6, min_samples_split=3, n_estimators=150;, score=0.256 total time= 2
[CV 1/3] END max_depth=None, max_features=0.5, min_samples_leaf=4, min_samples_split=2, n_estimators=110;, score=0.257 total time=
[CV 2/3] END max_depth=None, max_features=0.5, min_samples_leaf=4, min_samples_split=2, n_estimators=110;, score=0.262 total time=
[CV 3/3] END max_depth=None, max_features=0.5, min_samples_leaf=4, min_samples_split=2, n_estimators=110;, score=0.255 total time=
[CV 1/3] END max_depth=10, max_features=0.5, min_samples_leaf=3, min_samples_split=2, n_estimators=190;, score=0.261 total time= 1
[CV 2/3] END max_depth=10, max_features=0.5, min_samples_leaf=3, min_samples_split=2, n_estimators=190;, score=0.264 total time= 1
[CV 3/3] END max_depth=10, max_features=0.5, min_samples_leaf=3, min_samples_split=2, n_estimators=190;, score=0.256 total time= 1
[CV 1/3] END max_depth=10, max_features=1.0, min_samples_leaf=1, min_samples_split=7, n_estimators=120;, score=0.257 total time= 2
[CV 2/3] END max_depth=10, max_features=1.0, min_samples_leaf=1, min_samples_split=7, n_estimators=120;, score=0.259 total time= 1
[CV 3/3] END max_depth=10, max_features=1.0, min_samples_leaf=1, min_samples_split=7, n_estimators=120;, score=0.253 total time= 1
[CV 1/3] END max_depth=20, max_features=sqrt, min_samples_leaf=5, min_samples_split=6, n_estimators=90;, score=0.254 total time=
[CV 2/3] END max_depth=20, max_features=sqrt, min_samples_leaf=5, min_samples_split=6, n_estimators=90;, score=0.258 total time=
[CV 3/3] END max_depth=20, max_features=sqrt, min_samples_leaf=5, min_samples_split=6, n_estimators=90;, score=0.247 total time=
[CV 1/3] END max_depth=20, max_features=sqrt, min_samples_leaf=3, min_samples_split=8, n_estimators=20;, score=0.240 total time=
[CV 2/3] END max_depth=20, max_features=sqrt, min_samples_leaf=3, min_samples_split=8, n_estimators=20;, score=0.236 total time=
[CV 3/3] END max_depth=20, max_features=sqrt, min_samples_leaf=3, min_samples_split=8, n_estimators=20;, score=0.230 total time=
[CV 1/3] END max_depth=20, max_features=sqrt, min_samples_leaf=2, min_samples_split=2, n_estimators=160;, score=0.253 total time=
[CV 2/3] END max_depth=20, max_features=sqrt, min_samples_leaf=2, min_samples_split=2, n_estimators=160;, score=0.256 total time=
[CV 3/3] END max_depth=20, max_features=sqrt, min_samples_leaf=2, min_samples_split=2, n_estimators=160;, score=0.246 total time=
[CV 1/3] END max_depth=20, max_features=0.5, min_samples_leaf=1, min_samples_split=8, n_estimators=10;, score=0.204 total time=
[CV 2/3] END max_depth=20, max_features=0.5, min_samples_leaf=1, min_samples_split=8, n_estimators=10;, score=0.206 total time=
[CV 3/3] END max_depth=20, max_features=0.5, min_samples_leaf=1, min_samples_split=8, n_estimators=10;, score=0.206 total time=
[CV 1/3] END max_depth=None, max_features=1.0, min_samples_leaf=3, min_samples_split=2, n_estimators=140;, score=0.251 total time=
[CV 2/3] END max_depth=None, max_features=1.0, min_samples_leaf=3, min_samples_split=2, n_estimators=140;, score=0.253 total time=
[CV 3/3] END max_depth=None, max_features=1.0, min_samples_leaf=3, min_samples_split=2, n_estimators=140;, score=0.248 total time=
[CV 1/3] END max_depth=None, max_features=0.5, min_samples_leaf=6, min_samples_split=2, n_estimators=100;, score=0.259 total time=
[CV 2/3] END max_depth=None, max_features=0.5, min_samples_leaf=6, min_samples_split=2, n_estimators=100;, score=0.264 total time=
[CV 3/3] END max_depth=None, max_features=0.5, min_samples_leaf=6, min_samples_split=2, n_estimators=100;, score=0.255 total time=
[CV 1/3] END max_depth=20, max_features=1.0, min_samples_leaf=7, min_samples_split=7, n_estimators=10;, score=0.223 total time=  1
[CV 2/3] END max_depth=20, max_features=1.0, min_samples_leaf=7, min_samples_split=7, n_estimators=10;, score=0.223 total time=  1
[CV 3/3] END max_depth=20, max_features=1.0, min_samples_leaf=7, min_samples_split=7, n_estimators=10;, score=0.225 total time=  1
[CV 1/3] END max_depth=10, max_features=sqrt, min_samples_leaf=1, min_samples_split=7, n_estimators=150;, score=0.245 total time=
[CV 2/3] END max_depth=10, max_features=sqrt, min_samples_leaf=1, min_samples_split=7, n_estimators=150;, score=0.248 total time=
[CV 3/3] END max_depth=10, max_features=sqrt, min_samples_leaf=1, min_samples_split=7, n_estimators=150;, score=0.235 total time=
[CV 1/3] END max_depth=None, max_features=1.0, min_samples_leaf=8, min_samples_split=2, n_estimators=20;, score=0.242 total time=
[CV 2/3] END max_depth=None, max_features=1.0, min_samples_leaf=8, min_samples_split=2, n_estimators=20;, score=0.246 total time=
[CV 3/3] END max_depth=None, max_features=1.0, min_samples_leaf=8, min_samples_split=2, n_estimators=20;, score=0.241 total time=
```

RandomizedSearchCV

best_estimator_: RandomForestRegressor

RandomForestRegressor

```
rs_model.best_params_
```

```
{'n_estimators': np.int64(140),
 'min_samples_split': np.int64(5),
 'min_samples_leaf': np.int64(9),
 'max_features': 0.5,
 'max_depth': 10}
```

```
rs_model_scores = show_scores(model=rs_model,
                              train_features=X_train,
                              train_labels=y_train,
                              test_features=X_test,
                              test_labels=y_test)
rs_model_scores
```

```
{'Training MAE': 0.5473572853514472,
 'Test MAE': 0.5809276716238158,
```

```
    'Training RMSLE': 0.08870563578612232,
    'Test RMSLE': 0.09300200957031936,
    'Training R^2': 0.35659503446317753,
    'Test R^2': 0.25901433056358125}
```

This code block takes the best hyperparameters found during a previous randomized search on a sample of the data (sample_rs_model.best_params_) and uses them to train a new RandomForestRegressor model on the full training dataset (X_train, y_train).

```
np.random.seed(42)

rf_best_model_rs = RandomForestRegressor(**sample_rs_model.best_params_, n_jobs=-1)

rf_best_model_rs.fit(X=X_train,
                     y=y_train)

rf_best_model_rs_scores = show_scores(model=rf_best_model_rs,
                                      train_features=X_train,
                                      train_labels=y_train,
                                      test_features=X_test,
                                      test_labels=y_test)

rf_best_model_rs_scores
```

```
{'Training MAE': 0.5624191487918332,
 'Test MAE': 0.5876372043698553,
 'Training RMSLE': 0.09061482159477974,
 'Test RMSLE': 0.09415714889156918,
 'Training R^2': 0.32594390438005383,
 'Test R^2': 0.24094986223080406}
```

This code block trains a machine learning model, specifically a Random Forest Regressor, using the optimal hyperparameters found during a previous Grid Search performed on a sample of the data. It then evaluates the performance of this trained model.

```
np.random.seed(42)

rf_best_model_gs = RandomForestRegressor(**sample_gs_model.best_params_, n_jobs=-1)

rf_best_model_gs.fit(X=X_train,
                     y=y_train)

rf_best_model_gs_scores = show_scores(model=rf_best_model_gs,
                                      train_features=X_train,
                                      train_labels=y_train,
                                      test_features=X_test,
                                      test_labels=y_test)

rf_best_model_gs_scores
```

```
{'Training MAE': 0.5435080531798863,
 'Test MAE': 0.5814392662242237,
 'Training RMSLE': 0.08762451259715064,
 'Test RMSLE': 0.09304934505106814,
 'Training R^2': 0.3703261958364007,
 'Test R^2': 0.2581559797832569}
```

This code block trains a machine learning model, specifically a Random Forest Regressor, using the optimal hyperparameters found during a previous Halving Randomized Search performed on a sample of the data. It then evaluates the performance of this trained model.

```
np.random.seed(42)

rf_best_model_hrs = RandomForestRegressor(**sample_hrs_model.best_params_, n_jobs=-1)

rf_best_model_hrs.fit(X=X_train,
                     y=y_train)

rf_best_model_hrs_scores = show_scores(model=rf_best_model_hrs,
                                      train_features=X_train,
                                      train_labels=y_train,
                                      test_features=X_test,
                                      test_labels=y_test)

rf_best_model_hrs_scores
```

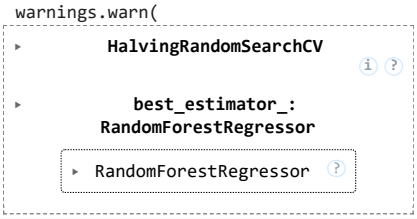
```
{'Training MAE': 0.565585096031433,
 'Test MAE': 0.5875540100160795,
 'Training RMSLE': 0.09170307199110485,
 'Test RMSLE': 0.09412788456318996,
 'Training R^2': 0.3115352189683698,
 'Test R^2': 0.24132504820335732}
```

This code block is setting up and running a HalvingRandomSearchCV for a RandomForestRegressor model. The goal is to efficiently find the best combination of hyperparameters for the model by exploring a range of possible values.

```
np.random.seed(42)

rf_grid = {"n_estimators": np.arange(50, 300, 25),
           "max_depth": [10, 20, 30, None],
           "min_samples_split": np.arange(5, 20, 2),
           "min_samples_leaf": np.arange(3, 15, 2),
           "max_features": [0.5, 0.7, 1.0, "sqrt"],
           "bootstrap": [True, False],
           "oob_score": [True, False]}

halving_rs_model = HalvingRandomSearchCV(estimator=RandomForestRegressor(n_jobs=-1),
                                         param_distributions=rf_grid,
                                         cv=5,
                                         verbose=3,
                                         n_jobs=-1)

halving_rs_model.fit(X_train, y_train)
```

```
n_iterations: 8
n_required_iterations: 8
n_possible_iterations: 8
min_resources_: 10
max_resources_: 48739
aggressive_elimination: False
factor: 3
----------
iter: 0
n_candidates: 4873
n_resources: 10
Fitting 5 folds for each of 4873 candidates, totalling 24365 fits
/usr/local/lib/python3.11/dist-packages/sklearn/model_selection/_validation.py:528: FitFailedWarning:
6055 fits failed out of a total of 24365.
The score on these train-test partitions for these parameters will be set to nan.
If these failures are not expected, you can try to debug them by setting error_score='raise'.

Below are more details about the failures:
--------------------------------------------------------------------------------
6055 fits failed with the following error:
Traceback (most recent call last):
  File "/usr/local/lib/python3.11/dist-packages/sklearn/model_selection/_validation.py", line 866, in _fit_and_score
    estimator.fit(X_train, y_train, **fit_params)
  File "/usr/local/lib/python3.11/dist-packages/sklearn/base.py", line 1389, in wrapper
    return fit_method(estimator, *args, **kwargs)
           ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
  File "/usr/local/lib/python3.11/dist-packages/sklearn/ensemble/_forest.py", line 448, in fit
    raise ValueError("Out of bag estimation only available if bootstrap=True")
ValueError: Out of bag estimation only available if bootstrap=True

  warnings.warn(some_fits_failed_message, FitFailedWarning)
/usr/local/lib/python3.11/dist-packages/sklearn/model_selection/_search.py:1108: UserWarning: One or more of the test scores are n
  warnings.warn(
/usr/local/lib/python3.11/dist-packages/sklearn/model_selection/_search.py:1108: UserWarning: One or more of the train scores are
 -4.27193078e-04 -4.44089210e-17]
  warnings.warn(
----------
iter: 1
n_candidates: 1625
n_resources: 30
Fitting 5 folds for each of 1625 candidates, totalling 8125 fits
/usr/local/lib/python3.11/dist-packages/sklearn/model_selection/_validation.py:528: FitFailedWarning:
2005 fits failed out of a total of 8125.
The score on these train-test partitions for these parameters will be set to nan.
If these failures are not expected, you can try to debug them by setting error_score='raise'.

Below are more details about the failures:
--------------------------------------------------------------------------------
2005 fits failed with the following error:
Traceback (most recent call last):
  File "/usr/local/lib/python3.11/dist-packages/sklearn/model_selection/_validation.py", line 866, in _fit_and_score
    estimator.fit(X_train, y_train, **fit_params)
  File "/usr/local/lib/python3.11/dist-packages/sklearn/base.py", line 1389, in wrapper
    return fit_method(estimator, *args, **kwargs)
           ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
  File "/usr/local/lib/python3.11/dist-packages/sklearn/ensemble/_forest.py", line 448, in fit
    raise ValueError("Out of bag estimation only available if bootstrap=True")
ValueError: Out of bag estimation only available if bootstrap=True

  warnings.warn(some_fits_failed_message, FitFailedWarning)
/usr/local/lib/python3.11/dist-packages/sklearn/model_selection/_search.py:1108: UserWarning: One or more of the test scores are n
 -0.01605811]
  warnings.warn(
/usr/local/lib/python3.11/dist-packages/sklearn/model_selection/_search.py:1108: UserWarning: One or more of the train scores are
  3.41004882e-01  2.90062201e-01]
  warnings.warn(
----------
iter: 2
n_candidates: 542
n_resources: 90
Fitting 5 folds for each of 542 candidates, totalling 2710 fits
/usr/local/lib/python3.11/dist-packages/sklearn/model_selection/_search.py:1108: UserWarning: One or more of the test scores are n
 -0.00157941]
  warnings.warn(
/usr/local/lib/python3.11/dist-packages/sklearn/model_selection/_search.py:1108: UserWarning: One or more of the train scores are
  7.06175891e-01  7.62732700e-01]
  warnings.warn(
----------
iter: 3
n_candidates: 181
n_resources: 270
Fitting 5 folds for each of 181 candidates, totalling 905 fits
/usr/local/lib/python3.11/dist-packages/sklearn/model_selection/_search.py:1108: UserWarning: One or more of the test scores are n
  warnings.warn(
/usr/local/lib/python3.11/dist-packages/sklearn/model_selection/_search.py:1108: UserWarning: One or more of the train scores are
  7.25171879e-01  7.30219253e-01]
  warnings.warn(
----------
iter: 4
n_candidates: 61
```

```
n_resources: 810
Fitting 5 folds for each of 61 candidates, totalling 305 fits
/usr/local/lib/python3.11/dist-packages/sklearn/model_selection/_search.py:1108: UserWarning: One or more of the test scores are n
  warnings.warn(
/usr/local/lib/python3.11/dist-packages/sklearn/model_selection/_search.py:1108: UserWarning: One or more of the train scores are
  5.94897933e-01  5.76805513e-01]
  warnings.warn(
----------
iter: 5
n_candidates: 21
n_resources: 2430
Fitting 5 folds for each of 21 candidates, totalling 105 fits
/usr/local/lib/python3.11/dist-packages/sklearn/model_selection/_search.py:1108: UserWarning: One or more of the test scores are n
  warnings.warn(
/usr/local/lib/python3.11/dist-packages/sklearn/model_selection/_search.py:1108: UserWarning: One or more of the train scores are
  5.65216244e-01  5.52050012e-01]
  warnings.warn(
----------
iter: 6
n_candidates: 7
n_resources: 7290
Fitting 5 folds for each of 7 candidates, totalling 35 fits
/usr/local/lib/python3.11/dist-packages/sklearn/model_selection/_search.py:1108: UserWarning: One or more of the test scores are n
  warnings.warn(
/usr/local/lib/python3.11/dist-packages/sklearn/model_selection/_search.py:1108: UserWarning: One or more of the train scores are
  4.93965097e-01  6.15766271e-01]
  warnings.warn(
----------
iter: 7
n_candidates: 3
n_resources: 21870
Fitting 5 folds for each of 3 candidates, totalling 15 fits
/usr/local/lib/python3.11/dist-packages/sklearn/model_selection/_search.py:1108: UserWarning: One or more of the test scores are n
  warnings.warn(
/usr/local/lib/python3.11/dist-packages/sklearn/model_selection/_search.py:1108: UserWarning: One or more of the train scores are
  5.76771625e-01  4.24070309e-01]
  warnings.warn(
```



```
halving_rs_model.best_params_
```

```
{'oob_score': False,
 'n_estimators': np.int64(275),
 'min_samples_split': np.int64(17),
 'min_samples_leaf': np.int64(7),
 'max_features': 0.5,
 'max_depth': 30,
 'bootstrap': True}
```

```
halving_rs_model_scores = show_scores(model=halving_rs_model,
                              train_features=X_train,
                              train_labels=y_train,
                              test_features=X_test,
                              test_labels=y_test)
halving_rs_model_scores
```

```
{'Training MAE': 0.43239679105730994,
 'Test MAE': 0.5809039460064926,
 'Training RMSLE': 0.07387080889863588,
 'Test RMSLE': 0.09281565264812022,
 'Training R^2': 0.5763891058035406,
 'Test R^2': 0.26164274423707845}
```

This code block trains a Ridge Regression model. It then evaluates the trained model's performance using show_scores function and prints the resulting scores.

```
ridge_model = Ridge()

ridge_model.fit(X_train, y_train)

ridge_scores = show_scores(model=ridge_model,
                           train_features=X_train,
                           train_labels=y_train,
                           test_features=X_test,
                           test_labels=y_test)

print("Ridge Regression Scores:")
ridge_scores
```

```
Ridge Regression Scores:
/usr/local/lib/python3.11/dist-packages/sklearn/linear_model/_ridge.py:215: LinAlgWarning: Ill-conditioned matrix (rcond=2.93359e-27
  return linalg.solve(A, Xy, assume_a="pos", overwrite_a=True).T
{'Training MAE': 0.6231777443912474,
 'Test MAE': 0.6186031282587662,
 'Training RMSLE': 0.10036177894638101,
 'Test RMSLE': 0.09854144256145399,
 'Training R^2': 0.17322049052005628,
 'Test R^2': 0.16280710973535883}
```

This code block trains and evaluates a Support Vector Regression (SVR) model with a Radial Basis Function (RBF) kernel.

```
svr_model = SVR(kernel="rbf")

svr_model.fit(X_train, y_train)

svr_scores = show_scores(model=svr_model,
                         train_features=X_train,
                         train_labels=y_train,
                         test_features=X_test,
                         test_labels=y_test)

print("SVR (RBF Kernel) Scores:")
svr_scores
```

```
SVR (RBF Kernel) Scores:
{'Training MAE': 0.6881027560787162,
 'Test MAE': 0.6842749615898589,
 'Training RMSLE': 0.10811746913004266,
 'Test RMSLE': 0.1062625410687166,
 'Training R^2': 0.008715864813863994,
 'Test R^2': 0.007762053219004805}
```

This code block trains and evaluates a Gradient Boosting Regressor.

```
gbr_model = GradientBoostingRegressor(random_state=42)

```

```
gbr_model.fit(X_train, y_train)

gbr_scores = show_scores(model=gbr_model,
                         train_features=X_train,
                         train_labels=y_train,
                         test_features=X_test,
                         test_labels=y_test)

print("Gradient Boosting Regressor Scores:")
gbr_scores
```

```
Gradient Boosting Regressor Scores:
{'Training MAE': 0.5751491075187253,
 'Test MAE': 0.5798634603002869,
 'Training RMSLE': 0.09284064173529244,
 'Test RMSLE': 0.09280764947156771,
 'Training R^2': 0.2898418118980568,
 'Test R^2': 0.262580555628648}
```

This code block focuses on training and evaluating a Lasso Regression model.

```
lasso_model = Lasso(random_state=42)

lasso_model.fit(X_train, y_train)

lasso_scores = show_scores(model=lasso_model,
                           train_features=X_train,
                           train_labels=y_train,
                           test_features=X_test,
                           test_labels=y_test)

print("Lasso Regression Scores:")
lasso_scores
```

```
Lasso Regression Scores:
{'Training MAE': 0.632726509690792,
 'Test MAE': 0.6292975515719712,
 'Training RMSLE': 0.10120086801639935,
 'Test RMSLE': 0.09936992092910946,
 'Training R^2': 0.15270338476070877,
 'Test R^2': 0.14448708096918006}
```

This code block focuses on training and evaluating a Linear Regression model.

```
linear_model = LinearRegression()

linear_model.fit(X_train, y_train)

linear_scores = show_scores(model=linear_model,
                            train_features=X_train,
                            train_labels=y_train,
                            test_features=X_test,
                            test_labels=y_test)

print("Linear Regression Scores:")
linear_scores
```

```
Linear Regression Scores:
{'Training MAE': 0.6245620575467123,
 'Test MAE': 0.6199371872971146,
 'Training RMSLE': 0.10063056015042172,
 'Test RMSLE': 0.09867805024108317,
 'Training R^2': 0.16845493968968595,
 'Test R^2': 0.16032442404375724}
```

This code block trains and evaluates a XGBoost (Extreme Gradient Boosting) Regressor model.

```
xgb_model = xgb.XGBRegressor(objective='reg:squarederror', random_state=42)

xgb_model.fit(X_train, y_train)

xgb_scores = show_scores(model=xgb_model,
                         train_features=X_train,
                         train_labels=y_train,
                         test_features=X_test,
                         test_labels=y_test)

print("XGBoost Regressor Scores:")
xgb_scores
```

```
XGBoost Regressor Scores:
{'Training MAE': 0.4955529241429421,
 'Test MAE': 0.5951668821028818,
 'Training RMSLE': 0.0790799696419051,
 'Test RMSLE': 0.09462947333056976,
 'Training R^2': 0.47928815972210703,
 'Test R^2': 0.22986016432730916}
```

This code block trains and evaluates a LightGBM Regressor model.

```
base_lgb_model  = lgb.LGBMRegressor(objective='regression', random_state=42)

base_lgb_model .fit(X_train, y_train)

base_lgb_scores = show_scores(model=base_lgb_model ,
                          train_features=X_train,
                          train_labels=y_train,
                          test_features=X_test,
                          test_labels=y_test)

print("LightGBM Regressor Scores:")
base_lgb_scores
```

```
[LightGBM] [Info] Auto-choosing row-wise multi-threading, the overhead of testing was 0.019830 seconds.
You can set `force_row_wise=true` to remove the overhead.
And if memory is not enough, you can set `force_col_wise=true`.
[LightGBM] [Info] Total Bins 7622
[LightGBM] [Info] Number of data points in the train set: 48739, number of used features: 64
[LightGBM] [Info] Start training from score 7.906696
LightGBM Regressor Scores:
{'Training MAE': 0.5525830835730449,
 'Test MAE': 0.5788805861720081,
 'Training RMSLE': 0.08900097029380527,
 'Test RMSLE': 0.09255008213444889,
 'Training R^2': 0.34836009160766324,
 'Test R^2': 0.2661600517020939}
```

This code block trains and evaluates a Decision Tree Regressor model.

```
dt_model = DecisionTreeRegressor(random_state=42)

dt_model.fit(X_train, y_train)

dt_scores = show_scores(model=dt_model,
                        train_features=X_train,
                        train_labels=y_train,
                        test_features=X_test,
                        test_labels=y_test)

print("Decision Tree Regression Scores:")
dt_scores
```

```
Decision Tree Regression Scores:
{'Training MAE': 2.186778767804326e-19,
 'Test MAE': 0.8620016413623306,
 'Training RMSLE': 0.0,
 'Test RMSLE': 0.1367951210369189,
 'Training R^2': 1.0,
 'Test R^2': -0.5571233874357828}
```

Because LightGBM Regressor produced the best score, we will try to tune it to get an even better score.

This code block focuses on tuning the hyperparameters of a LightGBM Regressor model using Randomized Search Cross-Validation.

```
np.random.seed(42)

lgb_grid = {
    'n_estimators': [100, 200, 300],
    'learning_rate': [0.01, 0.1, 0.2],
    'num_leaves': [31, 50, 100],
    'max_depth': [-1, 10, 20],
    'min_child_samples': [20, 50, 100],
    'subsample': [0.8, 1.0],
    'colsample_bytree': [0.8, 1.0],
}

tuned_lgb_model = RandomizedSearchCV(estimator=lgb.LGBMRegressor(random_state=42),
                                     param_distributions=lgb_grid,
                                     n_iter=20,
```

```
                                    cv=3,
                                    verbose=3,
                                    n_jobs=-1)

tuned_lgb_model.fit(X=X_train, y=y_train)
```

```
Fitting 3 folds for each of 20 candidates, totalling 60 fits
[LightGBM] [Info] Auto-choosing col-wise multi-threading, the overhead of testing was 0.056658 seconds.
You can set `force_col_wise=true` to remove the overhead.
[LightGBM] [Info] Total Bins 7622
[LightGBM] [Info] Number of data points in the train set: 48739, number of used features: 64
[LightGBM] [Info] Start training from score 7.906696
```

RandomizedSearchCV

best_estimator_: LGBMRegressor

LGBMRegressor

```
tuned_lgb_model.best_params_
```

```
{'subsample': 1.0,
 'num_leaves': 31,
 'n_estimators': 100,
 'min_child_samples': 100,
 'max_depth': 20,
 'learning_rate': 0.1,
 'colsample_bytree': 0.8}
```

```
tuned_lgb_scores = show_scores(model=tuned_lgb_model,
                               train_features=X_train,
                               train_labels=y_train,
                               test_features=X_test,
                               test_labels=y_test)
tuned_lgb_scores
```

```
{'Training MAE': 0.555199268874275,
 'Test MAE': 0.5779629516602062,
 'Training RMSLE': 0.08976384401293586,
 'Test RMSLE': 0.09243194321941373,
 'Training R^2': 0.33928446621819053,
 'Test R^2': 0.26818849514884713}
```

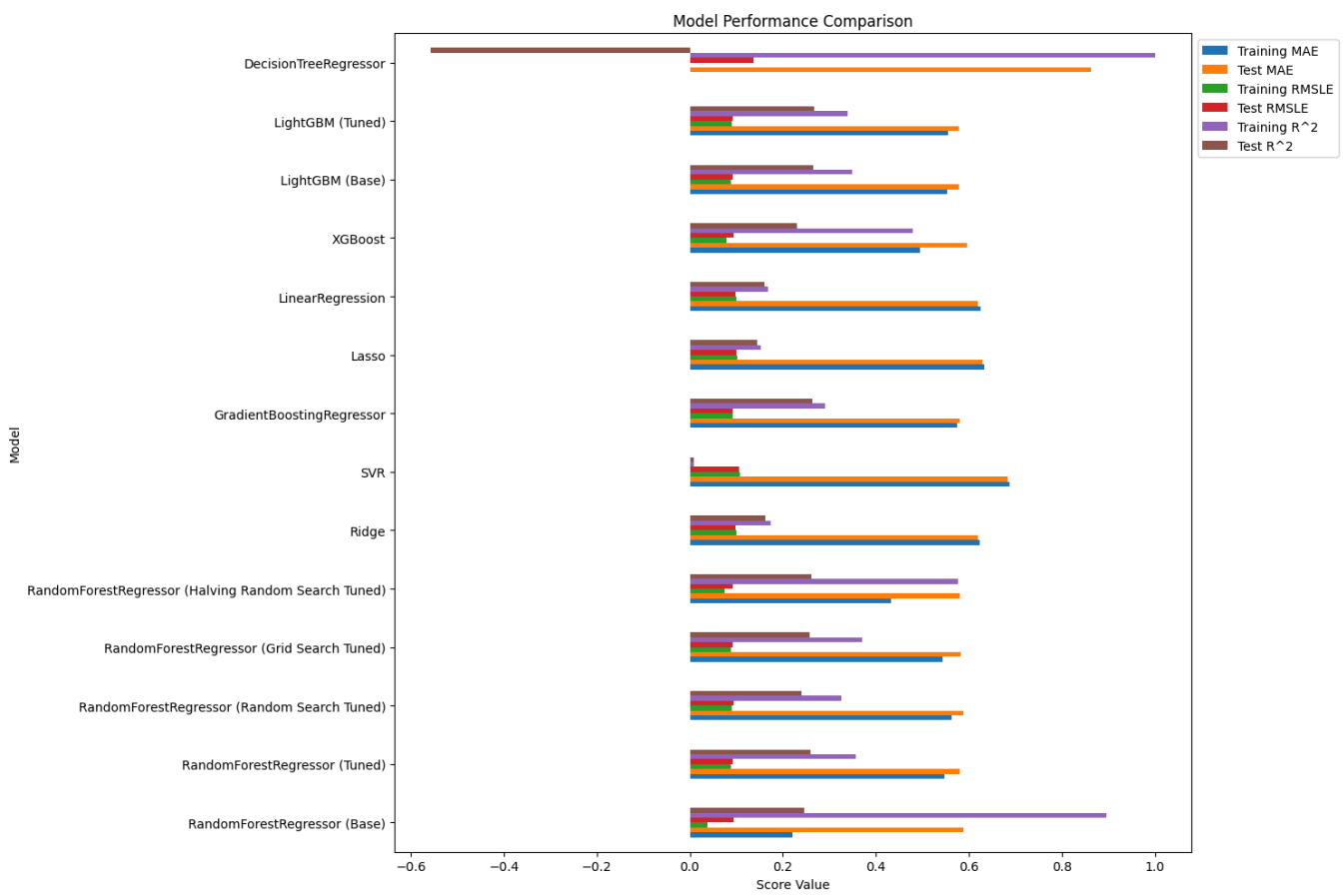
This code block is used to organize and compare the performance scores of various machine learning models that were trained and evaluated in the preceding cells.

```
base_model_scores["model_name"] = "RandomForestRegressor (Base)"
rs_model_scores["model_name"] = "RandomForestRegressor (Tuned)"
ridge_scores["model_name"] = "Ridge"
svr_scores["model_name"] = "SVR"
gbr_scores["model_name"] = "GradientBoostingRegressor"
lasso_scores["model_name"] = "Lasso"
linear_scores["model_name"] = "LinearRegression"
xgb_scores["model_name"] = "XGBoost"
base_lgb_scores["model_name"] = "LightGBM (Base)"
tuned_lgb_scores["model_name"] = "LightGBM (Tuned)"
dt_scores["model_name"] = "DecisionTreeRegressor"
rf_best_model_rs_scores["model_name"] = "RandomForestRegressor (Random Search Tuned)"
rf_best_model_gs_scores["model_name"] = "RandomForestRegressor (Grid Search Tuned)"
halving_rs_model_scores["model_name"] = "RandomForestRegressor (Halving Random Search Tuned)"


all_model_scores = [base_model_scores,
                    rs_model_scores,
                    rf_best_model_rs_scores,
                    rf_best_model_gs_scores,
                    halving_rs_model_scores,
                    ridge_scores,
                    svr_scores,
                    gbr_scores,
                    lasso_scores,
                    linear_scores,
                    xgb_scores,
                    base_lgb_scores,
                    tuned_lgb_scores,
                    dt_scores]

model_comparison_df = pd.DataFrame(all_model_scores).sort_values(by="Test R^2", ascending=True)
model_comparison_df
```

	Training MAE	Test MAE	Training RMSLE	Test RMSLE	Training R^2	Test R^2	model_name
13	2.186779e-19	0.862002	0.000000	0.136795	1.000000	-0.557123	DecisionTreeRegressor
6	6.881028e-01	0.684275	0.108117	0.106263	0.008716	0.007762	SVR
8	6.327265e-01	0.629298	0.101201	0.099370	0.152703	0.144487	Lasso
9	6.245621e-01	0.619937	0.100631	0.098678	0.168455	0.160324	LinearRegression
5	6.231777e-01	0.618603	0.100362	0.098541	0.173220	0.162807	Ridge
10	4.955529e-01	0.595167	0.079080	0.094629	0.479288	0.229860	XGBoost
2	5.624191e-01	0.587637	0.090615	0.094157	0.325944	0.240950	RandomForestRegressor (Random Search Tuned)
0	2.205965e-01	0.588709	0.038303	0.093677	0.894671	0.245387	RandomForestRegressor (Base)
3	5.435081e-01	0.581439	0.087625	0.093049	0.370326	0.258156	RandomForestRegressor (Grid Search Tuned)
1	5.473573e-01	0.580928	0.088706	0.093002	0.356595	0.259014	RandomForestRegressor (Tuned)
4	4.323968e-01	0.580904	0.073871	0.092816	0.576389	0.261643	RandomForestRegressor (Halving Random Search T...
7	5.751491e-01	0.579863	0.092841	0.092808	0.289842	0.262581	GradientBoostingRegressor
11	5.525831e-01	0.578881	0.089001	0.092550	0.348360	0.266160	LightGBM (Base)
12	5.551993e-01	0.577963	0.089764	0.092432	0.339284	0.268188	LightGBM (Tuned)

This code block focuses on visualizing the performance of the various machine learning models trained in the previous steps. It creates a bar chart to easily compare the scores of each model.

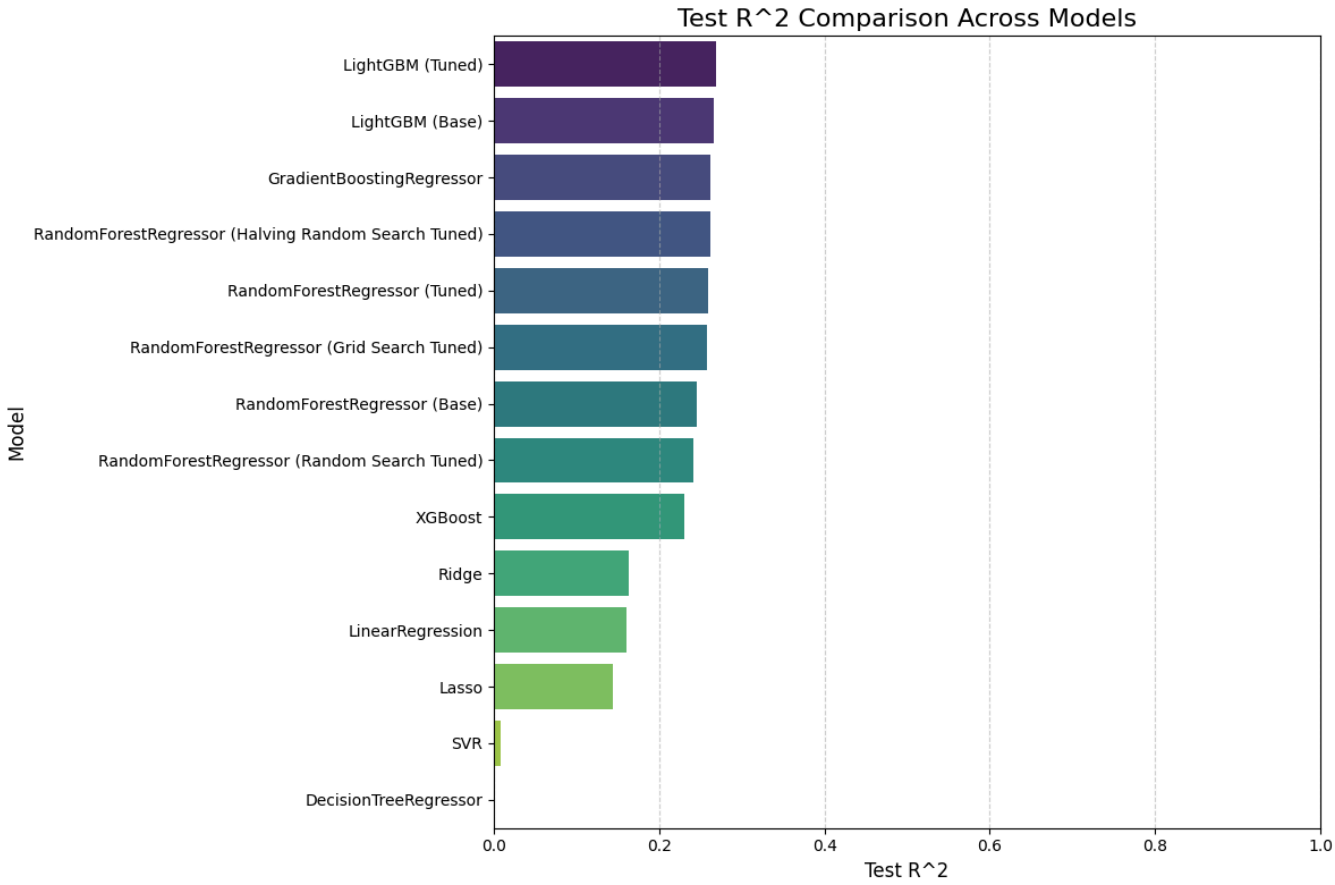
```
model_scores = pd.DataFrame({
    "RandomForestRegressor (Base)": base_model_scores,
    "RandomForestRegressor (Tuned)": rs_model_scores,
    "RandomForestRegressor (Random Search Tuned)": rf_best_model_rs_scores,
    "RandomForestRegressor (Grid Search Tuned)": rf_best_model_gs_scores,
    "RandomForestRegressor (Halving Random Search Tuned)": halving_rs_model_scores,
    "Ridge": ridge_scores,
    "SVR": svr_scores,
    "GradientBoostingRegressor": gbr_scores,
    "Lasso": lasso_scores,
    "LinearRegression": linear_scores,
    "XGBoost": xgb_scores,
    "LightGBM (Base)": base_lgb_scores,
    "LightGBM (Tuned)": tuned_lgb_scores,
    "DecisionTreeRegressor": dt_scores
})

model_scores = model_scores.transpose()

fig, ax = plt.subplots(figsize=(15, 10))
model_scores.plot(kind="barh", ax=ax)
ax.set_title("Model Performance Comparison")
ax.set_xlabel("Score Value")
ax.set_ylabel("Model")
plt.legend(loc="upper left", bbox_to_anchor=(1, 1))
plt.tight_layout()
plt.show()
```



This code block focuses on visualizing the performance of the machine learning models trained and evaluated previously. It specifically creates a horizontal bar chart to compare the Test R^2 score for each model.

```
model_comparison_df_sorted = model_comparison_df.sort_values(by="Test R^2", ascending=False)

plt.figure(figsize=(12, 8))
sns.barplot(x="Test R^2", y="model_name", data=model_comparison_df_sorted, palette="viridis")
plt.title("Test R^2 Comparison Across Models", fontsize=16)
plt.xlabel("Test R^2", fontsize=12)
plt.ylabel("Model", fontsize=12)
plt.xlim(0, 1)
plt.grid(axis='x', linestyle='--', alpha=0.6)
plt.tight_layout()
plt.show()
```

```
<ipython-input-107-43a2ba204d6a>:4: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `y` variable to `hue` and set `le

  sns.barplot(x="Test R^2", y="model_name", data=model_comparison_df_sorted, palette="viridis")
```



Machine Learning Model (Total Completed)

This code block prepares the sample data for training a machine learning model to predict the 'user_completed'.

```
sample = user_country_stats.sample(n=1000, random_state=42)

x_sample = sample.drop(labels="user_completed", axis=1)
y_sample = np.log1p(sample["user_completed"])
```

This code block splits the data into training and testing sets.

```
x_sample_train, x_sample_test, y_sample_train, y_sample_test = train_test_split(x_sample, y_sample, test_size=0.2, random_state=42)
```

This code block splits the main dataset into features and the target variable, and then further divides these into training and testing sets.

```
X = user_country_stats.drop('user_completed', axis=1)
y = np.log1p(user_country_stats['user_completed'])

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

This code block focuses on cleaning the column names of training and testing feature sets, X_train and X_test.

```
X_train.columns = [re.sub('[^A-Za-z0-9_]+', '_', col) for col in X_train.columns]
X_test.columns = [re.sub('[^A-Za-z0-9_]+', '_', col) for col in X_test.columns]
```