Predicting Olympic Medal Counts

Using Machine Learning and Deep Learning

By Balkrishna M Mottannavar

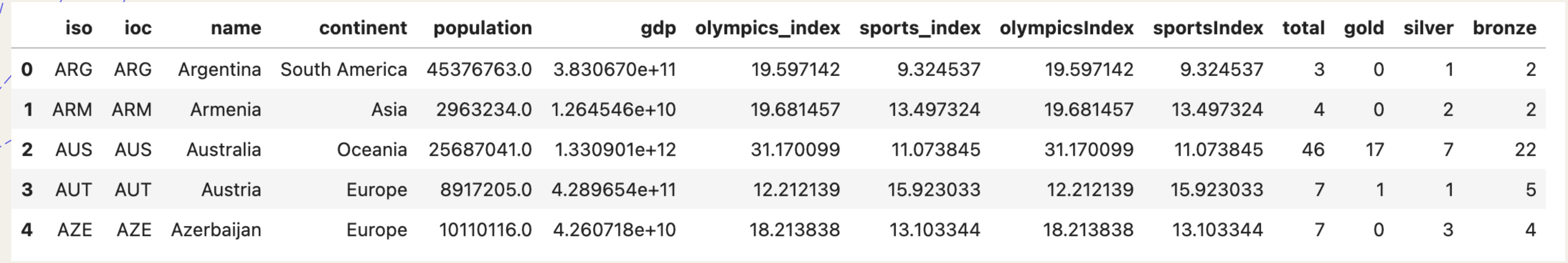
Project Overview

- In this project, we will explore basic machine learning (ML) and deep learning (DL) techniques to predict the number of Olympic medals a country will win.
- We will build and evaluate different models to understand which factors are most influential in predicting Olympic success.

About the Dataset

- The dataset provided includes features such as:
  - iso: Country ISO code
  - ioc: International Olympic Committee code
  - name: Country name
  - continent: Continent of the country
  - population: Population of the country
  - gdp: Gross Domestic Product (GDP) of the country
  - olympics_index: An index indicating the country's overall performance in the Olympics
  - sports_index: An index indicating the country's sports infrastructure and support
  - olympicsIndex: A calculated index related to Olympic performance
  - sportsIndex: A calculated index related to sports
  - total: Total number of medals won
  - gold: Number of gold medals won
  - silver: Number of silver medals won
  - bronze: Number of bronze medals won

Data Preprocessing



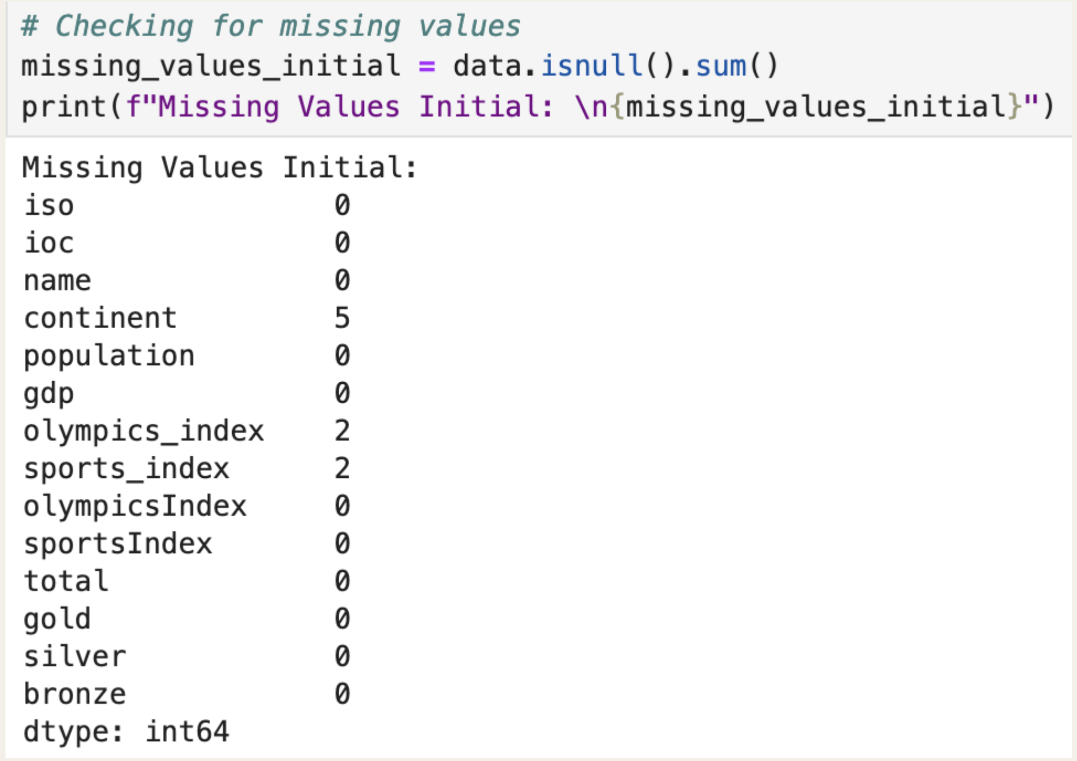
	iso	ioc	name	continent	population	gdp	olympics_index	sports_index	olympicsIndex	sportsIndex	total	gold	silver	bronze
0	ARG	ARG	Argentina	South America	45376763.0	3.830670e+11	19.597142	9.324537	19.597142	9.324537	3	0	1	2
1	ARM	ARM	Armenia	Asia	2963234.0	1.264546e+10	19.681457	13.497324	19.681457	13.497324	4	0	2	2
2	AUS	AUS	Australia	Oceania	25687041.0	1.330901e+12	31.170099	11.073845	31.170099	11.073845	46	17	7	22
3	AUT	AUT	Austria	Europe	8917205.0	4.289654e+11	12.212139	15.923033	12.212139	15.923033	7	1	1	5
4	AZE	AZE	Azerbaijan	Europe	10110116.0	4.260718e+10	18.213838	13.103344	18.213838	13.103344	7	0	3	4

Fig. First 5 Rows (out of 93) and 14 features of the Dataset

Feature Duplication:
In the above data we can clearly see that the features ‘olympicsIndex’ and ‘sportsIndex’ are the duplications of ‘Olympics_index’ and ‘sports_index’ respectively.

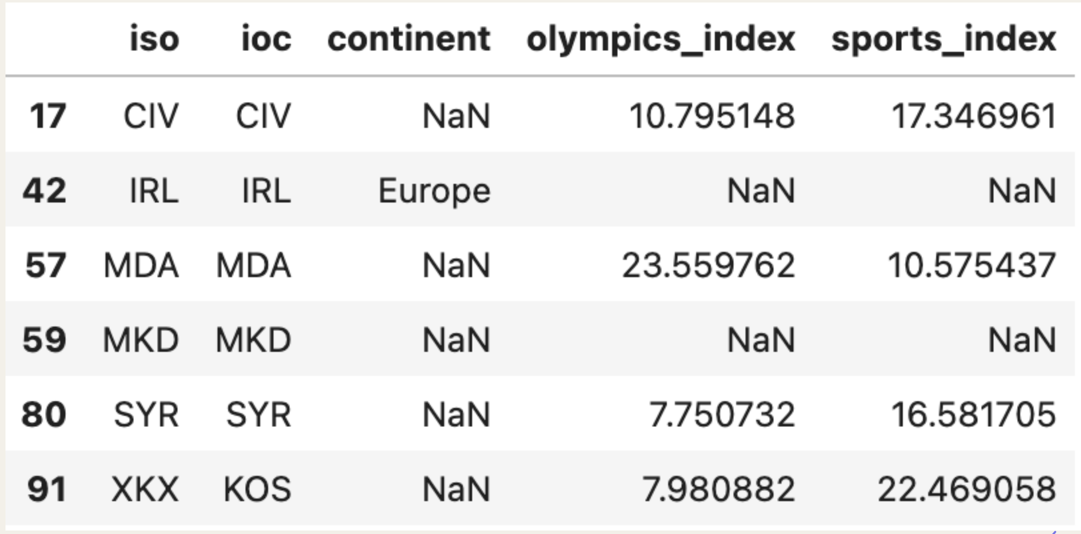
Data Preprocessing

Checking for any Missing Values



```
# Checking for missing values
missing_values_initial = data.isnull().sum()
print(f"Missing Values Initial: \n{missing_values_initial}")
```

```
Missing Values Initial:
iso               0
ioc               0
name              0
continent         5
population        0
gdp               0
olympics_index    2
sports_index      2
olympicsIndex     0
sportsIndex       0
total             0
gold              0
silver            0
bronze            0
dtype: int64
```



	iso	ioc	continent	olympics_index	sports_index
17	CIV	CIV	NaN	10.795148	17.346961
42	IRL	IRL	Europe	NaN	NaN
57	MDA	MDA	NaN	23.559762	10.575437
59	MKD	MKD	NaN	NaN	NaN
80	SYR	SYR	NaN	7.750732	16.581705
91	XKX	KOS	NaN	7.980882	22.469058

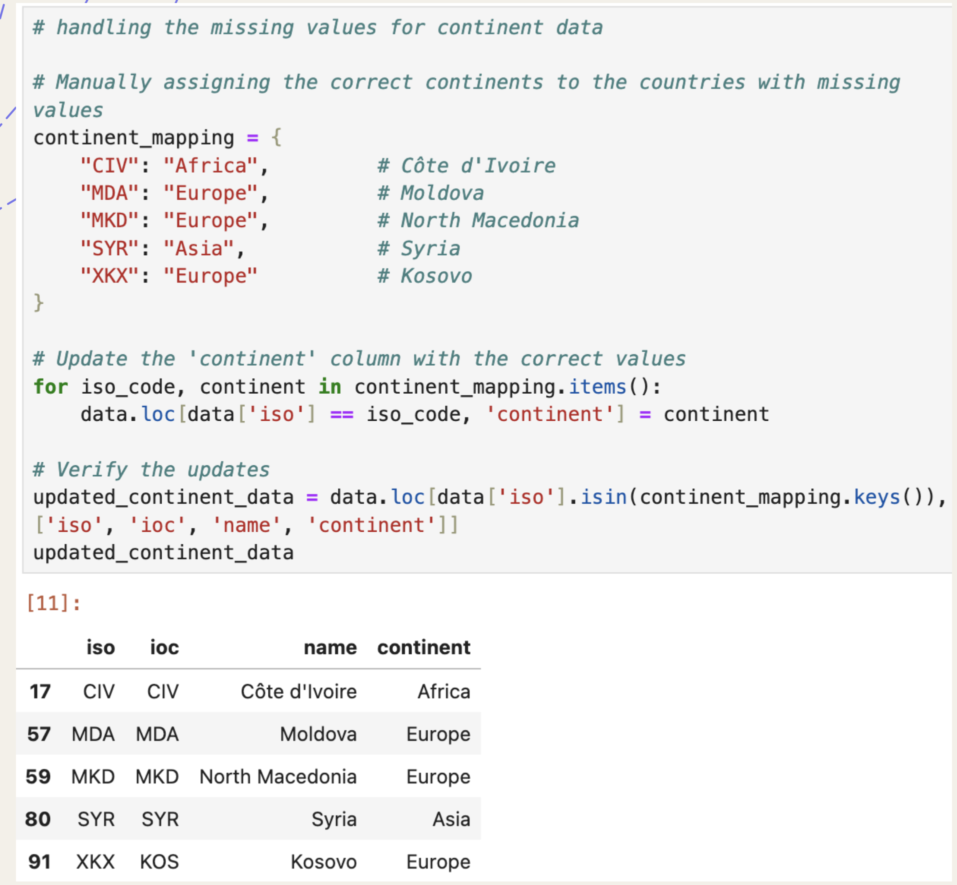
We can find that 'continent', 'olympics_index' and 'sports_index' Features have missing values

Data Preprocessing

Data Cleaning

Features	Cleaning Approach
Continent	Manually: As the name of the countries are provided under the feature called 'name' we can choose to fill the missing continent data manually with respect to their country names.
olympics_index Sports_index	K Nearest Neighbours: as these indices represents countries overall performance in olympics and countries sports infrastucture and support respectivelly, the best approach to fill in is Imputation Based on Similar Countries i.e., We could estimate the values based on countries with similar characteristics (e.g., GDP, population, or continent). This approach could provide more accurate imputations by considering factors that influence sports performance and infrastructure.

Data Preprocessing

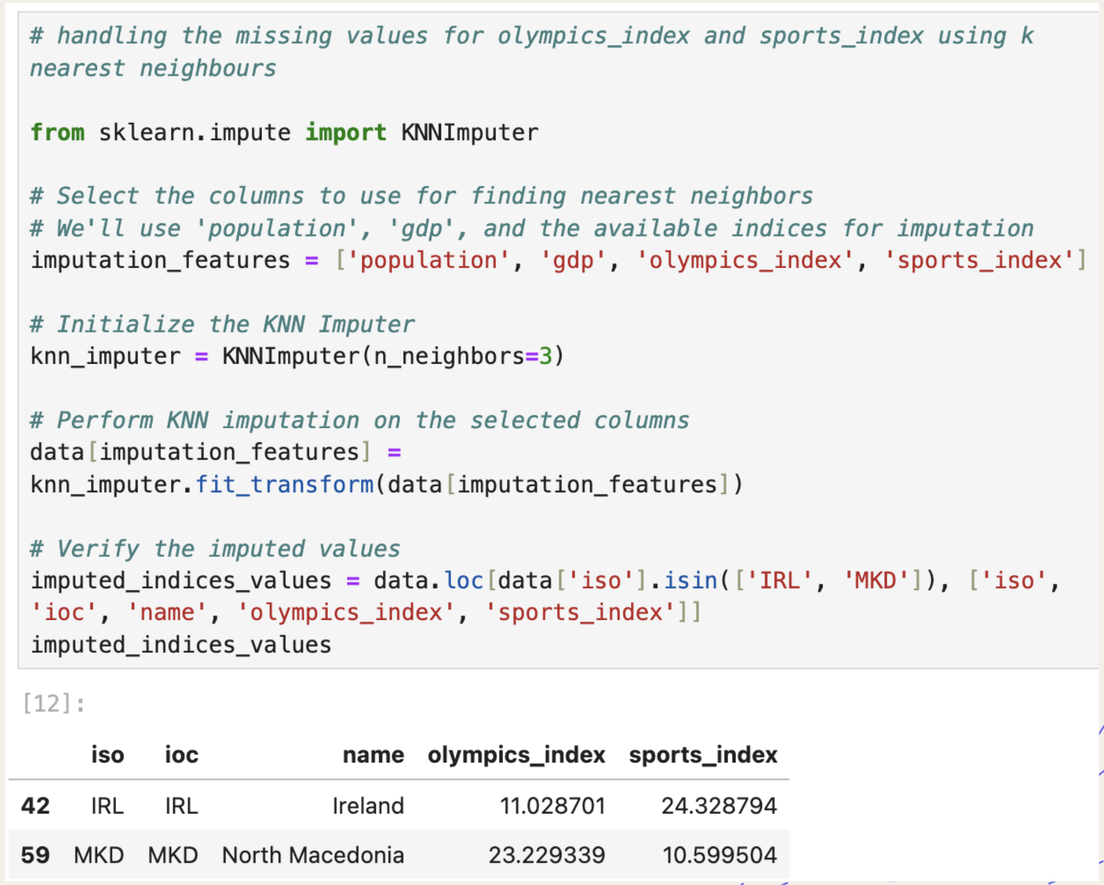


```
# handling the missing values for continent data

# Manually assigning the correct continents to the countries with missing
values
continent_mapping = {
    "CIV": "Africa",         # Côte d'Ivoire
    "MDA": "Europe",         # Moldova
    "MKD": "Europe",         # North Macedonia
    "SYR": "Asia",           # Syria
    "XKX": "Europe"          # Kosovo
}

# Update the 'continent' column with the correct values
for iso_code, continent in continent_mapping.items():
    data.loc[data['iso'] == iso_code, 'continent'] = continent

# Verify the updates
updated_continent_data = data.loc[data['iso'].isin(continent_mapping.keys()),
['iso', 'ioc', 'name', 'continent']]
updated_continent_data
```

[11]:

	iso	ioc	name	continent
17	CIV	CIV	Côte d'Ivoire	Africa
57	MDA	MDA	Moldova	Europe
59	MKD	MKD	North Macedonia	Europe
80	SYR	SYR	Syria	Asia
91	XKX	KOS	Kosovo	Europe

Fig. 'Continent' Missing Data Filling Manually



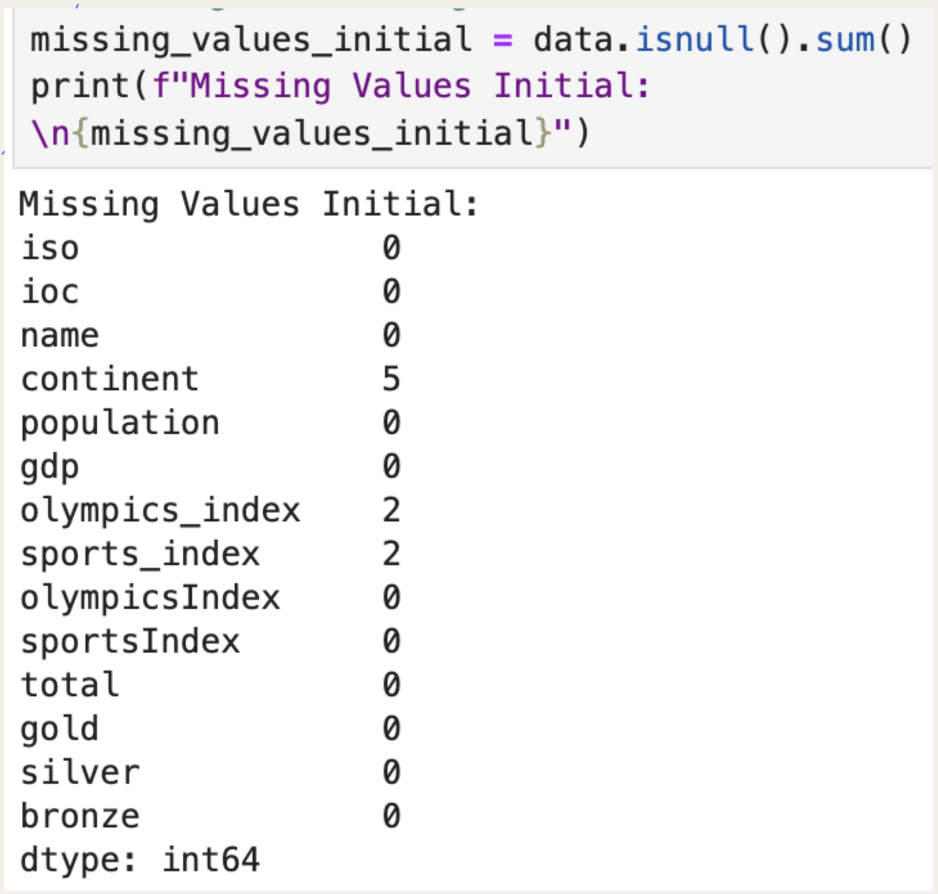
```
# handling the missing values for olympics_index and sports_index using k
nearest neighbours

from sklearn.impute import KNNImputer

# Select the columns to use for finding nearest neighbors
# We'll use 'population', 'gdp', and the available indices for imputation
imputation_features = ['population', 'gdp', 'olympics_index', 'sports_index']

# Initialize the KNN Imputer
knn_imputer = KNNImputer(n_neighbors=3)

# Perform KNN imputation on the selected columns
data[imputation_features] =
knn_imputer.fit_transform(data[imputation_features])

# Verify the imputed values
imputed_indices_values = data.loc[data['iso'].isin(['IRL', 'MKD']), ['iso',
'ioc', 'name', 'olympics_index', 'sports_index']]
imputed_indices_values
```

[12]:

	iso	ioc	name	olympics_index	sports_index
42	IRL	IRL	Ireland	11.028701	24.328794
59	MKD	MKD	North Macedonia	23.229339	10.599504

Fig. 'olympics_index' and 'sports_index' Missing Data Filling using KNN

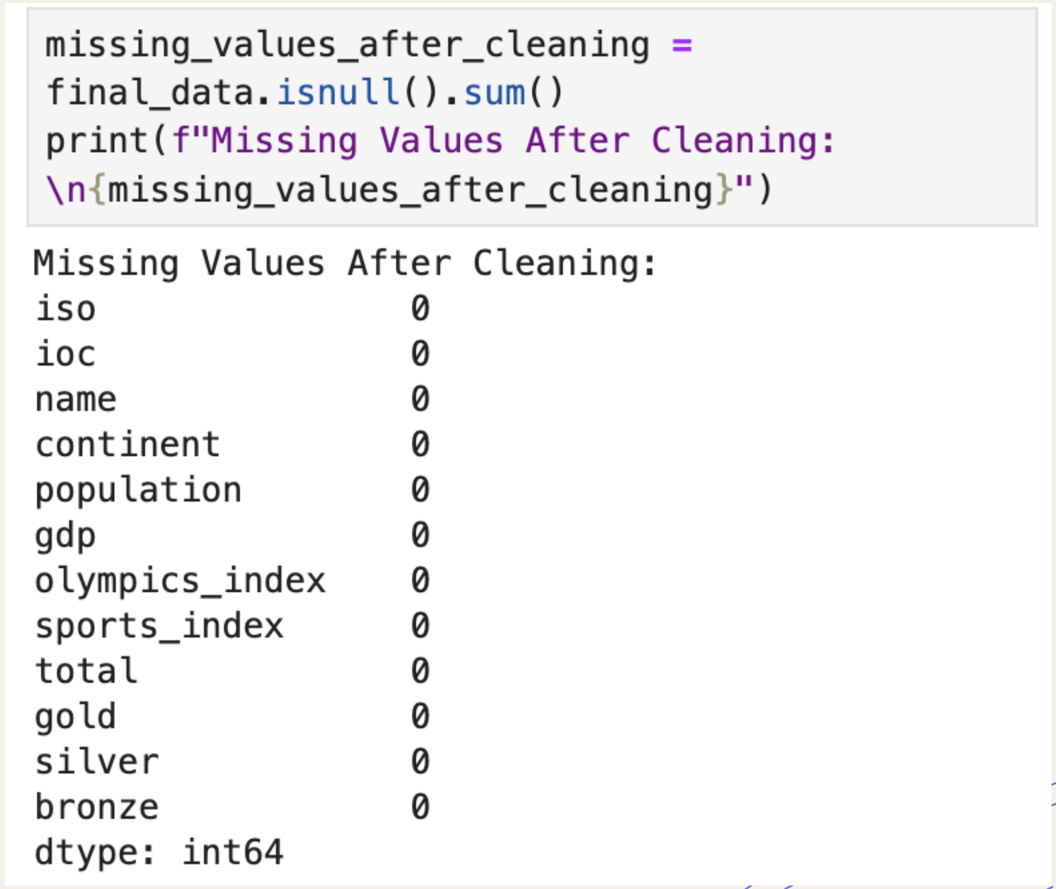
Verifying the Data Preprocessing



```
missing_values_initial = data.isnull().sum()
print(f"Missing Values Initial:
\n{missing_values_initial}")
```

```
Missing Values Initial:
iso               0
ioc               0
name              0
continent         5
population        0
gdp               0
olympics_index    2
sports_index      2
olympicsIndex     0
sportsIndex       0
total             0
gold              0
silver            0
bronze            0
dtype: int64
```

Fig. Before Data Preprocessing

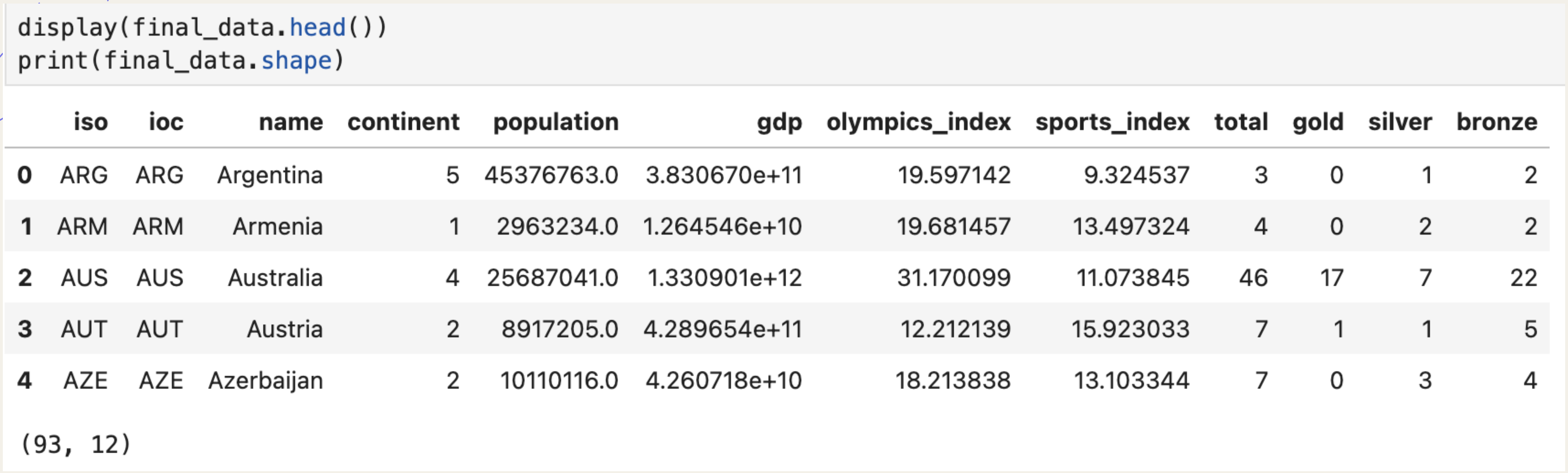


```
missing_values_after_cleaning =
final_data.isnull().sum()
print(f"Missing Values After Cleaning:
\n{missing_values_after_cleaning}")
```

```
Missing Values After Cleaning:
iso               0
ioc               0
name              0
continent         0
population        0
gdp               0
olympics_index    0
sports_index      0
total             0
gold              0
silver            0
bronze            0
dtype: int64
```

Fig. After Data Preprocessing

Verifying the Data Preprocessing



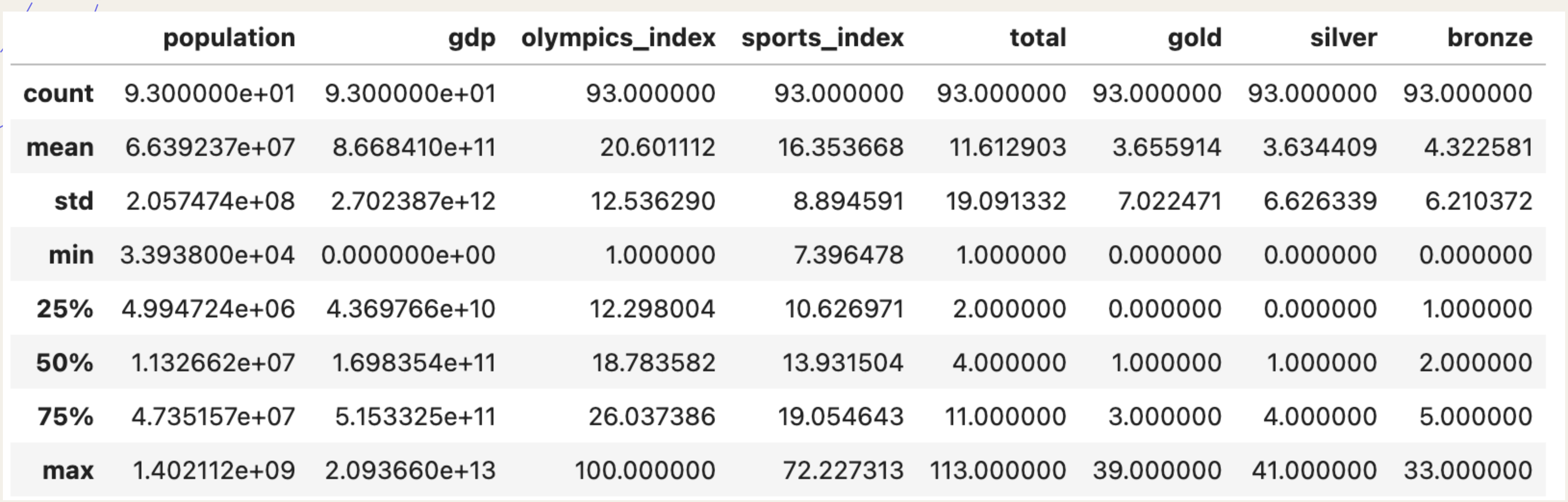
```
display(final_data.head())
print(final_data.shape)
```

	iso	ioc	name	continent	population	gdp	olympics_index	sports_index	total	gold	silver	bronze
0	ARG	ARG	Argentina	5	45376763.0	3.830670e+11	19.597142	9.324537	3	0	1	2
1	ARM	ARM	Armenia	1	2963234.0	1.264546e+10	19.681457	13.497324	4	0	2	2
2	AUS	AUS	Australia	4	25687041.0	1.330901e+12	31.170099	11.073845	46	17	7	22
3	AUT	AUT	Austria	2	8917205.0	4.289654e+11	12.212139	15.923033	7	1	1	5
4	AZE	AZE	Azerbaijan	2	10110116.0	4.260718e+10	18.213838	13.103344	7	0	3	4

```
(93, 12)
```

Fig. Final Data After Cleaning and Handling the Missing Values

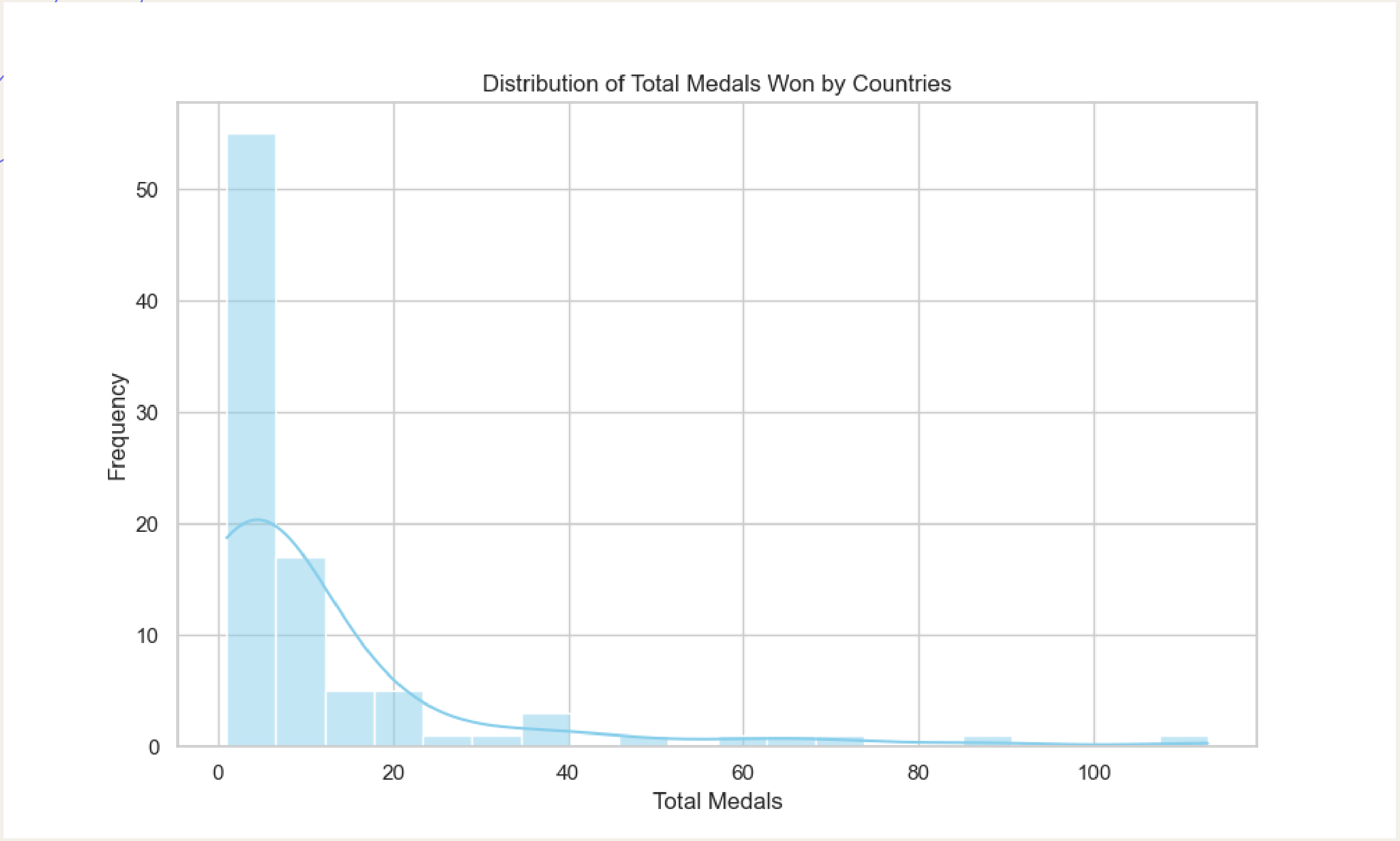
Exploratory Data Analysis (EDA)



	population	gdp	olympics_index	sports_index	total	gold	silver	bronze
count	9.300000e+01	9.300000e+01	93.000000	93.000000	93.000000	93.000000	93.000000	93.000000
mean	6.639237e+07	8.668410e+11	20.601112	16.353668	11.612903	3.655914	3.634409	4.322581
std	2.057474e+08	2.702387e+12	12.536290	8.894591	19.091332	7.022471	6.626339	6.210372
min	3.393800e+04	0.000000e+00	1.000000	7.396478	1.000000	0.000000	0.000000	0.000000
25%	4.994724e+06	4.369766e+10	12.298004	10.626971	2.000000	0.000000	0.000000	1.000000
50%	1.132662e+07	1.698354e+11	18.783582	13.931504	4.000000	1.000000	1.000000	2.000000
75%	4.735157e+07	5.153325e+11	26.037386	19.054643	11.000000	3.000000	4.000000	5.000000
max	1.402112e+09	2.093660e+13	100.000000	72.227313	113.000000	39.000000	41.000000	33.000000

Fig. Summary Statistics for Numerical Features of the Dataset

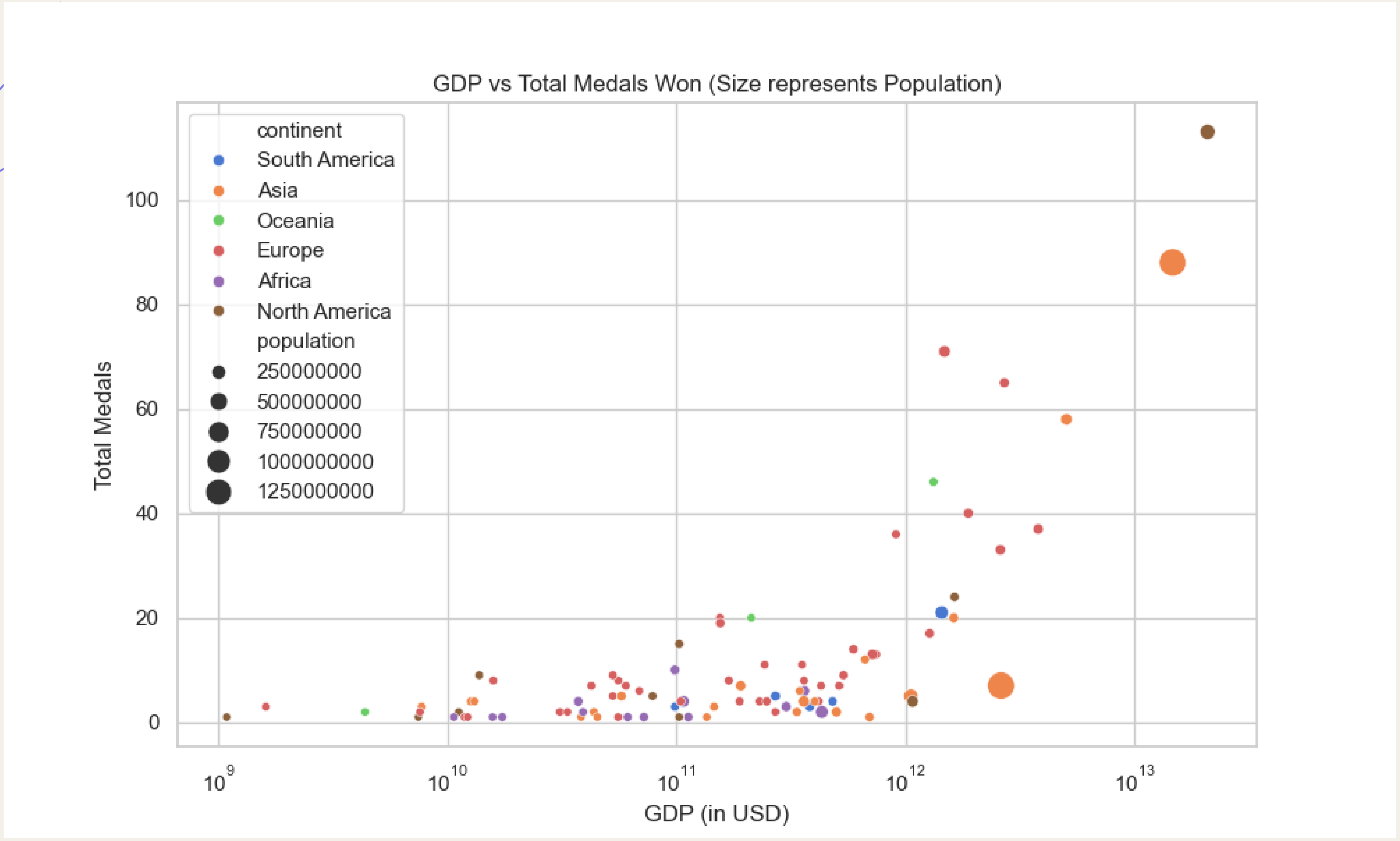
Data Visualization



Findings:

- No medals or zero medals are won by most of the time.
- 0 to 20 medals are won more frequently.
- This is because winning the medal is a rare scenario which comes at the cost of competition, countries' funds over sports department for training and maintenance, and moreover the Olympics is not conducted very frequently but only once in every four years.

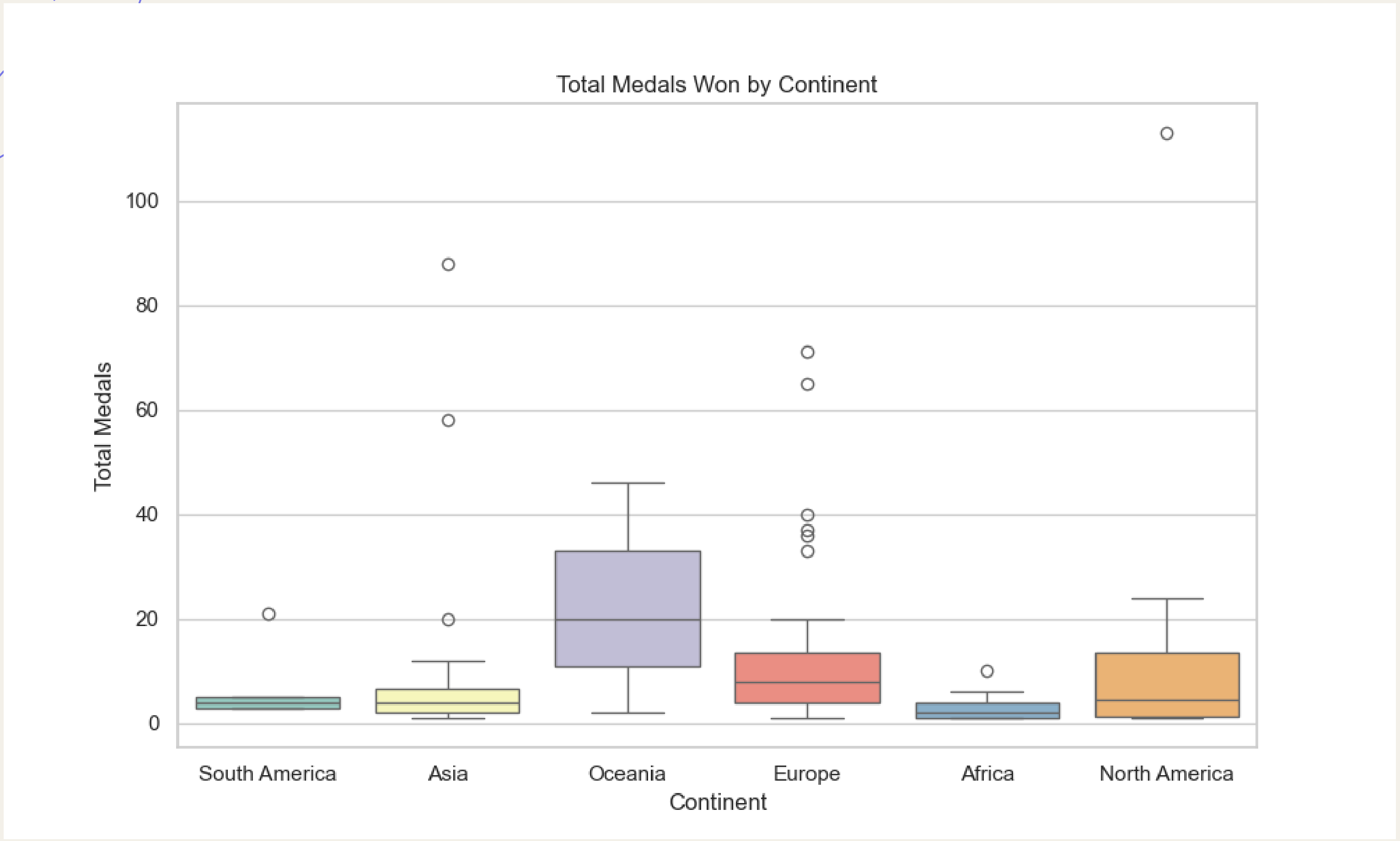
Data Visualization



Findings:

- Highest GDP Continents won more medals than lower GDP Continents.
- Hence GDP is found to be the impactful feature on number of medals won by a country.
- The graph also shows a positive relationship between population and the medals won by the country.

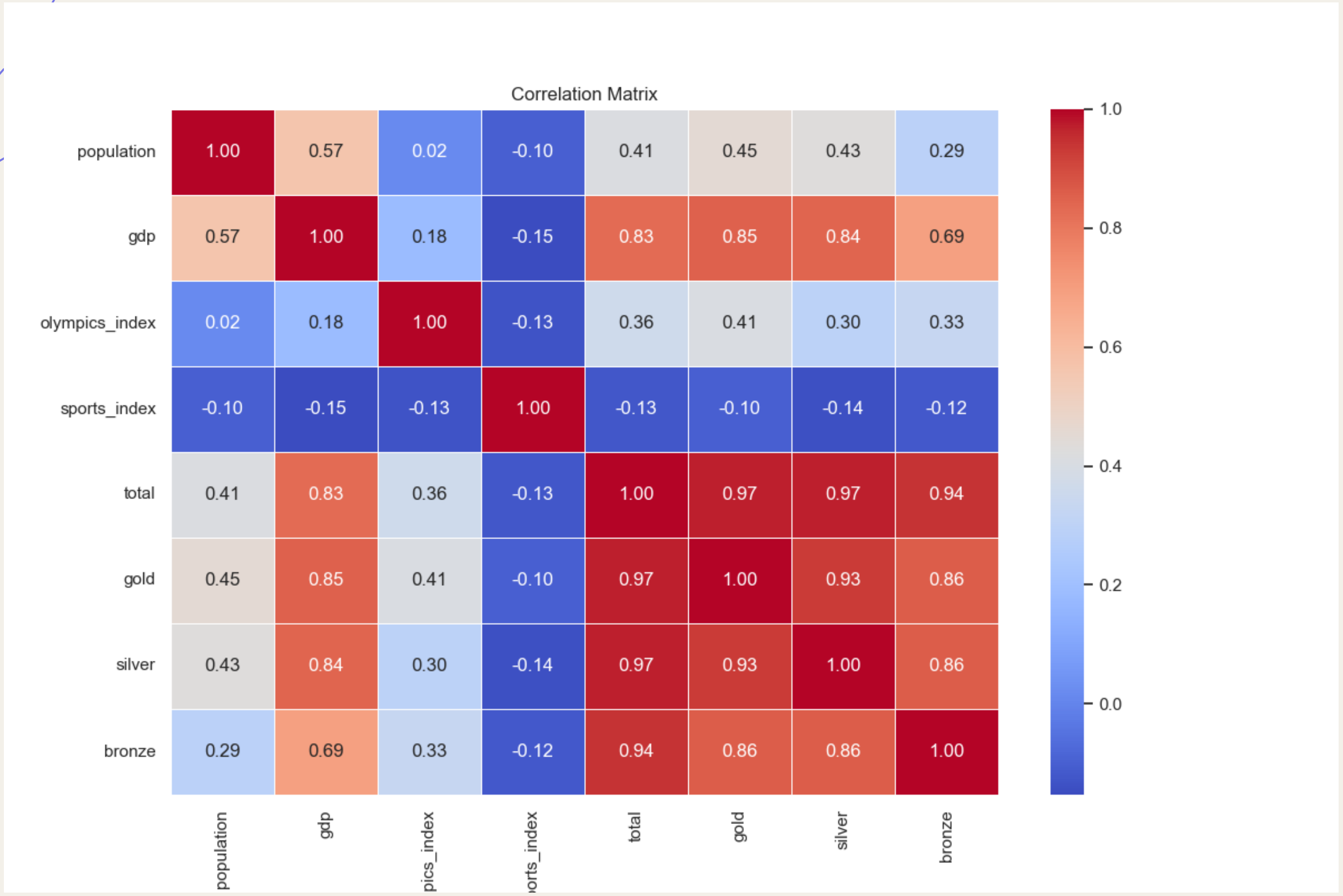
Data Visualization



Findings:

- In support with GDP vs Total Medals won graph this also shows the higher GDP continents have won more medals.
- We can find that Oceania has one the most medals but can also find huge outliers in North America, Europe, and Asia.
- South America is the least continent that won the medals.

Data Visualization



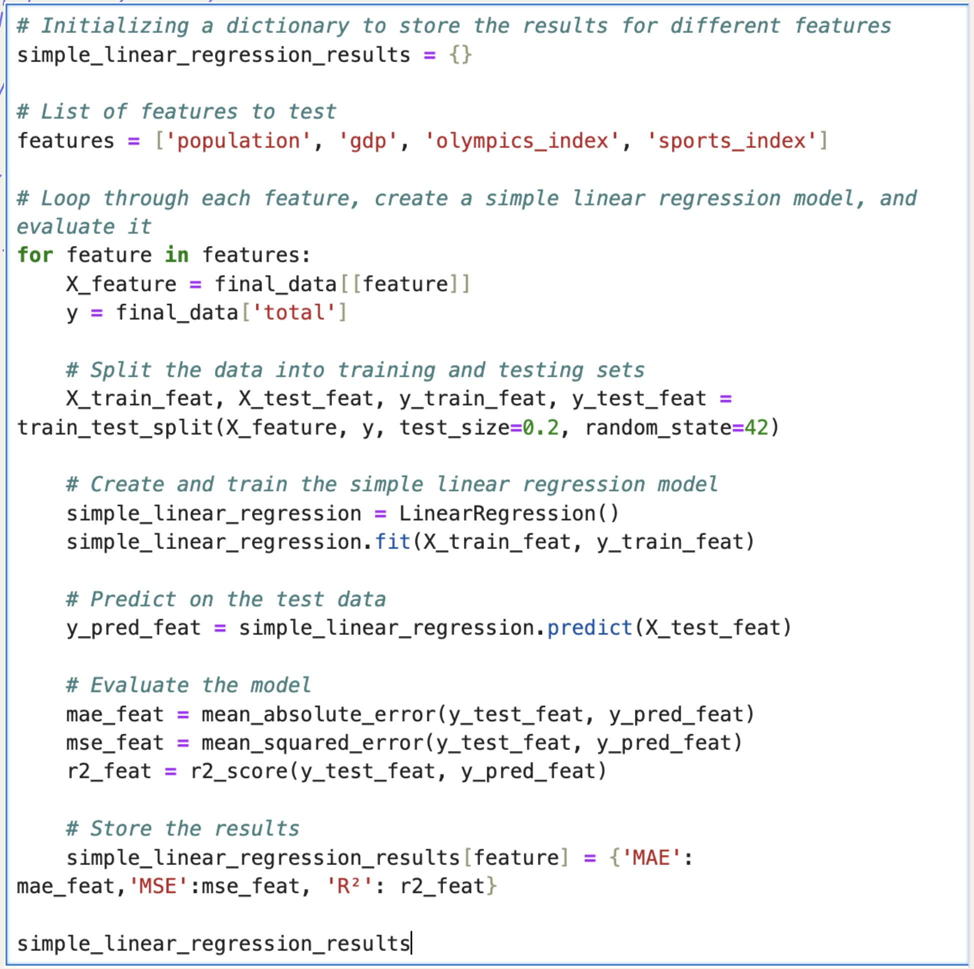
	population	gdp	pics_index	orts_index	total	gold	silver	bronze
population	1.00	0.57	0.02	-0.10	0.41	0.45	0.43	0.29
gdp	0.57	1.00	0.18	-0.15	0.83	0.85	0.84	0.69
olympics_index	0.02	0.18	1.00	-0.13	0.36	0.41	0.30	0.33
sports_index	-0.10	-0.15	-0.13	1.00	-0.13	-0.10	-0.14	-0.12
total	0.41	0.83	0.36	-0.13	1.00	0.97	0.97	0.94
gold	0.45	0.85	0.41	-0.10	0.97	1.00	0.93	0.86
silver	0.43	0.84	0.30	-0.14	0.97	0.93	1.00	0.86
bronze	0.29	0.69	0.33	-0.12	0.94	0.86	0.86	1.00

Findings:

- 0.83 that 83% relationship between GDP and Total Medals won.
- Followed by Population and Total Medals won which is 41%.
- It is shocking that there is a poor relationship between sports_index and total medal won by that country i.e., -0.13. As this index indicates country's sports infrastructure and support.

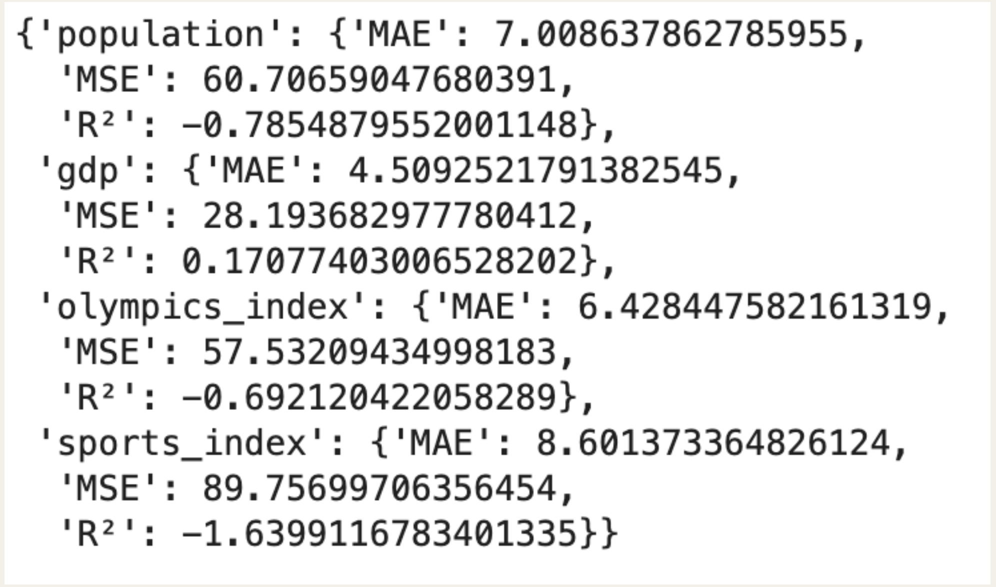
Machine Learning

Simple Linear Regression



```
# Initializing a dictionary to store the results for different features
simple_linear_regression_results = {}

# List of features to test
features = ['population', 'gdp', 'olympics_index', 'sports_index']

# Loop through each feature, create a simple linear regression model, and
evaluate it
for feature in features:
    X_feature = final_data[[feature]]
    y = final_data['total']

    # Split the data into training and testing sets
    X_train_feat, X_test_feat, y_train_feat, y_test_feat =
train_test_split(X_feature, y, test_size=0.2, random_state=42)

    # Create and train the simple linear regression model
    simple_linear_regression = LinearRegression()
    simple_linear_regression.fit(X_train_feat, y_train_feat)

    # Predict on the test data
    y_pred_feat = simple_linear_regression.predict(X_test_feat)

    # Evaluate the model
    mae_feat = mean_absolute_error(y_test_feat, y_pred_feat)
    mse_feat = mean_squared_error(y_test_feat, y_pred_feat)
    r2_feat = r2_score(y_test_feat, y_pred_feat)

    # Store the results
    simple_linear_regression_results[feature] = {'MAE':
mae_feat,'MSE':mse_feat, 'R²': r2_feat}

simple_linear_regression_results
```

Fig: Simple Linear Regression Code Snippet



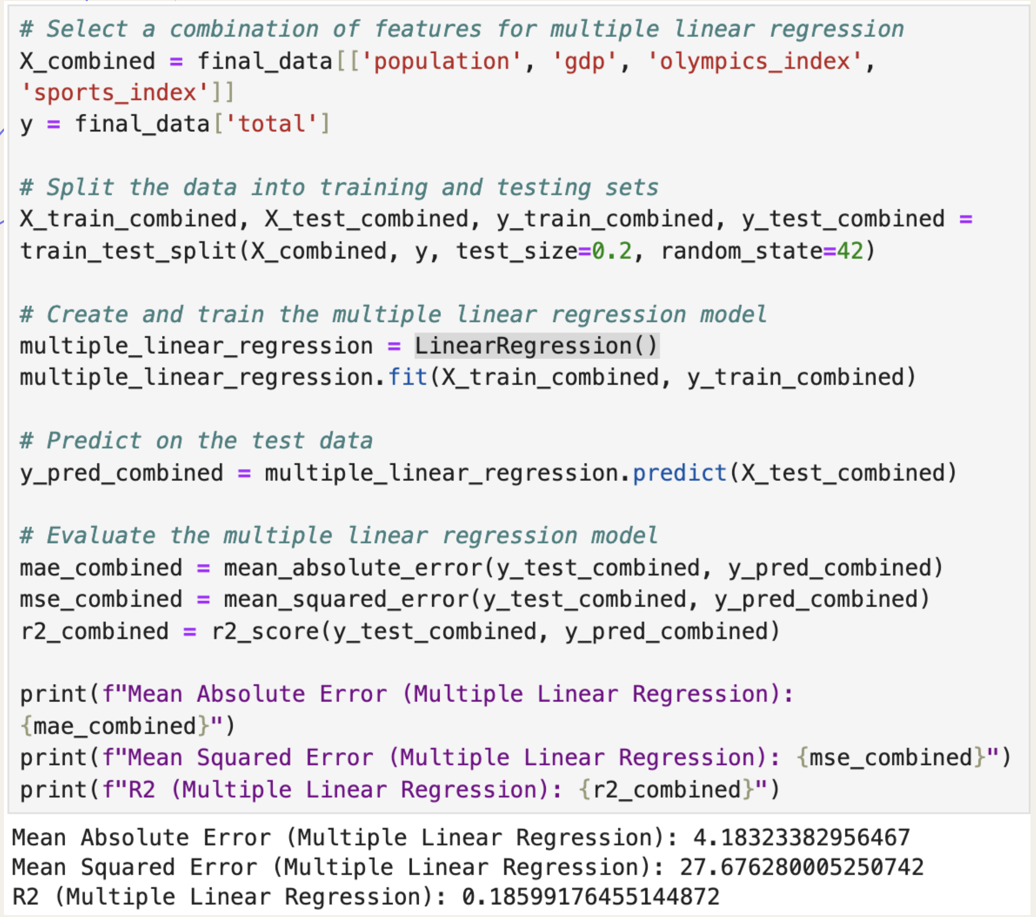
```
{'population': {'MAE': 7.008637862785955,
  'MSE': 60.70659047680391,
  'R²': -0.7854879552001148},
 'gdp': {'MAE': 4.5092521791382545,
  'MSE': 28.193682977780412,
  'R²': 0.17077403006528202},
 'olympics_index': {'MAE': 6.428447582161319,
  'MSE': 57.53209434998183,
  'R²': -0.692120422058289},
 'sports_index': {'MAE': 8.601373364826124,
  'MSE': 89.75699706356454,
  'R²': -1.6399116783401335}}
```

Fig: Simple linear Regression Output

Interpretation:

- GDP appears to be the best single predictor among the features, with a positive R² value, though the overall predictive power is still limited.
- Population and Olympics Index have slightly better performance than Sports Index, but they still do not explain much of the variance.

Multiple Linear Regression



```
# Select a combination of features for multiple linear regression
X_combined = final_data[['population', 'gdp', 'olympics_index',
'sports_index']]
y = final_data['total']

# Split the data into training and testing sets
X_train_combined, X_test_combined, y_train_combined, y_test_combined =
train_test_split(X_combined, y, test_size=0.2, random_state=42)

# Create and train the multiple linear regression model
multiple_linear_regression = LinearRegression()
multiple_linear_regression.fit(X_train_combined, y_train_combined)

# Predict on the test data
y_pred_combined = multiple_linear_regression.predict(X_test_combined)

# Evaluate the multiple linear regression model
mae_combined = mean_absolute_error(y_test_combined, y_pred_combined)
mse_combined = mean_squared_error(y_test_combined, y_pred_combined)
r2_combined = r2_score(y_test_combined, y_pred_combined)

print(f"Mean Absolute Error (Multiple Linear Regression):
{mae_combined}")
print(f"Mean Squared Error (Multiple Linear Regression): {mse_combined}")
print(f"R2 (Multiple Linear Regression): {r2_combined}")
```

```
Mean Absolute Error (Multiple Linear Regression): 4.18323382956467
Mean Squared Error (Multiple Linear Regression): 27.676280005250742
R2 (Multiple Linear Regression): 0.18599176455144872
```

Fig: Multiple Linear Regression Code Snippet and Output

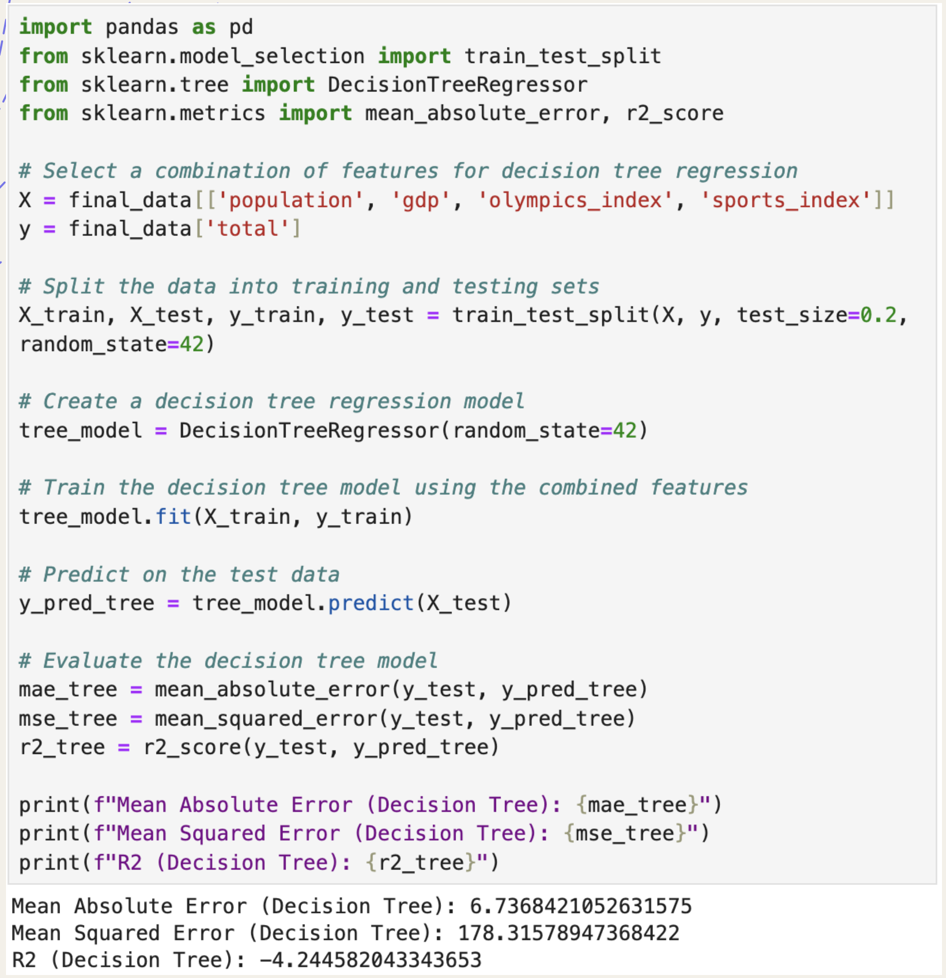
Interpretation:

- Improved MAE: The MAE has improved compared to the individual feature models, indicating that the combined model is more accurate in predicting the total number of medals.
- Positive R² Score: The R² score of 0.18 suggests that the combined model explains around 18% of the variance in the total number of medals, which is an improvement over the individual models but still leaves a lot of unexplained variance.

Conclusion:

- The multiple linear regression model using a combination of features provides a better fit and more accurate predictions than models using single features alone. However, the relatively low R² score indicates that there may be other important factors not captured by the current features.

Decision Tree Regression



```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeRegressor
from sklearn.metrics import mean_absolute_error, r2_score

# Select a combination of features for decision tree regression
X = final_data[['population', 'gdp', 'olympics_index', 'sports_index']]
y = final_data['total']

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)

# Create a decision tree regression model
tree_model = DecisionTreeRegressor(random_state=42)

# Train the decision tree model using the combined features
tree_model.fit(X_train, y_train)

# Predict on the test data
y_pred_tree = tree_model.predict(X_test)

# Evaluate the decision tree model
mae_tree = mean_absolute_error(y_test, y_pred_tree)
mse_tree = mean_squared_error(y_test, y_pred_tree)
r2_tree = r2_score(y_test, y_pred_tree)

print(f"Mean Absolute Error (Decision Tree): {mae_tree}")
print(f"Mean Squared Error (Decision Tree): {mse_tree}")
print(f"R2 (Decision Tree): {r2_tree}")
```

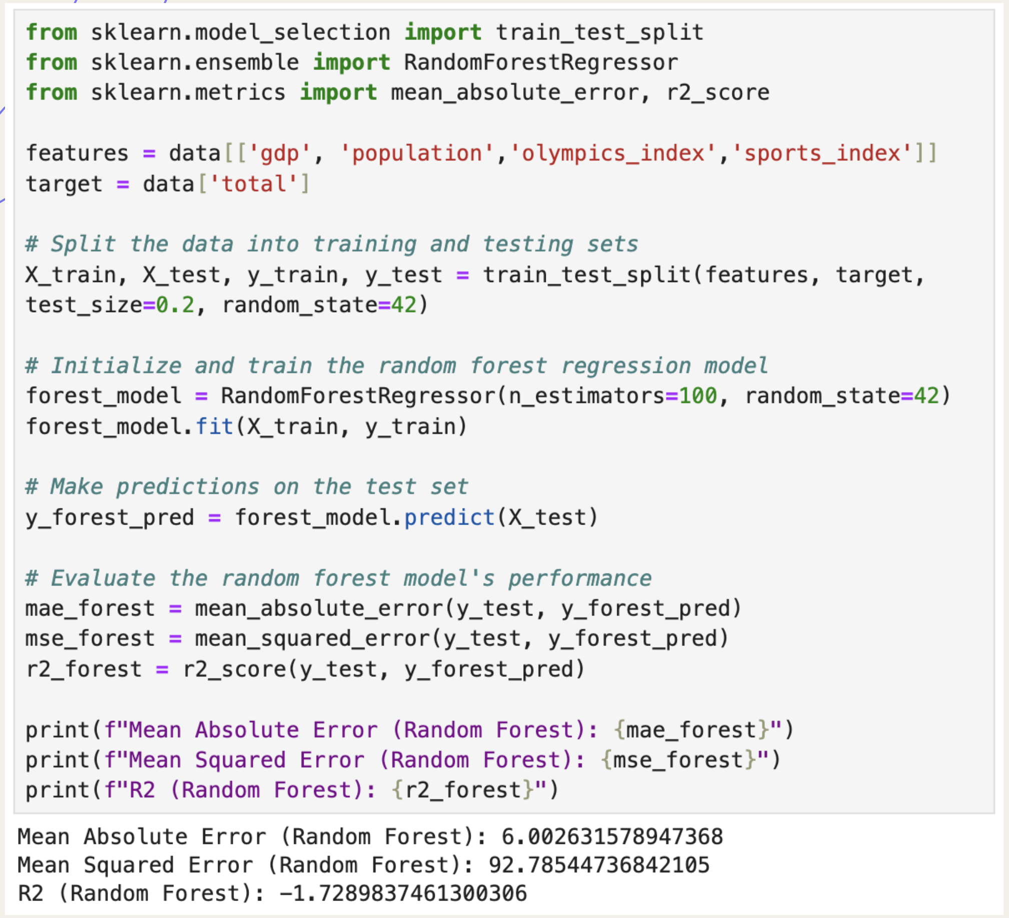
```
Mean Absolute Error (Decision Tree): 6.7368421052631575
Mean Squared Error (Decision Tree): 178.31578947368422
R2 (Decision Tree): -4.244582043343653
```

Fig: Decision Tree Regression Code Snippet and Output

Interpretation:

- The negative R² score suggests that the model is performing poorly, significantly worse than a simple mean prediction.
- Overfitting: The decision tree model may have learned the training data too well, capturing noise rather than the underlying pattern. This often leads to poor performance on unseen data.
- Non-linear Relationships: Although decision trees can model non-linear relationships, the current setup might not be optimal for this dataset, leading to poor generalization.

Random Forest Regression



```
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_absolute_error, r2_score

features = data[['gdp', 'population','olympics_index','sports_index']]
target = data['total']

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(features, target,
test_size=0.2, random_state=42)

# Initialize and train the random forest regression model
forest_model = RandomForestRegressor(n_estimators=100, random_state=42)
forest_model.fit(X_train, y_train)

# Make predictions on the test set
y_forest_pred = forest_model.predict(X_test)

# Evaluate the random forest model's performance
mae_forest = mean_absolute_error(y_test, y_forest_pred)
mse_forest = mean_squared_error(y_test, y_forest_pred)
r2_forest = r2_score(y_test, y_forest_pred)

print(f"Mean Absolute Error (Random Forest): {mae_forest}")
print(f"Mean Squared Error (Random Forest): {mse_forest}")
print(f"R2 (Random Forest): {r2_forest}")
```

```
Mean Absolute Error (Random Forest): 6.002631578947368
Mean Squared Error (Random Forest): 92.78544736842105
R2 (Random Forest): -1.7289837461300306
```

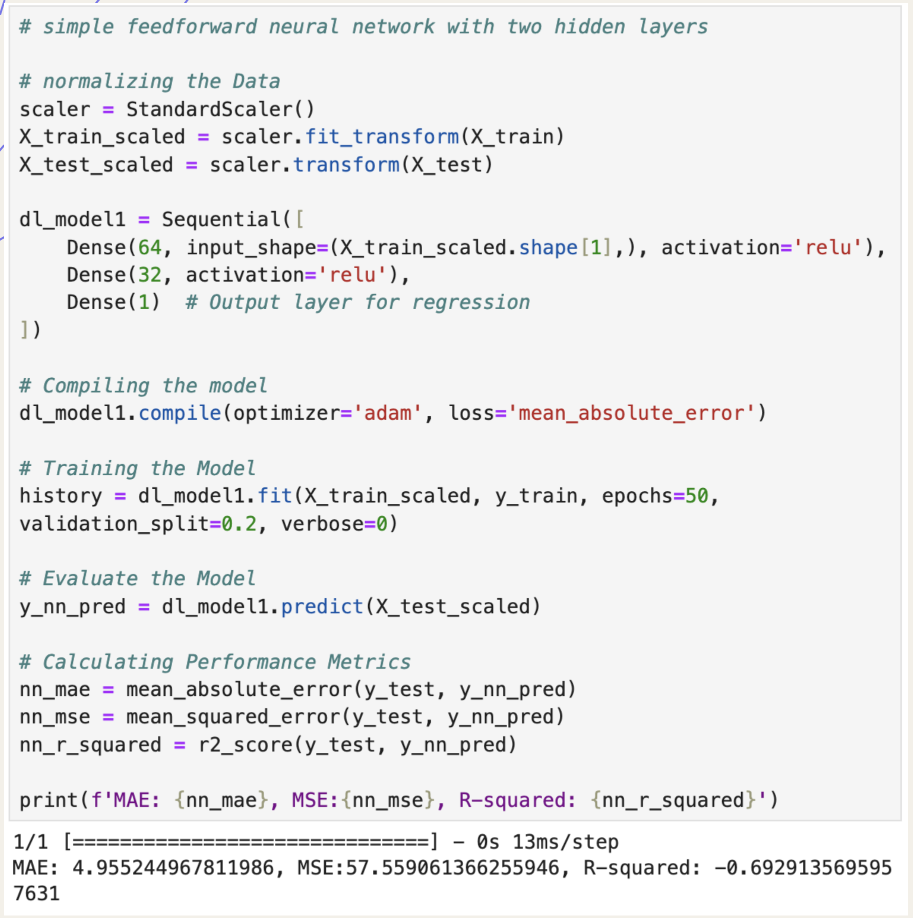
Fig: Random Regression Code Snippet and Output

Interpretation:

- Improved MAE: The MAE of 6.73 is an improvement over the decision tree model's MAE of 6.00, indicating that the random forest model is more accurate in predicting the total number of medals.
- Negative R² Score: The R² score is still negative (-1.72), which suggests that while the model has improved in accuracy, it is still not generalizing well to the test data. This could indicate that the model may still be overfitting or that the features used are not sufficiently capturing the underlying patterns.

Deep Learning

Artificial Neural Network (ANN)



```
# simple feedforward neural network with two hidden layers

# normalizing the Data
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

dl_model1 = Sequential([
    Dense(64, input_shape=(X_train_scaled.shape[1],), activation='relu'),
    Dense(32, activation='relu'),
    Dense(1)  # Output layer for regression
])

# Compiling the model
dl_model1.compile(optimizer='adam', loss='mean_absolute_error')

# Training the Model
history = dl_model1.fit(X_train_scaled, y_train, epochs=50,
validation_split=0.2, verbose=0)

# Evaluate the Model
y_nn_pred = dl_model1.predict(X_test_scaled)

# Calculating Performance Metrics
nn_mae = mean_absolute_error(y_test, y_nn_pred)
nn_mse = mean_squared_error(y_test, y_nn_pred)
nn_r_squared = r2_score(y_test, y_nn_pred)

print(f'MAE: {nn_mae}, MSE:{nn_mse}, R-squared: {nn_r_squared}')
```

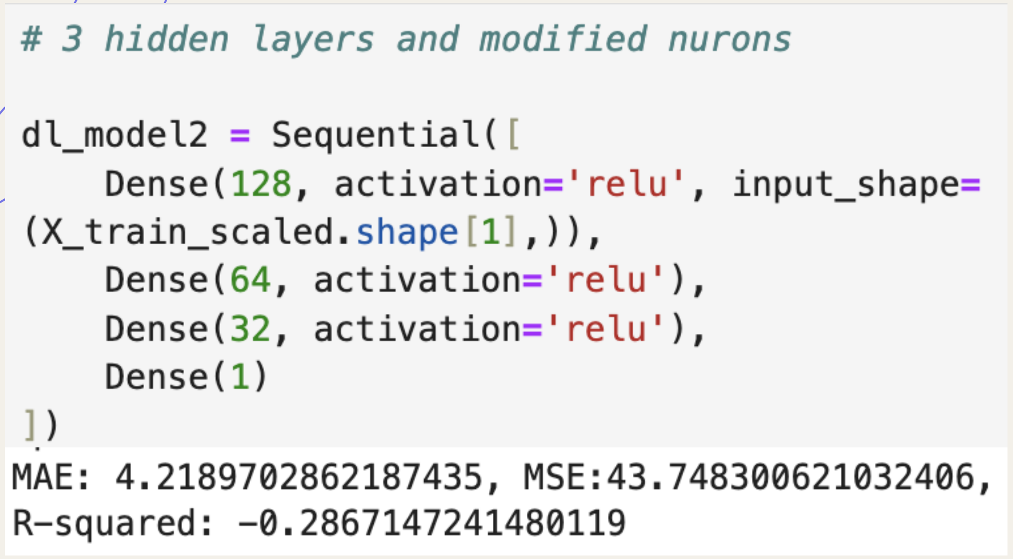
```
1/1 [==============================] - 0s 13ms/step
MAE: 4.955244967811986, MSE:57.559061366255946, R-squared: -0.692913569595
7631
```

Fig: ANN Code Snippet and Output

Interpretation:

- Increase in model performance from previous Random Forest Model i.e., MAE from 60% to 49%.
- R² score also approached near zero but still in negative value.

ANN – Experimenting with different Architectures



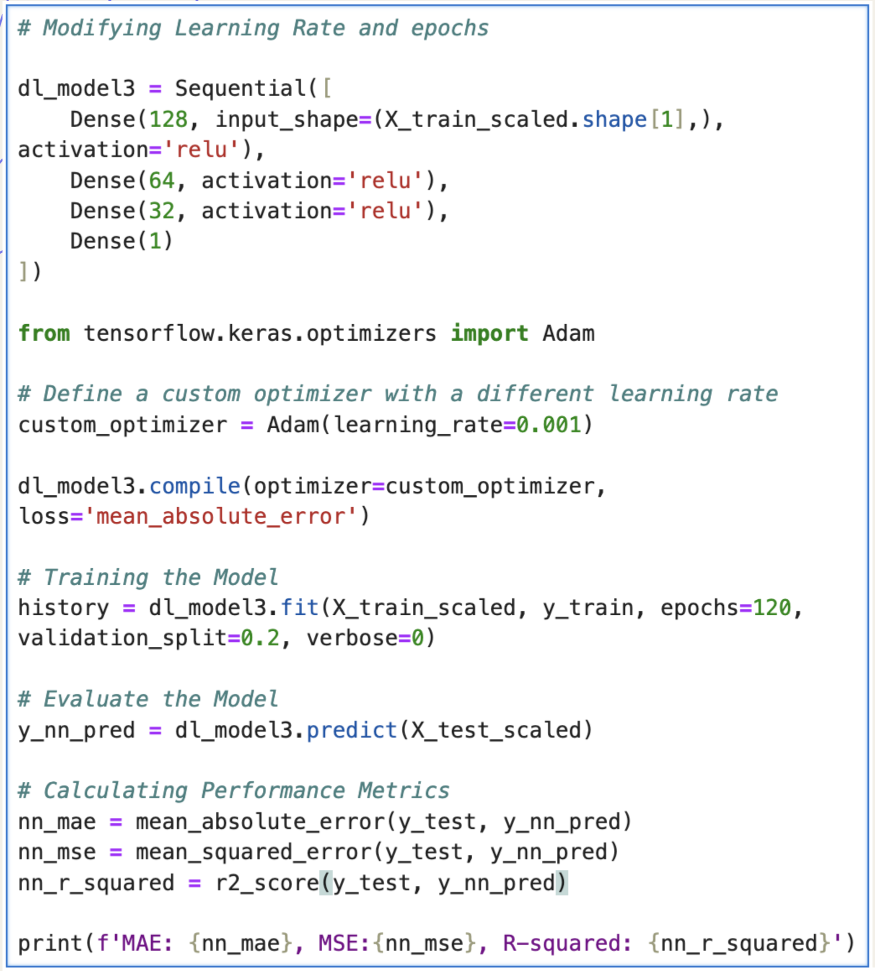
```
# 3 hidden layers and modified nurons

dl_model2 = Sequential([
    Dense(128, activation='relu', input_shape=
(X_train_scaled.shape[1],)),
    Dense(64, activation='relu'),
    Dense(32, activation='relu'),
    Dense(1)
])
```

```
MAE: 4.2189702862187435, MSE:43.748300621032406,
R-squared: -0.2867147241480119
```

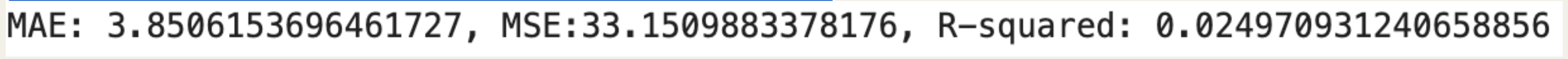
Fig. ANN with 3 hidden layers and modified nurons with its output

- We can find improved MAE and R² value
- Adding 3rd hidden layer increases the network's capacity to learn from the data. With more layers, the network can store more information, which might be necessary for accurately modeling complex datasets.
- hidden layer typically applies a non-linear transformation (e.g., ReLU activation), allowing the network to model non-linear relationships between inputs and outputs more effectively.

Hyperparameter Tuning



```
# Modifying Learning Rate and epochs

dl_model3 = Sequential([
    Dense(128, input_shape=(X_train_scaled.shape[1],),
activation='relu'),
    Dense(64, activation='relu'),
    Dense(32, activation='relu'),
    Dense(1)
])

from tensorflow.keras.optimizers import Adam

# Define a custom optimizer with a different learning rate
custom_optimizer = Adam(learning_rate=0.001)

dl_model3.compile(optimizer=custom_optimizer,
loss='mean_absolute_error')

# Training the Model
history = dl_model3.fit(X_train_scaled, y_train, epochs=120,
validation_split=0.2, verbose=0)

# Evaluate the Model
y_nn_pred = dl_model3.predict(X_test_scaled)

# Calculating Performance Metrics
nn_mae = mean_absolute_error(y_test, y_nn_pred)
nn_mse = mean_squared_error(y_test, y_nn_pred)
nn_r_squared = r2_score(y_test, y_nn_pred)

print(f'MAE: {nn_mae}, MSE:{nn_mse}, R-squared: {nn_r_squared}')
```



```
MAE: 3.8506153696461727, MSE:33.1509883378176, R-squared: 0.024970931240658856
```

Fig. Further Modification in Learning Rate and Epochs

- The learning rate was set to 0.001 and epochs was set to 120 from 100.
- We can encounter the better MAE i.e., 38% and positive R^2 score.
- learning rate is a hyperparameter that determines the size of the steps the model takes when updating the weights in response to the calculated gradient during training. Reducing the learning rate as training progresses, can help the model fine-tune the weights as it gets closer to the optimal solution.
- The number of epochs is the total number of passes the model makes through the data during training. The optimal number of epochs allows the model to learn enough from the data to generalize well to new data without overfitting.

Model Comparison

Models	MAE	MSE	R^2
ANN Learning rate = 0.001 Epochs = 120	3.85	33.15	0.024
Multi Linear Regression	4.18	27.68	0.19
ANN 3 hidden layers	4.22	43.75	-0.29
ANN 2 hidden layers	4.95	57.55	-0.69
Random Forest Regression	6.00	92.79	-1.73
Decision Tree Regression	6.74	178.31	-4.24

Model Comparison

- By above table we can conclude that the ANN model after Hyperparameter tuning performs better than among all the models we tested.
- Decision Tree Regression Model performs poorly among all.
- Random Forest Regression Model performs better than Decision Tree Regression Model.

Feature Importance

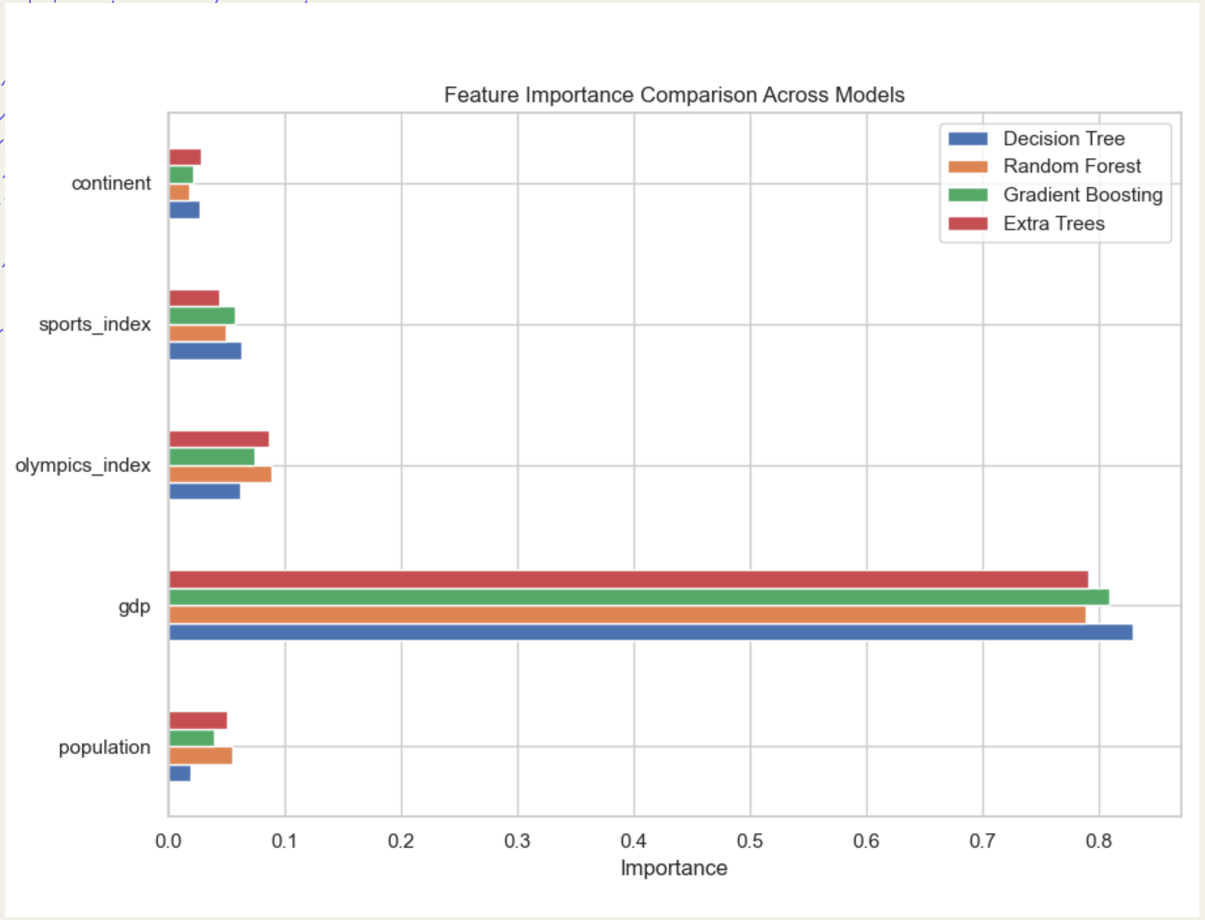


Fig. Feature Importance by Machine Learning Models

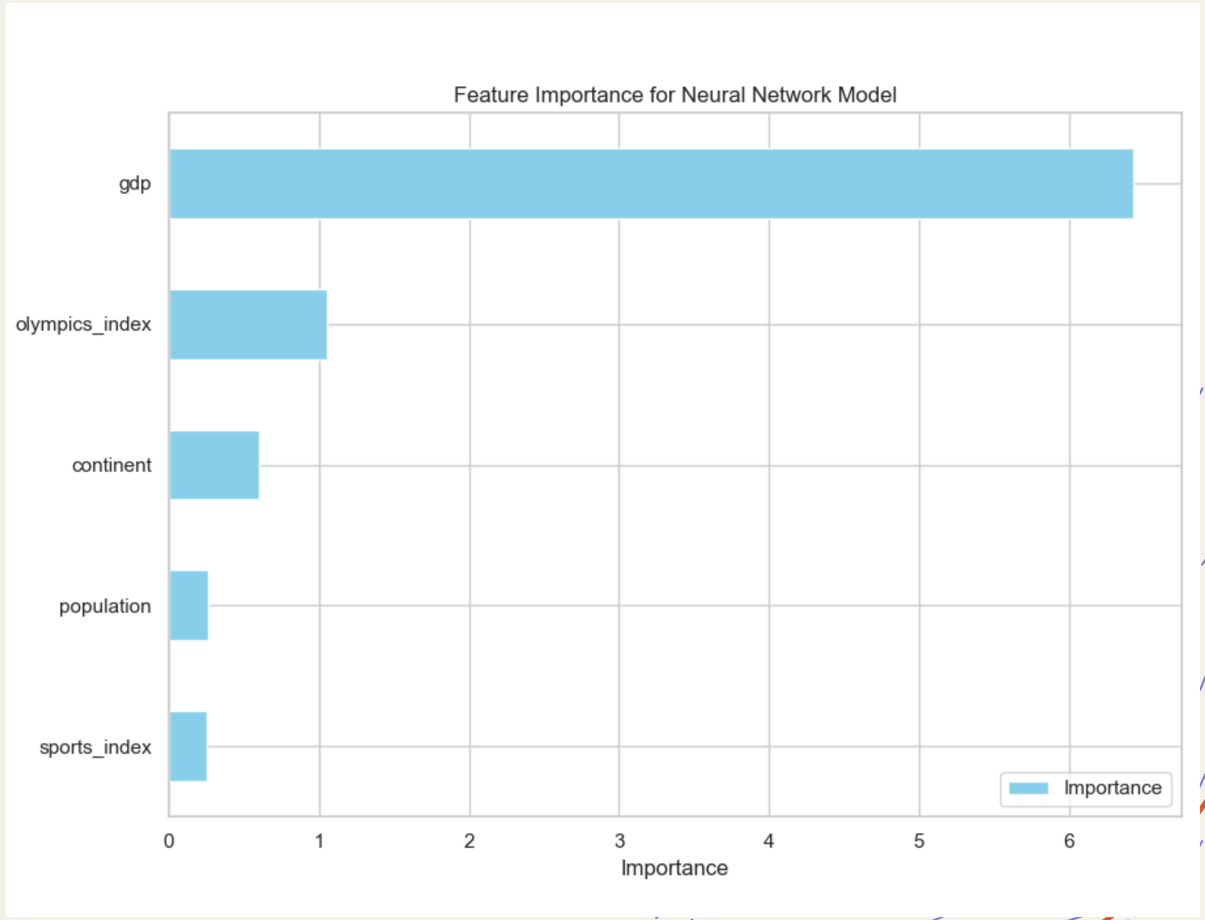


Fig. Feature Importance by Neural Network Model

Feature Importance

Findings:

- GDP is the most significant factor in determining a country’s medal-winning potential
- followed by the Olympics Index, which reflects overall sports performance, and then population size.

Model Interpretation

- We can conclude that countries with a higher GDP tend to win a greater number of Olympic medals. As a country's GDP increases, more funds can be allocated across various sectors, including sports. This enables the government to invest more in sports infrastructure, equipment, coaching, and overall athletic development. Consequently, these improvements can enhance the country's performance in the Olympics, potentially leading to the winning of more medals.
- We can conclude that, in addition to GDP, a country's overall performance in sports, as indicated by the Olympics Index, is the second most important factor in winning Olympic medals. A higher Olympics Index reflects a strong and consistent performance across various sports disciplines. As a country's GDP increases, it enables more investment in sports infrastructure, coaching, and athlete development, which in turn improves the country's overall sports performance. This enhanced performance, as captured by the Olympics Index, significantly contributes to the likelihood of winning more Olympic medals.

Way Forward

To enhance a country's chances of winning more Olympic medals, it is crucial to focus on the following areas:

1. Economic Growth: Continue to strengthen the economy, as a higher GDP enables greater investment in sports infrastructure, training facilities, and athlete support programs.
2. Investment in Sports Performance: Prioritize improving the Olympics Index by investing in talent identification, development programs, and international competition exposure. This will help to elevate the overall sports performance of the country.
3. Population Engagement: encouraging widespread participation in sports at the grassroots level. This can be achieved through initiatives that promote physical education, community sports programs, and national sports campaigns to discover and nurture talent from a larger pool.
4. Strategic Allocation of Resources: Allocate resources strategically across different sports, focusing on areas where the country has traditionally excelled, while also identifying and supporting emerging sports with high medal potential.

Thank you