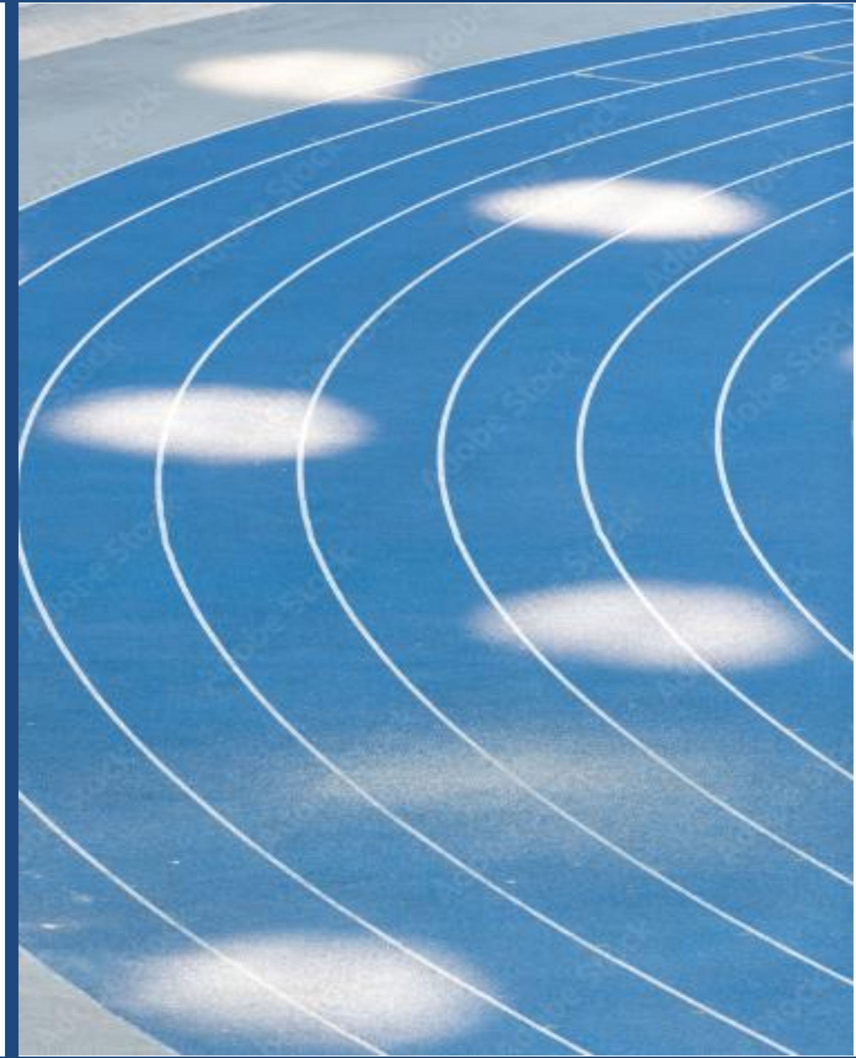
ML-DL PROJECT PREDICTING OLYMPIC MEDAL COUNTS

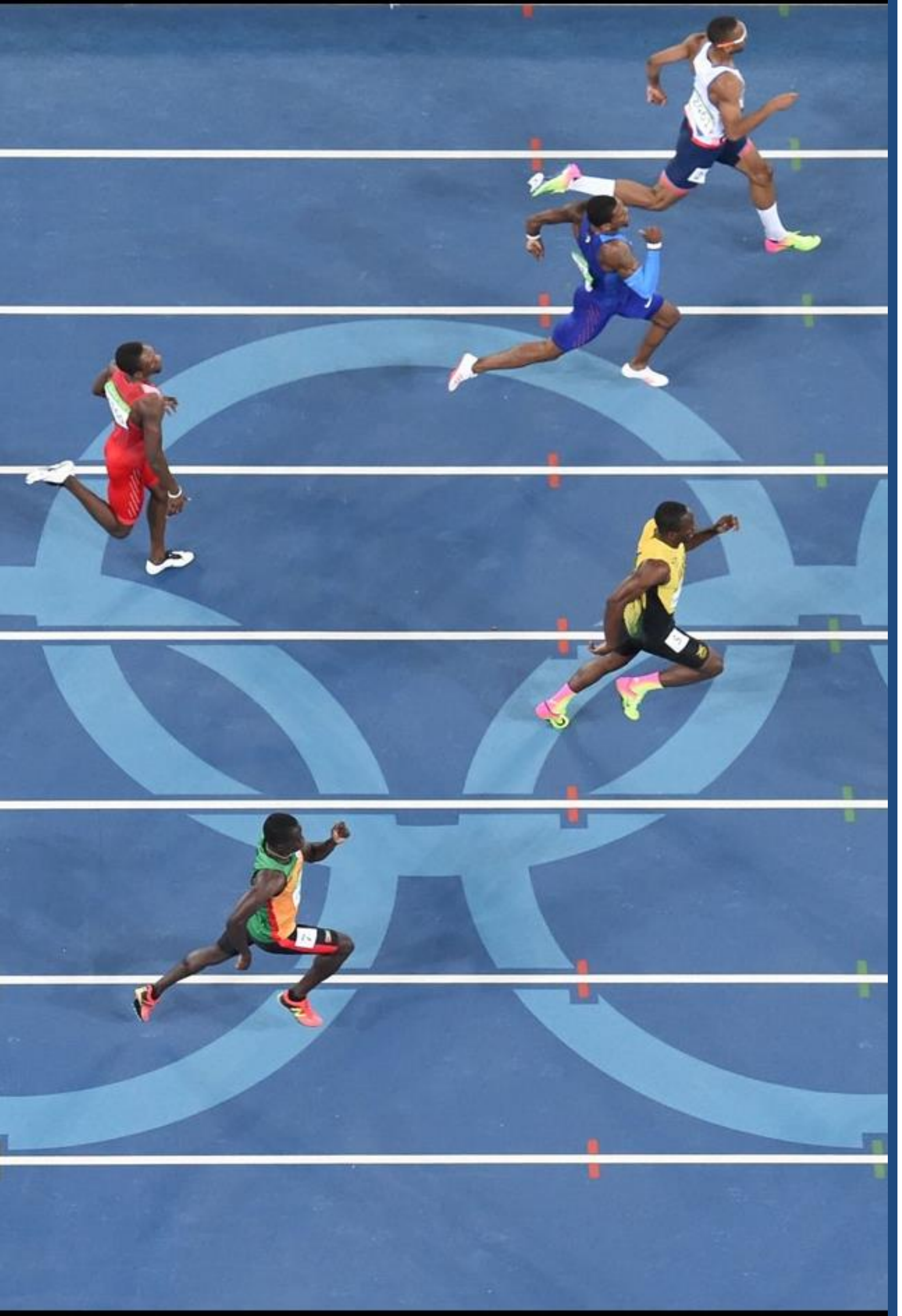
ASSIGNMENT 4

- By Rina Irene Rafalski
- Intern Code: OGTIPIRDS835
- Mentor: Manisha Anand

Table of contents

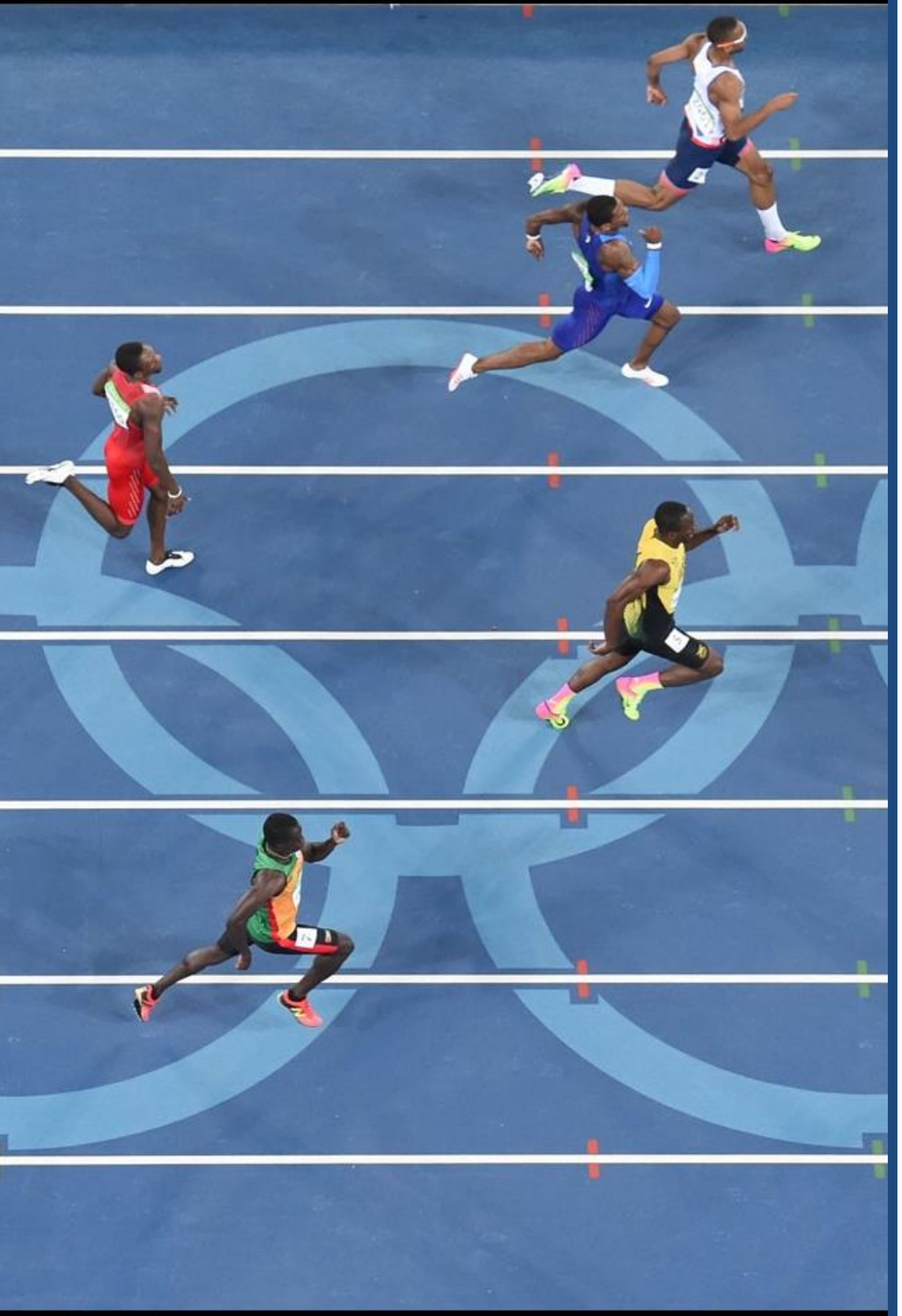
- Project Overview ……………………………………….…….…..…….3
- Dataset Description …………………………………………..…. 4
- 1. Data Pre-processing …………………………………….…. 5
- 2. Exploratory Data Analysis ……………………………… 9
- 3. Machine Learning Models ……………………………… 13
- 4. Deep Learning Models …………………………………… 17
- 5. Model Evaluation ……………….……………………….. 20
- 6. Interpretation and Insights ……………………….… 22





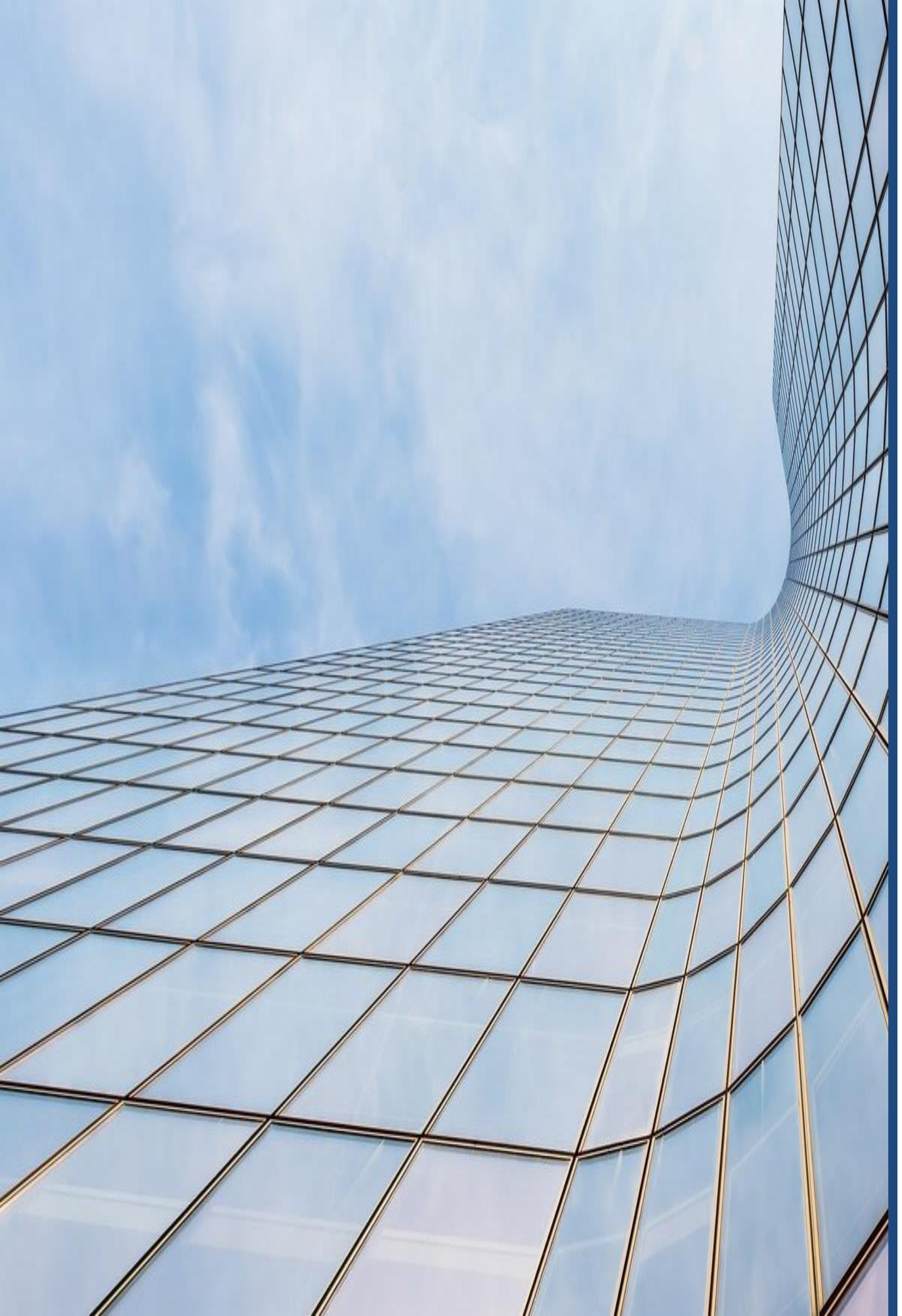
Project Overview

- In this project, you will be explored basic machine learning (ML) and deep learning (DL) techniques to predict the number of Olympic medals a country will win. The dataset provided includes features such as GDP, population, and sports index, along with the actual number of medals won. You will build and evaluate different models to understand which factors are most influential in predicting Olympic success.



Dataset Description

- The dataset includes 14 columns:
  - **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



1. Data Preprocessing

Data Preprocessing

Data observation

- The shape of this dataframe is 93 observations with 14 features

- Columns with null (missing) values:
  1) continent: 5
  2) olympics_index: 2
  3) sports_index: 2

- Redundant columns (columns with identical values except for 2 that are missing):
  1) olympics_index | olympicsIndex
  2) sports_index | sportsIndex

- Columns with 0 (incorect) values
  1) gdp
  2) olympics_index
  3) sports_index

```
Shape of this dataset is (93, 14).
==================================================
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 93 entries, 0 to 92
Data columns (total 14 columns):
 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
 0   iso             93 non-null     object
 1   ioc             93 non-null     object
 2   name            93 non-null     object
 3   continent       88 non-null     object
 4   population      93 non-null     int64
 5   gdp             93 non-null     int64
 6   olympics_index  91 non-null     float64
 7   sports_index    91 non-null     float64
 8   olympicsIndex   93 non-null     float64
 9   sportsIndex     93 non-null     float64
 10  total           93 non-null     int64
 11  gold            93 non-null     int64
 12  silver          93 non-null     int64
 13  bronze          93 non-null     int64
dtypes: float64(4), int64(6), object(4)
memory usage: 10.3+ KB
None
==================================================
```

```
       olympicsIndex  sportsIndex
count      93.000000    93.000000
mean       20.232746    15.978095
std        12.852103     9.150623
min         0.000000     0.000000
25%        12.212139    10.607469
50%        18.213838    13.891772
75%        26.037386    18.984764
max       100.000000    72.227313
```

```
         population           gdp
count  9.300000e+01  9.300000e+01
mean   6.639237e+07  8.668410e+11
std    2.057474e+08  2.702387e+12
min    3.393800e+04  0.000000e+00
25%    4.994724e+06  4.369766e+10
50%    1.132662e+07  1.698354e+11
75%    4.735157e+07  5.153325e+11
max    1.402112e+09  2.093660e+13
```

Data Preprocessing

Data Cleaning

- **Handling the redundant columns**
  1) Combine the 'olympics_index' and 'olympicsIndex' columns
  2) Combine the 'sports_index' and 'sportsIndex' columns
  3) Drop the original redundant columns
  4) Rename the combined columns to the original names

olympics_index	sports_index	olympicsIndex	sportsIndex
19.597142	9.324537	19.597142	9.324537
19.681457	13.497324	19.681457	13.497324
31.170099	11.073845	31.170099	11.073845
12.212139	15.923033	12.212139	15.923033
18.213838	13.103344	18.213838	13.103344

- **Handling the missing continents**
  1) Rename the 'name' column to 'country'
  2) Retrieve the 'country' values where 'continent' is missing
  3) Mapping of countries to their respective continents
  4) Fill in missing 'continent' values based on the 'country' column

```
# Retrieve the 'country' values where 'continent' is missing
missing_continent_countries = df[df['continent'].isnull()]['country']

missing_continent_countries.tolist()
```

```
["Côte d'Ivoire", 'Moldova', 'North Macedonia', 'Syria', 'Kosovo']
```

- **Handling 0 value of GDP**
  1) Filter the countries with GDP of 0
  2) Update the gdp for Syria with value from google search

```
# Filter the countries with GDP of 0
countries_with_zero_gdp = df[df['gdp'] == 0]['country']
countries_with_zero_gdp
```

```
80    Syria
Name: country, dtype: object
```

- **Handling 0 value of olympics_index and sports_index**
  1) Filter the countries with olympics_index of 0
  2) Remove rows where 'olympics_index' and 'sports_index' is 0

```
# Filter the countries with olympics_index of 0
countries_with_zero_olympics_index = df[(df['olympics_index'] == 0) | (df['sports_index'] == 0)]

countries_with_zero_olympics_index
```

	iso	ioc	country	continent	population	gdp	total	gold	silver	bronze	olympics_index	sports_index
42	IRL	IRL	Ireland	Europe	4994724	418621818482	4	2	0	2	0.0	0.0
59	MKD	MKD	North Macedonia	Europe	2083380	12266949805	1	0	1	0	0.0	0.0

Data Preprocessing

Data Feature Engineering

- Gross domestic product (GDP) is a measurement that describes the value of a geographic location’s total goods and services, and how it relates to the population of the region. GDP per capita is an evolution of this metric, it is obtained by dividing a country's GDP by its population.

Data Normalization

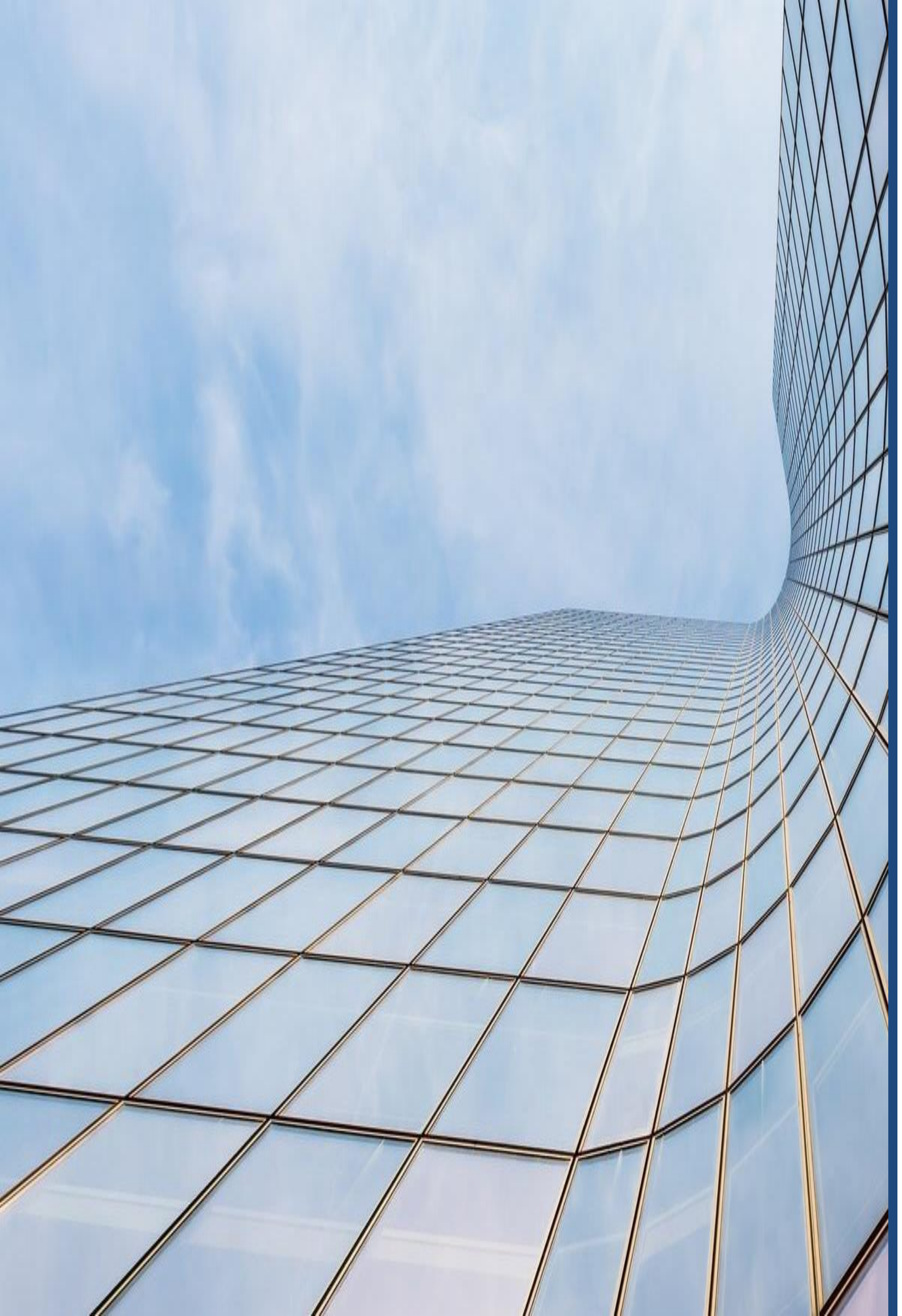
- Normalizing the dataset’s numerical features to ensure that all features contribute equally to the model training process and that the optimization algorithms perform effectively.

```
# Create a new feature 'GDP per capita' by dividing GDP by population.
# It is a measure of the relative health of that country’s overall economy and industry.

cleaned_df = cleaned_df.copy()
cleaned_df['gdp_per_capita'] = cleaned_df['gdp'] / cleaned_df['population']
```

```
# Data Normalization
# ensure that all features contribute equally to the model training process
# and that the optimization algorithms perform effectively.
scaler = StandardScaler()
features = ['gdp',
            'population',
            'gdp_per_capita',
            'olympics_index',
            'sports_index']
cleaned_df[features] = scaler.fit_transform(cleaned_df[features])
cleaned_df[features]
```

	gdp	population	gdp_per_capita	olympics_index	sports_index
0	-0.183479	-0.108376	-0.559011	-0.085998	-0.788494
1	-0.319902	-0.313609	-0.751413	-0.079286	-0.318780
2	0.165600	-0.203652	1.439932	0.835289	-0.591581
3	-0.166575	-0.284799	1.269086	-0.673895	-0.045728
4	-0.308867	-0.279026	-0.753862	-0.196118	-0.363129
...	...	...	...	...	...
88	7.386210	1.266381	1.980636	0.436569	-0.736330
89	-0.303306	-0.162304	-0.870403	0.330597	-0.528841
90	-0.146895	-0.190350	-0.166206	0.354589	-0.641884
91	-0.321756	-0.319357	-0.750502	-1.010732	0.691132
92	-0.213363	-0.040961	-0.713468	-0.731041	-0.256176

91 rows × 5 columns



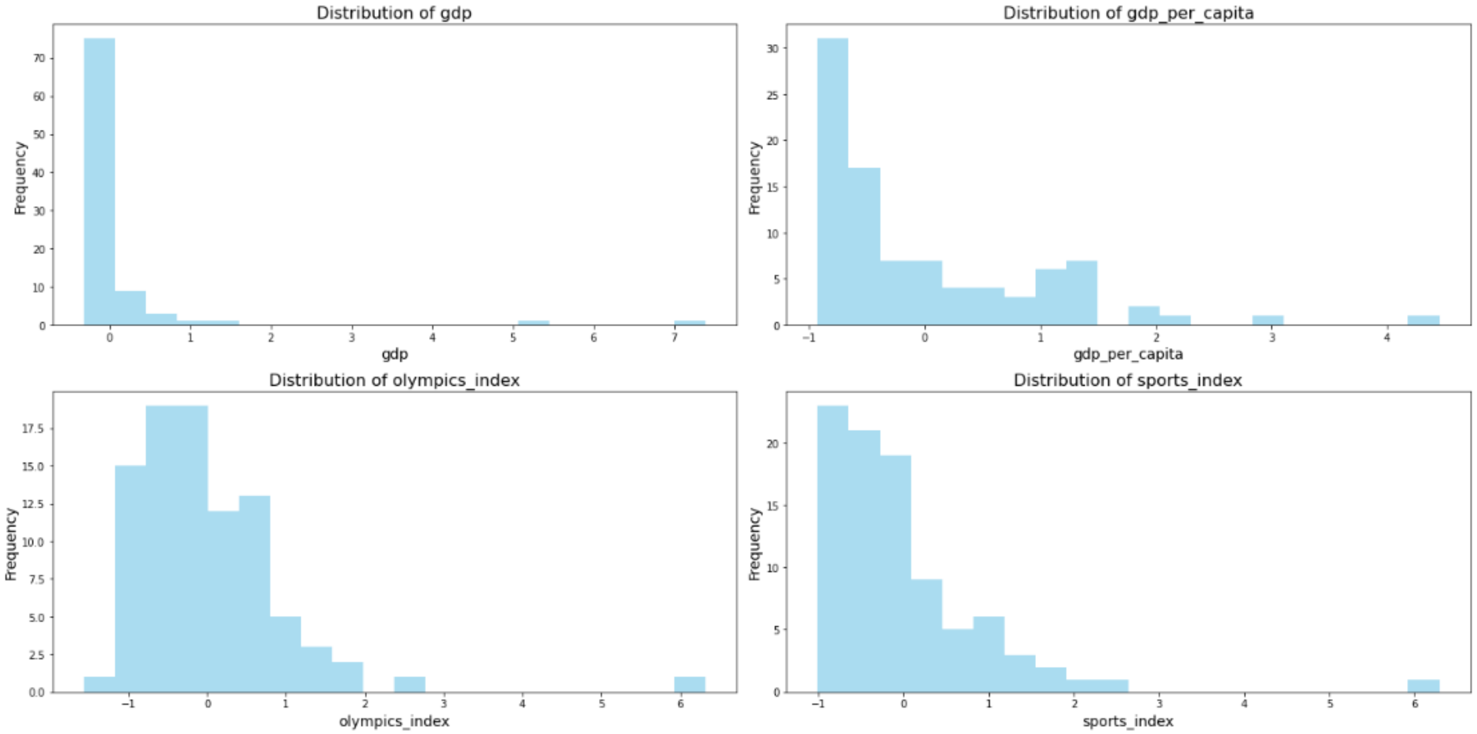
2. Exploratory Data Analysis (EDA)

Exploratory Data Analysis (EDA)

- **Descriptive Statistics**

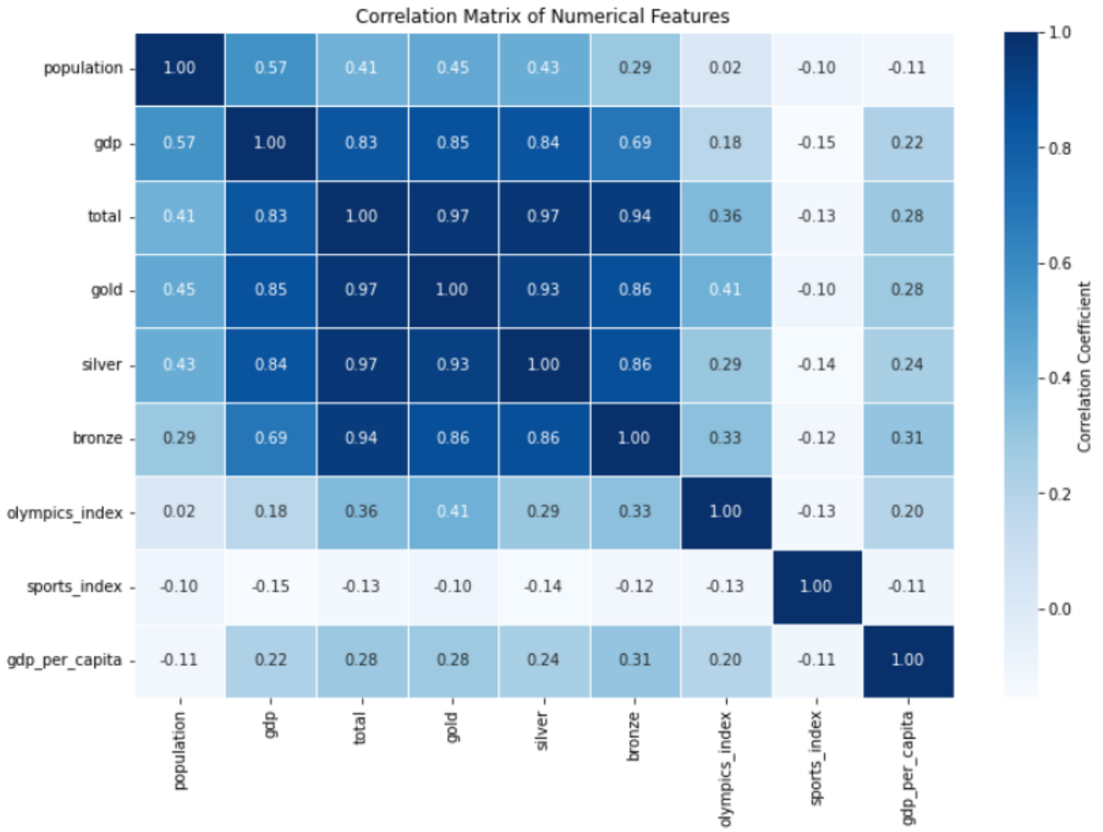
	total	gdp	population	gdp_per_capita	olympics_index	sports_index
count	91.000000	9.100000e+01	9.100000e+01	9.100000e+01	9.100000e+01	9.100000e+01
mean	11.813187	1.891039e-17	9.912706e-18	3.782078e-17	-7.320152e-18	6.710139e-18
std	19.252078	1.005540e+00	1.005540e+00	1.005540e+00	1.005540e+00	1.005540e+00
min	1.000000	-3.241578e-01	-3.277837e-01	-9.244767e-01	-1.566459e+00	-1.005528e+00
25%	2.000000	-3.081830e-01	-3.026316e-01	-7.507967e-01	-6.354884e-01	-6.197658e-01
50%	5.000000	-2.620100e-01	-2.720301e-01	-4.736188e-01	-1.507628e-01	-2.699060e-01
75%	11.000000	-1.306642e-01	-9.027641e-02	5.168352e-01	4.316293e-01	3.028522e-01
max	113.000000	7.386210e+00	6.456680e+00	4.449835e+00	6.314625e+00	6.292219e+00

- **Visualizations**



Exploratory Data Analysis (EDA)

Correlation Matrix of Numerical Features



Feature Analysis

- GDP shows a strong positive correlation Total Medals Won (0.83), indicating that countries with higher GDPs tend to win more medals.
- Population has a moderate positive correlation (0.41), suggesting some relationship but not as strong as GDP.
- The olympics_index has a moderate positive correlation (0.36) with Total Medals Won.
- GDP per capita measures (normalized, standardized) show weak positive correlations (0.28) with total medal counts.
- The sports_index shows very weak negative correlations (-0.13) total with medal counts.
- In summary, GDP shows the strongest correlation with Total Medals Won, followed by population to a lesser extent.

Exploratory Data Analysis (EDA)

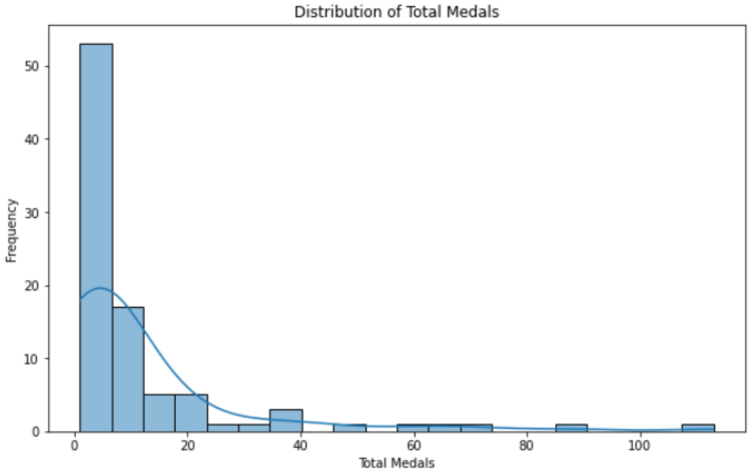
Descriptive Statistics of Total Medals Won

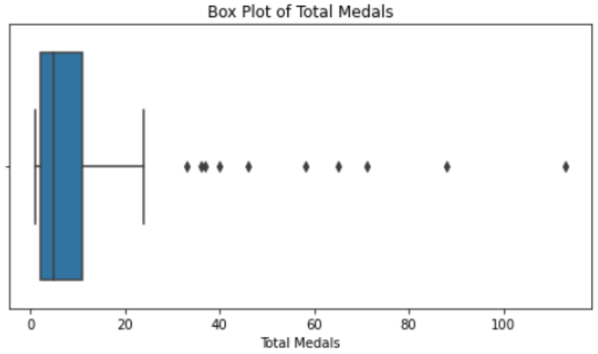
```python
# Descriptive statistics for the 'total' medals column
total_medals_stats = cleaned_df['total'].describe()
print(total_medals_stats)
```

```
count     91.000000
mean      11.813187
std       19.252078
min        1.000000
25%        2.000000
50%        5.000000
75%       11.000000
max      113.000000
Name: total, dtype: float64
```

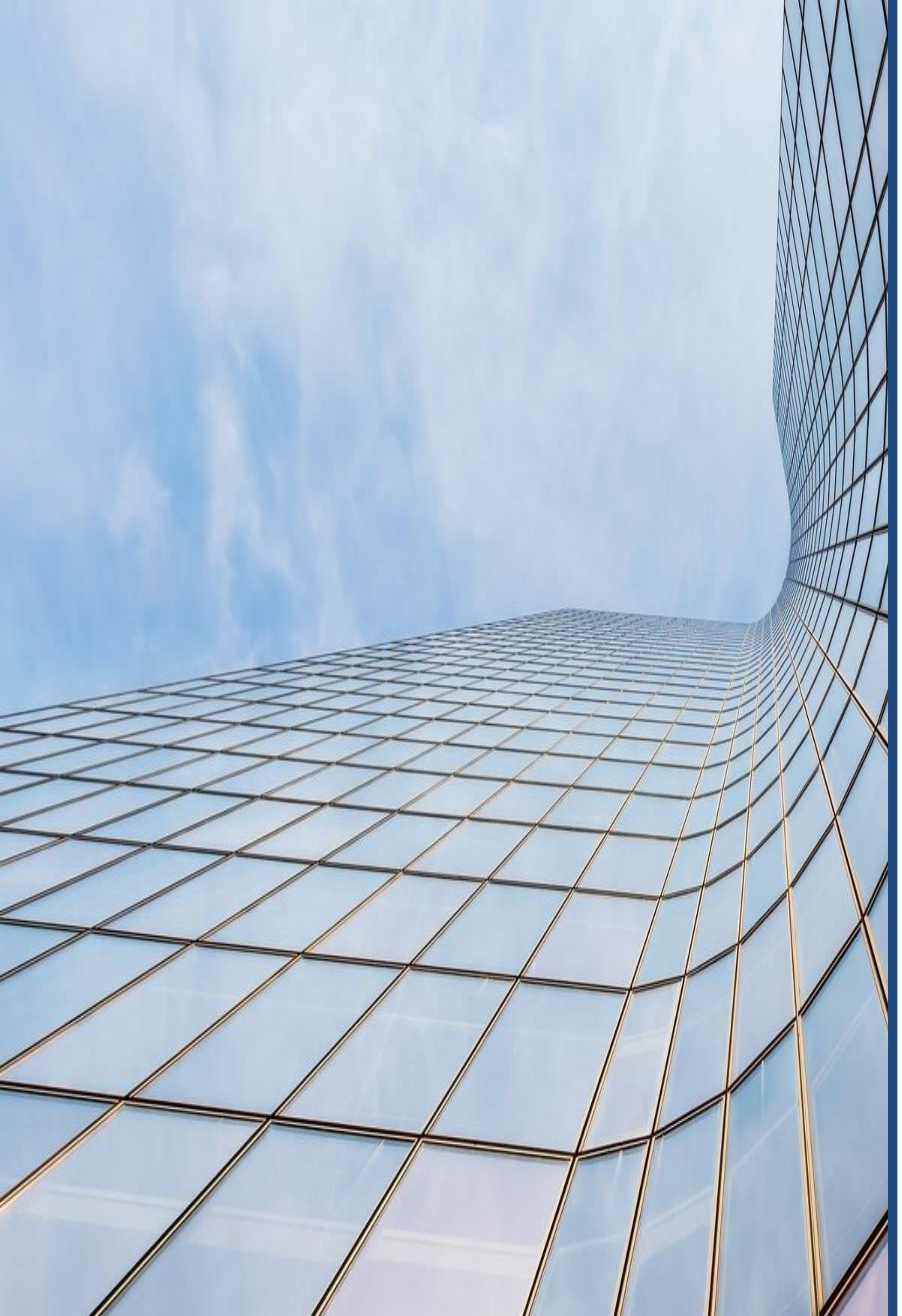
- Median (50%): 5 medals. This suggests that **more than half of the countries win fewer than 5 medals.**
- The difference between the 75% (11 medals) and the 25% (2 medals) is 9 medals, showing that the middle 50% of countries have a **range of medal counts that is quite narrow** compared to the overall range.

- ### Distribution of Total Medals Won





- The histogram shows a **right-skewed distribution**, with most countries winning fewer than 20 medals. A few countries win significantly more, creating a long tail to the right.
- The box plot also indicates the **presence of outliers** (countries with exceptionally high medal counts) and a skewed distribution. Most data points are concentrated at the lower end (closer to 0 medals), with a few extreme values far above the median.



3. Machine Learning Models

Machine Learning Models

Linear Regression

```python
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score

# Select features and target variable
X = cleaned_df[['gdp', 'population', 'gdp_per_capita', 'olympics_index']]  # Features
y = cleaned_df['total']  # Target variable: total number of medals

# Split the dataset into training and testing sets (80% training, 20% testing)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Initialize and fit the linear regression model
model = LinearRegression()
model.fit(X_train, y_train)

# Make predictions
y_pred = model.predict(X_test)

# Evaluate the model's performance
mae = mean_absolute_error(y_test, y_pred)
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

print(f"Linear Regression - Mean Absolute Error (MAE): {mae:.2f}")
print(f"Linear Regression - Mean Squared Error (MSE): {mse:.2f}")
print(f"Linear Regression - R-squared: {r2:.2f}")
```

```
Linear Regression - Mean Absolute Error (MAE): 5.86
Linear Regression - Mean Squared Error (MSE): 81.15
Linear Regression - R-squared: 0.30
```

Model Performance Evaluation

- **MAE is 5.86** indicates that the predictions of the total number of medals are off by about 5.86 medals. Considering that the mean number of medals is around 11.81, this error is nearly 50% of the mean, which could be considered significant given the data distribution.
- **MSE is 81.15** suggests that the squared differences between predicted and actual values are relatively large. Given the skewed distribution of the medals, this could indicate that the model struggles to predict the total medals accurately for countries with very high or very low medal counts.
- **R-squared is 0.30** indicates that the model explains only 30% of the variance in the total number of medals. There might be non-linear relationships or other variables that are not included in the model, which could better explain the variance

Machine Learning Models

Decision Tree

```python
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeRegressor
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score

# Select features and target variable
X = cleaned_df[['gdp', 'population', 'gdp_per_capita', 'olympics_index']]  # Features
y = cleaned_df['total']  # Target variable: total number of medals

# Split the dataset into training and testing sets (80% training, 20% testing)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Initialize and fit the Decision Tree Regressor
tree_model = DecisionTreeRegressor(random_state=42)
tree_model.fit(X_train, y_train)

# Make predictions
y_pred_tree = tree_model.predict(X_test)

# Evaluate the model's performance
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"Decision Tree Regressor - Mean Absolute Error (MAE): {mae_tree:.2f}")
print(f"Decision Tree Regressor - Mean Squared Error (MSE): {mse_tree:.2f}")
print(f"Decision Tree Regressor - R-squared: {r2_tree:.2f}")
```

```
Decision Tree Regressor - Mean Absolute Error (MAE): 8.74
Decision Tree Regressor - Mean Squared Error (MSE): 224.11
Decision Tree Regressor - R-squared: -0.94
```

Model Performance Evaluation

- **MAE is 8.74** higher than the MAE of 5.86 from the Linear Regression model, suggesting that the Decision Tree model is less accurate in predicting the total number of medals on average.
- **MSE is 224.11** is significantly higher than the MSE of 81.15 from the Linear Regression model indicates that there are some large errors that the model fails to handle well.
- **R-squared is -0.94** indicates a very poor fit, a negative R-squared suggests that the model is performing worse than a simple horizontal line (mean of the target values) and is likely overfitting the training data while failing to generalize to the test data.

Machine Learning Models

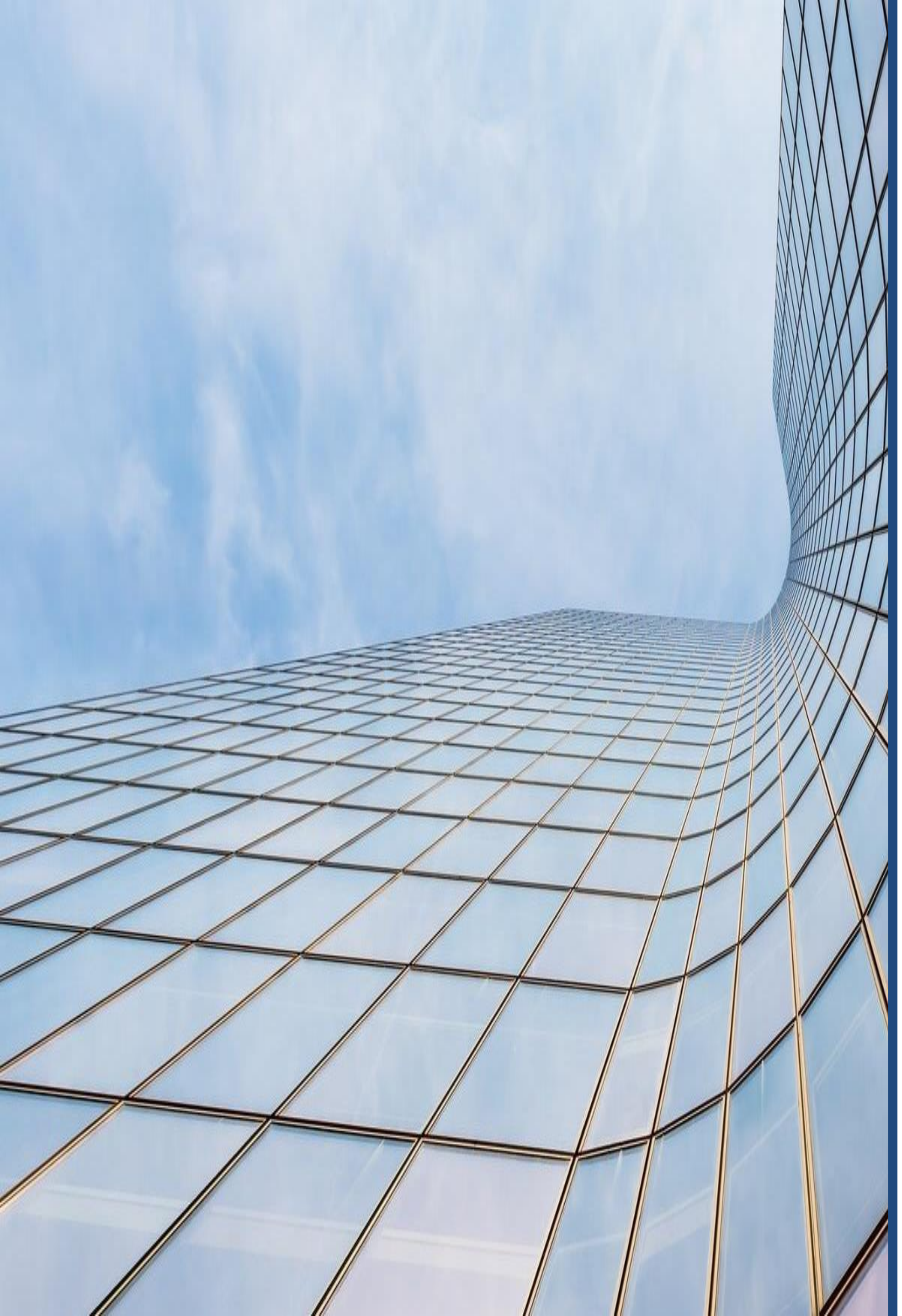
Random Forest

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

# Select features and target variable
X = cleaned_df[['gdp', 'population', 'gdp_per_capita', 'olympics_index']]  # Features
y = cleaned_df['total']  # Target variable: total number of medals

# Split the dataset into training and testing sets (80% training, 20% testing)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Initialize and fit the Random Forest Regressor
rf_model = RandomForestRegressor(n_estimators=100, random_state=42)
rf_model.fit(X_train, y_train)

# Make predictions
y_pred_rf = rf_model.predict(X_test)

# Evaluate the model's performance
mae_rf = mean_absolute_error(y_test, y_pred_rf)
mse_rf = mean_squared_error(y_test, y_pred_rf)
r2_rf = r2_score(y_test, y_pred_rf)

print(f"Random Forest Regressor - Mean Absolute Error (MAE): {mae_rf:.2f}")
print(f"Random Forest Regressor - Mean Squared Error (MSE): {mse_rf:.2f}")
print(f"Random Forest Regressor - R-squared: {r2_rf:.2f}")
```

```
Random Forest Regressor - Mean Absolute Error (MAE): 7.44
Random Forest Regressor - Mean Squared Error (MSE): 130.80
Random Forest Regressor - R-squared: -0.13
```

Model Performance Evaluation

- **MAE is 7.44** slightly better than the Decision Tree Regressor's MAE of 8.74 but still higher than the Linear Regression model's MAE of 5.86, it is still not the most accurate among the models tested.
- **MSE is 130.80** lower than the Decision Tree Regressor's MSE of 224.11 but higher than the Linear Regression model's MSE of 81.15, it is still not performing as well as the Linear Regression model in terms of overall squared error.
- **R-squared is -0.13** suggests that the model is performing worse than a horizontal line (mean of the target values). The negative R-squared suggests that the Random Forest model, like the Decision Tree, fails to generalize well to the test data



4. Deep Learning Models

Deep Learning Models

Neural Network

```python
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Input, Dense
from tensorflow.keras.optimizers import Adam

# Select features and target variable
X = cleaned_df[['gdp_per_capita', 'olympics_index', 'sports_index']].values  # Features
y = cleaned_df['total'].values  # Target variable: total number of medals

# Convert data to appropriate type for TensorFlow
X = X.astype(np.float32)
y = y.astype(np.float32)

# Split the dataset into training and testing sets (80% training, 20% testing)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Function to build and compile a neural network model
def build_model(input_shape, num_layers=2, num_neurons=32, activation='relu', learning_rate=0.001):
    model = Sequential()
    model.add(Input(shape=input_shape))

    # Add hidden layers
    for _ in range(num_layers):
        model.add(Dense(num_neurons, activation=activation))

    # Output layer
    model.add(Dense(1))

    # Compile the model
    optimizer = Adam(learning_rate=learning_rate)
    model.compile(optimizer=optimizer, loss='mean_squared_error')

    return model
```

Process Exploration

1) Importing Libraries and Prepare Data

- TensorFlow and Keras Imports: Sequential, Input, Dense, and Adam
- Feature and Target Selection: The features ['gdp_per_capita', 'olympics_index', 'sports_index'] are selected as inputs (X), and total (the number of medals) is the target variable (y).
- Data Type Conversion: The features and target variables are converted to float32 for compatibility with TensorFlow.
- Data Splitting: The dataset is split into training and testing sets (80% training, 20% testing) using train_test_split from Scikit-Learn.

2) Defining a Function to Build the Neural Network Model

- build_model Function: The function creates and compiles a neural network model with customizable parameters: input_shape, num_layers, num_neurons, activation (Activation function for the hidden layers (default is ReLU)), learning_rate: Learning rate for the optimizer (Adam).
- Hidden Layers: A loop adds the specified number of hidden layers (num_layers), each with the specified number of neurons (num_neurons) and activation function.
- The model is compiled using the Adam optimizer with the specified learning rate and mean squared error (MSE) as the loss function

Deep Learning Models

Neural Network

```python
# Experiment with different architectures and hyperparameters
results = []
layer_options = [1, 2, 3]
neuron_options = [32, 64, 128]
learning_rate_options = [0.01, 0.001, 0.0001]
epoch_options = [40, 100]
batch_size_options = [8, 16]

for num_layers in layer_options:
    for num_neurons in neuron_options:
        for learning_rate in learning_rate_options:
            for epochs in epoch_options:
                for batch_size in batch_size_options:
                    print(f"Training model with {num_layers} layers, {num_neurons} neurons, learning

                    # Build the model
                    model = build_model(input_shape=(X_train.shape[1],), num_layers=num_layers, num_n
                                        learning_rate=learning_rate)

                    # Train the model
                    history = model.fit(X_train, y_train, epochs=epochs, batch_size=batch_size, valid

                    # Make predictions
                    y_pred_nn = model.predict(X_test).flatten()

                    # Evaluate the model's performance
                    mae_nn = mean_absolute_error(y_test, y_pred_nn)
                    mse_nn = mean_squared_error(y_test, y_pred_nn)
                    r2_nn = r2_score(y_test, y_pred_nn)

                    # Save the results
                    results.append({
                        'num_layers': num_layers,
                        'num_neurons': num_neurons,
                        'learning_rate': learning_rate,
                        'epochs': epochs,
                        'batch_size': batch_size,
                        'MAE': mae_nn,
                        'MSE': mse_nn,
                        'R-squared': r2_nn
                    })

# Convert results to a DataFrame for better visualization
results_df = pd.DataFrame(results)
print(results_df.sort_values(by='MSE', ascending=True).head())  # Display top 5 configurations with l
```

Process Exploration

3) Experimenting with Different Architectures and Hyperparameters

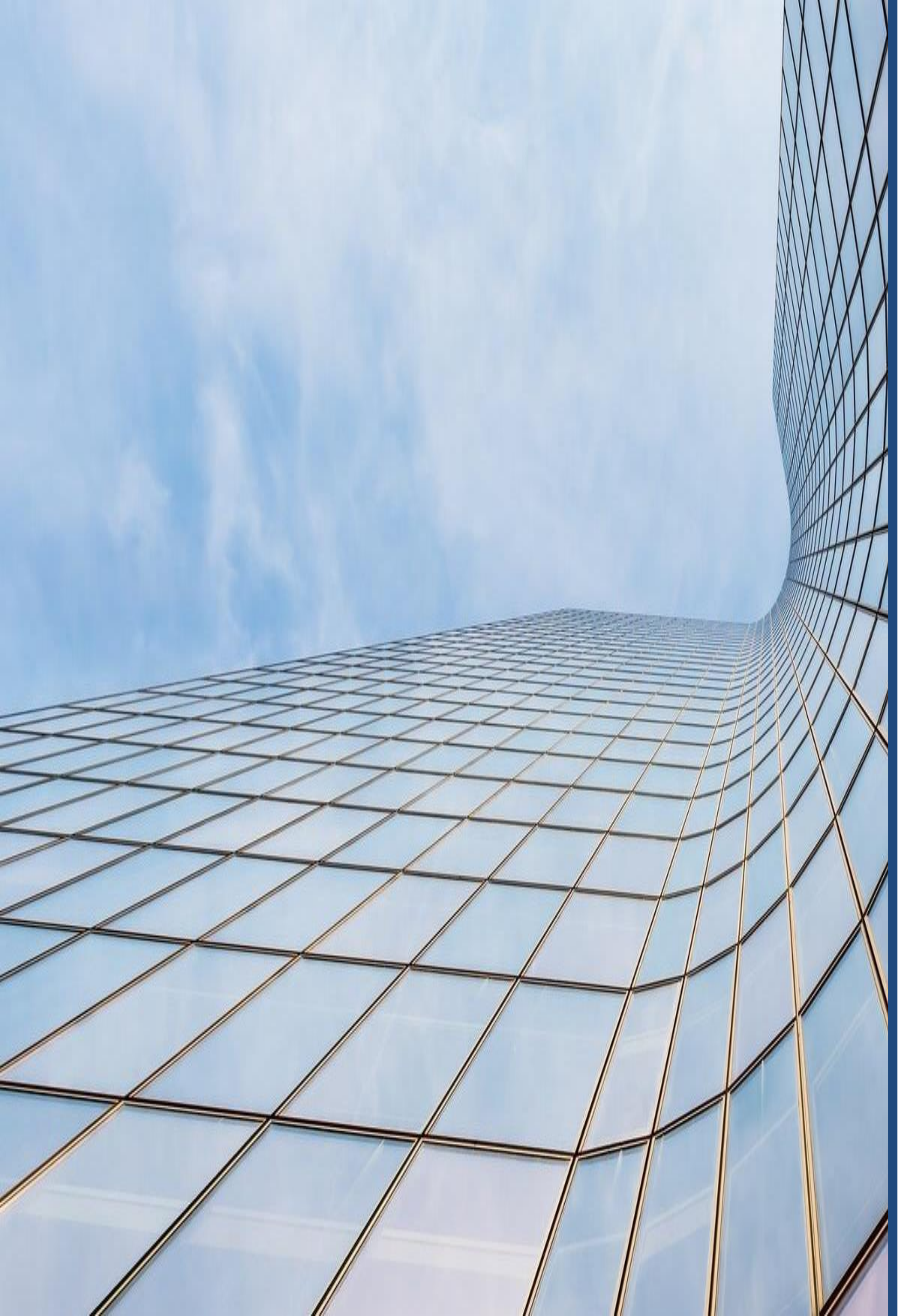
- Hyperparameter Options: layer_options (1, 2, or 3), neuron_options (32, 64, 128), learning_rate_options (0.01, 0.001, 0.0001), epoch_options: (40, 100), batch_size_options (8, 16).
- Experiment Loop: Nested loops iterate over all combinations of these hyperparameters to train and evaluate different neural network models.

  - Model Training: For each combination, the model is built using build_model, trained using the .fit() method, and evaluated using MAE, MSE, and R-squared metrics.

  - Performance Evaluation: Predictions are made on the test set, and performance metrics are calculated and stored in the results list.

4) Saving and Displaing the Results

- Results Storage: The performance metrics and corresponding hyperparameters for each model configuration are stored in a list of dictionaries (results).
- DataFrame Conversion: The list of results is converted into a pandas DataFrame (results_df) for easy visualization and sorting.
- Sorting and Displaying: The DataFrame is sorted by MSE in ascending order to display the top 5 configurations with the lowest MSE, indicating the best-performing models.



5. Model Evaluation

Model Evaluation

Model	Mean Absolute Error (MAE)	Mean Squared Error (MSE)	R-squared (R^2)
Linear Regression	5.86	81.15	0.30
Decision Tree	8.74	224.11	-0.94
Random Forest	7.44	130.80	-0.13
Neural Network (Best Configuration)	5.38	62.86	0.46

Performance Metrics

- **Linear Regression**: MAE: 5.86, MSE: 81.15, R^2: 0.30

  Performance Summary: Linear Regression still has the lowest MAE and MSE among all models. Its positive R-squared value indicates that it explains some variance in the target variable reasonably well. This model is useful when simplicity and interpretability are crucial.

- **Decision Tree**: MAE: 8.74, MSE: 224.11, R^2: -0.94

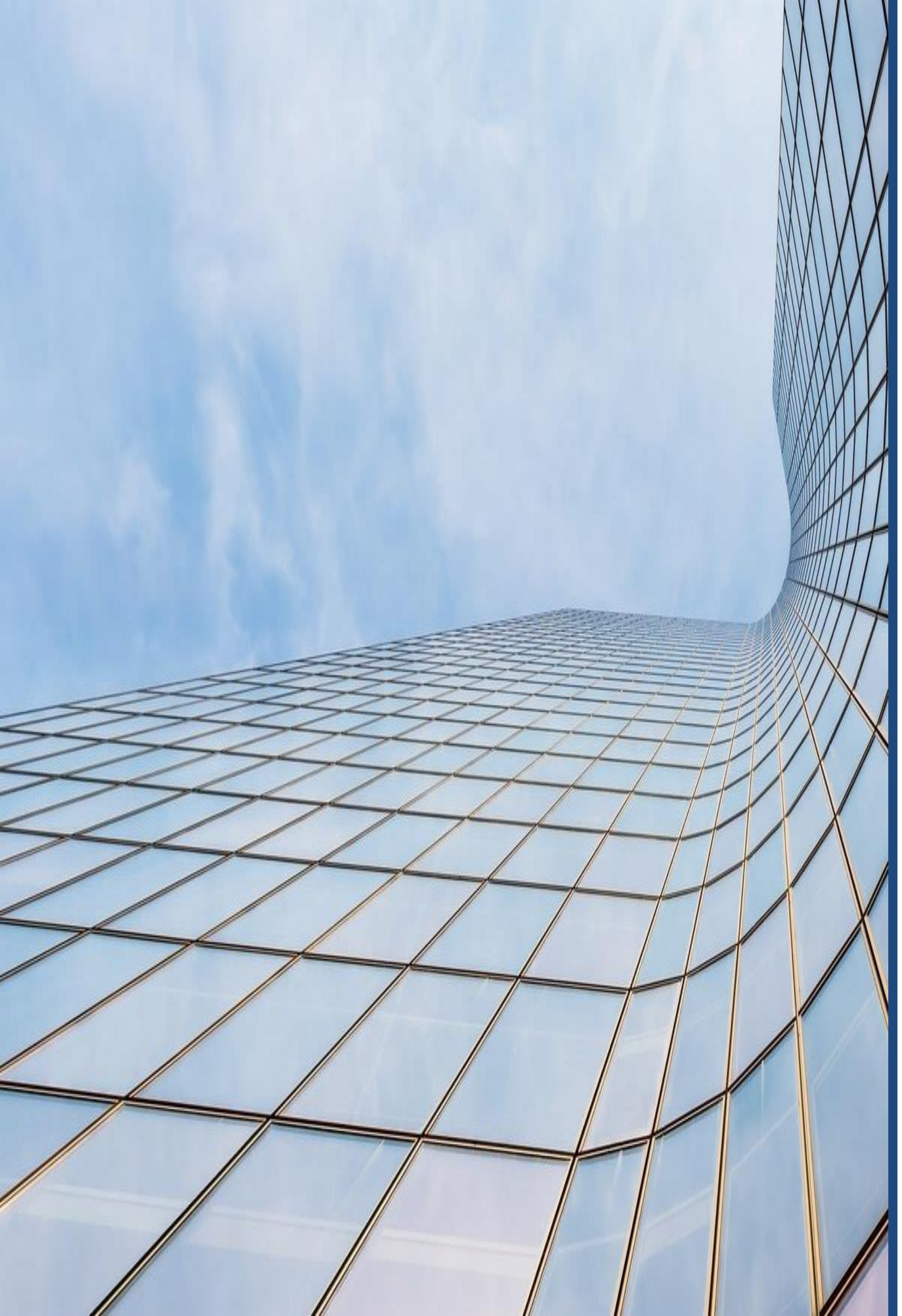
  Performance Summary: The Decision Tree model continues to show the worst performance, with the highest MAE and MSE and a negative R-squared value, indicating severe overfitting and poor generalization to the test data.

- **Random Forest**: MAE: 7.44, MSE: 130.80, R^2: -0.13

  Performance Summary: Random Forest performs better than the Decision Tree but worse than the Linear Regression model. Its negative R-squared value still suggests overfitting or insufficient generalization. It is not the best model but provides some improvement over the Decision Tree

- **Neural Network** (Revised Best Configuration): MAE: 5.38, MSE: 62.86, R^2: 0.46

  Performance Summary: The Neural Network model outperforms all other models, including Linear Regression, with the lowest MAE (5.38) and MSE (62.86). The R-squared value of 0.46 indicates that the model explains 46% of the variance in the target variable, this suggests that the Neural Network is effectively capturing complex, non-linear relationships between the features and the target variable.



6. Interpretation and Insights

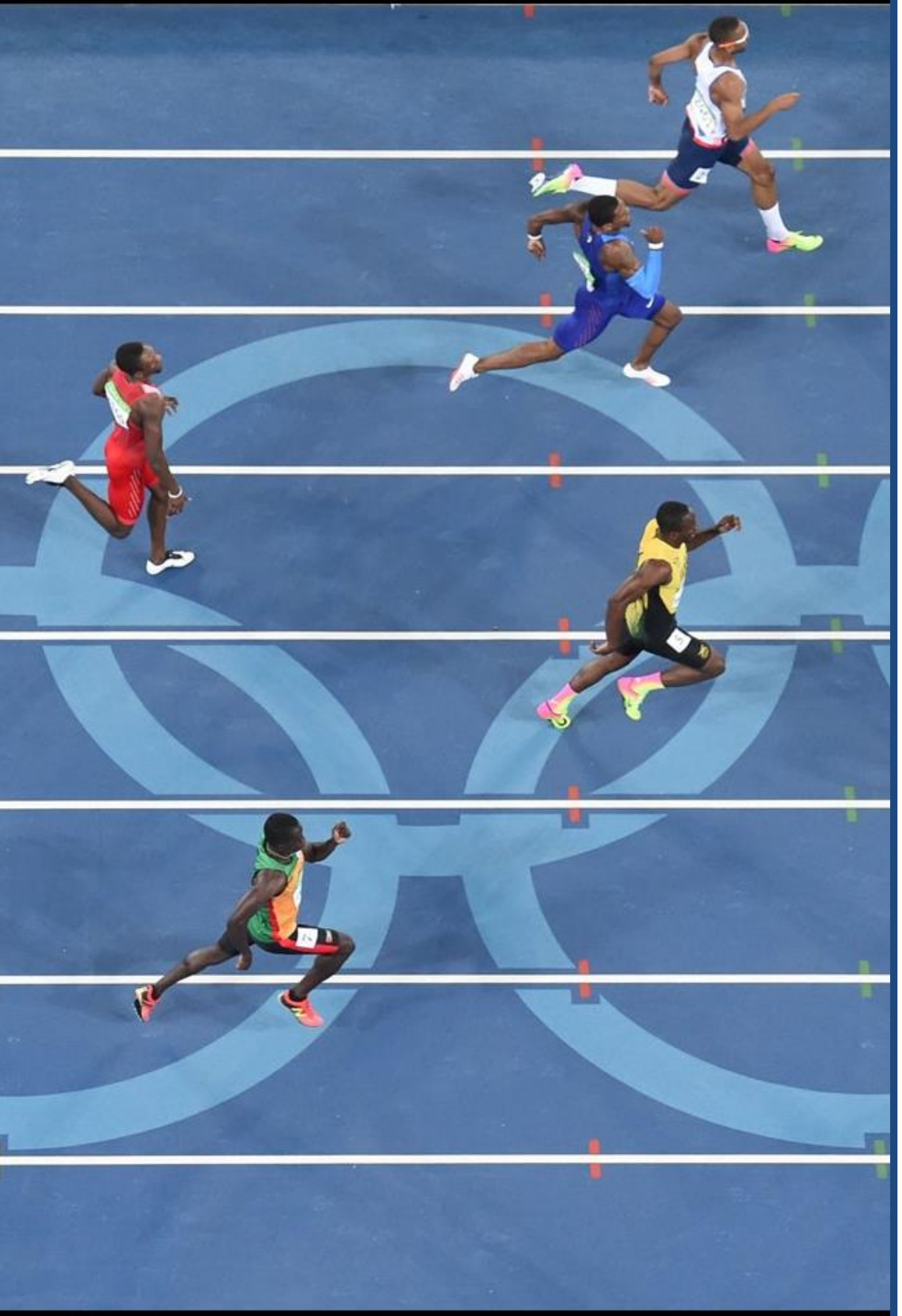
Interpretation and Insights

Feature Importance

Since Random Forests provide a straightforward method for calculating feature importances, we can use it to understand which features are most influential.

```
# After fitting the RandomForest model:
importances = rf_model.feature_importances_
feature_names = ['gdp', 'population', 'gdp_per_capita', 'olympics_index']
importance_df = pd.DataFrame({'Feature': feature_names, 'Importance': importances})
importance_df = importance_df.sort_values(by='Importance', ascending=False)

print(importance_df)

```

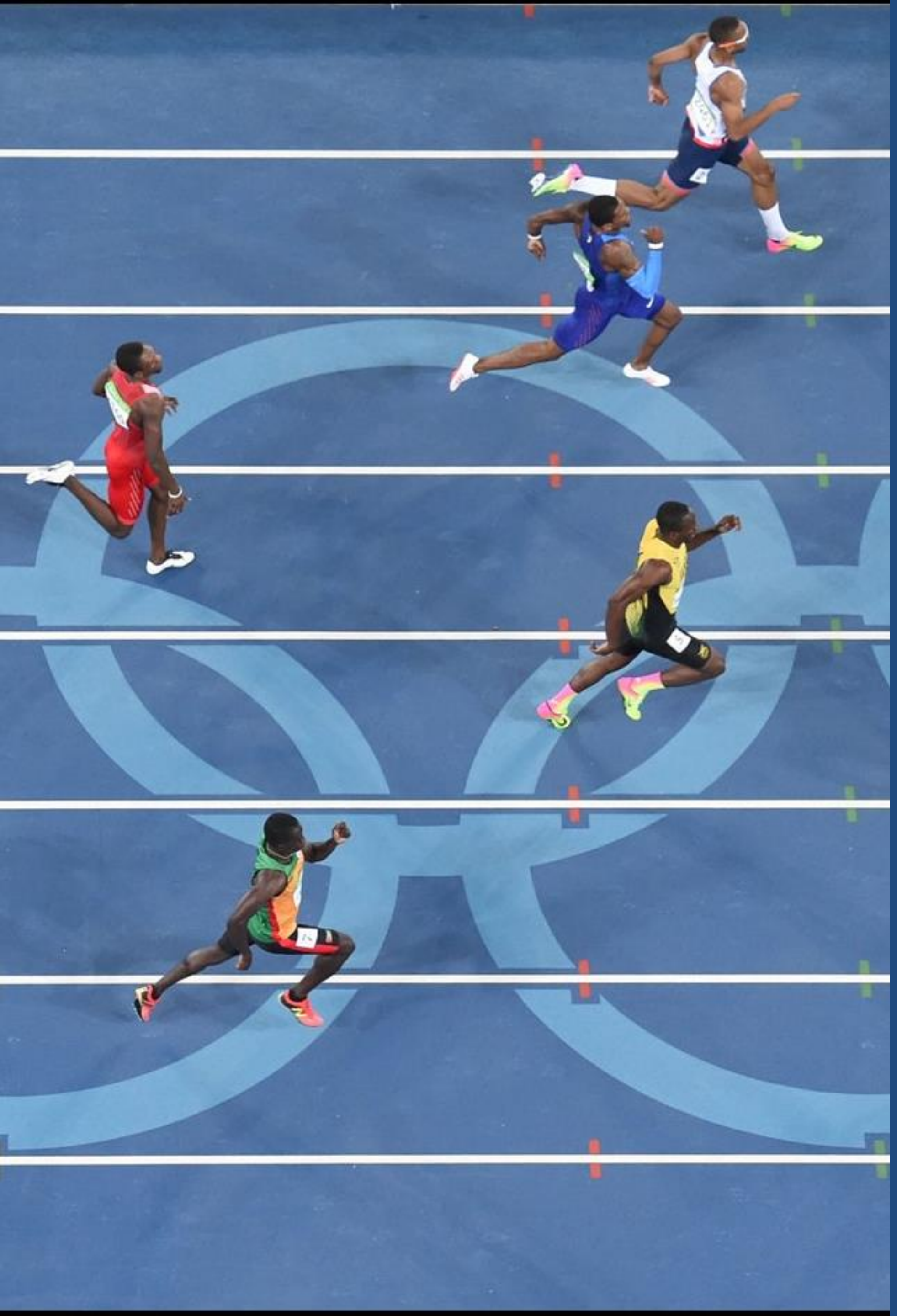
```
          Feature  Importance
0             gdp    0.798998
3  olympics_index    0.094474
1      population    0.065810
2  gdp_per_capita    0.040718
```



Feature Importance insights

- **Feature Importance and Interpretation:**
  Features such as gdp_per_capita, olympics_index are likely influential in predicting Olympic success, with the Neural Network model likely leveraging interactions between these features to provide more accurate predictions.

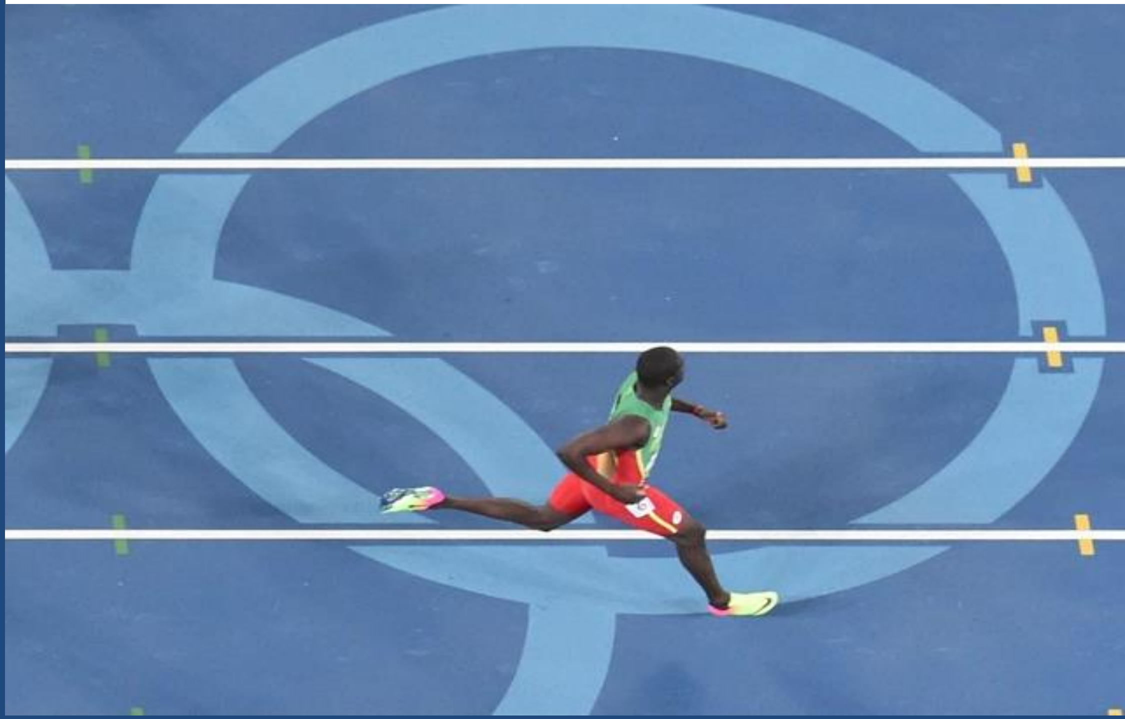
- **Practical Implications:**
  For countries aiming to improve their Olympic performance, focusing on improving GDP per capita, investing in sports infrastructure, and leveraging historical success can be strategic priorities. The insights from the Neural Network model suggest that a combination of economic and sports-specific factors is critical to Olympic success.



Model Interpretation insights

- **Best Performing Model**: The Neural Network is the best-performing model with lowest MSE and the highest R^2 value, providing more accurate predictions.
- **Linear Regression vs. Neural Network**: While the Linear Regression model performs reasonably well, the Neural Network outperforms it by capturing non-linear patterns, The Neural Network's ability to model complex relationships makes it more suitable for this dataset.
- **Decision Trees and Random Forest**: Both tree-based models performed poorly, especially the Decision Tree. Further tuning might be necessary to improve their performance, but they are unlikely to outperform the Neural Network.
- **Further Improvements for Neural Networks**: The Neural Network could potentially be further optimized by experimenting with more sophisticated architectures, such as deeper networks, different activation functions.

THANK YOU



CONTACT

Rina Irene Rafalski

@ rinaraf@gmail.com