Oeson Project 3:
Machine Learning/Deep Learning

—

By: Christopher Gonzalez

Introduction

- **Machine learning is a subset of AI, which uses algorithms that learn from data (input) to make inferences.**
- **On a similar note, deep learning is a subset of machine learning which uses layers of neural networks to learn.**
- **When combined, machine and deep learning are capable of learning independently with minimal human intervention.**
- **This presentation will demonstrate the use of machine and deep learning in predicting olympic medal counts.**

Background of the Dataset

- **The Olympic Games is an international multi-sports event to determine the best athletes in the world.**
- **Information in the dataset includes country names, gdp, olympics index, sports index, medals won, etc.**
- **For this project, machine learning and deep learning will be used to understand which factors are most influential in predicting Olympic success.**

A Portion of the Dataset

OlympicMedals

iso	ioc	name	continent	population	gdp	olympics_index	sports_index	olympicsIndex	sportsIndex	total	gold	silver	bronze
ARG	ARG	Argentina	South America	45376763	383066977654	19.5971422135	9.3245368874	19.5971422135	9.3245368874	3	0	1	2
ARM	ARM	Armenia	Asia	2963234	12645459214	19.6814569179	13.4973244517	19.6814569179	13.4973244517	4	0	2	2
AUS	AUS	Australia	Oceania	25687041	1330900925057	31.1700989204	11.0738450323	31.1700989204	11.0738450323	46	17	7	22
AUT	AUT	Austria	Europe	8917205	428965397959	12.2121386941	15.9230332938	12.2121386941	15.9230332938	7	1	1	5
AZE	AZE	Azerbaijan	Europe	10110116	42607176471	18.2138376188	13.1033442207	18.2138376188	13.1033442207	7	0	3	4
BEL	BEL	Belgium	Europe	11555997	515332499628	15.0063004202	14.3137622475	15.0063004202	14.3137622475	7	3	1	3
BFA	BUR	Burkina Faso	Africa	20903278	17369059295	14.9097873817	12.9365811886	14.9097873817	12.9365811886	1	0	0	1
BGR	BUL	Bulgaria	Europe	6927288	69105101090	22.6840327909	19.6560277704	22.6840327909	19.6560277704	6	3	1	2
BHR	BRN	Bahrain	Asia	1701583	38474521277	14.4225607577	13.9315039882	14.4225607577	13.9315039882	1	0	1	0
BHS	BAH	Bahamas	North America	393248	11250000000	18.0891842171	9.6301739642	18.0891842171	9.6301739642	2	2	0	0
BLR	BLR	Belarus	Europe	9398861	60258239056	30.9792870137	11.6659341893	30.9792870137	11.6659341893	7	1	3	3
BMU	BER	Bermuda	North America	63903	7484113000	51.5360263804	23.0599913923	51.5360263804	23.0599913923	1	1	0	0
BRA	BRA	Brazil	South America	212559409	1444733258972	34.4918726569	22.3539370623	34.4918726569	22.3539370623	21	7	6	8
BWA	BOT	Botswana	Africa	2351625	15781732826	17.4364279827	15.4157605082	17.4364279827	15.4157605082	1	0	0	1
CAN	CAN	Canada	North America	38005238	1643407977069	24.4017497237	9.1549797055	24.4017497237	9.1549797055	24	7	6	11

Data Preprocessing

- **The first step is to preprocess the data by handling missing values, checking for duplicates, and address inconsistencies.**
- **Next, features should be modified through normalization/standardization.**
- **Finally, the data will split into training and testing sets, where 80% of all data will go into training.**

Step 1: Data Preprocessing

```
#Check for Missing Values
print(data.isnull().sum())
```

```
iso              0
ioc              0
name             0
continent        0
population       0
gdp              0
olympics_index   0
sports_index     0
olympicsIndex    0
sportsIndex      0
total            0
gold             0
silver           0
bronze           0
dtype: int64
```

Note: Missing values were handled manually.

```
#Drop Irrelevant/Duplicate Columns
data = data.drop(columns = ['iso', 'ioc', 'name', 'continent', 'olympicsIndex', 'sportsIndex'])
data.head()
```

	population	gdp	olympics_index	sports_index	total	gold	silver	bronze
0	45376763	383066977654	19.597142	9.324537	3	0	1	2
1	2963234	12645459214	19.681457	13.497324	4	0	2	2
2	25687041	1330900925057	31.170099	11.073845	46	17	7	22
3	8917205	428965397959	12.212139	15.923033	7	1	1	5
4	10110116	42607176471	18.213838	13.103344	7	0	3	4

```
#Extract features and target
x = data.drop(columns = ['total'])
y = data['total']

#Standardizing Data
scaler = StandardScaler()
x = scaler.fit_transform(x)
```

```
#Splitting the Dataset Into Training and Test sets
from sklearn.model_selection import train_test_split


x_train, x_test, y_train, y_test = train_test_split(x, y, test_size = 0.2, random_state = 0)
```

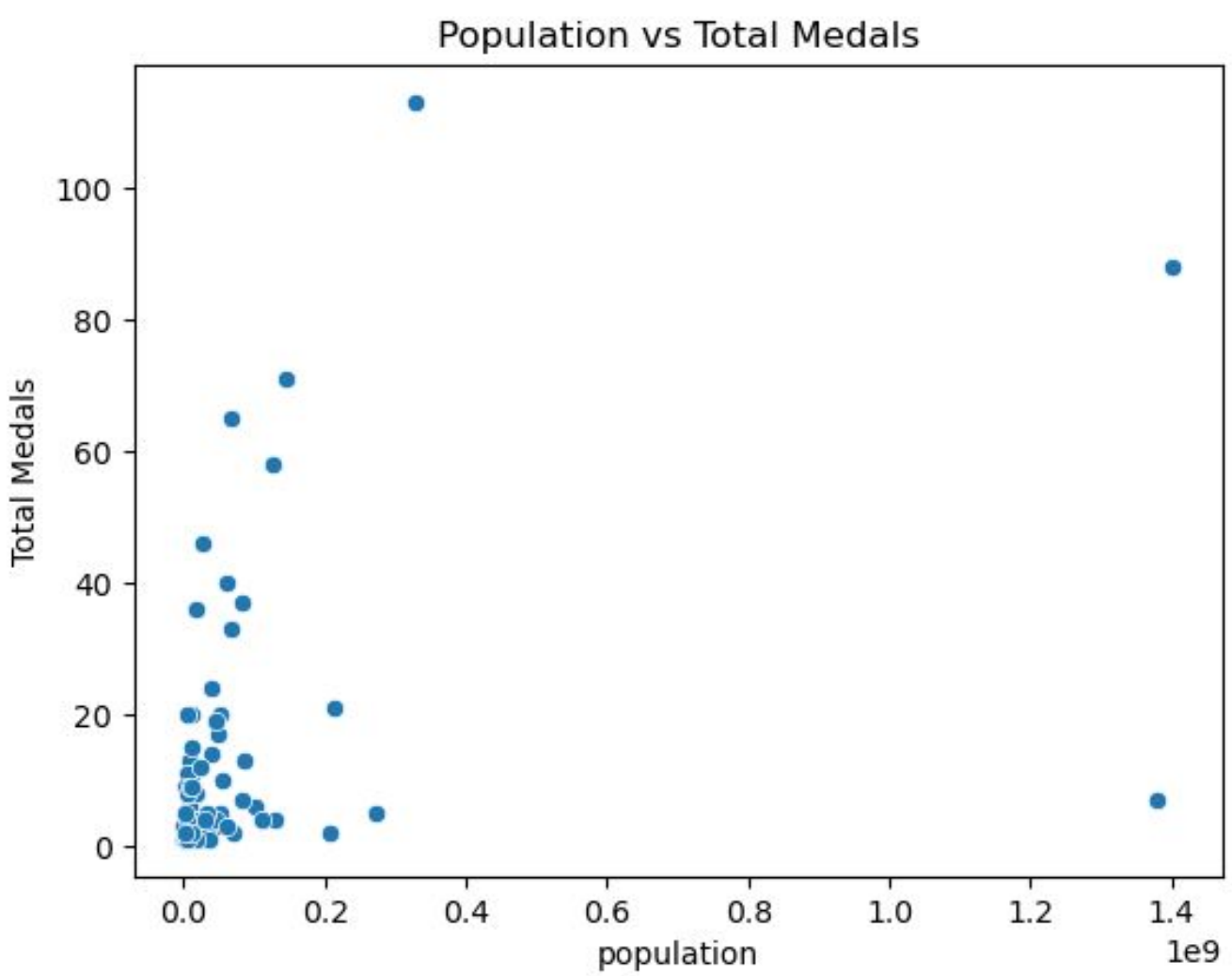
Exploratory Data Analysis (EDA)

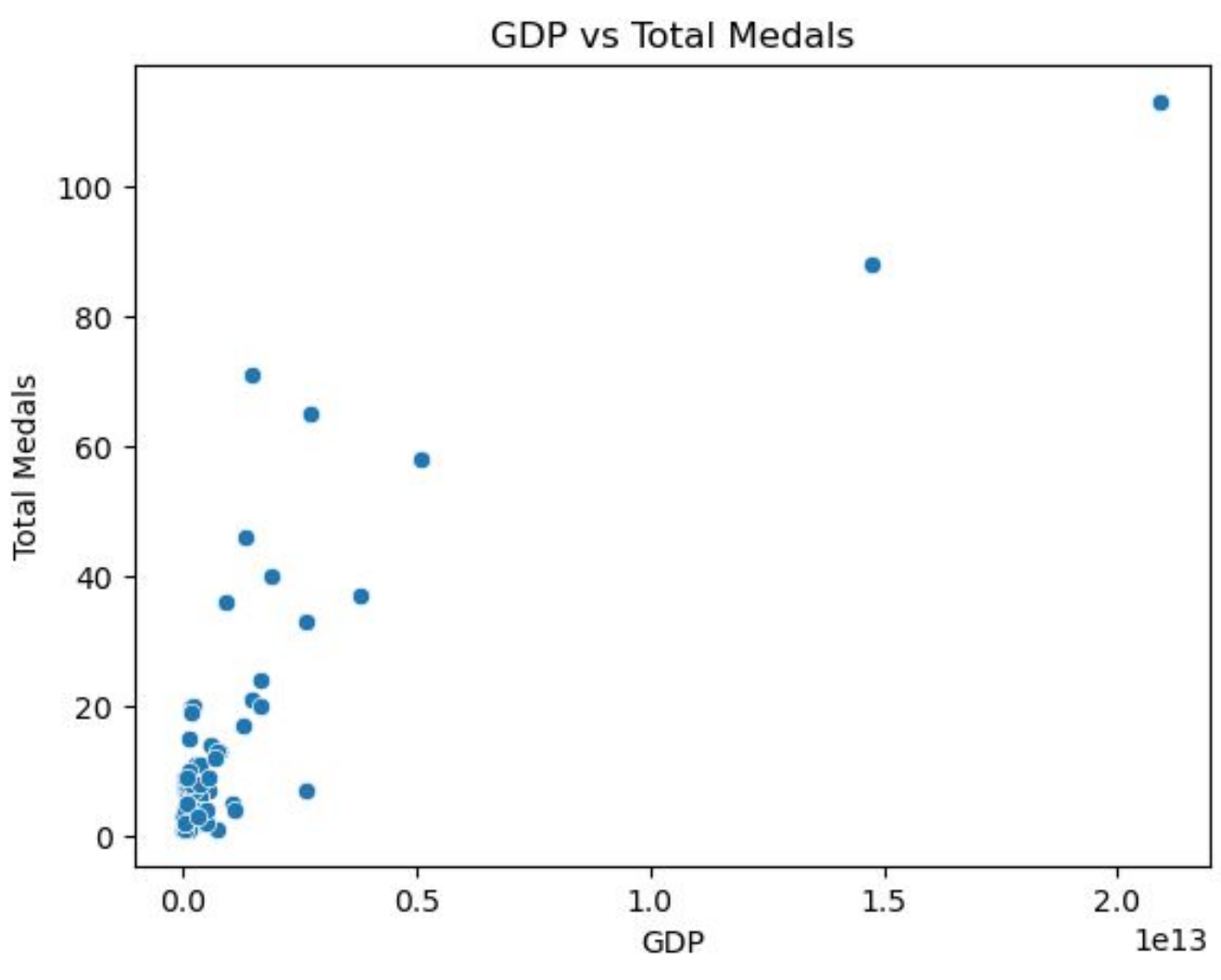
- **Exploratory Data Analysis is the analysis of a given dataset for significant patterns visible in the data.**
- **It is ideal to begin by computing basic statistics of the data such as mean, median, and standard deviation.**
- **Then, the data should be visualized to better understand the relationship between features and medal counts.**
- **For this project, scatter plots and a correlation matrix was created as visualizations.**

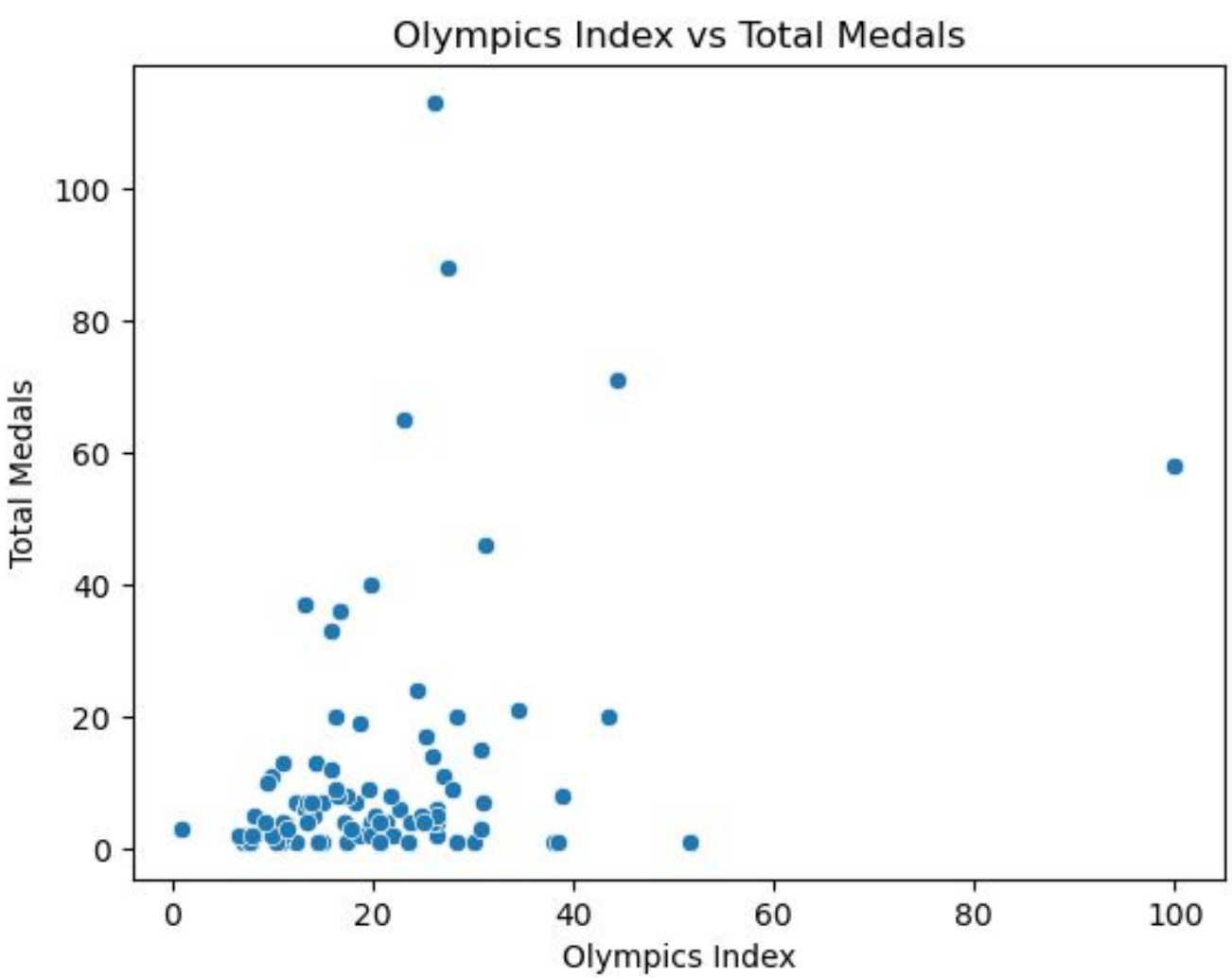
Step 2: Explatory Data Analysis (EDA)

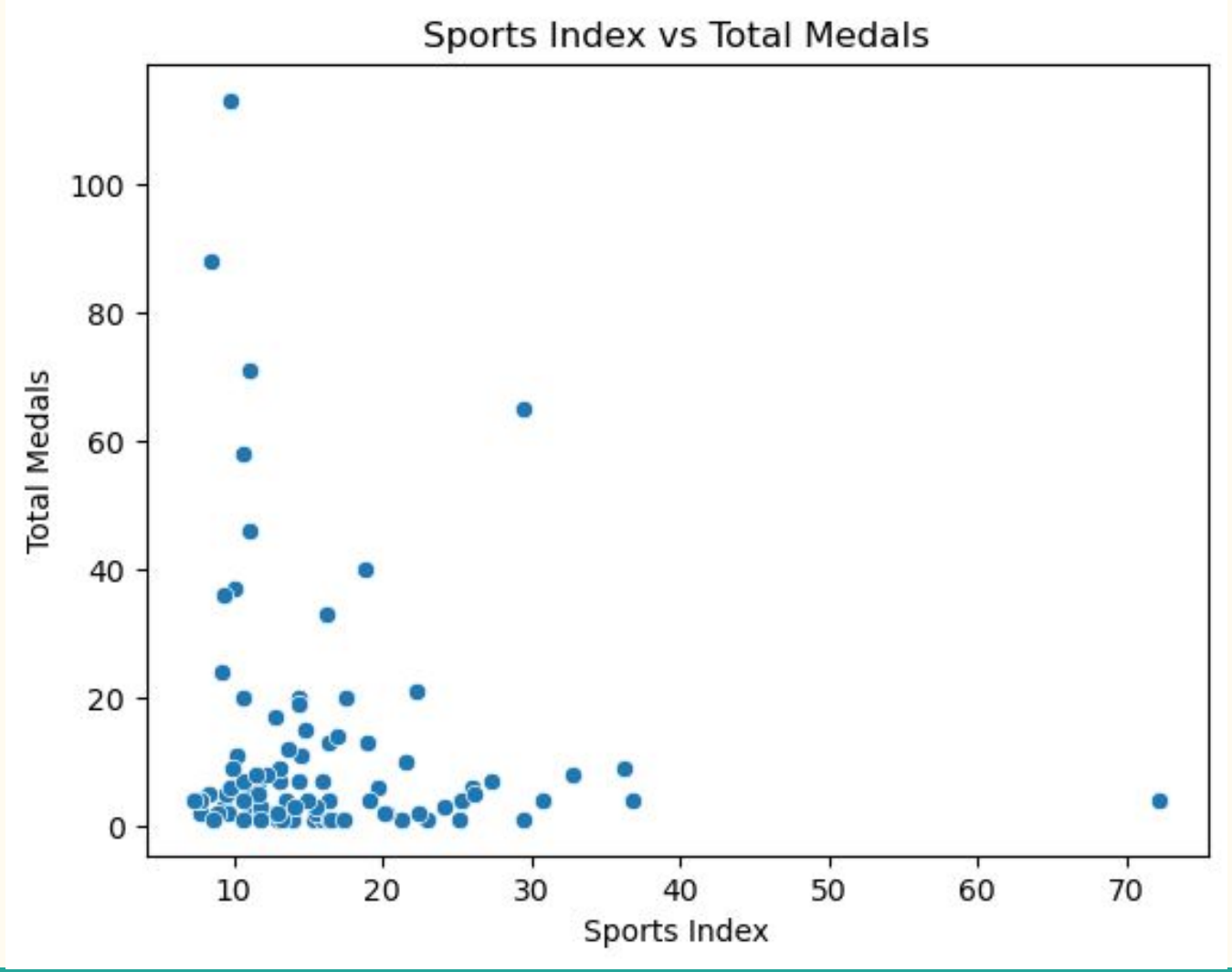
```
#General Statistics of Data
data.describe()
```

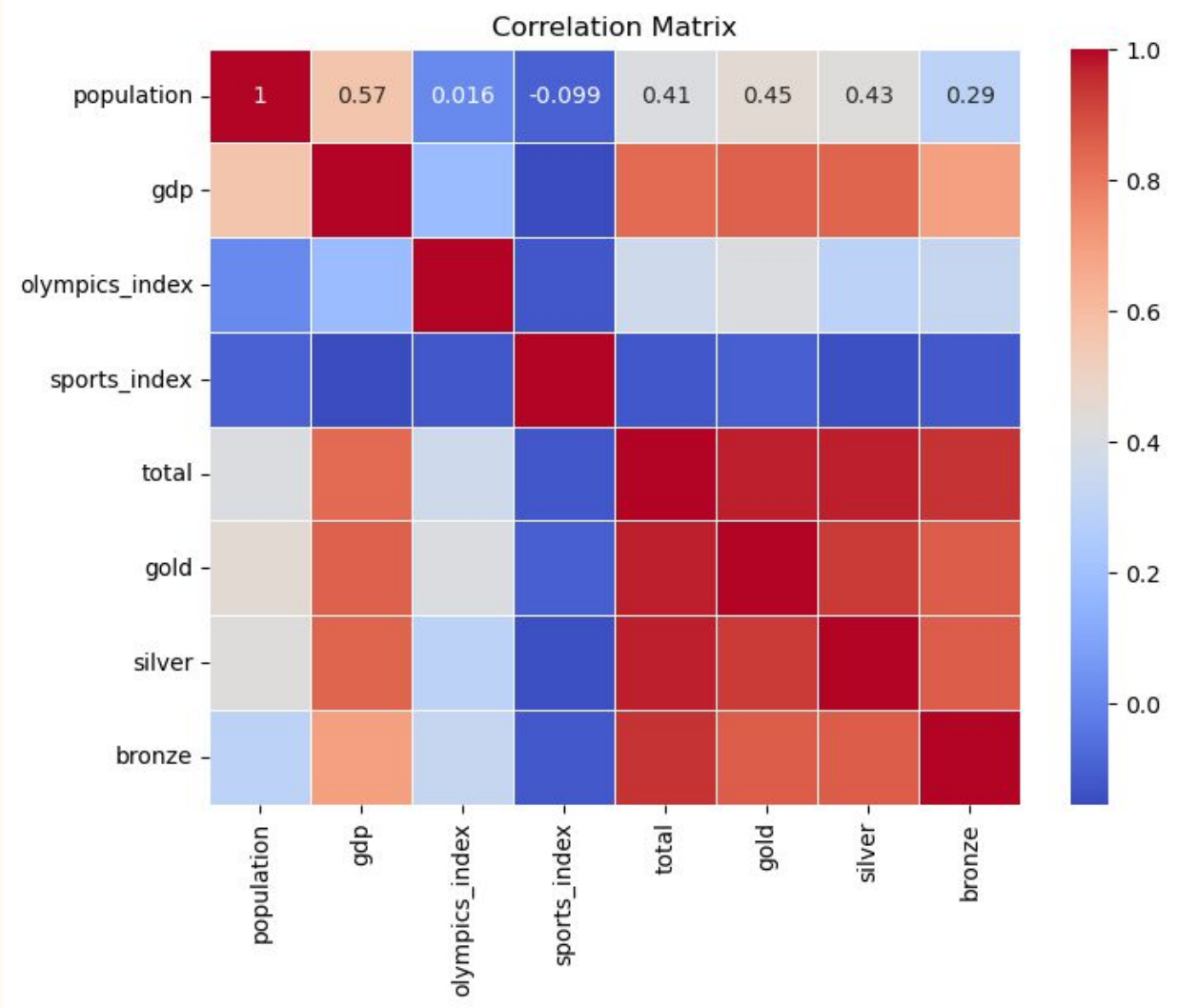
	population	gdp	olympics_index	sports_index	total	gold	silver	bronze
count	9.300000e+01	9.300000e+01	93.000000	93.000000	93.000000	93.000000	93.000000	93.000000
mean	6.639237e+07	8.668410e+11	20.677422	16.329262	11.612903	3.655914	3.634409	4.322581
std	2.057474e+08	2.702387e+12	12.493268	8.835266	19.091332	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	13.091179	11.019952	2.000000	0.000000	0.000000	1.000000
50%	1.132662e+07	1.698354e+11	18.787691	13.993115	4.000000	1.000000	1.000000	2.000000
75%	4.735157e+07	5.153325e+11	26.037386	18.984764	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


Population vs Total Medals
Total Medals
100
80
60
40
20
0
0.0
0.2
0.4
0.6
0.8
1.0
1.2
1.4
population
1e9


GDP vs Total Medals
Total Medals
100
80
60
40
20
0
0.0
0.5
1.0
1.5
2.0
GDP
1e13


Olympics Index vs Total Medals
Total Medals
Olympics Index
0
20
40
60
80
100
0
20
40
60
80
100


Sports Index vs Total Medals
Total Medals
Sports Index
0
20
40
60
80
100
10
20
30
40
50
60
70


Correlation Matrix
population
gdp
olympics_index
sports_index
total
gold
silver
bronze
1
0.57
0.016
-0.099
0.41
0.45
0.43
0.29
1.0
0.8
0.6
0.4
0.2
0.0

Significant Findings

- **The range of medals won per country is from 1-113, with the top 3 countries being United States, China, and Russia.**
- **The standard deviation of total medals won per country is approximately 19.**
- **According to the scatterplots, it can be argued that a higher gdp have a correlation with medals won.**
- **Most countries have a olympic and sports index of approximately 15.**

Significant Findings (Continued)

- **When viewing the correlation matrix correctly, it can be seen that gdp has the highest correlation with medals won.**
- **On the contrast, sports index has the lowest correlation with medals won per country.**
- **These findings are consistent across the different medal types as well (gold, silver, bronze).**
- **This solidifies that gdp is highly influential in predicting total medals won.**

Machine Learning Models

- **As mentioned previously, machine learning uses algorithms that learn from data and make predictions.**
- **For this project, three machine learning models were used (Linear Regression, Decision Tree, Random Forest).**
- **The purpose of using three models is to compare the results and later determine which is best for the data.**
- **The performance of each model is done using Mean Absolute Error, Mean Squared Error, and R-Squared.**

[33]:
```
#Linear Regression
lr_model = LinearRegression()
lr_model.fit(x_train, y_train)

#Make Predictions on the Test Set
lr_predict = lr_model.predict(x_test)

#Evaluate
print('Linear Regression Model: ')
print(f'Mean Absolute Error : {mean_absolute_error(y_test, lr_predict)}')
print(f'Mean Squared Error : {mean_squared_error(y_test, lr_predict)}')
print(f'R-Squared : {r2_score(y_test, lr_predict)}')
```

```
Linear Regression Model:
Mean Absolute Error : 1.5426256763212702e-15
Mean Squared Error : 1.22481035284315e-29
R-Squared : 1.0
```

[35]:
```
#Decision Tree
dt_model = DecisionTreeRegressor()
dt_model.fit(x_train, y_train)

#Make Predictions
dt_predict = dt_model.predict(x_test)

#Evaluate
print('Decision Tree Model: ')
print(f'Mean Absolute Error : {mean_absolute_error(y_test, dt_predict)}')
print(f'Mean Squared Error : {mean_squared_error(y_test, dt_predict)}')
print(f'R-Squared : {r2_score(y_test, dt_predict)}')
```

```
Decision Tree Model:
Mean Absolute Error : 3.5789473684210527
Mean Squared Error : 42.73684210526316
R-Squared : 0.9075082132322966
```

[37]:

```
#Random Forest
rfr_model = RandomForestRegressor()
rfr_model.fit(x_train, y_train)

#Make Predictions
rfr_predict = rfr_model.predict(x_test)

#Evaluate
print('Random Forest Regressor Model: ')
print(f'Mean Absolute Error : {mean_absolute_error(y_test, rfr_predict)}')
print(f'Mean Squared Error : {mean_squared_error(y_test, rfr_predict)}')
print(f'R-Squared : {r2_score(y_test, rfr_predict)}')
```

```
Random Forest Regressor Model:
Mean Absolute Error : 0.7884210526315791
Mean Squared Error : 1.234694736842107
R-Squared : 0.9973278530490876
```

Understanding the Results

- **The Mean Absolute Error reflects the average absolute difference between predicted and actual values.**
- **Similarly, Mean Squared Error is the average of the differences between predicted and actual values squared.**
- **Numerical values for R-Squared ranges from 0-1, with higher values indicating better performance.**
- **Looking at the results, the best model for the data is linear regression, while the worst model is decision tree.**

Deep Learning Models

- **To further the initial conclusions discovered in machine learning, we use deep learning.**
- **For this project, a Recurrent Neural Network was used along with the Keras library.**
- **Afterwards, hyperparameter tuning will be done on the model to optimize performance.**

```
#Data Preprocessing
features = data[['population', 'gdp', 'olympics_index', 'sports_index', 'total', 'gold', 'silver', 'bronze']]


#Scale data
scaler = MinMaxScaler(feature_range = (0, 1))
scaled_data = scaler.fit_transform(features)
```

```python
#Prepare Training Data
def create_sequences(data, seq_length):
    xs, ys = [], []

    for i in range(len(data) - seq_length):
        x = data[i:i + seq_length]
        y = data[i + seq_length][4]
        xs.append(x)
        ys.append(y)
    return np.array(xs), np.array(ys)

seq_length = 60
x, y = create_sequences(scaled_data, seq_length)
```

```
#Split Data Into Training and Test Sets
split = int(0.8 * len(x))
x_train, x_test = x[:split], x[split:]
y_train, y_test = y[:split], y[split:]
```

[48]:
```
#Building the Model
model = Sequential()
model.add(LSTM(units = 50, return_sequences = True, input_shape = (x_train.shape[1], x_train.shape[2])))
model.add(Dropout(0.2))
model.add(LSTM(units = 50, return_sequences = False))
model.add(Dropout(0.2))
model.add(Dense(units = 1))

model.compile(optimizer = 'adam', loss = 'mean_squared_error')
```

```
/opt/anaconda3/lib/python3.11/site-packages/keras/src/layers/rnn/rnn.py:204: UserWarning: Do not pass an `input_shape`/`input_dim` argument t
o a layer. When using Sequential models, prefer using an `Input(shape)` object as the first layer in the model instead.
  super().__init__(**kwargs)
```

[50]:
```
#Training the Model
history = model.fit(x_train, y_train, epochs = 100, batch_size = 32, validation_split = 0.1)
```

```
Epoch 1/100
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step - loss: 0.0257 - val_loss: 0.0042
Epoch 2/100
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 270ms/step - loss: 0.0203 - val_loss: 0.0020
Epoch 3/100
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 80ms/step - loss: 0.0195 - val_loss: 0.0029
Epoch 4/100
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 75ms/step - loss: 0.0210 - val_loss: 0.0039
Epoch 5/100
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 77ms/step - loss: 0.0224 - val_loss: 0.0037
Epoch 6/100
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 72ms/step - loss: 0.0224 - val_loss: 0.0029
Epoch 7/100
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 76ms/step - loss: 0.0199 - val_loss: 0.0023
Epoch 8/100
```

[52]:
```
#Evaluating the Model
predicted_medals = model.predict(x_test)
predicted_medals = scaler.inverse_transform(np.concatenate((np.zeros((predicted_medals.shape[0], 7)), predicted_medals), axis = 1))[:, 7]

#Inverse transform the actual medals
actual_medals = scaler.inverse_transform(np.concatenate((np.zeros((y_test.shape[0], 7)), y_test.reshape(-1, 1)), axis = 1))[:, 7]
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 241ms/step
```

[54]:
```
#Calculate Performance Metrics
mae = mean_absolute_error(actual_medals, predicted_medals)
mse = mean_squared_error(actual_medals, predicted_medals)
r2 = r2_score(actual_medals, predicted_medals)

print(f'MSE: {mse}')
print(f'MAE: {mae}')
print(f'R2: {r2}')
```

```
MSE: 170.48149675824234
MAE: 8.93203161063851
R2: -0.376589078079983
```

Model Evaluation

- **Slides 19-21 uses Mean Absolute Error, Mean Squared Error, and R-Squared to assess the three models used.**
- **From the results, it was evident that a Linear Regression model was best to use for the dataset.**
- **When compared to the Deep Learning Model, Linear Regression was still better to use.**

Conclusion

- **All of the evidence shows that gdp is the most influential in predicting the medals won in a given country.**
- **Upon outside research, the United States also has the highest GDP as of 2024, followed by China and Russia further down.**
- **GDP may be an extremely influential factor in predicting medals won because it reflects a countries economy. It is likely some money is used to support country athletes.**