```
import pandas as pd
import numpy as np
import warnings
import matplotlib.pyplot as plt
warnings.filterwarnings('ignore')
%matplotlib inline

import seaborn as sns


file_path = '/content/most-visited-countries-2024.csv'
df = pd.read_csv(file_path)


def df_info(df):
    print(f"Data Shape: {df.shape}")
    df_chack = pd.DataFrame(df.dtypes, columns=['dtypes'])
    df_chack = df_chack.reset_index()
    df_chack['name'] = df_chack['index']
    df_chack = df_chack[['name', 'dtypes']]
    df_chack['isnull'] = df.isnull().sum().values
    df_chack['% null'] = round((df_chack['isnull']/len(df))*100,2)
    df_chack['num_unique'] = df.nunique().values
    df_chack['first_value'] = df.values[0]
    return df_chack


df_info(df)
```

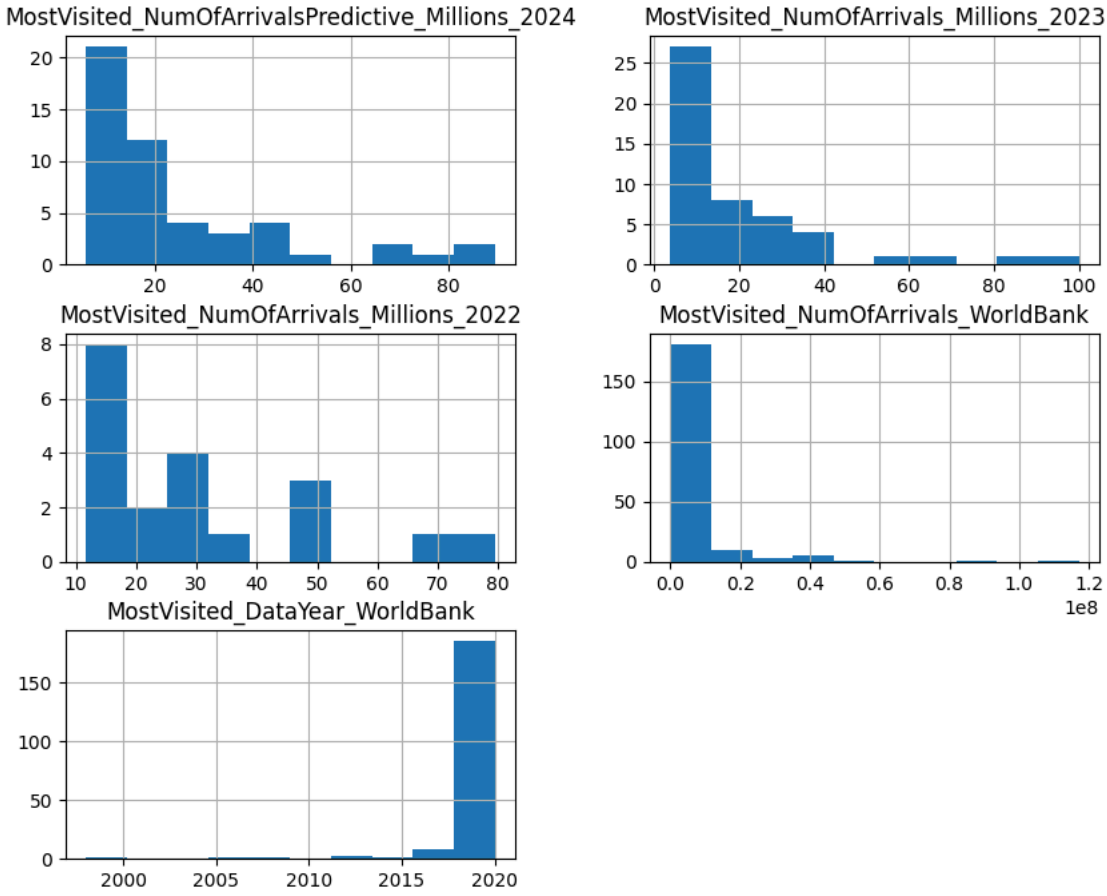
```
Data Shape: (203, 6)
```

	name	dtypes	isnull	% null	num_unique	first_value
0	country	object	0	0.00	203	India
1	MostVisited_NumOfArrivalsPredictive_Millions_2024	float64	153	75.37	47	17.9
2	MostVisited_NumOfArrivals_Millions_2023	float64	154	75.86	47	NaN
3	MostVisited_NumOfArrivals_Millions_2022	float64	183	90.15	20	NaN
4	MostVisited_NumOfArrivals_WorldBank	float64	1	0.49	198	17914000.0
5	MostVisited_DataYear_WorldBank	float64	1	0.49	12	2019.0

It seems like there are a lot of missing values

```
df.hist(figsize=(10,8))
```

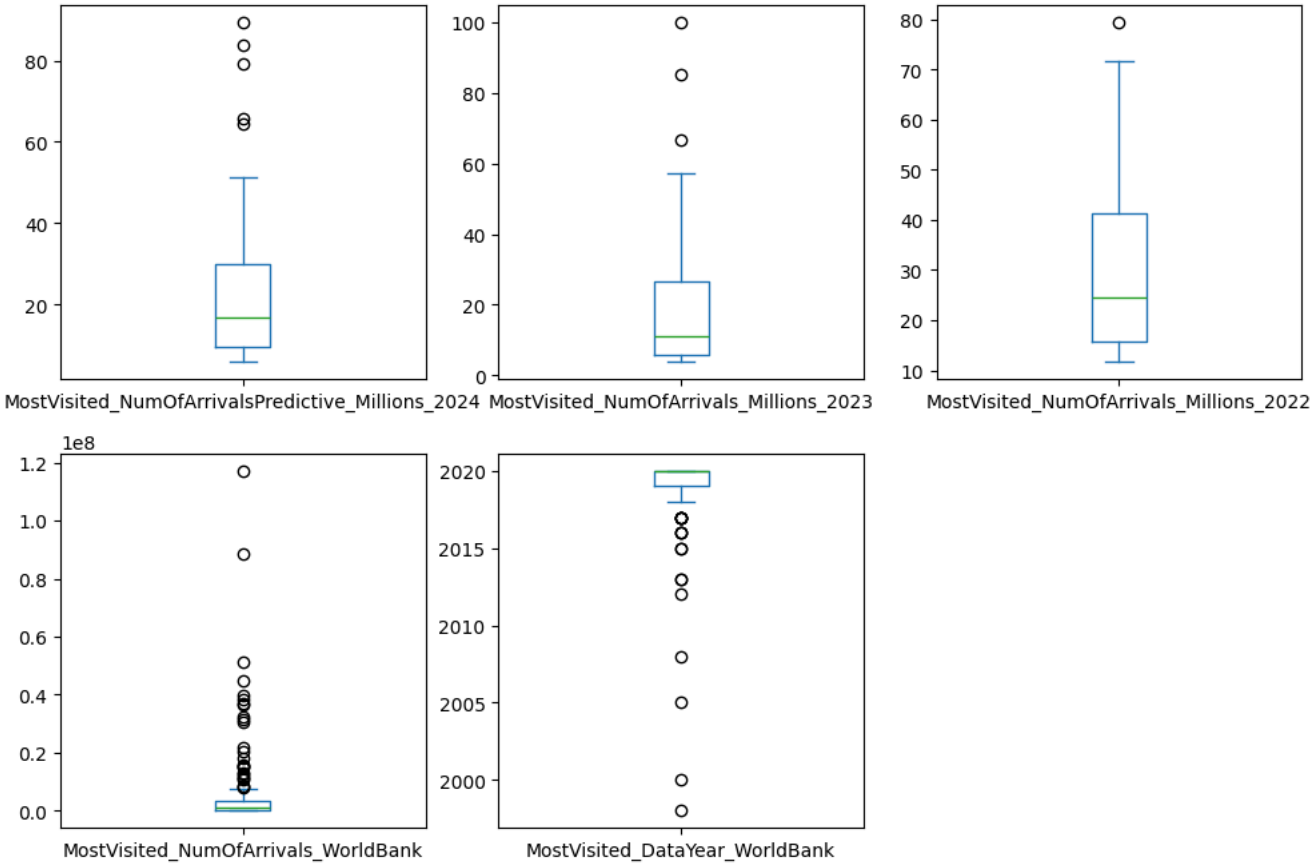
```
array([[<Axes: title={'center': 'MostVisited_NumOfArrivalsPredictive_Millions_2024'}>,
        <Axes: title={'center': 'MostVisited_NumOfArrivals_Millions_2023'}>],
       [<Axes: title={'center': 'MostVisited_NumOfArrivals_Millions_2022'}>,
        <Axes: title={'center': 'MostVisited_NumOfArrivals_WorldBank'}>],
       [<Axes: title={'center': 'MostVisited_DataYear_WorldBank'}>,
        <Axes: >]], dtype=object)
```




```
df.plot(kind='box', subplots=True, layout=(2,3), figsize=(12,8))
```

	0
MostVisited_NumOfArrivalsPredictive_Millions_2024	Axes(0.125,0.53;0.227941x0.35)
MostVisited_NumOfArrivals_Millions_2023	Axes(0.398529,0.53;0.227941x0.35)
MostVisited_NumOfArrivals_Millions_2022	Axes(0.672059,0.53;0.227941x0.35)
MostVisited_NumOfArrivals_WorldBank	Axes(0.125,0.11;0.227941x0.35)
MostVisited_DataYear_WorldBank	Axes(0.398529,0.11;0.227941x0.35)

dtype: object

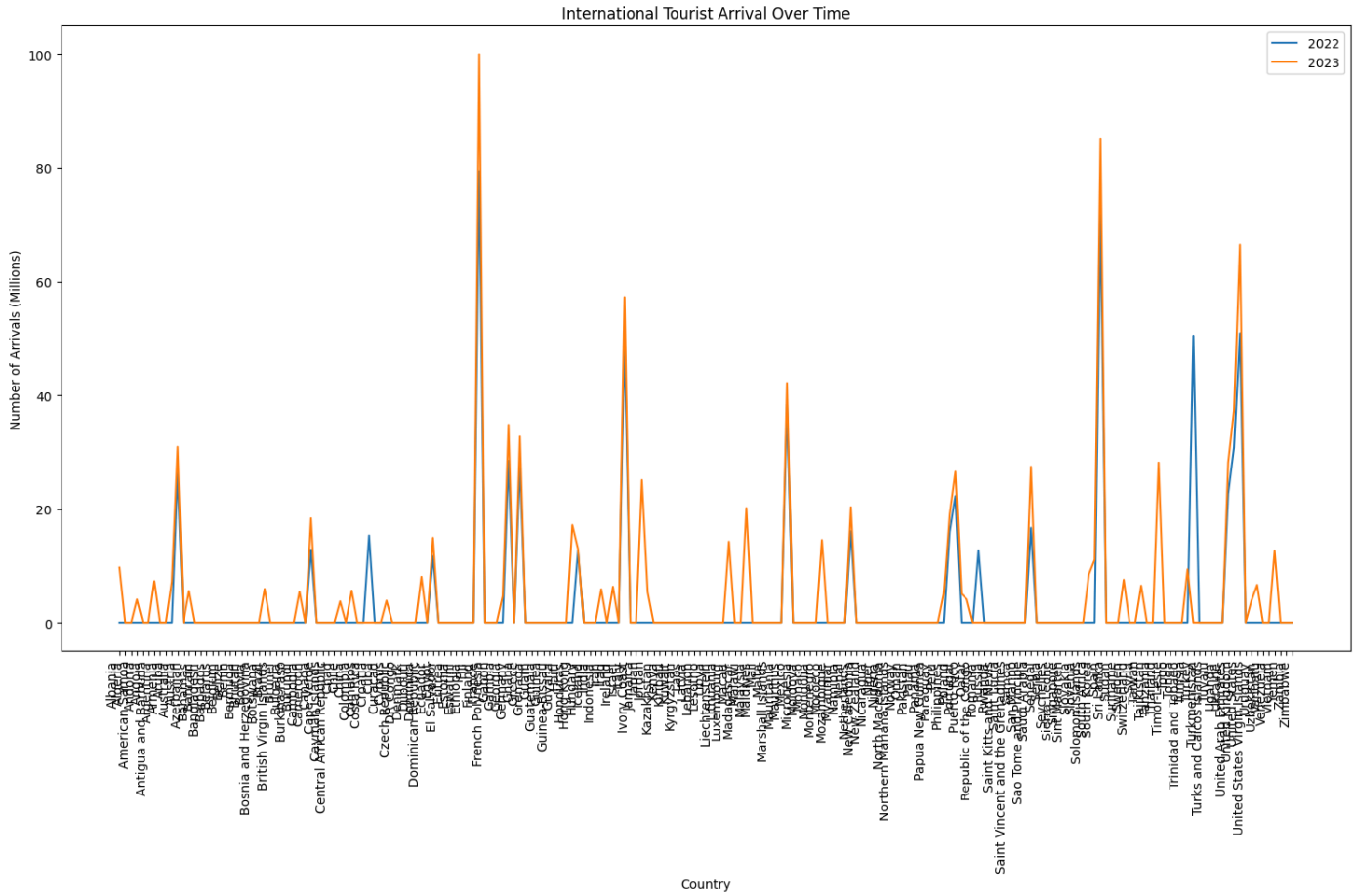



```
df_arrivals = df[['country','MostVisited_NumOfArrivals_Millions_2022', 'MostVisited_NumOfArrivals_Millions_2023']]
```

Compering the arrivals between 2022 and 2023¶

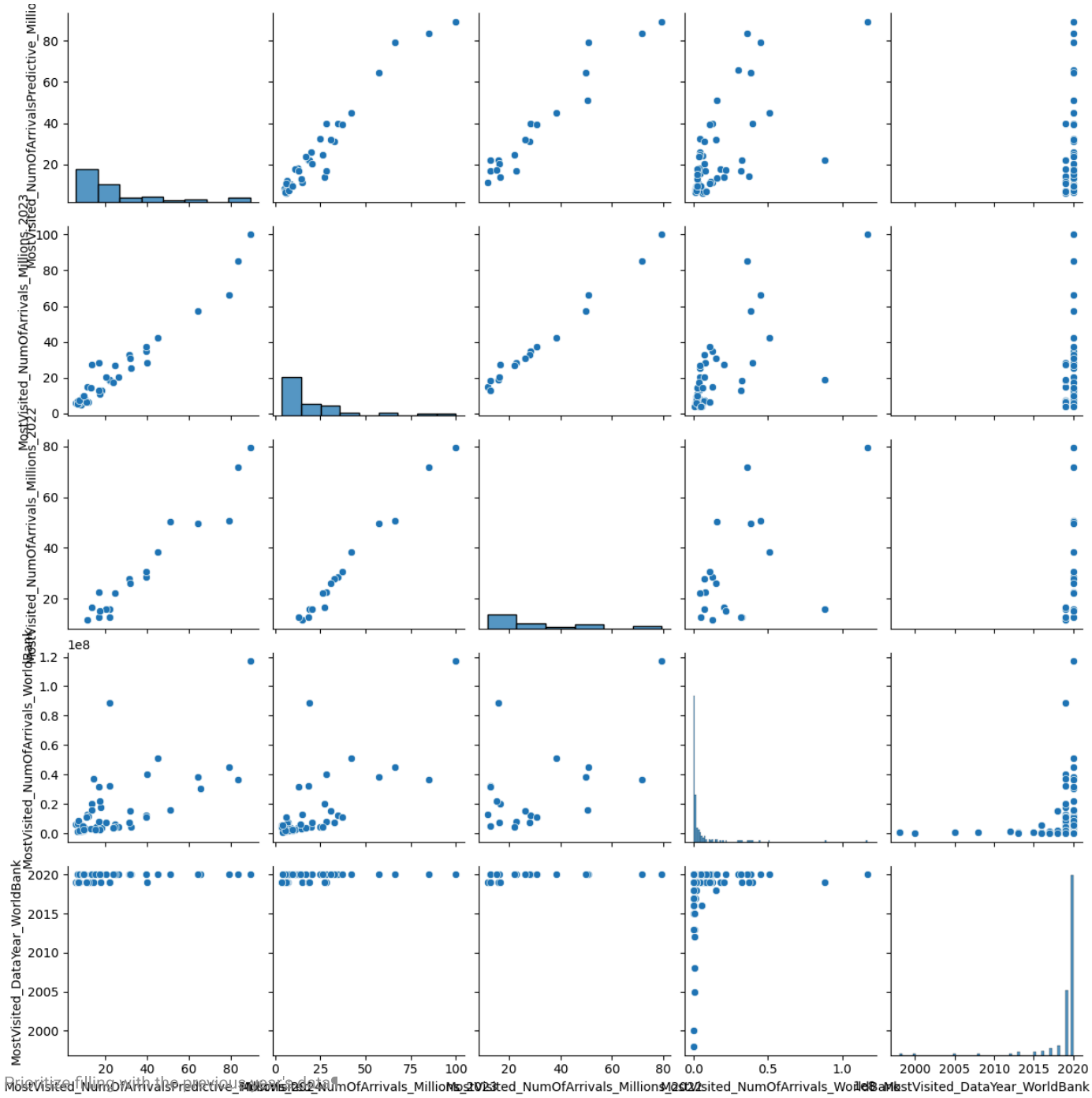
```
import matplotlib.pyplot as plt

yearly_arrivals = df_arrivals.groupby('country').sum()
plt.figure(figsize=(15, 10))
plt.plot(yearly_arrivals.index, yearly_arrivals['MostVisited_NumOfArrivals_Millions_2022'], label='2022')
plt.plot(yearly_arrivals.index, yearly_arrivals['MostVisited_NumOfArrivals_Millions_2023'], label='2023')
plt.xlabel('Country')
plt.ylabel('Number of Arrivals (Millions)')
plt.title('International Tourist Arrival Over Time')
plt.xticks(rotation=90, ha='right')
plt.legend()
plt.tight_layout()
plt.show()
```




```
# pairplot

df_numeric = df.select_dtypes(include=[np.number])
sns.pairplot(df_numeric)
plt.show()
```



1. Prioritize filling with the previous year's data

```
df['MostVisited_NumOfArrivalsPredictive_Millions_2024'].fillna(df['MostVisited_NumOfArrivals_Millions_2023'],inplace=True)
df['MostVisited_NumOfArrivals_Millions_2023'].fillna(df['MostVisited_NumOfArrivals_Millions_2022'],inplace=True)
```

```
df_info(df)
```

```
Data Shape: (203, 6)
```

	name	dtypes	isnull	% null	num_unique	first_value
0	country	object	0	0.00	203	India
1	MostVisited_NumOfArrivalsPredictive_Millions_2024	float64	141	69.46	58	17.9
2	MostVisited_NumOfArrivals_Millions_2023	float64	151	74.38	50	NaN
3	MostVisited_NumOfArrivals_Millions_2022	float64	183	90.15	20	NaN
4	MostVisited_NumOfArrivals_WorldBank	float64	1	0.49	198	17914000.0
5	MostVisited_DataYear_WorldBank	float64	1	0.49	12	2019.0

2. Then fill with the WorldBank data, but only if the column is numeric¶

```
if pd.api.types.is_numeric_dtype(df['MostVisited_NumOfArrivals_WorldBank']):
    df['MostVisited_NumOfArrivals_Millions_2022'].fillna(df['MostVisited_NumOfArrivals_WorldBank'],inplace=True)
else:
    # If WorldBank data is not numeric, try converting it first
    try:
        df['MostVisited_NumOfArrivals_WorldBank'] = pd.to_numeric(df['MostVisited_NumOfArrivals_WorldBank'], errors='coerce')
        df['MostVisited_NumOfArrivals_Millions_2022'].fillna(df['MostVisited_NumOfArrivals_WorldBank'],inplace=True)
    except:
        print("Warning: 'MostVisited_NumOfArrivals_WorldBank' could not be converted to numeric, skipping fillna.")
```

```
df_info(df)
```

```
Data Shape: (203, 6)
```

	name	dtypes	isnull	% null	num_unique	first_value
0	country	object	0	0.00	203	India
1	MostVisited_NumOfArrivalsPredictive_Millions_2024	float64	141	69.46	58	17.9
2	MostVisited_NumOfArrivals_Millions_2023	float64	151	74.38	50	NaN
3	MostVisited_NumOfArrivals_Millions_2022	float64	1	0.49	198	17914000.0
4	MostVisited_NumOfArrivals_WorldBank	float64	1	0.49	198	17914000.0
5	MostVisited_DataYear_WorldBank	float64	1	0.49	12	2019.0

3. For remaining NAs, using a more KNN imputation method¶

```
for col in ['MostVisited_NumOfArrivalsPredictive_Millions_2024', 'MostVisited_NumOfArrivals_Millions_2023', 'MostVisited_NumOfArrivals_Milli
    if df[col].isnull().any():
        from sklearn.impute import KNNImputer
        imputer = KNNImputer(n_neighbors=5)
        df[col] = imputer.fit_transform(df[[col]])[:, 0]
        print(f"Filled missing values in '{col}' using KNN imputation.")
```

```
Filled missing values in 'MostVisited_NumOfArrivalsPredictive_Millions_2024' using KNN imputation.
Filled missing values in 'MostVisited_NumOfArrivals_Millions_2023' using KNN imputation.
Filled missing values in 'MostVisited_NumOfArrivals_Millions_2022' using KNN imputation.
```

```
# Convert year columns to integer
df['MostVisited_DataYear_WorldBank'] = df['MostVisited_DataYear_WorldBank'].astype('Int64')
```

```
df_info(df)
```

```
Data Shape: (203, 6)
```

	name	dtypes	isnull	% null	num_unique	first_value
0	country	object	0	0.00	203	India
1	MostVisited_NumOfArrivalsPredictive_Millions_2024	float64	0	0.00	59	17.9
2	MostVisited_NumOfArrivals_Millions_2023	float64	0	0.00	51	19.628654
3	MostVisited_NumOfArrivals_Millions_2022	float64	0	0.00	199	17914000.0
4	MostVisited_NumOfArrivals_WorldBank	float64	1	0.49	198	17914000.0
5	MostVisited_DataYear_WorldBank	Int64	1	0.49	12	2019

Predictive Modeling¶

```
import pandas as pd
from sklearn.linear_model import LinearRegression
from sklearn.ensemble import RandomForestRegressor
from sklearn.svm import SVR
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error, r2_score
```

```
# Prepare data
```

```
X = df[['MostVisited_NumOfArrivals_Millions_2022', 'MostVisited_NumOfArrivals_Millions_2023']]
y = df['MostVisited_NumOfArrivalsPredictive_Millions_2024']

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


# Linear Reggration

model = LinearRegression()
model.fit(X_train, y_train)

y_pred_lr = model.predict(X_test)

mse_lr = mean_squared_error(y_test, y_pred_lr)
r2_lr = r2_score(y_test, y_pred_lr)


# Random Forest Regressor

model_rf = RandomForestRegressor()
model_rf.fit(X_train, y_train)

y_pred_rf = model_rf.predict(X_test)

mse_rf = mean_squared_error(y_test, y_pred_rf)
r2_rf = r2_score(y_test, y_pred_rf)


# SVR

scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

svr_model = SVR(kernel='rbf')
svr_model.fit(X_train, y_train)

y_pred_svr = svr_model.predict(X_test)

mse_svr = mean_squared_error(y_test, y_pred_svr)
r2_svr = r2_score(y_test, y_pred_svr)


# Decision Tree Regressor

from sklearn.tree import DecisionTreeRegressor

model_dt = DecisionTreeRegressor(random_state=42)
model_dt.fit(X_train, y_train)

y_pred_dt = model_dt.predict(X_test)

mse_dt = mean_squared_error(y_test, y_pred_dt)
r2_dt = r2_score(y_test, y_pred_dt)


import pandas as pd
from typing import Dict, List, Tuple

def summarize_models(model_metrics):
    summary_df = pd.DataFrame([
        {
            "Model": model_name,
            "MSE": metrics[0],
            "R2 Score": metrics[1]
        }
        for model_name, metrics in model_metrics.items()
    ])

    return summary_df.sort_values("MSE")


import tensorflow as tf
from tensorflow import keras
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from sklearn.metrics import r2_score # Import r2_score here

# Artificial Neural Network (ANN)
```

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

model_ann.compile(optimizer='adam', loss='mse')

history = model_ann.fit(X_train, y_train, epochs=100, batch_size=32, validation_split=0.15, verbose=0)

loss_ann = model_ann.evaluate(X_test, y_test, verbose=0)
print(f"Artificial Neural Network MSE: {loss_ann}")

y_pred_ann = model_ann.predict(X_test)
r2_ann = r2_score(y_test, y_pred_ann)
print(f"Artificial Neural Network R2 Score: {r2_ann}")

# Assign the loss_ann value to mse_ann
mse_ann = loss_ann
```

```
Artificial Neural Network MSE: 19.420209884643555
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 49ms/step
Artificial Neural Network R2 Score: 0.5636840531294429
```

```
# Recurrent Neural Network (RNN)

from tensorflow.keras.layers import SimpleRNN
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from sklearn.metrics import r2_score

X_train_rnn = X_train.reshape(X_train.shape[0], 1, X_train.shape[1])
X_test_rnn = X_test.reshape(X_test.shape[0], 1, X_test.shape[1])


model_rnn = Sequential([
    # Add a SimpleRNN layer. units is the dimensionality of the output space.
    # input_shape is (timesteps, features).
    SimpleRNN(64, activation='relu', input_shape=(X_train_rnn.shape[1], X_train_rnn.shape[2])),
    # Add a Dense layer with ReLU activation
    Dense(32, activation='relu'),
    # Output layer for regression
    Dense(1)
])

model_rnn.compile(optimizer='adam', loss='mse')

# Train the RNN model
history_rnn = model_rnn.fit(X_train_rnn, y_train, epochs=100, batch_size=32, validation_split=0.2, verbose=0)

# Evaluate the RNN model
loss_rnn = model_rnn.evaluate(X_test_rnn, y_test, verbose=0)
print(f"Recurrent Neural Network MSE: {loss_rnn}")

# Make predictions with the RNN model
y_pred_rnn = model_rnn.predict(X_test_rnn)

# Calculate R2 score for the RNN model
r2_rnn = r2_score(y_test, y_pred_rnn)
print(f"Recurrent Neural Network R2 Score: {r2_rnn}")

# Assign the loss_rnn value to mse_rnn
mse_rnn = loss_rnn
```

```
Recurrent Neural Network MSE: 24.797306060791016
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 149ms/step
Recurrent Neural Network R2 Score: 0.44287615021002835
```

```
metrics = {
    "Linear Regression": (mse_lr, r2_lr),
    "Random Forest": (mse_rf, r2_rf),
    "Support Vector Regression": (mse_svr, r2_svr),
    "Decision Tree": (mse_dt, r2_dt),
    "Artificial Neural Network" : (mse_ann, r2_ann),
    "Recurrent Neural Network (RNN)": (mse_rnn, r2_rnn
```

```
}

summary = summarize_models(metrics)
```

```
                             Model        MSE  R2 Score
2       Support Vector Regression  14.664297  0.670536
0                Linear Regression  14.857431  0.666196
4       Artificial Neural Network  19.420210  0.563684
1                   Random Forest  23.319272  0.476083
5  Recurrent Neural Network (RNN)  24.797306  0.442876
3                   Decision Tree  67.700071 -0.521025
```

```
import matplotlib.pyplot as plt
import pandas as pd
from IPython.display import display, HTML


metrics = {
    "Linear Regression": (mse_lr, r2_lr),
    "Random Forest": (mse_rf, r2_rf),
    "Support Vector Regression": (mse_svr, r2_svr),
    "Decision Tree": (mse_dt, r2_dt),
    "Artificial Neural Network" : (mse_ann, r2_ann),
    "Recurrent Neural Network (RNN)": (mse_rnn, r2_rnn)
}

summary_df = pd.DataFrame([
    {
        "Model": model_name,
        "MSE": metrics[0],
        "R2 Score": metrics[1]
    }
    for model_name, metrics in metrics.items()
])

summary_df = summary_df.sort_values("MSE")

colors = ['skyblue', 'lightgreen', 'salmon', 'gold', 'purple', 'orange']

plt.figure(figsize=(12, 6))
plt.bar(summary_df["Model"], summary_df["MSE"], color=colors)
plt.xlabel("Model")
```