In [ ]:
```
import pandas as pd
from sklearn.preprocessing import MinMaxScaler
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_squared_error, r2_score
import matplotlib.pyplot as plt
```

In [ ]:
```
import plotly.express as px
import plotly.graph_objects as go
import plotly.offline as pyo
#import dash
```

In [ ]:
```
# Load the dataset as a CSV file into a Pandas DataFrame
data_path = '/Users/Elena/Documents/GitHub/steel_strength/metals_data.csv'  # Updat
data_csv_df = pd.read_csv(data_path)
data_csv_df
```

Out[ ]:

	c	mn	si	cr	ni	mo	v	n	nb	co	w	al	ti
0	0.02	0.05	0.05	0.01	19.70	2.95	0.01	0.00	0.01	15.00	0.00	0.15	1.55
1	0.18	0.01	0.01	13.44	0.01	3.01	0.46	0.04	0.01	19.46	2.35	0.04	0.00
2	0.00	0.01	0.01	8.67	13.45	0.82	0.01	0.00	0.01	13.90	0.00	0.39	0.57
3	0.01	0.05	0.05	0.01	17.70	3.95	0.01	0.00	0.01	15.00	0.00	0.13	1.47
4	0.01	0.05	0.05	0.01	19.40	1.45	0.01	0.00	0.01	14.90	0.00	0.13	1.55
...	...	...	...	...	...	...	...	...	...	...	...	...	...
307	0.38	0.18	0.01	7.27	0.01	3.77	0.96	0.00	0.01	4.90	0.00	0.03	0.00
308	0.00	0.06	0.05	5.15	10.20	3.20	0.01	0.00	0.01	0.01	0.00	0.03	0.09
309	0.37	0.17	0.01	5.20	0.01	5.84	1.05	0.00	0.91	4.88	0.00	0.03	0.00
310	0.41	0.19	0.01	6.99	0.01	5.84	0.92	0.00	0.01	0.02	0.00	0.03	0.00
311	0.27	0.27	0.01	0.41	8.28	0.49	0.07	0.00	0.01	3.90	0.00	0.03	0.00

312 rows × 13 columns

In [ ]:
```
type(data_csv_df)
```

Out[ ]:
```
pandas.core.frame.DataFrame
```

In [ ]:
```
data_csv_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 312 entries, 0 to 311
Data columns (total 13 columns):
 #   Column  Non-Null Count  Dtype  
---  ------  --------------  -----  
 0   c       312 non-null    float64
 1   mn      312 non-null    float64
 2   si      312 non-null    float64
 3   cr      312 non-null    float64
 4   ni      312 non-null    float64
 5   mo      312 non-null    float64
 6   v       312 non-null    float64
 7   n       312 non-null    float64
 8   nb      312 non-null    float64
 9   co      312 non-null    float64
 10  w       312 non-null    float64
 11  al      312 non-null    float64
 12  ti      312 non-null    float64
dtypes: float64(13)
memory usage: 31.8 KB
```

In [ ]:

```python
# Load the dataset
data_path = '/Users/Elena/Documents/GitHub/steel_strength/metals_data.xlsx'  # Upda
data_xlsx_df = pd.read_excel(data_path)
print(f"data_xlsx_df: {data_xlsx_df.shape} rows and columns")
print(f"data_xlsx_df: {data_xlsx_df.columns} column names")
print(f"data_xlsx_df: {data_xlsx_df.index} row indices")
print()
print(f"data_xlsx_df: {data_xlsx_df.dtypes} data types")
print()
print(f"data_xlsx_df: {data_xlsx_df.info()} dataframe profile information")
print()
print(f"data_xlsx_df: {data_xlsx_df.describe()} descriptive statistics for the data
print()
print(f"data_xlsx_df: {data_xlsx_df.head()} data types \n\n")

data_xlsx_df
```

```
data_xlsx_df: (312, 13) rows and columns
data_xlsx_df: Index(['c', 'mn', 'si', 'cr', 'ni', 'mo', 'v', 'n', 'nb', 'co', 'w',
'al',
       'ti'],
      dtype='object') column names
data_xlsx_df: RangeIndex(start=0, stop=312, step=1) row indices

data_xlsx_df: c     float64
mn    float64
si    float64
cr    float64
ni    float64
mo    float64
v     float64
n     float64
nb    float64
co    float64
w     float64
al    float64
ti    float64
dtype: object data types

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 312 entries, 0 to 311
Data columns (total 13 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   c       312 non-null    float64
 1   mn      312 non-null    float64
 2   si      312 non-null    float64
 3   cr      312 non-null    float64
 4   ni      312 non-null    float64
 5   mo      312 non-null    float64
 6   v       312 non-null    float64
 7   n       312 non-null    float64
 8   nb      312 non-null    float64
 9   co      312 non-null    float64
 10  w       312 non-null    float64
 11  al      312 non-null    float64
 12  ti      312 non-null    float64
dtypes: float64(13)
memory usage: 31.8 KB
data_xlsx_df: None dataframe profile information

data_xlsx_df:                 c          mn          si          cr          ni
mo  \
count  312.000000  312.000000  312.000000  312.000000  312.000000  312.000000
mean     0.096442    0.146250    0.221218    8.043830    8.184006    2.766090
std      0.109008    0.397102    0.580796    5.426169    6.337055    1.832908
min      0.000000    0.010000    0.010000    0.010000    0.010000    0.020000
25%      0.010000    0.010000    0.010000    3.100000    0.960000    1.500000
50%      0.030000    0.010000    0.010000    9.050000    8.500000    2.210000
75%      0.182500    0.080000    0.110000   12.520000   12.117500    4.090000
max      0.430000    3.000000    4.750000   17.500000   21.000000    9.670000

                v           n          nb          co           w          al  \
```

```
count  312.000000  312.000000  312.000000  312.000000  312.000000  312.000000
mean     0.183750    0.005545    0.035449    7.008782    0.161282    0.239135
std      0.452462    0.018331    0.161537    6.254431    0.920211    0.340110
min      0.000000    0.000000    0.000000    0.010000    0.000000    0.010000
25%      0.010000    0.000000    0.010000    0.010000    0.000000    0.030000
50%      0.010000    0.000000    0.010000    7.085000    0.000000    0.050000
75%      0.127500    0.000000    0.010000   13.480000    0.000000    0.300000
max      4.320000    0.150000    2.500000   20.100000    9.180000    1.800000

              ti
count  312.000000
mean     0.310897
std      0.556593
min      0.000000
25%      0.000000
50%      0.030000
75%      0.232500
max      2.500000   descriptive statistics for the dataframe (for numeric columns)

data_xlsx_df:       c    mn    si     cr     ni    mo     v     n    nb     co     w
al  \
0  0.02  0.05  0.05   0.01  19.70  2.95  0.01  0.00  0.01  15.00  0.00  0.15
1  0.18  0.01  0.01  13.44   0.01  3.01  0.46  0.04  0.01  19.46  2.35  0.04
2  0.00  0.01  0.01   8.67  13.45  0.82  0.01  0.00  0.01  13.90  0.00  0.39
3  0.01  0.05  0.05   0.01  17.70  3.95  0.01  0.00  0.01  15.00  0.00  0.13
4  0.01  0.05  0.05   0.01  19.40  1.45  0.01  0.00  0.01  14.90  0.00  0.13

     ti
0  1.55
1  0.00
2  0.57
3  1.47
4  1.55   data types
```

Out[ ]:

	c	mn	si	cr	ni	mo	v	n	nb	co	w	al	ti
0	0.02	0.05	0.05	0.01	19.70	2.95	0.01	0.00	0.01	15.00	0.00	0.15	1.55
1	0.18	0.01	0.01	13.44	0.01	3.01	0.46	0.04	0.01	19.46	2.35	0.04	0.00
2	0.00	0.01	0.01	8.67	13.45	0.82	0.01	0.00	0.01	13.90	0.00	0.39	0.57
3	0.01	0.05	0.05	0.01	17.70	3.95	0.01	0.00	0.01	15.00	0.00	0.13	1.47
4	0.01	0.05	0.05	0.01	19.40	1.45	0.01	0.00	0.01	14.90	0.00	0.13	1.55
...	...	...	...	...	...	...	...	...	...	...	...	...	...
307	0.38	0.18	0.01	7.27	0.01	3.77	0.96	0.00	0.01	4.90	0.00	0.03	0.00
308	0.00	0.06	0.05	5.15	10.20	3.20	0.01	0.00	0.01	0.01	0.00	0.03	0.09
309	0.37	0.17	0.01	5.20	0.01	5.84	1.05	0.00	0.91	4.88	0.00	0.03	0.00
310	0.41	0.19	0.01	6.99	0.01	5.84	0.92	0.00	0.01	0.02	0.00	0.03	0.00
311	0.27	0.27	0.01	0.41	8.28	0.49	0.07	0.00	0.01	3.90	0.00	0.03	0.00

312 rows × 13 columns

In [ ]:
```
# Sort by column: 'co' (descending)
data = data_xlsx_df.sort_values(['co'], ascending=[False])
```

In [ ]:
```
data
```

Out[ ]:

	c	mn	si	cr	ni	mo	v	n	nb	co	w	al	ti
5	0.19	0.02	0.49	12.56	0.94	1.96	0.01	0.00	0.01	20.10	0.00	0.03	0.00
1	0.18	0.01	0.01	13.44	0.01	3.01	0.46	0.04	0.01	19.46	2.35	0.04	0.00
85	0.35	0.01	0.01	9.00	1.50	2.00	0.30	0.00	0.01	18.00	0.00	0.03	0.02
25	0.20	0.01	0.51	12.51	0.95	1.97	0.01	0.00	0.08	17.06	0.00	0.03	0.00
15	0.20	0.01	0.01	14.13	0.92	1.99	0.01	0.00	0.08	17.00	0.00	0.03	0.00
...	...	...	...	...	...	...	...	...	...	...	...	...	...
227	0.01	0.05	0.09	7.65	12.20	3.05	0.01	0.00	0.01	0.01	0.00	0.46	0.20
226	0.01	0.12	0.06	10.30	10.20	2.05	0.01	0.00	0.01	0.01	0.00	0.31	0.21
225	0.01	0.03	0.03	10.20	10.50	2.20	0.01	0.00	0.01	0.01	0.00	0.23	0.24
224	0.01	0.11	0.18	10.20	10.30	2.05	0.01	0.00	0.01	0.01	0.00	0.27	0.22
187	0.04	0.01	0.01	12.49	8.30	2.23	0.01	0.00	0.01	0.01	0.00	1.07	0.01

312 rows × 13 columns

In [ ]:
```
# Normalize the dataset
scaler = MinMaxScaler()
data_normalized = scaler.fit_transform(data)
data_normalized_df = pd.DataFrame(data_normalized, columns=data.columns)
```

In [ ]:
```
# Split the dataset into features and target variable
# Assuming the target variable is the last column
X = data_normalized_df.iloc[:, :-1]  # Features
y = data_normalized_df.iloc[:, -1]   # Target
```

In [ ]:
```
# Split the data into training and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_sta
```

In [ ]:
```
X_train, X_test, y_train, y_test
```

Out[ ]:
```
(           c        mn        si        cr        ni        mo         v  \
111  0.000000  0.013378  0.008439  0.000000  0.857075  0.370984  0.115741
208  0.023256  0.016722  0.008439  0.445397  0.485469  0.329534  0.002315
145  0.023256  0.000000  0.000000  0.626072  0.379228  0.516062  0.002315
203  0.023256  0.023411  0.008439  0.265294  0.576465  0.322280  0.002315
78   0.279070  0.000000  0.000000  0.829617  0.000000  0.525389  0.099537
..        ...       ...       ...       ...       ...       ...       ...
188  0.000000  0.000000  0.000000  0.817038  0.394950  0.267358  0.002315
71   0.511628  0.000000  0.000000  0.177244  0.534064  0.121244  0.002315
106  0.023256  0.013378  0.008439  0.000000  0.818961  0.407254  0.002315
270  0.023256  0.020067  0.021097  0.645512  0.490233  0.210363  0.002315
102  0.558140  0.000000  0.000000  0.514008  0.132920  0.132642  0.067130

       n     nb        co    w        al
111  0.0  0.004  0.596814  0.0  0.005587
208  0.0  0.004  0.000000  0.0  0.027933
145  0.0  0.004  0.425087  0.0  0.670391
203  0.0  0.004  0.000000  0.0  0.312849
78   0.2  0.004  0.670483  0.0  0.011173
..   ...    ...       ...  ...       ...
188  0.0  0.004  0.213539  0.0  0.050279
71   0.0  0.004  0.671976  0.0  0.000000
106  0.0  0.004  0.606770  0.0  0.122905
270  0.0  0.004  0.000000  0.0  0.128492
102  0.0  0.004  0.616725  0.0  0.011173

[249 rows x 12 columns],
           c        mn        si        cr        ni        mo         v  \
228  0.162791  0.197324  0.061181  0.971412  0.333016  0.000000  0.002315
9    0.372093  0.000000  0.000000  0.831904  0.000000  0.505699  0.111111
57   0.488372  0.000000  0.116034  0.716409  0.046689  0.202073  0.078704
60   0.488372  0.000000  0.103376  0.716981  0.044307  0.203109  0.002315
25   0.465116  0.003344  0.101266  0.716981  0.044307  0.202073  0.002315
..        ...       ...       ...       ...       ...       ...       ...
196  0.023256  0.030100  0.035865  0.277873  0.575036  0.305699  0.002315
211  0.000000  0.020067  0.014768  0.428245  0.499762  0.339896  0.002315
225  0.000000  0.016722  0.008439  0.293882  0.485469  0.329534  0.002315
109  0.279070  0.056856  0.010549  0.114351  0.479276  0.101554  0.002315
114  0.372093  0.000000  0.000000  0.822756  0.000000  0.545078  0.094907

           n     nb        co    w        al
228  0.000000  0.004  0.000000  0.0  0.664804
9    0.333333  0.004  0.776008  0.0  0.016760
57   0.000000  0.032  0.684420  0.0  0.011173
60   0.000000  0.032  0.683922  0.0  0.011173
25   0.000000  0.008  0.750124  0.0  0.011173
..        ...    ...       ...  ...       ...
196  0.000000  0.004  0.000000  0.0  0.134078
211  0.000000  0.004  0.000000  0.0  0.111732
225  0.000000  0.004  0.000000  0.0  0.011173
109  0.000000  0.004  0.601792  0.0  0.000000
114  0.866667  0.004  0.583873  0.0  0.016760

[63 rows x 12 columns],
111    0.760
208    0.040
```

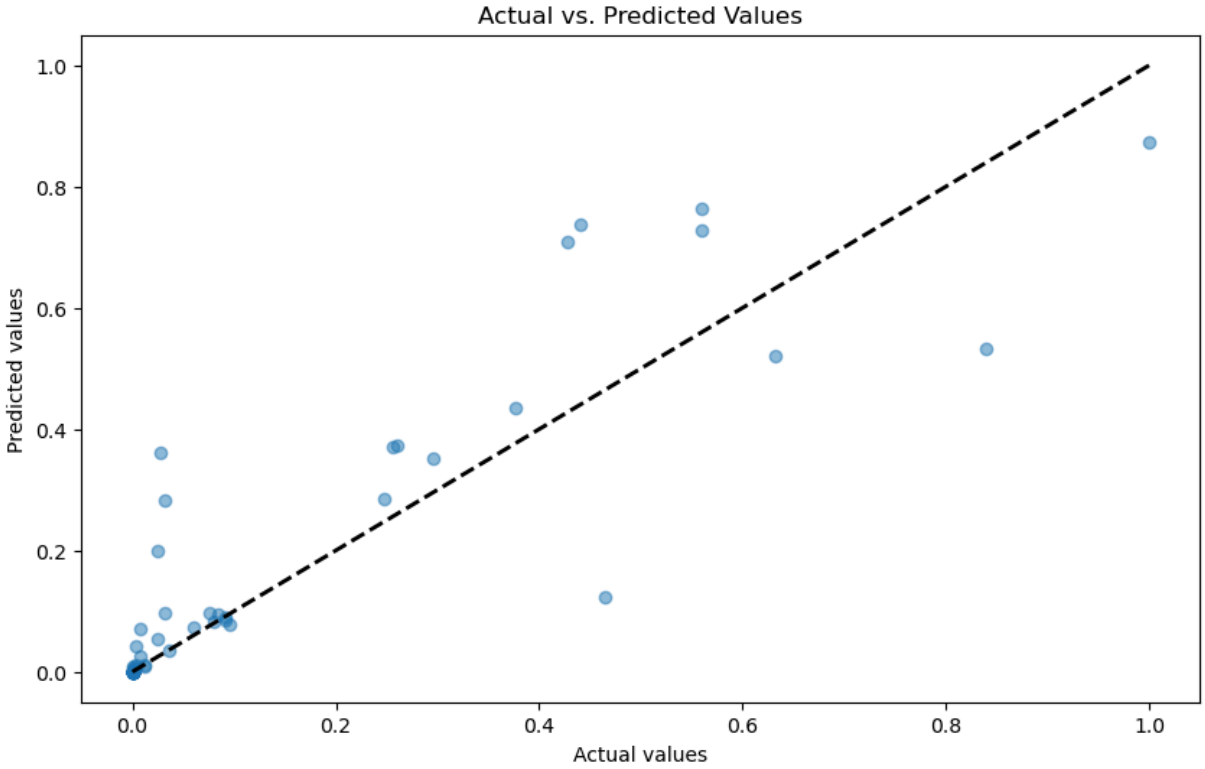
```
145    0.004
203    0.080
78     0.000
       ...
188    0.196
71     0.004
106    0.712
270    0.092
102    0.008
Name: ti, Length: 249, dtype: float64,
228    0.000
9      0.000
57     0.000
60     0.000
25     0.000
       ...
196    0.060
211    0.092
225    0.036
109    0.000
114    0.000
Name: ti, Length: 63, dtype: float64)
```

In [ ]:
```python
type(X_train), type(X_test), type(y_train), type(y_test)
```

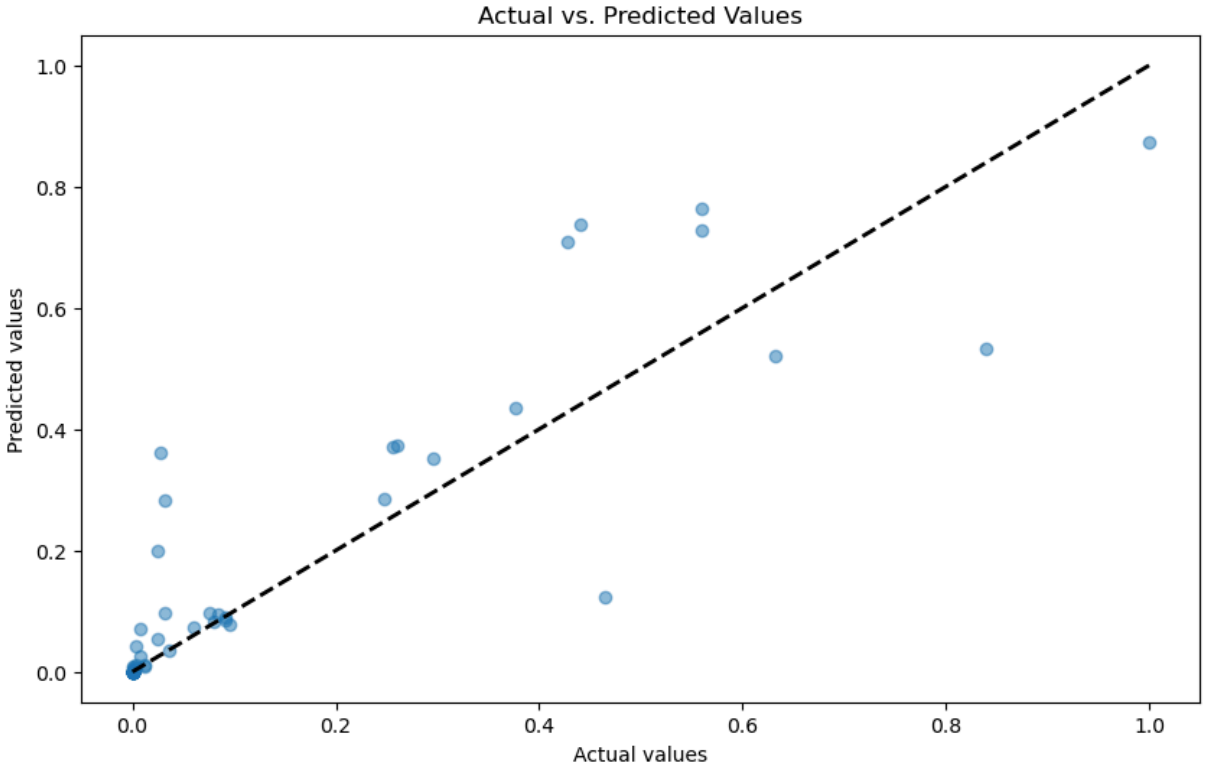
Out[ ]:
```
(pandas.core.frame.DataFrame,
 pandas.core.frame.DataFrame,
 pandas.core.series.Series,
 pandas.core.series.Series)
```

In [ ]:
```python
# Initialize the Random Forest Regressor
model = RandomForestRegressor(n_estimators=100, random_state=42)

# Fit the model on the training data
model.fit(X_train, y_train)

# Make predictions on the test set
y_pred = model.predict(X_test)

# Evaluate the model
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

print(f"Mean Squared Error: {mse}")
print(f"R² Score: {r2}")

# Plotting actual vs. predicted values
plt.figure(figsize=(10, 6))
plt.scatter(y_test, y_pred, alpha=0.5)
plt.xlabel('Actual values')
plt.ylabel('Predicted values')
plt.title('Actual vs. Predicted Values')
plt.plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max()], 'k--', lw=2)
plt.show()
```

```
Mean Squared Error: 0.011563619707936503
R² Score: 0.7561150278352762
```



In [ ]:
```python
# Plotting actual vs. predicted values
plt.figure(figsize=(10, 6))
plt.scatter(y_test, y_pred, alpha=0.5)
plt.xlabel('Actual values')
plt.ylabel('Predicted values')
plt.title('Actual vs. Predicted Values')
plt.plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max()], 'k--', lw=2)
plt.show()
```



Here is how you can create an interactive scatter plot using Plotly in Python for your data:

In [ ]:
```python
import plotly.graph_objects as go

# Create a trace for the scatter plot
trace1 = go.Scatter(
    x=y_test,
    y=y_pred,
    mode='markers',
    name='Data',
    marker=dict(color='rgba(152, 0, 0, .8)'),
)

# Create a trace for the line
trace2 = go.Scatter(
    x=[y_test.min(), y_test.max()],
    y=[y_test.min(), y_test.max()],
    mode='lines',
    name='Fit',
    line=dict(color='black', dash='dash')
)

data = [trace1, trace2]

layout = go.Layout(
    title='Actual vs. Predicted Values',
    xaxis=dict(title='Actual values'),
    yaxis=dict(title='Predicted values'),
)

fig = go.Figure(data=data, layout=layout)
```

```
fig.show()
```

This code creates an interactive scatter plot of `y_test` vs `y_pred` and a line from the minimum to the maximum of `y_test`. The plot is displayed in the Jupyter notebook and is interactive, meaning you can zoom, pan, hover over data points to see their values, and so on.