Predicting strength of Al Alloys

Importing Libraries

In [1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split,GridSearchCV,RandomizedSearchCV
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import MinMaxScaler
from sklearn.pipeline import make_pipeline
from sklearn.ensemble import RandomForestRegressor,BaggingRegressor
from sklearn.metrics import mean_absolute_error,mean_squared_error,r2_score,mean_squared_log_error
import plotly.express as px
```

Reading .csv file

In [2]:
```
df_raw = pd.read_csv('al-alloys.csv')
df_raw.head()
```

Out[2]:

	ID	X	Fe (wt%)	Mn (wt%)	Si (wt%)	Al (wt%)	Mg (wt%)	Ti (wt%)	Cu (wt%)	Cr (wt%)	V (wt%)	Zr (wt%)	Zn (wt%)	2% proof stress (Mpa)	Tensile strength (Mpa)	Elongation (%)
0	A 5005 P	1	0.35	0.1	0.15	98.33	0.8	0.0	0.1	0.05	0.0	0.0	0.125	95	125	2
1	A 5005 P	1	0.35	0.1	0.15	98.33	0.8	0.0	0.1	0.05	0.0	0.0	0.125	120	145	2
2	A 5005 P	1	0.35	0.1	0.15	98.33	0.8	0.0	0.1	0.05	0.0	0.0	0.125	145	165	2
3	A 5005 P	1	0.35	0.1	0.15	98.33	0.8	0.0	0.1	0.05	0.0	0.0	0.125	165	185	2
4	A 5005 P	2	0.35	0.1	0.15	98.33	0.8	0.0	0.1	0.05	0.0	0.0	0.125	85	120	4

In [3]:
```
df_raw.shape
```

Out[3]:
```
(173, 16)
```

Refining dataset for Tensile Strength

In [4]:
```
df_raw.columns
```

Out[4]:
```
Index(['ID', 'X', 'Fe (wt%)', 'Mn (wt%)', 'Si (wt%)', 'Al (wt%)', 'Mg (wt%)',
       'Ti (wt%)', 'Cu (wt%)', 'Cr (wt%)', 'V (wt%)', 'Zr (wt%)', 'Zn (wt%)',
       '2% proof stress (Mpa)', 'Tensile strength (Mpa)', 'Elongation (%)'],
      dtype='object')
```

In [5]:
```
df = df_raw.drop(['ID','X','2% proof stress (Mpa)','Elongation (%)'],axis=1)
df.head()
```

Out[5]:

	Fe (wt%)	Mn (wt%)	Si (wt%)	Al (wt%)	Mg (wt%)	Ti (wt%)	Cu (wt%)	Cr (wt%)	V (wt%)	Zr (wt%)	Zn (wt%)	Tensile strength (Mpa)
0	0.35	0.1	0.15	98.33	0.8	0.0	0.1	0.05	0.0	0.0	0.125	125
1	0.35	0.1	0.15	98.33	0.8	0.0	0.1	0.05	0.0	0.0	0.125	145
2	0.35	0.1	0.15	98.33	0.8	0.0	0.1	0.05	0.0	0.0	0.125	165
3	0.35	0.1	0.15	98.33	0.8	0.0	0.1	0.05	0.0	0.0	0.125	185
4	0.35	0.1	0.15	98.33	0.8	0.0	0.1	0.05	0.0	0.0	0.125	120

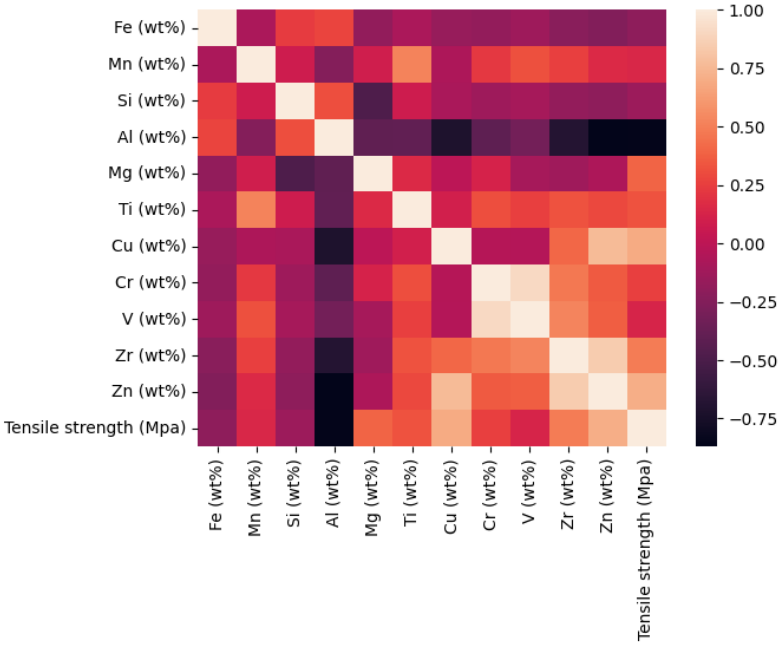
In [6]:
```
df.shape
```

Out[6]:
```
(173, 12)
```

In [7]:
```
attribute = df.corr()
```

In [8]:
```
sns.heatmap(attribute)
```

Out[8]:
```
<Axes: >
```



In [9]:
```
df.corr()['Tensile strength (Mpa)'].sort_values(ascending=False)
```

Out[9]:
```
Tensile strength (Mpa)    1.000000
Zn (wt%)                  0.700871
Cu (wt%)                  0.687078
Zr (wt%)                  0.485760
Mg (wt%)                  0.393391
Ti (wt%)                  0.326048
Cr (wt%)                  0.253523
Mn (wt%)                  0.142214
V (wt%)                   0.128569
Si (wt%)                 -0.145395
Fe (wt%)                 -0.203622
Al (wt%)                 -0.860886
Name: Tensile strength (Mpa), dtype: float64
```

In [10]:
```
fig = px.scatter(data_frame=df, x="Tensile strength (Mpa)", y= "Zn (wt%)", size= "Cu (wt%)", color= "Al (wt%)",
                 labels={"Al (wt%)":"Aluminium"}, title="Tensile Strength vs Zinc vs Copper vs Aluminium")
fig.update_layout(yaxis_title="Zinc (wt%)")
fig.show()
```

Getting metrics about the dataset

In [11]:

```python
unique_val = []

for cols in df.columns:
    uniq = df[cols].value_counts().unique().sum()
    unique_val.append(uniq)

identical_val = []

for i in unique_val:
    j = 173-i
    identical_val.append(j)

null_val = []

for null in df.columns:
    null_values = df[cols].isnull().sum()
    null_val.append(null_values)
```

In [12]:

```python
unique_values = pd.DataFrame({'Attribute': df.columns, 'Unique Values': unique_val, 'Identical Values': identical_val,
                              'Null Values': null_val})
unique_values.T
```

Out[12]:

	0	1	2	3	4	5	6	7	8	9	10	11
Attribute	Fe (wt%)	Mn (wt%)	Si (wt%)	Al (wt%)	Mg (wt%)	Ti (wt%)	Cu (wt%)	Cr (wt%)	V (wt%)	Zr (wt%)	Zn (wt%)	Tensile strength (Mpa)
Unique Values	142	162	153	92	120	171	161	162	173	173	124	37
Identical Values	31	11	20	81	53	2	12	11	0	0	49	136
Null Values	0	0	0	0	0	0	0	0	0	0	0	0

In [13]:

```python
df.describe()
```

Out[13]:

	Fe (wt%)	Mn (wt%)	Si (wt%)	Al (wt%)	Mg (wt%)	Ti (wt%)	Cu (wt%)	Cr (wt%)	V (wt%)	Zr (wt%)	Zn (wt%)	Tensi streng (Mp
count	173.000000	173.000000	173.000000	173.000000	173.000000	173.000000	173.000000	173.000000	173.000000	173.000000	173.000000	173.00000
mean	0.227514	0.164393	0.253873	95.952659	2.153295	0.045318	0.186705	0.175723	0.001734	0.014682	0.817168	261.43352
std	0.068303	0.170069	0.199373	2.541454	1.249819	0.040870	0.417126	0.204431	0.009175	0.041993	1.887479	93.98714
min	0.060000	0.010000	0.050000	87.050000	0.050000	0.000000	0.020000	0.000000	0.000000	0.000000	0.015000	110.00000
25%	0.200000	0.050000	0.130000	95.630000	0.900000	0.000000	0.050000	0.050000	0.000000	0.000000	0.050000	200.00000
50%	0.200000	0.080000	0.200000	96.780000	2.250000	0.050000	0.050000	0.150000	0.000000	0.000000	0.100000	255.00000
75%	0.250000	0.250000	0.230000	97.680000	3.100000	0.080000	0.100000	0.250000	0.000000	0.000000	0.125000	305.00000
max	0.500000	0.750000	1.000000	99.190000	5.050000	0.170000	2.300000	1.150000	0.050000	0.150000	7.800000	580.00000

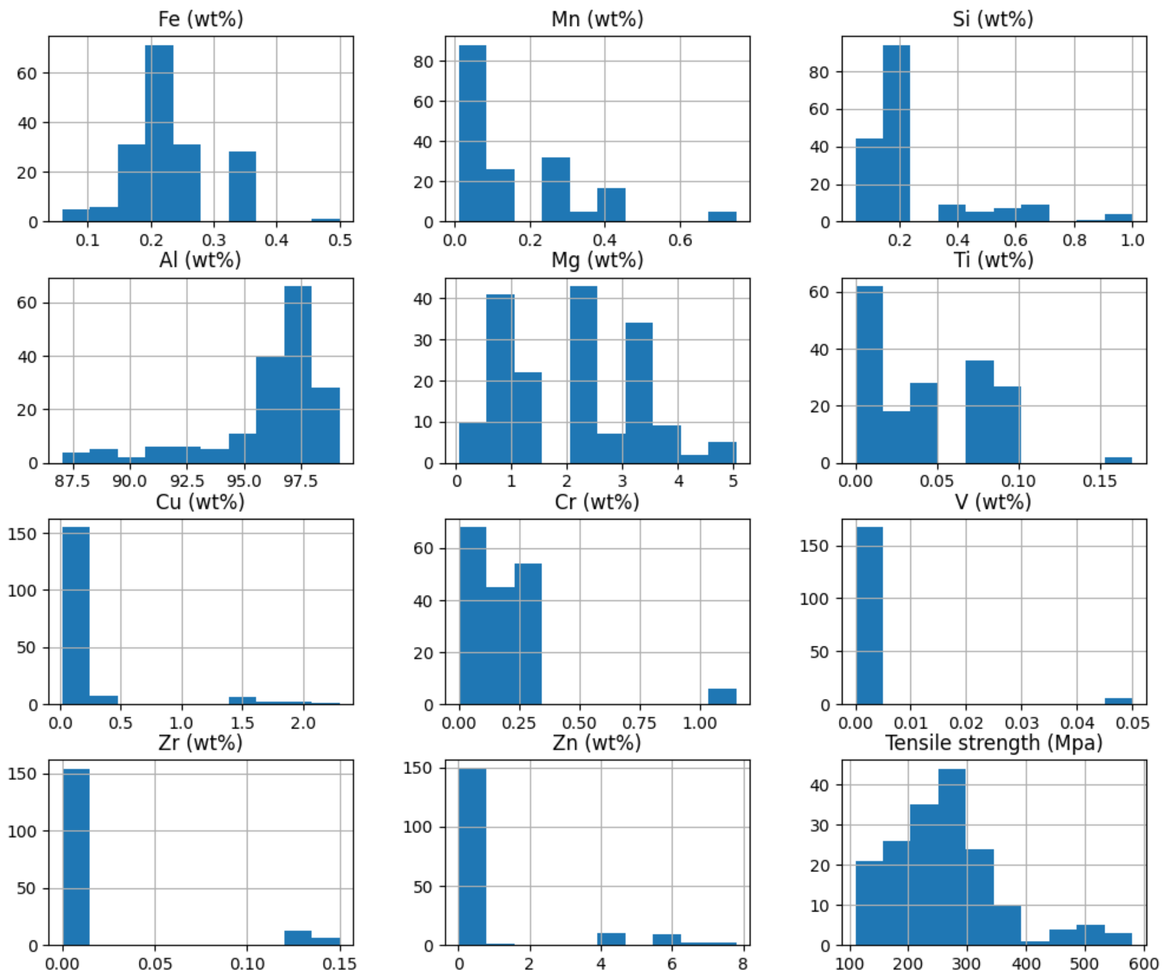
Histograms

In [14]:

```python
df.hist(figsize=(12,10))
```

Out[14]:

```
array([[<Axes: title={'center': 'Fe (wt%)'}>,
        <Axes: title={'center': 'Mn (wt%)'}>,
        <Axes: title={'center': 'Si (wt%)'}>],
       [<Axes: title={'center': 'Al (wt%)'}>,
        <Axes: title={'center': 'Mg (wt%)'}>,
        <Axes: title={'center': 'Ti (wt%)'}>],
       [<Axes: title={'center': 'Cu (wt%)'}>,
        <Axes: title={'center': 'Cr (wt%)'}>,
        <Axes: title={'center': 'V (wt%)'}>],
       [<Axes: title={'center': 'Zr (wt%)'}>,
        <Axes: title={'center': 'Zn (wt%)'}>,
        <Axes: title={'center': 'Tensile strength (Mpa)'}>]], dtype=object)
```



From this we can tell that there are a lot of columns with outliers and none of the columns have a Normal distribution.

Boxplots

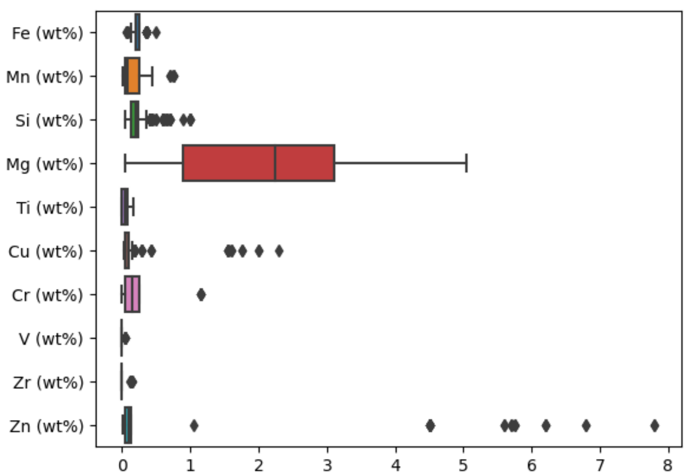
In [15]:
```
df.head()
```

Out[15]:

	Fe (wt%)	Mn (wt%)	Si (wt%)	Al (wt%)	Mg (wt%)	Ti (wt%)	Cu (wt%)	Cr (wt%)	V (wt%)	Zr (wt%)	Zn (wt%)	Tensile strength (Mpa)
0	0.35	0.1	0.15	98.33	0.8	0.0	0.1	0.05	0.0	0.0	0.125	125
1	0.35	0.1	0.15	98.33	0.8	0.0	0.1	0.05	0.0	0.0	0.125	145
2	0.35	0.1	0.15	98.33	0.8	0.0	0.1	0.05	0.0	0.0	0.125	165
3	0.35	0.1	0.15	98.33	0.8	0.0	0.1	0.05	0.0	0.0	0.125	185
4	0.35	0.1	0.15	98.33	0.8	0.0	0.1	0.05	0.0	0.0	0.125	120

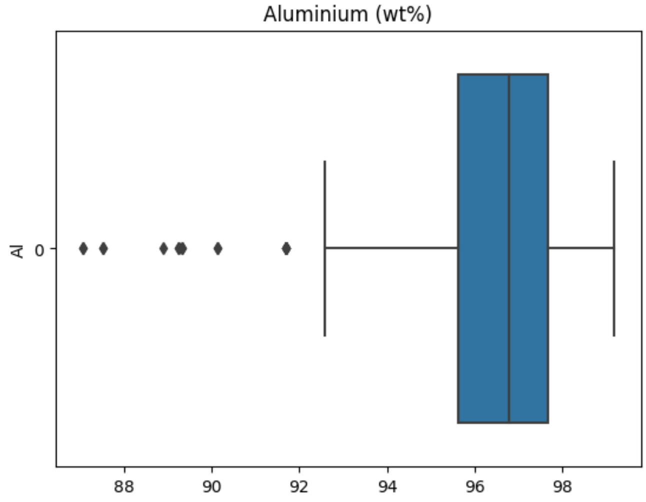
In [16]:
```
sns.boxplot(data=df.drop(['Tensile strength (Mpa)','Al (wt%)'],axis=1),orient='h')
```

Out[16]: <Axes: >



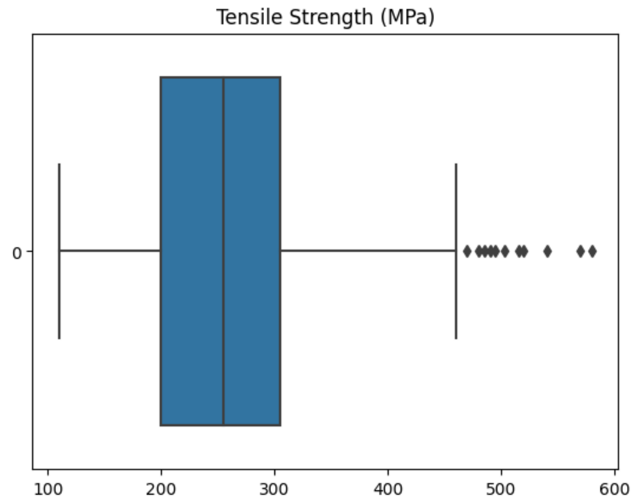
In [17]:
```
sns.boxplot(data=df['Al (wt%)'],orient='h')
plt.title('Aluminium (wt%)')
plt.ylabel('Al')
```

Out[17]: Text(0, 0.5, 'Al')



In [18]:
```
sns.boxplot(data=df['Tensile strength (Mpa)'],orient='h')
plt.title('Tensile Strength (MPa)')
```

Out[18]: Text(0.5, 1.0, 'Tensile Strength (MPa)')



Detecting outliers

In [19]:
```
def outliers(data,feature):
    q1 = data[feature].quantile(0.25)
    q3 = data[feature].quantile(0.75)

    IQR = q3-q1

    lower_bound = q1-1.5*IQR
    upper_bound = q3+1.5*IQR

    outlier_list = df.index[(data[feature]<lower_bound)|(data[feature]>upper_bound)]

    return outlier_list
```

In [20]:
```
df.columns
```

Out[20]:
```
Index(['Fe (wt%)', 'Mn (wt%)', 'Si (wt%)', 'Al (wt%)', 'Mg (wt%)', 'Ti (wt%)',
       'Cu (wt%)', 'Cr (wt%)', 'V (wt%)', 'Zr (wt%)', 'Zn (wt%)',
       'Tensile strength (Mpa)'],
      dtype='object')
```

In [21]:
```
out_index_list = []

cols = df.columns

for i in cols:
    for j in i:
        arr = outliers(df,i)
        out_index_list.extend(arr)
```

In [22]:
```
len(out_index_list)
```

Out[22]: 1588

In [23]:

```python
def unique_outlier_list(duplis):
    noduplist = []
    for element in duplis:
        if element not in noduplist:
            noduplist.append(element)

    return noduplist
```

In [24]:

```python
len(unique_outlier_list(out_index_list))
```

Out[24]:

```
81
```

In [25]:

```python
out_lis = unique_outlier_list(out_index_list)
```

In [26]:

```python
df.shape
```

Out[26]:

```
(173, 12)
```

In [27]:

```python
out_lis.sort()
```

In [28]:

```python
df.shape
```

Out[28]:

```
(173, 12)
```

In [29]:

```python
new_df = df.drop(index=out_lis)
```

In [30]:

```python
new_df.shape
```

Out[30]:

```
(92, 12)
```

In [31]:

```python
new_df.head()
```

Out[31]:

	Fe (wt%)	Mn (wt%)	Si (wt%)	Al (wt%)	Mg (wt%)	Ti (wt%)	Cu (wt%)	Cr (wt%)	V (wt%)	Zr (wt%)	Zn (wt%)	Tensile strength (Mpa)
8	0.18	0.35	0.10	95.58	3.5	0.05	0.08	0.05	0.0	0.0	0.125	240
9	0.18	0.35	0.10	95.58	3.5	0.05	0.08	0.05	0.0	0.0	0.125	330
10	0.18	0.35	0.10	95.58	3.5	0.05	0.08	0.05	0.0	0.0	0.125	350
11	0.18	0.35	0.10	95.58	3.5	0.05	0.08	0.05	0.0	0.0	0.125	280
24	0.20	0.05	0.13	96.78	2.5	0.00	0.05	0.25	0.0	0.0	0.050	175

In [32]:

```python
new_df.describe()
```

Out[32]:

	Fe (wt%)	Mn (wt%)	Si (wt%)	Al (wt%)	Mg (wt%)	Ti (wt%)	Cu (wt%)	Cr (wt%)	V (wt%)	Zr (wt%)	Zn (wt%)	Tensile strength (Mpa)
count	92.000000	92.000000	92.000000	92.000000	92.000000	92.000000	92.000000	92.000000	92.0	92.0	92.000000	92.000000
mean	0.209130	0.151196	0.168261	96.160978	2.972283	0.044239	0.052826	0.174891	0.0	0.0	0.079511	257.282609
std	0.030836	0.150551	0.046139	1.151676	1.041194	0.039786	0.022401	0.085401	0.0	0.0	0.032157	55.842267
min	0.130000	0.010000	0.080000	94.250000	0.400000	0.000000	0.020000	0.000000	0.0	0.0	0.015000	110.000000
25%	0.200000	0.040000	0.130000	95.630000	2.500000	0.000000	0.045000	0.080000	0.0	0.0	0.050000	230.000000
50%	0.200000	0.050000	0.200000	95.880000	3.100000	0.040000	0.050000	0.250000	0.0	0.0	0.075000	255.000000
75%	0.230000	0.300000	0.200000	96.780000	3.500000	0.080000	0.080000	0.250000	0.0	0.0	0.100000	295.000000
max	0.250000	0.450000	0.230000	99.190000	5.050000	0.100000	0.100000	0.250000	0.0	0.0	0.125000	380.000000

In [33]:

```python
new_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 92 entries, 8 to 114
Data columns (total 12 columns):
 #   Column                  Non-Null Count  Dtype
---  ------                  --------------  -----
 0   Fe (wt%)                92 non-null     float64
 1   Mn (wt%)                92 non-null     float64
 2   Si (wt%)                92 non-null     float64
 3   Al (wt%)                92 non-null     float64
 4   Mg (wt%)                92 non-null     float64
 5   Ti (wt%)                92 non-null     float64
 6   Cu (wt%)                92 non-null     float64
 7   Cr (wt%)                92 non-null     float64
 8   V (wt%)                 92 non-null     float64
 9   Zr (wt%)                92 non-null     float64
 10  Zn (wt%)                92 non-null     float64
 11  Tensile strength (Mpa)  92 non-null     int64
dtypes: float64(11), int64(1)
memory usage: 9.3 KB
```

In [34]:

```python
df_final = pd.DataFrame(np.random.permutation(new_df),columns=new_df.columns)
```

In [35]:

```python
df_final.head()
```

Out[35]:

	Fe (wt%)	Mn (wt%)	Si (wt%)	Al (wt%)	Mg (wt%)	Ti (wt%)	Cu (wt%)	Cr (wt%)	V (wt%)	Zr (wt%)	Zn (wt%)	Tensile strength (Mpa)
0	0.23	0.01	0.23	95.65	3.50	0.03	0.03	0.25	0.0	0.0	0.100	255.0
1	0.25	0.30	0.20	96.90	2.05	0.08	0.08	0.08	0.0	0.0	0.075	180.0
2	0.25	0.45	0.20	94.70	4.00	0.08	0.05	0.15	0.0	0.0	0.125	325.0
3	0.20	0.05	0.13	96.78	2.50	0.00	0.05	0.25	0.0	0.0	0.050	235.0
4	0.13	0.10	0.08	99.19	0.40	0.00	0.10	0.00	0.0	0.0	0.015	140.0

In [36]:
```
df_final.shape
```
Out[36]:
```
(92, 12)
```

In [37]:
```
scaler = MinMaxScaler()

final_df = pd.DataFrame(scaler.fit_transform(df_final),columns=df_final.columns)
```

In [38]:
```
X = final_df.drop('Tensile strength (Mpa)',axis=1)
y = final_df['Tensile strength (Mpa)']
```

In [39]:
```
X.shape, y.shape
```
Out[39]:
```
((92, 11), (92,))
```

In [40]:
```
x_train,x_test,y_train,y_test = train_test_split(X,y,test_size=0.1,random_state=42)
x_train.shape,y_train.shape,x_test.shape,y_test.shape
```
Out[40]:
```
((82, 11), (82,), (10, 11), (10,))
```

Defining Baseline metrics

In [41]:
```
y_mean = final_df['Tensile strength (Mpa)'].mean()
y_mae = [y_mean]*len(df_final)
```

In [42]:
```
baseline_mae = mean_absolute_error(final_df['Tensile strength (Mpa)'],y_mae)
baseline_mae
```
Out[42]:
```
0.1596040747742071
```

In [43]:
```
final_df.describe()
```
Out[43]:

	Fe (wt%)	Mn (wt%)	Si (wt%)	Al (wt%)	Mg (wt%)	Ti (wt%)	Cu (wt%)	Cr (wt%)	V (wt%)	Zr (wt%)	Zn (wt%)	Tensile strength (Mpa)
count	92.000000	92.000000	92.000000	92.000000	92.000000	92.000000	92.000000	92.000000	92.0	92.0	92.000000	92.000000
mean	0.659420	0.320899	0.588406	0.386838	0.553179	0.442391	0.410326	0.699565	0.0	0.0	0.586462	0.545491
std	0.256970	0.342161	0.307594	0.233133	0.223913	0.397861	0.280016	0.341605	0.0	0.0	0.292340	0.206823
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.0	0.0	0.000000	0.000000
25%	0.583333	0.068182	0.333333	0.279352	0.451613	0.000000	0.312500	0.320000	0.0	0.0	0.318182	0.444444
50%	0.583333	0.090909	0.800000	0.329960	0.580645	0.400000	0.375000	1.000000	0.0	0.0	0.545455	0.537037
75%	0.833333	0.659091	0.800000	0.512146	0.666667	0.800000	0.750000	1.000000	0.0	0.0	0.772727	0.685185
max	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	0.0	0.0	1.000000	1.000000

Model 1: Random Forest

Setting Parameters

In [44]:
```
params = {
    "n_estimators": range(450,1000,100),
    "max_depth": range(20,61,5),
    "criterion": ["squared_error", "absolute_error"],
    "min_samples_split": [2,4],
    "min_samples_leaf": [1,2,4]
}
```

Model Building

In [45]:
```
model_rf = RandomizedSearchCV(
    RandomForestRegressor(random_state=42),
    params,
    cv=5,
    n_jobs=-1,
    n_iter=35,
    scoring=["neg_mean_absolute_error", "r2"],
    refit="neg_mean_absolute_error",
    verbose=1
)
```

Model Fitting

In [46]:
```
model_rf.fit(x_train, y_train)
```
```
Fitting 5 folds for each of 35 candidates, totalling 175 fits
```

Out[46]:

```
▸ RandomizedSearchCV
▸ estimator: RandomForestRegressor
  ▸ RandomForestRegressor
```

Gettting results

In [47]:

```
cv_results = pd.DataFrame(model_rf.cv_results_)
cv_results.sort_values("rank_test_neg_mean_absolute_error").head()
```

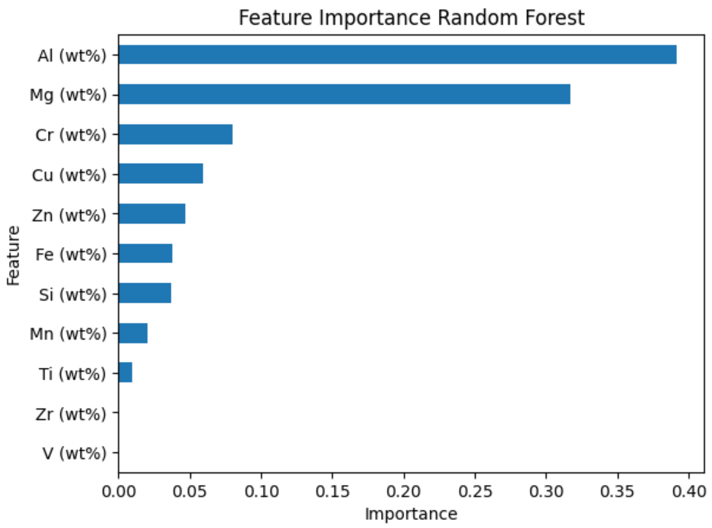
Out[47]:

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_n_estimators	param_min_samples_split	param_min_samples_leaf	param_ma
11	1.146316	0.033647	0.073919	0.012078	450	4	1	
13	1.330234	0.195825	0.060930	0.000874	450	2	1	
6	1.732617	0.042191	0.099637	0.004304	650	4	1	
27	1.695145	0.062812	0.101138	0.011278	650	4	1	
1	1.833172	0.020518	0.102227	0.006295	750	2	1	

5 rows × 26 columns

In [48]:

```
# Get feature names from training data
features = x_train.columns
# Extract importances from model
importances = model_rf.best_estimator_.feature_importances_
# Create a series with feature names and importances
feat_imp = pd.Series(importances, index=features)
# Plot 10 most important features
feat_imp.sort_values().plot(kind="barh")
plt.xlabel("Importance")
plt.ylabel("Feature")
plt.title("Feature Importance Random Forest");
```



In [49]:

```
mae_rf_train = mean_absolute_error(y_train, model_rf.predict(x_train))
mae_rf_test = mean_absolute_error(y_test, model_rf.predict(x_test))
mse_rf_train = mean_squared_error(y_train, model_rf.predict(x_train))
mse_rf_test = mean_squared_error(y_test, model_rf.predict(x_test))
lmse_rf_train = mean_squared_log_error(y_train, model_rf.predict(x_train))
lmse_rf_test = mean_squared_log_error(y_test, model_rf.predict(x_test))

print("Random Forest:")
print("Training Mean Absolute Error:", round(mae_rf_train, 4))
print("Test Mean Absolute Error:", round(mae_rf_test, 4))
print("Training Root Mean Squared Errror:", np.sqrt(mse_rf_train))
print("Testing Root Mean Squared Errror:", np.sqrt(mse_rf_test))
print("Training Mean Squared Log Errror:", round(lmse_rf_train,4))
print("Testing Mean Squared Log Errror:", round(lmse_rf_test,4))
print("Baseline Mean Absolute Error:", round(baseline_mae, 4))
```

```
Random Forest:
Training Mean Absolute Error: 0.0893
Test Mean Absolute Error: 0.1392
Training Root Mean Squared Errror: 0.10830064669797645
Testing Root Mean Squared Errror: 0.16630918389976498
Training Mean Squared Log Errror: 0.005
Testing Mean Squared Log Errror: 0.0127
Baseline Mean Absolute Error: 0.1596
```

In [50]:

```
r2_rf_train = r2_score(y_train, model_rf.predict(x_train))
r2_rf_test = r2_score(y_test, model_rf.predict(x_test))

print("Random Forest:")
print("Training R2:", round(r2_rf_train, 4))
print("Test R2:", round(r2_rf_test, 4))
```

```
Random Forest:
Training R2: 0.7319
Test R2: -0.5567
```

Model 2: Bagging Regressor

In [51]:

```
params_br = {
    "n_estimators": range(5,50,5)
}
```

In [52]:

```
model_br = GridSearchCV(
    BaggingRegressor(random_state=42),
    params_br,
    cv=5,
    n_jobs=-1,
    scoring=["neg_mean_absolute_error", "r2"],
    refit="neg_mean_absolute_error",
    verbose=1
)
```

In [53]:

```
model_br.fit(x_train, y_train)
```

```
Fitting 5 folds for each of 9 candidates, totalling 45 fits
```

Out[53]:

```
▸ GridSearchCV
▸ estimator: BaggingRegressor
  ▸ BaggingRegressor
```

In [54]:

```
cv_results = pd.DataFrame(model_br.cv_results_)
cv_results.sort_values("rank_test_neg_mean_absolute_error").head()
```

Out[54]:

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_n_estimators	params	split0_test_neg_mean_absolute_error	split1_test_
3	0.061065	0.000660	0.006609	0.000313	20	{'n_estimators': 20}	-0.102461	
8	0.120793	0.022037	0.009946	0.004038	45	{'n_estimators': 45}	-0.103147	
2	0.050575	0.004429	0.006348	0.000853	15	{'n_estimators': 15}	-0.101872	
1	0.046165	0.006989	0.005340	0.000582	10	{'n_estimators': 10}	-0.102988	
7	0.130474	0.008899	0.010826	0.002083	40	{'n_estimators': 40}	-0.103006	

5 rows × 22 columns

In [55]:

```
mae_br_train = mean_absolute_error(y_train, model_br.predict(x_train))
mae_br_test = mean_absolute_error(y_test, model_br.predict(x_test))

mse_br_train = mean_squared_error(y_train, model_br.predict(x_train))
mse_br_test = mean_squared_error(y_test, model_br.predict(x_test))

lmse_br_train = mean_squared_log_error(y_train, model_br.predict(x_train))
lmse_br_test = mean_squared_log_error(y_test, model_br.predict(x_test))


print("Bagging Regressor:")
print("Training Mean Absolute Error:", round(mae_br_train, 4))
print("Test Mean Absolute Error:", round(mae_br_test, 4))

print("Training Root Mean Squared Errror:", np.sqrt(mse_br_train))
print("Testing Root Mean Squared Errror:", np.sqrt(mse_br_test))

print("Training Mean Squared Log Errror:", round(lmse_br_train,4))
print("Testing Mean Squared Log Errror:", round(lmse_br_test,4))

print("Baseline Mean Absolute Error:", round(baseline_mae, 4))
```

```
Bagging Regressor:
Training Mean Absolute Error: 0.0893
Test Mean Absolute Error: 0.1394
Training Root Mean Squared Errror: 0.10859593849089692
Testing Root Mean Squared Errror: 0.16327267099342815
Training Mean Squared Log Errror: 0.005
Testing Mean Squared Log Errror: 0.0123
Baseline Mean Absolute Error: 0.1596
```

In [56]:
```
r2_br_train = r2_score(y_train, model_br.predict(x_train))
r2_br_test = r2_score(y_test, model_br.predict(x_test))

print("Bagging Regressor:")
print("Training R2:", round(r2_br_train, 4))
print("Test R2:", round(r2_br_test, 4))
```

```
Bagging Regressor:
Training R2: 0.7304
Test R2: -0.5004
```

Model 3: Linear Regressor

In [57]:
```
params_lr = {
    'fit_intercept': [True, False],
}
```

In [58]:
```
model_lr = GridSearchCV(
    LinearRegression(),
    params_lr,
    cv=5
)
```

In [59]:
```
model_lr.fit(x_train, y_train)
```

Out[59]:
```
▸ GridSearchCV
▸ estimator: LinearRegression
  ▸ LinearRegression
```

In [60]:
```
cv_results = pd.DataFrame(model_lr.cv_results_)
```

In [61]:
```
cv_results.head()
```

Out[61]:

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_fit_intercept	params	split0_test_score	split1_test_score	split2_test_sc
0	0.007109	0.002166	0.002843	0.001142	True	{'fit_intercept': True}	0.696472	0.612375	0.624
1	0.004045	0.002364	0.002542	0.001176	False	{'fit_intercept': False}	0.698448	0.612336	0.624

In [62]:
```
mae_lr_train = mean_absolute_error(y_train, model_lr.predict(x_train))
mae_lr_test = mean_absolute_error(y_test, model_lr.predict(x_test))

mse_lr_train = mean_squared_error(y_train, model_lr.predict(x_train))
mse_lr_test = mean_squared_error(y_test, model_lr.predict(x_test))

lmse_lr_train = mean_squared_log_error(y_train, model_lr.predict(x_train))
lmse_lr_test = mean_squared_log_error(y_test, model_lr.predict(x_test))


print("Linear Regressor:")
print("Training Mean Absolute Error:", round(mae_lr_train, 4))
print("Test Mean Absolute Error:", round(mae_lr_test, 4))

print("Training Root Mean Squared Errror:", np.sqrt(mse_lr_train))
print("Testing Root Mean Squared Errror:", np.sqrt(mse_lr_test))

print("Training Mean Squared Log Errror:", round(lmse_lr_train,4))
print("Testing Mean Squared Log Errror:", round(lmse_lr_test,4))

print("Baseline Mean Absolute Error:", round(baseline_mae, 4))
```

```
Linear Regressor:
Training Mean Absolute Error: 0.092
Test Mean Absolute Error: 0.1329
Training Root Mean Squared Errror: 0.11121386326773937
Testing Root Mean Squared Errror: 0.15503144787204423
Training Mean Squared Log Errror: 0.0052
Testing Mean Squared Log Errror: 0.0111
Baseline Mean Absolute Error: 0.1596
```

In [63]:
```
r2_lr_train = r2_score(y_train, model_lr.predict(x_train))
r2_lr_test = r2_score(y_test, model_lr.predict(x_test))

print("Linear Regressor:")
print("Training R2:", round(r2_lr_train, 4))
print("Test R2:", round(r2_lr_test, 4))
```

```
Linear Regressor:
Training R2: 0.7173
Test R2: -0.3527
```

Getting best parameters and scores after parameters optimization

In [64]:
```python
print("RANDOM FOREST:")
print("Best Parameters: ", model_rf.best_params_)
print("Best Score: ", model_rf.best_score_)

print("BAGGING REGRESSOR:")
print("Best Parameters: ", model_br.best_params_)
print("Best Score: ", model_br.best_score_)

print("LINEAR REGRESSOR:")
print("Best Parameters: ", model_lr.best_params_)
print("Best Score: ", model_lr.best_score_)
```

```
RANDOM FOREST:
Best Parameters:  {'n_estimators': 450, 'min_samples_split': 4, 'min_samples_leaf': 1, 'max_depth': 45, 'criterion': 'squared_error'}
Best Score:  -0.10195666418063265
BAGGING REGRESSOR:
Best Parameters:  {'n_estimators': 20}
Best Score:  -0.10300185651331994
LINEAR REGRESSOR:
Best Parameters:  {'fit_intercept': False}
Best Score:  0.6506888939650379
```

Conclusions

Mean Absolute Error Comparision

In [65]:
```python
fig = px.bar(y=["Random Forest", "Bagging", "Linear", "Baseline"],
             x=[mae_rf_test, mae_br_test, mae_lr_test, baseline_mae],
             color=["Random Forest", "Bagging", "Linear", "Baseline"],
             color_discrete_map={ "Random Forest": "green",
                                  "Bagging": "blue",
                                  "Linear": "grey",
                                  "Baseline": "pink"
                                },
            title="Mean Absolute Error comparison (lower is better)")
fig.update_layout(yaxis={'categoryorder':'total descending'}, xaxis_title="MAE", yaxis_title="Models")
fig.show()
```

R2 Score Comparision

In [66]:
```python
fig = px.bar(y=["Random Forest", "Bagging", "Linear"],
             x=[r2_rf_test, r2_br_test, r2_lr_test],
             color=["Random Forest", "Bagging", "Linear"],
             color_discrete_map={ "Random Forest": "green",
                                  "Bagging": "blue",
                                  "Linear": "grey",
                                },
            title="R2-Score comparison (higher is better)")
fig.update_layout(yaxis={'categoryorder':'total ascending'}, xaxis_title="R2-score", yaxis_title="Models")
fig.show()
```