Mechanical properties of low alloy steels


```
Name: Naman Tamrakar
Roll no.: 193152
Department: Mechanical Engineering
```


Summary

The mechanical property of steel is decided by doing mutiple test on it using different methods. Since the mechanical property of alloy steel depends on elements composition, temprature and dimension change.

So using a machine learning one can easily predict the mechanical property using these paramaters. The aim of this machine learning model is to predict the mechanical property of Steel Alloy by using element composition, temprature, tensile strengh, yeild strength and elogation.

As this problem is a **classification problem** so this model can be trained using **Logistic Classification**, **Support vector classification**, **Randome forest classification** and **Decision Tree classification**.

Dataset

The dataset contains columns having various elements composition, temprature, mechanical property of Stell alloy. The **Alloy code** is a string unique to each alloy. The model is expected to predict the alloy code based on above paramaters.

> Source: https://www.kaggle.com/datasets/rohannemade/mechanical-properties-of-low-alloy-steels

Data Preprocessing

Since **Alloy code** is string value so it is converted to numerical lable using **Label Encoding**.

It is observed that the performance of model is very bad when train without **standarizing dataset** i.e. 0.9% So standarize the data by removing mean and scaling to unit variance using **StandardScaler**.

Model Training

1. Logisitic Regression

When trained model with 80% training data and tested with 20% data, An accuracy of **98.4%** is achived which can further improved by changing some paramters like *regularization strength*.

2. Support Vector Classification

When train using default kernel i.e, `rbf`, It gives an accuracy of *40%*. To improve the accuracy try with different kernels.

The `linear` kernel found to gives better accuracy of **99%** which can futher improved by changing *regularization strength (C)*.

Conclusion

So one can easily predict the mechanical property of the steel alloy by using this model very accurately.

Code

In [ ]:
```
import pandas as pd # for data handling

from sklearn.model_selection import train_test_split # for splitting data
from sklearn.svm import SVC # SVM classifier
from sklearn.linear_model import LogisticRegression # logisitic classifier
from sklearn.preprocessing import LabelEncoder, StandardScaler
from sklearn.pipeline import Pipeline # creating pipeling
from sklearn.metrics import confusion_matrix

import matplotlib.pyplot as plt # for plotting data
import seaborn as sns
```

In [ ]:
```
# Importing data from CSV file
# which contains nearly 900+ data points

df = pd.read_csv("steel-alloy.csv")
df.head()
```

Out[ ]:

	Alloy code	C	Si	Mn	P	S	Ni	Cr	Mo	Cu	V	Al	N	Ceq	Nb + Ta	Tempe
0	MBB	0.12	0.36	0.52	0.009	0.003	0.089	0.97	0.61	0.04	0.0	0.003	0.0066	0.0	0.0	
1	MBB	0.12	0.36	0.52	0.009	0.003	0.089	0.97	0.61	0.04	0.0	0.003	0.0066	0.0	0.0	
2	MBB	0.12	0.36	0.52	0.009	0.003	0.089	0.97	0.61	0.04	0.0	0.003	0.0066	0.0	0.0	
3	MBB	0.12	0.36	0.52	0.009	0.003	0.089	0.97	0.61	0.04	0.0	0.003	0.0066	0.0	0.0	
4	MBB	0.12	0.36	0.52	0.009	0.003	0.089	0.97	0.61	0.04	0.0	0.003	0.0066	0.0	0.0	

Data analysing

In [ ]:
```
df.describe()
```

Out[ ]:

	C	Si	Mn	P	S	Ni	Cr	Mo
count	915.000000	915.000000	915.000000	915.000000	915.000000	915.000000	915.000000	915.00000
mean	0.174929	0.310918	0.812962	0.014543	0.010602	0.143016	0.427861	0.44287
std	0.059674	0.086871	0.342775	0.005244	0.004024	0.172746	0.457568	0.39438
min	0.090000	0.180000	0.420000	0.006000	0.003000	0.000000	0.000000	0.00500
25%	0.130000	0.240000	0.500000	0.010000	0.008000	0.023000	0.040000	0.05000
50%	0.160000	0.300000	0.680000	0.014000	0.010000	0.050000	0.110000	0.50000
75%	0.200000	0.370000	1.210000	0.018000	0.012000	0.210000	1.000000	0.56000
max	0.340000	0.520000	1.480000	0.030000	0.022000	0.600000	1.310000	1.35000

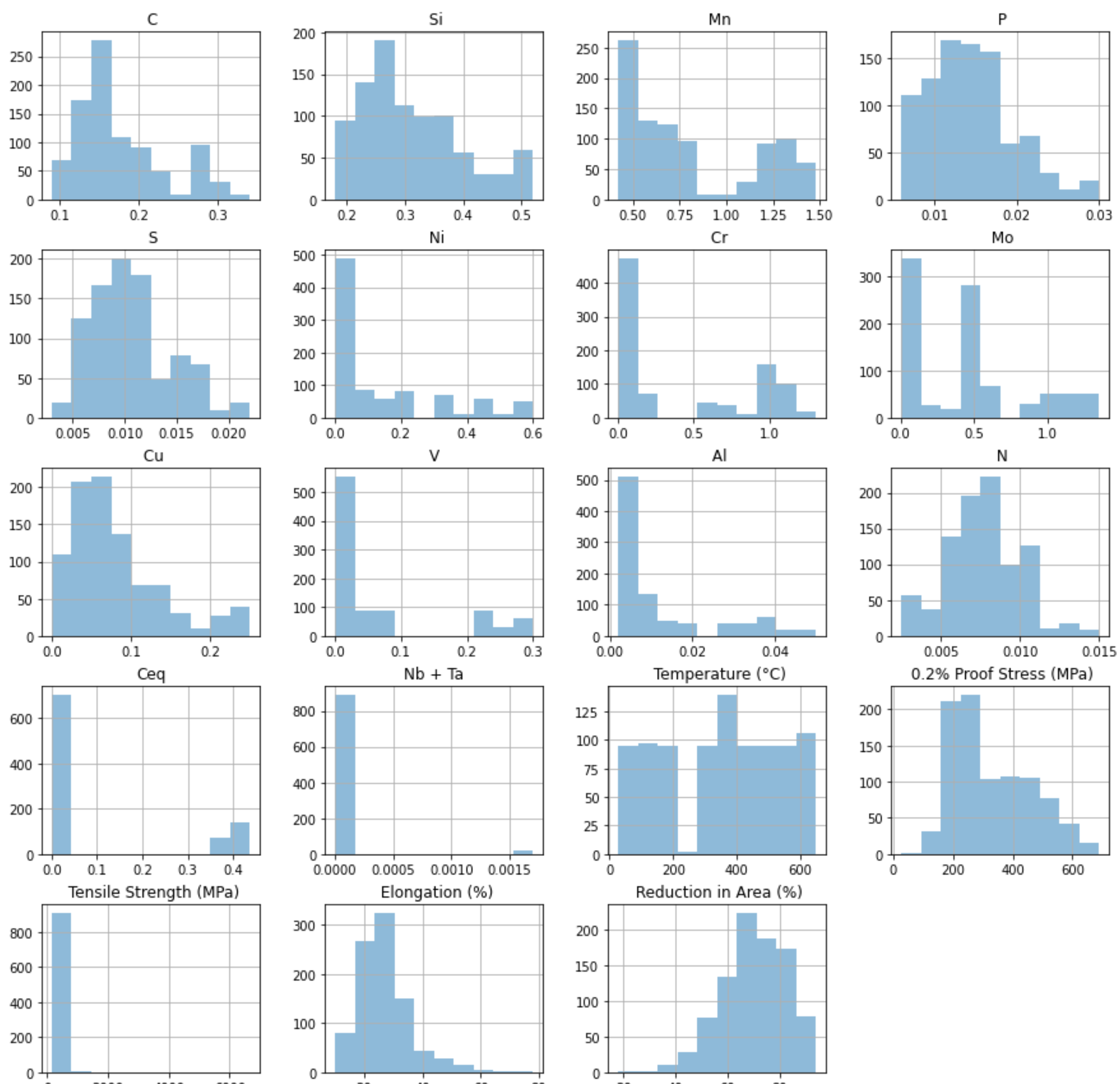
In [ ]:
```
# check if there is any data missing row
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 915 entries, 0 to 914
Data columns (total 20 columns):
 #   Column                    Non-Null Count  Dtype
---  ------                    --------------  -----
 0   Alloy code                915 non-null    object
 1    C                        915 non-null    float64
 2    Si                       915 non-null    float64
 3    Mn                       915 non-null    float64
 4    P                        915 non-null    float64
 5    S                        915 non-null    float64
 6    Ni                       915 non-null    float64
 7    Cr                       915 non-null    float64
 8    Mo                       915 non-null    float64
 9    Cu                       915 non-null    float64
 10  V                         915 non-null    float64
 11   Al                       915 non-null    float64
 12   N                        915 non-null    float64
 13  Ceq                       915 non-null    float64
 14  Nb + Ta                   915 non-null    float64
 15   Temperature (°C)         915 non-null    int64
 16   0.2% Proof Stress (MPa)  915 non-null    int64
 17   Tensile Strength (MPa)   915 non-null    int64
 18   Elongation (%)           915 non-null    int64
 19   Reduction in Area (%)    915 non-null    int64
dtypes: float64(14), int64(5), object(1)
memory usage: 143.1+ KB
```

In [ ]:

```
# Distribution of various data

df.hist(alpha=0.5, figsize=(15, 15))
plt.show()
```

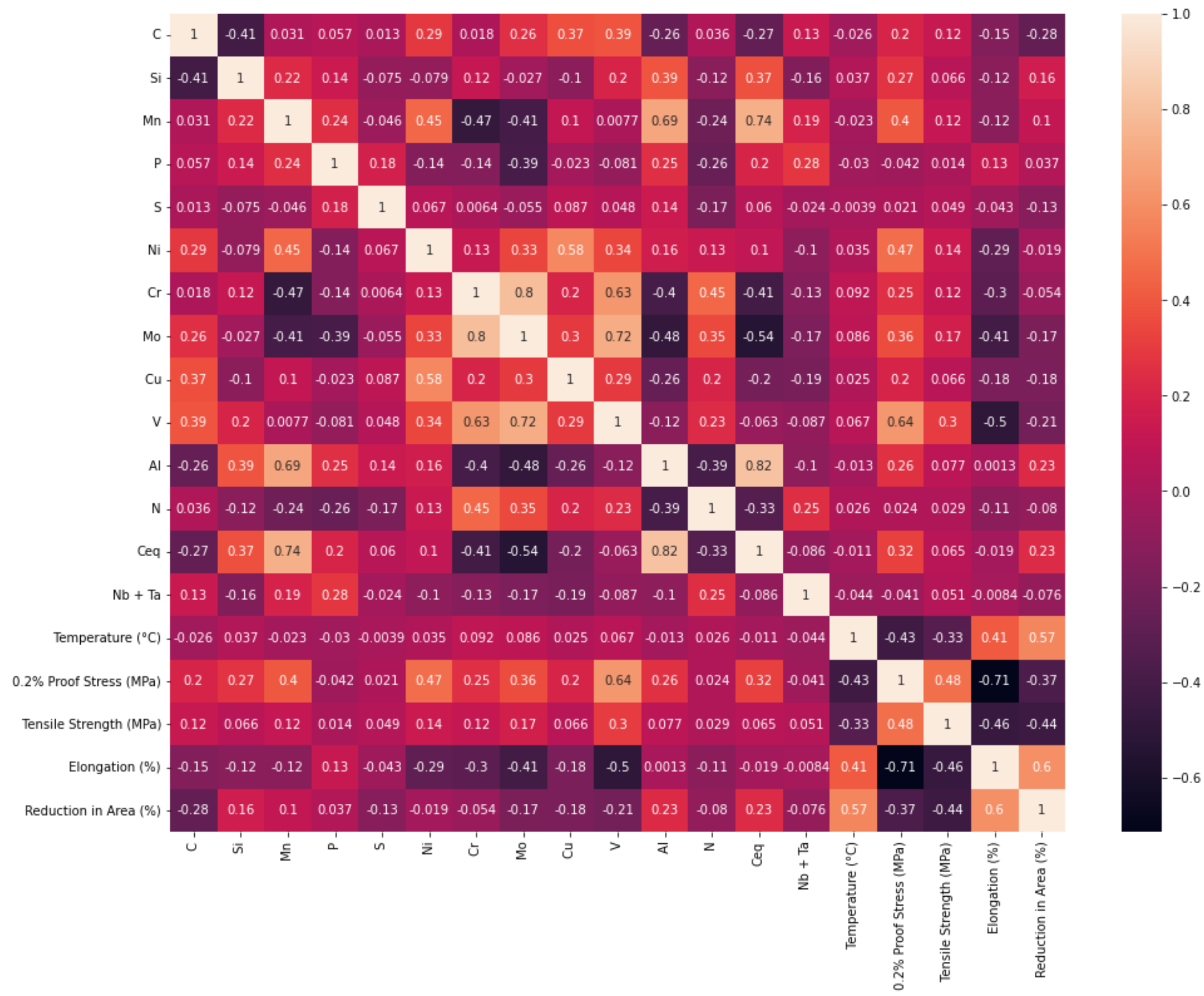


In [ ]:

```
# Let's plot a heatmap to get the idea of correlation
# b/w mutiple variables

plt.figure(figsize=(16, 12))
sns.heatmap(df.corr(), annot=True)
```

Out[ ]: <AxesSubplot:>



In [ ]:

In [ ]:

```
# check no of alloy code available

df["Alloy code"].describe()
```

Out[ ]:

```
count     915
unique     95
top       CCB
freq       11
Name: Alloy code, dtype: object
```

In [ ]:

```
# Label encode Alloy column
encoder = LabelEncoder()
df["Alloy code"] = encoder.fit_transform(df["Alloy code"])

df.head()
```

Out[ ]:

	Alloy code	C	Si	Mn	P	S	Ni	Cr	Mo	Cu	V	Al	N	Ceq	Nb + Ta	Tempe
0	57	0.12	0.36	0.52	0.009	0.003	0.089	0.97	0.61	0.04	0.0	0.003	0.0066	0.0	0.0	
1	57	0.12	0.36	0.52	0.009	0.003	0.089	0.97	0.61	0.04	0.0	0.003	0.0066	0.0	0.0	
2	57	0.12	0.36	0.52	0.009	0.003	0.089	0.97	0.61	0.04	0.0	0.003	0.0066	0.0	0.0	
3	57	0.12	0.36	0.52	0.009	0.003	0.089	0.97	0.61	0.04	0.0	0.003	0.0066	0.0	0.0	
4	57	0.12	0.36	0.52	0.009	0.003	0.089	0.97	0.61	0.04	0.0	0.003	0.0066	0.0	0.0	

In [ ]:

```
X = df.drop(columns=["Alloy code"])
y = df["Alloy code"]

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

Model training using Logistic Regression

In [ ]:

```python
# Check model accuracy without standarizing data

# create model pipeline with standarizing data
log_model = Pipeline([
    ("logisitic", LogisticRegression())
])

# Train model using training data
log_model.fit(X_train, y_train)

# Check accuracy of model
log_model.score(X_test, y_test)
```

```
c:\Users\Naman Tamrakar\Desktop\ML-CCPD\.venv\lib\site-packages\sklearn\linear_model
\_logistic.py:818: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  extra_warning_msg=_LOGISTIC_SOLVER_CONVERGENCE_MSG,
```

Out[ ]:

```
0.00546448087431694
```

In [ ]:

```python
# create model pipeline with standarizing data
log_model = Pipeline([
    ("scaler", StandardScaler()), # standarising the data
    ("logisitic", LogisticRegression(max_iter=3000, C=10)) # twiking param to get bett
])

# Train model using training data
log_model.fit(X_train, y_train)

# Check accuracy of model
log_model.score(X_test, y_test)
```

Out[ ]:

```
0.9836065573770492
```

In [ ]:

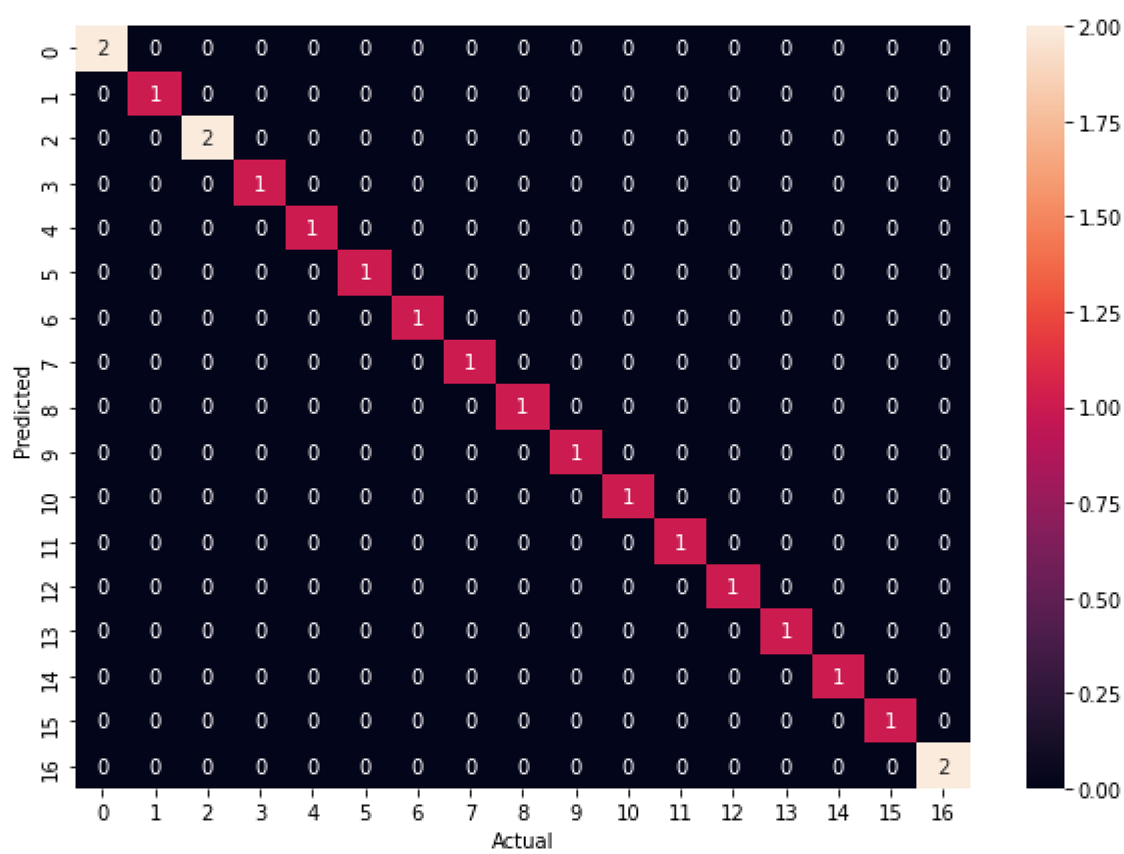
```python
y_pred = log_model.predict(X_test)

# plot only initial 20 results
matrix = confusion_matrix(y_test[:20], y_pred[:20])

plt.figure(figsize=(10, 7))
sns.heatmap(matrix, annot=True)
plt.xlabel("Actual")
plt.ylabel("Predicted")
```

Out[ ]:

```
Text(69.0, 0.5, 'Predicted')
```



Model training using Support vector maching classification

In [ ]:
```
# Check model accuracy with default kernel

# create model pipeline with standarizing data
svc_model = Pipeline([
    ("scaler", StandardScaler()),
    ("svc", SVC())
])

# fitting model using training data
svc_model.fit(X_train, y_train)

# Check model accuracy
svc_model.score(X_test, y_test)
```

Out[ ]:
```
0.3989071038251366
```

In [ ]:
```
# create model pipeline with standarizing data
svc_model = Pipeline([
    ("scaler", StandardScaler()),
    ("svc", SVC(kernel="linear"))
])

# fitting model using training data
svc_model.fit(X_train, y_train)

# Check model accuracy
svc_model.score(X_test, y_test)
```

Out[ ]:
```
0.994535519125683
```

In [ ]:
```
y_pred = svc_model.predict(X_test)

matrix = confusion_matrix(y_test[:20], y_pred[:20])

plt.figure(figsize=(10, 7))
sns.heatmap(matrix, annot=True)
plt.xlabel("Actual")
plt.ylabel("Predicted")
```

Out[ ]:
```
Text(69.0, 0.5, 'Predicted')
```