Name:Shwetank Deshmukh
Enrollment No:23118026
Branch:Metallurgy(2nd year)

<u>Predicting Electrical Conductivity of various alloy compositions</u>

Dataset

The dataset consists of metallurgical alloy samples with detailed information about their chemical composition, processing parameters, and mechanical properties. Below are the columns available in the dataset:

Features:

- **Alloy formula** – Chemical formula of the alloy
- **Alloy class** – Type or category of the alloy
- **Elemental Composition:** Cu, Al, Ag, B, Be, Ca, Co, Ce, Cr, Fe, Hf, La, Mg, Mn, Mo, Nb, Nd, Ni, P, Pb, Pr, Si, Sn, Ti, V, Zn, Zr (Weight fractions of elements in the alloy)
- **Processing Conditions:**
  - **Tss (K)** – Solution treatment temperature in Kelvin
  - **tss (h)** – Solution treatment time in hours
  - **CR reduction (%)** – Cold rolling reduction percentage
  - **Aging** – Whether aging treatment was performed
  - **Tag (K)** – Aging temperature in Kelvin
  - **tag (h)** – Aging time in hours
  - **Secondary thermo-mechanical process** – Additional processing details

Target Variable:

- **Electrical Conductivity (%IACS)** – The electrical conductivity of the alloy, measured as a percentage of the International Annealed Copper Standard (IACS)

Additional Properties:

- **Hardness (HV)** – Hardness in Vickers
- **Yield Strength (MPa)** – The yield strength of the alloy in MPa
- **Ultimate Tensile Strength (MPa)** – The ultimate tensile strength in MPa

The goal is to minimize MAE(Mean Absolute Error), ensuring predictions are as close as possible to actual values.

STEP 1:DATA PREPROCESSING AND CLEANING

Dataset Overview

The dataset consists of multiple material properties and processing conditions related to alloy compositions. The objective of this preprocessing was to clean the data, handle missing values, detect and treat outliers, and prepare the dataset for machine learning models.

TRAIN DATA PREPROCESSING

1. Data Loading

- Imported necessary libraries: `pandas`, `numpy`, `matplotlib.pyplot`, `seaborn`.
- Read the training dataset (`train.csv`) and test dataset (`test.csv`) from the given file paths using `pd.read_csv()`.

---

2. Handling Missing Values

- Checked for missing values in the dataset using `df_train.isnull().sum()`.
- Dropped columns:
  - 'Alloy formula' and 'Alloy class' (Reasons not specified, but possibly not useful for numerical analysis).
  - 'Ultimate tensile strength (MPa)' and 'Yield strength (MPa)' due to a large number of missing values.
- Imputed missing values:
  - Continuous numerical variables (`Tss (K)`, `tss (h)`, `Tag (K)`, `tag (h)`) were filled with their respective median values due to the presence of outliers.
  - `Hardness (HV)`, which had no significant outliers, was filled using the mean value.
  - The categorical column 'Secondary thermo-mechanical process' was filled with the most frequent value (mode) as it contained 'Y' (Yes) and 'N' (No).
  - Rows with missing values in 'Electrical conductivity (%IACS)' (target variable) were dropped since there were only two missing instances.

---

3. Outlier Detection and Treatment

- Selected continuous numerical variables for outlier detection:
  - `Tss (K)`, `tss (h)`, `Tag (K)`, `tag (h)`, `Hardness (HV)`.
- Used Interquartile Range (IQR) method to identify outliers:
  - Computed Q1 (25th percentile) and Q3 (75th percentile).
  - Calculated IQR = Q3 - Q1.
  - Defined lower and upper bounds:
    - Lower bound = Q1 - 1.5 * IQR
    - Upper bound = Q3 + 1.5 * IQR
  - Counted the number of outliers per column:
    - `Tss (K)`: 81 outliers
    - `tss (h)`: 19 outliers
    - `Tag (K)`: 126 outliers
    - `tag (h)`: 132 outliers
    - `Hardness (HV)`: 0 outliers

- Visualized outliers using a boxplot.
- Since outliers were present in most numerical columns, missing values were filled using the median instead of the mean to avoid skewing the data.

---

4. Encoding Categorical Variables

- Converted categorical variables into dummy variables using `pd.get_dummies()`:
  - 'Alloy class', 'Secondary thermo-mechanical process', and 'Aging' were one-hot encoded with `drop_first=True` to avoid multicollinearity.
  - Dropped 'Alloy formula' after encoding.

---

5. Feature Selection

- Defined features (X) and target variable (y):
  - Features (`X_train`): All columns except `Electrical conductivity (%IACS)`.
  - Target (`y_train`): `Electrical conductivity (%IACS)`.

---

6. Summary of Cleaning Steps

- Removed unnecessary columns (`Alloy formula`, `Alloy class`, `Ultimate tensile strength (MPa)`, `Yield strength (MPa)`).
- Handled missing values by dropping or imputing with appropriate statistical measures (median for outliers, mean for normally distributed data, mode for categorical data).
- Identified and visualized outliers using IQR and boxplots.
- Encoded categorical variables using one-hot encoding.
- Defined feature and target variables for further modeling.

This cleaned dataset is now ready for exploratory data analysis (EDA) and machine learning modeling.

Test Data Preprocessing

Following similar preprocessing steps, the test dataset was cleaned and transformed for model evaluation.

1. Handling Missing Values

- *Hardness (HV)* missing values were replaced with the mean.
- *Secondary thermo-mechanical process* missing values were replaced with the mode.
- Dropped *Yield strength (MPa)* and *Ultimate tensile strength (MPa)* to maintain consistency with the training dataset.
- *Tss (K)*, *tss (h)*, *Tag (K)*, *tag (h)* missing values were filled with their respective median values.
- Dropped the *Alloy formula* column as done in training data.

2. Encoding Categorical Variables

- Applied one-hot encoding to *Alloy class*, *Secondary thermo-mechanical process*, and *Aging* using `pd.get_dummies()` with `drop_first=True` to maintain consistency with the training dataset.

STEP 2:TRAIN VARIOUS ML MODELS ON THE TRAIN DATASET

1) Baseline Models

Linear Regression

- A simple linear regression model was implemented.

- **Training MAE:** `[13.6708]`

- **Validation MAE:** `{lr_mae:.4f}`

Ridge and Lasso Regression

- Ridge Regression (**alpha=1.0**) and Lasso Regression (**alpha=0.01**) were tested to handle multicollinearity.
- **Ridge Training MAE:** `[13.7139]`
- **Ridge Validation MAE:** `{ridge_mae:.4f}`
- **Lasso Training MAE:** `[13.7319]`
- **Lasso Validation MAE:** `{lasso_mae:.4f}`

2) Tree-Based Models

Decision Tree Regressor

- A **Decision Tree Regressor** with `max_depth=5` was used to prevent overfitting.
- **Training MAE:** `[12.3516]`
- **Validation MAE:** `{dt_mae:.4f}`

Random Forest Regressor

- A **Random Forest model** (`n_estimators=100, max_depth=10`) was trained to improve predictive power.
- **Training MAE:** `[7.9457]`

- **Validation MAE:** `{rf_mae:.4f}`

3.3 Boosting Models

XGBoost

- Multiple **XGBoost** models were trained with different hyperparameters:
  - `n_estimators=1200, learning_rate=0.008, max_depth=5`
  - **Validation MAE:** `{xgb_mae:.4f}`
  - **Training MAE:** `[6.1039]`
- Optimized XGBoost:
  - `n_estimators=1200, learning_rate=0.008, max_depth=7, subsample=0.8, colsample_bytree=0.8, reg_alpha=0.05, reg_lambda=2.0, gamma=0.3, min_child_weight=6`
  - **Training MAE:** `[6.1437]`
  - **Validation MAE:** `{xgb_mae_valid:.4f}`

CatBoost

- **CatBoost** was trained with `iterations=2000, depth=7, learning_rate=0.008, L2 regularization=3`.
- **Training MAE:** `[13.8819]`
- **Validation MAE:** `{catboost_mae:.4f}`

3.4 Other Regression Models

Support Vector Regression (SVR)

- **SVR (RBF kernel, C=10, epsilon=0.1)** was applied after **StandardScaler** normalization.
- **Training MAE:** `[14.5884]`
- **Validation MAE:** `{svr_mae:.4f}`

K-Nearest Neighbors (KNN) Regressor

- **KNN (k=10, distance-weighted)** was used for prediction.
- **Training MAE:** `[15.1558]`
- **Validation MAE:** `{knn_mae:.4f}`

Neural Network (MLP Regressor)

- A **Multi-Layer Perceptron (MLP)** with architecture `(128, 64, 32)` was trained.
- **Training MAE:** `[15.4751]`
- **Validation MAE:** `{nn_mae:.4f}`

3.5 Cross-Validation Performance

- A **5-fold cross-validation** was performed using **CatBoost** to ensure robustness.

- **Average MAE across folds:** `{np.mean(mae_scores):.4f}`

Final Observations:

1. **Tree-based models (XGBoost, Random Forest, CatBoost)** outperformed linear models due to their ability to capture non-linearity in the data.
2. **XGBoost (Optimized-CatBoost) provided the best validation MAE**, indicating strong generalization.
3. **Neural Networks (MLP) showed competitive performance but required more hyperparameter tuning.**
4. **SVR and KNN performed moderately well but were not the best choices for the dataset.**
5. **Cross-validation results confirmed that CatBoost provided stable and consistent predictions.**

<u>STEP 3:TESTING THE MAE SCORE FOR test.csv</u>

VARIOUS MODEL	MAE SCORE ON test_csv
Linear Regression	14.13389
Decision Tree Regressor	14.94030
Random Forest Regressor	14.28859
XGBoost	13.76118

CatBoost	13.49917
Support Vector Regression (SVR)	14.0250
Neural Network (MLP Regressor)	15.40987

Code for CatBoost model

[91]:

```
from catboost import CatBoostRegressor

cat_model = CatBoostRegressor(
    iterations=2000, learning_rate=0.008, depth=7,
    l2_leaf_reg=3, loss_function='MAE', random_seed=42,
    verbose=100
)

cat_model.fit(X_train, y_train, eval_set=(X_valid, y_valid))
y_valid_pred = cat_model.predict(X_valid)
print(f"CatBoost MAE: {mean_absolute_error(y_valid, y_valid_pred):.4f}")
```

```
0:	learn: 13.7072964	test: 14.3039938	best: 14.3039938 (0)	total: 7.78ms	remaining: 15.6s
100:	learn: 12.5550830	test: 14.1272414	best: 14.1272414 (100)	total: 584ms	remaining: 11s
200:	learn: 11.6934679	test: 14.0296240	best: 14.0213314 (190)	total: 1.16s	remaining: 10.4s
300:	learn: 10.9405221	test: 14.0163888	best: 13.9944239 (258)	total: 1.77s	remaining: 9.99s
400:	learn: 10.3132596	test: 13.9833135	best: 13.9772714 (392)	total: 2.35s	remaining: 9.36s
500:	learn: 9.7618092	test: 13.9771933	best: 13.9671072 (480)	total: 2.93s	remaining: 8.77s
600:	learn: 9.2456662	test: 13.9864747	best: 13.9671072 (480)	total: 3.51s	remaining: 8.17s
700:	learn: 8.7950517	test: 13.9816339	best: 13.9671072 (480)	total: 4.09s	remaining: 7.59s
800:	learn: 8.3798870	test: 13.9380532	best: 13.9335149 (789)	total: 4.69s	remaining: 7.01s
900:	learn: 8.0122339	test: 13.9227547	best: 13.9186219 (870)	total: 5.26s	remaining: 6.42s
1000:	learn: 7.6891823	test: 13.9220017	best: 13.9147241 (915)	total: 5.84s	remaining: 5.83s
1100:	learn: 7.4054810	test: 13.9128691	best: 13.9123088 (1099)	total: 6.43s	remaining: 5.25s
1200:	learn: 7.1253364	test: 13.9072067	best: 13.9004642 (1118)	total: 7s	remaining: 4.66s
1300:	learn: 6.8510901	test: 13.9056529	best: 13.9004642 (1118)	total: 7.58s	remaining: 4.07s
1400:	learn: 6.5944222	test: 13.9043626	best: 13.9004642 (1118)	total: 8.16s	remaining: 3.49s
1500:	learn: 6.3714230	test: 13.8981528	best: 13.8965049 (1474)	total: 8.74s	remaining: 2.91s
1600:	learn: 6.1863690	test: 13.8898422	best: 13.8896977 (1598)	total: 9.32s	remaining: 2.32s
1700:	learn: 6.0111125	test: 13.8849922	best: 13.8819120 (1689)	total: 9.92s	remaining: 1.74s
1800:	learn: 5.8270244	test: 13.8850576	best: 13.8819120 (1689)	total: 10.5s	remaining: 1.16s
1900:	learn: 5.6451417	test: 13.9027413	best: 13.8819120 (1689)	total: 11.1s	remaining: 579ms
1999:	learn: 5.5011045	test: 13.9116717	best: 13.8819120 (1689)	total: 11.7s	remaining: 0us

bestTest = 13.881912
bestIteration = 1689

Shrink model to first 1690 iterations.
CatBoost MAE: 13.8819
```

Below is the link for the excel in which their are ID of alloys and their corresponding Electrical Conductivity the model had predicted .This gave me the least MAE score of 13.49917

Cat_model.xlsx