Concrete
Compressive Strength
Prediction

Problem Statement:

- The objective is to predict the concrete compressive strength of concrete block.

- Prediction results tells the optimal way to get the maximum compressive strength for a unit block of concrete using the ingredients.

Data:

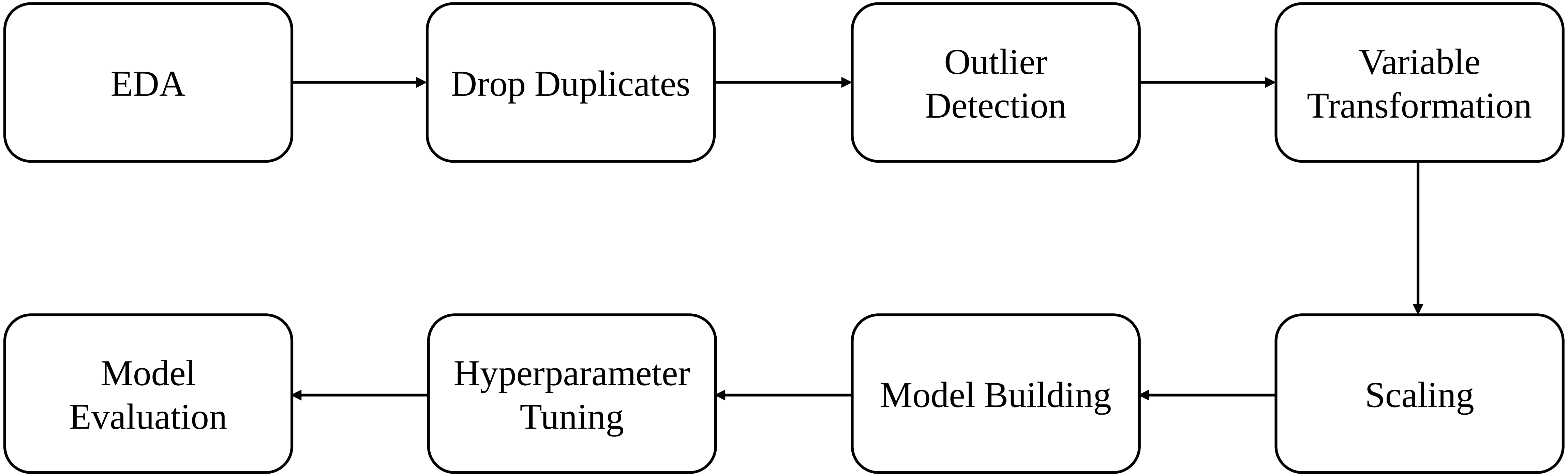
- Dataset is taken from UCI Machine Learning Repository.
- It’s a Supervised – Regression problem.
- Shape of the data (rows, columns): (1030, 9)
- 8 Quantitative input variable, and 1 Quantitative output variable
- **Input Features**: Cement, Furnace Slag, Fly Ash, Water, Superplasticizer, Coarse Aggregate, Fine Aggregate, and Age.
- **Target Feature**: Concrete Compressive Strength.

Process:

- In Jupyter Notebook
  - Explored the data.
  - Tried and tested all the methods.
  - Streamlined the process and used the results in modular coding.

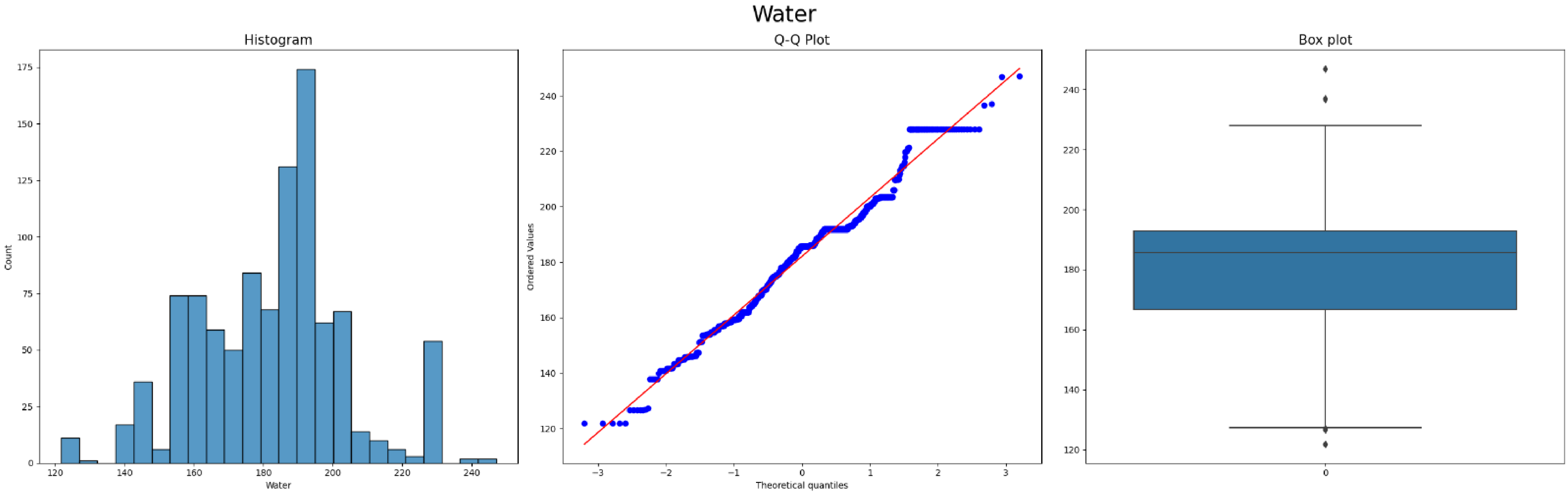
- Modular Coding
  - Complete End to End Project is made with all the Exception and Logging included.
  - Also Streamlit app is created for prediction and deployed it.

Project Flow:



Outlier Detection:

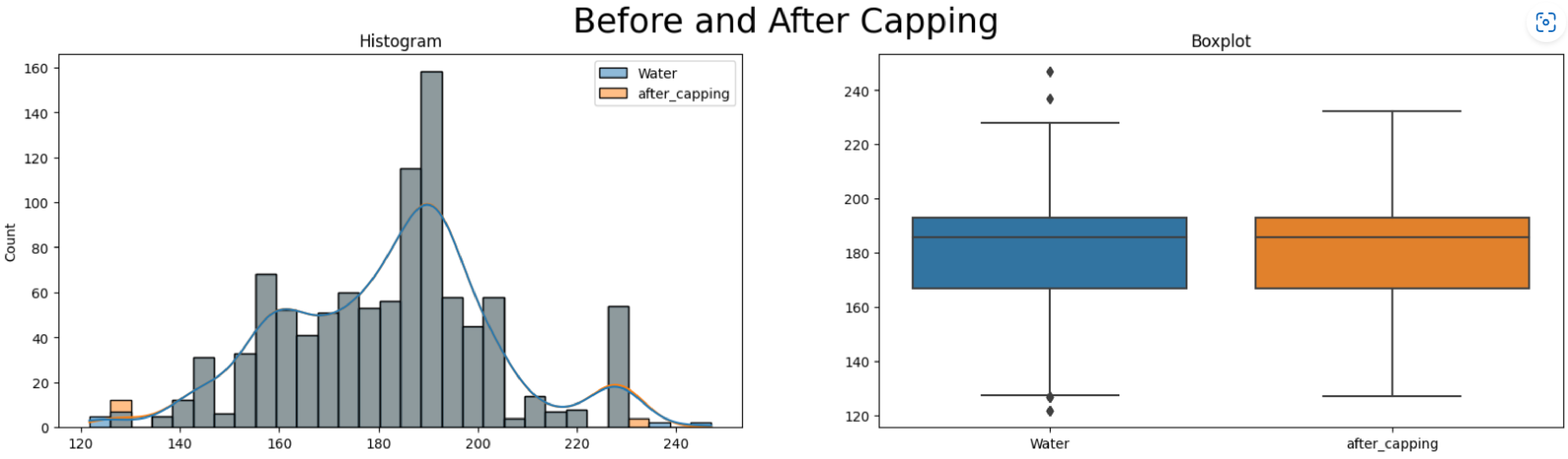
- First, we check the distribution, Q-Q plot and Box plot for every feature. depending on the distribution, we proceed further.



- Three methods to cap the outliers.
  - Gaussian (Z-Score)
  - IQR
  - Percentile (Winsorization)

- Depending on the distribution for every feature, we perform capping and check the results.
- For detailed analysis, check the Jupyter notebook here.

Results:




```
Using IQR method for outlier detection
The feature Water contains 1.493% of outliers.
The lower_limit for Water is 127.14999999999998
The upper_limit for Water is 232.35000000000002
```

Variable Transformation:

- Almost all the features are skewed, as some ML models assume normality, we need to transform the distribution.
- Some popular transformation techniques are:
  - Logarithmic Transformation
  - Logarithmic Transformation plus constant
  - Exponential
  - Reciprocal
  - Box-Cox
  - Yeo-Johnson

- Performed all the transformation techniques on every feature and check the skewness of the distribution.
- Skewness ranges from (-inf to +inf), but for all practical purposes it ranges from (-3 to +3).

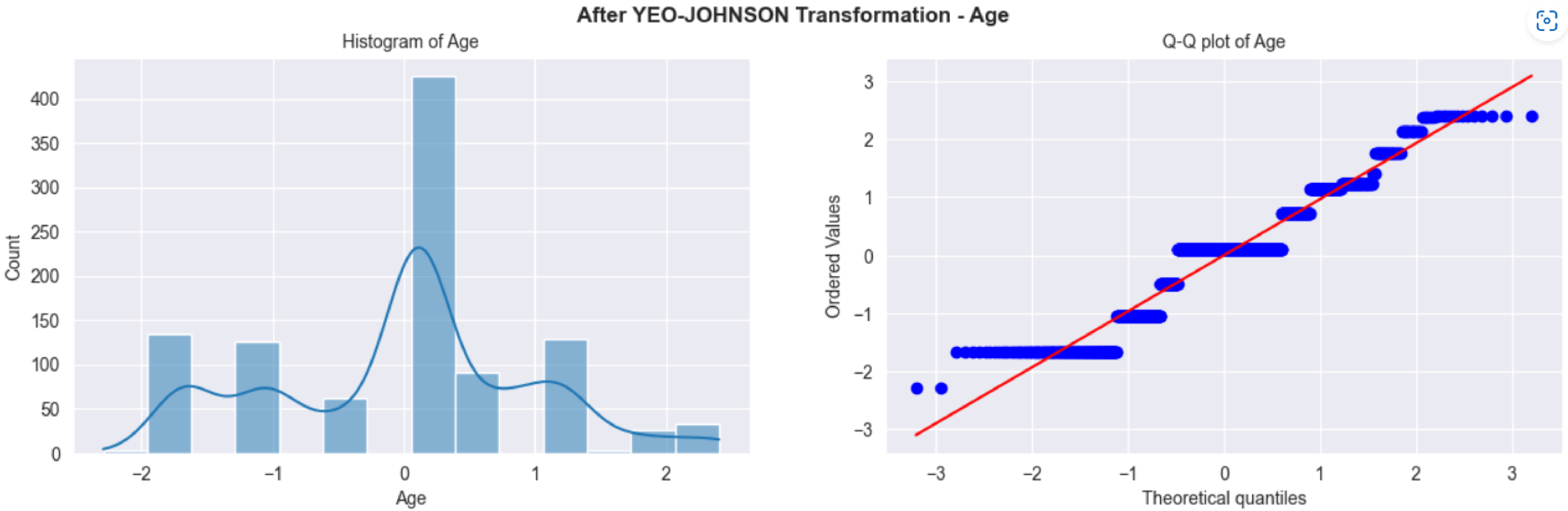
For Age:

	Transformation Name	Skewness
5	Yeo-Johnson	-0.046405
6	Box-Cox	-0.061648
1	log_with_constant	-0.331182
3	square_root	0.499612
0	Original	1.178188
4	Power_Transformation	1.562422
2	Reciprocal	1.688860

Age:


Before Transformation - Age
Histogram of Age
Count
400
350
300
250
200
150
100
50
0
0
50
100
150
200
250
300
350
Age
Q-Q plot of Age
Ordered Values
300
200
100
0
−100
−3
−2
−1
0
1
2
3
Theoretical quantiles

After Applying Yeo-Johnson:



Scaling:

- As some ML models are sensitive towards scale of the variable, scaling the features to same scale is must.
- There are many scaling methods:
    - MinMaxScaler
    - StandardScaler
    - AbsMaxScaler

Model Building:

- Performed 7 Machine learning algorithms on the data.
  - Linear Regression
  - Lasso Regression
  - Ridge Regression
  - Support Vector Regressor (SVR)
  - Decision Tree Regressor
  - Random Forest Regressor
  - XGBoost Regressor

Model Evaluation Metrics:

- R2 Score
  - No units, as it’s ratio
  - Ranges from 0 to 1
  - 1 – Perfect Fit
  - 0 – Not a fit
  - Closer to 1, better is the model at prediction.
- Root Mean Squared Error
  - Closer to zero, better is the model.
  - Units are same as target feature unit.

Results for all the 7 ML Models:

	Algorithm	RMSE	train_R2	test_R2	diff_R2
1	Linear Regression	7.127761	0.814293	0.792959	0.021335
2	Ridge	7.133896	0.813659	0.792602	0.021057
3	Lasso	10.171401	0.583986	0.578389	0.005597
4	Decision Tree	6.322161	0.995233	0.837115	0.158118
5	SVR	7.957428	0.778819	0.741955	0.036864
6	Random Forest	4.746714	0.981075	0.908180	0.072895
7	XGBoost	4.173695	0.994830	0.929011	0.065819

	Algorithm	RMSE	train_R2	test_R2	diff_R2
7	XGBoost	4.173695	0.994830	0.929011	0.065819
6	Random Forest	4.746714	0.981075	0.908180	0.072895
4	Decision Tree	6.322161	0.995233	0.837115	0.158118
1	Linear Regression	7.127761	0.814293	0.792959	0.021335
2	Ridge	7.133896	0.813659	0.792602	0.021057
5	SVR	7.957428	0.778819	0.741955	0.036864
3	Lasso	10.171401	0.583986	0.578389	0.005597

- Clearly, XGBoost and Random Forest performs better of all.
- Decision Tree is overfitting.

Hyperparameter Tuning:

- Hyperparameter tuning is performed on XGBoost and Random Forest.

Random Forest:

```
param_grid = {
    "n_estimators":np.linspace(100, 1000,10, dtype=int),
    'max_depth':np.linspace(2,30,15),
    'criterion':['mse', 'mae']
}
```

```
grid = GridSearchCV(estimator=RandomForestRegressor(),
                    param_grid=param_grid,
                    cv=5,
                    return_train_score=True, verbose=4,
                    n_jobs=-1)
```

Best Parameters:

```
grid.best_params_
```

```
{'criterion': 'mse', 'max_depth': 20.0, 'n_estimators': 100}
```

XGBoost:

```
param_grid = {
    "n_estimators": np.linspace(100, 1000, 10, dtype=int),
    'max_depth': np.linspace(2, 30, 15, dtype=int),
    'learning_rate': np.linspace(0.1, 1.0, 10)
}
```

```
grid = GridSearchCV(estimator=XGBRegressor(),
                    param_grid=param_grid,
                    cv=5,
                    return_train_score=True,
                    verbose=4)
```

Best Parameters:

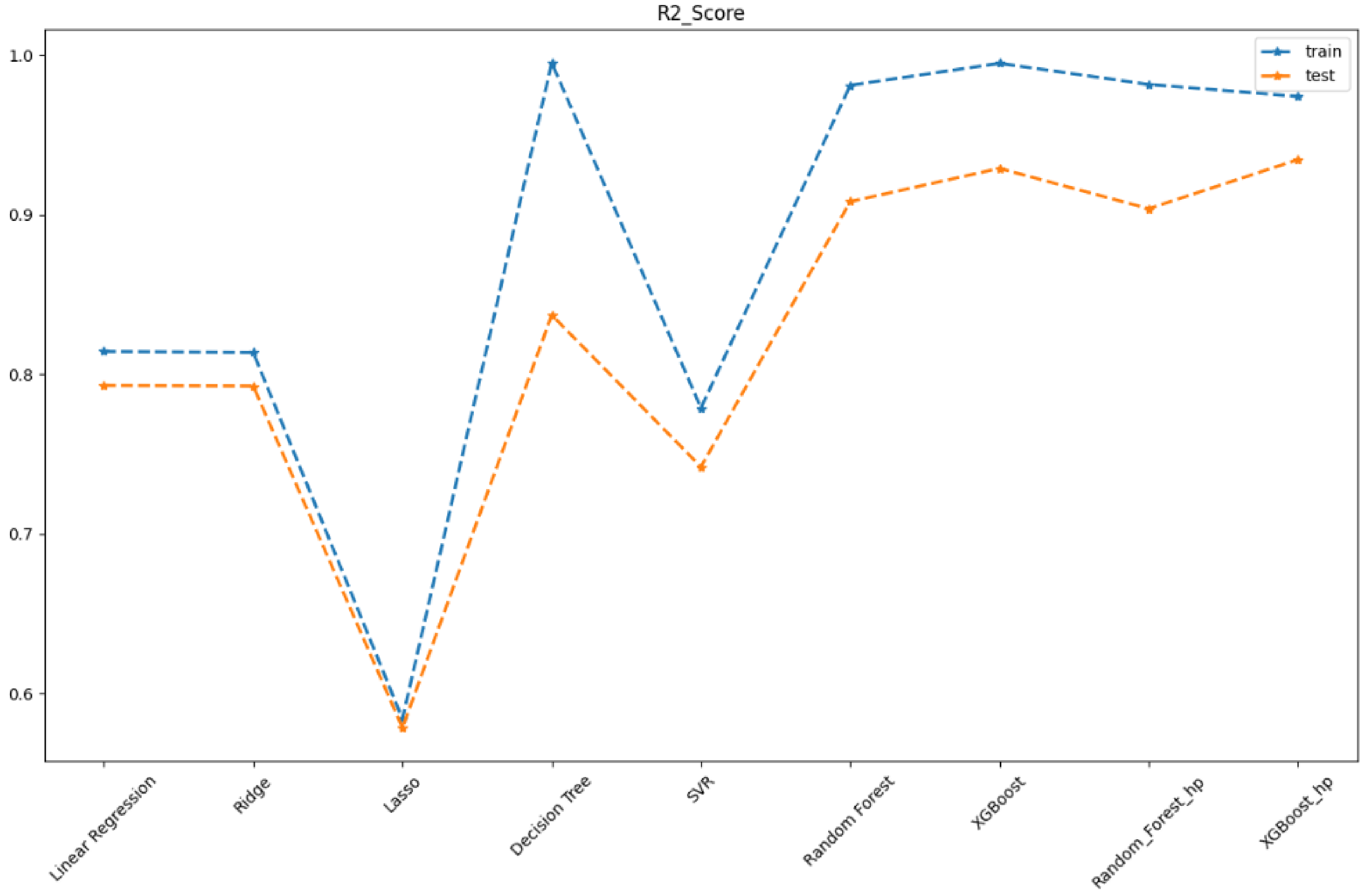
```
# best paramters
grid.best_params_
```

```
{'learning_rate': 0.1, 'max_depth': 2, 'n_estimators': 1000}
```

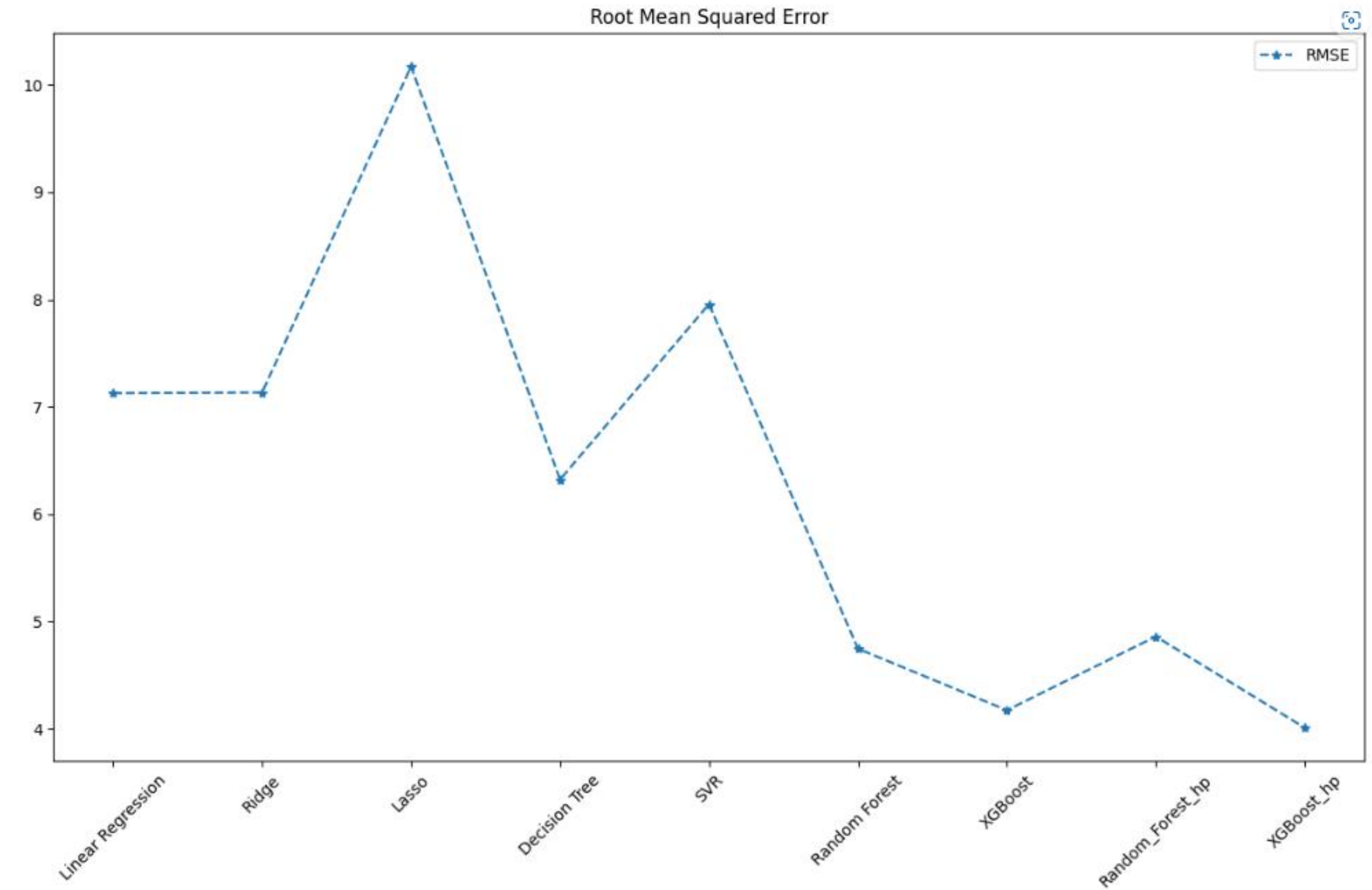
Results:

	Algorithm	RMSE	train_R2	test_R2	diff_R2
9	XGBoost_hp	4.009815	0.974111	0.934476	0.039635
7	XGBoost	4.173695	0.994830	0.929011	0.065819
6	Random Forest	4.746714	0.981075	0.908180	0.072895
8	Random_Forest_hp	4.860422	0.981592	0.903728	0.077864
4	Decision Tree	6.322161	0.995233	0.837115	0.158118
1	Linear Regression	7.127761	0.814293	0.792959	0.021335
2	Ridge	7.133896	0.813659	0.792602	0.021057
5	SVR	7.957428	0.778819	0.741955	0.036864
3	Lasso	10.171401	0.583986	0.578389	0.005597

R Squared Score:

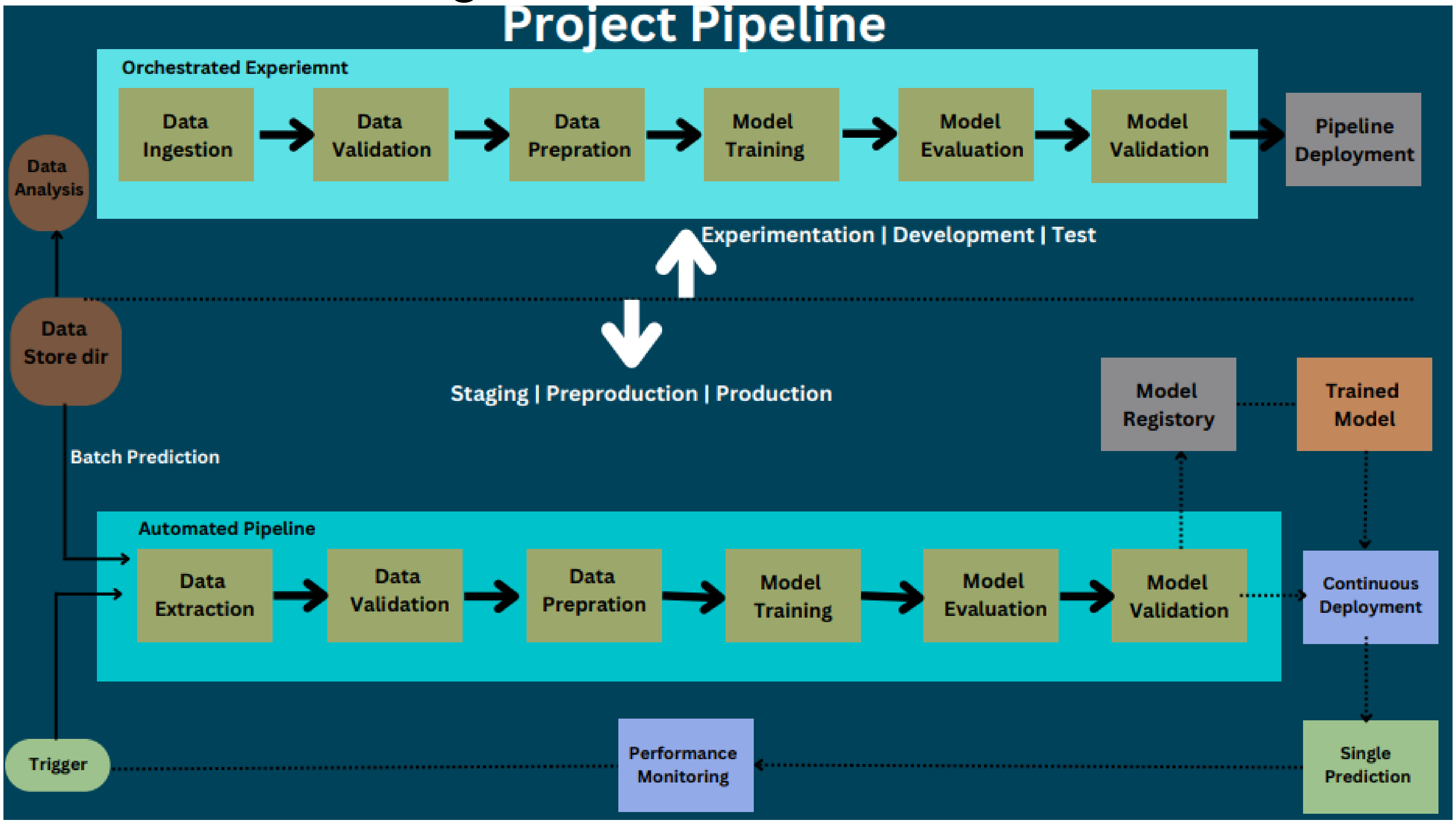


RMSE:



Root Mean Squared Error
RMSE
10
9
8
7
6
5
4
Linear Regression
Ridge
Lasso
Decision Tree
SVR
Random Forest
XGBoost
Random_Forest_hp
XGBoost_hp

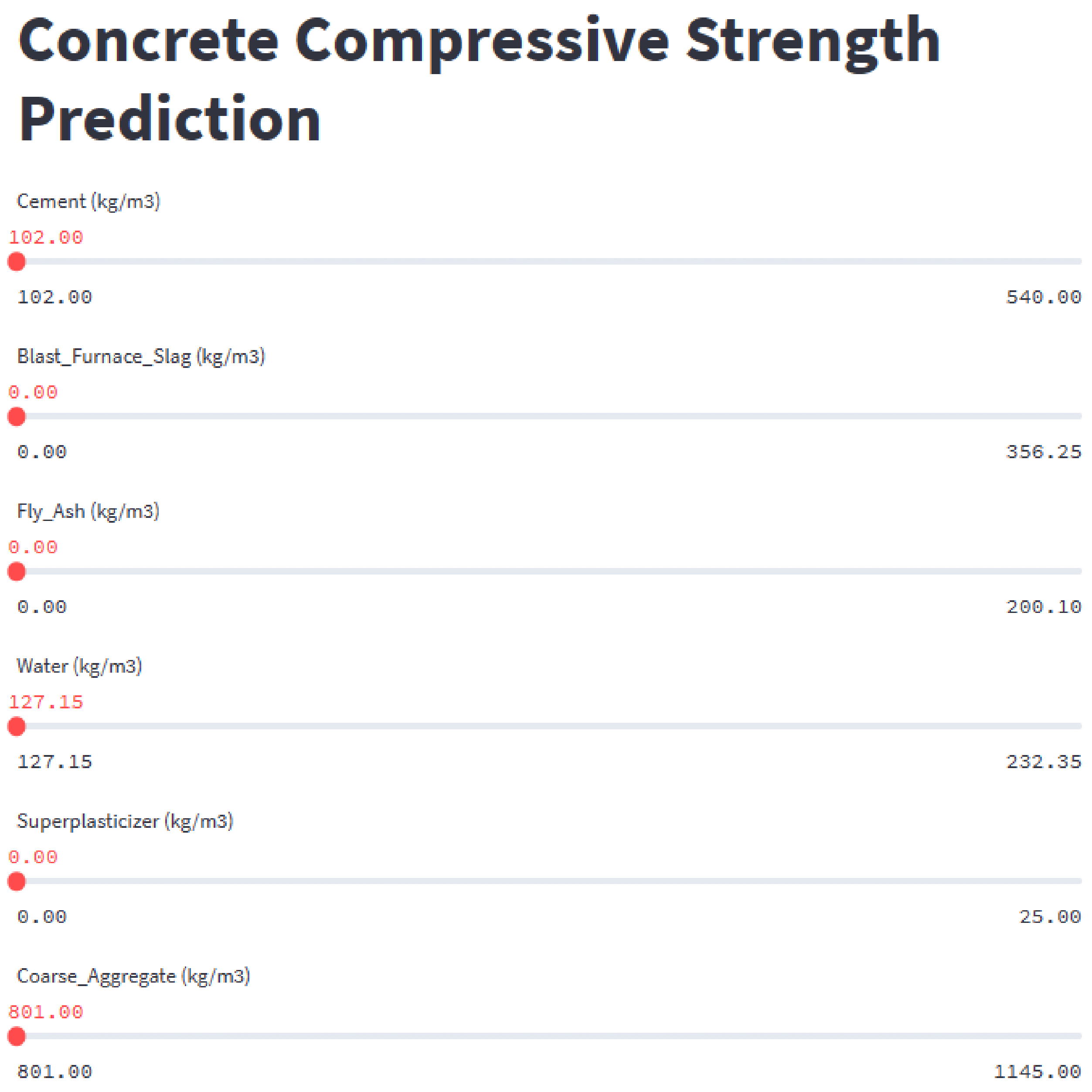
Modular Coding:



Project File Structure:

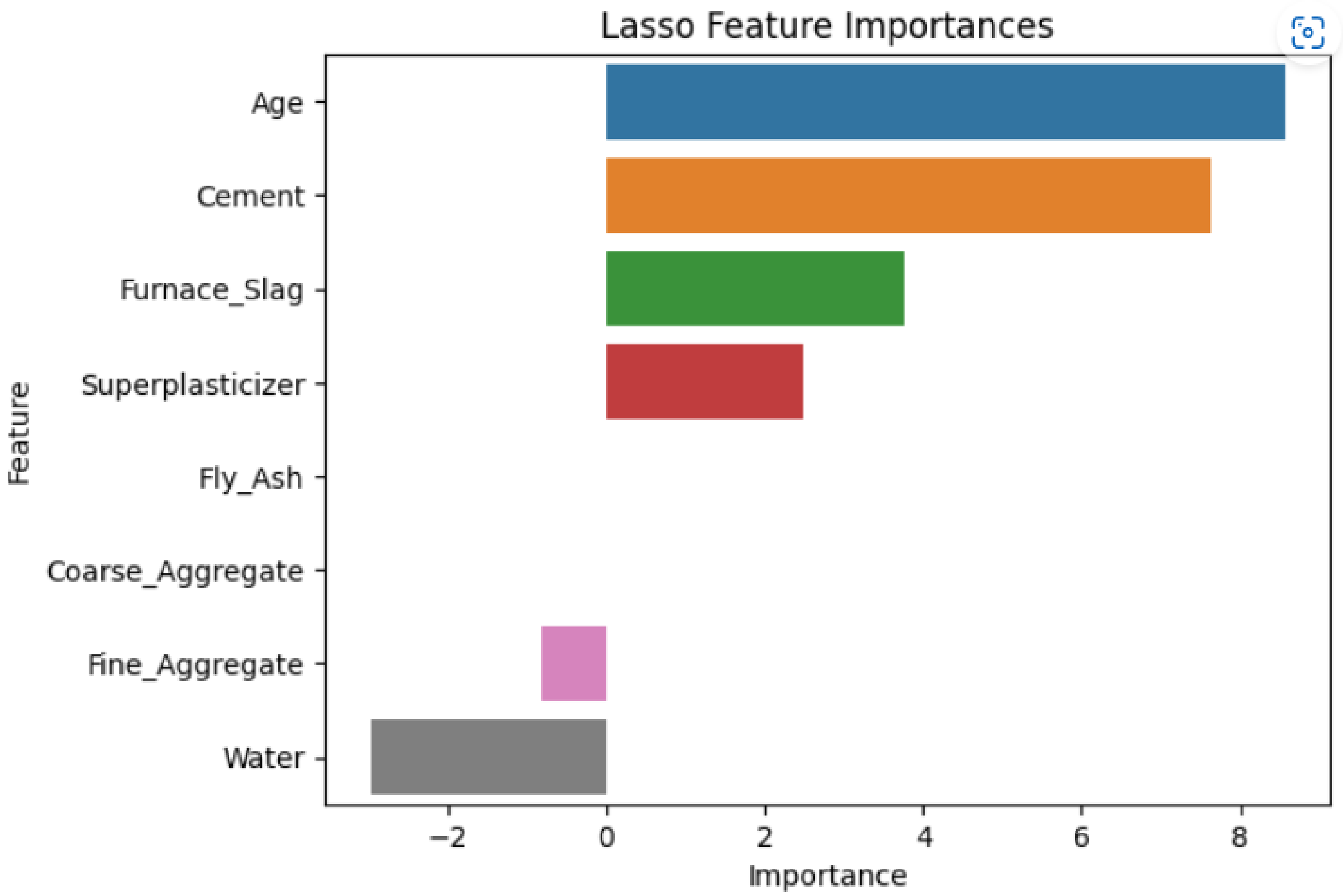
```
Concrete_Strength_Prediction  D:\DS_Live_Cla
  .github
  concrete
    components
      __init__.py
      data_ingestion.py
      data_transformation.py
      data_validation.py
      model_evaluation.py
      model_pusher.py
      model_trainer.py
    config
      __init__.py
      configuration.py
    constant
      __init__.py
    entity
      __init__.py
      artifact_entity.py
      config_entity.py
      model_factory.py
      raw_data_validation.py
    exception
    logger
    pipeline
      __init__.py
      pipeline.py
    utils
    __init__.py
  concrete_strength
```

Streamlit App:



Further Improvements:

- Only MinMaxScaler is applied, we can apply other scaling methods and check whether is there any significant change.
- Only Single Prediction Pipeline is created, Batch Prediction can be added and also, retraining the model can be added.
- Feature Selection can be performed, retrain the model and check the results.



Lasso Feature Importances
Age
Cement
Furnace_Slag
Superplasticizer
Fly_Ash
Coarse_Aggregate
Fine_Aggregate
Water
Feature
−2
0
2
4
6
8
Importance