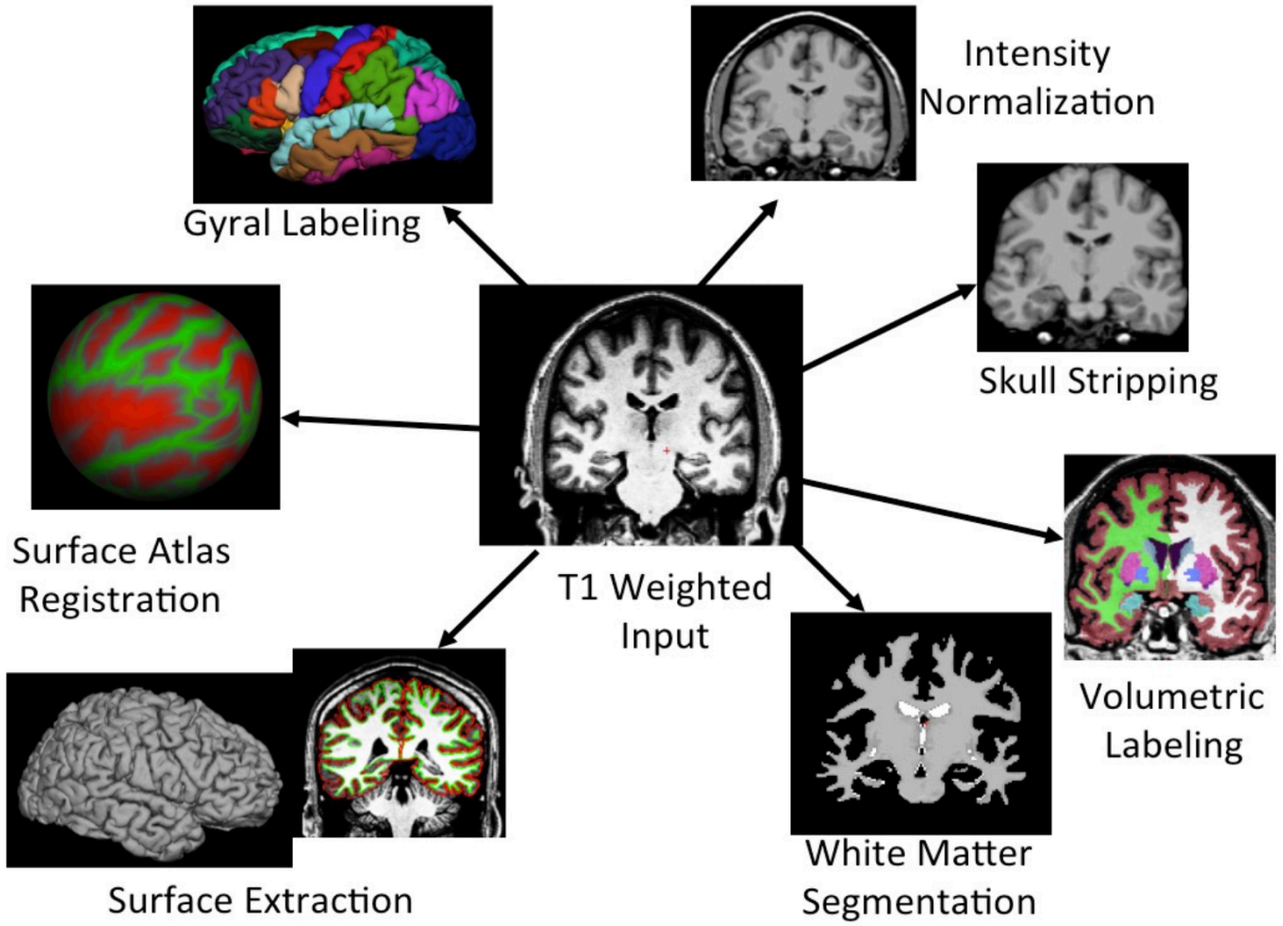
BRAIN AGE PREDICTION WITH PROGRESSION MODELS

Angela Corvino e Agata Minnocci

Esame di CMEPDA, 24 Giugno 2022

DATA SEGMENTATION

T1 weighted Brain MRIs + Freesurfer brain segmentation software →Brain features



- Number of Surface feature: 62
- Number of Thickness feature: 62
- Number of Volume feature: 108
- Number of Curvature feature: 124

- Cerebral cortex parcelled into a number of structures (depending on the atlas)
- For each structure several ROIs defined and relative features computed (area; volume; average thickness; thickness standard deviation; mean curvature)
- Global features can be considered, e.g.: white surface total area and mean thickness of cerebral cortex for both hemisphere

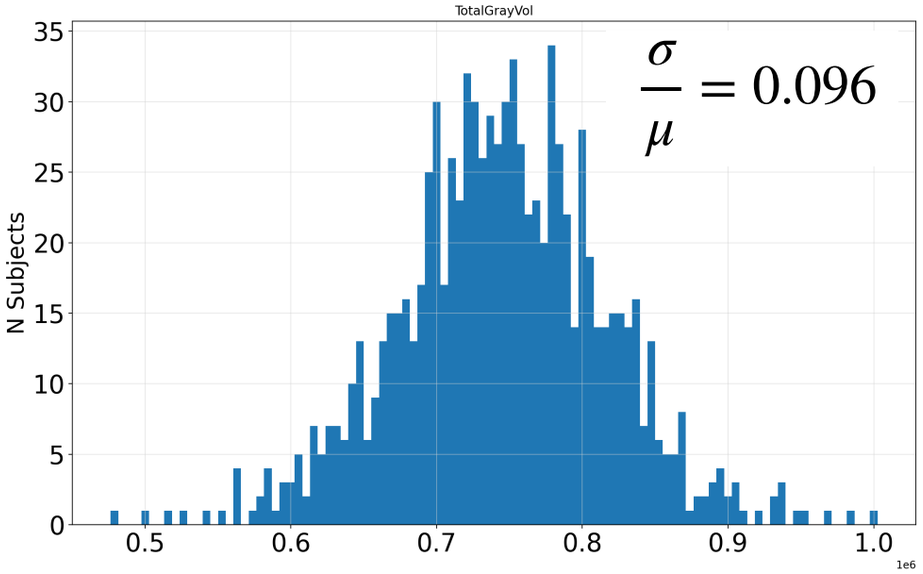
CODE REPOSITORY

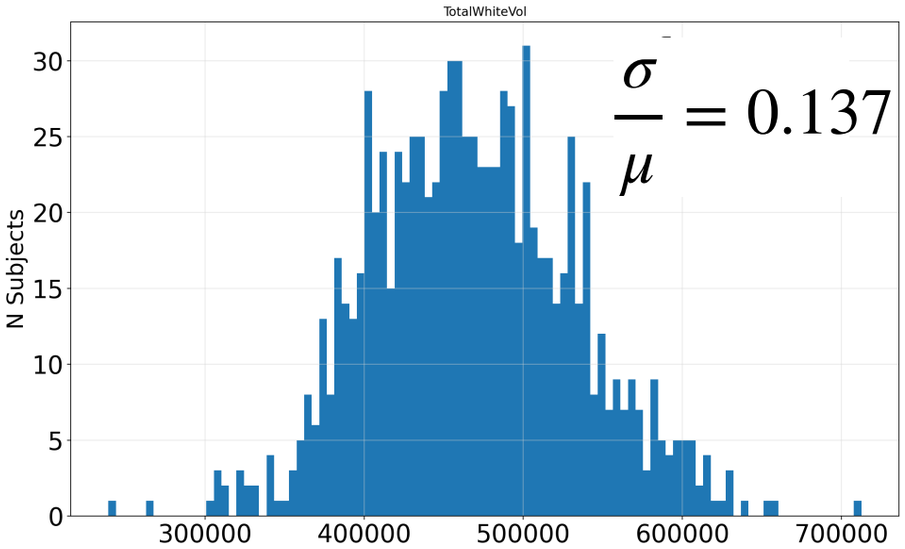
Repository

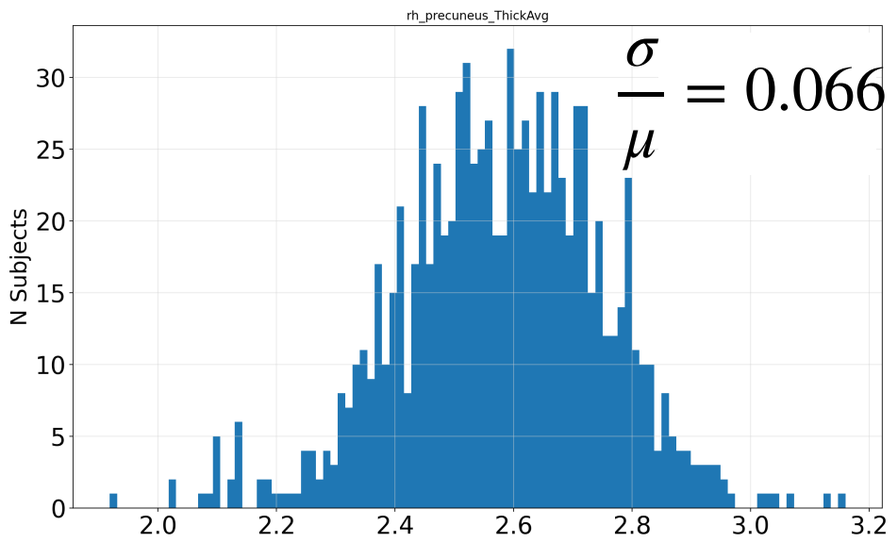
Documentation

Continuous integration

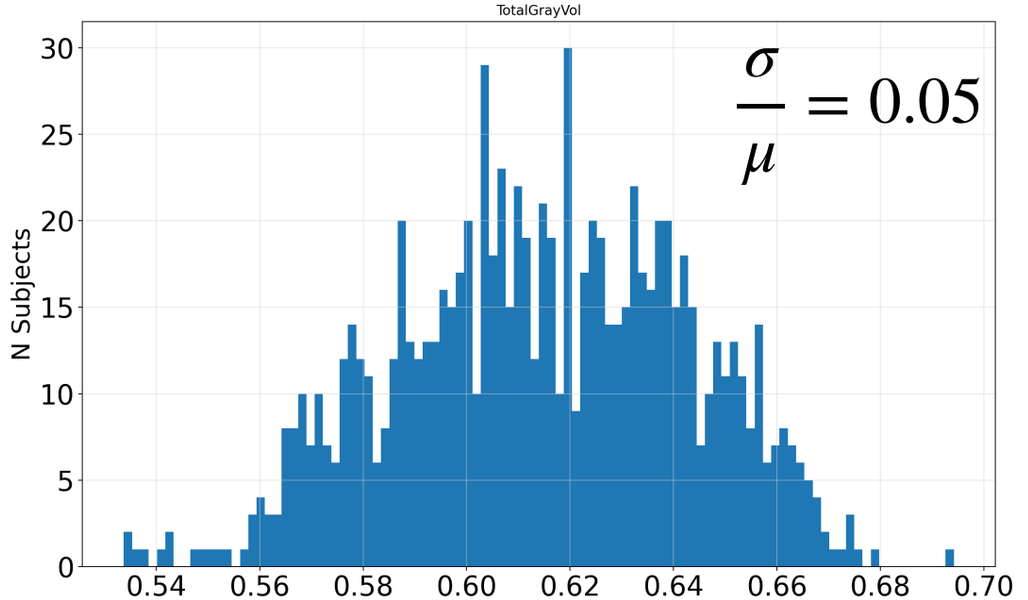
SELF NORMALIZATION

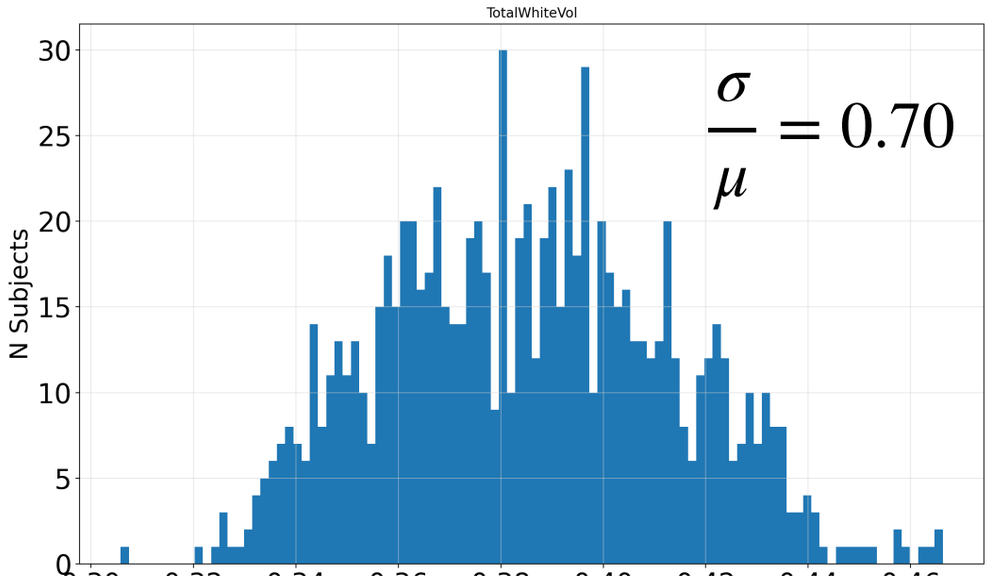


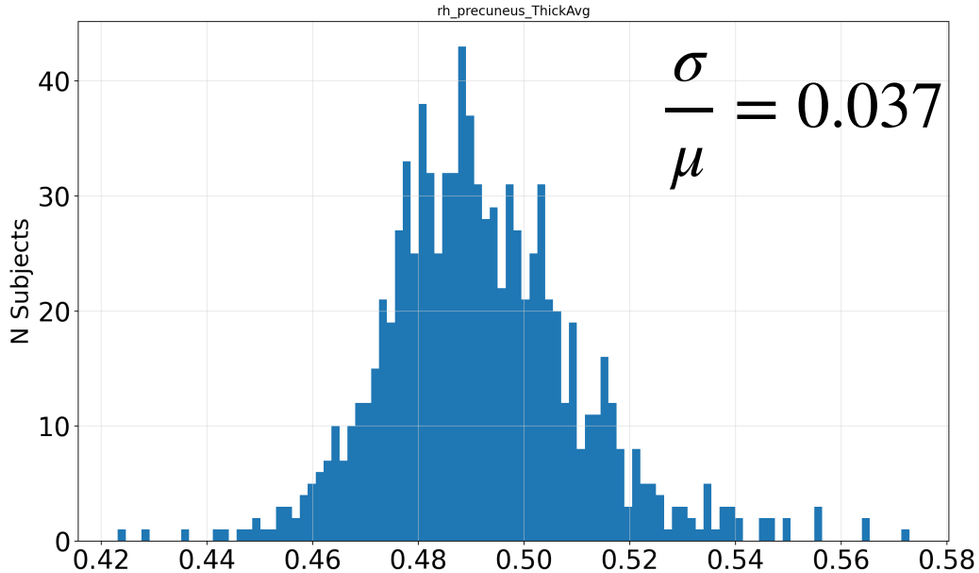




Variance Reduction





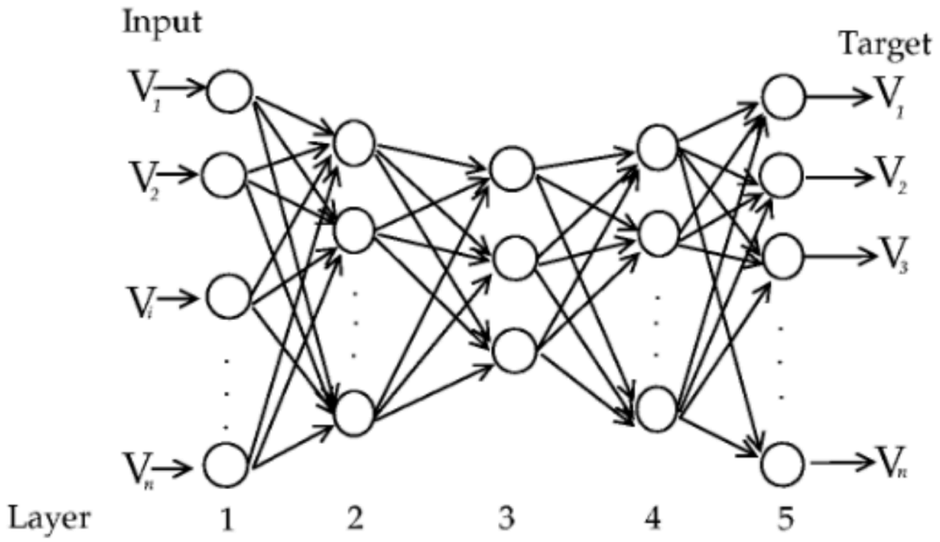


RNN FOR OUTLIERS DETECTION

RNN(feed-forward multi-layer perceptron)

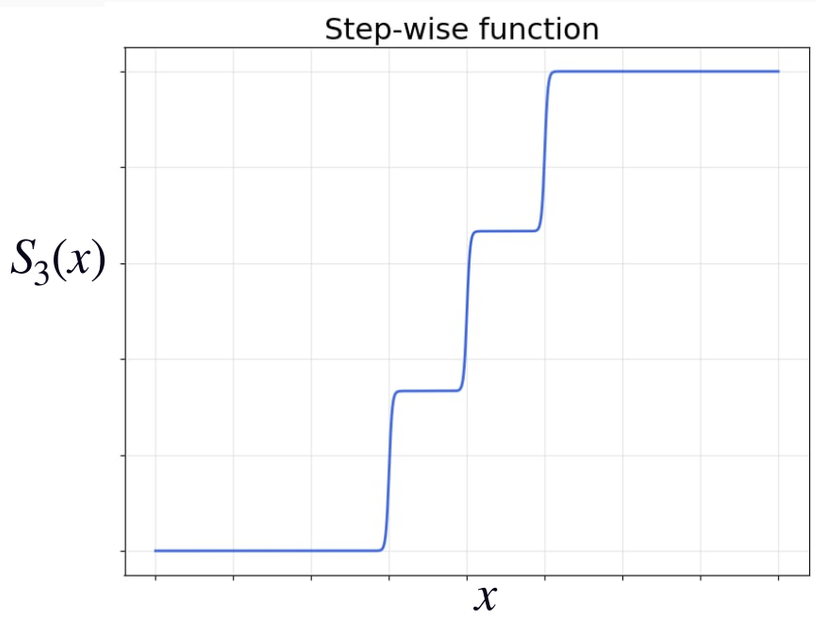
- 3 hidden layers sandwiched between input layer and output layer
- Step-wise activation function for the middle hidden layer (k = 3)

```python
def make_autoencoder(self):
    """doctsring"""
    get_custom_objects().update({"step_wise": Activation(step_wise)})

    inputs = Input(shape=self.X_train.shape[1])
    hidden = Dense(30, activation="tanh")(inputs)
    hidden = Dense(2, activation="step_wise")(hidden)
    hidden = Dense(30, activation="tanh")(hidden)
    outputs = Dense(self.X_train.shape[1], activation="linear")(hidden)

    model = Model(inputs=inputs, outputs=outputs)
    model.compile(loss="mean_absolute_error", optimizer="adam", metrics=["MAE"])
    model.summary()
    return model
```

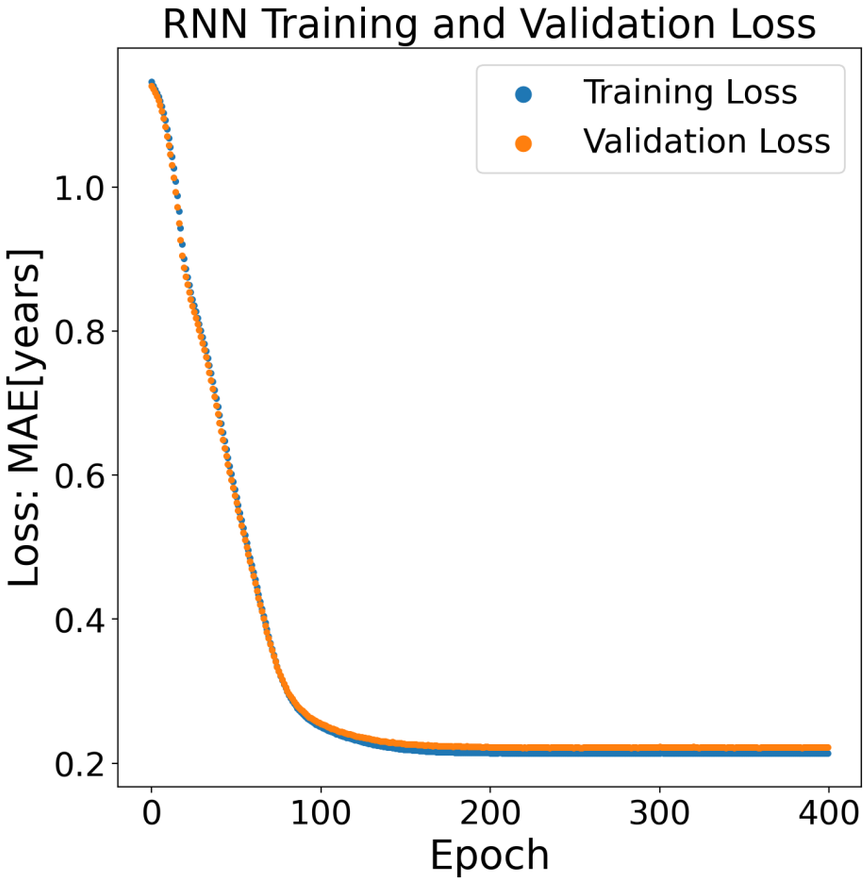


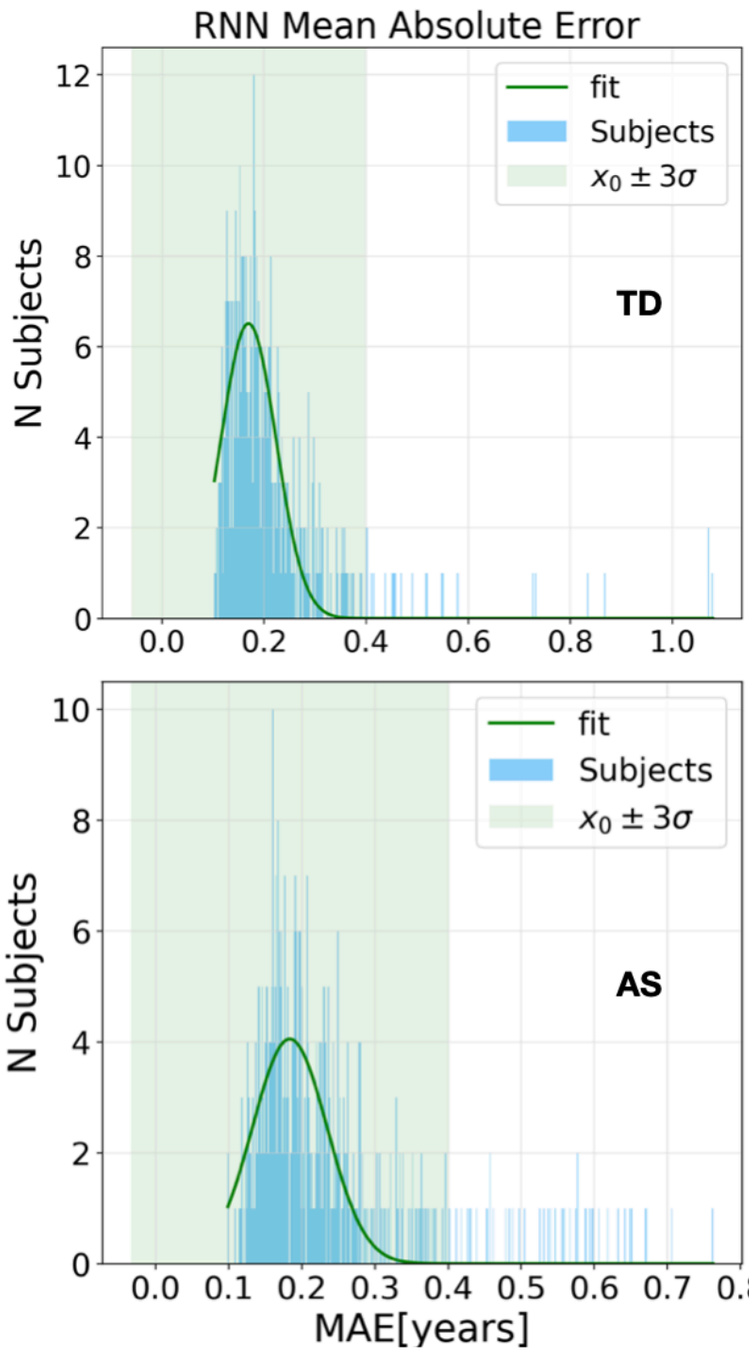

```python
def step_wise(x, N=4, a=100):
    """docstring"""
    y = 1 / 2
    for j in range(1, N):
        y += (1 / (2 * (N - 1))) * (K.tanh(a * (x - (j / N))))
    return y
```

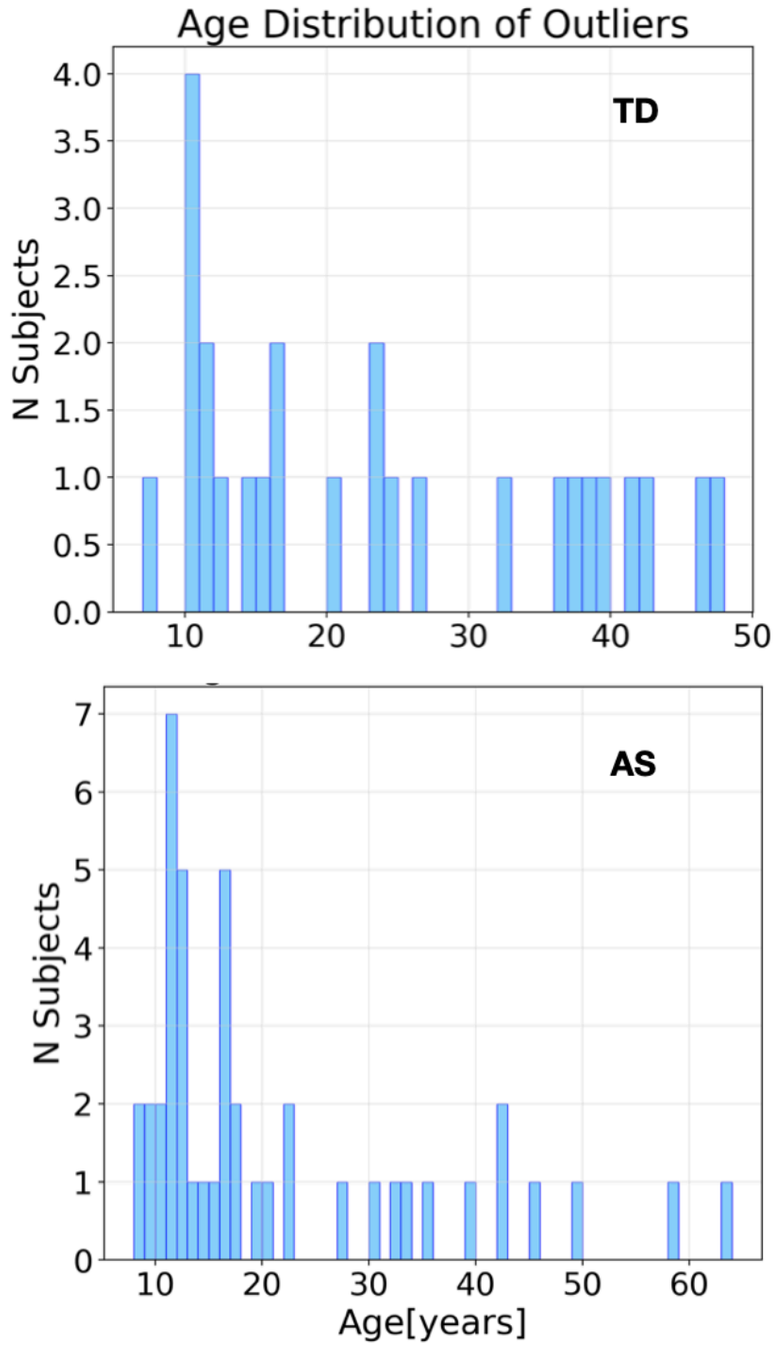
```python
def fit_autoencoder(self, model, epochs):
    """docstring"""
    # Define callbacks
    early_stopping = tf.keras.callbacks.EarlyStopping(
        monitor="val_loss", patience=10, verbose=1
    )

    history = model.fit(
        self.X_train,
        self.X_train,
        validation_split=0.4,
        epochs=epochs,
        batch_size=50,
        callbacks=[early_stopping],
        verbose=1,
    )
    with open(
        "models/autoencoder_pkl", "wb"
    ) as files:
        pickle.dump(model, files)
    return history
```



OUTLIERS DETECTION

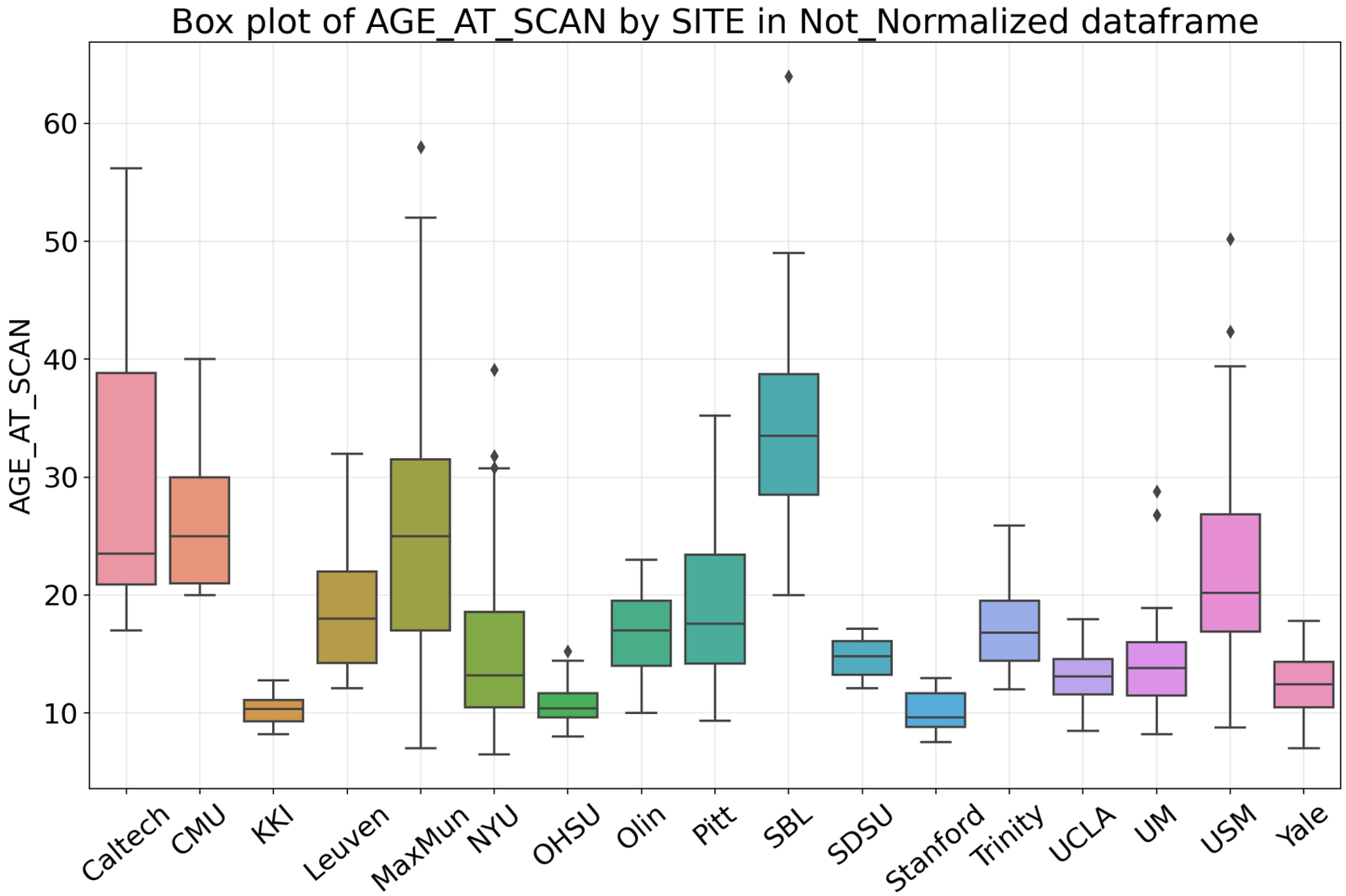






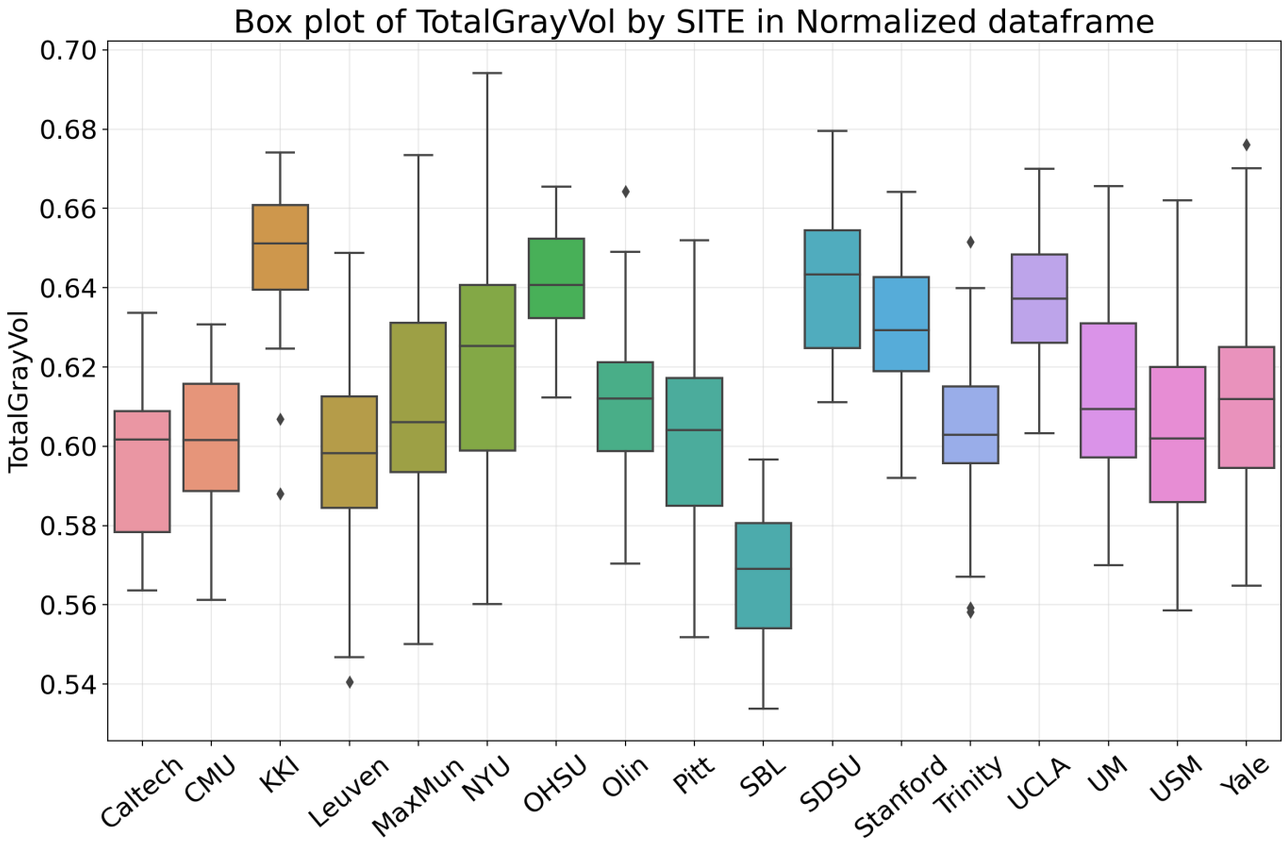
CONFOUNDER VARIABLES

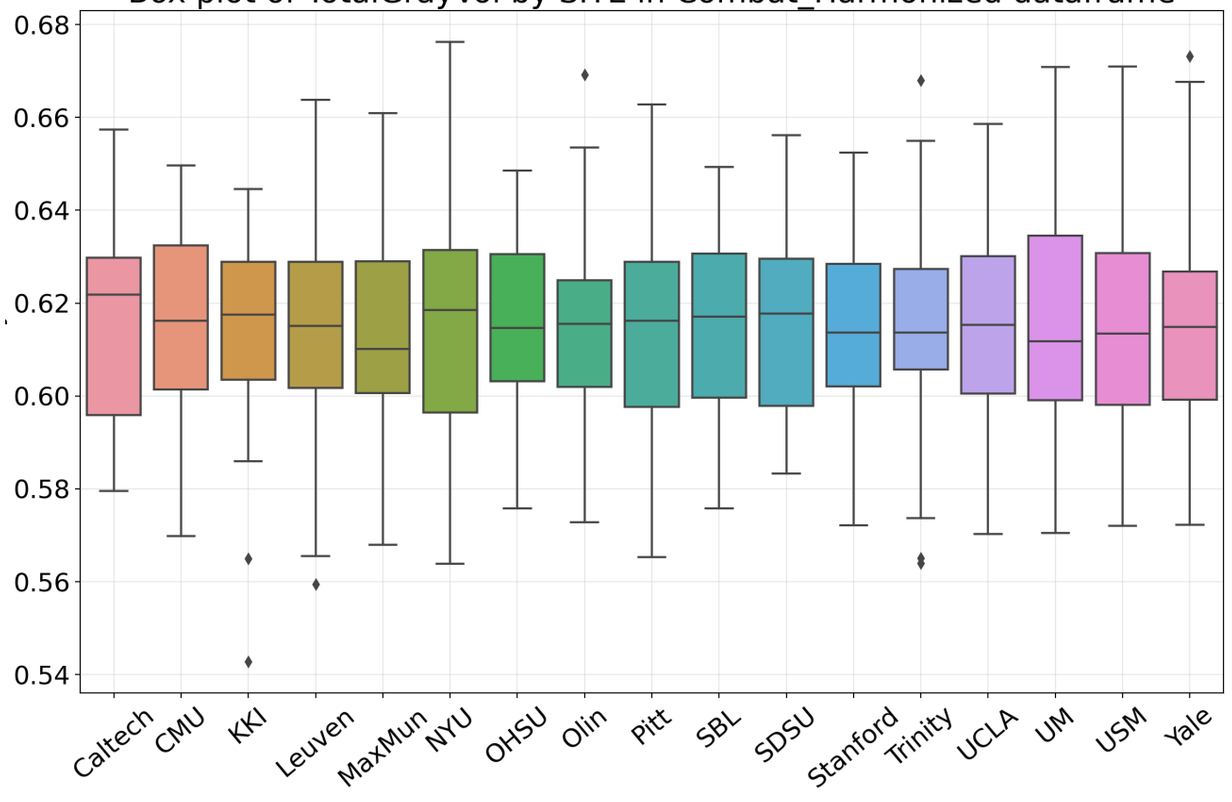
Confounders: characteristics of the samples that are not clinically relevant, but can mislead the classifier training process



DATA HARMONIZATION

Combat model $y_{ijv} = \alpha_v + W_{ij}\beta_v + \gamma_{iv} + \delta_{iv}\epsilon_{ijv}$



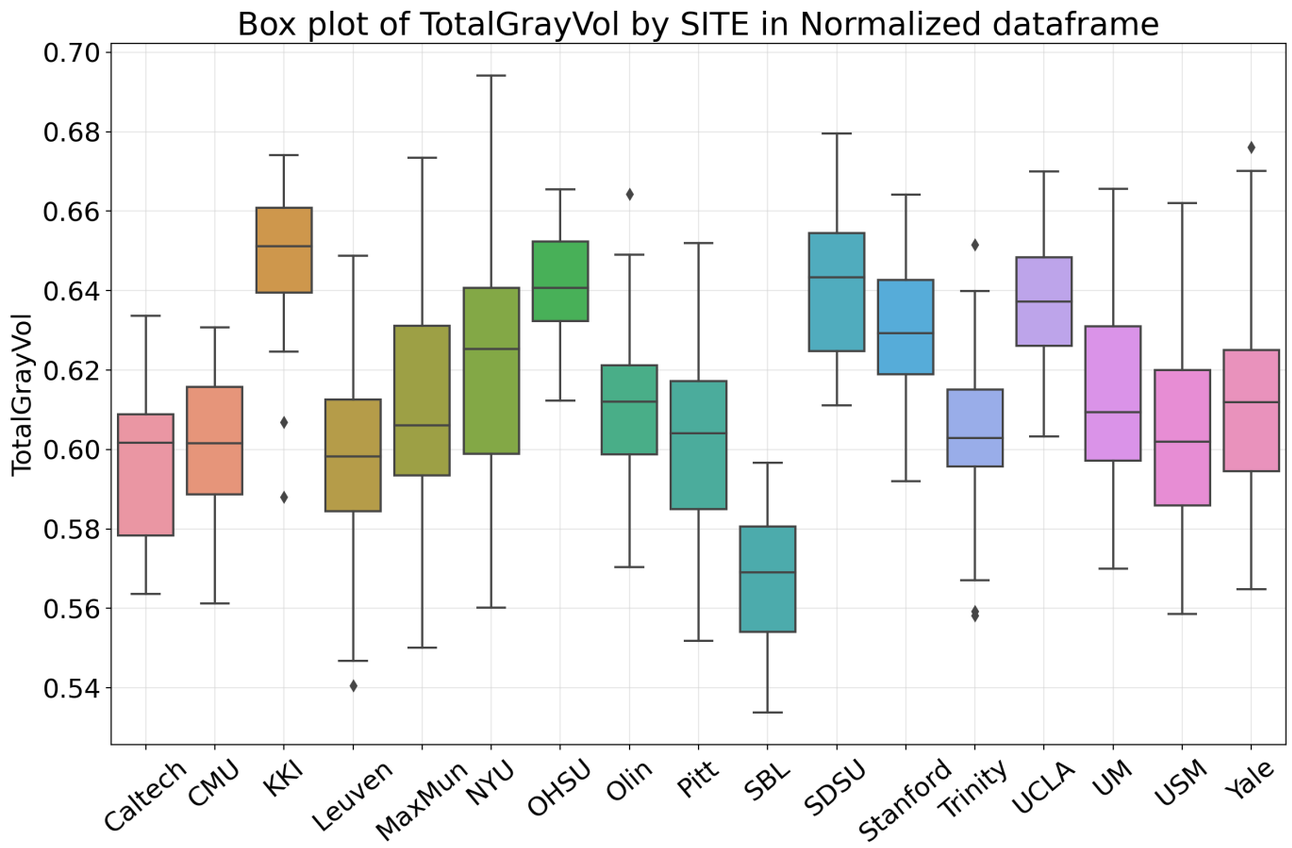


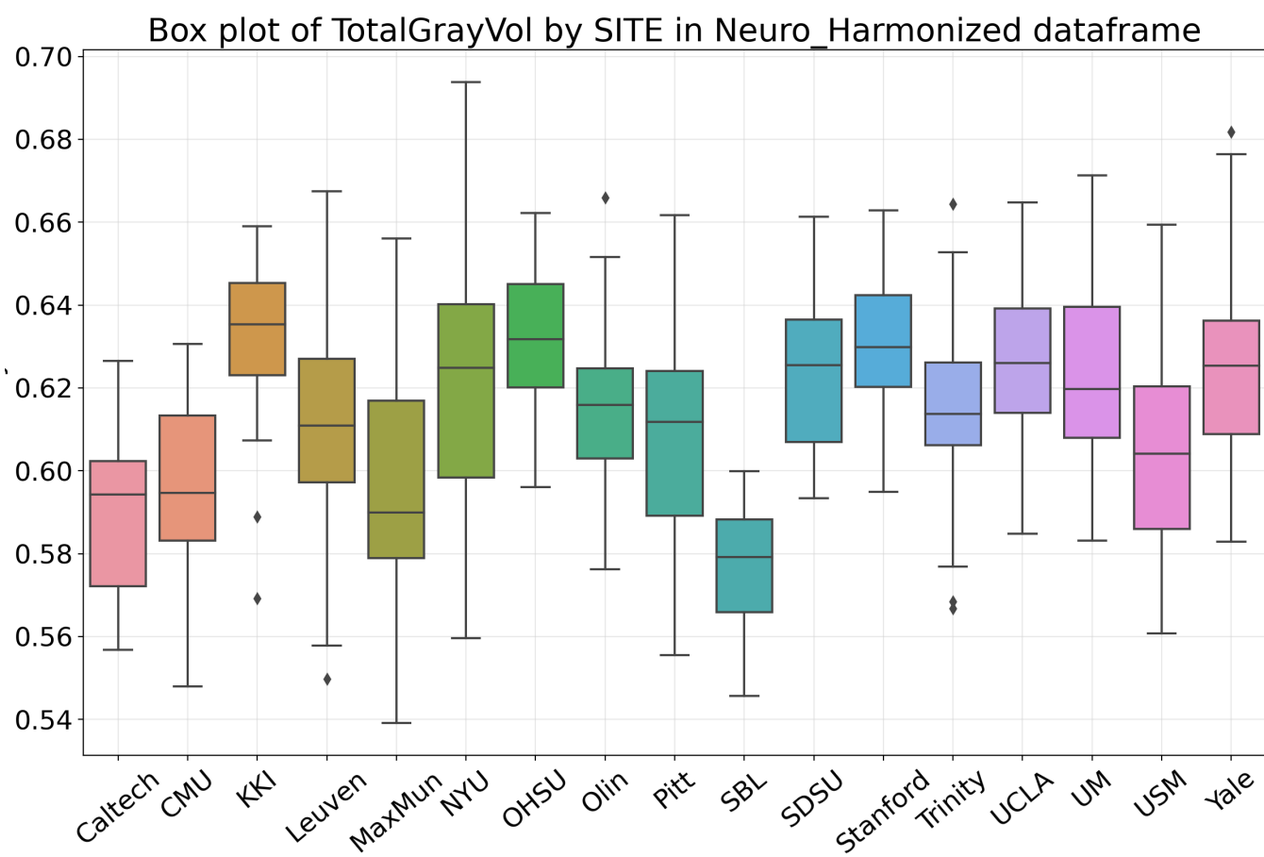
Limitations :centers the data to the overall, grand mean of all samples

→ adjusted data matrix shifted to an arbitrary location (no longer coincides with the location of any of the original centers)

→ harmonized features losing their original physical meanin

NeuroHarmonize model





MAIN

1. Define pipeline
2. Tune hyper-parameters with Grid Search (Cross Validation)
3. Refit a final model with the entire training dataset using the best hyper-parameters.
4. Save the best performing model

```python
def tune_model(dataframe_train, model, hyparams, harmonize_option):
    """docstring"""

    pipe = Pipeline(
        steps=[
            ("Feature", SelectKBest()),
            ("Scaler", RobustScaler()),
            ("Model", model),
        ]
    )

    x_train = dataframe_train.drop(
        ["AGE_AT_SCAN", "SEX", "DX_GROUP", "AGE_CLASS"], axis=1
    )
    y_train = dataframe_train["AGE_AT_SCAN"]
    y_train_class = dataframe_train["AGE_CLASS"]

    print("Cross validation for regression model")
    model_cv = GridSearchCV(
        pipe,
        cv=10,
        n_jobs=-1,
        param_grid=hyparams,
        scoring="neg_mean_absolute_error",
        verbose=True,
    )

    model_cv.fit(x_train, y_train)

    print("Best combination of hyperparameters:", model_cv.best_params_)

    crossvalidation = Crossvalidation()
    model_fit, MSE, MAE, PR = crossvalidation.stratified_k_fold(
        x_train, y_train, y_train_class, 10, model_cv.best_estimator_
    )
```

1

2

3

```python
    # Save the metrics in txt_file
    header = "MSE\t" + "MAE\t" + "PR\t"
    metrics = np.array([MSE, MAE, PR])
    metrics = np.array(metrics).T
    np.savetxt(
        "models/metrics/metrics_%s_%s.txt"
        % (model.__class__.__name__, harmonize_option),
        metrics,
        header=header,
    )

    # Save the best performing model fitted in stratifiedkfold cross validation
    with open(
        "models/%s_%s_pkl" % (model.__class__.__name__, harmonize_option), "wb"
    ) as files:
        pickle.dump(model_fit, files)


def predict_model(dataframe, model, harmonize_option):
    """docstring"""
    with open(
        "models/%s_%s_pkl" % (model.__class__.__name__, harmonize_option), "rb"
    ) as f:
        model_fit = pickle.load(f)
    x_test = dataframe.drop(["AGE_AT_SCAN", "SEX", "DX_GROUP", "AGE_CLASS"], axis=1)
    y_test = dataframe["AGE_AT_SCAN"]

    predict_y = model_fit.predict(x_test)
    predict_y = np.squeeze(predict_y)
    metric_test = np.array(
        [
            mean_squared_error(y_test, predict_y),
            mean_absolute_error(y_test, predict_y),
            pearsonr(y_test, predict_y)[0],
        ]
    )
    return predict_y, y_test, metric_test
```

4

PIPELINE AND TUNING

➤ **FEATURE SELECTION METHOD** : Best number of feature is chosen according to model and harmonization option

Model	N Not Harmonized	N Combat Harminized	N Neuro harmonized
MLP	50	100	50
LR	20	20	30
GR	20	20	20
RF	30	30	30
LA	20	30	30
SVR	20	30	30

➤ **ROBUTST SCALER**

- Features represent different kinds of quantities and have extremely different ranges of values →standardization is needed
- Robust Scaler:
  - ✳ Remove the median and scales according to the IQR
  - ✳ It is robust to outliers

Perché abbiamo fatto Scelta funzione di merito?

➤ **REGRESSION MODELS**

- Several regression model are compared
- Hyper-parameters are tuned for each model in cross validation

1.2 MLP

```python
class DeepRegression(BaseEstimator):
    """docstring"""

    def __init__(
        self,
        epochs=100,
        drop_rate=0.2,
        plot_loss=False,
    ):
        self.epochs = epochs
        self.drop_rate = drop_rate
        self.plot_loss = plot_loss
        super().__init__()

    def fit(self, X, y):
        """docstring"""
        inputs = Input(shape=X.shape[1])
        hidden = Dense(128, activation="relu")(inputs)
        hidden = Dense(12, activation="relu")(hidden)
        hidden = Dense(12, activation="relu")(hidden)
        hidden = Dropout(self.drop_rate)(hidden)
        hidden = Dense(12, activation="relu")(hidden)
        outputs = Dense(1, activation="linear")(hidden)

        self.model = Model(inputs=inputs, outputs=outputs)
        self.model.compile(
            loss="mean_absolute_error", optimizer="adam", metrics=["MAE"]
        )
        self.model.summary()
        # Define callbacks
        early_stopping = tf.keras.callbacks.EarlyStopping(
            monitor="val_loss", patience=10, verbose=1
        )
        history = self.model.fit(
            X,
            y,
            validation_split=0.3,
            epochs=self.epochs,
            batch_size=32,
            callbacks=[early_stopping],
            verbose=0,
        )
```

Class inherits from Base Estimator and implements fit and predict methods→can be used in Pipeline

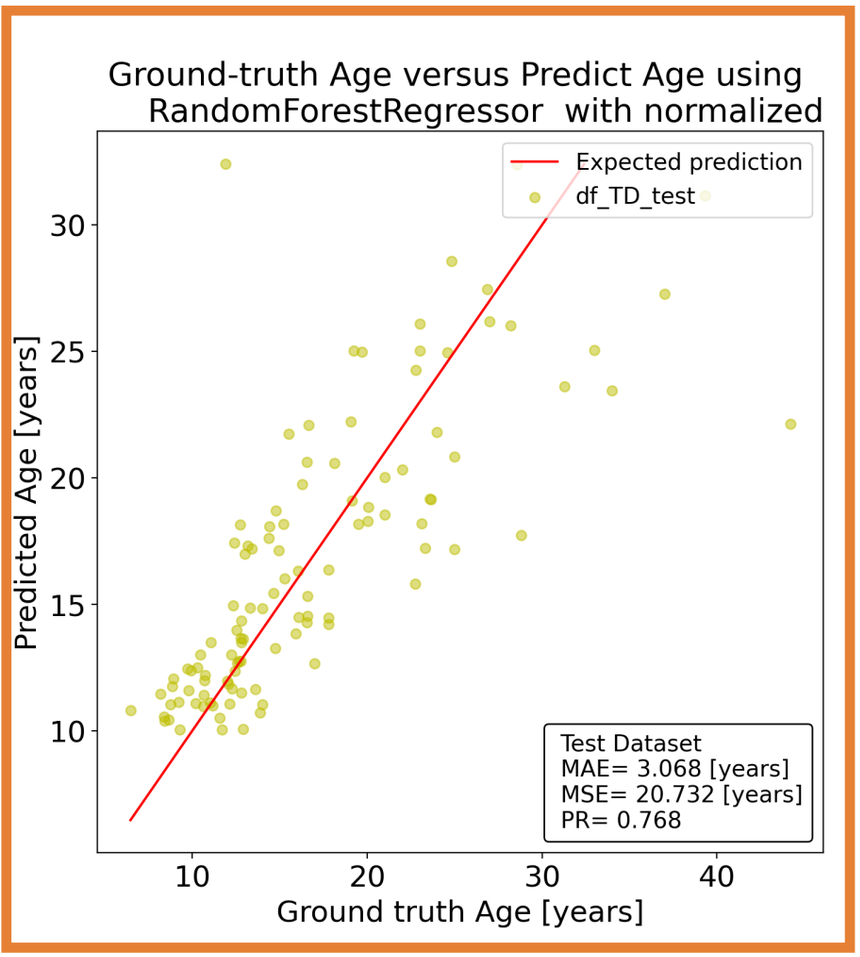
CROSS VALIDATION

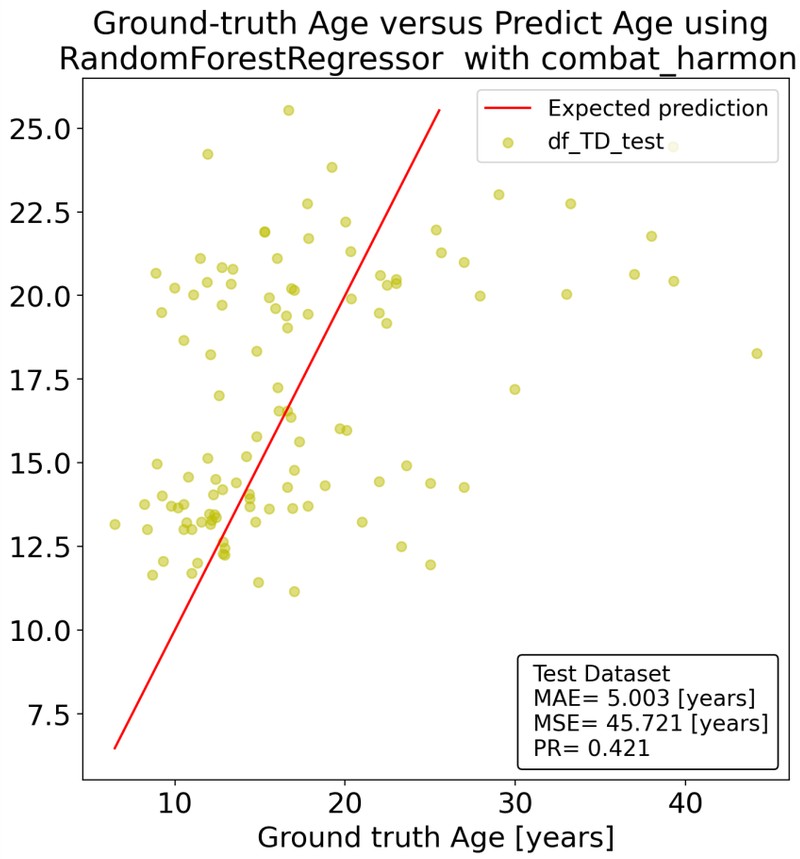
```
def stratified_k_fold(self, X, y, y_bins, n_splits, model):
    """docstring"""

    try:
        y = y.to_numpy()
        y_bins = y_bins.to_numpy()
        X = X.to_numpy()
    except AttributeError:
        pass
    cv = StratifiedKFold(n_splits)

    MSE = []
    MAE = []
    PR = []
    for train_index, validation_index in cv.split(X, y_bins):
        predict_y = model.fit(X[train_index], y[train_index]).predict(X[validation_index])
        print(f"MAE: {mean_absolute_error(y[validation_index], predict_y):0.3f}")

        MSE.append(mean_squared_error(y[validatio
        MAE.append(mean_absolute_error(y[validati
        PR.append(pearsonr(y[validation_index], p

        predict_y = np.squeeze(predict_y)
    return (model,MSE,MAE,PR,)
```

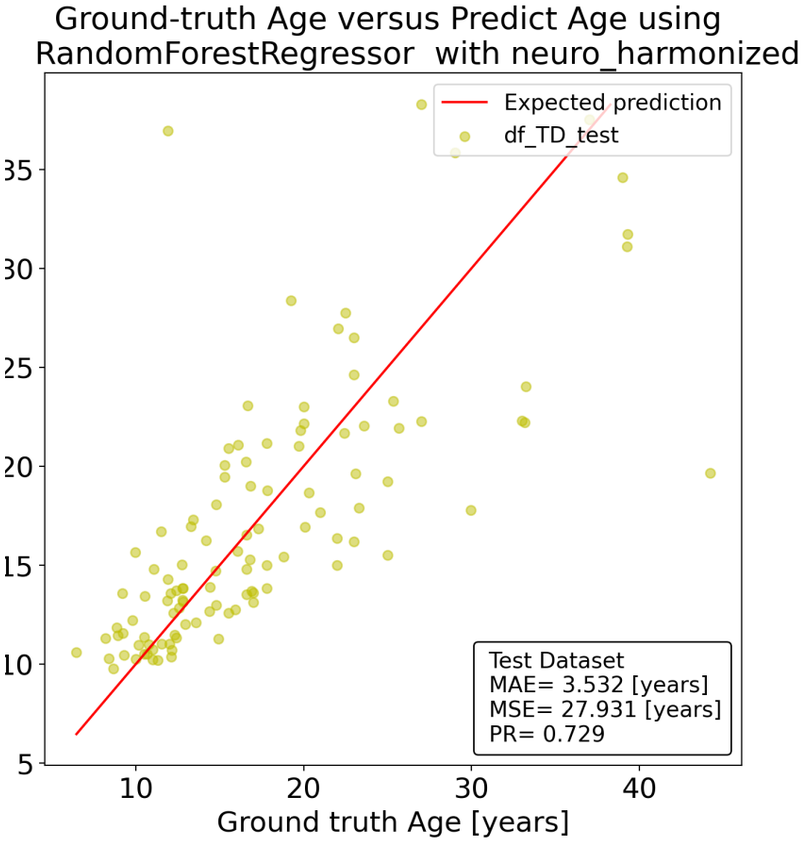
- Random Forest present overall best performances for all harmonization options
- Not harmonizing data seems to improves the performance in terms of metric

Algorithm	MSE	MAE	ρ	
Deep Regression	31.12 ± 10.62	3.98 ± 0.53	0.766 ± 0.07	Not Harmonized
	76.34 ± 29.40	6.04 ± 0.944	0.322 ± 0.152	Combat Harmonized
	35.29 ± 6.84	4.523 ± 0.317	0.785 ± 0.064	Neuro Harmonized
Linear Regression	20.63 ± 8.99	3.30 ± 0.47	0.809 ± 0.053	Not Harmonized
	48.78 ± 15.72	5.102 ± 0.736	0.501 ± 0.089	Combat Harmonized
	23.85 ± 8.55	3.540 ± 0.510	0.801 ± 0.0521	Neuro Harmonized
Gaussian Process Regressor	63.60 ± 11.2	6.22 ± 0.49	0.626 ± 0.082	Not Harmonized
	130.5 ± 34.48	8.86 ± 0.940	0.318 ± 0.185	Combat Harmonized
	91.95 ± 22.01	7.456 ± 0.959	0.517 ± 0.137	Neuro Harmonized
Random Forest Regressor	18.19 ± 6.40	2.91 ± 0.33	0.828 ± 0.052	Not Harmonized
	48.64 ± 16.23	5.025 ± 0.673	0.495 ± 0.122	Combat Harmonized
	21.74 ± 7.84	3.28 ± 0.46	0.821 ± 0.049	Neuro Harmonized
Lasso	20.48 ± 7.38	3.186 ± 0.39	0.808 ± 0.043	Not Harmonized
	48.13 ± 15.99	5.05 ± 0.72	0.507 ± 0.096	Combat Harmonized
	22.57 ± 7.73	3.458 ± 0.447	0.812 ± 0.0285	Neuro Harmonized
SVR	23.56 ± 8.37	3.177 ± 0.307	0.800 ± 0.0528	Not Harmonized
	51.89 ± 17.87	4.85 ± 0.628	0.506 ± 0.096	Combat Harmonized
	23.83 ± 8.47	3.465 ± 0.505	0.807 ± 0.039	Neuro Harmonized

TEST

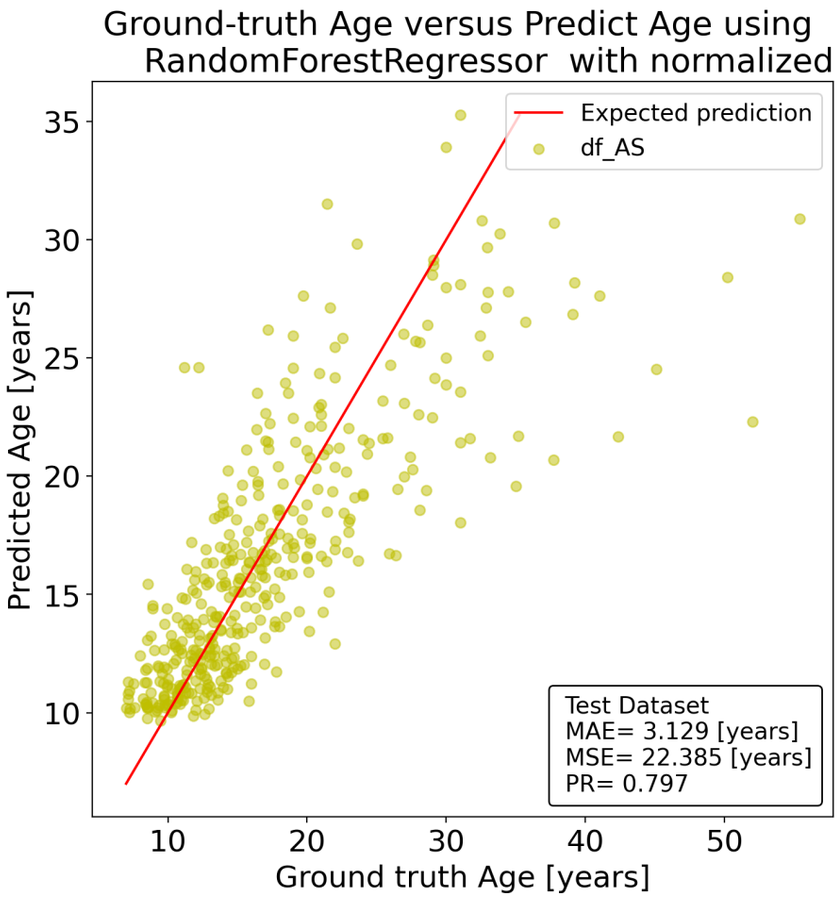


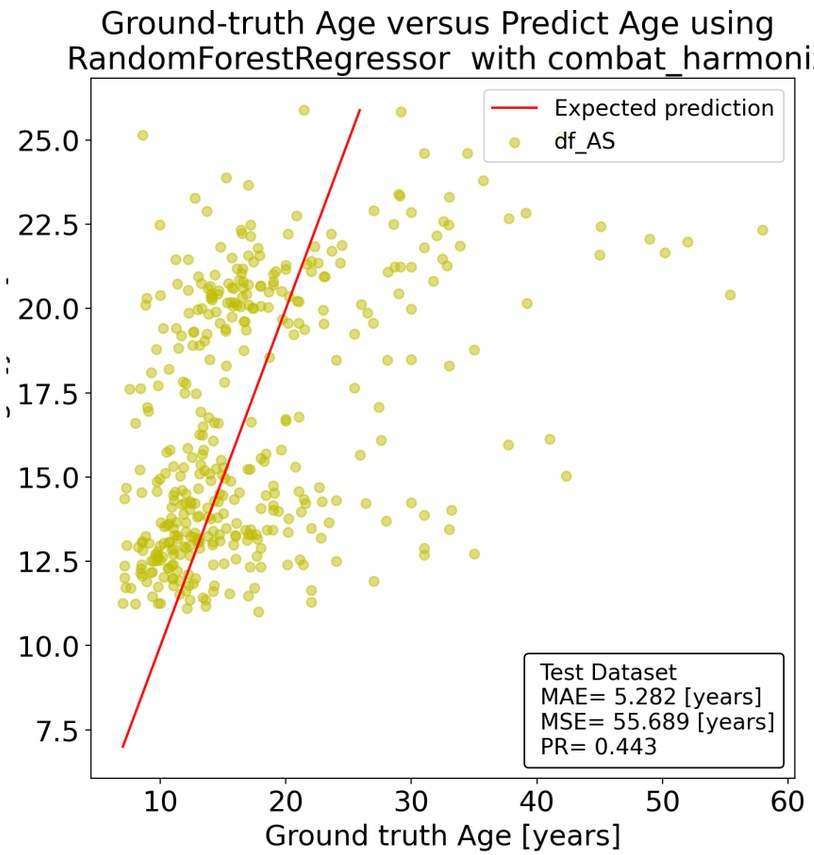


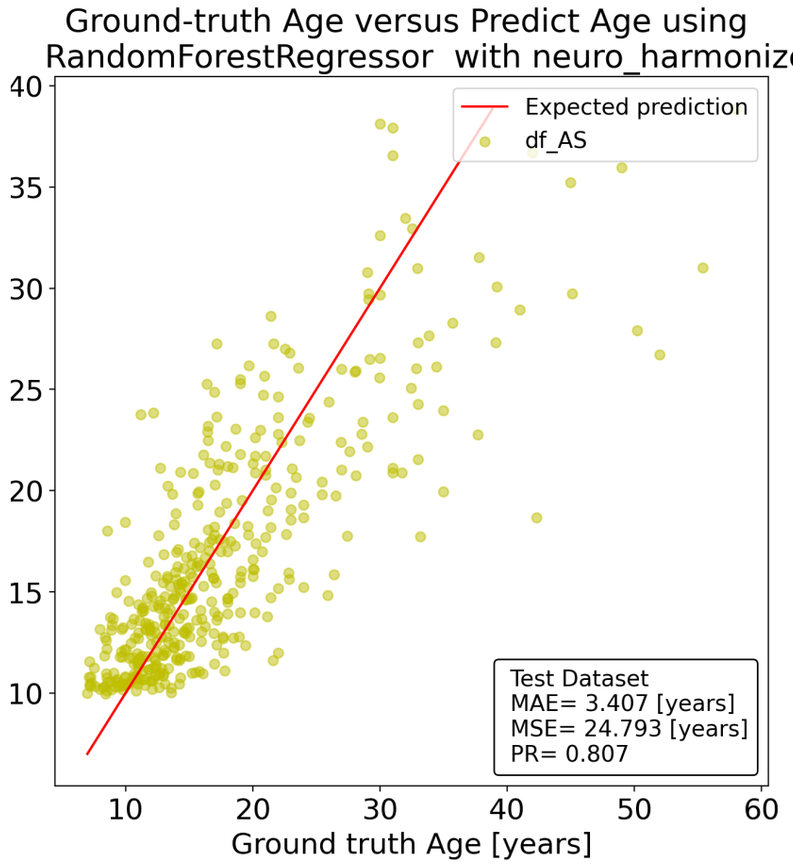


Dataset	MSE	MAE	ρ
Training	18.19	2.91	0.828
Test	20.732	3.068	0.768

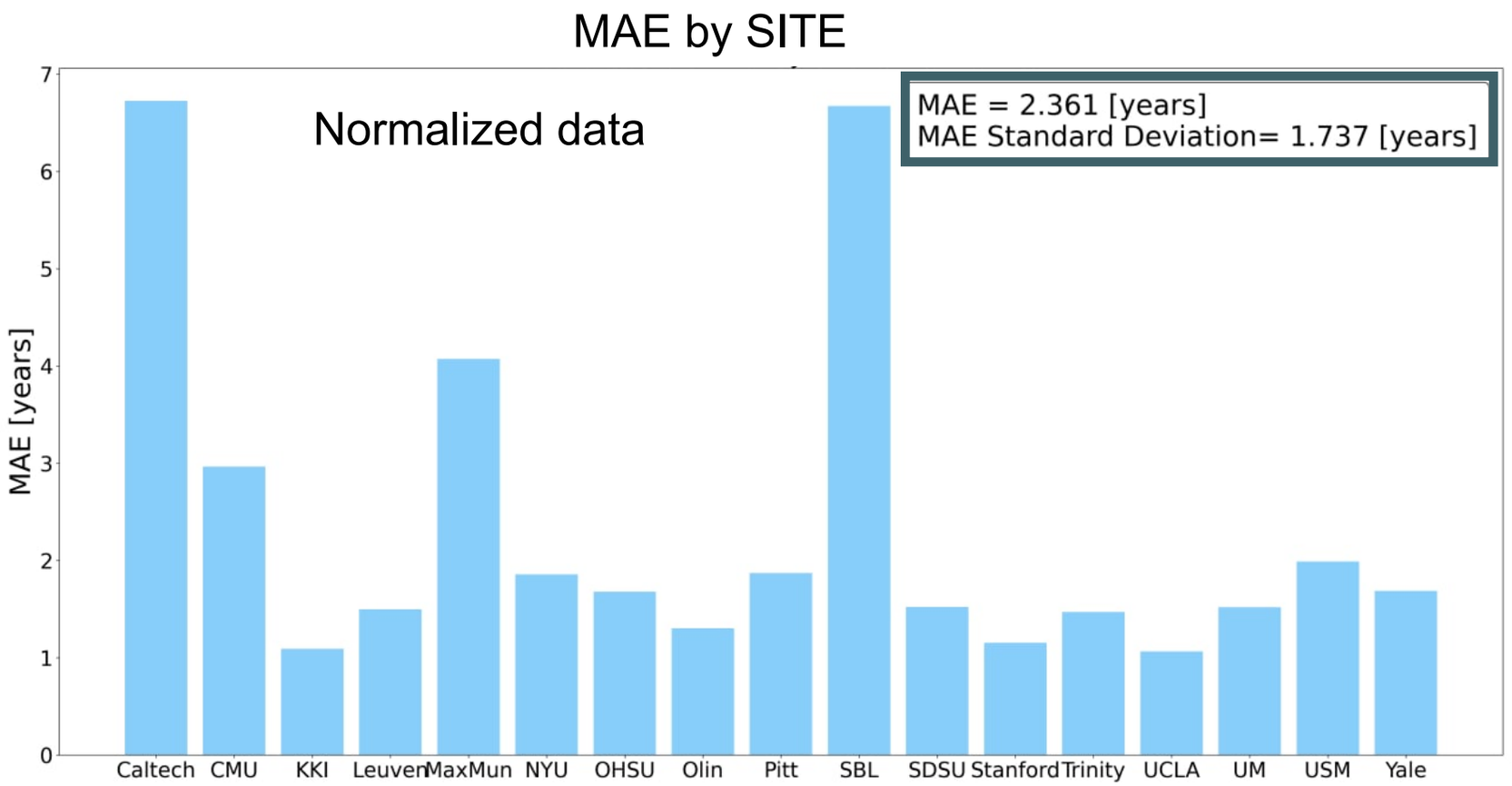
AUTISTIC SUBJECTS

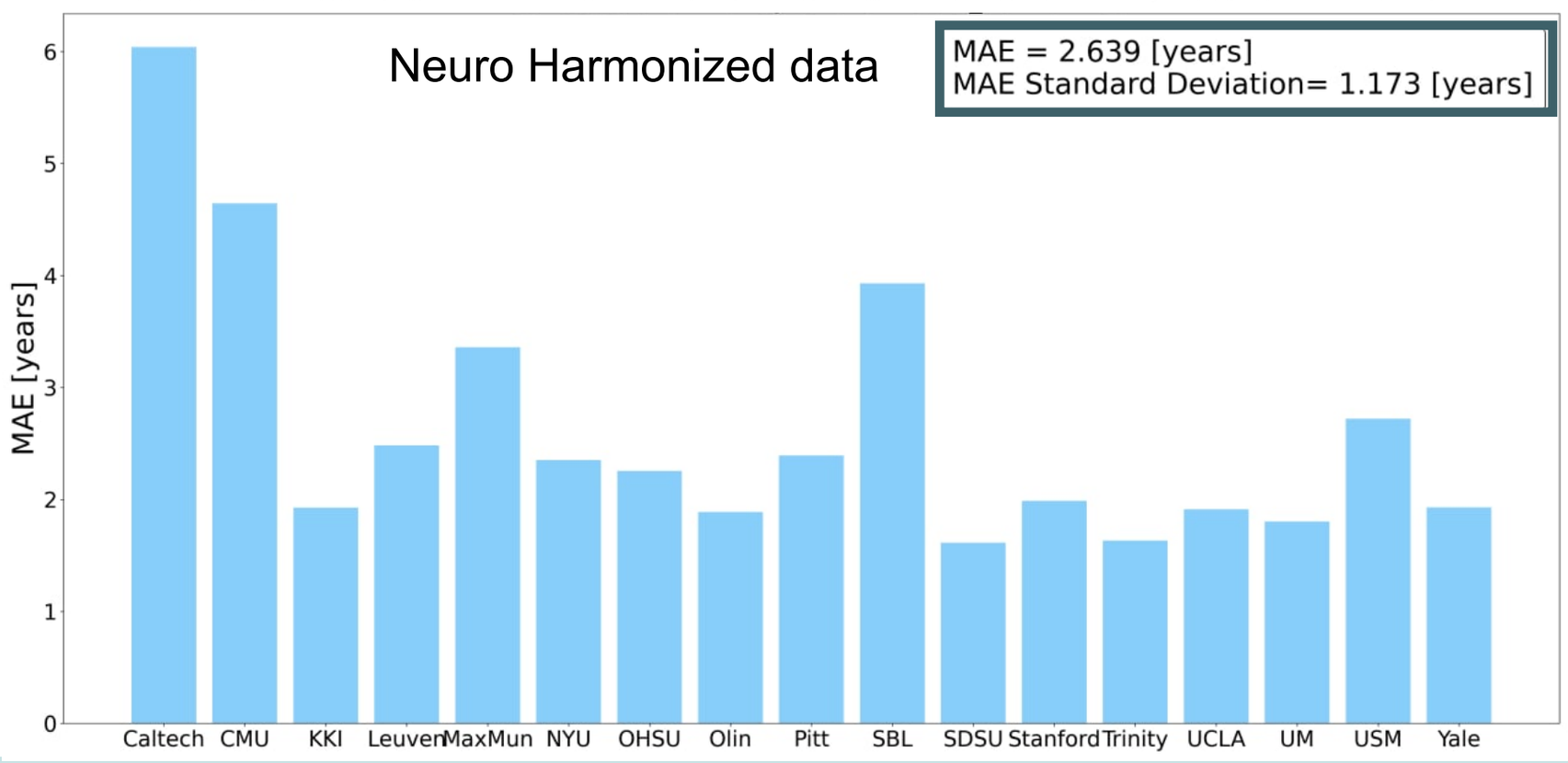






REPRODUCIBILITY





- Entire control dateset has been used wit Random Forest model
- Harmonizing data with neuro harmonize improves the reproducibility over different sites

FUTURE IDEAS AND IMPROVEMENTS

Pesare le cose che diamo alla rete , perché alcune classi sono sottorappresentate