BRAIN AGE PREDICTION WITH REGRESSION MODELS

Angela Corvino e Agata Minnocci

Esame di CMEPDA, 24 Giugno 2022

CODE REPOSITORY

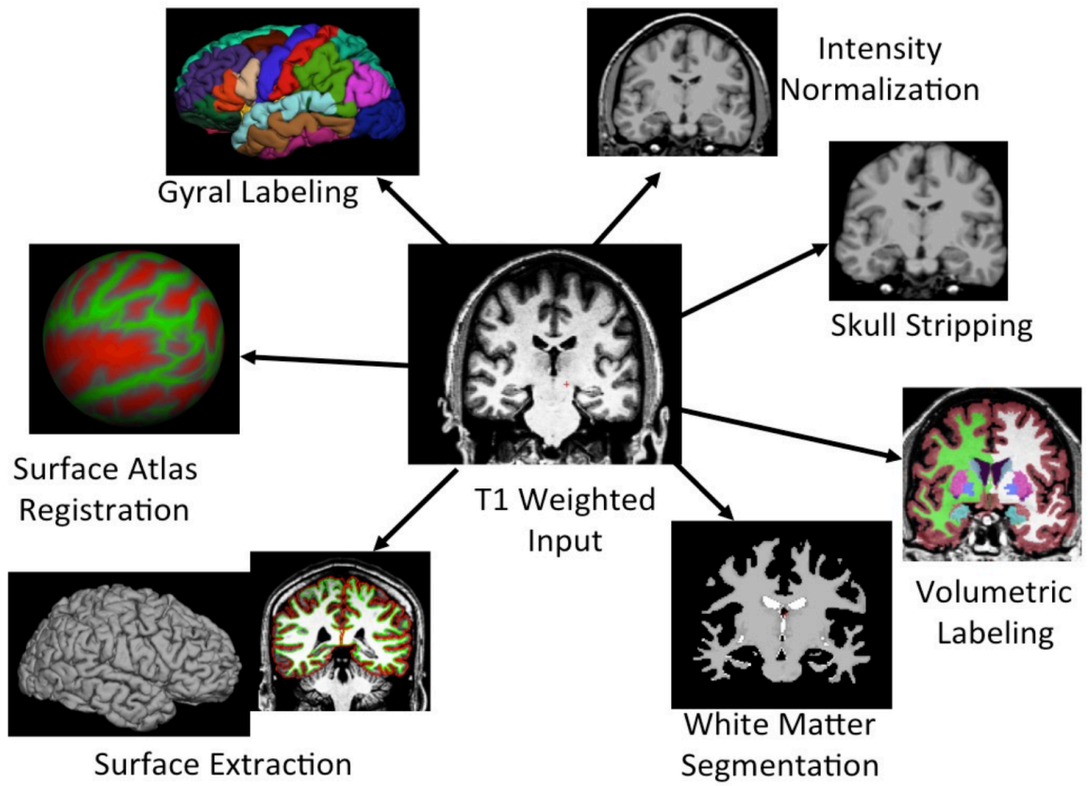
Repository

Documentation

Continuous integration

DATA SEGMENTATION

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



- 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

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

Dataframe

	FILE_ID	AGE_AT_SCAN	SEX	FIQ	DX_GROUP	...	lhCortexVol	rhCortexVol	lhCerebralWhiteMatterVol	rhCerebralWhiteMatterVol	TotalGrayVol
0	Caltech_0051456	55.40	1	126.0	1	...	258703.4443	262670.1207	262254.9768	265741.1295	701211.5651
1	Caltech_0051457	22.90	1	107.0	1	...	321132.1284	317004.7945	304528.2513	301555.0187	841485.9229
2	Caltech_0051458	39.20	1	93.0	1	...	266418.9917	266456.1701	271683.1723	266494.7088	726728.1619
3	Caltech_0051459	22.80	1	106.0	1	...	307157.4469	310540.0389	258618.7918	255480.2694	794676.4859
4	Caltech_0051461	37.70	1	99.0	1	...	263932.5269	269417.2519	242330.5838	232809.9268	714122.7788
..	...	...	...	...	...	...	...	...	...	...	...
910	Yale_0050618	12.75	1	76.0	1	...	192826.5499	195519.7899	181671.5502	178748.1592	550196.3398
911	Yale_0050622	9.92	1	89.0	1	...	252293.1567	257263.1314	210779.7950	210714.3714	691881.2881
912	Yale_0050625	7.00	1	99.0	1	...	264309.1936	268447.9865	186742.5963	189804.7237	694421.1801
913	Yale_0050626	11.08	1	61.0	1	...	296770.5313	309780.9732	255228.1165	260365.0738	803490.5045
914	Yale_0050628	14.42	1	77.0	1	...	227194.7780	238810.8206	219704.4479	221946.7201	627204.5987

PREPROCESSING.PY

1. Adding/removing features
2. Choosing harmonization option and harmonizing data
3. Splitting dataset in cases and controls (only controls will be used for training)

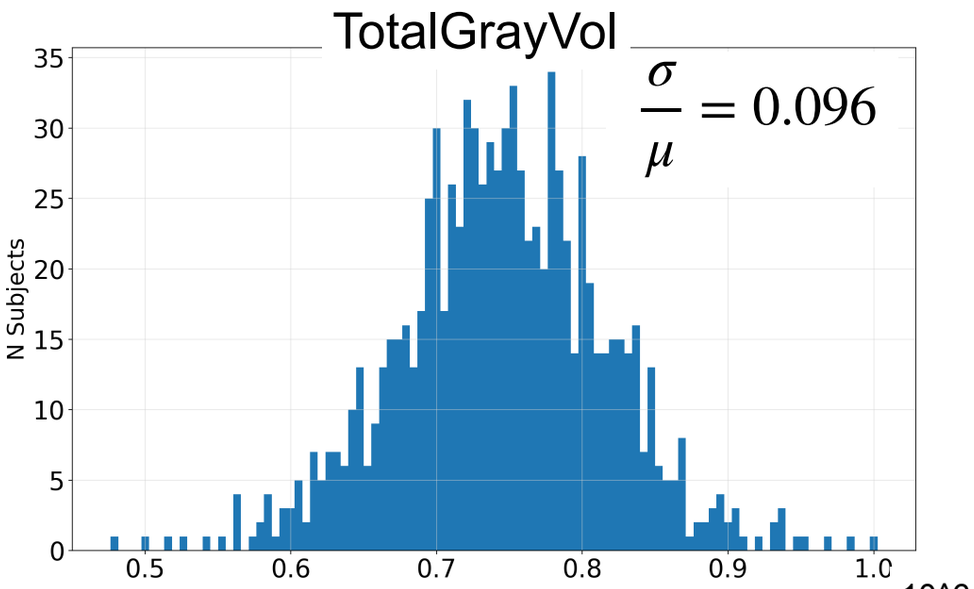
```python
def __call__(self, dataframe, prep_option, plot_option=True):
    """docstring"""
    self.add_TotalWhiteVol(dataframe)
    self.add_age_binning(dataframe)
    self.add_site(dataframe)
    dataframe= self.add_site_binning(dataframe)
    dataframe = self.remove_FIQ(dataframe)
    # PLOTTING DATA
    if plot_option is True:
        self.plot_histogram(dataframe, "AGE_AT_SCAN")
    # PROCESSING DATA
    if prep_option == "not_normalized":
        print("Dataframe is not normalised")
    elif prep_option == "normalized":
        self.self_normalize(dataframe)
        print("Dataframe is only normalized")
    elif prep_option == "combat_harmonized":
        self.self_normalize(dataframe)
        try:
            assert "SITE_CLASS" in dataframe.keys(), "There is no site class"
        except AssertionError as msg:
            print(msg)
        dataframe_combat = self.com_harmonize(
            dataframe,
            confounder="SITE_CLASS",
            covariate="AGE_AT_SCAN",
        )
        try:
            assert (
                np.sum(np.sum(dataframe.isna())) == 0
            ), "There are NaN values in the dataframe!"
        except AssertionError as msg:
            print(msg)
        dataframe = dataframe_combat
        print("Dataframe is normalized and harmonized with NeuroCombat")

    elif prep_option == "neuro_harmonized":
        self.self_normalize(dataframe)
        dataframe_neuro = self.neuro_harmonize(
            dataframe,
            confounder="SITE",
            covariate="AGE_AT_SCAN",
        )
        dataframe = dataframe_neuro
        print("Dataframe is normalized and harmonized with NeuroHarmonize")

    dataframe = dataframe.drop(["FILE_ID", "SITE_CLASS"], axis=1)
    return dataframe
```

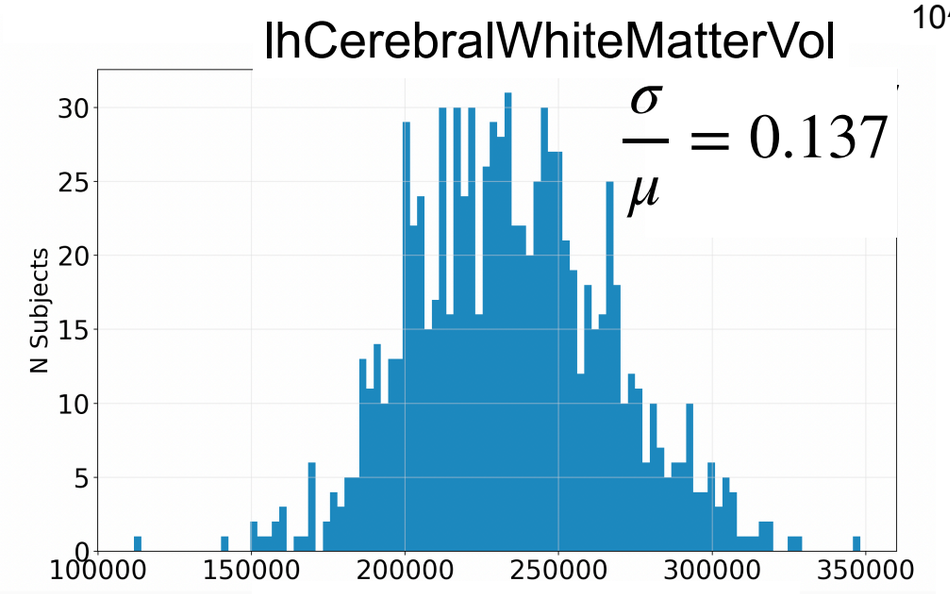
1

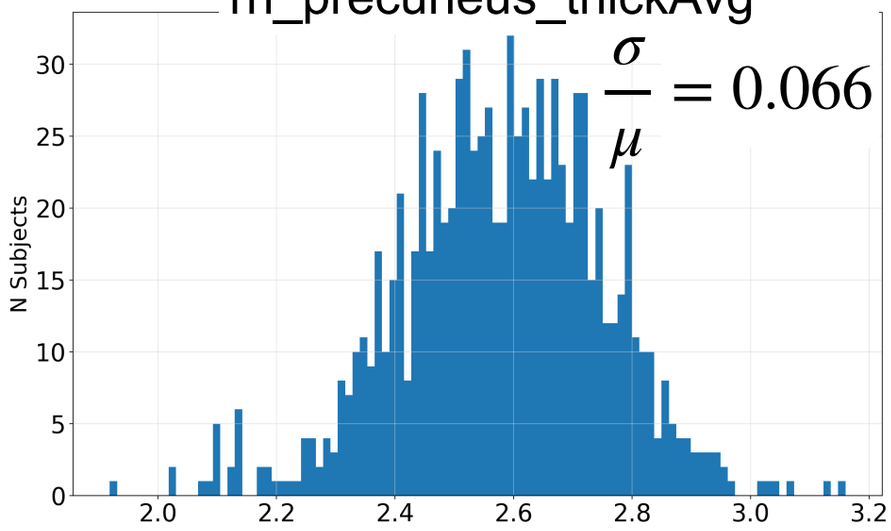
2

```python
def self_normalize(self, dataframe):
    """docstring"""
    # SURFACE
    column_list = dataframe.loc[
        :, ["SurfArea" in i for i in dataframe.columns]
    ].columns.tolist()
    dataframe.loc[:, ["SurfArea" in i for i in dataframe.columns]] = dataframe.loc[
        :, ["SurfArea" in i for i in dataframe.columns]
    ].divide(dataframe[column_list].sum(axis=1), axis=0)

    # THICKNESS
    dataframe.loc[:, ["ThickAvg" in i for i in dataframe.columns]] = dataframe.loc[
        :, ["ThickAvg" in i for i in dataframe.columns]
    ].divide(
        (dataframe["lh_MeanThickness"] + dataframe["rh_MeanThickness"]), axis=0
    )

    # VOLUME
    dataframe.loc[:, ["Vol" in i for i in dataframe.columns]] = dataframe.loc[
        :, ["Vol" in i for i in dataframe.columns]
    ].divide((dataframe["TotalGrayVol"] + dataframe["TotalWhiteVol"]), axis=0)
```

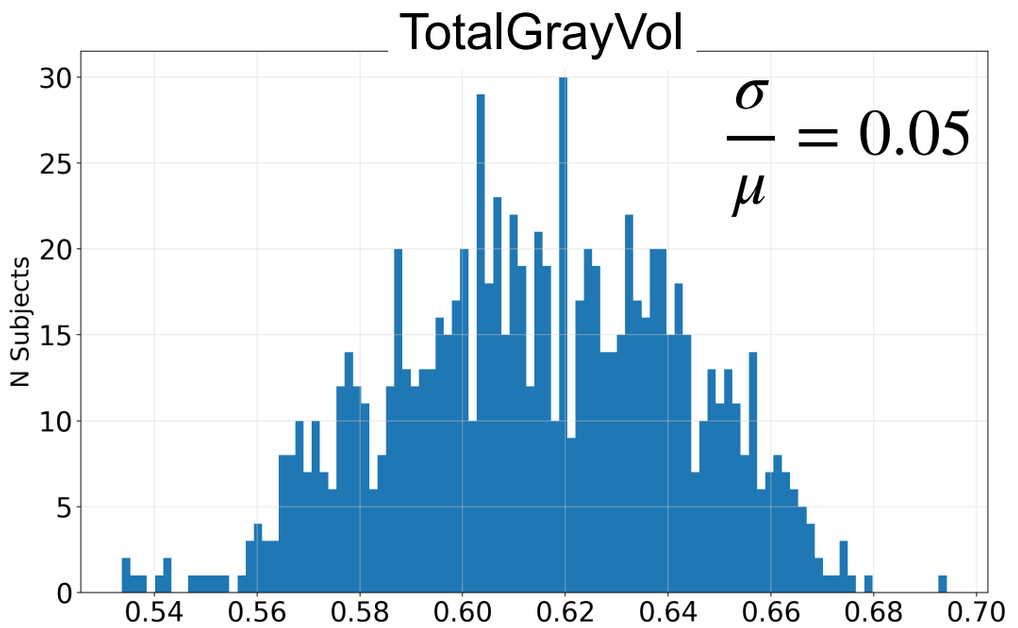
SELF NORMALIZATION

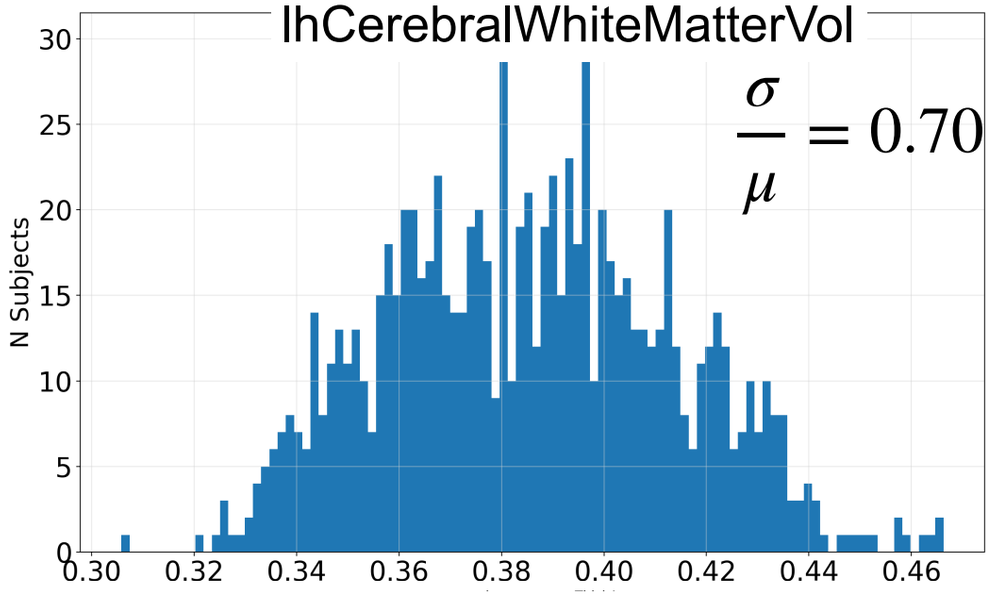


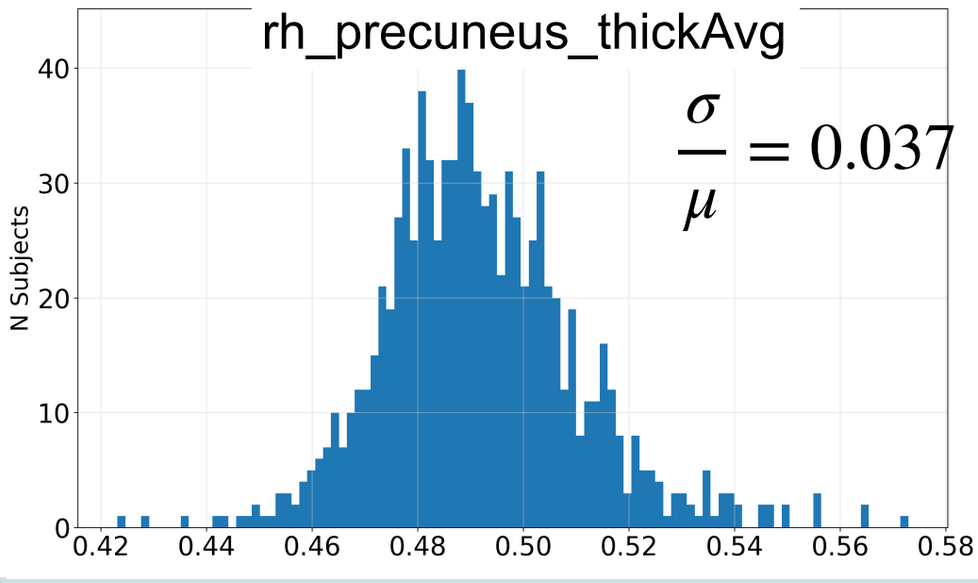




Normalized Variance Reduction





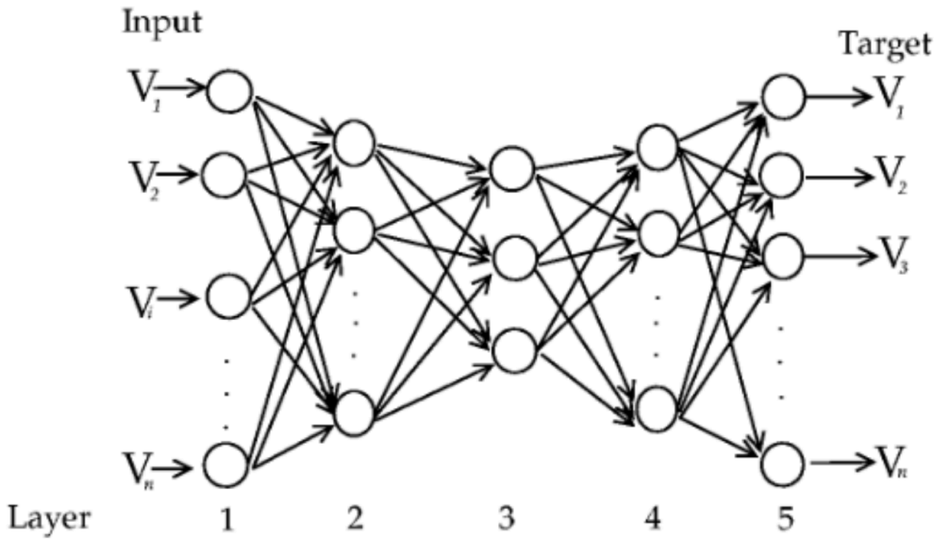


RNN FOR OUTLIERS DETECTION

RNN (feed-forward multi-layer perceptron)

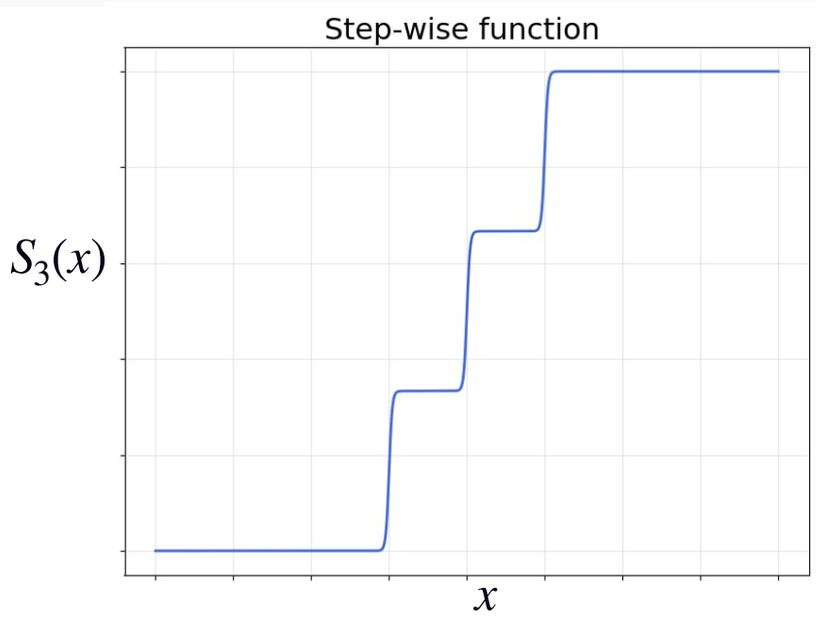
- 3 hidden layers sandwiched between input layer and output layer
- step-wise activation function for the middle hidden layer (k = 3)

```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
```

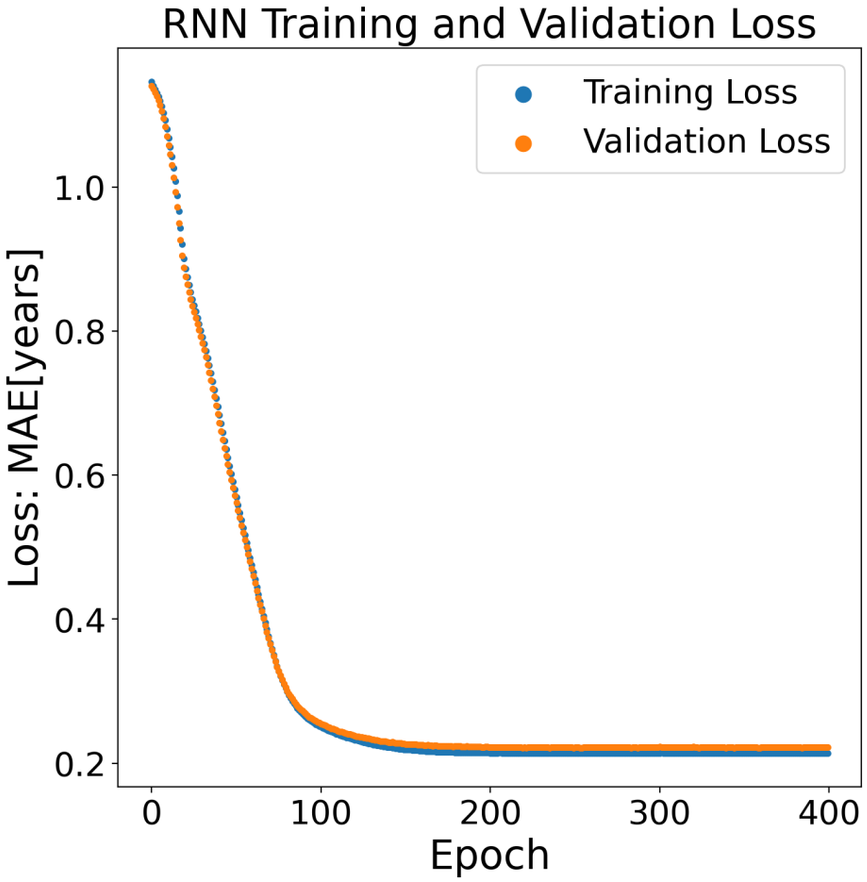



```python
def step_wise(x, N=4, a=100):
    """docstring"""
    y = 1 / 2
    for j in range(1, N):
        y += (1 / (2 * (N - 1))) * (K.tanh(a * (x - (j / N))))
    return y
```

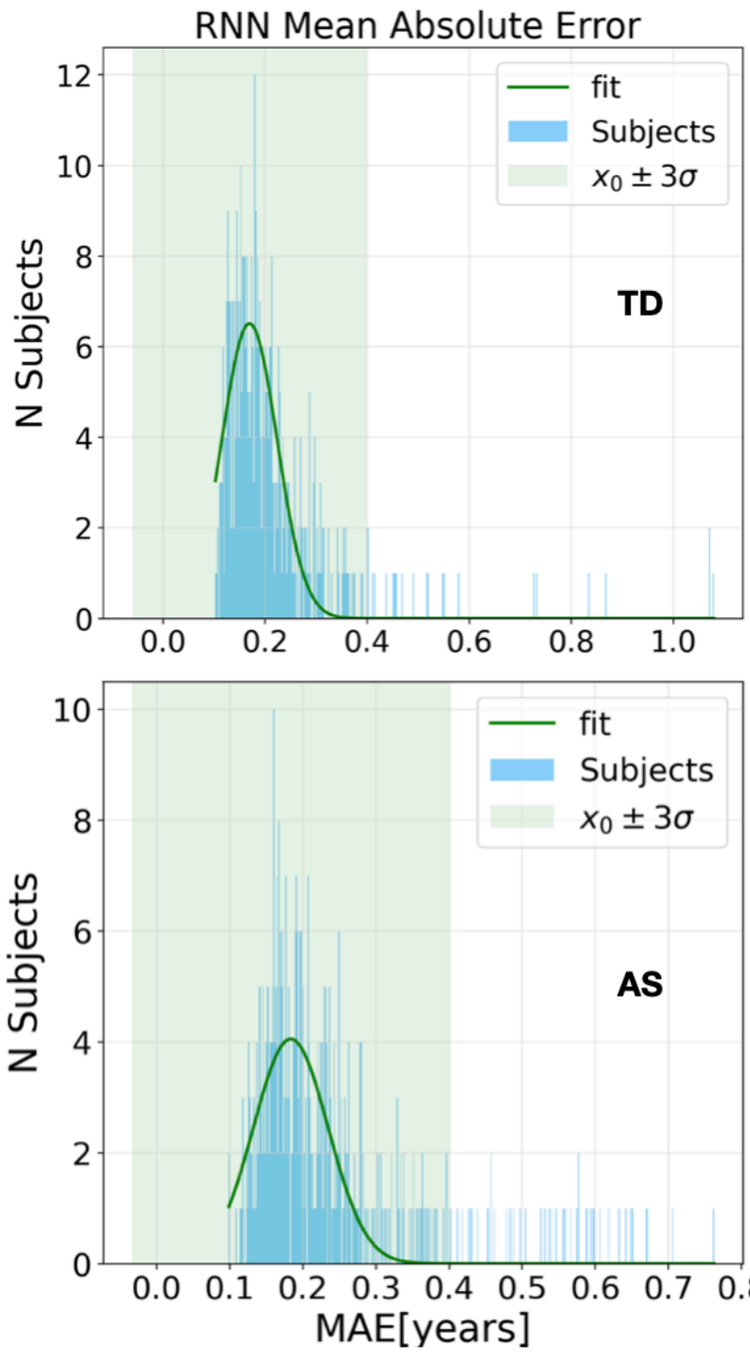
```python
def fit_autoencoder(self, model, epochs):
    """docstring"""
    # Define callbacks
    early_stopping = tf.keras.callbacks.EarlyStopping(
        monitor="val_loss", patience=10, verbose=1
    )

    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
```

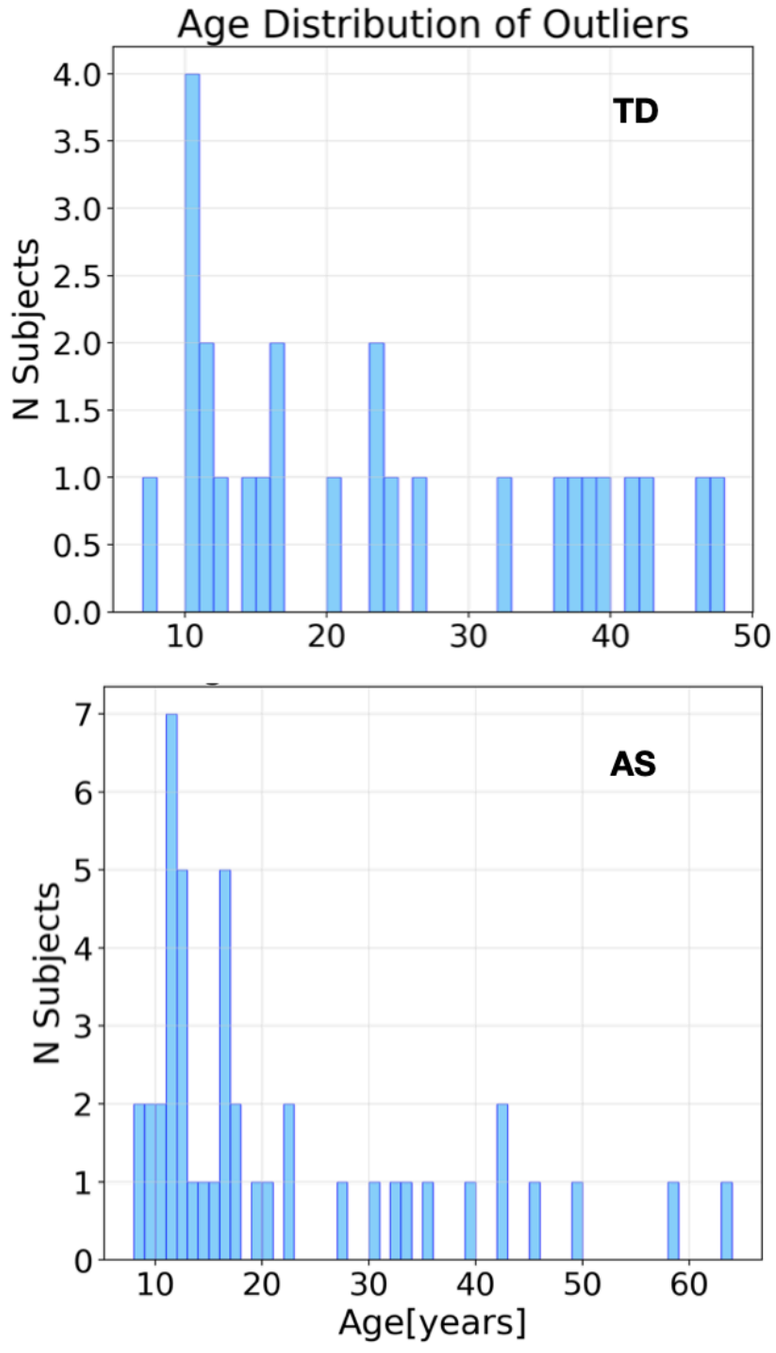


OUTLIERS DETECTION: OUTLIERS.PY



RNN performances on TD

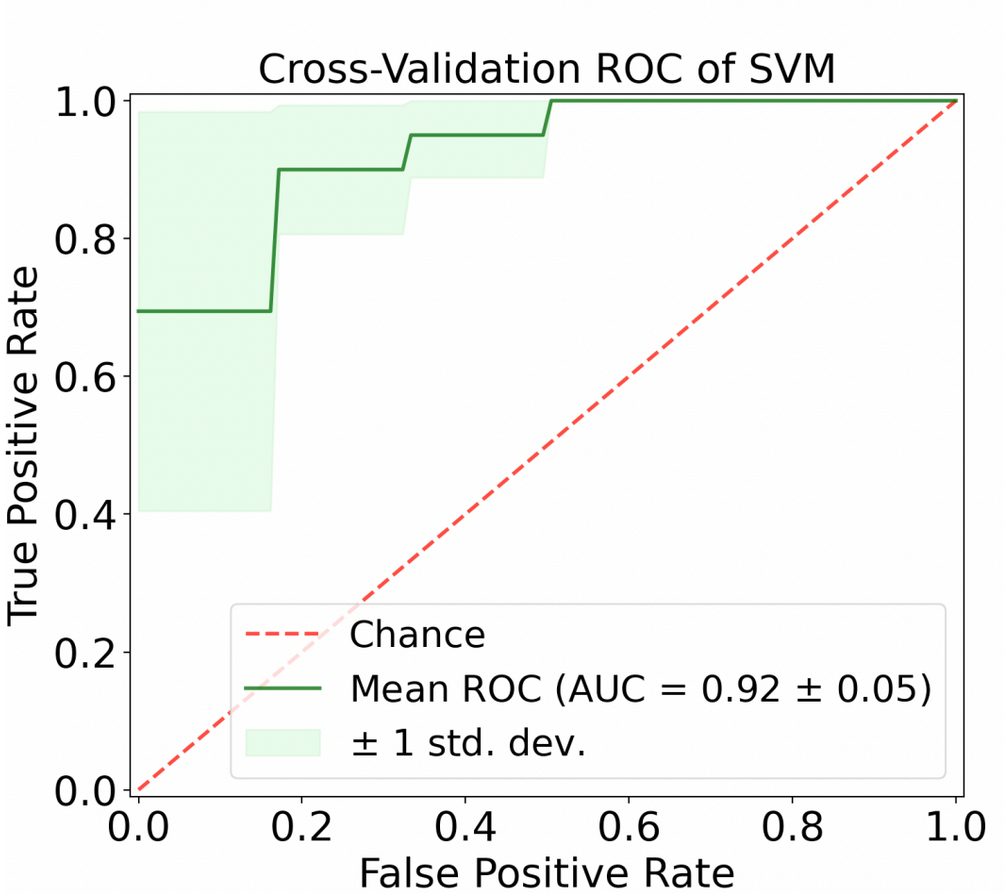




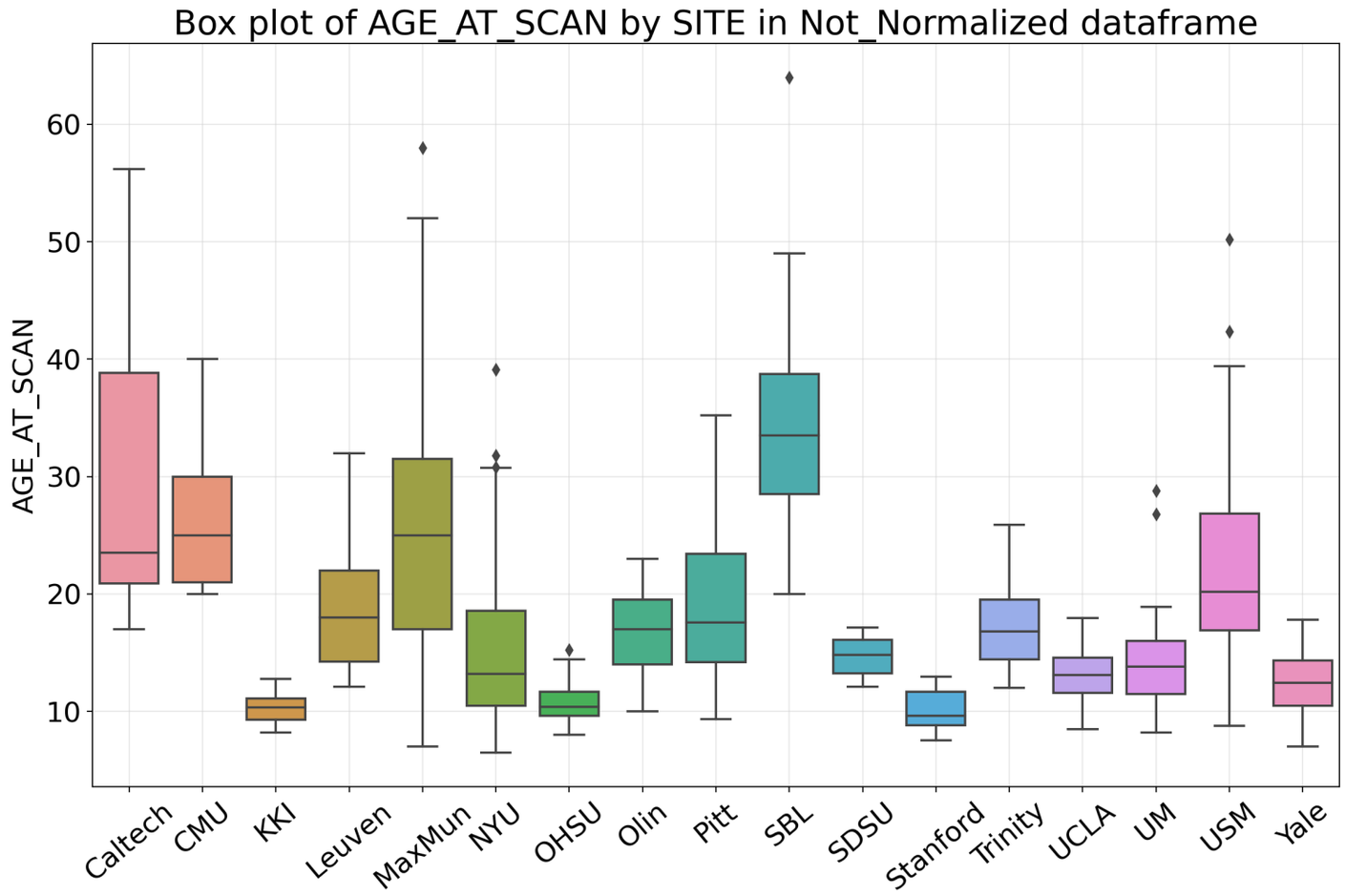
CONFOUNDER VARIABLES: SITE

Confounders:

- characteristics of the samples that are not clinically relevant, but can mislead the training process
- can be identified with a classification study

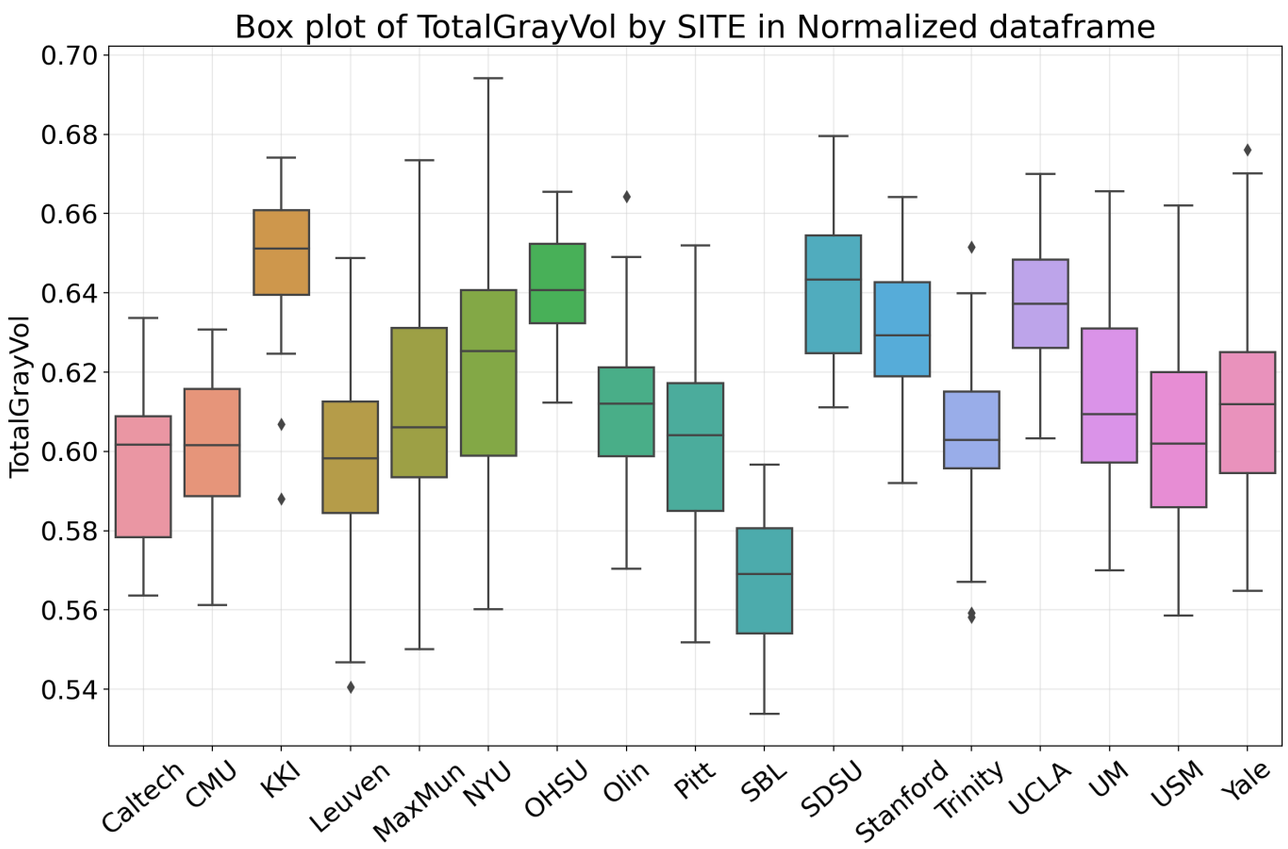


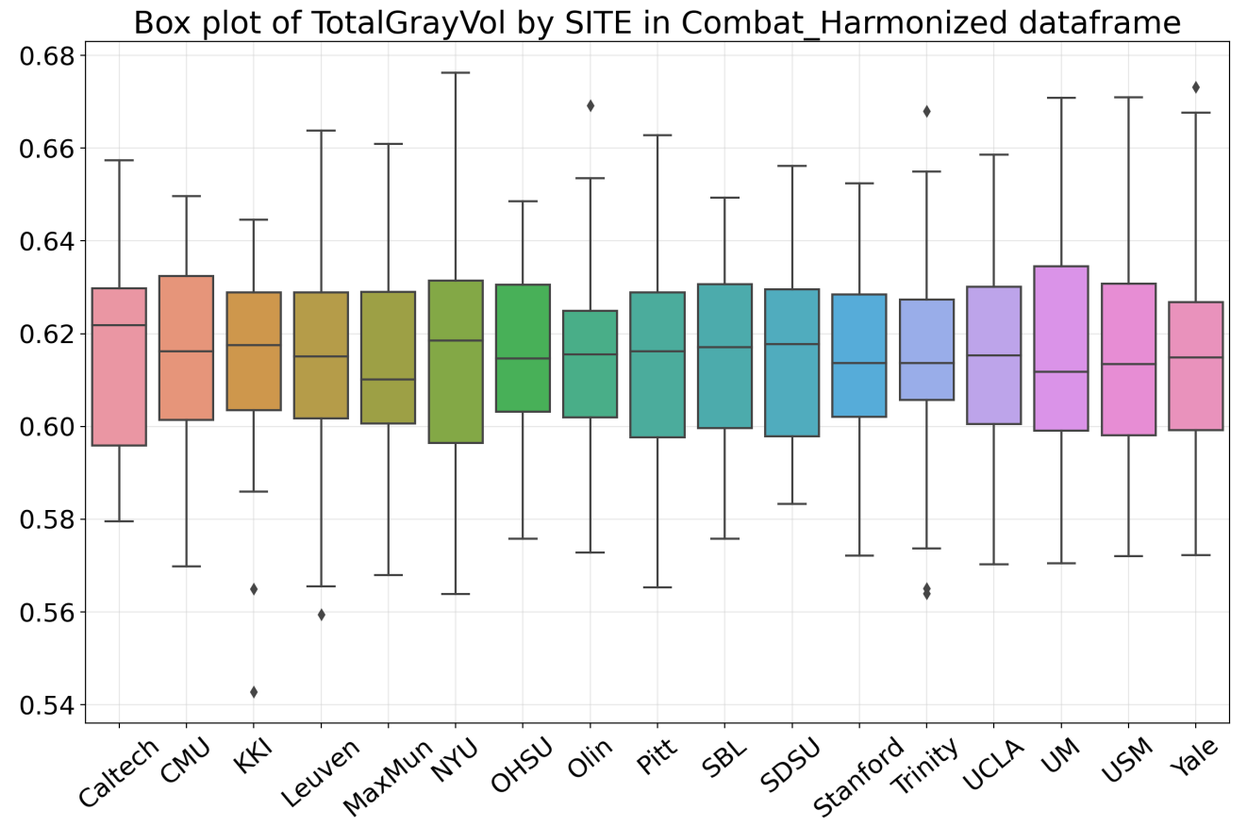
Classification study for subjects belonging to two different sites



Effects non-biological in nature and associated with differential scanning equipment or parameter configurations increase bias and variance in the measurement of brain features.

DATA HARMONIZATION: NEUROCOMBAT MODEL





Characteristics

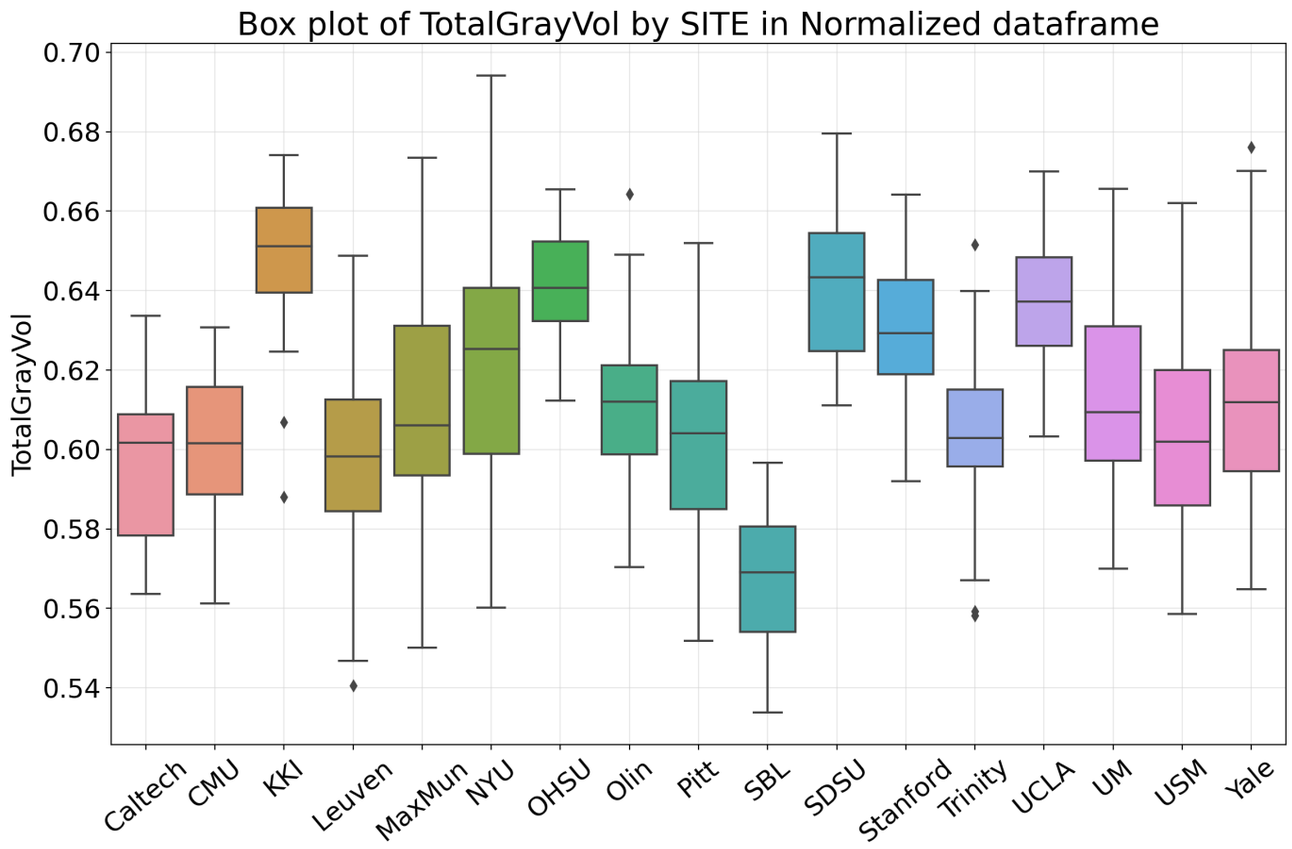
- Models site-specific scaling factors and uses empirical Bayes to improve the estimation of the site parameters for small sample sizes.
- (Should be) robust to unbalanced studies, in which the biological covariate of interest is not balanced across sites.
- Removes unwanted variation associated with site and preserves biological associations in the data.

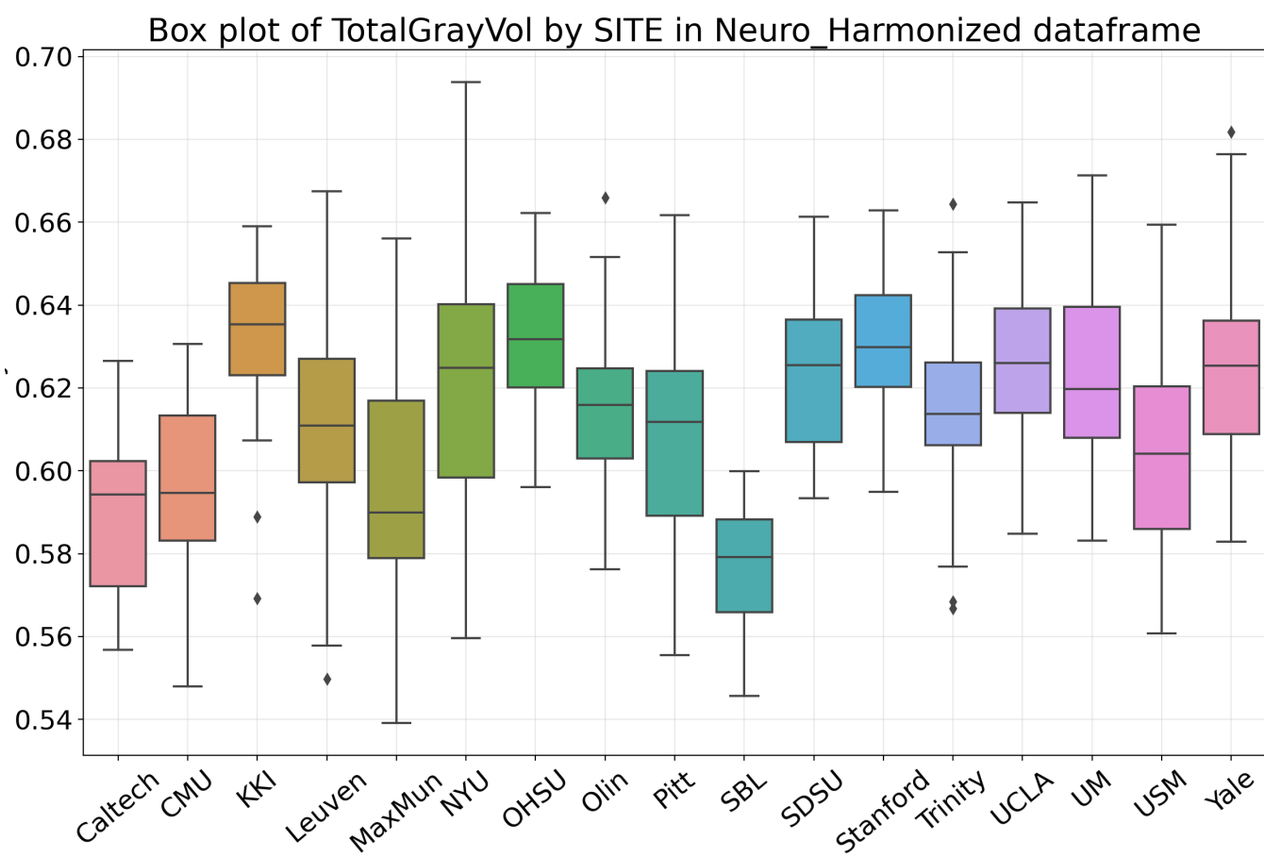
Limitations: centers the data to the overall, grand mean of all samples

- Can only harmonize by a single confounding variable at a time.
- Adjusted data matrix shifted to an arbitrary location

→ harmonized features losing their original physical meaning, impossible values

DATA HARMONIZATION: NEUROHARMONIZE MODEL





Characteristics

- Combat + (GAM) with a penalized nonlinear term to describe age effect
- Can harmonize by multiple batch effects (confounding variables) at a time. (We only use one)

REGRESSION MODELS

Comparing different regression models:

```
models = [
    DeepRegression(plot_loss=False),
    LinearRegression(),
    GaussianProcessRegressor(),
    RandomForestRegressor(),
    Lasso(),
    SVR()
]
```

MSE (Mean Squared Error)

- tells you how close a regression line is to a set of points (distance is the root)

$$\mathrm{MSE} = \frac{\sum_{i=1}^{n} (x_i - y_i)^2}{n}$$

MAE (Mean Absolute Error) is also used as loss function

- measures the average magnitude of the residuals between chronological ages and predicted ages.

$$\mathrm{MAE} = \frac{\sum_{i=1}^{n} |x_i - y_i|}{n}$$

Pearson correlation coefficient

- measure of linear correlation between predicted age and chronological age
- higher Pearson correlation coefficient ρ means better predictive performance.

$$r = \frac{\sum_{i=1}^{n} (x_i - \overline{x})(y_i - \overline{y})}{\sqrt{\sum_{i=1}^{n} (x_i - \overline{x})^2} \sqrt{\sum_{i=1}^{n} (y_i - \overline{y})^2}}$$

MAIN: PREDICTAGE.PY

1. Define pipeline
2. Tune hyper-parameters with Grid Search (Cross Validation)
3. Fit 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 using SelectKBest (from sklearn)

Model	N Not Harmonized	N Combat Harminized	N Neuro harmonized
MLP	128	128	128
LR	20	20	30
GR	20	20	20
RF	30	30	30
LA	20	30	30
SVR	20	30	30

➤ **ROBUST SCALER**

- Features represent different kinds of quantities and have extremely different ranges of values → standardization is needed
- Robust Scaler:
  - ✳ It is robust to outliers
  - ✳ Remove the median and scales according to the IQR. $\frac{x - \hat{x}}{IQR}$

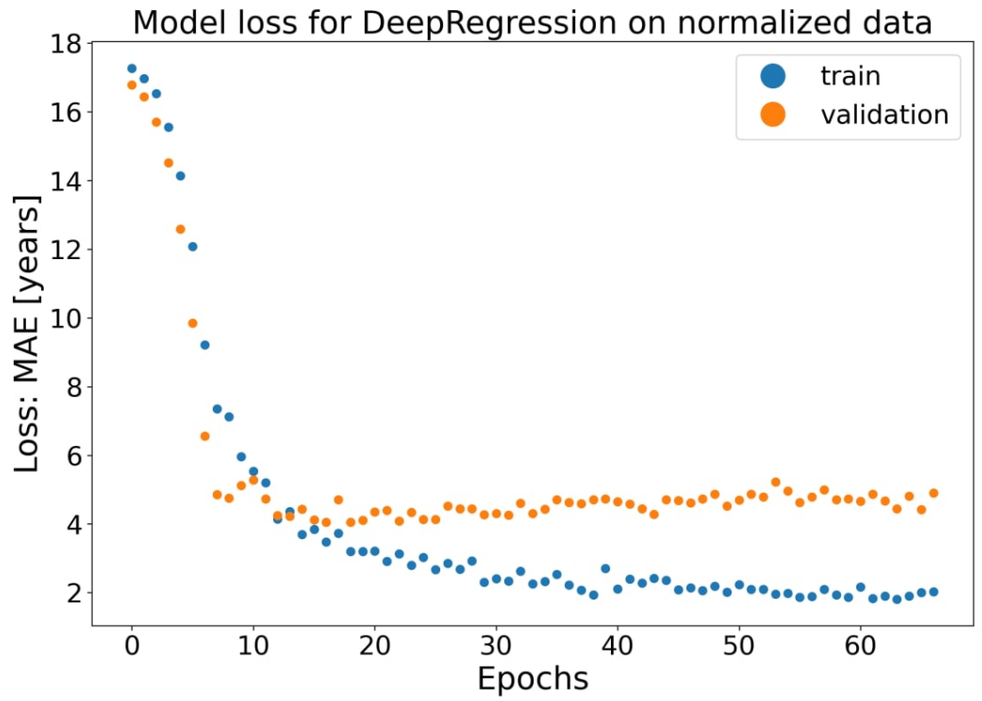
➤ **REGRESSION MODELS**

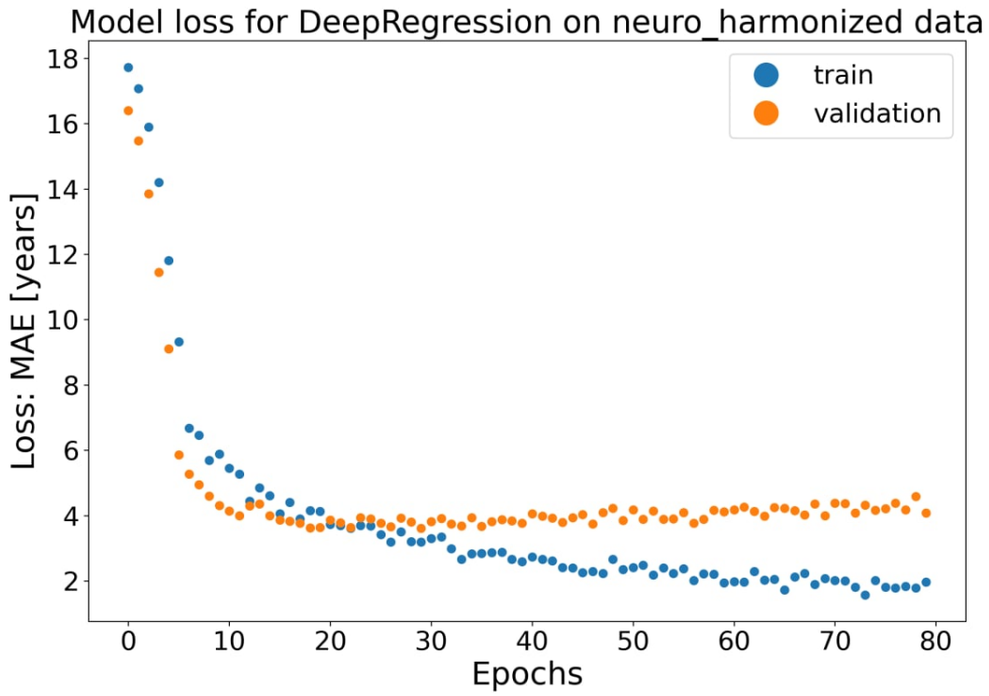
- Several regression model are compared
- Hyper-parameters are tuned for each model in cross validation

1.2 MLP: DEEPREGRESSION.PY

```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(64, activation="relu")(hidden)
        hidden = Dropout(self.drop_rate)(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=50, verbose=1
        )
        self.history = self.model.fit(
            X,
            y,
            validation_split=0.3,
            epochs=self.epochs,
            callbacks=[early_stopping],
            batch_size=32,
            verbose=0,
        )
```

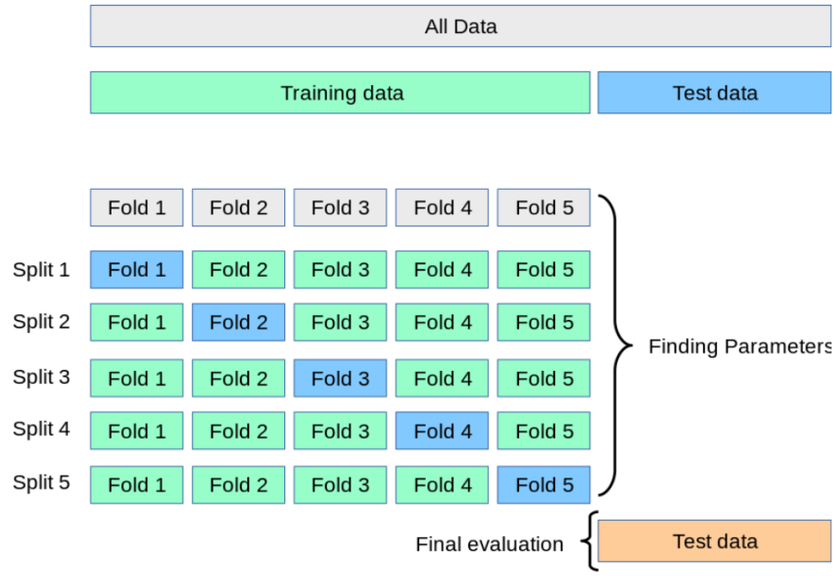
Class inherits from Base Estimator and implements fit and predict methods→can be used in Pipeline





CROSS VALIDATION: CROSSVALIDATION.PY

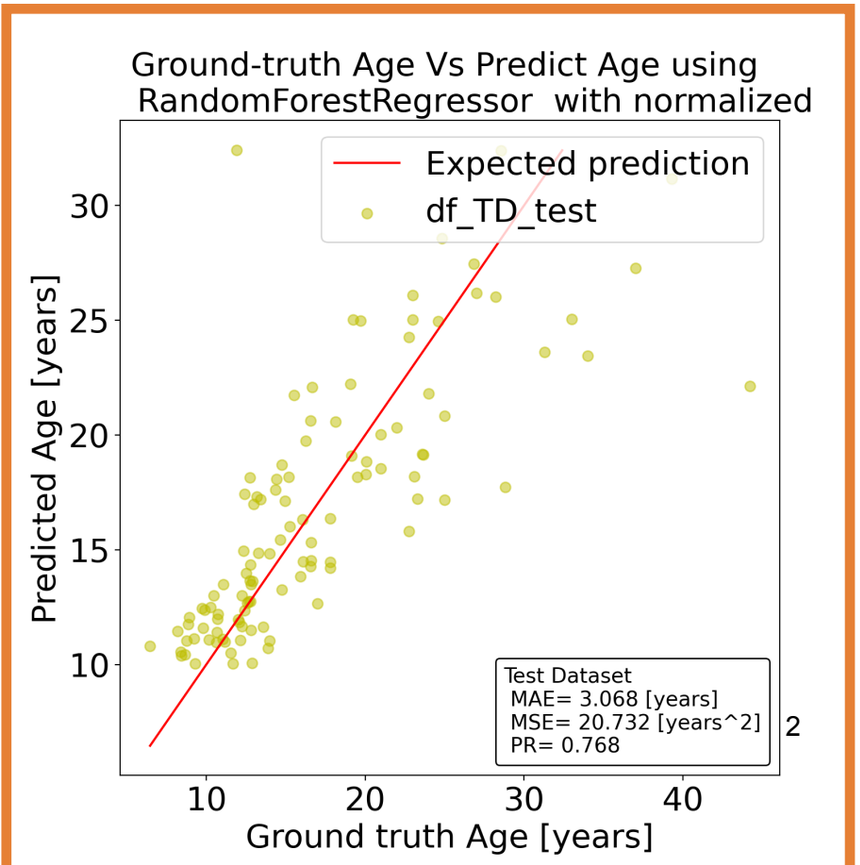
```python
def stratified_k_fold(self, X, y, y_bins, n_splits, model):
    """docstring"""

    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,)
```

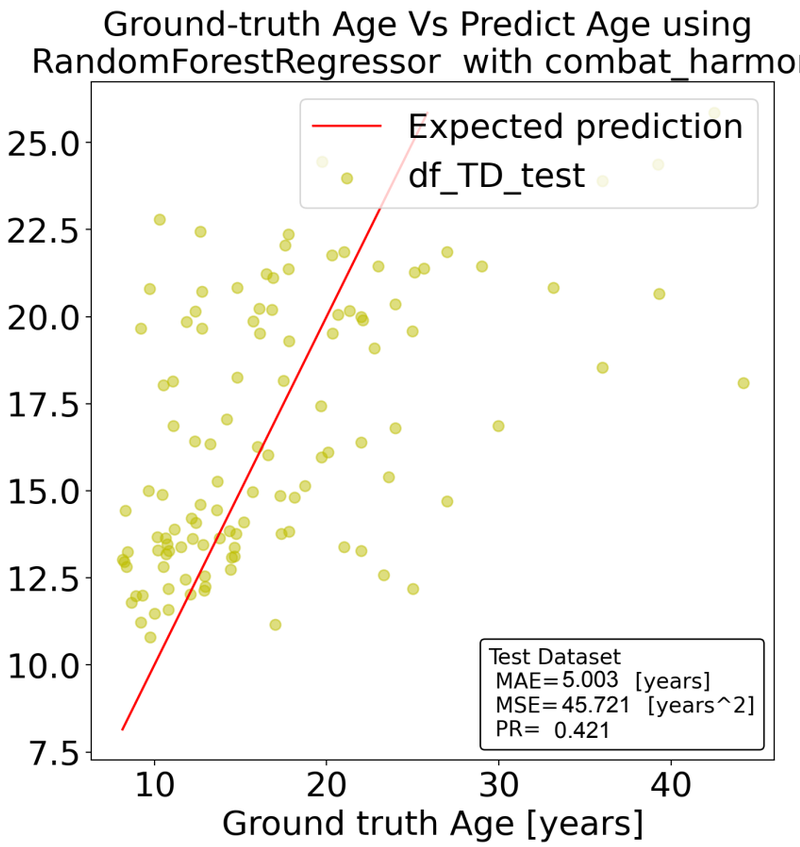


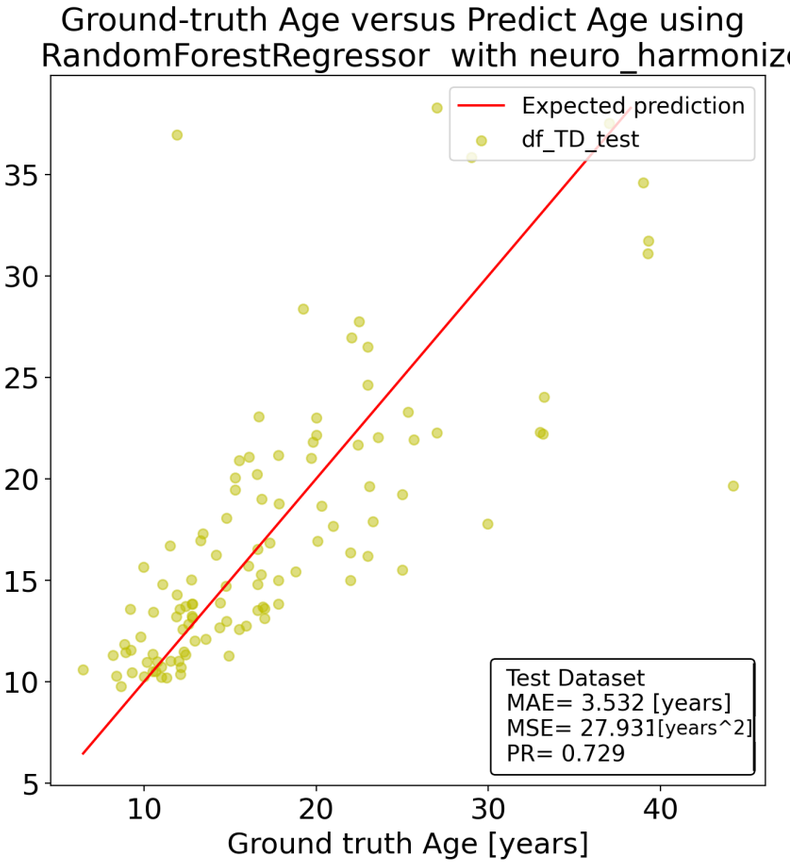
- Random Forest presents overall best performances for all harmonization options
- Not harmonizing data seems to improves the performance in terms of metric

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

RANDOM FOREST MODEL ON TEST DATASET

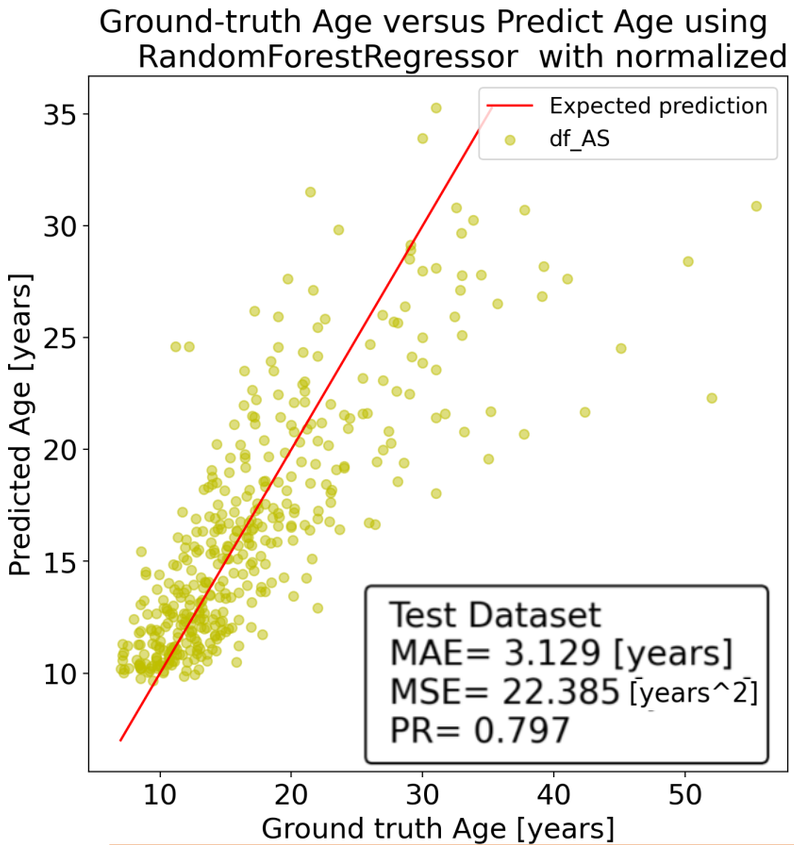


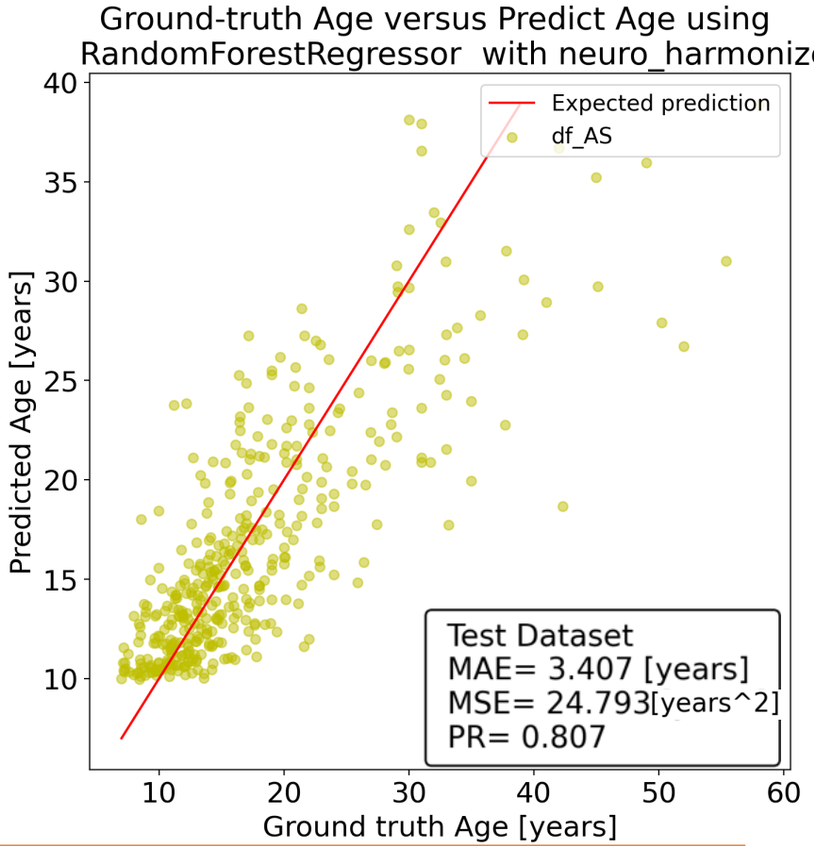




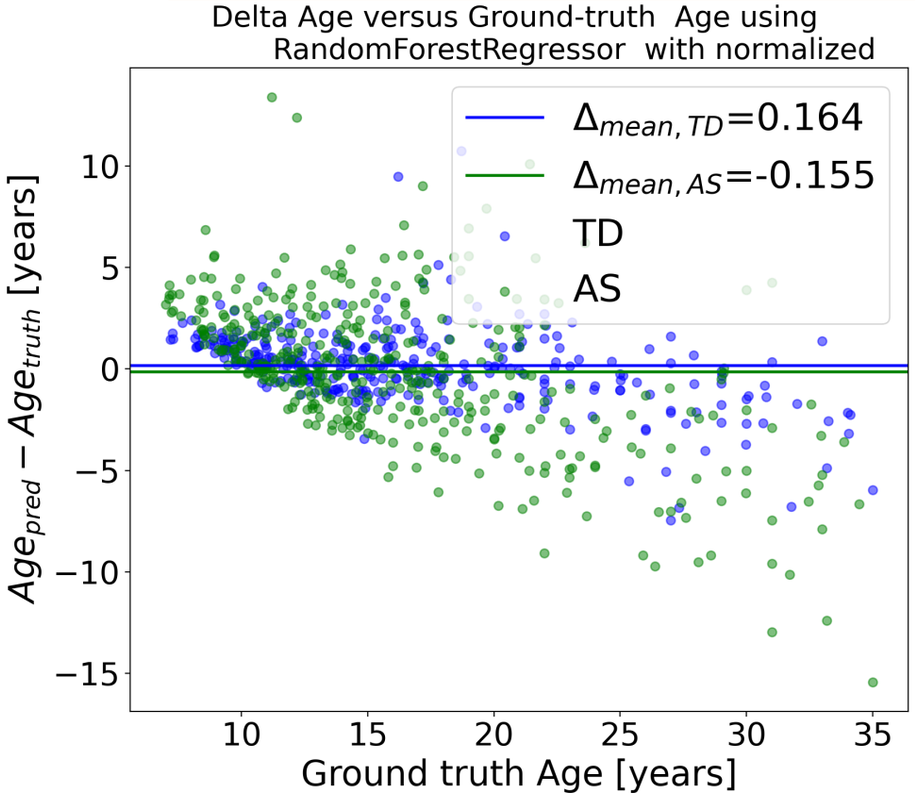
Dataset	MSE	MAE	ρ
Training	18.19	2.91	0.828
Test	20.732	3.068	0.768

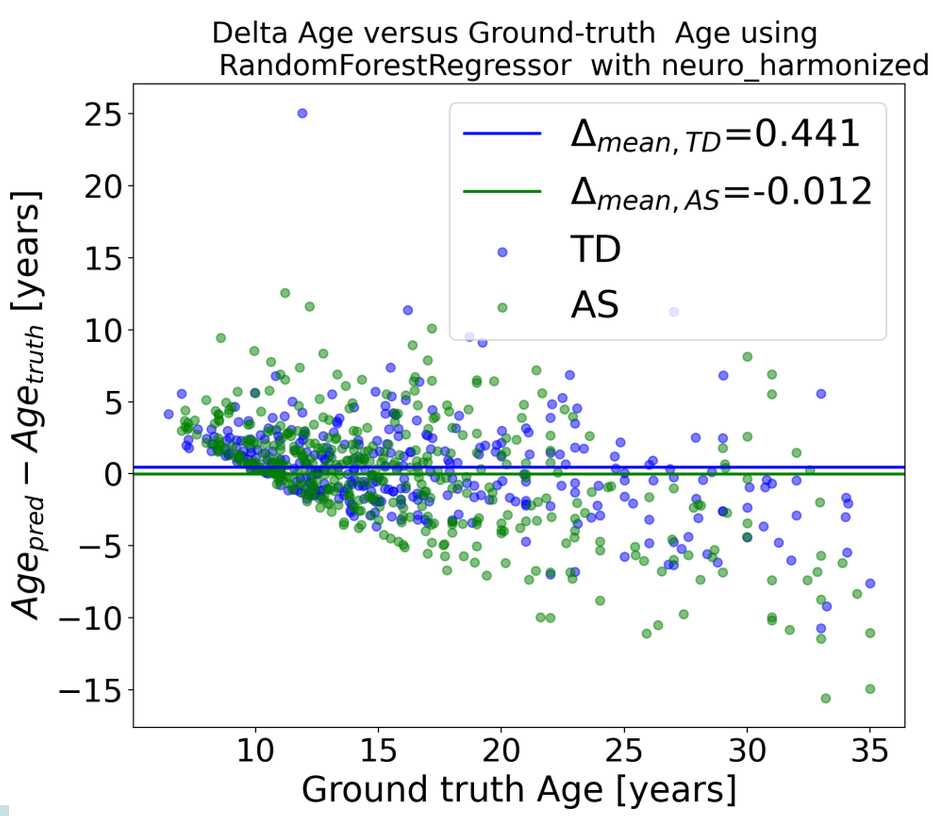
AUTISTIC SUBJECTS



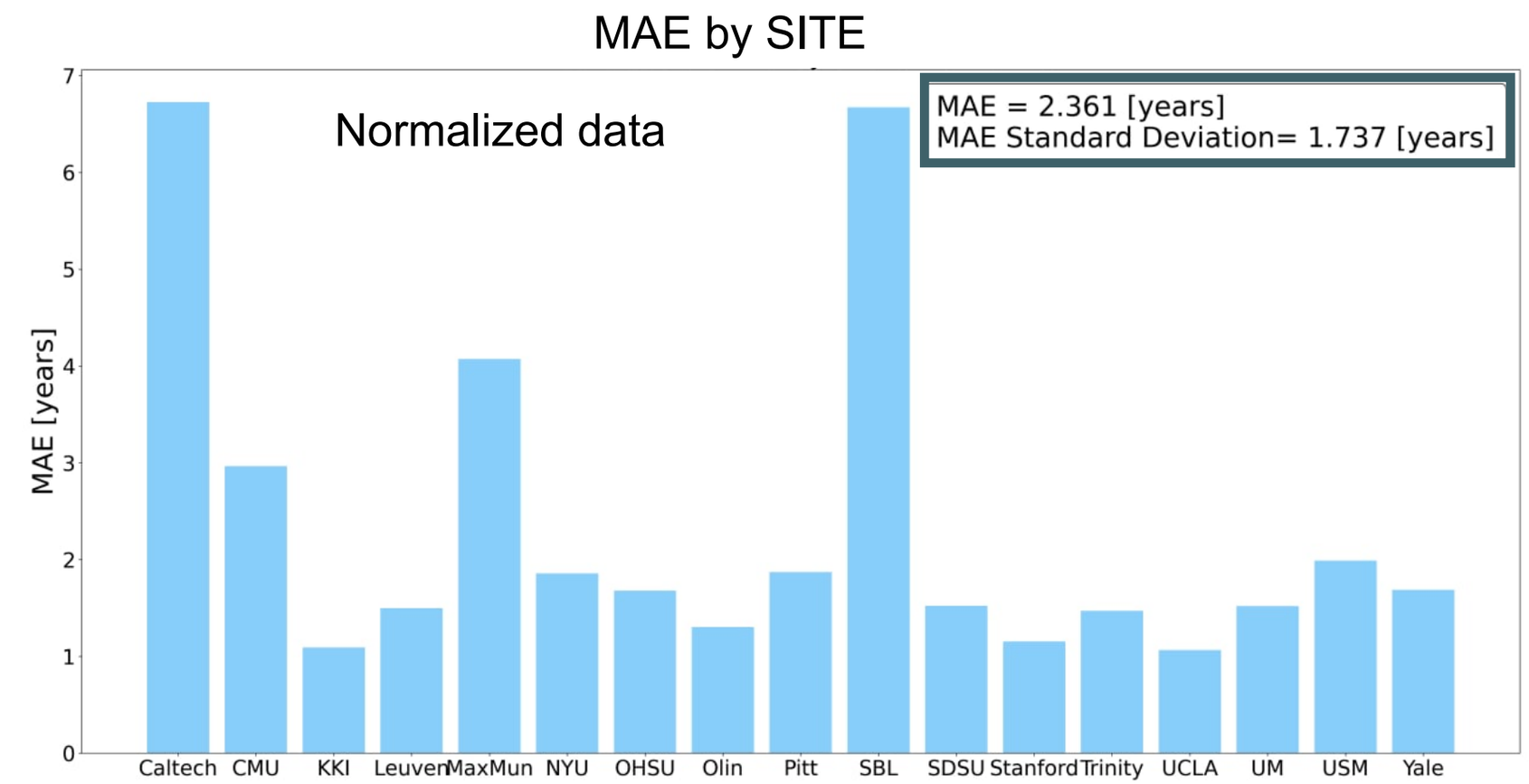


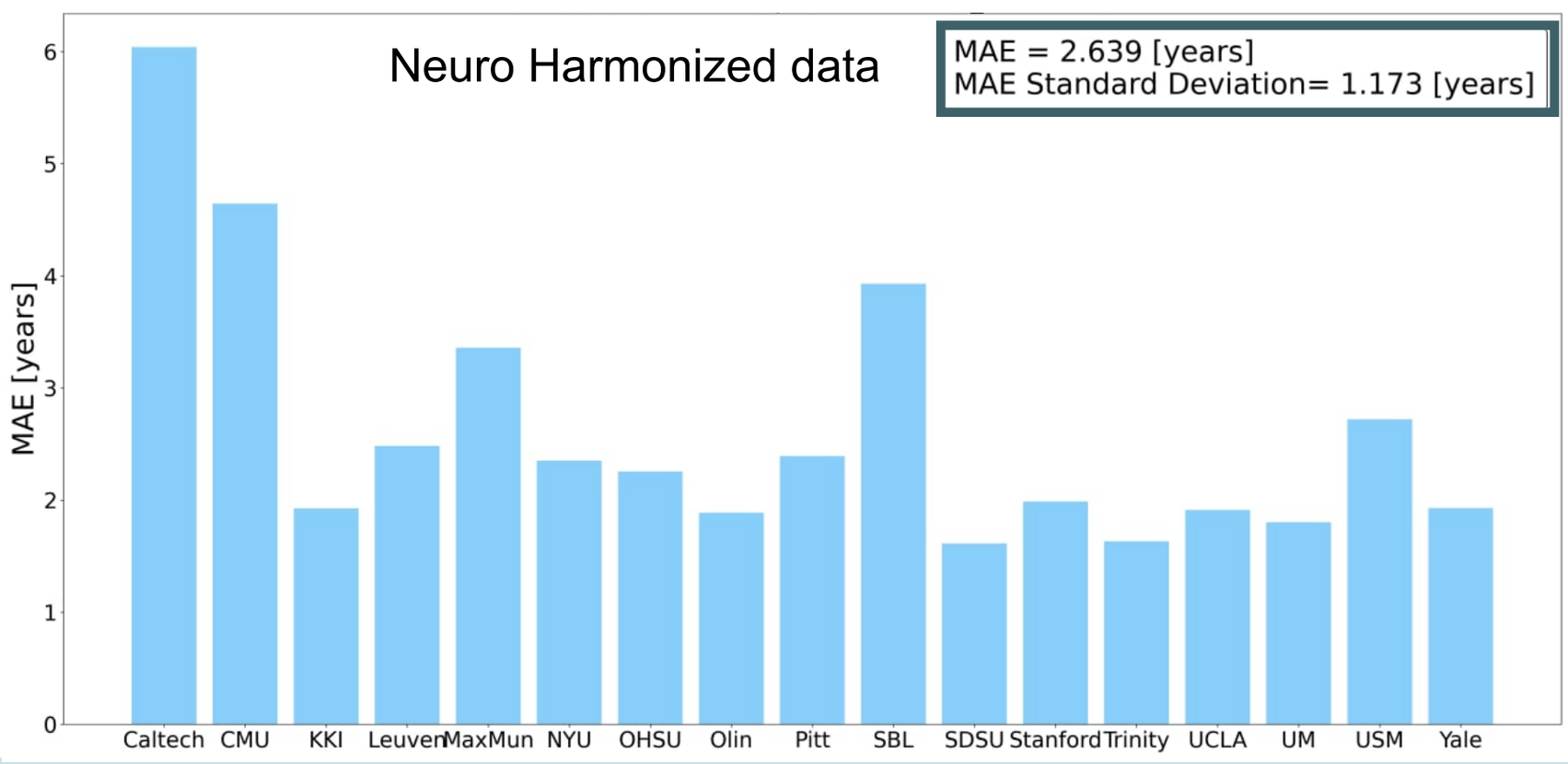
Autistic subjects are predicted to be slightly younger while controls older.





REPRODUCIBILITY





- Entire control dataset with Random Forest model
- Harmonizing data with neuro harmonize improves the reproducibility over different sites

MULTITHREADING FOR IO OPERATION

- Access to saved models is implemented with multithreading

```python
def import_model(harmonize_option, model_name):
    with open("models/%s_%s_pkl" % (model_name, harmonize_option), "rb") as f:
        model_fit = pickle.load(f)

if __name__ == "__main__":

    threads1 = [
        thr.Thread(
            target=import_model,
            args=(
                "normalized",
                model_list[x],
            ),
        )
        for x in range(len(model_list))
    ]

    threads2 = [
        thr.Thread(
            target=import_model,
            args=(
                "combat_harmonized",
                model_list[x],
            ),
        )
        for x in range(len(model_list))
    ]
    threads3 = [
        thr.Thread(
            target=import_model,
            args=(
                "neuro_harmonized",
                model_list[x],
            ),
        )
        for x in range(len(model_list))
    ]
    threads = list(chain(threads1, threads2, threads3))
    t0 = perf_counter()
    # start threads
    for thread in threads:
        thread.start()
        # join threads
    for thread in threads:
        thread.join()
    print("Time multithreading: " + str(perf_counter() - t0))

    # SEQUENTIAL
    t1 = perf_counter()

    for harmonize_option in harmonize_list:
        for model_name in ["SVR", "Lasso", "LinearRegression"]:
            with open("models/%s_%s_pkl" % (model_name, harmonize_option), "rb") as f:
                model_fit = pickle.load(f)

    print("Time sequential: " + str(perf_counter() - t1))
```

0.7 s

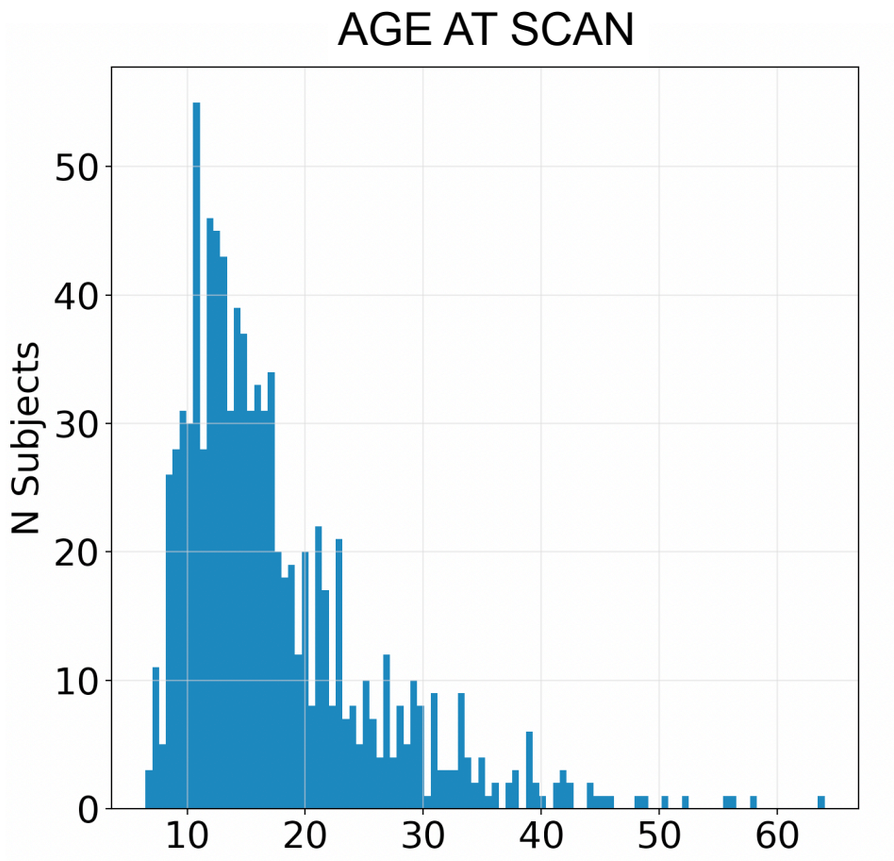
0.015 s

FUTURE IDEAS AND IMPROVEMENTS

- Age range >30 years is strongly under-represented

→ sample weight when training the models

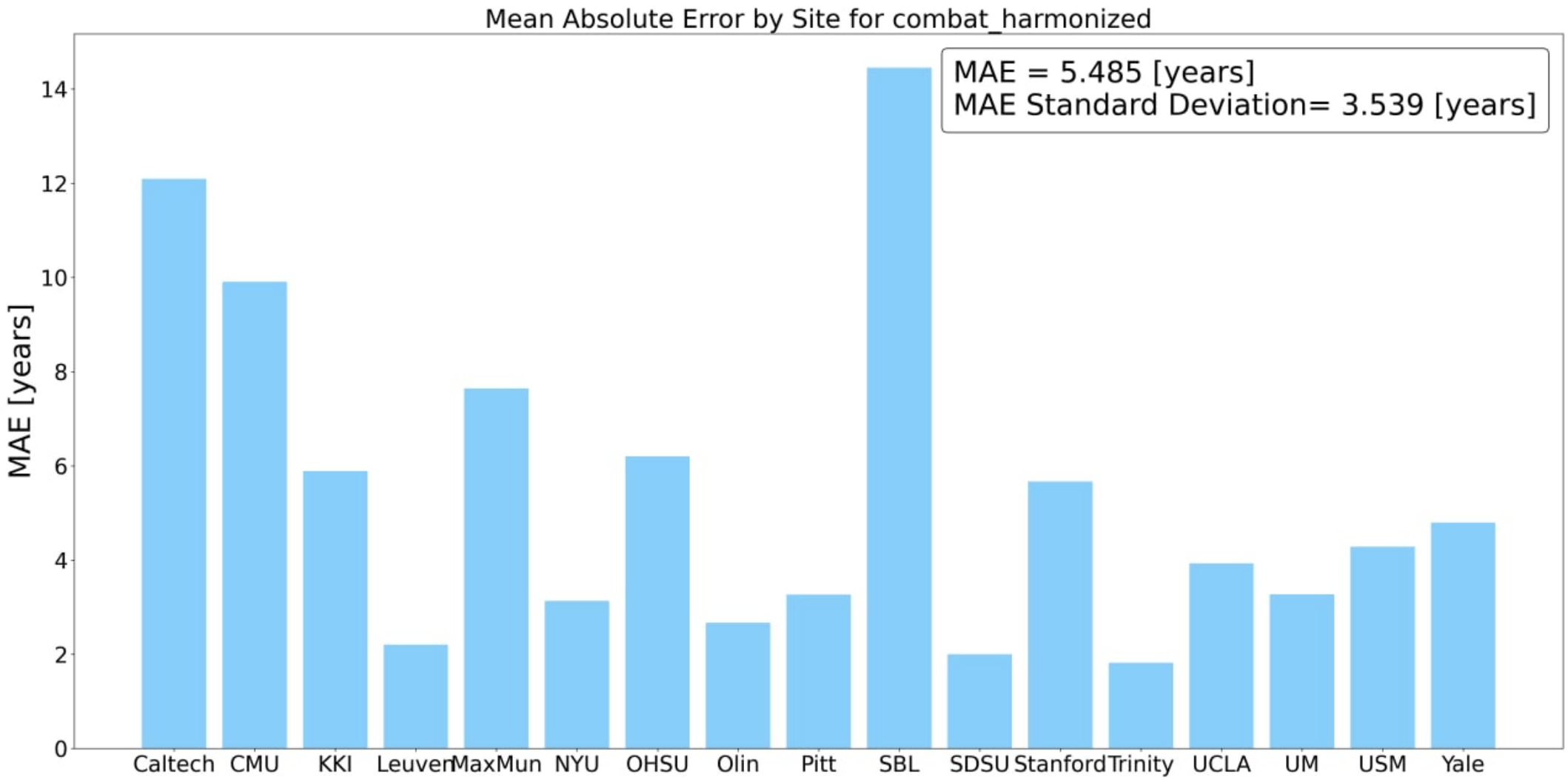
- Implement multiprocessing instead of multithreading for I/O
- Understanding why combat is performing so bad



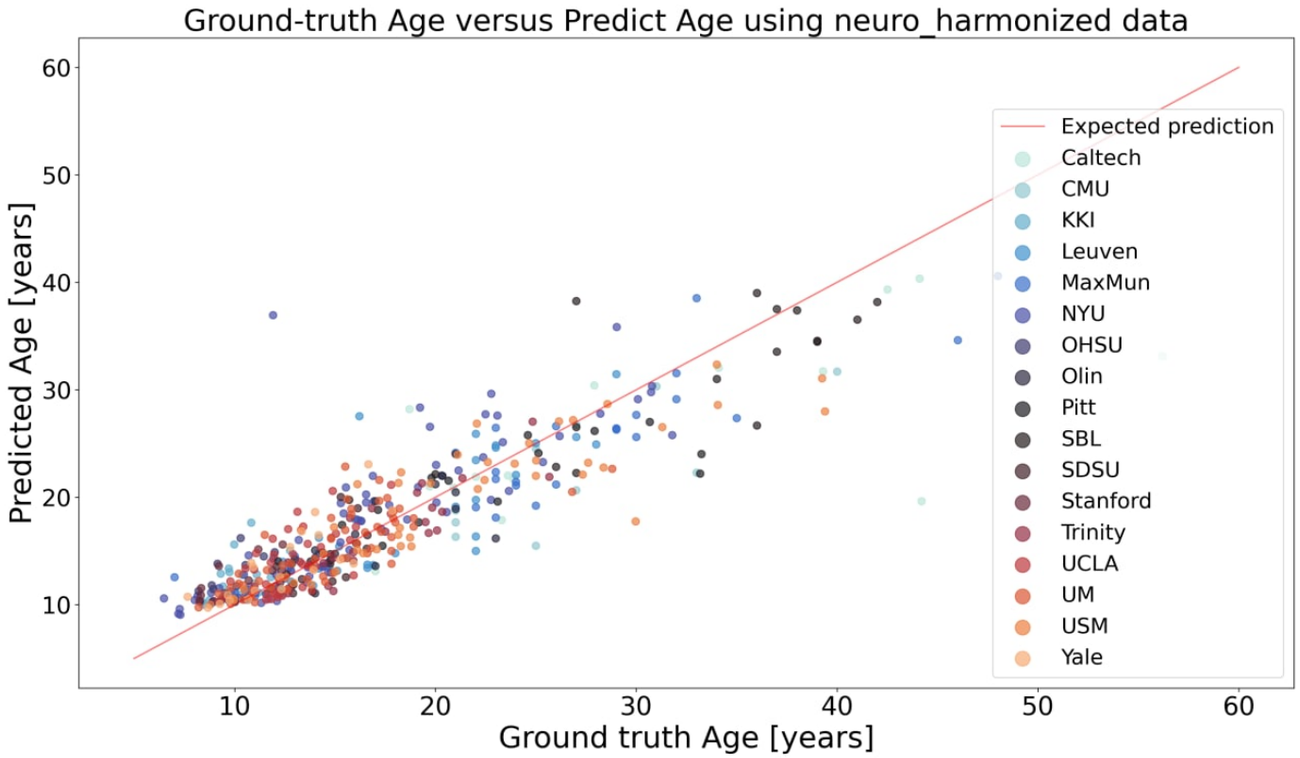
THANK YOU FOR YOUR ATTENTION!

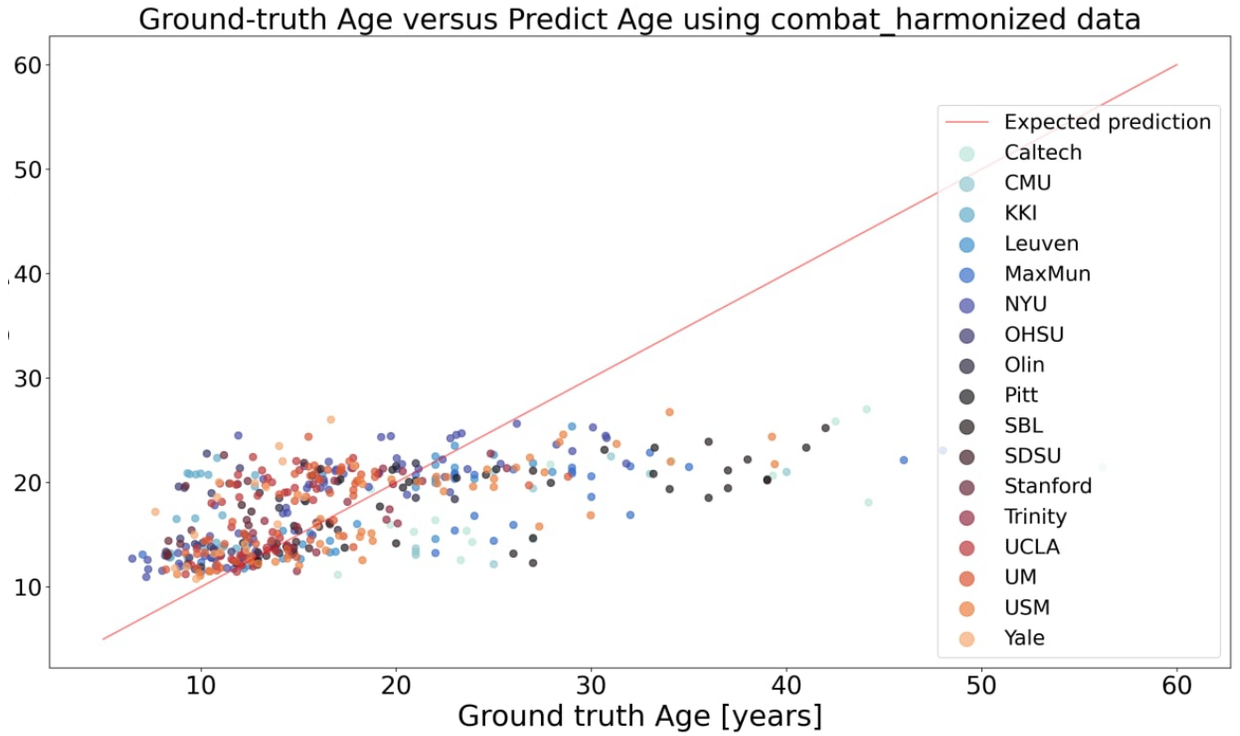
BACKUP SLIDES

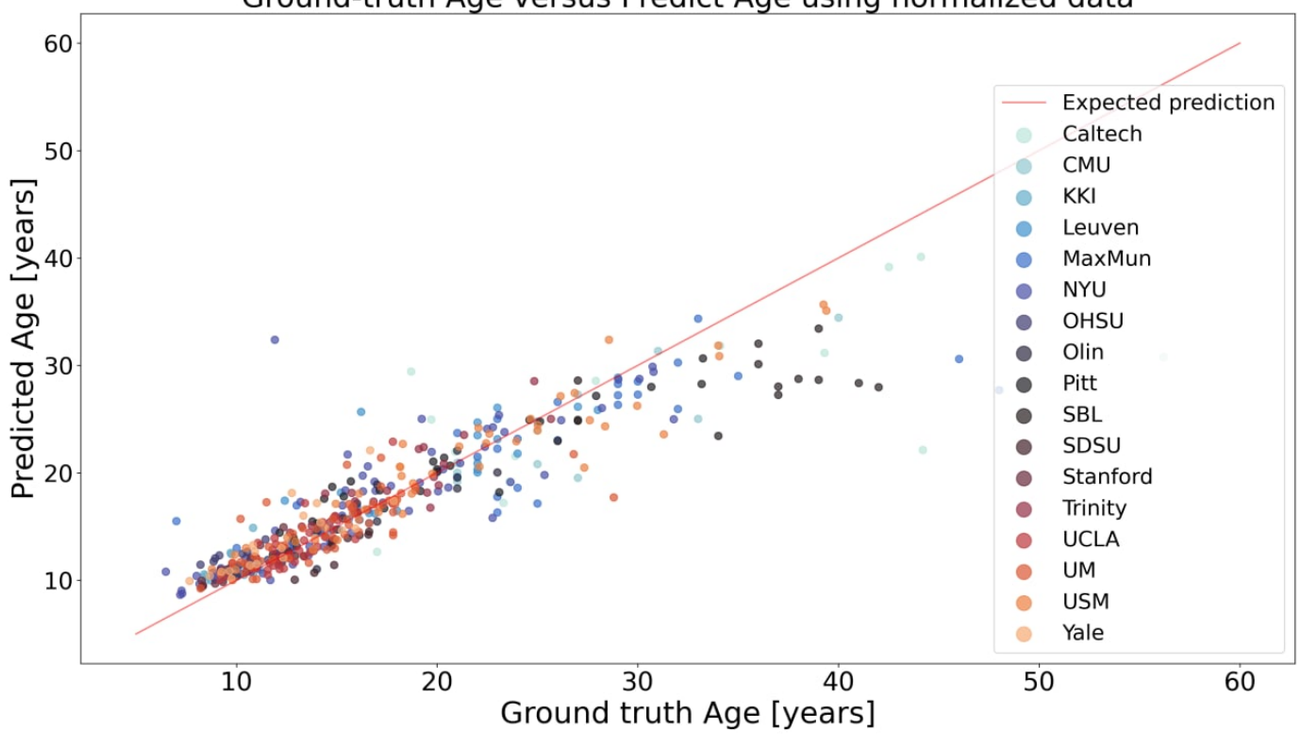
REPRODUCIBILITY



PREDICTED AGE MAE BY SITE







RNN TRAINING ON TPU WITH COLAB : RNN.IPYNB