Introduction to OmniAge

Authors:

- **Zhaozhen Du** (Shanghai Institute of Nutrition and Health, CAS) - duzhaozhen2022@sinh.ac.cn
- **Andrew E. Teschendorff** (Shanghai Institute of Nutrition and Health, CAS) - andrew@sinh.ac.cn

Package: `omniage`

Overview of Implemented Clocks

The `omniage` package provides a comprehensive Python framework for aging research. It offers high-level interfaces for calculating a vast suite of aging-related clocks and biomarkers. These tools are broadly categorized into three main groups:

1. **Epigenetic (DNAm) Clocks**
2. **Transcriptomic (RNA) Clocks**
3. **Other Aging-Related Surrogate Biomarkers**

The suite of **epigenetic aging clocks** is particularly extensive, organized into several functional categories, including predictors for chronological age, biological age, cellular division (mitotic clocks), and gestational age.

Epigenetic (DNAm) Aging Clocks

Chronological Age Clocks

These DNAm clocks are designed for the prediction of chronological age based on DNA methylation data.

- **Horvath2013**: (Horvath 2013) The original pan-tissue clock from 353 CpGs.
- **Hannum**: (Hannum et al. 2013) Blood-specific clock from 71 CpGs.
- **Lin**: (Lin et al. 2016) Clock based on 99 CpGs.
- **VidalBralo**: (Vidal-Bralo et al. 2016) Clock based on 8 CpGs.
- **ZhangClock**: (Zhang et al. 2019) Improved-precision clock from 514 CpGs.
- **Horvath2018**: (Horvath et al. 2018) Skin & blood clock.
- **Bernabeu_cAge**: (Bernabeu et al. 2022) Clock based on 3225 CpGs.
- **CorticalClock**: (Shireby et al. 2020) Brain cortical clock based on 347 CpGs.
- **PedBE**: (McEwen et al. 2019) Pediatric buccal epithelial clock for children.

- **CentenarianClock**: (Eric Dec et al. 2023) Centenarian Epigenetic Clocks.
- **Retro_age**: (Ndhlovu et al. 2024) Retroelement-based clock developed on the EPIC v1/v2 arrays.
- **PC Clocks** (PCHorvath2013, PCHorvath2018, PCHannum): (Higgins-Chen et al. 2022) Computationally-bolstered versions retrained using principal components to minimize technical noise.

Biological Age Clocks

These advanced clocks are designed to capture biological aging rather than chronological time, often showing stronger associations with health outcomes and mortality.

- **Zhang10**: (Zhang et al. 2017) 10-CpG clock associated with mortality.
- **PhenoAge**: (Levine et al. 2018) Predicts phenotypic age from 513 CpGs.
- **DunedinPACE**: (Belsky et al. 2022) Quantifies the pace of biological aging.
- **GrimAge1**: The original GrimAge clock (Lu et al. 2019).
- **GrimAge2**: (Lu et al. 2022) Updated composite biomarker of mortality risk.
- **PC Biological Clocks** (PCPhenoAge, PCGrimAge1): (Higgins-Chen et al. 2022) Computationally-bolstered PC versions.
- **DNAmFitAge**: (McGreevy et al. 2023) Indicator incorporating DNAmGrimAge and physical fitness markers.
- **IC_Clock**: (Fuentealba et al. 2025) The Intrinsic Capacity (IC) Clock based on 91 CpGs.
- **SystemsAge**: (Sehgal et al. 2025) The Systems Age and 11 System-Specific Scores.

Cellular Aging Clocks

This category groups biomarkers that measure two key forms of cellular aging: cell proliferation (mitotic clocks) and telomere attrition (DNAmTL).

Mitotic Clocks (Cellular Division)

A specialized set of clocks that measure cell division history or stem cell divisions.

- **epiTOC1**: (Yang et al. 2016) Average beta value of 385 promoter CpGs.
- **epiTOC2**: (Teschendorff et al. 2020) Estimates stem cell divisions using a dynamic model.
- **epiTOC3**: Estimates stem cell divisions based on unmethylated population doubling associated CpGs.
- **stemTOCvitro**: (Zhu et al. 2024) The 0.95 upper quantile of 629 stemTOCvitro CpGs.
- **stemTOC**: (Zhu et al. 2024) The 0.95 upper quantile of 371 stemTOC CpGs, filtered for in-vivo hypermethylation.

- **RepliTali**: (Endicott et al. 2022) Based on 87 population doubling associated hypomethylated CpGs.
- **HypoClock**: (Teschendorff et al. 2020) Based on hypomethylation at 678 solo-WCGW sites.
- **EpiCMIT** (hyper/hypo): (Duran-Ferrer et al. 2020) Average beta values of age-associated hyper/hypomethylated CpGs.

DNAm Telomere Length (TL) Clocks

- **DNAmTL**: (Lu AT et al. 2019) Calculates the Leukocyte telomere length.
- **PCDNAmTL**: (Lu AT et al. 2019; Higgins-Chen et al. 2022) A computationally-bolstered "PC" version of the original DNAmTL clock.

Causal Clocks

A new generation of clocks developed to reflect potential causal drivers of aging.

- **CausalAge, DamAge, AdaptAge**: (Ying et al. 2024) Three clocks derived from a causality-enriched model.

Stochastic Clocks

- **StocH, StocP, StocZ**: (Tong et al. 2024) Stochastic analogues of the Horvath, PhenoAge, and Zhang clocks.

Cell-Type Specific Clocks

- **Neu-In, Glia-In**: (Tong et al. 2024) Intrinsic clocks for neurons and glia.
- **Neu-Sin, Glia-Sin, Hep**: (Tong et al. 2024) Semi-intrinsic clocks for neurons, glia, and hepatocytes.

Gestational Age Clocks

This collection includes DNAm clocks designed to predict gestational age at birth.

- **Bohlin_GA**: (Bohlin et al. 2016)
- **EPIC_GA**: (Haftorn et al. 2021)
- **Lee_GA**: (Lee et al. 2019)
- **Knight_GA**: (Knight et al. 2016)
- **Mayne_GA**: (Mayne et al. 2017)

Cross-Species Support

- **EnsembleAge**: (Haghani et al. 2025) Multi-clock framework (Dynamic, Static, HumanMouse).

- **UniversalPanMammalianClocks**: (Lu et al. 2023) Applicable across all mammalian tissues.
- **PanMammalianBlood / Skin**: (Lu et al. 2023) Tissue-specific pan-mammalian clocks.

Transcriptomic Clocks

In addition to DNAm-based models, `omniage` implements cutting-edge transcriptomic (RNA-seq) clocks:

- **sc-ImmuAging**: A single-cell, immune-cell-type-specific aging clock (Li et al. 2025).
- **Brain_CT_clock**: A brain-cell-type-specific aging clock based on Single-Nuclei Transcriptomics (Muralidharan et al. 2025).
- **Pasta**: A robust, multi-tissue transcriptomic clock based on a novel age-shift learning strategy (Salignon et al. 2025).

Surrogate Biomarkers & Scores

Besides aging clocks, `omniage` includes functions to compute important surrogate markers associated with lifestyle and health status from DNAm data.

- **CompCRP**: DNAm score for C-reactive protein (CRP) levels.
- **CompCHIP**: Scores for Clonal Hematopoiesis of Indeterminate Potential (CHIP).
- **CompIL6**: DNAm surrogate score for Interleukin-6 (IL-6).
- **CompEpiScores**: 109 validated epigenetic scores serving as proxies for circulating plasma proteins (Gadd et al. 2022).
- **McCartney_Trait**: Surrogate scores for 10 complex traits (BMI, Smoking, Alcohol, Education, Cholesterol, WHR, Body Fat, etc.) (McCartney et al. 2018).

DNA Methylation Cell-Type Fraction

The package provides a DNA Methylation Cell-Type Fraction (CTF) clock (`DNAmCTFClock`), trained on large-scale blood samples. It incorporates 12 immune cell types to model aging based on cell-type composition.

Available Clocks and Functions

Epigenetic (DNAm) Aging Clocks

Clock Name	Category	Python Function / Class
`Horvath2013`	Chronological	`omniage.cal_epimarker(clocks='Horvath20` `omniage.Horvath2013`
`Hannum`	Chronological	`omniage.cal_epimarker(clocks='Hannum')` `omniage.Hannum`
`Lin`	Chronological	`omniage.cal_epimarker(clocks='Lin')`, `om`
`VidalBralo`	Chronological	`omniage.cal_epimarker(clocks='VidalBral` `omniage.VidalBralo`
`ZhangClock`	Chronological	`omniage.cal_epimarker(clocks='ZhangCloc` `omniage.ZhangClock`
`Horvath2018`	Chronological	`omniage.cal_epimarker(clocks='Horvath20` `omniage.Horvath2018`
`Bernabeu_cAge`	Chronological	`omniage.cal_epimarker(clocks='Bernabeu_` `omniage.Bernabeu_cAge`
`CorticalClock`	Chronological	`omniage.cal_epimarker(clocks='CorticalC` `omniage.CorticalClock`
`PedBE`	Chronological	`omniage.cal_epimarker(clocks='PedBE')`,
`CentenarianClock`	Chronological	`omniage.cal_epimarker(clocks='Centenari` `omniage.CentenarianClock`
`Retro_age`	Chronological	`omniage.cal_epimarker(clocks='Retro_age` `omniage.Retro_age`
`Zhang10`	Biological	`omniage.cal_epimarker(clocks='Zhang10')` `omniage.Zhang10`
`PhenoAge`	Biological	`omniage.cal_epimarker(clocks='PhenoAge'` `omniage.PhenoAge`
`DunedinPACE`	Biological	`omniage.cal_epimarker(clocks='DunedinPA` `omniage.DunedinPACE`
`GrimAge1`	Biological	`omniage.cal_epimarker(clocks='GrimAge1'` `omniage.GrimAge1`
`GrimAge2`	Biological	`omniage.cal_epimarker(clocks='GrimAge2'` `omniage.GrimAge2`
`DNAmFitAge`	Biological	`omniage.cal_epimarker(clocks='DNAmFitAg` `omniage.DNAmFitAge`
`IC_Clock`	Biological	`omniage.cal_epimarker(clocks='IC_Clock'` `omniage.IC_Clock`
`SystemsAge`	Biological	`omniage.cal_epimarker(clocks='SystemsAg` `omniage.SystemsAge`
`epiTOC1`	Mitotic	`omniage.cal_epimarker(clocks='EpiTOC1')` `omniage.EpiTOC1`
`epiTOC2`	Mitotic	`omniage.cal_epimarker(clocks='EpiTOC2')` `omniage.EpiTOC2`

Clock Name	Category	Python Function / Class
`epiTOC3`	Mitotic	`omniage.cal_epimarker(clocks='EpiTOC3')` `omniage.EpiTOC3`
`stemTOC`	Mitotic	`omniage.cal_epimarker(clocks='stemTOC')` `omniage.StemTOC`
`stemTOCvitro`	Mitotic	`omniage.cal_epimarker(clocks='stemTOCvi` `omniage.StemTOCvitro`
`RepliTali`	Mitotic	`omniage.cal_epimarker(clocks='RepliTali` `omniage.RepliTali`
`HypoClock`	Mitotic	`omniage.cal_epimarker(clocks='HypoClock` `omniage.HypoClock`
`EpiCMIT_hyper`	Mitotic	`omniage.cal_epimarker(clocks='EpiCMIT_h` `omniage.EpiCMIT_hyper`
`EpiCMIT_hypo`	Mitotic	`omniage.cal_epimarker(clocks='EpiCMIT_h` `omniage.EpiCMIT_hypo`
`DNAmTL`	Telomere length	`omniage.cal_epimarker(clocks='DNAmTL')` , `omniage.DNAmTL`
`PCHorvath2013`	Chronological	`omniage.cal_epimarker(clocks='PCClocks'` `omniage.PCHorvath2013`
`PCHorvath2018`	Chronological	`omniage.cal_epimarker(clocks='PCClocks'` `omniage.PCHorvath2018`
`PCHannum`	Chronological	`omniage.cal_epimarker(clocks='PCClocks'` `omniage.PCHannum`
`PCPhenoAge`	Biological	`omniage.cal_epimarker(clocks='PCClocks'` `omniage.PCPhenoAge`
`PCGrimAge1`	Biological	`omniage.cal_epimarker(clocks='PCClocks'` `omniage.PCGrimAge1`
`PCDNAmTL`	Telomere Length	`omniage.cal_epimarker(clocks='PCClocks'` `omniage.PCDNAmTL`
`CausalAge`	Causal	`omniage.cal_epimarker(clocks='CausalAge` `omniage.CausalAge`
`DamAge`	Causal	`omniage.cal_epimarker(clocks='DamAge')` , `omniage.DamAge`
`AdaptAge`	Causal	`omniage.cal_epimarker(clocks='AdaptAge'` `omniage.AdaptAge`
`StocH`	Stochastic	`omniage.cal_epimarker(clocks='StocH')` ,
`StocP`	Stochastic	`omniage.cal_epimarker(clocks='StocP')` ,
`StocZ`	Stochastic	`omniage.cal_epimarker(clocks='StocZ')` ,
`NeuIn`	Cell-Type Specific	`omniage.cal_epimarker(clocks='NeuIn')` ,
`NeuSin`	Cell-Type Specific	`omniage.cal_epimarker(clocks='NeuSin')` , `omniage.NeuSin`

Clock Name	Category	Python Function / Class
`GliaIn`	Cell-Type Specific	`omniage.cal_epimarker(clocks='GliaIn')` , `omniage.GliaIn`
`GliaSin`	Cell-Type Specific	`omniage.cal_epimarker(clocks='GliaSin')` `omniage.GliaSin`
`Hep`	Cell-Type Specific	`omniage.cal_epimarker(clocks='Hep')` , `om`
`CTS_Clocks`	Cell-Type Specific	`omniage.cal_epimarker(clocks='CTS_Clock` `omniage.CTS_Clocks`
`BohlinGA`	Gestational	`omniage.cal_epimarker(clocks='BohlinGA'` `omniage.BohlinGA`
`EPICGA`	Gestational	`omniage.cal_epimarker(clocks='EPICGA')` , `omniage.EPICGA`
`KnightGA`	Gestational	`omniage.cal_epimarker(clocks='KnightGA'` `omniage.KnightGA`
`MayneGA`	Gestational	`omniage.cal_epimarker(clocks='MayneGA')` `omniage.MayneGA`
`LeeControl`	Gestational	`omniage.cal_epimarker(clocks='LeeContro` `omniage.LeeControl`
`LeeRobust`	Gestational	`omniage.cal_epimarker(clocks='LeeRobust` `omniage.LeeRobust`
`LeeRefinedRobust`	Gestational	`omniage.cal_epimarker(clocks='LeeRefine` `omniage.LeeRefinedRobust`
`EnsembleAgeHumanMouse`	Mouse/Human	`omniage.cal_epimarker(clocks='EnsembleA` `omniage.EnsembleAgeHumanMouse`
`EnsembleAgeStatic`	Mouse	`omniage.cal_epimarker(clocks='EnsembleA` `omniage.EnsembleAgeStatic`
`EnsembleAgeDynamic`	Mouse	`omniage.cal_epimarker(clocks='EnsembleA` `omniage.EnsembleAgeDynamic`
`PanMammalianUniversal`	PanMammalian	`omniage.cal_epimarker(clocks='PanMammal` `omniage.PanMammalianUniversal`
`PanMammalianBlood`	PanMammalian	`omniage.cal_epimarker(clocks='PanMammal` `omniage.PanMammalianBlood`
`PanMammalianSkin`	PanMammalian	`omniage.cal_epimarker(clocks='PanMammal` `omniage.PanMammalianSkin`
`DNAm_CTF_Clock`	Cell-Type Fraction	`omniage.cal_epimarker(clocks='DNAmCTFCl` `omniage.DNAmCTFClock`

Transcriptomic Clocks

Clock Name	Category	Python Function / Class
`sc-ImmuAging`	Immune-cell-type-specific	`omniage.scImmuAging`

Clock Name	Category	Python Function / Class
`Brain_CT_clock`	Brain-cell-type-specific	`omniage.BrainCTClock`
`Pasta`	Multi-tissue	`omniage.PASTA_Clock`

Surrogate Biomarkers & Scores

Clock Name	Target	Python Function
`CompCRP`	CRP/intCRP score	`omniage.cal_epimarker(clocks='CompCRP'`
`CompCHIP`	CHIP score	`omniage.cal_epimarker(clocks='CompCHIP` `omniage.CompCHIP`
`CompIL6`	IL6 score	`omniage.cal_epimarker(clocks='CompIL6'`
`CompEpiScores`	109 validated epigenetic scores	`omniage.cal_epimarker(clocks='EpiScore` `omniage.EpiScores`
`McCartney_Trait`	Epigenetic surrogate traits	`omniage.cal_epimarker(clocks='DNAmCTFC` `omniage.calculate_mccartney_traits`
`McCartneyBMI`	Epigenetic surrogate traits	`omniage.cal_epimarker(clocks='McCartne` `omniage.McCartneyBMI`
`McCartneySmoking`	Epigenetic surrogate traits	`omniage.cal_epimarker(clocks='McCartne` `omniage.McCartneySmoking`
`McCartneyAlcohol`	Epigenetic surrogate traits	`omniage.cal_epimarker(clocks='McCartne` `omniage.McCartneyAlcohol`
`McCartneyEducation`	Epigenetic surrogate traits	`omniage.cal_epimarker(clocks='McCartne` `omniage.McCartneyEducation`
`McCartneyTotalCholesterol`	Epigenetic surrogate traits	`omniage.cal_epimarker(clocks='McCartne` `omniage.McCartneyTotalCholesterol`
`McCartneyHDL`	Epigenetic surrogate traits	`omniage.cal_epimarker(clocks='McCartne` `omniage.McCartneyHDL`
`McCartneyLDL`	Epigenetic surrogate traits	`omniage.cal_epimarker(clocks='McCartne` `omniage.McCartneyLDL`
`McCartneyTotalHDLRatio`	Epigenetic surrogate traits	`omniage.cal_epimarker(clocks='McCartne` `omniage.McCartneyTotalHDLRatio`
`McCartneyWHR`	Epigenetic surrogate traits	`omniage.cal_epimarker(clocks='McCartne` `omniage.McCartneyWHR`

Clock Name	Target	Python Function
`McCartneyBodyFat`	Epigenetic surrogate traits	`omniage.cal_epimarker(clocks='McCartney` `omniage.McCartneyBodyFat`

Tutorial Example 1: Examine the available clocks and extract their corresponding features.

In [1]:
```
%load_ext autoreload
%autoreload 2

import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
import statsmodels.formula.api as smf
import math
from scipy.stats import pearsonr
from sklearn.metrics import mean_absolute_error
from statannotations.Annotator import Annotator
import omniage
import scanpy as sc

sns.set(style="whitegrid")
```

In [2]:
```
available = omniage.list_available_clocks()
```

Category (Group)	Available Clocks
Chronological	Horvath2013, Hannum, Lin, VidalBralo, ZhangClock, Horvath2018, Bernabeu_cAge, CorticalClock, PedBE, CentenarianClock_40, CentenarianClock_100, Retro_age_V1, Retro_age_V2, PCHorvath2013, PCHorvath2018, PCHannum
Biological	Zhang10, PhenoAge, DunedinPACE, GrimAge1, GrimAge2, PCGrimAge1, DNAmFitAge, IC_Clock, SystemsAge
Telomere_Length	DNAmTL, PCDNAmTL
Mitotic	EpiTOC1, EpiTOC2, EpiTOC3, StemTOCvitro, StemTOC, RepliTali, HypoClock, EpiCMIT_Hyper, EpiCMIT_Hypo
Causal	CausalAge, DamAge, AdaptAge
Stochastic	StocH, StocZ, StocP
CellType_Specific	NeuIn, NeuSin, GliaIn, GliaSin, Hep
Transcriptomic	scImmuAging, BrainCTClock, PASTA_Clock
CellType_Fractin	DNAmCTFClock
Trait	McCartneyBMI, McCartneySmoking, McCartneyAlcohol, McCartneyEducation, McCartneyTotalCholesterol, McCartneyHDL, McCartneyLDL, McCartneyTotalHDLRatio, McCartneyWHR, McCartneyBodyFat
Gestational	BohlinGA, EPICGA, KnightGA, MayneGA, LeeControl, LeeRobust, LeeRefinedRobust
Surrogate_Biomarkers	CompCRP, CompCHIP, EpiScores, CompIL6
Cross_Species	PanMammalianUniversal, PanMammalianBlood, PanMammalianSkin, EnsembleAgeHumanMouse, EnsembleAgeStatic, EnsembleAgeDynamic

```
--------------------------------------------------------------------------------
------------------
================================================================================
==================
```

In [3]:
```
Clock_coef = omniage.get_clock_coefs(["Horvath2013","ZhangClock"])
print(Clock_coef)
```

```
Retrieving coefficients for 2 clocks...
100%|██████████| 2/2 [00:00<00:00, 182.72it/s]
{'Horvath2013':            probe      coef
0    (Intercept)  0.695507
1     cg00075967  0.129337
2     cg00374717  0.005018
3     cg00864867  1.599764
4     cg00945507  0.056852
..           ...       ...
349   cg26824091  0.549029
350   cg27015931 -0.466528
351   cg27016307 -0.035326
352   cg27202708 -0.138636
353   cg27544190 -0.869124

[354 rows x 2 columns], 'ZhangClock':            probe       coef
0    (Intercept)  65.792950
1     cg24611351  -0.001811
2     cg24173182  -0.203955
3     cg09604333  -0.703968
4     cg13617776  -0.011524
..           ...        ...
510   cg04749507  -0.357000
511   cg21674704   0.955665
512   cg25909396  -1.271815
513   cg12402251   0.758705
514   cg20515136  -0.093365

[515 rows x 2 columns]}
```

Tutorial Example 2: Apply mitotic clocks on Lung pre-cancerous lesions

This is an Illumina 450k dataset encompassing 21 normal lung tissue samples and 35 lung carcinoma in situ (LCIS) samples, of which 22 progressed to an invasive lung cancer (LC). Here we explore the correlation between mitotic age and cancer state with linear model, treating N, LCIS, and LC as ordinal variable (1,2,3) and adjusting for age.

In [4]:
```
print("1. Loading Lung Pre-cancerous Lesions Dataset...")

beta_df = pd.read_csv("../example/data/LungInv_beta.csv", index_col=0)

meta_df = pd.read_csv("../example/data/LungInv_pheno.csv", index_col=0)

print(meta_df['Group'].value_counts())
```

```
1. Loading Lung Pre-cancerous Lesions Dataset...
Group
LCIS->LC\nN=22    22
N\nN=21           21
LCIS\nN=13        13
Name: count, dtype: int64
```

In [5]:
```
print("2. Calculating Mitotic Ages...")

# Batch calculate all mitotic clocks using the "mitotic" group alias.
mitotic_ages = omniage.cal_epimarker(beta_df, clocks="Mitotic",ages=meta_df['Age

print("Calculation complete. Preview of results:")
print(mitotic_ages.head())

df_analysis = pd.concat([meta_df, mitotic_ages], axis=1)
```

```
2. Calculating Mitotic Ages...
Calculating 9 clocks: EpiTOC1, EpiTOC2, EpiTOC3, StemTOCvitro, StemTOC, RepliTal
i, HypoClock, EpiCMIT_Hyper, EpiCMIT_Hypo
Running EpiCMIT_Hypo: 100%|██████████| 9/9 [00:00<00:00, 134.90it/s]
Calculation complete. Preview of results:
       EpiTOC1  EpiTOC2_tnsc  EpiTOC2_tnsc2  EpiTOC2_irS  EpiTOC2_irS2  \
28_P  0.119201   4564.637928    5847.882204    78.700654    100.825555
22_P  0.240534  11718.464086   12786.691274   195.307735    213.111521
17_P  0.269947  13249.275847   14264.100680   189.275369    203.772867
51_P  0.197834   8654.082182    9799.413404   113.869502    128.939650
53_P  0.114486   4782.360555    6068.391990    64.626494     82.005297

      EpiTOC3_tnsc  EpiTOC3_tnsc2  EpiTOC3_avETOC3  EpiTOC3_irS  EpiTOC3_irS2  \
28_P   5823.244281    6843.955819         0.201146   100.400763    117.999238
22_P   8865.404239    9794.057491         0.296340   147.756737    163.234292
17_P   8830.046125    9763.777837         0.309714   126.143516    139.482541
51_P   6227.322570    7256.525208         0.235775    81.938455     95.480595
53_P   3452.675947    4581.561663         0.147211    46.657783     61.912995

      StemTOCvitro   StemTOC   RepliTali  HypoClock  EpiCMIT_Hyper  \
28_P      0.651052  0.732840   92.982206   0.157199       0.482281
22_P      0.720939  0.756422  104.562733   0.389383       0.553797
17_P      0.660729  0.687657  134.650485   0.358064       0.555286
51_P      0.449944  0.527809   95.380426   0.135785       0.466837
53_P      0.442461  0.525435  100.751312   0.383499       0.449277

      EpiCMIT_Hypo
28_P      0.282525
22_P      0.459155
17_P      0.502822
51_P      0.322827
53_P      0.482831
```

In [6]:
```
print("3. Statistical Analysis (Linear Model)...")

# --- 1. Data Cleaning & Ordinal Map ---
# Ensure Group column is clean
if df_analysis['Group'].str.contains('\n').any():
    df_analysis['Group'] = df_analysis['Group'].str.split('\n').str[0]

group_map = {"N": 1, "LCIS": 2, "LCIS->LC": 3}
df_analysis['Group_Ordinal'] = df_analysis['Group'].map(group_map)
```

```
# --- 2. Calculate P-values and Coefficients ---
results = []
target_clocks = [
    'EpiTOC1', 'EpiTOC2_irS', 'EpiTOC3_irS', 'StemTOCvitro',
    'StemTOC', 'HypoClock', 'RepliTali', 'EpiCMIT_Hyper', 'EpiCMIT_Hypo'
]

# Filter existing columns
plot_metrics = [m for m in target_clocks if m in df_analysis.columns]

for clock in plot_metrics:
    try:
        # Fit Age-Adjusted Linear Model
        formula = f"{clock} ~ Group_Ordinal + Age"
        model = smf.ols(formula=formula, data=df_analysis).fit()

        # Store results
        results.append({
            'Clock': clock,
            'Coef': model.params['Group_Ordinal'],
            'P-value': model.pvalues['Group_Ordinal']
        })
    except Exception as e:
        print(f"Skipping {clock}: {e}")

# Create the summary dataframe
res_df = pd.DataFrame(results).set_index('Clock') # Set index for faster lookup
print("Statistical analysis complete.")
print(res_df)
```

```
3. Statistical Analysis (Linear Model)...
Statistical analysis complete.
                   Coef       P-value
Clock
EpiTOC1        0.053434  1.754937e-08
EpiTOC2_irS   43.965747  3.557428e-08
EpiTOC3_irS   35.675438  4.213296e-10
StemTOCvitro   0.203316  5.064776e-13
StemTOC        0.222824  4.491548e-15
HypoClock      0.082805  4.477469e-09
RepliTali      7.514683  2.534508e-07
EpiCMIT_Hyper  0.127204  3.488365e-15
EpiCMIT_Hypo   0.063227  2.438297e-08
```

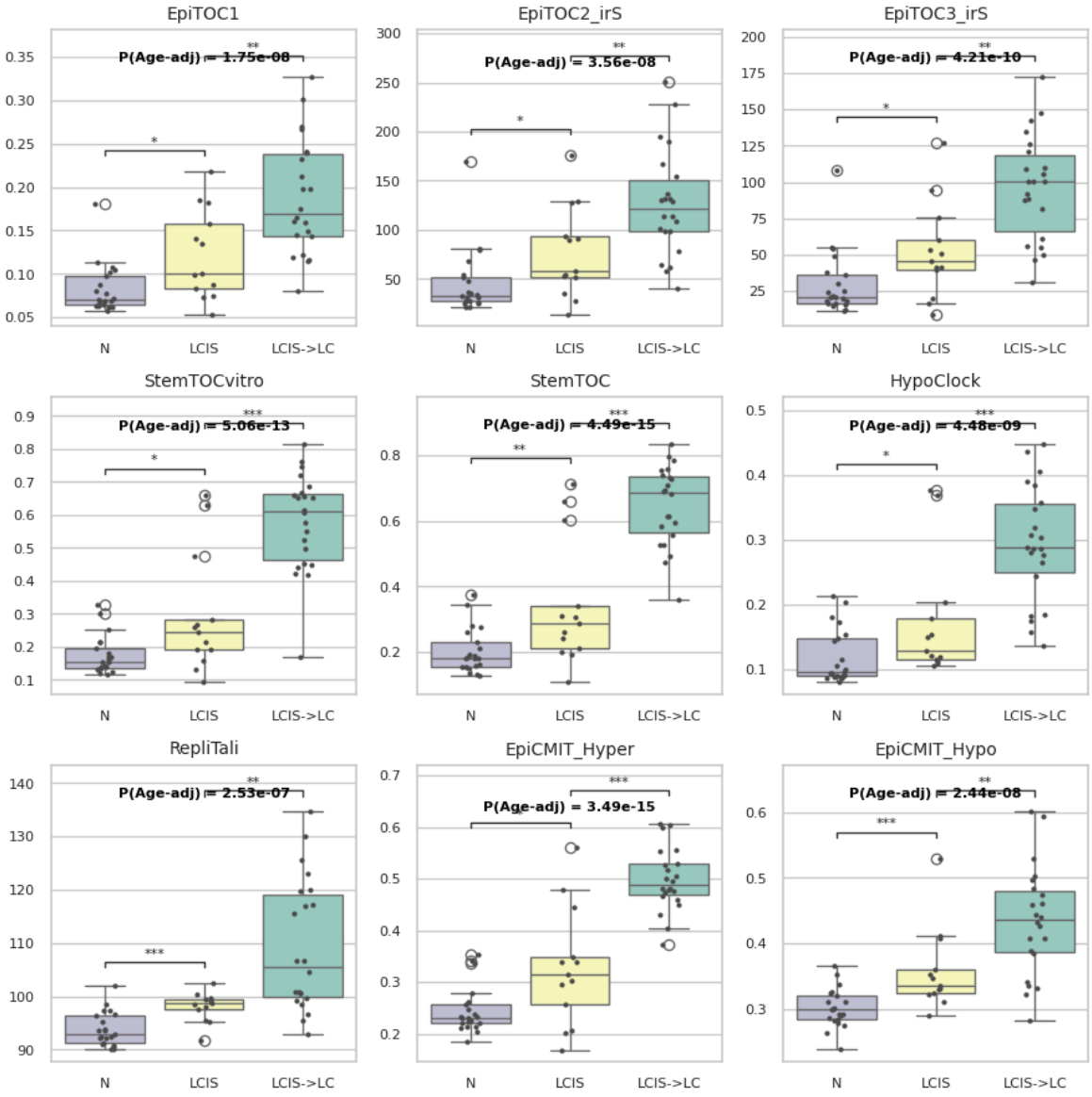
In [7]:

```
print("4. Visualization...")

sns.set_context("notebook", font_scale=0.8)

# --- Configuration ---
pairs = [("N", "LCIS"), ("LCIS", "LCIS->LC")]
order = ["N", "LCIS", "LCIS->LC"]

# --- Setup Canvas ---
n_plots = len(plot_metrics)
n_cols = 3
n_rows = math.ceil(n_plots / n_cols)

fig, axes = plt.subplots(n_rows, n_cols, figsize=(3 * n_cols, 3 * n_rows))
axes = axes.flatten()
```

```python
for i, metric in enumerate(plot_metrics):
    ax = axes[i]

    # --- A. Plotting (Boxplot + Strip) ---
    sns.boxplot(
        x='Group', y=metric, data=df_analysis,
        order=order, palette="Set3", ax=ax,
        hue='Group', legend=False,linewidth=1
    )
    sns.stripplot(
        x='Group', y=metric, data=df_analysis,
        order=order, color=".3", size=3, ax=ax
    )

    # --- B. Add Age-Adjusted P-value (Lookup from res_df) ---
    # Instead of recalculating, we just look it up!
    if metric in res_df.index:
        pval = res_df.loc[metric, 'P-value']

        # Format the P-value string
        if pval < 0.001:
            pval_text = f"P(Age-adj) = {pval:.2e}"
        else:
            pval_text = f"P(Age-adj) = {pval:.4f}"

        # Position the text at the top
        y_max = df_analysis[metric].max()
        y_min = df_analysis[metric].min()
        ax.text(
            x=1, y=y_max + (y_max - y_min) * 0.05,  # Slightly above the max
            s=pval_text, fontsize=8,
            ha='center', va='bottom', color='black', weight='bold'
        )

    # --- C. Add Pairwise Comparisons (Wilcoxon) ---
    # Note: This still runs on-the-fly because it's a different test (Group vs G
    try:
        annotator = Annotator(ax, pairs, data=df_analysis, x='Group', y=metric,
        annotator.configure(test='Mann-Whitney', text_format='star', loc='inside
        annotator.apply_and_annotate()
    except Exception as e:
        pass

    # Style adjustments
    ax.set_title(metric, fontsize=10)
    ax.set_xlabel("")
    ax.set_ylabel("")
    ax.tick_params(axis='both', labelsize=8)

# Remove empty subplots
for j in range(i + 1, len(axes)):
    fig.delaxes(axes[j])

plt.tight_layout()
plt.show()
```

```
4. Visualization...
p-value annotation legend:
      ns: 5.00e-02 < p <= 1.00e+00
       *: 1.00e-02 < p <= 5.00e-02
      **: 1.00e-03 < p <= 1.00e-02
     ***: 1.00e-04 < p <= 1.00e-03
    ****: p <= 1.00e-04

N vs. LCIS: Mann-Whitney-Wilcoxon test two-sided, P_val:1.187e-02 U_stat=6.500e+0
1
LCIS vs. LCIS->LC: Mann-Whitney-Wilcoxon test two-sided, P_val:5.396e-03 U_stat=
6.100e+01
p-value annotation legend:
      ns: 5.00e-02 < p <= 1.00e+00
       *: 1.00e-02 < p <= 5.00e-02
      **: 1.00e-03 < p <= 1.00e-02
     ***: 1.00e-04 < p <= 1.00e-03
    ****: p <= 1.00e-04

N vs. LCIS: Mann-Whitney-Wilcoxon test two-sided, P_val:1.934e-02 U_stat=7.000e+0
1
LCIS vs. LCIS->LC: Mann-Whitney-Wilcoxon test two-sided, P_val:4.855e-03 U_stat=
6.000e+01
p-value annotation legend:
      ns: 5.00e-02 < p <= 1.00e+00
       *: 1.00e-02 < p <= 5.00e-02
      **: 1.00e-03 < p <= 1.00e-02
     ***: 1.00e-04 < p <= 1.00e-03
    ****: p <= 1.00e-04

N vs. LCIS: Mann-Whitney-Wilcoxon test two-sided, P_val:2.557e-02 U_stat=7.300e+0
1
LCIS vs. LCIS->LC: Mann-Whitney-Wilcoxon test two-sided, P_val:1.413e-03 U_stat=
4.900e+01
p-value annotation legend:
      ns: 5.00e-02 < p <= 1.00e+00
       *: 1.00e-02 < p <= 5.00e-02
      **: 1.00e-03 < p <= 1.00e-02
     ***: 1.00e-04 < p <= 1.00e-03
    ****: p <= 1.00e-04

N vs. LCIS: Mann-Whitney-Wilcoxon test two-sided, P_val:2.125e-02 U_stat=7.100e+0
1
LCIS vs. LCIS->LC: Mann-Whitney-Wilcoxon test two-sided, P_val:5.299e-04 U_stat=
4.100e+01
p-value annotation legend:
      ns: 5.00e-02 < p <= 1.00e+00
       *: 1.00e-02 < p <= 5.00e-02
      **: 1.00e-03 < p <= 1.00e-02
     ***: 1.00e-04 < p <= 1.00e-03
    ****: p <= 1.00e-04

N vs. LCIS: Mann-Whitney-Wilcoxon test two-sided, P_val:7.074e-03 U_stat=6.000e+0
1
LCIS vs. LCIS->LC: Mann-Whitney-Wilcoxon test two-sided, P_val:1.617e-04 U_stat=
3.200e+01
p-value annotation legend:
      ns: 5.00e-02 < p <= 1.00e+00
       *: 1.00e-02 < p <= 5.00e-02
      **: 1.00e-03 < p <= 1.00e-02
```

```
     ***: 1.00e-04 < p <= 1.00e-03
    ****: p <= 1.00e-04

N vs. LCIS: Mann-Whitney-Wilcoxon test two-sided, P_val:1.447e-02 U_stat=6.700e+0
1
LCIS vs. LCIS->LC: Mann-Whitney-Wilcoxon test two-sided, P_val:7.717e-04 U_stat=
4.400e+01
p-value annotation legend:
      ns: 5.00e-02 < p <= 1.00e+00
       *: 1.00e-02 < p <= 5.00e-02
      **: 1.00e-03 < p <= 1.00e-02
     ***: 1.00e-04 < p <= 1.00e-03
    ****: p <= 1.00e-04

N vs. LCIS: Mann-Whitney-Wilcoxon test two-sided, P_val:5.870e-04 U_stat=3.900e+0
1
LCIS vs. LCIS->LC: Mann-Whitney-Wilcoxon test two-sided, P_val:2.004e-03 U_stat=
5.200e+01
p-value annotation legend:
      ns: 5.00e-02 < p <= 1.00e+00
       *: 1.00e-02 < p <= 5.00e-02
      **: 1.00e-03 < p <= 1.00e-02
     ***: 1.00e-04 < p <= 1.00e-03
    ****: p <= 1.00e-04

N vs. LCIS: Mann-Whitney-Wilcoxon test two-sided, P_val:4.719e-02 U_stat=8.000e+0
1
LCIS vs. LCIS->LC: Mann-Whitney-Wilcoxon test two-sided, P_val:1.227e-04 U_stat=
3.000e+01
p-value annotation legend:
      ns: 5.00e-02 < p <= 1.00e+00
       *: 1.00e-02 < p <= 5.00e-02
      **: 1.00e-03 < p <= 1.00e-02
     ***: 1.00e-04 < p <= 1.00e-03
    ****: p <= 1.00e-04

N vs. LCIS: Mann-Whitney-Wilcoxon test two-sided, P_val:9.815e-04 U_stat=4.300e+0
1
LCIS vs. LCIS->LC: Mann-Whitney-Wilcoxon test two-sided, P_val:7.363e-03 U_stat=
6.400e+01
```



Tutorial Example 3: Predict chronological age in blood tissue

The epigenetic age of 50 normal blood samples was estimated from their Illumina 450k methylation profiles using both the Horvath and Zhang epigenetic clocks.

In [8]:
```
# --- 1. Loading data ---
print("1. Loading Blood Tissue Dataset...")

beta_df = pd.read_csv("../example/data/hannum_50samples_beta.csv", index_col=0)
pheno_df = pd.read_csv("../example/data/hannum_50samples_pheno.csv", index_col=0
pheno_df['Age'] = pd.to_numeric(pheno_df['age'], errors='coerce')
```

```
1. Loading Blood Tissue Dataset...
```

In [9]:
```
# --- 2. Predict Chronological Age ---
print("\n2. Predicting Epigenetic Age...")
target_clocks = ["Horvath2013", "ZhangClock"]

pred_ages = omniage.cal_epimarker(beta_df, clocks=target_clocks,return_dict=Fals

print("Prediction preview:")
print(pred_ages.head())
```

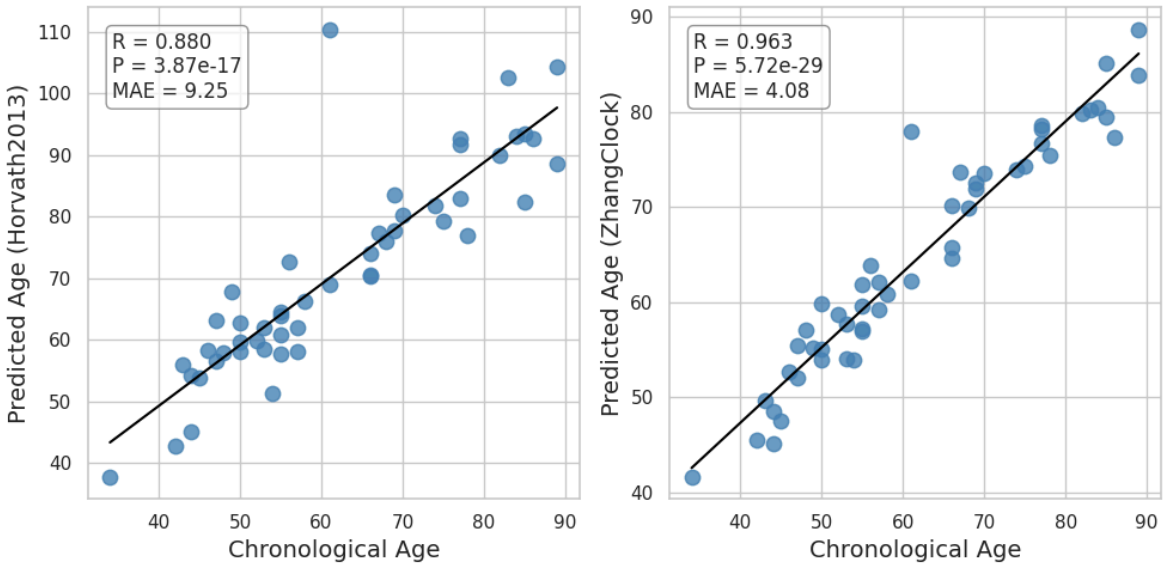
```
2. Predicting Epigenetic Age...
Calculating 2 clocks: Horvath2013, ZhangClock
Running ZhangClock: 100%|██████████| 2/2 [00:00<00:00,  4.80it/s]
Prediction preview:
           Horvath2013  ZhangClock
GSM990532    45.107894   45.162557
GSM990292    37.755842   41.681108
GSM989979    66.245765   60.825268
GSM989900    88.565909   83.879735
GSM990054    82.899726   76.667667
```

In [10]:

```python
# --- 3. Visualization & Statistics ---
print("\n3. Generating Correlation Plots...")

# Set plotting theme
sns.set_theme(style="whitegrid")
plt.rcParams.update({'font.size': 12})

# Initialize figure
# Adjusted figsize for better proportions (width=10, height=5)
fig, axes = plt.subplots(1, 2, figsize=(10, 5))
axes = axes.flatten()

for i, clock in enumerate(target_clocks):
    # Handle cases where there are fewer clocks than subplots
    if i >= len(axes): break

    ax = axes[i]

    # Skip if clock data is missing
    if clock not in pred_ages.columns:
        ax.set_visible(False)
        continue

    # --- Data Preparation ---
    # Align chronological age with predicted age based on Sample ID
    plot_data = pd.concat([pheno_df['Age'], pred_ages[clock]], axis=1, join='inn
    plot_data.columns = ['ActualAge', 'PredictedAge']

    x = plot_data['ActualAge']
    y = plot_data['PredictedAge']

    # Filter out missing values (NaNs)
    mask = ~np.isnan(x) & ~np.isnan(y)
    x_clean = x[mask]
    y_clean = y[mask]

    # --- Compute Statistical Metrics ---
    if len(x_clean) > 1:
        corr, p_value = pearsonr(x_clean, y_clean)
        mae = mean_absolute_error(x_clean, y_clean)

        # Format P-value (scientific notation for small values)
        p_text = f"{p_value:.2e}" if p_value < 0.001 else f"{p_value:.3f}"

        annotation_text = (f"R = {corr:.3f}\n"
                           f"P = {p_text}\n"
                           f"MAE = {mae:.2f}")
    else:
```

```
        annotation_text = "N/A"

    # --- Generate Plot ---
    # Use regplot for combined scatter plot and regression line
    sns.regplot(x=x_clean, y=y_clean, ax=ax,
                scatter_kws={'s': 80, 'alpha': 0.8, 'color': 'steelblue'},
                line_kws={'color': 'black', 'linewidth': 1.5}, ci=None)

    # --- Add Annotation ---
    # Anchor text at top-left relative to axes (0.05, 0.95)
    ax.text(0.05, 0.95, annotation_text, transform=ax.transAxes,
            fontsize=12, verticalalignment='top',
            bbox=dict(boxstyle="round,pad=0.3", fc="white", ec="gray", alpha=0.8

    # --- Set Labels ---
    # ax.set_title(f"{clock}", fontsize=14, fontweight='bold') # Optional
    ax.set_xlabel("Chronological Age", fontsize=14)
    ax.set_ylabel(f"Predicted Age ({clock})", fontsize=14)

plt.tight_layout()
plt.show()
```

```
3. Generating Correlation Plots...
```



Tutorial Example 4: Predict gestational age

A small demonstration dataset containing three samples, used to illustrate how to predict the gestational age of samples using DNAm Gestational Age Clocks

In [11]:
```
# --- 1. Loading data ---
print("Loading Dataset...")
beta_df = pd.read_csv("../example/data/GA_samples_beta.csv", index_col=0)
pheno_df = pd.read_csv("../example/data/GA_samples_pheno.csv", index_col=0)
if 'sample' in pheno_df.columns:
    pheno_df = pheno_df.set_index('sample')
```

```
Loading Dataset...
```

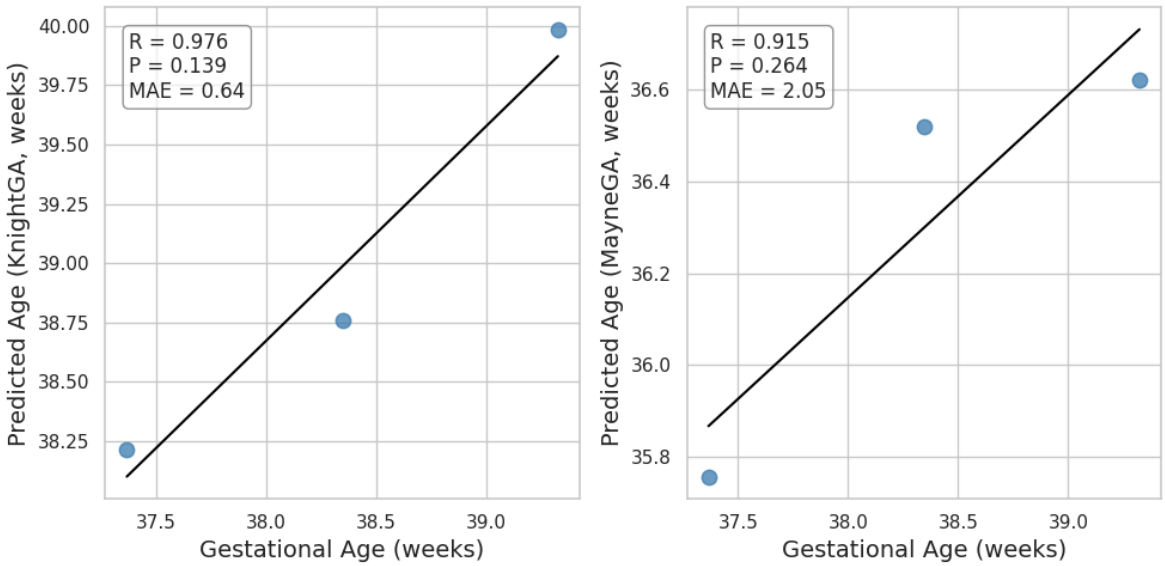
In [12]:
```
# --- 2. Predict gestational age ---
clock_names = ["KnightGA", "MayneGA"]
pred_df = omniage.cal_epimarker(beta_df, clocks=clock_names,return_dict=False)
```

```
Calculating 2 clocks: KnightGA, MayneGA
```

```
Running MayneGA: 100%|██████████| 2/2 [00:00<00:00, 70.96it/s]
```

In [13]:

```python
# --- 3. Visualization & Statistics ---
# Initialize subplots dynamically based on the number of clocks
fig, axes = plt.subplots(1, len(clock_names), figsize=(10, 5))
# Ensure axes is iterable even if there's only one plot
if len(clock_names) == 1: axes = [axes]

for i, clock in enumerate(clock_names):
    ax = axes[i]

    # 1. Extract predicted age (Y-axis)
    y_pred = pred_df[clock]

    # 2. Retrieve ground truth age (X-axis) aligned by sample index
    x_true = pheno_df.loc[y_pred.index, 'age']

    # 3. Filter out missing values (NaNs) for statistical validity
    mask = ~np.isnan(x_true) & ~np.isnan(y_pred)
    x_clean = x_true[mask]
    y_clean = y_pred[mask]

    # 4. Compute performance metrics (Pearson R, P-value, MAE)
    if len(x_clean) > 1:
        corr, p_value = pearsonr(x_clean, y_clean)
        mae = mean_absolute_error(x_clean, y_clean)

        # Format P-value (use scientific notation for p < 0.001)
        p_text = f"{p_value:.2e}" if p_value < 0.001 else f"{p_value:.3f}"

        annotation_text = (f"R = {corr:.3f}\n"
                           f"P = {p_text}\n"
                           f"MAE = {mae:.2f}")
    else:
        annotation_text = "N/A"

    # 5. Generate scatter plot with linear regression line
    sns.regplot(x=x_clean, y=y_clean, ax=ax,
                scatter_kws={'s': 80, 'alpha': 0.8, 'color': 'steelblue'},
                line_kws={'color': 'black', 'linewidth': 1.5}, ci=None)

    # 6. Annotate plot with statistical metrics
    ax.text(0.05, 0.95, annotation_text, transform=ax.transAxes,
            fontsize=12, verticalalignment='top',
            bbox=dict(boxstyle="round,pad=0.3", fc="white", ec="gray", alpha=0.8

    # 7. Configure axis labels
    ax.set_xlabel("Gestational Age (weeks)", fontsize=14)
    ax.set_ylabel(f"Predicted Age ({clock}, weeks)", fontsize=14)

plt.tight_layout()
plt.show()
```



Tutorial Example 5: Apply sc-ImmuAging to PBMC scRNA-seq dataset

In this example, we showcase the application of the cell-type-specific clock, scImmuAging, to a PBMC scRNA-seq dataset. The dataset includes 20 samples, profiled to contain only CD4+ and CD8+ T cells.

Loading data adata = sc.read_h5ad("../example/data//Yazar_CD4T_CD8T_example.h5ad") print(f"Data loaded successfully. The dataset contains {adata.n_obs} cells and {adata.n_vars} genes.")

In [15]:
```
seed = 42
np.random.seed(seed)
scImmuAging_model = omniage.models.scImmuAging()
target_cell_types = ["CD4T", "CD8T"]
# Execute scImmuAging prediction
print("\nInitiating scImmuAging prediction...")

try:
    sc_immu_out = scImmuAging_model.predict(
        adata,
        cell_types=target_cell_types,
        verbose=True
    )
    print("\nPrediction complete.")

    for ct in sc_immu_out:
        print(f"\n--- {ct} Results Preview (Donor-level) ---")
        # Access the 'Donor' level results
        donor_df = sc_immu_out[ct]['Donor']
        print(donor_df.head())

except Exception as e:
    print(f"\nExecution failed: {e}")
    import traceback
    traceback.print_exc()
```

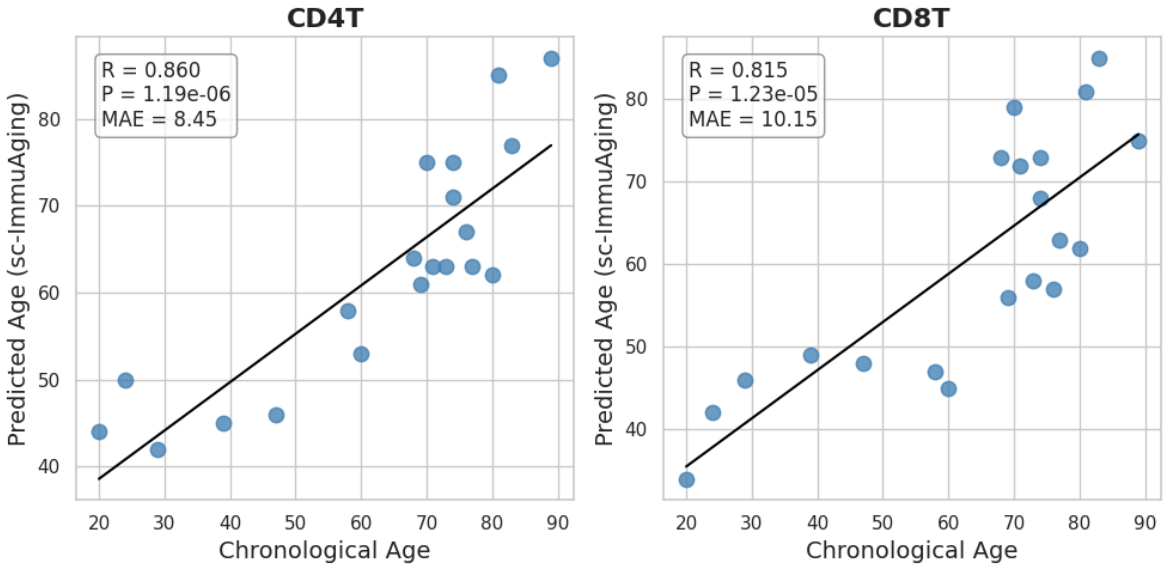
```
Initiating scImmuAging prediction...

--- Processing cell type: CD4T ---
Generating pseudocells (Bootstrap)...
```

```
/mnt/local-disk/data/duzhaozhen/OmniAge_py/src/omniage/models/transcriptomic.py:4
5: FutureWarning: The default of observed=False is deprecated and will be changed
to True in a future version of pandas. Pass observed=False to retain current beha
vior or observed=True to adopt the future default and silence this warning.
  grouped = adata.obs.groupby(['donor_id', 'age'])
Predicting age...
Aggregating per donor...
Prediction complete for CD4T!

--- Processing cell type: CD8T ---
Generating pseudocells (Bootstrap)...
/mnt/local-disk/data/duzhaozhen/OmniAge_py/src/omniage/models/transcriptomic.py:4
5: FutureWarning: The default of observed=False is deprecated and will be changed
to True in a future version of pandas. Pass observed=False to retain current beha
vior or observed=True to adopt the future default and silence this warning.
  grouped = adata.obs.groupby(['donor_id', 'age'])
Predicting age...
Aggregating per donor...
Prediction complete for CD8T!

Prediction complete.

--- CD4T Results Preview (Donor-level) ---
    donor_id  age  predicted
0  1006_1007   73       63.0
1  1037_1038   58       58.0
2  1055_1056   71       63.0
3    238_239   39       45.0
4    258_259   68       64.0

--- CD8T Results Preview (Donor-level) ---
    donor_id  age  predicted
0  1006_1007   73       58.0
1  1037_1038   58       47.0
2  1055_1056   71       72.0
3    238_239   39       49.0
4    258_259   68       73.0
```

In [16]:

```python
# Set plotting theme and parameters
sns.set_theme(style="whitegrid")
plt.rcParams.update({'font.size': 12})

# Initialize figure and subplots
fig, axes = plt.subplots(1, len(target_cell_types), figsize=(10, 5))
if len(target_cell_types) == 1: axes = [axes]

for i, ct in enumerate(target_cell_types):
    ax = axes[i]
    # Retrieve donor-level results for the current cell type
    df = sc_immu_out[ct]['Donor']

    # Extract data (Ground Truth vs. Predicted)
    x_true = df['age']
    y_pred = df['predicted']

    # Filter out missing values (NaNs)
    mask = ~np.isnan(x_true) & ~np.isnan(y_pred)
    x_clean = x_true[mask]
    y_clean = y_pred[mask]
```

```
    # --- Compute Performance Metrics ---
    if len(x_clean) > 1:
        corr, p_value = pearsonr(x_clean, y_clean)
        mae = mean_absolute_error(x_clean, y_clean)

        # Format P-value (use scientific notation for p < 0.001)
        p_text = f"{p_value:.2e}" if p_value < 0.001 else f"{p_value:.3f}"

        annotation_text = (f"R = {corr:.3f}\n"
                           f"P = {p_text}\n"
                           f"MAE = {mae:.2f}")
    else:
        annotation_text = "N/A"

    # --- Generate Plot ---
    # Create scatter plot with linear regression line
    sns.regplot(x=x_clean, y=y_clean, ax=ax,
                scatter_kws={'s': 80, 'alpha': 0.8, 'color': 'steelblue'},
                line_kws={'color': 'black', 'linewidth': 1.5}, ci=None)

    # --- Add Statistical Annotation ---
    ax.text(0.05, 0.95, annotation_text, transform=ax.transAxes,
            fontsize=12, verticalalignment='top',
            bbox=dict(boxstyle="round,pad=0.3", fc="white", ec="gray", alpha=0.8

    # --- Set Labels and Titles ---
    ax.set_title(ct, fontsize=16, fontweight='bold')
    ax.set_xlabel("Chronological Age", fontsize=14)
    ax.set_ylabel("Predicted Age (sc-ImmuAging)", fontsize=14)

plt.tight_layout()
plt.show()
```



Tutorial Example 6: Apply Brain_CT_clock to Brain snRNA-seq dataset

In this example, we showcase the application of the brain cell-type-specific aging Clocks to a brain snRNA-seq dataset. The dataset includes 15 donors, profiled to contain only Oligodendrocytes.

In [17]:
```
# --- 1. Load Data ---
# Load the example brain dataset (15 donors, Oligodendrocytes)
# Assuming omniage provides a loader, or use sc.read_h5ad()
adata =  sc.read_h5ad("../example/data/brain_frohlich_control_example.h5ad")
print(f"Data loaded successfully. Observations: {adata.n_obs}, Features: {adata.
```

```
Data loaded successfully. Observations: 12620, Features: 26195.
```

In [18]:
```
# --- 2. Run Brain_CT_clock ---
# Initialize the clock model
brain_clock = omniage.models.BrainCTClock()

# Execute prediction for all modes ('SC', 'Pseudobulk', 'Bootstrap')
# This returns a dictionary: {'SC': df1, 'Pseudobulk': df2, 'Bootstrap': df3}
brain_clock_results = brain_clock.predict(
    adata,
    cell_types=["Oligodendrocytes"],
    model_name="all"
)
```

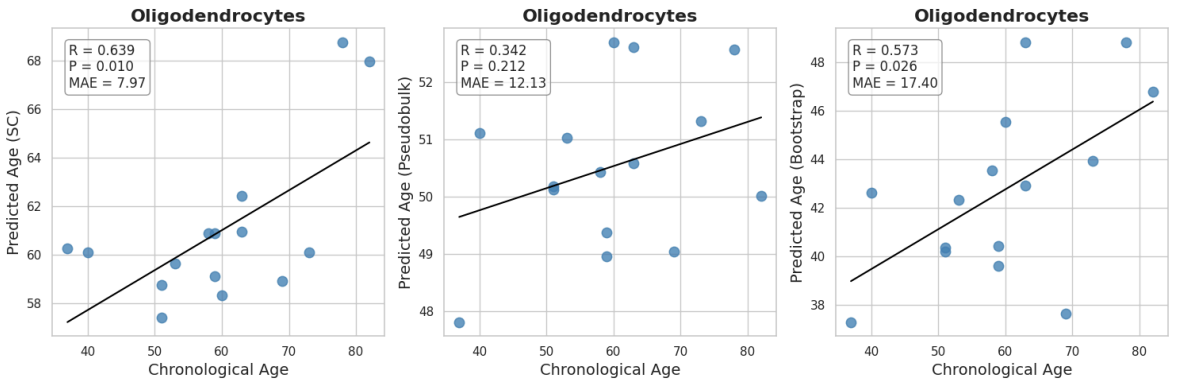
```
Starting Brain_CT_clock prediction...
Running mode: SC
Running mode: Pseudobulk
Running mode: Bootstrap
--- All Brain_CT_clock predictions complete! ---
```

In [19]:
```
# --- 3. Data Aggregation (Replicating dplyr logic) ---
# We need to average predictions per donor for robust validation
average_predictions_by_donor = {}

for mode, df in brain_clock_results.items():
    # Group by 'donors' and calculate mean of predictions and ages
    # (Age is constant per donor, so mean() keeps it unchanged)
    agg_df = df.groupby('donors')[['predictions', 'ages']].mean()
    average_predictions_by_donor[mode] = agg_df
```

```
/tmp/ipykernel_2187326/1072624082.py:8: FutureWarning: The default of observed=Fa
lse is deprecated and will be changed to True in a future version of pandas. Pass
observed=False to retain current behavior or observed=True to adopt the future de
fault and silence this warning.
  agg_df = df.groupby('donors')[['predictions', 'ages']].mean()
/tmp/ipykernel_2187326/1072624082.py:8: FutureWarning: The default of observed=Fa
lse is deprecated and will be changed to True in a future version of pandas. Pass
observed=False to retain current behavior or observed=True to adopt the future de
fault and silence this warning.
  agg_df = df.groupby('donors')[['predictions', 'ages']].mean()
```

In [20]:
```
# --- 4. Visualization ---
# Define modes to plot (ensure order matches R example)
modes = ['SC', 'Pseudobulk', 'Bootstrap']

# Set plotting theme
sns.set_theme(style="whitegrid")
plt.rcParams.update({'font.size': 12})

# Initialize figure (1 row, 3 columns)
fig, axes = plt.subplots(1, 3, figsize=(15, 5))

for i, mode in enumerate(modes):
    ax = axes[i]
```

```
    # Check if mode exists in results
    if mode not in average_predictions_by_donor:
        ax.text(0.5, 0.5, "Mode not run", ha='center')
        continue

    # Retrieve aggregated data
    df = average_predictions_by_donor[mode]
    x_true = df['ages']
    y_pred = df['predictions']

    # Filter out missing values
    mask = ~np.isnan(x_true) & ~np.isnan(y_pred)
    x_clean = x_true[mask]
    y_clean = y_pred[mask]

    # --- Compute Performance Metrics ---
    if len(x_clean) > 1:
        corr, p_value = pearsonr(x_clean, y_clean)
        mae = mean_absolute_error(x_clean, y_clean)

        # Format P-value
        p_text = f"{p_value:.2e}" if p_value < 0.001 else f"{p_value:.3f}"

        annotation_text = (f"R = {corr:.3f}\n"
                           f"P = {p_text}\n"
                           f"MAE = {mae:.2f}")
    else:
        annotation_text = "N/A"

    # --- Generate Plot ---
    sns.regplot(x=x_clean, y=y_clean, ax=ax,
                scatter_kws={'s': 80, 'alpha': 0.8, 'color': 'steelblue'},
                line_kws={'color': 'black', 'linewidth': 1.5}, ci=None)

    # --- Add Statistical Annotation ---
    ax.text(0.05, 0.95, annotation_text, transform=ax.transAxes,
            fontsize=12, verticalalignment='top',
            bbox=dict(boxstyle="round,pad=0.3", fc="white", ec="gray", alpha=0.8

    # --- Set Labels and Titles ---
    ax.set_title("Oligodendrocytes", fontsize=16, fontweight='bold')
    ax.set_xlabel("Chronological Age", fontsize=14)
    ax.set_ylabel(f"Predicted Age ({mode})", fontsize=14)

plt.tight_layout()
plt.show()
```



Tutorial Example 7: Apply Pasta to Brain (MTG) scRNA-seq dataset

In this example, we showcase the application of the Pasta clock to a single-nuclei RNA-seq dataset from the middle temporal gyrus (MTG) (Gabitto et al., 2024). The dataset includes 46 healthy donors and is filtered to contain extratelencephalic projecting glutamatergic cortical neurons.

In [21]:
```
# --- 1. Load Data ---
adata = sc.read_h5ad("../example/data/Gabitto_2024_filtered.h5ad")
# --- 2. Create Pseudobulk Samples ---
print("Generating pseudobulk samples (chunk_size=512)...")
seed = 42
np.random.seed(seed)
adata_bulk = omniage.utils.PASTA_create_pasta_pseudobulks(
    adata,
    pool_by=["cell_type", "age"],
    chunk_size=512
)
sc.pp.normalize_total(adata_bulk, target_sum=1e4)
sc.pp.log1p(adata_bulk)
```

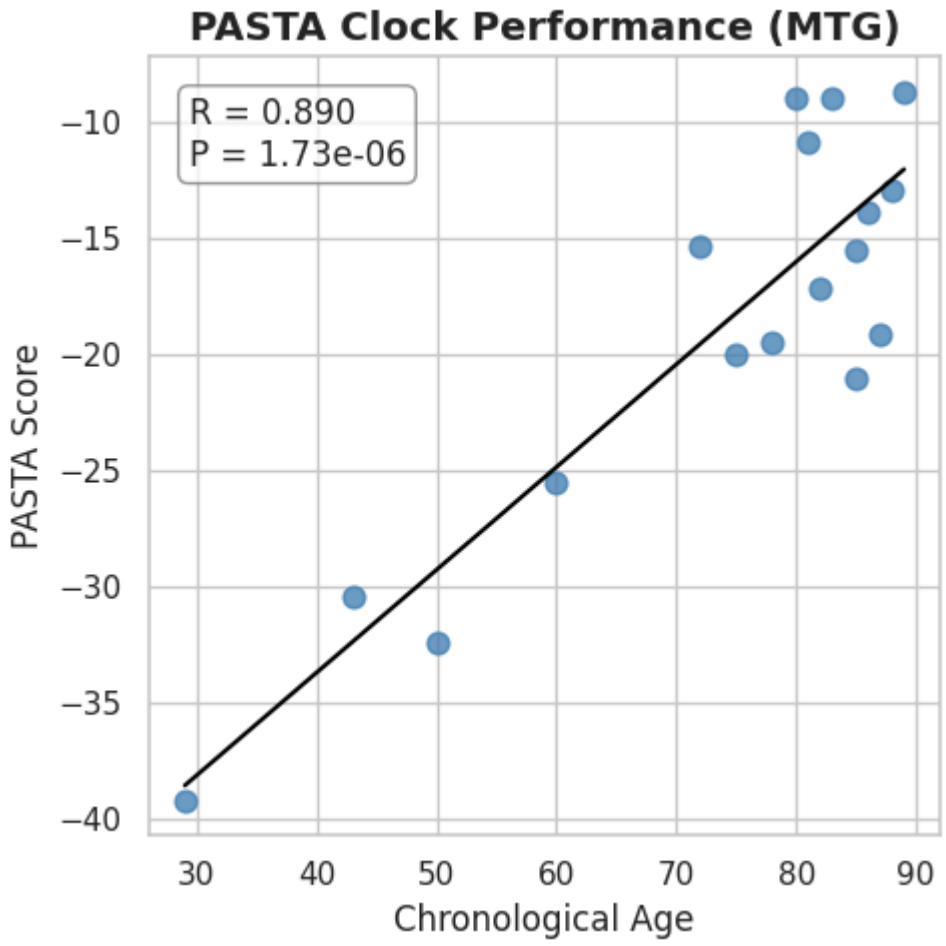
```
Generating pseudobulk samples (chunk_size=512)...
Aggregating 1385 cells into pseudobulks...
Created 17 pseudobulk samples.
```

In [22]:
```
# --- 3. Prediction ---
pasta = omniage.models.PASTA_Clock(model_type="PASTA")
pred_df = pasta.predict(adata_bulk, rank_norm=True)

print(pred_df.head())
```

```
                                                    Predicted_PASTA_Age
L5 extratelencephalic projecting glutamatergic ...           -39.156992
L5 extratelencephalic projecting glutamatergic ...           -30.426580
L5 extratelencephalic projecting glutamatergic ...           -32.358381
L5 extratelencephalic projecting glutamatergic ...           -25.486817
L5 extratelencephalic projecting glutamatergic ...           -15.357963
```

In [23]:
```
# --- 4. Visualization ---

# --- Data Preparation ---
# Align ground truth age with predicted scores based on sample index
x_true = adata_bulk.obs.loc[pred_df.index, 'age']
y_pred = pred_df.iloc[:, 0]  # Extract prediction scores

# Filter out missing values (NaNs) to ensure statistical validity
mask = ~np.isnan(x_true) & ~np.isnan(y_pred)
x_clean = x_true[mask]
y_clean = y_pred[mask]

# --- Plot Configuration ---
sns.set_theme(style="whitegrid")
plt.rcParams.update({'font.size': 12})
# Initialize figure (slightly larger than original R dimensions for clarity)
fig, ax = plt.subplots(figsize=(5, 5))

# --- Compute Statistical Metrics ---
if len(x_clean) > 1:
    corr, p_value = pearsonr(x_clean, y_clean)
```

```
    # Format P-value (scientific notation for p < 0.001)
    p_text = f"{p_value:.2e}" if p_value < 0.001 else f"{p_value:.3f}"

    annotation_text = (f"R = {corr:.3f}\n"
                       f"P = {p_text}")
else:
    annotation_text = "N/A"

# --- Generate Scatter Plot ---
# Plot data points with linear regression line
sns.regplot(x=x_clean, y=y_clean, ax=ax,
            scatter_kws={'s': 60, 'alpha': 0.8, 'color': 'steelblue'},
            line_kws={'color': 'black', 'linewidth': 1.5}, ci=None)

# --- Add Annotation ---
# Display correlation metrics in the top-left corner
ax.text(0.05, 0.95, annotation_text, transform=ax.transAxes,
        fontsize=12, verticalalignment='top',
        bbox=dict(boxstyle="round,pad=0.3", fc="white", ec="gray", alpha=0.8))

# --- Set Labels ---
ax.set_title("PASTA Clock Performance (MTG)", fontsize=14, fontweight='bold')
ax.set_xlabel("Chronological Age", fontsize=12)
ax.set_ylabel("PASTA Score", fontsize=12)

plt.tight_layout()
plt.show()
```



Tutorial Example 8: Calculate CPR and CHIP Scores

Here, we use DNAm data to calculate the CRP score and CHIP score.

In [24]:
```
# --- 1. Load Example Data ---
beta_df = pd.read_csv("../example/data/hannum_50samples_beta.csv", index_col=0)
pheno_df = pd.read_csv("../example/data/hannum_50samples_pheno.csv", index_col=0
```

In [25]:
```
# --- 2. Calculate CRP Score ---
print("Calculating CRP Score...")
crp_results = omniage.cal_epimarker(beta_df, clocks=["CompCRP"],return_dict=Fals

print("\n--- CRP Results  ---")
if isinstance(crp_results, dict):
    for key, df in crp_results.items():
        print(f"[{key}]:")
        print(df.head(5))
else:
    print(crp_results.head(5))
```

```
Calculating CRP Score...
Calculating 1 clocks: CompCRP
Running CompCRP: 100%|██████████| 1/1 [00:00<00:00, 14.77it/s]
--- CRP Results  ---
           CompCRP_CRP  CompCRP_intCRP
GSM990532    -0.406504       -0.296251
GSM990292     0.300220        0.237751
GSM989979    -0.146511       -0.064922
GSM989900    -0.149023        0.038612
GSM990054    -0.385788       -0.462057
```

In [26]:
```
# --- 3. Calculate CHIP Score ---
print("\nCalculating CHIP Score...")
chip_results = omniage.cal_epimarker(beta_df, clocks=["CompCHIP"],return_dict=Fa

print("\n--- CHIP Results ---")
if isinstance(chip_results, dict):
    for key, df in chip_results.items():
        print(f"[{key}]:")
        print(df.head(5))
else:
    print(chip_results.head(5))
```

```
Calculating CHIP Score...
Calculating 1 clocks: CompCHIP
Running CompCHIP: 100%|██████████| 1/1 [00:00<00:00,  8.72it/s]
--- CHIP Results ---
           CompCHIP_TET2  CompCHIP_AnyCHIP  CompCHIP_DNMT3A  CompCHIP_ASXL1
GSM990532      -0.187777         -0.503069        -1.568649       -0.133442
GSM990292      -0.541435         -0.359947        -0.867351       -0.538446
GSM989979       0.267374          0.006809        -0.109840        0.090336
GSM989900      -0.504409         -0.648675        -0.311165       -0.501661
GSM990054       0.178283          0.353747         1.181109        0.176207
```

Tutorial Example 9: Celltype fraction clock based on DNAm

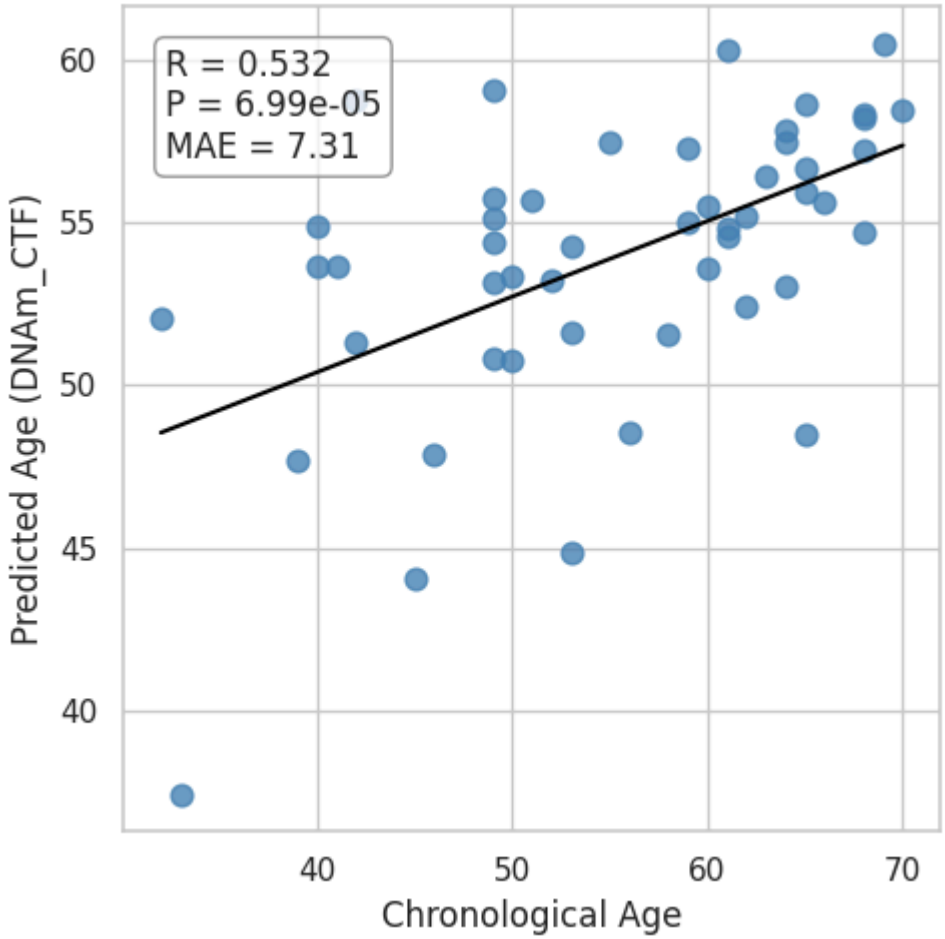
We apply the DNAm Cell-type Fraction (CTF) Clock to 50 blood samples (EPIC array) from the TZH cohort to predict chronological age.

In [27]:
```
# --- 1. Load Data ---
ctf_df = pd.read_csv("../example/data/TZH_50samples_Fraction.csv", index_col=0)
pheno_df = pd.read_csv("../example/data/TZH_50samples_pheno.csv", index_col=0)
pheno_df = pheno_df.set_index("Sample")
```

In [28]:
```
# --- 2. Run Prediction ---
print("Running DNAm CTF Clock prediction...")
#ctf_clock = omniage..DNAmCTFClock()
pred_ages = omniage.cal_epimarker(ctf = ctf_df,clocks=["DNAmCTFClock"], return_d
# --- 3. Data Alignment ---
pheno_df["PredictedAge"] = pred_ages["DNAmCTFClock"]
```

```
Running DNAm CTF Clock prediction...
Calculating 1 clocks: DNAmCTFClock
Running DNAmCTFClock: 100%|██████████| 1/1 [00:11<00:00, 11.51s/it]
```

In [29]:
```
# --- 3. Visualization ---
x_true = pheno_df["Age"]
y_pred = pheno_df["PredictedAge"]

mask = ~np.isnan(x_true) & ~np.isnan(y_pred)
x_clean = x_true[mask]
y_clean = y_pred[mask]


sns.set_theme(style="whitegrid")
plt.rcParams.update({'font.size': 12})
fig, ax = plt.subplots(figsize=(5, 5))

# --- Compute Statistical Metrics ---
if len(x_clean) > 1:
    corr, p_value = pearsonr(x_clean, y_clean)
    mae = mean_absolute_error(x_clean, y_clean)
    p_text = f"{p_value:.2e}" if p_value < 0.001 else f"{p_value:.3f}"

    annotation_text = (f"R = {corr:.3f}\n"
                       f"P = {p_text}\n"
                       f"MAE = {mae:.2f}")
else:
    annotation_text = "N/A"

# --- Generate Scatter Plot ---
sns.regplot(x=x_clean, y=y_clean, ax=ax,
            scatter_kws={'s': 60, 'alpha': 0.8, 'color': 'steelblue'},
            line_kws={'color': 'black', 'linewidth': 1.5}, ci=None)

# --- Add Annotation ---
ax.text(0.05, 0.95, annotation_text, transform=ax.transAxes,
        fontsize=12, verticalalignment='top',
        bbox=dict(boxstyle="round,pad=0.3", fc="white", ec="gray", alpha=0.8))

# --- Set Labels ---
ax.set_xlabel("Chronological Age", fontsize=12)
ax.set_ylabel("Predicted Age (DNAm_CTF)", fontsize=12)

plt.tight_layout()
plt.show()
```


R = 0.532
P = 6.99e-05
MAE = 7.31
Predicted Age (DNAm_CTF)
Chronological Age
60
55
50
45
40
40
50
60
70

In [ ]:

In [ ]:

In [ ]: