## Descriptive analysis

**proc** **tabulate** data=Original.MCI\_nested;

Class MCI APOE\_status Bloodtype breast n\_31\_0\_0 race\_n education1 income smoking\_B alcohol\_c PA\_level sedentary\_c sleep\_c diet\_c insulin\_c depressed\_c restless\_c anxious Lone ISO\_c hear\_c hearaid\_c Frailty\_cc CVD MS RD DD MSK Cancer ;

Table( APOE\_status Bloodtype breast n\_31\_0\_0 race\_n education1 income smoking\_B alcohol\_c PA\_level sedentary\_c sleep\_c diet\_c insulin\_c depressed\_c restless\_c anxious Lone ISO\_c hear\_c hearaid\_c Frailty\_cc CVD MS RD DD MSK Cancer ), MCI \*(n colpctn="%")all;

**run**;

**proc** **tabulate** data=Original.MCID\_nested;

Class MCI\_D APOE\_status Bloodtype breast n\_31\_0\_0 race\_n education1 income smoking\_B alcohol\_c PA\_level sedentary\_c sleep\_c diet\_c insulin\_c depressed\_c restless\_c anxious Lone ISO\_c hear\_c hearaid\_c Frailty\_cc CVD MS RD DD MSK Cancer;

Table( APOE\_status Bloodtype breast n\_31\_0\_0 race\_n education1 income smoking\_B alcohol\_c PA\_level sedentary\_c sleep\_c diet\_c insulin\_c depressed\_c restless\_c anxious Lone ISO\_c hear\_c hearaid\_c Frailty\_cc CVD MS RD DD MSK Cancer), MCI\_D \*(n colpctn="%")all;

**run**;

**PROC** **means** data=Original.MCI\_nested min q1 mean median q3 max std; where MCI=**0**; var Birthweight T1age n\_189\_0\_0 n\_24006\_0\_0 n\_24005\_0\_0 n\_24003\_0\_0 n\_24024\_0\_0 bmi\_B Grip DBP SBP LDL HbA1c VD;**run**;

**PROC** **means** data=Original.MCI\_nested min q1 mean median q3 max std; where MCI=**1**; var Birthweight T1age n\_189\_0\_0 n\_24006\_0\_0 n\_24005\_0\_0 n\_24003\_0\_0 n\_24024\_0\_0 bmi\_B Grip DBP SBP LDL HbA1c VD;**run**;

**PROC** **means** data=Original.MCID\_nested min q1 mean median q3 max std; where MCI\_D=**0**; var Birthweight T1age n\_189\_0\_0 n\_24006\_0\_0 n\_24005\_0\_0 n\_24003\_0\_0 n\_24024\_0\_0 bmi\_B Grip DBP SBP LDL HbA1c VD;**run**;

**PROC** **means** data=Original.MCID\_nested min q1 mean median q3 max std; where MCI\_D=**1**; var Birthweight T1age n\_189\_0\_0 n\_24006\_0\_0 n\_24005\_0\_0 n\_24003\_0\_0 n\_24024\_0\_0 bmi\_B Grip DBP SBP LDL HbA1c VD;**run**;

## Nested case-control study

### Matched case-control

#### MCI

**data** Original.MCI;

set Original.MCI\_nested;

**%MACRO** GMATCH(DATA=, GROUP= , ID=,

MVARS= ,WTS=,DMAXK=,dmax= , DIST=,

NCONTLS=, TIME=timex,TRANSF=,

SEEDCA=, SEEDCO=, PRINT=,

OUT=,OUTNMCA=,OUTNMCO=);

%LET BAD=0;

%IF %LENGTH(&DATA)=**0** %THEN %DO;

%PUT ERROR: NO DATASET SUPPLIED;

%LET BAD=1;

%END;

%IF %LENGTH(&ID)=**0** %THEN %DO;

%PUT ERROR: NO ID VARIABLE SUPPLIED;

%LET BAD=1;

%END;

%IF %LENGTH(&GROUP)=**0** %THEN %DO;

%PUT ERROR: NO CASE(1)/CONTROL(0) GROUP VARIABLE SUPPLIED;

%LET BAD=1;

%END;

%IF %LENGTH(&MVARS)=**0** %THEN %DO;

%PUT ERROR: NO MATCHING VARIABLES SUPPLIED;

%LET BAD=1;

%END;

%IF %LENGTH(&WTS)=**0** %THEN %DO;

%PUT ERROR: NO WEIGHTS SUPPLIED;

%LET BAD=1;

%END;

%LET NVAR=0;

%DO %UNTIL(%SCAN(&MVARS,&NVAR+**1**,' ')= );

%LET NVAR=%EVAL(&NVAR+1);

%END;

%LET NWTS=0;

%DO %UNTIL(%QSCAN(&WTS,&NWTS+**1**,' ')= );

%LET NWTS=%EVAL(&NWTS+1);

%END;

%IF &NVAR^= &NWTS %THEN %DO;

%PUT ERROR: #VARS MUST EQUAL #WTS;

%LET BAD=1;

%END;

%LET NK=0;

%IF %QUOTE(&DMAXK)^= %THEN %DO %UNTIL(%QSCAN(&DMAXK,&NK+**1**,' ')= );

%LET NK=%EVAL(&NK+1);

%END;

%IF &NK>&NVAR %THEN %LET NK=&NVAR;

%DO I=**1** %TO &NVAR;

%LET V&I=%SCAN(&MVARS,&I,' ');

%END;

%IF &NWTS>**0** %THEN %DO;

DATA \_NULL\_;

%DO I=**1** %TO &NWTS;

%LET W&I=%SCAN(&WTS,&I,' ');

IF &&W&I<**0** THEN DO;

PUT 'ERROR: WEIGHTS MUST BE NON-NEGATIVE';

CALL SYMPUT('BAD','1');

END;

%END;

RUN;

%END;

%IF &NK>**0** %THEN %DO;

DATA \_NULL\_;

%DO I=**1** %TO &NK;

%LET K&I=%SCAN(&DMAXK,&I,' ');

IF &&K&I<**0** THEN DO;

PUT 'ERROR: DMAXK VALUES MUST BE NON-NEGATIVE';

CALL SYMPUT('BAD','1');

END;

%END;

RUN;

%END;

**%MACRO** ***MAX1***;

%IF &DMAX^= %THEN %DO;

& \_\_D<=&DMAX

%END;

%DO I=**1** %TO &NK;

& ABS(\_\_CA&I-\_\_CO&I)<=&&K&I

%END;

**%MEND** MAX1;

**%macro** ***greedy***;

%GLOBAL BAD2;

data \_\_CHECK; set &DATA;

\_\_id=&id;

if \_\_id="" then delete;

%DO I=**1** %TO &NVAR;

IF %scan(&mvars,&i)=**.** THEN DELETE;

%END;

%IF &TIME^= %THEN %DO;

IF &TIME=**.** THEN DELETE;

%END;

run;

%if &transf=**1** %then %do;

proc standard data=\_\_check m=**0** s=**1** out=\_stdzd; var &mvars;

data \_caco;

set \_stdzd;

%end;

%if &transf=**2** %then %do;

proc rank data=\_\_check out=\_ranks; var &mvars;

data \_caco;

set \_ranks;

%end;

%if &transf=**0** %then %do;

data \_caco;

set \_\_check;

%end;

DATA \_\_CASE; SET \_caco;

if &group=**1**;

DATA \_\_CASE; SET \_\_CASE END=EOF;

KEEP \_\_IDCA \_\_CA1-\_\_CA&NVAR \_\_R &mvars

%if &time^= %then %do;

\_\_catime

%end;

;

\_\_IDCA=&ID;

%if &time^= %then %do;

\_\_catime=&time;

%end;

%DO I=**1** %TO &NVAR;

\_\_CA&I=&&V&I;

%END;

%if &seedca^= %then %do;

SEED=&SEEDCA;

\_\_R=RANUNI( SEED );

%end;

%else %do;

\_\_R=**1**;

%end;

IF EOF THEN CALL SYMPUT('NCA',\_N\_);

PROC SORT; BY \_\_R \_\_IDCA;

DATA \_\_CONT; SET \_caco;

if &group=**0**;

DATA \_\_CONT; SET \_\_CONT END=EOF;

KEEP \_\_IDCO \_\_CO1-\_\_CO&NVAR \_\_R &mvars

%if &time^= %then %do;

\_\_cotime

%end;

;

\_\_IDCO=&ID;

%if &time^= %then %do;

\_\_cotime=&time;

%end;

%DO I=**1** %TO &NVAR;

\_\_CO&I=&&V&I;

%END;

%if &seedco^= %then %do;

SEED=&SEEDCo;

\_\_R=RANUNI( SEED );

%end;

%else %do;

\_\_R=**1**;

%end;

IF EOF THEN CALL SYMPUT('NCO',\_N\_);

RUN;

%LET BAD2=0;

%IF &NCO < %EVAL(&NCA\*&NCONTLS) %THEN %DO;

%PUT ERROR: NOT ENOUGH CONTROLS TO MAKE REQUESTED MATCHES;

%LET BAD2=1;

%END;

%IF &BAD2=**0** %THEN %DO;

PROC SORT; BY \_\_R \_\_IDCO;

DATA \_\_MATCH;

KEEP \_\_IDCA \_\_CA1-\_\_CA&NVAR \_\_DIJ \_\_MATCH \_\_CONT\_N

%if &time^= %then %do;

\_\_catime \_\_cotime

%end;

;

ARRAY \_\_USED(&NCO) $ **1** \_TEMPORARY\_;

DO \_\_I=**1** TO &NCO;

\_\_USED(\_\_I)='0';

END;

DO \_\_I=**1** TO &NCONTLS;

DO \_\_J=**1** TO &NCA;

SET \_\_CASE POINT=\_\_J;

\_\_SMALL=**.**;

\_\_MATCH=**.**;

DO \_\_K=**1** TO &NCO;

IF \_\_USED(\_\_K)='0' THEN DO;

SET \_\_CONT POINT=\_\_K;

%if &dist=**2** %then %do;

\*\*wtd euclidian dist;

\_\_D= sqrt(

%do k=**1** %to &nvar;

%scan(&wts,&k)\*(\_\_ca&k - \_\_co&k)\*\***2**

%if &k<&nvar %then + ;

%end;

);

%end;

%else %do;

\_\_D=

%do k=**1** %to &nvar;

%scan(&wts,&k)\*abs(\_\_ca&k - \_\_co&k )

%if &k<&nvar %then + ;

%end;

;

%end;

IF \_\_d^=**.** & (\_\_SMALL=**.** | \_\_D<\_\_SMALL) %***MAX1***

%if &time^= %then %do;

& \_\_cotime > \_\_catime

%end;

THEN DO;

\_\_SMALL=\_\_D;

\_\_MATCH=\_\_K;

\_\_DIJ=\_\_D;

\_\_CONT\_N=\_\_I;

END;

END;

END;

IF \_\_MATCH^=**.** THEN DO;

\_\_USED(\_\_MATCH)='1';

OUTPUT;

END;

END;

END;

STOP;

DATA &OUT;

SET \_\_MATCH;

SET \_\_CONT POINT=\_\_MATCH;

KEEP \_\_IDCA \_\_IDCO \_\_CONT\_N \_\_DIJ \_\_CA1-\_\_CA&NVAR

\_\_CO1-\_\_CO&NVAR \_\_d1-\_\_d&nvar \_\_absd1-\_\_absd&nvar \_\_WT1-\_\_WT&NVAR

\_\_catime \_\_cotime \_\_dtime;

%if &time= %then %do;

\_\_cotime=**.**; \_\_catime=**.**;

%end;

LABEL

\_\_catime="&time/CASE"

\_\_cotime="&time/CONTROL"

\_\_dtime="&time/ABS. DIFF"

\_\_CONT\_N='CONTROL/NUMBER'

\_\_DIJ='DISTANCE/D\_IJ'

%DO I=**1** %TO &NVAR;

\_\_CA&I="&&V&I/CASE"

\_\_CO&I="&&V&I/CONTROL"

\_\_absd&I="&&V&I/ABS. DIFF "

\_\_d&I="&&V&I/DIFF "

\_\_WT&I="&&V&I/WEIGHT"

%END;

;

%DO I=**1** %TO &NVAR;

\_\_d&i= (\_\_CA&I-\_\_CO&I);

\_\_absd&I=abs(\_\_CA&I-\_\_CO&I);

\_\_WT&I=&&W&I;

%END;

\_\_dtime=\_\_cotime-\_\_catime;

PROC SORT DATA=&OUT; BY \_\_IDCA \_\_CONT\_N;

proc sort data=\_\_case; by \_\_IDCA;

data &outnmca; merge \_\_case

&out(in=\_\_inout where=(\_\_cont\_n=**1**)); by \_\_idca;

if \_\_inout=**0**; \*\*non-matches;

proc sort data=\_\_cont; by \_\_IDCO;

proc sort data=&out; by \_\_IDCO;

data &outnmco; merge \_\_cont

&out(in=\_\_inout); by \_\_idco;

if \_\_inout=**0**; \*\*non-matched controls;

proc sort data=&out; by \_\_IDCA; \*\*re-sort by case id;

%if %upcase(&print)=Y %then %do;

PROC PRINT data=&out LABEL SPLIT='/';

VAR \_\_IDCA \_\_IDCO \_\_CONT\_N

\_\_DIJ

%DO I=**1** %TO &NVAR;

\_\_absd&I

%END;

%if &time^= %then %do;

\_\_dtime

%end;

%DO I=**1** %TO &NVAR;

\_\_CA&I \_\_CO&I

%END;

%if &time^= %then %do;

\_\_catime \_\_cotime

%end;

;

sum \_\_dij;

title9'Data listing for matched cases and controls';

footnote"Greedy matching(gmatch) macro: data=&data group=&group id=&id ";

footnote2" mvars=&mvars wts=&wts dmaxk=&dmaxk dmax=&dmax ncontls=&ncontls";

footnote3" transf=&transf dist=&dist time=&time seedca=&seedca seedco=&seedco";

footnote4" out=&out outnmca=&outnmca outnmco=&outnmco";

run;

title9'Summary data for matched cases and controls--one obs/control';

%if &sysver ge **8** %then %do;

proc means data=&out maxdec=**3** fw=**8**

n mean median min p10 p25 p75 p90 max sum;

%end;

%else %do;

proc means data=&out maxdec=**3**

n mean min max sum;

%end;

class \_\_cont\_n;

var \_\_dij

%do I=**1** %TO &NVAR;

\_\_absd&I

%end;

%if &time^= %then %do;

\_\_dtime

%end;

%do I=**1** %TO &NVAR;

\_\_ca&I

%end;

%if &time^= %then %do;

\_\_catime

%end;

%do I=**1** %TO &NVAR;

\_\_co&I

%end;

%if &time^= %then %do;

\_\_cotime

%end;

;

run;

proc means data=&out n mean noprint; by \_\_idca;

var \_\_dij

%do i=**1** %to &nvar;

\_\_co&i

%end;

\_\_cotime

;

output out=\_mcont n=n\_co mean=\_\_dijm

%do i=**1** %to &nvar;

\_\_com&i

%end;

\_\_tcom

;

data \_onecase; set &out; by \_\_idca; if first.\_\_idca;

data \_\_camcon; merge \_onecase \_mcont; by \_\_idca;

keep \_\_idca n\_co \_\_dijm

\_\_dtime \_\_catime \_\_tcom

%do i=**1** %to &nvar;

\_\_ca&i \_\_com&i \_\_actd&i \_\_absd&i

%end;

;

%do i=**1** %to &nvar;

\_\_absd&i=abs(\_\_ca&i - \_\_com&i);

\_\_actd&i=(\_\_ca&i - \_\_com&i);

%end;

\_\_dtime=\_\_tcom-\_\_catime

;

label

n\_co="No./CONTROLS"

\_\_dijm="Average/Dij"

\_\_dtime="&time/Mean Time DIFF"

\_\_tcom="&time/Mean CONT TIME"

%do i=**1** %to &nvar; %let vvar=%scan(&mvars,&i);

\_\_absd&i="&vvar/Mean ABS. DIFF"

\_\_com&i="&vvar/Mean CONTROL"

%end;

;

title9'Summary data for matched cases and controls--one obs/case(using average control value)';

%if &sysver ge **8** %then %do;

proc means data=\_\_camcon maxdec=**3** fw=**8**

n mean median min p10 p25 p75 p90 max sum;

%end;

%else %do;

proc means data=\_\_camcon maxdec=**3**

n mean min max sum;

%end;

var n\_co \_\_dijm

%do i=**1** %to &nvar;

\_\_absd&i

%end;

%if &time^= %then %do;

\_\_dtime

%end;

%do i=**1** %to &nvar;

\_\_ca&i

%end;

%if &time^= %then %do;

\_\_catime

%end;

%do i=**1** %to &nvar;

\_\_com&i

%end;

%if &time^= %then %do;

\_\_tcom

%end;

;

%end;

%END;

run;

title9; footnote;

run;

**%mend** greedy;

%IF &BAD=**0** %THEN %DO;

%***GREEDY***

%END;

**%MEND** GMATCH;

**run**;

**data** Original.MCI;

set Original.MCI\_nested; timex=yearfollow\_MCI;

%***GMATCH*** (DATA=Original.MCI,

GROUP=MCI,

ID=n\_eid,

MVARS=agelastfollow\_n n\_31\_0\_0,

WTS=**2** **1**,

DMAXK=**2** **0**,

dmax= ,

DIST=**1**,

NCONTLS=**2**,

TIME=timex,

TRANSF=**0**,

SEEDCA=**11040**,

SEEDCO=**490132**,

PRINT= ,

OUT=OUT,

OUTNMCA=NMCA,

OUTNMCO=NMCO);

**run**;

**data** MCIcase;

merge

\_\_case (rename=(\_\_IDCA=IDCASE))

Original.MCI (rename=(n\_eid=IDCASE));

by IDCASE;

if \_\_catime=**.** then delete;

**run**;

**PROC** **SORT** DATA=OUT OUT=fst\_lst ;

BY \_\_IDCO;

**RUN** ;

**data** MCIcontrol;

merge

fst\_lst (rename=(\_\_IDCO=IDCO))

Original.MCI (rename=(n\_eid=IDCO));

by IDCO;

if \_\_catime=**.** then delete;

if \_\_IDCA ne **.** then IDCASE=\_\_IDCA;

**run**;

**PROC** **SORT** DATA=MCIcontrol OUT=MCIcontrol ;

BY IDCO;

**RUN** ;

**PROC** **SORT** DATA=MCIcase OUT=MCIcase ;

BY IDCASE;

**RUN** ;

**data** Original.MCI;

merge

MCIcase (rename=(IDCASE=ID))

MCIcontrol (rename=(IDCO=ID));

by ID;

**run**;

#### MCI-to-Dementia

**data** Original.MCID;

set Original.MCID\_nested;

**%MACRO** GMATCH(DATA=, GROUP= , ID=,

MVARS= ,WTS=,DMAXK=,dmax= , DIST=,

NCONTLS=, TIME=timex,TRANSF=,

SEEDCA=, SEEDCO=, PRINT=,

OUT=,OUTNMCA=,OUTNMCO=);

%LET BAD=0;

%IF %LENGTH(&DATA)=**0** %THEN %DO;

%PUT ERROR: NO DATASET SUPPLIED;

%LET BAD=1;

%END;

%IF %LENGTH(&ID)=**0** %THEN %DO;

%PUT ERROR: NO ID VARIABLE SUPPLIED;

%LET BAD=1;

%END;

%IF %LENGTH(&GROUP)=**0** %THEN %DO;

%PUT ERROR: NO CASE(1)/CONTROL(0) GROUP VARIABLE SUPPLIED;

%LET BAD=1;

%END;

%IF %LENGTH(&MVARS)=**0** %THEN %DO;

%PUT ERROR: NO MATCHING VARIABLES SUPPLIED;

%LET BAD=1;

%END;

%IF %LENGTH(&WTS)=**0** %THEN %DO;

%PUT ERROR: NO WEIGHTS SUPPLIED;

%LET BAD=1;

%END;

%LET NVAR=0;

%DO %UNTIL(%SCAN(&MVARS,&NVAR+**1**,' ')= );

%LET NVAR=%EVAL(&NVAR+1);

%END;

%LET NWTS=0;

%DO %UNTIL(%QSCAN(&WTS,&NWTS+**1**,' ')= );

%LET NWTS=%EVAL(&NWTS+1);

%END;

%IF &NVAR^= &NWTS %THEN %DO;

%PUT ERROR: #VARS MUST EQUAL #WTS;

%LET BAD=1;

%END;

%LET NK=0;

%IF %QUOTE(&DMAXK)^= %THEN %DO %UNTIL(%QSCAN(&DMAXK,&NK+**1**,' ')= );

%LET NK=%EVAL(&NK+1);

%END;

%IF &NK>&NVAR %THEN %LET NK=&NVAR;

%DO I=**1** %TO &NVAR;

%LET V&I=%SCAN(&MVARS,&I,' ');

%END;

%IF &NWTS>**0** %THEN %DO;

DATA \_NULL\_;

%DO I=**1** %TO &NWTS;

%LET W&I=%SCAN(&WTS,&I,' ');

IF &&W&I<**0** THEN DO;

PUT 'ERROR: WEIGHTS MUST BE NON-NEGATIVE';

CALL SYMPUT('BAD','1');

END;

%END;

RUN;

%END;

%IF &NK>**0** %THEN %DO;

DATA \_NULL\_;

%DO I=**1** %TO &NK;

%LET K&I=%SCAN(&DMAXK,&I,' ');

IF &&K&I<**0** THEN DO;

PUT 'ERROR: DMAXK VALUES MUST BE NON-NEGATIVE';

CALL SYMPUT('BAD','1');

END;

%END;

RUN;

%END;

**%MACRO** ***MAX1***;

%IF &DMAX^= %THEN %DO;

& \_\_D<=&DMAX

%END;

%DO I=**1** %TO &NK;

& ABS(\_\_CA&I-\_\_CO&I)<=&&K&I

%END;

**%MEND** MAX1;

**%macro** ***greedy***;

%GLOBAL BAD2;

data \_\_CHECK; set &DATA;

\_\_id=&id;

if \_\_id="" then delete;

%DO I=**1** %TO &NVAR;

IF %scan(&mvars,&i)=**.** THEN DELETE;

%END;

%IF &TIME^= %THEN %DO;

IF &TIME=**.** THEN DELETE;

%END;

run;

%if &transf=**1** %then %do;

proc standard data=\_\_check m=**0** s=**1** out=\_stdzd; var &mvars;

data \_caco;

set \_stdzd;

%end;

%if &transf=**2** %then %do;

proc rank data=\_\_check out=\_ranks; var &mvars;

data \_caco;

set \_ranks;

%end;

%if &transf=**0** %then %do;

data \_caco;

set \_\_check;

%end;

DATA \_\_CASE; SET \_caco;

if &group=**1**;

DATA \_\_CASE; SET \_\_CASE END=EOF;

KEEP \_\_IDCA \_\_CA1-\_\_CA&NVAR \_\_R &mvars

%if &time^= %then %do;

\_\_catime

%end;

;

\_\_IDCA=&ID;

%if &time^= %then %do;

\_\_catime=&time;

%end;

%DO I=**1** %TO &NVAR;

\_\_CA&I=&&V&I;

%END;

%if &seedca^= %then %do;

SEED=&SEEDCA;

\_\_R=RANUNI( SEED );

%end;

%else %do;

\_\_R=**1**;

%end;

IF EOF THEN CALL SYMPUT('NCA',\_N\_);

PROC SORT; BY \_\_R \_\_IDCA;

DATA \_\_CONT; SET \_caco;

if &group=**0**;

DATA \_\_CONT; SET \_\_CONT END=EOF;

KEEP \_\_IDCO \_\_CO1-\_\_CO&NVAR \_\_R &mvars

%if &time^= %then %do;

\_\_cotime

%end;

;

\_\_IDCO=&ID;

%if &time^= %then %do;

\_\_cotime=&time;

%end;

%DO I=**1** %TO &NVAR;

\_\_CO&I=&&V&I;

%END;

%if &seedco^= %then %do;

SEED=&SEEDCo;

\_\_R=RANUNI( SEED );

%end;

%else %do;

\_\_R=**1**;

%end;

IF EOF THEN CALL SYMPUT('NCO',\_N\_);

RUN;

%LET BAD2=0;

%IF &NCO < %EVAL(&NCA\*&NCONTLS) %THEN %DO;

%PUT ERROR: NOT ENOUGH CONTROLS TO MAKE REQUESTED MATCHES;

%LET BAD2=1;

%END;

%IF &BAD2=**0** %THEN %DO;

PROC SORT; BY \_\_R \_\_IDCO;

DATA \_\_MATCH;

KEEP \_\_IDCA \_\_CA1-\_\_CA&NVAR \_\_DIJ \_\_MATCH \_\_CONT\_N

%if &time^= %then %do;

\_\_catime \_\_cotime

%end;

;

ARRAY \_\_USED(&NCO) $ **1** \_TEMPORARY\_;

DO \_\_I=**1** TO &NCO;

\_\_USED(\_\_I)='0';

END;

DO \_\_I=**1** TO &NCONTLS;

DO \_\_J=**1** TO &NCA;

SET \_\_CASE POINT=\_\_J;

\_\_SMALL=**.**;

\_\_MATCH=**.**;

DO \_\_K=**1** TO &NCO;

IF \_\_USED(\_\_K)='0' THEN DO;

SET \_\_CONT POINT=\_\_K;

%if &dist=**2** %then %do;

\*\*wtd euclidian dist;

\_\_D= sqrt(

%do k=**1** %to &nvar;

%scan(&wts,&k)\*(\_\_ca&k - \_\_co&k)\*\***2**

%if &k<&nvar %then + ;

%end;

);

%end;

%else %do;

\_\_D=

%do k=**1** %to &nvar;

%scan(&wts,&k)\*abs(\_\_ca&k - \_\_co&k )

%if &k<&nvar %then + ;

%end;

;

%end;

IF \_\_d^=**.** & (\_\_SMALL=**.** | \_\_D<\_\_SMALL) %***MAX1***

%if &time^= %then %do;

& \_\_cotime > \_\_catime

%end;

THEN DO;

\_\_SMALL=\_\_D;

\_\_MATCH=\_\_K;

\_\_DIJ=\_\_D;

\_\_CONT\_N=\_\_I;

END;

END;

END;

IF \_\_MATCH^=**.** THEN DO;

\_\_USED(\_\_MATCH)='1';

OUTPUT;

END;

END;

END;

STOP;

DATA &OUT;

SET \_\_MATCH;

SET \_\_CONT POINT=\_\_MATCH;

KEEP \_\_IDCA \_\_IDCO \_\_CONT\_N \_\_DIJ \_\_CA1-\_\_CA&NVAR

\_\_CO1-\_\_CO&NVAR \_\_d1-\_\_d&nvar \_\_absd1-\_\_absd&nvar \_\_WT1-\_\_WT&NVAR

\_\_catime \_\_cotime \_\_dtime;

%if &time= %then %do;

\_\_cotime=**.**; \_\_catime=**.**;

%end;

LABEL

\_\_catime="&time/CASE"

\_\_cotime="&time/CONTROL"

\_\_dtime="&time/ABS. DIFF"

\_\_CONT\_N='CONTROL/NUMBER'

\_\_DIJ='DISTANCE/D\_IJ'

%DO I=**1** %TO &NVAR;

\_\_CA&I="&&V&I/CASE"

\_\_CO&I="&&V&I/CONTROL"

\_\_absd&I="&&V&I/ABS. DIFF "

\_\_d&I="&&V&I/DIFF "

\_\_WT&I="&&V&I/WEIGHT"

%END;

;

%DO I=**1** %TO &NVAR;

\_\_d&i= (\_\_CA&I-\_\_CO&I);

\_\_absd&I=abs(\_\_CA&I-\_\_CO&I);

\_\_WT&I=&&W&I;

%END;

\_\_dtime=\_\_cotime-\_\_catime;

PROC SORT DATA=&OUT; BY \_\_IDCA \_\_CONT\_N;

proc sort data=\_\_case; by \_\_IDCA;

data &outnmca; merge \_\_case

&out(in=\_\_inout where=(\_\_cont\_n=**1**)); by \_\_idca;

if \_\_inout=**0**; \*\*non-matches;

proc sort data=\_\_cont; by \_\_IDCO;

proc sort data=&out; by \_\_IDCO;

data &outnmco; merge \_\_cont

&out(in=\_\_inout); by \_\_idco;

if \_\_inout=**0**; \*\*non-matched controls;

proc sort data=&out; by \_\_IDCA; \*\*re-sort by case id;

%if %upcase(&print)=Y %then %do;

PROC PRINT data=&out LABEL SPLIT='/';

VAR \_\_IDCA \_\_IDCO \_\_CONT\_N

\_\_DIJ

%DO I=**1** %TO &NVAR;

\_\_absd&I

%END;

%if &time^= %then %do;

\_\_dtime

%end;

%DO I=**1** %TO &NVAR;

\_\_CA&I \_\_CO&I

%END;

%if &time^= %then %do;

\_\_catime \_\_cotime

%end;

;

sum \_\_dij;

title9'Data listing for matched cases and controls';

footnote"Greedy matching(gmatch) macro: data=&data group=&group id=&id ";

footnote2" mvars=&mvars wts=&wts dmaxk=&dmaxk dmax=&dmax ncontls=&ncontls";

footnote3" transf=&transf dist=&dist time=&time seedca=&seedca seedco=&seedco";

footnote4" out=&out outnmca=&outnmca outnmco=&outnmco";

run;

title9'Summary data for matched cases and controls--one obs/control';

%if &sysver ge **8** %then %do;

proc means data=&out maxdec=**3** fw=**8**

n mean median min p10 p25 p75 p90 max sum;

%end;

%else %do;

proc means data=&out maxdec=**3**

n mean min max sum;

%end;

class \_\_cont\_n;

var \_\_dij

%do I=**1** %TO &NVAR;

\_\_absd&I

%end;

%if &time^= %then %do;

\_\_dtime

%end;

%do I=**1** %TO &NVAR;

\_\_ca&I

%end;

%if &time^= %then %do;

\_\_catime

%end;

%do I=**1** %TO &NVAR;

\_\_co&I

%end;

%if &time^= %then %do;

\_\_cotime

%end;

;

run;

proc means data=&out n mean noprint; by \_\_idca;

var \_\_dij

%do i=**1** %to &nvar;

\_\_co&i

%end;

\_\_cotime

;

output out=\_mcont n=n\_co mean=\_\_dijm

%do i=**1** %to &nvar;

\_\_com&i

%end;

\_\_tcom

;

data \_onecase; set &out; by \_\_idca; if first.\_\_idca;

data \_\_camcon; merge \_onecase \_mcont; by \_\_idca;

keep \_\_idca n\_co \_\_dijm

\_\_dtime \_\_catime \_\_tcom

%do i=**1** %to &nvar;

\_\_ca&i \_\_com&i \_\_actd&i \_\_absd&i

%end;

;

%do i=**1** %to &nvar;

\_\_absd&i=abs(\_\_ca&i - \_\_com&i);

\_\_actd&i=(\_\_ca&i - \_\_com&i);

%end;

\_\_dtime=\_\_tcom-\_\_catime

;

label

n\_co="No./CONTROLS"

\_\_dijm="Average/Dij"

\_\_dtime="&time/Mean Time DIFF"

\_\_tcom="&time/Mean CONT TIME"

%do i=**1** %to &nvar; %let vvar=%scan(&mvars,&i);

\_\_absd&i="&vvar/Mean ABS. DIFF"

\_\_com&i="&vvar/Mean CONTROL"

%end;

;

title9'Summary data for matched cases and controls--one obs/case(using average control value)';

%if &sysver ge **8** %then %do;

proc means data=\_\_camcon maxdec=**3** fw=**8**

n mean median min p10 p25 p75 p90 max sum;

%end;

%else %do;

proc means data=\_\_camcon maxdec=**3**

n mean min max sum;

%end;

var n\_co \_\_dijm

%do i=**1** %to &nvar;

\_\_absd&i

%end;

%if &time^= %then %do;

\_\_dtime

%end;

%do i=**1** %to &nvar;

\_\_ca&i

%end;

%if &time^= %then %do;

\_\_catime

%end;

%do i=**1** %to &nvar;

\_\_com&i

%end;

%if &time^= %then %do;

\_\_tcom

%end;

;

%end;

%END;

run;

title9; footnote;

run;

**%mend** greedy;

%IF &BAD=**0** %THEN %DO;

%***GREEDY***

%END;

**%MEND** GMATCH;

**run**;

**data** Original.MCID;

set Original.MCID\_nested; timex=yearfollow\_MCID;

%***GMATCH*** (DATA=Original.MCID,

GROUP=MCI\_D,

ID=n\_eid,

MVARS=agelastfollow\_n n\_31\_0\_0,

WTS=**2** **1**,

DMAXK=**2** **0**,

dmax= ,

DIST=**1**,

NCONTLS=**2**,

TIME=timex,

TRANSF=**0**,

SEEDCA=**2877**,

SEEDCO=**9154**,

PRINT= ,

OUT=OUT,

OUTNMCA=NMCA,

OUTNMCO=NMCO);

**run**;

**data** MCIDcase;

merge

\_\_case (rename=(\_\_IDCA=IDCASE))

Original.MCID (rename=(n\_eid=IDCASE));

by IDCASE;

if \_\_catime=**.** then delete;

**run**;

**PROC** **SORT** DATA=OUT OUT=fst\_lst ;

BY \_\_IDCO;

**RUN** ;

**data** MCIDcontrol;

merge

fst\_lst (rename=(\_\_IDCO=IDCO))

Original.MCID (rename=(n\_eid=IDCO));

by IDCO;

if \_\_catime=**.** then delete;

if \_\_IDCA ne **.** then IDCASE=\_\_IDCA;

**run**;

**PROC** **SORT** DATA=MCIDcontrol OUT=MCIDcontrol ;

BY IDCO;

**RUN** ;

**PROC** **SORT** DATA=MCIDcase OUT=MCIDcase ;

BY IDCASE;

**RUN** ;

**data** Original.MCID;

merge

MCIDcase (rename=(IDCASE=ID))

MCIDcontrol (rename=(IDCO=ID));

by ID;

**run**;

### Logistic Regression

/\*Genetic: APOE\_status Bloodtype\*/

/\*sex: n\_31\_0\_0\*/

/\*race\_n\*/

**Proc** **logistic** data=Original.MCI\_analysis;

class APOE\_status (ref="0" param=ref) ;

Model MCI(event="1")= APOE\_status /expb;

**Run**;

**Proc** **logistic** data=Original.MCID\_analysis;

class APOE\_status (ref="0" param=ref) ;

Model MCI\_D(event="1")= APOE\_status /expb;

**Run**;

/\*Early-life:Birthweight\_c breast \*/

**Proc** **logistic** data=Original.MCI\_analysis;

class Bloodtype (ref="0" param=ref) APOE\_status (ref="0" param=ref) race\_n (ref="1" param=ref) n\_31\_0\_0 (ref="1" param=ref)

breast (ref="1" param=ref);

Model MCI(event="1") = breast APOE\_status Bloodtype n\_31\_0\_0 race\_n /expb;

**Run**;

**Proc** **logistic** data=Original.MCID\_analysis;

class Bloodtype (ref="0" param=ref) APOE\_status (ref="0" param=ref) race\_n (ref="1" param=ref) n\_31\_0\_0 (ref="1" param=ref)

breast (ref="1" param=ref);

Model MCI\_D(event="1") = breast APOE\_status Bloodtype n\_31\_0\_0 race\_n /expb;

**Run**;

/\*Midlife Demographics:age\_c education1 income TDI\*/

**Proc** **logistic** data=Original.MCI\_analysis;

class age\_c (ref="0" param=ref) APOE\_status (ref="0" param=ref) Bloodtype (ref="0" param=ref) race\_n (ref="1" param=ref) breast (ref="1" param=ref) ;

Model MCI(event="1") = age\_c APOE\_status Bloodtype n\_31\_0\_0 race\_n breast/expb;

**Run**;

**Proc** **logistic** data=Original.MCID\_analysis;

class age\_c (ref="0" param=ref) APOE\_status (ref="0" param=ref) Bloodtype (ref="0" param=ref) race\_n (ref="1" param=ref) breast (ref="1" param=ref) ;

Model MCI\_D(event="1") = age\_c APOE\_status Bloodtype n\_31\_0\_0 race\_n breast/expb;

**Run**;

/\*Environment:PM2 PM10 NO2 noise\*/

**Proc** **logistic** data=Original.MCI\_analysis;

class noise(ref="0" param=ref) APOE\_status (ref="0" param=ref) Bloodtype (ref="0" param=ref) race\_n (ref="1" param=ref)

n\_31\_0\_0 (ref="1" param=ref) breast (ref="1" param=ref)

age\_c (ref="2" param=ref) education1(ref="3" param=ref) income(ref="4" param=ref) TDI(ref="0" param=ref);

Model MCI(event="1")= noise age\_c education1 income TDI APOE\_status Bloodtype n\_31\_0\_0 race\_n breast /expb;

**Run**;

**Proc** **logistic** data=Original.MCID\_analysis;

class noise(ref="0" param=ref) APOE\_status (ref="0" param=ref) Bloodtype (ref="0" param=ref) race\_n (ref="1" param=ref)

n\_31\_0\_0 (ref="1" param=ref) breast (ref="1" param=ref)

age\_c (ref="2" param=ref) education1(ref="3" param=ref) income(ref="4" param=ref) TDI(ref="0" param=ref);

Model MCI\_D(event="1")= noise age\_c education1 income TDI APOE\_status Bloodtype n\_31\_0\_0 race\_n breast /expb;

**Run**;

/\*lifestyle:smoking\_B alcohol\_c PA\_level sedentary\_c sleep\_c diet\_c\*/

**Proc** **logistic** data=Original.MCI\_analysis;

class diet\_c (ref="0" param=ref) APOE\_status (ref="0" param=ref) Bloodtype (ref="0" param=ref) race\_n (ref="1" param=ref)

n\_31\_0\_0 (ref="1" param=ref) breast (ref="1" param=ref)

age\_c (ref="2" param=ref) education1(ref="3" param=ref) income(ref="4" param=ref) TDI(ref="0" param=ref)

PM2 (ref="0" param=ref) PM10 (ref="0" param=ref)

NO2 (ref="0" param=ref) noise (ref="0" param=ref);

Model MCI(event="1") =diet\_c age\_c education1 income TDI APOE\_status Bloodtype n\_31\_0\_0 race\_n breast PM2 PM10 NO2 noise/expb;

**Run**;

**Proc** **logistic** data=Original.MCID\_analysis;

class diet\_c (ref="0" param=ref) APOE\_status (ref="0" param=ref) Bloodtype (ref="0" param=ref) race\_n (ref="1" param=ref)

n\_31\_0\_0 (ref="1" param=ref) breast (ref="1" param=ref)

age\_c (ref="2" param=ref) education1(ref="3" param=ref) income(ref="4" param=ref) TDI(ref="0" param=ref)

PM2 (ref="0" param=ref) PM10 (ref="0" param=ref)

NO2 (ref="0" param=ref) noise (ref="0" param=ref);

Model MCI\_D(event="1") =diet\_c age\_c education1 income TDI APOE\_status Bloodtype n\_31\_0\_0 race\_n breast PM2 PM10 NO2 noise/expb;

**Run**;

/\*Biological:bmi4group\_b Grip\_c DBP\_C SBP\_C LDL\_C HbA1c\_c insulin\_c VD\_c\*/

/\*psychosocial:depressed\_c restless\_c anxious Lone ISO\_c\*/

**Proc** **logistic** data=Original.MCI\_analysis;

class ISO\_c (ref="0" param=ref) APOE\_status (ref="0" param=ref) Bloodtype (ref="0" param=ref) race\_n (ref="1" param=ref)

n\_31\_0\_0 (ref="1" param=ref) breast (ref="1" param=ref)

age\_c (ref="2" param=ref) education1(ref="3" param=ref) income(ref="4" param=ref) TDI(ref="0" param=ref)

PM2 (ref="0" param=ref) PM10 (ref="0" param=ref)

NO2 (ref="0" param=ref) noise (ref="0" param=ref)

smoking\_B (ref="0" param=ref) alcohol\_c (ref="4" param=ref) PA\_level (ref="2" param=ref) sedentary\_c (ref="1" param=ref) sleep\_c (ref="1" param=ref) diet\_c (ref="0" param=ref) ;

Model MCI(event="1") = ISO\_c age\_c education1 income TDI APOE\_status Bloodtype n\_31\_0\_0 race\_n breast PM2 PM10 NO2 noise smoking\_B alcohol\_c PA\_level sedentary\_c sleep\_c diet\_c /expb;

**Run**;

**Proc** **logistic** data=Original.MCID\_analysis;

class ISO\_c (ref="0" param=ref) APOE\_status (ref="0" param=ref) Bloodtype (ref="0" param=ref) race\_n (ref="1" param=ref)

n\_31\_0\_0 (ref="1" param=ref) breast (ref="1" param=ref)

age\_c (ref="2" param=ref) education1(ref="3" param=ref) income(ref="4" param=ref) TDI(ref="0" param=ref)

PM2 (ref="0" param=ref) PM10 (ref="0" param=ref)

NO2 (ref="0" param=ref) noise (ref="0" param=ref)

smoking\_B (ref="0" param=ref) alcohol\_c (ref="4" param=ref) PA\_level (ref="2" param=ref) sedentary\_c (ref="1" param=ref) sleep\_c (ref="1" param=ref) diet\_c (ref="0" param=ref) ;

Model MCI\_D(event="1") = ISO\_c age\_c education1 income TDI APOE\_status Bloodtype n\_31\_0\_0 race\_n breast PM2 PM10 NO2 noise smoking\_B alcohol\_c PA\_level sedentary\_c sleep\_c diet\_c /expb;

**Run**;

/\*Disability:Disability\_c hear\_c HA\_c Frailty\_cc\*/

/\*Medical condition:\*/

/\*CVD\_c CHD\_c stroke\_c HF\_c HP\_c \*/

/\*MS\_c DM\_c Obx\_c Dy\_c \*/

/\*RD\_c IP\_c COPD\_c AS\_c \*/

/\*DD\_c IBD\_c GORD\_c PUD\_c IBS\_c\*/

/\*MSK\_c Gout\_c OA\_c RA\_c\*/

/\*cancer\_c LHS\_c DS\_c ES\_c RPS\_c RS\_c US\_c\*/

**Proc** **logistic** data=Original.MCI\_analysis;

class US\_c (ref="0" param=ref) APOE\_status (ref="0" param=ref) Bloodtype (ref="0" param=ref) race\_n (ref="1" param=ref)

breast (ref="1" param=ref) age\_c (ref="2" param=ref) education1(ref="3" param=ref) income(ref="4" param=ref)

TDI (ref="0" param=ref) PM2 (ref="0" param=ref)

PM10 (ref="0" param=ref) NO2 (ref="0" param=ref)

noise (ref="0" param=ref) smoking\_B (ref="0" param=ref)

alcohol\_c (ref="4" param=ref) PA\_level (ref="2" param=ref)

sedentary\_c (ref="1" param=ref) sleep\_c (ref="1" param=ref) diet\_c (ref="0" param=ref) bmi4group\_b (ref="2" param=ref) Grip\_c (ref="3" param=ref) DBP\_C (ref="1" param=ref)

SBP\_C (ref="1" param=ref) LDL\_C (ref="1" param=ref)

HbA1c\_c (ref="0" param=ref) insulin\_c (ref="0" param=ref)

VD\_c (ref="2" param=ref) depressed\_c (ref="0" param=ref)

restless\_c (ref="0" param=ref) anxious(ref="0" param=ref) Lone(ref="0" param=ref) ISO\_c(ref="1" param=ref) ;

Model MCI(event="1") = US\_c age\_c education1 income TDI APOE\_status n\_31\_0\_0 race\_n Bloodtype breast PM2 PM10 NO2 noise smoking\_B alcohol\_c PA\_level sedentary\_c sleep\_c diet\_c bmi4group\_b Grip\_c DBP\_C SBP\_C LDL\_C HbA1c\_c insulin\_c VD\_c depressed\_c restless\_c anxious Lone ISO\_c /expb;

**Run**;

**Proc** **logistic** data=Original.MCID\_analysis;

class US\_c (ref="0" param=ref) APOE\_status (ref="0" param=ref) Bloodtype (ref="0" param=ref) race\_n (ref="1" param=ref)

breast (ref="1" param=ref) age\_c (ref="2" param=ref) education1(ref="3" param=ref) income(ref="4" param=ref)

TDI (ref="0" param=ref) PM2 (ref="0" param=ref)

PM10 (ref="0" param=ref) NO2 (ref="0" param=ref)

noise (ref="0" param=ref) smoking\_B (ref="0" param=ref)

alcohol\_c (ref="4" param=ref) PA\_level (ref="2" param=ref)

sedentary\_c (ref="1" param=ref) sleep\_c (ref="1" param=ref) diet\_c (ref="0" param=ref) bmi4group\_b (ref="2" param=ref) Grip\_c (ref="3" param=ref) DBP\_C (ref="1" param=ref)

SBP\_C (ref="1" param=ref) LDL\_C (ref="1" param=ref)

HbA1c\_c (ref="0" param=ref) insulin\_c (ref="0" param=ref)

VD\_c (ref="2" param=ref) depressed\_c (ref="0" param=ref)

restless\_c (ref="0" param=ref) anxious(ref="0" param=ref) Lone(ref="0" param=ref) ISO\_c(ref="1" param=ref) ;

Model MCI\_D(event="1") = US\_c age\_c education1 income TDI APOE\_status n\_31\_0\_0 race\_n Bloodtype breast PM2 PM10 NO2 noise smoking\_B alcohol\_c PA\_level sedentary\_c sleep\_c diet\_c bmi4group\_b Grip\_c DBP\_C SBP\_C LDL\_C HbA1c\_c insulin\_c VD\_c depressed\_c restless\_c anxious Lone ISO\_c /expb;

**Run**;

/\*female-specific: menarche\_cc agefirstbirth\_c agelastbirth\_c child\_c\*/

**Proc** **logistic** data=Original.MCI\_analysis;

where n\_31\_0\_0=**0**;

class child\_c(ref="0" param=ref) APOE\_status (ref="0" param=ref) Bloodtype (ref="0" param=ref) race\_n (ref="1" param=ref)

breast (ref="1" param=ref);

Model MCI(event="1") =child\_c APOE\_status Bloodtype n\_31\_0\_0 race\_n breast/expb;

**Run**;

**Proc** **logistic** data=Original.MCID\_analysis;

where n\_31\_0\_0=**0**;

class child\_c(ref="0" param=ref) APOE\_status (ref="0" param=ref) Bloodtype (ref="0" param=ref) race\_n (ref="1" param=ref)

breast (ref="1" param=ref);

Model MCI\_D(event="1") =child\_c APOE\_status Bloodtype n\_31\_0\_0 race\_n breast/expb;

**Run**;

/\*female-specific: menopauseage\_c Surgeryage\_c OC\_c HRT\_c\*/

**Proc** **logistic** data=Original.MCI\_analysis;

where n\_31\_0\_0=**0**;

class HRT\_c(ref="0" param=ref) APOE\_status (ref="0" param=ref) Bloodtype (ref="0" param=ref) race\_n (ref="1" param=ref)

breast (ref="1" param=ref) age\_c (ref="2" param=ref) education1(ref="3" param=ref) income(ref="4" param=ref) TDI(ref="0" param=ref) PM2 (ref="0" param=ref)

PM10 (ref="0" param=ref) NO2 (ref="0" param=ref)

noise (ref="0" param=ref) smoking\_B (ref="0" param=ref) alcohol\_c (ref="4" param=ref) PA\_level (ref="2" param=ref)

sedentary\_c (ref="1" param=ref) sleep\_c (ref="1" param=ref) diet\_c (ref="0" param=ref) bmi4group\_b (ref="2" param=ref) Grip\_c DBP\_C (ref="1" param=ref) SBP\_C (ref="1" param=ref)

LDL\_C (ref="1" param=ref) HbA1c\_c (ref="0" param=ref)

insulin\_c (ref="0" param=ref) VD\_c (ref="2" param=ref)

depressed\_c (ref="0" param=ref) restless\_c (ref="0" param=ref) anxious(ref="0" param=ref) Lone(ref="0" param=ref)

ISO\_c(ref="1" param=ref) ;

Model MCI(event="1") = HRT\_c age\_c education1 income TDI APOE\_status n\_31\_0\_0 race\_n Bloodtype breast PM2 PM10 NO2 noise smoking\_B alcohol\_c PA\_level sedentary\_c sleep\_c diet\_c bmi4group\_b Grip\_c DBP\_C SBP\_C LDL\_C HbA1c\_c insulin\_c VD\_c depressed\_c restless\_c anxious Lone ISO\_c /expb;

**Run**;

## VIF

### MCI

import pandas as pd  
from statsmodels.formula.api import ols  
from statsmodels.stats.outliers\_influence import variance\_inflation\_factor

data = pd.read\_sas(r'D:\nested\_case\_control\predication\MCI\mci\_lasso.sas7bdat')  
target\_variable = 'MCI'  
independent\_variables = ['age\_c','race\_n','Bloodtype','Birthweight\_c', 'education1', 'PA\_level', 'income', 'bmi4group\_b', 'smoking\_B', 'APOE\_status', 'MCI', 'TDI', 'Grip\_c', 'DBP\_c', 'SBP\_c', 'LDL\_c', 'HbA1c\_c', 'insulin\_c', 'alcohol\_c', 'sedentary\_c','HA\_c', 'hear\_c','depressed\_c', 'restless\_c', 'anxious', 'Lone', 'ISO\_c', 'PM2', 'CHD', 'Stroke', 'HF', 'HP', 'DM', 'Obx', 'Dy', 'IP', 'COPD', 'As', 'IBD', 'GORD', 'PUD', 'IBS', 'Gout', 'OA', 'RA', 'LHS', 'DS', 'ES', 'RS', 'US', 'RPS', 'Frai', 'sleep\_c']  
  
model = ols(f"{target\_variable} ~ {' + '.join(independent\_variables)}", data=data).fit()  
vif\_data = pd.DataFrame()  
vif\_data["Features"] = model.params.index  
vif\_data["VIF"] = [variance\_inflation\_factor(model.model.exog, i) for i in range(len(model.params))]  
print(vif\_data)  
vif\_data.to\_csv(r'D:\nested\_case\_control\MCI\_vif.csv')

### MCI (Delete VIF>10)

import pandas as pd  
from statsmodels.formula.api import ols  
from statsmodels.stats.outliers\_influence import variance\_inflation\_factor

data = pd.read\_sas(r'D:\nested\_case\_control\predication\MCI\mci\_lasso.sas7bdat')  
target\_variable = 'MCI'  
independent\_variables = ['age\_c','race\_n','Bloodtype','Birthweight\_c', 'education1', 'PA\_level', 'income', 'bmi4group\_b', 'smoking\_B', 'APOE\_status', 'MCI', 'TDI', 'Grip\_c', 'DBP\_c', 'SBP\_c', 'LDL\_c', 'HbA1c\_c', 'insulin\_c', 'alcohol\_c', 'sedentary\_c','hear\_c', 'depressed\_c', 'restless\_c', 'anxious', 'Lone', 'ISO\_c', 'PM2', 'CHD', 'Stroke', 'HF', 'HP', 'DM', 'Obx', 'Dy', 'IP', 'COPD', 'As', 'IBD', 'GORD', 'PUD', 'IBS', 'Gout', 'OA', 'RA', 'LHS', 'DS', 'ES', 'RS', 'US', 'RPS', 'Frai', 'sleep\_c']  
  
model = ols(f"{target\_variable} ~ {' + '.join(independent\_variables)}", data=data).fit()  
vif\_data = pd.DataFrame()  
vif\_data["Features"] = model.params.index  
vif\_data["VIF"] = [variance\_inflation\_factor(model.model.exog, i) for i in range(len(model.params))]  
print(vif\_data)  
vif\_data.to\_csv(r'D:\nested\_case\_control\MCI\_vif\_new.csv')

### MCI-to-dementia

import pandas as pd  
from statsmodels.formula.api import ols  
from statsmodels.stats.outliers\_influence import variance\_inflation\_factor

data = pd.read\_sas(r'D:\nested\_case\_control\predication\MCI\_D\mcid\_boruta.sas7bdat')  
target\_variable = 'MCI\_D'  
independent\_variables = ['race\_n', 'education1', 'PA\_level', 'income', 'bmi4group\_b', 'smoking\_B', 'APOE\_status', 'Grip\_c', 'HbA1c\_c', 'insulin\_c', 'alcohol\_c', 'sedentary\_c', 'hear\_c', 'restless\_c', 'Lone', 'ISO\_c', 'CHD', 'Stroke', 'HF', 'HP', 'DM', 'IP', 'COPD', 'OA',]  
model = ols(f"{target\_variable} ~ {' + '.join(independent\_variables)}", data=data).fit()  
vif\_data = pd.DataFrame()  
vif\_data["Features"] = model.params.index  
vif\_data["VIF"] = [variance\_inflation\_factor(model.model.exog, i) for i in range(len(model.params))]  
print(vif\_data)  
vif\_data.to\_csv(r'D:\nested\_case\_control\MCID\_vif.csv')

# Lasso

rm**(**list**=**ls**())**

setwd**(**"D:/SDU/UKB/nested case-control/Data/MCI\_nested"**)**

library**(**haven**)**

mcilasso **<-** read\_sas**(**'D:/SDU/UKB/nested case-control/Data/MCI\_nested/MCI\_lasso.sas7bdat'**)**

View**(**mcilasso**)**

clean\_data**<-**na.omit**(**mcilasso**)**

names**(**clean\_data**)**

clean\_data**$**APOE\_status **<-** as.factor**(**clean\_data**$**APOE\_status**)**

clean\_data**$**Birthweight\_c **<-** as.factor**(**clean\_data**$**Birthweight\_c**)**

clean\_data**$**age\_c **<-** as.factor**(**clean\_data**$**age\_c**)**

clean\_data**$**education1 **<-** as.factor**(**clean\_data**$**education1**)**

clean\_data**$**income **<-** as.factor**(**clean\_data**$**income**)**

clean\_data**$**TDI **<-** as.factor**(**clean\_data**$**TDI**)**

clean\_data**$**smoking\_B **<-** as.factor**(**clean\_data**$**smoking\_B**)**

clean\_data**$**alcohol\_c **<-** as.factor**(**clean\_data**$**alcohol\_c**)**

clean\_data**$**PA\_level **<-** as.factor**(**clean\_data**$**PA\_level**)**

clean\_data**$**sleep\_c **<-** as.factor**(**clean\_data**$**sleep\_c**)**

clean\_data**$**Grip\_c **<-** as.factor**(**clean\_data**$**Grip\_c**)**

clean\_data**$**sedentary\_c **<-** as.factor**(**clean\_data**$**sedentary\_c**)**

clean\_data**$**LDL\_c **<-** as.factor**(**clean\_data**$**LDL\_c**)**

clean\_data**$**HbA1c\_c **<-** as.factor**(**clean\_data**$**HbA1c\_c**)**

clean\_data**$**insulin\_c **<-** as.factor**(**clean\_data**$**insulin\_c**)**

clean\_data**$**depressed\_c **<-** as.factor**(**clean\_data**$**depressed\_c**)**

clean\_data**$**restless\_c **<-** as.factor**(**clean\_data**$**restless\_c**)**

clean\_data**$**Lone **<-** as.factor**(**clean\_data**$**Lone**)**

clean\_data**$**hear\_c **<-** as.factor**(**clean\_data**$**hear\_c**)**

clean\_data**$**HA\_c **<-** as.factor**(**clean\_data**$**HA\_c**)**

clean\_data**$**Frai **<-** as.factor**(**clean\_data**$**Frai**)**

clean\_data**$**CHD **<-** as.factor**(**clean\_data**$**CHD**)**

clean\_data**$**Stroke **<-** as.factor**(**clean\_data**$**Stroke**)**

clean\_data**$**HF **<-** as.factor**(**clean\_data**$**HF**)**

clean\_data**$**HP **<-** as.factor**(**clean\_data**$**HP**)**

clean\_data**$**DM **<-** as.factor**(**clean\_data**$**DM**)**

clean\_data**$**Obx **<-** as.factor**(**clean\_data**$**Obx**)**

clean\_data**$**Dy **<-** as.factor**(**clean\_data**$**Dy**)**

clean\_data**$**IP **<-** as.factor**(**clean\_data**$**IP**)**

clean\_data**$**COPD **<-** as.factor**(**clean\_data**$**COPD**)**

clean\_data**$**As **<-** as.factor**(**clean\_data**$**As**)**

clean\_data**$**IBD **<-** as.factor**(**clean\_data**$**IBD**)**

clean\_data**$**GORD **<-** as.factor**(**clean\_data**$**GORD**)**

clean\_data**$**PUD **<-** as.factor**(**clean\_data**$**PUD**)**

clean\_data**$**IBS **<-** as.factor**(**clean\_data**$**IBS**)**

clean\_data**$**Gout **<-** as.factor**(**clean\_data**$**Gout**)**

clean\_data**$**OA **<-** as.factor**(**clean\_data**$**OA**)**

clean\_data**$**RA **<-** as.factor**(**clean\_data**$**RA**)**

clean\_data**$**LHS **<-** as.factor**(**clean\_data**$**LHS**)**

clean\_data**$**DS **<-** as.factor**(**clean\_data**$**DS**)**

clean\_data**$**ES **<-** as.factor**(**clean\_data**$**ES**)**

clean\_data**$**RPS **<-** as.factor**(**clean\_data**$**RPS**)**

clean\_data**$**RS **<-** as.factor**(**clean\_data**$**RS**)**

clean\_data**$**US **<-** as.factor**(**clean\_data**$**US**)**

datax **<-** clean\_data**[**,**-**9**]**

library**(**tidyverse**)**

dummyAPOE\_status **<-** model.matrix**(~**APOE\_status, data**=**datax**)**

head**(**cbind**(**dummyAPOE\_status,clean\_data**$**APOE\_status**))**

dummyBirthweight\_c **<-** model.matrix**(~**Birthweight\_c, data**=**datax**)**

head**(**cbind**(**dummyBirthweight\_c,clean\_data**$**Birthweight\_c**))**

dummyBloodtype **<-** model.matrix**(~**Bloodtype, data**=**datax**)**

head**(**cbind**(**dummyBloodtype,clean\_data**$**Bloodtype**))**

dummyage\_c **<-** model.matrix**(~**age\_c, data**=**datax**)**

head**(**cbind**(**dummyage\_c,clean\_data**$**age\_c**))**

dummyeducation1 **<-** model.matrix**(~**education1, data**=**datax**)**

head**(**cbind**(**dummyeducation1,clean\_data**$**education1**))**

dummyincome **<-** model.matrix**(~**income, data**=**datax**)**

head**(**cbind**(**dummyincome,clean\_data**$**income**))**

dummyTDI **<-** model.matrix**(~**TDI, data**=**datax**)**

head**(**cbind**(**dummyTDI,clean\_data**$**TDI**))**

dummysmoking\_B **<-** model.matrix**(~**smoking\_B, data**=**datax**)**

head**(**cbind**(**dummysmoking\_B,clean\_data**$**smoking\_B**))**

dummyalcohol\_c **<-** model.matrix**(~**alcohol\_c, data**=**datax**)**

head**(**cbind**(**dummyalcohol\_c,clean\_data**$**alcohol\_c**))**

dummysleep\_c **<-** model.matrix**(~**sleep\_c, data**=**datax**)**

head**(**cbind**(**dummysleep\_c,clean\_data**$**sleep\_c**))**

dummyPA\_level **<-** model.matrix**(~**PA\_level, data**=**datax**)**

head**(**cbind**(**dummyPA\_level,clean\_data**$**PA\_level**))**

dummyGrip\_c **<-** model.matrix**(~**Grip\_c, data**=**datax**)**

head**(**cbind**(**dummyGrip\_c,clean\_data**$**Grip\_c**))**

dummysedentary\_c **<-** model.matrix**(~**sedentary\_c, data**=**datax**)**

head**(**cbind**(**dummysedentary\_c,clean\_data**$**sedentary\_c**))**

dummyLDL\_c **<-** model.matrix**(~**LDL\_c, data**=**datax**)**

head**(**cbind**(**dummyLDL\_c,clean\_data**$**LDL\_c**))**

dummyHbA1c\_c **<-** model.matrix**(~**HbA1c\_c, data**=**datax**)**

head**(**cbind**(**dummyHbA1c\_c,clean\_data**$**HbA1c\_c**))**

dummyinsulin\_c **<-** model.matrix**(~**insulin\_c, data**=**datax**)**

head**(**cbind**(**dummyinsulin\_c,clean\_data**$**insulin\_c**))**

dummydepressed\_c **<-** model.matrix**(~**depressed\_c, data**=**datax**)**

head**(**cbind**(**dummydepressed\_c,clean\_data**$**depressed\_c**))**

dummyrestless\_c **<-** model.matrix**(~**restless\_c, data**=**datax**)**

head**(**cbind**(**dummyrestless\_c,clean\_data**$**restless\_c**))**

dummyLone **<-** model.matrix**(~**Lone, data**=**datax**)**

head**(**cbind**(**dummyLone,clean\_data**$**Lone**))**

dummyhear\_c **<-** model.matrix**(~**hear\_c, data**=**datax**)**

head**(**cbind**(**dummyhear\_c,clean\_data**$**hear\_c**))**

dummyHA\_c **<-** model.matrix**(~**HA\_c, data**=**datax**)**

head**(**cbind**(**dummyHA\_c,clean\_data**$**HA\_c**))**

dummyFrai **<-** model.matrix**(~**Frai, data**=**datax**)**

head**(**cbind**(**dummyFrai,clean\_data**$**Frai**))**

dummyCHD **<-** model.matrix**(~**CHD, data**=**datax**)**

head**(**cbind**(**dummyCHD,clean\_data**$**CHD**))**

dummyStroke **<-** model.matrix**(~**Stroke, data**=**datax**)**

head**(**cbind**(**dummyStroke,clean\_data**$**Stroke**))**

dummyHF **<-** model.matrix**(~**HF, data**=**datax**)**

head**(**cbind**(**dummyHF,clean\_data**$**HF**))**

dummyHP **<-** model.matrix**(~**HP, data**=**datax**)**

head**(**cbind**(**dummyHP,clean\_data**$**HP**))**

dummyDM **<-** model.matrix**(~**DM, data**=**datax**)**

head**(**cbind**(**dummyDM,clean\_data**$**DM**))**

dummyObx **<-** model.matrix**(~**Obx, data**=**datax**)**

head**(**cbind**(**dummyObx,clean\_data**$**Obx**))**

dummyDy **<-** model.matrix**(~**Dy, data**=**datax**)**

head**(**cbind**(**dummyDy,clean\_data**$**Dy**))**

dummyIP **<-** model.matrix**(~**IP, data**=**datax**)**

head**(**cbind**(**dummyIP,clean\_data**$**IP**))**

dummyCOPD **<-** model.matrix**(~**COPD, data**=**datax**)**

head**(**cbind**(**dummyCOPD,clean\_data**$**COPD**))**

dummyAs **<-** model.matrix**(~**As, data**=**datax**)**

head**(**cbind**(**dummyAs,clean\_data**$**As**))**

dummyIBD **<-** model.matrix**(~**IBD, data**=**datax**)**

head**(**cbind**(**dummyIBD,clean\_data**$**IBD**))**

dummyGORD **<-** model.matrix**(~**GORD, data**=**datax**)**

head**(**cbind**(**dummyGORD,clean\_data**$**GORD**))**

dummyPUD **<-** model.matrix**(~**PUD, data**=**datax**)**

head**(**cbind**(**dummyPUD,clean\_data**$**PUD**))**

dummyIBS **<-** model.matrix**(~**IBS, data**=**datax**)**

head**(**cbind**(**dummyIBS,clean\_data**$**IBS**))**

dummyGout **<-** model.matrix**(~**Gout, data**=**datax**)**

head**(**cbind**(**dummyGout,clean\_data**$**Gout**))**

dummyOA **<-** model.matrix**(~**OA, data**=**datax**)**

head**(**cbind**(**dummyOA,clean\_data**$**OA**))**

dummyRA **<-** model.matrix**(~**RA, data**=**datax**)**

head**(**cbind**(**dummyRA,clean\_data**$**RA**))**

dummyLHS **<-** model.matrix**(~**LHS, data**=**datax**)**

head**(**cbind**(**dummyLHS,clean\_data**$**LHS**))**

dummyDS **<-** model.matrix**(~**DS, data**=**datax**)**

head**(**cbind**(**dummyDS,clean\_data**$**DS**))**

dummyES **<-** model.matrix**(~**ES, data**=**datax**)**

head**(**cbind**(**dummyES,clean\_data**$**ES**))**

dummyRPS **<-** model.matrix**(~**RPS, data**=**datax**)**

head**(**cbind**(**dummyRPS,clean\_data**$**RPS**))**

dummyRS **<-** model.matrix**(~**RS, data**=**datax**)**

head**(**cbind**(**dummyRS,clean\_data**$**RS**))**

dummyUS **<-** model.matrix**(~**US, data**=**datax**)**

head**(**cbind**(**dummyUS,clean\_data**$**US**))**

names\_factor**<-**c**(** "age\_c","race\_n","education1", "PA\_level" , "income" , "bmi4group\_b" , "smoking\_B",

"APOE\_status", "MCI" ,"Bloodtype" , "Birthweight\_c","TDI",

"Grip\_c","DBP\_c" , "SBP\_c", "LDL\_c" , "HbA1c\_c" , "insulin\_c" , "VD\_c" ,

"alcohol\_c","sedentary\_c" , "sleep\_c", "hear\_c" ,"depressed\_c" , "restless\_c",

"anxious" , "Lone","ISO\_c", "PM2" ,"CHD" , "Stroke", "HF" , "HP" , "DM" , "Obx",

"Dy","IP" ,"COPD" , "As" ,"IBD" , "GORD" , "PUD" ,"IBS" , "Gout" ,"OA", "RA","LHS",

"DS" , "ES","RS" ,"US" ,"RPS" , "HA\_c" , "Frai" **)**

dat\_model**<-**cbind**(**datax**[**,**-**c**(**1**:**53**)]**,dummyage\_c,dummyAPOE\_status,dummyBirthweight\_c,dummyBloodtype,

dummyeducation1,dummyincome,dummyTDI,dummysmoking\_B,

dummyalcohol\_c,dummysleep\_c, dummyPA\_level, dummyGrip\_c,dummysedentary\_c,

dummyLDL\_c, dummyHbA1c\_c,dummyinsulin\_c,dummydepressed\_c,dummyrestless\_c,

dummyLone, dummyhear\_c,dummyFrai,

dummyCHD,dummyStroke,dummyHF,dummyHP,dummyDM,dummyObx,

dummyDy, dummyIP, dummyCOPD, dummyAs,dummyIBD, dummyGORD,

dummyPUD,dummyIBS, dummyGout,dummyOA, dummyRA, dummyLHS,dummyDS,dummyES,

dummyDS,dummyRPS, dummyRS,dummyUS**)**

dat\_y**<-**as.matrix**(**clean\_data**[**,9**])**

library**(**glmnet**)**

lambdas**<-**seq**(**0.05,length.out**=**200**)**

set.seed**(**1234**)**

cv.lasso1**<-**cv.glmnet**(**data.matrix**(**dat\_model**)**,dat\_y,alpha**=**1,nfloods**=**5,family**=**'binomial'**)**

plot**(**cv.lasso1**)**

plot**(**cv.lasso1**$**glmnet.fit,xvar**=**'lambda',label**=**'T'**)**

plot**(**cv.lasso1**$**glmnet,xvar**=**'dev',label**=**'T'**)**

lasso1\_1se**<-**cv.lasso1**$**lambda.1se

lasso1.coef**<-**coef**(**cv.lasso1**$**glmnet.fit,s**=**lasso1\_1se,exact **=** F**)**

lasso1.coef

cv.lasso1**$**lambda.min

cv.lasso1**$**lambda.1se

model1**<-**glm**(**MCI**~**age\_c**+**APOE\_status**+**sleep\_c**+**Frai**+**Stroke**+**HP**+**HF**+**DM**+**Dy**+**IP**+**COPD**+**GORD**+**OA**+**LHS**+**DS**+**RS

, data**=**clean\_data,family**=**'binomial' **)**

summary**(**model1**)$**coefficients

a**<-**exp**(**cbind**(**'OR'**=**coef**(**model1**)**,confint**(**model1**)))**

a

write.csv**(**a,file **=** 'D:/SDU/UKB/nested case-control/Data/MCI\_nested/MCI.csv'**)**

## Boruta

# Boruta

rm**(**list**=**ls**())**

setwd**(**"D:/SDU/UKB/nested case-control/boruta"**)**

library**(**feather**)**

library**(**dplyr**)**

library**(**haven**)**

mcid **<-** read\_sas**(**'D:/SDU/UKB/nested case-control/boruta/mcid\_boruta.sas7bdat'**)**

colnames**(**mcid**)[**1**]** **<-** 'Race'

colnames**(**mcid**)[**2**]** **<-** 'Education levels'

colnames**(**mcid**)[**3**]** **<-** 'Physical activity levels'

colnames**(**mcid**)[**4**]** **<-** 'Income'

colnames**(**mcid**)[**5**]** **<-** 'BMI'

colnames**(**mcid**)[**6**]** **<-** 'Smoking status'

colnames**(**mcid**)[**7**]** **<-** 'APOE e4 status'

colnames**(**mcid**)[**8**]** **<-** 'Handgrip strength'

colnames**(**mcid**)[**9**]** **<-** 'HbA1c'

colnames**(**mcid**)[**10**]** **<-** 'Insulin use status'

colnames**(**mcid**)[**11**]** **<-** 'Alcohol'

colnames**(**mcid**)[**12**]** **<-** 'Sedentary'

colnames**(**mcid**)[**13**]** **<-** 'Anxious'

colnames**(**mcid**)[**14**]** **<-** 'Loneliness'

colnames**(**mcid**)[**15**]** **<-** 'Social isolation'

colnames**(**mcid**)[**16**]** **<-** 'CHD'

colnames**(**mcid**)[**17**]** **<-** 'Stroke'

colnames**(**mcid**)[**18**]** **<-** 'Heart failure'

colnames**(**mcid**)[**19**]** **<-** 'Hypertension'

colnames**(**mcid**)[**20**]** **<-** 'Diabetes'

colnames**(**mcid**)[**21**]** **<-** 'Influenza and pneumonia'

colnames**(**mcid**)[**22**]** **<-** 'COPD'

colnames**(**mcid**)[**23**]** **<-** 'Osteoarthritis Arthritis'

mcid\_boruta**<-**na.omit**(**mcid**)**

View**(**mcid\_boruta**)**

library**(**Boruta**)**

set.seed**(**123**)**

Boruta.mcid **<-** Boruta**(**MCI\_D **~** ., data **=** mcid\_boruta, doTrace **=** 2, maxRuns **=**100,getImp **=** getImpRfZ**)**

Boruta.mcid

attStats**(**Boruta.mcid**)**

par**(**oma**=**c**(**3.5,0.5,0.5,0.5**))**

plot**(**Boruta.mcid,xlab **=** "", xaxt **=** "n",ylab**=**"Importance: Z-Score"**)**

lz**<-**lapply**(**1**:**ncol**(**Boruta.mcid**$**ImpHistory**)**,**function(**i**)**

Boruta.mcid**$**ImpHistory**[**is.finite**(**Boruta.mcid**$**ImpHistory**[**,i**])**,i**])**

names**(**lz**)** **<-** colnames**(**Boruta.mcid**$**ImpHistory**)**

Labels **<-** sort**(**sapply**(**lz,median**))**

axis**(**side **=** 1,las**=**2,labels **=** names**(**Labels**)**,

at **=** 1**:**ncol**(**Boruta.mcid**$**ImpHistory**)**, cex.axis **=**0.6**)**

final.boruta **<-** TentativeRoughFix**(**Boruta.mcid**)**

print**(**final.boruta**)**

plotImpHistory**(**Boruta.mcid,ylab**=**"Importance: Z-Score"**)**

# Building the prediction models

### MCI — All participants

import time  
import pandas as pd  
import numpy as np  
import feather  
  
rawdata = pd.read\_sas(r'D:\nested\_case\_control\predication\MCI\mci\_prediction.sas7bdat')   
  
start = time.time()  
raw\_top5 = rawdata.head(500)  
print(rawdata.columns)  
null\_percentage = rawdata.isnull().sum()/len(rawdata)  
  
for col in null\_percentage.index:  
 if null\_percentage[col] > 0.1:  
 rawdata.drop(col, axis=1, inplace=True)  
  
null\_percentage1 = rawdata.isnull().sum(axis=1)/len(rawdata.columns)  
  
rawdata = rawdata[null\_percentage1 < 0.85]  
null\_percentage2 = rawdata.isnull().sum().sum()/rawdata.size  
num\_rows = rawdata[rawdata['MCI'] == 1].shape[0]  
print(num\_rows)  
unk\_ratio = rawdata[(rawdata == 'unknow') | (rawdata == 'unknown')].count().sum() / rawdata.size  
rawdata.replace(['unknow', 'unknown'], np.nan, inplace=True)  
null\_percentage2 = rawdata.isnull().sum().sum()/rawdata.size  
cleaned\_data = rawdata  
null\_percentage45 = cleaned\_data.isnull().sum().sum()/cleaned\_data.size  
print(cleaned\_data.shape)  
feather.write\_dataframe(cleaned\_data,r'D:\nested\_case\_control\predication\MCI\mci\_prediction.feather')  
num = cleaned\_data.MCI.value\_counts()  
print(num)  
print(cleaned\_data.columns)  
end = time.time()  
duration = end - start  
print('The code took %.2f seconds to run.' % duration)

from warnings import simplefilter  
import feather  
simplefilter(action='ignore', category=Warning)

import miceforest as mf  
import sys  
print(sys.getdefaultencoding())

num\_epochs = 20   
batch\_size = 64   
cleaned\_data = feather.read\_dataframe(r"D:\nested\_case\_control\predication\MCI\mci\_prediction.feather")  
print(cleaned\_data.head())  
  
kds2 = mf.ImputationKernel(  
 data=cleaned\_data,  
 datasets=4,

save\_models=1,

save\_all\_iterations=False,

random\_state=10 )  
  
optimal\_parameters, losses = kds2.tune\_parameters(  
 dataset=0,  
 optimization\_steps=5,  
 device='cpu'   
)  
  
kds2.mice(iterations=3,  
 variable\_parameters=optimal\_parameters,  
 n\_jobs=-1, n\_estimators=25,  
 min\_sum\_hessian\_in\_leaf=0.01) completed\_x\_train0 = kds2.complete\_data(dataset=0, inplace=False)  
  
print(completed\_x\_train0.isnull().sum(0))  
  
feather.write\_dataframe(completed\_x\_train0, r'D:\nested\_case\_control\predication\MCI\mci\_clean.feather')  
print(completed\_x\_train0.shape)  
num = completed\_x\_train0.MCI.value\_counts()  
print(num)

from collections import Counter  
import joblib  
import pandas as pd  
import numpy as np  
import feather  
import sklearn  
from sklearn import model\_selection  
from sklearn.inspection import permutation\_importance  
from sklearn.metrics import roc\_auc\_score, roc\_curve, auc, confusion\_matrix, f1\_score, precision\_score, recall\_score, \  
 classification\_report  
from random import random  
from sklearn.model\_selection import RepeatedStratifiedKFold  
from sklearn.model\_selection import train\_test\_split, GridSearchCV, cross\_val\_score, KFold, LeavePOut  
import matplotlib.pyplot as plt  
from catboost import CatBoostClassifier  
import os  
import sys  
from imblearn.ensemble import EasyEnsembleClassifier  
import pickle  
print(sys.getdefaultencoding())  
import warnings  
warnings.filterwarnings('ignore')

df = feather.read\_dataframe(r'D:\nested\_case\_control\predication\MCI\_new\mci\_clean.feather')  
print(df.columns)

df.rename(columns={df.columns[0]: 'APOE e4 carrier status',  
 df.columns[2]: 'Age at baseline',  
 df.columns[3]: 'Sleep duration',  
 df.columns[4]: 'Stroke',  
 df.columns[5]: 'Heart Failure',  
 df.columns[6]: 'Hypertension',  
 df.columns[7]: 'Diabetes',  
 df.columns[8]: 'Dyslipidemia',  
 df.columns[9]: 'Influenza and pneumonia',  
 df.columns[10]: 'Chronic obstructive pulmonary diseases',  
 df.columns[11]: 'Gastro-esophageal reflux disease',  
 df.columns[12]: 'Osteoarthritis Arthritis',  
 df.columns[13]: 'Lymphohematopoietic system cancer',  
 df.columns[14]: 'Digestive system cancer',  
 df.columns[15]: 'Respiratory system cancer',  
 df.columns[16]: 'Frailty',  
  
 },  
 inplace=True)  
feather.write\_dataframe(df, r'D:\nested\_case\_control\predication\MCI\_new\mci\_onehot.feather')  
print(df.head(10))  
print(df.shape)  
num = df.MCI.value\_counts()  
print(num)

X, y = df.drop('MCI',axis=1), df['MCI']  
X1\_train, X\_test, y1\_train, y\_test = train\_test\_split(X, y, train\_size=0.8, random\_state=123)

train\_val = pd.concat([X1\_train,y1\_train], axis=1)  
test\_val=pd.concat([X\_test,y\_test],axis=1)  
X1, y1 = train\_val.drop('MCI',axis=1), train\_val['MCI']  
print("y\_train",Counter(y1\_train))  
print("y\_test",Counter(y\_test))

KF = KFold(n\_splits =10)  
for train\_index, val\_index in KF.split(X1):  
 print("Train", train\_index, "Val", val\_index)  
 X\_train, X\_val = X1.iloc[train\_index], X1.iloc[val\_index]  
 y\_train, y\_val = y1.iloc[train\_index], y1.iloc[val\_index]  
 print("y\_train:", Counter(y\_train))  
 print("y\_val:", Counter(y\_val))

import tensorflow.\_api.v2.compat.v1 as tf  
B\_acc = tf.keras.metrics.BinaryAccuracy(  
 name='binary\_accuracy',  
 dtype=None,  
 threshold=0.25)  
cv = RepeatedStratifiedKFold(n\_splits=10, n\_repeats=3, random\_state=1)

#1 catboost  
catb = EasyEnsembleClassifier(base\_estimator=CatBoostClassifier(subsample=1.0, rsm=1.0, min\_data\_in\_leaf=310,max\_bin=100,  
 learning\_rate=0.1, l2\_leaf\_reg=1.0,  
 iterations= 100, depth=9),n\_estimators=20, n\_jobs=None,  
 random\_state=123, replacement=False, sampling\_strategy='auto', verbose=0, warm\_start=False)  
  
accuracy\_catb = cross\_val\_score(catb, X\_train, y\_train, scoring='accuracy', cv=cv)  
catb\_fit=catb.fit(X\_train,y\_train)   
catb\_predict = catb\_fit.predict(X\_val)   
catb\_val\_predprob = catb\_fit.predict\_proba(X\_val)[:,1]  
catb\_train\_predprob = catb\_fit.predict\_proba(X\_train)[:,1]  
print("catb\_bacc",B\_acc(y\_val, catb\_predict))  
print("precision ",precision\_score(y\_val, catb\_predict,average="weighted"))  
print("recall",recall\_score(y\_val, catb\_predict,average="weighted"))  
print("f1\_score",f1\_score(y\_val, catb\_predict,average="weighted"))  
joblib.dump(catb\_fit, '../web\_app/MCI/catb.pkl', compress=3)

#2 RF  
from sklearn.ensemble import RandomForestClassifier  
rf = EasyEnsembleClassifier(base\_estimator=RandomForestClassifier(n\_estimators=20),n\_estimators=20, n\_jobs=None,  
 random\_state=123, replacement=False, sampling\_strategy='auto', verbose=0, warm\_start=False)  
accuracy\_rf= cross\_val\_score(rf, X\_train, y\_train, cv=cv, scoring='accuracy')  
rf\_fit=rf.fit(X\_train,y\_train)   
rf\_predict = rf\_fit.predict(X\_val)   
rf\_val\_predprob = rf\_fit.predict\_proba(X\_val)[:,1]  
rf\_train\_predprob = rf\_fit.predict\_proba(X\_train)[:,1]  
print("rf\_Bacc",B\_acc(y\_val, rf\_predict))  
print("precision ",precision\_score(y\_val, rf\_predict,average="weighted"))  
print("recall",recall\_score(y\_val, rf\_predict,average="weighted"))  
print("f1\_score",f1\_score(y\_val, rf\_predict,average="weighted"))

#3 LGB  
import lightgbm as lgb  
lgb = EasyEnsembleClassifier(base\_estimator=lgb.LGBMClassifier(n\_estimators=20,boosting\_type='gbdt', num\_leaves=55, reg\_alpha=0.0, reg\_lambda=1,  
 max\_depth=15, objective='binary',  
 subsample=0.8, colsample\_bytree=0.8, subsample\_freq=1,  
 learning\_rate=0.06, min\_child\_weight=1, random\_state=20, n\_jobs=4),n\_estimators=20,  
 n\_jobs=None,random\_state=123,  
 replacement=False, sampling\_strategy='auto', verbose=0, warm\_start=False)  
accuracy\_lgb= cross\_val\_score(lgb, X\_train, y\_train, cv=cv, scoring='accuracy')  
lgb\_fit=lgb.fit(X\_train, y\_train)  
lgb\_predict = lgb\_fit.predict(X\_val)  
lgb\_val\_predprob = lgb\_fit.predict\_proba(X\_val)[:,1]  
lgb\_train\_predprob = lgb\_fit.predict\_proba(X\_train)[:,1]  
print("lgb\_bacc",B\_acc(y\_val, lgb\_predict))  
print("precision",precision\_score(y\_val, lgb\_predict,average="weighted"))  
print("recall",recall\_score(y\_val, lgb\_predict,average="weighted"))  
print("f1\_score",f1\_score(y\_val, lgb\_predict,average="weighted"))

#4 GBDT  
from sklearn.ensemble import GradientBoostingClassifier  
gbdt = EasyEnsembleClassifier(base\_estimator=GradientBoostingClassifier(n\_estimators=20 ),n\_estimators=20, n\_jobs=None,  
 random\_state=123, replacement=False, sampling\_strategy='auto', verbose=0, warm\_start=False)  
accuracy\_gbdt= cross\_val\_score(gbdt, X\_train, y\_train, cv=cv, scoring='accuracy')  
gbdt\_fit=gbdt.fit(X\_train, y\_train)  
gbdt\_predict = gbdt\_fit.predict(X\_val)  
gbdt\_val\_predprob = gbdt\_fit.predict\_proba(X\_val)[:,1]  
gbdt\_train\_predprob = gbdt\_fit.predict\_proba(X\_train)[:,1]  
print("gbdt\_bacc",B\_acc(y\_val, gbdt\_predict))  
print("precision",precision\_score(y\_val, gbdt\_predict,average="weighted"))  
print("recall",recall\_score(y\_val, gbdt\_predict,average="weighted"))  
print("f1\_score",f1\_score(y\_val, gbdt\_predict,average="weighted"))

#5 dt  
from sklearn.tree import DecisionTreeClassifier  
dt =EasyEnsembleClassifier(base\_estimator=DecisionTreeClassifier(criterion="gini",splitter="best"),n\_estimators=20, n\_jobs=None,  
 random\_state=123, replacement=False, sampling\_strategy='auto', verbose=0, warm\_start=False)  
cv = RepeatedStratifiedKFold(n\_splits=10, n\_repeats=3, random\_state=1)  
accuracy\_dt= cross\_val\_score(dt, X\_train, y\_train, cv=cv, scoring='accuracy')  
dt\_fit=dt.fit(X\_train, y\_train)  
dt\_predict = dt\_fit.predict(X\_val)  
dt\_val\_predprob = dt\_fit.predict\_proba(X\_val)[:,1]  
dt\_train\_predprob = dt\_fit.predict\_proba(X\_train)[:,1]  
print("dt\_bacc",B\_acc(y\_val, dt\_predict))  
print("precision",precision\_score(y\_val, dt\_predict,average="weighted"))  
print("recall",recall\_score(y\_val, dt\_predict,average="weighted"))  
print("f1\_score",f1\_score(y\_val, dt\_predict,average="weighted"))

#6 ad  
from sklearn.ensemble import AdaBoostClassifier  
ad =EasyEnsembleClassifier(base\_estimator=AdaBoostClassifier(n\_estimators=20,learning\_rate=0.1),  
 n\_estimators=20, n\_jobs=None,random\_state=123, replacement=False,  
 sampling\_strategy='auto', verbose=0, warm\_start=False)  
cv = RepeatedStratifiedKFold(n\_splits=10, n\_repeats=3, random\_state=1)  
accuracy\_ad= cross\_val\_score(ad, X\_train, y\_train, cv=cv, scoring='accuracy')  
ad\_fit=ad.fit(X\_train, y\_train)  
ad\_predict = ad\_fit.predict(X\_val)   
ad\_val\_predprob = ad\_fit.predict\_proba(X\_val)[:,1]  
ad\_train\_predprob = ad\_fit.predict\_proba(X\_train)[:,1]  
print("ad\_bacc",B\_acc(y\_val, ad\_predict))  
print("precision",precision\_score(y\_val, ad\_predict,average="weighted"))  
print("recall",recall\_score(y\_val, ad\_predict,average="weighted"))  
print("f1\_score",f1\_score(y\_val, ad\_predict,average="weighted"))

#7 lg  
from sklearn.linear\_model import LogisticRegression  
lg=EasyEnsembleClassifier(base\_estimator=LogisticRegression(C=1.0,solver= "lbfgs",max\_iter=100),  
 n\_estimators=20, n\_jobs=None,random\_state=123, replacement=False,  
 sampling\_strategy='auto', verbose=0, warm\_start=False)  
accuracy\_lg= cross\_val\_score(lg, X\_train, y\_train, cv=cv, scoring='accuracy')  
lg\_fit=lg.fit(X\_train, y\_train)  
lg\_predict = lg\_fit.predict(X\_val)  
lg\_val\_predprob = lg\_fit.predict\_proba(X\_val)[:,1]  
lg\_train\_predprob = lg\_fit.predict\_proba(X\_train)[:,1]  
print("lg\_bacc",B\_acc(y\_val , lg\_predict))  
print("precision",precision\_score(y\_val, lg\_predict,average="weighted"))  
print("recall",recall\_score(y\_val, lg\_predict,average="weighted"))  
print("f1\_score",f1\_score(y\_val, lg\_predict,average="weighted"))

#8 svm  
from sklearn.svm import LinearSVC  
svm = EasyEnsembleClassifier(base\_estimator=LinearSVC(penalty='l2',loss='hinge',C=5.0,max\_iter=1000),  
 n\_estimators=20, n\_jobs=None,random\_state=123, replacement=False,  
 sampling\_strategy='auto', verbose=0, warm\_start=False)  
accuracy\_svm= cross\_val\_score(svm, X\_train, y\_train, cv=cv, scoring='accuracy')  
svm\_fit=svm.fit(X\_train, y\_train)  
svm\_predict = svm\_fit.predict(X\_val)  
svm\_val\_predprob = svm\_fit.predict\_proba(X\_val)[:,1]  
svm\_train\_predprob = svm\_fit.predict\_proba(X\_train)[:,1]  
print("svm\_bacc",B\_acc(y\_val, svm\_predict))  
print("precision",precision\_score(y\_val, svm\_predict,average="weighted"))  
print("recall",recall\_score(y\_val, svm\_predict,average="weighted"))  
print("f1\_score",f1\_score(y\_val, svm\_predict,average="weighted"))

#9 XGBoost  
import xgboost as xgb  
xgb= EasyEnsembleClassifier(base\_estimator=xgb.XGBClassifier(n\_estimators=20,max\_depth=4,learning\_rate=0.1,  
 subsample=0.7,colsample\_bytree=0.7,eval\_metric='error'),  
 n\_estimators=20, n\_jobs=None,random\_state=123, replacement=False,  
 sampling\_strategy='auto', verbose=0, warm\_start=False)  
accuracy\_xgb= cross\_val\_score(xgb, X\_train, y\_train, cv=cv, scoring='accuracy')  
xgb\_fit=xgb.fit(X\_train, y\_train)  
xgb\_predict = xgb\_fit.predict(X\_val)  
xgb\_val\_predprob = xgb\_fit.predict\_proba(X\_val)[:,1]  
xgb\_train\_predprob = xgb\_fit.predict\_proba(X\_train)[:,1]  
print("xgb\_bacc",B\_acc(y\_val, xgb\_predict))  
print("precision",precision\_score(y\_val, xgb\_predict,average="weighted"))  
print("recall",recall\_score(y\_val, xgb\_predict,average="weighted"))  
print("f1\_score",f1\_score(y\_val, xgb\_predict,average="weighted"))

#10 MLP  
from sklearn.neural\_network import MLPClassifier  
mlp = EasyEnsembleClassifier(base\_estimator=MLPClassifier(solver='lbfgs', alpha=1e-5),  
 n\_estimators=20, n\_jobs=None,random\_state=123, replacement=False,  
 sampling\_strategy='auto', verbose=0, warm\_start=False)  
  
hidden\_layer\_sizes=(5, 2)  
accuracy\_mlp= cross\_val\_score(mlp, X\_train, y\_train, cv=10, scoring='accuracy')  
mlp\_fit=mlp.fit(X\_train, y\_train)  
mlp\_predict = mlp\_fit.predict(X\_val)  
mlp\_val\_predprob = mlp\_fit.predict\_proba(X\_val)[:,1]  
mlp\_train\_predprob = mlp\_fit.predict\_proba(X\_train)[:,1]  
print("mlp\_bacc",B\_acc(y\_val, mlp\_predict))  
print("precision",precision\_score(y\_val, mlp\_predict,average="weighted"))  
print("recall",recall\_score(y\_val, mlp\_predict,average="weighted"))  
print("f1\_score",f1\_score(y\_val, mlp\_predict,average="weighted"))

def multi\_models\_roc2(names, sampling\_methods, colors,y\_predprobs, X\_train, y\_train, save=True, dpin=100):  
 plt.figure(figsize=(20, 20), dpi=dpin)  
 for (name, method, colorname,y\_train\_predprob) in zip(names, sampling\_methods, colors,y\_predprobs):  
  
 y\_train\_preds = method.predict(X\_train)  
 fpr, tpr, thresholds = roc\_curve(y\_train, y\_train\_predprob, pos\_label=1)  
 plt.plot(fpr, tpr, lw=5, label='{} (AUC={:.3f})'.format(name, auc(fpr, tpr)),color = colorname)  
 plt.plot([0, 1], [0, 1], '--', lw=5, color = 'grey')  
 plt.axis('square')  
 plt.xlim([0, 1])  
 plt.ylim([0, 1])  
 plt.xlabel('False Positive Rate',fontsize=40)  
 plt.ylabel('True Positive Rate',fontsize=40)  
 plt.title('ROC Curve (MCI-Training Set)',fontsize=40)  
 plt.legend(loc='lower right',fontsize=40)  
 if save:  
 plt.savefig('multi\_models\_roc2.png')  
 return plt  
names = ['CatBoost',  
 'RF',  
 'LightGBM',  
 'GBDT',  
 'Decision Tree',  
 'AdaBoost',  
 'LG',  
 'LinearSVC',  
 'XGBoost',  
 'MLP'  
 ]  
  
sampling\_methods = [catb,  
 rf,  
 lgb,  
 gbdt,  
 dt,  
 ad,  
 lg,  
 svm,  
 xgb,  
 mlp  
 ]  
  
colors = ['crimson',  
 'orange',  
 'yellow',  
 'mediumseagreen',  
 'steelblue',  
 'mediumpurple' ,  
 'black',  
 'silver',  
 'lightgreen',  
 'brown'  
 ]  
y\_predprobs = [catb\_train\_predprob,  
 rf\_train\_predprob,  
 lgb\_train\_predprob,  
 gbdt\_train\_predprob,  
 dt\_train\_predprob,  
 ad\_train\_predprob,  
 lg\_train\_predprob,  
 svm\_train\_predprob,  
 xgb\_train\_predprob,  
 mlp\_train\_predprob  
 ]  
  
train\_roc\_graph = multi\_models\_roc2(names, sampling\_methods, colors,y\_predprobs, X\_train, y\_train, save = True)  
train\_roc\_graph.savefig(r'D:\nested\_case\_control\MCI\_Train.png')  
  
  
def multi\_models\_roc(names, sampling\_methods, colors,y\_predprobs ,X\_val, y\_val, save=True, dpin=100):  
 plt.figure(figsize=(20, 20), dpi=dpin)  
 for (name, method, colorname,y\_val\_predprob) in zip(names, sampling\_methods, colors,y\_predprobs):  
 y\_val\_preds = method.predict(X\_val)  
 fpr, tpr, thresholds = roc\_curve(y\_val, y\_val\_predprob, pos\_label=1)  
  
 plt.plot(fpr, tpr, lw=5, label='{} (AUC={:.3f})'.format(name, auc(fpr, tpr)),color = colorname)  
 plt.plot([0, 1], [0, 1], '--', lw=5, color = 'grey')  
 plt.axis('square')  
 plt.xlim([0, 1])  
 plt.ylim([0, 1])  
 plt.xlabel('False Positive Rate',fontsize=40)  
 plt.ylabel('True Positive Rate',fontsize=40)  
 plt.title('ROC Curve (MCI——Cross Validation Set)',fontsize=40)  
 plt.legend(loc='lower right',fontsize=40)  
  
 if save:  
 plt.savefig('multi\_models\_roc.png')  
  
 return plt  
  
  
names = ['CatBoost',  
 'RF',  
 'LightGBM',  
 'GBDT',  
 'Decision Tree',  
 'AdaBoost',  
 'LG',  
 'LinearSVC',  
 'XGBoost',  
 'MLP'  
 ]  
sampling\_methods = [catb,  
 rf,  
 lgb,  
 gbdt,  
 dt,  
 ad,  
 lg,  
 svm,  
 xgb,  
 mlp  
 ]  
  
colors = ['crimson',  
 'orange',  
 'yellow',  
 'mediumseagreen',  
 'steelblue',  
 'mediumpurple' ,  
 'black',  
 'silver',  
 'lightgreen',  
 'brown'  
 ]  
y\_predprobs= [catb\_val\_predprob,  
 rf\_val\_predprob,  
 lgb\_val\_predprob,  
 gbdt\_val\_predprob,  
 dt\_val\_predprob,  
 ad\_val\_predprob,  
 lg\_val\_predprob,  
 svm\_val\_predprob,  
 xgb\_val\_predprob,  
 mlp\_val\_predprob  
 ]  
test\_roc\_graph = multi\_models\_roc(names, sampling\_methods, colors, y\_predprobs,X\_val, y\_val, save = True)  
test\_roc\_graph.savefig(r'D:\nested\_case\_control\MCI\_Val.png')  
  
catb = EasyEnsembleClassifier(base\_estimator=CatBoostClassifier(subsample=1.0, rsm=1.0, min\_data\_in\_leaf=310,max\_bin=100,  
 learning\_rate=0.1, l2\_leaf\_reg=1.0,  
 iterations= 100, depth=9),n\_estimators=20, n\_jobs=None,  
 random\_state=123, replacement=False, sampling\_strategy='auto', verbose=0, warm\_start=False)  
  
accuracy\_catb = cross\_val\_score(catb, X1\_train, y1\_train, scoring='accuracy', cv=cv)  
catb\_fit=catb.fit(X1\_train,y1\_train) # 拟合  
catb\_predict = catb\_fit.predict(X\_test) # 预测  
catb\_val\_predprob = catb\_fit.predict\_proba(X\_test)[:,1]  
catb\_train\_predprob = catb\_fit.predict\_proba(X1\_train)[:,1]  
print("catb\_bacc",B\_acc(y\_test, catb\_predict))  
print("precision",precision\_score(y\_test, catb\_predict,average="weighted"))  
print("recall",recall\_score(y\_test, catb\_predict,average="weighted"))  
print("f1\_score",f1\_score(y\_test, catb\_predict,average="weighted"))

def multi\_models\_roc(names, sampling\_methods, colors,y\_predprobs ,X\_tset, y\_test, save=True, dpin=100):  
 plt.figure(figsize=(20, 20), dpi=dpin)  
 for (name, method, colorname,y\_test\_predprob) in zip(names, sampling\_methods, colors,y\_predprobs):  
 y\_test\_preds = method.predict(X\_test)  
 fpr, tpr, thresholds = roc\_curve(y\_test, y\_test\_predprob, pos\_label=1)  
  
 plt.plot(fpr, tpr, lw=5, label='{} (AUC={:.3f})'.format(name, auc(fpr, tpr)),color = colorname)  
 plt.plot([0, 1], [0, 1], '--', lw=5, color = 'grey')  
 plt.axis('square')  
 plt.xlim([0, 1])  
 plt.ylim([0, 1])  
 plt.xlabel('False Positive Rate',fontsize=40)  
 plt.ylabel('True Positive Rate',fontsize=40)  
 plt.title('ROC Curve (MCI——Test Set)',fontsize=40)  
 plt.legend(loc='lower right',fontsize=40)  
 if save:  
 plt.savefig('multi\_models\_roc.png')  
 return plt  
names = ['CatBoost' ]  
sampling\_methods = [catb ]  
colors = [ 'steelblue']  
y\_predprobs= [ catb\_val\_predprob]  
test\_roc\_graph = multi\_models\_roc(names, sampling\_methods, colors, y\_predprobs,X\_test, y\_test, save = True)  
test\_roc\_graph.savefig(r'D:\nested\_case\_control\MCI\_test.png')  
  
#shap  
import shap  
shap.initjs()  
X\_train\_summary = shap.kmeans(X1\_train, 10)  
explainer = shap.KernelExplainer(catb.predict,X\_train\_summary,gpu\_model='Ture')  
shap\_values= explainer.shap\_values(X\_test)  
shap.summary\_plot(shap\_values,X\_test,max\_display=10)  
shap.summary\_plot(shap\_values,X\_test,plot\_type='bar',color='steelblue')  
# calibration\_curve  
from sklearn.calibration import CalibratedClassifierCV, CalibrationDisplay  
import matplotlib.pyplot as plt  
import lightgbm as lgb  
from matplotlib.gridspec import GridSpec  
clf\_list = [(CatBoostClassifier(subsample=1.0, rsm=1.0, min\_data\_in\_leaf=310,max\_bin=100,  
 learning\_rate=0.1, l2\_leaf\_reg=1.0,  
 iterations= 100, depth=9), "CatBoost")]  
fig = plt.figure(figsize=(10, 10))  
colors = 'steelblue'  
ax\_calibration\_curve = fig.add\_subplot(111)  
calibration\_displays = {}  
for i, (clf, name) in enumerate(clf\_list):  
 clf.fit(X1\_train, y1\_train)  
 display = CalibrationDisplay.from\_estimator(  
 clf,  
 X\_test,  
 y\_test,  
 n\_bins=6,  
 name=name,  
 ax=ax\_calibration\_curve,  
 color=colors,  
 )  
 calibration\_displays[name] = display  
  
ax\_calibration\_curve.grid()  
ax\_calibration\_curve.set\_title("Calibration plots (CatBoost)")  
ax\_calibration\_curve.set\_xscale('log',base=10,subs=[10,100])  
ax\_calibration\_curve.set\_yscale('log',base=10,subs=[10,100])  
ax\_calibration\_curve.set\_xlim(0.001,1)  
ax\_calibration\_curve.set\_ylim(0.001,1)  
ax\_calibration\_curve.set\_xticklabels(['0','0.001','0.01', '0.1', '1'])  
ax\_calibration\_curve.set\_yticklabels(['0','0.001','0.01', '0.1', '1'])  
ax\_calibration\_curve.set\_xlabel("Mean predicted probability (Positive class: 1:0)",fontsize=23)  
ax\_calibration\_curve.set\_ylabel("Fraction of positives (Positive class: 1:0)",fontsize=23)  
ax\_calibration\_curve.set\_title("Calibration plots (CatBoost)",fontsize=23)  
ax\_calibration\_curve.legend(loc='lower right',fontsize=23)  
plt.tight\_layout()  
plt.show()

### MCI — Female population

import time  
import pandas as pd  
import numpy as np  
import feather  
  
rawdata = pd.read\_sas(r'D:\nested\_case\_control\predication\MCI\_female\mci\_female\_prediction.sas7bdat')   
  
start = time.time()  
raw\_top5 = rawdata.head(500)  
print(rawdata.columns)  
null\_percentage = rawdata.isnull().sum()/len(rawdata)  
  
for col in null\_percentage.index:  
 if null\_percentage[col] > 0.1:  
 rawdata.drop(col, axis=1, inplace=True)  
  
null\_percentage1 = rawdata.isnull().sum(axis=1)/len(rawdata.columns)  
  
rawdata = rawdata[null\_percentage1 < 0.85]  
null\_percentage2 = rawdata.isnull().sum().sum()/rawdata.size  
num\_rows = rawdata[rawdata['MCI'] == 1].shape[0]  
print(num\_rows)  
unk\_ratio = rawdata[(rawdata == 'unknow') | (rawdata == 'unknown')].count().sum() / rawdata.size  
rawdata.replace(['unknow', 'unknown'], np.nan, inplace=True)  
null\_percentage2 = rawdata.isnull().sum().sum()/rawdata.size  
cleaned\_data = rawdata  
null\_percentage45 = cleaned\_data.isnull().sum().sum()/cleaned\_data.size  
print(cleaned\_data.shape)  
feather.write\_dataframe(cleaned\_data,r'D:\nested\_case\_control\predication\MCI\_female\mci\_female\_prediction.feather')  
num = cleaned\_data.MCI.value\_counts()  
print(num)  
print(cleaned\_data.columns)  
end = time.time()  
duration = end - start  
print('The code took %.2f seconds to run.' % duration)

from warnings import simplefilter  
import feather  
simplefilter(action='ignore', category=Warning)

import miceforest as mf  
import sys  
print(sys.getdefaultencoding())

num\_epochs = 20   
batch\_size = 64   
cleaned\_data = feather.read\_dataframe(r"D:\nested\_case\_control\predication\ MCI\_female\mci\_female\_prediction.feather")  
print(cleaned\_data.head())  
  
kds2 = mf.ImputationKernel(  
 data=cleaned\_data,  
 datasets=4,

save\_models=1,

save\_all\_iterations=False,

random\_state=10 )  
  
optimal\_parameters, losses = kds2.tune\_parameters(  
 dataset=0,  
 optimization\_steps=5,  
 device='cpu')  
  
kds2.mice(iterations=3,  
 variable\_parameters=optimal\_parameters,  
 n\_jobs=-1, n\_estimators=25,  
 min\_sum\_hessian\_in\_leaf=0.01) completed\_x\_train0 = kds2.complete\_data(dataset=0, inplace=False)  
print(completed\_x\_train0.isnull().sum(0))  
feather.write\_dataframe(completed\_x\_train0, r'D:\nested\_case\_control\predication\ MCI\_female\mci\_female\_clean.feather')  
print(completed\_x\_train0.shape)  
num = completed\_x\_train0.MCI.value\_counts()  
print(num)

from collections import Counter  
import pandas as pd  
import numpy as np  
import feather  
from sklearn.metrics import roc\_auc\_score, roc\_curve, auc, confusion\_matrix, f1\_score, precision\_score, recall\_score  
from sklearn.model\_selection import RepeatedStratifiedKFold  
from sklearn.model\_selection import train\_test\_split, GridSearchCV, cross\_val\_score, KFold, LeavePOut  
from catboost import CatBoostClassifier  
import sys  
print(sys.getdefaultencoding())  
import warnings  
warnings.filterwarnings('ignore')  
  
df = feather.read\_dataframe(r'D:\nested\_case\_control\predication\MCI\_female\mci\_female\_clean.feather')  
print(df.columns)  
  
df.rename(columns={ df.columns[0]: 'APOE e4 carrier status',  
 df.columns[2]: 'Age at baseline',  
 df.columns[3]: 'Sleep duration',  
 df.columns[4]: 'Age at first live birth',  
 df.columns[5]: 'Age at last live birth',  
 df.columns[6]: 'Age at nature menopause',  
 df.columns[7]: 'Stroke',  
 df.columns[8]: 'Heart Failure',  
 df.columns[9]: 'Hypertension',  
 df.columns[10]: 'Diabetes',  
 df.columns[11]: 'Dyslipidemia',  
 df.columns[12]: 'Influenza and pneumonia',  
 df.columns[13]: 'Chronic obstructive pulmonary diseases',  
 df.columns[14]: 'Gastro-esophageal reflux disease',  
 df.columns[15]: 'Osteoarthritis Arthritis',  
 df.columns[16]: 'Lymphohematopoietic system cancer',  
 df.columns[17]: 'Digestive system cancer',  
 df.columns[18]: 'Respiratory system cancer',

df.columns[19]: 'Age at menarche',  
 df.columns[20]: 'Oral contraceptive pills',  
 df.columns[21]: 'Hormone replacement therapy',  
 df.columns[22]: 'Number of children',  
 df.columns[23]: 'Age at surgical menopause',  
 df.columns[24]: 'Frailty',  
 },  
 inplace=True)  
feather.write\_dataframe(df, r'D:\nested\_case\_control\predication\MCI\_female\mci\_female\_onehot.feather')  
print(df.head(10))  
print(df.shape)  
num = onehot\_data.MCI.value\_counts()  
print(num)  
  
X, y = df.drop('MCI',axis=1), df['MCI']  
X1\_train, X\_test, y1\_train, y\_test = train\_test\_split(X, y, train\_size=0.8, random\_state=123)  
  
train\_val = pd.concat([X1\_train,y1\_train], axis=1)  
X1, y1 = train\_val.drop('MCI',axis=1), train\_val['MCI']  
print("y\_train",Counter(y1\_train))  
print("y\_test",Counter(y\_test))  
  
  
KF = KFold(n\_splits =10)  
for train\_index, val\_index in KF.split(X1):  
 print("Train", train\_index, "Val", val\_index)  
 X\_train, X\_val = X1.iloc[train\_index], X1.iloc[val\_index]  
 y\_train, y\_val = y1.iloc[train\_index], y1.iloc[val\_index]  
 print("X\_train:", X\_train)  
 print("X\_val:", X\_val)  
  
from imblearn.ensemble import EasyEnsembleClassifier  
import tensorflow.\_api.v2.compat.v1 as tf  
B\_acc = tf.keras.metrics.BinaryAccuracy(  
 name='binary\_accuracy',  
 dtype=None,  
 threshold=0.25)  
cv = RepeatedStratifiedKFold(n\_splits=10, n\_repeats=3, random\_state=1)  
  
##1 catboost  
catb = EasyEnsembleClassifier(base\_estimator=CatBoostClassifier(subsample=1.0, rsm=1.0, min\_data\_in\_leaf=310,max\_bin=100,  
 learning\_rate=0.1, l2\_leaf\_reg=1.0,  
 iterations= 100, depth=9),n\_estimators=20, n\_jobs=None,  
 random\_state=123, replacement=False, sampling\_strategy='auto', verbose=0, warm\_start=False)  
  
accuracy\_catb = cross\_val\_score(catb, X\_train, y\_train, scoring='accuracy', cv=cv)  
catb\_fit=catb.fit(X\_train,y\_train)  
catb\_predict = catb\_fit.predict(X\_val)  
catb\_val\_predprob = catb\_fit.predict\_proba(X\_val)[:,1]  
catb\_train\_predprob = catb\_fit.predict\_proba(X\_train)[:,1]  
print("catb\_bacc",B\_acc(y\_val, catb\_predict))  
print("precision",precision\_score(y\_val, catb\_predict,average="weighted"))  
print("recall",recall\_score(y\_val, catb\_predict,average="weighted"))  
print("f1\_score",f1\_score(y\_val, catb\_predict,average="weighted"))  
  
#2 rf  
from sklearn.ensemble import RandomForestClassifier  
rf = EasyEnsembleClassifier(base\_estimator=RandomForestClassifier(n\_estimators=100,min\_samples\_split=14, min\_samples\_leaf= 14,  
 max\_features= 'sqrt', max\_depth=1, criterion='entropy'),n\_estimators=20, n\_jobs=None,  
 random\_state=123, replacement=False, sampling\_strategy='auto', verbose=0, warm\_start=False)  
accuracy\_rf= cross\_val\_score(rf, X\_train, y\_train, cv=cv, scoring='accuracy')  
rf\_fit=rf.fit(X\_train,y\_train)  
rf\_predict = rf\_fit.predict(X\_val)  
rf\_val\_predprob = rf\_fit.predict\_proba(X\_val)[:,1]  
rf\_train\_predprob = rf\_fit.predict\_proba(X\_train)[:,1]  
print("rf\_Bacc",B\_acc(y\_val, rf\_predict))  
print("precision",precision\_score(y\_val, rf\_predict,average="weighted"))  
print("recall",recall\_score(y\_val, rf\_predict,average="weighted"))  
print("f1\_score",f1\_score(y\_val, rf\_predict,average="weighted"))  
  
#3 LGB  
import lightgbm as lgb  
lgb = EasyEnsembleClassifier(base\_estimator=lgb.LGBMClassifier(n\_estimators=20,boosting\_type='gbdt', num\_leaves=55, reg\_alpha=0.0, reg\_lambda=1,  
 max\_depth=15, objective='binary',  
 subsample=0.8, colsample\_bytree=0.8, subsample\_freq=1,  
 learning\_rate=0.06, min\_child\_weight=1, random\_state=20, n\_jobs=4),n\_estimators=20,  
 n\_jobs=None,random\_state=123,  
 replacement=False, sampling\_strategy='auto', verbose=0, warm\_start=False)  
accuracy\_lgb= cross\_val\_score(lgb, X\_train, y\_train, cv=cv, scoring='accuracy')  
lgb\_fit=lgb.fit(X\_train, y\_train)  
lgb\_predict = lgb\_fit.predict(X\_val)  
lgb\_val\_predprob = lgb\_fit.predict\_proba(X\_val)[:,1]  
lgb\_train\_predprob = lgb\_fit.predict\_proba(X\_train)[:,1]  
print("lgb\_acc",B\_acc(y\_val, lgb\_predict))  
print("precision",precision\_score(y\_val, lgb\_predict,average="weighted"))  
print("recall",recall\_score(y\_val, lgb\_predict,average="weighted"))  
print("f1\_score",f1\_score(y\_val, lgb\_predict,average="weighted"))  
  
  
  
#4GBDT  
from sklearn.ensemble import GradientBoostingClassifier  
gbdt = EasyEnsembleClassifier(base\_estimator=GradientBoostingClassifier(n\_estimators=20 ),n\_estimators=20, n\_jobs=None,  
 random\_state=123, replacement=False, sampling\_strategy='auto', verbose=0, warm\_start=False)  
accuracy\_gbdt= cross\_val\_score(gbdt, X\_train, y\_train, cv=cv, scoring='accuracy')  
gbdt\_fit=gbdt.fit(X\_train, y\_train)  
gbdt\_predict = gbdt\_fit.predict(X\_val)  
gbdt\_val\_predprob = gbdt\_fit.predict\_proba(X\_val)[:,1]  
gbdt\_train\_predprob = gbdt\_fit.predict\_proba(X\_train)[:,1]  
print("交叉验证后的gbdt平均准确率：", accuracy\_gbdt.mean())  
print("gbdt\_bacc",B\_acc(y\_val, gbdt\_predict))  
print("precision",precision\_score(y\_val, gbdt\_predict,average="weighted"))  
print("recall",recall\_score(y\_val, gbdt\_predict,average="weighted"))  
print("f1\_score",f1\_score(y\_val, gbdt\_predict,average="weighted"))  
  
#5 dt  
from sklearn.tree import DecisionTreeClassifier  
dt =EasyEnsembleClassifier(base\_estimator=DecisionTreeClassifier(min\_samples\_split=186, min\_samples\_leaf=468,  
 max\_features='log2', max\_depth=164),n\_estimators=20, n\_jobs=None,  
 random\_state=123, replacement=False, sampling\_strategy='auto', verbose=0, warm\_start=False)  
cv = RepeatedStratifiedKFold(n\_splits=10, n\_repeats=3, random\_state=1)  
accuracy\_dt= cross\_val\_score(dt, X\_train, y\_train, cv=cv, scoring='accuracy')  
dt\_fit=dt.fit(X\_train, y\_train)  
dt\_predict = dt\_fit.predict(X\_val)  
dt\_val\_predprob = dt\_fit.predict\_proba(X\_val)[:,1]  
dt\_train\_predprob = dt\_fit.predict\_proba(X\_train)[:,1]  
print("dt\_bacc",B\_acc(y\_val, dt\_predict))  
print("precision",precision\_score(y\_val, dt\_predict,average="weighted"))  
print("recall",recall\_score(y\_val, dt\_predict,average="weighted"))  
print("f1\_score",f1\_score(y\_val, dt\_predict,average="weighted"))  
  
#6 ad  
from sklearn.ensemble import AdaBoostClassifier  
ad =EasyEnsembleClassifier(base\_estimator=AdaBoostClassifier(n\_estimators=20,learning\_rate=0.1),  
 n\_estimators=20, n\_jobs=None,random\_state=123, replacement=False,  
 sampling\_strategy='auto', verbose=0, warm\_start=False)  
cv = RepeatedStratifiedKFold(n\_splits=10, n\_repeats=3, random\_state=1)  
accuracy\_ad= cross\_val\_score(ad, X\_train, y\_train, cv=cv, scoring='accuracy')  
ad\_fit=ad.fit(X\_train, y\_train)  
ad\_predict = ad\_fit.predict(X\_val)  
ad\_val\_predprob = ad\_fit.predict\_proba(X\_val)[:,1]  
ad\_train\_predprob = ad\_fit.predict\_proba(X\_train)[:,1]  
print("ad\_bacc",B\_acc(y\_val, ad\_predict))  
print("precision",precision\_score(y\_val, ad\_predict,average="weighted"))  
print("recall",recall\_score(y\_val, ad\_predict,average="weighted"))  
print("f1\_score",f1\_score(y\_val, ad\_predict,average="weighted"))  
  
#7 lg  
from sklearn.linear\_model import LogisticRegression  
lg=EasyEnsembleClassifier(base\_estimator=LogisticRegression(C=1.0,solver= "lbfgs",max\_iter=100),  
 n\_estimators=20, n\_jobs=None,random\_state=123, replacement=False,  
 sampling\_strategy='auto', verbose=0, warm\_start=False)  
accuracy\_lg= cross\_val\_score(lg, X\_train, y\_train, cv=cv, scoring='accuracy')  
lg\_fit=lg.fit(X\_train, y\_train)  
lg\_predict = lg\_fit.predict(X\_val)  
lg\_val\_predprob = lg\_fit.predict\_proba(X\_val)[:,1]  
lg\_train\_predprob = lg\_fit.predict\_proba(X\_train)[:,1]  
print("lg\_bacc",B\_acc(y\_val , lg\_predict))  
print("precision",precision\_score(y\_val, lg\_predict,average="weighted"))  
print("recall",recall\_score(y\_val, lg\_predict,average="weighted"))  
print("f1\_score",f1\_score(y\_val, lg\_predict,average="weighted"))  
  
#8 svm  
from sklearn.svm import LinearSVC  
svm = EasyEnsembleClassifier(base\_estimator=LinearSVC(penalty='l2',loss='hinge',C=5.0,max\_iter=1000),  
 n\_estimators=20, n\_jobs=None,random\_state=123, replacement=False,  
 sampling\_strategy='auto', verbose=0, warm\_start=False)  
accuracy\_svm= cross\_val\_score(svm, X\_train, y\_train, cv=cv, scoring='accuracy')  
svm\_fit=svm.fit(X\_train, y\_train)  
svm\_predict = svm\_fit.predict(X\_val)  
svm\_val\_predprob = svm\_fit.predict\_proba(X\_val)[:,1]  
svm\_train\_predprob = svm\_fit.predict\_proba(X\_train)[:,1]  
print("svm\_bacc",B\_acc(y\_val, svm\_predict))  
print("precision",precision\_score(y\_val, svm\_predict,average="weighted"))  
print("recall",recall\_score(y\_val, svm\_predict,average="weighted"))  
print("f1\_score",f1\_score(y\_val, svm\_predict,average="weighted"))  
  
#9 XGBoost  
import xgboost as xgb  
xgb= EasyEnsembleClassifier(base\_estimator=xgb.XGBClassifier(n\_estimators=20,max\_depth=4,learning\_rate=0.1,  
 subsample=0.7,colsample\_bytree=0.7,eval\_metric='error'),  
 n\_estimators=20, n\_jobs=None,random\_state=123, replacement=False,  
 sampling\_strategy='auto', verbose=0, warm\_start=False)  
accuracy\_xgb= cross\_val\_score(xgb, X\_train, y\_train, cv=cv, scoring='accuracy')  
xgb\_fit=xgb.fit(X\_train, y\_train)  
xgb\_predict = xgb\_fit.predict(X\_val)  
xgb\_val\_predprob = xgb\_fit.predict\_proba(X\_val)[:,1]  
xgb\_train\_predprob = xgb\_fit.predict\_proba(X\_train)[:,1]  
print("xgb\_bacc",B\_acc(y\_val, xgb\_predict))  
print("precision",precision\_score(y\_val, xgb\_predict,average="weighted"))  
print("recall",recall\_score(y\_val, xgb\_predict,average="weighted"))  
print("f1\_score",f1\_score(y\_val, xgb\_predict,average="weighted"))  
  
  
#10 MLP  
from sklearn.neural\_network import MLPClassifier  
mlp = EasyEnsembleClassifier(base\_estimator=MLPClassifier(solver='adam', max\_iter=10, learning\_rate='constant',  
 hidden\_layer\_sizes=(100, 100, 100), alpha=0.021544346900318846,  
 activation='logistic'),  
 n\_estimators=20, n\_jobs=None,random\_state=123, replacement=False,  
 sampling\_strategy='auto', verbose=0, warm\_start=False)  
  
hidden\_layer\_sizes=(5, 2)  
accuracy\_mlp= cross\_val\_score(mlp, X\_train, y\_train, cv=10, scoring='accuracy')  
mlp\_fit=mlp.fit(X\_train, y\_train)  
mlp\_predict = mlp\_fit.predict(X\_val)  
mlp\_val\_predprob = mlp\_fit.predict\_proba(X\_val)[:,1]  
mlp\_train\_predprob = mlp\_fit.predict\_proba(X\_train)[:,1]  
print("mlp\_bacc",B\_acc(y\_val, mlp\_predict))  
print("precision",precision\_score(y\_val, mlp\_predict,average="weighted"))  
print("recall",recall\_score(y\_val, mlp\_predict,average="weighted"))  
print("f1\_score",f1\_score(y\_val, mlp\_predict,average="weighted"))  
  
#ROC curves  
def multi\_models\_roc2(names, sampling\_methods, colors,y\_predprobs, X\_train, y\_train, save=True, dpin=100):  
 plt.figure(figsize=(20, 20), dpi=dpin)  
 for (name, method, colorname,y\_train\_predprob) in zip(names, sampling\_methods, colors,y\_predprobs):  
 y\_train\_preds = method.predict(X\_train)  
 fpr, tpr, thresholds = roc\_curve(y\_train, y\_train\_predprob, pos\_label=1)  
 plt.plot(fpr, tpr, lw=5, label='{} (AUC={:.3f})'.format(name, auc(fpr, tpr)),color = colorname)  
 plt.plot([0, 1], [0, 1], '--', lw=5, color = 'grey')  
 plt.axis('square')  
 plt.xlim([0, 1])  
 plt.ylim([0, 1])  
 plt.xlabel('False Positive Rate',fontsize=40)  
 plt.ylabel('True Positive Rate',fontsize=40)  
 plt.title('ROC Curve (MCI——Training Set)',fontsize=40)  
 plt.legend(loc='lower right',fontsize=40)  
 if save:  
 plt.savefig('multi\_models\_roc2.png')  
 return plt  
names = ['CatBoost',  
 'RF',  
 'LightGBM',  
 'GBDT',  
 'Decision Tree',  
 'AdaBoost',  
 'LG',  
 'LinearSVC',  
 'XGBoost',  
 'MLP'  
 ]  
  
sampling\_methods = [catb,  
 rf,  
 lgb,  
 gbdt,  
 dt,  
 ad,  
 lg,  
 svm,  
 xgb,  
 mlp  
 ]  
  
colors = ['crimson',  
 'orange',  
 'yellow',  
 'mediumseagreen',  
 'steelblue',  
 'mediumpurple' ,  
 'black',  
 'silver',  
 'lightgreen',  
 'brown'  
 ]  
y\_predprobs = [catb\_train\_predprob,  
 rf\_train\_predprob,  
 lgb\_train\_predprob,  
 gbdt\_train\_predprob,  
 dt\_train\_predprob,  
 ad\_train\_predprob,  
 lg\_train\_predprob,  
 svm\_train\_predprob,  
 xgb\_train\_predprob,  
 mlp\_train\_predprob  
 ]  
  
train\_roc\_graph = multi\_models\_roc2(names, sampling\_methods, colors,y\_predprobs, X\_train, y\_train, save = True)  
train\_roc\_graph.savefig(r'D:\nested\_case\_control\MCI\_Train\_female.png')  
  
  
  
def multi\_models\_roc(names, sampling\_methods, colors,y\_predprobs ,X\_val, y\_val, save=True, dpin=100):  
 plt.figure(figsize=(20, 20), dpi=dpin)  
 for (name, method, colorname,y\_val\_predprob) in zip(names, sampling\_methods, colors,y\_predprobs):  
 y\_val\_preds = method.predict(X\_val)  
 fpr, tpr, thresholds = roc\_curve(y\_val, y\_val\_predprob, pos\_label=1)  
 plt.plot(fpr, tpr, lw=5, label='{} (AUC={:.3f})'.format(name, auc(fpr, tpr)),color = colorname)  
 plt.plot([0, 1], [0, 1], '--', lw=5, color = 'grey')  
 plt.axis('square')  
 plt.xlim([0, 1])  
 plt.ylim([0, 1])  
 plt.xlabel('False Positive Rate',fontsize=40)  
 plt.ylabel('True Positive Rate',fontsize=40)  
 plt.title('ROC Curve (MCI——Cross Validation Set)',fontsize=40)  
 plt.legend(loc='lower right',fontsize=40)  
  
 if save:  
 plt.savefig('multi\_models\_roc.png')  
 return plt  
  
names = ['CatBoost',  
 'RF',  
 'LightGBM',  
 'GBDT',  
 'Decision Tree',  
 'AdaBoost',  
 'LG',  
 'LinearSVC',  
 'XGBoost',  
 'MLP'  
 ]  
  
sampling\_methods = [catb,  
 rf,  
 lgb,  
 gbdt,  
 dt,  
 ad,  
 lg,  
 svm,  
 xgb,  
 mlp  
 ]  
  
colors = ['crimson',  
 'orange',  
 'yellow',  
 'mediumseagreen',  
 'steelblue',  
 'mediumpurple' ,  
 'black',  
 'silver',  
 'lightgreen',  
 'brown'  
 ]  
y\_predprobs= [catb\_val\_predprob,  
 rf\_val\_predprob,  
 lgb\_val\_predprob,  
 gbdt\_val\_predprob,  
 dt\_val\_predprob,  
 ad\_val\_predprob,  
 lg\_val\_predprob,  
 svm\_val\_predprob,  
 xgb\_val\_predprob,  
 mlp\_val\_predprob  
 ]  
test\_roc\_graph = multi\_models\_roc(names, sampling\_methods, colors, y\_predprobs,X\_val, y\_val, save = True)  
test\_roc\_graph.savefig(r'D:\nested\_case\_control\MCI\_Val\_female.png')  
  
from sklearn.linear\_model import LogisticRegression  
lg=EasyEnsembleClassifier(base\_estimator=LogisticRegression(C=1.0,solver= "lbfgs",max\_iter=100),  
 n\_estimators=20, n\_jobs=None,random\_state=123, replacement=False,  
 sampling\_strategy='auto', verbose=0, warm\_start=False)  
accuracy\_lg= cross\_val\_score(lg, X1\_train, y1\_train, cv=cv, scoring='accuracy')  
lg\_fit=lg.fit(X1\_train, y1\_train)  
lg\_predict = lg\_fit.predict(X\_test)  
lg\_val\_predprob = lg\_fit.predict\_proba(X\_test)[:,1]  
lg\_train\_predprob = lg\_fit.predict\_proba(X1\_train)[:,1]  
print("lg\_bacc",B\_acc(y\_test , lg\_predict))  
print("precision",precision\_score(y\_test, lg\_predict,average="weighted"))  
print("recall",recall\_score(y\_test, lg\_predict,average="weighted"))  
print("f1\_score",f1\_score(y\_test, lg\_predict,average="weighted"))  
  
def multi\_models\_roc(names, sampling\_methods, colors,y\_predprobs ,X\_tset, y\_test, save=True, dpin=100):  
 plt.figure(figsize=(20, 20), dpi=dpin)  
 for (name, method, colorname,y\_test\_predprob) in zip(names, sampling\_methods, colors,y\_predprobs):  
 y\_test\_preds = method.predict(X\_test)  
 fpr, tpr, thresholds = roc\_curve(y\_test, y\_test\_predprob, pos\_label=1)  
 plt.plot(fpr, tpr, lw=5, label='{} (AUC={:.3f})'.format(name, auc(fpr, tpr)),color = colorname)  
 plt.plot([0, 1], [0, 1], '--', lw=5, color = 'grey')  
 plt.axis('square')  
 plt.xlim([0, 1])  
 plt.ylim([0, 1])  
 plt.xlabel('False Positive Rate',fontsize=40)  
 plt.ylabel('True Positive Rate',fontsize=40)  
 plt.title('ROC Curve (MCI——Test Set)',fontsize=40)  
 plt.legend(loc='lower right',fontsize=40)  
  
 if save:  
 plt.savefig('multi\_models\_roc.png')  
 return plt  
names = ['LG']  
sampling\_methods = [lg]  
colors = [ 'steelblue']  
y\_predprobs= [ lg\_val\_predprob]  
  
test\_roc\_graph = multi\_models\_roc(names, sampling\_methods, colors, y\_predprobs,X\_test, y\_test, save = True)  
test\_roc\_graph.savefig(r'D:\nested\_case\_control\MCI\_test\_female.png')

### MCI-to-dementia — All participants

import time  
import pandas as pd  
import numpy as np  
import feather  
  
rawdata = pd.read\_sas(r'D:\nested\_case\_control\predication\MCI\_D\mcid\_prediction.sas7bdat')   
start = time.time()  
raw\_top5 = rawdata.head(500)  
print(rawdata.columns)  
null\_percentage = rawdata.isnull().sum()/len(rawdata)  
for col in null\_percentage.index:  
 if null\_percentage[col] > 0.1:  
 rawdata.drop(col, axis=1, inplace=True)  
  
null\_percentage1 = rawdata.isnull().sum(axis=1)/len(rawdata.columns)  
rawdata = rawdata[null\_percentage1 < 0.85]  
null\_percentage2 = rawdata.isnull().sum().sum()/rawdata.size  
num\_rows = rawdata[rawdata['MCI\_D'] == 1].shape[0]  
print(num\_rows)  
unk\_ratio = rawdata[(rawdata == 'unknow') | (rawdata == 'unknown')].count().sum() / rawdata.size  
rawdata.replace(['unknow', 'unknown'], np.nan, inplace=True)  
null\_percentage2 = rawdata.isnull().sum().sum()/rawdata.size  
cleaned\_data = rawdata  
null\_percentage45 = cleaned\_data.isnull().sum().sum()/cleaned\_data.size  
print(cleaned\_data.shape)  
feather.write\_dataframe(cleaned\_data,r'D:\nested\_case\_control\predication\MCI\_D\mcid\_prediction.feather')  
num = cleaned\_data.MCI\_D.value\_counts()  
print(num)  
print(cleaned\_data.columns)  
end = time.time()  
duration = end - start  
print('The code took %.2f seconds to run.' % duration)

from warnings import simplefilter  
import feather  
simplefilter(action='ignore', category=Warning)

import miceforest as mf  
import sys  
print(sys.getdefaultencoding())

num\_epochs = 20   
batch\_size = 64   
cleaned\_data = feather.read\_dataframe(r"D:\nested\_case\_control\predication\MCI\_D\mcid\_prediction.feather")  
print(cleaned\_data.head())  
  
kds2 = mf.ImputationKernel(  
 data=cleaned\_data,  
 datasets=4,

save\_models=1,

save\_all\_iterations=False,

random\_state=10 )  
  
optimal\_parameters, losses = kds2.tune\_parameters(  
 dataset=0,  
 optimization\_steps=5,  
 device='cpu' )  
kds2.mice(iterations=3,  
 variable\_parameters=optimal\_parameters,  
 n\_jobs=-1, n\_estimators=25,  
 min\_sum\_hessian\_in\_leaf=0.01) completed\_x\_train0 = kds2.complete\_data(dataset=0, inplace=False)  
print(completed\_x\_train0.isnull().sum(0))  
feather.write\_dataframe(completed\_x\_train0, r'D:\nested\_case\_control\predication\MCI\_D\mcid\_clean.feather')  
print(completed\_x\_train0.shape)  
num = completed\_x\_train0.MCI\_D.value\_counts()  
print(num)

import joblib  
import pandas as pd  
import numpy as np  
import feather  
from imblearn.ensemble import EasyEnsembleClassifier  
from pyarrow import fs  
from sklearn import model\_selection  
from sklearn.ensemble import RandomForestClassifier  
from sklearn.inspection import permutation\_importance  
from sklearn.metrics import roc\_auc\_score,roc\_curve,auc, confusion\_matrix, f1\_score, precision\_score, recall\_score  
from random import random  
from sklearn.model\_selection import RepeatedStratifiedKFold  
from collections import Counter  
from imblearn.over\_sampling import SMOTE,RandomOverSampler  
from imblearn.under\_sampling import RandomUnderSampler  
from sklearn.model\_selection import train\_test\_split, GridSearchCV, cross\_val\_score, KFold, LeavePOut  
import matplotlib.pyplot as plt  
from catboost import CatBoostClassifier  
from tensorflow import keras  
import pickle  
import os  
import sys  
import seaborn as sns  
  
  
print(sys.getdefaultencoding())  
import warnings  
warnings.filterwarnings('ignore')  
  
df = feather.read\_dataframe(r'D:\nested\_case\_control\predication\MCID\_new\mcid\_clean.feather')  
df=pd.get\_dummies(df,columns=['ISO\_c','sedentary\_c'])  
df.drop(['ISO\_c\_0.0','sedentary\_c\_0.0'],axis=1,inplace=True)  
print(df.columns)  
  
df.rename(columns={df.columns[0]: 'Education levels',  
 df.columns[1]: 'Income',  
 df.columns[2]: 'BMI',  
 df.columns[3]: 'Smoking status',  
 df.columns[4]: 'APOE e4 carrier status',  
 df.columns[5]: 'HbA1c',  
 df.columns[6]: 'Insulin use',  
 df.columns[7]: 'Social isolation',  
 df.columns[8]: 'Coronary heart disease',  
 df.columns[9]: 'Stroke',  
 df.columns[10]: 'Heart Failure',  
 df.columns[11]: 'Hypertension',  
 df.columns[12]: 'Diabetes',  
 df.columns[13]: 'Influenza and pneumonia',  
 df.columns[14]: 'Chronic obstructive pulmonary diseases',  
 df.columns[16]: 'Osteoarthritis Arthritis',  
 df.columns[17]: 'Sedentary',},  
 inplace=True)  
  
feather.write\_dataframe(df, r'D:\nested\_case\_control\predication\MCID\_new\mcid\_onehot.feather')  
print(df.head(10))  
print(df.shape)  
num = df.MCI\_D.value\_counts()  
print(num)  
print(df.columns)  
  
X, y = df.drop('MCI\_D',axis=1), df['MCI\_D']  
print(y)  
over= RandomOverSampler(sampling\_strategy={1:9154},random\_state=666)  
X\_over, y\_over = over.fit\_resample(X, y)  
counter = Counter(y\_over)  
print(counter)  
  
X1\_train, X\_test, y1\_train, y\_test = train\_test\_split(X\_over, y\_over, train\_size=0.8,random\_state=123)  
train\_val = pd.concat([X1\_train,y1\_train], axis=1)  
test=pd.concat([X\_test,y\_test], axis=1)  
X1, y1 = train\_val.drop('MCI\_D',axis=1), train\_val['MCI\_D']  
print("y\_train",Counter(y\_train))  
print("y\_test",Counter(y\_test))  
  
  
KF = KFold(n\_splits =10)  
for train\_index, val\_index in KF.split(X1):  
 print("Train", train\_index, "Val", val\_index)  
 X\_train, X\_val = X1.iloc[train\_index], X1.iloc[val\_index]  
 y\_train, y\_val = y1.iloc[train\_index], y1.iloc[val\_index]  
print("y\_train",Counter(y\_train))  
print("y\_val",Counter(y\_val))  
  
import tensorflow.\_api.v2.compat.v1 as tf  
B\_acc = tf.keras.metrics.BinaryAccuracy(  
 name='binary\_accuracy',  
 dtype=None,  
 threshold=0.25)  
cv = RepeatedStratifiedKFold(n\_splits=10, n\_repeats=3, random\_state=1)  
  
#1 catboost  
catb =CatBoostClassifier(subsample=1.0, rsm=0.5, min\_data\_in\_leaf=10, max\_bin=20, learning\_rate= 0.07742636826811278,  
 l2\_leaf\_reg=1.0, iterations=210, depth= 8)  
accuracy\_catb = cross\_val\_score(catb, X\_train, y\_train, scoring='accuracy', cv=cv)  
catb\_fit=catb.fit(X\_train,y\_train)  
catb\_predict = catb\_fit.predict(X\_val)  
catb\_val\_predprob = catb\_fit.predict\_proba(X\_val)[:,1]  
catb\_train\_predprob = catb\_fit.predict\_proba(X\_train)[:,1]  
print("catb\_bacc",B\_acc(y\_val, catb\_predict))  
print("precision",precision\_score(y\_val, catb\_predict,average="weighted"))  
print("recall",recall\_score(y\_val, catb\_predict,average="weighted"))  
print("f1\_score",f1\_score(y\_val, catb\_predict,average="weighted"))  
  
#2 rf  
from sklearn.ensemble import RandomForestClassifier  
rf = RandomForestClassifier()  
accuracy\_rf= cross\_val\_score(rf, X\_train, y\_train, cv=cv, scoring='accuracy')  
rf\_fit=rf.fit(X\_train,y\_train)  
rf\_predict = rf\_fit.predict(X\_val)  
rf\_val\_predprob = rf\_fit.predict\_proba(X\_val)[:,1]  
rf\_train\_predprob = rf\_fit.predict\_proba(X\_train)[:,1]  
print("rf\_Bacc",B\_acc(y\_val, rf\_predict))  
print("precision",precision\_score(y\_val, rf\_predict,average="weighted"))  
print("recall",recall\_score(y\_val, rf\_predict,average="weighted"))  
print("f1\_score",f1\_score(y\_val, rf\_predict,average="weighted"))  
joblib.dump(rf\_fit, '../web\_app/MCI\_Dementia/rf.pkl', compress=3)  
  
  
#3 LGB  
import lightgbm as lgb  
lgb = lgb.LGBMClassifier()  
accuracy\_lgb= cross\_val\_score(lgb, X\_train, y\_train, cv=cv, scoring='accuracy')  
lgb\_fit=lgb.fit(X\_train, y\_train)  
lgb\_predict = lgb\_fit.predict(X\_val)  
lgb\_val\_predprob = lgb\_fit.predict\_proba(X\_val)[:,1]  
lgb\_train\_predprob = lgb\_fit.predict\_proba(X\_train)[:,1]  
print("lg\_bacc",B\_acc(y\_val, lgb\_predict))  
print("precision",precision\_score(y\_val, lgb\_predict,average="weighted"))  
print("recall",recall\_score(y\_val, lgb\_predict,average="weighted"))  
print("f1\_score",f1\_score(y\_val, lgb\_predict,average="weighted"))  
  
  
#4 GBDT  
from sklearn.ensemble import GradientBoostingClassifier  
gbdt = GradientBoostingClassifier(tol=4.094915062380427e-06, subsample= 1, n\_estimators=100, min\_samples\_split= 6,min\_samples\_leaf=2,  
 max\_features='log2', max\_depth= 3, learning\_rate=0.3906939937054621)  
accuracy\_gbdt= cross\_val\_score(gbdt, X\_train, y\_train, cv=cv, scoring='accuracy')  
gbdt\_fit=gbdt.fit(X\_train, y\_train)  
gbdt\_predict = gbdt\_fit.predict(X\_val)  
gbdt\_val\_predprob = gbdt\_fit.predict\_proba(X\_val)[:,1]  
gbdt\_train\_predprob = gbdt\_fit.predict\_proba(X\_train)[:,1]  
print("gbdt\_bacc",B\_acc(y\_val, gbdt\_predict))  
print("precision",precision\_score(y\_val, gbdt\_predict,average="weighted"))  
print("recall",recall\_score(y\_val, gbdt\_predict,average="weighted"))  
print("f1\_score",f1\_score(y\_val, gbdt\_predict,average="weighted"))  
  
#5 dt  
from sklearn.tree import DecisionTreeClassifier  
dt = DecisionTreeClassifier(min\_samples\_split=18, min\_samples\_leaf=18, max\_features=None, max\_depth=9)  
cv = RepeatedStratifiedKFold(n\_splits=10, n\_repeats=3, random\_state=1)  
accuracy\_dt= cross\_val\_score(dt, X\_train, y\_train, cv=cv, scoring='accuracy')  
dt\_fit=dt.fit(X\_train, y\_train)  
dt\_predict = dt\_fit.predict(X\_val)  
dt\_val\_predprob = dt\_fit.predict\_proba(X\_val)[:,1]  
dt\_train\_predprob = dt\_fit.predict\_proba(X\_train)[:,1]  
print("dt\_bacc",B\_acc(y\_val, dt\_predict))  
print("precision",precision\_score(y\_val, dt\_predict,average="weighted"))  
print("recall",recall\_score(y\_val, dt\_predict,average="weighted"))  
print("f1\_score",f1\_score(y\_val, dt\_predict,average="weighted"))  
  
#6 ad  
from sklearn.ensemble import AdaBoostClassifier  
ad = AdaBoostClassifier(n\_estimators=100, learning\_rate=0.05963623316594643, algorithm='SAMME.R')  
cv = RepeatedStratifiedKFold(n\_splits=10, n\_repeats=3, random\_state=1)  
accuracy\_ad= cross\_val\_score(ad, X\_train, y\_train, cv=cv, scoring='accuracy')  
ad\_fit=ad.fit(X\_train, y\_train)  
ad\_predict = ad\_fit.predict(X\_val)  
ad\_val\_predprob = ad\_fit.predict\_proba(X\_val)[:,1]  
ad\_train\_predprob = ad\_fit.predict\_proba(X\_train)[:,1]  
print("ad\_bacc",B\_acc(y\_val, ad\_predict))  
print("precision",precision\_score(y\_val, ad\_predict,average="weighted"))  
print("recall",recall\_score(y\_val, ad\_predict,average="weighted"))  
print("f1\_score",f1\_score(y\_val, ad\_predict,average="weighted"))  
  
#7 lg  
from sklearn.linear\_model import LogisticRegression  
lg =LogisticRegression(warm\_start=False, tol=0.01, penalty='l2', max\_iter= 1000, class\_weight= 'balanced', C= 0.1)  
accuracy\_lg= cross\_val\_score(lg, X1\_train, y1\_train, cv=cv, scoring='accuracy')  
lg\_fit=lg.fit(X\_train, y\_train)  
lg\_predict = lg\_fit.predict(X\_val)  
lg\_val\_predprob = lg\_fit.predict\_proba(X\_val)[:,1]  
lg\_train\_predprob = lg\_fit.predict\_proba(X\_train)[:,1]  
print("lg\_bacc",B\_acc(y\_val , lg\_predict))  
print("precision",precision\_score(y\_val, lg\_predict,average="weighted"))  
print("recall",recall\_score(y\_val, lg\_predict,average="weighted"))  
print("f1\_score",f1\_score(y\_val, lg\_predict,average="weighted"))  
  
#8 svm  
from sklearn.svm import LinearSVC  
svm = LinearSVC(tol=0.001, C=0.1)  
accuracy\_svm= cross\_val\_score(svm, X\_train, y\_train, cv=cv, scoring='accuracy')  
svm\_fit=svm.fit(X\_train, y\_train)  
svm\_predict = svm\_fit.predict(X\_val)  
svm\_val\_predprob = svm\_fit.\_predict\_proba\_lr(X\_val)[:,1]  
svm\_train\_predprob = svm\_fit.\_predict\_proba\_lr(X\_train)[:,1]  
print("svm\_bacc",B\_acc(y\_val, svm\_predict))  
print("precision",precision\_score(y\_val, svm\_predict,average="weighted"))  
print("recall",recall\_score(y\_val, svm\_predict,average="weighted"))  
print("f1\_score",f1\_score(y\_val, svm\_predict,average="weighted"))  
  
#9 XGBoost  
import xgboost as xgb  
xgb = xgb.XGBClassifier()  
accuracy\_xgb= cross\_val\_score(xgb, X\_train, y\_train, cv=cv, scoring='accuracy')  
xgb\_fit=xgb.fit(X\_train, y\_train)  
xgb\_predict = xgb\_fit.predict(X\_val)  
xgb\_val\_predprob = xgb\_fit.predict\_proba(X\_val)[:,1]  
xgb\_train\_predprob = xgb\_fit.predict\_proba(X\_train)[:,1]  
print("xgb\_bacc",B\_acc(y\_val, xgb\_predict))  
print("precision",precision\_score(y\_val, xgb\_predict,average="weighted"))  
print("recall",recall\_score(y\_val, xgb\_predict,average="weighted"))  
print("f1\_score",f1\_score(y\_val, xgb\_predict,average="weighted"))  
  
#10 MLP  
from sklearn.neural\_network import MLPClassifier  
mlp =MLPClassifier()  
accuracy\_mlp= cross\_val\_score(mlp, X\_train, y\_train, cv=10, scoring='accuracy')  
mlp\_fit=mlp.fit(X\_train, y\_train)  
mlp\_predict = mlp\_fit.predict(X\_val)  
mlp\_val\_predprob = mlp\_fit.predict\_proba(X\_val)[:,1]  
mlp\_train\_predprob = mlp\_fit.predict\_proba(X\_train)[:,1]  
print("mlp\_bacc",B\_acc(y\_val, mlp\_predict))  
print("precision",precision\_score(y\_val, mlp\_predict,average="weighted"))  
print("recall",recall\_score(y\_val, mlp\_predict,average="weighted"))  
print("f1\_score",f1\_score(y\_val, mlp\_predict,average="weighted"))  
  
  
def multi\_models\_roc2(names, sampling\_methods, colors,y\_predprobs, X\_train, y\_train, save=True, dpin=100):  
 plt.figure(figsize=(20, 20), dpi=dpin)  
 for (name, method, colorname,y\_train\_predprob) in zip(names, sampling\_methods, colors,y\_predprobs):  
 y\_train\_preds = method.predict(X\_train)  
 fpr, tpr, thresholds = roc\_curve(y\_train, y\_train\_predprob, pos\_label=1)  
 plt.plot(fpr, tpr, lw=5, label='{} (AUC={:.3f})'.format(name, auc(fpr, tpr)),color = colorname)  
 plt.plot([0, 1], [0, 1], '--', lw=5, color = 'grey')  
 plt.axis('square')  
 plt.xlim([0, 1])  
 plt.ylim([0, 1])  
 plt.xlabel('False Positive Rate',fontsize=40)  
 plt.ylabel('True Positive Rate',fontsize=40)  
 plt.title('ROC Curve (MCI conversion to Dementia——Training Set)',fontsize=40)  
 plt.legend(loc='lower right',fontsize=40)  
  
 if save:  
 plt.savefig('multi\_models\_roc2.png')  
 return plt  
  
  
names = ['CatBoost',  
 'RF',  
 'LightGBM',  
 'GBDT',  
 'Decision Tree',  
 'AdaBoost',  
 'LG',  
 'LinearSVC',  
 'XGBoost',  
 'MLP'  
 ]  
sampling\_methods = [catb,  
 rf,  
 lgb,  
 gbdt,  
 dt,  
 ad,  
 lg,  
 svm,  
 xgb,  
 mlp  
 ]  
colors = ['crimson',  
 'orange',  
 'yellow',  
 'mediumseagreen',  
 'steelblue',  
 'mediumpurple' ,  
 'black',  
 'silver',  
 'lightgreen',  
 'brown'  
 ]  
y\_predprobs = [catb\_train\_predprob,  
 rf\_train\_predprob,  
 lgb\_train\_predprob,  
 gbdt\_train\_predprob,  
 dt\_train\_predprob,  
 ad\_train\_predprob,  
 lg\_train\_predprob,  
 svm\_train\_predprob,  
 xgb\_train\_predprob,  
 mlp\_train\_predprob  
 ]  
  
train\_roc\_graph = multi\_models\_roc2(names, sampling\_methods, colors,y\_predprobs, X\_train, y\_train, save = True)  
train\_roc\_graph.savefig(r'D:\nested\_case\_control\MCID\_Train.png')  
  
  
def multi\_models\_roc(names, sampling\_methods, colors,y\_predprobs ,X\_val,y\_val, save=True, dpin=100):  
 plt.figure(figsize=(20, 20), dpi=dpin)  
 for (name, method, colorname,y\_val\_predprob) in zip(names, sampling\_methods, colors,y\_predprobs):  
 y\_val\_preds = method.predict(X\_val)  
 fpr, tpr, thresholds = roc\_curve(y\_val, y\_val\_predprob, pos\_label=1)  
 plt.plot(fpr, tpr, lw=5, label='{} (AUC={:.3f})'.format(name, auc(fpr, tpr)),color = colorname)  
 plt.plot([0, 1], [0, 1], '--', lw=5, color = 'grey')  
 plt.axis('square')  
 plt.xlim([0, 1])  
 plt.ylim([0, 1])  
 plt.xlabel('False Positive Rate',fontsize=40)  
 plt.ylabel('True Positive Rate',fontsize=40)  
 plt.title('ROC Curve (MCI conversion to Dementia——Cross Validation Set)',fontsize=40)  
 plt.legend(loc='lower right',fontsize=40)  
   
 if save:  
 plt.savefig('multi\_models\_roc.png')  
 return plt  
  
names = ['CatBoost',  
 'RF',  
 'LightGBM',  
 'GBDT',  
 'Decision Tree',  
 'AdaBoost',  
 'LG',  
 'LinearSVC',  
 'XGBoost',  
 'MLP'  
 ]  
sampling\_methods = [catb,  
 rf,  
 lgb,  
 gbdt,  
 dt,  
 ad,  
 lg,  
 svm,  
 xgb,  
 mlp  
 ]  
colors = ['crimson',  
 'orange',  
 'yellow',  
 'mediumseagreen',  
 'steelblue',  
 'mediumpurple' ,  
 'black',  
 'silver',  
 'lightgreen',  
 'brown'  
 ]  
y\_predprobs= [catb\_val\_predprob,  
 rf\_val\_predprob,  
 lgb\_val\_predprob,  
 gbdt\_val\_predprob,  
 dt\_val\_predprob,  
 ad\_val\_predprob,  
 lg\_val\_predprob,  
 svm\_val\_predprob,  
 xgb\_val\_predprob,  
 mlp\_val\_predprob  
 ]  
  
  
test\_roc\_graph = multi\_models\_roc(names, sampling\_methods, colors, y\_predprobs,X\_val, y\_val, save = True)  
test\_roc\_graph.savefig(r'D:\nested\_case\_control\MCID\_Val.png')  
  
  
#2 rf  
from sklearn.ensemble import RandomForestClassifier  
rf = RandomForestClassifier()  
accuracy\_rf= cross\_val\_score(rf, X1\_train, y1\_train, cv=cv, scoring='accuracy')  
rf\_fit=rf.fit(X1\_train,y1\_train)   
rf\_predict = rf\_fit.predict(X\_test)   
rf\_val\_predprob = rf\_fit.predict\_proba(X\_test)[:,1]  
rf\_train\_predprob = rf\_fit.predict\_proba(X1\_train)[:,1]  
print("rf\_Bacc",B\_acc(y\_test, rf\_predict))  
print("precision",precision\_score(y\_test, rf\_predict,average="weighted"))  
print("recall",recall\_score(y\_test, rf\_predict,average="weighted"))  
print("f1\_score",f1\_score(y\_test, rf\_predict,average="weighted"))  
  
  
def multi\_models\_roc(names, sampling\_methods, colors,y\_predprobs ,X\_tset, y\_test, save=True, dpin=100):  
 plt.figure(figsize=(20, 20), dpi=dpin)  
 for (name, method, colorname,y\_test\_predprob) in zip(names, sampling\_methods, colors,y\_predprobs):  
 y\_test\_preds = method.predict(X\_test)  
 fpr, tpr, thresholds = roc\_curve(y\_test, y\_test\_predprob, pos\_label=1)  
 plt.plot(fpr, tpr, lw=5, label='{} (AUC={:.3f})'.format(name, auc(fpr, tpr)),color = colorname)  
 plt.plot([0, 1], [0, 1], '--', lw=5, color = 'grey')  
 plt.axis('square')  
 plt.xlim([0, 1])  
 plt.ylim([0, 1])  
 plt.xlabel('False Positive Rate',fontsize=40)  
 plt.ylabel('True Positive Rate',fontsize=40)  
 plt.title('ROC Curve (MCI conversion to Dementia——Test Set)',fontsize=40)  
 plt.legend(loc='lower right',fontsize=40)  
  
 if save:  
 plt.savefig('multi\_models\_roc.png')  
 return plt  
names = ['RF' ]  
  
sampling\_methods = [rf ]  
colors = [ 'steelblue']  
y\_predprobs= [ rf\_val\_predprob]  
  
test\_roc\_graph = multi\_models\_roc(names, sampling\_methods, colors, y\_predprobs,X\_test, y\_test, save = True)  
test\_roc\_graph.savefig(r'D:\nested\_case\_control\MCID\_test.png')  
  
#shap  
import shap  
shap.initjs()  
from sklearn.ensemble import RandomForestClassifier  
rf\_fit=rf.fit(X1\_train, y1\_train)  
X\_train\_summary = shap.kmeans(X1\_train, 10)  
explainer = shap.KernelExplainer(rf\_fit.predict,X\_train\_summary , gpu\_model='Ture')   
shap\_values= explainer.shap\_values(X\_test)  
shap.summary\_plot(shap\_values,X\_test,max\_display=10)#  
shap.summary\_plot(shap\_values,X\_test,plot\_type='bar',color='steelblue')  
  
#calibration  
from sklearn.calibration import CalibratedClassifierCV, CalibrationDisplay  
import matplotlib.pyplot as plt  
from matplotlib.gridspec import GridSpec  
clf\_list = [(rf, "RF")]  
fig = plt.figure(figsize=(10, 10))  
colors ='steelblue'  
ax\_calibration\_curve = fig.add\_subplot(111)  
calibration\_displays = {}  
for i, (clf, name) in enumerate(clf\_list):  
 clf.fit(X1\_train, y1\_train)  
 display = CalibrationDisplay.from\_estimator(  
 clf,  
 X\_test,  
 y\_test,  
 n\_bins=5,  
 name=name,  
 ax=ax\_calibration\_curve,  
 color=colors,  
 )  
 calibration\_displays[name] = display  
  
ax\_calibration\_curve.grid()  
ax\_calibration\_curve.set\_xlabel("Mean predicted probability (Positive class: 1:0)",fontsize=23)  
ax\_calibration\_curve.set\_ylabel("Fraction of positives (Positive class: 1:0)",fontsize=23)  
ax\_calibration\_curve.set\_title("Calibration plots (RF)",fontsize=23)  
ax\_calibration\_curve.legend(loc='lower right',fontsize=23)  
plt.tight\_layout()  
plt.show()

### MCI-to-dementia — Female population

import time  
import pandas as pd  
import numpy as np  
import feather  
  
rawdata = pd.read\_sas(r'D:\nested\_case\_control\predication\MCID\_female\mcid\_female\_prediction.sas7bdat')   
start = time.time()  
raw\_top5 = rawdata.head(500)  
print(rawdata.columns)  
null\_percentage = rawdata.isnull().sum()/len(rawdata)  
for col in null\_percentage.index:  
 if null\_percentage[col] > 0.1:  
 rawdata.drop(col, axis=1, inplace=True)  
  
null\_percentage1 = rawdata.isnull().sum(axis=1)/len(rawdata.columns)  
rawdata = rawdata[null\_percentage1 < 0.85]  
null\_percentage2 = rawdata.isnull().sum().sum()/rawdata.size  
num\_rows = rawdata[rawdata['MCI\_D'] == 1].shape[0]  
print(num\_rows)  
unk\_ratio = rawdata[(rawdata == 'unknow') | (rawdata == 'unknown')].count().sum() / rawdata.size  
rawdata.replace(['unknow', 'unknown'], np.nan, inplace=True)  
null\_percentage2 = rawdata.isnull().sum().sum()/rawdata.size  
cleaned\_data = rawdata  
null\_percentage45 = cleaned\_data.isnull().sum().sum()/cleaned\_data.size  
print(cleaned\_data.shape)  
feather.write\_dataframe(cleaned\_data,r'D:\nested\_case\_control\predication\MCID\_female\mcid\_female\_prediction.feather')  
num = cleaned\_data.MCI\_D.value\_counts()  
print(num)  
print(cleaned\_data.columns)  
end = time.time()  
duration = end - start  
print('The code took %.2f seconds to run.' % duration)

from warnings import simplefilter  
import feather  
simplefilter(action='ignore', category=Warning)

import miceforest as mf  
import sys  
print(sys.getdefaultencoding())

num\_epochs = 20   
batch\_size = 64   
cleaned\_data = feather.read\_dataframe(r"D:\nested\_case\_control\predication\ MCID\_female\mcid\_female\_prediction.feather")  
print(cleaned\_data.head())  
  
kds2 = mf.ImputationKernel(  
 data=cleaned\_data,  
 datasets=4,

save\_models=1,

save\_all\_iterations=False,

random\_state=10 )  
  
optimal\_parameters, losses = kds2.tune\_parameters(  
 dataset=0,  
 optimization\_steps=5,  
 device='cpu')  
  
kds2.mice(iterations=3,  
 variable\_parameters=optimal\_parameters,  
 n\_jobs=-1, n\_estimators=25,  
 min\_sum\_hessian\_in\_leaf=0.01) completed\_x\_train0 = kds2.complete\_data(dataset=0, inplace=False)  
  
print(completed\_x\_train0.isnull().sum(0))  
  
feather.write\_dataframe(completed\_x\_train0, r'D:\nested\_case\_control\predication\ MCID\_female\mcid\_female\_clean.feather')  
print(completed\_x\_train0.shape)  
num = completed\_x\_train0.MCI\_D.value\_counts()  
print(num)

df = feather.read\_dataframe(r'D:\nested\_case\_control\predication\MCID\_female\mcid\_female\_clean.feather')  
df=pd.get\_dummies(df,columns=['ISO\_c','sedentary\_c'])  
df.drop(['ISO\_c\_0.0','sedentary\_c\_0.0' ],axis=1,inplace=True)  
print(df.columns)  
  
df.rename(columns={df.columns[0]: 'Education levels',  
 df.columns[1]: 'Income',  
 df.columns[2]: 'BMI',  
 df.columns[3]: 'Smoking status',  
 df.columns[4]: 'APOE e4 carrier status',  
 df.columns[5]: 'HbA1c',  
 df.columns[6]: 'Insulin use',  
 df.columns[7]: 'CHD',  
 df.columns[8]: 'Stroke',  
 df.columns[9]: 'Heart Failure',  
 df.columns[10]: 'Hypertension',  
 df.columns[11]: 'Diabetes',  
 df.columns[12]: 'Influenza and pneumonia',  
 df.columns[13]: 'Chronic obstructive pulmonary diseases',  
 df.columns[14]: 'Osteoarthritis Arthritis',  
 df.columns[16]: 'Hormone replacement therapy',  
 df.columns[17]: 'Social isolation',  
 df.columns[18]: 'Sedentary',  
 },  
 inplace=True)  
feather.write\_dataframe(df, r'D:\nested\_case\_control\predication\MCID\_female\mcid\_female\_onehot.feather')  
print(df.head(10))  
print(df.shape)  
num = df.MCI\_D.value\_counts()  
print(num)  
print(df.columns)  
  
X, y = df.drop('MCI\_D',axis=1), df['MCI\_D']  
print(y)  
over= RandomOverSampler(sampling\_strategy={1:4104},random\_state=666)  
X\_over, y\_over = over.fit\_resample(X, y)  
counter = Counter(y)  
print(counter)  
X1\_train, X\_test, y1\_train, y\_test = train\_test\_split(X\_over, y\_over, train\_size=0.8,random\_state=42)  
  
train\_val = pd.concat([X1\_train,y1\_train], axis=1)  
X1, y1 = train\_val.drop('MCI\_D',axis=1), train\_val['MCI\_D']  
print("y\_train",Counter(y1\_train))  
print("y\_test",Counter(y\_test))  
  
KF = KFold(n\_splits =10)  
for train\_index, val\_index in KF.split(X1):  
 print("Train", train\_index, "Val", val\_index)  
 X\_train, X\_val = X1.iloc[train\_index], X1.iloc[val\_index]  
 y\_train, y\_val = y1.iloc[train\_index], y1.iloc[val\_index]  
print("y\_train",Counter(y\_train))  
print("X\_train",Counter(X\_train))  
  
import tensorflow.\_api.v2.compat.v1 as tf  
B\_acc = tf.keras.metrics.BinaryAccuracy(  
 name='binary\_accuracy',  
 dtype=None,  
 threshold=0.25)  
cv = RepeatedStratifiedKFold(n\_splits=10, n\_repeats=3, random\_state=1)  
  
  
#1 catboost  
catb =CatBoostClassifier(subsample=1.0, rsm=0.5, min\_data\_in\_leaf=10, max\_bin=20, learning\_rate= 0.07742636826811278,  
 l2\_leaf\_reg=1.0, iterations=210, depth= 8)  
accuracy\_catb = cross\_val\_score(catb, X\_train, y\_train, scoring='accuracy', cv=cv)  
catb\_fit=catb.fit(X\_train,y\_train)   
catb\_predict = catb\_fit.predict(X\_val)   
catb\_val\_predprob = catb\_fit.predict\_proba(X\_val)[:,1]  
catb\_train\_predprob = catb\_fit.predict\_proba(X\_train)[:,1]  
print("catb\_bacc",B\_acc(y\_val, catb\_predict))  
print("precision",precision\_score(y\_val, catb\_predict,average="weighted"))  
print("recall",recall\_score(y\_val, catb\_predict,average="weighted"))  
print("f1\_score",f1\_score(y\_val, catb\_predict,average="weighted"))  
  
  
#2 rf  
from sklearn.ensemble import RandomForestClassifier  
rf = RandomForestClassifier()  
accuracy\_rf= cross\_val\_score(rf, X\_train, y\_train, cv=cv, scoring='accuracy')  
rf\_fit=rf.fit(X\_train,y\_train)   
rf\_predict = rf\_fit.predict(X\_val)  
rf\_val\_predprob = rf\_fit.predict\_proba(X\_val)[:,1]  
rf\_train\_predprob = rf\_fit.predict\_proba(X\_train)[:,1]  
print("rf\_Bacc",B\_acc(y\_val, rf\_predict))  
print("precision",precision\_score(y\_val, rf\_predict,average="weighted"))  
print("recall",recall\_score(y\_val, rf\_predict,average="weighted"))  
print("f1\_score",f1\_score(y\_val, rf\_predict,average="weighted"))  
  
#3 LGB  
import lightgbm as lgb  
lgb = lgb.LGBMClassifier()  
accuracy\_lgb= cross\_val\_score(lgb, X\_train, y\_train, cv=cv, scoring='accuracy')  
lgb\_fit=lgb.fit(X\_train, y\_train)  
lgb\_predict = lgb\_fit.predict(X\_val)  
lgb\_val\_predprob = lgb\_fit.predict\_proba(X\_val)[:,1]  
lgb\_train\_predprob = lgb\_fit.predict\_proba(X\_train)[:,1]  
print("lgb\_bacc",B\_acc(y\_val, lgb\_predict))  
print("precision",precision\_score(y\_val, lgb\_predict,average="weighted"))  
print("recall",recall\_score(y\_val, lgb\_predict,average="weighted"))  
print("f1\_score",f1\_score(y\_val, lgb\_predict,average="weighted"))  
  
#4 GBDT  
from sklearn.ensemble import GradientBoostingClassifier  
gbdt = GradientBoostingClassifier(tol=4.094915062380427e-06, subsample= 1, n\_estimators=100, min\_samples\_split= 6,min\_samples\_leaf=2,  
 max\_features='log2', max\_depth= 3, learning\_rate=0.3906939937054621)  
accuracy\_gbdt= cross\_val\_score(gbdt, X\_train, y\_train, cv=cv, scoring='accuracy')  
gbdt\_fit=gbdt.fit(X\_train, y\_train)  
gbdt\_predict = gbdt\_fit.predict(X\_val)  
gbdt\_val\_predprob = gbdt\_fit.predict\_proba(X\_val)[:,1]  
gbdt\_train\_predprob = gbdt\_fit.predict\_proba(X\_train)[:,1]  
print("gbdt\_bacc",B\_acc(y\_val, gbdt\_predict))  
print("precision",precision\_score(y\_val, gbdt\_predict,average="weighted"))  
print("recall",recall\_score(y\_val, gbdt\_predict,average="weighted"))  
print("f1\_score",f1\_score(y\_val, gbdt\_predict,average="weighted"))  
  
#5 dt  
from sklearn.tree import DecisionTreeClassifier  
dt = DecisionTreeClassifier(min\_samples\_split=18, min\_samples\_leaf=18, max\_features=None, max\_depth=9)  
cv = RepeatedStratifiedKFold(n\_splits=10, n\_repeats=3, random\_state=1)  
accuracy\_dt= cross\_val\_score(dt, X\_train, y\_train, cv=cv, scoring='accuracy')  
dt\_fit=dt.fit(X\_train, y\_train)  
dt\_predict = dt\_fit.predict(X\_val)  
dt\_val\_predprob = dt\_fit.predict\_proba(X\_val)[:,1]  
dt\_train\_predprob = dt\_fit.predict\_proba(X\_train)[:,1]  
print("dt\_bacc",B\_acc(y\_val, dt\_predict))  
print("precision",precision\_score(y\_val, dt\_predict,average="weighted"))  
print("recall",recall\_score(y\_val, dt\_predict,average="weighted"))  
print("f1\_score",f1\_score(y\_val, dt\_predict,average="weighted"))  
  
#6 ad  
from sklearn.ensemble import AdaBoostClassifier  
ad = AdaBoostClassifier(n\_estimators=100, learning\_rate=0.05963623316594643, algorithm='SAMME.R')  
cv = RepeatedStratifiedKFold(n\_splits=10, n\_repeats=3, random\_state=1)  
accuracy\_ad= cross\_val\_score(ad, X\_train, y\_train, cv=cv, scoring='accuracy')  
ad\_fit=ad.fit(X\_train, y\_train)  
ad\_predict = ad\_fit.predict(X\_val)  
ad\_val\_predprob = ad\_fit.predict\_proba(X\_val)[:,1]  
ad\_train\_predprob = ad\_fit.predict\_proba(X\_train)[:,1]  
print("ad\_bacc",B\_acc(y\_val, ad\_predict))  
print("precision",precision\_score(y\_val, ad\_predict,average="weighted"))  
print("recall",recall\_score(y\_val, ad\_predict,average="weighted"))  
print("f1\_score",f1\_score(y\_val, ad\_predict,average="weighted"))  
  
#7 lg  
from sklearn.linear\_model import LogisticRegression  
lg =LogisticRegression(warm\_start=False, tol=0.01, penalty='l2', max\_iter= 1000, class\_weight= 'balanced', C= 0.1)  
accuracy\_lg= cross\_val\_score(lg, X1\_train, y1\_train, cv=cv, scoring='accuracy')  
lg\_fit=lg.fit(X\_train, y\_train)  
lg\_predict = lg\_fit.predict(X\_val)  
lg\_val\_predprob = lg\_fit.predict\_proba(X\_val)[:,1]  
lg\_train\_predprob = lg\_fit.predict\_proba(X\_train)[:,1]  
print("lg\_bacc",B\_acc(y\_val , lg\_predict))  
print("precision",precision\_score(y\_val, lg\_predict,average="weighted"))  
print("recall",recall\_score(y\_val, lg\_predict,average="weighted"))  
print("f1\_score",f1\_score(y\_val, lg\_predict,average="weighted"))  
  
#8 svm  
from sklearn.svm import LinearSVC  
svm = LinearSVC(tol=0.001, C=0.1)  
accuracy\_svm= cross\_val\_score(svm, X\_train, y\_train, cv=cv, scoring='accuracy')  
svm\_fit=svm.fit(X\_train, y\_train)  
svm\_predict = svm\_fit.predict(X\_val)  
svm\_val\_predprob = svm\_fit.\_predict\_proba\_lr(X\_val)[:,1]  
svm\_train\_predprob = svm\_fit.\_predict\_proba\_lr(X\_train)[:,1]  
print("svm\_bacc",B\_acc(y\_val, svm\_predict))  
print("precision",precision\_score(y\_val, svm\_predict,average="weighted"))  
print("recall",recall\_score(y\_val, svm\_predict,average="weighted"))  
print("f1\_score",f1\_score(y\_val, svm\_predict,average="weighted"))  
  
#9 XGBoost  
import xgboost as xgb  
xgb = xgb.XGBClassifier()  
accuracy\_xgb= cross\_val\_score(xgb, X\_train, y\_train, cv=cv, scoring='accuracy')  
xgb\_fit=xgb.fit(X\_train, y\_train)  
xgb\_predict = xgb\_fit.predict(X\_val)  
xgb\_val\_predprob = xgb\_fit.predict\_proba(X\_val)[:,1]  
xgb\_train\_predprob = xgb\_fit.predict\_proba(X\_train)[:,1]  
print("xgb\_bacc",B\_acc(y\_val, xgb\_predict))  
print("precision",precision\_score(y\_val, xgb\_predict,average="weighted"))  
print("recall",recall\_score(y\_val, xgb\_predict,average="weighted"))  
print("f1\_score",f1\_score(y\_val, xgb\_predict,average="weighted"))  
  
#10 MLP   
from sklearn.neural\_network import MLPClassifier  
mlp =MLPClassifier()  
accuracy\_mlp= cross\_val\_score(mlp, X\_train, y\_train, cv=10, scoring='accuracy')  
mlp\_fit=mlp.fit(X\_train, y\_train)  
mlp\_predict = mlp\_fit.predict(X\_val)  
mlp\_val\_predprob = mlp\_fit.predict\_proba(X\_val)[:,1]  
mlp\_train\_predprob = mlp\_fit.predict\_proba(X\_train)[:,1]  
print("mlp\_bacc",B\_acc(y\_val, mlp\_predict))  
print("precision",precision\_score(y\_val, mlp\_predict,average="weighted"))  
print("recall",recall\_score(y\_val, mlp\_predict,average="weighted"))  
print("f1\_score",f1\_score(y\_val, mlp\_predict,average="weighted"))  
  
  
def multi\_models\_roc2(names, sampling\_methods, colors,y\_predprobs, X\_train, y\_train, save=True, dpin=100):  
 plt.figure(figsize=(20, 20), dpi=dpin)  
 for (name, method, colorname,y\_train\_predprob) in zip(names, sampling\_methods, colors,y\_predprobs):  
 y\_train\_preds = method.predict(X\_train)  
 fpr, tpr, thresholds = roc\_curve(y\_train, y\_train\_predprob, pos\_label=1)  
 plt.plot(fpr, tpr, lw=5, label='{} (AUC={:.3f})'.format(name, auc(fpr, tpr)),color = colorname)  
 plt.plot([0, 1], [0, 1], '--', lw=5, color = 'grey')  
 plt.axis('square')  
 plt.xlim([0, 1])  
 plt.ylim([0, 1])  
 plt.xlabel('False Positive Rate',fontsize=40)  
 plt.ylabel('True Positive Rate',fontsize=40)  
 plt.title('ROC Curve (MCI conversion to Dementia——Training Set)',fontsize=40)  
 plt.legend(loc='lower right',fontsize=40)  
 if save:  
 plt.savefig('multi\_models\_roc2.png')  
 return plt  
names = ['CatBoost',  
 'RF',  
 'LightGBM',  
 'GBDT',  
 'Decision Tree',  
 'AdaBoost',  
 'LG',  
 'LinearSVC',  
 'XGBoost',  
 'MLP'  
 ]  
sampling\_methods = [catb,  
 rf,  
 lgb,  
 gbdt,  
 dt,  
 ad,  
 lg,  
 svm,  
 xgb,  
 mlp  
 ]  
colors = ['crimson',  
 'orange',  
 'yellow',  
 'mediumseagreen',  
 'steelblue',  
 'mediumpurple' ,  
 'black',  
 'silver',  
 'lightgreen',  
 'brown'  
 ]  
y\_predprobs = [catb\_train\_predprob,  
 rf\_train\_predprob,  
 lgb\_train\_predprob,  
 gbdt\_train\_predprob,  
 dt\_train\_predprob,  
 ad\_train\_predprob,  
 lg\_train\_predprob,  
 svm\_train\_predprob,  
 xgb\_train\_predprob,  
 mlp\_train\_predprob  
 ]  
train\_roc\_graph = multi\_models\_roc2(names, sampling\_methods, colors,y\_predprobs, X\_train, y\_train, save = True)  
train\_roc\_graph.savefig(r'D:\nested\_case\_control\MCID\_female\_Train.png')  
  
def multi\_models\_roc(names, sampling\_methods, colors,y\_predprobs ,X\_val,y\_val, save=True, dpin=100):  
 plt.figure(figsize=(20, 20), dpi=dpin)  
 for (name, method, colorname,y\_val\_predprob) in zip(names, sampling\_methods, colors,y\_predprobs):  
 y\_val\_preds = method.predict(X\_val)  
 fpr, tpr, thresholds = roc\_curve(y\_val, y\_val\_predprob, pos\_label=1)  
 plt.plot(fpr, tpr, lw=5, label='{} (AUC={:.3f})'.format(name, auc(fpr, tpr)),color = colorname)  
 plt.plot([0, 1], [0, 1], '--', lw=5, color = 'grey')  
 plt.axis('square')  
 plt.xlim([0, 1])  
 plt.ylim([0, 1])  
 plt.xlabel('False Positive Rate',fontsize=40)  
 plt.ylabel('True Positive Rate',fontsize=40)  
 plt.title('ROC Curve (MCI conversion to Dementia——Cross Validation Set)',fontsize=40)  
 plt.legend(loc='lower right',fontsize=40)  
 if save:  
 plt.savefig('multi\_models\_roc.png')  
 return plt  
names = ['CatBoost',  
 'RF',  
 'LightGBM',  
 'GBDT',  
 'Decision Tree',  
 'AdaBoost',  
 'LG',  
 'LinearSVC',  
 'XGBoost',  
 'MLP'  
 ]  
sampling\_methods = [catb,  
 rf,  
 lgb,  
 gbdt,  
 dt,  
 ad,  
 lg,  
 svm,  
 xgb,  
 mlp  
 ]  
colors = ['crimson',  
 'orange',  
 'yellow',  
 'mediumseagreen',  
 'steelblue',  
 'mediumpurple' ,  
 'black',  
 'silver',  
 'lightgreen',  
 'brown'  
 ]  
y\_predprobs= [catb\_val\_predprob,  
 rf\_val\_predprob,  
 lgb\_val\_predprob,  
 gbdt\_val\_predprob,  
 dt\_val\_predprob,  
 ad\_val\_predprob,  
 lg\_val\_predprob,  
 svm\_val\_predprob,  
 xgb\_val\_predprob,  
 mlp\_val\_predprob  
 ]  
test\_roc\_graph = multi\_models\_roc(names, sampling\_methods, colors, y\_predprobs,X\_val, y\_val, save = True)  
test\_roc\_graph.savefig(r'D:\nested\_case\_control\MCID\_female\_Val.png')  
  
from sklearn.ensemble import RandomForestClassifier  
rf = RandomForestClassifier()  
accuracy\_rf= cross\_val\_score(rf, X1\_train, y1\_train, cv=cv, scoring='accuracy')  
rf\_fit=rf.fit(X1\_train,y1\_train)   
rf\_predict = rf\_fit.predict(X\_test)  
rf\_val\_predprob = rf\_fit.predict\_proba(X\_test)[:,1]  
rf\_train\_predprob = rf\_fit.predict\_proba(X1\_train)[:,1]  
print("rf\_Bacc",B\_acc(y\_test, rf\_predict))  
print("precision",precision\_score(y\_test, rf\_predict,average="weighted"))  
print("recall",recall\_score(y\_test, rf\_predict,average="weighted"))  
print("f1\_score",f1\_score(y\_test, rf\_predict,average="weighted"))  
  
def multi\_models\_roc(names, sampling\_methods, colors,y\_predprobs ,X\_tset, y\_test, save=True, dpin=100):  
 plt.figure(figsize=(20, 20), dpi=dpin)  
 for (name, method, colorname,y\_test\_predprob) in zip(names, sampling\_methods, colors,y\_predprobs):  
 y\_test\_preds = method.predict(X\_test)  
 fpr, tpr, thresholds = roc\_curve(y\_test, y\_test\_predprob, pos\_label=1)  
 plt.plot(fpr, tpr, lw=5, label='{} (AUC={:.3f})'.format(name, auc(fpr, tpr)),color = colorname)  
 plt.plot([0, 1], [0, 1], '--', lw=5, color = 'grey')  
 plt.axis('square')  
 plt.xlim([0, 1])  
 plt.ylim([0, 1])  
 plt.xlabel('False Positive Rate',fontsize=40)  
 plt.ylabel('True Positive Rate',fontsize=40)  
 plt.title('ROC Curve (MCI conversion to Dementia——Test Set)',fontsize=40)  
 plt.legend(loc='lower right',fontsize=40)  
 if save:  
 plt.savefig('multi\_models\_roc.png')  
 return plt  
names = ['RF' ]  
sampling\_methods = [rf,]  
colors = [ 'steelblue']  
y\_predprobs= [ rf\_val\_predprob]  
test\_roc\_graph = multi\_models\_roc(names, sampling\_methods, colors, y\_predprobs,X\_test, y\_test, save = True)  
test\_roc\_graph.savefig(r'D:\nested\_case\_control\MCID\_female\_test.png')

### RandomizedSearchCV

from sklearn.model\_selection import RandomizedSearchCV  
from catboost import CatBoostClassifier  
catb\_params={'iterations': np.arange(10,100,10),  
 'depth': np.arange(2, 10, 1),  
 'learning\_rate': np.logspace(-10,0, 2),  
 'l2\_leaf\_reg': np.logspace(-10, 0, 2),  
 'subsample':(0.5, 1.0),  
 'rsm':(0.5, 1.0),  
 'max\_bin':np.arange(10, 500,100),  
 'min\_data\_in\_leaf':np.arange( 10, 500,100)  
 }  
catb\_cv=RandomizedSearchCV(estimator=CatBoostClassifier(),param\_distributions=catb\_params,cv=5)  
catb\_cv\_fit=catb\_cv.fit(X\_train,y\_train)  
catb\_best=catb\_cv\_fit.best\_params\_  
print('catb:', catb\_best)  
yprob\_train\_catb = catb\_cv\_fit.predict\_proba(X\_train)[:,1]  
print("catb\_train\_AUC",roc\_auc\_score(y\_true=y\_train, y\_score=yprob\_train\_catb))  
yprob\_val\_catb = catb\_cv\_fit.predict\_proba(X\_val)[:,1]  
print("catb\_val\_AUC",roc\_auc\_score(y\_true=y\_val, y\_score=yprob\_val\_catb))  
  
  
from sklearn.ensemble import RandomForestClassifier  
rf\_params={ 'n\_estimators': (10, 100,5),  
 'criterion': ['gini', 'entropy'],  
 'max\_depth': [None] + list(np.arange(1, 100, 10).astype(int)),  
 'min\_samples\_split': np.arange(2, 20, 2),  
 'min\_samples\_leaf': np.arange(2, 20, 2),  
 'max\_features': ['sqrt', 'log2', None]}  
rf\_cv=RandomizedSearchCV(estimator=RandomForestClassifier(random\_state=44),param\_distributions=rf\_params,cv=5)  
rf\_cv\_fit=rf\_cv.fit(X\_train,y\_train)  
rf\_best=rf\_cv\_fit.best\_params\_  
print('rf',rf\_best)  
yprob\_train\_rf = rf\_cv\_fit.predict\_proba(X\_train)[:,1]  
print("rf\_train\_AUC",roc\_auc\_score(y\_true=y\_train, y\_score=yprob\_train\_rf))  
yprob\_val\_rf = rf\_cv\_fit.predict\_proba(X\_val)[:,1]  
print("rf\_val\_AUC",roc\_auc\_score(y\_true=y\_val, y\_score=yprob\_val\_rf))  
  
import lightgbm as lgb  
lgb\_params={ 'num\_leaves': (10,500,100),  
 'learning\_rate': np.logspace(-5, 0, 20),  
 'n\_estimators': (10,500,100),  
 'max\_depth': np.arange(2, 10, 2),  
 'min\_split\_gain': np.logspace(-5, 0, 20),  
 'subsample': (0.5,1),  
 'colsample\_bytree':(0.5,1),  
 'reg\_alpha': np.logspace(-5, 0, 20),  
 'reg\_lambda': np.logspace(-5, 0, 20)}  
lgb\_cv=RandomizedSearchCV(estimator = lgb.LGBMClassifier(random\_state=123),param\_distributions=lgb\_params,cv=5)  
lgb\_cv\_fit=lgb\_cv.fit(X\_train,y\_train)  
lgb\_best=lgb\_cv\_fit.best\_params\_  
print('lgb:', lgb\_best)  
yprob\_train\_lgb = lgb\_cv\_fit.predict\_proba(X\_train)[:,1]  
print("lgb\_train\_AUC",roc\_auc\_score(y\_true=y\_train, y\_score=yprob\_train\_lgb))  
yprob\_val\_lgb = lgb\_cv\_fit.predict\_proba(X\_val)[:,1]  
print("lgb\_val\_AUC",roc\_auc\_score(y\_true=y\_val, y\_score=yprob\_val\_lgb))  
  
from sklearn.ensemble import GradientBoostingClassifier  
gbdt\_params={ 'learning\_rate': np.logspace(-5, 0, 10),  
 'n\_estimators': (10,500,100),  
 'max\_depth': np.arange(2, 10, 2),  
 'min\_samples\_split': np.arange(2, 10, 2),  
 'min\_samples\_leaf': np.arange(2, 10, 2),  
 'max\_features': ['sqrt', 'log2', None],  
 'subsample': [0.5, 0.7, 1.0],  
 'tol': np.logspace(-6, -1, 50)}  
gbdt\_cv=RandomizedSearchCV(estimator = GradientBoostingClassifier(random\_state=44),param\_distributions=gbdt\_params,cv=5)  
gbdt\_cv\_fit=gbdt\_cv.fit(X\_train,y\_train)  
gbdt\_best=gbdt\_cv\_fit.best\_params\_  
print('gbdt:', gbdt\_best)  
yprob\_train\_gbdt = gbdt\_cv\_fit.predict\_proba(X\_train)[:,1]  
print("gbdt\_train\_AUC",roc\_auc\_score(y\_true=y\_train, y\_score=yprob\_train\_gbdt))  
yprob\_val\_gbdt = gbdt\_cv\_fit.predict\_proba(X\_val)[:,1]  
print("gbdt\_val\_AUC",roc\_auc\_score(y\_true=y\_val, y\_score=yprob\_test\_gbdt))  
  
from sklearn.tree import DecisionTreeClassifier  
dt\_params={ 'max\_depth': np.arange(2, 500, 2),  
 'min\_samples\_split': np.arange(2, 500, 1),  
 'min\_samples\_leaf': np.arange(2, 500, 1),  
 'max\_features': ['sqrt', 'log2', None],  
 }  
dt\_cv=RandomizedSearchCV(estimator =DecisionTreeClassifier(random\_state=44),param\_distributions=dt\_params,cv=5)  
dt\_cv\_fit=dt\_cv.fit(X\_train,y\_train)  
dt\_best=dt\_cv\_fit.best\_params\_  
print('dt:', dt\_best)  
yprob\_train\_dt = dt\_cv\_fit.predict\_proba(X\_train)[:,1]  
print("dt训练集AUC",roc\_auc\_score(y\_true=y\_train, y\_score=yprob\_train\_dt))  
yprob\_val\_dt = dt\_cv\_fit.predict\_proba(X\_val)[:,1]  
print("dt\_val\_AUC",roc\_auc\_score(y\_true=y\_val, y\_score=yprob\_val\_dt))  
  
  
from sklearn.ensemble import AdaBoostClassifier  
from sklearn.tree import DecisionTreeClassifier  
ad\_params={ 'n\_estimators': (10,1000,100),  
 'learning\_rate': np.logspace(-5,0, 2),  
 'algorithm': ['SAMME', 'SAMME.R'],  
 'estimator': [DecisionTreeClassifier(max\_depth=1), DecisionTreeClassifier(max\_depth=3)]}  
ad\_cv=RandomizedSearchCV(estimator =AdaBoostClassifier(random\_state=44),param\_distributions=ad\_params,cv=5)  
ad\_cv\_fit=ad\_cv.fit(X\_train,y\_train)  
ad\_best=ad\_cv\_fit.best\_params\_  
print('ad:', ad\_best)  
yprob\_train\_ad = ad\_cv\_fit.predict\_proba(X\_train)[:,1]  
print("ad\_train\_AUC",roc\_auc\_score(y\_true=y\_train, y\_score=yprob\_train\_ad))  
yprob\_val\_ad = ad\_cv\_fit.predict\_proba(X\_val)[:,1]  
print("ad\_val\_AUC",roc\_auc\_score(y\_true=y\_val, y\_score=yprob\_val\_ad))  
  
from sklearn.linear\_model import LogisticRegression  
lg\_params={ 'penalty': ['l2', 'none'],  
 'C': np.logspace(-3, 10, 1),  
 'class\_weight': [None, 'balanced'],  
 'max\_iter': [1000, 5000, 1000],  
 'warm\_start': [True, False],  
 'tol': [0.01, 0.001, 0.0001]}  
lg\_cv=RandomizedSearchCV(estimator=LogisticRegression(random\_state=44),param\_distributions=lg\_params,cv=5)  
lg\_cv\_fit=lg\_cv.fit(X\_train,y\_train)  
lg\_best=lg\_cv\_fit.best\_params\_  
print('lg:', lg\_best)  
yprob\_train\_lg = lg\_cv\_fit.predict\_proba(X\_train)[:,1]  
print("lg\_train\_AUC",roc\_auc\_score(y\_true=y\_train, y\_score=yprob\_train\_lg))  
yprob\_val\_lg = lg\_cv\_fit.predict\_proba(X\_val)[:,1]  
print("lg\_val\_AUC",roc\_auc\_score(y\_true=y\_val, y\_score=yprob\_val\_lg))  
  
from sklearn.svm import LinearSVC  
svm\_params={ "C": np.logspace(-3, 3, 7),  
 'tol': [0.01, 0.001, 0.0001],  
 }  
svm\_cv=RandomizedSearchCV(estimator=LinearSVC(random\_state=44),param\_distributions=svm\_params,cv=5)  
svm\_cv\_fit=svm\_cv.fit(X\_train,y\_train)  
svm\_best=svm\_cv\_fit.best\_params\_  
print('svm:', svm\_best)  
yprob\_train\_svm = svm\_cv\_fit.\_predict\_proba\_lr(X\_train)[:,1]  
print("svm\_train\_AUC",roc\_auc\_score(y\_true=y\_train, y\_score=yprob\_train\_svm))  
yprob\_val\_svm = svm\_cv\_fit.\_predict\_proba\_svm(X\_val)[:,1]  
print("svm\_val\_AUC",roc\_auc\_score(y\_true=y\_val, y\_score=yprob\_val\_svm))  
  
import xgboost as xgb  
xgb\_params={ 'max\_depth': (2,10,1),  
 'min\_child\_weight': (1, 10,1),  
 'gamma':(0,1),  
 "subsample": (0.5,1),  
 "colsample\_bytree": (0.5,1),  
 'n\_estimators': (10, 100, 10),  
 'booster': ( ['gbtree', 'gblinear']),  
 'eta': np.logspace(0.5, 1.0),  
 'alpha': np.logspace(0.5, 1.0),  
 'learning\_rate': (0.01, 0.015, 0.05, 0.1),  
 'grow\_policy': ( ["depthwise", "lossguide"])}  
xgb\_cv=RandomizedSearchCV(estimator= xgb.XGBClassifier(random\_state=44),param\_distributions=xgb\_params,cv=5)  
xgb\_cv\_fit=xgb\_cv.fit(X\_train,y\_train)  
xgb\_best=xgb\_cv\_fit.best\_params\_  
print('xgb:', xgb\_best)  
yprob\_train\_xgb = xgb\_cv\_fit.predict\_proba(X\_train)[:,1]  
print("xgb\_train\_AUC",roc\_auc\_score(y\_true=y\_train, y\_score=yprob\_train\_xgb))  
yprob\_val\_xgb = xgb\_cv\_fit.predict\_proba(X\_val)[:,1]  
print("xgb\_val\_AUC",roc\_auc\_score(y\_true=y\_val, y\_score=yprob\_val\_xgb))  
  
from sklearn.neural\_network import MLPClassifier  
mlp\_params={ 'hidden\_layer\_sizes': [(50, 50, 50), (50, 100, 50), (100,100,100)],  
 'activation': ['identity', 'logistic','tanh', 'relu'],  
 'solver': ['sgd', 'adam','lbfgs'],  
 'learning\_rate': (['constant','invscaling','adaptive']),  
 'alpha': np.logspace(-5, 0, 10),  
 'max\_iter':(10,500,100)}  
mlp\_cv=RandomizedSearchCV(estimator= MLPClassifier(random\_state=44),param\_distributions=mlp\_params,cv=5)  
mlp\_cv\_fit=mlp\_cv.fit(X\_train,y\_train)  
mlp\_best=mlp\_cv\_fit.best\_params\_  
print('mlp:', mlp\_best)  
yprob\_train\_mlp = mlp\_cv\_fit.predict\_proba(X\_train)[:,1]  
print("mlp\_train\_AUC",roc\_auc\_score(y\_true=y\_train, y\_score=yprob\_train\_mlp))  
yprob\_val\_mlp = mlp\_cv\_fit.predict\_proba(X\_val)[:,1]  
print("mlp\_val\_AUC",roc\_auc\_score(y\_true=y\_val, y\_score=yprob\_val\_mlp))

# Web app

### MCI

import numpy as np  
import stcli as stcli  
import streamlit  
import streamlit as st  
import pandas as pd  
import joblib  
import sys  
  
st.markdown('<center><font size=15 color=steelblue>Risk of Mild Cognitive Impairment</font></center>', unsafe\_allow\_html=True)  
  
# Dropdown input  
Age = st.selectbox("1.Age", ("<=50 years", "51-59 years", ">=60 years"))  
Sleep = st.selectbox("2.How many hours of sleep do you get in a 24-hour period, including any naps?", ("<6 hours/day", "6-8 hours/day", ">=9 hours/day"))  
APOE = st.selectbox("3.How many APOE e4 alleles do you carry?", ("None", "One", "Two"))  
Frailty = st.selectbox("4.Physical frailty (Such as walking less than 30 minutes a day, having low handgrip strength, feeling tired often over the past two weeks, experiencing weight loss compared to one year ago, or having a slow walking pace of less than 3 miles per hour.)", ("No", "Yes"))  
HP = st.selectbox("5.Hypertension", ("No", "Yes"))  
DM = st.selectbox("6.Diabetes",("No", "Yes"))  
IP = st.selectbox("7.Influenza and pneumonia", ("No", "Yes"))  
OA = st.selectbox("8.Osteoarthritis Arthritis",("No", "Yes"))  
Stroke = st.selectbox("9.Stroke", ("No", "Yes"))  
Dy = st.selectbox("10.Dyslipidemia", ("No", "Yes"))  
HF = st.selectbox("11.Heart Failure", ("No", "Yes"))  
COPD = st.selectbox("12.Chronic obstructive pulmonary diseases", ("No", "Yes"))  
GORD = st.selectbox("13.Gastro-esophageal reflux disease", ("No", "Yes"))  
DS = st.selectbox("14.Digestive system cancer", ("No", "Yes"))  
RPS = st.selectbox("15.Respiratory system cancer", ("No", "Yes"))  
LHS = st.selectbox("16.Lymphohematopoietic system cancer", ("No", "Yes"))  
  
if st.button("Submit"):  
 clf = joblib.load(r'D:\nested\_case\_control\predication\web\_app\MCI\catb.pkl')  
 int\_features= pd.DataFrame([[APOE,Age,Sleep,Stroke,HF,HP,DM,Dy,IP,  
 COPD,GORD,OA,LHS,DS,RPS, Frailty]],  
 columns=['APOE e4 carrier status', 'Age at baseline', 'Sleep duration', 'Stroke',  
 'Heart Failure','Hypertension', 'Diabetes','Dyslipidemia', 'Influenza and pneumonia',  
 'Chronic obstructive pulmonary diseases','Gastro-esophageal reflux disease',  
 'Osteoarthritis Arthritis','Lymphohematopoietic system cancer','Digestive system cancer',  
 'Respiratory system cancer','Frailty'])  
 final\_features = int\_features.replace(["<=50 years", "51-59 years", ">=60 years","<6 hours/day", "6-8 hours/day", ">=9 hours/day",  
 "None", "One", "Two","Yes", "No"], [0,1,2,0,1,2,0,1,2,1,0])  
 prediction = clf.predict\_proba(final\_features)[:,1]  
 output=np.around(prediction, decimals=2)  
 if output<0.5:  
 st.markdown(f'<font size=5>Risk of Mild Cognitive Impairment: :green[${output}$].</font>', unsafe\_allow\_html=True)  
 else:  
 st.markdown(f'<font size=5>Risk of Mild Cognitive Impairment: :red[${output}$].</font>', unsafe\_allow\_html=True)  
st.markdown('<font size=3 color=gray>The calculated risk on this page is based on the probability derived from a 8.0-year follow-up.</font>', unsafe\_allow\_html=True)

### MCI-to-dementia

import numpy as np  
import stcli as stcli  
import streamlit  
import streamlit as st  
import pandas as pd  
import joblib  
import sys  
  
st.markdown('<center><font size=15 color=steelblue>Risk of Mild Cognitive Impairment conversion to dementia</font></center>', unsafe\_allow\_html=True)  
Edu = st.selectbox("1.Years of education", ("<= 10 years", "11–12 years", ">12 years"))  
Income = st.selectbox("2.What is the average total income before tax received by your household?", ("Less than 18,000 pound",  
 "18,000 to 30,999 pound",  
 "31,000 to 51,999 pound",  
 "Greater than 52,000 pound"))  
BMI = st.selectbox("3.Body mass index", ("Underweight (<18.5 Kg/m2)", "Normal (18.5-24.9 Kg/m2)",  
 "Overweight (25.0-29.9 Kg/m2)","Obese (>=30.0 Kg/m2)"))  
Smoking = st.selectbox("4.Smoking status", ("Never", "Previous","Current"))  
Sedentary = st.selectbox("5.How long do you usually sit in a 24-hour period?", ("<4 hours/day", ">=4 hours/day"))  
APOE = st.selectbox("6.How many APOE e4 alleles do you carry?",("None", "One","Two"))  
ISO = st.selectbox("7.Social isolation status (Such as living alone, not participating in social activities at least once a week, and receiving visits from family and friends less than once a month.)", ("No", "Yes"))  
Insulin = st.selectbox("8.Do you regularly use insulin?",("No", "Yes"))  
HbA1c = st.selectbox("9.Glycated haemoglobin (HbA1c) level", ("<53 mmol/L (<7%)", ">=53 mmol/L (>=7%)"))  
HP = st.selectbox("10.Hypertension", ("No", "Yes"))  
DM = st.selectbox("11.Diabetes", ("No", "Yes"))  
CHD = st.selectbox("12.Coronary heart disease", ("No", "Yes"))  
Stroke = st.selectbox("13.Stroke", ("No", "Yes"))  
HF = st.selectbox("14.Heart Failure", ("No", "Yes"))  
IP = st.selectbox("15.Influenza and pneumonia", ("No", "Yes"))  
COPD = st.selectbox("16.Chronic obstructive pulmonary diseases", ("No", "Yes"))  
OA = st.selectbox("17.Osteoarthritis Arthritis", ("No", "Yes"))  
  
if st.button("Submit"):  
 clf = joblib.load(r'D:\nested\_case\_control\predication\web\_app\MCI\_Dementia\rf.pkl')  
 int\_features= pd.DataFrame([[APOE,Edu,Income,Smoking,Sedentary,Stroke,HF,HP,DM,IP,ISO,Insulin,HbA1c,  
 COPD,BMI,OA,CHD]],  
 columns=['APOE e4 carrier status', 'Education levels', 'Income', 'Smoking status','Sedentary',  
 'Stroke', 'Heart Failure','Hypertension', 'Diabetes', 'Influenza and pneumonia',  
 'Social isolation','Insulin use','HbA1c', 'Chronic obstructive pulmonary diseases','BMI',  
 'Osteoarthritis Arthritis','Coronary heart disease'])  
 final\_features = int\_features.replace(["<= 10 years", "11–12 years", ">12 years",  
 "Less than 18,000 pound","18,000 to 30,999 pound","31,000 to 51,999 pound","Greater than 52,000 pound",  
 "Underweight (<18.5 Kg/m2)", "Normal (18.5-24.9 Kg/m2)",  
 "Overweight (25.0-29.9 Kg/m2)","Obese (>=30.0 Kg/m2)",  
 "Never", "Previous","Current",  
 "<4 hours/day", ">=4 hours/day",  
 "None", "One","Two",  
 "<53 mmol/L (<7%)", ">=53 mmol/L (>=7%)",  
 "Yes", "No"],  
 [1,2,3,  
 1,2,3,4,  
 1,2,3,4,  
 0,1,2,  
 0,1,  
 0,1,2,  
 0,1,  
 1,0])  
  
 prediction = clf.predict\_proba(final\_features)[:,1]  
 output=np.around(prediction, decimals=2)  
 if output<0.5:  
 st.markdown(f'<font size=5>Risk of Mild Cognitive Impairment conversion to dementia: :green[${output}$].</font>', unsafe\_allow\_html=True)  
 else:  
 st.markdown(f'<font size=5>Risk of Mild Cognitive Impairment conversion to dementia: :red[${output}$].</font>', unsafe\_allow\_html=True)  
st.markdown('<font size=3 color=gray>The calculated risk on this page is based on the probability derived from a 4.0-year follow-up.</font>', unsafe\_allow\_html=True)