Supplementary Materials for the manuscript titled *Tri-Variate Stochastic Weather Model for Predicting Maize Yield*

Patrick Chidzalo [1], Phillip O. Ngare[2], Joseph K. Mung'atu [1]

[1] Pan African University Institute of Basic Sciences, Technology and Innovation, Juja, Kenya

[2] School of Mathematics, University of Nairobi, Nairobi, Kenya

Abstract

This document presents supplementary materials for the manuscript *Tri-Variate Stochastic Weather Model for Predicting Maize Yield*. It consists of algorithms written in MATLAB and python. Data sets can be obtained through writing to the authors, the Ministry of Agriculture; and the Department of Meteorological Services and Climate Change in Malawi. Weather and crop yield data from somewhere else can also be used with the algorithms.

1 MATLAB2021a Code for Checking Stochastic Models

The quantiles are calculated using the original algorithm from first principles rather than using in built commands.

1.1 Rainfall Amount

```
function exploring_rain_algmically
clc;
close all;
rng(80,'philox')
A=xlsread('weatherdatacleaned');
x=A(:,4);
m1=mean(x);
v1=var(x);
beta_h1=v1/m1
alpha_h1=m1/beta_h1
z1=gamrnd(alpha_h1,beta_h1, 500,1);
m=length(x);
p=((1:m)-0.5)/m;
xns=sort(x);
z1ns=quantile(z1,p);
figure(1)

plot(z1ns,xns,'b+')
cof=polyfit(z1ns,xns,1);
Lnn=polyval(cof,z1ns);
```

```
hold on
plot(z1ns,Lnn,'r-.', 'LineWidth',2)
xlabel('Theoretical quantiles')
ylabel('Sample quantiles')
legend('','Fit line', 'Location', 'northwest')
grid on
ylim([0,60])
hold off
figure(2)
h=histogram(x,'normalization','pdf');
h.NumBins=22;
m=150;
t=min(x):(max(x)-min(x))/m:max(x);

f=gampdf(t, alpha_h1,beta_h1);
hold on
plot(t,f,'r-', 'LineWidth',2)
ylabel('Frequency density')
xlabel('Rainfall amount (mm/day)')
grid on
legend('Raw Data','\Gamma (\alpha, \beta)', 'Location', 'northeast')
```

1.2 Daily Temperature

```
function exploring_Temp_algmically
clc;
%close all;

rng(250,'philox')


A=xlsread('weatherdatacleaned');
A(A(:,4)<=2.54,:)=[];
T_min=A(:,2); %%%Minimum Temperature
T_max=A(:,3); %%%Maximum Temperature
T= 0.5.*(T_min+T_max);

x=T;
m1=mean(x);
v1=sum((x-m1).^2)/(length(x)-1);

sig1=sqrt(v1)/1.16;
z1=normrnd(m1,sig1,500,1);
m=length(x);
p=((1:m)-0.5)/m;
xns=sort(x);
z1ns=quantile(z1,p);
figure(1)

plot(z1ns,xns,'b+')
```

```
cof=polyfit(z1ns,xns,1);
Lnn=polyval(cof,z1ns);
hold on
plot(z1ns,Lnn,'r-.', 'LineWidth',2)
xlabel('Theoretical quantiles')
ylabel('Sample quantiles')
legend('','Fit line', 'Location', 'northwest')
grid on
ylim([18,26])
hold off

figure(2)
h=histogram(x,'normalization','pdf');
h.NumBins=18;
m=200;
t=min(x):(max(x)-min(x))/m:max(x);

f=normpdf(t,m1,sig1);
hold on
plot(t,f,'r-', 'LineWidth',2)
ylabel('Frequency density')
xlabel('Temperature (^o C)')
grid on
legend('Raw Data','N(\mu, \sigma ^2 )', 'Location', 'northeast')
xlim([17,26])
```

1.3 Reference Evapotranspiration

```
function exploring_ET0_algmically
clc;
close all;
rng(200,'philox')
A=xlsread('weatherdatacleaned');
x=A(:,7);
m1=mean(x);
v1=var(x);
beta_h1=v1/m1
alpha_h1=m1/beta_h1
z1=gamrnd(alpha_h1,beta_h1, 500,1);
m=length(x);
p=((1:m)-0.5)/m;
xns=sort(x);
z1ns=quantile(z1,p);
figure()
subplot(1,2,2)
plot(z1ns,xns,'b+')
cof=polyfit(z1ns,xns,1);
Lnn=polyval(cof,z1ns);
hold on
plot(z1ns,Lnn,'r-.', 'LineWidth',2)
```

```
xlabel('Theoretical quantiles')
ylabel('Sample quantiles')
legend('','Fit line', 'Location', 'northwest')
hold off
subplot(1,2,1)
h=histogram(x,'normalization','pdf');
h.NumBins=8;
t=min(x):(max(x)-min(x))/m:max(x);

f=gampdf(t, alpha_h1,beta_h1);
hold on
plot(t,f,'r-', 'LineWidth',2)
ylabel('Frequency density')
xlabel('ET_0 (mm/day)')
legend('Raw Data','\Gamma (\alpha, \beta)', 'Location', 'northeast')
```

2 MATLAB2021a Codes for Copulas

The MALTLAB codes have been constructed from the mathematical algorithms presented in Zhang and Singh (2019), Joe and Kurowicka (2011), and Joe (1996).

2.1 Calculation of Parameters for Bi-variate Copulas

2.1.1 Rainfall Amount and Reference Evapotranspiration

```
function fitting_FrCop_Evapo_Rain
clc; close all;
clear;
tic
%% Data
Dta=xlsread('weatherdatacleaned');
Dta(Dta(:,4)<=2.54,:)=[];
y=Dta(:,4);
x=Dta(:,7);

%%%% STEP ONE: Find the Kendall's tau
n=length(x);
w=1:n;
A=nchoosek(w,2);
m=size(A); C=[]; D=[]; prd=[];
for i=1:m(1)
tt1=A(i,1);
tt2=A(i,2);
D(i)=x(tt1)-x(tt2);
C(i)=y(tt1)-y(tt2);
prd(i)=sign(D(i)*C(i));
end
cc1=2*(1/n)*(1/(n-1));
```

```
Ktau=cc1.*sum(prd)

%%%% STEP TWO: Find Theta

syms k t
g=t/(exp(t)-1);
D1F=(-1./k)*int(g,t, 0,-k);
Ftali=1-(4/k).*(D1F-1);

theta=-vpasolve(Ftali==Ktau)

%%%% STEP THREE: Find the copula
f=@(u,v,ta) ((-1/ta).*log(1+((exp(-ta.*u)-1).*(exp(-ta.*v)-1))./(exp(-ta)-1)));

%%%% STEP FOUR: Find the u's and the v's by using appropriate
%%%% distribution function
u=gamcdf(x,13.6095,0.2507);
v=gamcdf(y,1.3083,8.2941);
table(u,v);

%%%% STEP FOUR: You can now find the copula
C=f(u,v,theta);
C=eval(C);
table(u,v,C);

%%%% You can also sketch the copula
us=linspace(min(u),max(u),15);

vs=linspace(min(v),max(v),15);


theta=eval(theta);
[X,Y]=meshgrid(us,vs);
Z=(-1/theta).*log(1+((exp(-theta.*X)-1).*(exp(-theta.*Y)-1))./(exp(-theta)-1));
figure()
subplot(1,2,1)
surf(X,Y,Z)
xlabel('u')
ylabel('v')
subplot(1,2,2)
contour(X,Y,Z,'ShowText','on','LineWidth',1.5)
xlabel('u')
ylabel('v')
toc
```

2.1.2 Rainfall Amount and Daily Temperature

```
function fitting_FrCop_Temp_Rain
clc; close all;
clear;
```

```
%% Data
Dta=xlsread('weatherdatacleaned');
Dta(Dta(:,4)<=2.54,:)=[];
y=Dta(:,4);
x=0.5.*Dta(:,2)+0.5.*Dta(:,3);
format long
%%%% STEP ONE: Find the Kendall's tau
n=length(x);
w=1:n;
A=nchoosek(w,2);
m=size(A); C=[]; D=[]; prd=[];
for i=1:m(1)
tt1=A(i,1);
tt2=A(i,2);
D(i)=x(tt1)-x(tt2);
C(i)=y(tt1)-y(tt2);
prd(i)=sign(D(i)*C(i));
end
cc1=2*(1/n)*(1/(n-1));
Ktau=cc1.*sum(prd)

%%%% STEP TWO: Find Theta

syms k t
g=t/(exp(t)-1);
D1F=(-1./k)*int(g,t, 0,-k);
Ftali=1-(4/k).*(D1F-1);

theta=-vpasolve(Ftali==Ktau)

%%%% STEP THREE: Find the copula
f=@(u,v,ta) ((-1/ta).*log(1+((exp(-ta.*u)-1).*(exp(-ta.*v)-1))./(exp(-ta)-1)));

%%%% STEP FOUR: Find the u's and the v's by using appropriate
%%%% distribution function
u=normcdf(x,20.5442,3.0166);
v=gamcdf(y,1.3083,8.2941);
table(u,v);

%%%% STEP FOUR: You can now find the copula
C=f(u,v,theta);
C=eval(C);
table(u,v,C);
%%%% You can also sketch the copula
us=linspace(min(u),max(u),10);

vs=linspace(min(v),max(v),10);


theta=eval(theta);
```

```
[X,Y]=meshgrid(us,vs);
Z=(-1/theta).*log(1+((exp(-theta.*X)-1).*(exp(-theta.*Y)-1))./(exp(-theta)-1));
figure()
subplot(1,2,1)
surf(X,Y,Z)
xlabel('u')
ylabel('v')
subplot(1,2,2)
contour(X,Y,Z,'ShowText','on','LineWidth',1.5)
xlabel('u')
ylabel('v')
```

2.1.3 Daily Temperature and Reference Evapotranspiration

```
function fitting_FrCop_Temp_Evap
clc; close all;
clear;
tic
%% Data
Dta=xlsread('weatherdatacleaned');
Dta(Dta(:,4)<=2.54,:)=[];
y=Dta(:,7);
x=0.5.*Dta(:,2)+0.5.*Dta(:,3);

%%%% STEP ONE: Find the Kendall's tau
n=length(x);
w=1:n;
A=nchoosek(w,2);
m=size(A); C=[]; D=[]; prd=[];
for i=1:m(1)
tt1=A(i,1);
tt2=A(i,2);
D(i)=x(tt1)-x(tt2);
C(i)=y(tt1)-y(tt2);
prd(i)=sign(D(i)*C(i));
end
cc1=2*(1/n)*(1/(n-1));
Ktau=cc1.*sum(prd)

%%%% STEP TWO: Find Theta

syms k t
g=t/(exp(t)-1);
D1F=(-1./k)*int(g,t, 0,-k);
Ftali=1-(4/k).*(D1F-1);

theta=-vpasolve(Ftali==Ktau)

%%%% STEP THREE: Find the copula
f=@(u,v,ta) ((-1/ta).*log(1+((exp(-ta.*u)-1).*(exp(-ta.*v)-1))./(exp(-ta)-1)));
```

```
%%%% STEP FOUR: Find the u's and the v's by using appropriate
%%%% distribution function
u=normcdf(x,20.5442,3.0166);
v=gamcdf(y,13.6095,0.2507);
table(u,v);

%%%% STEP FOUR: You can now find the copula
C=f(u,v,theta);
C=eval(C);
table(u,v,C);
%%%% You can also sketch the copula
us=linspace(min(u),max(u),10);
vs= linspace(min(v),max(v),10);
theta=eval(theta);
[X,Y]=meshgrid(us,vs);
Z=(-1/theta).*log(1+((exp(-theta.*X)-1).*(exp(-theta.*Y)-1))./(exp(-theta)-1));
figure()
subplot(1,2,1)
surf(X,Y,Z)
xlabel('w')
ylabel('v')
zlabel('C_{\theta}(v,w)' )
subplot(1,2,2)
contour(X,Y,Z,'ShowText','on','LineWidth',2)
xlabel('w')
ylabel('v')
toc
```

2.2 KK-Plots

2.2.1 Reference Evapotranspiration and Daily Temperature

```
function K_plot_Evap_Temp
clc;
close all;
AB=xlsread('weatherdatacleaned');
AB(AB(:,4)<=2.54,:)=[];
x=0.5.*AB(:,2)+0.5.*AB(:,3);
y=AB(:,7);
x=x(1:39:3992);
y=y(1:39:3992);
n=length(x)
w=1:n;
A=nchoosek(w,2);
m=size(A);
C=[];
B=[];
prd=[];
```

```
B=A(:,1);
C=A(:,2);
Kss=[];
for i=1:m(1)
t1=A(i,1);
t2=A(i,2);
for j=1:m(1)
t3=B(j);
t4=C(j);
prd(j)=(x(t3)<=x(t1))&&(y(t3)<=y(t1))&&(i~=j);
end
Kss(i)=mean(prd);
end

Kss=Kss';
H=sort(Kss);
Hs=unique(H);
n2=length(Hs);
n=n2;
format long
F= @(x,k,n) (-x.*log(x).*(x-x.*log(x)).^(k-1).*(1-x+x.*log(x)).^(n-k));
W=[];
for k=1:n
num1=n*factorial(n-1);
den1=factorial(n-k).*factorial(k-1);
CC1=num1/den1;
W(k)= CC1.*integral(@(x) F(x,k,n),0,1);
end
W=W';
PD=W-W.*log(W);



plot(W,Hs,'k+',W, PD,'r -' ,[W;1],[W;1],'k--','LineWidth',2)
ylabel('H_{(s)}')
xlabel('W_{1:n}')
title('C_{\theta}(ET_0, Temp)')
legend('Empirical Data', 'Perfect Positive Dependence', 'Independence (h=w)',...
'Location','SouthEast');
```

2.2.2 Rainfall Amount and Reference Evapotranspiration

```
function K_plot_Rain_Evap
clc;
close all;
AB=xlsread('weatherdatacleaned');
AB(AB(:,4)<=2.54,:)=[];
x=AB(:,4);
y=AB(:,7);
x=x(1:39:3992);
```

```
y=y(1:39:3992);
n=length(x);
w=1:n;
A=nchoosek(w,2);
m=size(A);
C=[];
B=[];
prd=[];
B=A(:,1);
C=A(:,2);
Kss=[];
for i=1:m(1)
t1=A(i,1);
t2=A(i,2);
for j=1:m(1)
t3=B(j);
t4=C(j);
prd(j)=(x(t3)<=x(t1))&&(y(t3)<=y(t1))&&(i~=j);
end
Kss(i)=mean(prd);
end

Kss=Kss';
H=sort(Kss);
Hs=unique(H);
n2=length(Hs);
n=n2;
format long
F= @(x,k,n) (-x.*log(x).*(x-x.*log(x)).^(k-1).*(1-x+x.*log(x)).^(n-k));
W=[];
for k=1:n
num1=n*factorial(n-1);
den1=factorial(n-k).*factorial(k-1);
CC1=num1/den1;
W(k)= CC1.*integral(@(x) F(x,k,n),0,1);
end
W=W';
PD=W-W.*log(W);



plot(W,Hs,'k+',W, PD,'r -' ,[W;1],[W;1],'k--','LineWidth',2)
ylabel('H_{(s)}')
xlabel('W_{1:n}')
title('C_{\theta} (ET_0, Rain)')
legend('Empirical Data', 'Perfect Positive Dependence (h=w)', 'Independence',...
'Location','southeast');
```

2.2.3 Daily Temperature and Rainfall Amount

```
function K_plot_Rain_Temp
clc;
close all;
AB=xlsread('weatherdatacleaned');
AB(AB(:,4)<=2.54,:)=[];
x=0.5.*AB(:,2)+0.5.*AB(:,3);
y=AB(:,4);
x=x(1:35:3992);
y=y(1:35:3992);
n=length(x);
w=1:n;
A=nchoosek(w,2);
m=size(A);
C=[];
B=[];
prd=[];
B=A(:,1);
C=A(:,2);
Kss=[];
for i=1:m(1)
t1=A(i,1);
t2=A(i,2);
for j=1:m(1)
t3=B(j);
t4=C(j);
prd(j)=(x(t3)<=x(t1))&&(y(t3)<=y(t1))&&(i~=j);
end
Kss(i)=mean(prd);
end

Kss=Kss';
H=sort(Kss);
Hs=unique(H);
n2=length(Hs);
n=n2;
format long
F= @(x,k,n) (-x.*log(x).*(x-x.*log(x)).^(k-1).*(1-x+x.*log(x)).^(n-k));
W=[];
for k=1:n
num1=n*factorial(n-1);
den1=factorial(n-k).*factorial(k-1);
CC1=num1/den1;
W(k)= CC1.*integral(@(x) F(x,k,n),0,1);
end
W=W';
PD=W-W.*log(W);
```

```
plot(W,Hs,'k+',W, PD,'r -' ,[W;1],[W;1],'k--','LineWidth',2)
ylabel('H_{(s)}')
xlabel('W_{1:n}')
title('C_{\theta}(Rain, Temp)')
legend('Empirical Data', 'Perfect Positive Dependence', 'Independence (h=w)',...
'Location','SouthEast');
```

2.3 Cross Product Ratio's and Plackett Copula Determination

```
function Trivariate_measure_of_dependence_product_Ratios
clear; close all; clc;
tic
%%% STEP ONE: Call the random variables and clean them
dta=xlsread('weatherdatacleaned');
dta(dta(:,4)<=2.54,:)=[];
R=dta(:,4); ET=dta(:,7); T=0.5.*dta(:,2)+0.5.*dta(:,3);

%%% STEP TWO: Evaluating u's
u=gamcdf(R,1.3083,8.2941);
v=gamcdf(ET,13.6095,0.2507);
w=normcdf(T,20.5442,3.0166);

%%% STEP THREE: Finding 2-copulas
ta=[-2.0176338567;-1.2185606113; 3.6538800162];
f=@(u1,v1,ta1) ((-1/ta1).*log(1+((exp(-ta1.*u1)-1).*(exp(-ta1.*v1)-1))./(exp(-ta1)-1)));
C_R_ET=f(u,v,ta(1));
C_R_T=f(u,w,ta(2));
C_ET_T=f(v,w,ta(3));

%%% STEP FOUR: Measures of dependence
num1=C_R_ET.*(1-u-v+C_R_ET);
num2=C_R_T.*(1-u-w+C_R_T);
num3=C_ET_T.*(1-v-w+C_ET_T);
den1=(u-C_R_ET).*(v-C_R_ET);
den2=(u-C_R_T).*(w-C_R_T);
den3=(v-C_ET_T).*(w-C_ET_T);

psI1=num1./den1;
cfs1=mean(psI1);
N=length(psI1);
psI1=psI1(1:39:N);
N=length(psI1);
t=linspace(0,1,N);
Z_test1=LinearModel.fit(t,psI1);
Lss1=cfs1.*t.^(0);
figure()
subplot(3,1,1)
plot(t,psI1,'bo',t,Lss1,'k', 'LineWidth',2)
ylabel('\psi _{R, ET}','Rotation', 90)
```

```
xlabel('t')
ylim([0 1]);

psI2=num2./den2;
cfs2=mean(psI2);
N=length(psI2);
psI2=psI2(1:39:N);
N=length(psI2);
t=linspace(0,1,N);
Z_test2=LinearModel.fit(t,psI2);
Lss2=cfs2.*t.^(0);
subplot(3,1,2)
plot(t,psI2,'bo',t,Lss2,'k', 'LineWidth',2)
ylabel('\psi _{R, T}','Rotation', 90)
xlabel('t')
ylim([0 1]);

psI3=num3./den3;
cfs3=mean(psI3);
N=length(psI3);
psI3=psI3(1:39:N);
N=length(psI3);
t=linspace(0,1,N);
Z_test=LinearModel.fit(t,psI3);
Lss3=cfs3.*t.^(0);
subplot(3,1,3)
plot(t,psI3,'bo',t,Lss3,'k', 'LineWidth',2)
ylabel('\psi _{ET_0, T}','Rotation', 90)
xlabel('t')
ylim([1 20]);

%%% STEP FIVE: Estimating Non-Parametric estimates product ratio
X1=[]; X2=[]; X3=[]; X4=[]; Y1=[]; Y2=[]; Y3=[];
Z1=[]; Z2=[]; Z3=[];
NX1=length(R); mx1=median(R); mx2=median(ET); mx3=median(T);
for i=1:NX1
%Rainfall VS ET
X1(i)=(R(i)<mx1)&&(ET(i)<mx2);
X2(i)=(R(i)>mx1)&&(ET(i)>mx2);
X3(i)=(R(i)<mx1)&&(ET(i)>mx2);
X4(i)=(R(i)>mx1)&&(ET(i)<mx2);
%Rainfall VS Temperature
Y1(i)=(R(i)<mx1)&&(T(i)<mx3);
Y2(i)=(R(i)>mx1)&&(T(i)>mx3);
Y3(i)=(R(i)<mx1)&&(T(i)>mx3);
Y4(i)=(R(i)>mx1)&&(T(i)<mx3);
%Rainfall VS Temperature
Z1(i)=(ET(i)<mx2)&&(T(i)<mx3);
Z2(i)=(ET(i)>mx2)&&(T(i)>mx3);
Z3(i)=(ET(i)<mx2)&&(T(i)>mx3);
```

```
Z4(i)=(ET(i)>mx2)&&(T(i)<mx3);
end
RETn00=sum(X1);
RETn11=sum(X2);
RETn01=sum(X3);
RETn10=sum(X4);
cfsnp1=(RETn00*RETn11)/(RETn01*RETn10);


RTn00=sum(Y1);
RTn11=sum(Y2);
RTn01=sum(Y3);
RTn10=sum(Y4);
cfsnp2=(RTn00*RTn11)/(RTn01*RTn10);


ETTn00=sum(Z1);
ETTn11=sum(Z2);
ETTn01=sum(Z3);
ETTn10=sum(Z4);
cfsnp3=(ETTn00*ETTn11)/(ETTn01*ETTn10);

vRTET=categorical({' psi(R,ET)';' psi(R,T)';'psi (T,ET)'});
vRTENP=[cfsnp1;cfsnp2;cfsnp3];
vRTEP=[cfs1;cfs2;cfs3];

table(vRTET,vRTENP,vRTEP,'VariableNames'...
,{'Cross product Ratio', 'Non-Parametric', 'Parametric' })


%%% STEP SIX: Estimating Non-Parametric estimates product ratio
X11=[]; X12=[]; X13=[]; X14=[]; Y11=[]; Y12=[]; Y13=[];
NX11=length(R); mx11=median(R); mx12=median(ET); mx13=median(T);
for i=1:NX11
%%% Numerator determination
%%%(0,0,0)
X11(i)=(R(i)<mx11)&&(ET(i)<mx12)&&(T(i)<mx13);
%%%(0,1,1)
X12(i)=(R(i)<mx11)&&(ET(i)>mx12)&&(T(i)>mx13);
%%%(1,0,1)
X13(i)=(R(i)>mx11)&&(ET(i)<mx12)&&(T(i)>mx13);
%%%(1,1,0)
X14(i)=(R(i)>mx11)&&(ET(i)>mx12)&&(T(i)<mx13);

%%% Numerator determination
%%%(1,1,1)
Y11(i)=(R(i)>mx11)&&(ET(i)>mx12)&&(T(i)>mx13);
%%%(1,0,0)
Y12(i)=(R(i)>mx11)&&(ET(i)<mx12)&&(T(i)<mx13);
%%%(0,1,0)
```

```
Y13(i)=(R(i)<mx11)&&(ET(i)>mx12)&&(T(i)<mx13);
%%%(0,0,1)
Y14(i)=(R(i)<mx11)&&(ET(i)<mx12)&&(T(i)>mx13);

end
RETTn000=sum(X11);
RETTn011=sum(X12);
RETTn101=sum(X13);
RETTn110=sum(X14);


RETTn111=sum(Y11);
RETTn100=sum(Y12);
RETTn010=sum(Y13);
RETTn001=sum(Y14);

psi_RETT=(RETTn000*RETTn011*RETTn101*RETTn110)...
/(RETTn111*RETTn100*RETTn010*RETTn001);


%%% STEP SIXb: Estimating semi-Parametric estimates product ratio
X11=[]; X12=[]; X13=[]; X14=[]; Y11=[]; Y12=[]; Y13=[];
NX11=length(u);
for i=1:NX11
%%% Numerator determination
%%%(0,0,0)
X11(i)=(u(i)<0.5)&&(v(i)<0.5)&&(w(i)<0.5);
%%%(0,1,1)
X12(i)=(u(i)<0.5)&&(v(i)>0.5)&&(w(i)>0.5);
%%%(1,0,1)
X13(i)=(u(i)>0.5)&&(v(i)<0.5)&&(w(i)>0.5);
%%%(1,1,0)
X14(i)=(u(i)>0.5)&&(v(i)>0.5)&&(w(i)<0.5);

%%% Numerator determination
%%%(1,1,1)
Y11(i)=(u(i)>0.5)&&(v(i)>0.5)&&(w(i)>0.5);
%%%(1,0,0)
Y12(i)=(u(i)>0.5)&&(v(i)<0.5)&&(w(i)<0.5);
%%%(0,1,0)
Y13(i)=(u(i)<0.5)&&(v(i)>0.5)&&(w(i)<0.5);
%%%(0,0,1)
Y14(i)=(u(i)<0.5)&&(v(i)<0.5)&&(w(i)>0.5);

end
RETTn000=sum(X11);
RETTn011=sum(X12);
RETTn101=sum(X13);
RETTn110=sum(X14);
```

```
RETTn111=sum(Y11);
RETTn100=sum(Y12);
RETTn010=sum(Y13);
RETTn001=sum(Y14);

psi_RETT2=(RETTn000*RETTn011*RETTn101*RETTn110)...
/(RETTn111*RETTn100*RETTn010*RETTn001);

vRTET=categorical({'Non-Parametric';'Semi-Parametric'});
vRTENP=[psi_RETT;psi_RETT2];


table(vRTET,vRTENP,'VariableNames',{'Method of Calculation', 'psi(R,ET,T)'  })

%%% STEP8: Finding 3-copula of Plackett Family

A1=C_ET_T;        %%% from the fact that a1=c(v,w)
A2=C_R_T;         %%% from the fact that a2=c(u,w)
A3=C_R_ET;        %%% from the fact that a3=c(u,v)
A4=1-u-v-w+A1+A2;
B1=-w+A2+A1;
B2=-v+A3+A1;
B3=-u+A2+A3;

%%%Purely Parametric  and semi-parametric
PlackT=zeros(NX1,8);
for j=1:NX1
syms z
PfT1= psi_RETT.*(A1(j)-z).*(A2(j)-z).*(A2(j)-z).*(A4(j)-z)...
-z.*(z-B1(j))*(z-B2(j)).*(z-B3(j));
PfT2= psi_RETT2.*(A1(j)-z).*(A2(j)-z).*(A2(j)-z).*(A4(j)-z)...
-z.*(z-B1(j))*(z-B2(j)).*(z-B3(j));
ZP=vpasolve(PfT1==0);
ZP2=vpasolve(PfT2==0);
PlackT(j,1)=abs(real(ZP(1)));
PlackT(j,2)=abs(real(ZP(2)));
PlackT(j,3)=abs(real(ZP(3)));
PlackT(j,4)=abs(real(ZP(4)));
PlackT(j,5)=abs(real(ZP2(1)));
PlackT(j,6)=abs(real(ZP2(2)));
PlackT(j,7)=abs(real(ZP2(3)));
PlackT(j,8)=abs(real(ZP2(4)));
end
PlackT;

BBB=[T R ET PlackT(:,3) PlackT(:,7) ];
xlswrite('pat1',BBB);
close all;
figure()
```

```
subplot(1,2,1)
plot(u,PlackT(:,3),'k.')
xlabel('u')
ylabel('C (R, ET_0,T)')
title('\psi_{(R, ET_0,T)} =1.1424')
subplot(1,2,2)
plot(u,PlackT(:,7),'b.')
xlabel('u')
title('\psi_{(R, ET_0,T)} = 1.3914')
toc
```

2.4 MATLAB2021a Code for Frank Asymmetric 3-Copula

```
function copdensity_By_Nesting_Method
clc; clear; close all;
tic %%
syms u v w
a=-2.0176338567; b=3.653800162;

Anum=(1-exp(-b.*u)).*(1-exp(-b.*v));
Aden=1-exp(-b);
A=Anum./Aden;
B1=1-A;
C1=1-exp(-a*w);
C2=C1*B1;
Cden=1-exp(-a);
Cff=C2./Cden;
Cin1=1-Cff;
C=(-1/a).*log(Cin1);



Bu=diff(C,w);
Buv=diff(Bu,u);
Buvw=diff(Buv,v);
bf=simplify(Buvw);


Dta=xlsread('weatherdatacleaned');
Dta(Dta(:,4)<=2,:)=[];
z=0.5*Dta(:,2)+0.5*Dta(:,3);
mu=20.5442; sg=3.01666;n=length(z);
t=1:n; r=Dta(:,4);

alpha1=1.3083; beta1=8.3941;
y=Dta(:,8);
alpha2=13.6085; beta2=0.2507;

U=gamcdf(r, alpha1,beta1);
V=gamcdf(y, alpha2,beta2);
```

```
W=normcdf(z,mu,sg);
fr=gampdf(r, alpha1,beta1);
fy=gampdf(y, alpha2,beta2);
fz=normpdf(z,mu,sg);
Cxxx=[];
for j=1:n
Cx=subs(bf,u,U(j));
Cxx=subs(Cx,v,V(j));
Cxxx(j)=eval(subs(Cxx,w,W(j)))*fr(j)*fy(j)*fz(j);
Cxxx(Cxxx<0)=0.0000000001;
end
Cxxx=Cxxx';
AIC=6-2*sum(log(Cxxx))
BIC=3*log(n)-2*sum(log(Cxxx))
Cxxx=Cxxx.*0.025./max(Cxxx);
AIC2=6-2*sum(log(Cxxx))
BIC2=3*log(n)-2* sum(log(Cxxx))
figure(1)
plot(r,Cxxx, 'b o')
xlabel('Rainfall in mm')
ylabel('f(r,x,y)')
figure(2)
plot(y,Cxxx, 'k o')
xlabel('Reference Evapotranspiration')
ylabel('f(r,x,y)')
figure(3)
plot(z,Cxxx, 'r o')
xlabel('Temperature in ^o C')
ylabel('f(r,x,y)')

toc %%
```

2.5 Tri-variate Pdf for Vine Copula

```
function ConditioningMethodTrivariateDensity
clc; close all;
tic
%%%%%DATA
Dta=xlsread('weatherdatacleaned');
Dta(Dta(:,4)<=2,:)=[];
z=0.5*Dta(:,2)+0.5*Dta(:,3); mu=20.5442; sg=3.01666;n=length(z);
t=1:n; r=Dta(:,4); alpha1=1.3083; beta1=8.3941;
y=Dta(:,8); alpha2=13.6085; beta2=0.2507; U=gamcdf(r, alpha1,beta1);
V=gamcdf(y, alpha2,beta2); W=normcdf(z,mu,sg); fr=gampdf(r, alpha1,beta1);
fy=gampdf(y, alpha2,beta2); fz=normpdf(z,mu,sg);

%%%%% Arguments of Conditioning Trivariate Density
theta1=-2.01763385668883331966612147309 61;
theta2=-1.21856061130989519454384162342 15;
theta3=3.65388001616074726214641121140 6;
```

```
theta4=0.1996;
fd=@(u,v,ta) ((ta.*(exp(ta)-1)*exp(ta*(1+u+v)))...
./(exp(ta.*(u+v))-exp(ta.*(1+u))+exp(ta.*(1))).^2);
f=@(u,v,ta) ((-1/ta).*log(1+((exp(-ta.*u)-1).*(exp(-ta.*v)-1))...
./(exp(-ta)-1)));
fwv=@(u,v,ta) ( (2.*u.*v+(u-3.*u.*v+v).*exp(-ta)+(1-u).*(1-v).*exp(-2.*ta))...
./(1+(1-u).*(1-v).*(exp(-ta)-1)).^3);
CUV=f(U,V,theta1);   %%F(X,Y)
CWU=f(W,U,theta2);   %%F(Z,X)
s1=CWU./U;           %%F(Z|X)
s2=CUV./U;           %%F(Y|X)
figure(1)            %%simply to check consistence
plot(1:n,s1)         %%simply to check consistence
figure(2)            %%simply to check consistence
plot(1:n,s2)         %%simply to check consistence
cUV=fd(U,V,theta1);  %%c(u,v) copula density
figure(3)
plot(1:n,cUV)        %%simply to check consistence: positivity
cWU=fd(W,U,theta2);   %%c(w,u) copula density
figure(4)
plot(1:n,cWU)
cWV_U=fwv(s1,s2,theta4);
figure(5)
plot(1:n,cWV_U);
%%%%% Calculating the copula density
fxyz=fr.*fy.*fz.*cUV.*cWU.*cWV_U;
AIC=2*6-2*sum(log(fxyz))
BIC=6*log(n)-2*sum(log(fxyz))
fxyz=fxyz.*0.025./max(fxyz)
AIC2=2*6-2*sum(log(fxyz))
BIC2=6*log(n)-2*sum(log(fxyz))
figure(6)
plot(z,fxyz,'ro')
xlabel('Temperature')
ylabel('f(r,y,z)')
figure(7)
plot(r,fxyz,'bo')
xlabel('Rainfall')
ylabel('f(r,y,z)')
figure(8)
plot(y,fxyz,'ko')
xlabel('Evapotranspiration')
ylabel('f(r,y,z)')
toc
```

3 Stochastic Differential Equations

3.1 Prediction of Weather Elements

```
function Rainfall_Temp_EvapoTransp_SDE_Model
%%%%% STEP ONE: Clean the command lines
clc; clear; close all; rng('default');

%%%%% STEP TWO: Invite data to the command line
Dta=xlsread('weatherdatacleaned');
Dta(Dta(:,4)<=2.54,:)=[];
z=0.5*Dta(:,2)+0.5*Dta(:,3);  %z=z(3861:1:3992);
mu=20.5442; sg=3.01666;n=length(z);
t=1:n; r=Dta(:,4);  %r=r(3861:1:3992);
y=Dta(:,8);%y=y(3861:1:3992);
%%%% SDE for temperature
w=2*pi/n;
Z=[]; B=[]; A=[]; m=20;
for i=1:n
Z(i)=z(i);
for j=1:m
B(i,j)=cos(j*w*i);
A(i,j)=sin(j*w*i);
end
end
Z3=LinearModel.fit([A, B], Z);
mu2=predict(Z3,[A,B]); ee1=z-mu2;sgg=dot(ee1,ee1)/(n-m);
Hi=xlsread('hessn'); gam3=1-Hi; sggt2=sgg.*gam3;sggt=sqrt(sggt2);
%sgtt=sggt(3861:1:3992);
%%%% Rainfall SDE Paramters
Z4=LinearModel.fit([A, B], r);
mu3=predict(Z4,[A,B]);
mu3(mu3<0)=2.54;
ee2=r-mu3;
vaRi=sum(ee2.^2)/(n-m);
ssR=sqrt(vaRi.*(1-Hi));
%ssR=ssR(3861:1:3992);
beta1=ssR./mu3;
alpha1=mu3./beta1;
%%%% Evapotranspiration SDE Parameters
Z5=LinearModel.fit([A, B], y);
mu4=predict(Z5,[A,B]);
err3=y-mu4;
vaYi=sum(err3.^2/(n-m));
ssY=sqrt(vaYi.*(1-Hi));
%ssY=ssY(3861:1:3992);
beta2=ssY./mu4;
alpha2=mu4./beta2;
```

```
%%%Next section of Brownian Motions
X=[]; X(1)=Z(1);
dt=1;
W=mean([zeros(100,1),sqrt(sg^3/n)*cumsum(randn(100,n-1))]);
%%%Focusing on Rainfall
X2=[]; X2(1)=r(1);

dt2=1;
W2=mean([zeros(100,1),sqrt(mean(ssR)^3/n)*cumsum(randn(100,n-1))]);
gam1=1-Hi;
%%% Focusing on Evapotranspiration
X3=[]; X3(1)=y(1);
dt3=1;
W3=mean([zeros(100,1),sqrt(mean(ssY)^10/n)*cumsum(randn(100,n-1))]);
gam2=1-Hi;

V=[]; V(1)=r(1)*y(1)*z(1);
for j=1:n-1
X3(j+1)=X3(j)+gam2(j)*(alpha2(j)*beta2(j)-beta2(j)-X3(j))...
*dt3+sqrt(2*gam2(j)*X3(j))*(W3(j+1)-W3(j));
X3(X3<0)=mu4(j+1);
X2(j+1)=X2(j)+gam1(j)*(alpha1(j)*beta1(j)-beta1(j)-X2(j))...
*dt2+sqrt(2*gam1(j)*X2(j))*(W2(j+1)-W2(j));
X2(X2<0)=mu3(j+1);
X(j+1)=X(j)+sggt(j)*(mu2(j)-X(j))*dt +sqrt(2*sggt(j))*(W(j+1)-W(j));
V(j+1)=X(j+1)*X2(j+1)*X3(j+1);
end

figure(1)
scatter(r,z,'ko')
hold on
scatter(X2,X,'ro', 'filled')
hold off
legend('Exact data','Predicted data')
xlabel('Rainfall in mm')
ylabel('Temperature in ^o C')


figure(2)
plot(r,y,'bo')
hold on
scatter(X2,X3,'r o','filled')
hold off
legend('Exact data','Predicted data')
xlabel('Rainfall in mm')
ylabel('Reference Evapotranspiration in mm')

figure(3)
```

```
plot(z,y,'ro')
hold on
scatter(X,X3,'b o','filled')
hold off
legend('Exact data','Predicted data')
xlabel('Temperature in ^oC')
ylabel('Reference Evapotranspiration in mm')

figure(4)
plot(t,r,'k-')
hold on
plot(t,X2, 'b--.')
hold off
legend('Exact data','Predicted data')
xlabel('Time in Days')
ylabel('Rainfall in mm')


figure(5)
plot(t,y,'k-')
hold on
plot(t,X3,'b--.')
hold off
legend('Exact data','Predicted data')
ylabel('Time in days')
ylabel('Reference Evapotranspiration in mm')

figure(6)
plot(t,z,'k-')
hold on
plot(t,X,'b--.')
hold off
legend('Exact data','Predicted data')
ylabel('Temperature in ^oC')
xlabel('Time in days')

% V_exact=r.*y.*z;
%  V=V'
% rs=sort(r);
% ts=sort(z);
% es=sort(y);
% Vs=sort(V);
%
% Vs=sort(V);
% Ves=sort(V_exact);
%  [rho,pval]=corr(r,V_exact,'Type','Pearson')
```

3.2 Tri-variate Stochastic Process

```
function Trivariate_Stochastic_Model
```

```
clc;close all; clear;


%%%%%DATA COLLECTION
Dta=xlsread('weatherdatacleaned');
Dta(Dta(:,4)<=2,:)=[];
z=0.5*Dta(:,2)+0.5*Dta(:,3);
r=Dta(:,4);
y=Dta(:,8);

L=112;L2=39;
w=2*pi/L; %%

B=[]; A=[]; m=24;
for i=1:L
for j=1:m
B(i,j)=cos(j*w*i);
A(i,j)=sin(j*w*i);
end
end


gam=leverage([A,B],'linear');
%%%%%Rainfall
X=r;
V1=zeros(L,L2);
V=V1;
Rsq=[];
Y=[]; Y2=[];
j1=1; j2=L;
%%%%%Evapotranspiration
B1=[]; A1=[]; m1=24;
for i=1:L
for j=1:m1
B1(i,j)=cos(j*w*i);
A1(i,j)=sin(j*w*i);
end
end
Y3=[]; V2=zeros(L,L2);V3=zeros(L,L2);
Y4=[];
%%%%%Temperature
B2=[]; A2=[]; m2=20;
for i=1:L
for j=1:m2
B2(i,j)=cos(j*w*i);
A2(i,j)=sin(j*w*i);
end
end
Y4=[];V4=zeros(L,L2); V5=zeros(L,L2); Y5=[];
```

```
for k=1:L2
for i=j1:j2
Y(i)=X(i);
Y3(i)=y(i);
Y4(i)=z(i);

end
Y(Y==0)=[];
Y3(Y3==0)=[];
Y4(Y4==0)=[];
md1=LinearModel.fit([A, B], Y);
md2=LinearModel.fit([A1, B1], Y3);
md3=LinearModel.fit([A2,B2],Y4);
Rsq(k)=md3.Rsquared.Ordinary;
V(:,k)=predict(md1,[A,B]);
V3(:,k)=predict(md2,[A1,B1]);
V5(:,k)=predict(md3,[A2,B2]);
V1(:,k)=Y';
V2(:,k)=Y3';
V4(:,k)=Y4';

j1=j2+1;
j2=j2+L;
Y=[];
Y3=[];
Y4=[];
end
NumRsq=4.*Rsq.^2.*(1-Rsq.^2).^2.*(L-m2-1).^2;
DenRsq=(L^2-1)*(L+3);
SderrRsq=2.*sqrt(NumRsq./DenRsq); %%Confidence interval
lerr=Rsq-Rsq+SderrRsq;
Uerr=SderrRsq;
figure(1)
bar(1:L2,Rsq)
xlabel('Growing Season')
ylabel('R^2')
hold on
er = errorbar(1:L2,Rsq,lerr,Uerr);
er.Color = [0 0 0];
er.LineStyle = 'none';
hold off
VF=zeros(L,L2);
VASM=zeros(L,L2);
VLASM=zeros(L,L2);
for i=1:L2
%%%%RAINFALL
r1=V1(:,i);
mu1=V(:,i);
mu1(mu1<0)=2.54;
```

```
err2=V1(:,i)-mu1;
VaRi=sum(err2.^2)/(L-m);
SSR=sqrt(VaRi.*(1-gam));
beta1=SSR./mu1;
alpha1=mu1./beta1;
XR=[]; XR(1)=r1(1);
%%%%EVAPOTRANSPIRATION
y1=V2(:,i);
mu2=V3(:,i);
XY=[]; XY(1)=y1(1);
err3=V2(:,i)-mu2;
VaYi=sum(err3.^2)/(L-m);
SSY=sqrt(VaYi.*(1-gam));
beta2=SSY./mu2;
alpha2=mu2./beta2;
%%%%TEMPERATURE
z1=V4(:,i);
mu3=V5(:,i);
XZ=[]; XZ(1)=z1(1);
errz=(z1-mu3).^2./(L-m2);
md4=LinearModel.fit([A2,B2],errz);
sggt=predict(md4,[A2,B2]);
sggt(sggt<0)=sqrt(mean(errz))/L;
sggt=sqrt(sggt);
%%%%Modeling V


for j=1:L-1
XR(j+1)=XR(j)+(beta1(j)*(alpha1(j)-1)-XR(j))+sqrt(2*gam(j)*beta1(j)*XR(j))*randn;
XR(XR<0)=mu1(j);
XY(j+1)=XY(j)+(beta2(j)*(alpha2(j)-1)-XY(j))+sqrt(2*gam(j)*beta2(j)*XY(j))*randn/L;
XZ(j+1)=XZ(j)+  (mu3(j)-XZ(j))+sggt(j)*sqrt(gam(j))*randn;



end
%%%Assuming Non_uniform Noise
VHN=XR.*XY.*XZ;
VASM(:,i)=VHN';
%%%%%Exact
VE=r1.*y1.*z1;
VF(:,i)=VE';
%%%%Asumming Uniform Noise
VD=mu1.*mu2.*mu3;
VLASM(:,i)=VD';
%    figure(i+1)
%    plot(1:L,VE,1:L,VD,1:L,VHN,'k-')
%    legend('Raw','Estimate','HN','Location','NW')
```

```
end
Vexact=reshape(VF,L*L2,1);
Vest1=reshape(VASM,L*L2,1);
Vest2=reshape(VLASM,L*L2,1);
figure(2)
plot(1:L*L2, Vexact,'b-',1:L*L2,Vest1,'r-','LineWidth',2)

xlim([0,4000])
legend('Raw Data', 'Model under  Uniform Noise Assumption','Location','NW')

figure(3)
plot(1:L*L2, Vexact,'b-',1:L*L2,Vest2,'r-','LineWidth',2)
xlabel('Time in Days')
ylabel('V')
xlim([0,4000])
legend('Raw Data', 'Model under Correlated Noises Assumption','Location','NW')

xlswrite('Patdata1',VASM)
xlswrite('Patdata2',VLASM)
```

4 Python Codes for Estimating Maize Yield

The mathematics behind the algorithms can be found in Bishop and Nasrabadi (2006), Samikwa et al. (2020), and Suzuki (2021).

4.1 Neural Network Algorithm

```
from __future__ import print_function


import os
os.environ["THEANO_FLAGS"] = "mode=FAST_RUN,device=gpu,floatX=float32"

from math import sqrt
import numpy as np
import matplotlib.pyplot as plt
import pandas
import math
import io
import keras
import requests
from matplotlib import pyplot
import matplotlib.patches as mpatches
from numpy import concatenate
from tensorflow.python.keras.models import Sequential
from tensorflow.python.keras.layers import Dense
from tensorflow.python.keras.layers import Dropout
from tensorflow.python.keras.layers import Activation
```

```
from tensorflow.python.keras.layers import LSTM
from tensorflow.keras.callbacks import EarlyStopping
from sklearn.preprocessing import MinMaxScaler
from sklearn.metrics import mean_squared_error
from matplotlib.lines import Line2D
from keras import metrics
from keras.layers import LeakyReLU
import tensorflow as tf

# fix random seed for reproducibility
np.random.seed(10)

#Data uploading
df = pandas.read_excel('C:/Users/User/Desktop/EricData.xls')
df2=pandas.read_excel('C:/Users/User/Desktop/betheldata.xls')
# dataset = df.drop(columns=['V=r*y*z/1000','V','PlacketPDF', 'VinePDF','FrankPDf', 'time
dataset = np.array(df['FrankPDf'])
dataset2 = np.array(df2['V'])
#crop yield
crop_yield = np.array(df['res1'])
crop_yield2 = np.array(df2['yield'])
# normalize the dataset (Neural Networks are sensitive to the scale of the input data)
scaler  = MinMaxScaler(feature_range=(0, 1))
# dataset = scaler.fit_transform(dataset)
#normalisation function for 1D array (i.e. for crop yields, the MinMaxScaler only works f
def normalize(arr, t_min, t_max):
norm_arr = []
diff = t_max - t_min
diff_arr = max(arr) - min(arr)
for i in arr:
temp = (((i - min(arr))*diff)/diff_arr) + t_min
norm_arr.append(temp)
return norm_arr

range_to_normalize = (0, 1)
crop_yield = normalize(
crop_yield, range_to_normalize[0],
range_to_normalize[1])

dataset = normalize(
dataset, range_to_normalize[0],
range_to_normalize[1])

crop_yield = np.array(crop_yield)
# dataset = np.array(dataset)
dataset = dataset[:3885]
new_dataset = np.array(np.array_split(dataset, 37))

# split into train, validation and test sets
# values = new_dataset.values
```

```
num_train_size = 26 # approx 70% of the dataset


#separated datasets
train = new_dataset[:num_train_size]
validate= new_dataset[num_train_size:]


#traning set
trainX, trainY = train, crop_yield[:26]
#validation set
validateX, validateY = validate, crop_yield[26:37]
print(trainX.shape, len(trainX), trainY.shape)
print(validateX.shape, len(validateX), validateY.shape)


# deep neural network: feed forward neural network <multilayer perceptron>
model = Sequential()
model.add(Dense(10, activation='relu', input_dim = 105))
model.add(Dense(10, activation='relu'))
model.add(Dense(10, activation='relu'))
model.add(Dropout(0.2))
model.add(Dense(1))

model.compile(loss='mse', optimizer='adam')


#early stopping regularization used to avoid overfitting
callback = EarlyStopping(monitor='val_loss', mode='min',verbose=1,
                         patience=20, baseline=0.000073750)


#training :  epochs and batch size 10
history = model.fit(trainX, trainY, epochs=200, batch_size=50, callbacks=[callback],
validation_data=(validateX, validateY), shuffle=True)

# plot loss during training
plt.figure(
figsize=(7, 5))
pyplot.plot(history.history['loss'], label='training')
pyplot.plot(history.history['val_loss'], label='validation')
pyplot.title('Model train vs Validation loss')
plt.xlabel('no of epochs')
plt.ylabel('loss')
pyplot.legend(['Train', 'Validation'], loc='upper right')
pyplot.show()

# predictions - with validation set :-(
predictions = model.predict(validateX)
```

```
predictions = predictions.flatten()

# root mean squared error (RMSE)
rmse = sqrt(mean_squared_error(validateY, predictions))
print('Test RMSE: %.3f  ' % (rmse))
print('Test RMSE: %.3f %% ' % ((rmse)*100))

# R
def R(y, y_pred):
residual = tf.reduce_sum(tf.square(tf.subtract(y, y_pred)))
total = tf.reduce_sum(tf.square(tf.subtract(y, tf.reduce_mean(y))))
r2 = tf.subtract(1.0, tf.truediv(residual, total))
return tf.math.sqrt(r2)

print('R score is: ', R(validateY, predictions))



corr_matrix = np.corrcoef(validateY, predictions)
corr = corr_matrix[0,1]
R_sq = corr**2

print(R_sq)

plt.figure(
figsize=(9, 6))
plt.plot(validateY, color='#017C8F')
plt.plot(predictions, color='coral')
print(predictions)
print(validateY)
plt.xlabel('Year ', fontsize=18)
plt.ylabel('Yeild', fontsize=18)
plt.tight_layout()
Original = mpatches.Patch(color='#017C8F', label='Actual')
Forecast = mpatches.Patch(color='coral', label='Predicted')
plt.legend(handles=[Original,Forecast])
# positions = (0, 2000, 4000, 6000, 8000, 10000,12000)
# labels = ("Dec", "Feb", "April", "June", "Aug", "Oct","Dec")
# plt.xticks(positions, labels)
plt.show()

print(predictions)
print(validateY)
```

4.2 Decision Trees and Random Forest

```
#Importing necessary libraries
import pandas as pd
import sklearn.model_selection as model_selection
import numpy as np
```

```
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error
from sklearn.ensemble import RandomForestRegressor
from sklearn.tree import DecisionTreeRegressor
import matplotlib.pyplot as plt
import seaborn as sns


#reading the data and converting it to pandas dataframe
bdata=pd.read_excel('betheldata.xls')
#dropping rows with missing data
bdata=bdata.dropna()

#spliting the data into independent and dependent variables
X=bdata[['V']].values
y=bdata['Maize yield'].values

#splitting the 2 datasets into training and testing sets
X_train, X_test, y_train, y_test = model_selection.train_test_split(X, y,
           train_size=0.8,test_size=0.2, random_state=101)

# fitting a random forest model with high depth
model_forest = RandomForestRegressor(max_depth=10, random_state=0)
model_forest.fit(X, y)

#predicting using i. all the values of 'V', ii. Training set only, iii. testing set only
y_pred_forest = model_forest.predict(X)
y_pred_train_forest = model_forest.predict(X_train)
y_pred_test_forest= model_forest.predict(X_test)

#monitoring model performance for the whole data(i.e. all Vs), training and testing sets
mse_forest = mean_squared_error(y, y_pred_forest)
mse_train_forest = mean_squared_error(y_train, y_pred_train_forest)
mse_test_forest = mean_squared_error(y_test, y_pred_test_forest)

#printing root mean squared error for all the three data sets
print('rmse:',(mse_forest)**0.5)
print('Training rmse:',
      ((mse_train_forest)**0.5).round(2),'\n',
     'Testing rmse:',((mse_test_forest)**0.5).round(2))

#plotting the distribution of true and predicted values of the target variable(i.e distri
plt.plot(bdata['Year'].values,bdata['Maize yield'].values)
plt.plot(bdata['Year'].values,y_pred_forest,color='black',linestyle='dotted')
plt.xlabel('Years')
plt.ylabel('Maize yield')
plt.legend(['True values [Maize yield]', 'Predicted values [Maize yield]'], loc=2)
plt.title('Simulation of Random Forest Regressor model')
```

```
# fitting a decision tree model with high depth
model_tree1 = DecisionTreeRegressor(max_depth=10, random_state=0)
model_tree1.fit(X, y)

#predicting using i. all the values of 'V', ii. Training set only, iii. testing set only
y_pred_tree1 = model_tree1.predict(X)
y_pred_train_tree1 = model_tree1.predict(X_train)
y_pred_test_tree1= model_tree1.predict(X_test)

#monitoring model performance for the whole data(i.e. all Vs), training and testing sets
mse_tree1 = mean_squared_error(y, y_pred_tree1)
mse_train_tree1 = mean_squared_error(y_train, y_pred_train_tree1)
mse_test_tree1 = mean_squared_error(y_test, y_pred_test_tree1)

#printing root mean squared error for all the three data sets
print('rmse for the whole dataset:',(mse_tree1)**0.5)
print('Training rmse:',((mse_train_tree1)**0.5).round(2),'\n',
                'Testing rmse:',((mse_test_tree1)**0.5).round(2))

#plotting the distribution of true and predicted values of the target variable(i.e distri
plt.scatter(bdata['Year'].values,bdata['Maize yield'].values)
plt.plot(bdata['Year'].values,y_pred_tree1,color='black',linestyle='dotted')
plt.xlabel('Years')
plt.ylabel('Maize yield')
plt.legend(['True values [Maize yield]', 'Predicted values [Maize yield]'], loc=2)
plt.title('Simulation of Decision Tree Regressor model')


# fitting a decision tree model with high depth
model_tree2 = DecisionTreeRegressor(max_depth=3, random_state=0)
model_tree2.fit(X, y)

#predicting using i. all the values of 'V', ii. Training set only, iii. testing set only
y_pred_tree2 = model_tree2.predict(X)
y_pred_train_tree2 = model_tree2.predict(X_train)
y_pred_test_tree2= model_tree2.predict(X_test)

#monitoring model performance for the whole data(i.e. all Vs), training and testing sets
mse_tree2 = mean_squared_error(y, y_pred_tree2)
mse_train_tree2 = mean_squared_error(y_train, y_pred_train_tree2)
mse_test_tree2 = mean_squared_error(y_test, y_pred_test_tree2)

#printing root mean squared error for all the three data sets
print('rmse for the whole dataset:',(mse_tree2)**0.5)
print('Training rmse:',((mse_train_tree2)**0.5).round(2),'\n',
'Testing rmse:',((mse_test_tree2)**0.5).round(2))

#plotting the distribution of true and predicted values of the target variable(i.e distri
plt.scatter(bdata['Year'].values,bdata['Maize yield'].values)
plt.plot(bdata['Year'].values,y_pred_tree2,color='black',linestyle='dotted')
```

```
plt.xlabel('Years')
plt.ylabel('Maize yield')
plt.legend(['True values [Maize yield]', 'Predicted values [Maize yield]'], loc=2)
plt.title('Simulation of Decision Tree Regressor model')

# fitting a decision tree model with high depth
model_tree3 = DecisionTreeRegressor(max_depth=6, random_state=0)
model_tree3.fit(X, y)

#predicting using i. all the values of 'V', ii. Training set only, iii. testing set only
y_pred_tree3 = model_tree3.predict(X)
y_pred_train_tree3 = model_tree3.predict(X_train)
y_pred_test_tree3= model_tree3.predict(X_test)

#monitoring model performance for the whole data(i.e. all Vs), training and testing sets
mse_tree3 = mean_squared_error(y, y_pred_tree3)
mse_train_tree3 = mean_squared_error(y_train, y_pred_train_tree3)
mse_test_tree3 = mean_squared_error(y_test, y_pred_test_tree3)

#printing root mean squared error for all the three data sets
print('rmse for the whole dataset:',(mse_tree3)**0.5)
print('Training rmse:',((mse_train_tree3)**0.5).round(2),'\n',
'Testing rmse:',((mse_test_tree3)**0.5).round(2))

#plotting the distribution of true and predicted values of the target variable(i.e distri
plt.scatter(bdata['Year'].values,bdata['Maize yield'].values)
plt.plot(bdata['Year'].values,y_pred_tree3,color='black',linestyle='dotted')
plt.xlabel('Years')
plt.ylabel('Maize yield')
plt.legend(['True values [Maize yield]', 'Predicted values [Maize yield]'], loc=2)
plt.title('Simulation of Decision Tree Regressor model')
```

4.3 Linear Regression

```
# fitting a linear model with high depth
model_linear = LinearRegression()
model_linear.fit(X, y)

#predicting using i. all the values of 'V', ii. Training set only, iii. testing set only
y_pred_linear = model_linear.predict(X)
y_pred_train_linear = model_linear.predict(X_train)
y_pred_test_linear= model_linear.predict(X_test)

#monitoring model performance for the whole data(i.e. all Vs), training and testing sets
mse_linear = mean_squared_error(y, y_pred_linear)
mse_train_linear = mean_squared_error(y_train, y_pred_train_linear)
mse_test_linear = mean_squared_error(y_test, y_pred_test_linear)
```

```
#printing root mean squared error for all the three data sets
print('rmse:',(mse_linear)**0.5)
print('Training rmse:',((mse_train_linear)**0.5).round(2),'\n','Testing rmse:',((mse_test

#plotting the distribution of true and predicted values of the target variable(i.e distri
plt.scatter(bdata['time'].values,bdata['Yield'].values)
plt.plot(bdata['time'].values,y_pred_linear,color='black',linestyle='dotted')
plt.xlabel('time')
plt.ylabel('Maize yield')
plt.legend(['True values [Maize yield]', 'Predicted values [Maize yield]'], loc=2)
plt.title('Simulation of Linear Regressor model')
```

4.4 K-Nearest Neighbors

```
# fitting a KNearestNeighbor model with high depth
model_knn = KNeighborsRegressor()
model_knn.fit(X, y)

#predicting using i. all the values of 'V', ii. Training set only, iii. testing set only
y_pred_knn = model_knn.predict(X)
y_pred_train_knn = model_knn.predict(X_train)
y_pred_test_knn= model_knn.predict(X_test)

#monitoring model performance for the whole data(i.e. all Vs), training and testing sets
mse_knn = mean_squared_error(y, y_pred_knn)
mse_train_knn = mean_squared_error(y_train, y_pred_train_knn)
mse_test_knn = mean_squared_error(y_test, y_pred_test_knn)

#printing root mean squared error for all the three data sets
print('rmse:',(mse_knn)**0.5)
print('Training rmse:',((mse_train_knn)**0.5).round(2),'\n','Testing rmse:',((mse_test_k

#plotting the distribution of true and predicted values of the target variable(i.e distr
plt.scatter(bdata['time'].values,bdata['Yield'].values)
plt.plot(bdata['time'].values,y_pred_knn,color='black',linestyle='dotted')
plt.xlabel('time')
plt.ylabel('Maize yield')
plt.legend(['True values [Maize yield]', 'Predicted values [Maize yield]'], loc=2)
plt.title('Simulation of Linear Regressor model')
```

4.5 Support Vector Machines

```
# fitting a support vector machine model with high depth
model_svm = svm.SVR(kernel='linear')
model_svm.fit(X, y)

#predicting using i. all the values of 'V', ii. Training set only, iii. testing set onl
```

```
y_pred_svm = model_svm.predict(X)
y_pred_train_svm = model_svm.predict(X_train)
y_pred_test_svm= model_svm.predict(X_test)

#monitoring model performance for the whole data(i.e. all Vs), training and testing set
mse_svm = mean_squared_error(y, y_pred_svm)
mse_train_svm = mean_squared_error(y_train, y_pred_train_svm)
mse_test_svm = mean_squared_error(y_test, y_pred_test_svm)

#printing root mean squared error for all the three data sets
print('rmse:',(mse_svm)**0.5)
print('Training rmse:',((mse_train_svm)**0.5).round(2),'\n','Testing rmse:',((mse_test_

#plotting the distribution of true and predicted values of the target variable(i.e dist
plt.scatter(bdata['time'].values,bdata['Yield'].values)
plt.plot(bdata['time'].values,y_pred_svm,color='black',linestyle='dotted')
plt.xlabel('time')
plt.ylabel('Maize yield')
plt.legend(['True values [Maize yield]', 'Predicted values [Maize yield]'], loc=2)
plt.title('Simulation of SVM model')
```

4.6 Measures of Accuracy, Uncertainty, Reliability and Precision

The mathematical formulas behind the calculations in this section can be found in Barzkar et al. (2022), (Saberi-Movahed et al., 2020) and Chicco et al. (2021).

```
function preci_calculations
clc;
close all;
raw= xlsread('maizedata')
ybar=mean(raw);
Den1=sum((raw-ybar).^2);
%%%Plackett
X= xlsread('plpreddata');
Num1=sum ((X-raw).^2);
Rsplack=1-Num1/Den1;
RMSE=sqrt(Num1/length(X));
MAPE=mean(abs((raw-X)./raw));
BIASPrack=mean(X-raw);
%%%SI Calculation
Xbar=mean(X);
SIsquare=mean( ((X-Xbar)-(raw-ybar)).^2);
SIpracket=sqrt(SIsquare);

%%%Frank
X2=xlsread('frpreddata');
Num2=sum ((X2-raw).^2);
Rsfrank=1-Num2/Den1;
RMSE2=sqrt(Num2/length(X2));
```

```
MAPE2=mean(abs((raw-X2)./raw));
BIASfra=mean(X2-raw);
t=2007:2017;
%%%SI Calculation
Xbar=mean(X2);
SIsquare=mean( ((X2-Xbar)-(raw-ybar)).^2);
SIX2=sqrt(SIsquare);



%%%%Vine
Vine=xlsread('vinpreddata');
Num3=sum ((Vine-raw).^2);


RsVine=1-Num3/Den1
RMSEVine=sqrt(Num3/length(Vine))
MAPEvine=mean(abs((raw-Vine)./raw))
BiasVine=mean(Vine-raw)
%%%SI Calculation
Xbar=mean(Vine);
SIsquare=mean( ((Vine-Xbar)-(raw-ybar)).^2);
SIVine=sqrt(SIsquare);



%SMM(stochastic Maize Model)
Vtree=xlsread('smpreddata');
Num4=sum ((Vtree-raw).^2);
RsTree=1-Num4/Den1;
RMSEtree=sqrt(Num4/length(Vtree));
MAPEtree=mean(abs((raw-Vtree)./raw));
BIASVtree=mean(Vtree-raw);
%%%SI Calculation
Xbar=mean(Vtree);
SIsquare=mean( ((Vtree-Xbar)-(raw-ybar)).^2);
SISMM=sqrt(SIsquare)







%Decision tree
VVtree=xlsread('dtmpreddata');
Num5=sum ((VVtree-raw).^2);
RsVsc=1-Num5/Den1;
RMSEVsc=sqrt(Num5/length(VVtree));
MAPEVsc=mean(abs((raw-VVtree)./raw));
BIAStree=mean(VVtree-raw);
```

```
%%%SI Calculation
Xbar=mean(VVtree);
SIsquare=mean( ((VVtree-Xbar)-(raw-ybar)).^2);
SIVVtree=sqrt(SIsquare);




%Support Vector Machines
XVSM=xlsread('svmpreddata');

Num6=sum ((XVSM-raw).^2);
RsVSM=1-Num6/Den1;
RMSEVSM=sqrt(Num6/length(XVSM));
MAPEVSM=mean(abs((raw-XVSM)./raw));
BIAS=mean(XVSM-raw);
%%%SI Calculation
Xbar=mean(XVSM);
SIsquare=mean( ((XVSM-Xbar)-(raw-ybar)).^2);
SIXVSM=sqrt(SIsquare);

  %%%%Uncertainity at 95 \% level of confidence
%%%SVM
U95squre=sum((raw-ybar).^2)+ sum((XVSM-raw).^2);
U95z=sqrt(U95squre)/length(raw).*1.96;
%%%Decision tree
U95squre=sum((raw-ybar).^2)+ sum((VVtree-raw).^2);
U95ztree=sqrt(U95squre)/length(raw).*1.96;
%%%%SMM
U95squre=sum((raw-ybar).^2)+ sum((Vtree-raw).^2);
U95zsmm=sqrt(U95squre)/length(raw).*1.96;
%%%%Placket
U95squre=sum((raw-ybar).^2)+ sum((X-raw).^2);
U95zplac=sqrt(U95squre)/length(raw).*1.96;
%%%%Vine
U95squre=sum((raw-ybar).^2)+ sum((Vine-raw).^2);
U95zvine=sqrt(U95squre)/length(raw).*1.96;
%%%%Frank
U95squre=sum((raw-ybar).^2)+ sum((X2-raw).^2);
U95zfrank=sqrt(U95squre)/length(raw).*1.96;
table(U95z, U95ztree, U95zsmm,U95zplac, U95zvine, U95zfrank, ...
'VariableNames',{'SVM U95', 'DTM U95', 'SMM U95', 'Plack U95'...
 ,'Vine U95', 'Frank U95'})


%%%%Reliability
KSVM1=abs((XVSM-raw)./raw);
W=(KSVM1<0.05);
ksv=mean(W);
```

```
Kvvt=abs((VVtree-raw)./raw);
Wvvt=(Kvvt<0.05);
ksvvvt=mean(Wvvt);

Kvt=abs((Vtree-raw)./raw);
Wvt=(Kvt<0.05);
ksvvt=mean(Wvt);

KX=abs((X-raw)./raw);
WX=(KX<0.05);
ksX=mean(WX);

KVine=abs((Vine-raw)./raw);
WVine=(KVine<0.05);
ksVine=mean(WVine);

KX2=abs((X2-raw)./raw);
WX2=(KX2<0.05);
ksX2=mean(WX2);

table(ksv, ksvvvt, ksvvt,ksX, ksVine, ksX2, ...
'VariableNames',{'SVM Re', 'DTM Re', 'SMM Re', 'Plack Re'...
 ,'Vine Re', 'Frank Re'})




figure(1)
plot(t,raw,'o-',t,Vine,'+-',t, X,'o-',t,X2,'>-','LineWidth',1.5)
legend('Raw Data','Vine PDF','Plackett','Frank','Location','SW')
xlabel('Year')
ylabel('Maize yield (kg/ha)')
ylim([0,95])
grid on
xlim([2006, 2018])
t2=1:19;
Loss1=readmatrix('lossdata');
figure(2)
plot(t2,Loss1(:,2),'+-', t2, Loss1(:,4),'<-',t2,Loss1(:,6),'o-', 'LineWidth',1.5)
xlabel('Number of epochs')
ylabel('Loss')
grid on
legend('Vine PDF','Plackett PDF','Frank PDF' )

figure(2)
plot(t,raw,'o-', t, VVtree, 'k o-', 'LineWidth',1.5)
xlabel('Year')
ylabel('Maize yield (kg/ha)')
legend('raw', 'prediction')
ylim([0,95])
```

```
xlim([2006, 2018])
grid on
figure(3)
plot(t,raw,'o-', t, Vtree, 'k o-', 'LineWidth',1.5)
xlabel('Year')
ylabel('Maize yield (kg/ha)')
legend('raw', 'prediction')
ylim([0,95])
xlim([2006, 2018])
grid on

figure(4)
plot(t,raw,'o-', t, XVSM, 'k o-', 'LineWidth',1.5)
xlabel('Year')
ylabel('Maize yield (kg/ha)')
legend('raw', 'prediction')
ylim([0,95])
xlim([2006, 2018])
grid on
```

References

Barzkar, A., Najafzadeh, M., and Homaei, F. (2022). Evaluation of drought events in various climatic conditions using data-driven models and a reliability-based probabilistic model. *Natural Hazards*, 110(3):1931–1952.

Bishop, C. M. and Nasrabadi, N. M. (2006). *Pattern recognition and machine learning*, volume 4. Springer.

Chicco, D., Warrens, M. J., and Jurman, G. (2021). The coefficient of determination r-squared is more informative than smape, mae, mape, mse and rmse in regression analysis evaluation. *PeerJ Computer Science*, 7:e623.

Joe, H. (1996). Families of m-variate distributions with given margins and m (m-1)/2 bivariate dependence parameters. *Lecture notes-monograph series*, pages 120–141.

Joe, H. and Kurowicka, D. (2011). *Dependence modeling: vine copula handbook*. World Scientific.

Saberi-Movahed, F., Najafzadeh, M., and Mehrpooya, A. (2020). Receiving more accurate predictions for longitudinal dispersion coefficients in water pipelines: training group method of data handling using extreme learning machine conceptions. *Water Resources Management*, 34(2):529–561.

Samikwa, E., Voigt, T., and Eriksson, J. (2020). Flood prediction using iot and artificial neural networks with edge computing. In *2020 International Conferences on Internet of Things (iThings) and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom) and IEEE Smart Data (SmartData) and IEEE Congress on Cybermatics (Cybermatics)*, pages 234–240. IEEE.

Suzuki, J. (2021). *Statistical Learning with Math and Python*. Springer.

Zhang, L. and Singh, V. P. (2019). *Copulas and their applications in water resources engineering*. Cambridge University Press.