Everybody welcome.

This demonstration will introduce Copula Methods in R.

In particular, we will go through the example that I went through way back

at the start of this course about modelling stocks and combinations of stocks.

Now, we'll be using a few specialised packages here.

One is reshape.

Reshape, it's not an analysis package,

it's more about reorganising your data in particular ways that will be useful to us.

The package copula is of course, for copula modelling in R.

Package stats4 for the function mle,

which I will explain how and why we're using it later.

Then the package Rsolnp which is good for solving constrained non-linear problems.

We have a helper function to

talk about copula functions and just explore their different structures.

You've seen these figures in the slides as well.

But basically, solnp produce these figures function.

Well, what it sets up is a two-by-two grid of plots and persp

here plots one of these cubes like the one here and plots a function inside those cubes.

That function happens to be pCopula for the given copula.

This is an R method for copula objects.

It will plot several things.

Will plot the copula function, C,

will plot the copula density function i.e. dCopula,

little c, and will also make contour plots for these copula functions.

Also have a title on top explaining what the copula is.

Let's start with independence copula in two-dimensions.

This is what it looks like.

Here's just a product of the individual,

a uniform density functions, standard uniform functions.

This is the density.

Again, it looks like it's just a product,

and is just a constant density over this particular as your one block.

Contour plots is what you would expect as well.

Now, we can make a normal copula.

This is a Gaussian copula with correlation parameter 0.9.

In two-dimensions, we'll have to scroll past all these messages.

Unfortunately, we know.

No, we don't have to scroll past these messages unfortunately,

so please cut up to this point.

I'm going to narrate to the normal copula

again. Next copula we'll

talk about is the normal copula, the Gaussian copula.

We'll use the correlation of 0.9 just to make it very visible.

Now it's a little bit hard to eyeball it for

the copula function and code differs from the independence copula,

but if you look at the contour plot,

there is a bit more density where both values are high.

In terms of the density,

we can now see a ridge here that corresponds to where both u1 and u2 have high values.

The spikes are weird-looking,

but they are basically there

because all of the normal density going off into infinity,

gets smushed onto those quantiles.

That's the intuition there.

Here's the tCopula.

I think the spikes are even bigger there,

although it is a bit hard to distinguish.

We have to specify to which freedom for t. Well,

we don't have to specify,

it defaults to four.

As you can see that we couldn't specify it.

Here's the gumbelCopula. Unlike the others,

you can see that it's not symmetrical.

Here it does have a similar shape to the others,

but the spikes are a bit different.

The spikes have different spikiness at different ends.

We'll apply this to microwave ovens example as before.

We've played around with the data quite a lot below this dataset.

Then this is what it looks like.

Now, before we work with it by transforming it.

Now, let's work with it directly.

In particular, we'll obtain the empirical quantiles and observations.

Here they are. Let's fit a Gaussian copula to it using the inverse correlation method.

That is, it'll play around with the copula parameters

until it gets to the point

where the observed correlation matches the predicted.

We can test that this is what it's doing or something very

close by napping the empirical quantiles onto normal quantiles,

that's what qnorm function does,

and then computing the sample correlation there.

As you can see, the values are very,

very similar, not that identical,

but very similar, 0.81.

Alternatively, we could choose to model these directly.

Now we have this type of skewed to

the right distribution so maybe we can use something like Gamma.

Gamma is a pretty flexible distribution which can represent

a range of different skewnesses and also has a scale parameter,

so it's a pretty good one to work with.

The function to do that is called mvdc.

It specifies the copula,

the margin families in this case is Gamma and their parameters.

Incidentally, we could mix the margin families if you wanted to.

The function that fits in is called fitMvdc,

and we do have to specify initial parameter values.

That produces what's called a parametric copula fit and it gives you a lot of

information including confidence intervals and a model from which we can simulate.

Here's what a typical fit looks like.

First we define our

multivariate copula or Marginal Valued copula.

I believe that's what MV stands for,

and we will use a normal copula with two dimensions.

We'll use Gamma margins,

we'll use parameters, and we have to specify basically lists for each of these four.

For the first one we use shape equals 1 or rate equals 1, shape equals.

For the second one the same.

We don't really worry about specifying them because we'll just actually estimate

the correct one later using the fitMvdc function.

We use the data ovens.

Note this is not the transformed data, not the quantile data.

This is the original data.

We specify the copula,

and then we specify the starting values so that it knows where to start,

and also we specify the initial copula correlation.

What do we see? We see the shape parameters being pretty low and

the rate parameters having these values but these are Gamma distribution parameters.

Then we have the correlation for the copula,

which is actually pretty close to what we had before.

We have the likelihood too so we can,

if we wanted to get AIC,

BIC, and other model selection tools.

If we call the summary function,

we could also get the standard errors for

these estimates because this is just a likelihood estimation.

Or we can even get confidence intervals.

If we wanted to obtain the fitted model we could use @mvdc.

It's complicated, but basically R has two ways of having

object-oriented programming and one of them uses

this @ instead of dollar signs to extract elements.

This is and one of those.

This is a way to simulate random datasets.

All we're doing here is using rMvdc.

We specify the numbers and we specify the fitted copula.

We convert it to a DataFrame,

and these are simulated values.

They look quite a bit like the original looks.

Now let's proceed with stocks example.

Here, I'll illustrate exactly how I got the results I

got in the demonstration at the start of the term.

Let's say that we have five different stocks: IBM,

Microsoft, British Petroleum, Coca-Cola and Duke Energy.

As you might expect, two of these,

for example, are tech stocks,

they might be correlated with each other in some way.

Two of them are energy stocks and so

they might be correlated and one of them

is not really related to the others, at least not directly.

Now, we're going to load the data,

combine them and sort them by date now.

This is all data management and we'll briefly talk through what they do.

First of all, we have the five symbols and we have the data set stocks.

You can download these data sets in the data set folder or in the data set zip file.

Then we'll have a function called readstock.

What it does is it gets us to

the folder dataset subfolder stocks,

and then within that, the ticker symbol .csv.

Grab one of those files,

read the CSV file,

get the date and the adjusted closing price,

then set the column names

to date and the ticker symbol.

Readstock will take a ticker symbol and return a dataframe with two columns,

date and the closing price with the column names being date and the ticker symbol.

Then what we'll do is ticker symbols, a map.

Now map is a function, the per package.

What it'll do is for every symbol,

it'll evaluate this function with that particular symbol and this argument.

It's a short version of L apply. We get that.

We have a list of these tables of stock prices,

one for each stock.

Then we'll pass it on to reduce.

Now reduce what it does is it takes a list,

takes the first two elements and it runs whatever function you give it to.

In this case it's left_join on those first two elements.

Takes the result, runs the function again,

this time on the result and the third element,

then the result in the fourth element and so on until you have the final answer.

What this is actually doing is it's taking the first stock output for IBM,

say, and it merges it with one from Microsoft by date.

Then it merges it with BP by date and then Coca-Cola and then Duke Energy.

Then arrange that, just sort it by date just to make sure.

The final result is going to be a table

with the date column and then a column for the closing price feature stocks.

Six columns total.

Now, let's visualise these.

We can do this using some other functions.

This is in reshape tool, it's called melt.

The idea then is that we're going to have this data frame of

six columns and one row.

Let me start over.

Now visualise this and you are going to use

among other things this melt function and some ggplot functions.

Melt is basically a function for transforming data frames.

We start off with a data frame that has

a column date and then a column for each of the stocks.

But in order to plot them using R's facilities,

we need to essentially stack these.

We need to have a new data frame with three columns and the three columns are date,

stock name and the stock price.

We essentially want to stack the stock prices on top of each other.

The way we're going to do that,

is using the melt function and ID date means that

this is essentially we want to keep that as an ID variable for all of them.

Then we have a variable name stock,

which certainly use,

so it'll keep the date and then when it does the stacking,

it'll put the names of the stocks,

the ticker symbols into the stock column and

then we'll have a third column value name which has an actual stock values.

I would actually recommend running this.

We're just running this line after running

all the lines before it and just seeing what it gives you.

It'll give you a pretty good idea of what this function can do.

Now we pass to ggplot.

We saved that on our horizontal axis, we have the date.

On the vertical axis we have the value.

We'll colour according to the stock and we'll group in according to

a stock and then we'll use a line plot.

Conveniently enough, it also produces a legend here.

Here are our stocks. You can

see the IBM and Microsoft are going to be pretty correlated in some way.

When we analyse this, we could work with the daily return investment.

This is a recap of what we talked about at the start of the term,

which is price today over price yesterday minus one.

But it's more convenient to work in log scale,

so like a log price today over price

yesterday because then what we can do is we can write,

essentially, the return over two days.

Price today over price two days ago.

We can write that as the price today

over yesterday plus the log price today over yesterday,

plus log of price yesterday over price two days ago.

In other words, we can model these log returns over

long periods as sums of log-returns over short periods.

Let's calculate the log daily returns

and we're going to do that by taking the stocks data frame,

getting rid of the date.

What this does, essentially,

for each column in the table.

We're going to execute this expression.

We're going to take the values in that column,

divide them by those values lagged by one,

so price today over price yesterday,

and then we're going to take a log of the result.

Then, we'll have a list of these daily log returns.

Then we're going to run this through a table,

so to put together into a table and run into na.omit,

when you shift all data points,

you end up with an empty row somewhere.

We'll just drop that and this is what the distribution of the returns now looks like.

Now, they are correlated with each other,

as we would expect. There are outliers.

Although they're not too badly behaved,

there are also outliers and there are

long tails and that's even after log transformation.

We need to convert observations into quantiles and while preserving these correlations.

What we're going to do is we're going to use pobs to do

that and this is the user or the quantile versions of that.

Let's use a multivariate t-Copula with an unstructured covariance matrix.

That means that this covariance matrix for

the t-copula could be anything, it's positive definite.

We could have used exchangeable,

which would have assumed all pairwise correlations are equal.

That is probably not a good assumption for stocks,

because stocks and coalitions might be a bit different from each other.

With that, we can see pretty well here.

We can also specify maximum likelihood to be used here as opposed to other methods,

and then we simulated from the fitted copula and plotted it.

Here's the fit for the copula.

It's pretty straightforward.

We've refit a copula,

t-copula with these many dimensions and this structure using maximum likelihood.

By the way, if you wanted to get the estimated correlation matrix for this t- copula,

we could use getSigma, the function.

Next, we can take the fitted model and we can simulate empirical quantiles

using our copula and we can extract it from the fit by using this add copula symbol.

Actually, here's a function that really ought to be in

the copula package, even though it isn't.

What it does is basically does the opposite of PR.

You give it the quantiles,

you give it the original data,

and then it maps them back using the quantile function.

I'm not going to get into details here,

but if you want,

you can go through and parse it

out and look into help for each over what each of these does.

But basically, what this does is we take

the simulated returns and we map them through to have the same marginal distribution as

the original returns using the qobs function converted to a table and to

a data framework table and then again set the ticker names and make a pairwise plot.

This is what it looks like, and the idea is we want to

reproduce the structure we see here.

We have some values, strong values here,

but then we also have quite a few outliers out here, which is.

Now, we fit this copula,

but we always have to ask as we do in statistics,

does our model actually fit the data.

In this particular case,

we're using the non-parametric copula.

We're not assuming any distribution for the margins,

but we still assume a certain structure to the correlation.

Let's use goodness of fit testing.

The function that does it is called gofCopula.

It does have some limitations at the moment because, for example,

it cannot handle non-integer degrees of freedom,

so we're going to have to extract them from the fit and round.

Also, unfortunately, it is very time-consuming.

We're going to set the simulation size to 40 in practice.

In real-life applications, you would probably use it

more and you might run into a more powerful computer.

But copula methods are pretty modern methods.

As one might expect it, pretty computationally-intensive.

What does this line of code do?

What it does is, first of all,

it defines a copula

with five variables unstructured,

and its degrees of freedom for the t-copula are going to be

extracted from the copula with parameter number 11 that

turns out to be the degrees of freedom,

except it's rounded and also we'll set the df.fixed as true.

Basically when we try to fit this copula,

it will not try to fit degrees of freedom.

Then, returns is our raw data,

and N equals 40,

that's the simulation size and estimation method of maximum likelihood.

What do we see? We see a parametric bootstrap goodness-of-fit test.

Method equals Sn,

there are others that are possible,

but for now the default one is decent.

Ultimately though we end up with a p-value of one half,

which means that we're probably okay.

That means that we're probably representing the correlation

between the stock returns decently well.

Now, let's talk about how we might simulate the different portfolio strategies.

Now, we have the simulated returns,

so let's play around with those.

First let's talk about the equally weighted portfolio.

That's pretty straightforward.

These are the log returns,

so we'll just exponentiate them to get the raw returns.

Since each stock has equal weight,

we'll just take the mean across each row,

and that'll give us the equally weighted portfolio return.

We'll use row means for that and subtract one.

If the stock gains 10 percent then this is going to be 1.1,

so we subtract one to get the return,

then get the mean return and the standard deviation of the returns of daily.

What does that mean?

Well, suppose that we invest one dollar and split it equally among the stock,

so put in 0.20 cents into each stock.

Then the average daily return will be about this much,

that is the mean, and its standard deviation will be about this much.

We could also ask other questions.

For example, we could look at what fraction of these returns are less than zero.

In fact, we would expect to lose money by 48 percent of the days.

If we're worried about losing a lot of money,

we'd expect to lose more than one percent on about 12 percent of the days.

We can also plot the density of these daily returns.

Here is the ggplot way of doing it.

We use the qplot function of the returns and we use the geometric shape density,

and I set a label for daily portfolio return.

One thing I wanted to mention before I move on is that,

I don't provide code for these figures in this file,

but they are in fact in the R Markdown file that you could

download and you can see how you can embed these types of

mathematical expressions that are computed using R into the text.

Can we do better than that?

The stocks are correlated,

so maybe you can try hedging.

Can we, for example, get the return that we got from this portfolio more reliably.

What we will do is we'll use constrained optimisation and that's where

that package that we loaded at the start,

the Rsolnp, that's where that's going to come in useful.

This is not examinable,

this is not a course on optimisation,

this is a course on something else, multivariate analysis.

What we're doing here is,

we're going to essentially create an objective function,

which will compute the standard deviation

of the returns as a function of the stock weights.

What the sweep does is,

basically takes the simulator, determines,

exponentiate them, and then it multiplies each column,

or multiplies each column by W,

which is given here,

and then subtract one to get the return.

Then it gets passed here and it's used there.

This function is basically the difference between

the return for under the weight is W,

and the equally weighed portfolio,

which we've computed back here.

This is a bit counter-intuitive,

the other element we have here is sum of W minus one.

The reason is that this EQ function is actually a function for constraints.

It specifies which of

the functions of the data this optimisation we'll have to keep at zero,

and in particular, we want the return for under

our scheme to be equal to the return under the equally-weighted scheme.

That difference has to be zero.

Also, we want the sum of the weights minus one to equal zero,

that is the sum of the weights to equal one.

Then we use the function gosolnp,

Gosol this nonlinear program,

give it the objective function,

the standard deviation,

which it will try to minimise.

Give it the EQ function,

that is the function of things that should equal to zero,

and actually tell it that these have to equal to zero.

Then we'll say the lower bound for our parameters, well, they are proportion,

so there must be at least zero and the upper bounds must be at most one.

N.sim I think is a helper argument.

What parameters do we get?

Well, it does some iteration and finds a solution that's happy with,

and the pars is the parameters that gives you the best weights it can get.

We are going to attach the symbol names to them

and also get the best returns using the rets function here.

Get the returns for these weights.

Here are the weights, what can we see here?

Well, Microsoft's weight is almost zero,

IBM's weight is about 13 cents of the dollar.

The idea is that because they are so correlated,

it only makes sense to take one of them.

It doesn't really give you any additional hedging to take both.

The average daily return is still the same as it was before.

Again, if you look at the R Markdown file,

it'll show you how that's computed.

The standard deviation is now a bit smaller

and the probability of loss have decreased significantly,

in particular, the probability of a major loss,

has decreased by about 1/3.

One interesting, perhaps slightly counter-intuitive aspect of

this is that IBM got picked over Microsoft.

Not quite sure why that happened,

but I think that it might be because of differences in volatility,

although I'm not sure.

That concludes this demonstration.

Again, I would encourage you to work through it on your own,

maybe try different numbers,

different combinations of these,

and see if you can get a different result.

Again, this portfolio optimisation task is not examinable.