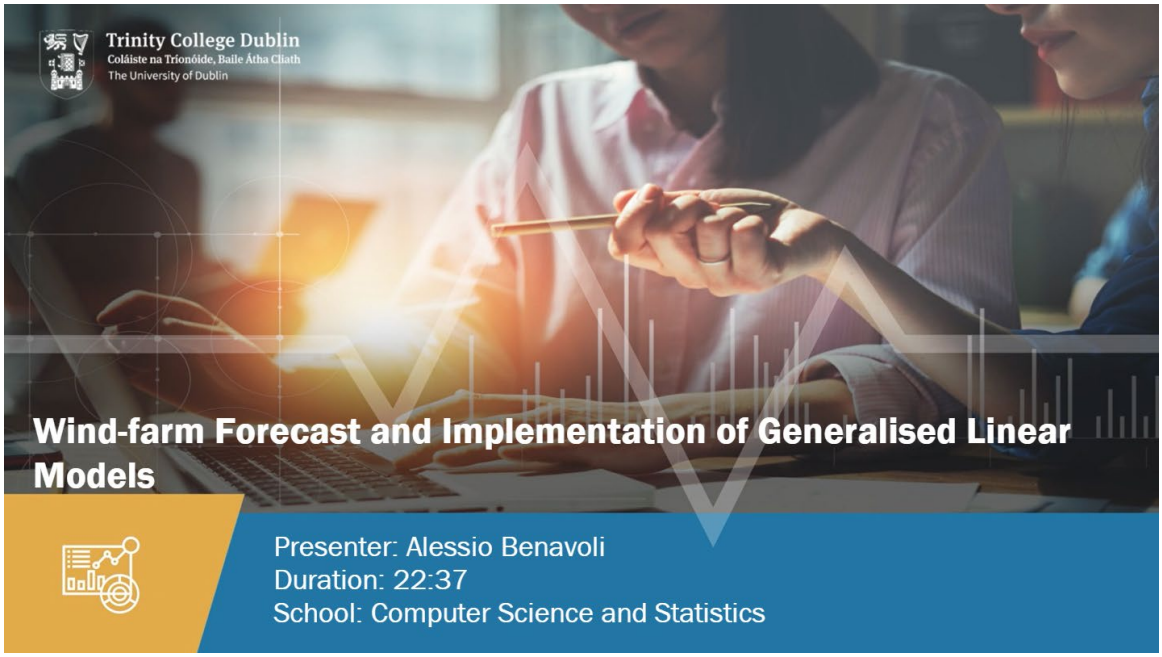
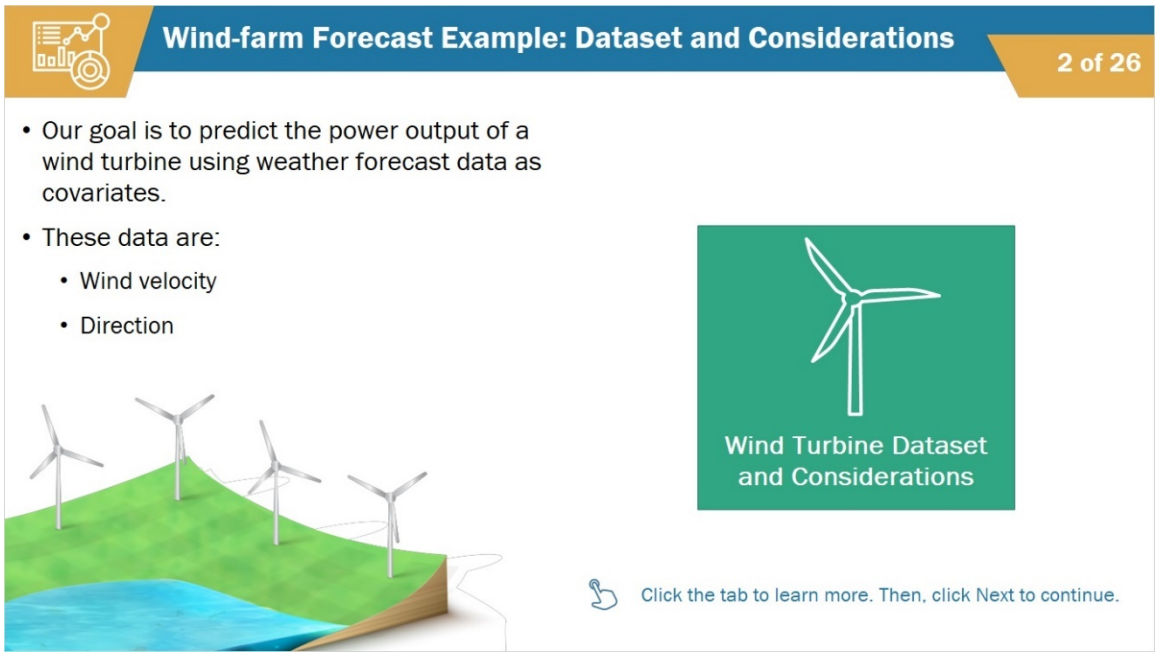
Wind-Farm Forecast and Implementation of Generalised Linear Models

Slide 1: Introduction



My name is **Alessio Benavoli** and I am **the instructor for this session.** In this session, we apply generalised linear models to the problem of wind power forecasting. This problem cannot be formulated as linear regression, or as logistic regression, or as any other generalised linear model implemented in the R function "glm". For this reason, we will derive the model from scratch both theoretically and practically in R. This will allow us to address the wind power forecasting problem but also to understand (indirectly) what is inside the "glm" function in R.

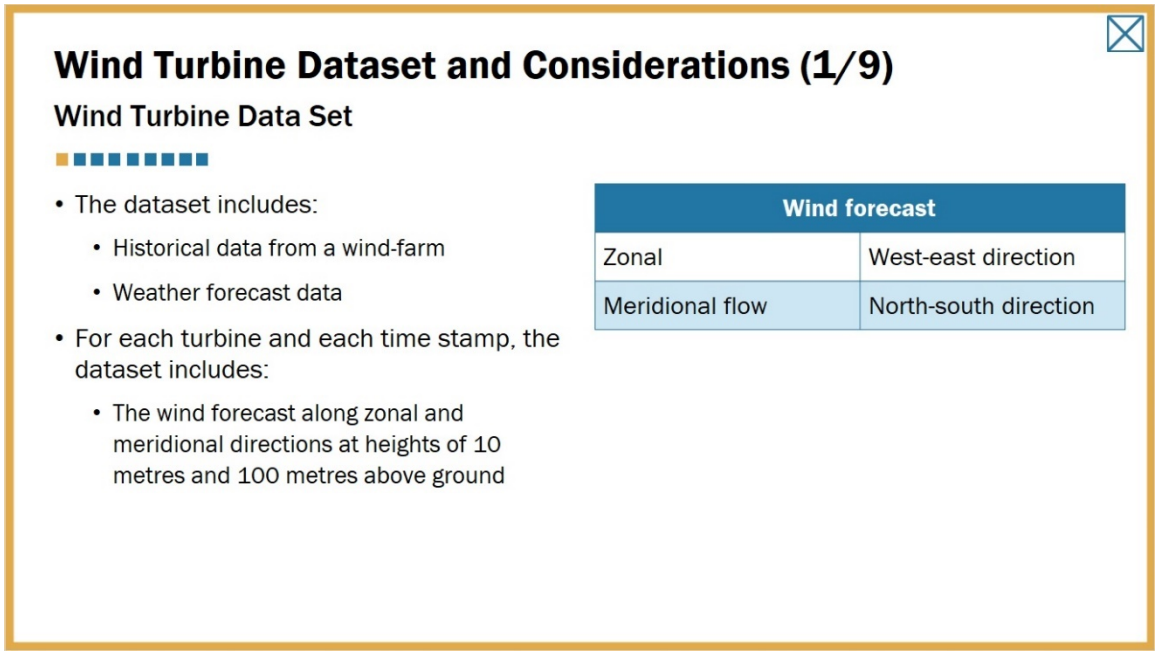
Slide 2: Wind Farm Forecast Example: Dataset and Considerations



In this session, we will face the problem of predicting the power generated by a wind farm. More precisely, our Goal is to predict the power output of a wind turbine using weather-forecast data (wind velocity and direction) as covariates.

Click the tab to learn more about this dataset and other considerations.

Tab 1: The Wind Turbine Dataset and Considerations

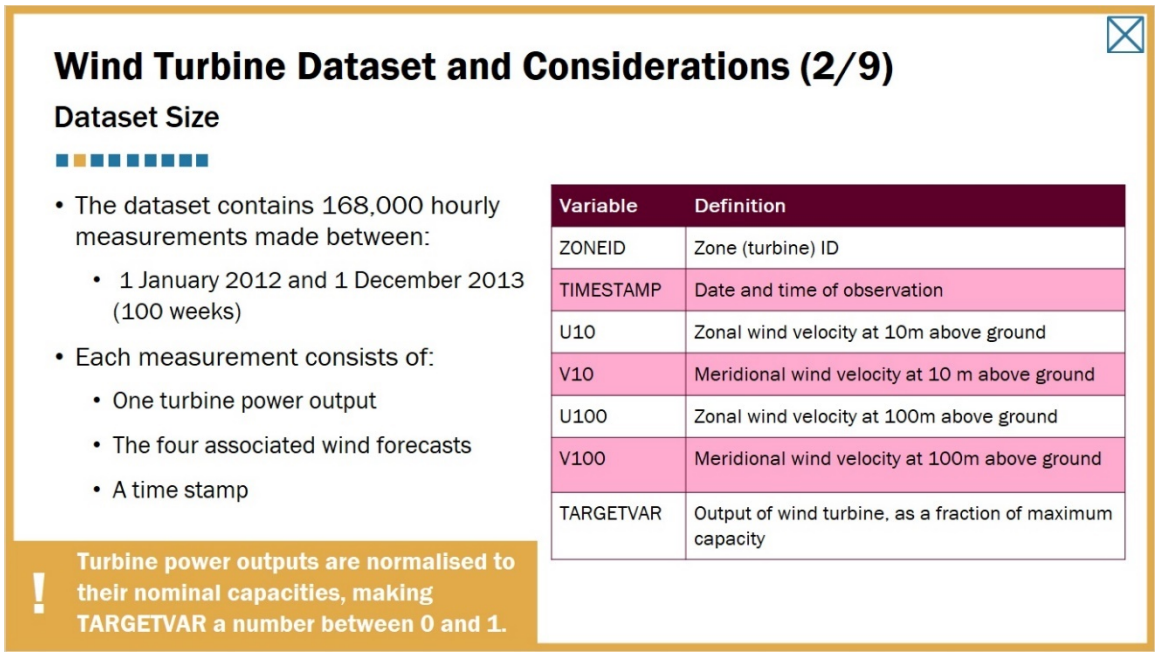


We will use a dataset which includes historical data from a wind-farm, together with weather-forecast data.

The data also includes the wind forecast (delivered by a Weather-forecast provider) along the zonal and meridional directions at heights of 10 and 100 meters above ground (four measurements) for each turbine and for each time stamp.

Note that Zonal denotes the west-east direction and Meridional flow denotes the north-south direction.

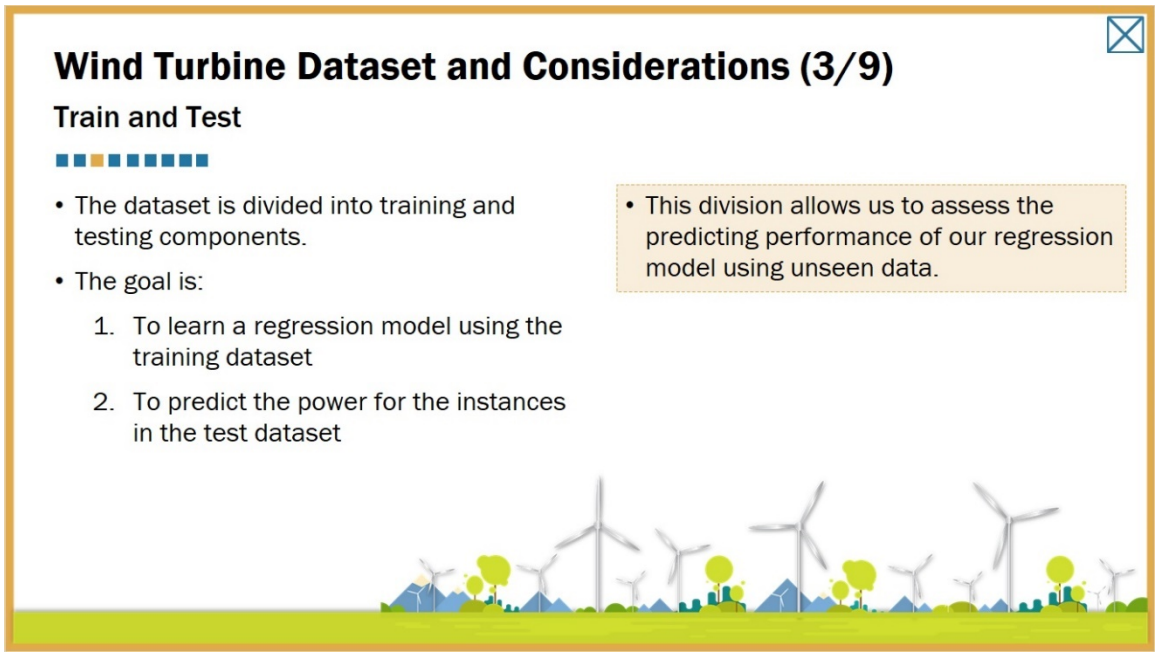
Tab 1.1: Dataset Size



The data set contains 168,000 hourly measurements made between January 1, 2012 and December 1, 2013 (100 weeks). Each measurement consisted of one turbine power output, plus the four associated wind forecasts, plus a time stamp. You can see the variables in the table u10 and u100 correspond to the zonal component of wind at 10m and, respectively, 100m. v10 and v100 correspond to the meridional component of wind at 10m and, respectively, 100m.

The TARGET variable is the produced power. Note that, the turbine power outputs are normalized to their nominal capacities, making TARGETVAR a number between 0 and 1.

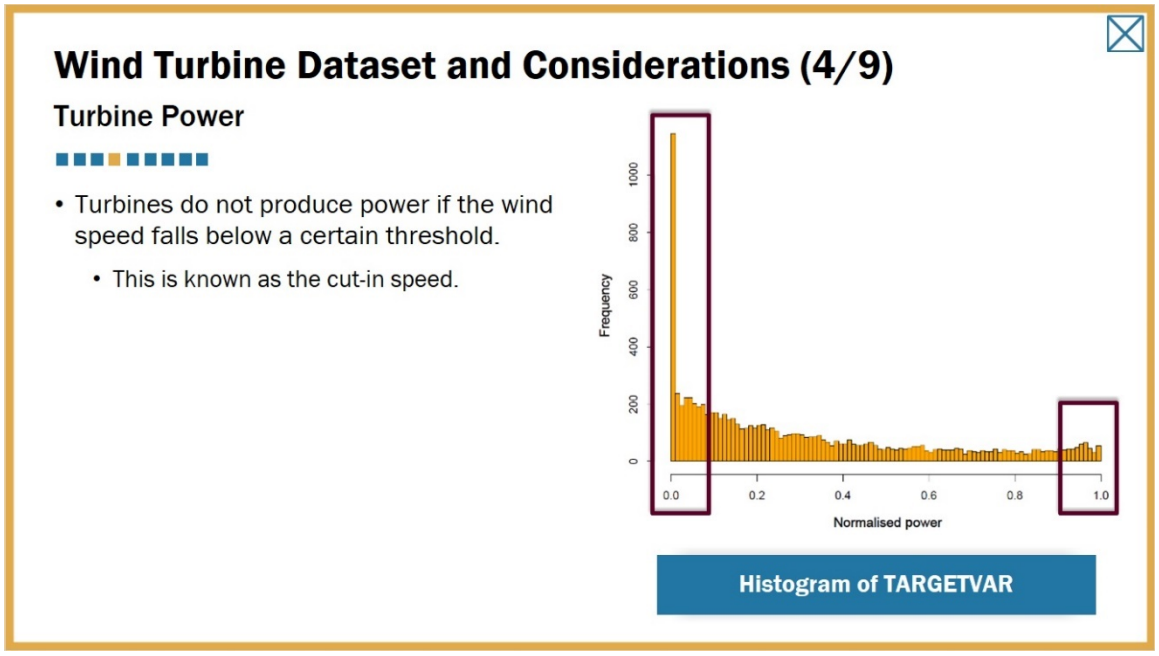
Tab 1.2: Train and Test



The dataset is divided into training and testing components. Our goal is to learn a regression model using the data in the training dataset and, then, predict the power for the instances in the test dataset.

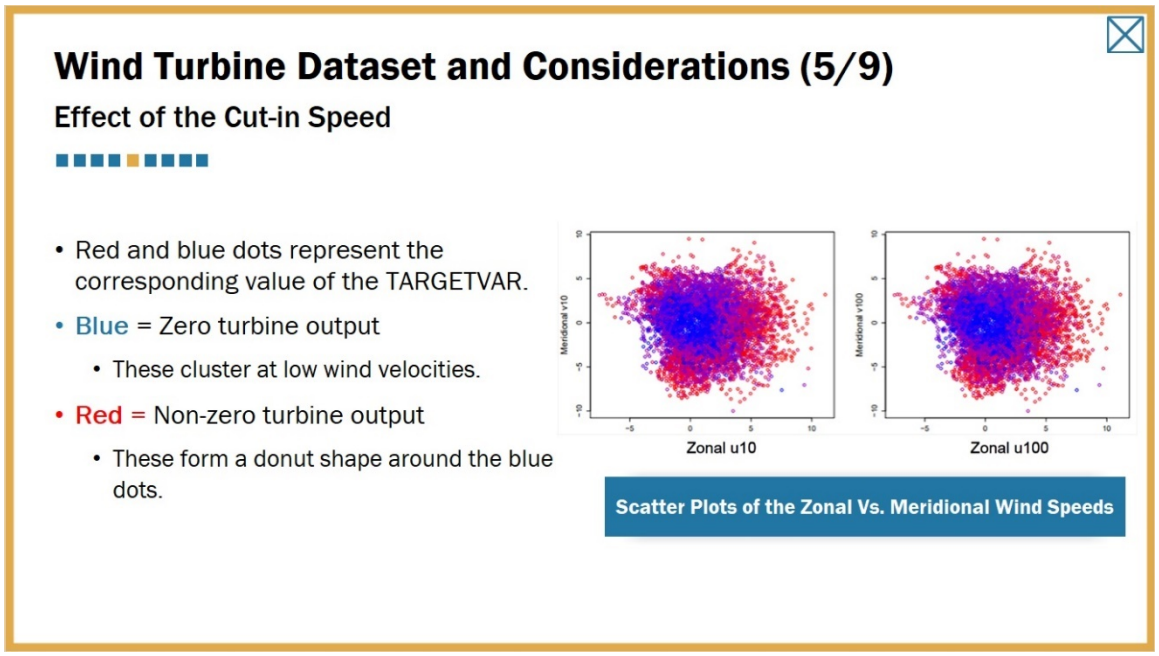
In this way we can assess the predicting performance of our regression model using unseen data, that is data were not used during fitting (that is computing MLE).

Tab 1.3: Turbine Power



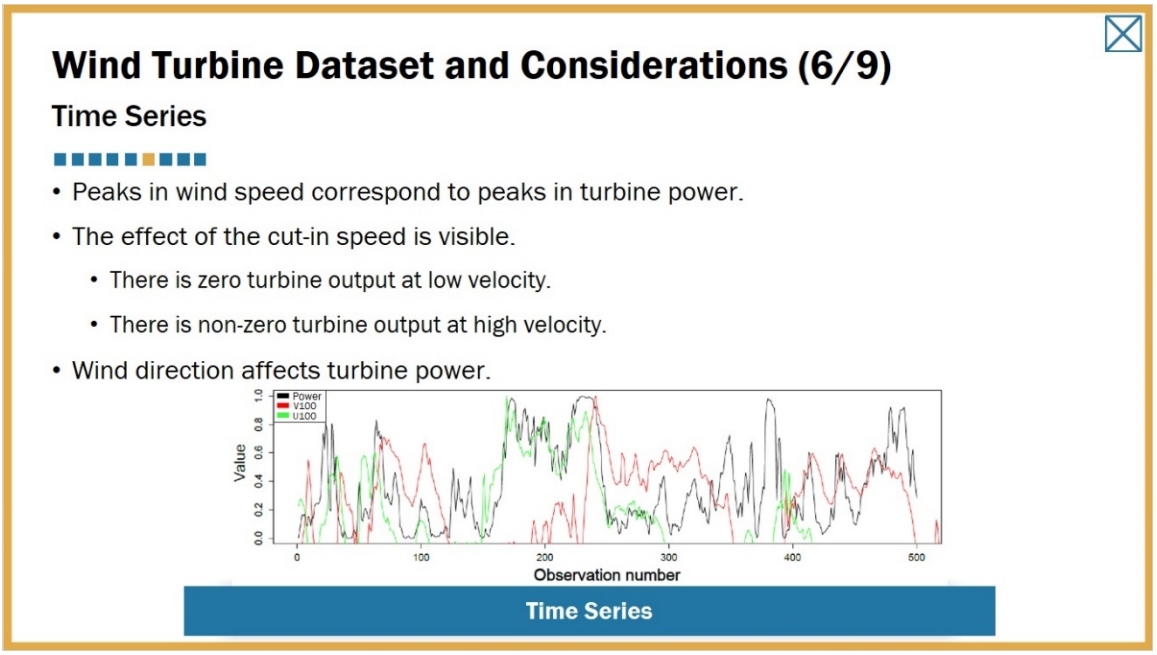
The histogram shows two characteristics of this dataset: a large spike near 0, and a bump near 1. The bump corresponds to turbines working near full capacity. The large spike corresponds to low wind velocities. Turbines won't produce any power if the wind speed falls below a threshold known as the *cut-in speed.*

Tab 1.4: Effect of the Cut-in Speed



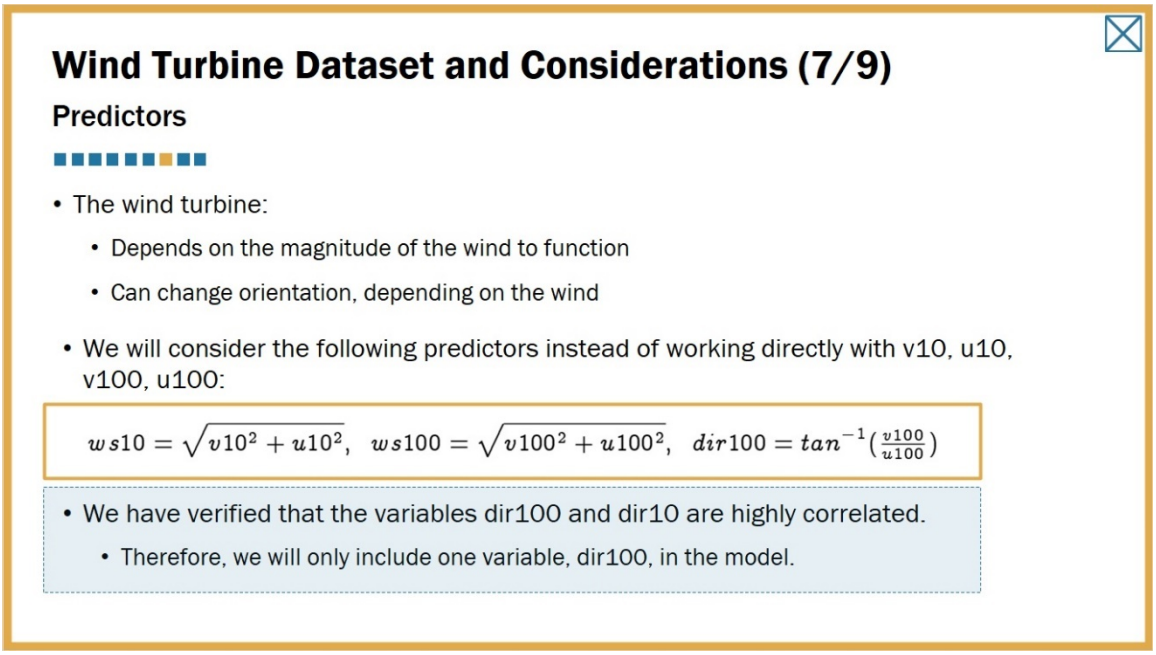
These are the scatter plots of the zonal versus meridional wind speed at 10meters (left) and 100meters (right). The colours represent the corresponding value of the TARGETVAR: blue dots correspond to zero turbine output and cluster at low wind velocities. Red dots correspond to non-zero turbine output and form a donut shape around the blue ones.

Tab 1.5: Time Series



Peaks in wind speed correspond to peaks in turbine power, and the effect of the cut-in speed is also visible (periods of zero turbine output associated with low, but non-zero wind speed). Note also the effect of the wind direction, that is the effect of the zonal (u100) or meridional wind (v100) on the turbine power. As can be seen, a strong wind corresponds to a large produced power.

Tab 1.6: Predictors

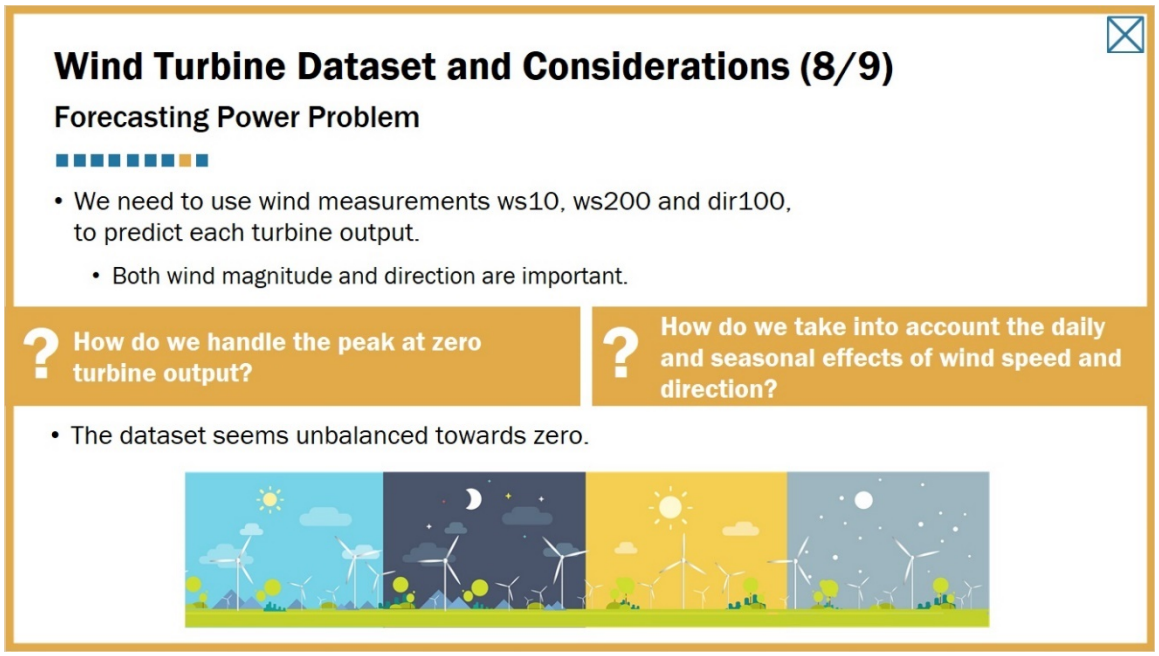


From knowledge about how a wind-turbine works, it is clear that the power mainly depends on the magnitude of the wind. The turbine can change orientation depending on the wind. So, we will consider the following predictors:

ws10 which is magnitude of the wind at 10m, ws100 which is magnitude of the wind at 100m, and dir100 which is the direction of the wind at 100m, instead of working directly with v10, u10, v100, u100.

We consider only dir100 and not dir10, because we have verified that these two variables are highly correlated. So, we should only include one of them in the model.

Tab 1.7: Forecasting Power Problem



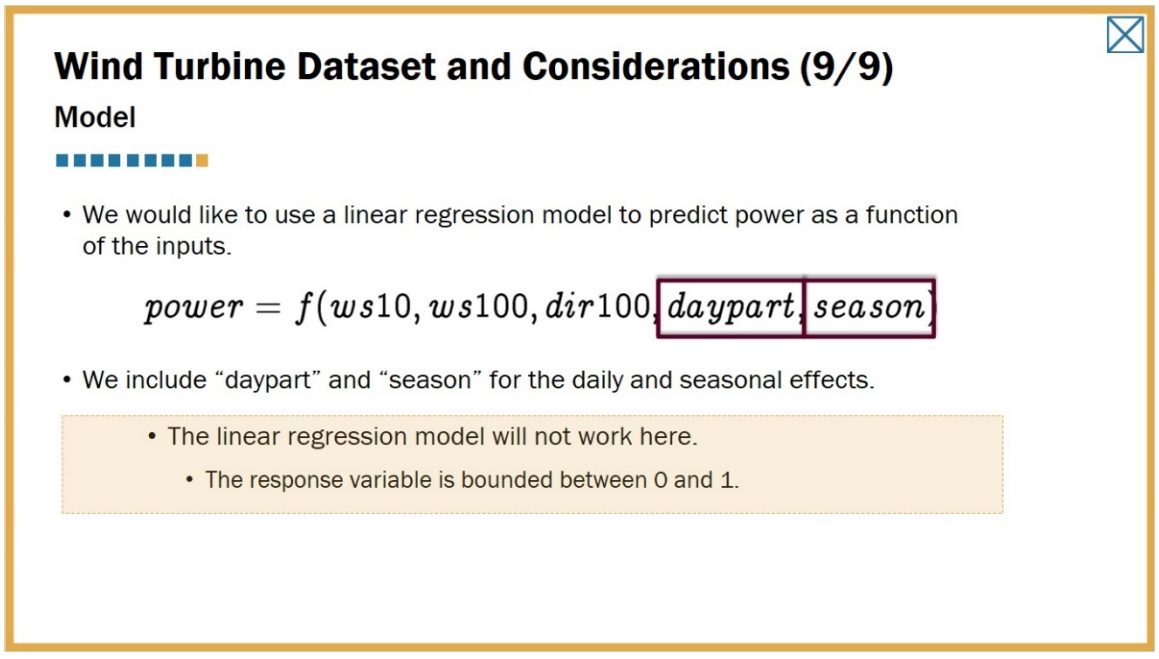
We need to use wind measurements (*ws10, ws100, dir100)* for predicting each turbine output (because both magnitude and direction are important).

But this may cause difficulties because of the following aspects:

How should we handle the peak at zero turbine output? It seems the dataset is unbalanced towards zero. This may be a problem for our regression model.

Wind speed/direction has daily and seasonal effects (in winter wind is on average stronger), how do we take into account of these effects?

Tab 1.8: Model



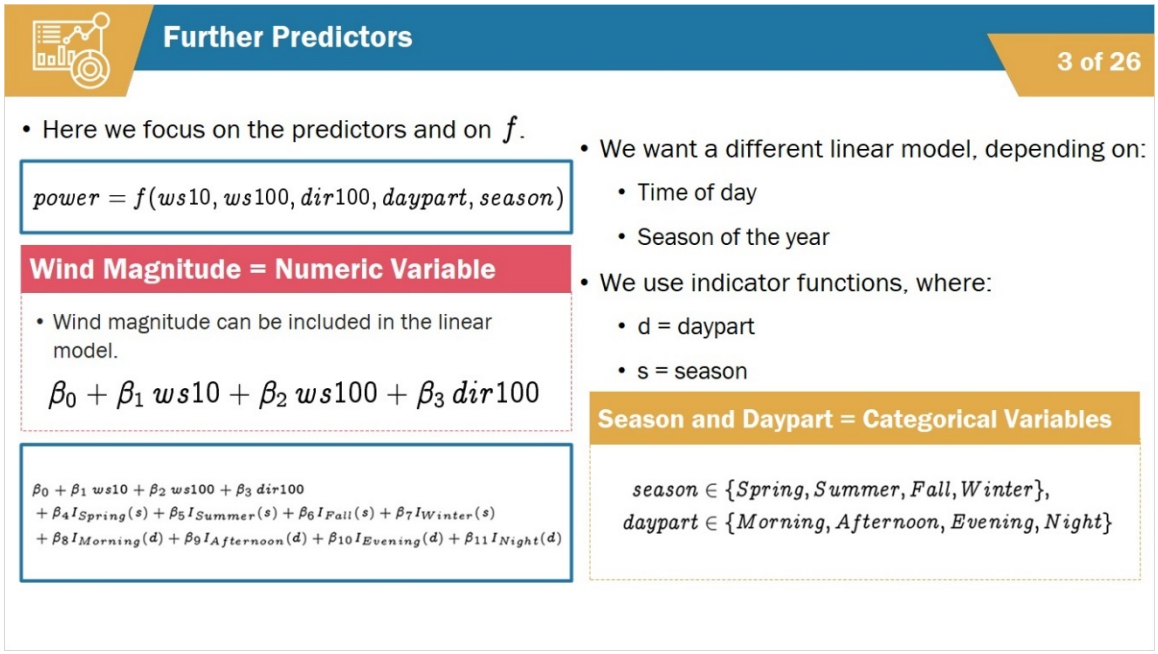
Ideally, we would like to use a linear regression model to predict power as a function of the inputs

power = f (ws10 ; ws100 ; dir100 ; daypart; season)

We have included time (“day part” and “season” to account for the daily and seasonality effects).

The linear regression model will not work here since the response variable is bounded between 0 and 1, we have seen a similar issue in the lesson about logistic regression. Linear regression predictions can be below 0 or above 1 and these predictions do not make sense for this problem (where the response variable is bounded between 0 and 1).

Slide 3: **Further Predictors**



Before discussing the issue with the response variable further, let us focus on the predictors and on f .in

power = f (ws10 ; ws100 ; dir100 ; daypart; season)

The wind magnitude is a numeric variable and so we can include it in the linear model as shown in the top formula *β0 + β1 ws10 + β2 ws100 + _β3 dir100*

while season can be Spring; Summer; Fall; Winter

season ϵ {Spring; Summer; Fall;Winter};

daypart ϵ {Morning; Afternoon;Evening;Night}

daypart can be Morning; Afternoon; Evening; Night

So, they are categorical (discrete) variables. Our goal here is to have a different linear model depending on the time of the day and on the season of the year and so we can use indicator functions, as seen in this last formula:

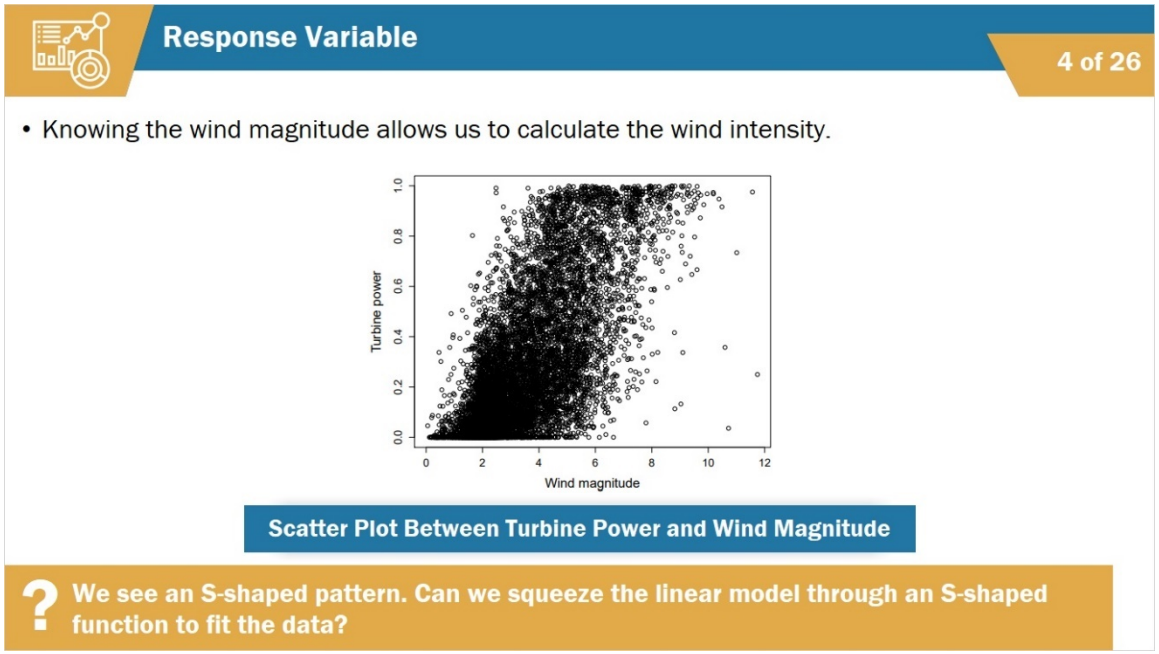
$$\_\beta_0 + \beta_1 \text{ws10} + \beta_2 \text{ws100} + \beta_3 \text{dir100}$$

$$+ \beta_4 I\text{Spring}^{(s)} + \beta_5 I\text{Summer}^{(s)} + \beta_6 I\text{Fall}^{(s)} + \beta_7 I\text{Winter}^{(s)}$$

$$+ \beta_8 I\text{Morning}^{(d)} + \beta_9 I\text{Afternoon}^{(d)} + \beta_{10} I\text{Evening}^{(d)} + \beta_{11} I\text{Night}^{(d)}$$

where d stands for daypart and s for season.

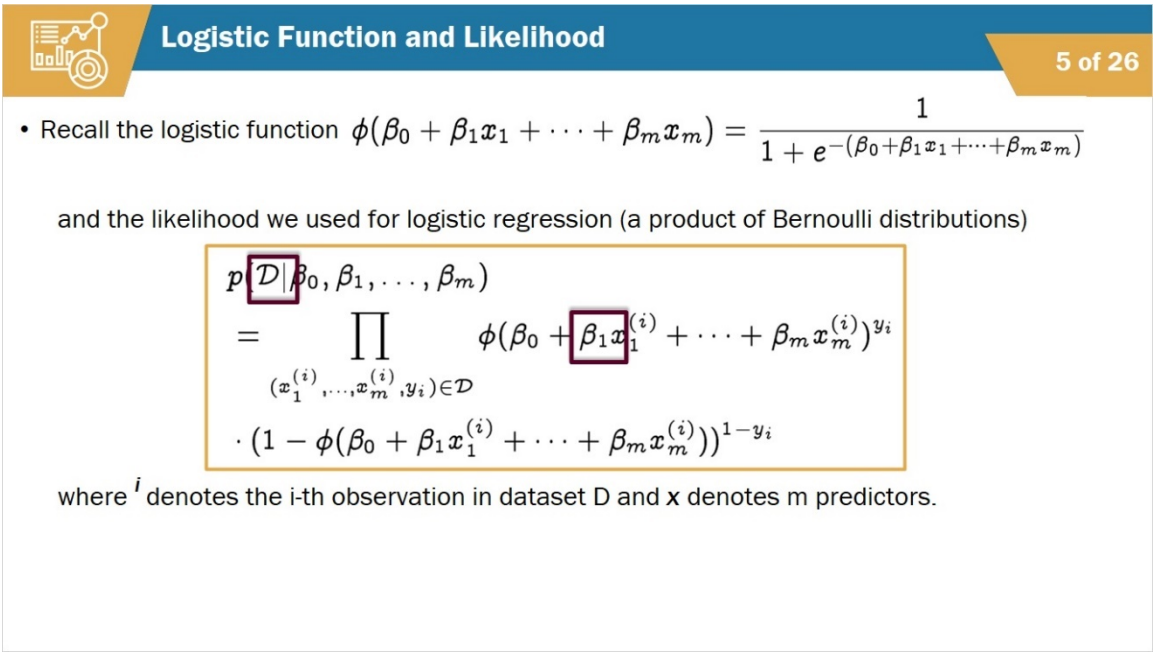
Slide 4: Response Variable



Looking at the scatter plot between turbine power (x-axis) and wind magnitude at 10m (y-axis), where the wind magnitude is defined as the square-root of the sum of the zonal and meridional wind components squared.

We can see an S-shape pattern, so can we use an approach similar to logistic regression to deal with the normalised turbine power? That is, we can try to squeeze the linear model through a S-shaped function to fit the data.

Slide 5: Logistic Function and Likelihood



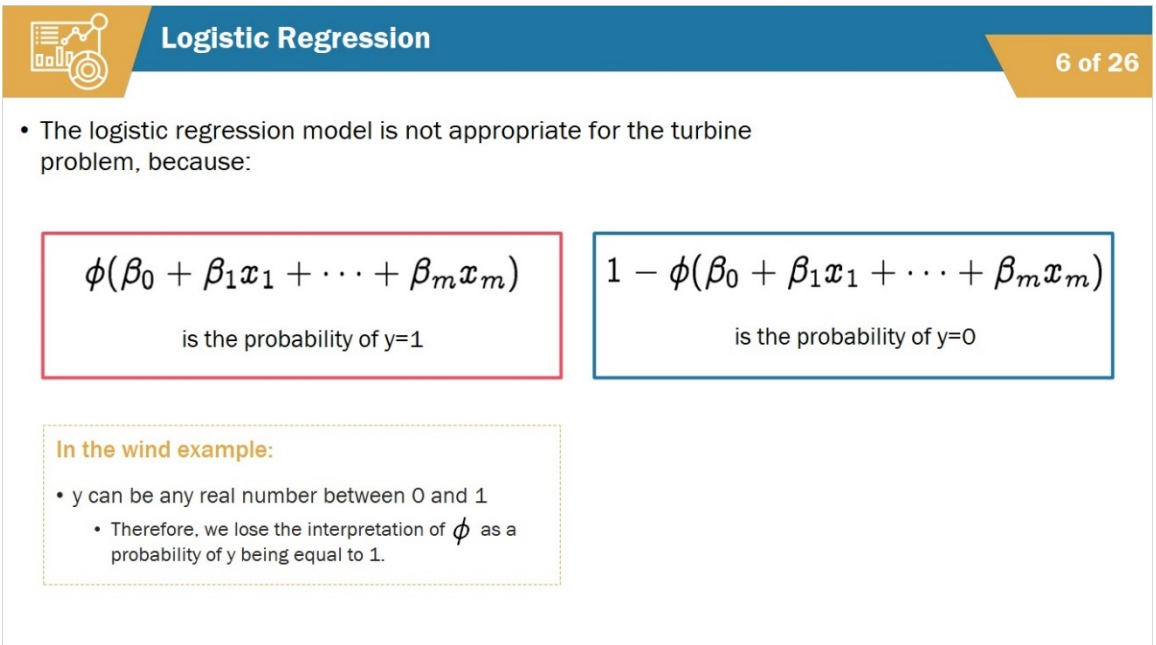
We recall here from the session on logistic regression the logistic function and the likelihood we used for logistic regression, which is a product of Bernoulli distributions, where I used the superscript i to denote the i-th observation in the dataset.

$\Phi(\beta_0 + \beta_1 \beta^{x_1} + \ldots + \beta_m x_m) =$ 1

$1 + e^{-(\beta 0 + \beta_1 x_1 + \ldots + \beta_m x_m)}$

I have assumed we have m predictors denoted with x. In the turbine power prediction problem defined previously, m=11 and the predictors are ws10, ws100, dir100 and the indicators.

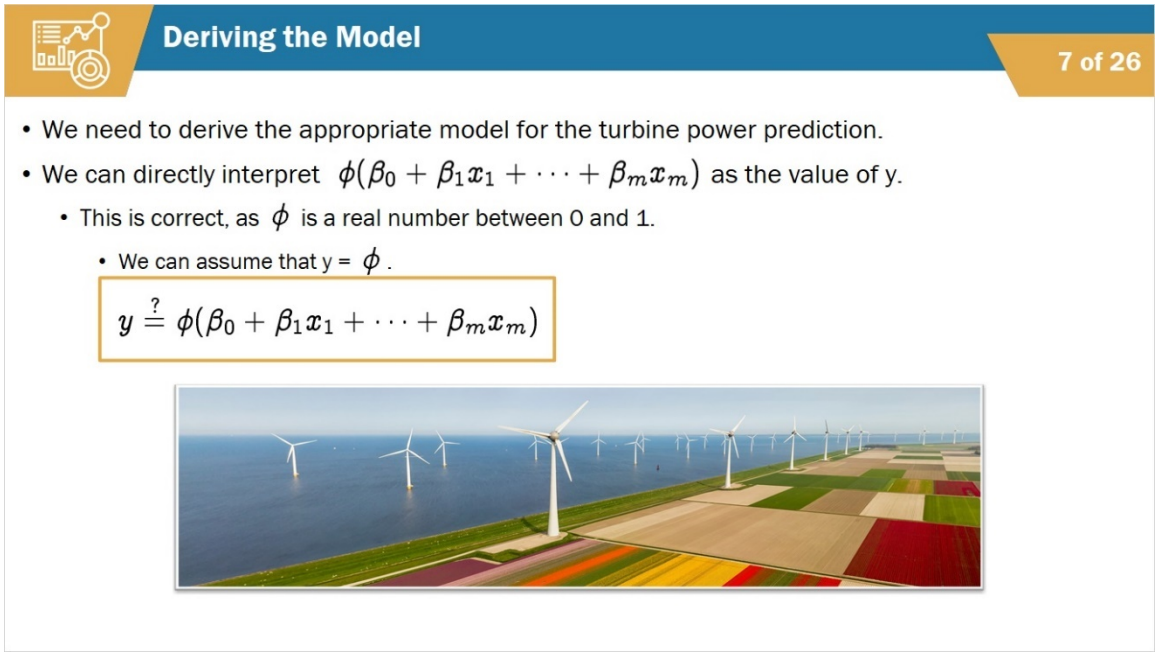
Slide 6: Logistic Regression



Is a Logistic regression model appropriate for the turbine power prediction problem? No, because in logistic regression, we interpreted Phi as the probability that y is equal to one and 1-Phi as the probability that y is zero.

But in the wind example y can be any real number between 0 and 1, and so we lose this interpretation of Phi as a probability of y being equal to one.

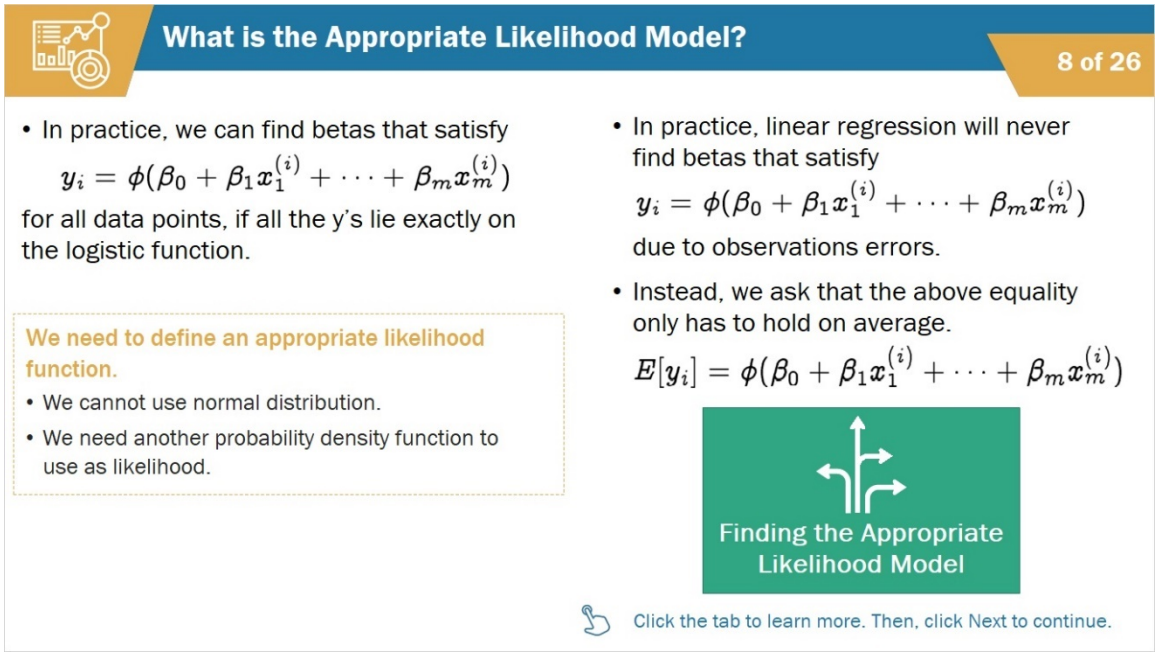
Slide 7: Slide 7: Deriving the Model



We need to derive the appropriate model for the turbine power prediction. We could directly interpret the output of the logistic function, that is the value Phi

as the value of y. This is correct because Phi is a real number between 0 and 1, so we could assume that y is equal to Phi.

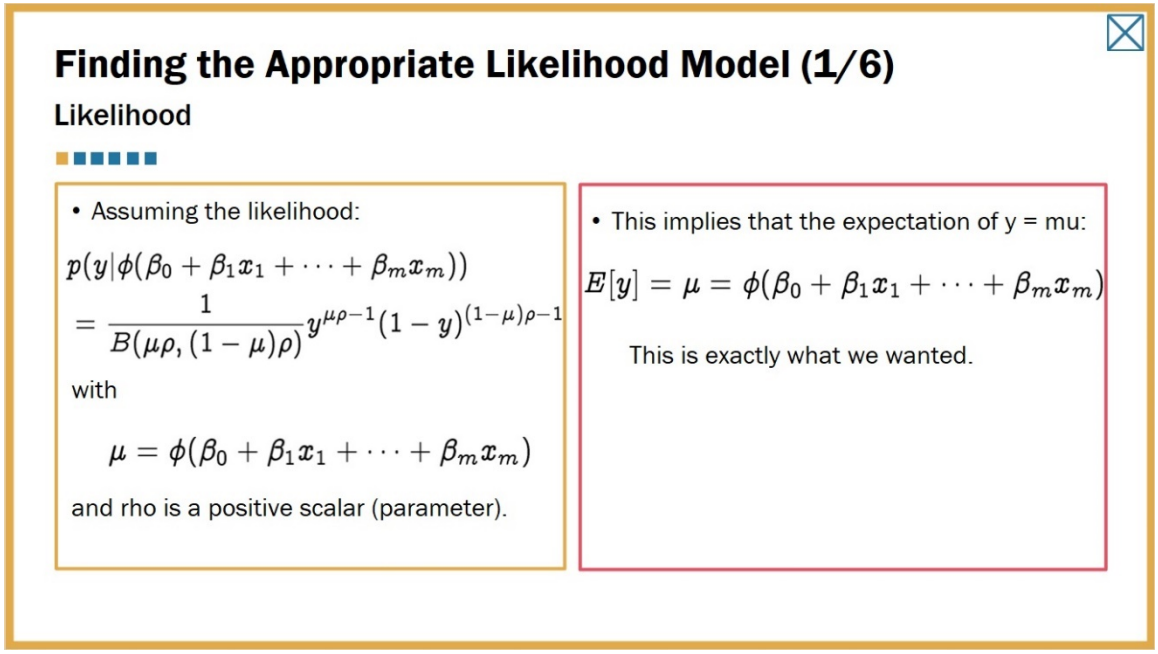
Slide 8: What is the Appropriate Likelihood Model?



In practice, we can find betas that satisfy the set of equations in the slide. Note that we have an equation for each observation "i" in the dataset. These N equations can all be satisfied only if all the ys lie exactly on the logistic function. As for linear regression, this will never happen in practice due to observation errors. So, to account for these errors, we won't ask our model to satisfy the above set of equations. Similar to what was done for linear regression, we will ask the above equality only to hold on average, so that Phi is considered only be the average value of the variable y, as you can see from the formula.

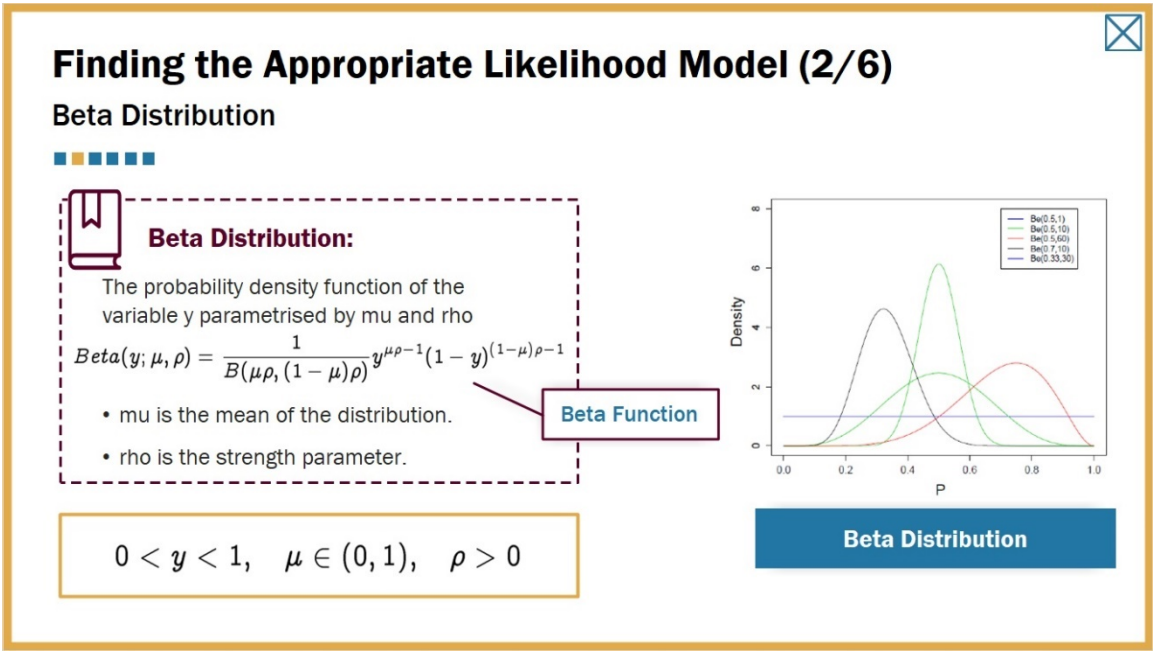
Click the tab to learn more about how we do this. Then click next to continue.

Tab 1: Likelihood



We will do that by defining an appropriate likelihood function. We cannot use a Normal distribution because both y and Phi are numbers between 0 and 1. We need another probability density function to use as likelihood. We will consider the likelihood you see in the first equation on this slide, where mu is equal to the logistic function Phi. This is a probability density function of y, and one can show that the mean of this probability density function is equal to mu, that is the logistic function. This is exactly what we want.

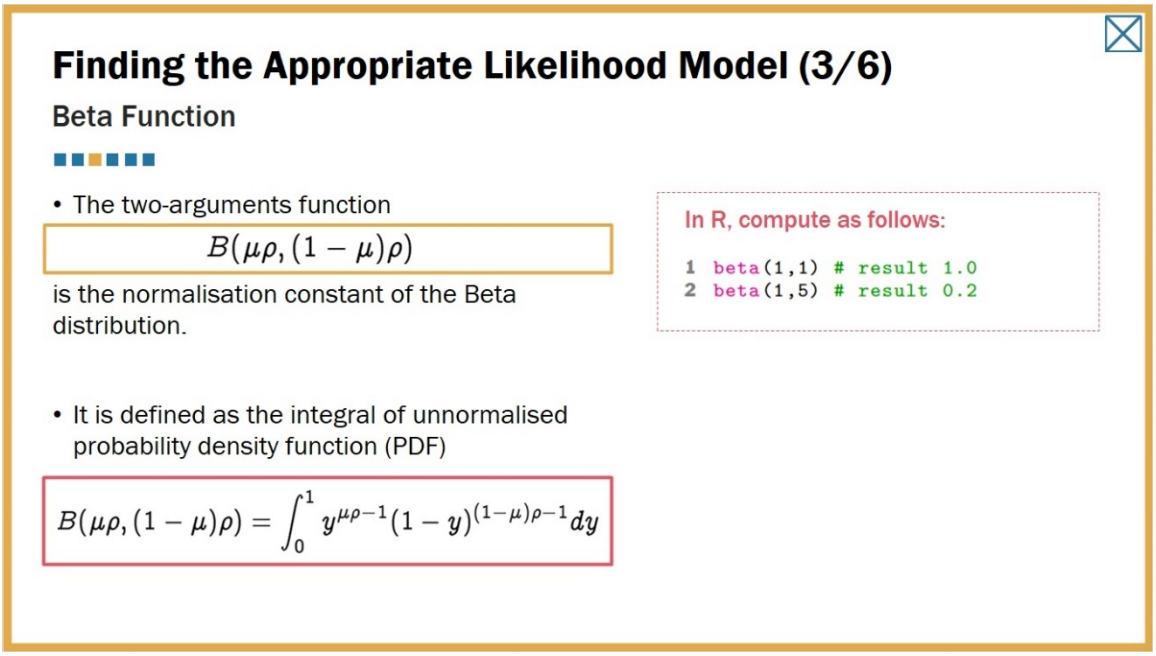
Tab 1.1: Beta Distribution



This probability density function of the variable y parametrized by mu and rho is called Beta distribution. You can see its probability density function in the formula. It has two parameters mu, which is the mean of the distribution, and rho which is the strength parameter, whose domain is defined below. At the increase of rho, the Beta density concentrates more around the mean. As you can see from the plots, comparing for instance the two green lines corresponding to a Beta with the same mu=0.5 and different rho.

This will be our likelihood. What is the denominator in the fraction in the top formula? It is called Beta function.

Tab 1.2: Beta Function



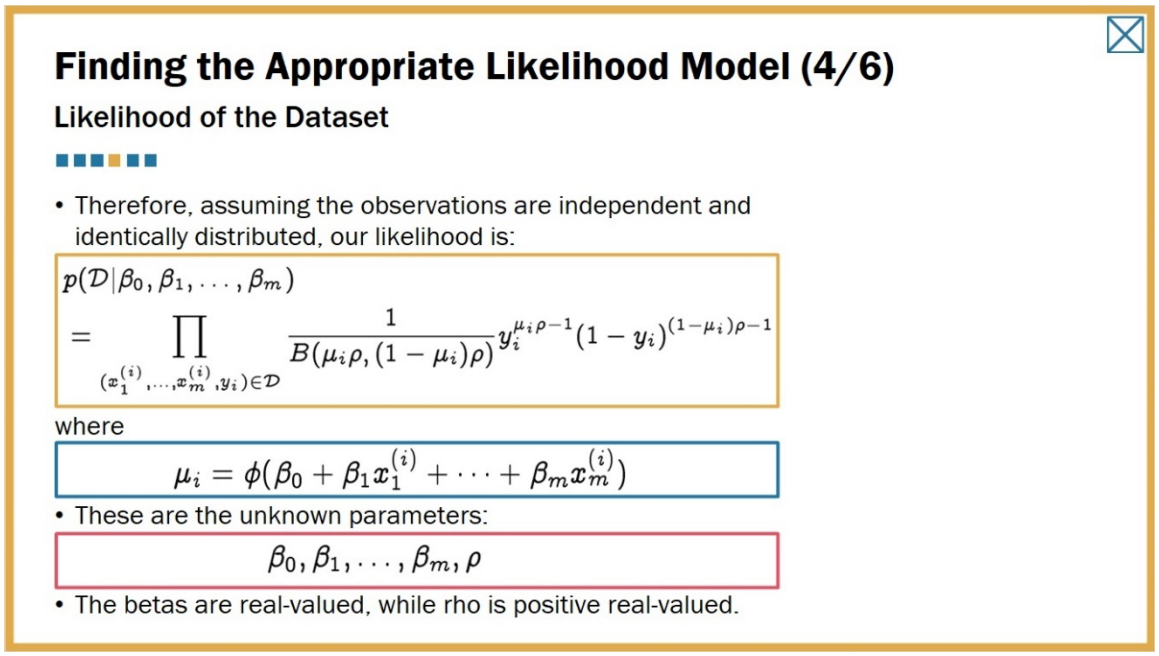
The two-argument function you saw in the previous slide in the denominator of the fraction appearing in the formula of the Beta density and that you see in the first equation here

is the normalization constant of the Beta distribution. It is defined as the integral of the unnormalised PDF.

In R, you can compute calling the function “beta” as you can see in the slide.

```
1 beta(1,1) # result 1.0
2 beta(1,5) # result 0.2
```

Tab 1.3: Likelihood of the Dataset

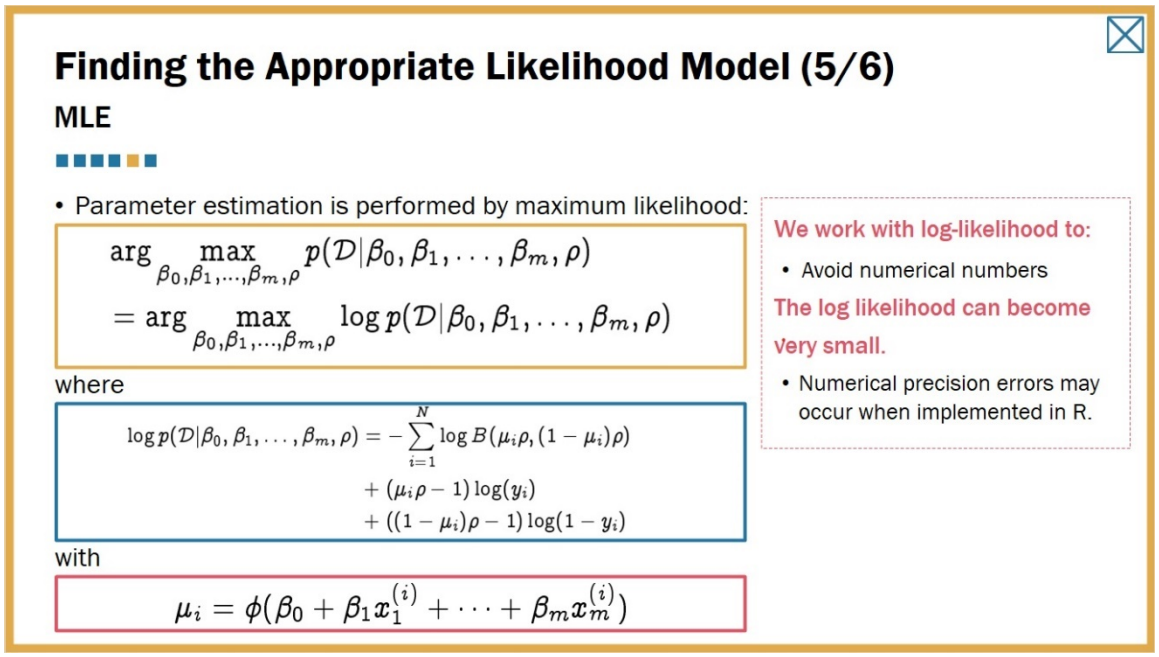


Therefore, assuming the observations are independent and identically distributed, our likelihood is a product of Beta density, with one term for each observation in the dataset. The mean mu depends on i and it is equal to the logistic function Phi we use to squeeze the linear model into 0 and 1.

The unknown parameters for this regression model are the Betas and rho.

The betas are real-valued, while rho is positive real-valued.

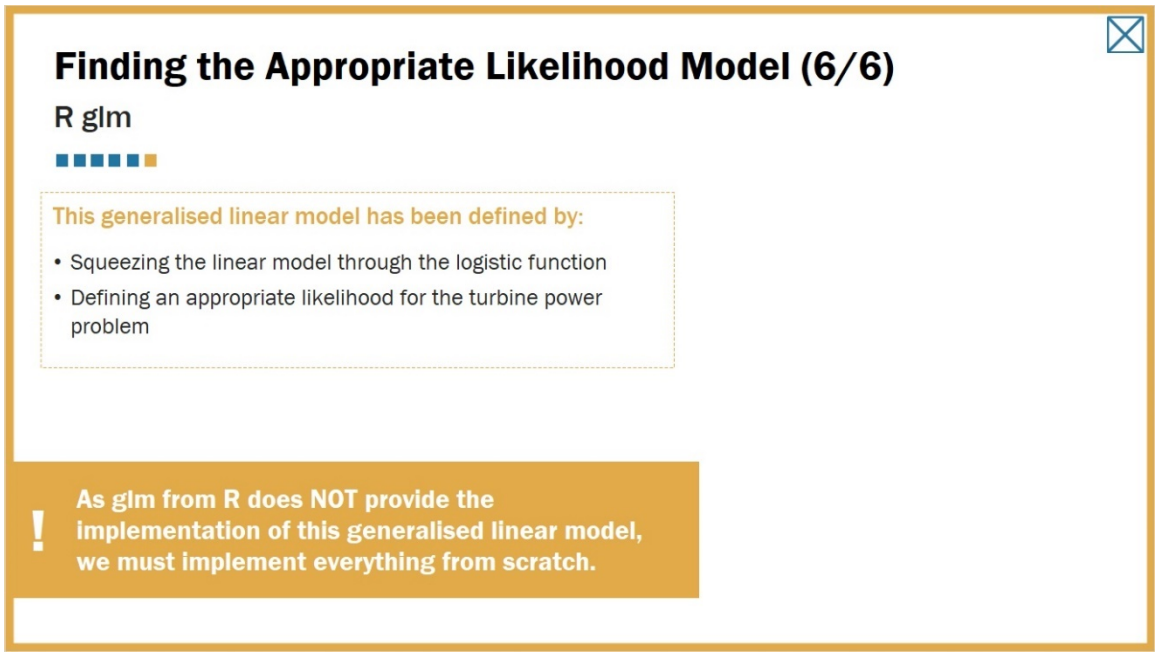
Tab 1.4: MLE



Parameter estimation is performed by maximum likelihood, that is we want to find the values (called argument) of the betas and rho which maximize the likelihood. This is equivalent to maximizing the log-likelihood. We work with the log-likelihood to avoid numerical problems. The likelihood can become very very small and so we can have numerical precision errors when we implement it in R.

The expression of the log-likelihood is given in the second formula.

Tab 1.5: R glm

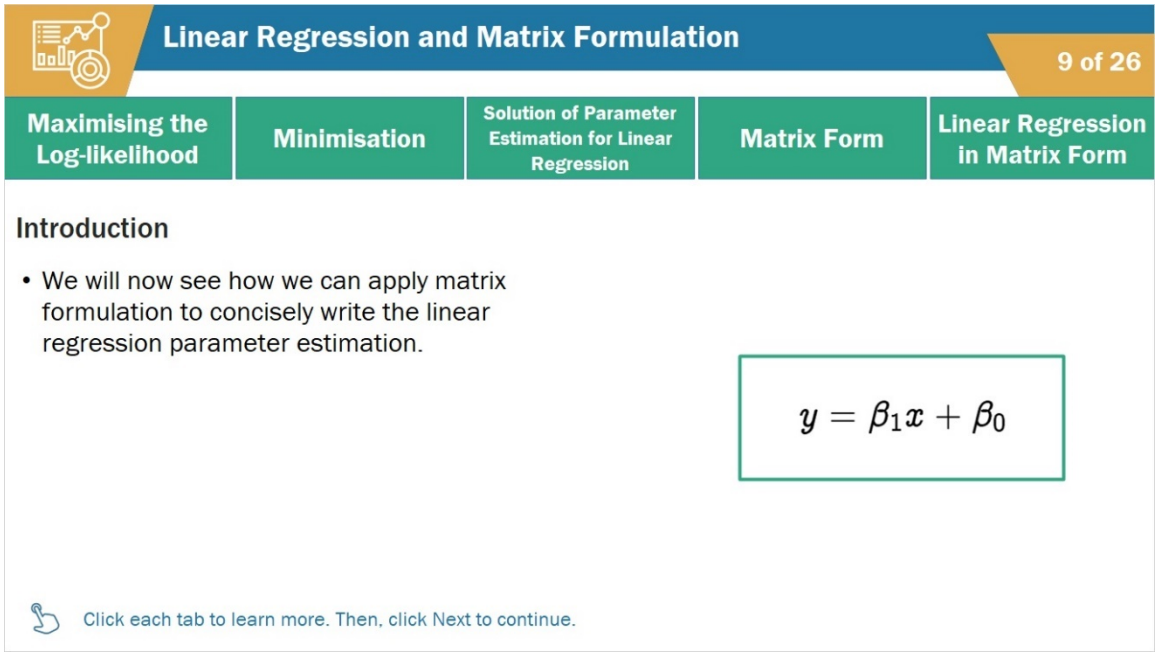


We have just defined another generalised linear model, by squeezing the linear model through the logistic function and by defining an appropriate likelihood for our turbine power problem.

Does glm from R provide the implementation of this generalised linear model? No, it doesn't.

What do we do then? We can implement everything from scratch.

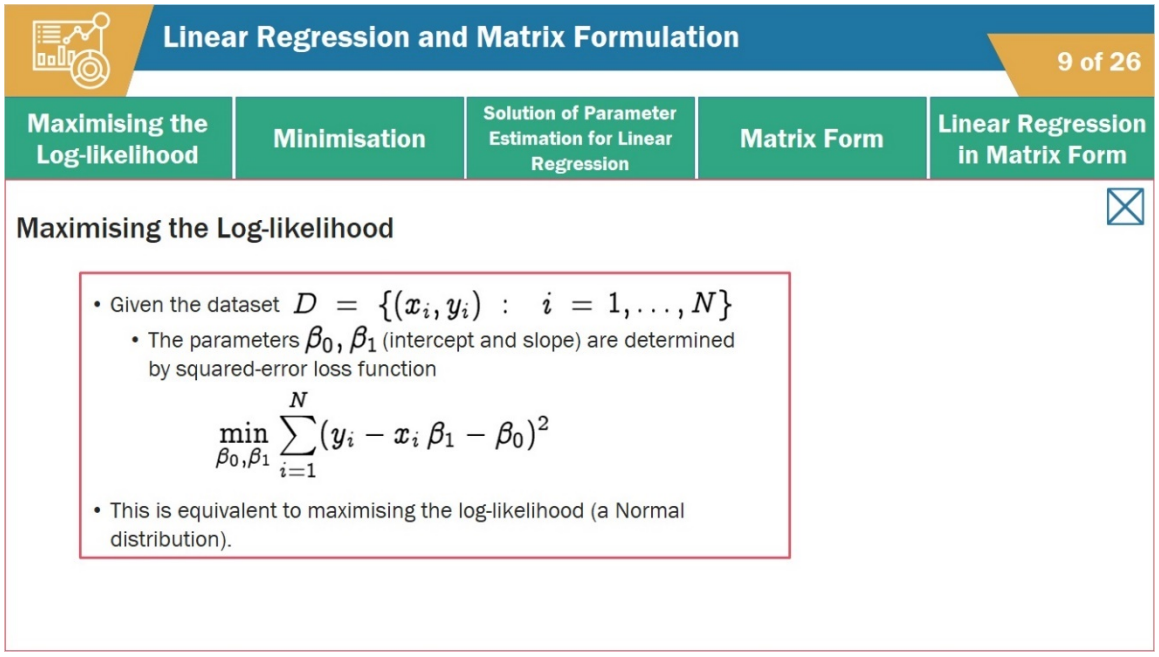
Slide 9: Linear Regression and Matrix Formulation



In a previous session we looked at linear regression. Let's recall what we did for standard linear regression and how we can apply the matrix formulation to compactly write the solution of linear regression parameter estimation.

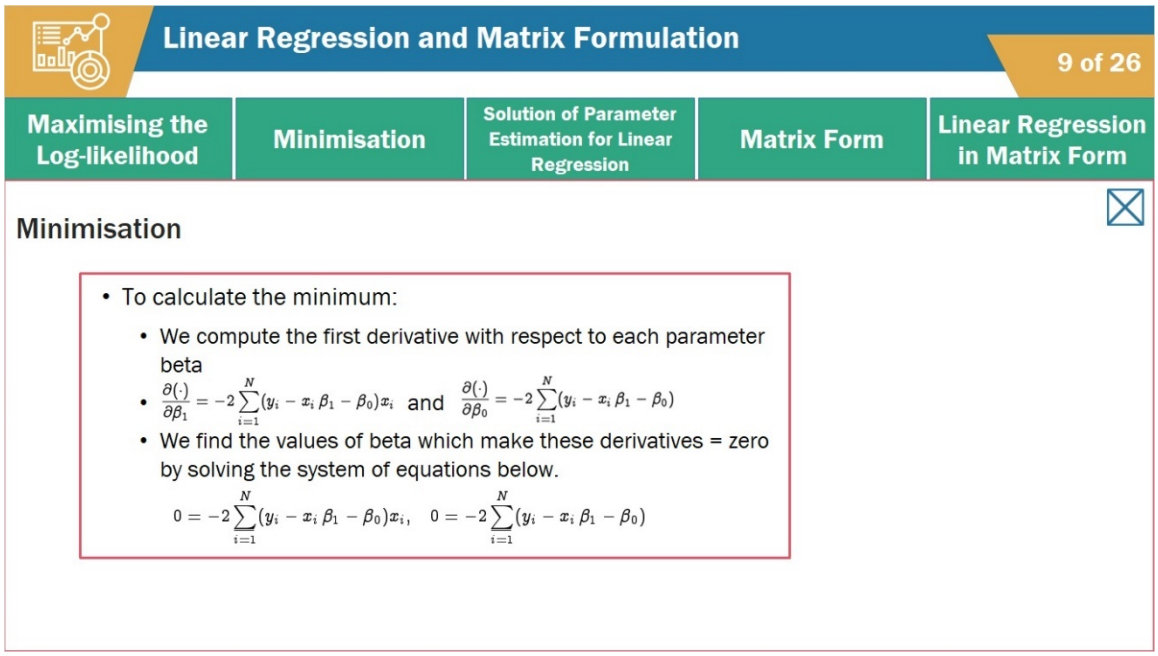
Click each tab to learn more. Then click next to continue.

Tab 1: Maximising the Log-Likelihood



In linear regression, given the dataset D, the parameters beta (intercept and slope) are determined by minimising the squared-error loss function which is equivalent to maximising the log-likelihood (which is a Norman distribution).

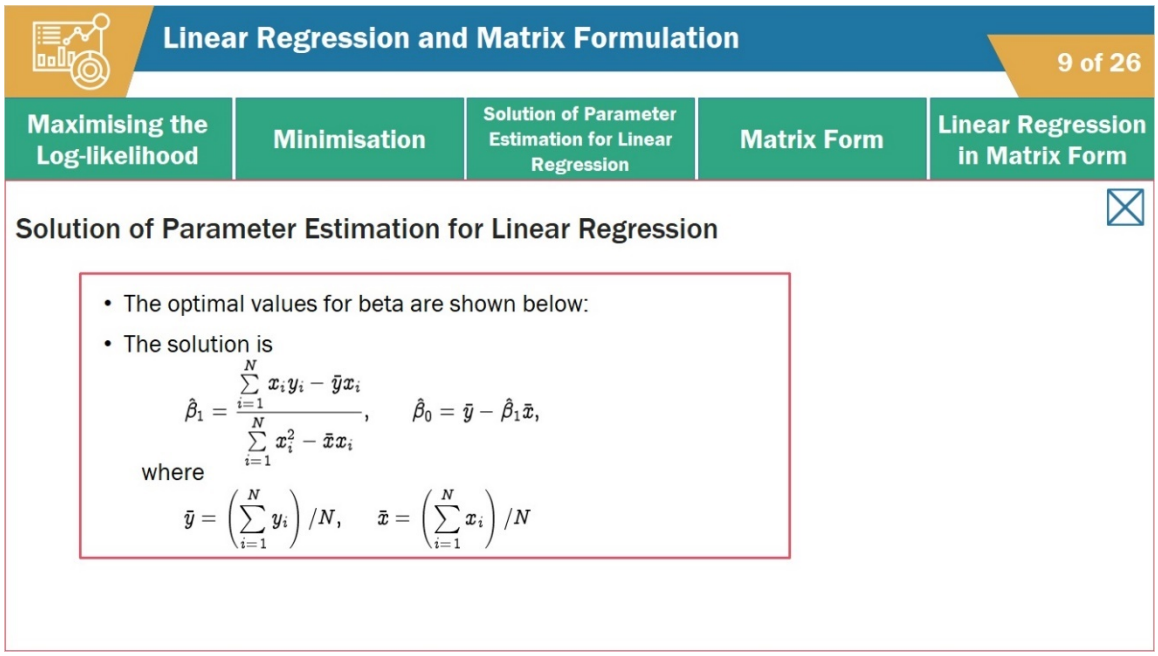
Tab 2: **Minimisation**



To compute the minimum we compute the first derivative with respect to each parameter alpha,beta. The derivatives are shown in the first two equations in the slide.

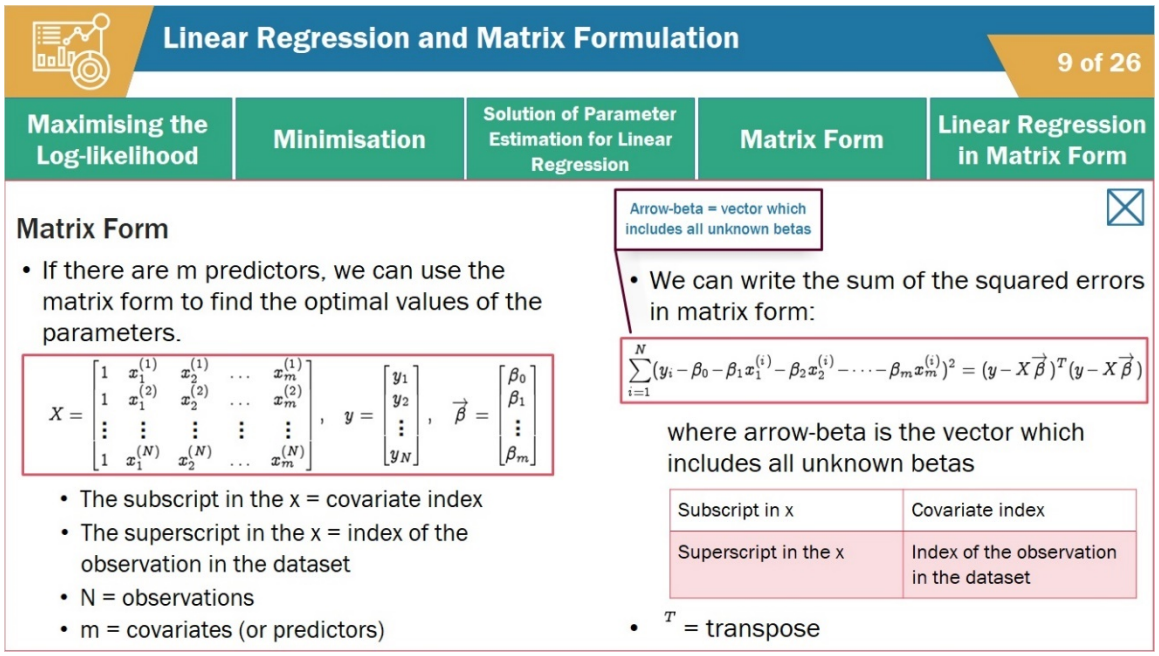
To find the minimum, we look for the values of alpha,beta which make the above derivatives equal to zero. In linear regression, we can find these values analytically by solving the system of equations:

Tab 3: **Solution of Parameter Estimation for Linear Regression**



The optimal values for beta are shown in the slide.

Tab 4: **Matrix Form**

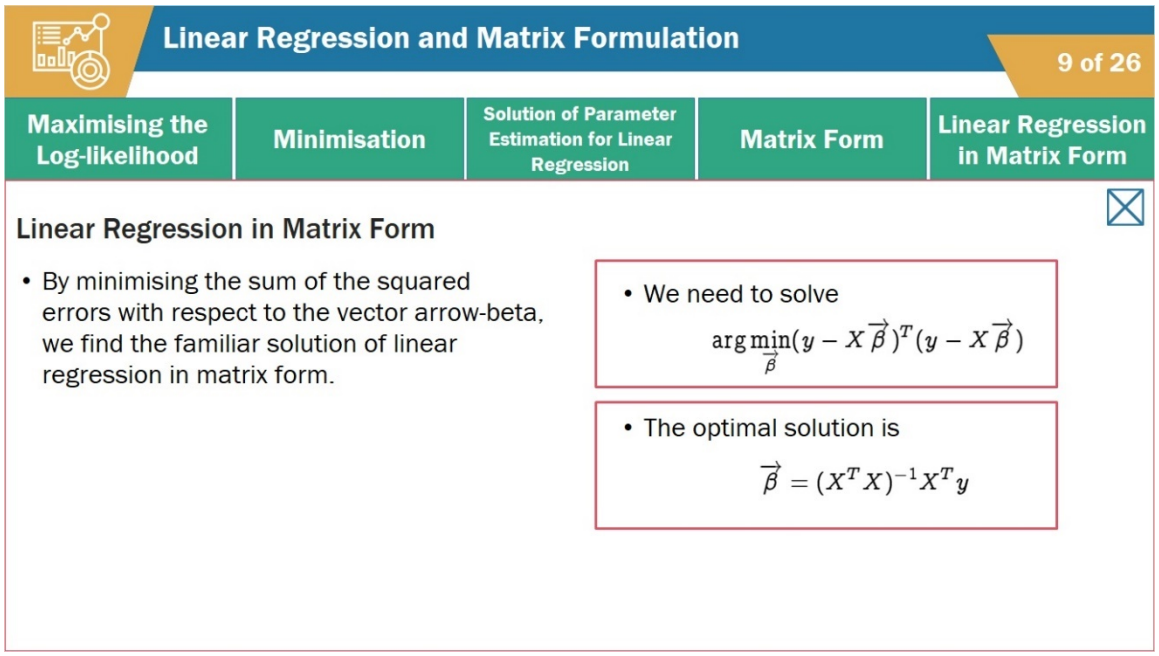


If there are m predictors (instead of only one as in the previous slide), we can use the matrix form to find the optimal values of the parameters. We define the matrix capital X and the vector y as in the slide,

where the subscript in the x denotes the index of the covariate and the superscript in the x the index of the observation in the dataset. I am assuming there are N observations and m covariates (also called predictors).

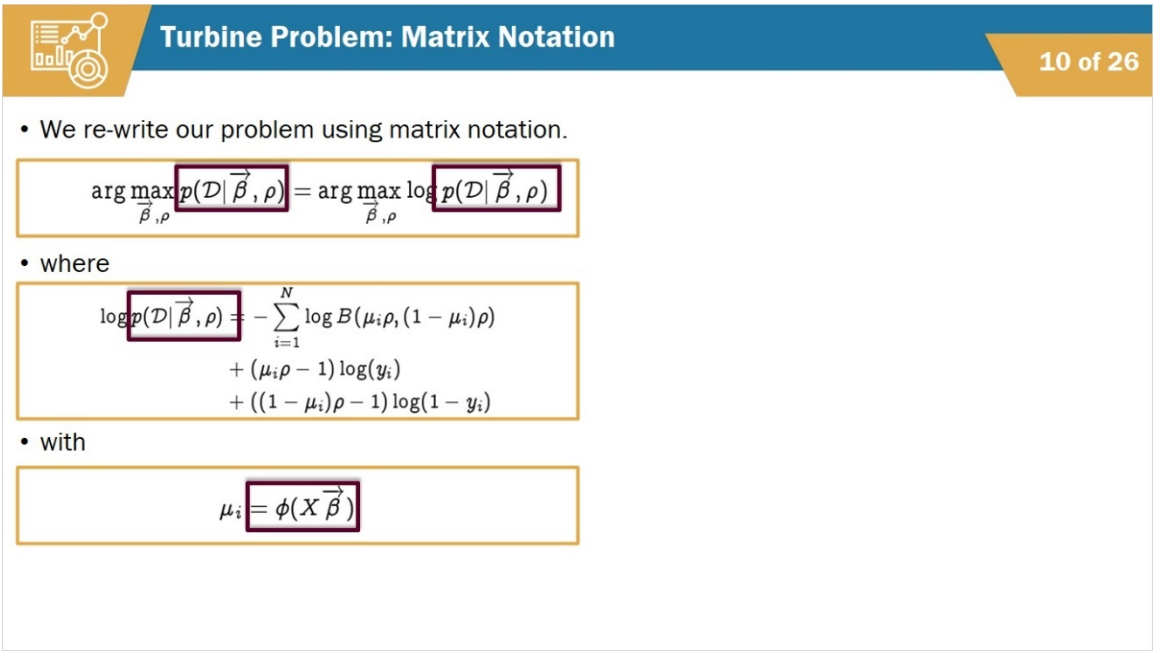
We can write the sum of the squared errors in this matrix form, where the arrow above the beta is used to denote the vector which includes all the unknown betas.

Tab 5: **Linear Regression in Matrix Form**



By minimizing the sum of the squared errors with respect to the vector arrow-beta, we find the familiar solution of linear regression in matrix form

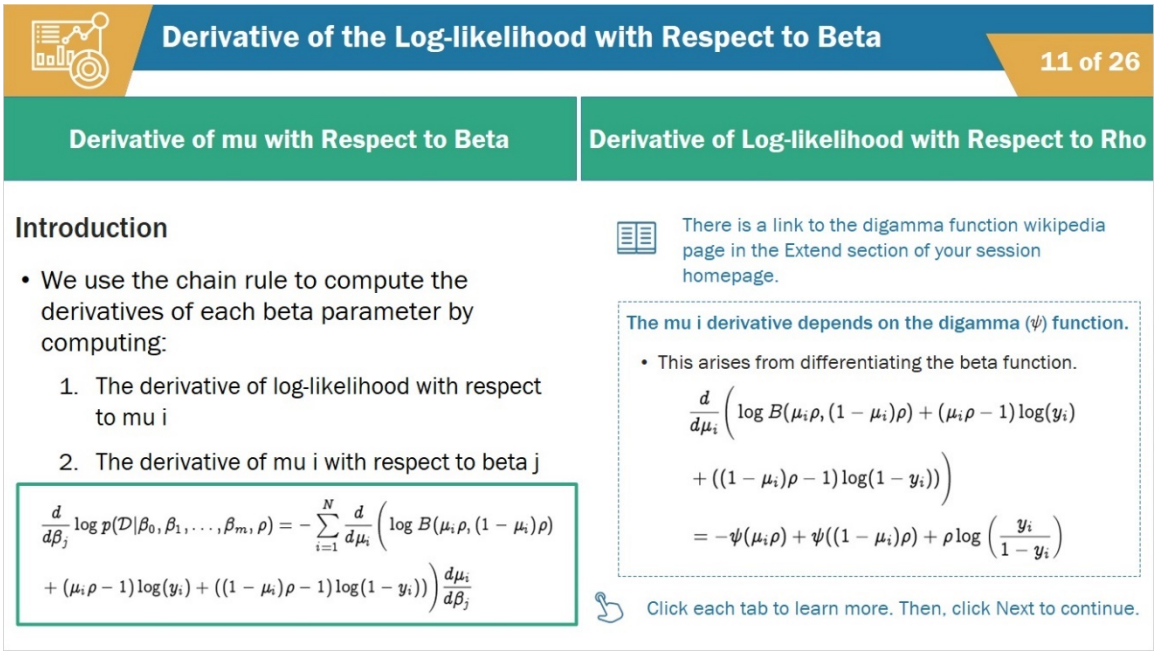
Slide 10: Turbine Problem: Matrix Notation



Now that we have explained the matrix form (which we need to solve the turbine problem), let's go back to our problem, which I have rewritten using the matrix notation. Also, in this case the unknown is the vector arrow-beta.

Take time to view the information on this slide. Then click next to continue.

Slide 11: Derivative of the Log-likelihood with Respect to Beta

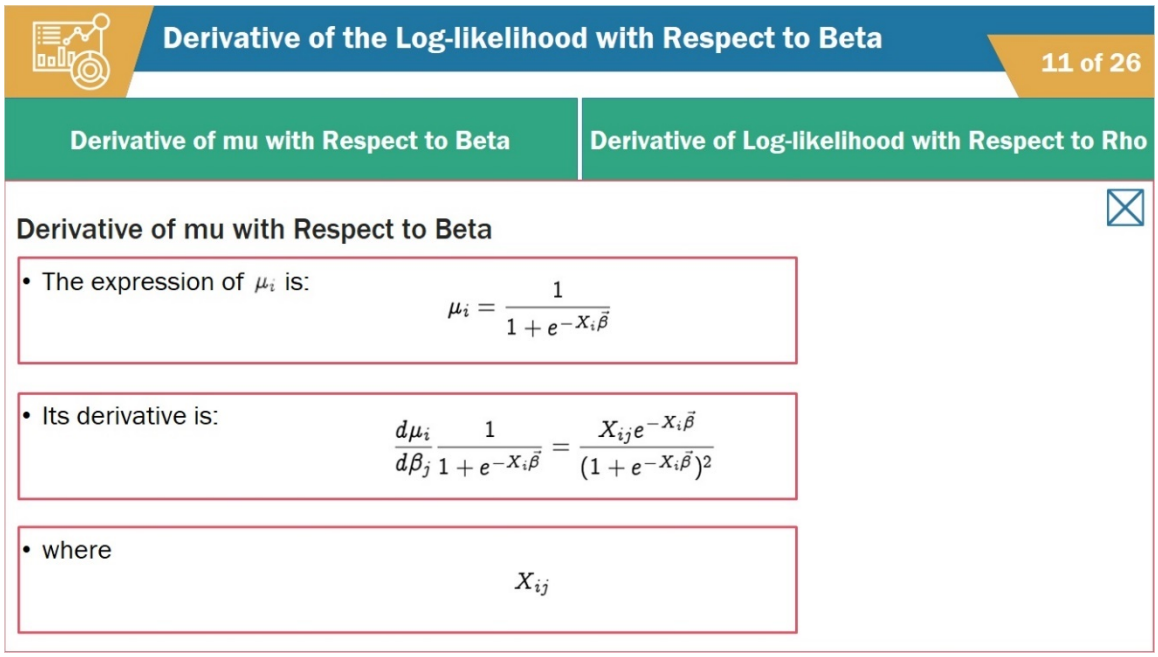


We use the chain rule to compute the derivatives of each beta parameter, by first computing the derivative of the log-likelihood with respect to mu i and then computing the derivative of mu_i with respect to beta_j, as you see in the formula below.

The derivative of the log-likelihood with respect to mu_i shown on the right depends on the digamma function, which arises from differentiating the Beta function (see wikipedia for details).

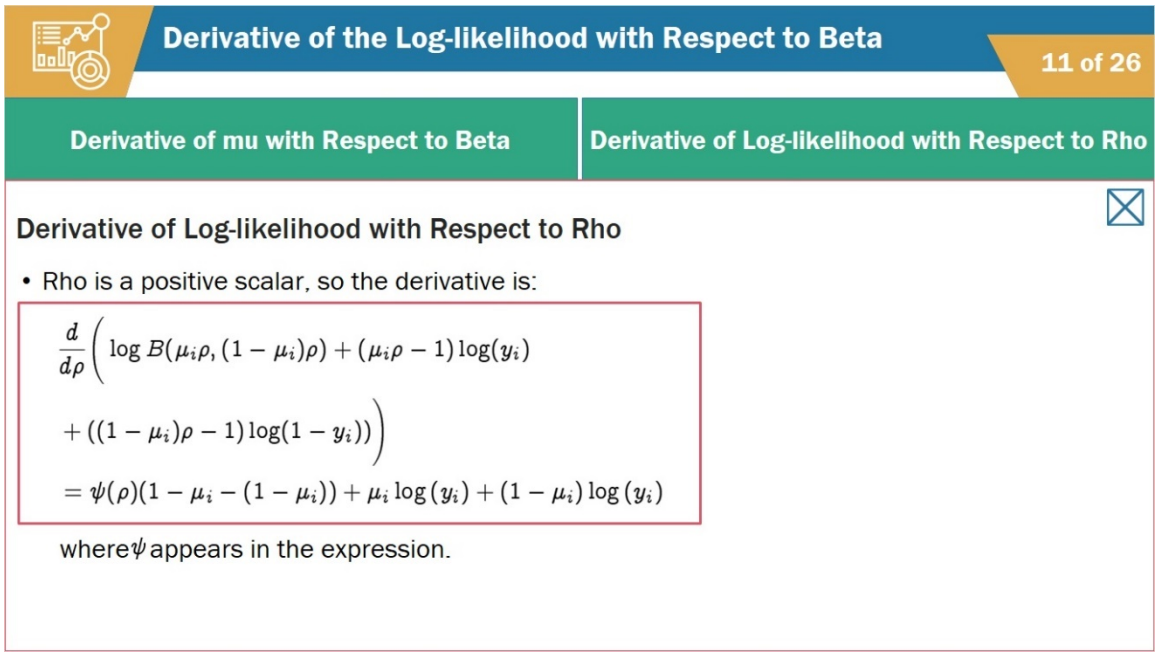
Click each tab to learn more about calculating these derivatives. Then, click next to continue.

Tab 1: Derivative of mu With Respect to Beta



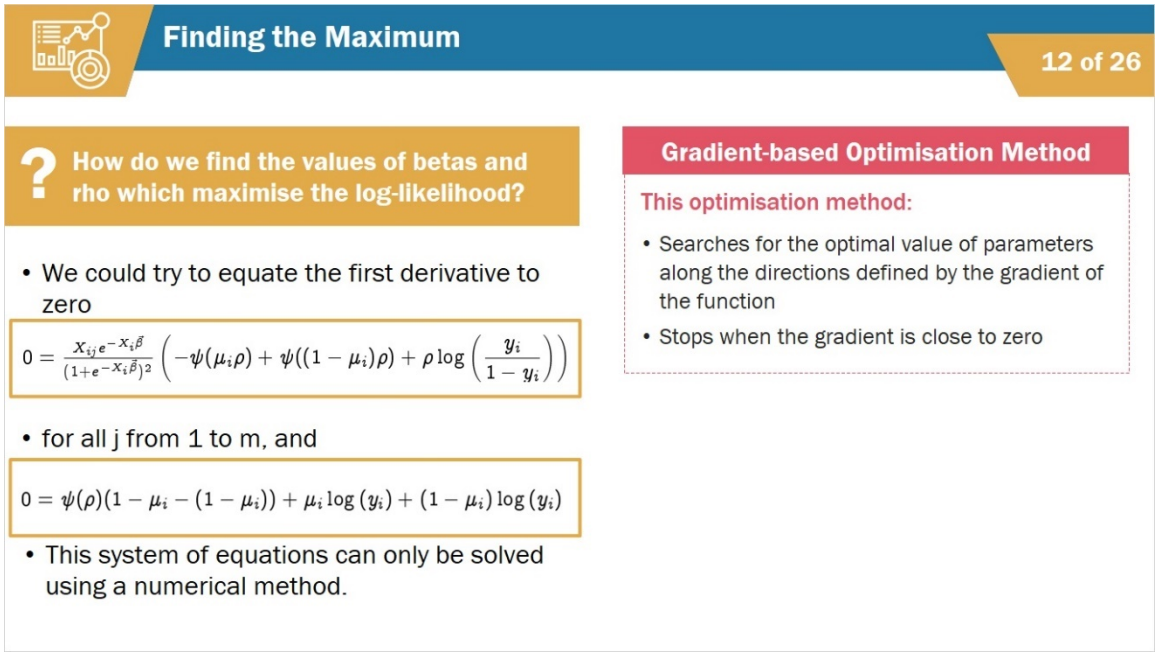
The derivative of mu_i with respect to beta_j can be easily calculated from the expression of the logistic function.

Tab 2: Derivative of the Log-likelihood With Respect to Rho



Rho is a positive scalar, so the derivative can be computed easily and again the digamma function appears in the expression of the derivative.

Slide 12: Finding the Maximum



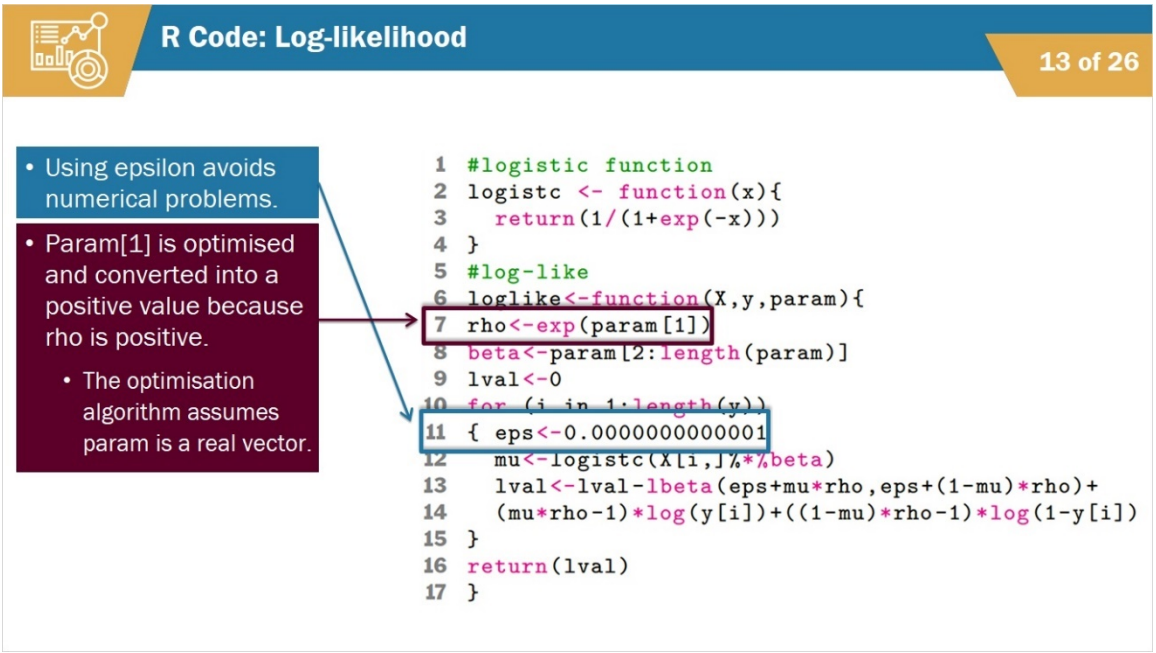
We want to find the values of betas and rho which maximize the log-likelihood. We could try to equate the first derivative to zero to do that and solve the system of equations you see in the slide.

Unfortunately, we cannot solve this system of equations analytically. Therefore, we use another strategy, a numerical method. We use a gradient-based optimisation method, that is an optimisiation method which searches for the optimal value of the parameters

by iteratively changing the parameters along the directions defined by the gradient of the function (at the current value of the parameter). It stops when the gradient is close to zero.

We will now implement it in R.

Slide 13: R Code: Log-likelihood



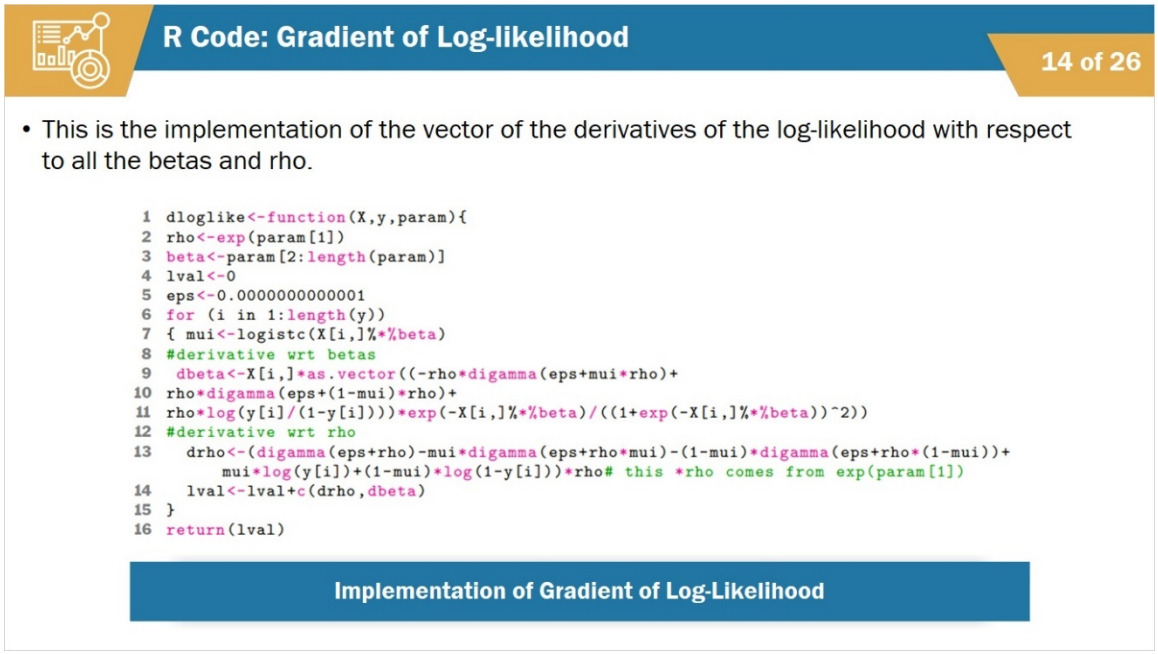
The following code implements the log-likelihood in R. AT the top you can see the implementation of the logistic function. The second R function is the log-likelihood.

```
#logistic function
logistc <- function(x){
return(1/(1+exp(-x)))
}
#log -like
loglike <-function(X,y,param){
rho<-exp(param [1])
beta<-param[2: length(param)]
lval<-0
for (i in 1:length(y))
{ eps<-0.0000000000001
mu<-logistc(X[i,]%*%beta)
lval<-lval -lbeta(eps+mu*rho ,eps+(1-mu)*rho)+
```

```
14 (mu*rho -1)*log(y[i])+((1-mu)*rho -1)*log(1-y[i])
15 }
16 return(lval)
17 }
```

We have introduced the epsilon to avoid numerical problems. Note the term exp(param[1]), we optimize param[1] which is real-variable and we use the exp transform to convert it to a positive value because rho is positive. This is necessary because the optimization algorithm, we will use later, assumes param is a real vector.

Slide 14: R Code: Gradient of Log-likelihood

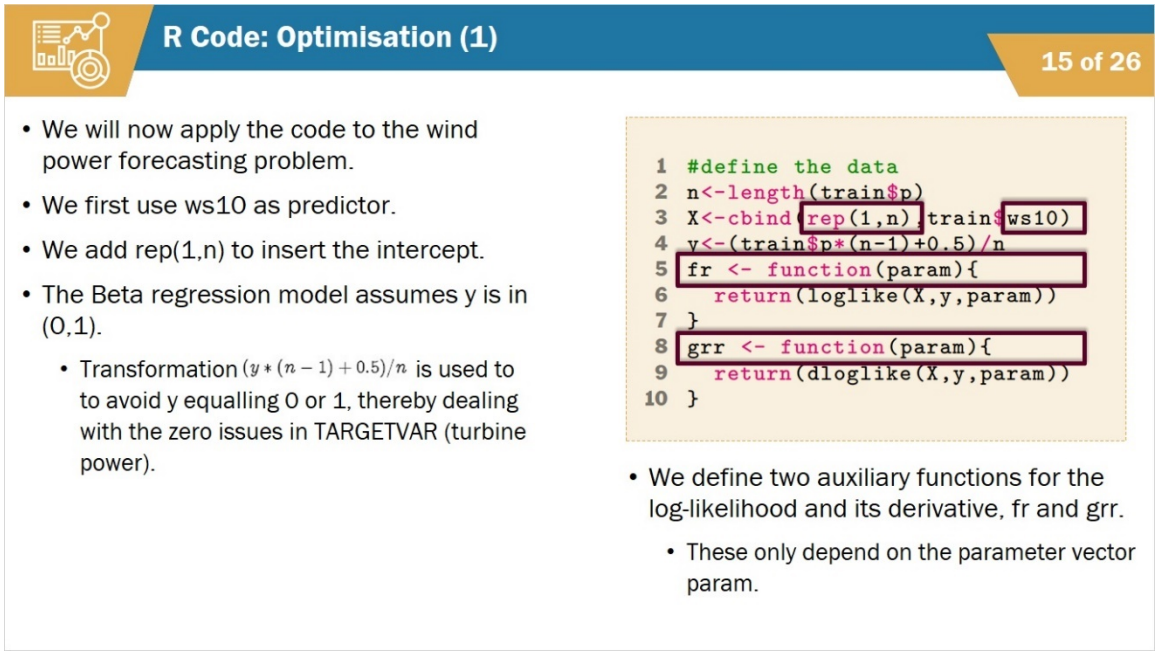


This is the implementation of the gradient of the log-likelihood, that is the vector that includes the derivatives of the log-likelihood with respect to all the betas and rho.

```
1 dloglike <-function(X,y,param){
2 rho<-exp(param [1])
3 beta<-param[2: length(param)]
4 lval<-0
5 eps<-0.0000000000001
6 for (i in 1:length(y))
7 { mui<-logistc(X[i,]%*%beta)
8 #derivative wrt betas
9 dbeta <-X[i,]*as.vector((-rho*digamma(eps+mui*rho)+
10 rho*digamma(eps+(1-mui)*rho)+
```

```
11 rho*log(y[i]/(1-y[i])))*exp(-X[i,]%*%beta)/((1+exp(-X[i,]%*%beta))^2))
12 #derivative wrt rho
13 drho<-(digamma(eps+rho)-mui*digamma(eps+rho*mui) -(1-
mui)*digamma(eps+rho*(1-mui))+
mui*log(y[i])+(1-mui)*log(1-y[i]))*rho# this *rho comes from exp(param [1])
14 lval<-lval+c(drho ,dbeta)
15 }
16 return(lval)
```

Slide 15: **R Code: Optimisation (1)**



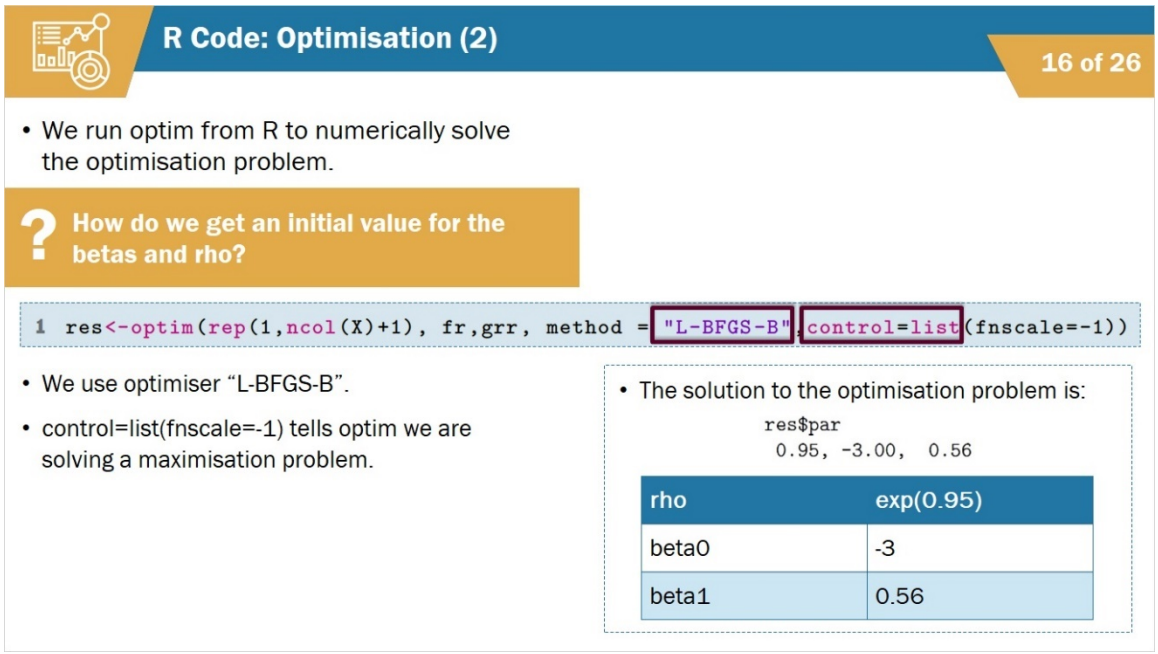
We are now going to apply the code to the wind power forecasting problem.

```
1 #define the data
2 n<-length(train$p)
3 X<-cbind(rep(1,n),train$ws10)
4 y<-(train$p*(n-1)+0.5)/n
5 fr <- function(param){
6 return(loglike(X,y,param))
7 }
8 grr <- function(param){
9 return(dloglike(X,y,param))
10 }
```

For the moment we only use ws10 as predictor. We have added rep(1,n) which is a column of ones to insert the intercept term. The Beta regression model we defined assume y is in (0,1), so we use the transformation (y*(n-1)+0.5)/n

where n is the number of observations, to avoid that y is equal to either 0 and 1. In this way, we deal with the issue of all the zeroes in TARGETVAR (turbine-power). We then define two auxiliary functions for the log-likelihood and its derivative, fr,grr, which only depend on the parameter vector param. This because the only variable we aim to optimize is param.

Slide 16: R code: Optimisation (2)



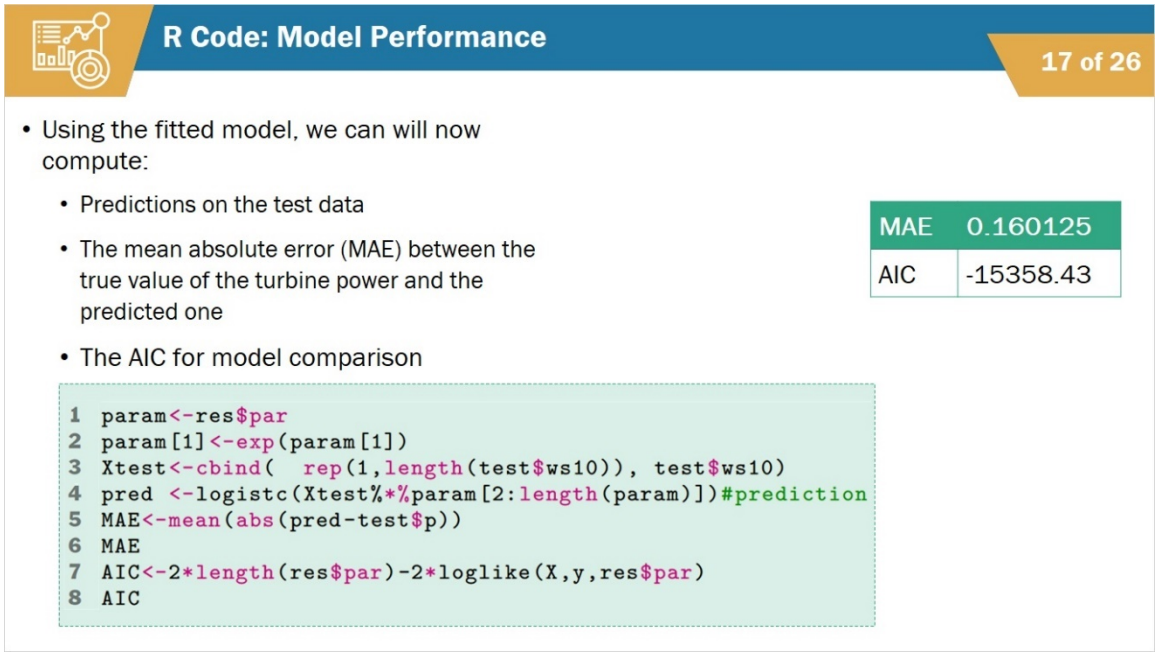
We then run optim from R which numerically solves the optimization problem. We need an initial value for the betas and rho: a vector of ones in the code seen here (this is arbitrary).

```
1 res<-optim(rep(1,ncol(X)+1), fr,grr , method = "L-BFGS -B",control=list(fnscale=-1))
```

"L-BFGS-B" is the name of the optimizer we use and control=list(fnscale=-1) tells optim we are solving a maximization problem.

The solution to the optimisation problem is rho equal to exp(0.95), beta0 is -3 and beta1 0.56

Slide 17: R Code: Model Performance



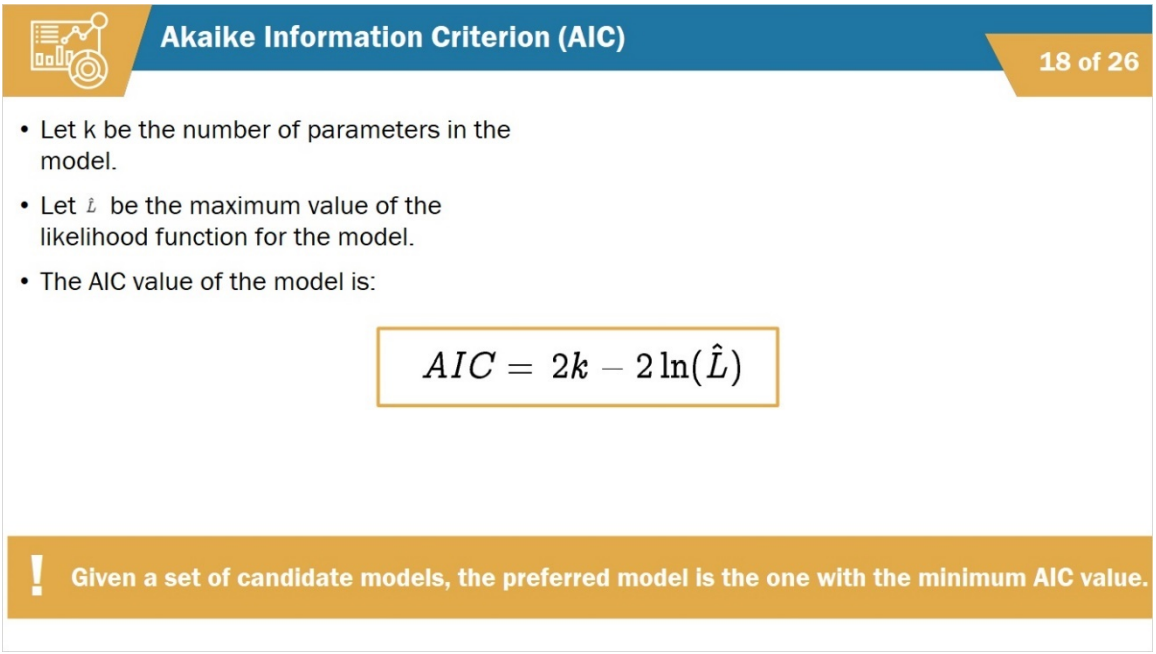
We can now use the fitted model to compute predictions on the test data. We then use these predictions to compute the mean absolute error (MAE) between the true value of the turbine power and the predicted one. This allows us to quantify the accuracy of the model. Finally we compute the AIC for model comparison. You can see the result on this slide.

```
1 param <-res$par
2 param[1]<-exp(param [1])
3 Xtest <-cbind( rep(1,length(test$ws10)), test$ws10)
4 pred <-logistc(Xtest%*%param[2: length(param)])#prediction
5 MAE<-mean(abs(pred -test$p))
6 MAE
7 AIC<-2*length(res$par)-2*loglike(X,y,res$par)
8 AIC
```

MAE = 0.1601025

AIC = 15358.43

Slide 18: Akaike Information Criterion (AIC)

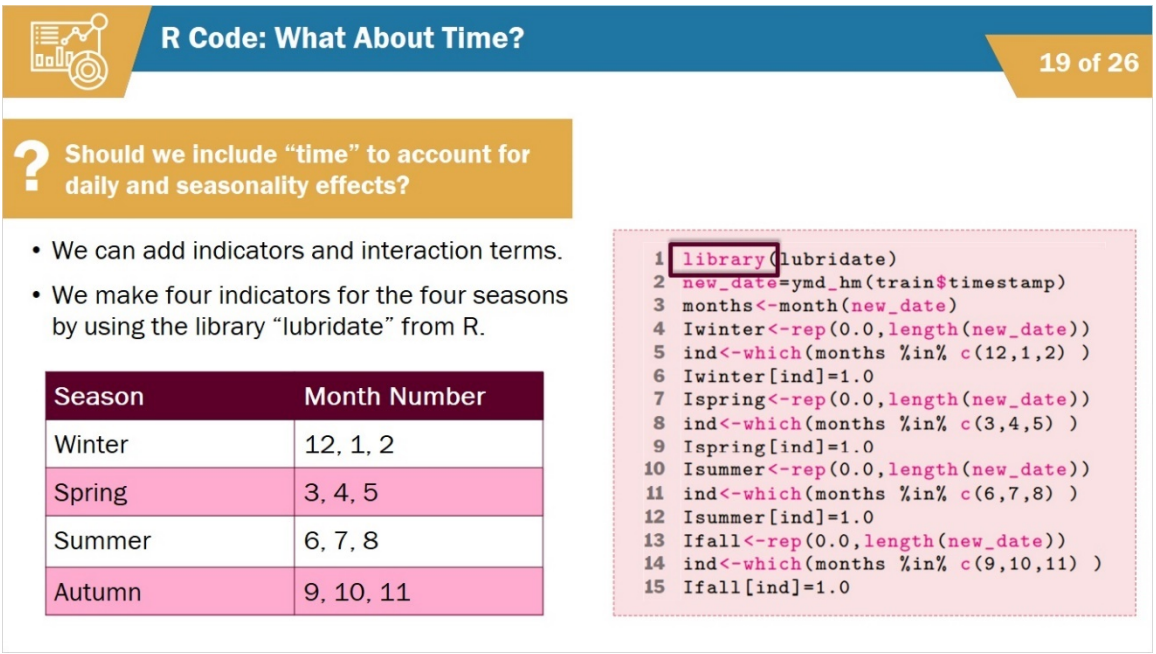


In an earlier presentation, we describe the AIC to perform model selection, in particular to select the most important covariates for predicting the response variable. Recall: let k be the number of parameters in the model. Let

hat-L be the maximum value of the likelihood function for the model. Then the AIC value of the model is 2*k-2 log-likelihood

Given a set of candidate models, the preferred model is the one with the minimum AIC value.

Slide 19: R Code: What About Time?



Should we include “time” to account for daily and seasonality effects? We may try to add indicators and interaction terms.

First, we need to make the indicators. We make 4 indicators for the 4 seasons, for Winter (month number 12,1,2), for Spring (month number 3,4,5) etc. To do that, we need to parse the dates in the timestamps, we use the library ``lubridate'' from R.

```
1 library(lubridate)
2 new_date=ymd_hm(train$timestamp)
3 months <-month(new_date)
4 lwinter <-rep(0.0,length(new_date))
5 ind<-which(months %in% c(12,1,2) )
6 lwinter[ind]=1.0
7 lspring <-rep(0.0,length(new_date))
8 ind<-which(months %in% c(3,4,5) )
9 lspring[ind]=1.0
10 lsummer <-rep(0.0,length(new_date))
11 ind<-which(months %in% c(6,7,8) )
12 lsummer[ind]=1.0
13 lfall <-rep(0.0,length(new_date))
14 ind<-which(months %in% c(9,10,11) )
15 lfall[ind]=1.0
```

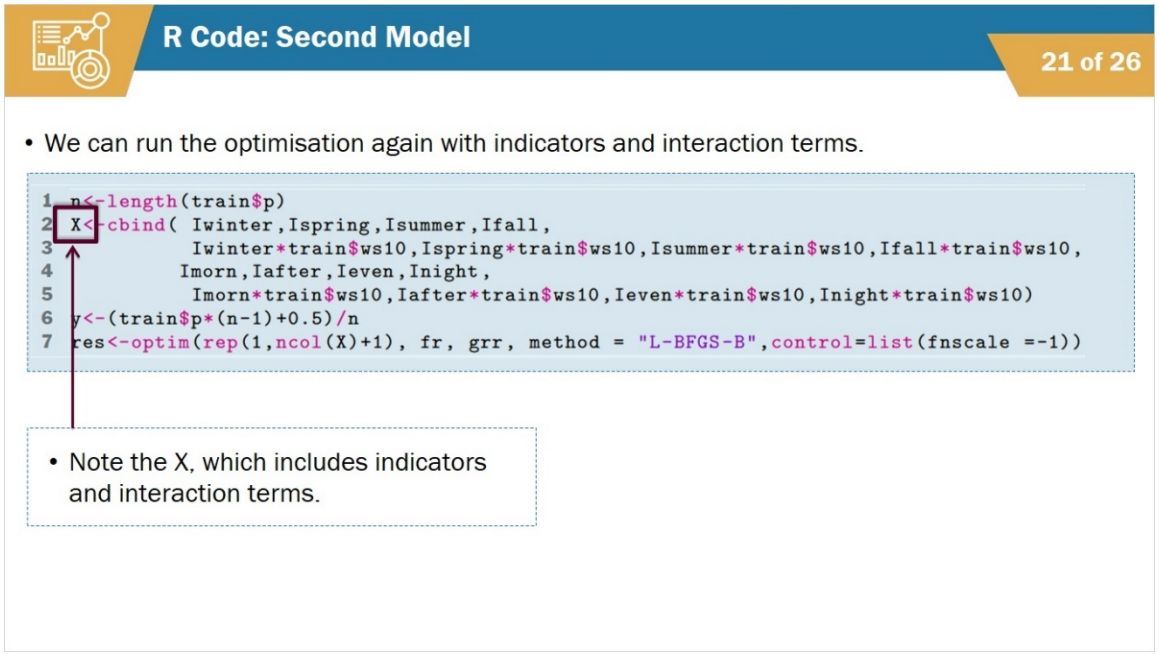
Slide 20: R Code: Hours

We use a similar code for dayparts.

```
1 library(lubridate)
2 new_date=ymd_hm(train$timestamp)
3 hours <-hour(new_date)
4 Imorn <-rep(0.0,length(new_date))
5 ind<-which(hours %in% c(6,7,8,9,10,11) )
6 Imorn[ind]=1.0
7 Iafter <-rep(0.0,length(new_date))
8 ind<-which(months %in% c(12,13,14,15,16) )
9 Iafter[ind]=1.0
10 Ieven <-rep(0.0,length(new_date))
11 ind<-which(months %in% c(17,18,19,20,21,22) )
12 Ieven[ind]=1.0
13 Inight <-rep(0.0,length(new_date))
14 ind<-which(months %in% c(23,1,2,3,4,5) )
15 Inight[ind]=1.0
```

We make 4 indicators for morning, afternoon, evening and night.

Slide 21: R Code: Second Model

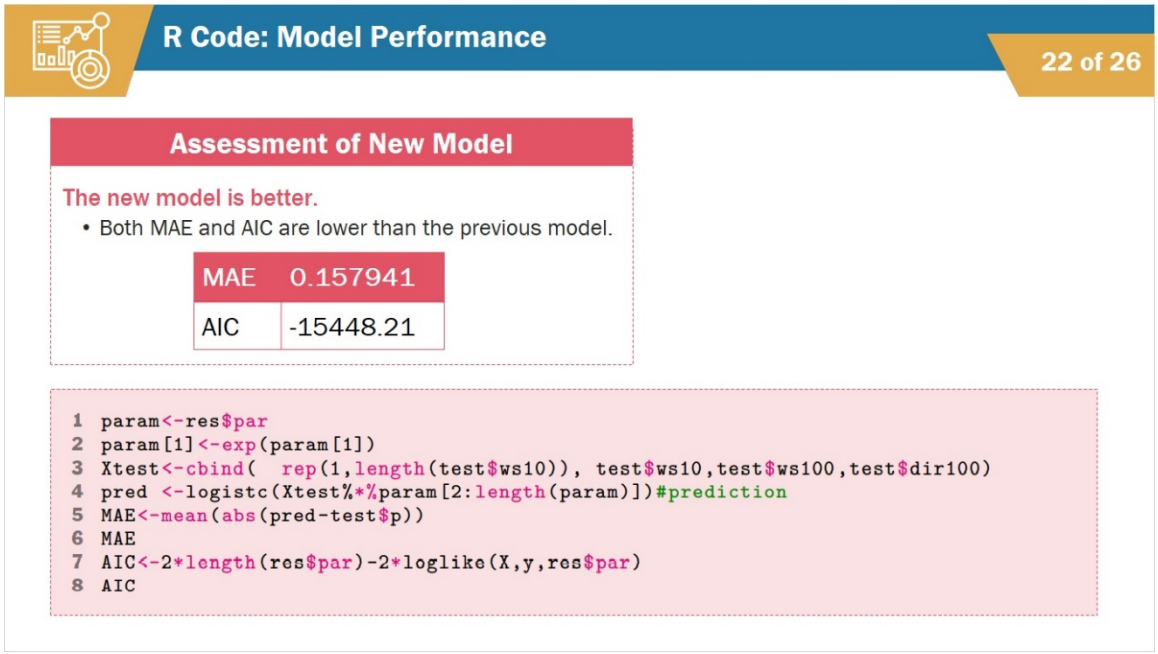


This is our model which has indicators and interaction terms.

```
1 n<-length(train$p)
2 X<-cbind( Iwinter ,Ispring ,Isummer ,Ifall ,
3 Iwinter*train$ws10 ,Ispring*train$ws10 ,Isummer*train$ws10 ,Ifall*train$ws10 ,
4 Imorn ,Iafter ,Ieven ,Inight ,
5 Imorn*train$ws10 ,Iafter*train$ws10 ,Ieven*train$ws10 ,Inight*train$ws10)
6 y<-(train$p*(n-1)+0.5)/n
7 res<-optim(rep(1,ncol(X)+1), fr, grr , method = "L-BFGS -B",control=list(fnscale =-1))
```

Note in the code that, we only need to change X, so our code is very general (similar to glm in R).

Slide 22: R Code: Model Performance



We can assess the performance of the new model.

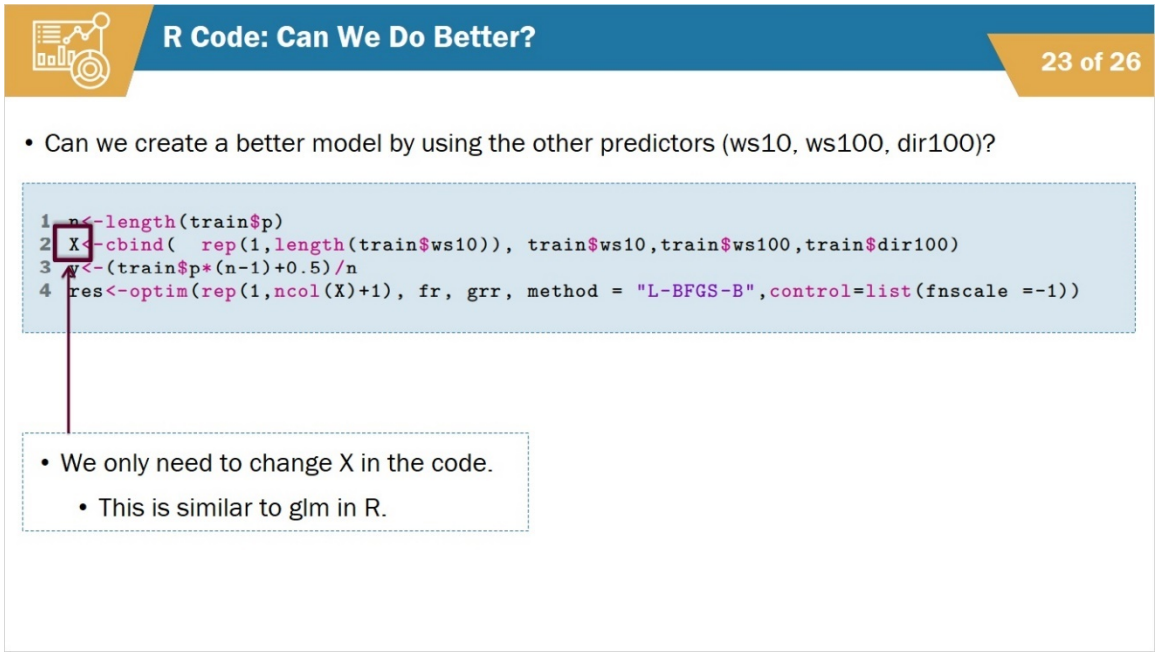
```
1 param <-res$par
2 param[1]<-exp(param [1])
3 Xtest <-cbind( rep(1,length(test$ws10)), test$ws10 ,test$ws100 ,test$dir100)
4 pred <-logistc(Xtest%*%param[2: length(param)])#prediction
5 MAE<-mean(abs(pred -test$p))
6 MAE
7 AIC<-2*length(res$par)-2*loglike(X,y,res$par)
8 AIC
```

MAE = 0.1579741

AIC = -15448.21

The new model has a lower MAE and a lower AIC, so it is definitely better.

Slide 23: R Code: Can We Do Better?



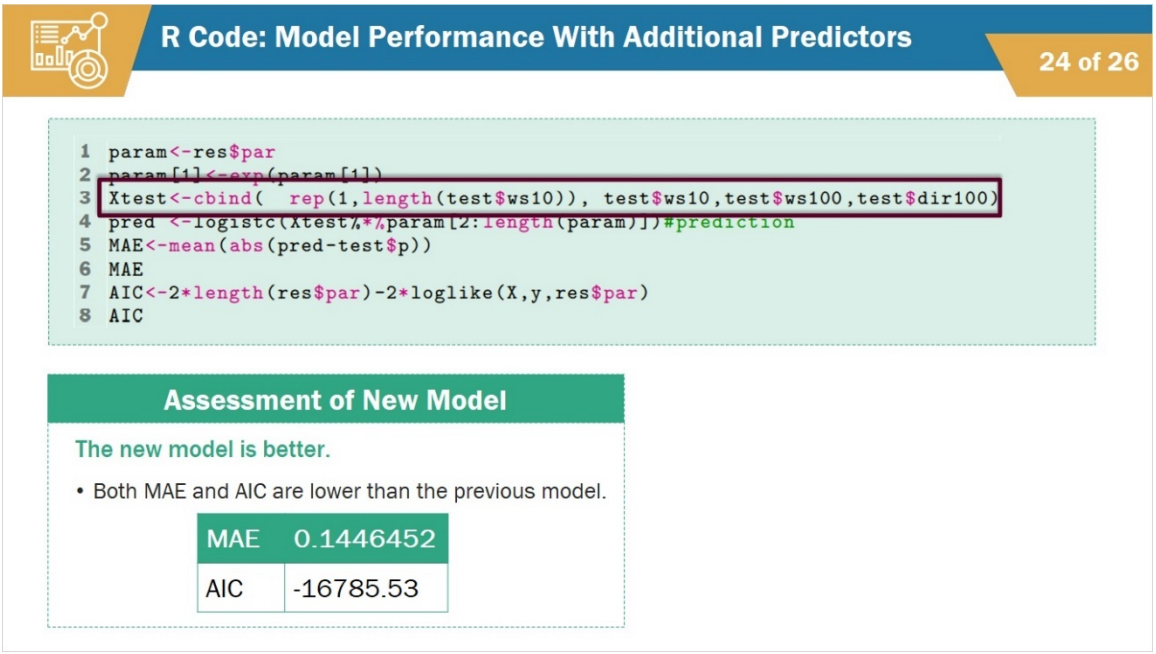
Can we create a better model for instance, by using the other predictors (ws10, ws100,dir100)?

```
1 n<-length(train$p)
2 X<-cbind( rep(1,length(train$ws10)), train$ws10 ,train$ws100 ,train$dir100)
3 y<-(train$p*(n-1)+0.5)/n
4 res<-optim(rep(1,ncol(X)+1), fr, grr , method = "L-BFGS -B",control=list(fnscale =-1))
```

Note that we only need to change x, so our code is very general. It is similar to glm in R.

Take time to view the information on this slide. When you are ready, click next to continue.

Slide 24: R Code: Model Performance with Additional Predictors



We can assess the performance of the new model that includes the predictors (ws10, ws100, dir100), note we changed the way Xtest is defined because the covariates of the new model include ws10, ws100 and dir100.

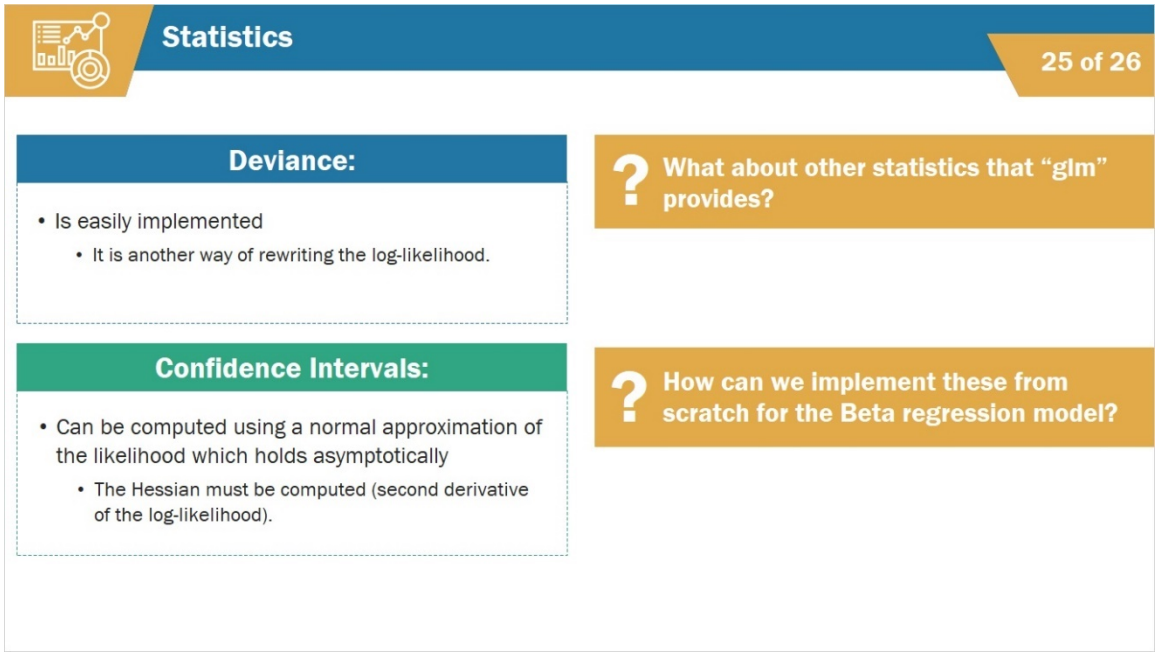
```
1 param <-res$par
2 param[1]<-exp(param [1])
3 Xtest <-cbind( rep(1,length(test$ws10)), test$ws10 ,test$ws100 ,test$dir100)
4 pred <-logistc(Xtest%*%param[2: length(param)])#prediction
5 MAE<-mean(abs(pred -test$p))
6 MAE
7 AIC<-2*length(res$par)-2*loglike(X,y,res$par)
8 AIC
```

MAE = 0.1446452

AIC = -16785.53

The new model has a lower MAE and a lower AIC, so it is definitely better than both the previous ones.

Slide 25: Statistics

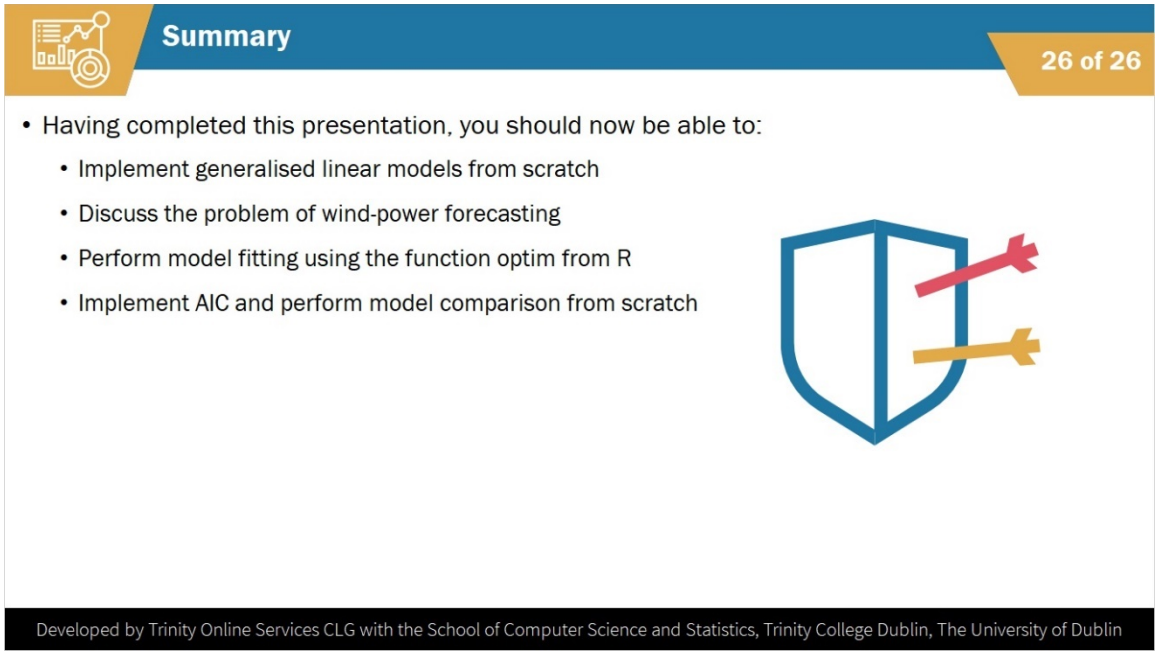


What about the other statistics "glm" provides us, for instance, deviance, confidence intervals etc., **how can we implement them from scratch for the Beta regression model?**

Deviance can be easily implemented because, as we saw for logistic regression, deviance is just a way of rewriting the log-likelihood.

Confidence intervals can be computed using a Normal approximation of the likelihood which holds asymptotically. In order to do that, we need to compute the Hessian of the log-likelihood, that is the second derivatives of the log-likelihood. ***It is not difficult, but it requires lengthy calculations and so we won't do it in this lesson.***

Slide 26: Summary



In this presentation, we have developed and implemented from scratch

in R a new generalised linear model to solve a -wind-power forecasting problem.

Having completed this presentation, you should now be able to:

Implement generalised linear models from scratch

Discuss the problem of wind-power forecasting

Perform model fitting using the function optim from R and

Implement AIC and perform model comparison from scratch