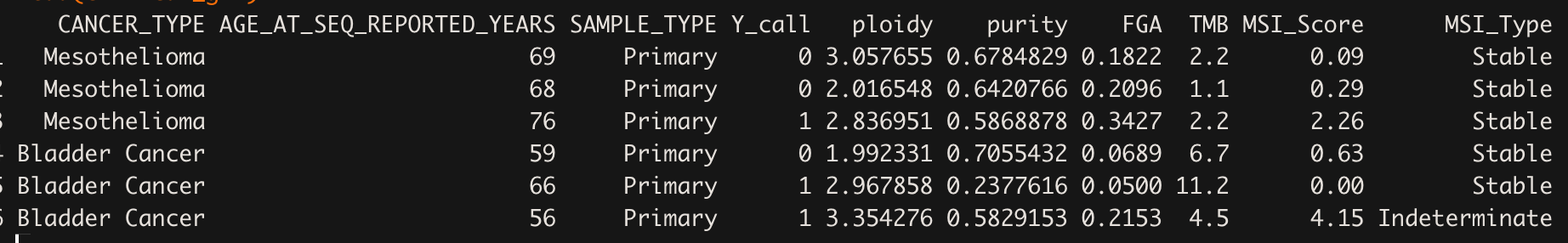
**Logistic Regression Models: a short primer (12/2021)**

**Motivation:** Let’s start with a real problem that has engaged me lately. We are interested in which covariates may predict (or influence) the occurrence of a Y-chromosome loss (Y\_call) in solid cancer tissues. For this purpose, we created a dataset which comprises 12,405 individual cancer samples. Variables which were determined included **i)** cancer-type, **ii)** fraction genome altered (FGA), **iii)** age, and so on (see below for a snapshot).



The most general form of our problem can be stated as follows:

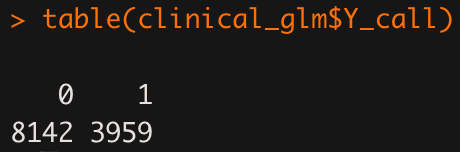
Note, since the dependent variable *Ycall* is binary [0, 1] we are using a generalized linear model (aka glm). Importantly, our dependent variable will be binomially distributed, hence we are specifying ‘*family’* as binomial. *‘family’* is a generic function, and whenever we are using the *‘binomial-family’* we can truly speak about fitting a logistic regression model. Translating this in R means:

Importantly, since *‘Ycall’* is binary, make sure that it is properly encoded in R (*integer*;

1 = success, Y-chromosome loss AND 0 = failure, intact Y-chromosome).

**Observation 1:** Now we can naively ask ourselves: *What is the probability of observing an individual cancer specimen with a Y-chromosome loss (without considering any covariate)?*

It's simple. Create a table and calculate the probability.

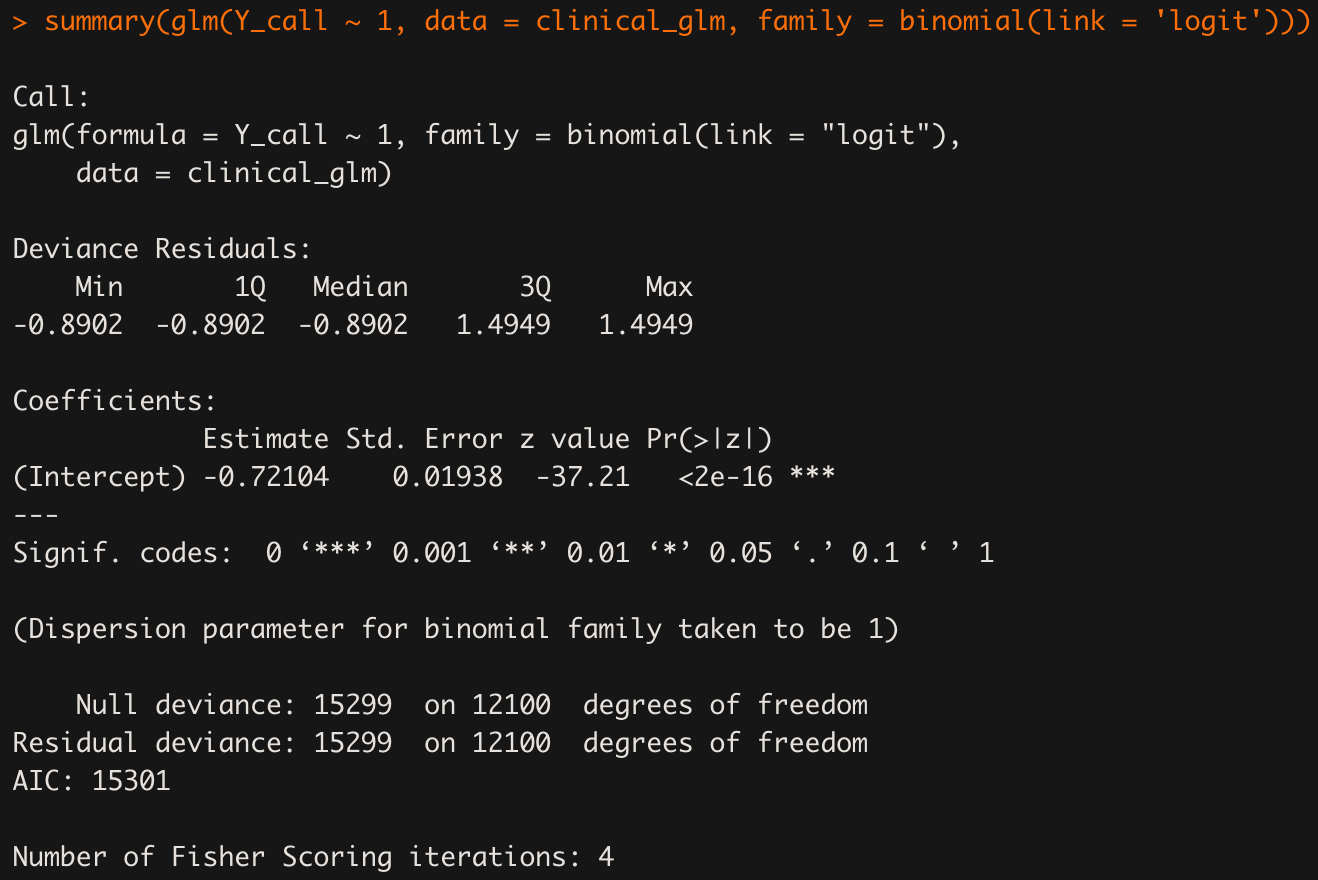


3959 / (3959 + 8142) = 0.327 or **32.7%** to draw a sample with a Y-chromosome loss.

Can we confirm our naive calculation with a sophisticated model in R? Yes, we can.

a. #’ beta0 is the intercept

b.

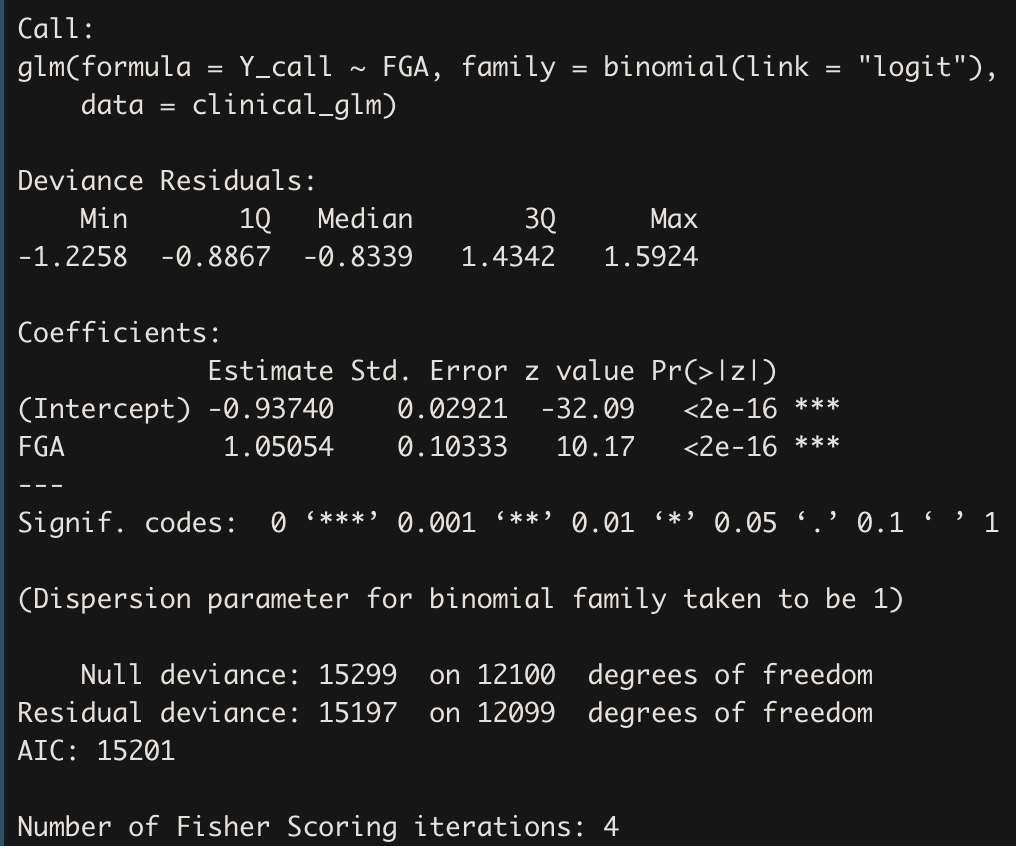


**log(ODDS) by default**

* we specify ***‘1’*** as a predictor variable () to estimate the intercept without any covariate
* you may realize that the model output (summary()) only include the intercept
* the intercept was **estimated at -0.72104** (with associated std.error, z-value and Pr)**.**
* Let's concentrate on the estimate first:
  + **-0.72104 = log(ODDS)** or the ‘logit’ (estimate)
  + if we **exponentiate** the **log(ODDS)**, we obtain the **ODDS**
  + exp(-0.72104) = 0.4862463 = **ODDS**
  + **ODDS / (1 + ODDS) = probability** (see Appendix)
  + **probability = 0.327164 = 32.7%** (exactly what we calculated above in our naive implementation)
* Std.Error is a associated with the estimate
* z value (*how much uncertainty ‘surrounds’ the point estimate*):
  + the z value is simply the **Estimate / Std.Error**
  + rule of thumb: if *abs(z value) > 2*, there is a good chance that the point-estimate of the predictor variable is reasonable. The larger the *z value* gets, the less uncertainty there is.
* Pr(>|z|) reports a p-value derived from z-statistics (details will not be covered here)

**Model performance.** The *Null* and *Residual* deviance is actually the same in our model, as we didn’t include any predictor variable. Hence, the residual deviance can’t be smaller than the Null deviance (i.e. **Null = 15299** AND **Residual = 15299**).

**Observation 2:** *How does the model change if we include one continuous predictor variable (e.g. FGA)*?



* Firstly, the intercept changed from the previous model (-0.72104 → -0.93740)
* exp(-0.93740) / (1+exp(-0.93740)) = **0.281** or **28.1%**. This means that, by including one predictor variable (in this case FGA), the chance to observe a cancer specimen with a Y-chromosome loss, given that **FGA = 0** is 28.1%. Hence, the probability decreased by ~ 4% when compared to the previous model (without any predictor variable). This further suggests that FGA may have some ‘positive’ influence on the outcome (also indicated through a positive *estimate*.
* How can we interpret the ‘FGA: estimate’ of 1.05054?

Recall our general model formula:

1. {FGA = 0.1} → -0.93740 + 1.05054\*0.1 = -0.832346
2. {FGA = 0.2} → -0.93740 + 1.05054\*0.2 = -0.727292
3. {FGA = 0.3} → -0.93740 + 1.05054\*0.3 = -0.622238

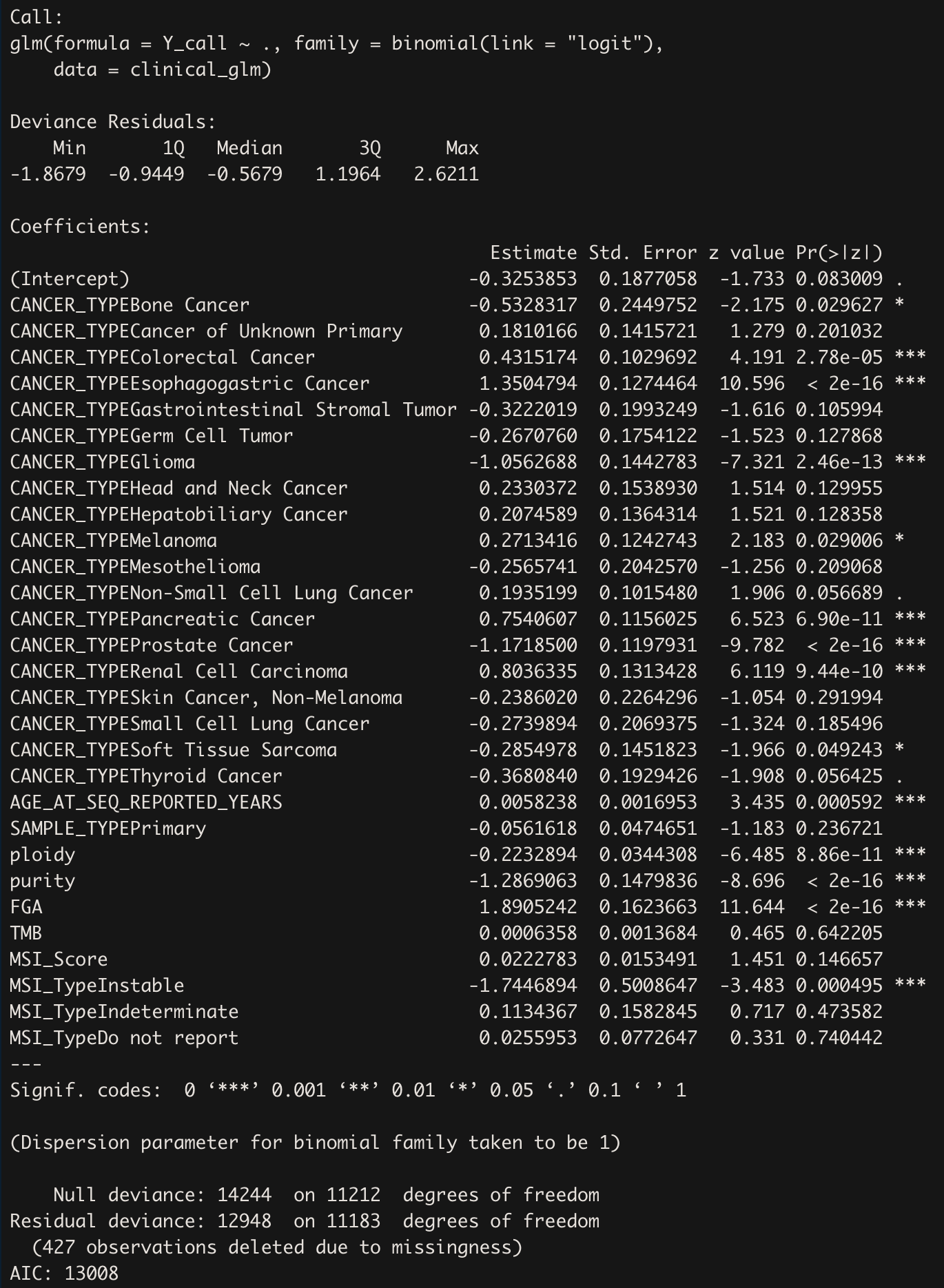
If we now want to know the difference between FGA = 0.3 and FGA = 0.2, we simply subtract the two values and obtain **0.105054**. If we want to know the difference between FGA = 0.2 and FGA = 0.1, we subtract the two values and again obtain **0.105054**. You get the idea. Generally speaking - whenever I am *increasing* or *decreasing* ONE UNIT in FGA (in our case we are using a **1/10-scale**, since FGA is in the range of {0,1}) we exactly obtain our model estimate → **0.105054 \* 10 = 1.05054.**

We can also translate our FGA-Estimate into a probability (according to the formulas described in the Appendix) and obtain **11.1%**. Meaning that whenever I am increasing one unit of FGA increases the probability of observing a cancer specimen with a Y-chromosome loss by ~ 11%.

05054)

One additional thing to mention. Compare the Residual deviance in this model with the Residual deviance in the previous model. You clearly see that it is reduced, due to the incorporation of one predictor variable.

**Observation 3:** *What does a full saturated model look like?*



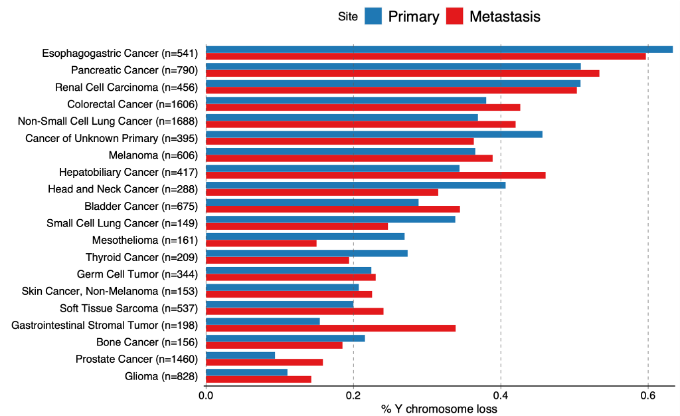
From this model, there are a few things to mention:

1. General terms:

* **‘~ .’** in the formula indicates that all variables in the data frame should be used
* predictor variables in this model are both, numerical (ploidy, purity, etc.) and categorical (SAMPLE\_TYPE, MSI\_TYPE)

1. Cancer\_TYPE:

* I converted the variable ‘CANCER\_TYPE’ from the input table into a factor and used ‘Bladder Cancer’ as the reference group.
* This means that every cancer type estimate must be interpreted relative to bladder cancer.
* If we look into the distribution of Y-chromosome loss across different cancer types, the individual estimates make quite sense.
* Those with a **positive** estimate and those ranked higher in the plot below (e.g. Esophagastric, Pancreatic, etc.). Those cancer types are significantly different from Bladder Cancer (which in the figure below is somehow in the middle).
* You can also choose a different reference level. By default, R is using an alphabetic order. But you can also compare everything against ‘Gliomas’ for example.



1. Overall effect of CANCER\_TYPE:

We have seen that the ODDS of observing a Y-chromosome loss among individual cancer types differs. We can furthermore test for an overall effect of CANCER\_TYPE using a wald.test function (aod package). We use the wald.test function. b supplies the coefficients, while Sigma supplies the variance covariance matrix of the error terms, finally Terms tells R which terms in the model are to be tested, in this case, terms 2 - 20 are used.

wald.test(b=coef(model\_full), Sigma=vcov(model\_full), Terms=2:20)

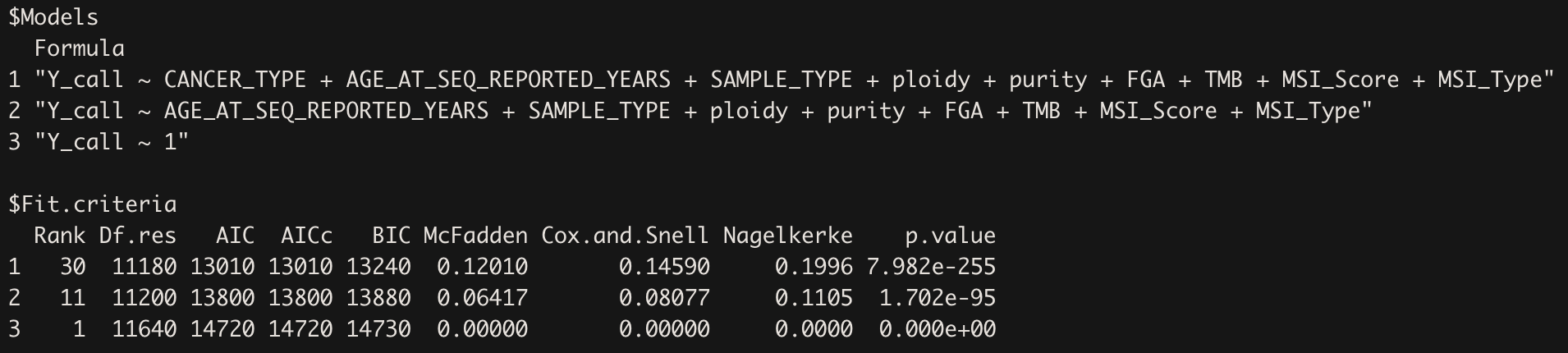
Running this model, we obtain p = 0. This means that Y chromosome loss rates among different cancer types are significantly different and that there is evidence for direct selection or maintaining the Y-chromosome.

There are more things we could spend some time on, but for the sake of time, we neglect the interpretation of the further terms.

Again - pay attention to the AIC in this case. It's lower than in the previous model, indicating that the current model is better suited to describe the data

**Observation 4:** *How can we measure the goodness of fit of respective models and how can we compare two (or several) among each other?*

Actually there are several attempts described in the literature. Since there are many suggestions, I will not focus on one specific approach, rather, I will propose one way. compareGLM() is a function provided by the R rcompanion package. This function calculates several ‘goodness-of-fit’ estimates (e.g. AIC, BIC, McFadden, etc.). If we are about to compare our Full-model (i.e. including all predictor variables) with a ‘reduced-model’ (i.e., excluding ‘CancerType’) and the Null-model (i.e. just including the intercept), we get the following output.



Some naive interpretations:

* The lower the AIC estimate the better the model performance
* The higher the Pseudo-R2 (McFadden, Cox.and.Snell, Nagelkerke) the better

You clearly see that the more predictor variables we include (ie. full model) the more variance can be explained. This means that our full model is an appropriate approximation of the dataset. For those who are more interested in assessing the goodness of fit of a model, I provide a small summary about the McFadden estimate.

**McFadden:** McFadden's 𝑅2 is defined as 1−𝐿𝐿𝑚𝑜𝑑 / 𝐿𝐿0, where 𝐿𝐿𝑚𝑜𝑑 is the log likelihood value for the fitted model and 𝐿𝐿0 is the log likelihood for the null model which includes only an intercept as predictor (so that every individual is predicted the same probability of 'success'). For a logistic regression model the log likelihood value is always negative (because the likelihood contribution from each observation is a probability between 0 and 1). If your model doesn't really predict the outcome better than the null model, 𝐿𝐿𝑚𝑜𝑑 will not be much larger than 𝐿𝐿0 and so 𝐿𝐿𝑚𝑜𝑑 / 𝐿𝐿0 ≈ 1 and McFadden's pseudo-𝑅2 is close to 0 (your model has no predictive value).

Conversely if your model was really good, those individuals with a success (1) outcome would have a fitted probability close to 1, and vice versa for those with a failure (0) outcome. In this case if you go through the likelihood calculation the likelihood contribution from each individual for your model will be close to zero, such that 𝐿𝐿𝑚𝑜𝑑 is close to zero, and McFadden's pseudo- 𝑅2 squared is close to 1, indicating very good predictive ability.

**Likelihood Ratio Tests:** Likelihood ratio tests are used to compare the goodness of fit of two statistical models. The LRT compares two hierarchically nested models to determine whether or not adding complexity to your model (i.e., adding more parameters) makes your model significantly more accurate. The “hierarchically nested models” simply means that the complex model differs only from the simpler (or “nested”) model by the addition of one or more parameters.

In summary, the LRT tells us if it is beneficial to add parameters to our model, or if we should stick with our simpler model.

The test statistics here follow a chi-squared distribution with degrees of freedom equal to the difference in the number of free parameters between the complex model and the reduced model. With this information, we may calculate the p-value, and if it is less than our significance level, we reject the null hypothesis.

A = logLik(reduced)

B = logLik(complex)

LLR = -2\*(as.numeric(A) - as.numeric(B))

p.val = pchisq(LLR, df = df(B) - df(A), lower.tail = FALSE)

p.val = 0

A common significance level to use is .05. Under that significance level, we would reject the null hypothesis and conclude that we should use the more complex model. In this example we obtain a p-value of 0, meaning that the more complex model is doing better than the reduced model.

There is also a convenient function available in R, which automatically calculates the LRT for you.

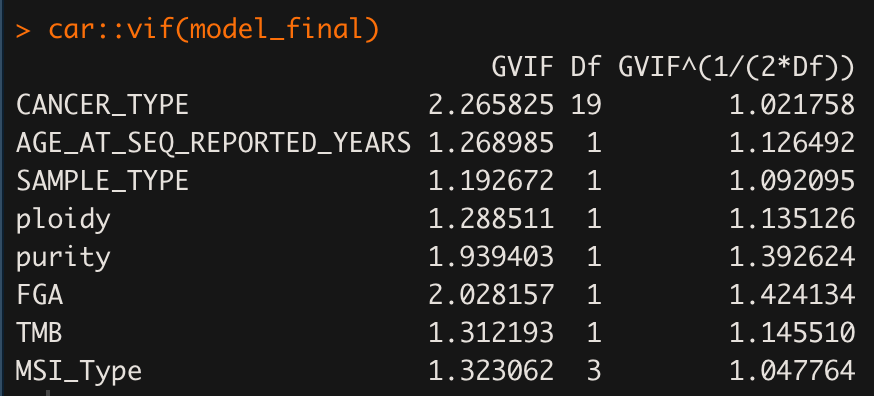
lrtest() from the lmtest package

**Observation 5:** Multicollinearity among predictor variables.

Simply speaking. If there is extensive correlation among predictor variables (especially when a lot of variables are included in the model), the outcome is highly skewed and interpretations may be grossly misleading. To check for the multicollinearity problem, we can take advantage of the vif() function from the car package in R. VIF stands for variance inflation factor. It measures how much the variance of any one of the coefficients is inflated due to multicollinearity in the overall model. As a rule of thumb, a vif score over 5 is a problem. A score over 10 should be remedied and you should consider dropping the problematic variable from the regression model or creating an index of all the closely related variables.



We see that MSI\_Score as well as MSI\_Type both show very high (VIF >> 5) estimates and this is problematic (but somehow also makes sense). So, what can we do? Let’s exclude the MSI\_Score predictor variable from our full model and see what is happening.



Looks good :)

Well, I know you will ask now → what’s happening with the model performance. Technically speaking they are the same (with some modest exceptions, not shown). However, as MSI\_TYPE and MSI\_Score were highly correlated, the performance didn’t go down significantly.

**Take home messages**

* Make sure that your variables are properly encoded in R
  + **as.integer** for response **{0 ,1}**
  + **as.factor** for categorical variables **{specify reference group}**
  + **as.numeric** for numerical variables
* provided model estimates are in **logit-units (NOT ODDS)**
  + if you are interested in ODDS, you need to exponentiate the logit-units
  + if you want to translate ODDS into probabilities take a look into the Appendix
* Use may consider the wald.test() for assessing the overall effect of one predictor variable
* There are several ways to compare the model fit
  + Metrics described (McFadden, AIC, etc.) should be interpreted with caution. There is no ‘one size fits all’ metric, rather, use different estimates to compare model among each other.
  + Check for multicollinearity among predictor variables\

Why are we even transforming *‘probabilities’* from a logistic regression model to ODDS or log(ODDS)? Firstly, probabilities are **always** bound in the range of 0-1. Secondly, ODDS range from 0 to positive infinity. Thirdly, log(ODDS) range **from negative to positive infinity** (and this makes the usage of log(ODDS); logit) quite cool. The transformation is an attempt to get around the restricted range problem with probabilities.

**Appendix:**

**,**

,