Prediction of the 2016 Olympic Medals

Author: Gabriel Berardi

University of Glasgow
School of Mathematics - Data Analytics (ODL)

Submission Date: 17 June 2021

Contents

1. Introduction

The goal of this paper is to find a suitable model to predict the total number of medals won by each country at the Rio Olympics in 2016 using information that was available prior to the Rio Games. More specifically, we have a dataset containing data for the Olympic games between 2000 and 2016, for 108 countries.

The attributes include the following numerical data for each country:

- The GDP at the time of each Olympic games [in millions of US dollars]
- The population at the time of each Olympic games [in thousands]
- The number of total and gold medals in each Olympic games
- The altitude of the country's capital city [in m]
- The number of athletes representing the country in each Olympic games

The attributes further include the following categorical data about each country:

- Whether or not the country was part of the soviet union
- Whether or not the country is/was a communist state
- Whether or not the country is a Muslim majority country
- Whether or not the country is a one-party state
- Whether or not the country has hosted/is hosting/will be hosting the Olympics

Last but not least, the dataset also contains information about:

- The overall total number of gold medals awarded in the Olympics between 2000 and 2016
- The overall total number of all medals awarded in the Olympics between 2000 and 2016

The emphasis is on predicting the total number of medals won by each country and not so much on interpreting the coefficients for our variables.

2. Validity Check

1. Looking at the original data, we realize that there are missing values in the dataset which are encoded as `#N/A`. We can specify this when we load the data into R with `oldat<-read.csv(..., na.strings = "#NA")` As such, we can see that the column `gdp00` contains 1 missing value for Afghanistan and `gdp16` contains 2 missing values for Cuba and the Syrian Arab Republic.
2. Moreover, by comparing the sum of all observations in the `goldYY` and `totYY` columns, we can see that the total number of gold medals and the total number of all medals seem to be wrong for 2016.
3. Looking at `summary(oldat)` we can see that all other columns contain data in a sensible range and there seems to be no faulty entries.

3. Handling Data Issues

As we only have three missing values, it is best to find a sensible way of estimating these data points. Since the GDP of a country in a specific year is usually highly dependent on its neighbouring years, we could try to interpolate the data given existing data points for the GDP in 2004 (for Afghanistan) and 2012 (for Cuba and Syria). Alternatively, it might be worth investigating whether the missing data points can be obtained from another source.

For the missing value of Cuba's GDP in 2016, the World Bank reports this to be 91.37 billion USD, which also seems reasonable when looking at Cuba's GDP in 2012.[1] Therefore, we will us this to substitute the missing value.

For Afghanistan's GDP in 2000 and Syria's GDP in 2016, there is no available data from the World Bank. Due to the political situation in the two countries in the two years mentioned, it would be inadvisable to assume that an interpolation of the data could yield sensible results. Instead, we will use an estimate made by the website countryeconomy.com which assumes a GDP of 3.53 billion USD for Afghanistan in 2000[2] and 12.38 billion USD for Syria in 2016[3]. These assumptions seem reasonable given the hardship the two countries encountered in the relative years.

With regard to the wrong sum of total number of gold medals and the total number of all medals, we can simply replace the original sum with a newly calculated sum of the medals in 2016 with `oldat$totgold16<-sum(oldat$gold16)` and `oldat$totmedals16<-sum(oldat$tot16)` respectively.

4. Arrange Data

In its original form, the data contains many columns in wide format, such as `gdpYY, popYY` and others. We will transform this to a long format, in order to faciliate visualization with `ggplot` and later model building.This can be done using the `melt` function from the `data.table` package. After transforming the data, the structure of our dataset look like this:

```
## Classes 'data.table' and 'data.frame':   540 obs. of  16 variables:
##  $ country         : Factor w/ 108 levels "Afghanistan",..: 1 1 1 1 1 2 2 2 2 2 ...
##  $ country.code    : Factor w/ 108 levels "AFG","ARE","ARG",..: 1 1 1 1 1 28 28 28 28 28 ...
##  $ soviet          : Factor w/ 2 levels "0","1": 1 1 1 1 1 1 1 1 1 1 ...
##  $ comm            : Factor w/ 2 levels "0","1": 1 1 1 1 1 1 1 1 1 1 ...
##  $ muslim          : Factor w/ 2 levels "0","1": 2 2 2 2 2 2 2 2 2 2 ...
##  $ oneparty        : Factor w/ 2 levels "0","1": 1 1 1 1 1 1 1 1 1 1 ...
##  $ altitude        : num  1790 1790 1790 1790 1790 1 1 1 1 1 ...
##  $ host            : Factor w/ 2 levels "0","1": 1 1 1 1 1 1 1 1 1 1 ...
##  $ year            : chr  "2000" "2004" "2008" "2012" ...
##  $ gdp             : int  3532 5285 10191 20537 19469 54790 85325 171001 209059 159049 ...
##  $ pop             : int  20094 24119 27294 30697 34656 31184 32831 34861 37566 40606 ...
##  $ gold_medals_won : int  0 0 0 0 0 1 0 0 1 0 ...
##  $ total_medals_won: int  0 0 1 1 0 5 0 2 1 2 ...
##  $ total_golds     : int  298 301 301 301 305 298 301 301 301 305 ...
##  $ total_medals    : int  915 924 949 956 968 915 924 949 956 968 ...
##  $ athletes        : int  0 5 4 6 3 47 61 56 38 64 ...
##  - attr(*, ".internal.selfref")=<externalptr>
```

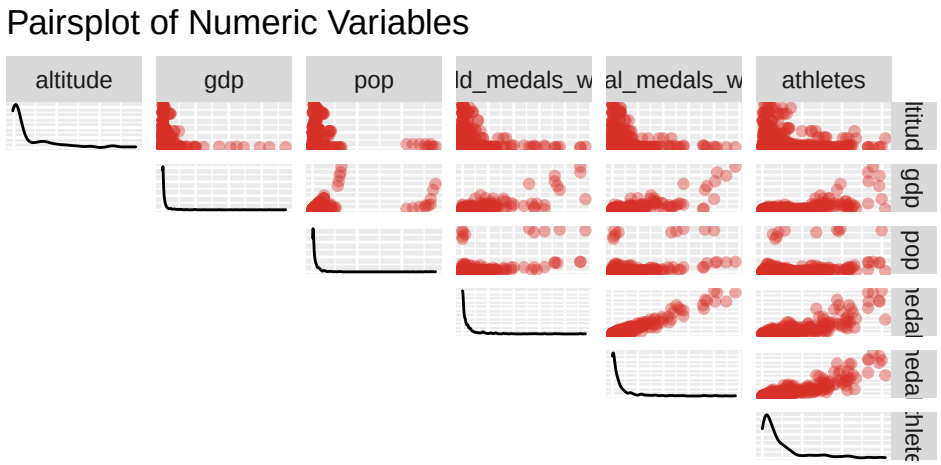
[1] https://data.worldbank.org/indicator/NY.GDP.MKTP.CD?locations=CU

[2] https://countryeconomy.com/gdp/afghanistan

[3] https://countryeconomy.com/gdp/syria

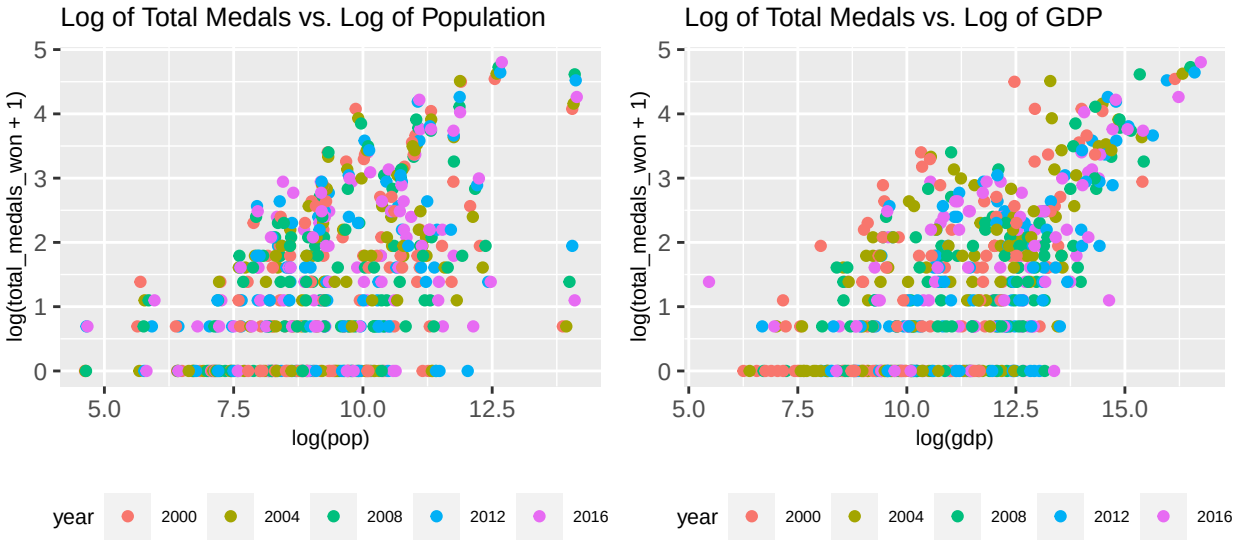
5. Exploratory Data Analysis

Let's start this short EDA by looking at a pairsplot of all the numerical variables we have in our dataset:



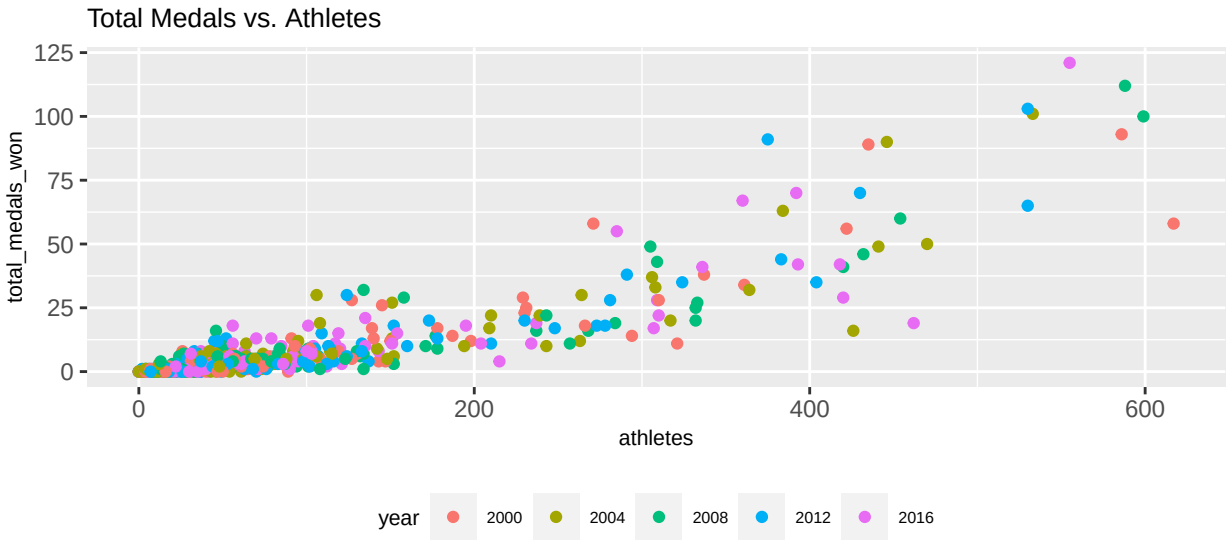
From this pairsplot, we can immediately see that all numerical variables are extremely right-skewed, giving evidence of some outliers with very large values. We can also see a strong linear relationship between the total medals won by each country (fifth column) and the total gold medals won by each country (fourth row). This is not surprising as we would expect a very strong correlation between these two variables. Last but not least, we can see that there are also some linear relationships between the total number of medals won and the number of athletes, the GDP and the population of each country.

When looking at the scatter plot of the total number of medals won and the population of each country, we can see that the points are very crowded and close to each other with regard to the population and GDP, with a few outliers. Let's therefore log-transform these two variables and plot their relationship again. Note that there are many countries who did not win any medal. We therefore need to add 1 to this variable, otherwise the log-transformation does not work.



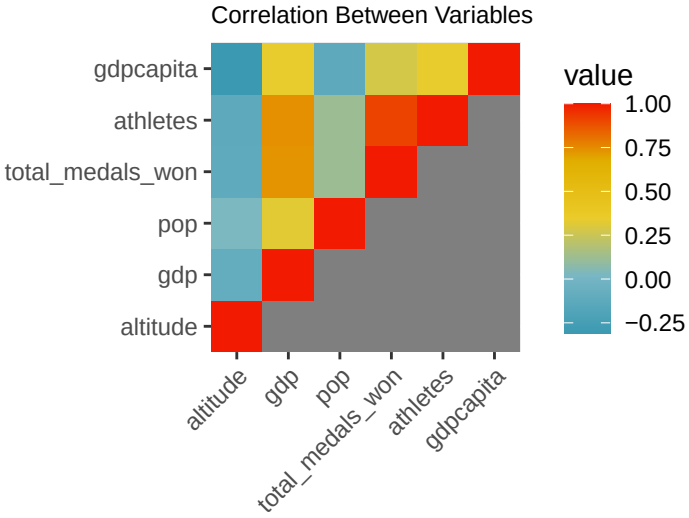
As we can see there seems to be some relationship between the total number medals won and the population and GDP by each country, but it doesn't seem to be very strong.

Let's also look at the relationship between the number of athletes and the total number of medals won:



Looking at the scatter plot of the number of total medals against the athletes, we can see some relationship as well.

Let's conclude this short EDA by looking at the correlation between all numerical variables in the dataset:



As we can see, the total number of medals seems to correlate most with the number of athletes and the GDP of a country. We also notice a linear correlation between the number of athletes and the GDP, as well as some weak correlations between the population and the GDP. However, the correlation does not seem to be too large and we will therefore ignore its impact when building models.

6. Splitting the Data into a Training and a Test set

After the EDA, let's split the data into a training and a test set to evaluate our final model. We will only keep the data for the year 2012 to train our models, as we would otherwise violate the assumption of independent observations (i.e. if a country had a large number of athletes competing in the 2000 Olympics, it had most likely also sent many athletes to the 2004 Olympics and so on). We can therefore assign out `train.data` and `test.data` by filtering for the year `2012` and `2016` respectively. We will then also drop the columns `gold_medals_won, total_medals and total_golds` as this would obviously not be known prior to the outcome of the Olympic games. Finally, we can also drop the column `year` as it holds no useful information anymore.

7. Model Building

Before we start building a model, it is always good to look at our variables and see if we can construct any new features from them. In our case, we can add a new column to the dataset which shows the GDP per capita for each country, by dividing the `gdp` column by the `pop` column.

Also, we should decide what measure to use in order to compare the predictive power of different models we are going to build. Let's use the Root Mean Squared Error (RMSE), which is the square root of the mean of the square of all the residuals, i.e. the standard deviation of the prediction errors and is given by the formula:

$$RMSE = \sqrt{\sum_{i=1}^{n} \frac{(\hat{y}_i - y_i)^2}{n}}$$

where $\hat{y}_i$ are the predicted values, y_i are the observed values and n is the number of observations.

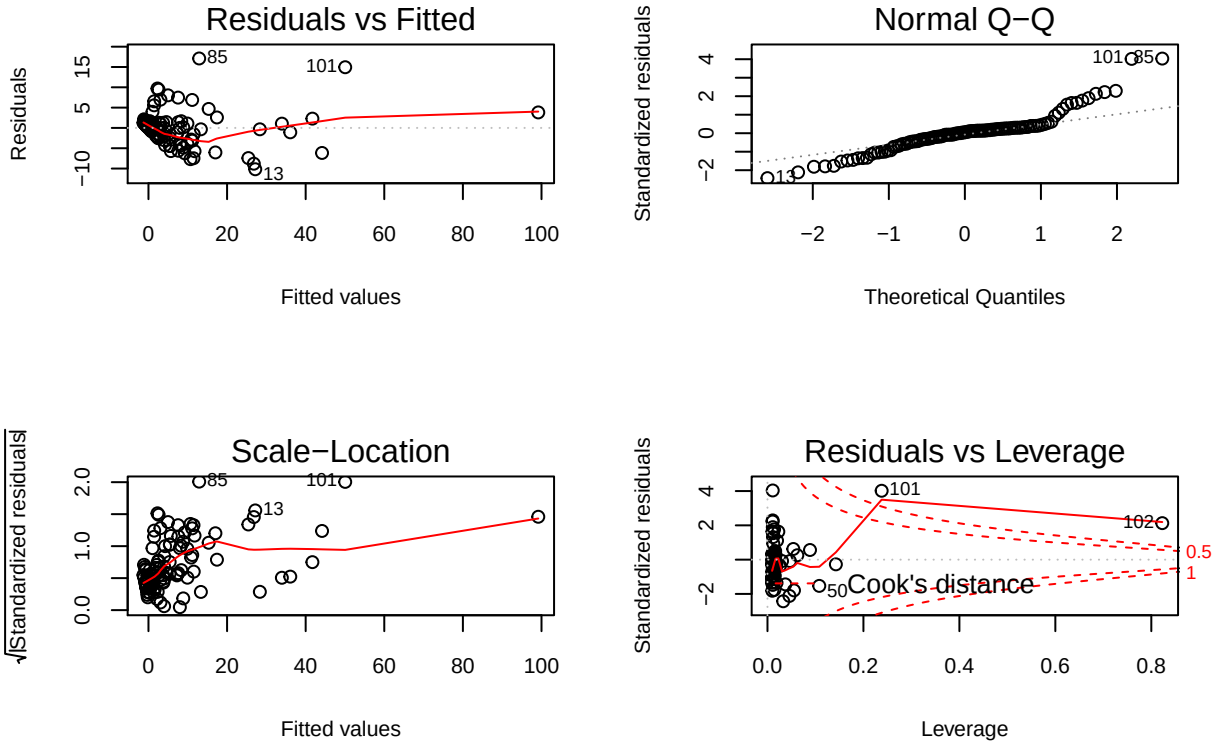
7.1 Linear Regression

Let's start our model building process with a very simple linear model. We can fit a model using all variables using `lm(total_medals_won ~ . -country -country.code, data = train.data)`. This model has an adjusted R-squared of 0.90 but there seem to be many non-significant variables. Let's first find and remove outliers using the `outlierTest()` function from the `car` package, which reports Bonferroni p-values for testing each observation in turn to be a mean-shift outlier. We can spot two outliers in this manner. After removal, we can go ahead and use the `ols_step_forward_p()` function from the `olsrr` package, which builds up the linear regression model by entering predictors based on p values, in a stepwise manner until there is no variable left to enter any more.

```
##
##                            Selection Summary
## ------------------------------------------------------------------------
##         Variable                 Adj.
## Step    Entered     R-Square    R-Square      C(p)        AIC        RMSE
## ------------------------------------------------------------------------
##    1    athletes      0.8018      0.7999     43.1088    694.8041    6.2944
##    2    gdp           0.9100      0.9083    -34.1416    613.0665    4.2610
##    3    pop           0.9128      0.9102    -34.1732    611.7451    4.2153
##    4    comm          0.9142      0.9108    -33.2078    612.0129    4.2016
##    5    oneparty      0.9154      0.9112    -32.0786    612.5326    4.1932
## ------------------------------------------------------------------------
```

As we can see, the adjusted R-squared increases up to 0.91 if we include `athletes, gdp, pop, comm and oneparty` into our model, and the RMSE sinks from 6.29 to 4.19. However, when we fit a linear model with these 5 variables and look at the `summary(linear.regression)`, we can see that `oneparty` is actually not a significant predictor. We can remove this variable and check the p-values of the predictors in the resulting model and keep removing variables until all remaining predictors are significant. When we do this, we end up with a model that has `athletes and gdp` as predictors with an adjusted R-squared of 0.91.

However, if we look at `plot(linear.regression)`, we can see that some assumptions of a linear model are dubious and violated.



For example, the assumption of normally distributed residuals is clearly dubious and their variance also seems to be not constant. Moreover, there is a correlation of 0.69 between the number of athletes and the GDP, so there might be an issue with multi-colinearity here. Last but not least, there still seem to be more outliers. Therefore, we remove two more outliers based on Cook's distance and re-fit the model. We end up with a model that has an adjusted R-squared of 0.81 and an RMSE of 3.80 (see Appendix A).

Apart from the dubious assumptions, our linear model has one significant downside, which is that it does not respect the bounds of zero with regard to the count data we are dealing with. This means, that the linear regression might predict negative values for the total number of medals won by a country, which is clearly problematic.

Therefore, we will now look at models that are more suitable for this type of counts data we have here.

7.2 Poisson Regression

The first model that comes to mind when dealing with counts data is the Poisson regression model. We will reset the training data and remove two outliers with regard to the response variable that lie above the 99% quantile and fit a Poisson regression with all variables using `glm(total_medals_won ~ ..., family = poisson(link = "log"), data = train.data)`. Looking at the `summary(poisson.regression)`, we can see that there seem to be many predictors that are not significant. We will therefore remove the predictor with the highest p-value, re-fit the model, analyze the summary and repeat this procedure until we only have significant predictors left.

In our case, the remaining predictors are `comm, gdp, altitude and athletes` and the model has an RMSE of 4.08 (see Appendix B).

However, the model has a residual deviance of 252.43 on 101 degrees of freedom. We can calculate the dispersion parameter with `sum(resid(poisson.regression, type = "pearson")^2) / X2/poisson.regression$df.res` which yields 3.05. It seems that we might have a problem with overdispersion.

There can be various reasons why we could encounter overdispersion here, such as:

- The model might be misspecified, in the sense that there might be a missing interaction term, a missing non-linear relationship or a missing predictor
- Excessive zeros in the response variable
- The theoretical Poisson distribution simply does not fit our data well

Since our goal is to develop a model that can accurately predict the number of medals won by each country in the 2016 Olympics, we will focus on dealing with the overdispersion and try to find an alternative model that makes better predictions.

There are many different ways to deal with overdispersion, such as the Quasi-Poisson and the Negative-Binomial regression model. Let's first look at the Quasi-Poisson regression.

7.3 Quasi-Poisson Regression

We will start by fitting a Quasi-Poisson regression model, which introduces a dispersion parameter that can be used to adjust the standard errors in the summary of our Poisson regression, using `glm(total_medals_won ~ ..., family = quasipoisson(link = "log"), data = train.data)`. With the use of the estimated dispersion parameter, the Wald tests are not very reliable, so we turn to an F test to determine the significance of the regression coefficients and remove non-significant predictors from the model stepwise. When we repeat this procedure, we finally end up with a Quasi-Poisson regression model with `comm, host and athletes` as predictors (see Appendix C).

Unfortunately, the deviance is still very large compared to the degrees of freedom (and even higher than the Poisson regression) and the RMSE of the Quasi-Poisson model is 4.88, which is higher than both the linear and the Poisson regression model.

Finally, let's also try to fit a Negative-Binomial regression model to our data.

7.4 Negative-Binomial Regression

We can fit a Negative-Binomial, which introduces an additional free parameter and allows for an adjusted variance, by using the `glm.nb()` function from the `MASS` package. Similarly as before, we will stepwise select predictors that are significant. Additionally, we can also compare the deviance and AIC for different link functions such as the default log-link and the sqrt-link. The latter led to better results in this example and using the method described, we end up with a model using only `athletes` as its predictor and an RMSE 6.48 which is higher than all other models (see Appendix D). In contrast, the Negative-Binomial model has a lower AIC of 499 and a lower deviance of 120 compared to the Poisson regression which has an AIC of 543 and a deviance of 252.

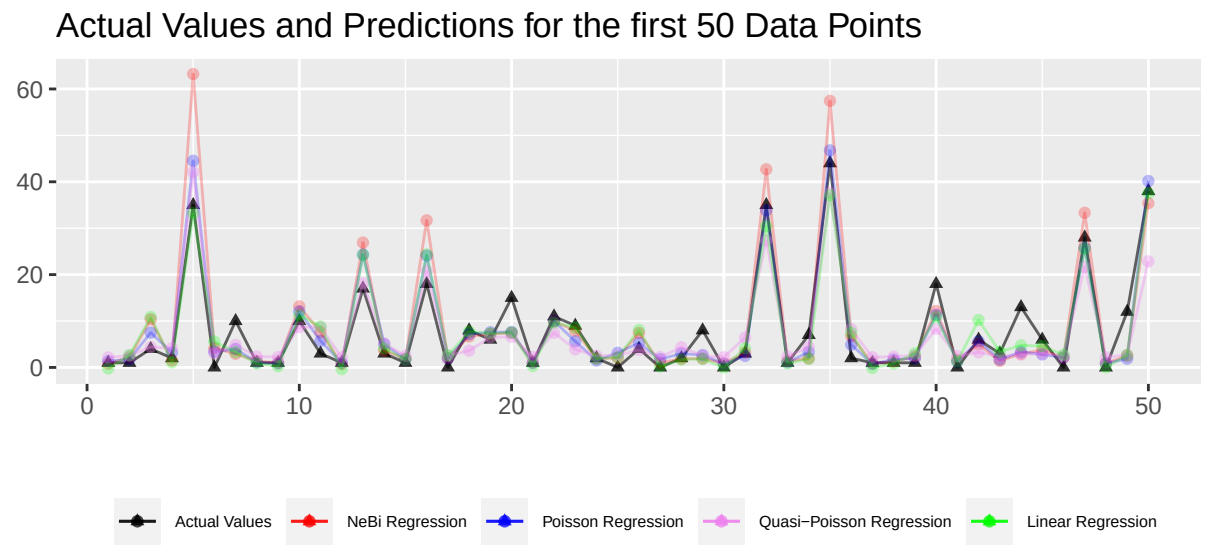
8. Model Evaluation

Let's look at the selected variables and the RMSE for each of the four models we have built:

Model	Variables	RMSE
Linear	athletes, gdp	3.80
Poisson	athletes, gdp, comm, altitude	4.08
Quasi-Poisson	athletes, comm, host	4.88
Negative-Binomial	athletes	6.48

Interestingly, the linear regression model appears to have the highest predictive power (on the training set), despite the fact that many of it's assumptions are dubious.

Let's plot the performance of the models:

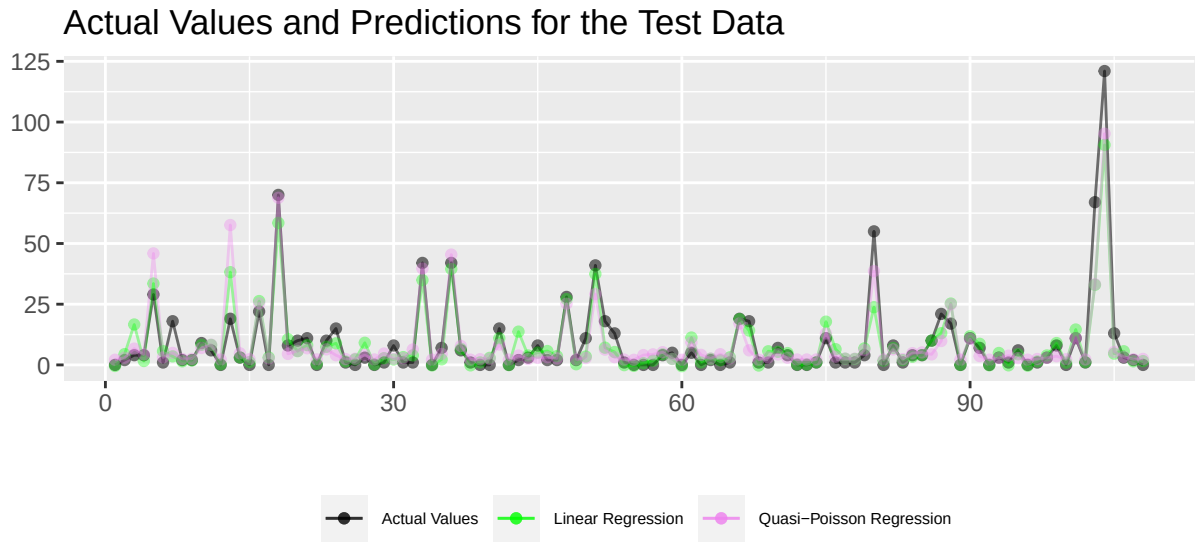


As we can see from the plot above, all models actually seem to do a fairly good job predicting the data points in the training set. However, the Negative-Binomial regression seems to overestimate the performance of some countries by a large amount, which leads to the higher RMSE on the training set. We can also see that the Poisson- and the Quasi-Poisson regression models seem to

have estimates that are quite close to each other in many cases, but the Poisson regression seems tends to have higher estimates than the Quasi-Poisson regression. Let's take the linear regression model as the model with the lowest RMSE and the Quasi-Poisson regression model as the model with the lowest RMSE of the models for counts data that ease the assumption of equal mean and variance and see how the two models perform on the test data set.

9. Predictions

First of all, let's calculate the RMSE for the linear and the Quasi-Poisson regression model on the test data set. We get an RMSE of 6.90 for the linear regression and 7.20 for the Quasi-Poisson regression model. It is not surprising that both model perform slightly worse on the test set compared to the training set. Let's plot the predictions of both models versus the actual data in the test data set:



As we can see, both models seem to be able to predict the total number of medals won by each country at the 2016 Olympic Games quite well. This is especially surprising with regard to the simple linear model that only uses the number of competing athletes and the GDP of each country. From the plot above, we can also see that the linear model makes a few negative predictions. While this is theoretically problematic, in practice we could simply take negative predictions as a prediction of zero.

Let's calculate for how many of the 108 countries in the test set, the two models made a prediction within the range of 2, within the range of 5 and outside the range of 5 from the actual. Let's furthermore label these predictions as `Good Predcition`, `Moderate Prediction` and `Bad Prediction`. The two models would then be evaluated as:

	Linear Regression	Quasi-Poisson Regression
Good	66	65
Moderate	22	26
Bad	20	17

As we can see, the linear regression model makes slightly more predictions that are within the range of 2 to the actual values, but it also makes more predictions were it's prediction is more than 5 off from the actual values, compared to the Quasi-Poisson regression. Nevertheless, it is surprising to see how both model can predict the actual number of medals won by each country within a range of only 2 in 65/66, and within a range of 5 in 88/91 out of 108 cases.

10. Conclusions

We have fitted four different models trying to predict the total number of medals that each country won at the 2016 Olympics. Surprisingly, a simple linear regression model that only used the GDP of the country and the number of athletes as predictors was able to predict this fairly well. We compared its performance to a Quasi-Poisson regression model that used the number of athletes and whether or not a country has ever been the host of the Olympics and whether it has ever been a communist state. The Quasi-Poisson regression model also performed quite well.

Despite the theoretical limitations of the linear model with regard to the dubious assumptions mentioned above, we might actually prefer it in practice, due to its simplicity and ease of interpretation. Nevertheless, many things could be improved to find a better performing model for this particular prediction task.

Tweak Models The first thing that could be done is to further tweak our existing models, for example by investigating whether any of the following tweaks could lead to better predictions:

- Introduce different interaction terms into our model
- Apply different transformations on our data
- Introduce non-linear terms into the models
- Use different link-functions
- Experiment with different approaches of variable selection

Fit New Models Another option we have is to fit other types of models on our data. As there are quite many countries that did not win any medal in the Olympic games, we might go ahead and try out Zero-Inflated models or Hurdle models next. Of course, we could also try completely different kinds of models for this prediction problem, such as Regression Trees or a simple Artificial Neural Network.

Get More Data Last but not least, we should also consider including more data into our model. For starters, we only have data on 108 of the 207 nations that took part in the Olympics. More importantly, we might think of other variables that influence the number of medals a country wins at the Olympics and include them into our data, for example the amount of money that is being invested by each country into national sports promotions programs, how many sports facilities a country has etc.

Appendix

A. Summary of linear Regression

```
##
## Call:
## lm(formula = total_medals_won ~ athletes + gdp, data = train.data)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -7.2404 -1.8520  0.0373  0.9948 18.2574
##
## Coefficients:
##               Estimate Std. Error t value Pr(>|t|)
## (Intercept) -7.022e-01  5.116e-01  -1.373    0.173
## athletes     7.367e-02  6.342e-03  11.617  < 2e-16 ***
## gdp          2.707e-06  6.413e-07   4.220 5.34e-05 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 3.853 on 101 degrees of freedom
## Multiple R-squared:  0.8157, Adjusted R-squared:  0.812
## F-statistic: 223.5 on 2 and 101 DF,  p-value: < 2.2e-16
```

B. Summary of Poisson Regression

```
##
## Call:
## glm(formula = total_medals_won ~ comm + gdp + altitude + athletes,
##     family = poisson(link = "sqrt"), data = train.data)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -2.5988  -1.3113  -0.5826   0.3124   6.1357
##
## Coefficients:
##              Estimate Std. Error z value Pr(>|z|)
## (Intercept) 7.633e-01  7.895e-02   9.668  < 2e-16 ***
## comm1       5.266e-01  1.202e-01   4.381 1.18e-05 ***
## gdp         2.736e-07  8.649e-08   3.163  0.00156 **
## altitude    1.776e-04  7.800e-05   2.277  0.02276 *
## athletes    1.332e-02  7.425e-04  17.934  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for poisson family taken to be 1)
##
##     Null deviance: 1396.66  on 105  degrees of freedom
## Residual deviance:  252.43  on 101  degrees of freedom
## AIC: 542.59
##
## Number of Fisher Scoring iterations: 5
```

C. Summary of Quasi-Poisson Regression

```
##
## Call:
## glm(formula = total_medals_won ~ comm + host + athletes, family = quasipoisson(link = "log"),
##     data = train.data)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -2.8530  -1.9824  -0.8924   0.5794   5.3820
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept) 0.7394129  0.1297838   5.697 1.18e-07 ***
## comm1       0.5561297  0.1610091   3.454 0.000806 ***
## host1       0.8184607  0.2468099   3.316 0.001265 **
## athletes    0.0054021  0.0006671   8.098 1.25e-12 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for quasipoisson family taken to be 3.500819)
##
##     Null deviance: 1396.66  on 105  degrees of freedom
## Residual deviance:  339.04  on 102  degrees of freedom
## AIC: NA
##
## Number of Fisher Scoring iterations: 5
```

D. Summary of Negative-Binomial Regression

```
##
## Call:
## glm.nb(formula = total_medals_won ~ athletes, data = train.data,
##     link = sqrt, init.theta = 2.867337387)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -2.2630  -1.1653  -0.2667   0.2607   2.6886
##
## Coefficients:
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept) 0.793597   0.091315   8.691   <2e-16 ***
## athletes    0.017717   0.001433  12.363   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for Negative Binomial(2.8673) family taken to be 1)
##
##     Null deviance: 413.45  on 105  degrees of freedom
## Residual deviance: 119.55  on 104  degrees of freedom
## AIC: 498.58
##
## Number of Fisher Scoring iterations: 1
##
##
##               Theta:  2.867
##           Std. Err.:  0.769
##
##  2 x log-likelihood:  -492.584
```