Loan repayment plan using `R` and `Python`

Andrija Djurovic*

*www.linkedin.com/in/andrija-djurovic

The following example illustrates creating a loan repayment plan using `R` and `Python`.

Loan inputs:

```
#loan amount
amount <- 5000
#maturity (in months)
maturity <- 18
#yearly interest rate
ir.y <- 0.0649
#monthly interest rate
ir.m <- ir.y / 12
```

User-defined function that generates the loan repayment plan:

```
create.repayment.plan <- function(p, r, m) {
      #p - loan amount
      #r - monthly interest rate
      #m - loan maturity in months

      #annuity
      annuity <- p * r / (1 - (1 + r)^(-m))
      #prepare the repayment plan data frame
      rp <- data.frame(month = 1:m,
                       remaining.principal = NA,
                       monthly.principal = NA,
                       monthly.interest = NA,
                       annuity = rep(annuity, m))
      #assign the loan amount as a starting principal
      rp$remaining.principal[1] <- p
      #construct the repaymen plan
      for   (i in 1:m) {
            if    (i == m) {
                  rp$monthly.principal[i] <- rp$remaining.principal[i]
                  rp$monthly.interest[i] <- rp$annuity[i] -
                                            rp$monthly.principal[i]
                  } else {
                  rp$monthly.interest[i] <- rp$remaining.principal[i] * r
                  rp$monthly.principal[i] <- rp$annuity[i] -
                                             rp$monthly.interest[i]
                  rp$remaining.principal[i + 1] <- rp$remaining.principal[i] -
```

```
                                                    rp$monthly.principal[i]
                }
            }
return(rp)
}
```

Loan repayment plan:

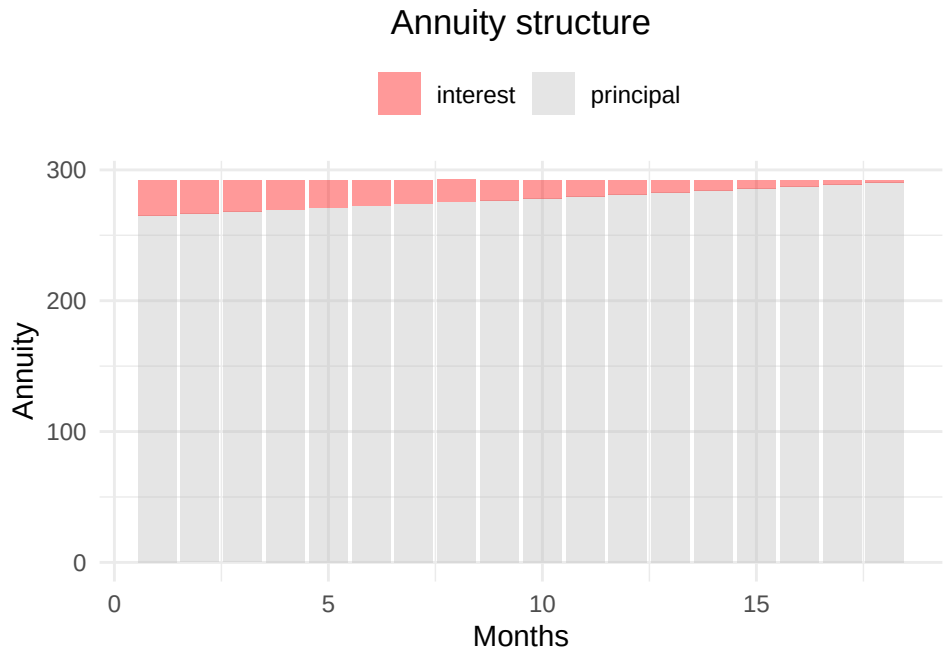
```
rp <- create.repayment.plan(p = amount,
                            r = ir.m,
                            m = maturity)
rp
```

```
##    month remaining.principal monthly.principal monthly.interest  annuity
## 1      1           5000.0000          265.2262        27.041667 292.2678
## 2      2           4734.7738          266.6606        25.607235 292.2678
## 3      3           4468.1132          268.1028        24.165046 292.2678
## 4      4           4200.0104          269.5528        22.715056 292.2678
## 5      5           3930.4576          271.0106        21.257225 292.2678
## 6      6           3659.4470          272.4763        19.791509 292.2678
## 7      7           3386.9707          273.9500        18.317866 292.2678
## 8      8           3113.0207          275.4316        16.836254 292.2678
## 9      9           2837.5891          276.9212        15.346628 292.2678
## 10    10           2560.6679          278.4189        13.848946 292.2678
## 11    11           2282.2490          279.9247        12.343163 292.2678
## 12    12           2002.3243          281.4386        10.829237 292.2678
## 13    13           1720.8857          282.9607         9.307124 292.2678
## 14    14           1437.9250          284.4911         7.776778 292.2678
## 15    15           1153.4339          286.0297         6.238155 292.2678
## 16    16            867.4042          287.5766         4.691211 292.2678
## 17    17            579.8276          289.1319         3.135901 292.2678
## 18    18            290.6957          290.6957         1.572179 292.2678
```

```
#check net present value of the cash flow
sum(rp$annuity / ((1 + ir.m)^(1:maturity)))
```

```
## [1] 5000
```

Annuity structure:



An alternative approach is available for those seeking a more advanced solution that offers precise values for the remaining principal, principal, and interest portion in an annuity at a certain period. The subsequent code demonstrates the calculation of the principal portion in the annuity for the 5th month. The remaining steps in the repayment plan calculation are straightforward and are left for the reader to complete.

```
#annuity
a <-  amount * ir.m / (1 - (1 + ir.m)^(-maturity))
#define the nth period
n <- 5
#principal portion in the annuity at the 5th period
(a - amount * ir.m) * ((1 + ir.m)^(n - 1))
```

```
## [1] 271.0106
```

Loan repayment plan in `Python`:

```
import pandas as pd
import numpy as np

#input parameters
#loan amount
amount = 5000
#maturity (in months)
```

```
maturity = 18
#yearly interest rate
ir_y = 0.0649
#monthly interest rate
ir_m = ir_y / 12

#loan repayment function
def create_repayment_plan(p, r, m):
    #p - loan amount
    #r - monthly interest rate
    #m - loan maturity in months

    #annuity
    annuity = p * r / (1 - (1 + r)**(-m))
    #prepare the repayment plan data frame
    rp = pd.DataFrame({"month" : [*range(1, m + 1)],
                       "remaining_principal" : [None]*m,
                       "monthly_principal" : [None]*m,
                       "monthly_interest" : [None]*m,
                       "annuity" : [annuity]*m
                      })
    #assign loan amount as a starting principal
    rp.loc[0, "remaining_principal"] = p
    #construct the repaymen plan
    for i in rp.index:
        if (i == rp.index[-1]):
            rp.loc[i, "monthly_principal"] = rp.remaining_principal[i]
            rp.loc[i, "monthly_interest"] = rp.annuity[i] - \
                                            rp.monthly_principal[i]
        else:
            rp.loc[i, "monthly_interest"] = rp.remaining_principal[i] * r
            rp.loc[i, "monthly_principal"] = rp.annuity[i] - \
                                             rp.monthly_interest[i]
            rp.loc[i + 1, "remaining_principal"] = rp.remaining_principal[i] - \
                                                   rp.monthly_principal[i]
    return(rp)


#create repayment plan
```

```
rp = create_repayment_plan(p = amount,
                           r = ir_m,
                           m = maturity)
rp
```

```
##     month remaining_principal monthly_principal monthly_interest     annuity
## 0       1                5000        265.226177        27.041667  292.267843
## 1       2         4734.773823        266.660608        25.607235  292.267843
## 2       3         4468.113215        268.102798        24.165046  292.267843
## 3       4         4200.010417        269.552787        22.715056  292.267843
## 4       5          3930.45763        271.010618        21.257225  292.267843
## 5       6         3659.447012        272.476334        19.791509  292.267843
## 6       7         3386.970678        273.949977        18.317866  292.267843
## 7       8         3113.020701         275.43159        16.836254  292.267843
## 8       9         2837.589112        276.921216        15.346628  292.267843
## 9      10         2560.667896        278.418898        13.848946  292.267843
## 10     11         2282.248998         279.92468        12.343163  292.267843
## 11     12         2002.324318        281.438606        10.829237  292.267843
## 12     13         1720.885712         282.96072         9.307124  292.267843
## 13     14         1437.924992        284.491066         7.776778  292.267843
## 14     15         1153.433927        286.029688         6.238155  292.267843
## 15     16          867.404239        287.576632         4.691211  292.267843
## 16     17          579.827607        289.131942         3.135901  292.267843
## 17     18          290.695664        290.695664         1.572179  292.267843
```

```
#check net present value of the cash flow
np.sum(rp["annuity"] / (1 + ir_m) ** np.arange(1, maturity + 1))
```

```
## 5000.000000000088
```

```
#annuity
a = amount * ir_m / (1 - (1 + ir_m)**(-maturity))
#define the nth period
n = 5
#principal portion in the annuity at the nth period
(a - amount * ir_m) * ((1 + ir_m)**(n - 1))
```

```
## 271.0106182999124
```