STONE CHALLENGE - FASE 2

1) Objetivos

Objetivo primário

Desenvolver uma metodologia para avaliar a efetividade na comunicação com clientes que apresentam atraso na quitação do saldo devedor, avaliando os diferentes perfis para cada cliente.

Objetivos secundários:

1. Prever um fluxo de pagamento sustentável, com base no fluxo de caixa do cliente e levando em conta a sazonalidade.
2. Análise de correlações que possam trazer algum valor para o processo de comunicação atual.
3. Explorar possíveis correlações de dados internos da Stone com dados de fontes públicas.

2) Validações Iniciais

a) Fazendo joins de todas as tabelas (ajustando as relações one-to-many) entre as chaves, teríamos as seguintes etapas:

1. *portfolio_geral* ⇒ 14265 clientes distintos
2. portfolio_geral inner join *portfolio_comunicados* ⇒ 11871 clientes distintos
3. merge anterior inner join *portfolio_tpv* ⇒ 11865 clientes distintos
4. merge anterior inner join com *portfolio_clientes* ⇒ 11865 clientes distintos

b) Seguem os principais filtros que aplicamos para garantir a integridade dos dados analisados:

Foram removidos todos os contratos que com DSP > DSPP na *portfolio_geral.* Inicialmente foram retirados 30 contratos com inconsistências nos dados.

Como iremos fazer os cálculos de de ciclos com base na variável DSP, fizemos uma validação, para cada contrato, da variação entre DSP entre dois registros consecutivos. Para todos os registros em que DSP é diferente de **zero**, a variação consecutiva de DSP entre dois registros deve ser obrigatoriamente 1 dentro do mesmo contrato. Aplicando essa validação, no total foram removidos mais alguns contratos da análise, totalizando 457 removidos da tabela.

Na tabela de *portfolio_comunicados,* removemos todos os comunicados que não foram entregues aos clientes, conforme veremos em detalhes.

3) Fluxo de Etapas Realizadas

A fim de mensurar a efetividade das ações de comunicação, primeiramente iremos focar nossa análise nas tabelas de *portfolio_comunicados* e *portfolio_geral* , pois será a partir dessas tabela que iremos inferir a nossa variável target (relacionada com a efetividade da comunicação) que será utilizada na nossa modelagem preditiva.

Para o carregamento e manipulação dos dados, utilizamos o pacote *data.table* da linguagem R. Esse pacote apresenta um processamento de dados otimizado e robusto, permitindo o processamento de grandes volumes de dados. Após carregarmos as 4 tabelas, fizemos uma ordenação pelas respectivas variáveis chaves de cada uma, a fim de garantir que os registros dos nossos conjuntos de dados estejam organizados da forma correta.

```
#carregamento dos arquivos
portfolio_clientes = fread(paste0(caminho_arquivos,'\\portfolio_clientes.csv'), encoding = "UTF-8")
portfolio_comunicados = fread(paste0(caminho_arquivos,'\\portfolio_comunicados.csv'), encoding = "UTF-8")
portfolio_geral = fread(paste0(caminho_arquivos,'\\portfolio_geral.csv'), encoding = "UTF-8")
portfolio_tpv = fread(paste0(caminho_arquivos,'\\portfolio_tpv.csv'), encoding = "UTF-8")
```

Desenvolvemos uma função para percorrer o nosso dataset e identificar a ocorrência de valores missing. Aplicamos essa função aos 4 datasets compartilhados e não encontramos nenhum valor missing.

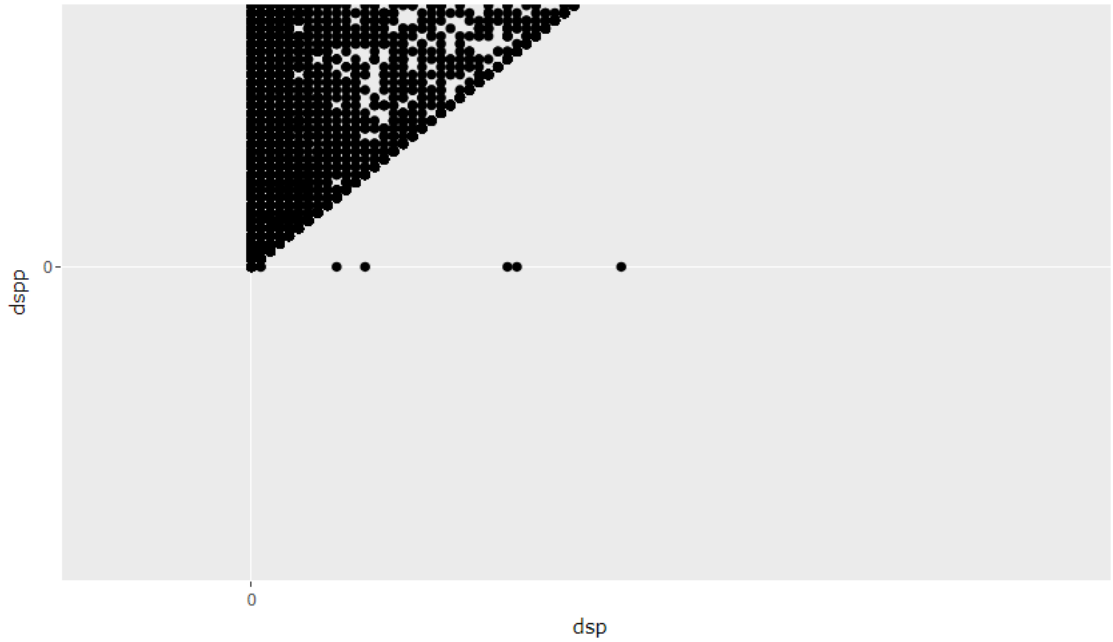
Uma boa prática para verificar se as variáveis estão codificadas da forma correta, é fazer uma contagem dos valores distintos para cada variável. Em geral, quando temos muitos valores distintos, a variável pode se tratar de uma variável numérica. Portanto, fizemos essa análise para todas as tabelas. Segue abaixo, como exemplo, as primeiras 5 variáveis com mais valores distintos para a tabela *portfolio_comunicados* :

VARIÁVEL	TIPO	VALORES ÚNICOS
vlr_saldo_devedor_esperado	numeric	3050274
vlr_saldo_devedor	numeric	2879232
vlr_pgto_realizado	numeric	111430
contrato_id	character	14756
nr_documento	character	14265

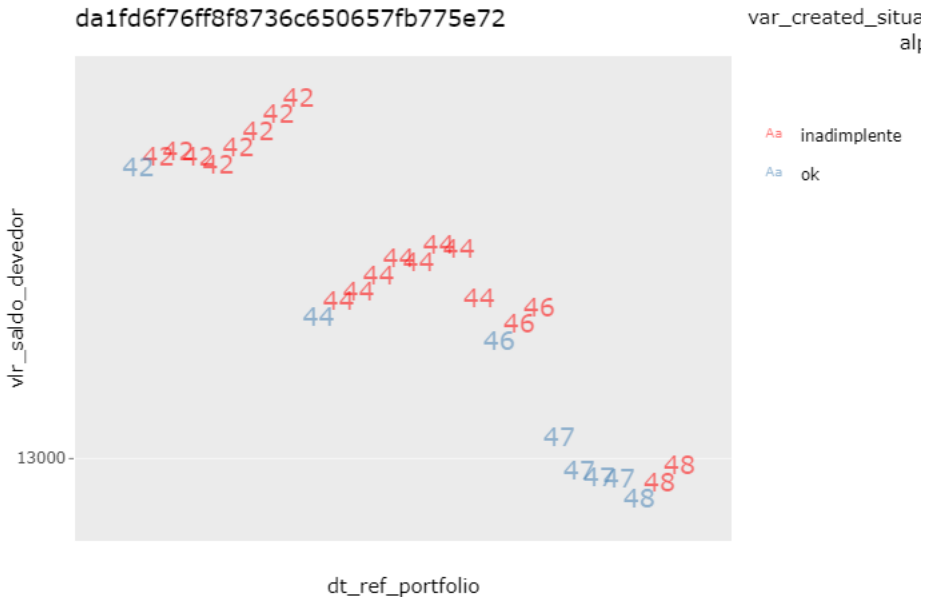
Notamos que alguns clientes possuem mais de um contrato em aberto. Por esse motivo, iremos criar uma variável que contabiliza o número de contratos disponíveis por clientes, conforme o script abaixo:

```
portfolio_geral[,var_created_contratos_por_cliente:=uniqueN(contrato_id), by = nr_documento]
```

Encontramos no total de 706 registros de DSP que estão com valores inconsistentes (DSPP>DSP), como podemos notar pelo gráfico abaixo. Os 30 contratos que apresentaram essa inconsistência foram removidos.



Notamos que a variável *vlr_saldo_devedor* apresenta ciclos, conforme o pagamento do cliente, ou seja, quando DSPP = 0 ou DSP = 0. Sendo assim, nosso desafio inicial será desenvolver uma forma de mapear esses ciclos de pagamento para cada contrato, tendo em vista o objetivo de entender como funciona essa dinâmica do fluxo de pagamentos. Na imagem abaixo, temos a representação dos ciclos de pagamento para um cliente. Esses ciclos foram gerados a partir de um algoritmo que desenvolvemos e que será detalhado mais adiante. Os números representam cada ciclo. A cor azul representa quando há pagamento do principal; e o vermelho representa os dias que o cliente está inadimplente considerando o critério da DSPP. Apesar dos ciclos serem definidos quando há algum pagamento, seja do principal ou não, por questão de processamento de dados (e de inconsistências nos valores de DSP), iremos definir apenas as transições quando há o pagamento do principal (DSPP=0). Veremos uma forma de incorporar a DSP nos cálculos.



Inicialmente vamos criar as variáveis a variável categórica *var_created_situacao_inadimplencia*, para facilitar a identificação da situação diária do cliente conforme a métrica de inadimplência DSPP. Basicamente, teremos o valor '**ok**' para quando houver a quitação do principal. O código abaixo resume esse processo. Note que, o valor de '**erro**' pode ser usado para remover as inconsistências já mencionadas (DSPP>DSP).

```
portfolio_geral[,var_created_situacao_inadimplencia:=fcase(

  dspp == 0 & dsp == 0, "ok",
  dspp > 0 & dsp  > 0, "inadimplente",
  dspp > 0 & dsp  == 0, "inadimplente",
  default = "erro"

)]
```

Notamos que para a tabela *portfolio_comunicados*, as chaves mostradas no diagrama de MER (*contrato_id* e *dt_ref_portfolio*) apresentam valores repetidos. Por conta disso, teremos de tratar essa tabela com intuito de ter apenas valores únicos nas chaves antes fazer o join essa tabela com a *portfolio_geral*. Os passos para essa finalidade estão descritos em detalhes na tabela abaixo:

Conforme vemos na imagem abaixo, a coluna *tipo_acao* contém informações que podem ser separadas em duas variáveis distintas (2 colunas separadas). Para isso, vamos converter a tabela para o formato wide, usando a função *dcast*.

contrato_id	dt_ref_portfolio	data_acao	tipo_acao	acao	status	conta
000c35a61297edadc2842f6d5b4028e1	2021-04-24	2021-04-26	HSM	campanhaboletoquitado	ENTREGUE	4
000c35a61297edadc2842f6d5b4028e1	2021-04-24	2021-04-26	HSM	campanhaobservacao	NAO ENTREGUE	4
000c35a61297edadc2842f6d5b4028e1	2021-04-24	2021-04-26	EMAIL	campanhaboletoquitado	NAO ENTREGUE	4
000c35a61297edadc2842f6d5b4028e1	2021-04-24	2021-04-26	EMAIL	campanhaobservacao	ENTREGUE	4

```
comunicados_wide = dcast(portfolio_comunicados,formula = contrato_id + dt_ref_portfolio + data_acao + acao ~ tipo_acao, value.var = 'status
```

Mesmo após transformação da tabela, ainda temos alguns registros **duplicados nas chaves**. Após análises, percebemos ocorre por conta do conflito entre as duas réguas. Teremos de lidar com essa questão antes de trabalhar com esses dados.

contrato_id	dt_ref_portfolio	data_acao	acao	EMAIL	HSM	conta
000c35a61297edadc2842f6d5b4028e1	2021-04-24	2021-04-26	campanhaboletoquitado	NAO ENTREGUE	ENTREGUE	2
000c35a61297edadc2842f6d5b4028e1	2021-04-24	2021-04-26	campanhaobservacao	ENTREGUE	NAO ENTREGUE	2
000c35a61297edadc2842f6d5b4028e1	2021-05-09	2021-05-11	campanhaparcelamento	ENTREGUE	ENTREGUE	2
000c35a61297edadc2842f6d5b4028e1	2021-05-09	2021-05-11	campanhaprenegativacao	NAO ENTREGUE	LIDO	2

Como nosso objetivo é tratar da efetividade da comunicação, faremos as suposições abaixo para os valores de comunicação. Logo, todos os valores de '**LIDO**' / '**RESPONDIDO**' serão considerados como ações efetivas ('**OK**'), ou seja, que muito provavelmente tiveram algum impacto no nosso cliente.

VALOR	NOVO VALOR
LIDO	OK
RESPONDIDO	OK
ENTREGUE	NAO_CONCLUSIVO
NAO ENTREGUE	FALHA

Como nosso objetivo é tratar da efetividade da comunicação, podemos aplicar um filtro na tabela de comunicados para manter somente os registros que tiveram êxito na comunicação em pelo menos algum dos canais (EMAIL e HSM), considerando o novo valor como **OK**. Essas linhas poderão ter algum valor preditivo devido à efetividade na comunicação. O script abaixo irá fazer somente o filtro dos comunicados efetivos, que serão usados na nossa análise.

```
comunicados_wide = comunicados_wide[EMAIL_EFETIVO=='OK' | HSM_EFETIVO == 'OK']
```

O próximo passo foi fazer um filtro na tabela de *portfolio_geral* para considerar **somente os clientes que foram comunicados pelo menos uma vez,** com base na tabela de *comunicados_wide.* Atribuímos a essa tabela filtrada o nome de *portifolio_analisado*. Após isso, iremos performar um left join (*portifolio_analisado x comunicados_wide*). Como há chaves duplicadas na tabela *comunicados_wide,* haverá um aumento no total de registros (relação one-to-many), o que teremos de tratar posteriormente. Como já mencionamos, essa duplicidade se trata quando as 2 réguas apresentam resultados conflitantes. Vamos armazenar esse join na tabela temporária/auxiliar *target_intermed*, de onde iremos identificar qual o tipo de campanha foi informado ao cliente e resolver a ambiguidade do valor *campanhaboletoquitado*). Os códigos para performar o processo descrito estão abaixo:

```
#Filtro de contatos com pelo menos 1 contato
contratos_questionados = unique(comunicados_wide[,contrato_id])

#Aplicacao do filtro na tabela portifolio geral
portifolio_analisado = copy(portfolio_geral[contrato_id %in% contratos_questionados,.(contrato_id,dt_ref_portfolio,nr_documento,dsp,dspp,re

#Left Join portfolio x comunicados
target_intermed = merge(portifolio_analisado[,.(contrato_id,dt_ref_portfolio,dsp,dspp)],comunicados_wide[,.(contrato_id,dt_ref_portfolio,ac
```

Como agora temos os valores de DSP e DSSP atrelados aos valores de ação da comunicação, iremos fazer a codificação conforme os dados expressos nas réguas, a fim de eliminar a ambiguidade que há com a valor de ***campanhaboletoquitado*** (ocorre em DSP = 15 e DSP = 90, mas pode representar campanhas diferentes). Usando uma estrutura '**case when**', criamos uma escala de campanha (de 1 a 6), está descrito abaixo:

```
target_intermed[,codificao_acao_comunicado:=fcase(

  dsp==5  & acao == 'campanhaobservacao',1,
  dsp==10 & acao ==  'campanhaparcelamento',2,
  dsp==15 & acao == 'campanhaboletoquitado',3,
  dsp==30 & acao == 'campanhaprenegativacao',4,
  dsp==60 & acao ==  'campanhanegativacao',5,
  dsp==90 & acao == 'campanhaboletoquitado',6,
```

```
  dspp==15 & acao == 'campanhaobservacao',1,
  dspp==30 & acao ==  'campanhaparcelamento',2,
  dspp==45 & acao == 'campanhaboletoquitado',3,
  default = NA

)]
```

Existem dias em que os clientes foram aparentemente contactados 2 vezes devido a conflitos entre as duas réguas. Visto isso, vamos fazer a suposição de que a **campanha mais drástica (conforme a escala acima) é o mais representativo**. Por exemplo, se o cliente receber no mesmo dia um aviso de negativação e outro de observação, será considerada apenas a negativação). Dessa forma, iremos fazer um filtro na nossa tabela, **resolvendo o problema da chave duplicada**. Seguem abaixo os códigos para executar essa operação:

```
#Selecao da acao mais drastica para o mesmo dia (caso haja 2)
target_intermed[,acao_mais_drastica:=max(codificao_acao_comunicado), by=.(contrato_id,dt_ref_portfolio)]

#Aplicacao da selecao mais drastica
target_intermed = target_intermed[contatos_por_dia==1 | (codificao_acao_comunicado==acao_mais_drastica)]

#Limpeza das linhas nulas e das colunas nao necessarias
target_intermed = target_intermed[!is.na(acao_mais_drastica)]
target_intermed = unique(target_intermed[,.(contrato_id,data_acao,contatos_por_dia,acao_mais_drastica)])
```

Note que agora temos exatamente qual foi a campanha mais drástica (sem ambiguidade) que foi informada ao cliente em relação ao dia exato da comunicação. Como poder ser visto na tabela abaixo, o cliente **000dcdc93a545ee45a1aee85ef85c34a**, em 10/11/2020, foi contactado com duas campanhas (**contatos_por_dia = 2**), com a campanha de **boleto quitado (3)** e **campanha de observação (1)**. Neste exemplo, adotaremos a campanha mais drástica, que foi o boleto quitado, que será criada em uma nova variável, **campanha_mais_drastica = 3**.

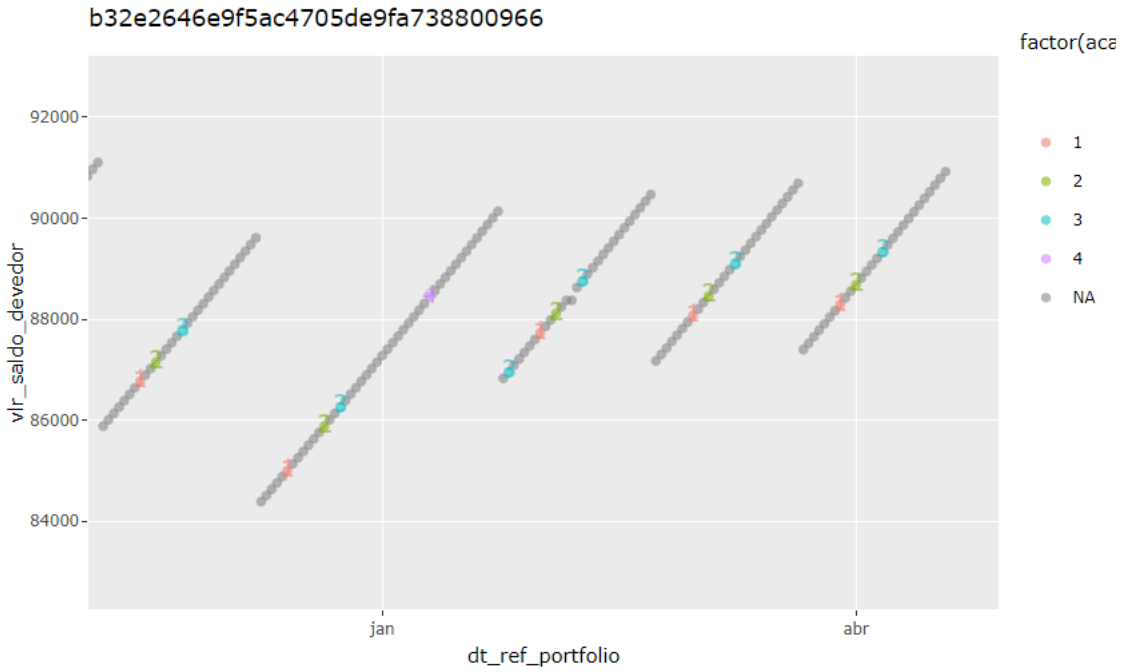
dt_ref_portfolio	dsp	dspp	data_acao	acao	contatos_por_dia	codificao_acao_comunicado	campanha_mais_drastica
	All	All		All	All	All	All
2020-11-09	14	14	NA	NA	1	NA	NA
2020-11-10	15	15	2020-11-10	campanhaboletoquitado	2	3	3
2020-11-10	15	15	2020-11-10	campanhaobservacao	2	1	3
2020-11-11	16	16	NA	NA	1	NA	NA
2020-11-12	17	17	NA	NA	1	NA	NA
2020-11-13	18	18	NA	NA	1	NA	NA

Agora iremos performar um novo left join entre a tabela *target_intermed* (contém a informação codificada do tipo de campanha e sem ambiguidade) e a tabela *portifolio_analisado* (filtro da *portfolio_geral*). Com base nisso, teremos a junção da informação de **comunicação da campanha** com os **dados financeiros para cada cliente**, o que vai nos ajudar a obter a variável target (relação entre comunicação e efetividade do pagamento). Após esse join, obtivemos a tabela *target,* que tem 5.616.176 registros. Podemos agora fazer o join com a tabela *portfolio_tpv*, para agregar as informações de transações à nossa análise.

```
#left Join entre os derivados das tabelas de comunicados e a geral
target = merge(portifolio_analisado,target_intermed, by.x = c('contrato_id','dt_ref_portfolio'),by.y = c('contrato_id','data_acao') , all.x

#left join com a target com a portfolio_tpv
target = merge(target, portfolio_tpv, by.x = c('nr_documento','dt_ref_portfolio'), by.y = c('nr_documento','dt_transacao'),all.x = T, suffi

#ordenando a target
target = target[order(contrato_id,dt_ref_portfolio)]
```

A tabela de *portfolio_clientes* apresenta problema de chaves duplicadas. Por esse motivo iremos deixá-la para uma segunda etapa e focar apenas na tabela *target*, que é a basicamente uma junção das outras 3 tabelas.

Observamos um forte padrão cíclico na Iremos focar nos ciclos de pagamento dos clientes com base na variável **vlr_saldo_devedor.** Ou seja, sempre que há algum tipo de pagamento, a dívida é recalculada é determinada de forma linear, conforme podemos observar na imagem abaixo. No caso, podemos ver 5 ciclos completos (5 retas). Os números representam o momento a campanha e, conforme vimos, 1 = observação, 2 = observação e assim por diante. Note que os ciclos de pagamento podem apresentam padrões de repetição, e poderemos usar isso em nossa modelagem preditiva.



A partir de agora, iremos trabalhar com os dados agregados em ciclos. O script abaixo faz essa identificação dos ciclos tomando como critério os pontos onde DSPP = 0. A variável *dia_ciclo* irá definir uma contagem para cada ciclo criado, ou seja, a partir do dia do pagamento, teremos uma contagem de dias. Essa variável será extremamente útil na determinação da nossa variável alvo, conforme veremos mais adiante.

```
#determinacao os ciclos continuos (pagamento/divida)
target[,ciclo_ativo:=rleid(var_created_situacao_inadimplencia), by = contrato_id]

#incluindo o dia de pagamento (ok) no inicio do ciclo de divida
target[,ciclo_ativo:=ifelse(var_created_situacao_inadimplencia=='ok' & shift(var_created_situacao_inadimplencia, type = 'lead')!='ok', shif

#incluindo o dia de pagamento parcial no inicio do cilo da divida
target[,ciclo_ativo:=ifelse(var_created_situacao_inadimplencia=='inadimplente_ativo' & shift(var_created_situacao_inadimplencia, type = 'le

#criando uma sequencia de dias para cada ciclo
target[,dia_ciclo:=sequence(.N),by = .(ciclo_ativo,contrato_id)]
```

Agora, iremos agregar a tabela de *target* a nível de **ciclo**. Basicamente estamos agregando as variáveis numéricas em somas, mínimos, máximos ou médias e valores de data inicial/final, com base em cada um dos ciclos identificados para cada contrato. O

tipo de operação de agrupamento das variáveis foi feito após um estudo de como a variável se comporta dentro de um ciclo. O script abaixo resume todas as operações que fizemos.

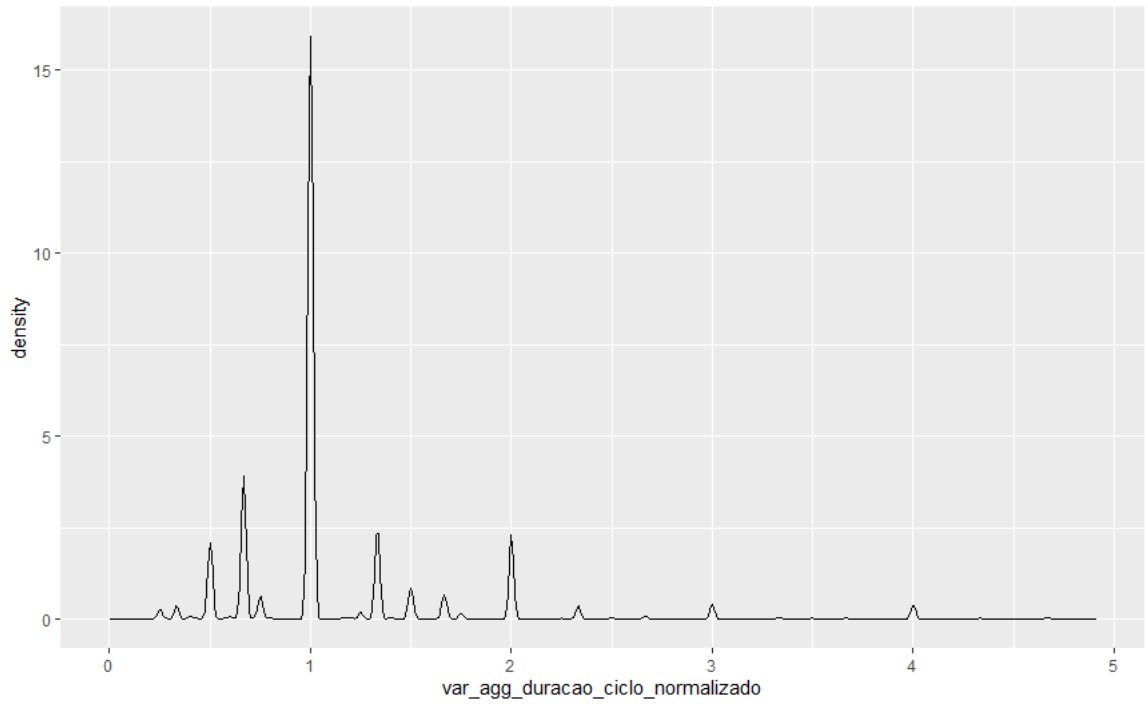
```
#fazendo uma copia da tabela target
preditivas_agregadas = copy(target)

#definindo as variaveis chave
vars_chave = c('contrato_id','ciclo_ativo')

#agregando valores
preditivas_agregadas[,var_agg_dt_ref_portfolio_min:=min(dt_ref_portfolio), by = .(ciclo_ativo,contrato_id) ]
preditivas_agregadas[,var_agg_dt_ref_portfolio_max:=max(dt_ref_portfolio), by = .(ciclo_ativo,contrato_id) ]
preditivas_agregadas[,var_agg_vlr_pgto_realizado_sum:=sum(vlr_pgto_realizado, na.rm = T),by = .(ciclo_ativo,contrato_id)]
preditivas_agregadas[,var_agg_dsp_max:=max(dsp), by = .(ciclo_ativo,contrato_id) ]
preditivas_agregadas[,var_agg_flag_transacao_med:=mean(flag_transacao), by = .(ciclo_ativo,contrato_id) ]
preditivas_agregadas[,var_agg_qtd_transacoes_sum:=sum(qtd_transacoes, na.rm = T),by = .(ciclo_ativo,contrato_id)]
preditivas_agregadas[,var_agg_vlr_tpv_sum:=sum(vlr_tpv, na.rm = T),by = .(ciclo_ativo,contrato_id)]
preditivas_agregadas[,var_agg_duracao_ciclo:=.N, by = .(ciclo_ativo,contrato_id) ]
preditivas_agregadas[,var_agg_num_pagamentos_parciais:=sum(dsp_binaria), by = .(ciclo_ativo,contrato_id) ]
preditivas_agregadas[,var_agg_campanha_mais_drastica_ciclo:=max(campanha_mais_drastica, na.rm = T), by = .(ciclo_ativo,contrato_id) ]
preditivas_agregadas[,var_agg_campanha_mais_drastica_ciclo:=ifelse(is.infinite(var_agg_campanha_mais_drastica_ciclo),0,var_agg_campanha_mai

variveis_agregadas = stringr::str_subset(colnames(preditivas_agregadas),'var_agg_')

#Aplicando o filtro de colunas e executando o distinct
filtro_colunas = c(vars_chave,variveis_agregadas)
preditivas_agregadas = unique(preditivas_agregadas[ ,colnames(preditivas_agregadas) %in% filtro_colunas, with=FALSE])
setcolorder(preditivas_agregadas,filtro_colunas)


#valores iniciais e finais para cada ciclo
estado_inicial = copy(target[,.SD[1],by = .(ciclo_ativo,contrato_id),.SDcols=c('var_created_situacao_inadimplencia','vlr_saldo_devedor')])
estado_final = copy(target[,.SD[.N],by = .(ciclo_ativo,contrato_id),.SDcols=c('var_created_situacao_inadimplencia','vlr_saldo_devedor')])

#definindo novos nomes para colunas
setnames(estado_inicial, old = c('var_created_situacao_inadimplencia','vlr_saldo_devedor'), new = c('situacao_inicial','var_agg_vlr_saldo_d
setnames(estado_final, old = c('var_created_situacao_inadimplencia','vlr_saldo_devedor'), new = c('situacao_final','var_agg_vlr_saldo_deved


#status inicial de cada ciclo
status_inicial = copy(target[,.SD[1],by = .(ciclo_ativo,contrato_id),.SDcols=c('status_contrato','prazo','juros_mes','perc_retencao')])


#agregando os dados
preditivas_agregadas = merge(preditivas_agregadas,estado_inicial, by=c('ciclo_ativo','contrato_id'))
preditivas_agregadas = merge(preditivas_agregadas,estado_final, by=c('ciclo_ativo','contrato_id'))
preditivas_agregadas = merge(preditivas_agregadas,status_inicial, by=c('ciclo_ativo','contrato_id'))
```

Note que, como fizemos a nossa separação com ciclos em DSPP, mas na verdade sabemos que o ciclos de dívida são determinados pelo DSP, tivemos de criar uma forma de incorporar esses subciclos de DSP dentro dos ciclos de DSPP. Nesse caso, tivemos de binarizar a variável *dsp* (*dsp_binaria*) e fazer uma agrupamento na variável *var_agg_num_pagamentos_parciais*, de forma que temos o número de pagamentos parciais (DSP=0) dentro do ciclo de DSPP.

Após o processo, a nova tabela *preditivas_agregadas* apresentou 620.481 registros. Conforme podemos ver na tabela abaixo, nossos dados agora estão devidamente organizados em ciclos:

ciclo_ativo	contrato_id	var_agg_dt_ref_portfolio_min	var_agg_dt_ref_portfolio_max	var_agg_vlr_pgto_realizado_sum
2	000180509391a5ac66ff83cae603ffb8	2020-06-15	2020-06-17	25.43
3	000180509391a5ac66ff83cae603ffb8	2020-06-18	2020-06-18	41.40
4	000180509391a5ac66ff83cae603ffb8	2020-06-19	2020-06-21	61.49
5	000180509391a5ac66ff83cae603ffb8	2020-06-22	2020-06-25	257.05
6	000180509391a5ac66ff83cae603ffb8	2020-06-26	2020-07-01	117.51
7	000180509391a5ac66ff83cae603ffb8	2020-07-02	2020-07-02	64.19
8	000180509391a5ac66ff83cae603ffb8	2020-07-03	2020-07-06	88.62
9	000180509391a5ac66ff83cae603ffb8	2020-07-07	2020-07-07	43.28
10	000180509391a5ac66ff83cae603ffb8	2020-07-08	2020-07-16	182.74
12	000180509391a5ac66ff83cae603ffb8	2020-07-17	2020-07-20	121.67
13	000180509391a5ac66ff83cae603ffb8	2020-07-21	2020-07-23	140.24
14	000180509391a5ac66ff83cae603ffb8	2020-07-24	2020-07-26	74.23
15	000180509391a5ac66ff83cae603ffb8	2020-07-27	2020-07-29	151.33
16	000180509391a5ac66ff83cae603ffb8	2020-07-30	2020-08-02	55.60
17	000180509391a5ac66ff83cae603ffb8	2020-08-03	2020-08-06	149.20
18	000180509391a5ac66ff83cae603ffb8	2020-08-07	2020-08-09	52.99

Identificamos um padrão interessante nos dados, normalizando a duração de cada ciclo do contrato pela duração mediana. O gráfico abaixo representa a distribuição dessa variável normalizada (var_agg_duracao_ciclo_mediana). Notamos que a maioria dos ciclos se concentram próximos da mediana (pico em 1) e que os demais ciclos se concentram em valores específicos, como o dobro da mediana (pico em 2). Por esse motivo, utilizaremos essa variável como preditora na nossa modelagem, pois temos uma padrão para saber se o ciclo está acima ou abaixo dos ciclos históricos para o mesmo contrato.



Agora iremos fazer a limpeza na tabela de *portfolio_clientes*. Apenas os últimos dados cadastrados serão considerados por clientes. Faremos um join com um filtro a tabela *portfolio_geral* para obter a chave por contrato, conforme explicitamos no script

abaixo:

```
#limpando a tabela clientes -----
clientes_limpa = unique(portfolio_clientes[,.(nr_documento,tipo_empresa,estado,subsegmento)])
clientes_limpa = clientes_limpa[,.SD[.N],.SDcols=c('tipo_empresa','estado','subsegmento'),by=nr_documento]
chave_cliente_contrato = unique(portfolio_geral[,.(nr_documento,contrato_id)])
clientes_limpa = merge(clientes_limpa,chave_cliente_contrato, by = 'nr_documento')
```

Quando trabalhamos com machine learning, mais especificamente com aprendizado supervisionado, buscamos analisar padrões nos dados históricos para treinar um modelo preditivo, que nos ajudará a fazer previsões do comportamento futuro quando recebermos novos dados. Para termos insumos para realizar o aprendizado, primeiramente iremos deslocar temporalmente a nossa tabela de *preditivas_agregadas*, que contém informações a nível ciclo. Para isso, desenvolvemos o script automatizado, onde podemos controlar o número de informações históricas que obteremos. Nesse caso, iremos trabalhar com **3 registros predecessores para cada ciclo**, porém esse número pode ser facilmente modificado conforme o intuito do analista. Note que, fizemos diversos filtros com o objetivo de reduzir o ruídos nos dados. Foram removidos os ciclos de pagamentos seguidos (dias consecutivos aonde DSPP=0); foram considerados somente os ciclos com no mínimo 10 observações por contrato (garantir que tenhamos uma amostra significativa por contrato); iremos considerar somente o período ativo no contrato e, por fim, iremos limitar o nossa range de análise para os ciclos de 5 a 120 dias. O código abaixo realiza todo o processo descrito nesse parágrafo:

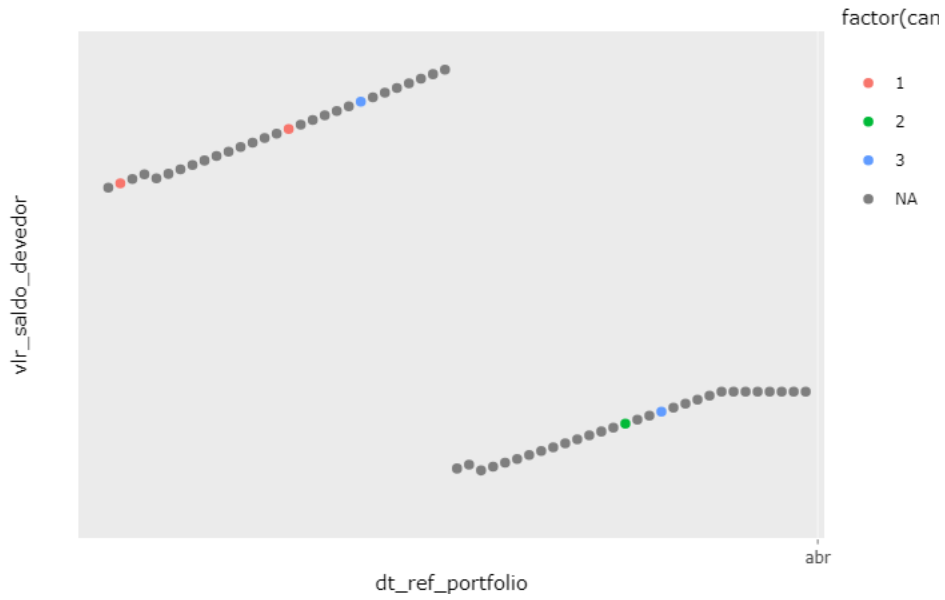
```
preditivas_lag = copy(preditivas_agregadas)

variaveis_preditivas_historicas = c('var_agg_qtd_transacoes_sum','var_agg_vlr_pgto_realizado_sum','var_agg_vlr_tpv_sum','var_agg_duracao_ci

variaveis_chave_lag = c('ciclo_ativo','contrato_id')

summary(preditivas_lag)

#filtros
#considera somente ciclos de inadimplencia
preditivas_lag = preditivas_lag[situacao_final!='ok']
#numero de obs mínima de 10
preditivas_lag = preditivas_lag[var_agg_tipo_ciclo_numero_obs>10]
#filtro de somente contratos ativos
preditivas_lag = preditivas_lag[status_contrato=='Active']
#determinacao do range de interesse
preditivas_lag = preditivas_lag[var_agg_duracao_ciclo>=5]
preditivas_lag = preditivas_lag[var_agg_duracao_ciclo<=120]


#correcao de inconsistencias
colunas_preditivas_numericas=get_num_cols(preditivas_lag)

for (coluna in colunas_preditivas_numericas){

  preditivas_lag[,(coluna):=ifelse(get(coluna)>0,get(coluna),0)]


}

summary(preditivas_lag)

#filtrar numero minimo de obs para cada ciclo


for (lag_var in 1:3) {

  print(lag_var)


  for (coluna_preditiva_historica in variaveis_preditivas_historicas){


    preditivas_lag[,(paste0('lag_',lag_var,'_',coluna_preditiva_historica)):=shift(get(coluna_preditiva_historica),n = lag_var,type = 'lag'
```

```
  }


  #fazendo variaveis por variacao percentual
  # preditivas_lag[,(paste0('lag_',lag_var,'_','var_agg_vlr_saldo_devedor_inicial')):=log( (var_agg_vlr_saldo_devedor_inicial+1)/(shift(var
  preditivas_lag[,(paste0('lag_',lag_var,'_','var_agg_vlr_saldo_devedor_inicial')):=( (var_agg_vlr_saldo_devedor_inicial)-(shift(var_agg_vl


  #considerar apenas o anterior, pois caso contrario vai usar dados futuros
  # preditivas_lag[,(paste0('lag_',lag_var,'_','spread_vlr_saldo_devedor')):=log( (shift(var_agg_vlr_saldo_devedor_final,n = lag_var,type =
  preditivas_lag[,(paste0('lag_',lag_var,'_','spread_vlr_saldo_devedor')):=( (shift(var_agg_vlr_saldo_devedor_final,n = lag_var,type = 'lag


}


#retirando as linhas com NA (sem info historica minima)
preditivas_lag = na.omit(preditivas_lag)

#removendo variaveis futuras
variaveis_lag = stringr::str_subset(colnames(preditivas_lag),'^lag_')

filtro_colunas_lag = c(variaveis_chave_lag,variaveis_lag)


preditivas_lag = preditivas_lag[ ,colnames(preditivas_lag) %in% filtro_colunas_lag, with=FALSE]
setcolorder(preditivas_lag,filtro_colunas_lag)
```

Agora que já temos a nossa tabela principal com as variáveis preditivas (preditivas_lag), iremos focar no cálculo da variável target, que será obtida a partir da relação entre o tipo de campanha executada em cada regime. Com base na tabela *target*, agora iremos extrair uma outra tabela, chamada *estado_presente*, porém com filtro apenas nos dias em que houve alguma comunicação efetiva com o cliente. Nessa tabela, além de determinar a nossa variável alvo, usaremos as informações presentes que estavam disponíveis na data daquele comunicado. Removeremos os últimos ciclos para cada contrato, pois eles ainda não estão completos.

```
estado_presente = unique(copy(target[!is.na(campanha_mais_drastica),.(contrato_id,ciclo_ativo,dt_ref_portfolio,campanha_mais_drastica,dia_c
estado_presente[,ultimo_ciclo:=max(ciclo_ativo), by = .(contrato_id)]
estado_presente = estado_presente[status_contrato=='Active']

#ultimo comunicado do ciclo
estado_presente[,ultimo_comunicado_flag:=ifelse(dia_contato==max(dia_contato),T,F), by = .(contrato_id,ciclo_ativo)]

#remove o ulimo ciclo
estado_presente = estado_presente[ciclo_ativo!=ultimo_ciclo]
estado_presente[,tempo_resposta_cliente:=duracao_ciclo-dia_contato]
```

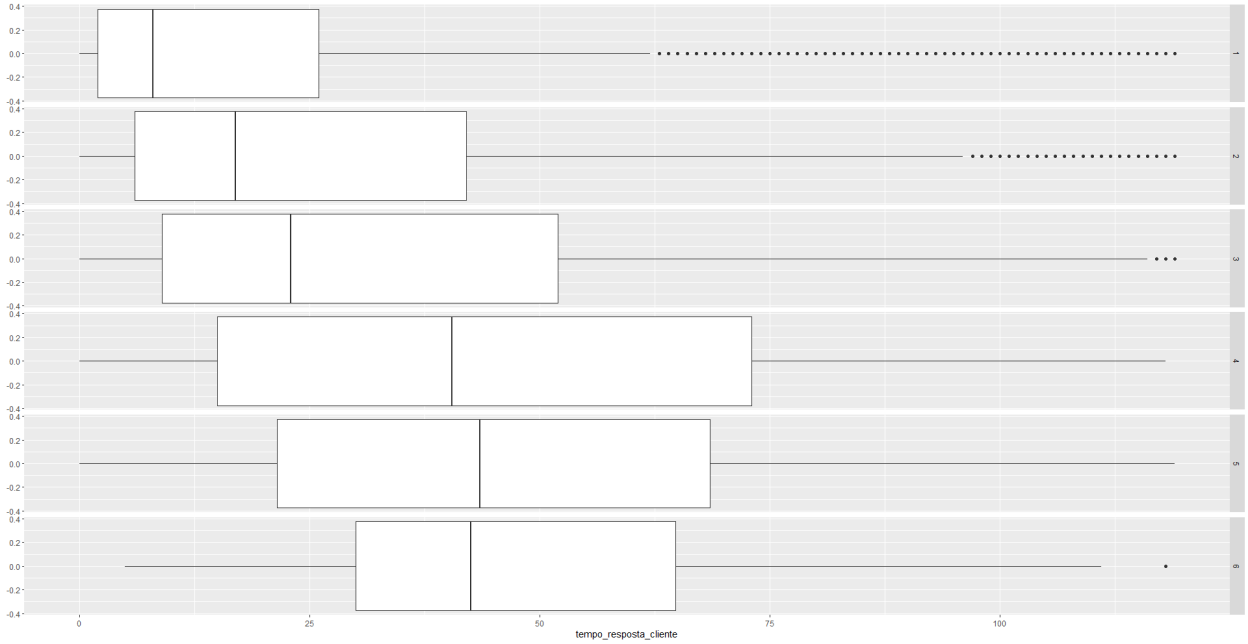
Agora, iremos dedicar esforços para determinar a nossa variável alvo. Primeiramente, vamos analisar os dados para o cliente 5724f8503dc5bc301ce9bb6819b761e4. Note que no período analisado, 01/02/2021 até 31/03/2021, tivemos 3 campanhas praticadas nos dois ciclos representados: campanha de observação (1), campanha de parcelamento (2) e campanha de boleto quitado (3).



Para esses 2 ciclos representados, definiremos o tempo de resposta como a diferença entre a duração de ciclo até o pagamento do principal (***duracao_ciclo***) e dia que o cliente foi contactado (***dia_ciclo***), conforme podemos ver na tabela abaixo:

dt_ref_portfolio	ciclo_ativo	campanha_mais_drastica	duracao_ciclo	dia_ciclo	tempo_resposta
2021-02-02	16	1	29	2	27
2021-02-16	16	1	29	16	13
2021-02-22	16	3	29	22	7
2021-03-16	20	2	28	13	15
2021-03-19	20	3	28	16	12

Conforme podemos verificar pelo gráfico abaixo, em geral as campanhas mais drásticas tem um tempo de resposta maior, ou seja, o cliente demora um pouco mais na quitação do principal:



Teremos de determinar um limite para considerar que ação foi efetiva. Dessa forma, estabeleceremos os limites abaixo para o tempo de resposta. Se cliente fizer a quitação até esse limite de dias especificado, iremos considerar que realmente a campanha teve efeito, caso contrário a comunicação será considerada sem efeito.

campanha_mais_drastica	tempo_resposta_minimo
1	5
2	5
3	10
4	10
5	10
6	10

Pelo código abaixo, finalmente podemos definir a nossa variável alvo (**comunicacao_efetiva**)! Essa varíavel é booleana (verdeiro ou falso) e será usada no nosso modelo de machine learning.

```
#extraindo todos os comunciados ----
estado_presente = unique(copy(target[!is.na(campanha_mais_drastica),.(contrato_id,ciclo_ativo,dt_ref_portfolio,campanha_mais_drastica,dia_c
estado_presente[,ultimo_ciclo:=max(ciclo_ativo), by = .(contrato_id)]
estado_presente = estado_presente[status_contrato=='Active']

#ultimo comunicado do ciclo
estado_presente[,ultimo_comunicado_flag:=ifelse(dia_contato==max(dia_contato),T,F), by = .(contrato_id,ciclo_ativo)]

#remove o ulimo ciclo
estado_presente = estado_presente[ciclo_ativo!=ultimo_ciclo]

#caculo do tempo de resposta
```

```
estado_presente[,tempo_resposta_cliente:=duracao_ciclo-dia_contato]

#definindo finalmente a nossa variavel target
estado_presente[,comunicacao_efetiva:=fcase(
  campanha_mais_drastica==1 & tempo_resposta_cliente < 5, T,
  campanha_mais_drastica==2 & tempo_resposta_cliente < 5, T,
  campanha_mais_drastica==3 & tempo_resposta_cliente < 10, T,
  campanha_mais_drastica==4 & tempo_resposta_cliente < 10, T,
  campanha_mais_drastica==5 & tempo_resposta_cliente < 10, T,
  campanha_mais_drastica==6 & tempo_resposta_cliente < 10, T,
  default = F)
]
```

Para avaliar se há algum padrão sazonal na data de comunicação, iremos criar uma variável que representa o mês do ano. Note que usamos essa variável como categórica (factor):

```
estado_presente[,mes_comunicado:=as.factor(month(dt_ref_portfolio))]
```

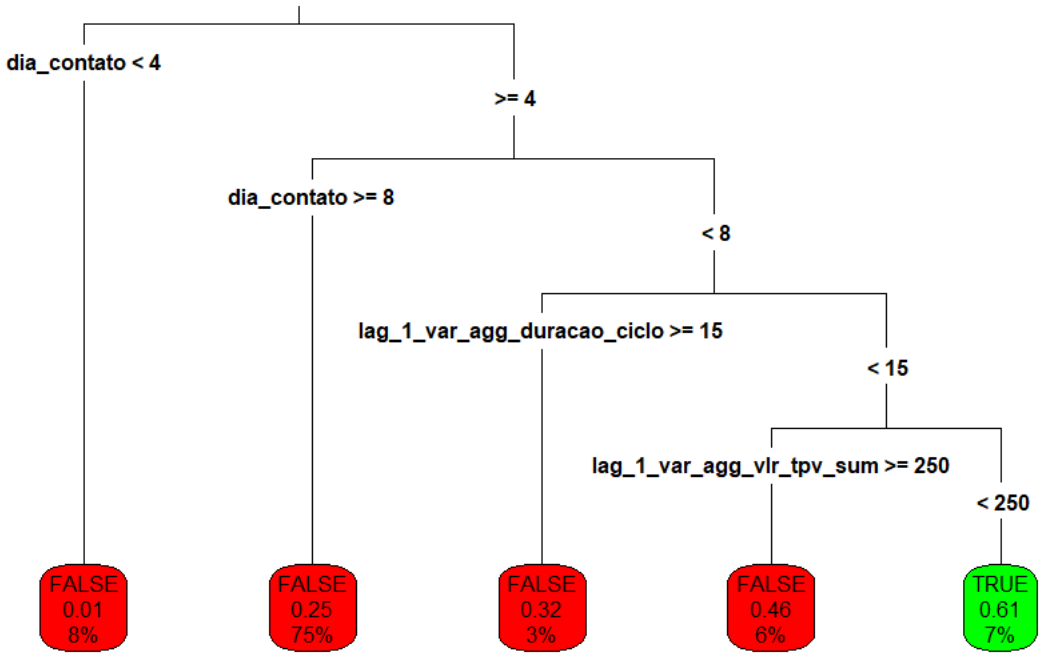
Agora, iremos unir o nosso dataset contendo a variável alvo (*estado_presente*) com o dataset que contém as variáveis preditoras, criando *target_final,* que contém 13.290 registros. Nessa tabela, temos os dados das 4 tabelas originais devidamente tratados e prontos para a nossa análise preditiva!

4) Insights

Iniciaremos nossa modelagem com um modelo mais simples de árvore de decisão, implementando o código abaixo. Note que o ***method = "class"*** indica que estamos fazendo uma classificação e que a nossa variável target é a ***comunicacao_efetiva*** .

```
arvore_previsao_target_final = rpart::rpart(data = target_final,
   method = "class",
   formula = comunicacao_efetiva ~ .
)
```

Nosso modelo gerou a seguinte árvore:



Em vermelho estão as comunicações que, segundo os nossos critérios, não foram efetivas; e em verde as que surtiram algum possível efeito. Note que as variáveis denotadas com '*lag_1*' são as variáveis das informações do ciclo anterior, ou seja, o modelo está nos dizendo que a duração do ciclo e a soma TPV do ciclo anterior (lembre-se que trabalhamos com até 3 valores passados) nos dizem muito de como o pagamento de um cliente vai ser comportar no próximo ciclo.

Aplicando o modelo nos dados, obtivemos uma acurácia de 74,1%. Fazendo separação em treino (70% dos dados) e teste (30% dos dados), podemos fazer uma avaliação mais justa do modelo: 75.1%. Logo, esse modelo está aparentemente generalizável para esse conjunto de dados, ou seja, se aplicarmos esse modelo aos dados, a cada 75 de 100, ele irá acertar se teremos ou não uma comunicação efetiva.

Agora, vamos aplicar um outro modelo mais complexo, o Random Forest, que é composto de diversas árvores de decisão. Note que usaremos 100 árvores de decisão no nosso modelo!

```
random_forest_final = ranger::ranger(formula  = comunicacao_efetiva ~ .,
                         target_final,
                         num.trees = 100
                         )
```

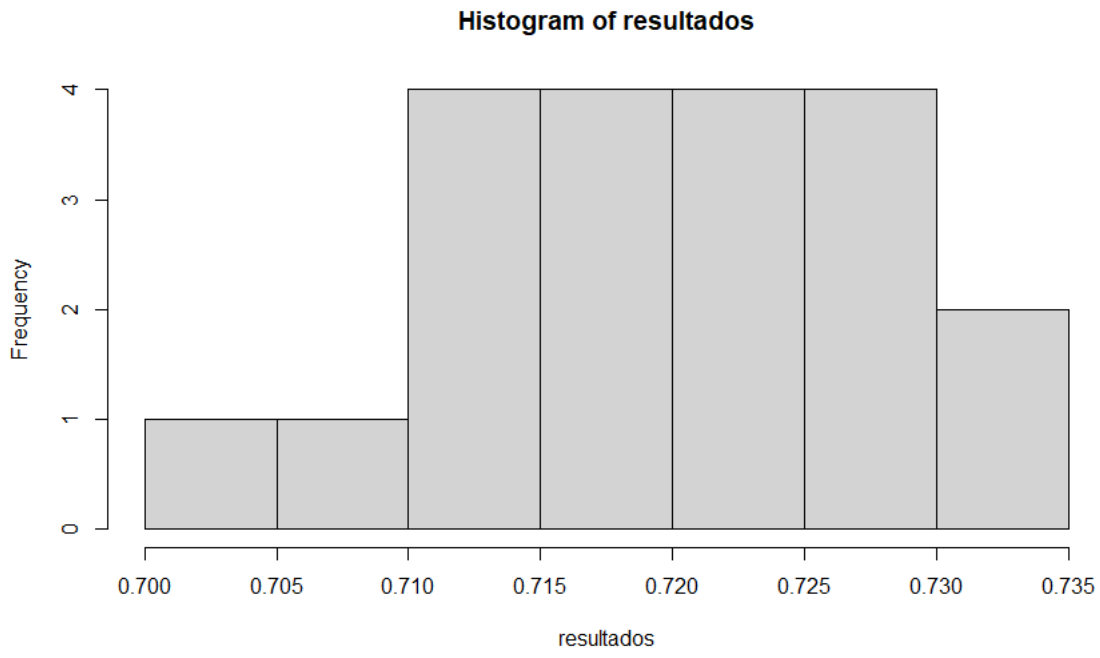
Aplicando esse modelo aos dados, tivemos uma acurácia de 99,9%! Porém, essa não é uma avaliação justa, pois o modelo está sendo avaliado dados que ele foi treinado, logo essas performance pode estar superestimada.

Vamos fazer 20 validações separando e treino (80%) e teste (20%) para verificar a performance real:

```
resultados = c()

for (i in 1:20) {

target_final_split = rsample::initial_split(target_final,prop = 0.80)
```

```
training_target_final = training(target_final_split)

testing_target_final = testing(target_final_split)


random_forest_treino = ranger::ranger(formula  = comunicacao_efetiva ~ .,
                                      training_target_final,
                                      num.trees = 100
)



rm(resultado)
resultado = data.table(pred=predict(random_forest_treino, testing_target_final)[[1]],
                       testing_target_final[,.(comunicacao_efetiva)])

resultado[,comunicacao_efetiva:=as.numeric(comunicacao_efetiva)]
result_int = mean(resultado[,pred]==resultado[,comunicacao_efetiva])

resultados =  c(resultados,result_int)
rm(result_int)
}
```

A performance mediana do nosso modelo foi de 71.9%, não sendo relativamente melhor do que a árvore de decisão que criamos. Segue abaixo o histograma com resultados obtidos das 20 passadas:



5) Considerações sobre os objetivos secundários

Em um segundo momento, podemos pensar em incluir fontes de dados pública na tomada de decisão. Porém, para o desenvolvimento do nosso modelo de Machine Learning, esse não será o caminho ideal por dois motivos: Primeiro, porque vamos criar uma dependência externa em relação ao processo de tomada de decisão, o que pode ser prejudicial caso essas fontes não atualizem os dados. Segundo, porque iremos incluir mais estimadores no nosso modelo, o que pode enviesar ainda mais a nossa decisão, sendo ideal incluir o menor número de variáveis possíveis para tomarmos as decisões. Apenas complementar o processo de tomada de decisão conforme a previsão do modelo, podemos posteriormente validar algumas fontes de dados macroeconômicas, como por exemplo:

- Previsões da taxa Selic, IPCA, PIB informadas semanalmente no boletim Focus. Podemos automatizar a extração desses dados com linguagem R.
- Análise dos PMI.
- Fazer uma estimativa da necessidade de capital de giro para cada segmento, a fim suavizar o pagamento das empresas que têm ramos de atuação mais arriscados. Dados de balanço de empresas de capital aberto podem ser utilizados para isso.
- Taxa de desemprego do IBGE
- Monitoramento dos agregados monetários M1, M2 etc, fornecidos regularmente pelo Banco Central

6) Conclusões e Insights

Conseguimos desenvolver um framework que nos permite analisar os dados com base nos ciclos de pagamento. Com os dados criados na tabela *target_final*, além dos dois modelos que fizemos, poderemos desenvolver muitos outros, pois temos os dados devidamente preparados para isso.

O modelo de árvore de classificação se mostrou promissor para avaliar o problema. Apesar de todo script e a modelagem apresentarem um framework poderoso para a solução do modelo, ainda não temos uma resposta definitiva para a solução final e enxergamos 2 oportunidades de melhoria: Primeiramente, o processo de modelagem está apenas no começo (teremos de realizar outras etapas mais rigorosas, como a validação cruzada, tune dos hiperparâmetros com Grid Search etc) para garantir que temos um modelo generalizável e tentar aumentar a nossa acurácia. Segundo, precisaremos do apoio da equipe da Stone para avaliar a qualidade das inúmeras suposições que fizemos para progredir como o problema. Até agora, temos um modelo razoável que apenas nos responde, se, para um determinado cliente, caso fizer o contato com alguma campanha, se terá êxito ou não na quitação do principal. Em outras palavras, conseguimos saber relativamente bem, com base nos dados históricos e no modelo de árvore de decisão, se teremos êxito comunicando o cliente em um determinado dia.