Pequeno Guia com indicações e sugestões sobre o projeto de TIAPOSE 2023 (versão 1.2):

1) Na execução do trabalho em grupo deve haver uma maior responsabilização do trabalho individual. Quem "sabe" de antemão que vai faltar a uma sessão prática deve contactar os restantes elementos do grupo, justificando a falta, bem como entregando e explicando o resultado do seu trabalho individual. Isto para que o grupo possa explicar ao docente o que o colega ausente realmente fez. Por outro lado, aquilo que aparece no "semanário" deve corresponder à realidade e não ser algo inventado ("para não parecer mal"). Todas dúvidas que tenham devem estar sumariadas no final da entrada do semanário. Todos resultados relevantes, devem estar via "copy e paste" na entrada do semanário. Quando um grupo faz a sua apresentação do trabalho, aquilo que apresentam deve ser coerente. Não faz sentido mostrar 3 portáteis, cada um com código diferente e que não é coerente entre si. Uma boa prática que poderiam instituir é ter 2 reuniões por semana de sessão de aulas. Uma mais inicial, para distribuir tarefas, e outra antes da sessão da aula, para agruparem os trabalhos individuais de um modo mais agregado e terem uma única visão do que é o vosso trabalho em grupo. Estas reuniões não têm de ser presenciais e nem têm de ser muito longas. E todos deveriam estar presentes nestas reuniões.

2) Na escrita do código R, devem usar sempre que possível bons princípios de programação (Engenharia de Software). Por exemplo, não faz muito sentido ter 10 ficheiros quase iguais onde uma a três linhas de código são diferentes. Também não faz muito sentido ter diversos pedaços de código que são praticamente iguais e repetidos devido ao uso exagerado de "*copy & paste*". É preferível ter um código mais genérico e reduzido, com uso de objetos em memória ("variáveis") que possam ser instanciadas de modos diferentes, conforme o que se pretende executar, e numa única *script* R, com reuso de funções. Isto evita erros de programação, simplifica o ter-se uma "única visão" para o grupo e facilita a interação com o docente.

   Eis um pequeno exemplo (que não tem a ver com este projeto):

```
# auxiliar show function (confusion matrixes and some classification measures):
showres=function(model,Y,Pred)
{
 cat("> ",model,":\n")
 print(mmetric(Y,PDT,metric="CONF")$conf)
 print(round(mmetric(Y,Pred,metric=c("ACC","ACCLASS"))))
 print(round(mmetric(Y,Pred,metric=c("macroF1","F1"))))
 cat(" --- \n")
}

model="rpart"; showres(model,Y,PDT)
model="randomForest"; showres(model,Y,PRF)
model="mlpe"; showres(model,Y,PNN)
```

   Neste caso, os objectos Y, PDT, PRF e PNN foram já previamente criados. As três últimas linhas executam aquilo que sem a função auxiliar exigiria 5x3=15 linhas. Realço que este

exemplo nem está muito otimizado, as três últimas linhas poderiam ser substituídas por esta linha:

```
for(i in 1:length(model)) showres(model[i],Y,Pred[[i]])
```

No caso dos objectos model (vector) e Pred (vector list) serem indexáveis, tipo:

```
model=c("rpart","randomForest","mlpe")
Pred=vector("list",3); Pred[[1]]=PDT; Pred[[2]]=PRF; Pred[[3]]=PNN
```

3) Para a parte de previsão, devem explorar métodos univariados (mais fáceis de serem codificados em R) e depois, se tiverem tempo, multivariados.

   Os métodos univariados podem ser implementados com 2 pacotes R: **forecast** e **rminer** (entre outros). Cada pacote tem um modo próprio de funcionamento. No caso **forecast**, devem criar um objeto do tipo **ts**, se usarem o **rminer** devem criar um **data.frame** via função **CasesSeries**. As *scripts* R demonstradas nas aulas exemplificam como usar ambos tipos de modelos univariados de previsão. As previsões *multi-step ahead* (em avanço) de um método do pacote **forecast** são obtidas via a função **forecast,** enquanto que as previsões *multi-step ahead* do pacote **rminer** são obtidas via função **lforecast**. Para mais explicações dos métodos, consultar estes links:

   forecast: https://otexts.com/fpp2/
   rminer: https://dx.doi.org/10.21814/1822.36210

   Os métodos multivariados assumem mais do que um atributo temporal para prever outro atributo temporal. Para a base de dados fornecida, existem múltiplas combinações. Na aula de previsão, foi demonstrado como usar 3 métodos multivariados: 1) VAR; 2) ARIMAX e 3) Machine Learning (ML) via pacote **rminer.** As demonstrações assumem somente 2 variáveis em simultâneo para prever os valores futuros das mesmas 2 variáveis. No que diz respeito ao método 3), ML, não se pode usar a função **lforecast**, porque esta não foi concebida para modelos de previsão multivariados. Por isso, tem de usar a função **predict**, construindo um objecto a prever (até uma semana em avanço) e que possa ser criado com informação conhecida até ao último instante de treino. Alguns atributos podem ser utilizados diretamente nesse **predict** mesmo para se realizarem previsões *multi-step ahead*. Por exemplo: DIA_SEMANA e PRECIPITACAO. Quanto ao uso de atributos *lagged*, tem de se ter algum cuidado e adaptação. Seja por exemplo t o primeiro dia que queremos prever o valor de STELLA, portanto $STELLA_{t.}$ Como o valor de BUD (vendas desse dia) ainda não é conhecido para t, não faz sentido ter esse valor direto como entrada do modelo ML, embora se possa considerar um valor *lagged* de BUD, por exemplo BUD_{t-1}. Assim, a título de exemplo, o modelo ML pode ter estas entradas: TEMP_MAX, $STELLA_{t-1}$ e BUD_{t-1}, pretendendo-se prever $STELLA_t$. A ideia da previsão *multi-step ahead* é realizar até 7 previsões (uma semana) usando só dados conhecidos até t-1. Seja t-1 um domingo. Pode-se usar os valores de TEMP_MAX em qualquer dos dias da semana a prever (conforme enunciado, assume-se que a previsão meteorológica dá este valor). Para ter a previsão da segunda-feira, usa-se então $TEMP\_MAX_t$,$STELLA_{t-1}$,BUD_{t-1} para obter uma previsão de STELLAt (segunda), seja ^$STELLA_t$. Para ter uma previsão de terça, usa-se então

TEMP_MAX$_{t+1}$, ^STELLA$_t$ e ^BUD$_t$. Ou seja, é preciso ter uma previsão de segunda para a STELLA e para o BUD, uma vez que ambas variáveis não são conhecidas nesse período. A *script* lecionada na aula demonstra este processo, sendo que esta *script* deverá ter de ser adaptada para funcionar bem no caso do projeto.

4) Em termos de implementação dos métodos de previsão, devem testar numa primeira fase algo mais simples, por exemplo, treinar com todos dados excepto os da última semana e tentar fazer previsões para essa semana. Verificar se cada modelo de previsão foi bem construído e se faz sentido. Obter um gráfico de valores previstos versus desejados. Calcular uma ou mais medidas de qualidade preditiva. A sazonalidade neste projeto, a existir, será semanal quando muito, portanto com K=7 e nunca com K=52 ou K=12. Não faz sentido usar um "copy & paste" direto das scripts das aulas sem perceber o que se está a fazer. Realço que os modelos do pacote **forecast** só usam a versão sazonal se existir um **frequency** na criação do objecto ts em que **frequency > 1**. Quanto aos modelos do pacote rminer, estes só irão beneficiar de informação sazonal se esta for introduzida na função CasesSeries, por exemplo via os *time lags* 1:7, c(1,2,3,7) ou 1:14 ou outra combinação em que surja o valor de 7 ou seus múltiplos.

   Depois de verificarem e perceberem bem como se cria um modelo de previsão, como se obtêm previsões, qual a qualidade das mesmas e medidas de avaliação de capacidade preditiva, então devem avançar para fase seguinte: estudo mais robusto de comparação de métodos de previsão. Podem usar OU um *growing window* OU um *rolling window,* sendo que tenho aconselhado aplicar um destes métodos de validação para as últimas 20 semanas, prevendo uma semana de cada vez e fazendo um salto de uma semana (portanto 7 dias). Conforme explicado em aula, em cada iteração obtêm-se um conjunto de 7 previsões, logo pode calcular-se uma medida de qualidade da previsão para cada iteração. Em termos globais, pode-se agregar as diferentes medidas via uma mediana ou média dos valores. Como não existe sobreposição entre os dados de teste (são semanas diferentes), podem também criar um gráfico global com todas as previsões das 20 semanas, comparando-as com os valores reais. Por uma questão de coerência, todos métodos de previsão devem ser comparados nas mesmas condições.

   Se quiserem ainda afinar mais elementos da modelação, podem testar mais combinações distintas para o mesmo modelo de previsão. Por exemplo:
   - HoltWinters com (e.g., frequency=7) e sem sazonalidade (sem frequency), modelo univariado.
   - Rede neuronal do tipo "mlpe" com diferentes números de nodos intermédios (univariado).
   - Random Forest com as entradas t-1 a t-7 (univariado), com o uso da opção 1:7 na função CasesSeries.

   Idealmente, esta afinação deveria ser efetuada dentro dos dados de treino, por exemplo separando os dados de treino em dados de fit (primeiros 90%) e validação (últimos 10%), monitorizando os dados de validação para escolher a melhor combinação do modelo. Caso não consigam realizar isto, podem testar todas as combinações na avaliação robusta, como por exemplo o growing window, escolhendo a que dá melhores resultados.

No estudo, é interessante para cada bebida, avaliar: qual o melhor método univariado, multivariado (se testarem estes) e o melhor método (de entre todos). No final, é relevante avaliar a "qualidade" das previsões obtidas. Uma forma de fazer isto é comparar estas previsões com um método "simples", por exemplo o *Weekly Naive*, que corresponde a repetir os últimos 7 valores conhecidos (da última semana) como as próximas 7 previsões. Por exemplo, quando comparado com este método simples *Weekly Naive*, quão melhores são as previsões em %?

5) Convém não esquecer que também existe a componente de **otimização**, que já pode ser iniciada. Tipicamente, o arranque da componente de otimização é um processo algo demorado devido à falta de compreensão do que é para ser feito. Conforme explicado nas aulas, este componente deve ser iniciado pela definição de uma representação de soluções, que corresponde ao plano semanal do enunciado. Depois, deve ser criada uma função de avaliação, tipo: **eval(s)**, que devolve o lucro (conforme as regras do enunciado) para um dado plano **s** que já esteja definido.

   Representação da solução: De acordo com o enunciado, um plano **s** deve ter 6 componentes (recursos de armazém; recursos de distribuição em termos dos veículos v1, v2 e v3; entregas diárias para as lojas em termos de cerveja Stella e Bud). Para cada componente ter os valores planeados para uma semana, portanto 7 dias. Logo, uma solução **s** deve ter um total de 6 x 7 = **42** valores inteiros (0,1,2,...). Como muitos métodos de optimização já implementados em R assumem o uso de soluções vetoriais, a solução s também deverá ser um vetor, de inteiros (42 valores, se for utilizada uma representação de reais ou inteiros) ou binários (42 x *nbits*, se for uma representação binária simples, onde *nbits* representa o número de bits utilizados para representar cada inteiro).

   Função de avaliação: Uma forma simples de testar a função de avaliação é usar os dados do slide 15 do pdf do enunciado. Por exemplo:

```
# variáveis globais
vendas_stella=c(141,154,...)
vendas_bud=c(211,172,...)
s1=c(6,1,0,1,...)
# eval
eval=function(s)
{ profit=0
  s=round(s) # arredondamento local, não altera s1 fora da funcao
...
 return(profit)
}
# teste
print(eval(s1))
# deve mostrar o valor 2619 para os dados do slide 15
```

   Quando esta função estiver bem programada, depois é muito fácil implementar um método de otimização. Podem por exemplo iniciar o teste pelo método de Monte Carlo, embora convenha tratar das soluções inválidas. O objetivo do método de otimização é encontrar os melhores valores de **s**, de modo a maximizar ou minimizar a função **eval**.

Soluções inválidas: Realça-se ainda que uma solução **s**, um planeamento, tem diversos componentes para os 7 dias da semana: recursos armazenamento, recursos de veículos v1, v2 e v3, entregas às lojas das quantidades de stella e bud. Ora, nem todas soluções **s** são válidas, uma vez que existem restrições entre as componentes. Por exemplo, se para um dado dia existirem 0 recursos de armazenamento (arm), não é possível transportar cervejas via veículos, nem sequer entregar quantidades de stella (stella) nas lojas. O tópico de tratar soluções inválidas é especialmente relevante quando muitas das soluções iniciais são geradas de modo aleatório (por exemplo, como acontece no método de Monte Carlo), uma vez que existem muitas soluções inválidas. Para tratar soluções inválidas, existem 2 métodos lecionados nas aulas e que podem ser usados dentro da função de avaliação, eval: **death penalty** ou **repair**. O death penalty consiste num simples:

```
# retornar o pior valor possivel, invalid é função a codificar pelo grupo
if (invalid(s)) profit = - Inf
```

Quando ao repair, consiste em usar um método que altera a solução s, tornando-a válida. Podem por exemplo ajustar a quantidade de armazenamento ou dos veículos ou das entregas. Sejam por exemplo as quandidades arm=0 e stella=33 para um dado dia.
arm=0 e stella=0 # é valido
arm=1 e stella=33 # também é válido.
Dentro da função de avaliação, ter-se-ia algo do género:

```
…
s=round(s)
s=repair(s) # reparação da solução
```

Métodos de otimização possíveis de serem aplicados: Para o objetivo O1, Quase todos lecionados nas aulas. Começar pelos métodos que utilizam uma representação de números reais. É necessário definir os vetores de **lower** e **upper** para este problema. No caso do **upper**, os valores máximos podem ser ajustados aos valores previstos para essa semana. Exemplos de métodos a testar:

- Monte Carlo (procura cega);
- *Hill Climbing*, *Simulated Annealing* (procura mais local);
- Algoritmos genéticos, *Differential Evolution*, *Particle Swarm Optimization* (procura mais global).

Nota: Não convém usar métodos de procura completa, nem o *grid search*, por serem muito exigentes em termos computacionais. No caso do *Simulated Annealing*, convém preencher o argumento **gr=**, de modo a usar uma função *change* de procura na vizinhança que garanta que as soluções estão dentro do **lower** e **upper** (ver a script de demonstração do *Simulated Annealing* para o problema *sphere* lecionada nas aulas: `demo-4-sann.R`).

Numa segunda fase, podem também testar métodos baseados em representações binárias, como o *Tabu Search* ou `rbga.bin`. Para usarem estes métodos, têm de ter uma forma de representar um inteiro (dentro do lower e upper) via um conjunto de bits.

Análise de convergência: este é um elemento opcional do projeto. Se quiserem, poderá ser interessante monitorizar a "evolução" de um método de otimização ao longo das suas

iterações. Chama-se a isto uma análise de convergência. Esta análise permite verificar se o método otimiza bem ao longo das suas iterações ou se estagna rapidamente, não adiantando muito ter mais iterações. Esta análise pode ser realizada por gráficos ou por tabelas.

Que semana otimizar? Numa primeira fase, podem assumir só uma dada semana (por exemplo a última ou penúltima). Se for mais fácil, podem usar de início os valores reais das vendas e não os valores previstos, sendo que mais tarde podem substituir o objeto que contem as vendas reais pelas previsões. O que é importante é implementar diversos tipos de métodos de otimização e certificarem-se que funcionam bem.

Depois numa segunda fase, e já usando valores previstos (e não reais), podem correr uma execução de um mesmo método de otimização para cada uma das últimas 20 semanas (de modo semelhante ao que foi feito em termos de avaliação robusta para a previsão). Em cada execução vão ter um valor otimizado (por exemplo, lucro final). E podem agregar os diversos valores de lucro (os tais 20) utilizando uma mediana ou média. A comparação final entre métodos de otimização pode então ser realizada usando uma tabela e estes valores agregados obtidos.

Otimização multi-objetivo: no caso do objetivo O2, existem formas diferentes de modelar a optimização. Uma forma é assumir um único objetivo, onde atribuem um peso ao lucro e outro peso ao somatório dos recursos. Por exemplo, se o método de otimização só minimiza, seria algo tipo: -w1 x lucro + w2 x recursos. Esta forma implica que cada execução otimiza sempre uma combinação específica (via os pesos) de ambos objetivos. Outra forma, à partida mais interessante, consiste em ter uma função objetivo que retorna um vector com 2 valores, por exemplo **eval2**, relativas ao lucro (ou -lucro) e o número total de recursos de armazém e de distribuição. Usando um método multi-objectivo (e.g., NSGA-II), é então possível numa única execução otimizar de forma simultânea ambos objetivos, retornando no final uma curva de Pareto (conforme explicado nas aulas).

6) Existe ainda um terceiro componente a realizar, que é a construção de um sistema funcional, que assume ter a parte de previsão e otimização a ser executada em R, idealmente via um interface, que pode ser gráfico (via por exemplo pacote **shiny**) ou em modo de consola (via funções **cat**, **print** e **readLines**). Normalmente, a construção deste componente é realizada quando os componentes anteriores (previsão e optimização) estão mais finalizados e tende a demorar 1 a 2 semanas. No caso deste projeto, um modo simples de iteração com o utilizador é este:

   - o utilizador escolhe semana para a qual quer um planeamento.
   - para essa semana, o sistema mostra quais as previsões de vendas, bem como qual ou quais os planeamentos aconselhados.
   - para cada planeamento, o sistema mostra (para cada dia), as vendas potenciais, as vendas reais, os proveitos diários, os custos diários, o valor de bebidas em stock, etc, bem como os proveitos, custos e lucros totais.