Credit card data Analysis

Question 2.1

Here are some examples of classification from my personal / work life,

1. Taking a COVID PCR test before flying out internationally. The predictor variable is PCR Test Result (Positive/Negative) and the response variable is if I can fly out or not (True/False)
2. On a long road trip, I usually look at different route options before deciding with routes to take. The predictor variables in this case would be:
    a. How long is the route (miles - number)
    b. How many accidents / incidents are reported along the way (number)
    c. How scenic is the route (Boolean – Categorical)

   And the response variable is a route (set of roads) to travel by.
3. At work, applications usually emit log messages / events in SYSLOG format and we must decide whether to alert the operational teams based on the
    a. severity of the event
    b. number of occurrences of the event in a specific time window
    c. if these events have self-healed in the past.

       The response variables are a) to alert or not to alert b) whom to alert

   A special case of this is when we get some cumulative stats from the application.
   E.g.: number of events processed or processing errors by the application since the start.
   This is an example of time series data.
4. Planning lunch is always an unnecessarily complex classification problem. Predictor variables include:
    a. Kind of cuisine preferences? there is a sub-classification: what cuisines have most people had the previous few days.
    b. How far is the restaurant? There is often a sub-classification here: do we feel like walking vs driving.
    c. Covid satiety protocols followed at the restaurant
    d. Word of mouth references / if we really want to waste time Yelp/Google reviews.
    e. Cost
    f. Cleanliness
    g. Friendliness
    h. Quality of food
    i. Time to serve

    And the response variable, of course is the name of the restaurant and usually a measure of how long we have wasted in deciding this.

5. Almost every day I must decide when and how much to water the plants. Predictor variables are:
    a. If it rained the previous day or if there is rain prediction of that day

b. How many days I have spent too much time with this, or other equally unnecessary classification problems - instead of watering the plants
c. How dry the plants look (which usually is unhealthily dry owning to previous predictor variable)
d. Any water uses restriction (more popular in Texas summer)

Response variables are decision to water them and the amount to water.

Question 2.2.1

Problem Statement

The purpose of this assignment is to analyze the input data and find good classifier for the data – specially using ksvm function from kernlab and further examine how well kknn function classifies the dataset.

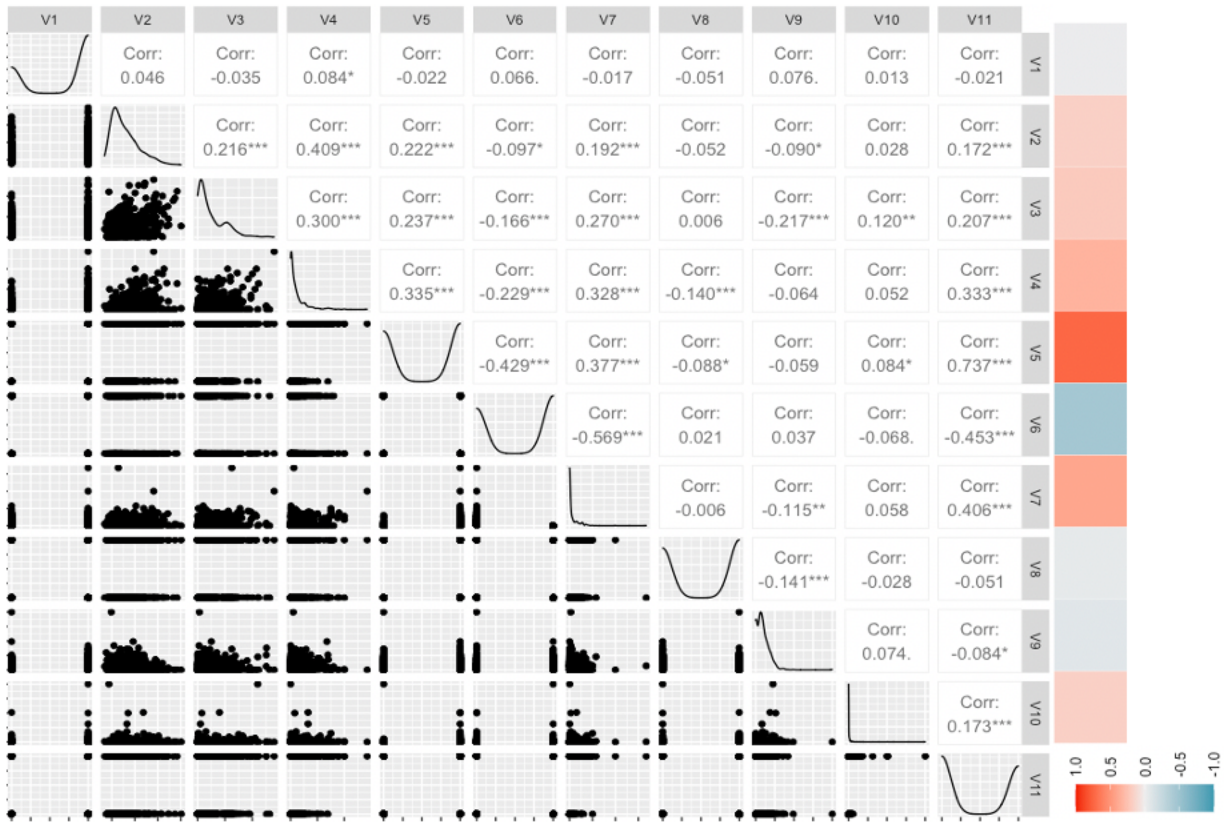
Input data Analysis:

We are presented with an input data on credit card application information that includes 10 attributes and one response variable. The response variable is Boolean value of 1 and 0 indicating if the credit card application was approved or declined.

Using R functions such as head(), summary(), I infer that some attributes are quantitative while others are categorical, binary to be specific; some are integers (discrete values) while others are continuous. Also, the range of these attributes is variable – from very small to very large (100000s). For this reason, **we will be required to scale the data so attributes with uneven range don't have uneven influence in our model**.

```
> summary(dat)
       V1                V2                V3                V4        
V5                V6                V7       
 Min.   :0.0000   Min.   :13.75   Min.   : 0.000   Min.   : 0.000   
Min.   :0.0000   Min.   :0.0000   Min.   : 0.000  
 1st Qu.:0.0000   1st Qu.:22.58   1st Qu.: 1.040   1st Qu.: 0.165   
1st Qu.:0.0000   1st Qu.:0.0000   1st Qu.: 0.000  
 Median :1.0000   Median :28.46   Median : 2.855   Median : 1.000   
Median :1.0000   Median :1.0000   Median : 0.000  
 Mean   :0.6896   Mean   :31.58   Mean   : 4.831   Mean   : 2.242   
Mean   :0.5352   Mean   :0.5612   Mean   : 2.498  
 3rd Qu.:1.0000   3rd Qu.:38.25   3rd Qu.: 7.438   3rd Qu.: 2.615   
3rd Qu.:1.0000   3rd Qu.:1.0000   3rd Qu.: 3.000  
 Max.   :1.0000   Max.   :80.25   Max.   :28.000   Max.   :28.500   
Max.   :1.0000   Max.   :1.0000   Max.   :67.000  
       V8                V9                V10                V11        
 Min.   :0.0000   Min.   :  0.00   Min.   :     0   Min.   :0.0000  
 1st Qu.:0.0000   1st Qu.: 70.75   1st Qu.:     0   1st Qu.:0.0000  
 Median :1.0000   Median : 160.00   Median :     5   Median :0.0000  
 Mean   :0.5382   Mean   : 180.08   Mean   :  1013   Mean   :0.4526  
 3rd Qu.:1.0000   3rd Qu.: 271.00   3rd Qu.:   399   3rd Qu.:1.0000  
 Max.    :1.0000   Max.    :2000.00   Max.    :100000   Max.    :1.0000
```

Not all attributes have same level of correlation to the response variables – as illustrated in the below ggpairs() and ggcorr(). So**, it is possible to drop some variables (eg: V1 and V8) in our model, but for the purposes of this assignment, we will include all attributes**.



Support Vector Method (SVM)

A support vector method (SVM) is a generalization of a maximal margin classifier. (For brevity, we will not explain terms such as margin, hard/soft margin, margin classifier, maximum message classifier, hyperplane, support vector classifier, etc.). Class notes, ISLR and StatQuest, among other resources has pretty good explanation of all these required terms.

Informally speaking, SVM looks at data points as p-dimensional vectors, defines a p-1 dimensional hyperplane (a line in the case of two-dimensional data space) that divides the data points into two categories. With this, we would be able to predict which side of the hyperplane a new data point fits in. The challenge of course is to pick the optimal hyperplane that will minimize the classification errors. We do this by calculating the accuracy of the model - by comparing the predicted value vs actual value for known data points.

Based on the sample code provided in the assignment, I wrote a function that would calculate the accuracy for a given kernel, type and cost and store it a data frame. Then I could use standard R filters to extract the data point that maximizes the accuracy.

```
#-------------------------------------------------
# Parameterized model
#
my_model_1 <- function(cost_, kernel_, type_) {
  cat ('\nCalculating for Kernel:', kernel_, 'Type:', type_, 'Cost :', cost_)
  t0 <- proc.time() # track how long it takes
  model <- ksvm(as.matrix(data[,1:10]),
                as.factor(data[,11]),
                type=type_,
                kernel=kernel_,
                C=cost_,
                scaled=TRUE)
  a <- colSums(model@xmatrix[[1]] * model@coef[[1]])
  a0 <- -model@b
  pred <- predict(model,data[,1:10])
  accuracy <- sum(pred == data[,11]) / nrow(data)
  cat ('... model:'); print(model)
  cat ('... A :'); print(a)
  #cat ('... Prediction: '); print(pred)
  cat ('... Accuracy :', accuracy, '\n')
  cat ('    elapsed time:', proc.time()-t0)
  data.frame(kernel_,type_,cost_,accuracy)
}
```

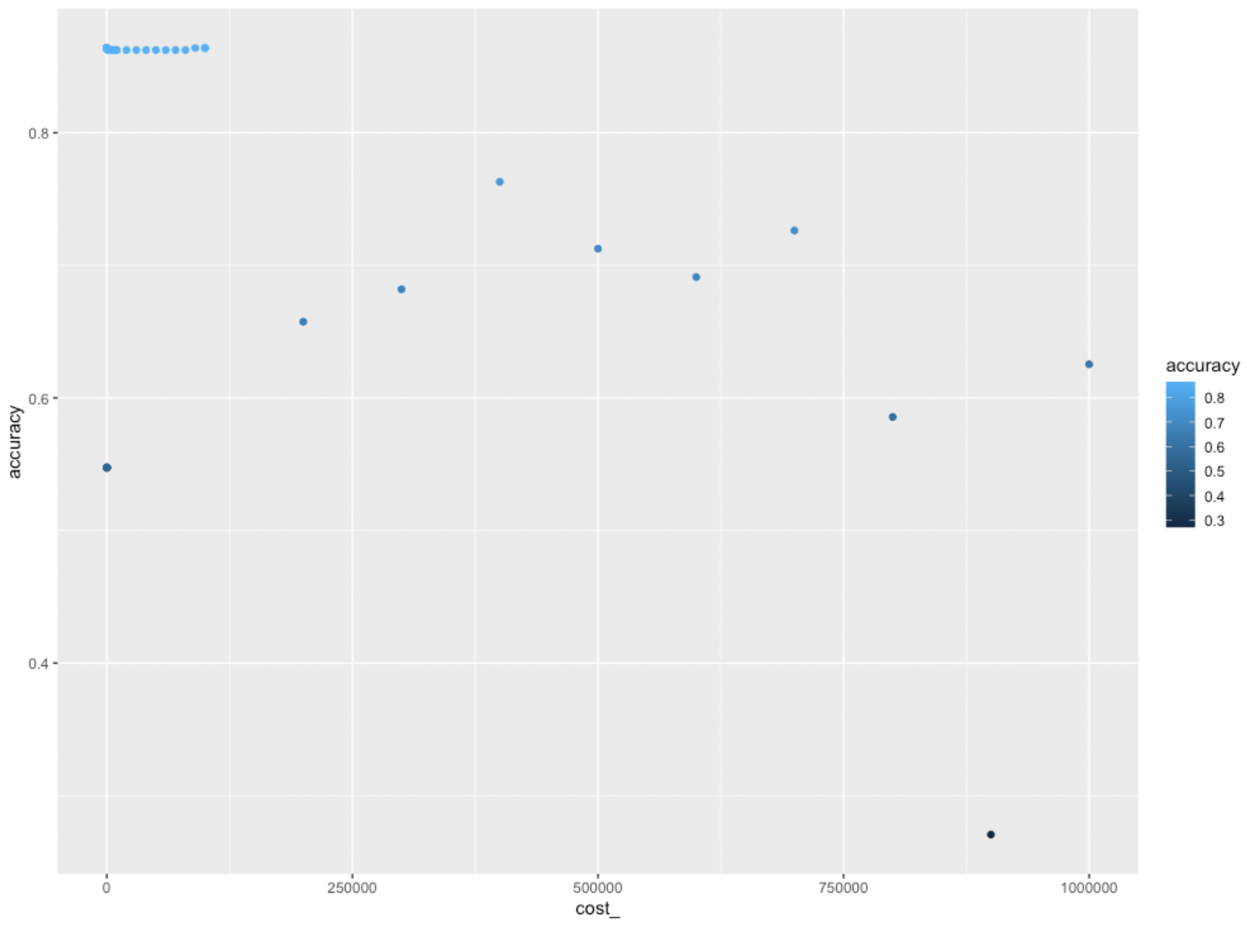
This make is easier to call this with different values such as different cost of constraint violation, kernel function or classification type. The results can be accumulated to standard data frame with different values like below:

```
results <- data.frame()
for (i in seq(1, 100, by=10)) {
  results <- rbind(results,
                   my_model_1(i, "vanilladot", "C-svc"))
}
for (i in seq(100000, 1000000, by=100000)) {
  results <- rbind(results,
                   my_model_1(i, "vanilladot", "C-svc"))
}

for (i in seq(.0000001, .000001, by=.0000001)) {
  results <- rbind(results,
                   my_model_1(i, "vanilladot", "C-svc"))
}
```

SVM Analysis

C (the Cost of constraints violation) allows us to adjust the bias-variance trade off. Higher values of C allow more variance and less bias (more flexible models).

Running with a wide range of C value resulted in a range of accuracy and the **best results are observed around 1 to 100 range**. Too small and too large values resulted in the model doing worse.



```
> max_results<-results[results$accuracy==max(results$accuracy),]
> head(max_results)
    kernel_ type_ cost_  accuracy
1 vanilladot C-svc     1 0.8639144
2 vanilladot C-svc    11 0.8639144
3 vanilladot C-svc    21 0.8639144
4 vanilladot C-svc    31 0.8639144
5 vanilladot C-svc    41 0.8639144
6 vanilladot C-svc    51 0.8639144
```

Since very low C values allow high bias with little attention to the training data, the low accuracy can be explained. Calculations with high C value allow high variance is something that's hard to explain. (TBD)

Non-Linear Kernels

In two class settings where the boundary between two classes is liner, support vector classifier is a natural choice. However, using Kernel trick extension, we can use SVMs for non-linear classification as well.
I tried couple of them using the same parameterized model function and I found that the "**splinedot**" kernel gets much better accuracy.

```
> my_model_1(100, "splinedot", "C-svc")

Calculating for Kernel: splinedot Type: C-svc Cost : 100 Setting default
kernel parameters
... model:Support Vector Machine object of class "ksvm"

SV type: C-svc  (classification)
 parameter : cost C = 100

Spline kernel function.

Number of Support Vectors : 172

Objective Function Value : -121162.2
Training error : 0.021407
... A :          V1           V2           V3           V4           V5
V6           V7
   6.035428  439.986669 -103.580060  -23.007520  -16.089700   60.721453  -
17.646918
         V8           V9          V10
 107.815329 -233.089446    7.757040
... Accuracy : 0.9785933
    elapsed time: 0.631 0.024 0.655 0 0    kernel_ type_ cost_  accuracy
1 splinedot C-svc   100 0.9785933
```

K Nearest Neighbor (KNN)

KNN Algorithm assigns class membership to data point by plurality vote of its neighbors. The neighbors are computed by distance from the datapoint (usually Euclidean but can be others). Selecting a small enough number of neighbors (K value) thus reduces the computational complexity, while large values of N reduce the effect of noise in the data.

Based on the code discussed in the office hours, I wrote a parameterized function to calculate the accuracy of the KNN model. As required by the assignment, for each of K value, the algorithm processes each datapoint, but remove that data-point from the training data and having just that data point for verification. Since KNN returns the probability of the result (V11) being 1, the algorithm assumes any probability > 0.5 is 1 and 0 otherwise.

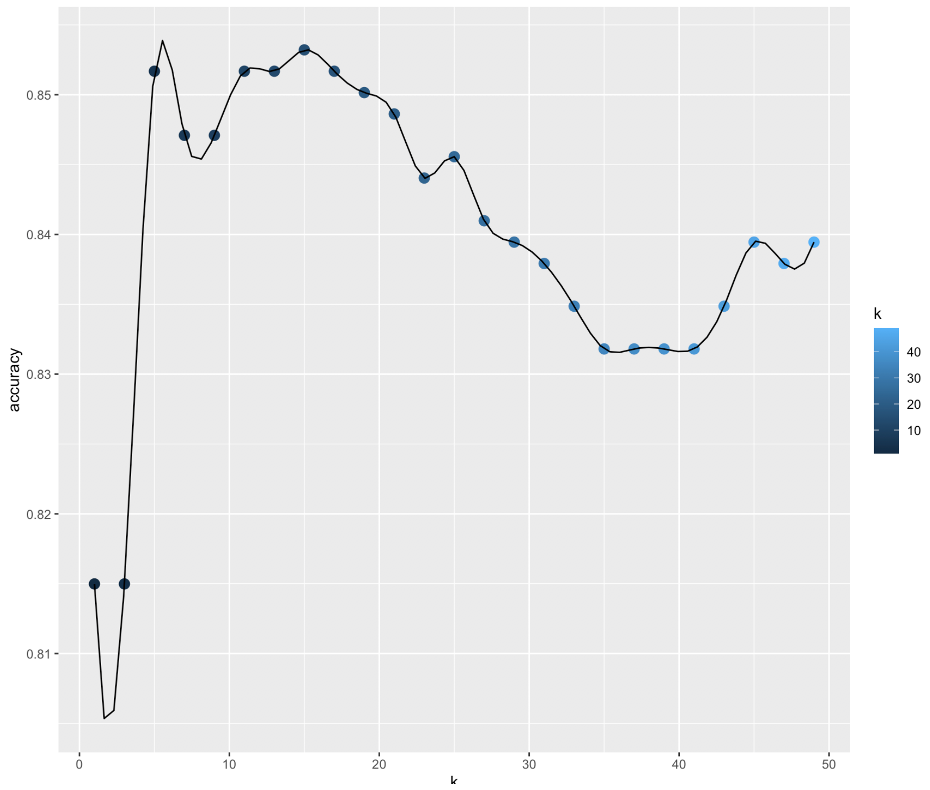
```
knn_model_1 <- function(i, neighbours_, distance_=2, kernel_='optimal') {
  #cat ('\nCalculating for i:',i, 'neighbours: ', neighbours_, 'distance:',
distance_, 'kernel:', kernel_)
  learn = data[-i,]
  test = data[i,]
  knn <- kknn(V11 ~ . , learn, test,
              k=neighbours_, distance = distance_, kernel = kernel_,
              scale = TRUE)
  fit <- fitted.values(knn)
  #cat ('\n... fit:'); print(fit)
  actual <- data[i,11]
  predicted <- if (fit > 0.5) 1 else 0
  data.frame(kernel_,distance_,neighbours_,i, fit, predicted, actual)

}
```

This function is called iteratively for all datapoints for some K values. For each K value, a mean of prediction accuracy over all datapoints is considered as below:

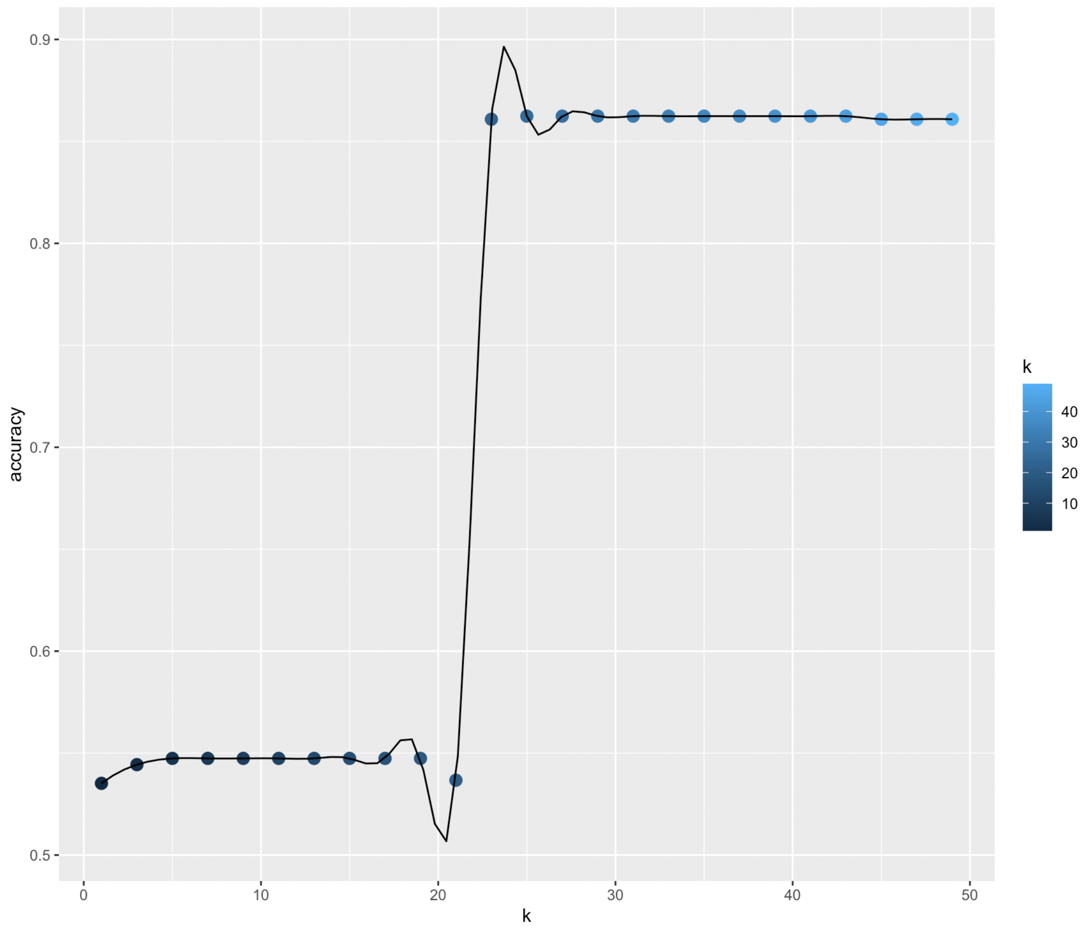
```
results_knn <- data.frame()
for (k in seq(1, 50, by=2)) {
  cat ('\n--- Calculating for k:', k)
  df <- data.frame()
  for (i in seq(1, nrow(data))) {
    df <- rbind(df, knn_model_1(i, k))
  }
  accuracy <- sum(df$predicted == df$actual)/nrow(df)
  results_knn <- rbind(results_knn, data.frame(k, accuracy))
}
```

KNN Analysis

As explained above, the algorithm performs poorly on very low K values (less than 5). Increasing the K value gives a better result, but only up to point (around 15). This may be due the presence of large number of attributes in the data, some with low correlation to the response variable.



For eg, if we plot only for strongly correlating attributes (V5:V7), we get a monotonically increasing accuracy as K goes up.



Other kernels

Checking the model for same optimal k (15) with different kernels, I see a slight difference in performance, with triangle kernel giving a small improvement.

```
results_knn <- data.frame()
k = 15
for (kernel in c("optimal", "triangular", "rectangular", "epanechnikov",
"biweight", "triweight", "cos", "inv", "gaussian", "rank")) {
  cat ("\nCalculating for kernel:", kernel)
  for (i in seq(1, nrow(data))) {
    df <- rbind(df, knn_model_1(i, k, kernel))
  }
  accuracy <- sum(df$predicted == df$actual)/nrow(df)
  results_knn <- rbind(results_knn, data.frame(kernel, k, accuracy))
}
```

```
> results_knn[order(-results_knn$accuracy),]
         kernel  k  accuracy
2    triangular 15 0.8480122
1       optimal 15 0.8478593
7           cos 15 0.8446483
5      biweight 15 0.8446101
3   rectangular 15 0.8442915
4  epanechnikov 15 0.8442551
6     triweight 15 0.8442066
9      gaussian 15 0.8436544
8           inv 15 0.8436197
10         rank 15 0.8422724
```