notebook0277ef60f2

January 17, 2023

In this notebook we want to show how data analysis can support an internal audit. Therefore, this notebook does not use the data as indicated in the description of the data. Furthermore, the text is in German since I want to link it to a small lecture I'll give to a German audience.

Comments are welcome!

0.1 target

The purpose of the notebook is to show how data analysis can support revision testing. Experience has shown that for a medium-sized test (30 - 50 PT), an initial data analysis, as shown below, should not take more than **around 3 - 5 days .**
This approach is applicable to a **wide range of cases** and requires only * access to data associated with the exam content, * standard techniques in a language suitable for data analysis (e.g. R or Python), and * a basic understanding of the data/common sense.

We perform data analysis for Incident Data available on Kaggle. The methods can also be used analogously for credit data, payment transaction data, personnel data and many other types of data.

In this notebook we want to show how data analysis can support an internal audit. Therefore, this notebook does not use the data as indicated in the description of the data. Furthermore, the text is in German since I want to link it to a small lecture I'll give to a German audience.

Comments are welcome!

In this notebook we want to show how data analysis can support an internal audit. Therefore, this notebook does not use the data as indicated in the description of the data. Furthermore, the text is in German since I want to link it to a small lecture I'll give to a German audience.

Comments are welcome!

In this notebook we want to show how data analysis can support an internal audit. Therefore, this notebook does not use the data as indicated in the description of the data. Furthermore, the text is in German since I want to link it to a small lecture I'll give to a German audience.

Comments are welcome!

In this notebook we want to show how data analysis can support an internal audit. Therefore, this notebook does not use the data as indicated in the description of the data. Furthermore, the text is in German since I want to link it to a small lecture I'll give to a German audience.

Comments are welcome!

0.2 0. Preliminary remark

Central to a good audit are: * a clear audit objective that derives from the risk of the audit field * audit questions derived from this and * a good understanding of the underlying processes from which the data originate.

None of this is the case for the present data publicly **available** on Kaggle . We therefore make certain **assumptions in** order to still be able to simulate a stringent analysis. * The aim of the audit is to check the effectiveness and regularity of the incident process. The audit questions are: * To what extent is the process suitable for closing incidents in a timely * manner, depending on their criticality? * To what extent is the processing quality satisfactory? * With regard to the process, we assume that it is a typical incident process (according to the description of the data, it is “data gathered from the audit system of an instance of the ServiceNowTM platform used by an IT company”)

The aim of the data analysis is, on the one hand, to generate comprehensive evaluations for the above questions. Typically, the incident management has also set itself goals that can be checked against - this cannot be considered in the example. On the other hand, outliers should be identified. These would then be examined more closely in practice in order to identify potential weaknesses in the process design. In the example given here, of course, only the definition of the risk-oriented random sample can be presented.

0.3 1. Preparations:

Loading the used packages and data. We are fortunate to have a complete data set here and not have to connect different data sources with each other. In real life, these steps often involve considerable effort.

```
[ ]: # Loading packages
library(tidyverse) # metapackage of all tidyverse packages library(plotly) # For
interactive graphics
```

```
[ ]: # Loading data
incident_event_log <- read.csv("../input/incident-response-log/
   ÿincident_event_log.csv") description
<- readLines("../input/incident-response-log/Incident_response.txt")
```

After looking at the file, we can ignore the warning. In addition to the actual data, there is a short description that we output:

```
[ ]: print(description[-2:-1]) # lines 1 and 2 without information about variables
```

Next, a quick look at the data:

```
[ ]: head(incident_event_log)
```

Apparently there are several lines per incident. The final entries then seem to be in the last line, with the lines before that already containing some entries that you might not have expected here (e.g. the “resolved_by” column when opening a ticket). Here is another graphic that shows the entries per ticket number:

```
[ ]: incident_event_log$number %>% table() %>% hist(main = "Number of entries perÿ
    ÿIncident")
```

We want to focus on the last line per incident. This should be the line where the status is Closed. We would have expected this to be equivalent to the active column being false. This is true in principle, but not for one line.

```
[ ]: incident_event_log %>% filter(incident_state == "Closed" & active == "true")ÿ
    ÿ%>% nrow()
  incident_event_log %>% filter(incident_state != "Closed" & active == "false") incident_event_log %>%
  filter(number == "INC0018594")
```

Observation 1: "INC0018594" (as the only incident) has the status "incidents_state" unequal to "Closed" and "active" equal to false.

In the following we reduce the data set to those rows in which the incident_state has the status "Closed".

```
[ ]: incident_event_log_save <- incident_event_log incident_event_log
  <- incident_event_log %>% filter(incident_state == "Closed")
```

We still check whether we now actually have a line for each incident:

```
[ ]: doppelter_name <-ÿ
    ÿincident_event_log$number[duplicated(incident_event_log$number)]
  print(paste("Number of incidents that occur more than once: ",ÿ
    ÿlength(duplicate_name)))
  print(paste("Total number of incidents in closed status: ",ÿ
    ÿlength(unique(incident_event_log$number))))
```

Given the scale and time constraints, we resist the urge to look for the cause of the multiple incidents. Rather, we reserve them for later for a random sample.

0.4 2. Univariate Outlier Analysis (Machining Quality)

For example, the following key figures could provide information on the **processing** quality: * reassign ment_count: number of times the incident has changed the group or the support analysts; open_count: number * re of times the incident resolution was rejected by the caller * Processing time

Let's take a look at the corresponding histograms. First for reassignment_count and reopen_count:

```
[ ]: incident_event_log %>% ggplot(aes(reassignment_count)) + geom_histogram() incident_event_log %>%
  ggplot(aes(reopen_count)) + geom_histogram()
```

The processing time must first be formatted:

```
[ ]: incident_event_log$h <- as.POSIXct(incident_event_log$resolved_at, format="%d/ ÿ%m/%Y %H:%M")
```

```
incident_event_log$h2 <- as.POSIXct(incident_event_log$opened_at, format="%d/%m/
   ÿ%Y %H:%M")
incident_event_log$settlement time <- difftime(incident_event_log$h,ÿ
   ÿincident_event_log$h2, unit = "days")
incident_event_log %>% ggplot(aes(settlement time)) + geom_histogram()
```

Obviously the graphics aren't pretty and a warning pops up. **My urgent recommendation, however, is not to beautify the graphics in a time-consuming manner.** Instead, the results are easy to read: There are individual tickets with high assignment scores (> 10) and also with reopen_count > 2. We create a sample in which we save these tickets. We set the limits based on the graphics based on expert estimates:

```
[ ]: sample <- incident_event_log %>% filter(reassignment_count > 10) print(paste("number of rows: ",
     nrow(sample))) sample$reason <- "Reassignment high"
```

```
[ ]: h <- incident_event_log %>% filter(reopen_count > 2) print(paste("number of
     rows: ",nrow(h))) h$reason <- "reopen high" sample <- rbind(sample, h)
```

We can also set the limits for a sample according to expert estimates for the processing time. Alternatively, you can also take the 0.1% of cases that take the longest:

```
[ ]: high_resolution_time <- quantile(incident_event_log$resolution_time, 0.999, na.
        ÿrm = T)
     print(paste("High handling time:", high_handling_time)) h <- incident_event_log
     %>% filter(handling time > high_handling_time) print(paste("Number of rows: ",nrow(h))) h$reason <-
     "Handling time high " sample <- rbind(sample, h)
```

At this point we also consider whether/why we have NA entries in the completion times:

```
[ ]: print(paste("Number of NA entries at completion times:",incident_event_log %>%ÿ
        ÿfilter(is.na(handling time)) %>% nrow())) print("Entries in the
     resolved_at column in this case:") incident_event_log %>% filter(is.na(handling
     time)) %>% select( resolved_at)ÿ
        ÿ%>% table()
     print("Entries in the resolved_by column in this case:") incident_event_log %>%
     filter(is.na(resolution time)) %>% select(resolved_by)ÿ
        ÿ%>% table()
```

So it's just a "?" entered in the "resolved_at" column, which is why no processing time can of course be calculated. In most cases, however, someone is entered who has dealt with the incident.

Observation 2: There is missing data on the date when an incident was resolved.

We select a sample, also considering cases where the person doing the work is unknown.

```
[ ]: h <- incident_event_log %>% filter(is.na(time to be resolved)) %>%ÿ
       ÿfilter(resolved_by == "?") h <- h[sample(1:nrow(h), 10), ] # take sample of 10
     elements h$reason <- "resolved_at and resolved_by missing" sample <-
     rbind(sample, h)


     h <- incident_event_log %>% filter(is.na(time to be resolved)) %>%ÿ
       ÿfilter(resolved_by != "?") h <- h[sample(1:nrow(h), 10),] # take sample of 10
     elements h$reason <- "resolved_at missing" sample <- rbind(sample, h)
```

In addition, as noted in Chapter 1, we wanted to include multiple incidents in the take a sample:

```
[ ]: h <- incident_event_log %>% filter(number %in% duplicate_name) h <-
     h[sample(1:nrow(h), 10),] # take sample of 10 elements h$reason <- "Incident
     multiple " sample <- rbind(sample, h)
```

0.5 3. Criticality

According to Section 0, an audit question was: *To what extent is the process suitable for closing incidents in a timely manner, depending on their criticality?*

Accordingly, it is first necessary to evaluate which incidents can be regarded as critical.

Auf eine hohe Kritikalität der Incidents könnten z.B. folgende Variablen hinweisen: * impact: description of the impact caused by the incident (values: 1â€“High; 2â€“Medium; 3â€“Low); urgency: description * of the urgency informed by the user for the incident resolution (values: 1â€“High; 2â€“Medium; 3â€“Low); * priority: calculated by the system based on 'impact' and 'urgency'; * made_sla: boolean attribute that shows whether the incident exceeded the target SLA;

The last variable is difficult to assess without further knowledge, and the completion time may also play a role here.

It is also interesting to see for which incidents a confirmation of priority was made.

In the following we show the distribution of the incidents on these variables using histograms.

```
[ ]: incident_event_log %>% ggplot(aes(impact, fill = u_priority_confirmation)) +ÿ
       ÿgeom_histogram(stat="count")
     incident_event_log %>% ggplot(aes(urgency, fill = u_priority_confirmation)) +ÿ
       ÿgeom_histogram(stat="count")
```

```
incident_event_log %>% ggplot(aes(priority, fill = u_priority_confirmation))+ÿ
  ÿgeom_histogram(stat="count")
incident_event_log %>% ggplot(aes(made_sla, fill=priority)) +ÿ
  ÿgeom_histogram(stat="count")
```

Here, too, we resist the temptation to make the graphs prettier, for example by arranging them more nicely (grid.arrange, plot_grid). Instead, let's take a closer look at the graphs:

Observation 3: The vast majority are rated medium/moderate. It should be verified that the relevant classification processes are appropriate.

As expected, the priority is confirmed, especially in the case of high-priority incidents. In the sample, we include a few cases that are highly prioritized, but where there was still no review.

```
[ ]: h <- incident_event_log %>% filter(priority == "1 - Critical") %>%ÿ ÿfilter(u_priority_confirmation ==
       "false")
     h
     h$reason <- "Critical priority not checked" sample <- rbind(sample,
      h)
```

As expected, made_sla is false for critical incidents.

Observation 4: There are low-priority incidents that still result in non-compliance with the SLA.

Let's take a sample of this:

```
[ ]: h <- incident_event_log %>% filter(priority == "4 - Low") %>% filter(made_slaÿ
       ÿ== "false")
     nrow(h)
```

121 cases are too many for the sample, so we draw a random selection here.

```
[ ]: h <- h[sample(1:nrow(h), 20),] # take sample of 20 elements
     h$reason <- "Low Priority Incident Failure to SLA" sample <- rbind(sample, h)
```

0.6 4. Criticality and processing time

According to Section 0, the first audit question was: To what extent is the process suitable for closing incidents in a timely manner, depending on their criticality?

Assuming that priority best represents criticality, the following boxplot provides a good approximation of the question:

```
[ ]: incident_event_log %>% ggplot(aes(priority, completion time)) + geom_boxplot()
```

Observation 5: On average, more critical incidents are not closed earlier than less critical ones. The process must be checked here for the control effect of the priority.

Let's look at this again in numbers:

```
[ ]: incident_event_log %>% group_by(priority) %>% summarize(average =ÿ
       ÿmean(completion time, na.rm=T), median = median(completion time, na.rm=T))
```

Here, too, we sample the critical ones with the longest processing time.

```
[ ]: h <- incident_event_log %>% filter(priority == "1 - Critical") %>%ÿ ÿfilter(settlement time > 100)

     h$reason <- "High resolution critical incident" sample <- rbind(sample, h)
```

0.7 5. Change of priority in the course

In the previous evaluations, we implicitly assumed that the priority over the incident was unchanged. However, it is quite possible that an incident changes its criticality, for example due to long processing times. We investigate this below for critical incidents.

```
[ ]: # All critical incidents: critical_incidents <-
     incident_event_log %>% filter(priority == "1 - Critical")

     # All critical incidents at the end with all lines before
     critical_incidents_alle_entries <- incident_event_log_save %>% filter(numberÿ
       ÿ%in% critical_incidents$number)

     # Incidents that were critical in the end but not critical_incidents_between_other_priority <-ÿ

       ÿcritical_incidents_alle_entries %>% filter(priority != "1 - Critical") print(paste("Number of incidents that were
     critical at the end, but in betweenÿ
       ÿnot: ",ÿ
       ÿlength(unique(critical_incidents_between_other_priority$number))))

     # Incident numbers that were always critical always_critical <-
     setdiff(critical_incidents$number,ÿ
       ÿcritical_incidents_between_other_priority$number)

     # Incidents (all columns) that were always critical always_critical_incidents <-
     incident_event_log %>% filter(number %in%ÿ ÿalways_critical)


     # Mean and median of these incidents
     print(paste("Average resolution time of incidents that were alwaysÿ ÿcritical: ",
       mean(always_critical_incidents$resolution time)))
     print(paste("Median of the completion time of the incidents that were always critical:
       ÿ ", median(always_critical_incidents$handling time)))
```

You can see that the results are better when the incidents were critical from the start.

However, the average time is still higher than High and no lower than Moderate, and the median is higher than Moderate and Low.

0.8 6. Multivariate Analyse

We have already made individual analyzes above that included several variables (e.g. completion time as a function of criticality). In this chapter, however, we would like to show how to deal with a relatively large number of variables.

First of all, we need to convert all relevant variables into numeric variables. the

For the sake of simplicity, we also convert ordinal variables as if they were metric (eg the distance between "Critical" and "High" is the same as between "High" and "Moderate").

Additionally, we remove the incidents whose resolution time is NA.

Output is a summary of the different variables.

[ ]:
```
df <- incident_event_log df <- df %>%
select(priority, reassignment_count, reopen_count,ÿ
   ÿcompletion time, made_sla, u_priority_confirmation)
df$priority[df$priority == "1 - Critical"] <- 3 df$priority[df$priority ==
"2 - High"] <- 2 df$priority[df$priority == "3 - Moderate"] <- 1
df$priority[df$priority == "4 - Low"] <- 0 df$priority <-
as.numeric(df$priority)


df$made_sla[df$made_sla == "false"] <- 0
df$made_sla[df$made_sla == "true"] <- 1 df$made_sla
<- as.numeric(df$made_sla)

df$u_priority_confirmation[df$u_priority_confirmation == "false"] <- 0
df$u_priority_confirmation[df$u_priority_confirmation == "true"] <- 1 df$u_priority_confirmation <-
as.numeric(df$u_priority_confirmation)

df$completion time <- as.numeric(df$completion time)

df <- df %>% filter(!is.na(completion time))

summary(df)
```

0.8.1 6.1 Principal Component-Analyse

We now perform a principal component analysis. This is one of the most common methods of displaying high-dimensional data in two dimensions with as little loss of information as possible.
Of course, the method also has disadvantages/limitations (in particular that only linear transformations are possible and the optimal choice of axes or the loss of information can only be proven under strict conditions), which we will not go into here. However, it makes sense to check how much "percent of information" can be seen with two axes.

[ ]:
```
df.pca <- prcomp(df, center = TRUE, scale. = TRUE)
summary(df.pca)
```

So with two axes you can see around 47% of the information, which is not a particularly high value with 6 dimensions (with random axes it would be on average 33%).

We now plot the first two axes.

[ ]:
```
# Convert df.pca to data.frame df.pca_data_frame
<- df.pca[["x"]] %>% data.frame()

# Log incident names names <-
incident_event_log %>% filter(!is.na(delivery time)) names <- names$number



#plot
plot_ly(df.pca_data_frame, x=~PC1, y=~PC2, text=namen, type="scatter")
```

We regard the individual points at the edge as outliers. As a pragmatic approach, we hover over these points and write them into a vector. We draw the graphic again (this time without an interactive element) to check it or to make it easier to understand, and color-code the elements of the vector.

[ ]:
```
outlier <- c("INC0012499", "INC0007521", "INC0005927", "INC0015902",ÿ
  ÿ"INC0002129","INC0007521", "INC0002780"                , "INC0011665", "INC0001929",ÿ
  ÿ"INC0007593","INC0002483","INC0003419", "INC0007229", "INC0019396",ÿ ÿ"INC0011206",
  "INC0019596", "1C902040", "1C900040", "1C900040" , "INC0020718",ÿ ÿ"INC0007349", "INC0012815",
  "INC0019131") df.pca_data_frame$names <- names df.pca_data_frame$outliers <- ifelse(names %in%
outliers, 1, 0) df.pca_data_frame % >% ggplot(aes(PC1, PC2, colour=as.factor(outlier))) +ÿ



  ÿgeom_point()
```

The essential question is: have we discovered new outliers through this analysis that we had not yet discovered in the univariate analysis?

[ ]:
```
print(paste("Number of multivariate outliers: ", length(unique(outlier))))
h <- setdiff(outliers, sample$number) print(paste("Number of
multivariate outliers not identified in the univariateÿ ÿoutlier analysis: ", length(h)))
```

So 4 out of 21 outliers are “new”. So the multivariate analysis provides new information, but not a lot.

[ ]:
```
h <- incident_event_log %>% filter(number %in% outliers)
h$reason <- "Outlier acc. PCA" sample <-
rbind(sample, h)
```

Of course, such a purely graphical analysis is vulnerable (although it is only a sample investigation goes). Of course, there are also algorithms that perform similar analyses, for example based on k-nearest neighborhoods.

0.8.2 6.2 Autoencoder

We addressed a few weaknesses of principal component analysis above. We therefore use a second, non-linear method for outlier analysis here. We train a small neural network that should compress the data as well as possible. Those records where the compression is far from the original are outliers.

First, we prepare the data and packages.

```
[ ]: # load library
library(autoencoder)
library(scales)

# Scale variables
df_scaled <- sapply(df, rescale)
summary(df_scaled)
```

Now let's fire up the model. According to the package, the model is especially for “sparse autoencoders” (to avoid overfitting with many variables). This is not necessary here, which is why we set the parameter beta (weight of sparsity penalty term) to 0. Of course you can also use other packages here.

```
[ ]: # Berechne Modell
set.seed(123) model <-
autoencode(df_scaled, beta = 0, rho = 0.01, epsilon = 0.001, N.hiddenÿ
  ÿ= 3, lambda = 0.0002, rescale.flag = FALSE)
# Calculate prediction of model result <-
predict(model, df_scaled, hidden.output = FALSE)
```

We now calculate the deviations between the model prediction and actual values and plot them in a histogram. We mark those incidents that are already in the sample:

```
[ ]: # Calculate the deviation between the forecast and the actual values (perÿ ÿvariable and per incident)
  deviation <- as.matrix(df_scaled) - result$X.output # Calculate the square of this deviation_2 <-
deviation^2 # Calculate the sum of all square deviations per incident deviation_total <- rowSums
(deviation_2)



# Take these with the names in a dataframe deviation_df <-
data.frame(deviation_total, names) colnames(deviation_df) <- c("deviation",
"IncidentName") deviation_df$Ist_in_Sample <- ifelse(deviation_df$IncidentName
%in%ÿ ÿsample$number, "yes", "no")
```

```
# Plot histogram g <-
deviation_df %>% ggplot(aes(deviation, fill = actual_in_sample)) +ÿ ÿgeom_histogram()
  ggplotly(g)
```

Compared to all incidents, there are only very few in the sample, which is why they are not visible in the diagram. However, if you select a range of approx. count 0 - 50, deviation 0.5 - 1.1, you can clearly see that incidents with the highest deviation are already included in the sample. From a deviation of about 0.5, however, there are also incidents that are not yet in the sample. We therefore set the limit for integration to 0.4.

```
[ ]: hh <- deviation_df %>% filter(deviation > 0.4)
print(paste("Number of incidents with deviation > 0.4:ÿ
  ÿ",length(unique(hh$IncidentName))))
sample_in_deviation <- sample %>% filter(number %in% hh$IncidentName) print(paste("Of which
already included in the sample: ",ÿ ÿlength(unique (sample_in_deviation$number))))
```

We add the points of the sample:

```
[ ]: h <- incident_event_log %>% filter(number %in% hh$IncidentName) h$reason <-
"Outlier acc. Autoencoder" sample <- rbind(sample, h)
```

0.9 completion

The brief analysis gave us important information for our audit: * A basic understanding of the quantity structure of the process * Several **observations that** can indicate process weaknesses * A **risk-oriented random sample that** can be the starting point for further investigations

Since only about 10% of the estimated time was used for the analysis, there is still enough time to pursue these points. Of course, one could do many more analyzes on this data, but experience has shown that it is difficult to actually investigate all anomalies with the required level of accuracy.

We conclude with a brief look at the distribution of the sample according to various reasons. It makes sense to look at the tickets first, which are in the sample for multiple reasons:

```
[ ]: sample$reason %>% table()
frequency_incident_in_sample <- sample %>% group_by(number) %>%ÿ ÿsummarise(count
  = n())
hist(frequency_incident_in_sample$count)
```

And now the detail work begins...