Is everything OK with that baggage?

Time series analysis and forecasting

Damiano Pincolini

2025-11-05

Table of contents

1. INTRODUCTION

1.1. PROJECT GOAL

The aim of this work is to follow a complete pipeline in order to study a time series, analyse it and make some predictions using R and, specifically, tidyverts ecosistem.

Defining the proper sequence of activities, choosing the right tools to use and clearing the differences with a "standard" ML project are the main goals.

The time series has been downloaded from a work by Gabriel Santello on Kaggle (https://www.kaggle.com/datasets/gabrielsantello/airline-baggage-complaints-time-series-dataset/data) and contains monthly observations of baggage complaints from 2004 to 2010 for United Airlines, American Eagle, and Hawaiian Airlines. The variables in the data set include:

- Baggage - The total number of passenger complaints for theft of baggage contents, or for lost, damaged, or misrouted luggage for the airline that month.

- Scheduled - The total number of flights scheduled by that airline that month.

- Cancelled - The total number of flights cancelled by that airline that month.

- Enplaned - The total number of passengers who boarded a plane with the airline that month.

1.2. PROBLEM DEFINITION AND CHALLENGES

Time series is quite a specific kind of dataset which requires a different approach that a "standard" machine learning workflow:

- The purpose of this work is to deal with a dive deep into sort of "pure" Time Series mdels (like ARIMA, ETS, etc.) which are specifically designed to tackle this kind of problems, since they are based on the mathematics of temporal dependencies.
- Therefore, no classical ML will be taken into account, even if, given some context (many strong exogenous predictors, large amounts of data, complex or non linear temporal relationship) this could be the go-to option.
  EDA is supposed to be executed on the whole dataset.
  Time series analysis and prediction is not based on a dataset composed of target/output and predictor features, since the forecast relies mostly on the past values of the observed variable, its trend and its seasonality: external factors are meant to be few or, more likely, absent.

- The purpose is to understand how complaints (meaning the "Baggage" column) are shaped during time, which components they are made of, if either a trend, or a seasonality or a "pure noise" is detectable and, finally, how this time series can be used to predict future trend.
- The main challenges are how to properly inspect and prepare the dataset, execute explorative data analysis and get useful insight, look for a suitable model and set its parameters, define the correct moment to split into train and test set and, train, test and evaluate model's forecasting performance.

2. DATA PREPARATION

2.1. LOADING PACKAGES

Naturally, this step is a basic prerequisite for every data science project. To simplify and save space, I avoid recalling any code. The most important packages to be used are: tidyverse, SmartEDA, tsibble, feasts, fable, fabletools,

2.2. DATA IMPORT

After downloading the dataset from Kaggle (https://www.kaggle.com/datasets/gabrielsantello/airline-baggage-complaints-time-series-dataset/data), I've simply created the very first "original" data set called DataOrigin.

2.3. DATASET INTEGRITY CHECK

The first things to inspect are dataset structure, datatype and quality.

```
options(scipen = 999)

str(DataOrigin)
```

```
spc_tbl_ [252 x 8] (S3: spec_tbl_df/tbl_df/tbl/data.frame)
 $ Airline  : chr [1:252] "American Eagle" "American Eagle" "American Eagle" "American Eagle"
 $ Date     : chr [1:252] "01/2004" "02/2004" "03/2004" "04/2004" ...
 $ Month    : num [1:252] 1 2 3 4 5 6 7 8 9 10 ...
 $ Year     : num [1:252] 2004 2004 2004 2004 2004 ...
 $ Baggage  : num [1:252] 12502 8977 10289 8095 10618 ...
 $ Scheduled: num [1:252] 38276 35762 39445 38982 40422 ...
 $ Cancelled: num [1:252] 2481 886 1346 755 2206 ...
```

```
$ Enplaned : num [1:252] 992360 1060618 1227469 1234451 1267581 ...
- attr(*, "spec")=
 .. cols(
 ..   Airline = col_character(),
 ..   Date = col_character(),
 ..   Month = col_double(),
 ..   Year = col_double(),
 ..   Baggage = col_double(),
 ..   Scheduled = col_double(),
 ..   Cancelled = col_double(),
 ..   Enplaned = col_double()
 .. )
 - attr(*, "problems")=<externalptr>
```

Clearly, "Date" column has character format that must be converted into date format.

```
DataOriginQuality1 <- ExpData (DataOrigin, type = 1)

DataOriginQuality1|>
  kable(format = "latex",
        booktabs = TRUE)|>
  kable_styling(latex_options = c("striped", "hold_position"),
                position = "left",
                full_width = FALSE)
```

Descriptions	Value
Sample size (nrow)	252
No. of variables (ncol)	8
No. of numeric/interger variables	6
No. of factor variables	0
No. of text variables	2
No. of logical variables	0
No. of identifier variables	2
No. of date variables	0
No. of zero variance variables (uniform)	0
%. of variables having complete cases	100% (8)
%. of variables having >0% and <50% missing cases	0% (0)
%. of variables having >=50% and <90% missing cases	0% (0)
%. of variables having >=90% missing cases	0% (0)

As far as data quality is concerned, 100% of rows are complete. So, everything appears fine: just to be on the safe side, let's count nas for every column which are expected to be all equal to zero.

```
DataOriginNA <- tibble(colNames = colnames(DataOrigin),
                       NAs = colSums(is.na(DataOrigin)))
DataOriginNA|>
  kable(format = "latex",
        booktabs = TRUE)|>
  kable_styling(latex_options = c("striped", "hold_position"),
                position = "left",
                full_width = FALSE)
```

colNames	NAs
Airline	0
Date	0
Month	0
Year	0
Baggage	0
Scheduled	0
Cancelled	0
Enplaned	0

It is confirmed there are no missing values.

It's now time to move one and dive deep into single column's content.

```
DataOriginQuality2 <- ExpData (DataOrigin, type = 2)

DataOriginQuality2|>
  kable(format = "latex",
        booktabs = TRUE)|>
  kable_styling(latex_options = c("striped", "scale_down"),
                position = "left",
                full_width = FALSE)
```

Index	Variable_Name	Variable_Type	Sample_n	Missing_Count	Per_of_Missing	No_of_distinct_values
1	Airline	character	252	0	0	3
2	Date	character	252	0	0	84
3	Month	numeric	252	0	0	12
4	Year	numeric	252	0	0	7
5	Baggage	numeric	252	0	0	252
6	Scheduled	numeric	252	0	0	248
7	Cancelled	numeric	252	0	0	198
8	Enplaned	numeric	252	0	0	252

There are three distinct values for "Airlines" column.

If I want to carry the analysis on the whole structure of the dataset, "Airlines" will be the key of the time series, but since I want to keep this project simple mainly focusing on methods and process, rather than the specific content, undefined I will choose one airline and work on its data so that I avoid replicating the same analysis on three sub-sets.

Since we work with a time series it is important to assess whether it is regular or not, but this check must be postponed after next step (feature engineering) where date will be converted into date format, which is necessary to this purpose.

2.4. "TIME-INDIPENDENT" FEATURE ENGINEERING

At this stage feature engineering is aimed only to create "deterministic" features or "clean" data set from useless columns:

1. Select only one airline. As previoulsy stated, since I want to highlight the very plain and simple workflow of time series analysis, I'm looking for simplifying whatever is possible, starting from the number of sub series. American Eagle is the chosen one by pure chance.

2. After this selection, the column "Airline" will be no longer useful, since it will store the one and only name of "American Eagle" (the airline I've decided to keep). Thus, it can be deselected.

3. "Month" and "Year" columns will probably be useless, but can be kept for possible uses.

4. Finally, the analysis will be carried on absolute values and not on proportions (i.e. complaints vs enplanned); this should keep, again, everything as simple as possible and allow to focus on the analysis process in itself.

After feature engineering the quality check will be re-run.

```
Data <- DataOrigin|>
  filter(Airline == "American Eagle")|>
  mutate(Date = my(Date))|>
  select(-Airline)
```

```r
str(Data)
```

```
tibble [84 x 7] (S3: tbl_df/tbl/data.frame)
 $ Date     : Date[1:84], format: "2004-01-01" "2004-02-01" ...
 $ Month    : num [1:84] 1 2 3 4 5 6 7 8 9 10 ...
 $ Year     : num [1:84] 2004 2004 2004 2004 2004 ...
 $ Baggage  : num [1:84] 12502 8977 10289 8095 10618 ...
 $ Scheduled: num [1:84] 38276 35762 39445 38982 40422 ...
 $ Cancelled: num [1:84] 2481 886 1346 755 2206 ...
 $ Enplaned : num [1:84] 992360 1060618 1227469 1234451 1267581 ...
```

```r
DataQuality1 <- ExpData(Data, type = 1)

DataQuality1|>
  kable(format = "latex",
        booktabs = TRUE)|>
  kable_styling(latex_options = c("striped", "hold_position"),
                position = "left",
                full_width = FALSE)
```

Descriptions	Value
Sample size (nrow)	84
No. of variables (ncol)	7
No. of numeric/interger variables	6
No. of factor variables	0
No. of text variables	0
No. of logical variables	0
No. of identifier variables	4
No. of date variables	1
No. of zero variance variables (uniform)	0
%. of variables having complete cases	100% (7)
%. of variables having >0% and <50% missing cases	0% (0)
%. of variables having >=50% and <90% missing cases	0% (0)
%. of variables having >=90% missing cases	0% (0)

```r
DataQuality2 <- ExpData(Data, type = 2)

DataQuality2|>
  kable(format = "latex",
```

```
        booktabs = TRUE)|>
  kable_styling(latex_options = c("striped", "scale_down"),
                position = "left",
                full_width = FALSE)
```

Index	Variable_Name	Variable_Type	Sample_n	Missing_Count	Per_of_Missing	No_of_distinct_values
1	Date	Date	84	0	0	84
2	Month	numeric	84	0	0	12
3	Year	numeric	84	0	0	7
4	Baggage	numeric	84	0	0	84
5	Scheduled	numeric	84	0	0	84
6	Cancelled	numeric	84	0	0	81
7	Enplaned	numeric	84	0	0	84

Data quality is fine, so it's now possible to check for regularity, in order to assess that the proper sequence of dates is respected.

```
DataInterval1 <- Data |>
  distinct(Date) |>
  pull(Date) |>
  diff()|>
  as_tibble()|>
  count(value)

DataInterval1|>
  kable(format = "latex",
        booktabs = TRUE)|>
  kable_styling(latex_options = c("striped", "hold_position"),
                position = "left",
                full_width = FALSE)
```

value	n
28 days	5
29 days	2
30 days	28
31 days	48

There are 83 gaps out of 84 values which is what we can expect from a regular time series.

2.5. TSIBBLE CREATION, INSPECTION AND ADJUSTMENT

At this point, the first thing to do is to convert the tibble Data into a tsibble object, which is the R data type properly created for time series.

```
DataTsibble <- Data|>
  mutate(Date = yearmonth(Date))|>
  as_tsibble(index = Date)
```

Let's control if the tsibble is effectively regular.

```
is_regular(DataTsibble)
```

```
[1] TRUE
```

It is now necessary check for NAs. The control will be done for every value in the dataset, even if the focus will be on "Baggage" (complaints).

```
columnName <- Data|>
  select("Baggage":"Enplaned")|>
  colnames()

FunctionCheckNAs <- function(columnName){

  columnNameSubset <- Data|>
    select(all_of(columnName))

  tibble(
    column = columnName,
    NAs = sum(is.na(columnNameSubset))
  )
}

DataTibbleNAs <-
  map(.x = columnName,
      .f = FunctionCheckNAs)|>
  list_rbind()

DataTibbleNAs|>
  kable(format = "latex",
        booktabs = TRUE)|>
  kable_styling(latex_options = c("striped", "hold_position"),
                position = "left",
                full_width = FALSE)
```

column	NAs
Baggage	0
Scheduled	0
Cancelled	0
Enplaned	0

After having confirmed that there aren't any null values, it is advisable to check for duplicate entries for each single date.

```
DataTsibbleDuplicates <- DataTsibble|>
  index_by(Date)|>
  summarize(count = n())|>
  filter(count > 1)

DataTsibbleDuplicates|>
  kable(format = "latex",
        booktabs = TRUE)|>
  kable_styling(latex_options = c("striped", "hold_position"),
                position = "left",
                full_width = FALSE)
```

Date	count

Again: no duplicates in Date column.

Now, let's move to detect gaps due to irregularities caused by missing observations in the column(s) containing the time series.

```
DataTsibbleGaps <- DataTsibble|>
  scan_gaps()|>
  as_tibble()|>
  summarize(gapsCount = n())|>
  arrange(desc(gapsCount))

DataTsibbleGaps|>
  kable(format = "latex",
        booktabs = TRUE)|>
  kable_styling(latex_options = c("striped", "hold_position"),
                position = "left",
                full_width = FALSE)
```

gapsCount
0

No gaps to fill

3. EXPLORATIVE DATA ANALYSIS

It's worth to keep in mind some key points about EDA (and what follows):

1. During EDA, every analysis will be executed on the whole dataset. In time series analysis, the goal of EDA is to inform the model about the entire temporal pattern. No data leakage risk exist as long as the analysis is restricted to structural discovery and visualization (e.g., ACF, decomposition, stationarity tests), which only inform the model's form, and does not involve the creation of time-dependent numerical features (e.g., rolling averages or explicit lagged predictors) used to fed any kind of "classical" ML's training algorithm.
2. EDA is supposed to be executed on the entire dataset, and only after EDA, data partitioning takes place following a chronological approach (train set is composed of least recent occurencies, while test set have more recent data). Specifically: the chronological split (train test made of least recent observations and test set composed of most recents ones) must occur before any step that transforms the target variable (like differencing or scaling) or creates value-dependent features (like lags/rolling averages) for the final model.
3. The main goal of EDA is to identify the full temporal signature before defining the forecasting task.
4. During EDA it might be useful to use the dataset either in tibble or in tsibble datatype. For example, SmartEDA works with tibble and not tsibble. It's appropriate to create the tibble as well to use if and when necessary alongside the tsibble.

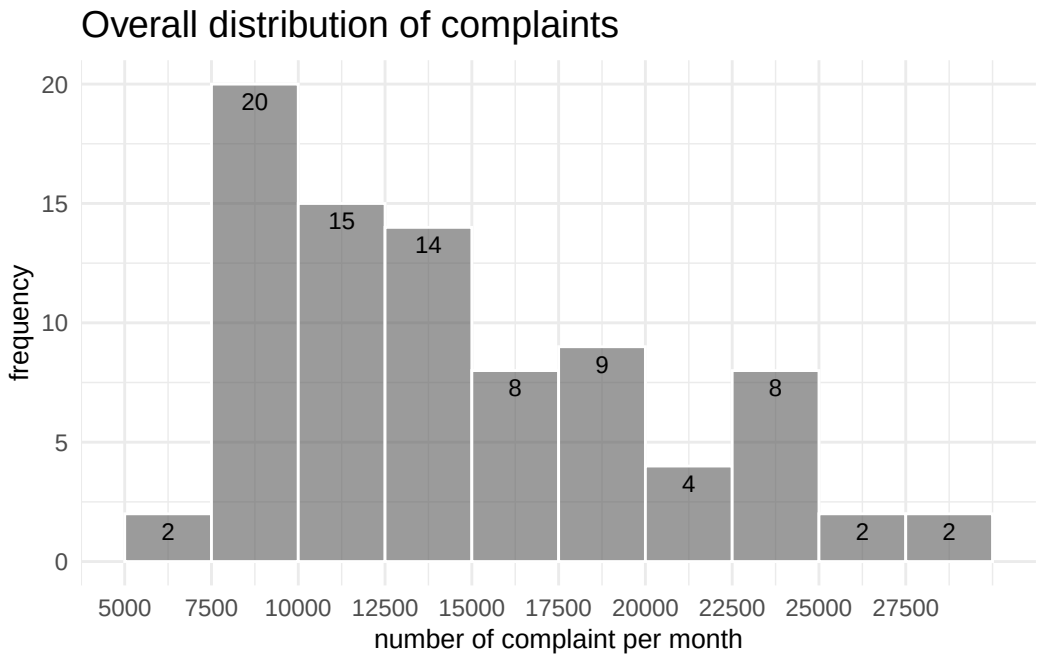
```
options(scipen = 999) # Avoid scientific notation
options(digits = 3)   # Set number of significant digits to 4

DataTibble <- DataTsibble|>
  as_tibble()
```

3.1. DISTRIBUTION ANALYSIS

3.1.1. Distribution visualization

The very first way to visualize distribution is by plotting an histogram with a give binwidth (2500 in this case).

```r
DataTibble |>
  ggplot(aes(x = Baggage)) +
  stat_bin(
    aes(y = after_stat(count)),
    binwidth = 2500,
    boundary = 0,
    alpha = 0.6,
    colour = "white",
    geom = "bar"
  ) +
  stat_bin(
    aes(y = after_stat(count), label = after_stat(count)),
    binwidth = 2500,
    boundary = 0,
    geom = "text",
    size = 3,
    vjust = 1.5
  ) +
  scale_x_continuous(breaks = seq(0, max(DataTibble$Baggage), by = 2500)) +
  labs(
    title = "Overall distribution of complaints",
    x = "number of complaint per month",
    y = "frequency"
    ) +
  theme_minimal() +
  theme(plot.title = element_text(size = 14),
        axis.title.x = element_text(size = 10),
        axis.title.y = element_text(size = 10)
        )
```



A normal distribution is not clearly detectable given bins of 2500 each.

3.1.2. Skewness, kurtosis and outlier

```
DataEdaNum <- DataTibble|>
  ExpNumStat(#by = "G",
             Qnt = c(0.25, 0.75),
             MesofShape = 2,
             Outlier = TRUE,
             round = 2)|>
  select(-c("Group", "TN", "nNeg", "nPos", "nZero",
            "NegInf", "PosInf", "NA_Value", "Per_of_Missing", "sum"))

DataEdaNum|>
  kable(format = "latex",
        booktabs = TRUE)|>
  kable_styling(latex_options = c("striped", "scale_down"),
                position = "left",
                full_width = FALSE)
```

	Vname	min	max	mean	median	SD	CV	IQR	Skewness	Kurtosis	25%	75%	LB.25%	UB.75%	nOutliers
2	Baggage	7052	28556	14619.2	13111.0	5695.84	0.39	9057.8	0.70	-0.58	9960.00	19017.75	-3626.6	32604.4	0
4	Cancelled	266	3712	1414.8	1341.0	717.42	0.51	943.2	0.76	0.38	885.25	1828.50	-529.6	3243.4	1
5	Enplaned	992360	1727570	1396725.6	1391112.5	166162.31	0.12	219492.8	-0.13	-0.35	1290414.75	1509907.50	961175.6	1839146.6	0
1	Month	1	12	6.5	6.5	3.47	0.53	5.5	0.00	-1.22	3.75	9.25	-4.5	17.5	0
3	Scheduled	31965	48096	41314.1	42021.0	4187.57	0.10	7635.0	-0.23	-1.21	37328.25	44963.25	25875.8	56415.8	0

Skewness is equal to 0.70. Values bigger than 1 or smaller than -1 recall a high skewness. Skwness is moderate. It is a slightly platykurtic distribution withthinner tails and extreme values (outliers) less likely. In fact, they are basically not detected by this analysis (especially in "Baggage" column).

3.1.3. Normality test

To asses distribution normality, we can switch from plots and tables to figure and run a statistical test (Shapiro Wilk).

```
shapiroTest <- shapiro.test(DataTibble$Baggage)

DataTibbleShapiroTest <- tibble(BaggageShapiroW = shapiroTest$statistic,
                                BaggageShapiroP_value = shapiroTest$p.value)

DataTibbleShapiroTest|>
  kable(format = "latex",
        booktabs = TRUE)|>
  kable_styling(latex_options = c("striped", "hold_position"),
                position = "left",
                full_width = FALSE)
```

BaggageShapiroW	BaggageShapiroP_value
0.922	0

p-value is massively below 0.05 treshold. We don't accept null hypotesis of normality.

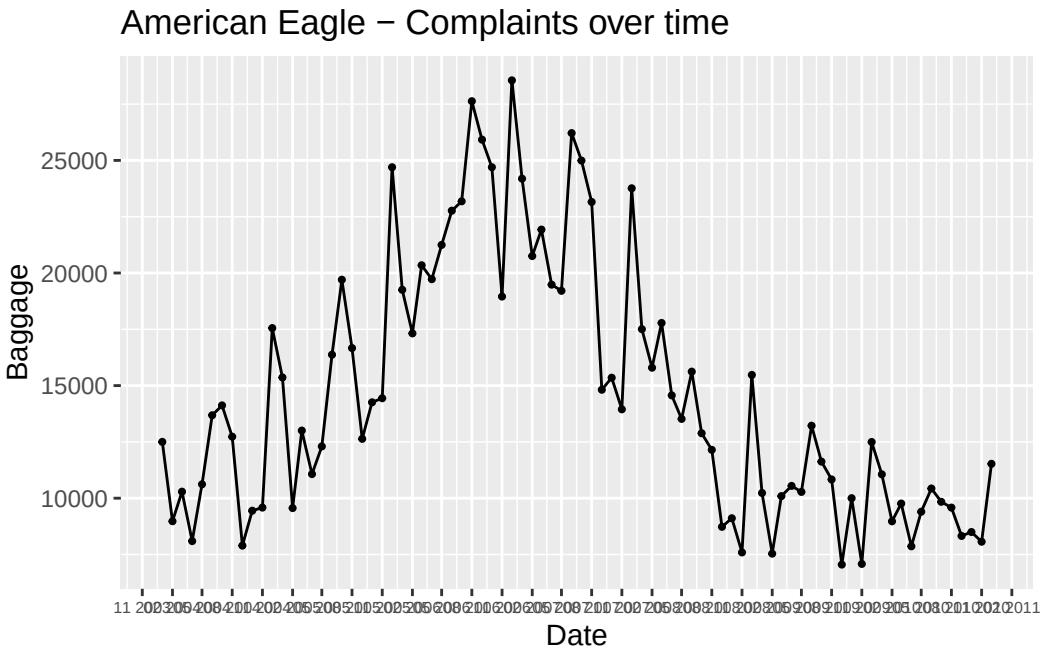
3.2. TIME SERIES VISUALIZATION

Visualization refers to both the whole time series and its components. The aim is to invetigate into time series' trend, seasonality and noise.

3.2.1. Trend and seasonality detection

Let's start from the overall trend.

```
DataTsibble|>
  ggplot(aes(x = Date,
             y = Baggage)
         )+
  geom_line(linewidth = 0.5)+
  geom_point(size = 0.75) +
  scale_x_yearmonth(
    date_breaks = "3 months",
    date_labels = "%m %Y"
    ) +
  ggtitle("American Eagle - Complaints over time") +
  theme(axis.text.x = element_text(size = 6))
```

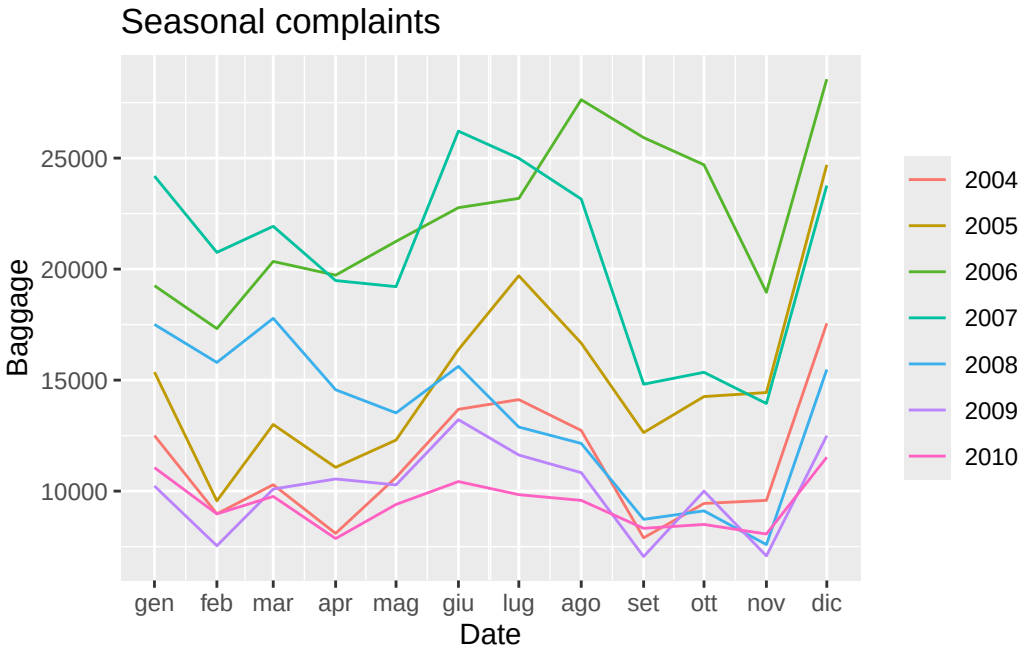


Complaints have increased from 2004 to 2007, when they reached their peak, and then have slowed down until 2010. From 2007 till the end of these observation, a downward trend is detectable.

Seasonality | The aim is to spot repeating cycles across seasons and years. The overall plot shows monthly data with quite regular peaks roughly occurring during summer and winter

season (july, august and december, january). This should point out some kind of seasonality. Let's consider the year as the seasonal unit.

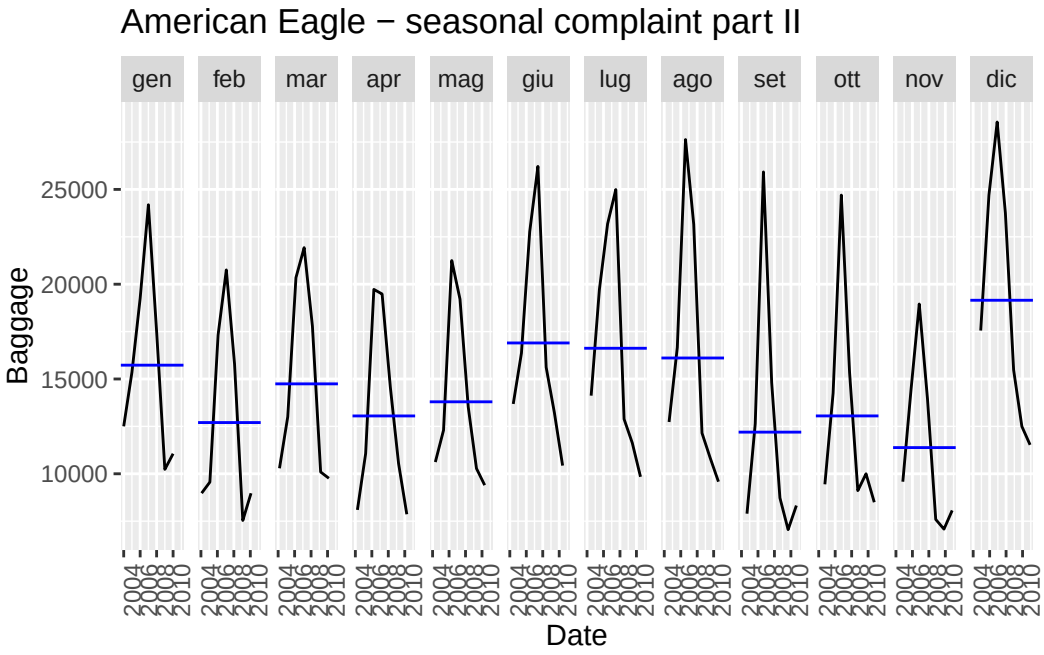
```
DataTsibble |>
  ggtime::gg_season(y = Baggage,
                    period = "1y",
                    #facet_period = "1m",
                    label_repel = TRUE)+
  ggtitle("Seasonal complaints")
```



As expected, the curves of every year are pretty similar and show similar paths alongside each month. A kind of seasonality seems to exist.

It is possible to check this insight using gg_subseries()

```
DataTsibble |>
  gg_subseries(y = Baggage,
               period = "1y")+
  ggtitle("American Eagle - seasonal complaint part II")
```



Looking at the monthly average of complaints, it seems confirmed that holiday period record more issues: dec, jan, june, july and august.

3.2.3. Decomposition

The decomposition and component analysis follow this steps:

1. define a model (STL can be a default choice at this stage)
2. decompose the time series into its components
3. visualize decomposition results
4. analyse components' features

3.2.3.1. Define the model

In order to set the STL model, the following parameters have to be set:

- trend(window): 13. This allows the trend to capture changes over approximately a year, but still be responsive to longer-term shifts. You should use an odd number.
- trend(degree): 1. This is the degree of the polynomial used for the LOESS trend smoothing. A degree of 1 uses a linear polynomial is the most common choice, computationally efficient and often sufficient.

- season(window): 13. A value of 13 is a common choice, as it's just over a full seasonal cycle and provides a good balance of smoothing without losing important seasonal details.
- season(degree): 1. This is the degree of the polynomial for the LOESS seasonal smoothing. Similar to the trend degree, a degree of 1 is standard and uses a linear polynomial.
- robust: TRUE. just to be on safe side even if no outliers have been detected.

```
DataDecStl <- DataTsibble|>
  model(STL(Baggage ~
              trend(window = 13,
                    degree = 1) +
              season(window = 13,
                     degree = 1),
            robust = TRUE))
```

3.2.3.2. Extract components

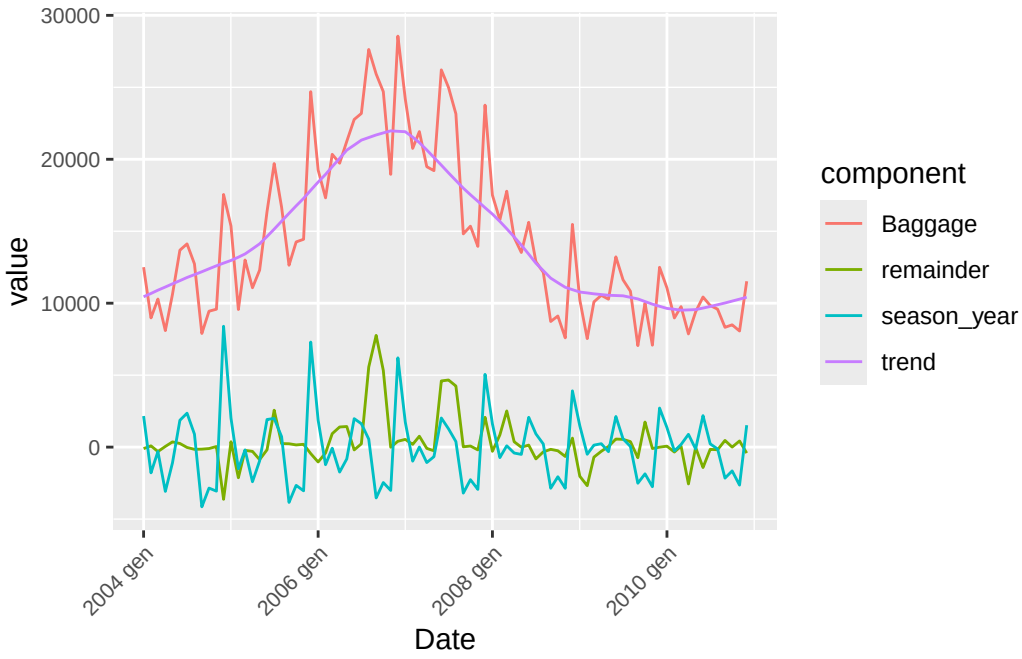
The goal here is to isolate every component of the "Baggage" time series. I also introduce a test comparing the sum of time series' components with its original values: if the difference is null, decomposition has worked fine.

```
ComponentStl <- components(DataDecStl)|>
  mutate(sum = (trend + season_year + remainder),
         check = (Baggage - sum))
```

3.2.3.3. Plot the components

Rather than using autoplot(), I prefer to pivot the dataset and use ggplot in order to get some better visual feedback.

```
ComponentStl|>
  pivot_longer(cols = Baggage:remainder,
               names_to = "component",
               values_to = "value")|>
  ggplot(aes(x = Date,
             y = value,
             color = component))+
  geom_line()+
  theme(axis.text.x = element_text(size = 8, angle = 45, vjust = 1, hjust = 1),
        axis.text.y = element_text(size = 8))
```



3.2.3.4. Analyse time series features

After quite a lot of attempts with features() from feasts{} package, I've found out that, since features() applies formulas on components resulting from default values decomposition, it is better, for explicability reasons, to compute manually every features (see below).

The access to transformed (if necessary) time series components and the ease of working with formulas I'm more familiar with are the two main reasons why I've opted for a manual calculation of features.

```
StlFeatures <- as_tibble(ComponentStl)|>
  summarize(StlTrendStrength  =
              max(0, 1 - var(remainder, na.rm = TRUE) / var(season_adjust, na.rm = TRUE)),
            StlSeasonalStrength =
              max(0, 1 - var(remainder, na.rm = TRUE) / var(season_year + remainder, na.rm =
            StlRemainderStrength =
              1 - var(trend + season_year, na.rm = TRUE) / var(Baggage, na.rm = TRUE),
            StlSeasonalPeak =
              which.max(season_year) %% frequency(ComponentStl),
            StlSeasonalTrough =
              which.min(season_year) %% frequency(ComponentStl),
            StlSpikiness = var(diff(remainder, differences = 2, na.rm = TRUE)),
            StlLinearity = summary(lm(trend ~ seq_along(trend)))$r.squared,
```

```
            StlCurvature = sum(abs(diff(trend, differences = 2)), na.rm = TRUE)
  )|>
  pivot_longer(cols = StlTrendStrength:StlCurvature,
               names_to = "featureName",
               values_to = "featureValue")

StlFeatures|>
  kable(format = "latex",
        booktabs = TRUE)|>
  kable_styling(latex_options = c("striped", "hold_position"),
                position = "left",
                full_width = FALSE)
```

featureName	featureValue
StlTrendStrength	0.892
StlSeasonalStrength	0.623
StlRemainderStrength	0.268
StlSeasonalPeak	0.000
StlSeasonalTrough	9.000
StlSpikiness	6968729.134
StlLinearity	0.143
StlCurvature	3051.153

Before analysing results, some disclaimer must be made:

1. peak and trough | seasonal peaks and troughs don't necessarily manifest as perfectly repeated patterns every single cycle. Instead, they represent the dominant recurring points where the seasonal component tends to be highest (peak) or lowest (trough) on average across time. These peaks and troughs aren't always exact, because external factors (random noise, demand variations, holidays, promotions, etc.) impact actual monthly values. The seasonal component is an average behavior, meaning it doesn't rigidly repeat every week: it's a general tendency, not a strict rule. Peaks/troughs might shift slightly depending on how decomposition was done (e.g., smoothing choices or window length). Finally, a peak and a trough can be spotted on the same day, even if it sounds contradictor at first; it's actually possible due to the way seasonal decomposition works.

2. StlRemainderStrength. It is calculated like: 1 - var(trend + season_week) / var(number) It measures the fraction of the total variance not explained by the trend and seasonality components. Higher values mean more noise remains in the data, suggesting weaker predictable structure. Outcome ranges between 0 and 1.

Now let's focus on the outcomes of component feature analysis.

- Trend Strength (0-1): 0.89 means that a clear trend is detectable.
- Seasonal Strength (0-1): 0.62 represents a quite strong seasonality component.
- Remainder Strength (0-1): 0.27. Not weak, but still less significant than trend and season. The remaining "noise" or irregular fluctuations in the data are a smaller, but still present, part of the overall variation.
- Seasonal Peak & Trough: 0 and 9. December (the result of 0 typically maps to the last month of the cycle) and September show, on average,the highest and the lowest number of complaints.
- Spikiness: 6968729.134. This value is very high and suggests the series has many sharp, sudden changes or outliers.
- Spikiness measures the roughness of the remainder component. A high value like this indicates that the irregular part of your time series is very "spiky" or volatile. It implies that your data is prone to large, unpredictable shocks that were not captured by the trend or seasonality.
- Linearity & Curvature: 0.14 and 3051.153. A low linearity value indicates that the trend is not linear. The trend is likely rising and falling, or changing direction, rather than following a simple straight line. A high curvature value like 3051.153 is consistent with the low linearity score. It confirms that the trend is indeed highly curved and not a straight line.

The analysis shows a time series with a very strong, non-linear trend and a moderately strong seasonal pattern. The irregular component is highly volatile and contains many spikes or outliers, which are a dominant feature of the series.

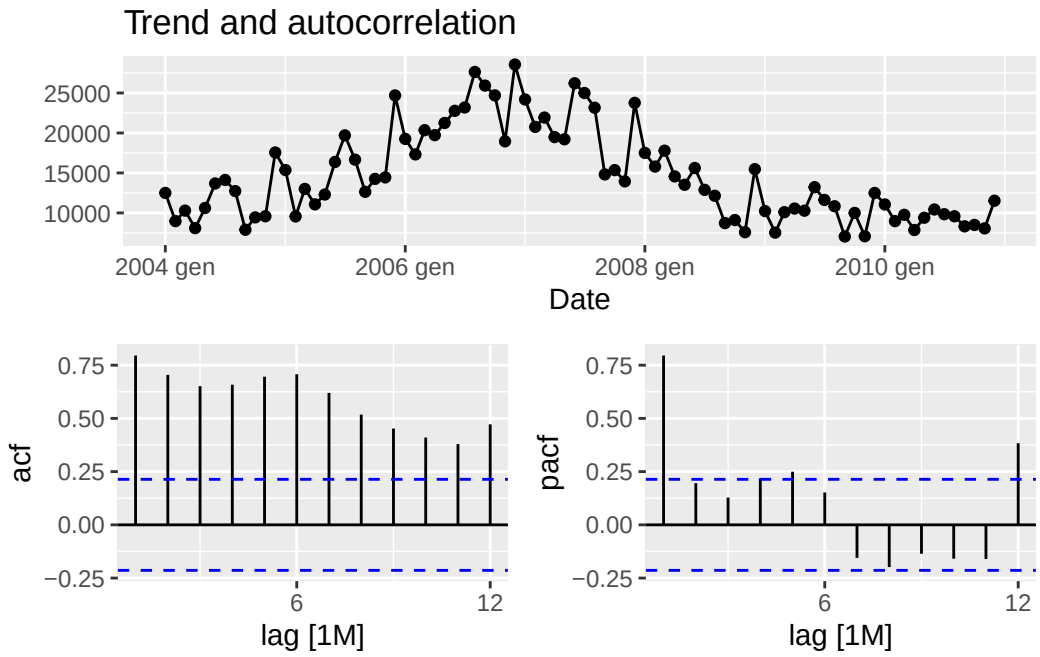
3.2.4. Autocorrelation analysis

A lag up to 12 periods (in this case: months) is consistent with data handled.

Since time series is monthly, a lag equal to 12 means that autocorrelation analysis will beexecuted for the first 12 months:

- ACF will show correlation between an observation and the observation of day -1, -2 until -12.
- PACF will will show correlation between an observation and the observation of 12 months before.

```r
DataTsibble|>
  gg_tsdisplay(y = Baggage,
               plot_type = 'partial',
               lag = 12) +
  labs(title = "Trend and autocorrelation", y = "")
```



3.2.4.1. Time Series Plot

The overall time series plot has already been observed (see 3.2.1.). Main points to recap are:

- Trend | Time seris trend has a visible upward trend from 2004, peaking around mid-2007. After the peak, there's a clear downward trend towards the end of the series. This suggests a non-stationary mean.
- Seasonality | The plot also shows a cyclical or seasonal pattern. Data seems to peak and trough on a regular, recurring basis, which suggests a seasonal component.
- Irregularity | There is some noise or random fluctuation around the trend and seasonal patterns. ACF (Autocorrelation Function)

3.2.4.2. ACF Plot

The ACF plot shows a slow decay in the correlation as the lag increases. All the lags up to 12 months are significant. This is a classic signature of a non-stationary series that has a trend and/or a seasonal component. The high correlation at lag 12 confirms the presence of strong yearly seasonality.

3.2.4.3. PACF Plot

The PACF plot shows two significant spike at lag 1 (the biggest) and 12 that confirm yearly seasonality. The current month's value is thus correlated with (and can be explained by) the values of 1 and 12 months ago. This appears consistent with a seasonal data structure.

3.2.4.4. Conclusions

This is a classic pattern for a series with a strong trend. After the first lag, the partial autocorrelation drops off quickly and stays within the significance lines, except for a small spike at lag 12. This further supports the conclusion of yearly seasonality.

3.3 STATIONARY TEST

A stationary test determines if a time series has consistent statistical properties over time: it sets a null hypotesis (H0) which states that the time series in non-stationary.

This is a critical step because many common forecasting models, such as ARIMA, assume the data is stationary to produce reliable results. A stationary test provides a statistical conclusion on whether the time series has:

- A constant mean: The average value of the data doesn't change over time. Non-stationary data often shows an upward or downward trend.
- A constant variance: The spread or volatility of the data around the mean stays the same. Non-stationary data might show increasing or decreasing variance over time.
- A constant autocorrelation: The relationship between a data point and its past values is consistent, regardless of the time period.

Like every hypotesis test, a test statistic and its p-value are returned and represent the elements on which decide whether to accept or reject the null hypothesis (H0).

```r
EdaStationarityTest <- DataTsibble|>
  features(Baggage,
           unitroot_pp)

EdaStationarityTest|>
  kable(format = "latex",
        booktabs = TRUE)|>
  kable_styling(latex_options = c("striped", "hold_position"),
                position = "left",
                full_width = FALSE)
```

pp_stat	pp_pvalue
-2.62	0.094

The value of statistic is -2.62: the more negative the statistic, the stronger the evidence to reject the null hypothesis. The value of -2.62 is not such a strong negative number.

P-value is 0.0944. With such a p-value (sigthly bigger than 0.05), null hypotesis (the series has a unit root, thus is non-stationary) can be accepted, not without paying any attention.

Stationary test has given clues not so clear and incontrovertible. All in all, accepting a non-stationary time series is actually consistent with other information we've gathered so far.

To sum up, the time series is likely non-stationary, which means that it's unsuitable for models that require stationarity, such as ARIMA.

To make the series stationary, it will be necessary differencing.

3.4. VARIABILITY ANALYSIS

Analysing variability using moving averages and moving standard deviation is aimed to check for a crucial assumption known as homoscedasticity.

Moving averages can be "trailing" or "centered".

Trailing moving average is calculated on the current observation plus the previous n values (i.e: 6). This is the most common approach in forecasting because it relies only on past values. It keeps the structure aligned with time-dependent models like ARIMA.

Centered Moving Average uses n/2 (i.e: 3, if we keep 6 observations) previous and n/2 future values around each point. It creates a smoother result by balancing past and future trends. It works well for retrospective analysis but is not usable in real-time forecasting since future values are needed.

Generally speaking, trailing moving average is used for forecasting because it relies only on past data, while centered moving average is good at analyzing patterns since it returns a more symmetric view. In this case I try both, even if the scope here would bring to the latter option.

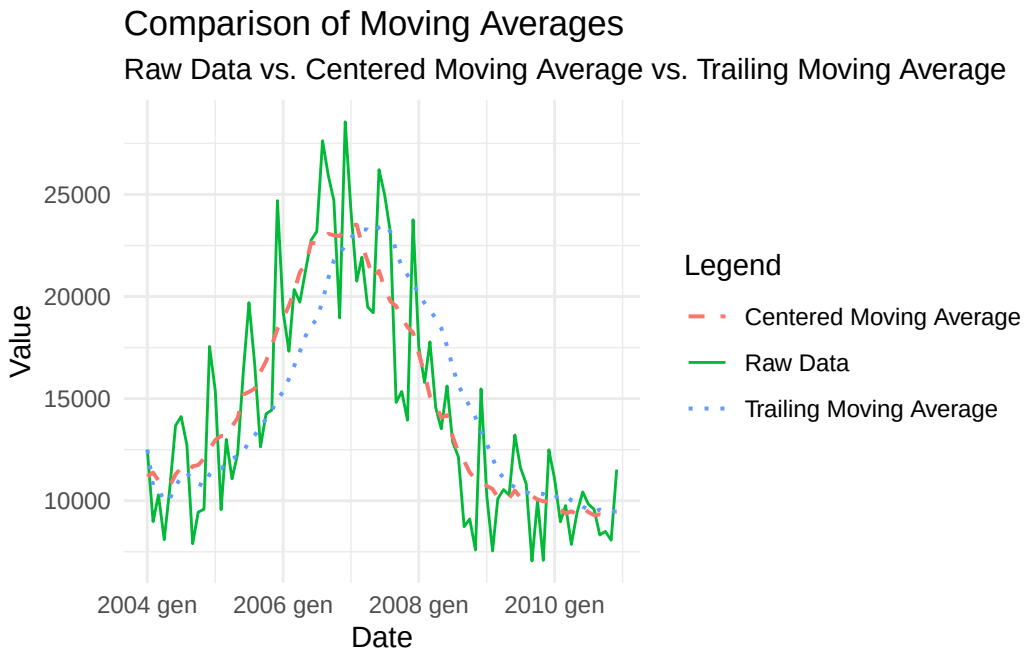
For a monthly time series, like the object of this analysis, a good practice is to choose a window that is either a multiple of the seasonal period (the most common seasonal period for monthly data is 12 months) and an odd number of months centered on the season expecially for centered moving averages to properly handle seasonality.

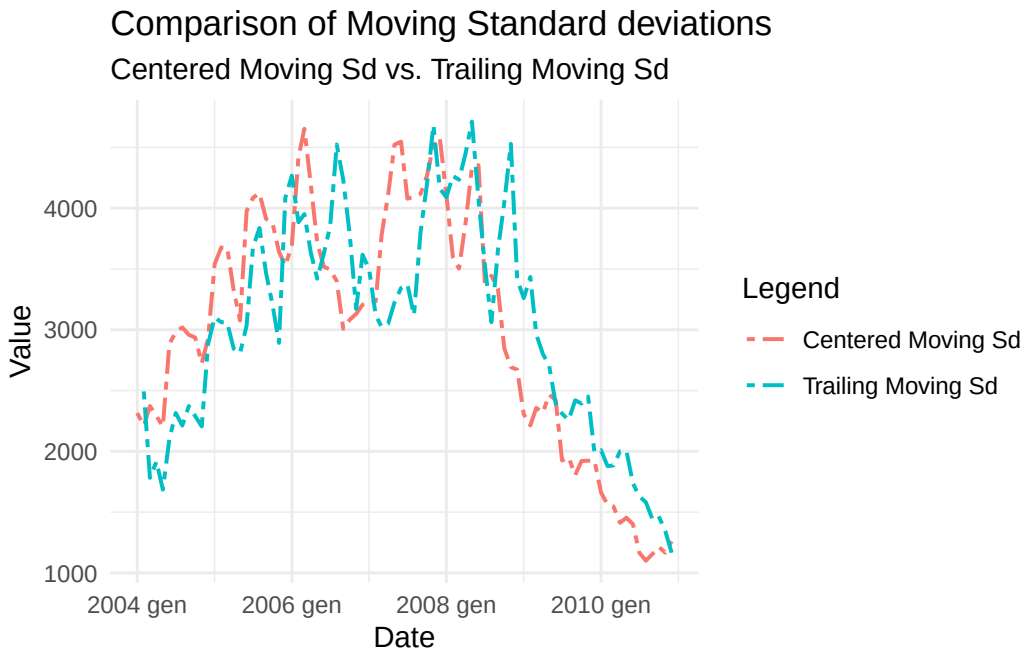
A window of 13 is a very common choice for monthly data, as it covers a full year plus one month, making the average centered on the middle of the year and effectively smoothing out the seasonal pattern.

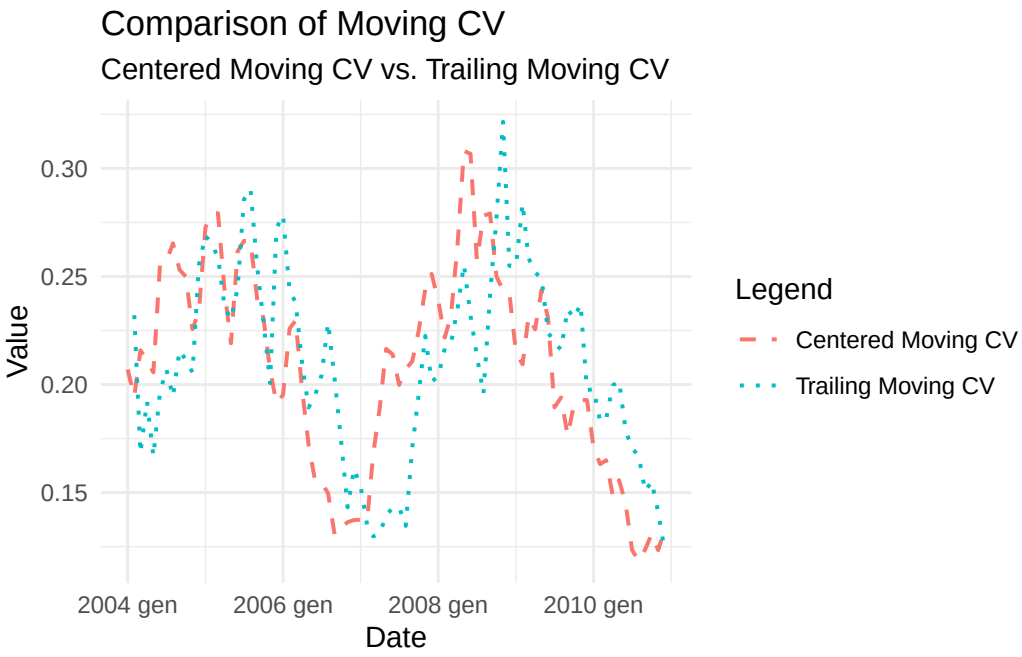
A window of 12 can be kept for a trailing average. I need to create a function for coefficient of variation since it's not a built in formula in slider

```
FunctionCv <- function(x) {
  sd(x, na.rm = TRUE) / mean(x, na.rm = TRUE)
}


EdaMovingAv <-  DataTsibble|>
  mutate(centeredBaggageAvg = slide_mean(Baggage,
                                         before = 6,
                                         after = 6),
         centeredBaggageSd = slide(Baggage, .f = sd,
                                   .before = 6,
                                   .after = 6),
         centeredBaggageCv = slide(Baggage, .f = FunctionCv,
                                   .before = 6,
                                   .after = 6),
         trailingBaggageAvg = slide_mean(Baggage,
                                         before = 11),
         trailingBaggageSd = slide(Baggage, .f = sd,
                                   .before = 11),
         trailingBaggageCv = slide(Baggage, .f = FunctionCv,
                                   .before = 11))|>
  mutate(centeredBaggageSd = unlist(centeredBaggageSd),
         centeredBaggageCv = unlist(centeredBaggageCv),
         trailingBaggageSd = unlist(trailingBaggageSd),
         trailingBaggageCv = unlist(trailingBaggageCv))
```

A further step is to visualize all of this.


Comparison of Moving Averages
Raw Data vs. Centered Moving Average vs. Trailing Moving Average
25000
20000
15000
10000
Value
2004 gen
2006 gen
2008 gen
2010 gen
Date
Legend
Centered Moving Average
Raw Data
Trailing Moving Average


Comparison of Moving Standard deviations
Centered Moving Sd vs. Trailing Moving Sd
4000
3000
2000
1000
Value
2004 gen
2006 gen
2008 gen
2010 gen
Date
Legend
Centered Moving Sd
Trailing Moving Sd



Center and trailing moving average walk along pretty similarly. They show a downward trend, after the peak at the beginning of 2007.

The most important thing to observe here are Moving Standard Deviation and Moving Coefficient of Variation which express the same concept (variability) with different scales. They are not constant and, generally speaking, follow the trend of observations. A non-constant Moving SD/CV, with varying values over time (increasing, decreasing, or fluctuating erratically) lead to assume that the series is heteroscedastic (non-constant variance). Non-constant standard deviation violates a fundamental assumption of many statistical models (including ARIMA). Implications are relevant and refer to:

- Parameter estimates (like AR/MA terms) are still unbiased but are inefficient (they have larger standard errors than necessary).
- Standard errors, t-statistics, and p-values are biased and unreliable, which can lead to incorrect conclusions about the significance of the model's coefficients.
- Forecast intervals (confidence bounds) will be too wide in some periods and too narrow in others, failing to capture the true changing risk.

Variance-stabilizing transformation is likely required.

3.5. IMPLICATIONS FOR MODELING

3.5.1. Distribution analysis

1. Both from a visual glance, and from Shapiro-Wilk test, the distribution doesn't clearly show normal distribution.
2. Skewness is moderate and there are no outliers.

3.5.2. Visualization

3.5.2.1. Trend and season

1. Regarding to trend, complaints have increased from 2004 to 2007, when they reached their peak, and then have slowed down until 2010. From 2007 till the end of these observation, a downward trend is detectable.

2. As far as seasonality is concerned, looking at the monthly average of complaints, it seems that holiday time records more issues: December, January, June, July and August.

Takeaway: a seasonal (monthly) model like SARIMA may be useful and consistent with data structure.

3.5.2.2. Component analysis

1. Time series' component analysis returns a very strong, non-linear trend and a moderately strong seasonal pattern.

2. The irregular component is highly volatile and contains many spikes, which are a dominant feature of the series.

Takeaway: volatility, along with seasonality, suggest differencing.

3.5.2.3. Autocorrelation

1. The ACF (Autocorrelation Function) plot shows a slow decay in the correlation as the lag increases. All the lags up to 12 months are significant. This is a classic signature of a non-stationary series that has a trend and/or a seasonal component. The high correlation at lag 12 confirms the presence of strong yearly seasonality.

2. The PACF (Partial Autocorrelation Function) plot shows a single significant spike at lag 1 and 12. This is a classic pattern for a series with a strong trend. After the first lag, the partial autocorrelation drops off quickly and stays within the significance lines, except for a spike at lag 12 which supports the conclusion of yearly seasonality.

Takeaway: trend e seasonality seem confirmed.

3.5.3. Stationary analysis

This time series is likely non-stationary, which means that it's unsuitable for models that require stationarity, such as ARIMA, unless differencing it.

Takeaway: the need for differencing is confirmed.

3.5.4. Variability analysis

This step have detected heteroscedasticity which suggest to use a power transformation to stabilize the variance.

3.5.5. Conclusions

Based on the exploratory analysis, the time series is characterized by a non-stationary process with both a clear trend and a strong yearly seasonal pattern. The Phillips-Perron unit root test, with a p-value greater than 0.05, confirms the non-stationarity.

- Takeaway 1: differencing is needed | The ACF plot's slow decay and the significant spikes in the PACF plot at lag 1 confirm the presence of a trend, while the significant correlation at lag 12 in both plots indicates strong yearly seasonality.
- Takeaway 2: the model should take seasonality into account. | Further analysis using moving averages confirmed these findings. The 13-month centered moving average effectively removed the seasonality, revealing a complex, non-linear trend that rises from 2004 to a peak in 2007 before declining. The moving standard deviation and coefficient of variation plots indicate that the series' variability is stable over time. This suggests that the assumption of constant variance is met.
- Takeaway 3: data transformation (such as a log or Box-Cox) is necessary.

In conclusion, the time series is a suitable candidate for a SARIMA model due to its non-stationarity, seasonality, and stable variance. The modeling process will begin by applying a transformation and a first-order seasonal differencing (D=1) with a lag of 12 to remove the strong seasonal component. The ACF and PACF plots of the seasonally differenced series will then be used to determine if an additional non-seasonal differencing is required and to identify the appropriate non-seasonal (p, q) and seasonal (P, Q) parameters for the model.

4. MODELING

This stage's aim is to train and test a SARIMA model.

4.1. DATA PARTITIONING AND FEATURE REFINEMENT

Just after splitting data into train and test, we need either to stabilize variance (via transformation) and to reduce/erase non-stationarity (via differencing). These activities are to be operated on the train set and in order to avoid data leakage risk.

Note that unlike standard machine learning models where data can often be randomly split, time series analysis requires chronological integrity in the split. Here's why:

1. In time series forecasting, the test set must be the most recent observations to simulate real-world predictive scenarios.
2. Preserving order ensures that both train and test data follow their natural sequence, maintaining temporal dependencies.
3. No random shuffling! Unlike classic ML models (classification or regression), time-dependent relationships mean that randomly splitting would break meaningful patterns.

To sum up, data partition must follow a chronological approach: train set is to be composed by least recent entries, while test set by the most recent.

In this phase, there could be a third phase which is the time-dependant feature engineering. If we want to rely on a "classical" ML supervised model and along with the target variable, we need to create any predictor, this is the point to act.

4.1.1 Data Partitioning

At this stage, the tsibble formatted time series is to be splitted; splitting tsibble is the best practice because it ensures that the crucial time index and key structure of the time series data are preserved and correctly handled during the partitioning process.

```
DataTrainSize <- round(nrow(DataTsibble) * 0.8)  # 80% for training

DataTrain <- DataTsibble|>
  slice(1:DataTrainSize) # slice() è tidyverse

DataTest <- DataTsibble|>
  slice((DataTrainSize + 1):nrow(DataTsibble))
```

Data partitioning has been carried on at this early stage of the project with the aim of deliver all the preparatory work.

Note that in a classical ML model (i.e. regression, classification), EDA is typically performed on the train set only in order to ensure that no insight from the test set leak into model building and, thus, to maintain the integrity of the validation process.

In the case of time series analysis, using the full dataset for **visual** exploratory analysis (decomposition, autocorrelation patterns, etc.) does not cause data leakage because we're only understanding structure, not estimating parameters. However, any **transformation parameters** (like Box-Cox) should still be calculated from the training set only.

4.1.2. Transformation and differencing

EDA has made clear that both transformation and differencing are needed because of heteroschedasticity (variance is not stable) and non-stationariety respectively.

As far as transformation is concerned a Box-Cox transformation will be used.

With regard of differencing, since both seasonality and non stationery have been detected, starting with seasonal differencing is the preferable because it often removes both the seasonality and a significant part of the non-seasonal trend in one step, potentially simplifying the required additional non-seasonal differencing. These are the potential steps to follow:

1. Seasonal differencing removes recurring fluctuations tied to the calendar (like monthly cycles or annual seasonality).
2. After removing seasonality, any remaining non-stationary trend becomes more visible and cleaner.
3. Reversing this order (trend differencing before seasonal differencing) could bring the trend component to interfere with detecting the seasonal structure clearly.

4.1.2.1 Transforming

Once the transofrmation recipe has been set (based on train set to avoid data leakage), it actually can be used to transform both train and test set.

It is important to highlight that the test set is transformed using the same parameters (lambda) learned from the training set to maintain consistency. However, note that final forecast comparison will be done on the original scale after inverse transformation.

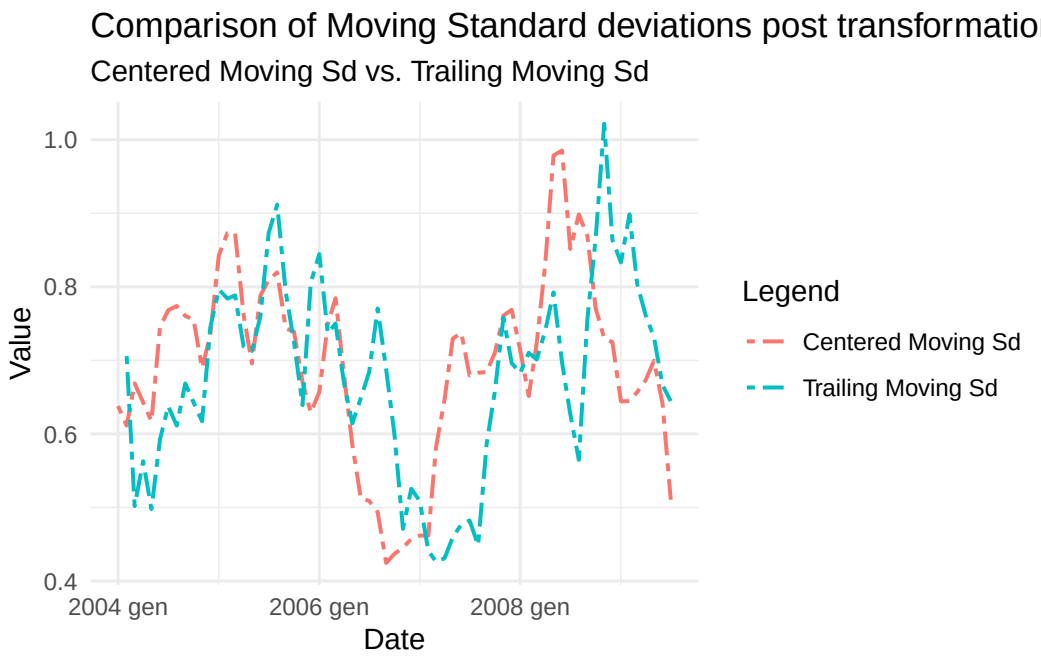
```
# recipe
TransformationRecipe <- DataTrain|>
  recipe(Baggage ~ Date)|>
  step_BoxCox(Baggage)|>
  prep()

# lambda extraction (to be used for reverse transformation)
TransformationLambdaValue <- TransformationRecipe$steps[[1]]$lambda
```

```
# check transformation's outcome
DataTrainTransformed <- TransformationRecipe|>
  juice()|>
  as_tsibble(index = Date)

# tsibble convertion
DataTestTransformed <- TransformationRecipe |>
  bake(new_data = DataTest)|>
  as_tsibble(index = Date)
```

Has transformation produced useful effects in terms of variability stabilization?

```
#| warning: false

DataTrainTransformedMovingSd <-  DataTrainTransformed|>
  mutate(
    centeredBaggageSd = slide(Baggage, .f = sd, .before = 6, .after = 6),
    trailingBaggageSd = slide(Baggage, .f = sd, .before = 11)
         )|>
  mutate(
    centeredBaggageSd = unlist(centeredBaggageSd),
    trailingBaggageSd = unlist(trailingBaggageSd)
         )

DataTrainTransformedMovingSd|>
  ggplot(aes(x = Date)) +
  geom_line(aes(y = centeredBaggageSd, color = "Centered Moving Sd"),
            linewidth = 0.7,
            linetype = "twodash") +  # Trailing Moving Sd
  geom_line(aes(y = trailingBaggageSd, color = "Trailing Moving Sd"),
            linewidth = 0.7,
            linetype = "twodash") +  # Trailing Moving Sd
  labs(title = "Comparison of Moving Standard deviations post transformation",
       subtitle = "Centered Moving Sd vs. Trailing Moving Sd",
       x = "Date", y = "Value",
       color = "Legend") +  # Legend title
  theme_minimal()
```

```
Warning: Removed 1 row containing missing values or values outside the scale range
(`geom_line()`).
```



Standard deviation has assumed a more regular shape with a downwards trend, it assumes values ranging from 0.65 to 0.35 (if we consider the centered SD).

The more stable standard deviation (ranging from 0.35-0.65) compared to the original data confirms successful variance stabilization, making the series more suitable for ARIMA modeling which assumes constant variance.

4.1.2.2 Differencing

Let's execeute a unit root test first.

```
DataTrainTransformed |>
  features(difference(Baggage,
                      lag = 12),
           unitroot_kpss)
```

```
# A tibble: 1 x 2
  kpss_stat kpss_pvalue
      <dbl>       <dbl>
1      1.13        0.01
```

Test hypotheses are:

- H (null): The series IS stationary.
- H (alternative): The series is NOT stationary.

With results such as 1.13 and 0.01 for statistic and p-value respectively, a p-value lower than 0.05 suggest to reject the null hypothesis.

This lead to the conclusion that the seasonally differenced series is still NOT stationary.

This suggests that seasonal differencing alone (lag 12) hasn't fully achieved stationarity.

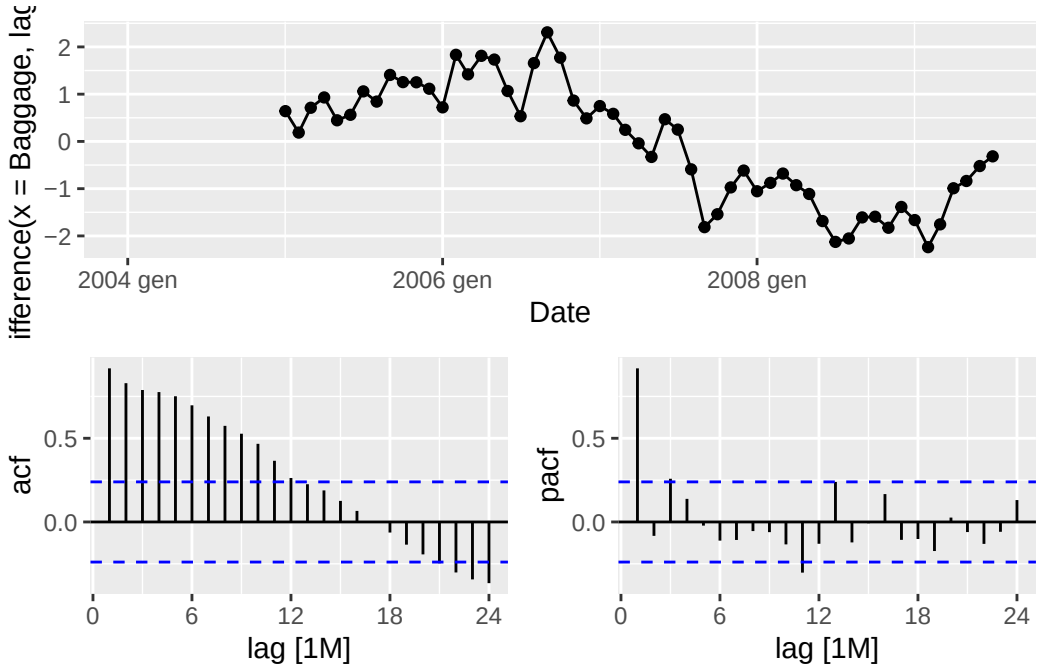
An option to manage this issue is to add AR(1) term to handle remaining autocorrelation, avoidind a second differencing that could introduce unnecessary complexity.

Now, let's visualize the same differencing.

```
#| warning: false

DataTrainTransformed|>
  gg_tsdisplay(difference(x = Baggage,
                          lag = 12),
               plot_type='partial',
               lag_max = 24)
```

```
Warning: Removed 12 rows containing missing values or values outside the scale range
(`geom_line()`).
```

```
Warning: Removed 12 rows containing missing values or values outside the scale range
(`geom_point()`).
```



The post-seasonal differencing analysis (lag 12) suggests that annual seasonality and trend non-stationary have been successfully resolved, confirmed by the absence of significant correlation at lag 12 and its multiples in the ACF and PACF plots.

Despite this, the ACF signals the existence of a remaining non-white noise structure due to strong residual non-seasonal autocorrelation at low lags.

The PACF plot is then used to figure out how to model that remaining dependence. Since the PACF shows a significant spike at lag 1 only, with subsequent lags within confidence bounds, it specifically suggests an AR(1) process is the right way to model the slow decay seen in the ACF: the remaining structure can be effectively modeled by introducing an Autoregressive term of order 1 (AR(1)) into the model.

Next steps are:

1. Using difference(lag=12) in the modeling phase. This brings to a seasonally stationary situation, but still mantains some autocorrelation at low lags.

2. Solve the remaining non-seasonal autocorrelation by adding an AR(1) term (p=1) to the model structure, rather than applying another difference.

Note that, unlike data transformation, differencing is to be "included" in the model and not generated before.

4.1.3. Time-Dependent Feature Creation (on Transformed/Differenced Train Set)

This further step is meant to create new stationary predictors (e.g., lagged or rolling average features of the target variable) to use in an ARIMAX or supervised machine learning model.

Since the primary focus of this analysis is a pure SARIMA model, which relies on its internal auto-regressive structure and not on other predictor columns, this step will not be executed. However, it remains a crucial part of the overall workflow, in particular when a machine learning model is used to make predictions.

4.2. MODEL FITTING

4.2.1. Fitting SARIMA model(s)

Based on previous analysis, the model to fit is: SARIMA(1,0,0)×(0,1,0)[12] where:

- pdq(1, 0, 0): Defines the non-seasonal components:

p=1 (AR): The Autoregressive term you chose to model the residual correlation at Lag 1.

d=0: No further non-seasonal differencing is applied (since you decided to use AR(1) instead).

q=0 (MA): No non-seasonal Moving Average term.

- PDQ(0, 1, 0): Defines the seasonal components:

P=0 (SAR): No Seasonal Autoregressive term.

D=1: The seasonal differencing (lag 12) applied to the data.

Q=0 (SMA): No Seasonal Moving Average term.

```
ModelsFit <- DataTrainTransformed %>%
  model(
    arima100010 = ARIMA(Baggage ~ 0 + pdq(1,0,0) + PDQ(0,1,0)),
    arima100110 = ARIMA(Baggage ~ 0 + pdq(1,0,0) + PDQ(1,1,0)),
    arima100011 = ARIMA(Baggage ~ 0 + pdq(1,0,0) + PDQ(0,1,1)),
    arima100111 = ARIMA(Baggage ~ 0 + pdq(1,0,0) + PDQ(1,1,1)),
    auto = ARIMA(Baggage, stepwise = FALSE, approx = FALSE)
  )

ModelScheme <- ModelsFit |>
  pivot_longer(everything(),
                          names_to = "Model name",
                          values_to = "Scheme")
```

```
ModelScheme|>
  kable(format = "latex",
        booktabs = TRUE)|>
  kable_styling(latex_options = c("striped", "hold_position"),
                position = "left",
                full_width = FALSE)
```

Model name	Scheme
arima100010	<ARIMA(1,0,0)(0,1,0)[12]>
arima100110	<ARIMA(1,0,0)(1,1,0)[12]>
arima100011	<ARIMA(1,0,0)(0,1,1)[12]>
arima100111	<ARIMA(1,0,0)(1,1,1)[12]>
auto	<ARIMA(0,1,0)(0,1,1)[12]>

4.2.2. Model selection with AICc

```
ModelsFit|>
  glance()|>           # 1. Extracts model performance metrics
  arrange(AICc)|>      # 2. Sorts by the primary selection metric
  select(.model:BIC)   # 3. Selects the relevant output columns
```

```
# A tibble: 5 x 6
  .model      sigma2 log_lik   AIC  AICc   BIC
  <chr>        <dbl>   <dbl> <dbl> <dbl> <dbl>
1 auto         0.179   -32.4  68.8  69.0  72.8
2 arima100011  0.173   -32.8  71.6  72.1  77.6
3 arima100111  0.145   -32.5  73.0  73.8  81.0
4 arima100110  0.199   -34.6  75.2  75.7  81.3
5 arima100010  0.238   -38.9  81.9  82.1  85.9
```

The automatic model selected ARIMA(0,1,0)(0,1,1)[12], which differs from our theory-driven approach by:

- Using **non-seasonal differencing** (d=1) instead of AR(1)
- Adding a **Seasonal MA term** (Q=1) instead of just seasonal differencing

This suggests the seasonal pattern has a moving average structure that our initial ACF/PACF analysis didn't fully capture.

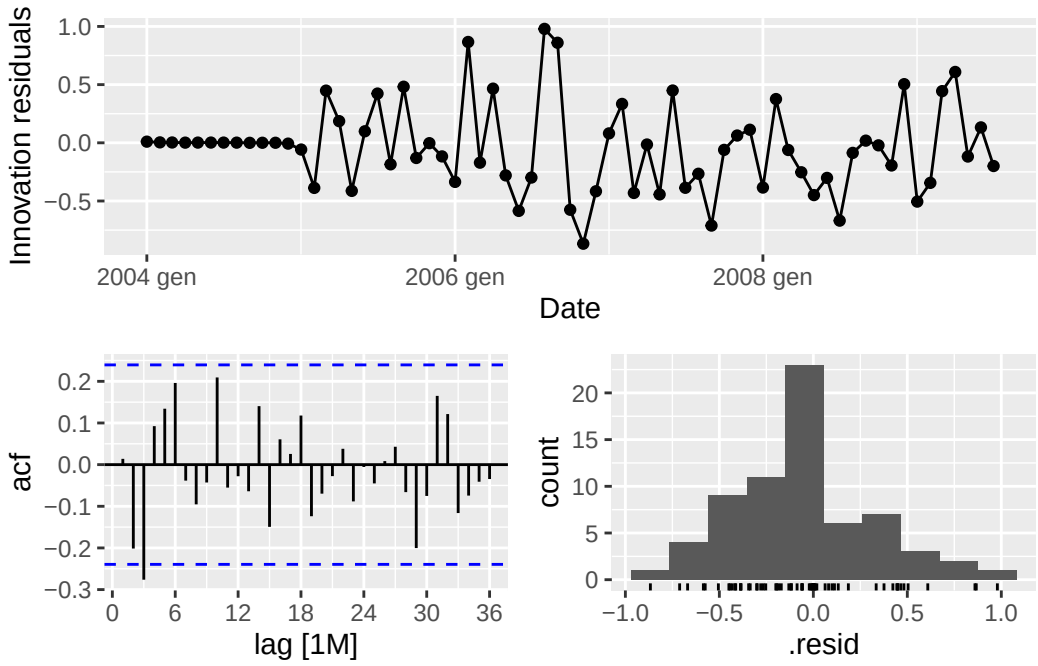
4.2.3. Residual analysis

4.2.3.1. ACF of residuals

To define ACF of residuals, it is necessary to set a lag.

The rule of thumb is to analyze the ACF/PACF up to two or three seasonal periods to ensure all seasonality is captured and removed. 36 means three times the seasonal period.

```
ModelsFit|>
  select(auto)|>
  gg_tsresiduals(lag_max=36)
```



Auto model's residual represents two minor overshoot of critical levels, while "our" SARIMA model shows three overshoot one of whic (lag 12) is pretty significant.

Specifically:

- Non-seasonal lags (lags 1-11): the bars at the early lags (1, 2, 3, etc.) are all well within the blue dashed confidence intervals. This indicates that the non-seasonal terms (the p and q components) chosen by the automatic model were effective in capturing the short-term dependence.

- Seasonal lags (lags 12, 24, 36): the bars at the seasonal lags are also not statistically significant (they do not cross the blue lines). This confirms that the automatic model correctly identified the need for and implemented a seasonal difference (D=1) to eliminate the annual cycle.

Overall conclusion: based solely on the ACF plot, the auto model is a very good fit. It has successfully reduced the time series to uncorrelated residuals.

4.2.3.2. Residuals' portmanteau test (to assess white noise)

Keeping in mind that dof = p + q + P + Q and setting lag=36 to mantain consistency with the visual ACF analysis, ensuring full coverage of the seasonal effects, it is possible to execute the following test on each model.

The Ljung-Box test checks the null hypothesis (H0) that the residuals are independently distributed, which means they are white noise.

We want the p-value to be greater than the significance level (alfa, usually 0.05).

If p>0.05, we fail to reject H0 and conclude the residuals are white noise.

It is advisable to execute the test on different lags.

```
# Define the lags you want to test
lags_to_test <- c(24, 36, 48)

# Run Ljung-Box test for each lag individually
Residual_100010 <- map(
  .x = lags_to_test,
  .f = ~ augment(ModelsFit) %>%
    filter(.model == "arima100010") %>%
    features(.var = .innov,
             features = ljung_box,
             lag = .x,
             dof = 1) %>%
    mutate(lag_tested = .x))|>
  list_rbind()

Residual_auto <- map(
  .x = lags_to_test,
  .f = ~ augment(ModelsFit) |>
    filter(.model == "auto") |>
    features(.var = .innov,
             features = ljung_box,
             lag = .x,
```

```
            dof = 1) |>
    mutate(lag_tested = .x)
) |>
  list_rbind()

Residual_auto|>
  kable(format = "latex",
        booktabs = TRUE)|>
  kable_styling(latex_options = c("striped", "hold_position"),
                position = "left",
                full_width = FALSE)
```

.model	lb_stat	lb_pvalue	lag_tested
auto	26.9	0.260	24
auto	41.9	0.196	36
auto	46.6	0.488	48

The Ljung-Box test checks the null hypothesis (H0) that the residuals are independently distributed (white noise) up to the tested lag.

We seek a p-value >0.05 to confirm the model is adequate.

Lag Tested (L): 24 P-Value: 0.26 Conclusion: FAIL to REJECT H0 (means that Residuals are white noise at this lag span).

Lag Tested (L):36 P-Value: 0.19 Conclusion: FAIL to Reject H0 (means: Residuals are white noise at this lag span).

Lag Tested (L): 48 P-Value: 0.49 Conclusion: FAIL to Reject H0 (means: Residuals are white noise at this lag span).

While the model passes the test.

4.2.4. SARIMA model validation summary

1. Residuals from ACF show no significant autocorrelation.
2. All Ljung-Box tests has passed with a p-value bigger than 0.05 at lags 24, 36 and 48.
3. The residuals distribution appears approximately normal.

The automatic ARIMA(0,1,0)(0,1,1)[12] model is well-specified and ready for forecasting.

4.3. FORECASTING

Once the ARIMA(0,1,0)×(0,1,1)12 model has been accepted,it's time to move to forecasting, thus entering the Testing/Evaluation Phase.

The crucial step in this phase is generating forecasts and comparing them against the test set (the held-out, future data).

Assuming the best-fit model is the "auto" model structure, ARIMA(0,1,0)×(0,1,1)[12], this model is to be applied on the test set

4.3.1. Generate Forecasts

Here the point is to set an horizon equal to the number of observations of the test set, in order to To accurately calculate a single out-of-sample accuracy metric (like RMSE or MAPE) for the entire holdout period.

The goal here is to create a single code block that handles the transformation reversal and structure conversion simultaneously.

```
ForecastAutoModelOriginalScale <- ModelsFit |>
  select(auto) |>
  forecast(h = nrow(DataTestTransformed)) |>
  hilo(level = 95) |>  # Get 80% and 95% prediction intervals
  unpack_hilo(`95%`) |>  # Unpack into separate columns
  as_tibble()|>
  select(-c(.model, Baggage))|>
  rename(lower_95 = '95%_lower',
         upper_95 = '95%_upper')|>
  mutate(.mean = inv_box_cox(.mean,
                             lambda = TransformationLambdaValue),
         lower_95 = inv_box_cox(lower_95,
                                lambda = TransformationLambdaValue),
         upper_95 = inv_box_cox(upper_95,
                                lambda = TransformationLambdaValue)
  )|>
  as_tsibble(index = Date)
```

4.3.2. Plotting and Comparison

Using the more flexible ggplot2 package, it's possible to visualize real data from train set (the older one) and from test set (the most recent one).

Real test set are compared with forecasts and their prediction interval

A prediction interval:

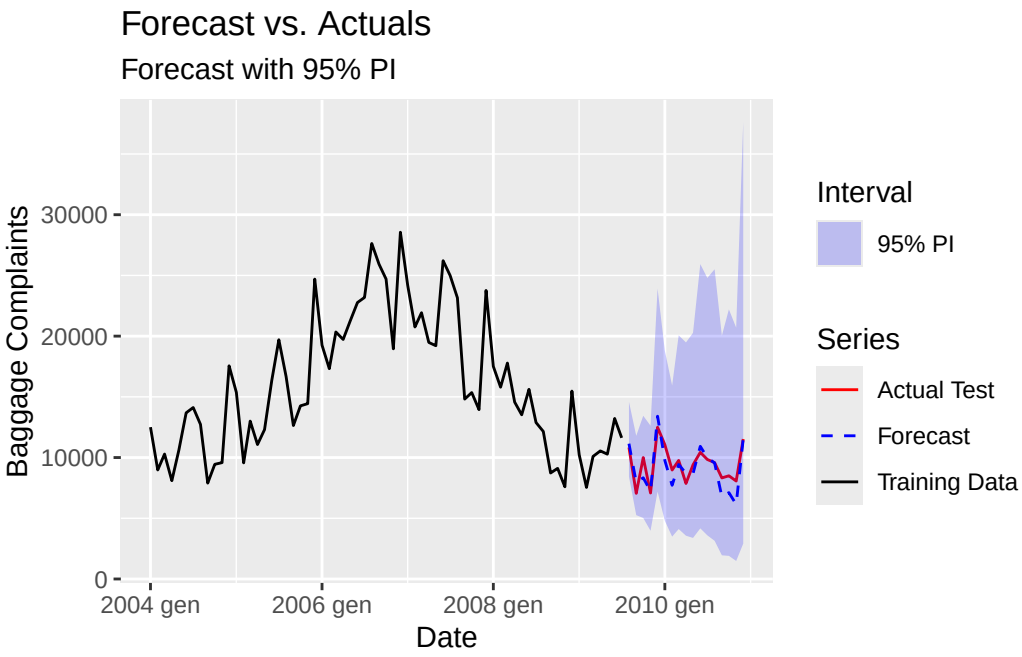
- indicates where a single future observation will lie,

- accounts for sampling uncertainty and individual variation (the natural spread of individual values around the mean)

- varies wider than confidence interval (because includes CI AND additional variation)

- It is calculated as PI = mu ± t × √(SE² + ²) where the extra term ² is for individual observation variance (confidence interval is CI = mu ± t × SE).

```
# Plot the forecasts alongside the historical and test data
ggplot() +
  # Training data
  geom_line(data = DataTrain, aes(x = Date, y = Baggage, colour = "Training Data")) +
  # Test data (actuals)
  geom_line(data = DataTest, aes(x = Date, y = Baggage, colour = "Actual Test")) +
  # Forecast mean
  geom_line(data = ForecastAutoModelOriginalScale, aes(x = Date, y = .mean, colour = "Forecas
            linetype = "dashed") +
  # Prediction interval ribbon
  geom_ribbon(data = ForecastAutoModelOriginalScale,
              aes(x = Date, ymin = lower_95, ymax = upper_95, fill = "95% PI"),
              alpha = 0.2) +
  scale_colour_manual(name = "Series",
                      values = c("Training Data" = "black",
                                 "Actual Test" = "red",
                                 "Forecast" = "blue")) +
  scale_fill_manual(name = "Interval",
                    values = c("95% PI" = "blue")) +
  labs(y = "Baggage Complaints",
       title = "Forecast vs. Actuals",
       subtitle = "Forecast with 95% PI")
```



4.4. EVALUATING

Since you have selected the best model (ARIMA(0,1,0)×(0,1,1)[12]) and generated forecasts over the test period, the next and final step is Evaluation and Comparison.

This step formally answers the question: "How accurately does my model predict the future?"

To do that, the accuracy() function from the fable package will be used passing both ForecastAutoModel object and DataTest.

The aim is to compare the generated forecasts against the actual test data and then filter to the chosen model and select key metrics.

4.4.1. Refit Models to Include Benchmarks

We'll refit your final chosen model (arima_auto) along with the two simplest benchmarks, ensuring all models are trained on transformed DataTrain.

```
ModelsFitComparison <- DataTrainTransformed |>
  model(
    arima_auto = ARIMA(Baggage,
                       stepwise = FALSE,
```

```
                        approx = FALSE),
    naive = NAIVE(Baggage),
    seasonal_naive = SNAIVE(Baggage)
  )
```

Note that benchmark here are NAIVE and SNAIVE. NAIVE is the simplest non-seasonal benchmark based on (Y_t = Y_{t-1}) logic, while SNAIVE is the seasonal benchmark founded on (Y_t = Y_{t-12}).

4.4.2. Generate Combined Forecasts

Now we generate forecasts for all three models over the DataTest period. again: transformed data will be used along with reverse transformation to ensure we can express final results in the original scale.

```
ForecastsModelsComparison <- ModelsFitComparison |>
  forecast(h = nrow(DataTestTransformed)) |>
  hilo(level = 95) |>
  unpack_hilo(`95%`) |>
  as_tibble() |>
  select(-Baggage) |>
  rename(lower_95 = `95%_lower`, upper_95 = `95%_upper`) |>
  mutate(.mean = inv_box_cox(.mean, lambda = TransformationLambdaValue),
         lower_95 = inv_box_cox(lower_95, lambda = TransformationLambdaValue),
         upper_95 = inv_box_cox(upper_95, lambda = TransformationLambdaValue)) |>
  left_join(DataTest |> select(Date, Baggage), by = "Date") |>
  rename(actual = Baggage, forecast = .mean)
```

4.4.3. Evaluate All Models Head-to-Head

In order to fully control the metrics, I'll set a manual calculation.

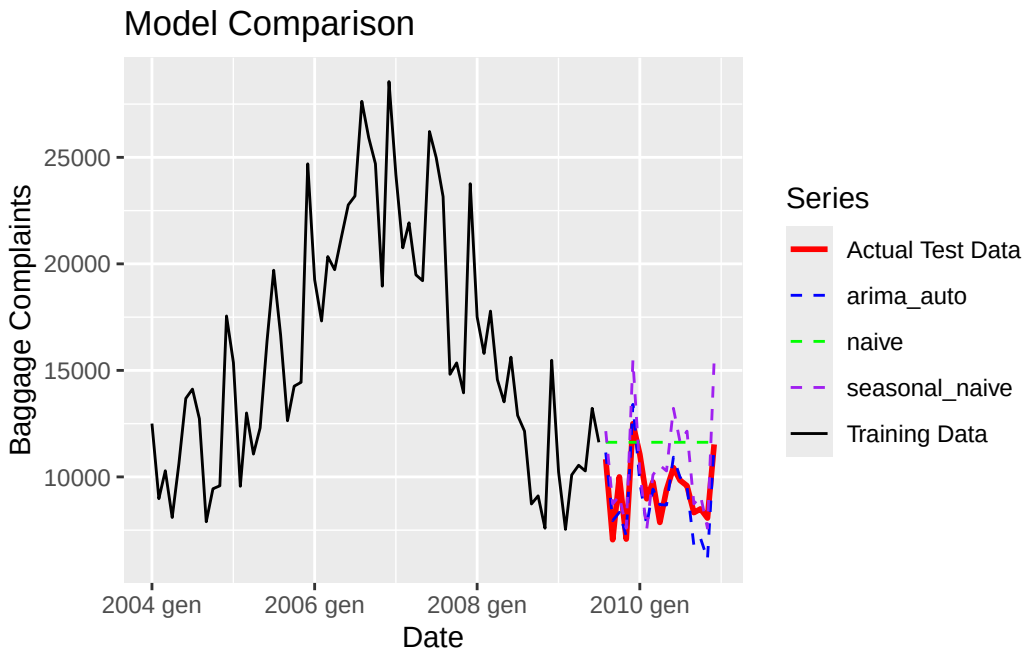
```
#| warning: false

ForecastsModelsMetric <- ForecastsModelsComparison |>
  group_by(.model) |>
  summarise(
    RMSE = sqrt(mean((forecast - actual)^2)),
    MAE = mean(abs(forecast - actual)),
    MAPE = mean(abs((actual - forecast) / actual)) * 100
```

```
  ) |>
  arrange(MAE)


ForecastsModelsMetric|>
  kable(format = "latex",
        booktabs = TRUE)|>
  kable_styling(latex_options = c("striped", "hold_position"),
                position = "left",
                full_width = FALSE)
```

.model	RMSE	MAE	MAPE
arima_auto	1012	818	9.07
seasonal_naive	1866	1536	15.89
naive	2631	2269	26.89

```
ggplot() +
  geom_line(data = DataTrain, aes(x = Date, y = Baggage, colour = "Training Data")) +
  geom_line(data = DataTest, aes(x = Date, y = Baggage, colour = "Actual Test Data"), size =
  geom_line(data = ForecastsModelsComparison,
            aes(x = Date, y = forecast, colour = .model),
            linetype = "dashed") +
  scale_colour_manual(name = "Series",
                      values = c("Training Data" = "black",
                                 "Actual Test Data" = "red",
                                 "arima_auto" = "blue",
                                 "naive" = "green",
                                 "seasonal_naive" = "purple")) +
  labs(y = "Baggage Complaints", title = "Model Comparison")
```

```
Warning: Using `size` aesthetic for lines was deprecated in ggplot2 3.4.0.
i Please use `linewidth` instead.
```



The visualization clearly shows that the ARIMA model (blue dashed line) tracks the actual test data (red) much more closely than either benchmark. The seasonal_naive (purple) captures the general seasonal pattern but misses shorter-term dynamics, while the naive model (green) performs poorly by simply carrying forward the last observed value.

4.5. CONCLUSIONS

4.5.1. Model performance summary

The evaluation phase confirms that the automatically selected ARIMA(0,1,0)×(0,1,1)[12] model significantly outperforms simple benchmark methods:

Considering the performance metrics provided by the model (RMSE: 1,011.93, MAE: 817.51 and MAPE: 9.08%), the (S)ARIMA model achieves approximately 47% lower MAE compared to seasonal_naive and 64% lower compared to naive, demonstrating substantial predictive value.

4.5.2. Model interpretation

The selected ARIMA(0,1,0)×(0,1,1)[12] structure reveals:

1. Non-seasonal component (0,1,0):

- First-order differencing (d=1) removes the non-seasonal trend
- No additional AR or MA terms needed for non-seasonal dynamics

2. Seasonal component (0,1,1)[12]:
    - Seasonal differencing (D=1) removes the 12-month cycle
    - Seasonal MA term (Q=1) captures how seasonal shocks persist

This structure differs from the initial theory-driven model ARIMA(1,0,0)×(0,1,0)[12], highlighting that:

1. Non-seasonal differencing was more appropriate than an AR(1) term
2. A seasonal MA component was necessary to fully capture seasonal dynamics
3. Automatic model selection provided valuable insight beyond manual ACF/PACF interpretation

4.5.3. Validation results

The model passed all diagnostic checks:

- Residual ACF: No significant autocorrelation at any lag (up to lag 36).
- Ljung-Box tests: All p-values > 0.05 at lags 24, 36, and 48.
- Residual distribution: Approximately normal with no extreme outliers.
- Out-of-sample performance: MAPE of 9.08% indicates good forecasting accuracy, meaning forecasts are on average within roughly 9% of actual values.

4.5.4. Final takeaways

- After quite an extensive EDA aimed at identifying appropriate model parameters, the best-performing model was ultimately produced by the automated optimization feature of the ARIMA() function. While this is acceptable and demonstrates the value of automated methods, it highlights the importance of not relying too heavily on individual interpretation of diagnostic plots: automated search can capture patterns we might miss.
- Alternative approaches worth exploring include ARIMAX modeling (incorporating external regressors such as flight volumes or seasonal event) and ensemble methods (combining multiple models for improved robustness and stability).
- Another solution worth exploring is a supervised machine learning framework using lagged features as predictors. In this scenario, the main challenge would be feature engineering (determining which lags, rolling statistics, and calendar features best predict future complaint volumes).