Fachhochschule Südwestfalen
University of Applied Sciences

South Westphalia University
Department of Engineering and Economics

Predictive Analytics / Forecasting

[redacted]

Analysis of ARAL's daily median prices in Germany

Patrick Adrian Ulbrich

[redacted]

Contents

1 Introduction

Most people own a car, so predicting gas prices to find the optimal time for buying gas is a topic that most people are interested in. The dataset by Tankerkönig is an ideal basis for forecasting gas prices as it provides exact, event based data going back to June of 2014. Additionally, besides home owning, a car and the expenses in relation to owning a car are the second biggest expense of an average household. Most people are interested in the exact price and are less interested in the theoretical uncertainty in prices. Therefore this analysis focuses on point forecasts and metrics that focus on the accuracy of the point forecast.

The aim of this work is to determine what the optimal forecasting method for gas prices is and how accurate this measure is. A natural assumption is that gas prices are highly correlated to crude oil prices. As crude oil is a publicly traded commodity, just as gas at gas stations is, the *Efficient Market Hypothesis* is a fitting hypothesis for gas prices. That would mean that all usable information is incorporated in the current price and a Naive Forecast outperforms most other forecasting methods and models.

Asking a small, non representative group of people how they decide on their gas buying behaviour, some people try to follow rules like "Weekends are always more expensive", "Monday is the cheapest day", or "Thursday is the cheapest day". Sometimes these day-based rules contradict each other, which can be interpreted as also being in line with the Efficient Market Hypothesis and a confirmation bias. As most people try to predict the best time to buy gas in some way, even a confirmation of an efficient market behaviour for day to day prices is an interesting result.

In this work the event based data by Tankerkönig is converted to a daily median gas price time series. Then an Exploratory Data Analysis (EDA) is performed to understand the dataset. Following the EDA basic forecasting methods (Mean, Naive and Drift) as well as more complex forecasting models (ETS and ARIMA) are created. Multiple evaluation metrics are calculated for each forecast. The AICc is used for training set accuracy and model selection and MAE, RMSE, MAPE, MASE and RMSSE are used for test set accuracy.

2 Dataset Creation

The data is provided by Tankerkoenig (2024). The data is separated in stations and event-based prices. For forecasting a time series is needed, therefore the event based data has to be aggregated on a specified granularity. A daily granularity is used here. First the stations that are the focus of the analysis have to be specified.

```
library(dplyr)
library(readr)
stations <- read_csv(
  Sys.getenv("STATIONS_ROOT_DIR")
)
```

```
## Rows: 15442 Columns: 9
## -- Column specification --------------------------------------------------------
## Delimiter: ","
## chr (7): uuid, name, brand, street, house_number, post_code, city
## dbl (2): latitude, longitude
##
## i Use `spec()` to retrieve the full column specification for this data.
## i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

```
betreiber <- stations %>%
  count(brand, name = "count") %>%
  arrange(desc(count))
```

Table 1: *Station Brands sorted by Count of Stations.*

brand	count
ARAL	2378
Shell	1791
ESSO	1066
TOTAL	893
AVIA	753
JET	671

ARAL is the focus of this analysis as it is the most common brand of gas stations in Germany, as shown in table 1.

```
selected_brand <- "ARAL" # ARAL is the most common gas station
focused_stations <- stations %>% filter(brand %in% selected_brand)
```

Next the prices have to be inspected to define an appropriate aggregation scheme. For this a single file is inspected.

```
root_dir <- Sys.getenv("PRICES_ROOT_DIR")
file_list <- list.files(
  path = root_dir, pattern = "*.csv", full.names = TRUE, recursive = TRUE
)
# read example file
# shorten station_uuid for better visualization
example_file <- read.csv(file_list[1]) %>%
  rename("station_uuid_short" = "station_uuid")
example_file$station_uuid_short <-
  substr(example_file$station_uuid_short, 1, 10)
example_file$date <-
  substr(example_file$date, 1, 10)

knitr::kable(
  head(example_file),
  booktabs = T,
  linesep = "",
  caption = "Example Rows of the Price Event Data."
) %>%
  kableExtra::kable_styling(latex_options = "H")
```

Table 2: *Example Rows of the Price Event Data.*

date	station_uuid_short	diesel	e5	e10	dieselchange	e5change	e10change
2014-06-08	00060029-0	1.329	1.549	1.509	1	1	1
2014-06-08	00060034-0	1.389	1.619	1.579	0	1	1
2014-06-08	00060059-0	1.369	1.569	1.529	1	1	1
2014-06-08	00060063-0	1.349	1.559	1.519	1	1	1
2014-06-08	00060146-0	1.339	1.569	1.529	1	1	1
2014-06-08	00060158-0	1.359	1.608	1.569	1	1	1

Table 2 shows the first 5 rows of the first price event file as an example. The description of the fields below is taken from Tankerkoenig (2024).

Field	meaning
date	Timestamp of Change
station_uuid	UUID of the Gas Stations from `stations`
diesel	Price Diesel
e5	Price Super E5
e10	Price Super E10
dieselchange	0=No Change, 1=Change, 2=Removed, 3=New
e5change	0=No Change, 1=Change, 2=Removed, 3=New
e10change	0=No Change, 1=Change, 2=Removed, 3=New

```
max(example_file$e10)
```

```
## [1] 1.999
```

```r
min(example_file$e10)
```

```
## [1] -0.001
```

For forecasting the data is aggregated on a daily basis. As the simple check for outliers via the max() and min() function above shows unrealistic outliers for the e10 price, probably "out of stock" default values, the median instead of the mean or mode is used to aggregate the prices. It is a reasonable assumption that most gas stations do some price changes within a day, so the median on the price events per day should result in a realistic level of the gas prices. This process makes the final price values equidistant.

The function below is used for the aggregation. The process filters the stations on the previously specified focused_stations list, so stations of *ARAL* in this case and loops over all files in the file_list. The dataset is also saved.

```r
library(data.table)
```

```
##
## Attache Paket: 'data.table'
```

```
## Das folgende Objekt ist maskiert 'package:tsibble':
##
##     key
```

```
## Die folgenden Objekte sind maskiert von 'package:lubridate':
##
##     hour, isoweek, mday, minute, month, quarter, second, wday, week,
##     yday, year
```

```
## Die folgenden Objekte sind maskiert von 'package:dplyr':
##
##     between, first, last
```

```r
process_raw_data <- function(
    calculate = 0,
    file_list,
    focused_stations,
    file_path = Sys.getenv("DAILY_DATA_FILE_PATH")) {
  # This function calculates the daily median prices for the specified
  # stations and saves the result in the file_path directory. The file
  # saved in the file_path can be read by setting calculate to 0 (default).
  # The arguments of file_list and focused_stations are only necessary
  # for calculating the daily data.

  if (calculate == 1) {
    daily_focused_prices <- tibble()

    total_time <- system.time({
      for (file in file_list) {
        # read the individual CSV files
        # separately tested: fread() is faster than read_csv() which is
        # important for the amount of data processed.
```

```
        daily_data <- fread(
          file,
          colClasses = list(character = "date")
        ) %>% # prevent unwanted implicit timezone conversions
          as_tibble()

        filtered_data <- daily_data %>%
          filter(station_uuid %in% focused_stations$uuid) %>%
          select(date, station_uuid, diesel, e5, e10) %>%
          # extract only the date portion without timestamp and timezone
          mutate(date = as.Date(substr(date, 1, 10)))

        daily_aggregation <- filtered_data %>%
          group_by(date) %>%
          summarize(
            # median used to remove outliers and out of stock prices
            median_e10_price = median(e10, na.rm = TRUE),
            median_e5_price = median(e5, na.rm = TRUE),
            median_diesel_price = median(diesel, na.rm = TRUE),
            .groups = "drop"
          )

        # combine the days data into the main data.table
        daily_focused_prices <- rbind(
          daily_focused_prices,
          daily_aggregation
        )

        message(file, " read") # tracking progress
      }
    })

    # output data to a csv
    daily_focused_ts <- daily_focused_prices %>%
      as_tsibble(index = date)
    write_csv(daily_focused_prices, file_path)

    # output time measurement for processing loop in readable format
    elapsed_time <- total_time["elapsed"]
    hours <- floor(elapsed_time / 3600)
    minutes <- floor(
      (elapsed_time %% 3600) / 60
    )
    seconds <- round(elapsed_time %% 60)
    formatted_time <- sprintf("%02d:%02d:%02d", hours, minutes, seconds)
    print(paste("Total time taken:", formatted_time))
  } else {
    # read file from previously calculated csv
    daily_focused_prices <- read_csv(file_path)
    daily_focused_ts <- daily_focused_prices %>%
      as_tsibble(index = date)
```

```
  }

  return(daily_focused_ts)
}
```

The function above is used for creating the time series below.

```
daily_focused_ts <- process_raw_data(
  calculate = 0,
  file_path = Sys.getenv("DAILY_DATA_FILE_PATH")
)
```

```
## Rows: 3726 Columns: 4
## -- Column specification --------------------------------------------------------
## Delimiter: ","
## dbl  (3): median_e10_price, median_e5_price, median_diesel_price
## date (1): date
##
## i Use ‘spec()‘ to retrieve the full column specification for this data.
## i Specify the column types or set ‘show_col_types = FALSE‘ to quiet this message.
```

3 Dataset Description and Exploratory Data Analysis

3.1 Dataset Description

The created time series contains the daily median prices for the fuel types e5, e10 and diesel. Figure 1 shows the daily median price for all three fuel types. From 2014 to 2021 the three fuel types were in the price range of 1.00 Euro to 1.60 Euros. After that period prices surged to a high of above 2.30 Euros with the special case that diesel, for a short time, was more expensive than the other two fuel types. After that, from about the beginning of 2023 up until now, prices seem to have settled on a new plateau between 1.60 Euros on the lower end and 1.90 on the upper end. In total there are 3726 observations. The time series does not exhibit clearly visible yearly seasonal patterns. The prices of the three fuel types are highly correlated. As the fuel types are highly correlated, all following analysis are based solely on the e10 price for easier understanding and easier visualization.

```
# convert the tsibble from wide to long format for autoplot
daily_long_ts <- daily_focused_ts %>%
  pivot_longer(
    cols = c(median_diesel_price, median_e5_price, median_e10_price),
    names_to = "fuel_type",
    # regex extracts everything between "median_" and "_price" as fuel_type
    names_pattern = "median_(.*)_price",
    values_to = "median_price"
  )

# visualize all three median prices in a single plot
autoplot(daily_long_ts, median_price) +
  labs(x = "Date", y = "Median Price", color = "Fuel Type") +
  theme_minimal() +
  theme(
    legend.position = "top",
    legend.title = element_blank()
  ) +
  # set specific color scheme
  scale_color_manual(
    values = c("diesel" = "blue", "e10" = "orange", "e5" = "green")
  )
```

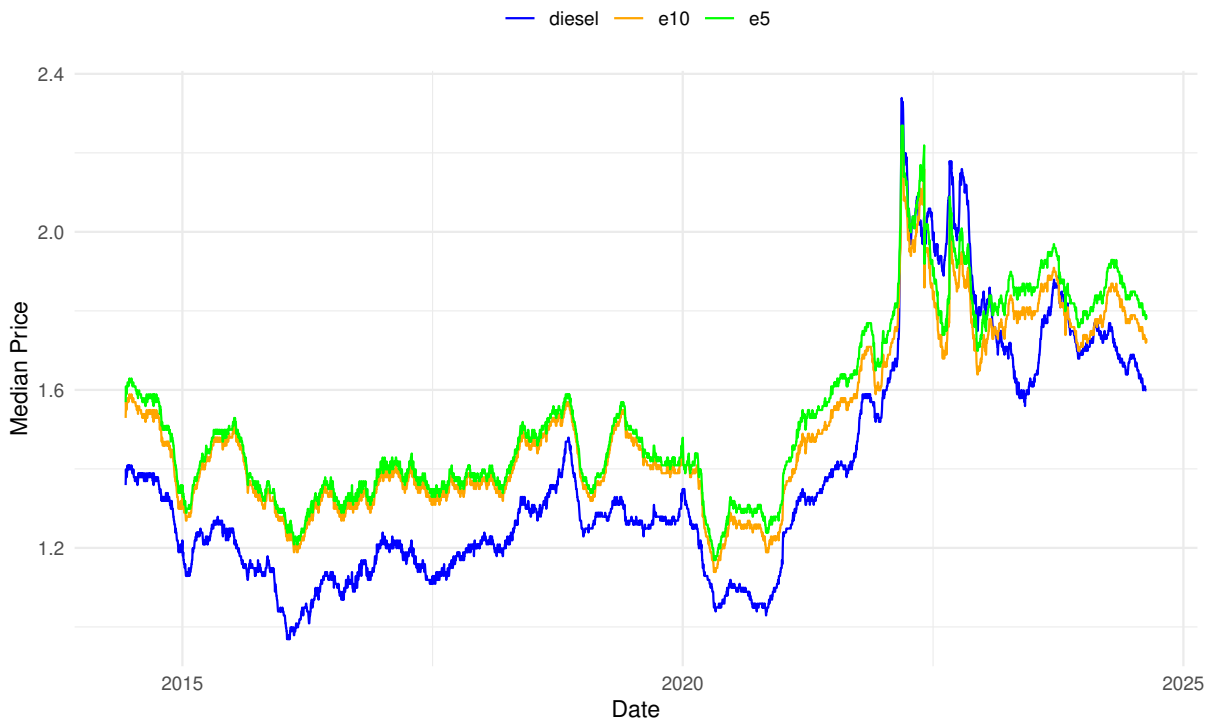

Figure 1: *Daily Median Fuel Prices.*

3.2 Visual Comparison of the Days of the Week

As mentioned in the introduction, there are multiple theories that say that some weekdays are generally cheaper or more expensive than others. This can be checked on the basis of the median prices. Figure 2 shows the Violin Plot for the median prices of e10.

```
daily_focused_ts <- daily_focused_ts %>%
  mutate(day_of_week = lubridate::wday(date, label = TRUE, abbr = FALSE))

ggplot(daily_focused_ts, aes(x = day_of_week, y = median_e10_price)) +
  geom_violin(trim = FALSE) +
  labs(
    x = "Day of the Week",
    y = "Median E10 Price"
  ) +
  theme_minimal()
```

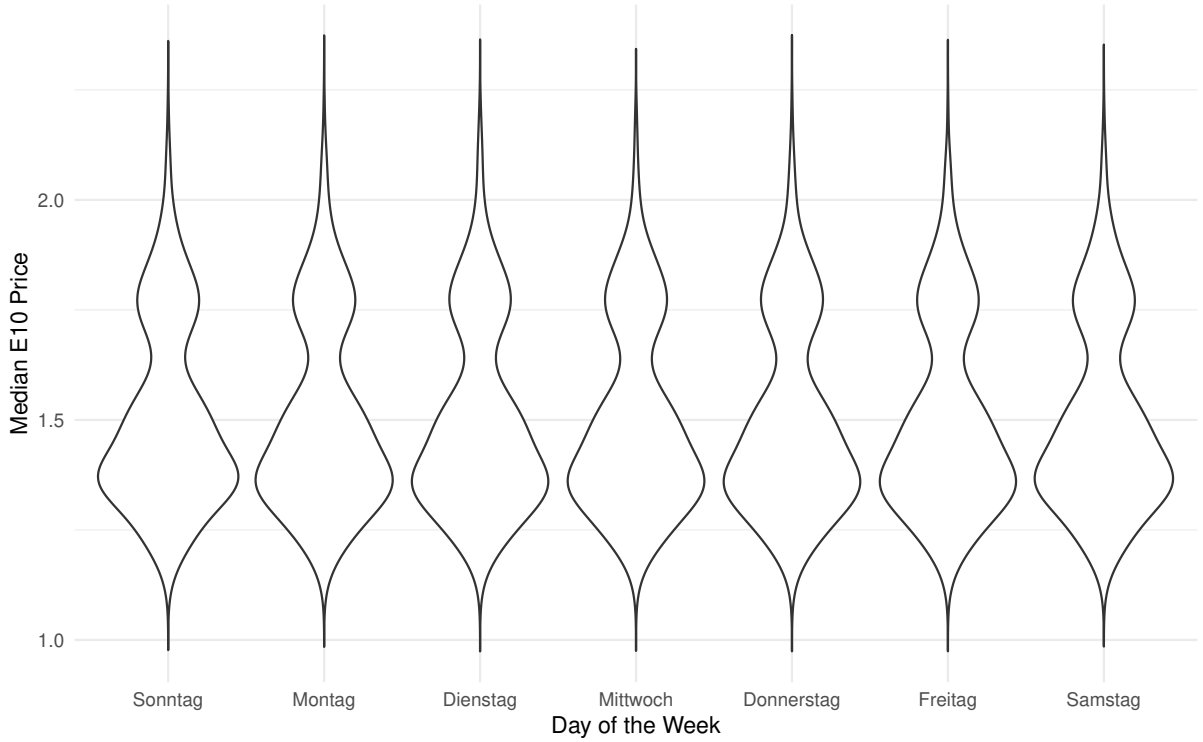

Figure 2: *Density and Distribution of Median E10 Prices by Day of the Week.*

A violin plot is comparable to a box plot, but it additionally includes visually easier to interpret information from a density plot. The width of each plot shows the frequency of the respective price. Visually the graphs for each day of the week look almost identical, which indicates that there are no meaningful differences between the prices on different week days.

3.3 ANOVA

The visual result is confirmed by a statistical test. The ANOVA (Analysis of Variance) is used to compare different groups, in this case the day of the week, to determine if there are significant differences. The null hypothesis of ANOVA is that there are no differences between the groups. A small p-value of under 0.05 would indicate that at least one of the days is significantly different from the other days (see Chambers, Freeny & Heiberger (1992) or alternatively for a quick overview Qualtrics (2024)). The aov() function in combination with the summary() function can be used to calculate the ANOVA for the days of the week in relation to the median e10 price.

```
summary(aov(median_e10_price ~ day_of_week, data = daily_focused_ts))
```

```
##                Df Sum Sq Mean Sq F value Pr(>F)
## day_of_week     6   0.01 0.00170   0.037      1
## Residuals    3719 169.52 0.04558
```

Df is the degrees of freedom, 6 in this case as there are 7 days in a week. Sum Sq is the Sum of Squares, a measure of the between-group variation. Mean Square (Mean Sq) is the Sum of Squares divided by the degrees of freedom. The F-value is the Mean Square of the days of the week divided by the Mean Square of the residuals. This value is very low with a value of 0.037. The final p-value is an indication if the null hypothesis can be rejected. The p-value shows a

value of 1 (100%). There is no statistical significant difference of the e10 price between the days of the week.

3.4 Seasonality

For focusing on the seasonal component it is necessary to make the data stationary by differencing it. This effectively removes the trend component from the upcoming plots. The unitroot_ndiffs() and unitroot_nsdiffs() functions can be used to calculate how often the dataset should be differenced or seasonally differenced (see Hyndman & Athanasopoulos (2021, Chapter 9.1) for an explanation of the functions). Based on the result below, the dataset has to be differenced one time.

```
daily_focused_ts %>% features(median_e10_price, unitroot_ndiffs)
```

```
## # A tibble: 1 x 1
##   ndiffs
##    <int>
## 1      1
```

```
daily_focused_ts %>% features(median_e10_price, unitroot_nsdiffs)
```

```
## # A tibble: 1 x 1
##   nsdiffs
##     <int>
## 1       0
```

The result can be double checked. As can be seen below, the functions now return that no additional differencing is necessary.

```
diff_series <- daily_focused_ts %>% mutate(
  diff_median_e10 = difference(median_e10_price)
)

diff_series %>% features(diff_median_e10, unitroot_ndiffs)
```

```
## # A tibble: 1 x 1
##   ndiffs
##    <int>
## 1      0
```

```
diff_series %>% features(diff_median_e10, unitroot_nsdiffs)
```

```
## # A tibble: 1 x 1
##   nsdiffs
##     <int>
## 1       0
```

The ACF (Autocorrelation Function) and PACF (Partial Autocorrelation Function) plots are used for visually inspecting seasonality. Even though there is no significant difference between the mean of the days as shown in figure 2 and the ANOVA test, the ACF on the bottom left in figure 3 shows significant autocorrelation at the weekly seasonal lags of 7, 14, 21, 28 and 35 days. The

PACF also shows the same result of significant weekly seasonal lags at 7, 14, 21, 28 and 35 days. This indicates that there is significant weekly seasonality in the data.

```
1 gg_tsdisplay(diff_series, diff_median_e10, plot_type = "partial")
```

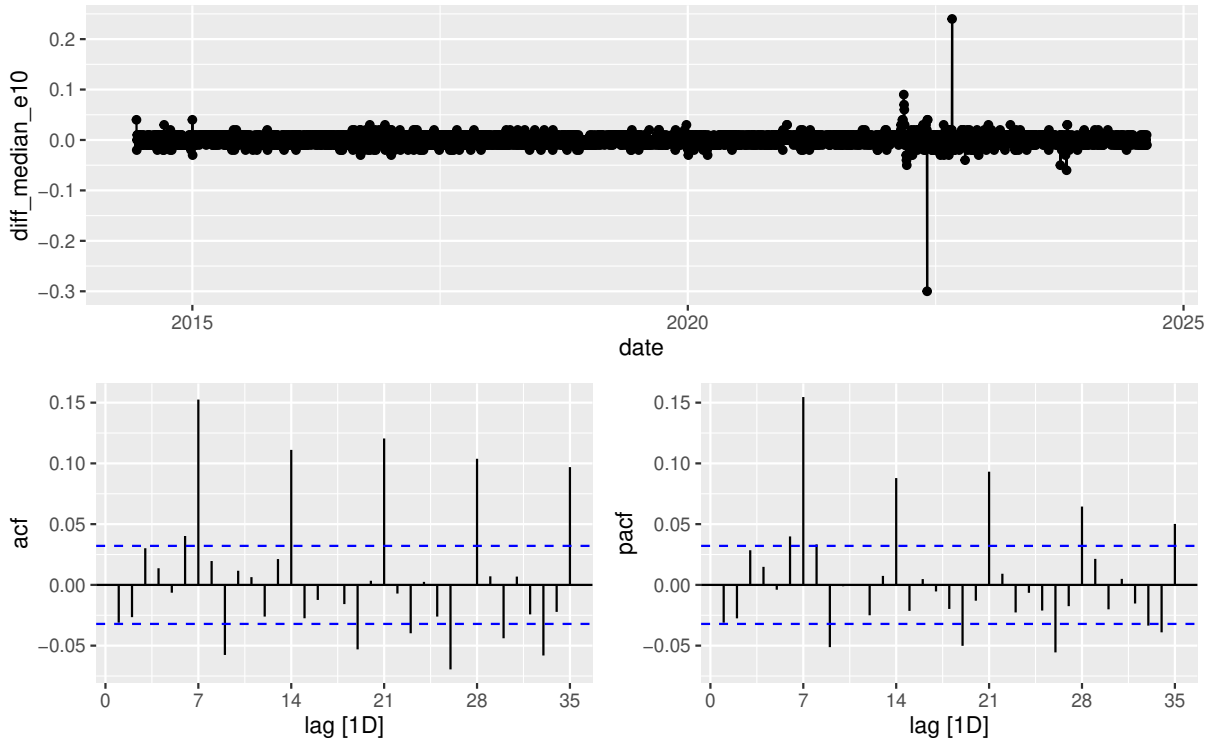

Figure 3: *Time Series, ACF, and PACF of the Median E10 Price.*

4 Forecasting

For calculating the forecast accuracy the available data has to be split in a training set and a test set. A typical rule of thumb is to use 80% of the data for training and 20% for testing. As this dataset contains 3726 observations, more of the data can be used for testing than in this rule of thumb. For this analysis 95% of the data is used for training. That still leaves 187 data points for testing.

```
split_point <- floor(nrow(daily_focused_ts) * 0.95)

training_set <- daily_focused_ts %>% slice(1:split_point)
test_set <- daily_focused_ts %>% slice((split_point + 1):n())
```

The code block below calculates all models. These models are:

- Mean Method
- Naive Method
- Drift Method
- ETS Model
- ARIMA Model

```
# calculate all models
e10_forecast_models <- training_set %>%
  model(
    mean = MEAN(median_e10_price),
    naive = NAIVE(median_e10_price),
    drift = RW(median_e10_price ~ drift()),
    ets = ETS(median_e10_price),
    arima = ARIMA(
      median_e10_price,
      stepwise = FALSE,
      approximation = FALSE,
      greedy = FALSE
    )
  )

# calculate forecasts
forecast_horizon <- nrow(test_set)
e10_forecasts <- e10_forecast_models %>% forecast(h = forecast_horizon)
```

4.1 Forecasting Method and Models

In this section the forecasting methods and forecasting models are described. The simple forecasting methods of Mean, Naive and Drift are referred to as *methods* instead of *models* as they do not contain statistical information. The forecasting methods are an algorithmic, recurring approach to forecasting. They don't inherently contain the forecast uncertainty. In comparison a statistical *model* is a stochastic process that produces the point forecast as the mean of a distribution as well as an entire forecast distribution (Hyndman & Athanasopoulos 2021, Chapter 8.5). There are also statistical models that generate the same point forecasts as exponential smoothing methods. These are referred to as state space models (Hyndman & Athanasopoulos 2021, Chapter 8.5).

4.1.1 Mean Method

The forecast via the Mean Method is the mean of the historical data, in this case the training set. Equation 1 is taken from Hyndman & Athanasopoulos (2021, Chapter 5.2).

$$\hat{y}_{T+h|T} = \bar{y} = (y_1 + \cdots + y_T)/T \tag{1}$$

4.1.2 Naive Method

There is no parameter in the Naive Method. The naive forecast is the value of the last observation. Equation 2 is taken from Hyndman & Athanasopoulos (2021, Chapter 5.2).

$$\hat{y}_{T+h|T} = y_T \tag{2}$$

4.1.3 Drift Method

The Drift Method is a variation of the Naive Method. The last observed value is taken and the average change from the first and last value of the historical data, here the training set, is added to the last observed value. Equation 3 is taken from Hyndman & Athanasopoulos (2021, Chapter 5.2).

$$\hat{y}_{T+h|T} = y_T + \frac{h}{T-1}\sum_{t=2}^{T}(y_t - y_{t-1}) = y_T + h\left(\frac{y_T - y_1}{T-1}\right) \tag{3}$$

4.1.4 ETS Model (Exponential Smoothing)

ETS Models are time series forecasting models. The ETS model stands for Error, Trend, and Seasonality. ETS models can accommodate additive, additive damped or multiplicative error, trend, and seasonal components, allowing for a flexible and robust approach to forecasting. The taxonomy of exponential smoothing methods is shown in Hyndman & Athanasopoulos (2021, Chapter 8.4) and is not be repeated here.

4.1.5 ARIMA Model (Autoregressive Integrated Moving Average)

ARIMA Models are time series forecasting models. The ARIMA model stands for Autoregression (AR), Integration (I) and Moving Average (MA). The autoregressive part forecasts the next value by using past values of the variable (Hyndman & Athanasopoulos 2021, Chapter 9.3). The moving average part forecasts the next value by using past forecasting errors (Hyndman & Athanasopoulos 2021, Chapter 9.4). The general formula in backshift notation (see Hyndman & Athanasopoulos (2021, Chapter 9.2)) was taken from Hyndman & Athanasopoulos (2021, Chapter 9.5) and is shown in equation 4. ARIMA models can be extended to incorporate seasonal information as shown in Hyndman & Athanasopoulos (2021, Chapter 9.9).

$$\underset{\substack{\uparrow\\ \text{AR}(p)}}{(1-\phi_1 B - \cdots - \phi_p B^p)} \quad \underset{\substack{\uparrow\\ d \text{ differences}}}{(1-B)^d y_t} \quad = \quad c + \underset{\substack{\uparrow\\ \text{MA}(q)}}{(1+\theta_1 B + \cdots + \theta_q B^q)\epsilon_t} \tag{4}$$

4.2 Forecasting Results

Table 4: *Created Forecasting Models.*

mean	naive	drift	ets	arima
<MEAN>	<NAIVE>	<RW w/ drift>	<ETS(M,N,A)>	<ARIMA(1,1,3)(2,0,0)[7]>

Table 4 shows the created forecasting models. The ETS() as well as the ARIMA() functions recognized the weekly seasonality automatically and incorporated it in their respective models. Figure 4 shows all forecasts. This includes the three basics models of Mean, Naive and Drift as well as the more sophisticated models of ETS and ARIMA. As multiple forecasts are plotted Only the point forecasts are visualized. Based on the visualization the Mean Forecast is not a good forecast. All other forecasts are very similar and visually almost not distinguishable. The test set is shown in grey.

```
e10_forecasts %>%
  autoplot(
    training_set,
    level = NULL # level = NULL to disable prediction intervals
  ) +
  autolayer(test_set, .vars = median_e10_price, color = "grey") +
  labs(y = "Median E10 Price") +
  guides(colour = guide_legend(title = "Forecast")) +
  theme_minimal() +
  theme(
    legend.position = "top",
    legend.title = element_blank()
  )
```

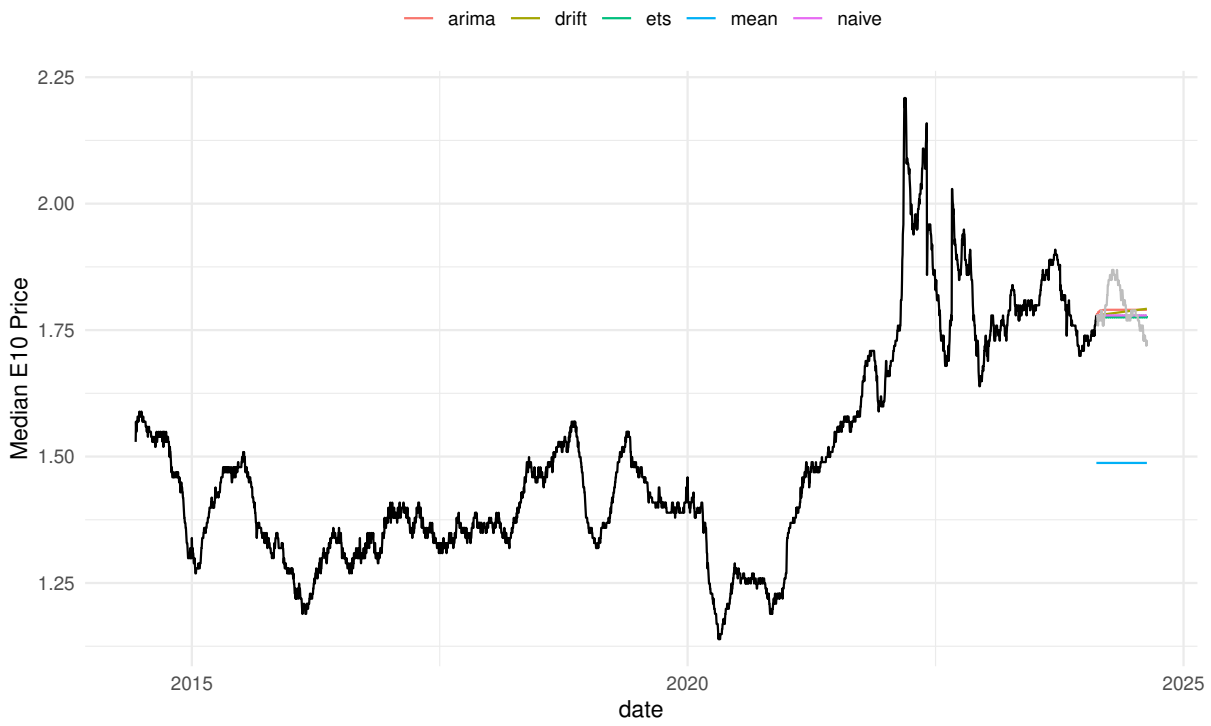

Figure 4: *Forecasts of Median E10 Prices Using All described Forecasting Methods.*

4.3 Metrics

To compare the methods in more detail specific metrics can be used. Some of the metrics are scale dependent, while others are scale independent. Scale dependent metrics cannot be used for comparison of forecasts using different units. All forecasts in this analysis are on the same time series. The metrics used here are:

- MAE
- RMSE
- MAPE
- MASE
- RMSSE
- AIC and AICc

4.3.1 MAE (Mean Absolute Error) and RMSE (Root Mean Squared Error)

The MAE (Mean Absolute Error) and the RMSE (Root Mean Squared Error) are scale dependent metrics. Minimizing the MAE leads to a median optimal forecast, while minimizing the RMSE leads to mean optimal forecasts. Equation 5 is taken from Hyndman & Athanasopoulos (2021, Chapter 5.8).

$$\begin{aligned} \text{Mean absolute error: MAE} &= \text{mean}(|e_t|) \\ \text{Root mean squared error: RMSE} &= \sqrt{\text{mean}(e_t^2)} \end{aligned} \tag{5}$$

4.3.2 MAPE (Mean Absolute Percentage Error)

The MAPE (Mean Absolute Percentage Error) can be used to compare forecasts between datasets as it is a percentage error and therefore unit-free. Percentage Errors in general can only be used if there is a meaningful zero, which is the case in gas prices. Equation 6 is taken from Hyndman & Athanasopoulos (2021, Chapter 5.8).

$$\text{Mean absolute percentage error: MAPE} = \text{mean}(|p_t|) \tag{6}$$

4.3.3 MASE (Mean Absolute Scaled Error) and RMSSE (Root Mean Squared Scaled Error)

The MASE (Mean Absolute Scaled Error) is a scaled measure and an alternative to the percentage error approach. It uses the training MAE for scaling. For a seasonal time series, the scaled error using a seasonal naive forecast is shown in equation 7 as taken from Hyndman & Athanasopoulos (2021, Chapter 5.8).

$$q_j = \frac{e_j}{\frac{1}{T-m}\sum_{t=m+1}^{T}|y_t - y_{t-m}|} \tag{7}$$

This scaled error is used similarly to the MAE and RMSE as shown in equation 8 as taken from Hyndman & Athanasopoulos (2021, Chapter 5.8).

$$\begin{aligned} \text{MASE} &= \text{mean}(|q_j|) \\ \text{RMSSE} &= \sqrt{\text{mean}(q_j^2)} \end{aligned} \tag{8}$$

4.3.4 AIC and AICc

The ETS() and ARIMA() functions used for finding the optimal model use the *AICc* metric for determining the best model. AICc measures the accuracy of a model and penalizes the amount of parameters used. It's important to note that AICc is only used on the training set for selecting the model. It contains no information about fit on the test set. The formulas in equation 9 and equation 10 are taken from Hyndman & Athanasopoulos (2021, Chapter 7.5).

$$\text{AIC} = T\log\left(\frac{\text{SSE}}{T}\right) + 2(k+2) \tag{9}$$

$$\text{AIC}_\text{c} = \text{AIC} + \frac{2(k+2)(k+3)}{T-k-3} \tag{10}$$

4.4 Model Comparison

The AICc is used for comparing models. The AICc can be accessed using the glance() function with the ETS and ARIMA models. However, as shown in table 5, the simpler methods of Mean, Naive and Drift don't contain an AICc, so their AICc has to be specifically calculated.

```
knitr::kable(
  e10_forecast_models %>% glance() %>% select(.model, AIC, AICc),
  booktabs = T,
  linesep = "",
  caption = "Result of the glance() Function on the Created Models."
) %>%
  kableExtra::kable_styling(latex_options = "H")
```

Table 5: *Result of the glance() Function on the Created Models.*

.model	AIC	AICc
mean	NA	NA
naive	NA	NA
drift	NA	NA
ets	-4032.856	-4032.793
arima	-21975.993	-21975.961

4.4.1 Calculation of AICc for simpler Models

The defined function below calculates the AIC and AICc for the simpler methods.

```
calculate_aic_aicc <- function(residuals, num_params, T) {
  sse <-
    sum(residuals^2, na.rm = TRUE) # sum of squared errors (SSE)
  aic <-
    T * log(sse / T) + 2 * (num_params + 2)
  aicc <-
    aic + (2 * (num_params + 2) * ((num_params + 3))) / (T - num_params - 3)
  return(list(AIC = aic, AICc = aicc))
}
```

This function is sued to calculate the AIC and AICc for the Mean, Naive and Drift method below.

```
# calculate residuals for mean model
mean_residuals <- e10_forecast_models %>%
  select(mean) %>%
  augment() %>%
  pull(.resid)

# calculate residuals for naive model
naive_residuals <- e10_forecast_models %>%
  select(naive) %>%
  augment() %>%
  pull(.resid)

# calculate residuals for drift model
drift_residuals <- e10_forecast_models %>%
  select(drift) %>%
```

```
  augment() %>%
  pull(.resid)

T <- nrow(training_set) # number of observations

# calculate AIC and AICc
# naive has no parameter as it is just the last observation
naive_aic_aicc <- calculate_aic_aicc(naive_residuals, num_params = 0, T)

# mean has 1 parameter: the mean, which is the forecast
mean_aic_aicc <- calculate_aic_aicc(mean_residuals, num_params = 1, T)

# drift has 2 parameters: 1 the intercept, 2 the slope of the drift
drift_aic_aicc <- calculate_aic_aicc(drift_residuals, num_params = 2, T)
```

4.4.2 Comparison of all Models

The code block below combines the AIC and AICc values for the ETS() and ARIMA() models with the manually calculated values for the simpler models of Mean, Naive and Drift. The results are shown in table 6.

```
# take AICc for ETS and ARIMA models from glance() function
model_metrics <- e10_forecast_models %>%
  select(-mean, -naive, -drift) %>%
  glance() %>%
  select(.model, AIC, AICc)

# create a tibble for calculated AIC and AICc values for simple models
simple_models_metrics <- tibble(
  .model = c("mean", "naive", "drift"),
  AIC = c(mean_aic_aicc$AIC, naive_aic_aicc$AIC, drift_aic_aicc$AIC),
  AICc = c(mean_aic_aicc$AICc, naive_aic_aicc$AICc, drift_aic_aicc$AICc)
)

# combine the two results
combined_metrics <-
  bind_rows(simple_models_metrics, model_metrics) %>% arrange(AICc)
```

Table 6: *AIC and AICc for all Created Models.*

.model	AIC	AICc
naive	-31906.218	-31906.215
drift	-31902.364	-31902.352
arima	-21975.993	-21975.961
mean	-11126.453	-11126.447
ets	-4032.856	-4032.793

Table 6 shows that the Naive Forecast has the lowest AICc of all created models. This is in line with the hypothesis from the introduction that gas prices are a publicly traded good and therefore should follow the efficient market hypothesis.

As described in the previous AICc section, the AICc is used for chosing a model, but it contains no information test set accuracy. The test metrics are shown in table 7.

```
knitr::kable(
  e10_forecasts %>%
    accuracy(daily_focused_ts) %>%
    select(.model, .type, MAE, RMSE, MAPE, MASE, RMSSE) %>%
    arrange(RMSE),
  booktabs = T,
  linesep = "",
  caption = "Test Metrics for all Created Models."
) %>%
  kableExtra::kable_styling(latex_options = "H")
```

Table 7: *Test Metrics for all Created Models.*

.model	.type	MAE	RMSE	MAPE	MASE	RMSSE
arima	Test	0.0315061	0.0395443	1.741526	1.849725	1.382341
drift	Test	0.0327301	0.0424467	1.805758	1.921587	1.483799
naive	Test	0.0316578	0.0428278	1.739260	1.858630	1.497124
ets	Test	0.0325672	0.0437617	1.788337	1.912023	1.529767
mean	Test	0.3087427	0.3112289	17.150185	18.126317	10.879560

Comparing the values shown in table 7 shows no model that is clearly better than the other models. Besides the Mean Method, that is clearly the worst forecast, all other forecasts are very close together. This is in line with figure 4. On MAPE the simplest Naive Method even is the best forecast.

5 Conclusion

The initial efficient market hypothesis for gas prices is confirmed. Gas prices are publicly traded, therefore they follow the efficient market hypothesis and the Naive Forecast is the best fit by terms of training set accuracy as determined by AICc and even in terms of one of the test set metrics. In terms of test set accuracy some of the more sophisticated models have better values in some metrics, but for interpretability and all practical purposes for most end consumers it is best to assume that there is no value in "waiting for day x of the week" to get gas.

Technical Appendix

```
Sys.time()
```

[redacted]

```
sessionInfo()
```

```
## R version 4.4.0 (2024-04-24 ucrt)
## Platform: x86_64-w64-mingw32/x64
## Running under: Windows [redacted]
##
## Matrix products: default
##
##
## locale:
## [1] LC_COLLATE=German_Germany.utf8  LC_CTYPE=German_Germany.utf8
## [3] LC_MONETARY=German_Germany.utf8 LC_NUMERIC=C
## [5] LC_TIME=German_Germany.utf8
##
## time zone: Europe/Berlin
## tzcode source: internal
##
## attached base packages:
## [1] stats     graphics  grDevices utils     datasets  methods   base
##
## other attached packages:
##  [1] data.table_1.15.4 readr_2.1.5       fable_0.3.4       feasts_0.3.2
##  [5] fabletools_0.4.2  tsibbledata_0.4.1 tsibble_1.1.4     ggplot2_3.5.1
##  [9] lubridate_1.9.3   tidyr_1.3.1       dplyr_1.1.4       tibble_3.2.1
## [13] fpp3_0.5          fhswf_0.0.7
##
## loaded via a namespace (and not attached):
##  [1] rappdirs_0.3.3       utf8_1.2.4           generics_0.1.3
##  [4] anytime_0.3.9        xml2_1.3.6           lattice_0.22-6
##  [7] stringi_1.8.4        hms_1.1.3            digest_0.6.35
## [10] magrittr_2.0.3       evaluate_0.24.0      grid_4.4.0
## [13] timechange_0.3.0     bookdown_0.39        fastmap_1.2.0
## [16] purrr_1.0.2          fansi_1.0.6          viridisLite_0.4.2
## [19] scales_1.3.0         cli_3.6.2            rlang_1.1.4
## [22] crayon_1.5.2         bit64_4.0.5          ellipsis_0.3.2
## [25] munsell_0.5.1        withr_3.0.0          yaml_2.3.8
## [28] parallel_4.4.0       tools_4.4.0          tzdb_0.4.0
## [31] colorspace_2.1-0     kableExtra_1.4.0     vctrs_0.6.5
## [34] R6_2.5.1             lifecycle_1.0.4      stringr_1.5.1
## [37] bit_4.0.5            vroom_1.6.5          urca_1.3-4
## [40] pkgconfig_2.0.3      progressr_0.14.0     pillar_1.9.0
## [43] gtable_0.3.5         glue_1.7.0           Rcpp_1.0.12
## [46] systemfonts_1.1.0    highr_0.11           xfun_0.44
## [49] tidyselect_1.2.1     rstudioapi_0.16.0    knitr_1.47
```

```
## [52] farver_2.1.2         nlme_3.1-164         htmltools_0.5.8.1
## [55] labeling_0.4.3       svglite_2.1.3        rmarkdown_2.27
## [58] compiler_4.4.0       distributional_0.4.0
```

References

Chambers, JM, AE Freeny & RM Heiberger (1992). "Analysis of Variance; Designed Experiments". In: *Statistical Models in S*. Wadsworth & Brooks/Cole. Chap. 5.

Hyndman, RJ & G Athanasopoulos (2021). *Forecasting: Principles and Practice*. Accessed: 2024-05-23. https://otexts.com/fpp3/.

Qualtrics (2024). *Varianzanalyse: Formen und Beispiele der ANOVA*. Accessed: 2024-09-05. https://www.qualtrics.com/de/erlebnismanagement/marktforschung/varianzanalyse/.

Tankerkoenig (2024). *Tankerkoenig Data Repository*. Accessed: 2024-09-05. https://dev.azure.com/tankerkoenig/_git/tankerkoenig-data.