R for Finance Report Examples

AS

Libraries

```
library(tidyverse)
```

```
Warning: package 'ggplot2' was built under R version 4.3.3

Warning: package 'purrr' was built under R version 4.3.3

Warning: package 'lubridate' was built under R version 4.3.3

-- Attaching core tidyverse packages ------------------------ tidyverse 2.0.0 --
v dplyr     1.1.4     v readr     2.1.5
v forcats   1.0.0     v stringr   1.5.1
v ggplot2   3.5.2     v tibble    3.2.1
v lubridate 1.9.4     v tidyr     1.3.1
v purrr     1.0.4
-- Conflicts ------------------------------------------ tidyverse_conflicts() --
x dplyr::filter() masks stats::filter()
x dplyr::lag()    masks stats::lag()
i Use the conflicted package (<http://conflicted.r-lib.org/>) to force all conflicts to becor
```

```
library(santimentR)
library(forecast)
```

```
Warning: package 'forecast' was built under R version 4.3.3

Registered S3 method overwritten by 'quantmod':
  method            from
  as.zoo.data.frame zoo
```

API-Key Management

The "Golden Rule" of API keys is: **Never hardcode credentials directly into your script.**

If you type api_key <- "abc12345" in your code, you risk accidentally committing it to GitHub or sharing it with a colleague, which compromises your security.

The easiest solution is to add it to `.Renviron`, which works the same way as `.env` file in Python. You store variables in a hidden text file that R loads automatically when it starts.

1. Setup your key
The easiest way to edit this file is using the `usethis` package:

```
# install.packages("usethis")
usethis::edit_r_environ()
```

```
[ ] Edit '/Users/semenoffalex/.Renviron'.
```

```
[ ] Restart R for changes to take effect.
```

This opens a file called `.Renviron`. Add your key there on a new line:

```
MY_API_KEY="12345-secret-key"
```

2. Restart R
Important: You must restart your R session for these changes to take effect (`Session -> Restart R`).

3. Use the key in your code
Now, you can access the key using `Sys.getenv()`:

```
my_key <- Sys.getenv("MY_API_KEY")

# Check if it loaded correctly
if (my_key == "") stop("API Key not found! Check .Renviron")
```

4. Safety Check
Ensure your `.gitignore` file includes `.Renviron` so git does not track it.

Getting crypto assets data via Santiment API

Initialize Santiment client

Get Data via SantimentR

We will fetch daily USD price data for four major coins: Bitcoin, Ethereum, Cardano, and Solana.

```
slugs <- c("bitcoin", "ethereum", "cardano", "solana")
from_date <- "2025-01-01T00:00:00Z"
to_date <- "2026-01-15T00:00:00Z"
interval <- "1d"
```

Fetching metrics for all slugs at once.

Sometimes it doesn't work (LoL) for long intervals ~1 year.

```
raw_data <- map_df(slugs, function(s) {
  get_metrics(client, metric = "price_usd", slug = s,
              from_date = from_date, to_date = to_date,
              interval = interval) %>%
    mutate(slug = s)
})
```

Alternative way: getting time series for each slug separately

```
btc <- get_metric(client, metric = 'price_usd',
                  slug = "bitcoin", from_date = from_date,
                  to_date = to_date,
                  interval = interval)

eth <- get_metric(client, metric = 'price_usd',
                  slug = "ethereum", from_date = from_date,
                  to_date = to_date,
                  interval = interval)

ada <- get_metric(client, metric = 'price_usd',
```

```
                    slug = "cardano", from_date = from_date,
                    to_date = to_date,
                    interval = interval)

sol <- get_metric(client, metric = 'price_usd',
                    slug = "solana", from_date = from_date,
                    to_date = to_date,
                    interval = interval)

# Combine 4 dataframes into 1 dataframe
raw_data <- bind_rows("btc"=btc,
                        "eth"=eth,
                        "ada"=ada,
                        "sol"=sol,
                        .id = "slug")
```

Another way to have some overview of the data

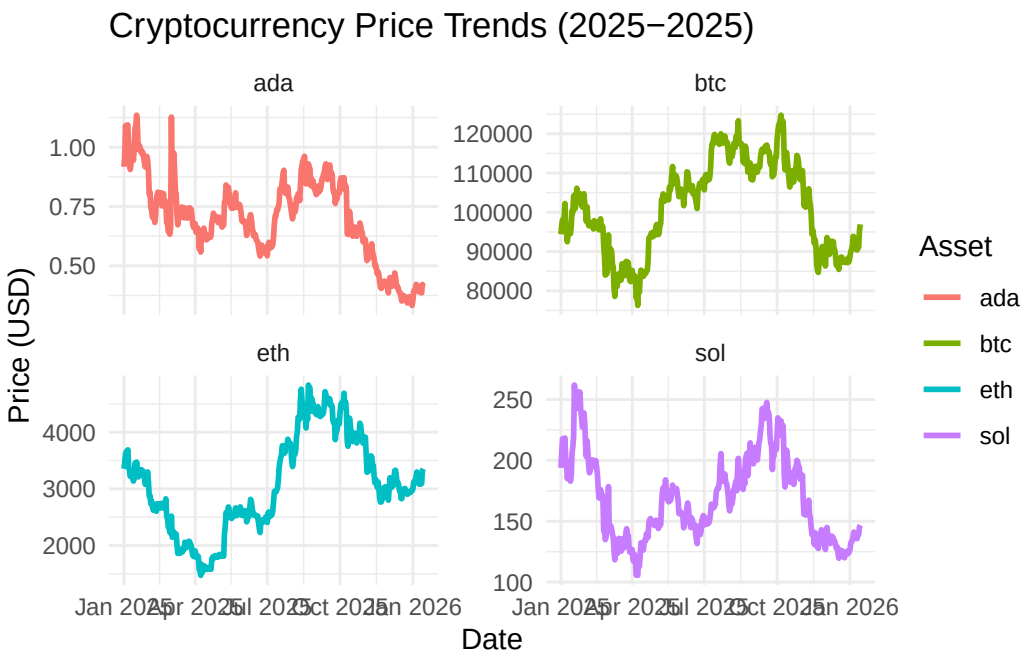
```
glimpse(raw_data)
```

```
Rows: 1,516
Columns: 3
$ slug     <chr> "btc", "btc", "btc", "btc", "btc", "btc", "btc", "btc", "btc"~
$ datetime <dttm> 2025-01-01, 2025-01-02, 2025-01-03, 2025-01-04, 2025-01-05, ~
$ value    <dbl> 94416.29, 96876.24, 98101.70, 98236.74, 98319.67, 102255.22, ~
```

Visualization

Separate graphs for the prices

```
ggplot(raw_data, aes(x = datetime, y = value, color = slug)) +
  geom_line(linewidth = 1) +
  facet_wrap(~ slug, scales = "free_y") + # Facet to see individual scales
  theme_minimal() +
  labs(title = "Cryptocurrency Price Trends (2025-2025)",
       x = "Date", y = "Price (USD)", color = "Asset")
```



Pivot data from long to wide format for convenience

```
wide_data <- raw_data %>%
  pivot_wider(names_from = slug, values_from = value) %>%
  drop_na()
```

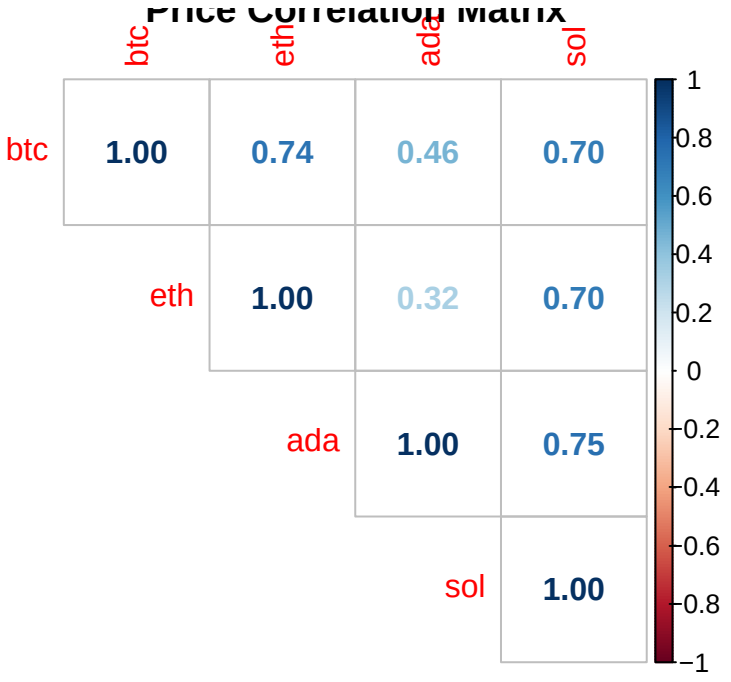
Making correlation matrix the tidy way

```
wide_data %>%
  select(-datetime) %>%
  cor() -> cor_matrix

cor_matrix
```

```
          btc       eth       ada       sol
btc 1.0000000 0.7385131 0.4638825 0.6955089
eth 0.7385131 1.0000000 0.3219278 0.7033782
ada 0.4638825 0.3219278 1.0000000 0.7541814
sol 0.6955089 0.7033782 0.7541814 1.0000000
```

Simple correlation visualization from correlation matrix

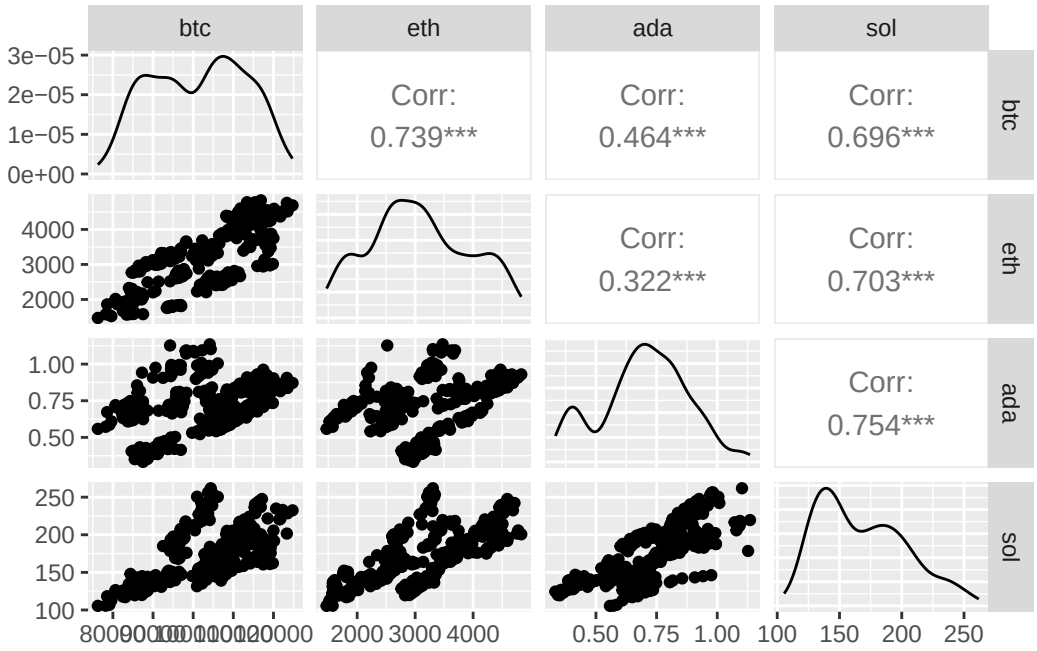
```
corrplot::corrplot(cor_matrix, method = 'number', type = 'upper',
                   title = "Price Correlation Matrix")
```



Fancy correlation visualization from the data in wide format

```
GGally::ggpairs(
  data = wide_data,
  columns = 2:5
) +
  tidyquant::scale_fill_tq() +
  tidyquant::scale_color_tq()
```

```
Registered S3 method overwritten by 'GGally':
  method from
  +.gg   ggplot2
```



Forecasting

Let's use BTC as an example

Getting the data

We did it already in this notebook, but just for a reminder:

```
btc <- get_metrics(client, metric = 'price_usd',
                   slug = "bitcoin",
                   from_date = from_date,
                   to_date = to_date,
                   interval = interval)
```

Create target variable

```
btc_data <- btc %>%
  arrange(datetime) %>%
  mutate(next_day_price = lead(price_usd)) %>% # Our target variable
  drop_na()
```

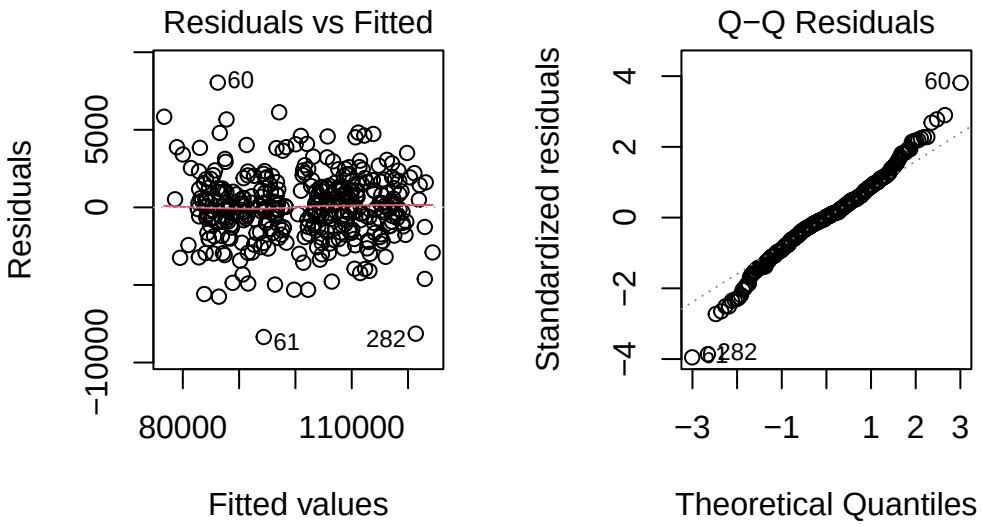
Linear regression model

```
lm_model <- lm(next_day_price ~ price_usd, data = btc_data)
summary(lm_model)
```

```
Call:
lm(formula = next_day_price ~ price_usd, data = btc_data)

Residuals:
    Min      1Q  Median      3Q     Max
-8355.3 -1133.8    45.9  1137.1  8042.4

Coefficients:
             Estimate Std. Error t value Pr(>|t|)
(Intercept) 1.703e+03  9.503e+02   1.792    0.074 .
price_usd   9.833e-01  9.320e-03 105.501   <2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 2118 on 376 degrees of freedom
Multiple R-squared:  0.9673,    Adjusted R-squared:  0.9672
F-statistic: 1.113e+04 on 1 and 376 DF,  p-value: < 2.2e-16
```

Plotting some model quality graphs

```
par(mfrow = c(1, 2))
plot(lm_model, which = 1:2)
```



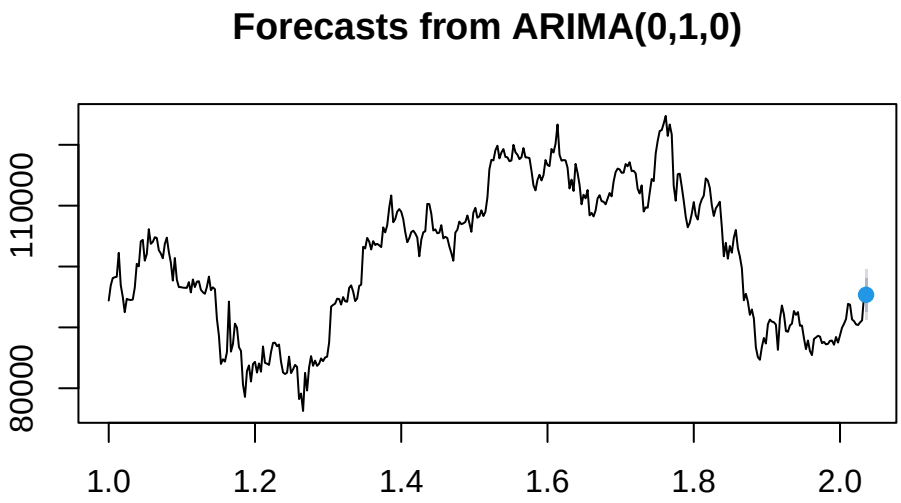
ARIMA model with Auto.ARIMA

```
btc_ts <- ts(btc_data$price_usd, frequency = 365)
arima_model <- auto.arima(btc_ts)
summary(arima_model)
```

```
Series: btc_ts
ARIMA(0,1,0)

sigma^2 = 4505747:  log likelihood = -3422.92
AIC=6847.84   AICc=6847.85   BIC=6851.78

Training set error measures:
                    ME     RMSE      MAE         MPE     MAPE      MASE
Training set 2.730626 2119.865 1546.494 -0.02077464 1.558181 0.3154826
                     ACF1
Training set -0.05463296
```

Plotting forcast from ARIMA model

```
arima_forecast <- forecast(arima_model, h = 1)
plot(arima_forecast)
```



Comparing forecast accuracy

```
lm_preds <- predict(lm_model, btc_data)
arima_preds <- fitted(arima_model)

rmse_lm <- sqrt(mean((btc_data$next_day_price - lm_preds)^2))
rmse_arima <- sqrt(mean((btc_data$price_usd - arima_preds)^2))

performance_comparison <- tibble(
  Model = c("Linear Regression", "ARIMA"),
  RMSE = c(rmse_lm, rmse_arima)
)
print(performance_comparison)
```

```
# A tibble: 2 x 2
```

```
  Model              RMSE
  <chr>             <dbl>
1 Linear Regression 2112.
2 ARIMA             2120.
```

Conclusion

This notebook successfully integrated Santiment data with Tidyverse tools to analyze and predict cryptocurrency movements via linear regression and ARIMA.

Key Considerations from the Sources:

- **Data Structure**: The code prioritizes the **long format** for visualization with `ggplot2` and the **wide format** for calculating correlations.
- **Modeling Assumptions**: When reviewing the Linear Regression results, pay attention to the **Residuals vs Fitted** plot to check for homoscedasticity and the **Normal Q-Q** plot for normality of residuals.
- **Tooling**: The use of `pivot_longer()` and `pivot_wider()` ensures the data is "tidy" before it enters the analytical pipeline.