Chapter 11

Bohan Zhu

2024-12-10

```
tourism <- read.csv("tourism_data.csv")
library(tidyverse)
```

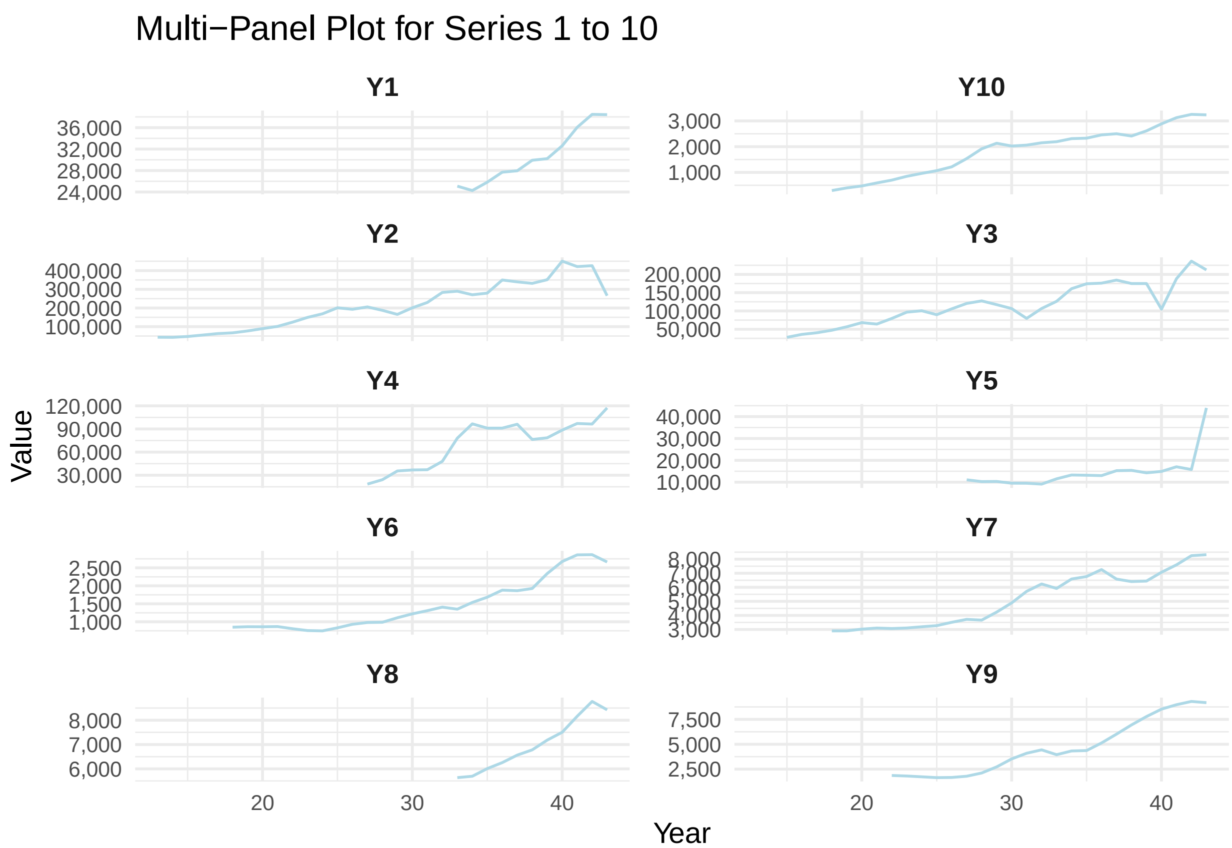
```
## -- Attaching core tidyverse packages ------------------------ tidyverse 2.0.0 --
## v dplyr     1.1.4     v readr     2.1.5
## v forcats   1.0.0     v stringr   1.5.1
## v ggplot2   3.5.1     v tibble    3.2.1
## v lubridate 1.9.3     v tidyr     1.3.1
## v purrr     1.0.2
## -- Conflicts ------------------------------------------ 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 become error
```

```
library(tidyr)
library(dplyr)

tourism_new <- tourism |>
  #pivot_longer: convert data into long formating
  pivot_longer(
    cols = everything(),
    names_to = "Series_Name",# third column
    values_to = "Value" # first column
  ) |>
  group_by(Series_Name) |>
  mutate(Time = row_number()) |> #second column
  ungroup() |>
  drop_na(Value) #drop all missing values

tourism_new <- tourism_new |>
  arrange(as.numeric(gsub("\\D", "", Series_Name)), Time)
```

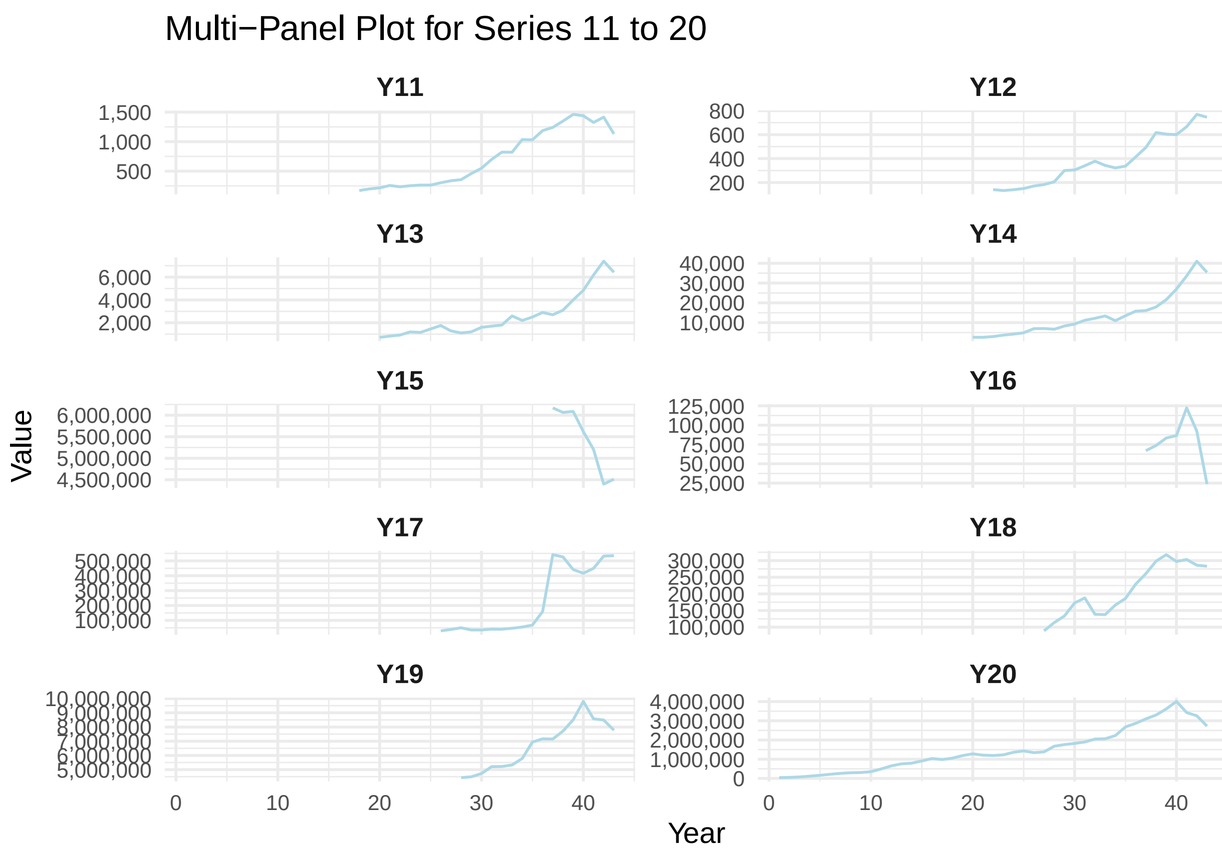
Step 1

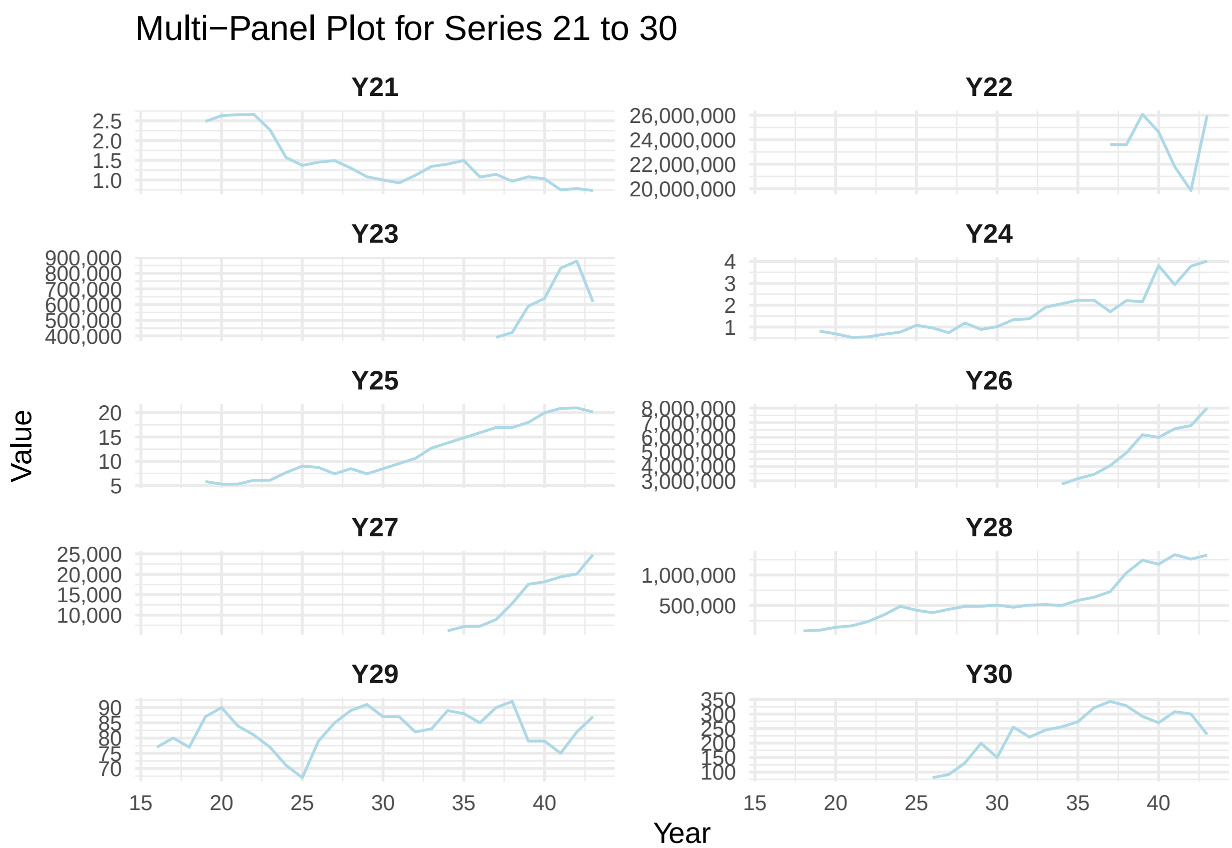
```
library(ggplot2)
library(dplyr)

# Multi-panel plot function for 10 series per plot
multi_panel_plot <- function(data, series_range) {
```

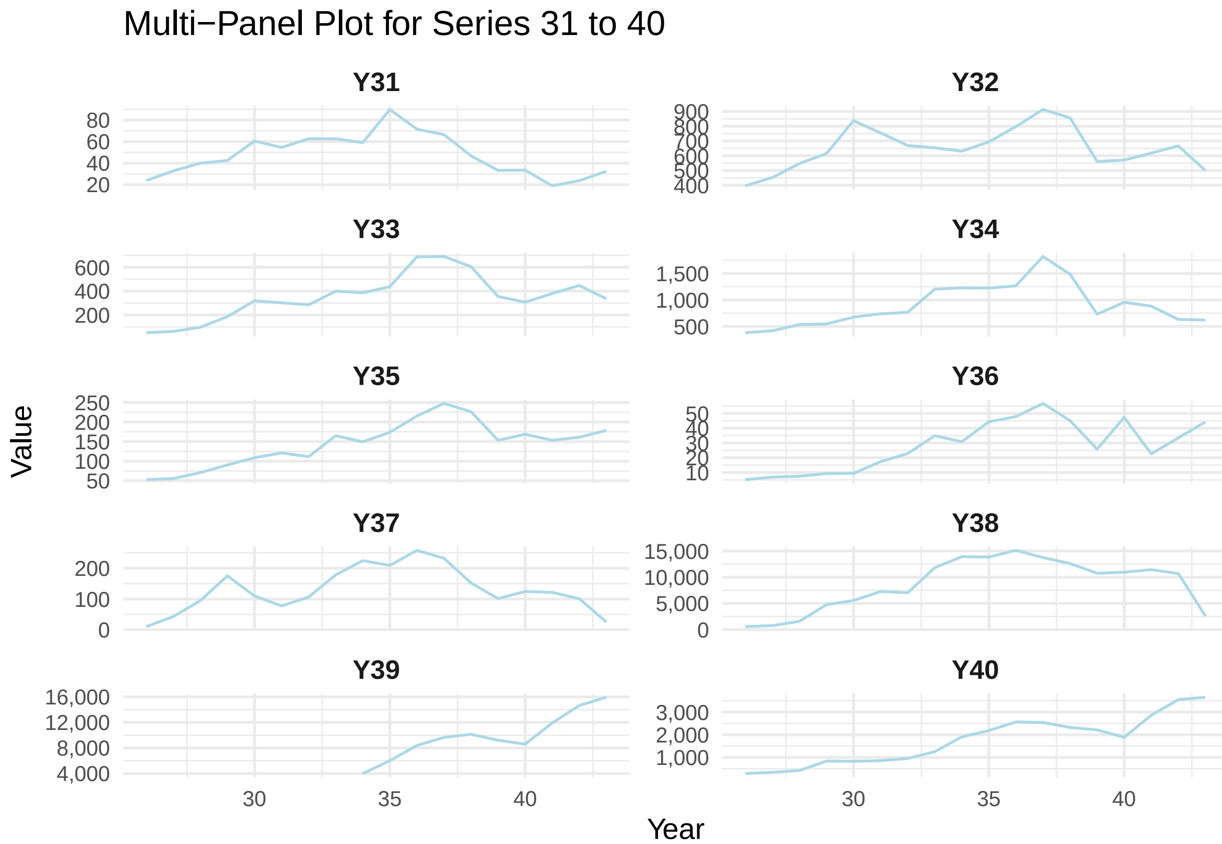
```r
  # Tip 1: Filtering data for the selected series range
  filtered_data <- data |>
    filter(Series_Name %in% unique(Series_Name)[series_range])

  ggplot(filtered_data, aes(x = Time, y = Value)) +
    geom_line(color = "lightblue") +
    # Tip 2: Use facet_wrap to create a multi-panel plot for each series
    facet_wrap(~Series_Name, ncol = 2, scales = "free_y") +   # Arrange plots in 1 column per row
    labs(
      title = paste("Multi-Panel Plot for Series", min(series_range), "to", max(series_range)),
      x = "Year",
      y = "Value"
    ) +
    theme_minimal() +
     # Tip 3: Adjust axis label sizes for better readability
    theme(
      axis.text.x = element_text(size = 8, face = "plain"),
      axis.text.y = element_text(size = 8, face = "plain"),
      strip.text = element_text(size = 10, face = "bold")
    ) +
    scale_y_continuous(labels = scales::comma)
}

# Plot all series from Y1 to Y256 in chunks of 10
for (i in seq(1, 256, by = 10)) {
  series_range <- i:min(i + 9, 256)
  print(multi_panel_plot(tourism_new, series_range))
}
```

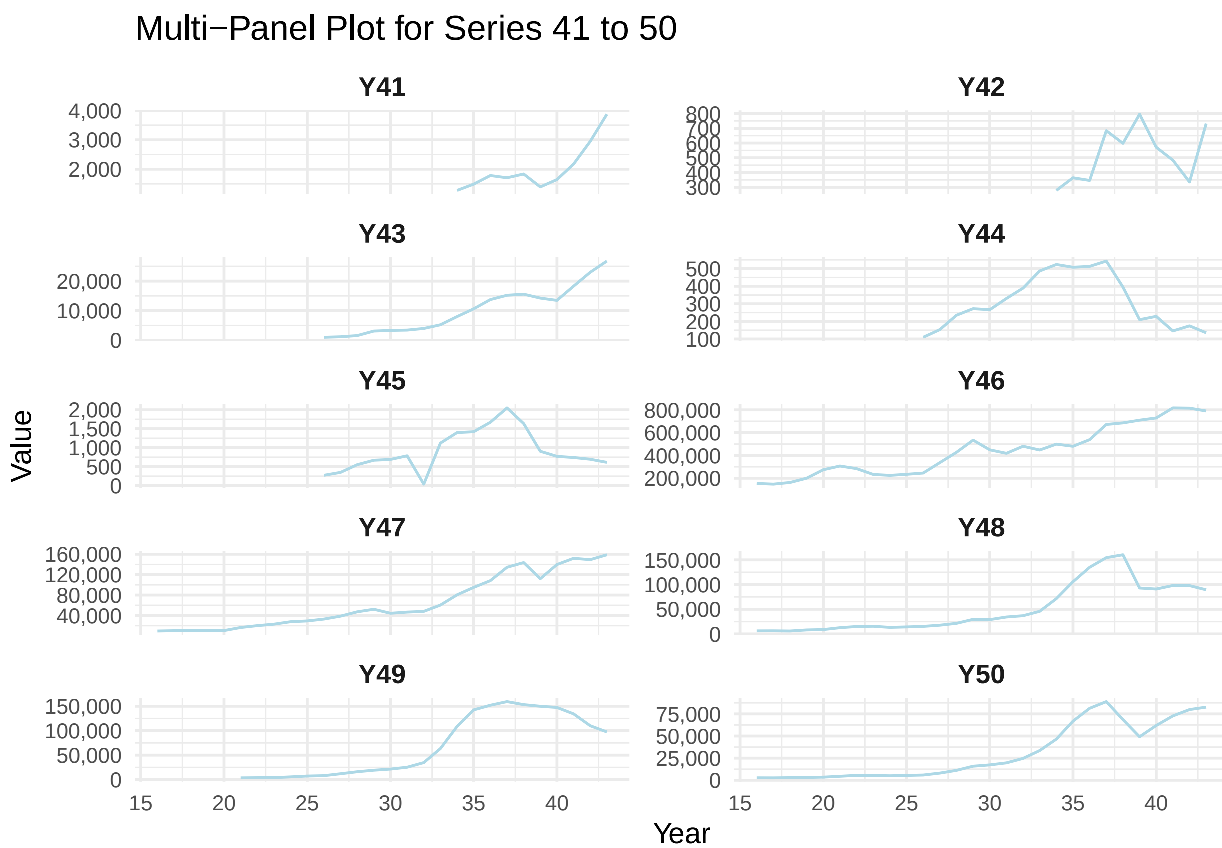


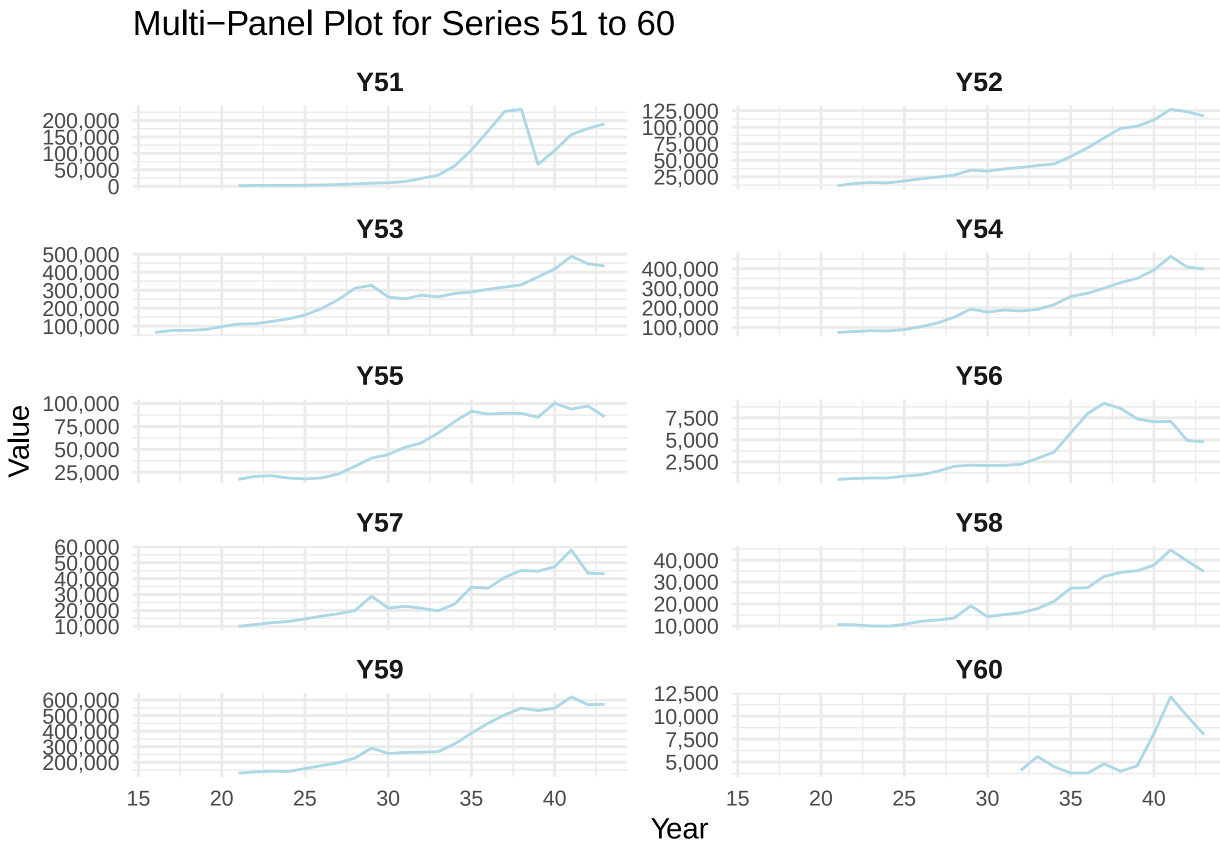
Multi−Panel Plot for Series 1 to 10
Y1
Y10
Y2
Y3
Y4
Y5
Y6
Y7
Y8
Y9
36,000
32,000
28,000
24,000
3,000
2,000
1,000
400,000
300,000
200,000
100,000
200,000
150,000
100,000
50,000
120,000
90,000
60,000
30,000
40,000
30,000
20,000
10,000
2,500
2,000
1,500
1,000
8,000
7,000
6,000
5,000
4,000
3,000
8,000
7,000
6,000
7,500
5,000
2,500
20
30
40
20
30
40
Value
Year

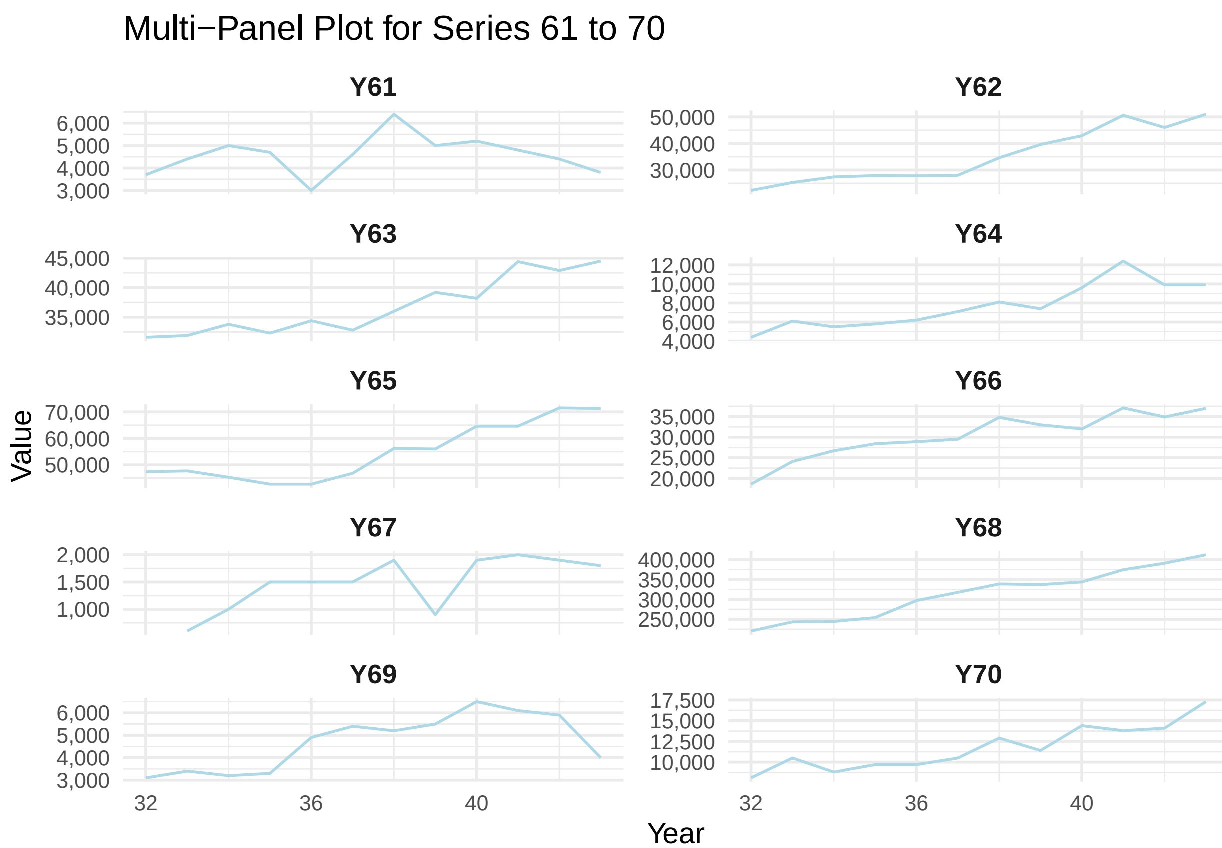

Multi−Panel Plot for Series 11 to 20
Y11
Y12
Y13
Y14
Y15
Y16
Y17
Y18
Y19
Y20
Value
Year


Multi−Panel Plot for Series 21 to 30
Y21
Y22
Y23
Y24
Y25
Y26
Y27
Y28
Y29
Y30
Value
Year

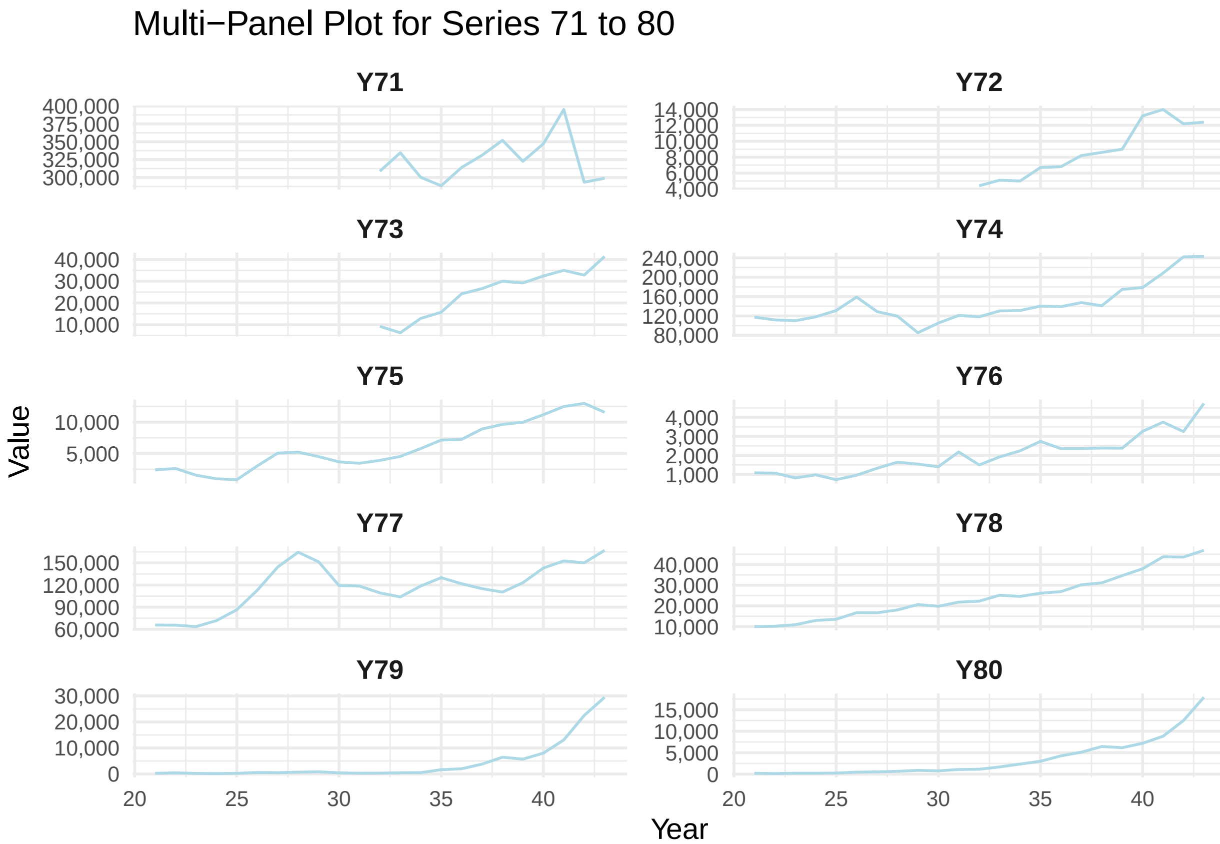

Multi−Panel Plot for Series 31 to 40
Y31
Y32
Y33
Y34
Y35
Y36
Y37
Y38
Y39
Y40
Value
Year


Multi−Panel Plot for Series 41 to 50
Y41
Y42
Y43
Y44
Y45
Y46
Y47
Y48
Y49
Y50
Value
Year

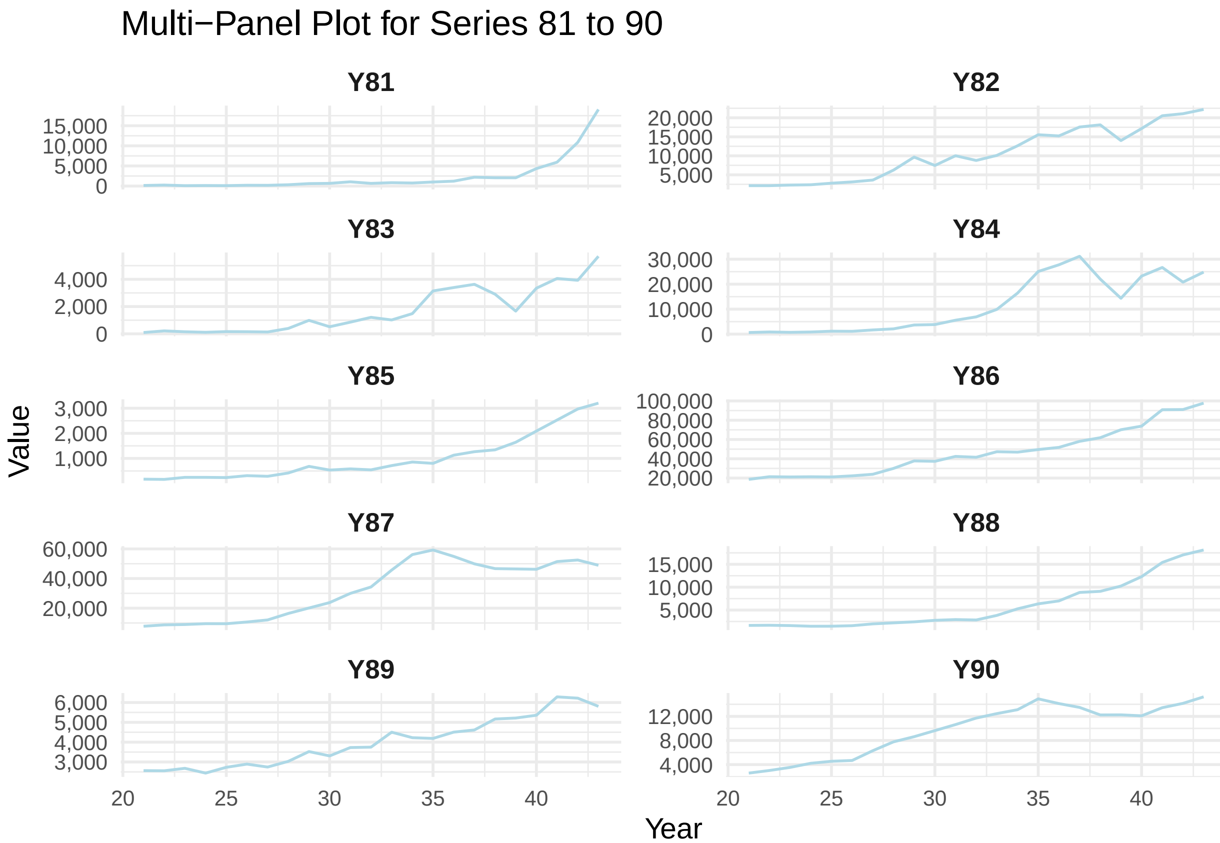

Multi−Panel Plot for Series 51 to 60
Y51
Y52
Y53
Y54
Y55
Y56
Y57
Y58
Y59
Y60
Value
Year

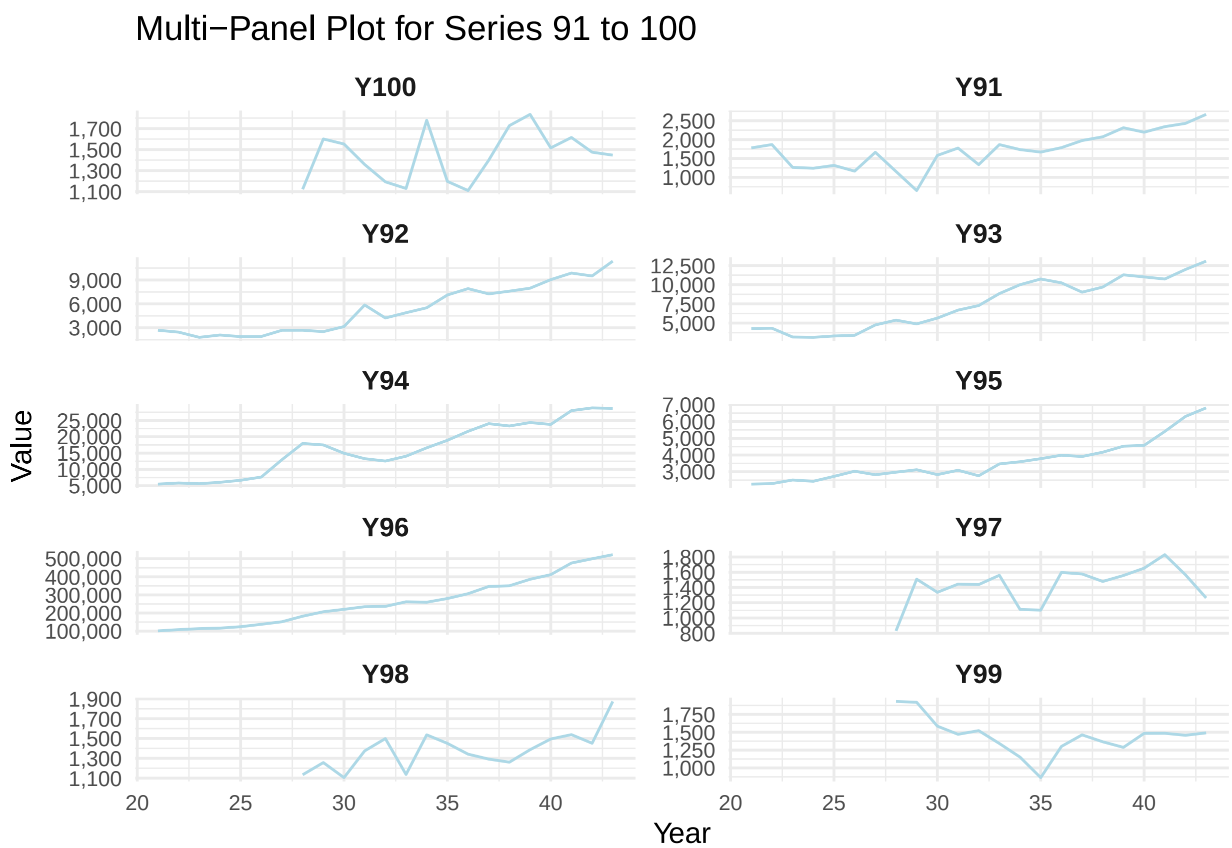


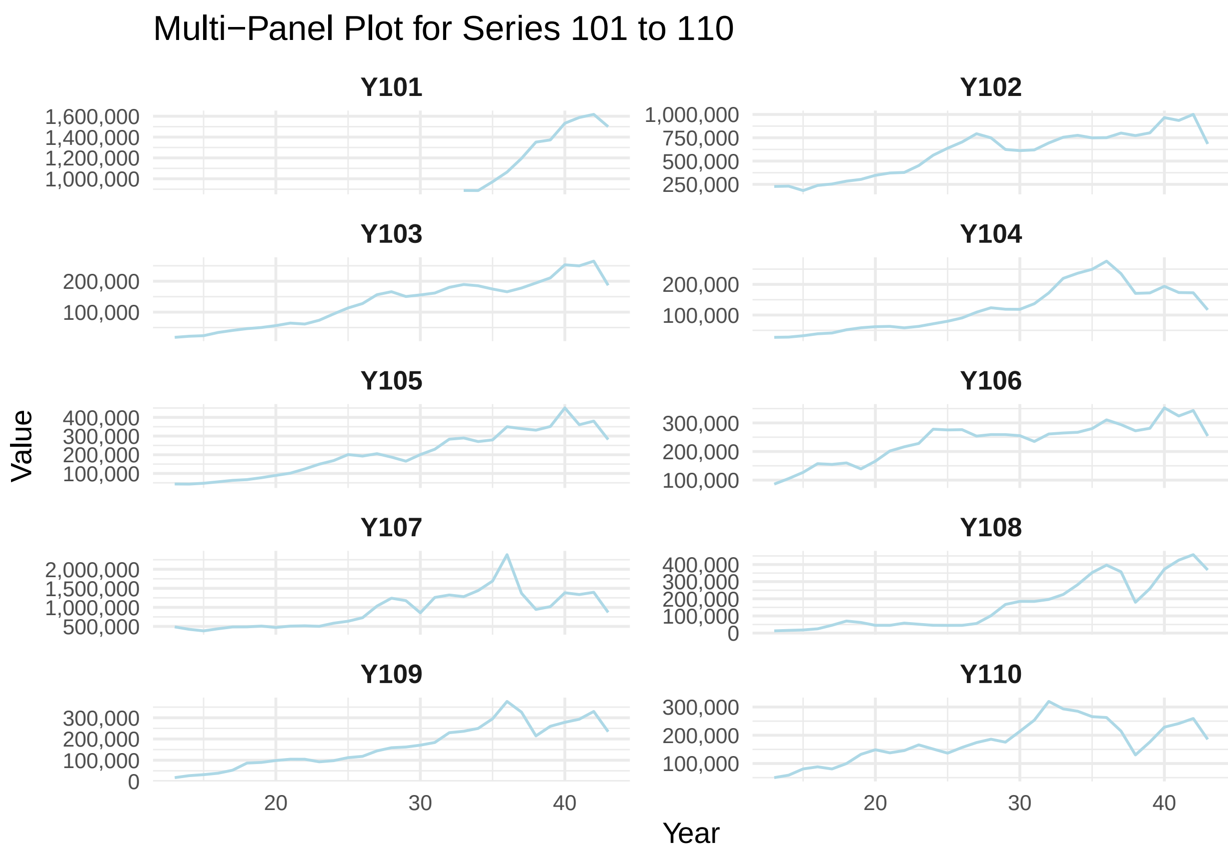
Multi−Panel Plot for Series 61 to 70
Y61
6,000
5,000
4,000
3,000
Y62
50,000
40,000
30,000
Y63
45,000
40,000
35,000
Y64
12,000
10,000
8,000
6,000
4,000
Y65
70,000
60,000
50,000
Y66
35,000
30,000
25,000
20,000
Y67
2,000
1,500
1,000
Y68
400,000
350,000
300,000
250,000
Y69
6,000
5,000
4,000
3,000
Y70
17,500
15,000
12,500
10,000
32
36
40
32
36
40
Value
Year



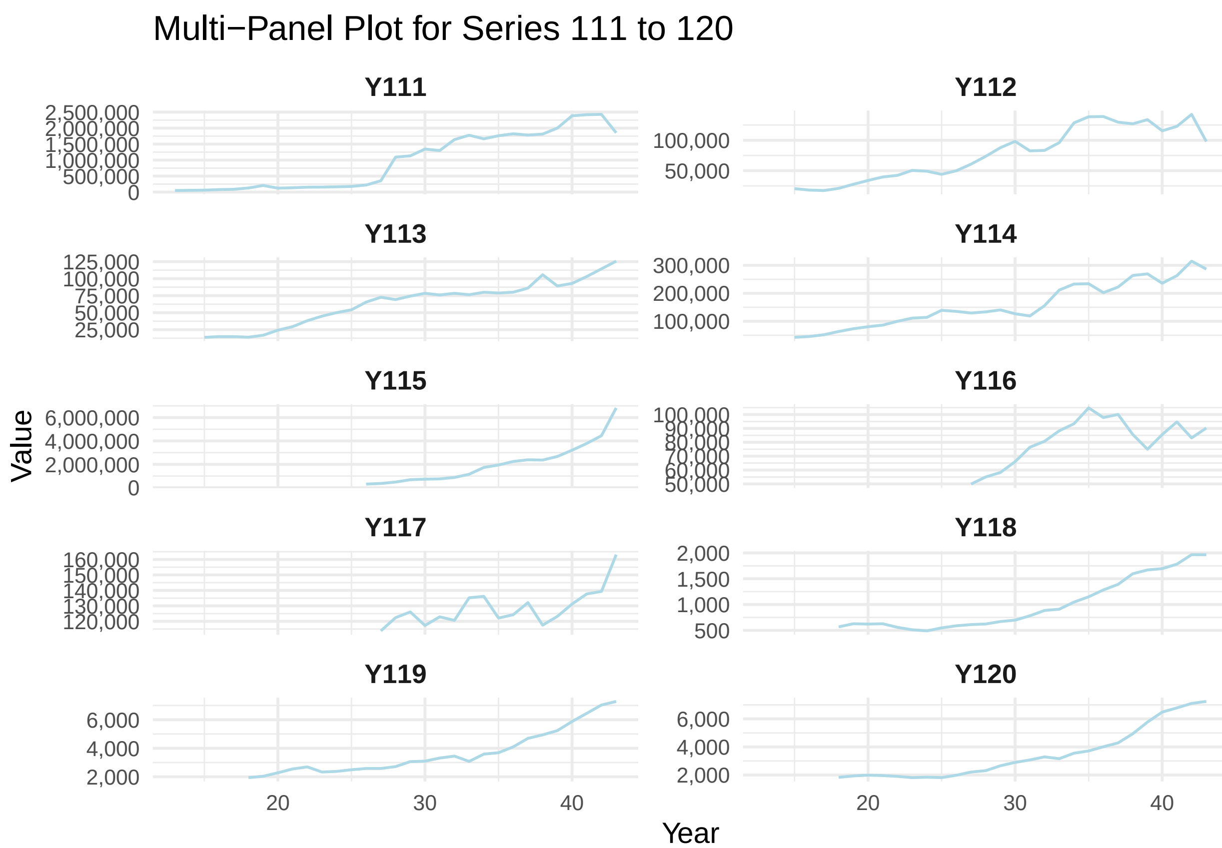
Multi−Panel Plot for Series 71 to 80
Y71
400,000
375,000
350,000
325,000
300,000
Y72
14,000
12,000
10,000
8,000
6,000
4,000
Y73
40,000
30,000
20,000
10,000
Y74
240,000
200,000
160,000
120,000
80,000
Y75
10,000
5,000
Y76
4,000
3,000
2,000
1,000
Y77
150,000
120,000
90,000
60,000
Y78
40,000
30,000
20,000
10,000
Y79
30,000
20,000
10,000
0
Y80
15,000
10,000
5,000
0
20
25
30
35
40
Value
Year

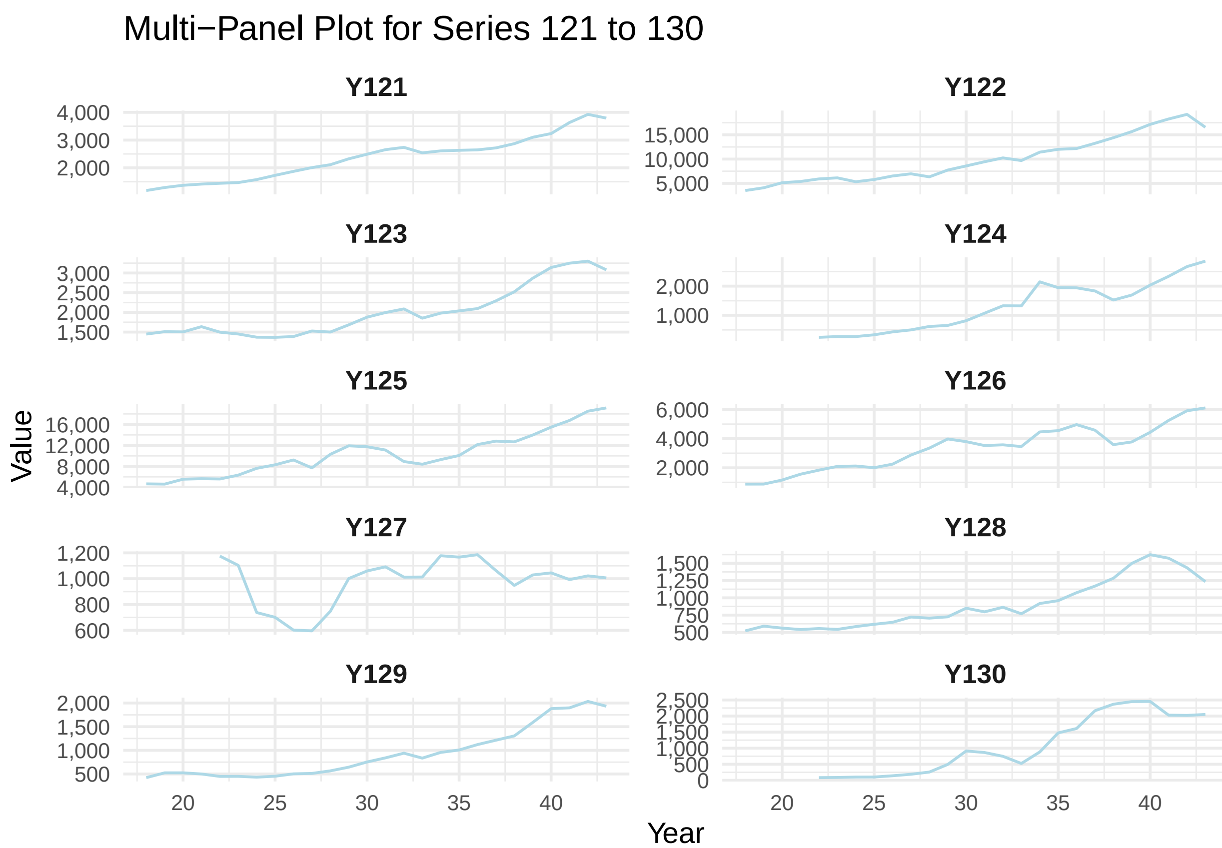

Multi−Panel Plot for Series 81 to 90
Y81
Y82
Y83
Y84
Y85
Y86
Y87
Y88
Y89
Y90
Value
Year

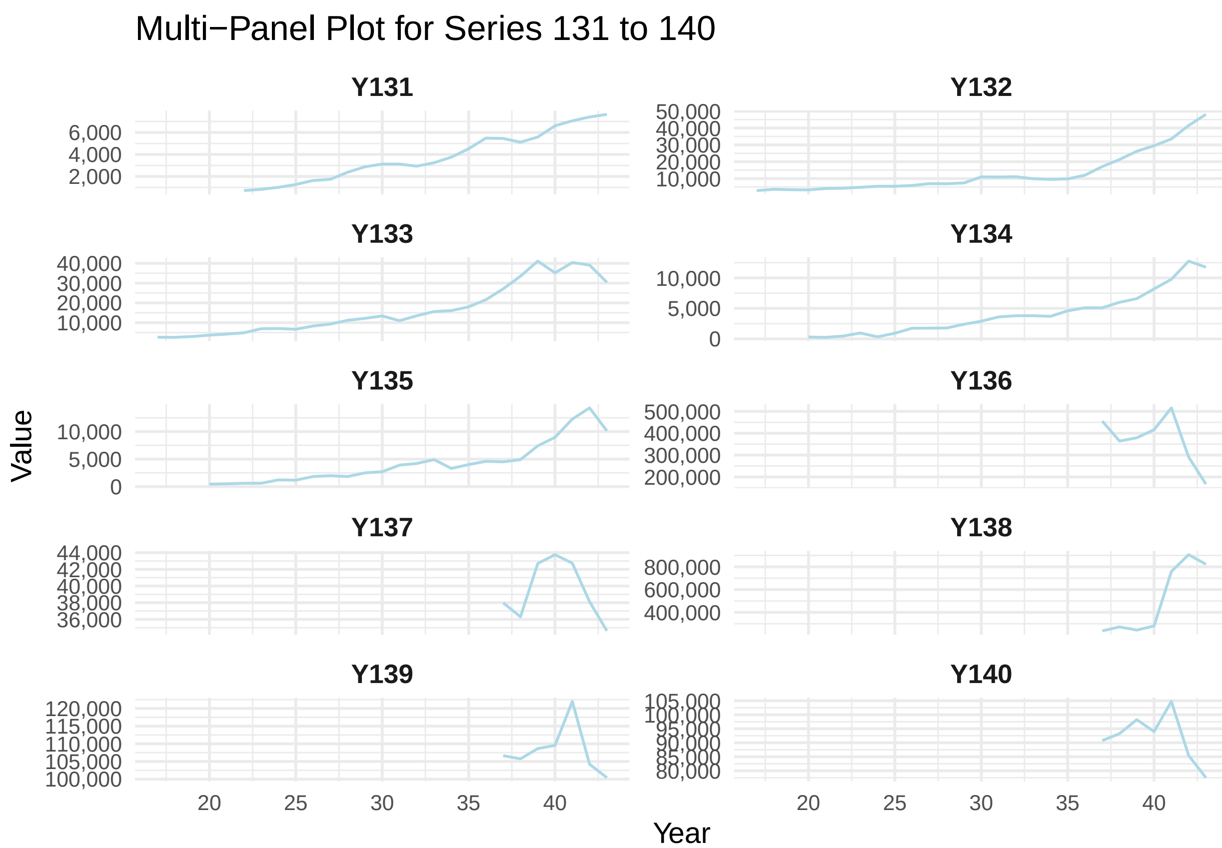

Multi−Panel Plot for Series 91 to 100
Y100
Y91
Y92
Y93
Y94
Y95
Y96
Y97
Y98
Y99
Value
Year



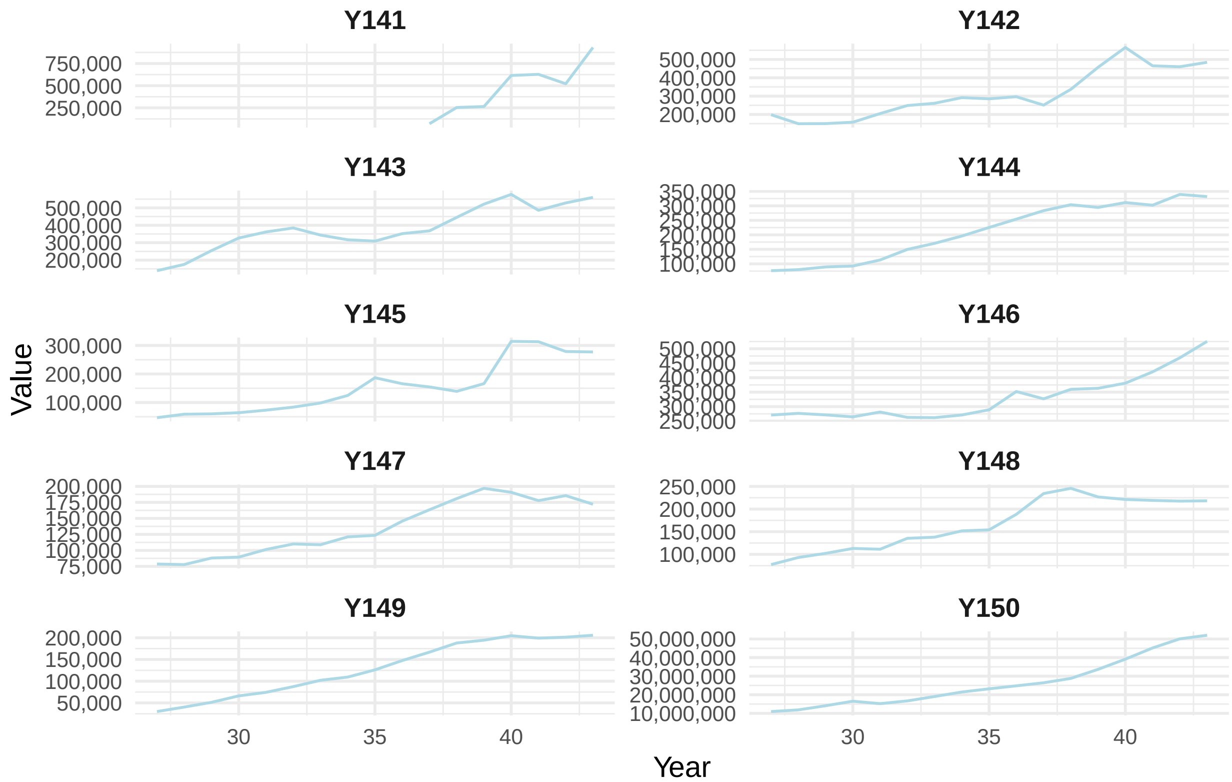
Multi−Panel Plot for Series 101 to 110
Y101
1,600,000
1,400,000
1,200,000
1,000,000
Y102
1,000,000
750,000
500,000
250,000
Y103
200,000
100,000
Y104
200,000
100,000
Y105
400,000
300,000
200,000
100,000
Y106
300,000
200,000
100,000
Y107
2,000,000
1,500,000
1,000,000
500,000
Y108
400,000
300,000
200,000
100,000
0
Y109
300,000
200,000
100,000
0
Y110
300,000
200,000
100,000
20
30
40
20
30
40
Value
Year

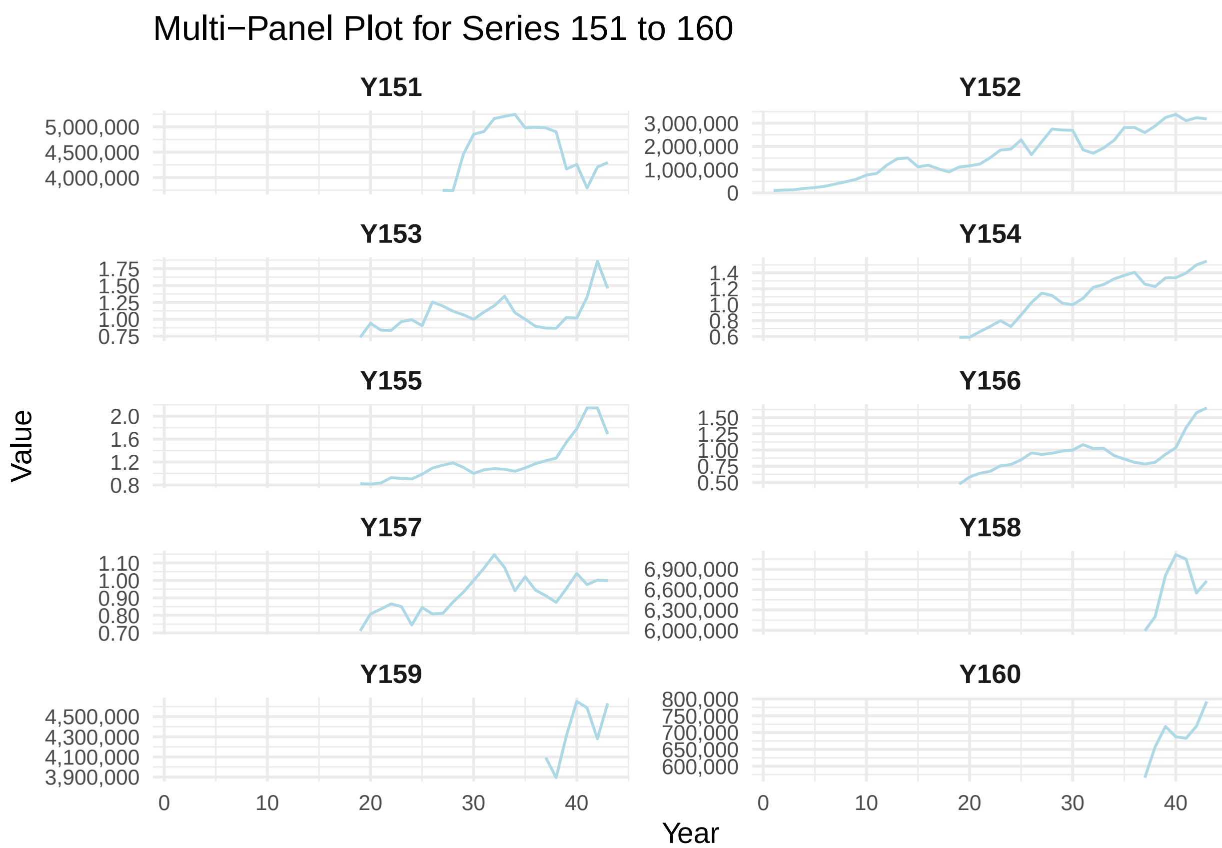

Multi−Panel Plot for Series 111 to 120
Y111
Y112
Y113
Y114
Y115
Y116
Y117
Y118
Y119
Y120
Value
Year
20
30
40


Multi−Panel Plot for Series 121 to 130
Y121
4,000
3,000
2,000
Y122
15,000
10,000
5,000
Y123
3,000
2,500
2,000
1,500
Y124
2,000
1,000
Y125
16,000
12,000
8,000
4,000
Y126
6,000
4,000
2,000
Y127
1,200
1,000
800
600
Y128
1,500
1,250
1,000
750
500
Y129
2,000
1,500
1,000
500
Y130
2,500
2,000
1,500
1,000
500
0
20
25
30
35
40
Value
Year

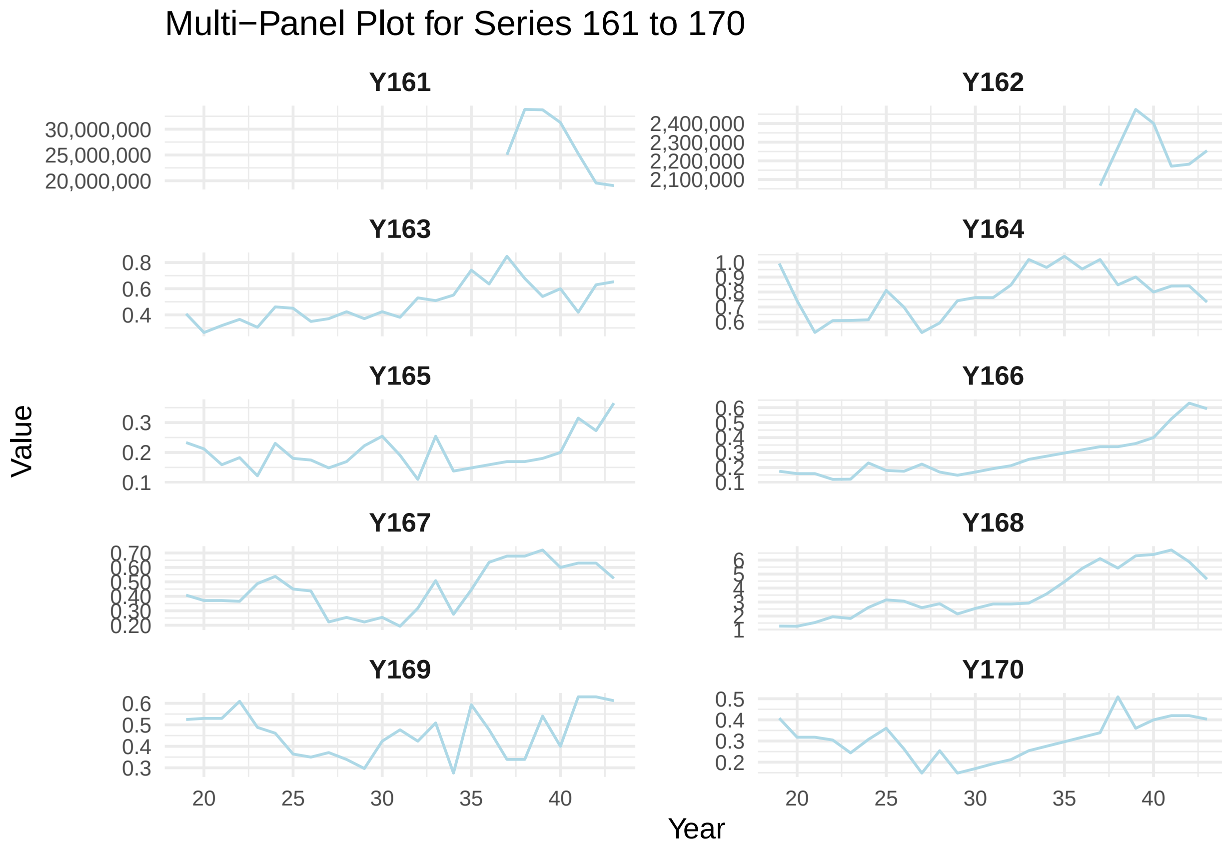

Multi−Panel Plot for Series 131 to 140
Y131
Y132
Y133
Y134
Y135
Y136
Y137
Y138
Y139
Y140
Value
Year

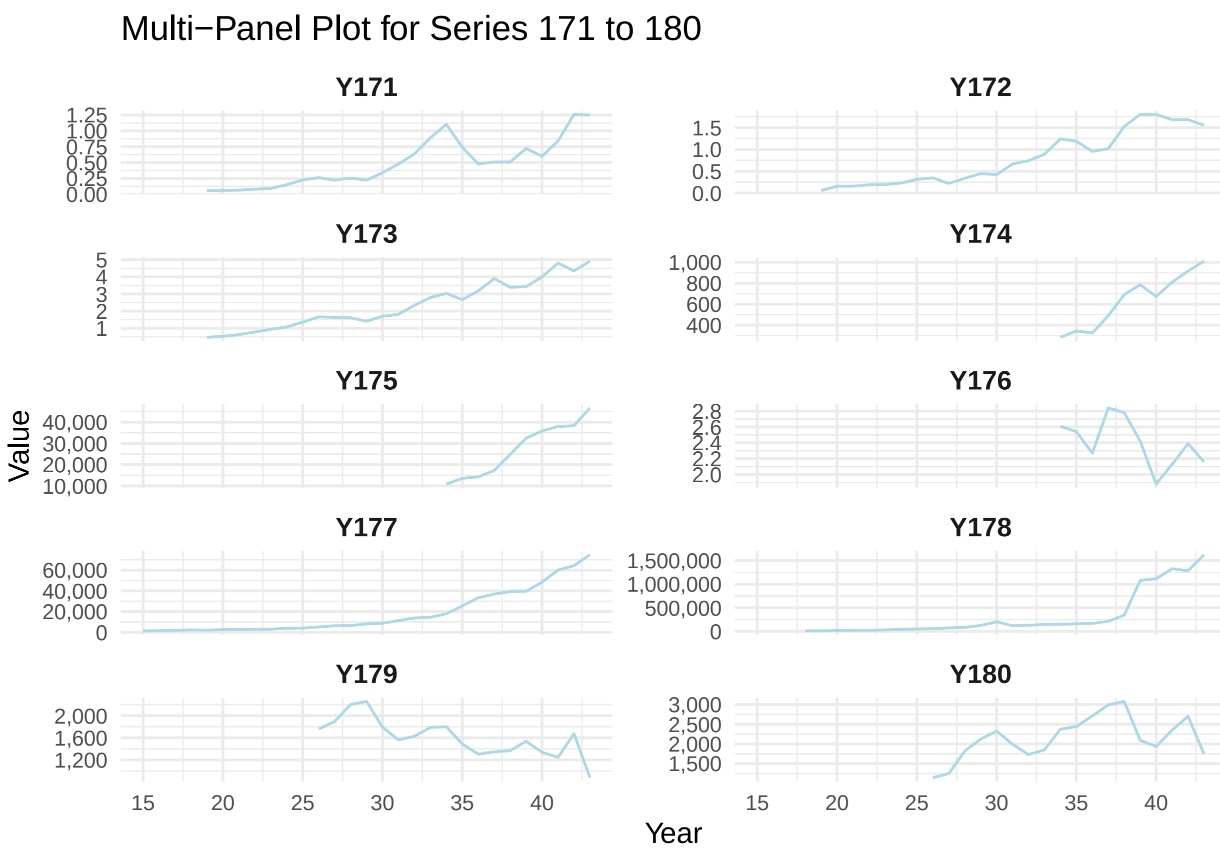
Multi−Panel Plot for Series 141 to 150

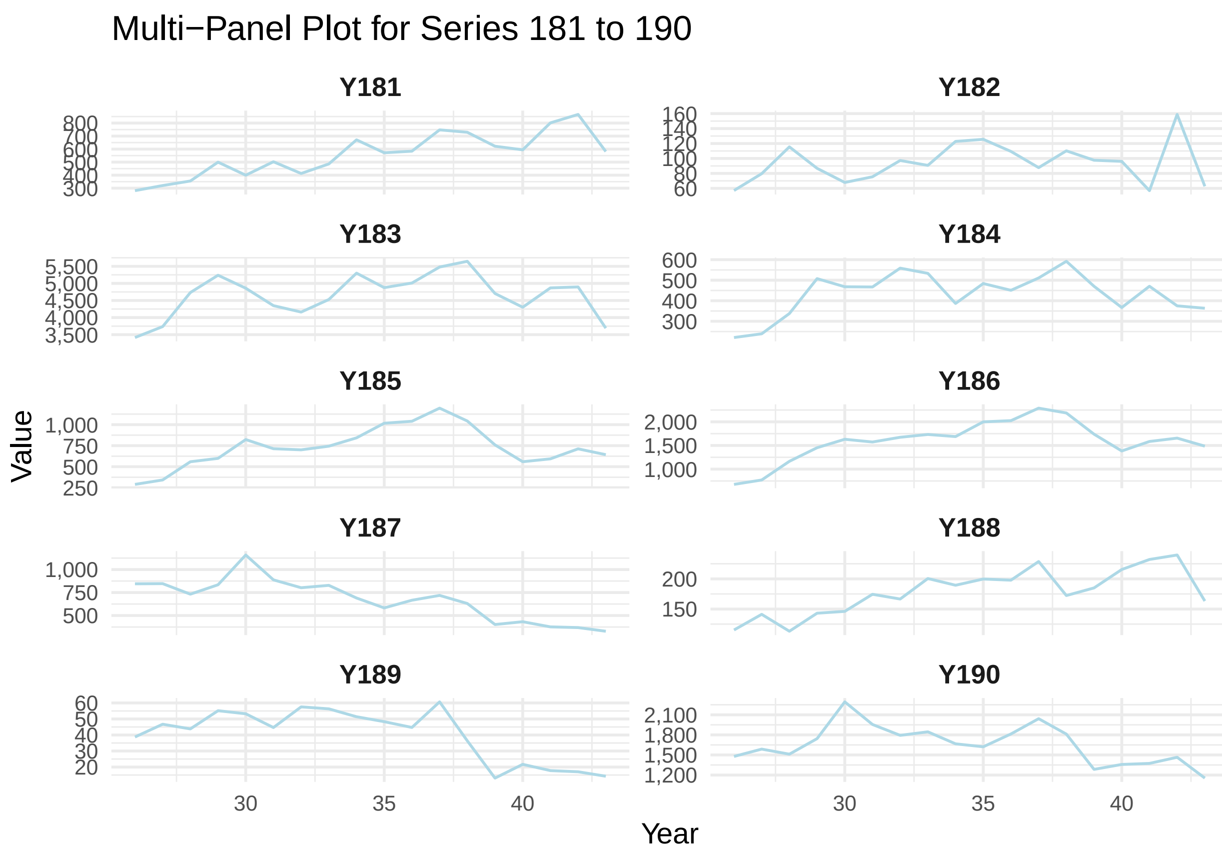




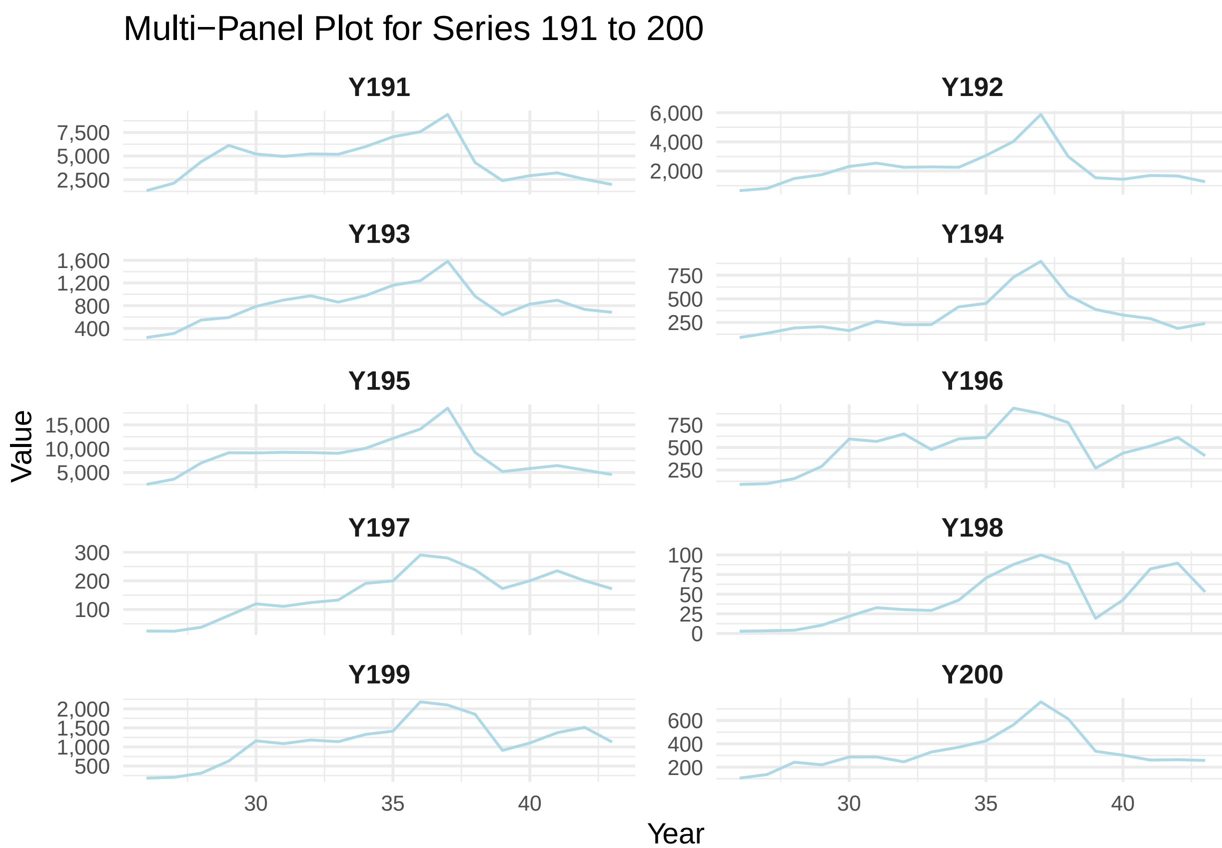
Multi−Panel Plot for Series 151 to 160
Y151
Y152
Y153
Y154
Y155
Y156
Y157
Y158
Y159
Y160
Value
Year
5,000,000
4,500,000
4,000,000
3,000,000
2,000,000
1,000,000
0
1.75
1.50
1.25
1.00
0.75
1.4
1.2
1.0
0.8
0.6
2.0
1.6
1.2
0.8
1.50
1.25
1.00
0.75
0.50
1.10
1.00
0.90
0.80
0.70
6,900,000
6,600,000
6,300,000
6,000,000
4,500,000
4,300,000
4,100,000
3,900,000
800,000
750,000
700,000
650,000
600,000
0
10
20
30
40


Multi−Panel Plot for Series 161 to 170
Y161
Y162
Y163
Y164
Y165
Y166
Y167
Y168
Y169
Y170
Value
Year
30,000,000
25,000,000
20,000,000
2,400,000
2,300,000
2,200,000
2,100,000
20
25
30
35
40

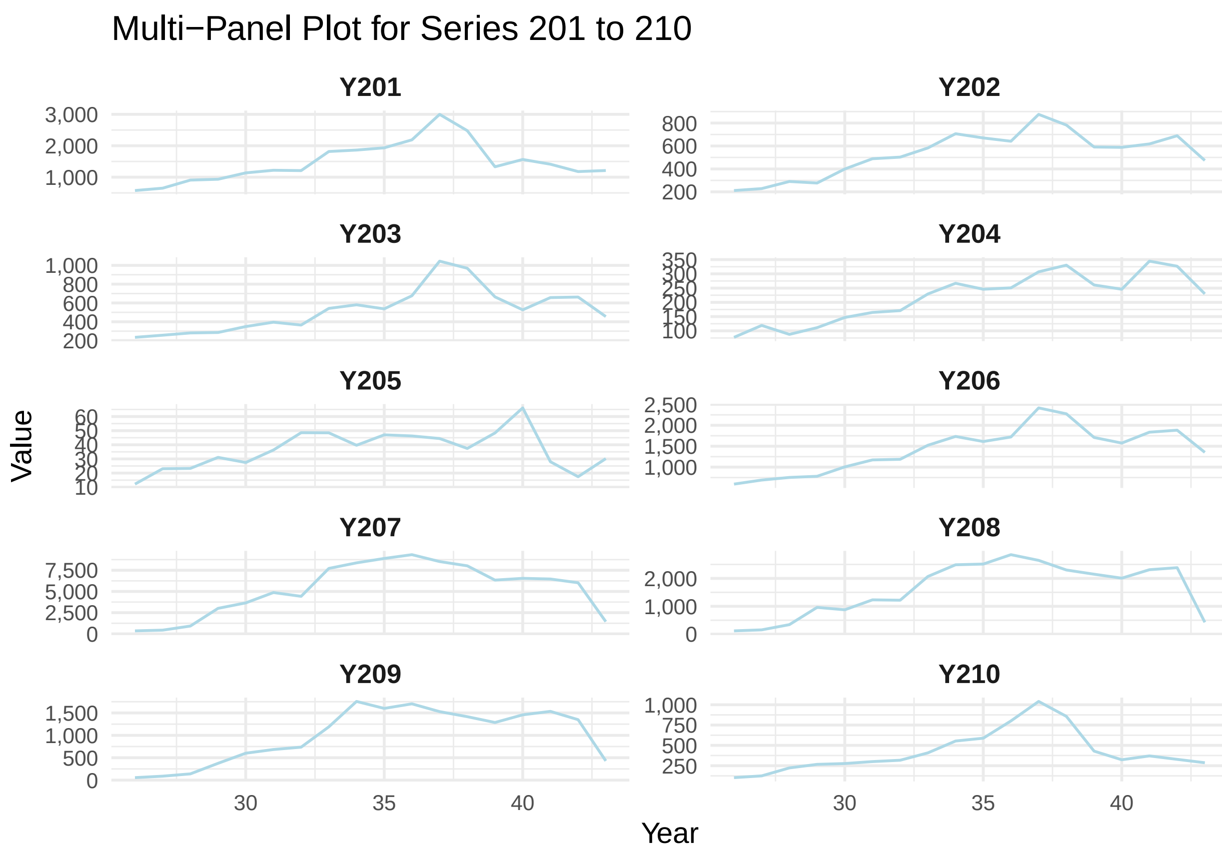

Multi−Panel Plot for Series 171 to 180
Y171
Y172
Y173
Y174
Y175
Y176
Y177
Y178
Y179
Y180
Value
Year

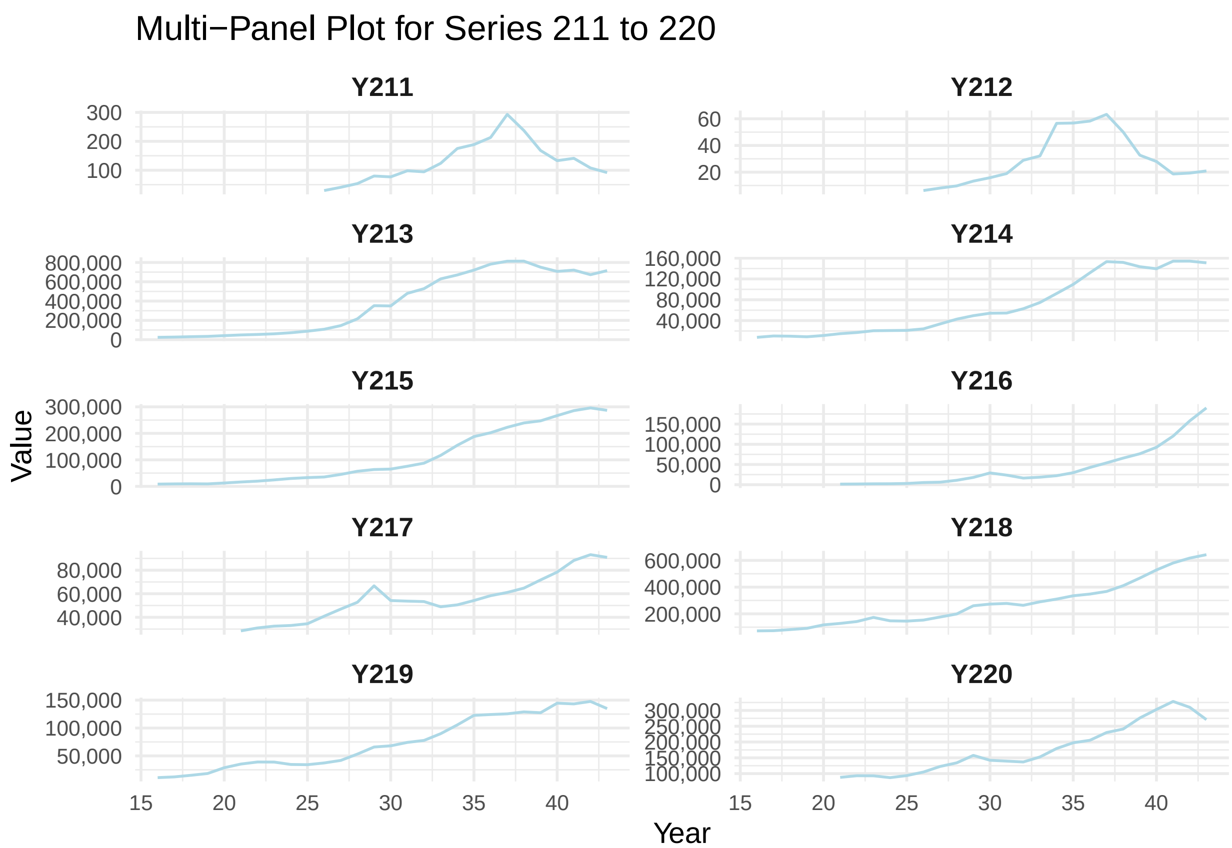


Multi−Panel Plot for Series 181 to 190
Y181
Y182
Y183
Y184
Y185
Y186
Y187
Y188
Y189
Y190
Value
Year
300
400
500
600
700
800
60
80
100
120
140
160
3,500
4,000
4,500
5,000
5,500
300
400
500
600
250
500
750
1,000
1,000
1,500
2,000
500
750
1,000
150
200
20
30
40
50
60
1,200
1,500
1,800
2,100
30
35
40

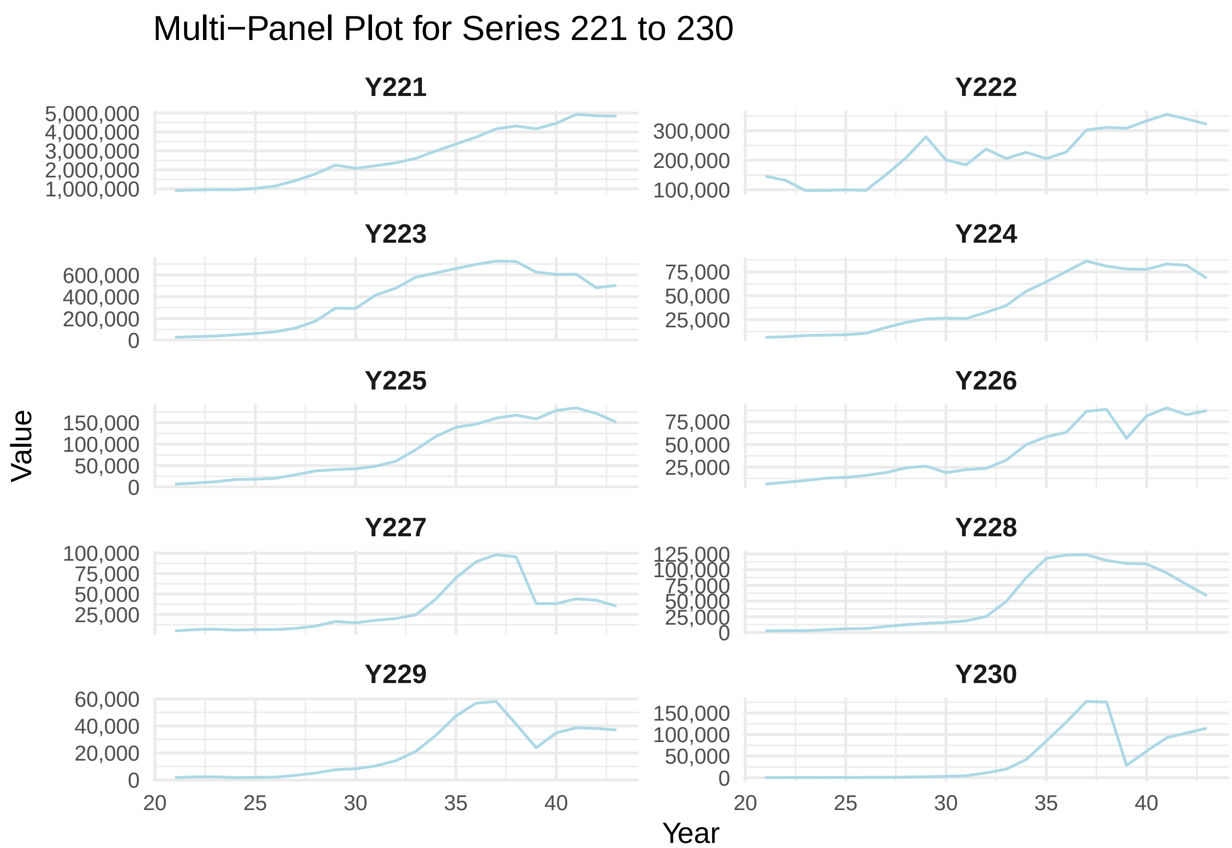


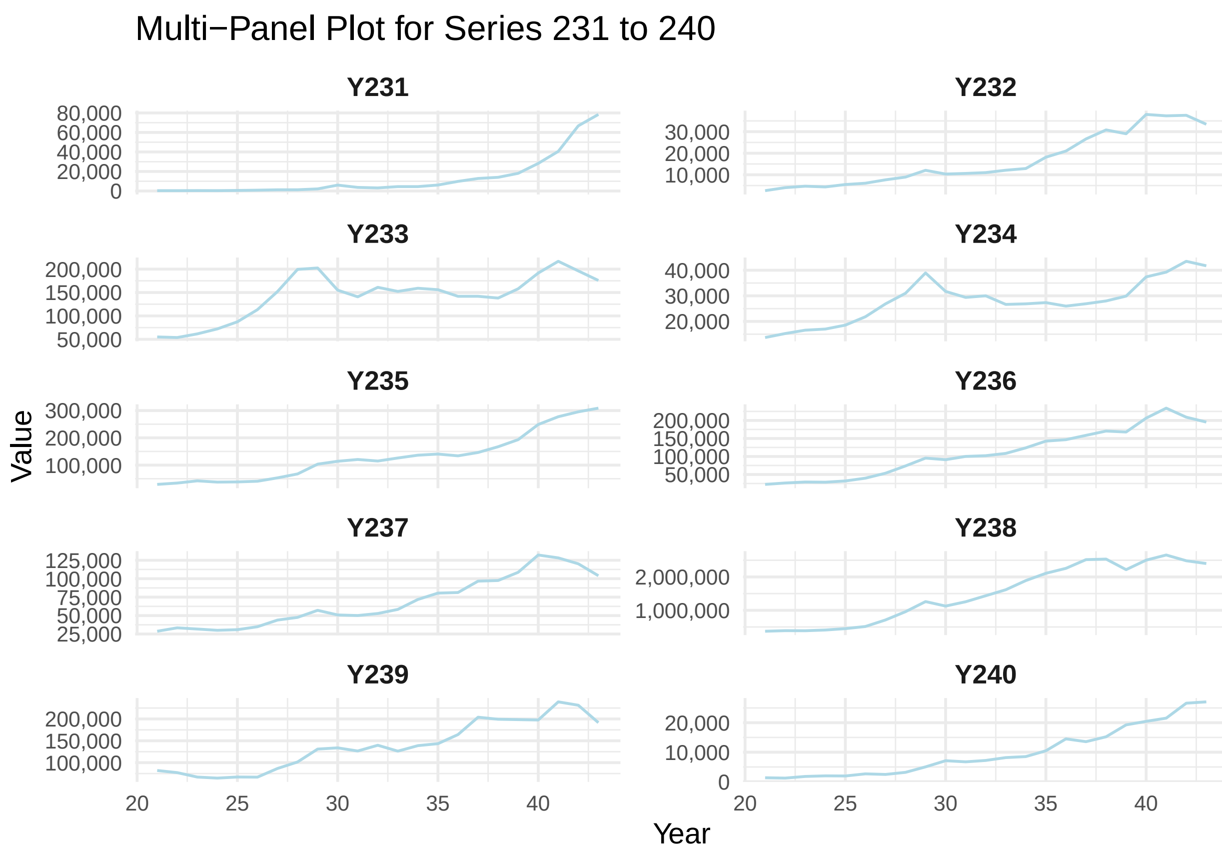
Multi−Panel Plot for Series 191 to 200
Y191
Y192
Y193
Y194
Y195
Y196
Y197
Y198
Y199
Y200
Value
Year

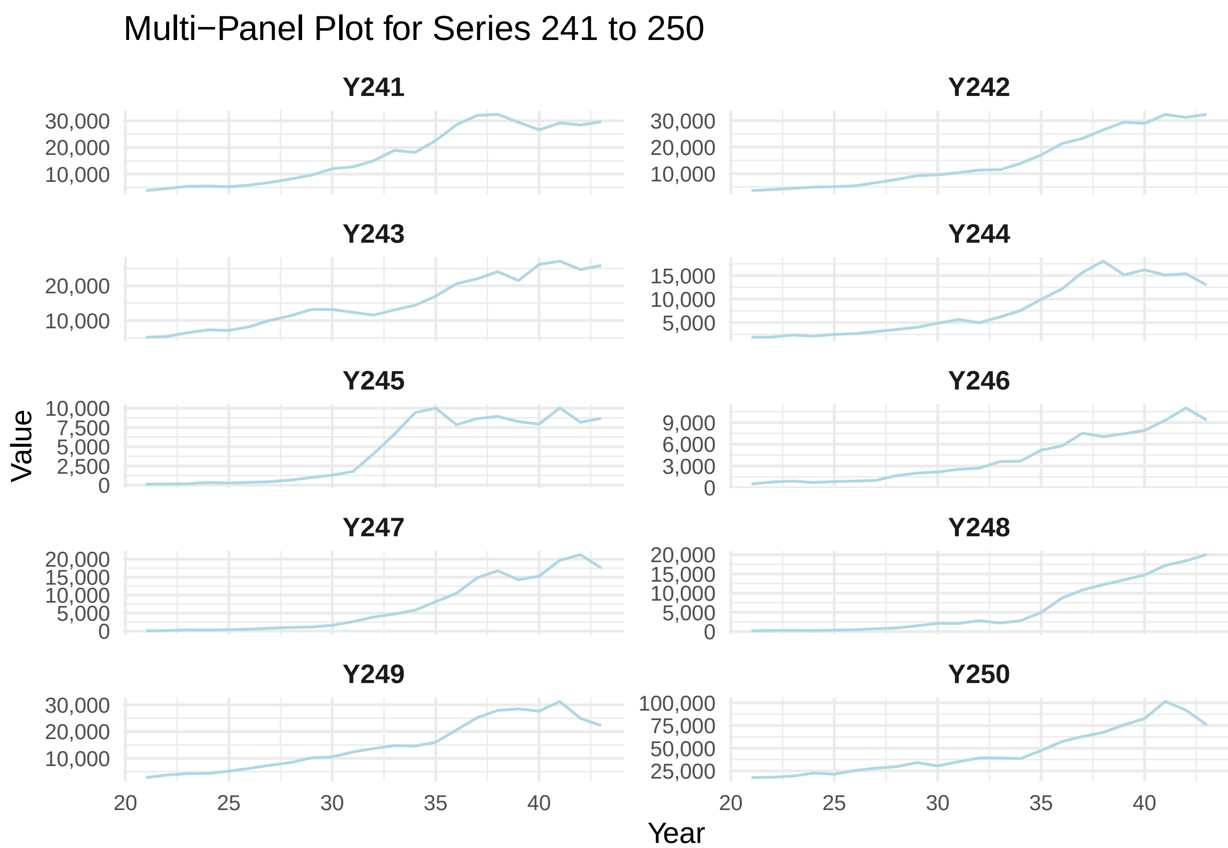

Multi−Panel Plot for Series 201 to 210
Y201
Y202
Y203
Y204
Y205
Y206
Y207
Y208
Y209
Y210
Value
Year

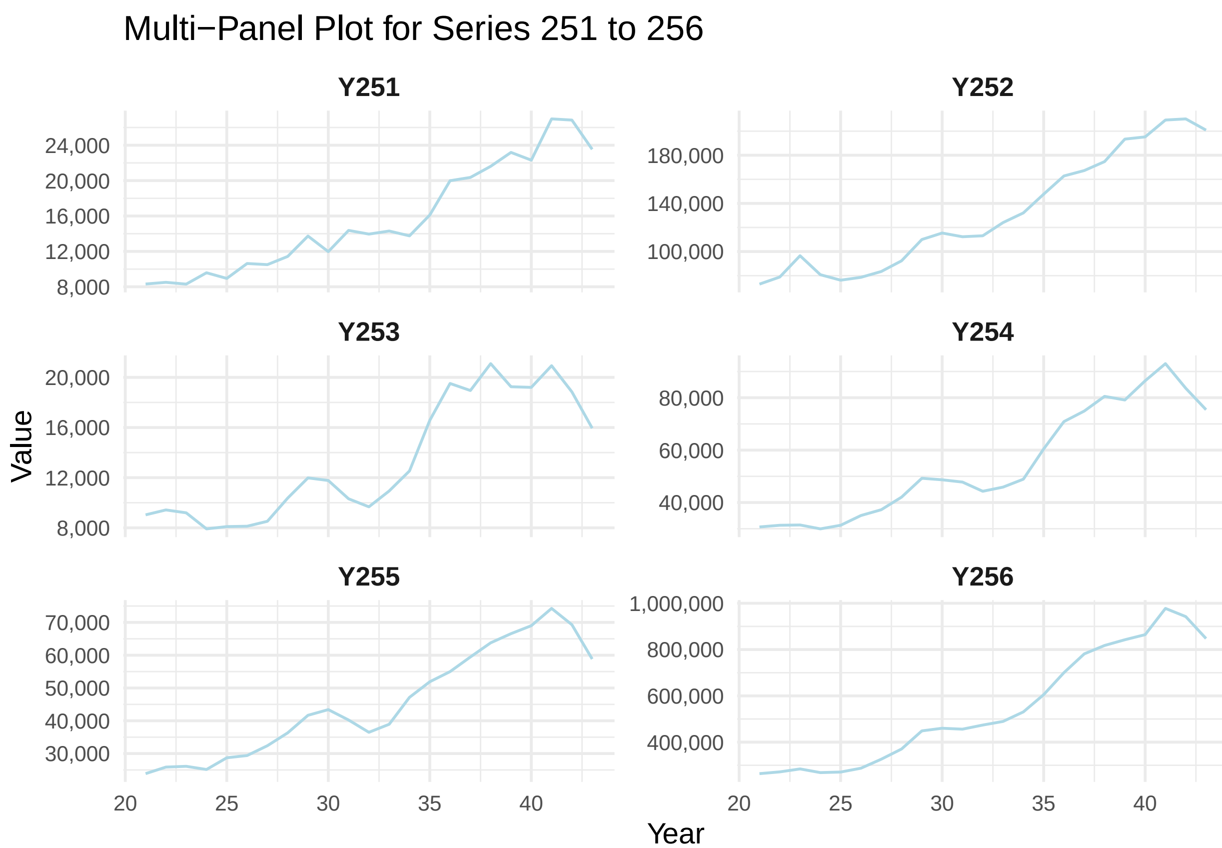


Multi−Panel Plot for Series 211 to 220
Y211
300
200
100
Y212
60
40
20
Y213
800,000
600,000
400,000
200,000
0
Y214
160,000
120,000
80,000
40,000
Y215
300,000
200,000
100,000
0
Y216
150,000
100,000
50,000
0
Y217
80,000
60,000
40,000
Y218
600,000
400,000
200,000
Y219
150,000
100,000
50,000
Y220
300,000
250,000
200,000
150,000
100,000
15
20
25
30
35
40
Value
Year


Multi−Panel Plot for Series 221 to 230
Y221
Y222
Y223
Y224
Y225
Y226
Y227
Y228
Y229
Y230
5,000,000
4,000,000
3,000,000
2,000,000
1,000,000
300,000
200,000
100,000
600,000
400,000
200,000
0
75,000
50,000
25,000
150,000
100,000
50,000
0
75,000
50,000
25,000
100,000
75,000
50,000
25,000
125,000
100,000
75,000
50,000
25,000
0
60,000
40,000
20,000
0
150,000
100,000
50,000
0
20
25
30
35
40
Value
Year


Multi−Panel Plot for Series 231 to 240
Y231
Y232
Y233
Y234
Y235
Y236
Y237
Y238
Y239
Y240
Value
Year
80,000
60,000
40,000
20,000
0
30,000
20,000
10,000
200,000
150,000
100,000
50,000
40,000
30,000
20,000
300,000
200,000
100,000
200,000
150,000
100,000
50,000
125,000
100,000
75,000
50,000
25,000
2,000,000
1,000,000
200,000
150,000
100,000
20,000
10,000
0
20
25
30
35
40


Multi−Panel Plot for Series 241 to 250
Y241
Y242
Y243
Y244
Y245
Y246
Y247
Y248
Y249
Y250
Value
Year
20
25
30
35
40



We can observe some general trends in the data. Overall, the data tend to increase regardless of their starting time. Most series exhibit a consistent upward trend until the end, with some fluctuations. There is no obvious seasonality in the data.

Most series share a similar trend, but slight differences in time-based changes do exist. The main variation lies in the starting point of each series, which is influenced by missing values. As a result, each series begins at a different point in time.

Missing values are present and affect the starting point of each series individually.

Step 2

```
train.tourism <- tourism_new |>
  group_by(Series_Name) |>
  mutate(Max_Time = max(Time)) |>
  filter(Time <= Max_Time - 4) |>
  select(-Max_Time)

valid.tourism <- tourism_new |>
  group_by(Series_Name) |>
  mutate(Max_Time = max(Time)) |>
  filter(Time > Max_Time - 4) |>
  select(-Max_Time)
```

We may use data partitioning before any forecasting to address the problem of over fitting.We split series into two periods, using one to build forecasting model and use the other one to test. We may measure the forecast errors by seeing the difference between predicated values and actual values.

Disadvantages of doing this are Validation May Not Represent Future Trends: The validation set reflects only the most recent period, which may not capture future trends or seasonality. And may also occur problems such as over fitting to recent data.

Step 3

```
library(fable)
```

```
## Loading required package: fabletools
```

```
## Registered S3 method overwritten by 'tsibble':
##   method               from
##   as_tibble.grouped_df dplyr
```

```
library(tsibble)
```

```
##
## Attaching package: 'tsibble'
```

```
## The following object is masked _by_ '.GlobalEnv':
##
##     tourism
```

```
## The following object is masked from 'package:lubridate':
##
##     interval
```

```
## The following objects are masked from 'package:base':
##
##     intersect, setdiff, union
```

```
train_tourism <- train.tourism |>
  as_tsibble(key = Series_Name, index = Time)

valid_tourism <- valid.tourism |>
  as_tsibble(key = Series_Name, index = Time)

# Fit naive and seasonal naive models on the training data
fit <- train_tourism |>
  model(
    naive_model = NAIVE(Value)
  )

# Forecast for the validation period
fc <- fit |>
  forecast(h = nrow(valid_tourism) / n_distinct(valid_tourism$Series_Name))
```

Step 4

```r
accuracy_results <- fc |>
  accuracy(valid_tourism) |>
  select(Series_Name, .model, MAE, AvgError = ME, MAPE, RMSE) |>
  filter(.model == "naive_model") |>
  select(-.model)


# Reorder the Series_Name
accuracy_results <- accuracy_results |>
  mutate(
    Series_Number = as.numeric(gsub("Y", "", Series_Name))
  ) |>
  arrange(Series_Number) |>
  select(-Series_Number)

head(accuracy_results, 20)
```

```
## # A tibble: 20 x 5
##    Series_Name      MAE  AvgError  MAPE     RMSE
##    <chr>          <dbl>     <dbl> <dbl>    <dbl>
##  1 Y1             6173.     6173. 16.6     6620.
##  2 Y2            82542.    39819. 22.1    83294.
##  3 Y3            45450.    10412. 29.3    50580.
##  4 Y4            21321.    21321. 20.5    23830.
##  5 Y5             8662.     8662. 24.4    14940.
##  6 Y6              426.      426. 15.3      436.
##  7 Y7             1366.     1366. 17.1     1459.
##  8 Y8             1034.     1034. 12.3     1134.
##  9 Y9             1213      1213  13.4     1248.
## 10 Y10             511.      511. 16.2      532.
## 11 Y11             134.     -134. 11.1      181.
## 12 Y12              93        91  12.6      113.
## 13 Y13            2212.     2212. 34.1     2393.
## 14 Y14           12642.    12642. 35.5    13608.
## 15 Y15         1153352. -1153352. 24.6  1257403.
## 16 Y16           27648.    -2180. 74.2    35932.
## 17 Y17           54261.    41919. 10.6    66330.
## 18 Y18           25294    -25294   8.73   26579.
## 19 Y19          527217.   155689.  5.90  744341.
## 20 Y20          457512.  -262097. 14.8   528040.
```

MAE and RMSE are useful because they capture the magnitude of forecast errors. MAE provides the average absolute error, making it easy to interpret in original units, while RMSE penalizes larger errors by squaring deviations, which is helpful when large errors are critical.

Average Error is not suitable because positive and negative errors cancel out, hiding the true error magnitude. MAPE is problematic for series when data include zero, it may work by dropping zeros, but then it may also exclude some useful information.

Step 5

```r
train_fc <- fit |>
  forecast(h = nrow(train_tourism) / n_distinct(train_tourism$Series_Name))
valid_fc <- fit |>
  forecast(h = nrow(valid_tourism) / n_distinct(valid_tourism$Series_Name))

# Compute MAPE for training data
train.mape <- train.tourism |>
  group_by(Series_Name) |>
  mutate(Forecast_Value = lag(Value)) |>
  filter(!is.na(Forecast_Value)) |>
  summarize(Training_MAPE = mean(abs((Value - Forecast_Value)/Value)) * 100)

# Compute MAPE for validation data
valid.mape <- valid_fc |>
  accuracy(valid_tourism) |>
  filter(.model == "naive_model") |>
  select(Series_Name, Validation_MAPE = MAPE)

train.mape
```

```
## # A tibble: 256 x 2
##    Series_Name Training_MAPE
##    <chr>               <dbl>
##  1 Y1                   4.10
##  2 Y10                 10.3
##  3 Y100                18.7
##  4 Y101                 6.93
##  5 Y102                 8.86
##  6 Y103                10.7
##  7 Y104                11.3
##  8 Y105                10.4
##  9 Y106                 7.84
## 10 Y107                16.0
## # i 246 more rows
```

```r
valid.mape
```

```
## # A tibble: 256 x 2
##    Series_Name Validation_MAPE
##    <chr>                 <dbl>
##  1 Y1                     16.6
##  2 Y10                    16.2
##  3 Y100                   21.5
##  4 Y101                   11.9
##  5 Y102                   17.1
##  6 Y103                   16.4
##  7 Y104                   14.8
##  8 Y105                   14.3
##  9 Y106                   15.5
## 10 Y107                   23.6
## # i 246 more rows
```

Step 6

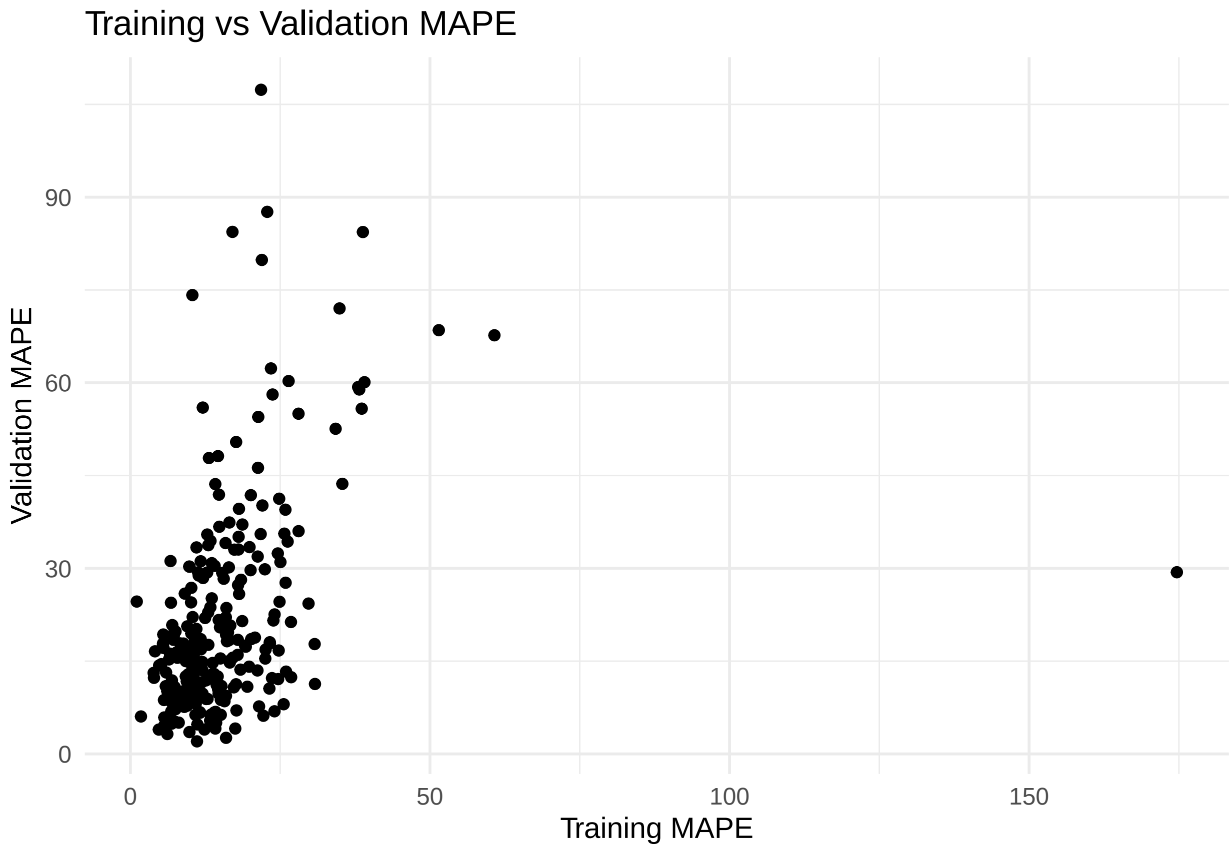
```r
train.mae <- train.tourism |>
  group_by(Series_Name) |>
  arrange(Time) |>
  mutate(Lagged_Value = lag(Value)) |>
  filter(!is.na(Lagged_Value)) |>
  summarize(
    Training_MAE = mean(abs(Value - Lagged_Value), na.rm = TRUE)
  )

valid.mae <- valid_fc |>
  accuracy(valid_tourism) |>
  filter(.model == "naive_model") |>
  select(Series_Name, Validation_MAE = MAE)

# Compute for training mase
train.mase <- train.mae |>
  mutate(Training_MASE = Training_MAE / Training_MAE)

# Compute for validation mase
valid.mase <- valid.mae |>
  left_join(train.mae, by = "Series_Name") |>
  mutate(Validation_MASE = Validation_MAE / Training_MAE)
```

The most major advantage of MASE is that MASE can handle with the zero counts.MASE can avoid a zero value in the denominator. Besides, MAPE gives a heavier penalty to positive errors (over-forecasts) than negative errors (under-forecasts), while MASE weighs both types of errors equally.

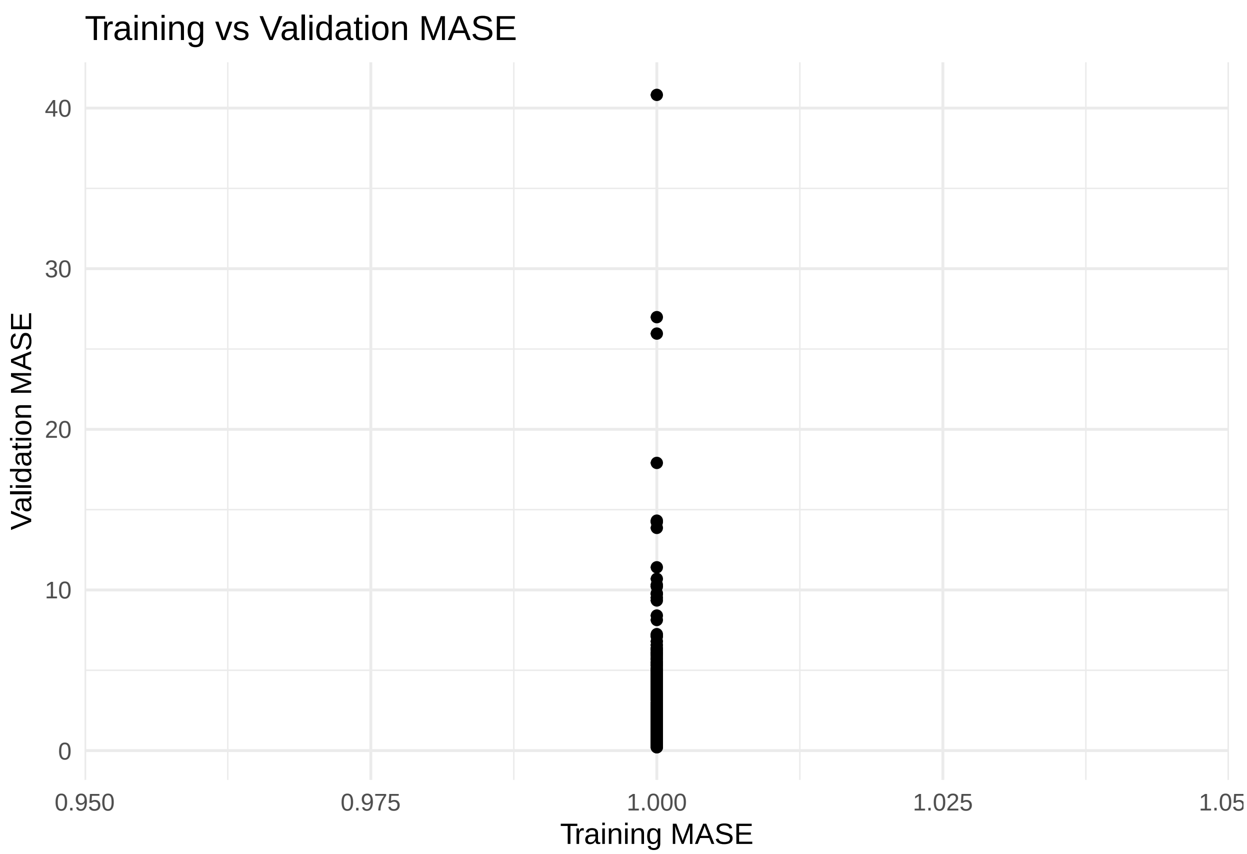
Compare to naive forecasts, MASE is more accurate.

Step 7

```r
ggplot(data = merge(train.mape, valid.mape, by = "Series_Name"), aes(x = Training_MAPE, y = Validation_M
  geom_point() +
  labs(title = "Training vs Validation MAPE", x = "Training MAPE", y = "Validation MAPE") +
  theme_minimal()
```




```
ggplot(data = merge(train.mase, valid.mase, by = "Series_Name"), aes(x = Training_MASE, y = Validation_M
  geom_point() +
  labs(title = "Training vs Validation MASE", x = "Training MASE", y = "Validation MASE") +
  theme_minimal()
```



The first plot shows the relationship between Training MAPE and Validation MAPE, where performance tends to degrade in validation as compared to training, with higher variance. Most series show a relatively low MAPE in training, but validation errors spread wider, indicating overfitting for some series. The second plot shows Training MASE vs. Validation MASE, highlighting more uniformity in training MASE but considerable variability in validation MASE. This implies that while training performance is consistent, the generalization across series during validation varies significantly, especially for poorly fitted models. The wide ranges in both metrics reflect differences in model performance across series.

Step 8

(a)

```r
# Define the global annual growth rate
annual_growth_rate <- 0.06
monthly_growth_rate <- (1 + annual_growth_rate)^(1 / 12)

# Fit naive model on the training data
fit <- train_tourism |>
  model(
    naive_model = NAIVE(Value)
  )

# Generate naive forecasts for the validation period
fc <- fit |>
  forecast(h = nrow(valid_tourism) / n_distinct(valid_tourism$Series_Name))
```

```r
# Apply the 6% annual growth adjustment
fc_adjusted <- fc |>
  as_tibble() |>
  group_by(Series_Name) |>
  mutate(
    Trend_Adjusted_Forecast = .mean * monthly_growth_rate^(row_number())
  )

# Extract adjusted forecasts
naive.forecasts <- fc_adjusted |>
  select(Series_Name, .model, Trend_Adjusted_Forecast)
```

(b) The rationale for multiplying the naive forecasts by a constant is to account for a known or observed trend in the data. This adjustment ensures the forecasts reflect expected changes over time, making them more aligned with the overall growth pattern instead of remaining static.

(c) Dependent variable should be the actual observed value, predicted variable will be the time_index, so that the model is using historical patterns and trends to predict future outcomes.

(d)

```r
fit_models <- split(train_tourism, train_tourism$Series_Name) |>
  lapply(function(df) lm(Value ~ lag(Value), data = df))

# Generate forecasts for each series in the validation data
forecast_list <- split(valid_tourism, valid_tourism$Series_Name) |>
  lapply(function(df) {
    series_name <- unique(df$Series_Name)
    if (!is.null(fit_models[[series_name]])) {
      preds <- tryCatch(
        predict(fit_models[[series_name]], newdata = df),
        error = function(e) rep(NA, nrow(df))
      )
    } else {
      preds <- rep(NA, nrow(df))
    }
    data.frame(
      Series_Name = df$Series_Name,
      Forecast_Value = preds,
      Actual_Value = df$Value
    )
  })

forecasts <- do.call(rbind, forecast_list)

# Compute forecast errors for each series
forecast.errors2<- forecasts |>
  group_by(Series_Name) |>
  summarize(
    MAE = mean(abs(Actual_Value - Forecast_Value), na.rm = TRUE),
    MAPE = mean(abs((Actual_Value - Forecast_Value) / Actual_Value), na.rm = TRUE) * 100,
    RMSE = sqrt(mean((Actual_Value - Forecast_Value)^2, na.rm = TRUE))
  ) |>
```

```r
  ungroup()

forecast.errors2
```

```
## # A tibble: 256 x 4
##    Series_Name      MAE  MAPE     RMSE
##    <chr>          <dbl> <dbl>    <dbl>
##  1 Y1            1732.   4.67   1994.
##  2 Y10            108.   3.40    120.
##  3 Y100            56.8  3.55     90.7
##  4 Y101        107991.   7.04 132860.
##  5 Y102        138123.  18.8  189078.
##  6 Y103         34047.  17.3   48825.
##  7 Y104         27601.  21.3   35828.
##  8 Y105         72992.  23.1   86291.
##  9 Y106         41713.  15.0   50282.
## 10 Y107        200826.  21.0  271064.
## # i 246 more rows
```

(e) Overfitting: Using high-order polynomials risks capturing noise instead of patterns, leading to poor generalization.

Dependence on R-squared. R-squared increases with model complexity, even if predictive accuracy does not improve. Cross-validation or metrics like MASE or RMSE are better for model evaluation.

(f) Holt-Winter's exponential smoothing will be the most reasonable method to be used here. We can see the data has clear trend but no seasonal patterns. Hence, using Holt-Winter's exponential smoothing can effectively captures the trend while providing smoothed forecasts.

(g) To automate the ensemble method and reduce manual tweaking, techniques like automated hyperparameter tuning and machine learning-based ensemble frameworks can be used. For example, algorithms like stacking or blending can combine forecasts from multiple models by assigning optimal weights based on cross-validation performance.

(h) The competition's goal of minimizing the average MAPE across all series emphasizes short-term forecast accuracy for the next four periods, prioritizing global consistency. In practice, tourism forecasting often focuses on longer-term trends, seasonality, and other practical factors like budgeting or resource planning. Real-life forecasting would require incorporating more domain-specific adjustments and iterative refinement beyond minimizing a single error metric.