STAT527 Term Project

Miller Kodish, Ian Ou, Vinay Pundith

12/17/2025

Part 1: Define Helper Functions

(a) These functions are used for tables. useful_columns() checks each column and keeps only the ones that don't have too many missing values, based on a threshold. df_stats() then uses that result to report simple summary info about the data, like how many rows it has and how many "useful" columns remain after filtering out columns with lots of NAs.

```
useful_columns <- function(df, na_threshold = 0.85) {
  na_fraction <- sapply(df, function(col) mean(is.na(col)))
  names(na_fraction[na_fraction <= na_threshold])
}

df_stats <- function(df, na_threshold=0.85) {
  cols <- useful_columns(df, na_threshold)
  list(rows = nrow(df), useful_cols = length(cols))
}
```

(b) These functions are used for plotting. int_plot() automatically turns ggplot figures into interactive Plotly plots when the output is HTML, but keeps them static for PDFs. plot_time_series() cleans out missing values, makes a simple time-series plot with points and a smooth trend line so we don't have to repeat the same code.

```
# make sure plots are interactive in the R when compiled but static when in PDF
int_plot <- function(p) {
  if (knitr::is_html_output()) {
    plotly::ggplotly(p)
  } else {
    p
  }
}

plot_time_series <- function(data, y, title, ylab, color_point, color_line) {
  clean_data <- data |> filter(!is.na(testDate), !is.na(.data[[y]]))
  p <- ggplot(clean_data, aes(x = testDate, y = .data[[y]])) +
    geom_point(alpha = 0.4, color = color_point) +
    geom_smooth(method = "loess", se = FALSE, color = color_line) +
    theme_minimal() +
    scale_x_continuous(breaks = seq(min(clean_data$testDate),
                                    max(clean_data$testDate), by = 1)) +
    labs(title = title, x = "Year", y = ylab)
  int_plot(p)
}
```

Part 2: Loading in Datasets

(a) Load in Geekbench

```
recent_cpu <- read.csv(here("Datasets", "Geekbench", "recent-cpu-v6.csv"))
recent_gpu <- read.csv(here("Datasets", "Geekbench", "recent-gpu-v6.csv"))
single_core <- read.csv(here("Datasets", "Geekbench", "single-core-v4.csv"))
top_multi <- read.csv(here("Datasets", "Geekbench", "top-multi-core-v6.csv"))
top_single <- read.csv(here("Datasets", "Geekbench", "top-single-core-v6.csv"))
```

(b) Load in Kaggle

```
gpu_benchmarks <- read.csv(here("Datasets", "Kaggle", "GPU_benchmarks_v7.csv"))
gpu_scores <- read.csv(here("Datasets", "Kaggle", "GPU_scores_graphicsAPIs.csv"))
```

Part 3: Preprocessing and Merging Datasets

(a) Make GPU names consistent across datasets (lowercase and trimmed). Merge the PassMark and Geekbench data based on the GPU name. After merging, removes duplicate name columns, prints out how many rows are in each dataset (before and after the merge), and shows a quick preview of the merged result.

```
gpu_benchmarks$gpu_name <- tolower(trimws(gpu_benchmarks$gpuName))
gpu_scores$gpu_name <- tolower(trimws(gpu_scores$Device))
merged_gpu <- merge(gpu_benchmarks, gpu_scores, by="gpu_name")
merged_gpu <- merged_gpu |> select(-gpuName, -Device)

cat("Rows in PassMark dataset:", nrow(gpu_benchmarks), "\n")
```

```
## Rows in PassMark dataset: 2317
```

```
cat("Rows in Geekbench dataset:", nrow(gpu_scores), "\n")
```

```
## Rows in Geekbench dataset: 1213
```

```
cat("Rows in merged dataset:", nrow(merged_gpu), "\n\n")
```

```
## Rows in merged dataset: 647
```

```
head(merged_gpu)
```

```
##        gpu_name G3Dmark G2Dmark price gpuValue TDP powerPerformance testDate
## 1       a40-12q    5573     198    NA       NA  NA                NA     2022
## 2 firepro m4000    1597     410 72.83    21.92  NA                NA     2012
## 3 firepro m4100    1059     623    NA       NA  NA                NA     2015
## 4 firepro m4150     999     207    NA       NA  NA                NA     2015
## 5 firepro m4170    1067     290    NA       NA  NA                NA     2015
## 6 firepro m5100    2103     800    NA       NA  NA                NA     2014
##      category Manufacturer  CUDA Metal OpenCL Vulkan
## 1     Unknown       Nvidia 95329    NA 156643     NA
## 2 Workstation          AMD    NA    NA   6494     NA
## 3 Workstation          AMD    NA    NA   5067     NA
## 4     Unknown          AMD    NA    NA   5063   6685
## 5     Unknown          AMD    NA    NA   6347     NA
## 6 Workstation          AMD    NA    NA   9305  10692
```

Part 4: Filtering the Datasets

(a) Split the GPU data by manufacturer and generate a small summary showing how many rows and usable columns each manufacturer has.

```
manufacturers <- unique(gpu_scores$Manufacturer)
gpu_split <- split(gpu_scores, factor(gpu_scores$Manufacturer, levels = manufacturers))

for (m in manufacturers) {
  assign(sprintf("%s_gpu_scores", tolower(m)), subset(gpu_scores, Manufacturer == m))
}

manufacturers <- unique(gpu_scores$Manufacturer)
manufacturer_summary <- map_df(manufacturers, function(m) {
  df <- gpu_scores |> filter(Manufacturer == m)
  s <- df_stats(df)
  tibble(Manufacturer = m, Rows = s$rows, UsefulCols = s$useful_cols)
})

manufacturer_summary
```

```
## # A tibble: 9 x 3
##   Manufacturer  Rows UsefulCols
##   <chr>        <int>      <int>
## 1 Nvidia         404          7
## 2 AMD            546          6
## 3 Apple           21          5
## 4 Qualcomm        22          4
## 5 Intel          144          6
## 6 Other            7          4
## 7 ARM             58          5
## 8 PowerVR         10          4
## 9 Samsung          1          5
```

(b) This splits the GPU benchmark data by supported API (CUDA, Metal, OpenCL, Vulkan) and summarizes each subset. For every test type, it keeps only GPUs with valid scores and reports how many useful rows and columns.

```
cuda_tests <- subset(gpu_scores, !is.na(CUDA))
metal_tests <- subset(gpu_scores, !is.na(Metal))
opencl_tests <- subset(gpu_scores, !is.na(OpenCL))
vulkan_tests <- subset(gpu_scores, !is.na(Vulkan))

test_types <- c("CUDA", "Metal", "OpenCL", "Vulkan")
test_summary <- map_df(test_types, function(t) {
  df <- gpu_scores |> filter(!is.na(.data[[t]]))
  if (nrow(df) == 0) return(NULL)
  s <- df_stats(df)
  tibble(Test = t, Rows = s$rows, UsefulCols = s$useful_cols)
})

test_summary
```

```
## # A tibble: 4 x 3
##   Test    Rows UsefulCols
##   <chr>  <int>      <int>
```

```
## 1 CUDA     266          7
## 2 Metal    241          7
## 3 OpenCL   976          7
## 4 Vulkan   629          7
```

(c) This builds a summary table by manufacturer and benchmark type (CUDA, Metal, OpenCL, Vulkan). For each valid combo, reports how many rows exist and how many columns useful.

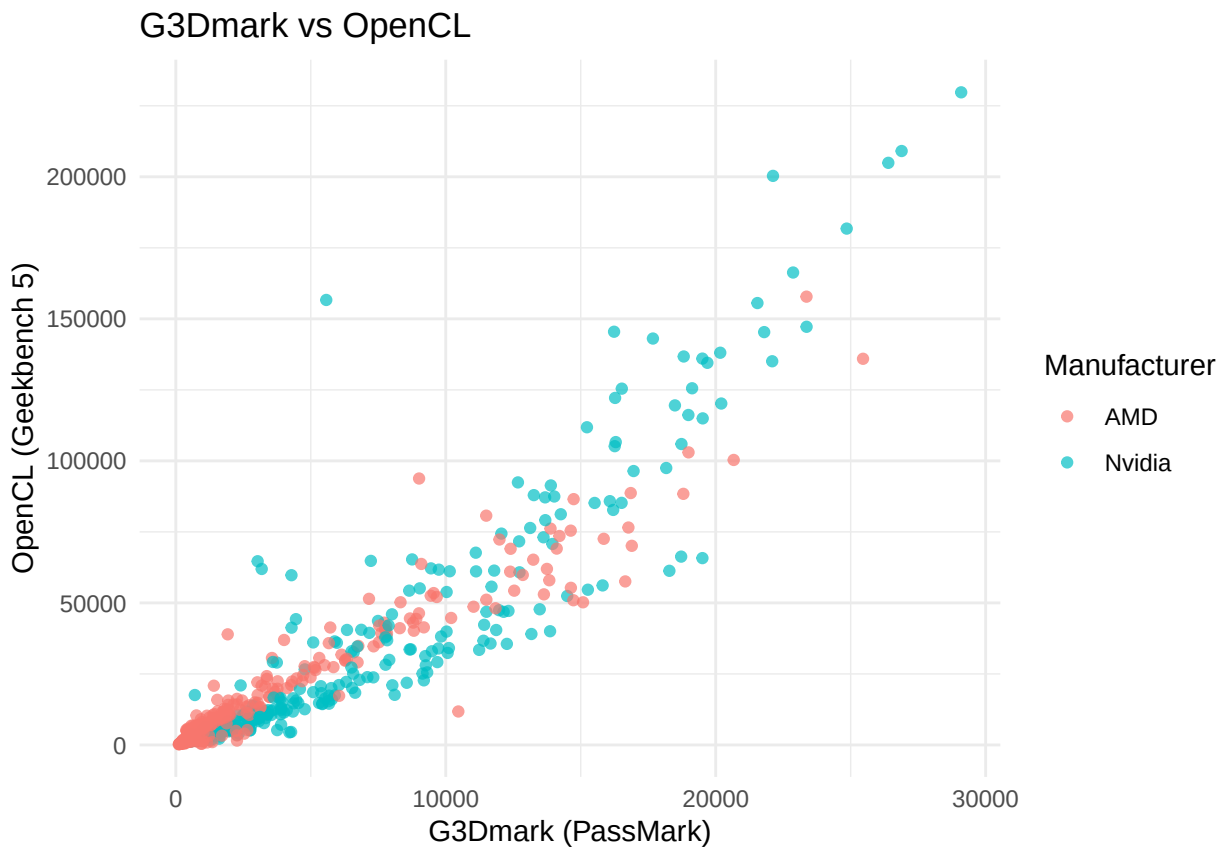
```
summary_table <- map_df(manufacturers, function(m) {
  map_df(test_types, function(t) {
    df <- gpu_scores |> filter(Manufacturer == m, !is.na(.data[[t]]))
    if (nrow(df) == 0) return(NULL)
      s <- df_stats(df)
      tibble(Manufacturer = m, Test = t, Rows = s$rows, UsefulCols = s$useful_cols)
  })
})

summary_table
```

```
## # A tibble: 21 x 4
##    Manufacturer Test    Rows UsefulCols
##    <chr>        <chr>  <int>      <int>
##  1 Nvidia       CUDA     266          7
##  2 Nvidia       Metal     73          7
##  3 Nvidia       OpenCL   381          7
##  4 Nvidia       Vulkan   225          7
##  5 AMD          Metal    123          6
##  6 AMD          OpenCL   452          6
##  7 AMD          Vulkan   251          6
##  8 Apple        Metal     20          5
##  9 Apple        OpenCL     5          5
## 10 Qualcomm     OpenCL     1          4
## # i 11 more rows
```

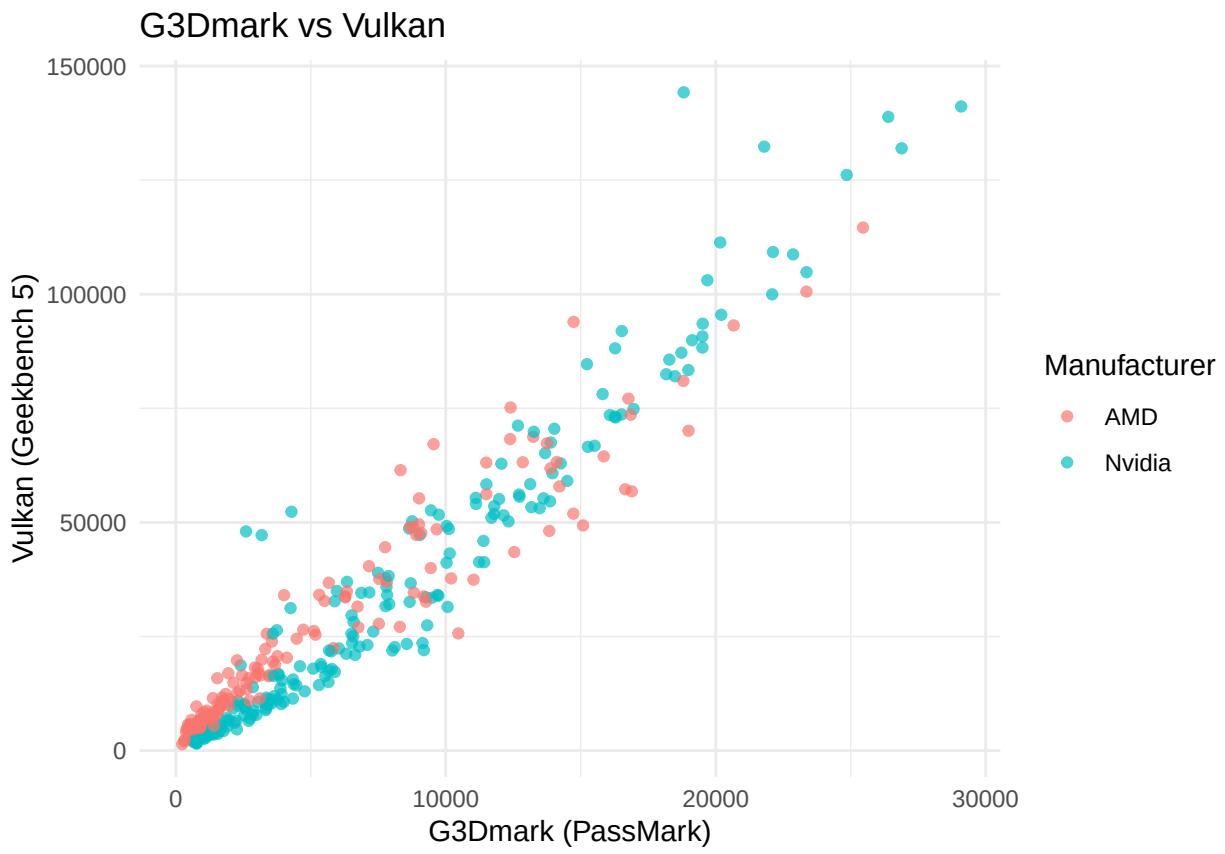
Part 5: Exploring Data Through ggplot() and plotly()

(a) Comparing AMD vs Nvidia (CUDA/OpenCL/Vulkan/G3dmark)

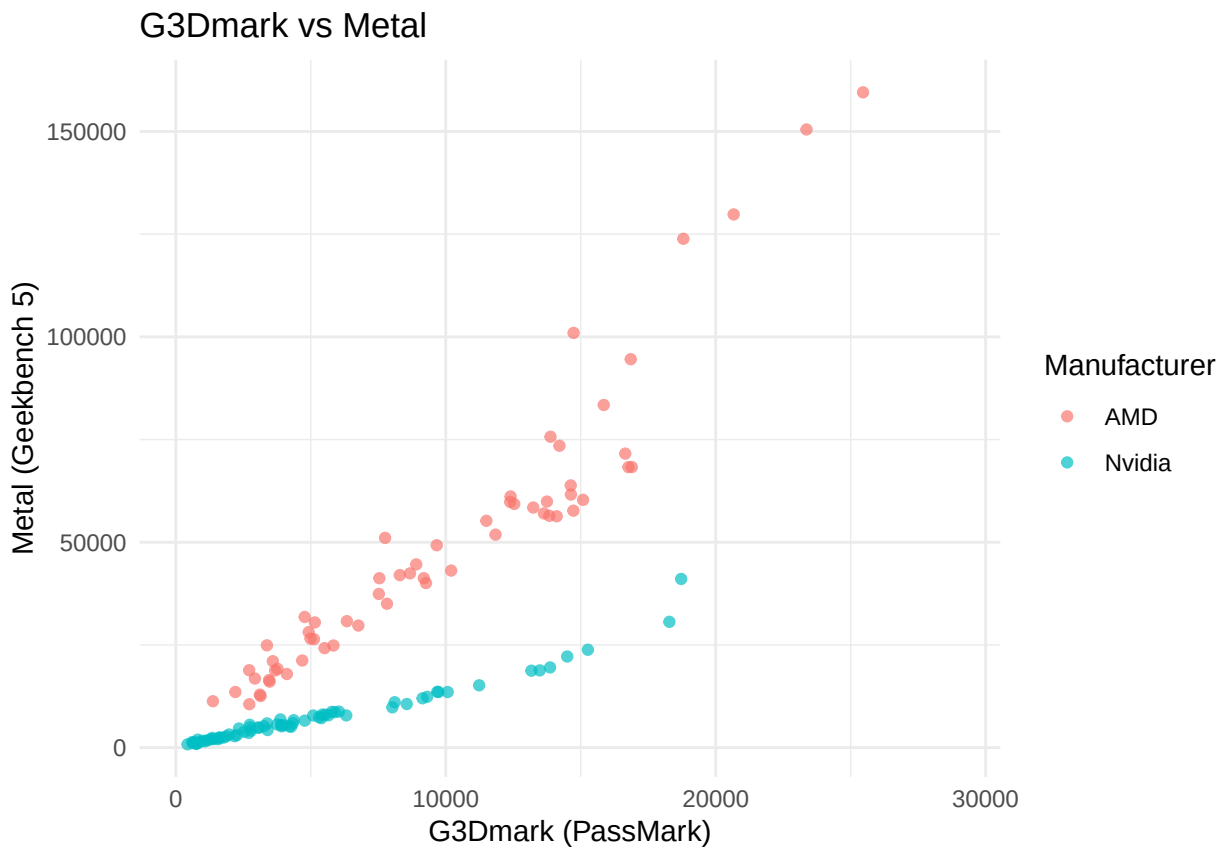
```
plot_scatter <- function(x_col, y_col, data) {
  p <- ggplot(data, aes_string(x = x_col, y = y_col, color = "Manufacturer")) +
    geom_point(alpha = 0.7) +
    theme_minimal() +
    labs(title = paste(x_col, "vs", y_col), x = paste(x_col, "(PassMark)"),
         y = paste(y_col, "(Geekbench 5)"))
  int_plot(p)
}

print(plot_scatter("G3Dmark", "OpenCL", merged_gpu))
```




```
print(plot_scatter("G3Dmark", "Vulkan", merged_gpu))
```



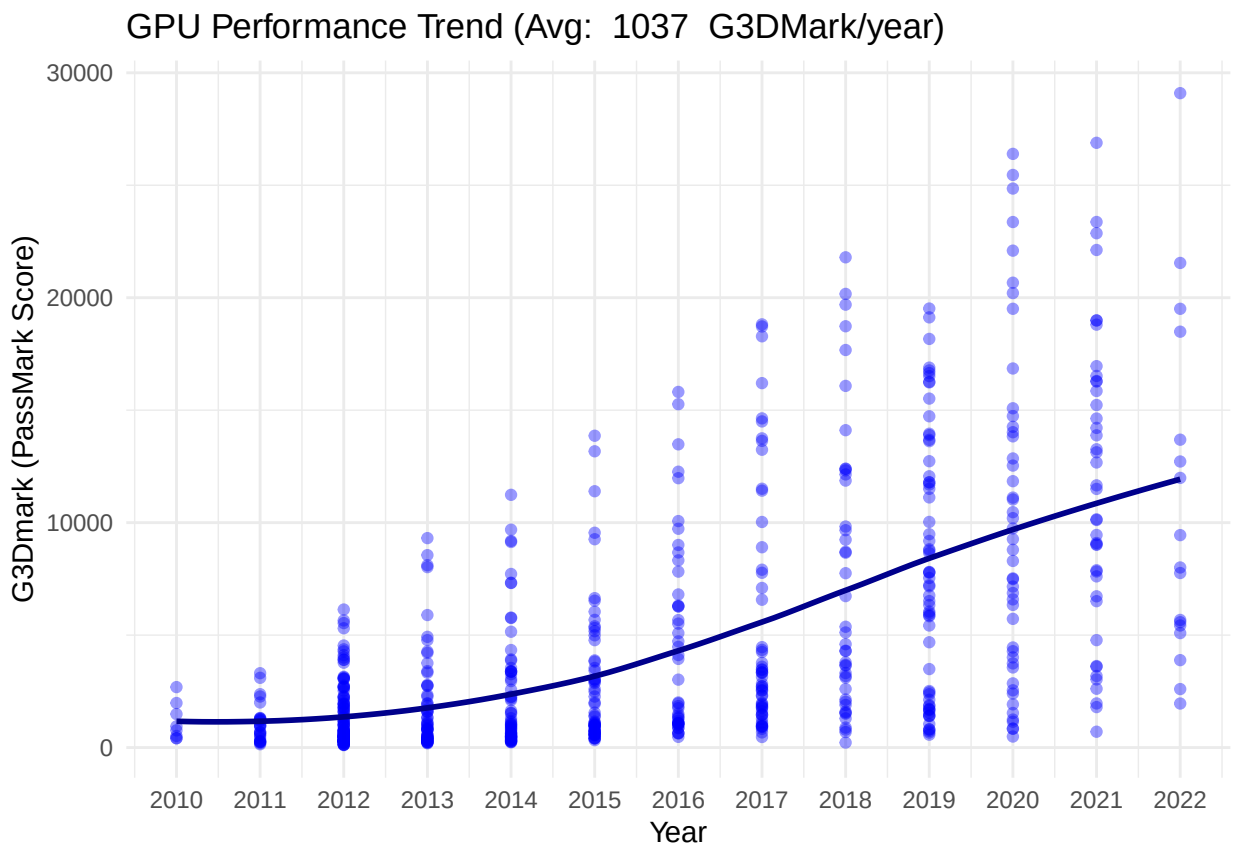

```
if (!is.null(merged_gpu$Metal)) {
  print(plot_scatter("G3Dmark", "Metal", merged_gpu))
}
```



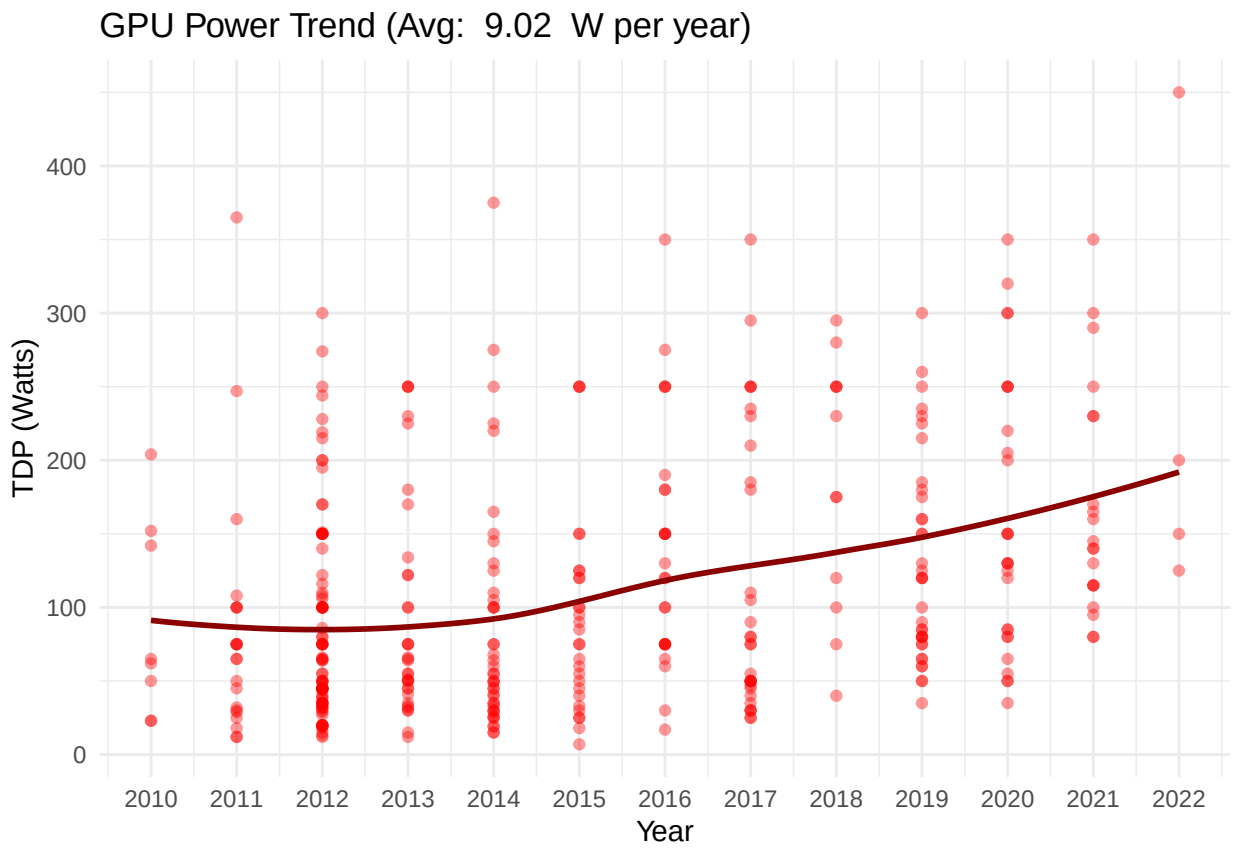
(b) Plotting different trends over time

```
# add PerfPerWatt column once
merged_gpu <- merged_gpu |> mutate(PerfPerWatt = G3Dmark / TDP)

get_slope <- function(data, y, manufacturer = NULL, digits = 0) {
  d <- data |> filter(!is.na(testDate), !is.na(.data[[y]]))

  if (!is.null(manufacturer)) {
    d <- d |> filter(Manufacturer == manufacturer)
  }

  fit <- lm(reformulate("testDate", response = y), data = d)
  round(coef(fit)["testDate"], digits)
}

slope_g3d <- get_slope(merged_gpu, "G3Dmark")
slope_tdp <- get_slope(merged_gpu, "TDP", digits = 2)
slope_eff <- get_slope(merged_gpu, "PerfPerWatt", digits = 2)
slope_amd <- get_slope(merged_gpu, "G3Dmark", manufacturer = "AMD")
slope_nv  <- get_slope(merged_gpu, "G3Dmark", manufacturer = "Nvidia")

# plots
plot_time_series(
  merged_gpu, "G3Dmark",
  paste("GPU Performance Trend (Avg: ", slope_g3d, " G3DMark/year)"),
  "G3Dmark (PassMark Score)", "blue", "darkblue"
```

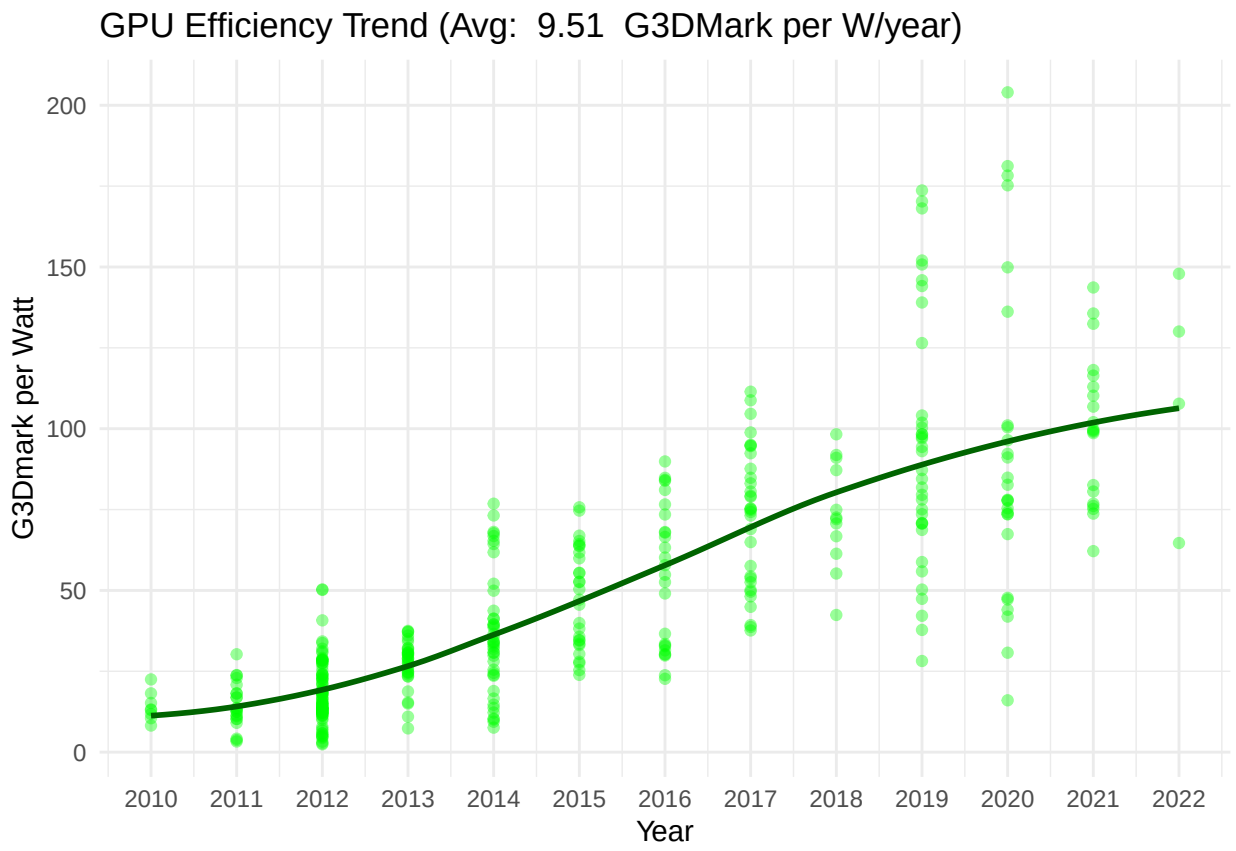
```
)
```



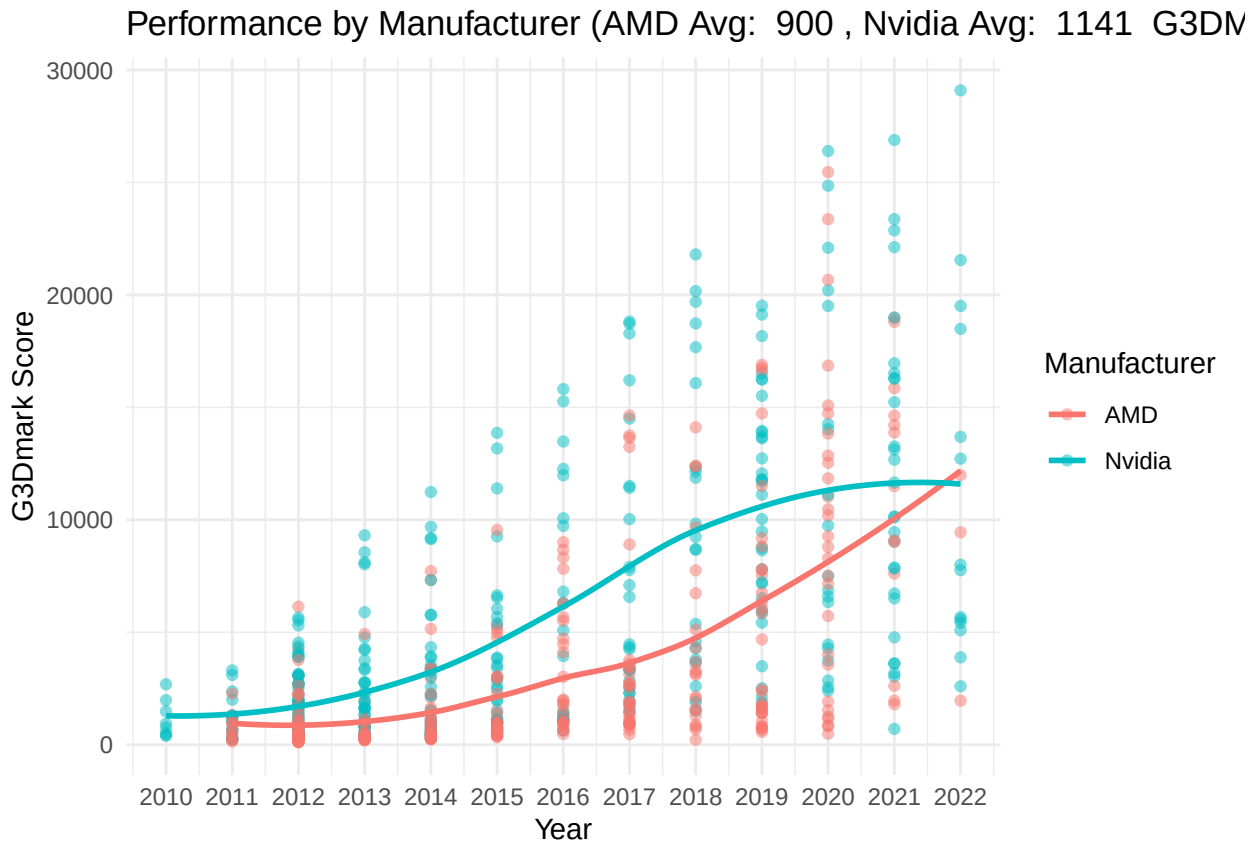

```
plot_time_series(
  merged_gpu, "TDP",
  paste("GPU Power Trend (Avg: ", slope_tdp, " W per year)"),
  "TDP (Watts)", "red", "darkred"
)
```




```
plot_time_series(
  merged_gpu, "PerfPerWatt",
  paste("GPU Efficiency Trend (Avg: ", slope_eff, " G3DMark per W/year)"),
  "G3Dmark per Watt", "green", "darkgreen"
)
```

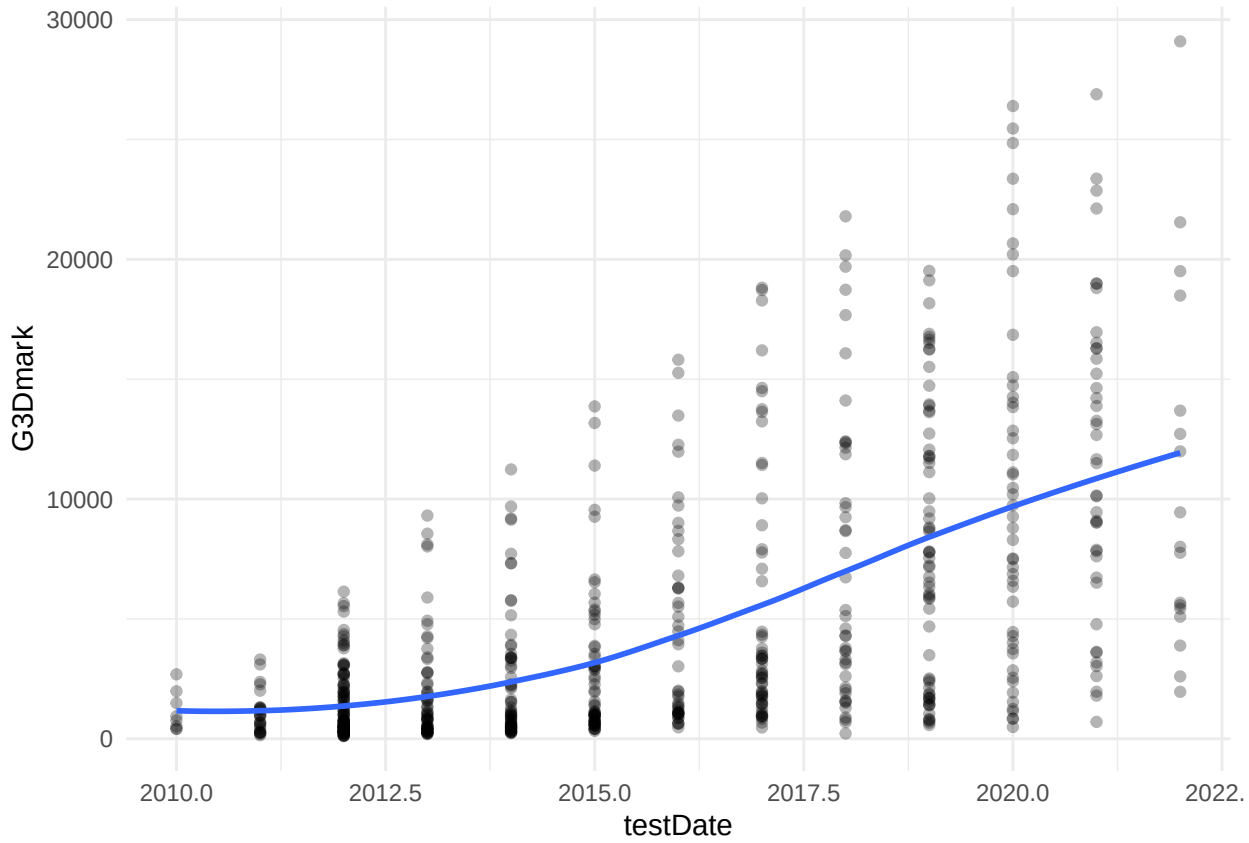



```
int_plot(
  ggplot(merged_gpu, aes(testDate, G3Dmark, color = Manufacturer)) +
    geom_point(alpha = 0.5) +
    geom_smooth(method = "loess", se = FALSE) +
    theme_minimal() +
    scale_x_continuous(breaks = seq(min(merged_gpu$testDate),
                                    max(merged_gpu$testDate), by = 1)) +
    labs(
      title = paste(
        "Performance by Manufacturer (AMD Avg: ",
        slope_amd, ", Nvidia Avg: ", slope_nv, " G3DMark/year)"),
      x = "Year", y = "G3Dmark Score")
)
```



(c) Plotting ALL GPU Trend Line

```
merged_gpu <- merged_gpu |> mutate(PerfPerWatt = G3Dmark / TDP)

int_plot(
  ggplot(merged_gpu, aes(testDate, G3Dmark)) +
    geom_point(alpha=0.3) +
    geom_smooth(se=FALSE) +
    theme_minimal()
)
```



Part 6: Exploring the data with TidyModels

(a) Split/train/test on the merged dataset for our tidymodels. We clean up the merged dataset, keep only the variables we care about, drop missing values, and split the data into training and testing sets so we can fairly evaluate our models. Also, create a common recipe to use for all models (except Random Forest). Dummy variables are needed for linear models, but random forests already know how to work with categories.

```
set.seed(123)
model_data <- merged_gpu |>
  select(G3Dmark, testDate, TDP, price, Manufacturer, category) |> drop_na()
split <- initial_split(model_data, prop = 0.8)
train <- training(split)
test <- testing(split)

common_recipe <- recipe(G3Dmark ~ ., data = train) |>
  step_dummy(all_nominal_predictors()) |> step_normalize(all_numeric_predictors())
```

Model 1: Linear Regression

(b) We start with a simple linear regression as a baseline model, using dummy variables for categorical features and checking how well it predicts GPU performance on the test set.

```
lin_spec <- linear_reg() |> set_engine("lm")
lin_wf <- workflow() |> add_recipe(common_recipe) |> add_model(lin_spec)
lin_fit <- lin_wf |> fit(data = train)
```

```
lin_preds <- predict(lin_fit, new_data = test) |> bind_cols(test)
metrics(lin_preds, truth = G3Dmark, estimate = .pred)
```

```
## # A tibble: 3 x 3
##   .metric .estimator .estimate
##   <chr>   <chr>          <dbl>
## 1 rmse    standard    2530.
## 2 rsq     standard       0.851
## 3 mae     standard    1927.
```

Model 2, 3: LASSO/Ridge Regression

(c) We try Ridge and LASSO regression, tuning the regularization strength with cross-validation to control overfitting and compare their predictive performance.

```
folds <- vfold_cv(train, v=10)
lambda_grid <- grid_regular(penalty(), levels= 30)

ridge_spec <-linear_reg(mixture=0, penalty =tune()) |> set_engine("glmnet")
lasso_spec <- linear_reg(mixture=1, penalty= tune()) |> set_engine("glmnet")

ridge_wf <- workflow() |> add_recipe(common_recipe) |> add_model(ridge_spec)
lasso_wf <- workflow() |> add_recipe(common_recipe) |> add_model(lasso_spec)

ridge_t <- tune_grid(ridge_wf, resamples=folds, grid=lambda_grid)
lasso_t <- tune_grid(lasso_wf, resamples=folds, grid=lambda_grid)

best_ridge <- select_best(ridge_t, metric="rmse")
best_lasso <- select_best(lasso_t, metric="rmse")

ridge_final_fit <- ridge_wf |> finalize_workflow(best_ridge) |> fit(data=train)
lasso_final_fit <- lasso_wf |> finalize_workflow(best_lasso) |> fit(data=train)

ridge_preds <- predict(ridge_final_fit, new_data=test) |> bind_cols(test)
lasso_preds <- predict(lasso_final_fit, new_data=test) |> bind_cols(test)

ridge_metrics <- ridge_preds |> metrics(truth=G3Dmark, estimate=.pred)
lasso_metrics <- lasso_preds |> metrics(truth=G3Dmark, estimate=.pred)

best_ridge
```

```
## # A tibble: 1 x 2
##        penalty .config
##          <dbl> <chr>
## 1 0.0000000001 pre0_mod01_post0
```

```
best_lasso
```

```
## # A tibble: 1 x 2
##        penalty .config
##          <dbl> <chr>
## 1 0.0000000001 pre0_mod01_post0
```

```
ridge_metrics
```

```
## # A tibble: 3 x 3
```

```
##   .metric .estimator .estimate
##   <chr>   <chr>          <dbl>
## 1 rmse    standard    2360.
## 2 rsq     standard       0.858
## 3 mae     standard    1751.
```

```
lasso_metrics
```

```
## # A tibble: 3 x 3
##   .metric .estimator .estimate
##   <chr>   <chr>          <dbl>
## 1 rmse    standard    2522.
## 2 rsq     standard       0.850
## 3 mae     standard    1922.
```

Model 4: KNN

(d) We use KNN model, tuning the number of neighbors with cross-validation to see how a distance-based approach performs on this dataset.

```
knn_folds <- vfold_cv(train, v = 10)
knn_model <- nearest_neighbor(neighbors = tune()) |>
  set_engine("kknn") |> set_mode("regression")
knn_wf <- workflow() |> add_recipe(common_recipe) |> add_model(knn_model)

knn_tuned <- tune_grid(knn_wf, resamples = knn_folds,
                       grid = tibble(neighbors = c(1, 3, 5, 10, 20, 50)),
                       metrics = metric_set(rmse, rsq, mae))

show_best(knn_tuned, metric = "rmse")
```

```
## # A tibble: 5 x 7
##   neighbors .metric .estimator  mean     n std_err .config
##       <dbl> <chr>   <chr>      <dbl> <int>   <dbl> <chr>
## 1         5 rmse    standard   2160.    10    270. pre0_mod3_post0
## 2         3 rmse    standard   2219.    10    254. pre0_mod2_post0
## 3        10 rmse    standard   2266.    10    262. pre0_mod4_post0
## 4        20 rmse    standard   2463.    10    252. pre0_mod5_post0
## 5         1 rmse    standard   2629.    10    223. pre0_mod1_post0
```

```
best_k <- select_best(knn_tuned, metric = "rmse")

knn_final <- finalize_workflow(knn_wf, best_k) |> fit(data = train)
knn_preds <- predict(knn_final, new_data = test) |> bind_cols(test)
metrics(knn_preds, truth = G3Dmark, estimate = .pred)
```

```
## # A tibble: 3 x 3
##   .metric .estimator .estimate
##   <chr>   <chr>          <dbl>
## 1 rmse    standard    1709.
## 2 rsq     standard       0.918
## 3 mae     standard    1172.
```

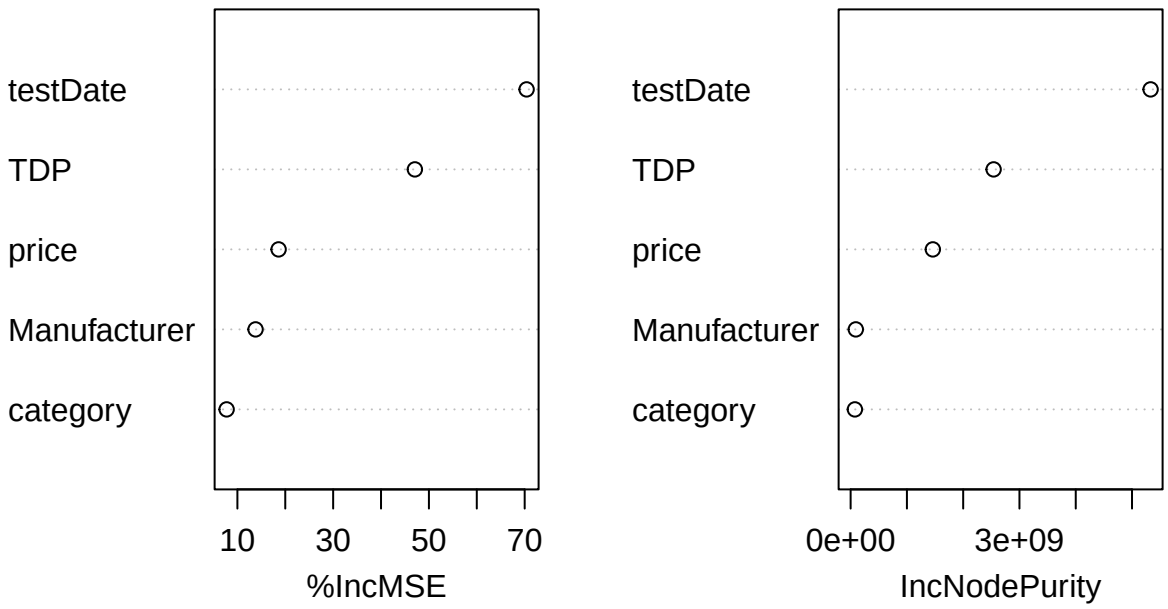
Model 5: Random Forest

(e) Finally, we fit a random forest model to see which features matter most.

```r
rf_recipe <- recipe(G3Dmark ~ ., data = train) |>
  step_normalize(all_numeric_predictors()) # we do not want dummy for rf.

rf_spec <- rand_forest(mtry = tune(), trees = tune(), min_n = tune()) |>
  set_engine("randomForest", importance = TRUE) |>
  set_mode("regression")

rf_wf <- workflow() |> add_recipe(rf_recipe) |> add_model(rf_spec)
folds <- vfold_cv(train, v = 10)

rf_params <- parameters(finalize(mtry(), train), trees(range = c(200, 600)),
  min_n(range = c(2, 25)))

rf_grid <- grid_regular(rf_params, levels = 4)

rf_tune <- tune_grid(rf_wf, resamples = folds, grid = rf_grid,
  metrics = metric_set(rmse, rsq, mae))

best_rf <- select_best(rf_tune, metric = "rmse")
final_rf <- finalize_workflow(rf_wf, best_rf)
rf_fit <- final_rf |> fit(data = train)
rf_preds <- predict(rf_fit, test) |> bind_cols(test)
rf_metrics <- metrics(rf_preds, truth = G3Dmark, estimate = .pred)
rf_metrics
```

```
## # A tibble: 3 x 3
##   .metric .estimator .estimate
##   <chr>   <chr>          <dbl>
## 1 rmse    standard    1466.
## 2 rsq     standard       0.946
## 3 mae     standard     998.
```

```r
rf_fit |> extract_fit_parsnip() |> pluck("fit") |> varImpPlot()
```


pluck(extract_fit_parsnip(rf_fit), "fit")
testDate
TDP
price
Manufacturer
category
10
30
50
70
%IncMSE
testDate
TDP
price
Manufacturer
category
0e+00
3e+09
IncNodePurity