Assignment 1: Exploratory data analysis


```
begin
    using PlutoUI
    using PlutoTest
    using Random, Statistics, LinearAlgebra, Dates
    using Distributions, StatsBase
    using DataFrames , XLSX
    using CairoMakie
    using Chain
    using TidierData
    using FileIO
    using Polynomials
    using Match
    using Latexify
    const colors = Makie.wong_colors()
end;
```

Table of Contents

Load the Melbourne Housing data set

This dataset was downloaded from Kaggle (`https://www.kaggle.com/datasets/anthonypino/melbourne-housing-market`). The data was scraped from publicly available results posted every week from Domain.com.au. The dataset includes Address, Type of Real estate, Suburb, Method of Selling, Rooms, Price, Real Estate Agent, Date of Sale and distance from C.B.D.

Load into a dataframe

df =

	Suburb	Address	Rooms	Type	Method	SellerG	Da
1	"Abbotsford"	"68 Studley St"	2	"h"	"SS"	"Jellis"	2016-
2	"Airport West"	"154 Halsey Rd"	3	"t"	"PI"	"Nelson"	2016-
3	"Albert Park"	"105 Kerferd Rd"	2	"h"	"S"	"hockingstuart"	2016-
4	"Albert Park"	"85 Richardson St"	2	"h"	"S"	"Thomson"	2016-
5	"Alphington"	"30 Austin St"	3	"h"	"SN"	"McGrath"	2016-
6	"Alphington"	"6 Smith St"	4	"h"	"S"	"Brace"	2016-
7	"Alphington"	"5/6 Yarralea St"	3	"h"	"S"	"Jellis"	2016-
8	"Altona"	"158 Queen St"	3	"h"	"VB"	"Greg"	2016-
9	"Altona North"	"1 Beuron Rd"	3	"h"	"SP"	"Williams"	2016-
10	"Altona North"	"45 Hearn St"	5	"h"	"S"	"FN"	2016-
more							
34857	"Westmeadows"	"42 Pascoe St"	4	"h"	"S"	"Barry"	2017-

```
1 df = DataFrame(XLSX.readtable("Melbourne_housing.xlsx", 1))
```

(1) Calculate the number of missing observations for each df column

Store the result in `dfMissing`. Use column names `Column` and `MissingValues`. It's important to use these names to facilitate grading. See the test below.

Tip

Use `count(ismissing, df[!, col])` to return the number of missing values in the column `col`.

Your code here

```
begin
    #New data frame for columns and their missing values
    dfMissing = DataFrame(Column = String[], MissingValues = Int[])
        #Loop through each column of original dataframe
        for col in names(df)
            missing_count = count(ismissing, df[!, col])
            #Add a new row when missing content is found
            push!(dfMissing, (col, missing_count))
        end
end
```

```
sum(dfMissing.MissingValues) == 100954
```

```
@test sum(dfMissing.MissingValues) == 100954
```

Generate a summary table of house types

(2) Re-Calculate the table of house types

With fewer missing observations excluded, the overall number of records should now be larger.

Tip

Use the `@chain` macro to extract and summarise the data from `df`.

Visit the `TidierData.jl` GitHub repo to see the macros for the various data operations you can perform (e.g., `@group_by`, `@mutate`, `@summarize`).

Your code here

`houseType =`

	Type	n
1	"h"	23980
2	"t"	3580
3	"u"	7297

```
houseType = @chain df begin
    #Select Type column
    @select(Type)
    @group_by(Type)
    #counting number of occurances
    @summarize(n = n())
end
```

```
sum(houseType.n) == 34857
```

```
1 @test sum(houseType.n) == 34857
```

(3) Create a dataframe for subsequent plots

Include only the variables you will need from the original df. You may also have to create one or two new ones to help with the plots. Note that the `Colour` column is used to provide a numerical code for `Region` for graphing purposes.

Include: `Regionname, Price, Rooms, Latitude, Longtitude, Distance, Type`.

Create: `Region, PriceK, TypeFull, Colour`

Select records for just the five metropolitan regions.

Tip

Be sure to name the new dataframe `dfPlot` to ensure the test below is passed.

Create `dfPlot`, a subset of data for plotting

Your code here

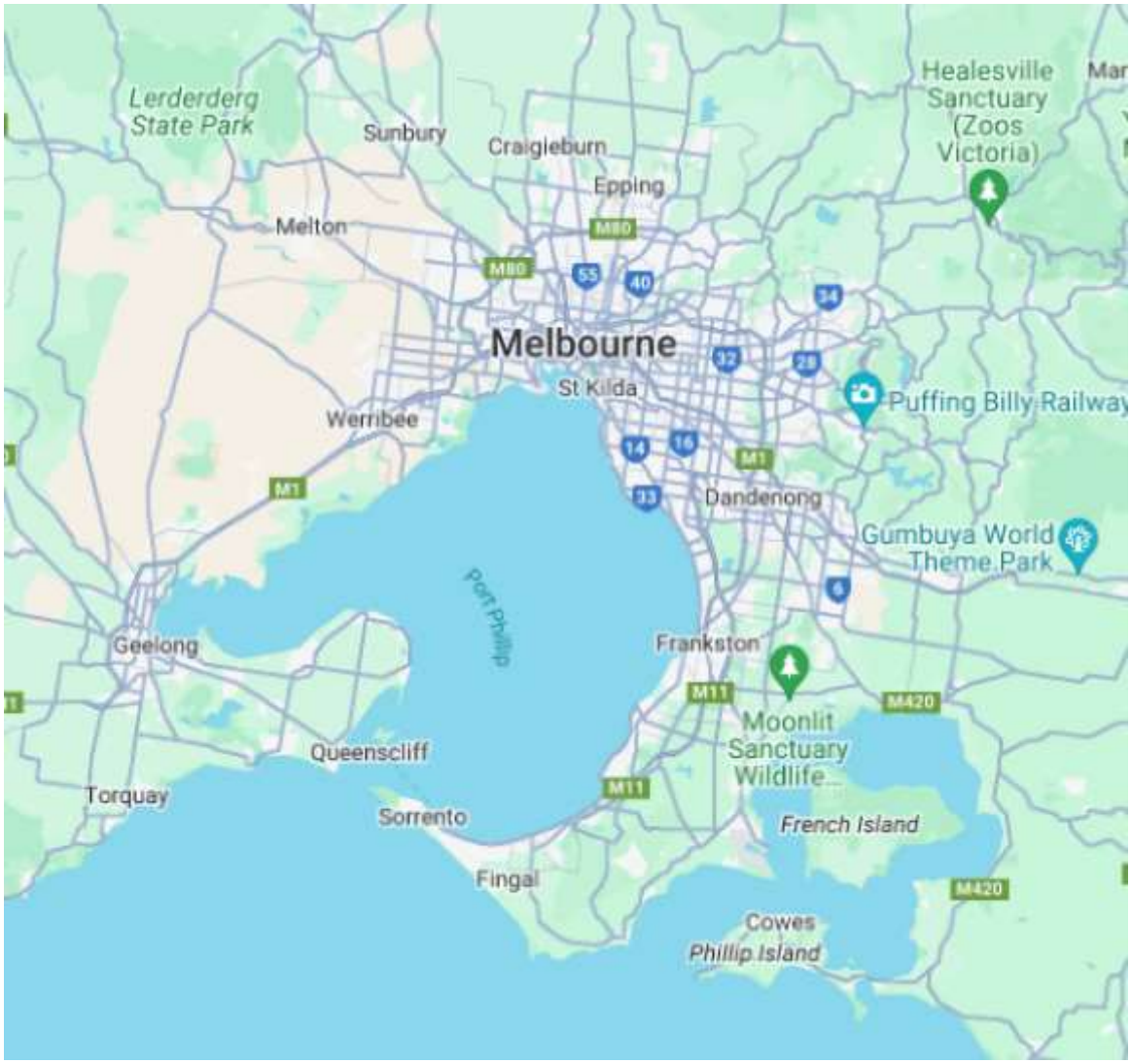
	Region	PriceK	TypeFull	Colour	Rooms	Latitude	Longtitude	Distance
1	"WM"	840.0	"Townhouse"	"2"	3	-37.718	144.878	13.5
2	"SM"	1275.0	"House"	"3"	2	-37.8459	144.956	3.3
3	"SM"	1455.0	"House"	"3"	2	-37.845	144.954	3.3
4	"NM"	2000.0	"House"	"1"	4	-37.7707	145.032	6.4
5	"NM"	1110.0	"House"	"1"	3	-37.7854	145.032	6.4
6	"WM"	520.0	"House"	"2"	3	-37.87	144.825	13.8
7	"WM"	1085.0	"House"	"2"	5	-37.8388	144.857	11.1
8	"WM"	781.0	"House"	"2"	3	-37.8291	144.835	11.1
9	"SM"	599.0	"Unit"	"3"	2	-37.8543	145.026	6.3
10	"WM"	455.0	"Unit"	"2"	2	-37.7756	144.917	5.9
more								
20622	"NM"	791.0	"House"	"1"	4	-37.6763	144.894	16.5

```
begin
    dfPlot = @chain df begin
        @select(Regionname, Price, Rooms, Latitude, Longtitude, Distance, Type)
        @filter(Regionname in ["Northern Metropolitan", "Western Metropolitan",
        "Southern Metropolitan", "Eastern Metropolitan", "South-Eastern
        Metropolitan"])
        @drop_missing()
        #Create new columns to replace old
        @mutate(
            Region = replace(Regionname,
            "Northern Metropolitan" => "NM",
            "Western Metropolitan" => "WM",
            "Southern Metropolitan" => "SM",
            "Eastern Metropolitan" => "EM",
            "South-Eastern Metropolitan" => "SEM"),
            PriceK = Price/1000,
            TypeFull= replace(Type,
            "h" => "House",
            "t" => "Townhouse",
            "u" => "Unit"),
            Colour = replace(Regionname,
            "Northern Metropolitan" => 1,
            "Western Metropolitan" => 2,
            "Southern Metropolitan" => 3,
            "Eastern Metropolitan" => 4,
            "South-Eastern Metropolitan" => 5))
    #Select new columns for output
    @select(Region, PriceK, TypeFull, Colour, Rooms, Latitude, Longtitude, Distance)
    end
end
```

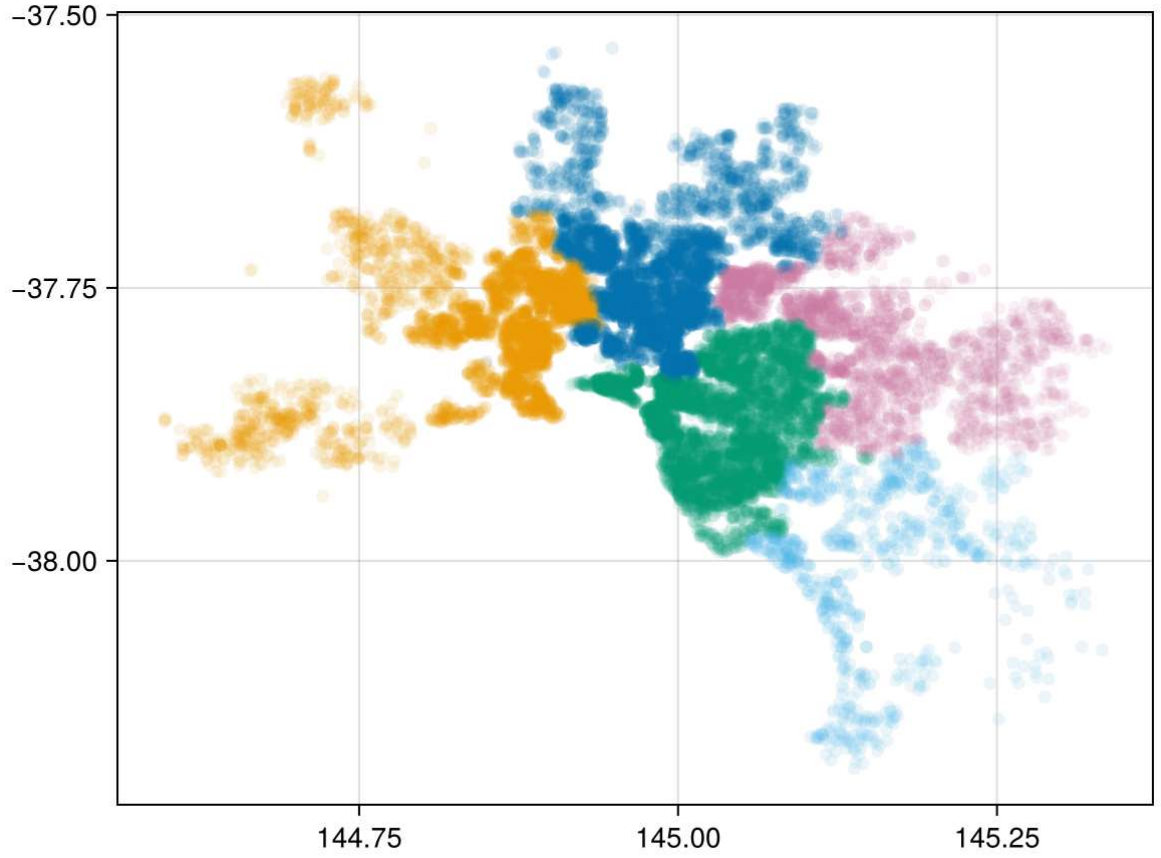
```
nrow(dfPlot) == 20622
```

```
1 @test nrow(dfPlot) == 20622
```

Geographical distribution and density of housing



This error will go once you define `dfPlot`.




```
let
    xs = dfPlot.Longtitude
    ys = dfPlot.Latitude
    points = Point2f.(xs, ys)
    scatter(points, alpha=0.1,
        color = colors[parse.(Int, dfPlot.Colour)])
end
```

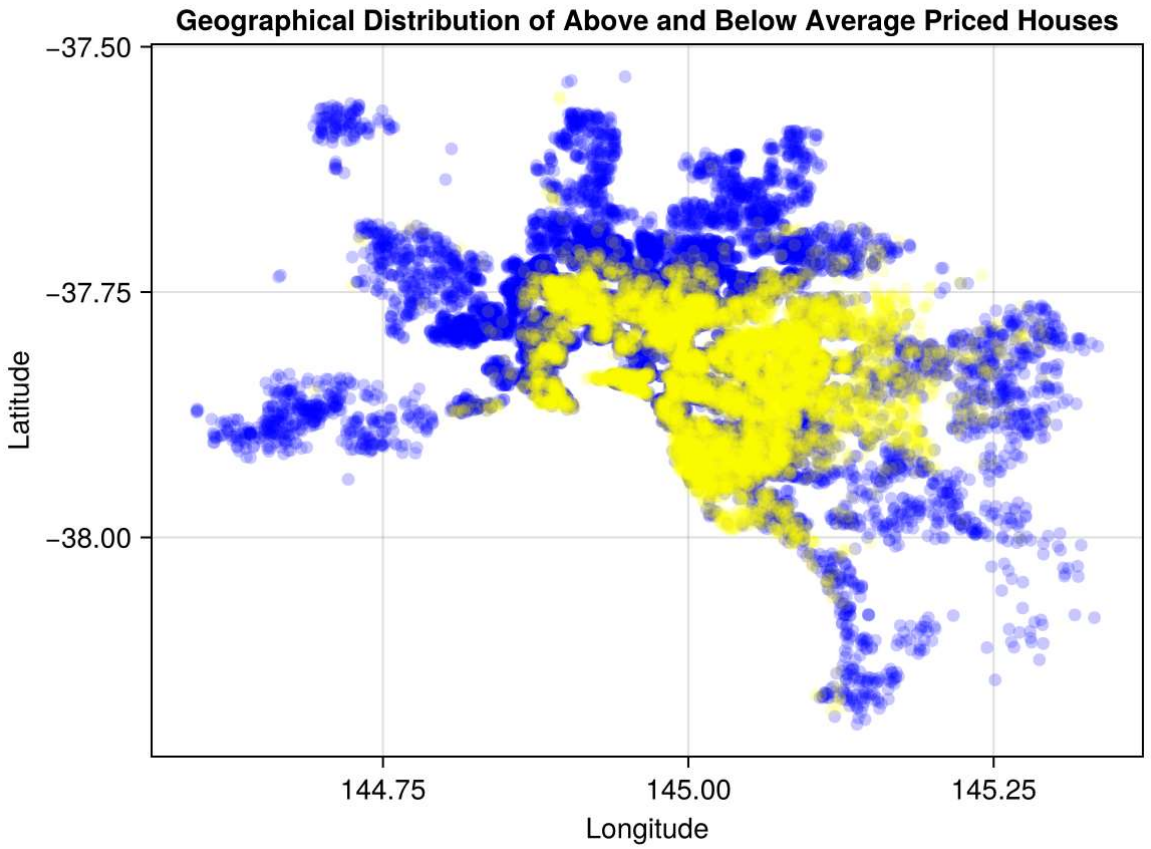
(4) Make an additional geographic plot of above- and below-average priced houses

Discuss the patterns you find.

Tip

You can use an expression like `avgPrice = mean(dfPlot.PriceK)` to calculate the average house price in the dataframe.

Your code here




```
begin
    #Calculate average
    avgPrice = mean(dfPlot.PriceK)
    #Create new subsets for those above average and below
    aboveAvg = dfPlot[dfPlot.PriceK .> avgPrice, :]
    belowAvg = dfPlot[dfPlot.PriceK .<= avgPrice, :]

    #Create points for subsets by their longtitude and latitude
    points_above = Point2f.(aboveAvg.Longtitude, aboveAvg.Latitude)
    points_below = Point2f.(belowAvg.Longtitude, belowAvg.Latitude)

let
    fig = Figure()
    ax = Axis(fig[1, 1], title="Geographical Distribution of Above and Below Average Priced Houses")

    #Plot points for those below
    scatter!(ax, points_below, label="Below Average Price", color=:blue, alpha=0.2)
    #Plot points for those above
    scatter!(ax, points_above, label="Above Average Price", color=:yellow, alpha=0.155)


    ax.xlabel = "Longitude"
    ax.ylabel = "Latitude"

    #Return the figure
    fig
    end
```

```
28 end
```

From the graph I noticed above-averaged priced homes clustered in areas closer to the city center and along the waterfront. As we move away from these two areas, the distribution shifts with below-averaged priced homes clustering farther out. Even though this clustering of below averaged homes starts away from the city center, there are still a few below average priced scattered in the more city areas

(5) Suggest why the South Metro distribution tends to be bimodal

Support your conjecture with data and/or graphs.

From the Melbourne House Prices by Region graph below, we can see that the Southern Metro area exhibits a bimodal distribution, with two distinct peaks: one at the lower end of the price spectrum (~500) and another at the higher end (~1500).

The Geographical Distribution and Density Housing graph shows that the Southern Metro area is situated between the city center and the coastline. Additionally,the Above and Below Average Cost graph shows that most of the housing in this area is on the above average side, yet there still remains a significant number of below-average priced options.

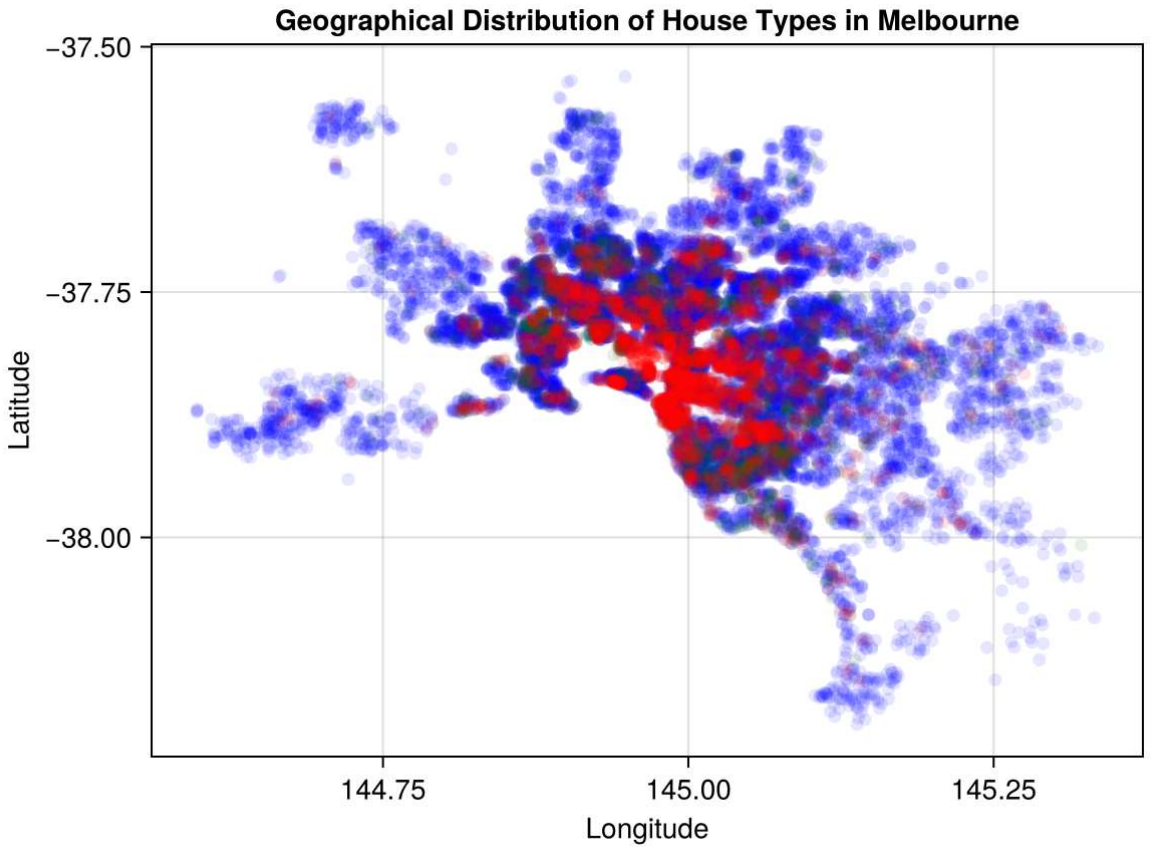
My hypothesis for this phenomenon is influenced by the fact that the models mentioned above don't take into consideration the different house types. Like I said before this area is located right by the city which typically tends to have more apartments (units) and townhouses, that tend to cost less due to their limited space.

To see if this hypothesis was right I organized the Southern Metro area by Type below:

	TypeFull	Count	AveragePriceK
1	"Townhouse"	55	889.368
2	"House"	896	933.828
3	"Unit"	78	551.971

From this updated dataset we can see that units cost significantly less than houses and townhouses.

To explore this further, I created the graph below, which shows the geographical distribution of different housing types across all areas of Melbourne (red = units, blue = houses, green = townhouses).

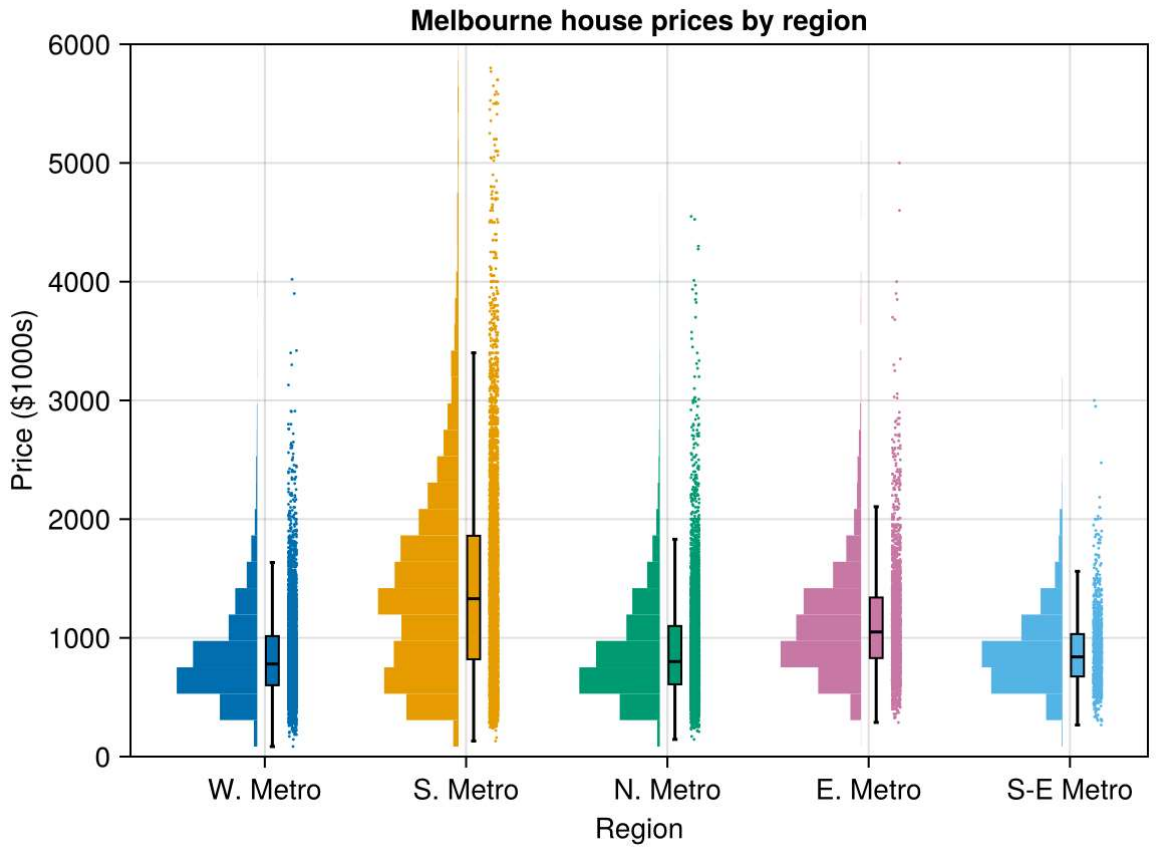


Most of these units are located in the part of the Southern Metro area closest to the city center.

In conclusion the SM's closeness to the city center causes a dip towards the cheaper end of housing because of its trend of apartments. Conversely, the higher-end prices are likely due to the presence of more expensive housing near the shoreline.

House price patterns by main city districts

This error will go once you define `dfPlot`.




```
let
    category_labels = dfPlot.Region
    data_array = dfPlot.PriceK

    rainclouds(category_labels, data_array;
            axis = (; xlabel = "Region",
                ylabel = "Price (\$1000s)",
                limits = (nothing, (0,6000)),
                title = "Melbourne house prices by region",
                xticks= (1:5, ["W. Metro", "S. Metro", "N. Metro", "E. Metro", "S-E
                Metro"])),
        plot_boxplots = true, cloud_width=0.5, clouds=hist, hist_bins=50,
        color = colors[indexin(category_labels, unique(category_labels))])

end
```

Relationship between price and proximity to the CBD

(6) Create a dataframe of CBD distance and average price

You should use `trunc()` to return the nearest integral value of the `Distance` measure in `dfPlot`, then group and summarise price data within each of the resulting kilometer blocks.

Be sure to name the new dataframe `distPrice` and the variables `Dist`, `n`, and `AvgPrice` for test purposes.

Your code here

	Dist	n	AvgPrice
19	16	589	1038.82
20	20	456	694.659
21	34	57	604.648
22	24	122	717.068
23	22	90	836.426
24	18	445	731.071
25	19	196	739.292
26	38	111	773.184
27	32	9	713.722
28	21	209	1066.74
29	35	31	747.597
30	31	134	576.498
31	27	107	745.518
32	28	39	657.256
33	0	22	586.682
34	26	16	745.562
35	33	6	586.333

```
begin
    distPrice = @chain dfPlot begin
        #return nearest integral value
        @mutate(Dist = trunc.(Distance))
        @mutate(Dist = Int.(Dist))
        @group_by(Dist)
        @summarize(
            n = n(),
            AvgPrice = mean(PriceK)
        )
    end
end
```

(7) Graph and fit a polynomial curve to these data

Use the `Polynomials.jl` library. Provide correlation and goodness-of-fit measures (R^2 and RMSE) on the graph.

Bind a slider to the variable controlling the polynomial order to show different fits.

Discuss what polynomial order best describes the relationship. Note that this does not necessarily mean gives the best R^2.

Tip

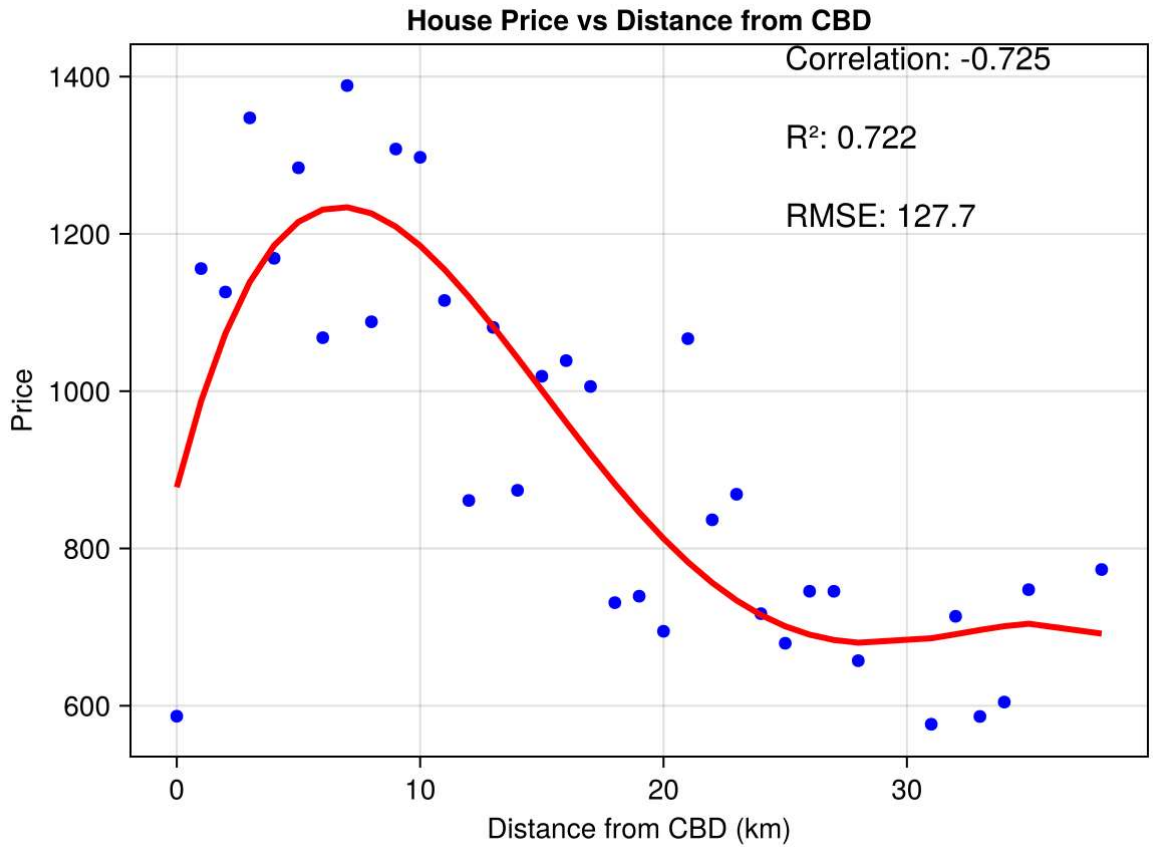
Use `corr()` from the `Statistics` library to calculate the correlation between distance and average price.

The R^2 goodness of fit measure is calculated using: $R^2 = 1 - \frac{\sum (y_{\text{actual}} - y_{\text{pred}})^2}{\sum (y_{\text{actual}} - \bar{y})^2}$

The RMSE measure is calculated using: $\text{RMSE} = \sqrt{\frac{\sum (y_{\text{actual}} - y_{\text{pred}})^2}{N}}$

Your code here

```
poly_order = 2;
```




```
1
2  let
3      xs = distPrice.Dist
4      ys = distPrice.AvgPrice
5
6      function fit_polynomial(xs, ys, degree)
7          #Creating the fitted line
8          p = Polynomials.fit(xs, ys, degree)
9          #Predict values using fitted polynomial
10         ys_pred = p.(xs)
11
12         # Calculate R², RMSE, and correlation
13         R2 = 1 - sum((ys .- ys_pred).^2) / sum((ys .- mean(ys)).^2)
14         rmse = sqrt(mean((ys .- ys_pred).^2))
15         correlation = cor(xs, ys)
16
17         fig = Figure()
18         ax = Axis(fig[1, 1],
19             title="House Price vs Distance from CBD",
20             xlabel="Distance from CBD (km)",
21             ylabel="Price"
22         )
23
24         #Original data points (distance & price)
25         scatter!(ax, xs, ys, color=:blue)
26
27         # Sort xs and ys_pred for proper line plotting
28         sorted_indices = sortperm(xs)
29         sorted_xs = xs[sorted_indices]
```

```
        sorted_ys_pred = ys_pred[sorted_indices]

        #Plot fitted line
        lines!(ax, sorted_xs, sorted_ys_pred, color=:red, linewidth=3)

        # Set custom y-ticks
        y_ticks = 600:200:1400
        ax.yticks = y_ticks

        text!(ax, 25, 1400, text="Correlation: $(round(correlation, digits=3))",
fontsize=16)
        text!(ax, 25, 1300, text="R²: $(round(R2, digits=3))", fontsize=16)
        text!(ax, 25, 1200, text="RMSE: $(round(rmse, digits=1))", fontsize=16)

        return fig
    end

    fit_polynomial(xs, ys, poly_order) #function call
end
```

```
@bind poly_order PlutoUI.Slider(1:1:5, default=3)
```

round(cor(distPrice.Dist, distPrice.AvgPrice), digits = 3) == -0.725

```
@test round(cor(distPrice.Dist, distPrice.AvgPrice),digits=3) == -0.725
```

The degree 3 polynomial best captures the underlying relationship between house prices and distance from the CBD. While the degree 5 polynomial provides the best statistical fit—reflected by a higher R^2 and lower RMSE—it runs the risk of overfitting. Overfitting occurs when a model is too finely tuned to the specific dataset it was trained on, leading to poor performance on new data.

In contrast, the degree 3 polynomial provides a better balance. It accurately reflects the general trend in the data without being overly sensitive to small variations, making it more likely to generalize well to additional data.