Seoul Bike Sharing Demand

This notebook demonstrates how to use GLM to analyze the Seoul Bike Sharing Demand data set.

Data definitions

The following definitions are captured from the **UCI Datasets Archive page**.

- Date: year-month-day
- Rented Bike count: Count of bikes rented at each hour
- Hour: Hour of he day
- Temperature: Temperature in Celsius
- Humidity: %
- Windspeed: m/s
- Visibility: 10m
- Dew point temperature: Celsius
- Solar radiation: MJ/m2
- Rainfall: mm
- Snowfall: cm
- Seasons: Winter, Spring, Summer, Autumn
- Holiday: Holiday/No holiday
- Functional Day: NoFunc(Non Functional Hours), Fun(Functional hours)

As the dataset is targeted for bike sharing demand, the key metrics should be the Rented Bike Count field. The rest of the columns are variables that may be correlated to bike sharing demand.

Taking a quick peek at the variables, it makes sense that they may directly or indirectly affect people's decision of renting a bike. For example, when it's very hot or very cold, then I may take on some other transportation. Likewise, humidity, solar radiation, visibility, rainfall, and snowfall probably play a part as well.

Most variables are continuous variables. The last three columns seasons, holiday, and functional day are discrete variables. As for the hour of day variable, we may want to consider that as discrete because increasing the value (later in the day) does not always affect the response variable (rented bike count) in the same direction.

Reading data

```
begin
    using CSV
    using DataFrames
    using Plots
    using StatsPlots
    using Pipe: @pipe
    using GLM
    using StatsBase
    using Statistics
    using Dates
end
```

```
file = "https://archive.ics.uci.edu/ml/machine-learning-databases/00560/SeoulBikeData.csv"
```

```
file = "https://archive.ics.uci.edu/ml/machine-learning-databases/00560/SeoulBikeData.csv"
```

```
df = DataFrame(CSV.File(download(file); dateformats = Dict(:Date => "dd/mm/yyyy")));
```

Data wrangling

Let's quickly examine the data frame and its columns.

14 rows × 6 columns

	variable Symbol	eltype DataType	min Any	mean Union…	max Any	nmissing Nothing
1	Date	Date	2017-12-01		2018-11-30	
2	Rented Bike Count	Int64	0	704.602	3556	
3	Hour	Int64	0	11.5	23	
4	Temperature(\xb0C)	Float64	-17.8	12.8829	39.4	
5	Humidity(%)	Int64	0	58.2263	98	
6	Wind speed (m/s)	Float64	0.0	1.72491	7.4	
7	Visibility (10m)	Int64	27	1436.83	2000	
8	Dew point temperature(\xb0C)	Float64	-30.6	4.07381	27.2	
9	Solar Radiation (MJ/m2)	Float64	0.0	0.569111	3.52	
10	Rainfall(mm)	Float64	0.0	0.148687	35.0	
11	Snowfall (cm)	Float64	0.0	0.0750685	8.8	
12	Seasons	String	Autumn		Winter	

	variable Symbol	eltype DataType	min Any	mean Union...	max Any	nmissing Nothing
13	Holiday	String	Holiday		No Holiday	
14	Functioning Day	String	No		Yes	

```
describe(df, :eltype, :min, :mean, :max, :nmissing)
```

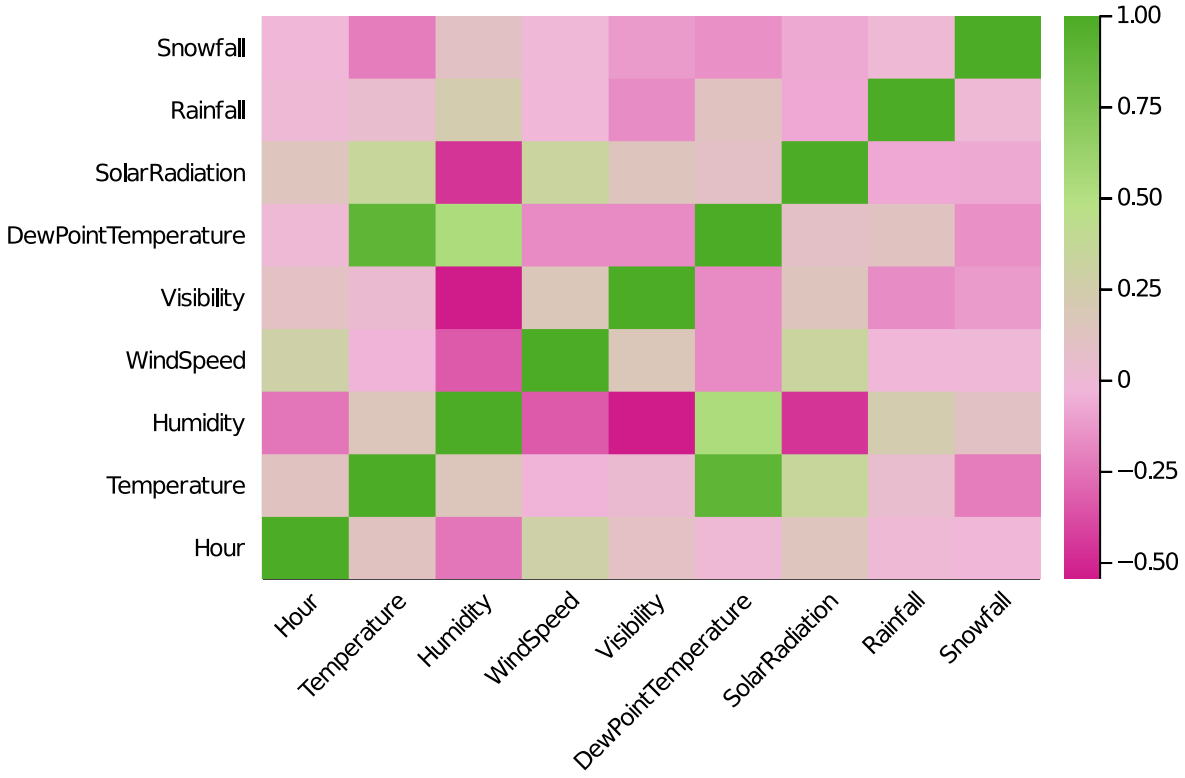
It looks pretty clean in general but let's rename some columns so they can be referenced more easily in the code below.

```
rename!(df,
    2 => :RentedBikeCount,
    4 => :Temperature, 5 => :Humidity, 6 => :WindSpeed, 7 => :Visibility,
    8 => :DewPointTemperature, 9 => :SolarRadiation, 10 => :Rainfall, 11 => :Snowfall,
    14 => :FunctioningDay
);
```

Correlation analysis

```
String["Date", "RentedBikeCount", "Hour", "Temperature", "Humidity", "WindSpeed", "
```

```
names(df)
```



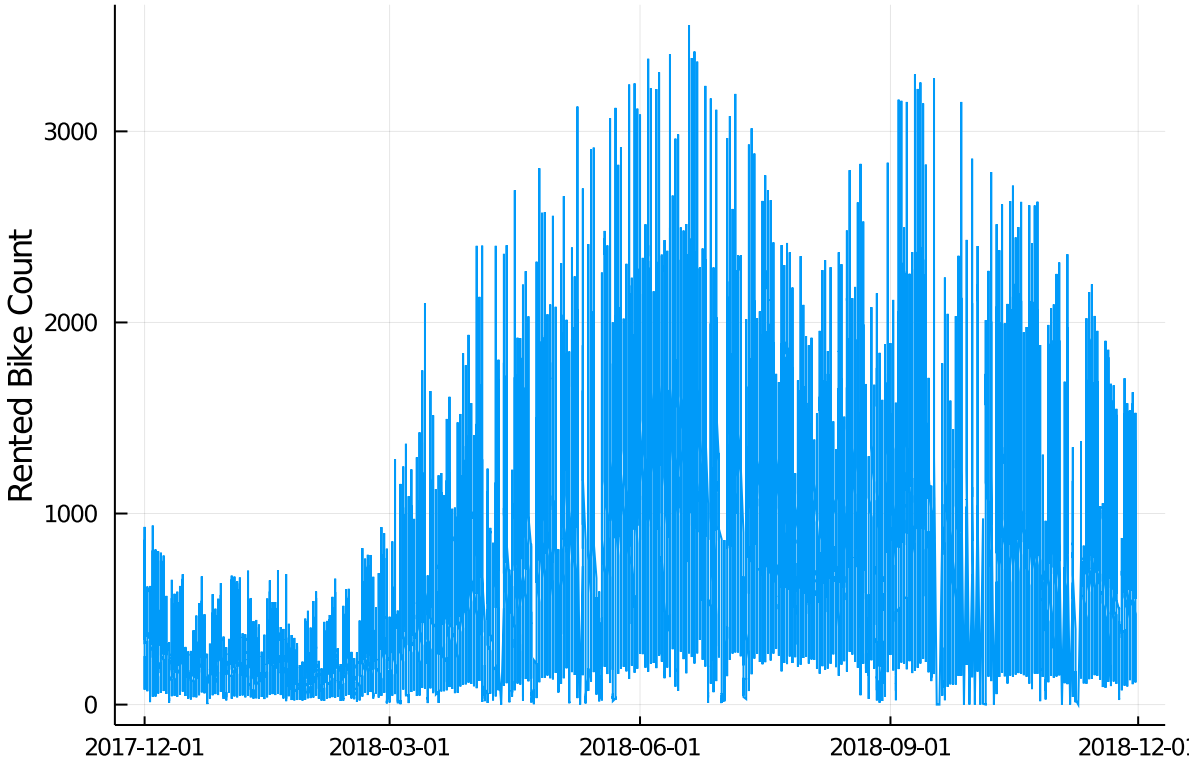

```
let
    cols = 3:11
    colnames = names(df)[cols]
```

```
    Σ = cor(Matrix(df[:, cols]))
    heatmap(colnames, colnames, Σ; xrotation = 45, seriescolor = :PiYG_4)
end
```

It appears that temperature and dew point temperature columns are highly correlated. For that reason, we may want to exclude dew point temperature in our model below.

Seasonal analysis

Since this is a time series, let's check if there's a trend.



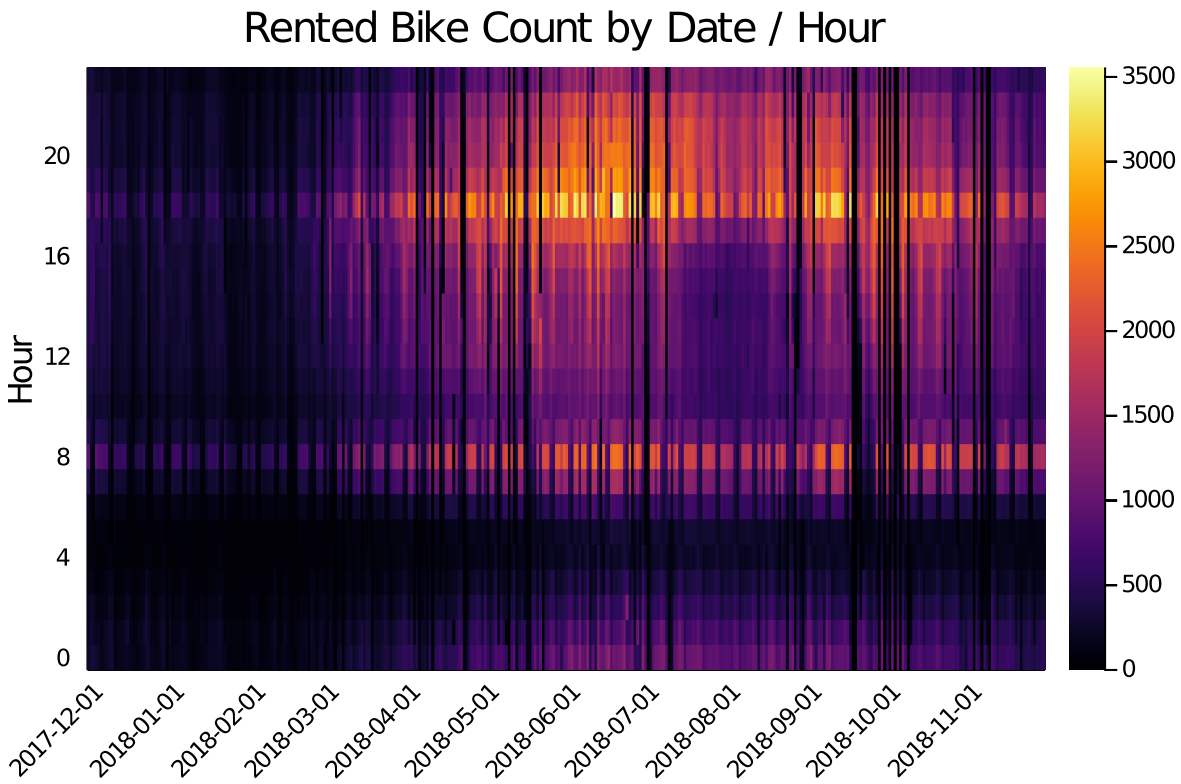

```
plot(df.Date, df.RentedBikeCount;
    legend = :none,
    ylabel = "Rented Bike Count")
```

As we only have 1 year of data, we cannot really see any seasonal trends. However, my intuition is that people don't tend to ride bikes during winter due to the low temperature. When spring comes around March/April, the demand picked up.

Now, December 2018 still look like a higher demand when compared with December 2017. A possibility is that biking may have gotten popular? Not sure...

Time of day

My initial guess is that people probably do not rent bikes early morning or late night. Let's verify that.

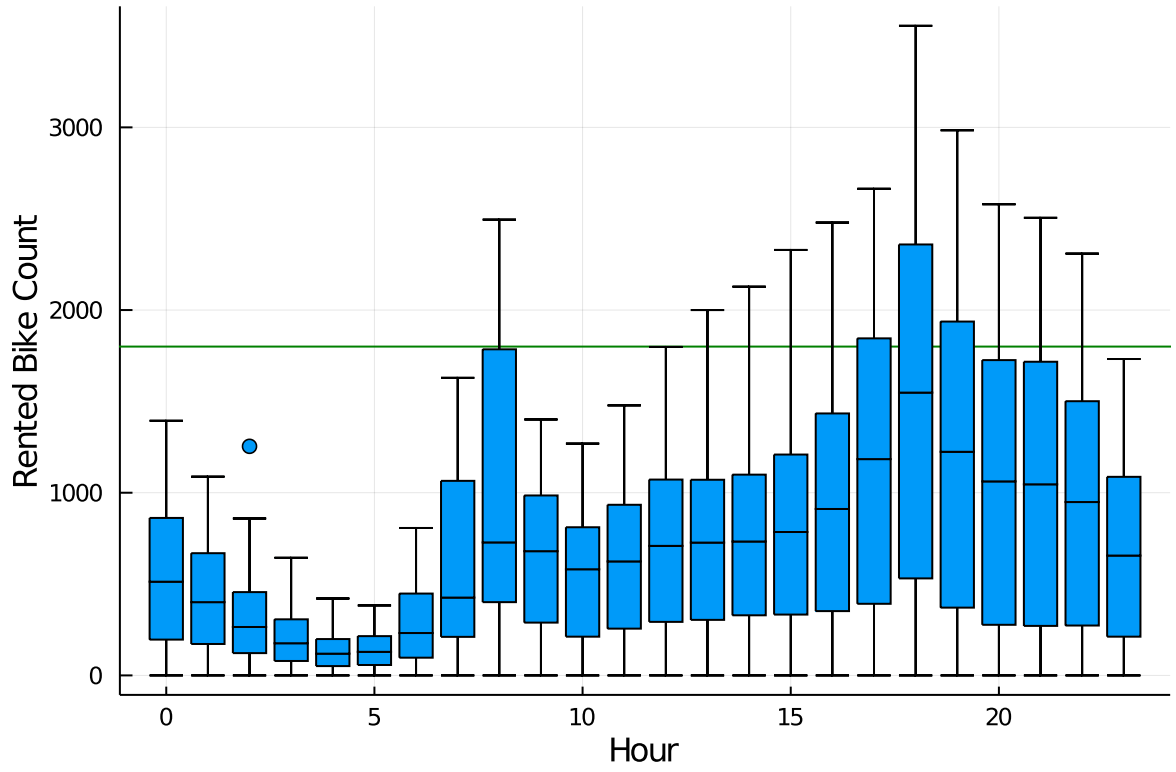



```
let
    data = @pipe df |>
        select(_, :Date, :Hour, :RentedBikeCount) |>
        unstack(_, :Hour, :RentedBikeCount) |>
        sort(_, :Date)

    matrix = Matrix(data[:, 2:end])'

    heatmap(data[:, 1], 0:23, matrix;
        title = "Rented Bike Count by Date / Hour",
        ylabel = "Hour",
        xrotation = 45.0,
        xticks = Date(2017,12,1):Month(1):Date(2019,1,1),
        yticks = 0:4:24)
end
```

The two horizontal stripes show that 8 AM and 6 PM are popular hours for bike rentals. That makes sense because people might want to do some exercise right before work or after work. Or, they may use it for transportation.

Korean seems to stay up late. Bike rental demand continues after 6 PM until almost midnight. By 3 AM, it's total silence.

Now, let's take a look at the same data from a different angle using a boxplot.




```
begin
    hline([1800], color = :green)
    @df df StatsPlots.boxplot!(:Hour, :RentedBikeCount;
        legend = :none,
        color = palette(:default)[1],  # reset to first color due to `hline` above
        xlabel = "Hour",
        ylabel = "Rented Bike Count")
end
```

The boxplot shows the quartiles for each hour across all dates in the year. It confirms our findings that 8 AM and 6 AM are popular times.

Categorical Variables

Some variables should be converted to categorical such that GLM.jl can encode dummy variables as such. Instead of mutating the existing `Hour` column, I will create a new `Hour2` column and make it categorical.

```
Int64[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19,
```

```
df.Hour2 = df.Hour
```

```
categorical!(df, :Hour2);
```

Linear Regression

Using linear regression, we can fit the data to a linear equation. Here, the response variable is `RentedBikeCount`. We can choose any of the other fields as explanatory variables. Let's start with something simple - an ordinary linear model.

```
hour_model = FormulaTerm
             Response:
               RentedBikeCount(unknown)
             Predictors:
               1
               Hour2(unknown)
```

```
hour_model = @formula(RentedBikeCount ~ 1 + Hour2)
```

To fit the model, we can use the `lm` function.

```
ols = StatsModels.TableRegressionModel{LinearModel{GLM.LmResp{Array{Float64,1}},GLM.Dens
      ePredChol{Float64,LinearAlgebra.Cholesky{Float64,Array{Float64,2}}}},Array{Float6
      4,2}}

      RentedBikeCount ~ 1 + Hour2

      Coefficients:
      ──────────────────────────────────────────────────────────────────────────────────
                        Coef.  Std. Error       t  Pr(>|t|)  Lower 95%  Upper 95%
      ──────────────────────────────────────────────────────────────────────────────────
      (Intercept)   541.46       28.4436   19.04    <1e-78   485.704    597.216
      Hour2: 1     -115.277      40.2253   -2.87    0.0042  -194.128    -36.4256
      Hour2: 2     -239.83       40.2253   -5.96    <1e-8   -318.681   -160.979
      Hour2: 3     -338.129      40.2253   -8.41    <1e-16  -416.98    -259.278
      Hour2: 4     -408.868      40.2253  -10.16    <1e-23  -487.72    -330.017
      Hour2: 5     -402.378      40.2253  -10.00    <1e-22  -481.229   -323.527
      Hour2: 6     -253.896      40.2253   -6.31    <1e-9   -332.747   -175.045
      Hour2: 7       64.5452     40.2253    1.60    0.1086   -14.3059   143.396
      Hour2: 8      474.241      40.2253   11.79    <1e-31   395.39     553.092
      Hour2: 9      104.523      40.2253    2.60    0.0094    25.6722   183.374
      Hour2: 10     -13.6384     40.2253   -0.34    0.7346   -92.4895    65.2128
      Hour2: 11      59.3918     40.2253    1.48    0.1399   -19.4593   138.243
      Hour2: 12     157.981      40.2253    3.93    <1e-4     79.1297   236.832
      Hour2: 13     191.786      40.2253    4.77    <1e-5    112.935    270.637
      Hour2: 14     217.364      40.2253    5.40    <1e-7    138.513    296.216
      Hour2: 15     287.726      40.2253    7.15    <1e-12   208.875    366.577
      Hour2: 16     389.162      40.2253    9.67    <1e-21   310.311    468.013
      Hour2: 17     597.049      40.2253   14.84    <1e-48   518.198    675.9
      Hour2: 18     961.466      40.2253   23.90    <1e-99   882.615   1040.32
      Hour2: 19     653.688      40.2253   16.25    <1e-57   574.837    732.539
      Hour2: 20     527.504      40.2253   13.11    <1e-38   448.653    606.355
      Hour2: 21     489.989      40.2253   12.18    <1e-33   411.138    568.84
      Hour2: 22     381.337      40.2253    9.48    <1e-20   302.486    460.188
      Hour2: 23     129.666      40.2253    3.22    0.0013    50.8146   208.517
      ──────────────────────────────────────────────────────────────────────────────────
```

```
ols = lm(hour_model, df)
```

We can now use the fitted model to predict bike demand.

```
(r_squared = 0.292047,  rmsd = 542.669)
```

```
let
    y = df.RentedBikeCount
    yhat = round.(Int, predict(ols))
    (r_squared = r2(ols), rmsd = rmsd(y, yhat))
```

```
end
```

The R^2 is very low, meaning that the current model cannot make very accurate prediction. That's understandable because we have only used a single explanatory variable.

The root mean squared deviation (`rmsd`) value shows how much the predicted values deviates from the actual values.

Now, let's design a more complex model but we will continue to use an ordinary linear model.

```
(r_squared = 0.661326,  rmsd = 375.337)
```

```
let
    model = @formula(RentedBikeCount ~
                         1 + Hour2 + Temperature + Humidity + WindSpeed +
                         Visibility + SolarRadiation + Rainfall + Snowfall +
                         Seasons + Holiday + FunctioningDay)
    fitted = lm(model, df)
    y = df.RentedBikeCount
    yhat = round.(Int, predict(fitted))
    (r_squared = r2(fitted), rmsd = rmsd(y, yhat))
end
```

That's great result. The R^2 has jumped to 0.66 now! The RMSD is also reduced quite significantly from 543 to 375.

Generalized Linear Model (GLM)

I wonder if we can do better. The GLM.jl package supports Generalized Linear Model (GLM) which is more flexible that linear regression. We will demonstrate how it works below.

Given that the response variable is a count, the general wisdom (not mine) is to design the model with Poisson distribution.

```
(rmsd = 317.52)
```

```
let
    model = @formula(RentedBikeCount ~
                         1 + Hour2 + Temperature + Humidity + WindSpeed +
                         Visibility + SolarRadiation + Rainfall + Snowfall +
                         Seasons + Holiday + FunctioningDay)
    fitted = glm(model, df, Poisson(), LogLink())
    y = df.RentedBikeCount
    yhat = round.(Int, predict(fitted))
    (rmsd = rmsd(y, yhat), )
end
```

There is no `r2` function defined for GLM models, so we just show RMSD here. As you can see, RMSD is further reduced using this model. That's an improvement.

What if we build an even more complex model? So far, all variables are independent. If we introduce interaction terms (multiple variables interacting with each other) then we may create a more powerful predictor.

The question is how to choose the right variables for the interaction terms. My gut feeling is that it would be appropriate to choose variables that are "orthogonal" to each other. For example, humidity and rainfall should be highly correlated and the interaction between them would be somewhat uninteresting. Hence, I have chosen to mix hour of day, temperature, and humidity in the following experiment.

```
(rmsd = 302.623)
```

```
let
    model = @formula(RentedBikeCount ~
                         1 + Hour2 + Temperature + Humidity + WindSpeed +
                         Visibility + SolarRadiation + Rainfall + Snowfall +
                         Seasons + Holiday + FunctioningDay +
                         Hour2 * Temperature * Humidity
    )
    fitted = glm(model, df, Poisson(), LogLink())
    y = df.RentedBikeCount
    yhat = round.(Int, predict(fitted))
    (rmsd = rmsd(y, yhat), )
end
```

That's great! The RMSD is now further reduced although not by a whole lot.

Todo's

So far, I have been fitting the model with the complete data set. In order to test the predictive power of the model, I should test it against unseen data. Of course, I don't have any more data than what I have downloaded. What I should do is to split the data set and do cross validation.

Resources

I don't know much about GLM before working on this. I found the following resources useful as I learn about the subject:

- Foundations of Linear and Generalized Linear Models by Alan Agresti (Wiley 2015)
- **Introduction to Generalized Linear Models**
- **MIT 18.650 Statistics for Applications, Phillipe Rigolle, Lecture 21-22**

Thanks you for reading. I hope you enjoy this notebook.

Tom Kwong, October 2020