Regression and correlation

Simple linear regression

Simple linear regression is a statistical technique used to model the relationship between two variables: a dependent variable (also called the response variable or outcome variable) and an independent variable (also called the predictor variable or explanatory variable). It assumes that the relationship between the variables can be approximated by a straight line.

In [19]:
```
using DataFrames

short_velocity = [1.76, 1.34, 1.27, 1.47, 1.27, 1.49, 1.31, 1.09, 1.18, 1.22, :
blood_glucose = [15.3, 10.8, 8.1, 19.5, 7.2, 5.3, 9.3, 11.1, 7.5, 12.2, 6.7, 5

df = DataFrame(short_velocity = short_velocity, blood_glucose = blood_glucose)
```

Out[19]: 24 rows × 2 columns

	short_velocity Float64?	blood_glucose Float64
1	1.76	15.3
2	1.34	10.8
3	1.27	8.1
4	1.47	19.5
5	1.27	7.2
6	1.49	5.3
7	1.31	9.3
8	1.09	11.1
9	1.18	7.5
10	1.22	12.2
11	1.25	6.7
12	1.19	5.2
13	1.95	19.0
14	1.28	15.1
15	1.52	6.7
16	missing	8.6
17	1.12	4.2
18	1.37	10.3
19	1.19	12.5
20	1.05	16.1
21	1.32	13.3
22	1.03	4.9
23	1.12	8.8
24	1.7	9.5

Train-Test Split

Split the data to train and test set.

In [20]:
```
import Pkg
```

In [21]:
```
Pkg.add("ScikitLearn")
```

```
   Resolving package versions...
   Installed Crayons ──────────── v4.1.1
   Installed Reexport ─────────── v1.2.2
   Installed PooledArrays ─────── v1.4.2
   Installed Compat ───────────── v4.7.0
   Installed Parsers ──────────── v2.7.1
   Installed DataStructures ───── v0.18.14
   Installed DataFrames ───────── v1.6.0
   Installed SortingAlgorithms ── v1.1.1
   Installed Missings ─────────── v1.1.0
   Installed StringManipulation ─ v0.3.0
   Installed PrettyTables ─────── v2.2.5
    Updating `C:\Users\Lenovo\path\to\new\environment\Project.toml`
  [3646fa90] + ScikitLearn v0.7.0
    Updating `C:\Users\Lenovo\path\to\new\environment\Manifest.toml`
  [d360d2e6] + ChainRulesCore v1.16.0
  [9e997f8a] + ChangesOfVariables v0.1.8
  [34da2185] + Compat v4.7.0
  [8f4d0f93] + Conda v1.9.0
```

In [18]:

```
# Train test split
using Lathe.preprocess: TrainTestSplit
train,test= TrainTestSplit(df,.75)
```

Out[18]:

(19×2 DataFrame

Row	short_velocity Float64?	blood_glucose Float64
1	1.76	15.3
2	1.27	8.1
3	1.27	7.2
4	1.49	5.3
5	1.31	9.3
6	1.09	11.1
7	1.18	7.5
8	1.22	12.2
9	1.25	6.7
10	1.19	5.2
11	1.95	19.0
12	1.28	15.1
13	1.52	6.7
14	missing	8.6
15	1.12	4.2
16	1.19	12.5
17	1.05	16.1
18	1.03	4.9
19	1.7	9.5

, 5×2 DataFrame

Row	short_velocity Float64?	blood_glucose Float64
1	1.34	10.8
2	1.47	19.5
3	1.37	10.3
4	1.32	13.3
5	1.12	8.8

)

In [1]:
```
import Pkg; Pkg.add("CRRao")
```

```
   Updating registry at `C:\Users\Lenovo\.julia\registries\General.toml`
  Resolving package versions...
```

```
Unsatisfiable requirements detected for package DataFrames [a93c6f00]:
 DataFrames [a93c6f00] log:
 ├─possible versions are: 0.11.7-1.5.0 or uninstalled
 ├─restricted to versions * by an explicit requirement, leaving only versions
0.11.7-1.5.0
 ├─restricted by compatibility requirements with RData [df47a6cb] to version
s: 0.13.0-1.5.0
 │ └─RData [df47a6cb] log:
 │   ├─possible versions are: 0.5.0-1.0.0 or uninstalled
 │   └─restricted to versions * by an explicit requirement, leaving only vers
ions 0.5.0-1.0.0
 ├─restricted by compatibility requirements with CRRao [49d1be55] to version
s: 1.0.0-1.5.0
 │ └─CRRao [49d1be55] log:
 │   ├─possible versions are: 0.1.0 or uninstalled
 │   └─restricted to versions * by an explicit requirement, leaving only vers
ions 0.1.0
 └─restricted by compatibility requirements with Lathe [38d8eb38] to version
s: 0.11.7-0.22.7 — no versions left
   └─Lathe [38d8eb38] log:
     ├─possible versions are: 0.0.3-0.1.8 or uninstalled
     └─restricted to versions * by an explicit requirement, leaving only vers
ions 0.0.3-0.1.8

Stacktrace:
  [1] propagate_constraints!(graph::Pkg.Resolve.Graph, sources::Set{Int64}; l
og_events::Bool)
    @ Pkg.Resolve C:\Users\Lenovo\AppData\Local\Programs\Julia-1.8.5\share\ju
lia\stdlib\v1.8\Pkg\src\Resolve\graphtype.jl:1072
  [2] propagate_constraints! (repeats 2 times)
    @ C:\Users\Lenovo\AppData\Local\Programs\Julia-1.8.5\share\julia\stdlib\v
1.8\Pkg\src\Resolve\graphtype.jl:1008 [inlined]
  [3] simplify_graph!(graph::Pkg.Resolve.Graph, sources::Set{Int64}; clean_gr
aph::Bool)
    @ Pkg.Resolve C:\Users\Lenovo\AppData\Local\Programs\Julia-1.8.5\share\ju
lia\stdlib\v1.8\Pkg\src\Resolve\graphtype.jl:1533
  [4] simplify_graph! (repeats 2 times)
    @ C:\Users\Lenovo\AppData\Local\Programs\Julia-1.8.5\share\julia\stdlib\v
1.8\Pkg\src\Resolve\graphtype.jl:1532 [inlined]
  [5] resolve_versions!(env::Pkg.Types.EnvCache, registries::Vector{Pkg.Regis
try.RegistryInstance}, pkgs::Vector{Pkg.Types.PackageSpec}, julia_version::Ve
rsionNumber)
    @ Pkg.Operations C:\Users\Lenovo\AppData\Local\Programs\Julia-1.8.5\share
\julia\stdlib\v1.8\Pkg\src\Operations.jl:352
  [6] targeted_resolve(env::Pkg.Types.EnvCache, registries::Vector{Pkg.Regist
ry.RegistryInstance}, pkgs::Vector{Pkg.Types.PackageSpec}, preserve::Pkg.Type
s.PreserveLevel, julia_version::VersionNumber)
    @ Pkg.Operations C:\Users\Lenovo\AppData\Local\Programs\Julia-1.8.5\share
\julia\stdlib\v1.8\Pkg\src\Operations.jl:1254
  [7] tiered_resolve(env::Pkg.Types.EnvCache, registries::Vector{Pkg.Registr
y.RegistryInstance}, pkgs::Vector{Pkg.Types.PackageSpec}, julia_version::Vers
ionNumber)
    @ Pkg.Operations C:\Users\Lenovo\AppData\Local\Programs\Julia-1.8.5\share
\julia\stdlib\v1.8\Pkg\src\Operations.jl:1239
  [8] _resolve(io::IJulia.IJuliaStdio{Base.PipeEndpoint}, env::Pkg.Types.EnvC
ache, registries::Vector{Pkg.Registry.RegistryInstance}, pkgs::Vector{Pkg.Typ
es.PackageSpec}, preserve::Pkg.Types.PreserveLevel, julia_version::VersionNum
```

```
ber)
    @ Pkg.Operations C:\Users\Lenovo\AppData\Local\Programs\Julia-1.8.5\share
\julia\stdlib\v1.8\Pkg\src\Operations.jl:1260
  [9] add(ctx::Pkg.Types.Context, pkgs::Vector{Pkg.Types.PackageSpec}, new_gi
t::Set{Base.UUID}; preserve::Pkg.Types.PreserveLevel, platform::Base.BinaryPl
atforms.Platform)
    @ Pkg.Operations C:\Users\Lenovo\AppData\Local\Programs\Julia-1.8.5\share
\julia\stdlib\v1.8\Pkg\src\Operations.jl:1276
 [10] add(ctx::Pkg.Types.Context, pkgs::Vector{Pkg.Types.PackageSpec}; preser
ve::Pkg.Types.PreserveLevel, platform::Base.BinaryPlatforms.Platform, kwarg
s::Base.Pairs{Symbol, IJulia.IJuliaStdio{Base.PipeEndpoint}, Tuple{Symbol}, N
amedTuple{(:io,), Tuple{IJulia.IJuliaStdio{Base.PipeEndpoint}}}})
    @ Pkg.API C:\Users\Lenovo\AppData\Local\Programs\Julia-1.8.5\share\julia
\stdlib\v1.8\Pkg\src\API.jl:275
 [11] add(pkgs::Vector{Pkg.Types.PackageSpec}; io::IJulia.IJuliaStdio{Base.Pi
peEndpoint}, kwargs::Base.Pairs{Symbol, Union{}, Tuple{}, NamedTuple{(), Tupl
e{}}})
    @ Pkg.API C:\Users\Lenovo\AppData\Local\Programs\Julia-1.8.5\share\julia
\stdlib\v1.8\Pkg\src\API.jl:156
 [12] add(pkgs::Vector{Pkg.Types.PackageSpec})
    @ Pkg.API C:\Users\Lenovo\AppData\Local\Programs\Julia-1.8.5\share\julia
\stdlib\v1.8\Pkg\src\API.jl:145
 [13] #add#27
    @ C:\Users\Lenovo\AppData\Local\Programs\Julia-1.8.5\share\julia\stdlib\v
1.8\Pkg\src\API.jl:144 [inlined]
 [14] add
    @ C:\Users\Lenovo\AppData\Local\Programs\Julia-1.8.5\share\julia\stdlib\v
1.8\Pkg\src\API.jl:144 [inlined]
 [15] #add#26
    @ C:\Users\Lenovo\AppData\Local\Programs\Julia-1.8.5\share\julia\stdlib\v
1.8\Pkg\src\API.jl:143 [inlined]
 [16] add(pkg::String)
    @ Pkg.API C:\Users\Lenovo\AppData\Local\Programs\Julia-1.8.5\share\julia
\stdlib\v1.8\Pkg\src\API.jl:143
 [17] top-level scope
    @ In[1]:1
```

Performed linear Regression

In [22]:
```
import Pkg
```

In [5]:
```
Pkg.add("LinearRegression")
```

```
   Resolving package versions...
  No Changes to `C:\Users\Lenovo\.julia\environments\v1.8\Project.toml`
  No Changes to `C:\Users\Lenovo\.julia\environments\v1.8\Manifest.toml`
```

In [19]:
```
using DataFrames
using GLM

# Define the data
short_velocity = [1.76, 1.34, 1.27, 1.47, 1.27, 1.49, 1.31, 1.09, 1.18, 1.22, 
blood_glucose = [15.3, 10.8, 8.1, 19.5, 7.2, 5.3, 9.3, 11.1, 7.5, 12.2, 6.7, 5

# Create the DataFrame
df = DataFrame(short_velocity = short_velocity, blood_glucose = blood_glucose)

# Remove rows with missing values
df = dropmissing(df)

# Perform linear regression
model = lm(@formula(blood_glucose ~ short_velocity), df)
```

Out[19]:
```
StatsModels.TableRegressionModel{LinearModel{GLM.LmResp{Vector{Float64}}, GL
M.DensePredChol{Float64, LinearAlgebra.CholeskyPivoted{Float64, Matrix{Float6
4}, Vector{Int64}}}}, Matrix{Float64}}

blood_glucose ~ 1 + short_velocity

Coefficients:
─────────────────────────────────────────────────────────────────────────────
─
                    Coef.  Std. Error      t  Pr(>|t|)    Lower 95%  Upper 9
5%
─────────────────────────────────────────────────────────────────────────────
─
(Intercept)     -0.109632     5.06302  -0.02    0.9829  -10.6388       10.419
5
short_velocity   7.90822      3.7641    2.10    0.0479    0.0803362    15.736
1
─────────────────────────────────────────────────────────────────────────────
─
```

Summay of Data A summary of a linear regression analysis typically includes several key components that provide an overview of the model's performance and statistical significance.

In [6]:
```
using DataFrames
using GLM

# Define the data
short_velocity = [1.76, 1.34, 1.27, 1.47, 1.27, 1.49, 1.31, 1.09, 1.18, 1.22, 
blood_glucose = [15.3, 10.8, 8.1, 19.5, 7.2, 5.3, 9.3, 11.1, 7.5, 12.2, 6.7, 5

# Create the DataFrame
df = DataFrame(short_velocity = short_velocity, blood_glucose = blood_glucose)

# Remove rows with missing values
df = dropmissing(df)

# Perform linear regression
model = lm(@formula(short_velocity ~ blood_glucose), df)

# Obtain a summary of the regression model
summary_table = coeftable(model)
println(summary_table)
```

	Estimate	Std.Error	t value	Pr(>|t|)
(Intercept)	1.09781	0.117481	9.3446	<1e-08
blood_glucose	0.0219625	0.0104536	2.10096	0.0479

In [1]:
```
import Pkg
Pkg.add("Plots")
```

```
    Updating registry at `C:\Users\Lenovo\.julia\registries\General.toml`
   Resolving package versions...
  No Changes to `C:\Users\Lenovo\.julia\environments\v1.8\Project.toml`
  No Changes to `C:\Users\Lenovo\.julia\environments\v1.8\Manifest.toml`
```

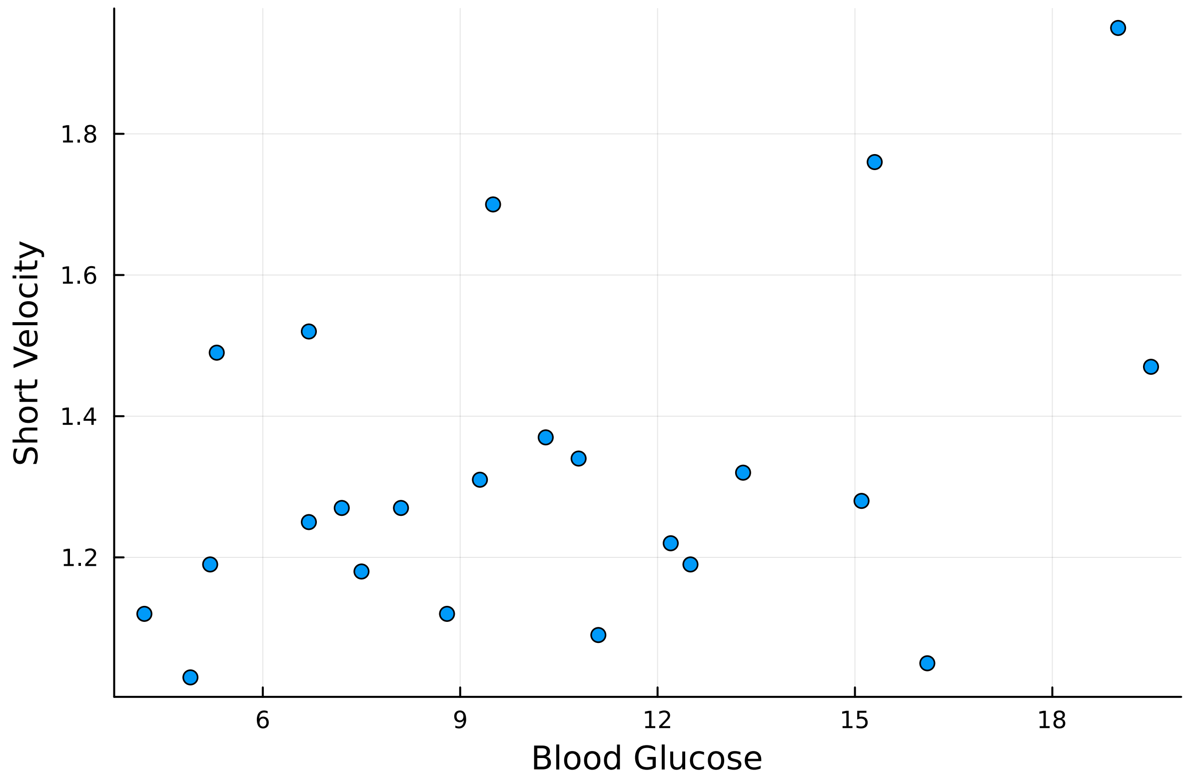
In [2]:
```
using Pkg
Pkg.activate("path/to/new/environment")
```

```
  Activating project at `C:\Users\Lenovo\path\to\new\environment`
```

In [3]:
```
using Plots

short_velocity = [1.76, 1.34, 1.27, 1.47, 1.27, 1.49, 1.31, 1.09, 1.18, 1.22, 
blood_glucose = [15.3, 10.8, 8.1, 19.5, 7.2, 5.3, 9.3, 11.1, 7.5, 12.2, 6.7, 5

scatter(blood_glucose, short_velocity, xlabel="Blood Glucose", ylabel="Short Ve
```

```
[ Info: Precompiling Plots [91a5bcdd-55d7-5caf-9e0b-520d859cae80]
```

Out[3]:



Plot of the data

When plotting data with confidence and prediction limits, you typically want to visualize the uncertainty associated with the model's predictions. Confidence limits represent the range within which you can be confident that the true population parameter lies, while prediction limits represent the range within which individual future observations are likely to fall.

abline Plot

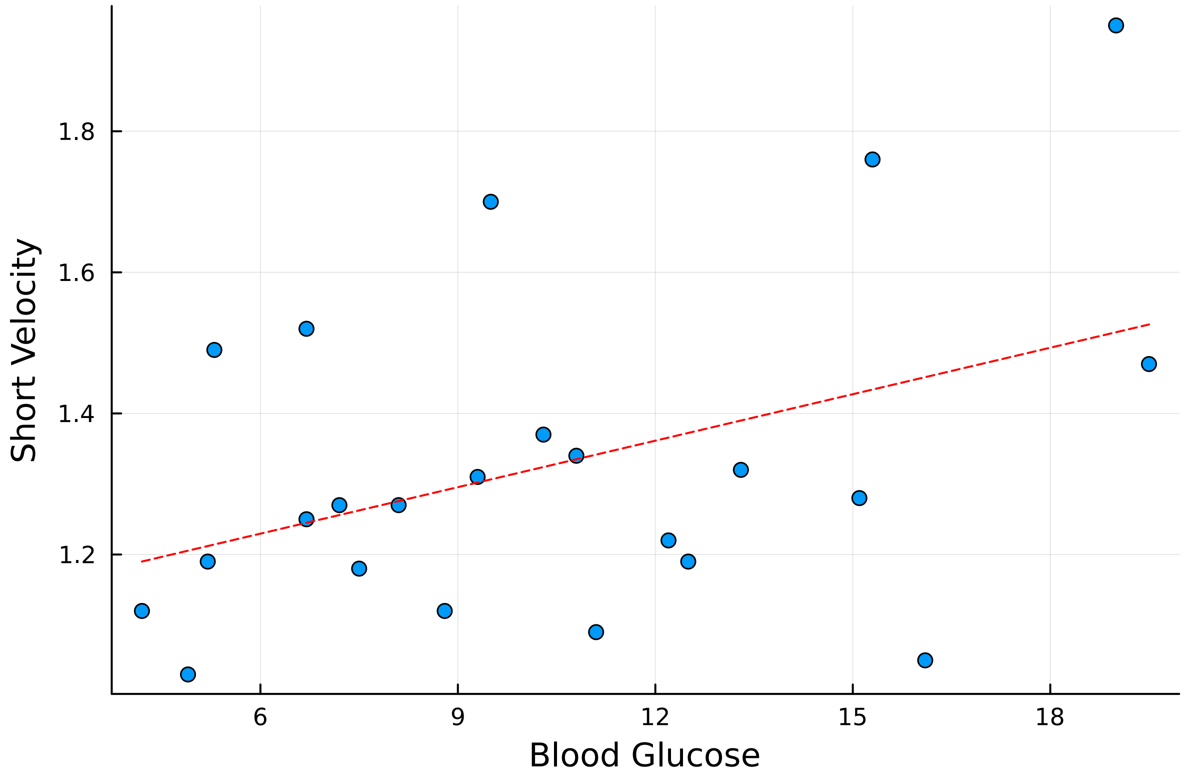
An "abline plot" typically refers to a plot that includes a straight line with a specified slope and intercept.

In [12]:
```
import Pkg
Pkg.add("GLM")
```

```
   Resolving package versions...
   Installed FillArrays ───────────── v1.4.0
   Installed DualNumbers ──────────── v0.6.8
   Installed StatsFuns ────────────── v1.3.0
   Installed HypergeometricFunctions ─ v0.3.20
   Installed Distributions ─────────── v0.25.98
    Updating `C:\Users\Lenovo\path\to\new\environment\Project.toml`
  [38e38edf] + GLM v1.8.3
    Updating `C:\Users\Lenovo\path\to\new\environment\Manifest.toml`
  [49dc2e85] + Calculus v0.5.1
  [b429d917] + DensityInterface v0.4.0
  [31c24e10] + Distributions v0.25.98
  [fa6b7ba4] + DualNumbers v0.6.8
  [1a297f60] + FillArrays v1.4.0
  [38e38edf] + GLM v1.8.3
  [34004b35] + HypergeometricFunctions v0.3.20
  [90014a1f] + PDMats v0.11.17
  [1fd47b50] + QuadGK v2.8.2
  [79098fc4] + Rmath v0.7.1
  [1277b4bf] + ShiftedArrays v2.0.0
  [4c63d2b9] + StatsFuns v1.3.0
  [3eaba693] + StatsModels v0.7.2
  [f50d1b31] + Rmath_jll v0.4.0+0
  [4607b0f0] + SuiteSparse
Precompiling project...
  ✓ PDMats
  ✓ DualNumbers
  ✓ FillArrays
  ✓ HypergeometricFunctions
  ✓ StatsFuns
  ✓ StatsModels
  ✓ Distributions
  ✓ GLM
  8 dependencies successfully precompiled in 41 seconds. 166 already precompi
led.
```

In [13]:
```
using DataFrames
using GLM
using Plots

# Define the data
short_velocity = [1.76, 1.34, 1.27, 1.47, 1.27, 1.49, 1.31, 1.09, 1.18, 1.22, 
blood_glucose = [15.3, 10.8, 8.1, 19.5, 7.2, 5.3, 9.3, 11.1, 7.5, 12.2, 6.7, 5

# Create the DataFrame
df = DataFrame(short_velocity = short_velocity, blood_glucose = blood_glucose)

# Remove rows with missing values
df = dropmissing(df)

# Perform linear regression
model = lm(@formula(short_velocity ~ blood_glucose), df)

# Create scatter plot
scatter(df.blood_glucose, df.short_velocity, xlabel = "Blood Glucose", ylabel 

# Extract regression coefficients
intercept = coef(model)[1]
slope = coef(model)[2]

# Generate points for the regression line
x_values = range(minimum(df.blood_glucose), stop=maximum(df.blood_glucose), le
y_values = intercept .+ slope .* x_values

# Plot the regression line
plot!(x_values, y_values, line=:dash, color=:red, label="Regression Line")
```

Out[13]:



Residuals and fitted values are key components in regression analysis and help assess the performance of the regression model

In [23]:
```
#using DataFrames
#using GLM

# Define the data
short_velocity = [1.76, 1.34, 1.27, 1.47, 1.27, 1.49, 1.31, 1.09, 1.18, 1.22, 
blood_glucose = [15.3, 10.8, 8.1, 19.5, 7.2, 5.3, 9.3, 11.1, 7.5, 12.2, 6.7, 5

# Create the DataFrame
df = DataFrame(short_velocity = short_velocity, blood_glucose = blood_glucose)

# Remove rows with missing values
df = dropmissing(df)

# Perform linear regression
model = lm(@formula(short_velocity ~ blood_glucose), df)

# Store the model
lm_velo = model
```

Out[23]:
```
StatsModels.TableRegressionModel{LinearModel{GLM.LmResp{Vector{Float64}}, GL
M.DensePredChol{Float64, LinearAlgebra.CholeskyPivoted{Float64, Matrix{Float6
4}, Vector{Int64}}}}, Matrix{Float64}}

short_velocity ~ 1 + blood_glucose

Coefficients:
```

	Coef.	Std. Error	t	Pr(>|t|)	Lower 95%	Upper 95%
(Intercept)	1.09781	0.117481	9.34	<1e-08	0.853499	1.34213
blood_glucose	0.0219625	0.0104536	2.10	0.0479	0.000223108	0.0437019

In [24]:
```
#using GLM

# Obtain the fitted values
fitted_values = fitted(lm_velo)

# Print the fitted values
println(fitted_values)
```

```
[1.4338414683503315, 1.3350101181803464, 1.2757113080783553, 1.52608406184231
75, 1.2559450380443584, 1.2142162457503647, 1.3020663347903514, 1.34159887485
83454, 1.2625337947223574, 1.3657576493443417, 1.24496377691436, 1.2120199935
24365, 1.5151028007123193, 1.429448963898332, 1.24496377691436, 1.19005747126
43683, 1.324028857050348, 1.372346406022341, 1.451411486158329, 1.38991642383
0338, 1.205431236846366, 1.291085073660353, 1.3064588392423508]
```

fitted line on the plot The "fitted line" refers to the line that represents the predicted values of the dependent variable based on a regression model. It is also known as the "regression line" or "best-fit line."

Prediction and confidence bands are graphical representations of uncertainty in the predictions made by a regression model. They are typically plotted around the fitted line to illustrate the range of possible values for future observations or the uncertainty in the estimated mean response.

In [26]:
```
thuesen = [1.433841, 1.335010, 1.275711, 1.526084, 1.255945, 1.214216, 1.30206
           1.515103, 1.429449, 1.244964, missing, 1.190057, 1.324029, 1.372346
```

Out[26]:
```
24-element Vector{Union{Missing, Float64}}:
 1.433841
 1.33501
 1.275711
 1.526084
 1.255945
 1.214216
 1.302066
 1.341599
 1.262534
 1.365758
 1.244964
 1.21202
 1.515103
 1.429449
 1.244964
  missing
 1.190057
 1.324029
 1.372346
 1.451411
 1.389916
 1.205431
 1.291085
 1.306459
```

Q-Q PLOT

A Q-Q plot, short for quantile-quantile plot, is a graphical tool used to assess if a given dataset follows a specific theoretical distribution, such as a normal distribution. The Q-Q plot compares the quantiles of the observed data against the quantiles expected from the theoretical distribution.

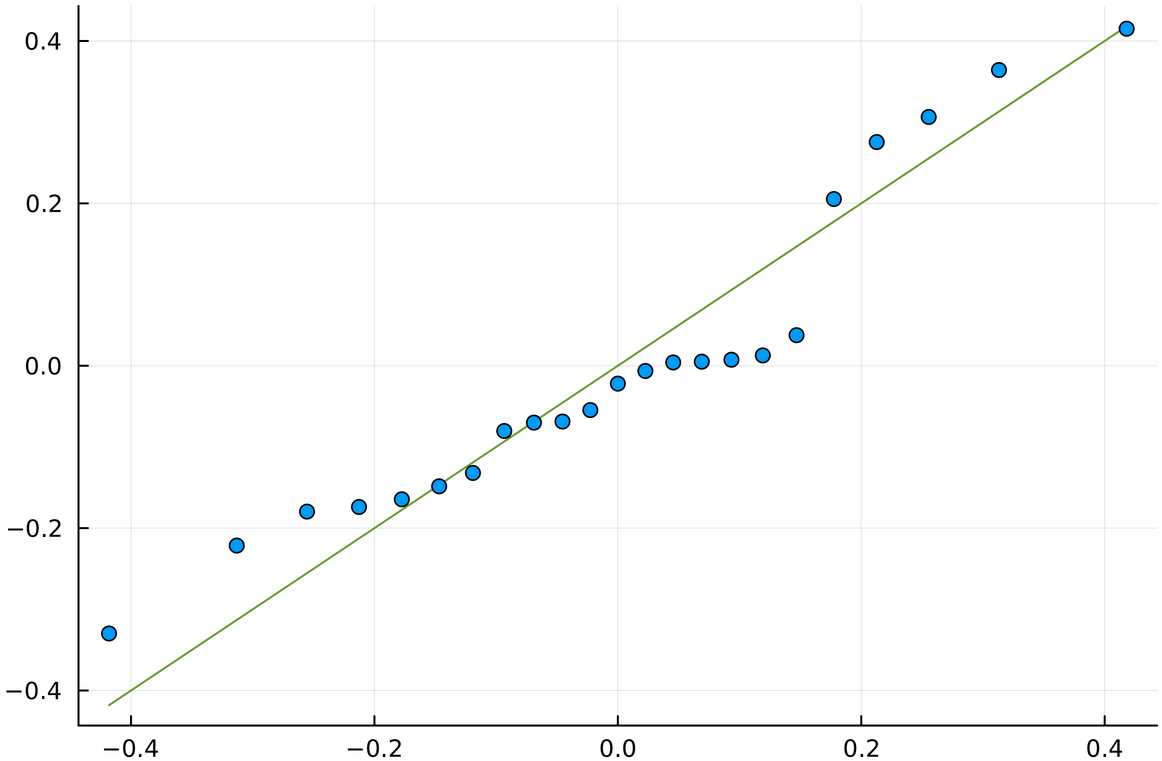
In [14]:
```
import Pkg
Pkg.add("StatsPlots")
```

```
   Resolving package versions...
   Installed StaticArrays ─ v1.6.1
   Installed Clustering ─── v0.15.3
    Updating `C:\Users\Lenovo\path\to\new\environment\Project.toml`
  [f3b207a7] + StatsPlots v0.15.5
    Updating `C:\Users\Lenovo\path\to\new\environment\Manifest.toml`
  [621f4979] + AbstractFFTs v1.4.0
  [79e6a3ab] + Adapt v3.6.2
  [7d9fca2a] + Arpack v0.5.4
  [13072b0f] + AxisAlgorithms v1.0.1
  [aaaa29a8] + Clustering v0.15.3
  [b4f34e82] + Distances v0.10.8
  [7a1cc6ca] + FFTW v1.7.1
  [a98d9a8b] + Interpolations v0.14.7
  [5ab0869b] + KernelDensity v0.6.7
  [6f286f6a] + MultivariateStats v0.10.2
  [b8a86587] + NearestNeighbors v0.4.13
  [510215fc] + Observables v0.5.4
  [6fe1bfb0] + OffsetArrays v1.12.10
  [c84ed2f1] + Ratios v0.4.5
  [90137ffa] + StaticArrays v1.6.1
  [1e83bf80] + StaticArraysCore v1.4.1
  [f3b207a7] + StatsPlots v0.15.5
  [ab02a1b2] + TableOperations v1.2.0
  [cc8bc4a8] + Widgets v0.6.6
  [efce3f68] + WoodburyMatrices v0.5.5
⌅ [68821587] + Arpack_jll v3.5.1+1
  [f5851436] + FFTW_jll v3.3.10+0
  [1d5cc7b8] + IntelOpenMP_jll v2023.1.0+0
  [856f044c] + MKL_jll v2023.1.0+0
  [4af54fe1] + LazyArtifacts
  [1a1011a3] + SharedArrays
        Info Packages marked with ⌅ have new versions available but compatibi
lity constraints restrict them from upgrading. To see why use `status --outda
ted -m`
Precompiling project...
  ✓ StaticArrays
  ✓ NearestNeighbors
  ✓ Interpolations
  ✓ Clustering
  ✓ KernelDensity
  ✓ StatsPlots
  6 dependencies successfully precompiled in 162 seconds. 192 already precomp
iled.
```

In [15]:
```
using DataFrames
using GLM
using Plots
using StatsPlots

# Define the data
short_velocity = [1.76, 1.34, 1.27, 1.47, 1.27, 1.49, 1.31, 1.09, 1.18, 1.22,
blood_glucose = [15.3, 10.8, 8.1, 19.5, 7.2, 5.3, 9.3, 11.1, 7.5, 12.2, 6.7, 5

# Create the DataFrame
df = DataFrame(short_velocity = short_velocity, blood_glucose = blood_glucose)

# Remove rows with missing values
df = dropmissing(df)

# Perform linear regression
model = lm(@formula(short_velocity ~ blood_glucose), df)

# Obtain the residuals
residuals_values = residuals(model)

# Create QQ plot
qqnorm(residuals_values)
```

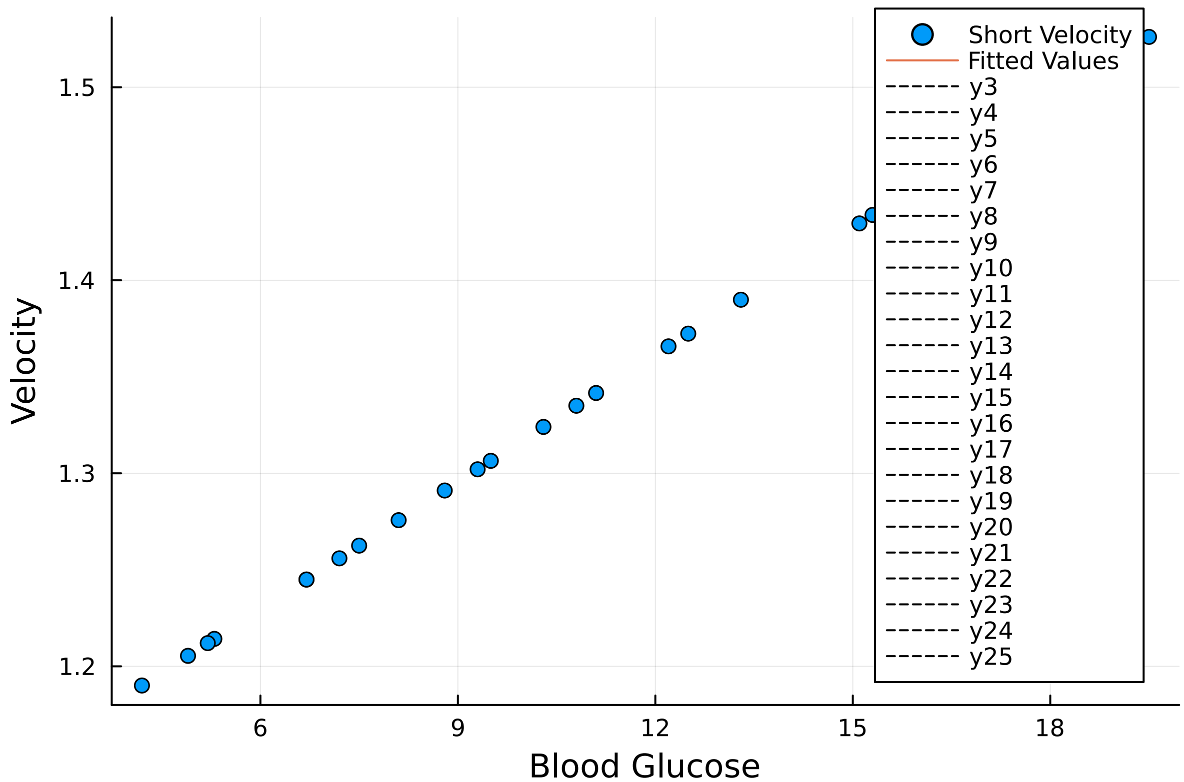
```
[ Info: Precompiling StatsPlots [f3b207a7-027a-5e70-b257-86293d7955fd]
┌ Warning: Module Plots with build ID 144100985896101 is missing from the cac
he.
│ This may mean Plots [91a5bcdd-55d7-5caf-9e0b-520d859cae80] does not support
precompilation but is imported by a module that does.
└ @ Base loading.jl:1325
[ Info: Skipping precompilation since __precompile__(false). Importing StatsP
lots [f3b207a7-027a-5e70-b257-86293d7955fd].
```

Out[15]:



In [16]:
```
using DataFrames
using GLM
using Plots

# Define the data
short_velocity = [1.433841, 1.335010, 1.275711, 1.526084, 1.255945, 1.214216, 
blood_glucose = [15.3, 10.8, 8.1, 19.5, 7.2, 5.3, 9.3, 11.1, 7.5, 12.2, 6.7, 5

# Create the DataFrame
df = DataFrame(short_velocity = short_velocity, blood_glucose = blood_glucose)

# Remove rows with missing values
df = dropmissing(df)

# Perform linear regression
model = lm(@formula(short_velocity ~ blood_glucose), df)

# Obtain the fitted values
fitted_values = fitted(model)

# Create the plot
scatter(df.blood_glucose, df.short_velocity, label = "Short Velocity")
plot!(blood_glucose, fitted_values, label = "Fitted Values")

# Filter out missing values for line segments
valid_indices = findall(!isnan, short_velocity)
valid_blood_glucose = blood_glucose[valid_indices]
valid_fitted_values = fitted_values[valid_indices]
valid_short_velocity = short_velocity[valid_indices]

for i in eachindex(valid_indices)
    plot!([valid_blood_glucose[i], valid_blood_glucose[i]], [valid_fitted_valu
end

xlabel!("Blood Glucose")
ylabel!("Velocity")
```

Out[16]:



In [23]:

```
#using DataFrames
#using Statistics

# Define the data
short_velocity = [1.76, 1.34, 1.27, 1.47, 1.27, 1.49, 1.31, 1.09, 1.18, 1.22,
blood_glucose = [15.3, 10.8, 8.1, 19.5, 7.2, 5.3, 9.3, 11.1, 7.5, 12.2, 6.7, 5
df = DataFrame(short_velocity = short_velocity, blood_glucose = blood_glucose)

# Remove missing values
df = dropmissing(df)

# Fit the linear regression model
lm_velo = lm(@formula(short_velocity ~ blood_glucose), df)

# Make predictions
new_data = DataFrame(blood_glucose = [7.8, 9.1, 11.5])
predicted_velo = predict(lm_velo, new_data, interval = :confidence)

# Print the predicted velocities
println(predicted_velo)
```

3×3 DataFrame

Row	prediction Float64?	lower Float64?	upper Float64?
1	1.26912	1.15976	1.37849
2	1.29767	1.19971	1.39564
3	1.35038	1.25328	1.44749

In [24]:
```
using Pkg
Pkg.add("GLM")
```

```
   Resolving package versions...
  No Changes to `C:\Users\Lenovo\.julia\environments\v1.8\Project.toml`
  No Changes to `C:\Users\Lenovo\.julia\environments\v1.8\Manifest.toml`
```

In [25]:
```
using DataFrames

pred_frame = DataFrame(blood_glucose = 4:20)
```

Out[25]: 17 rows × 1 columns

	blood_glucose
	Int64
1	4
2	5
3	6
4	7
5	8
6	9
7	10
8	11
9	12
10	13
11	14
12	15
13	16
14	17
15	18
16	19
17	20

In [16]:
```
using DataFrames
using GLM

# Assuming you have already defined the DataFrame df

lm_velo = lm(@formula(short_velocity ~ blood_glucose), df)
```

Out[16]:
```
StatsModels.TableRegressionModel{LinearModel{GLM.LmResp{Vector{Float64}}, GL
M.DensePredChol{Float64, LinearAlgebra.CholeskyPivoted{Float64, Matrix{Float6
4}, Vector{Int64}}}}, Matrix{Float64}}

short_velocity ~ 1 + blood_glucose

Coefficients:
```

	Coef.	Std. Error	t	Pr(>|t|)	Lower 95%	Upper 95%
(Intercept)	1.09781	0.117481	9.34	<1e-08	0.853499	1.34213
blood_glucose	0.0219625	0.0104536	2.10	0.0479	0.000223108	0.0437019

Fitted values, also known as predicted values, are the estimated values of the dependent variable obtained from a regression model. These values represent the model's predicted or expected values for the dependent variable given specific values of the independent variables.

In [17]:
```
fitted_vals = fitted(lm_velo)
```

Out[17]:
```
23-element Vector{Float64}:
 1.4338414683503315
 1.3350101181803464
 1.2757113080783553
 1.5260840618423175
 1.2559450380443584
 1.2142162457503647
 1.3020663347903514
 1.3415988748583454
 1.2625337947223574
 1.3657576493443417
 1.24496377691436
 1.212019993524365
 1.5151028007123193
 1.429448963898332
 1.24496377691436
 1.1900574712643683
 1.324028857050348
 1.372346406022341
 1.451411486158329
 1.389916423830338
 1.205431236846366
 1.29108507366035
 1.3064588392423508
```

In [10]:
```
import Pkg
Pkg.add("MLBase")
```

```
    Updating registry at `C:\Users\Lenovo\.julia\registries\General.toml`
   Resolving package versions...
  No Changes to `C:\Users\Lenovo\.julia\environments\v1.8\Project.toml`
  No Changes to `C:\Users\Lenovo\.julia\environments\v1.8\Manifest.toml`
```

In [7]:
```
] add DataFrames Plots GLM
```

```
    Updating registry at `C:\Users\Lenovo\.julia\registries\General.toml`
   Resolving package versions...
  No Changes to `C:\Users\Lenovo\.julia\environments\v1.8\Project.toml`
  No Changes to `C:\Users\Lenovo\.julia\environments\v1.8\Manifest.toml`
```

In [2]:
```
using Pkg
Pkg.add("GLM")
```

```
    Updating registry at `C:\Users\Lenovo\.julia\registries\General.toml`
   Resolving package versions...
  No Changes to `C:\Users\Lenovo\.julia\environments\v1.8\Project.toml`
  No Changes to `C:\Users\Lenovo\.julia\environments\v1.8\Manifest.toml`
```

All the elementary statistical functions in julia require either that all values be nonmissing or that you explicitly state what should be done with the cases with missing values

In [10]:
```
using Statistics

# Calculate correlation coefficient
correlation = cor(df.blood_glucose, df.short_velocity)

# Print the correlation coefficient
println("Correlation coefficient: $correlation")
```

```
Correlation coefficient: missing
```

The correlation coefficient is a statistical measure that quantifies the strength and direction of the linear relationship between two variables. It is denoted by the symbol "r" and can range from -1 to 1

In [13]:

```
#using Pkg
#Pkg.add("DataFrames")
#kg.add("Statistics")

using DataFrames
using Statistics

# Create a new DataFrame with the variables
df = DataFrame(short_velocity = [1.76, 1.34, 1.27, 1.47, 1.27, 1.49, 1.31, 1.0
               blood_glucose = [15.3, 10.8, 8.1, 19.5, 7.2, 5.3, 9.3, 11.1, 7.

# Drop rows with missing values
df_complete = dropmissing(df)

# Calculate correlation coefficient
correlation = cor(df_complete.blood_glucose, df_complete.short_velocity)

# Print the correlation coefficient
println("Correlation coefficient: $correlation")
```

```
Correlation coefficient: 0.4167545988606907
```