```
ingredients (generic function with 1 method)
```

Project: Titanic - Machine Learning from Disaster

Overview

1. Exploring the data
2. Data Cleaning / Preprocessing
3. Model Building
4. Results

```julia
using Plots
```

```julia
using CSV
```

```julia
using DataFrames
```

```julia
using ScikitLearn
```

```julia
using VegaLite
```

```julia
using Statistics
```

```julia
using Distributions
```

	PassengerId	Survived	Pclass	Name	Sex	
1	1	0	3	"Braund, Mr. Owen Harris"	"male"	22.
2	2	1	1	"Cumings, Mrs. John Bradley (Florence	"female"	38.
3	3	1	3	"Heikkinen, Miss. Laina"	"female"	26.
4	4	1	1	"Futrelle, Mrs. Jacques Heath (Lily Ma	"female"	35.
5	5	0	3	"Allen, Mr. William Henry"	"male"	35.
6	6	0	3	"Moran, Mr. James"	"male"	mis
7	7	0	1	"McCarthy, Mr. Timothy J"	"male"	54.
8	8	0	3	"Palsson, Master. Gosta Leonard"	"male"	2.0
9	9	1	3	"Johnson, Mrs. Oscar W (Elisabeth Vilh	"female"	27.
10	10	1	2	"Nasser, Mrs. Nicholas (Adele Achem)"	"female"	14.
more						
891	891	0	3	"Dooley, Mr. Patrick"	"male"	32.

```
begin
    # load Train Data
    path_train = "train.csv"
    data_train = CSV.read(path_train, DataFrame)
end
```

	PassengerId	Pclass	Name	Sex	Age	Sil
1	892	3	"Kelly, Mr. James"	"male"	34.5	0
2	893	3	"Wilkes, Mrs. James (Ellen Needs)"	"female"	47.0	1
3	894	2	"Myles, Mr. Thomas Francis"	"male"	62.0	0
4	895	3	"Wirz, Mr. Albert"	"male"	27.0	0
5	896	3	"Hirvonen, Mrs. Alexander (Helga E Lin	"female"	22.0	1
6	897	3	"Svensson, Mr. Johan Cervin"	"male"	14.0	0
7	898	3	"Connolly, Miss. Kate"	"female"	30.0	0
8	899	2	"Caldwell, Mr. Albert Francis"	"male"	26.0	1
9	900	3	"Abrahim, Mrs. Joseph (Sophie Halaut E	"female"	18.0	0
10	901	3	"Davies, Mr. John Samuel"	"male"	21.0	2
more						
418	1309	3	"Peter, Master. Michael J"	"male"	missing	1

```
begin
    # load Test Data
        path_test = "test.csv"
        data_test = CSV.read(path_test, DataFrame)
end
```

Exploring the data

	variable	mean	min	median	max
1	:PassengerId	446.0	1	446.0	891
2	:Survived	0.383838	0	0.0	1
3	:Pclass	2.30864	1	3.0	3
4	:Name	nothing	"Abbing, Mr. Anthony"	nothing	"van Melkebeke, Mr. Philemo
5	:Sex	nothing	"female"	nothing	"male"
6	:Age	29.6991	0.42	28.0	80.0
7	:SibSp	0.523008	0	0.0	8
8	:Parch	0.381594	0	0.0	6
9	:Ticket	nothing	"110152"	nothing	"WE/P 5735"
10	:Fare	32.2042	0.0	14.4542	512.329
11	:Cabin	nothing	"A10"	nothing	"T"
12	:Embarked	nothing	"C"	nothing	"S"

```
describe(data_train)
```

	variable	mean	min	median	m
1	:PassengerId	1100.5	892	1100.5	1309
2	:Pclass	2.26555	1	3.0	3
3	:Name	nothing	"Abbott, Master. Eugene Joseph"	nothing	"van Billiard, Ma
4	:Sex	nothing	"female"	nothing	"male"
5	:Age	30.2726	0.17	27.0	76.0
6	:SibSp	0.447368	0	0.0	8
7	:Parch	0.392344	0	0.0	9
8	:Ticket	nothing	"110469"	nothing	"W.E.P. 5734"
9	:Fare	35.6272	0.0	14.4542	512.329
10	:Cabin	nothing	"A11"	nothing	"G6"
11	:Embarked	nothing	"C"	nothing	"S"

```
describe(data_test)
```

Data Overview / Plots

Numeric Data

- Age
- SibSp (Siplings and Spouses)
- Parch (Partens and Children)
- Fare

Plots for numeric data

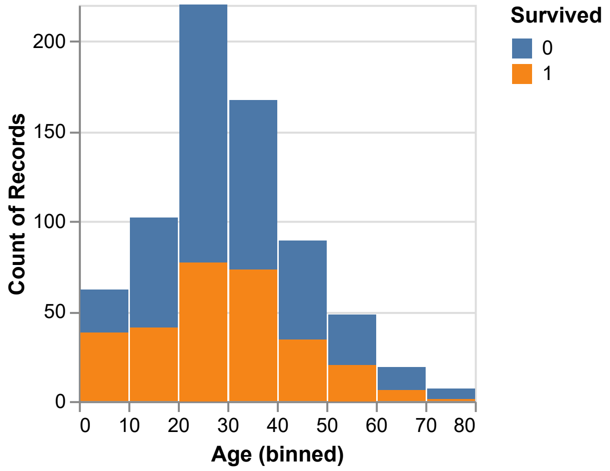
- Histograms to understand distributions
- Correlation Plot

numeric_df =

	Age	SibSp	Parch	Fare
1	22.0	1	0	7.25
2	38.0	1	0	71.2833
3	26.0	0	0	7.925
4	35.0	1	0	53.1
5	35.0	0	0	8.05
6	missing	0	0	8.4583
7	54.0	0	0	51.8625
8	2.0	3	1	21.075
9	27.0	0	2	11.1333
10	14.0	1	0	30.0708
more				
891	32.0	0	0	7.75

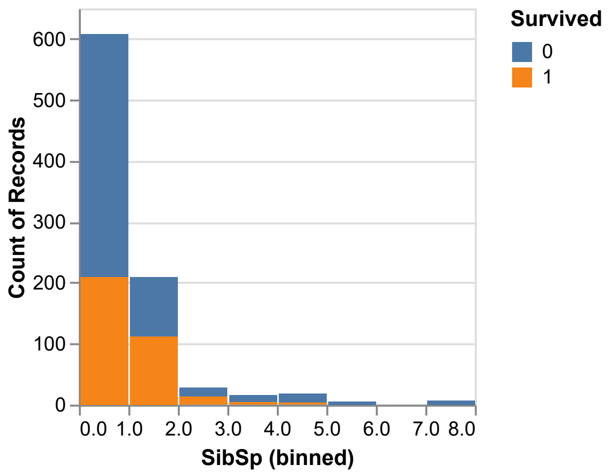
```
numeric_df = data_train[:,[:Age,:SibSp,:Parch,:Fare]]
```

hist_Age =



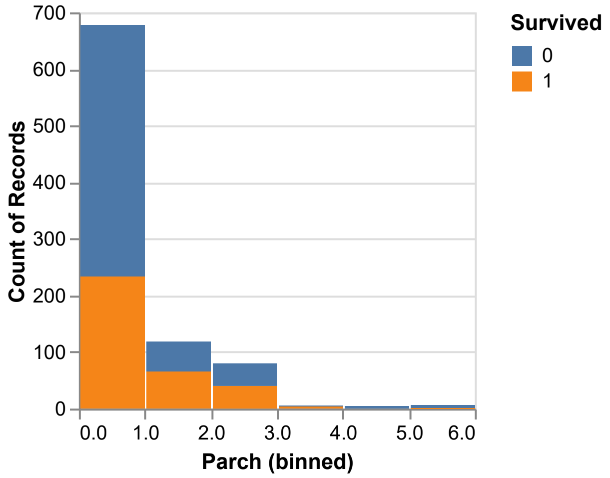

```
hist_Age= @vlplot(data=data_train)+
    @vlplot(:bar, x={:Age, bin=true}, y="count()", color={:Survived, type =
    "nominal"})
```

hist_SibSp =



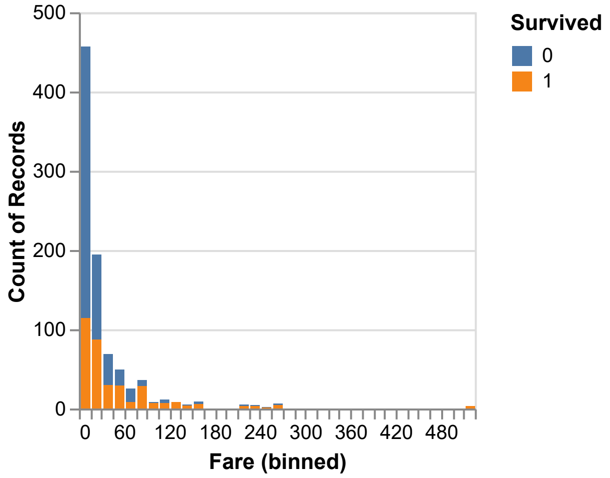

```
hist_SibSp= @vlplot(data=data_train)+
    @vlplot(:bar, x={:SibSp, bin=true}, y="count()", color={:Survived, type =
    "nominal"})
```

hist_Parch =




```
hist_Parch = @vlplot(data=data_train)+
    @vlplot(:bar, x={:Parch , bin=true}, y="count()", color={:Survived, type =
    "nominal"})
```

```
hist_Fare =
```



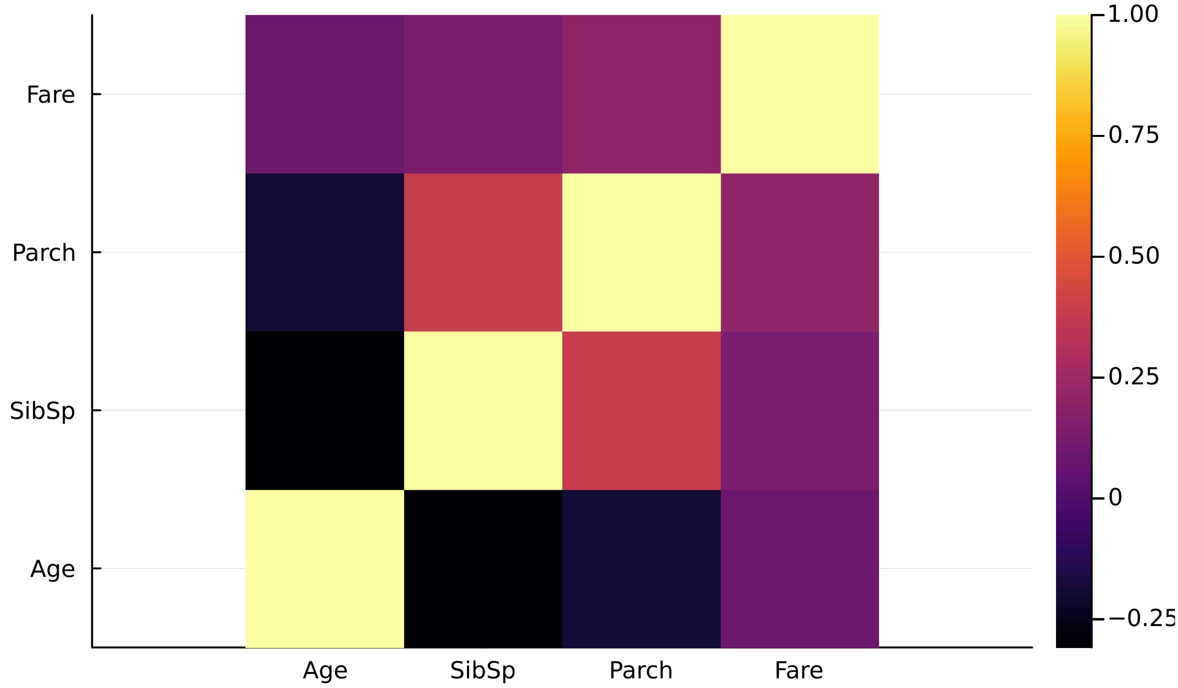

```
hist_Fare = @vlplot(data=data_train)+
    @vlplot(:bar, x={:Fare, bin={step=15}}, y="count()", color={:Survived, type =
    "nominal"})
```

```
cor_numeric = 4×4 Matrix{Float64}:
  1.0         -0.308247  -0.189119  0.0960667
 -0.308247     1.0        0.38382   0.138329
 -0.189119     0.38382    1.0       0.205119
  0.0960667    0.138329   0.205119  1.0
```

```
cor_numeric=cor(Matrix(dropmissing(numeric_df)))
```

```
GRBackend()
```

```
gr()
```




```
Plots.heatmap(names(numeric_df),names(numeric_df),cor_numeric,aspect_ratio = 1)
```

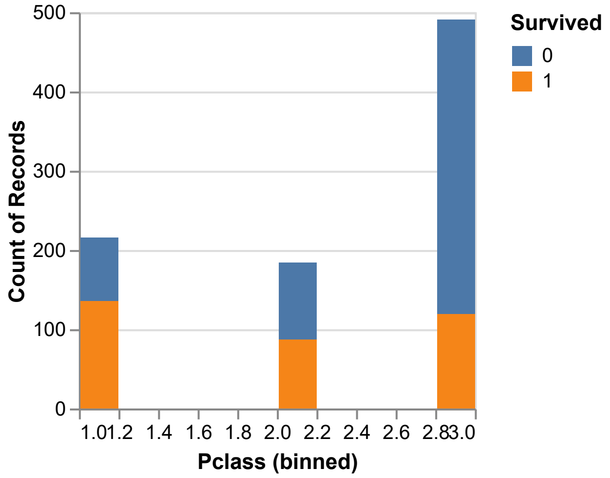
Categorical

- Survived
- Pclass
- Sex
- *(Cabin)*
- Embarked

Plots for Categorical Data

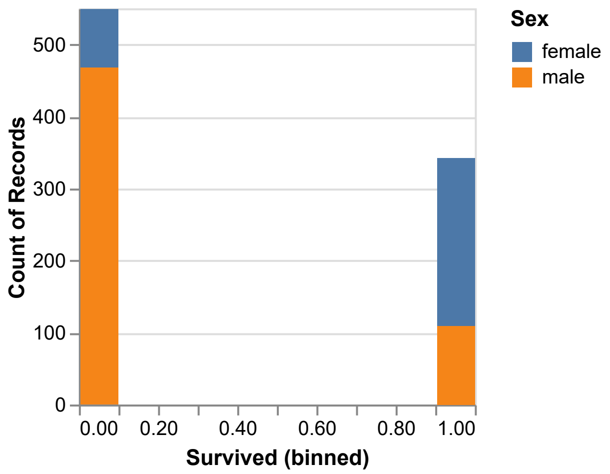
- Bar charts to understand balance of classes

bar_Pclass =



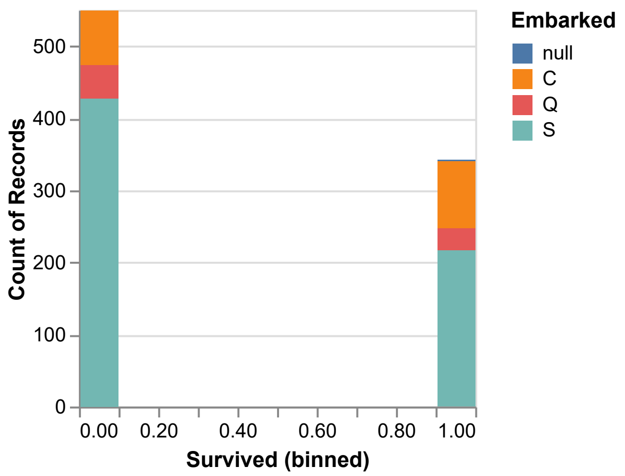

```
bar_Pclass = @vlplot(data=data_train)+
    @vlplot(:bar, x={:Pclass , bin=true}, y="count()", color={:Survived, type =
    "nominal"})
```

bar_Sex =




```
bar_Sex = @vlplot(data=data_train)+
    @vlplot(:bar, x={:Survived , bin=true}, y="count()", color={:Sex, type =
    "nominal"})
```

```
bar_Embarked =
```




```
bar_Embarked = @vlplot(data=data_train)+
    @vlplot(:bar, x={:Survived , bin=true}, y="count()", color={:Embarked, type =
    "nominal"})
```

Other Data

- Name
- Ticket

```
891-element SentinelArrays.ChainedVector{String, Vector{String}}:
 "Braund, Mr. Owen Harris"
 "Cumings, Mrs. John Bradley (Florence Briggs Thayer)"
 "Heikkinen, Miss. Laina"
 "Futrelle, Mrs. Jacques Heath (Lily May Peel)"
 "Allen, Mr. William Henry"
 "Moran, Mr. James"
 "McCarthy, Mr. Timothy J"
 ⋮
 "Rice, Mrs. William (Margaret Norton)"
 "Montvila, Rev. Juozas"
 "Graham, Miss. Margaret Edith"
 "Johnston, Miss. Catherine Helen \"Carrie\""
 "Behr, Mr. Karl Howell"
 "Dooley, Mr. Patrick"
```

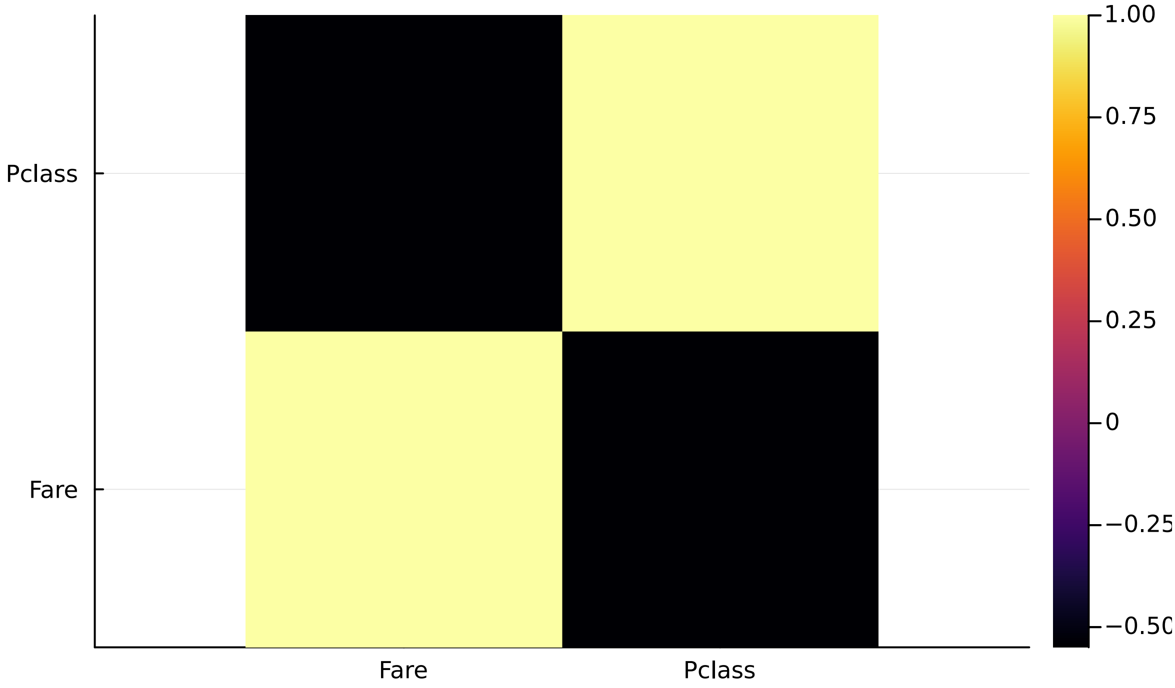
```
data_train.Name
```

```
891-element SentinelArrays.ChainedVector{String31, Vector{String31}}:
 "A/5 21171"
 "PC 17599"
 "STON/O2. 3101282"
 "113803"
 "373450"
 "330877"
 "17463"
 ⋮
 "382652"
 "211536"
 "112053"
 "W./C. 6607"
 "111369"
 "370376"
```

```
data_train.Ticket
```

```
2×2 Matrix{Float64}:
  1.0      -0.5495
 -0.5495    1.0
```

```
begin
    # correlation between Pclass and Fare
    dat = dropmissing(select(data_train, [:Fare, :Pclass]))
    cor_Pclass_Fare=cor(Matrix(dat))
end
```




```
begin
    gr()
    Plots.heatmap(names(dat),names(dat),cor_Pclass_Fare,aspect_ratio = 1)
end
```

Conclusion

Dealing with missing values

The training data set has a total of 891 samples.

- Age
  - 177 missing values (in training data)
  - replace missing values with mean $\pm$ sd
- Cabin
  - 687 missing values (in training data)
  - don't use Cabin as feature
- Embarked
  - 2 missing values (in training data)
  - drop missing
- Fare
  - high (negative) correlation with class
  - if missing: replace with mean for the corresponding class

Feature Selection

Y = Survived

- Exclude:
  - PassengerId
  - Name
  - Ticket
  - Cabin
- Include:
  - Pclass
  - Sex
  - Age
  - SibSp and Parch
  - Fare
  - Embarked

Data Cleaning

```
clean_function = Main.var"final_function.jl"
```

```
clean_function = ingredients("final_function.jl")
```

(

	Age	Relatives	pclass_3	pclass_1	pclass_2	sex_male	sex_female	Fa
1	0.334188	0.1	true	false	false	true	false	0.014
2	0.51786	0.1	false	true	false	false	true	0.139
3	0.380106	0.0	true	false	false	false	true	0.015
4	0.483422	0.1	false	true	false	false	true	0.103
5	0.483422	0.0	true	false	false	true	false	0.015
6	0.0582966	0.0	true	false	false	true	false	0.016
7	0.701532	0.0	false	true	false	true	false	0.101
8	0.104597	0.4	true	false	false	true	false	0.041
9	0.391585	0.2	true	false	false	false	true	0.021
10	0.242352	0.1	false	false	true	false	true	0.058
more								
889	0.448983	0.0	true	false	false	true	false	0.015

```
df_train_X, train_y = clean_function.final_function_train(data_train)
```

df_test_X =

	Age	Relatives	pclass_3	pclass_2	pclass_1	sex_male	sex_female	Fare
1	0.452723	0.0	true	false	false	true	false	0.0152816
2	0.617566	0.1	true	false	false	false	true	0.0136631
3	0.815377	0.0	false	true	false	true	false	0.0189087
4	0.353818	0.0	true	false	false	true	false	0.0169081
5	0.287881	0.2	true	false	false	false	true	0.0239836
6	0.182382	0.0	true	false	false	true	false	0.018006
7	0.39338	0.0	true	false	false	false	true	0.0148912
8	0.34063	0.2	false	true	false	true	false	0.0566042
9	0.235131	0.0	true	false	false	false	true	0.0141105
10	0.274693	0.2	true	false	false	true	false	0.0471377
more								
418	0.219546	0.2	true	false	false	true	false	0.0436405

```
df_test_X = clean_function.final_function_test(data_test)
```

```
(889×11 Matrix{Float64}:                                                          , 418×11 Matrix{Float
 0.334188   0.1  1.0  0.0  0.0  1.0  0.0  0.0141511  1.0  0.0  0.0   0.452723  0.0  1.0
 0.51786    0.1  0.0  1.0  0.0  0.0  1.0  0.139136   0.0  1.0  0.0   0.617566  0.1  1.0
 0.380106   0.0  1.0  0.0  0.0  0.0  1.0  0.0154686  1.0  0.0  0.0   0.815377  0.0  0.0
 0.483422   0.1  0.0  1.0  0.0  0.0  1.0  0.103644   1.0  0.0  0.0   0.353818  0.0  1.0
 0.483422   0.0  1.0  0.0  0.0  1.0  0.0  0.0157126  1.0  0.0  0.0   0.287881  0.2  1.0
 0.0582966  0.0  1.0  0.0  0.0  1.0  0.0  0.0165095  0.0  0.0  1.0   0.182382  0.0  1.0
 0.701532   0.0  0.0  1.0  0.0  1.0  0.0  0.101229   1.0  0.0  0.0   0.39338   0.0  1.0
 ⋮                              ⋮                              ⋮      ⋮
 0.52934    0.5  1.0  0.0  0.0  0.0  1.0  0.0568482  0.0  0.0  1.0   0.367005  0.0  1.0
 0.391585   0.0  0.0  0.0  1.0  1.0  0.0  0.0253743  1.0  0.0  0.0   0.610549  0.0  1.0
 0.299749   0.0  0.0  1.0  0.0  0.0  1.0  0.0585561  1.0  0.0  0.0   0.512066  0.0  0.0
 0.372144   0.3  1.0  0.0  0.0  0.0  1.0  0.0457714  1.0  0.0  0.0   0.505473  0.0  1.0
 0.380106   0.0  0.0  1.0  0.0  1.0  0.0  0.0585561  0.0  1.0  0.0   0.224796  0.0  1.0
 0.448983   0.0  1.0  0.0  0.0  1.0  0.0  0.015127   0.0  0.0  1.0   0.219546  0.2  1.0
```

```
train_X, test_X = Matrix{Float64}(df_train_X), Matrix{Float64}(df_test_X)
```

Model Building

- Linear Regression
- Ridge Regression
- LASSO Regression
- Logistic Regression
- K Nearest Neighbor
- Decision Tree
- Random Forest
- Naive Bayes

```
PyObject <class 'sklearn.linear_model._base.LinearRegression'>
```

```
@sk_import linear_model: LinearRegression
```

```
PyObject <class 'sklearn.linear_model._ridge.Ridge'>
```

```
@sk_import linear_model: Ridge
```

```
PyObject <class 'sklearn.linear_model._coordinate_descent.Lasso'>
```

```
@sk_import linear_model: Lasso
```

```
PyObject <class 'sklearn.linear_model._logistic.LogisticRegression'>
```

```
@sk_import linear_model: LogisticRegression
```

```
PyObject <class 'sklearn.neighbors._classification.KNeighborsClassifier'>
```

```
@sk_import neighbors: KNeighborsClassifier
```

```
PyObject <class 'sklearn.tree._classes.DecisionTreeClassifier'>
```

```
@sk_import tree: DecisionTreeClassifier
```

```
PyObject <class 'sklearn.ensemble._forest.RandomForestClassifier'>
```

```
@sk_import ensemble: RandomForestClassifier
```

```
PyObject <class 'sklearn.naive_bayes.GaussianNB'>
```

```
@sk_import naive_bayes: GaussianNB
```

```
PyObject <class 'sklearn.ensemble._voting.VotingClassifier'>
```

```
@sk_import ensemble: VotingClassifier
```

Gridsearching

```
bestfit = Main.var"find_best_fit.jl"
```

```
bestfit = ingredients("find_best_fit.jl")
```

```
Module model_selection has been ported to Julia - try `import ScikitLearn: CrossValidation` instead
```

```
findfit = find_best_fit (generic function with 1 method)
```

```
findfit = bestfit.find_best_fit
```

```
(LogisticRegression
LogisticRegression(C=1.4, class_weight='balanced'), GridSearchCV
estimator: LogisticRegression
LogisticRegression )
```

```
findfit(train_X,train_y,LogisticRegression(),Dict("C"=> 0.1:0.1:1.5, "class_weight"
=> ("balanced", nothing)))
# using default
```

```
(KNeighborsClassifier
KNeighborsClassifier(n_neighbors=3), GridSearchCV
estimator: KNeighborsClassifier
KNeighborsClassifier )
```

```
findfit(train_X,train_y,KNeighborsClassifier(),Dict("n_neighbors" => 1:10, "weights"
=> ("uniform", "distance")))
# using n_neighbors=3
```

```
begin
    minsplits_dt = []
    for i in 1:5
    push!(minsplits_dt,findfit(train_X,train_y,DecisionTreeClassifier(),
    Dict("criterion" => ("gini", "entropy", "log_loss"),"class_weight" =>
    ("balanced", nothing), "min_samples_split" => 6:2:20)))
    end
end
# using default
```

[(DecisionTreeClassifier
DecisionTreeClassifier(criterion='log_loss', min_samples_split=16), GridS
estimator: Decis
DecisionTr

```
minsplits_dt # using min_samples_split=16
```

```
begin
    minsplits = []
    for i in 1:5
        push!(minsplits,findfit(train_X,train_y,RandomForestClassifier(),
        Dict("min_samples_split" => 6:10)))
    end
end
```

[(RandomForestClassifier
RandomForestClassifier(min_samples_split=7), GridSearchCV
estimator: RandomForestClassifier
RandomForestClassifier), (Ra

```
minsplits #using min_samples_split=8
```

Crossvalidation Scoring

```
using ScikitLearn   .CrossValidation: cross_val_score
```

Linear Regression

cv_linear = [0.332458, 0.36783, 0.37807, 0.34009, 0.438664]

```
cv_linear = cross_val_score(LinearRegression(),train_X,train_y,cv=5)
```

```
print(mean(cv_linear))
```

0.3714223329442524

Ridge Regression

cv_ridge = [0.33108, 0.369516, 0.381255, 0.34096, 0.436708]

```
cv_ridge = cross_val_score(Ridge(),train_X,train_y,cv=5)
```

```
print(mean(cv_ridge))
```

0.3719035258800675

LASSO Regression

```
cv_lasso = [-0.0228073, -0.0238521, -0.000177285, -0.00315337, -0.00708916]
```

```
cv_lasso = cross_val_score(Lasso(),train_X,train_y,cv=5)
```

```
print(mean(cv_lasso))
```

```
-0.011415839277551543
```

Logistic Regression

```
cv_logistic = [0.792135, 0.808989, 0.775281, 0.775281, 0.819209]
```

```
cv_logistic = cross_val_score(LogisticRegression(),train_X,train_y,cv=5)
```

```
print(mean(cv_logistic))
```

```
0.7941788865612899
```

K Nearest Neighbor

```
cv_kneighbors = [0.764045, 0.786517, 0.820225, 0.792135, 0.79661]
```

```
cv_kneighbors =
cross_val_score(KNeighborsClassifier(n_neighbors=3),train_X,train_y,cv=5)
```

```
print(mean(cv_kneighbors))
```

```
0.7919063035612266
```

Decision Tree

```
cv_destree = [0.797753, 0.786517, 0.825843, 0.797753, 0.779661]
```

```
cv_destree =
cross_val_score(DecisionTreeClassifier(min_samples_split=16),train_X,train_y,cv=5)
```

```
print(mean(cv_destree))
```

```
0.7975052370976957
```

Random Forest

```
cv_randforest = [0.797753, 0.792135, 0.870787, 0.825843, 0.813559]
```

```
cv_randforest =
cross_val_score(RandomForestClassifier(min_samples_split=8),train_X,train_y,cv=5)
```

```
print(mean(cv_randforest))
```

```
0.8200152351932966
```

Naive Bayes

cv_naibay = [0.747191, 0.780899, 0.775281, 0.820225, 0.819209]

```
cv_naibay = cross_val_score(GaussianNB(),train_X,train_y,cv=5)
```

```
print(mean(cv_naibay))
```

0.7885609090332

Voting Classifier

voting_clf_soft =

VotingClassifier

lr | knn | dt

LogisticRegression | KNeighborsClassifier | DecisionTreeClassifier | Rando

```
voting_clf_soft = VotingClassifier(estimators = [
    ("lr", LogisticRegression()),
    ("knn", KNeighborsClassifier(n_neighbors=3)),
    ("dt", DecisionTreeClassifier(min_samples_split=16)),
    ("rf", RandomForestClassifier(min_samples_split=8)),
    ("gnb", GaussianNB())],
    voting = "soft")
```

```
voting_clf_hard = VotingClassifier(estimators = [
    ("lr", LogisticRegression()),
    ("knn", KNeighborsClassifier(n_neighbors=3)),
    ("dt", DecisionTreeClassifier(min_samples_split=16)),
    ("rf", RandomForestClassifier(min_samples_split=8)),
    ("gnb", GaussianNB())],
    voting = "hard");
```

```
voting_clf_test = VotingClassifier(estimators = [
    ("knn", KNeighborsClassifier(n_neighbors=3)),
    ("dt", DecisionTreeClassifier(min_samples_split=16)),
    ("rf", RandomForestClassifier(min_samples_split=8))],
    voting = "hard");
```

cv_test = [0.803371, 0.792135, 0.865169, 0.803371, 0.79661]

```
cv_test = cross_val_score(voting_clf_test,train_X,train_y,cv=5)
```

```
print(mean(cv_test))
```

0.8121310226623499

cv_soft = [0.808989, 0.797753, 0.814607, 0.825843, 0.80226]

```
cv_soft = cross_val_score(voting_clf_soft,train_X,train_y,cv=5)
```

cv_hard = [0.831461, 0.803371, 0.825843, 0.820225, 0.819209]

```
cv_hard = cross_val_score(voting_clf_hard,train_X,train_y,cv=5)
```

```
print(mean(cv_soft))
```

0.809890179648321

```
print(mean(cv_hard))
```

0.8200215831905033

Model Fitting

[0, 1, 0, 0, 2, 0, 0, 2, 0, 2, 0, 0, 1, 1, 1, 1, 0, 0, 1, 0, more ,0, 2, 0, 1, 0, 0, 0, 0,

```
data_test.Relatives = data_test.SibSp + data_test.Parch
```

prov_X_test = 418×5 Matrix{Float64}:

```
  892.0  3.0  34.5     0.0    7.8292
  893.0  3.0  47.0     1.0    7.0
  894.0  2.0  62.0     0.0    9.6875
  895.0  3.0  27.0     0.0    8.6625
  896.0  3.0  22.0     2.0   12.2875
  897.0  3.0  14.0     0.0    9.225
  898.0  3.0  30.0     0.0    7.6292
    ⋮
 1304.0  3.0  28.0     0.0    7.775
 1305.0  3.0  46.4679  0.0    8.05
 1306.0  1.0  39.0     0.0  108.9
 1307.0  3.0  38.5     0.0    7.25
 1308.0  3.0  17.2163  0.0    8.05
 1309.0  3.0  16.8182  2.0   22.3583
```

```
prov_X_test=Matrix(dropmissing(data_test[:,[:PassengerId, :Pclass, :Age, :Relatives,
:Fare ]]))
```

model =

▾ RandomForestClassifier

RandomForestClassifier(min_samples_split=8)

```
model = fit!(RandomForestClassifier(min_samples_split=8),train_X,train_y)
```

y_pred =

[0, 1, 0, 0, 1, 0, 1, 0, 1, 1, 0, 0, 1, 0, 1, 1, 0, 0, 1, 1, more ,1, 1, 0, 1, 1, 0, 1, 0,

```
y_pred = predict(model,test_X)
```

```
Enter cell code...
```

df_submission =

	PassengerId	Survived
1	892	0
2	893	1
3	894	0
4	895	0
5	896	1
6	897	0
7	898	1
8	899	0
9	900	1
10	901	1
more		
418	1309	0

```
df_submission = DataFrame("PassengerId" => 892:1309,
                          "Survived" => y_pred)
```

submissionfile = "submission.csv"

```
submissionfile = "submission.csv"
```

"submission.csv"

```
CSV.write(submissionfile, df_submission)
```