Ames House Price Data - Data Cleaning

> Juptyer notebook in Julia 0.5.2, using Machine Learning modules written by author

This notebook describes cleaning of the data. Additional model-dependent transformations to the data (eg, Cox-Box transformations and one-hot encodings for linear models) are described in the notebook for the relevant model.

Index

Reviewing data types

Some of our models are able to handle categorical and ordinal data simultaneously. These models will expect categorical data to be of `String` or `Char` type. In the house price data the attribute `MSSubClass` (building class) is categorical but is represented by an `Int` datatype. We fix this now, after loading the data:

In [20]:
```
push!(LOAD_PATH, pwd()) # Allow loading of modules from current dir
ectory
using DataFrames, Preprocess

train = readtable("0.kaggle/train.csv")
test = readtable("0.kaggle/test.csv")
combined = vcat(train,test)
head(combined)
```

Out[20]:

	Id	MSSubClass	MSZoning	LotFrontage	LotArea	Street	Alley	LotShape	Land
1	1	60	RL	65	8450	Pave	NA	Reg	Lvl
2	2	20	RL	80	9600	Pave	NA	Reg	Lvl
3	3	60	RL	68	11250	Pave	NA	IR1	Lvl
4	4	70	RL	60	9550	Pave	NA	IR1	Lvl
5	5	60	RL	84	14260	Pave	NA	IR1	Lvl
6	6	50	RL	85	14115	Pave	NA	IR1	Lvl

In [21]:
```
combined[:MSSubClass] = map(combined[:MSSubClass]) do x
    string("__", x)
end
combined[:MSSubClass]  = convert(DataArrays.DataArray{String,1}, co
mbined[:MSSubClass]);
```

N.B. Alternatively, we could have reviewed the datatypes using the interactive function `review_ordinals!`.

Appraising the quality of the data

The third column in the data frame constructed below shows the number of NA's in each field. Other columns give an idea of how they are distributed within the data. This information guides subsequent cleaning below. The various fields are described further here (0.kaggle/data_summary.md). (To see the full list of field meta data, remove the second line below.)

In [22]:
```
meta = get_meta(combined)
head(meta)
```

Out[22]:

	field	type	n_values	n_nas	percent_nas	row_of_first_non_na	nas_afte
1	Id	Int64	2919	0	0.0	1	0
2	MSSubClass	String	16	0	0.0	1	0
3	MSZoning	String	6	4	0.1	1	4
4	LotFrontage	Int64	129	486	16.6	1	486
5	LotArea	Int64	1951	0	0.0	1	0
6	Street	String	2	0	0.0	1	0

Determining outliers

The following plot reveals two properties with huge living area but sale prices around the median value.

In [23]:
```
using Plots
plotlyjs()

scatter(train[:GrLivArea], train[:SalePrice], markersize=1.5)
```

Out[23]:

In [24]:
```
median(train[:SalePrice])
```

Out[24]: 163000.0

We remove these from the data and record the new size of our combined data set (`Id=1461` is the first pattern in the test portion of the data):

In [25]:
```
mask_bad = (combined[:GrLivArea] .> 4000) .* (combined[:Id] .< 1461
)
mask_good = !mask_bad
combined = combined[mask_good,:]

nrows, ncols = size(combined)
```

Out[25]: (2915,81)

Cleaning

According to the documentation (0.kaggle/detailed_documentation.txt), "NA" in `:Alley` means "no access" not "unknown value". Correct this:

In [26]:
```
combined[:Alley] = convert(Array, combined[:Alley], "NoAccess");
```

Similar remarks apply to other fields:

In [27]:
```
combined[:PoolQC] = convert(Array, combined[:PoolQC], "None")
combined[:MiscFeature] = convert(Array, combined[:MiscFeature], "No
ne")
combined[:Fence] = convert(Array, combined[:Fence], "None")
combined[:FireplaceQu] = convert(Array, combined[:FireplaceQu], "No
ne")
combined[:Functional] = convert(Array, combined[:Functional], "Typ"
);
```

We dump ordinal fields with more than 50% NAs. (Note that `.*` is broadcast version of logical AND.)

In [28]:
```
bad_fields = meta[(meta[:percent_nas] .> 50) .* (meta[:type] .!= St
ring), :field]
bad_fields = collect(bad_fields)
filter!(x -> x != :SalePrice, bad_fields) # exclude target variable
which is undefined in test portion of data
delete!(combined, bad_fields);
```

Dump categorical fields with more than 80% NAs:

In [29]:
```
bad_fields = meta[(meta[:percent_nas] .> 80) .* (meta[:type] .== St
ring), :field]
delete!(combined, bad_fields);
```

Make "NA" a genuine value in remaining categoricals:

In [30]:
```
for f in names(combined)
    if eltype(combined[f]) == String
        combined[f] = convert(Array{String,1}, combined[f], "_NA")
    end
end
```

To clean `LotFrontage` we shall suppose that, within each neighborhood, `:LotFrontage` (which is measured in linear feet) correlates roughly with the *square root* of `:LotArea` (measured in square feet). So letting `ratio` refers to `:LotFrontage/sqrt(:LotArea)` we begin by finding the median value of `ratio` in each neighborhood and storing this in a dictionary:

In [31]:
```
temp = by(combined, [:Neighborhood], df -> median(dropna(df[:LotFro
ntage]./sqrt(df[:LotArea]))))
ratio = Dict{String,Float64}()
for i in 1:first(size(temp))
    ratio[temp[i,:Neighborhood]]=temp[i,:x1]
end
```

We now use this ratio to imputate NA values in `:LotFrontage`:

In [32]:
```
combined[:LotFrontage] =  convert(DataArray{Float64,1}, combined[:L
otFrontage])
for i in 1:nrows
    if isna(combined[i,:LotFrontage])
        combined[i,:LotFrontage] = ratio[combined[i,:Neighborhood]]
*sqrt(combined[i,:LotArea])
    end
end
```

Drop `:Utilities` as the field is virtually constant:

In [33]:
```
delete!(combined, :Utilities);
```

We replace `MasVnrArea` (Masonry Veneer area) NAs with zeros:

In [34]:
```
combined[:MasVnrArea] = convert(Array, combined[:MasVnrArea], 0.0);
```

Remaining NAs are replaced with median values:

In [35]:
```
meta = get_meta(combined)
fields_to_fix = collect(meta[meta[:n_nas] .> 0,:][:field])
filter!(x -> x != :SalePrice, fields_to_fix) # exclude target varia
ble
for f in fields_to_fix
    m = median(dropna(combined[f]))
    combined[f] = convert(Array{Float64, 1}, combined[f], m)
end
```

It turns out `:MSSubClass` takes on a value in the test set not appearing in the train set. We replace it with the mode value:

In [36]:
```
combined[:MSSubClass] = map(combined[:MSSubClass]) do x
    x == "_150" ? "_20" : x
end;
```

Transforming the target variable

As we shall be optimizing our models to minimize log-root-mean-squared error, it is convenient to replace the target variable in our modelling, `:SalePrice`, with its logarithm:

In [37]:
```
combined[:target] = log(combined[:SalePrice])
delete!(combined, :SalePrice);
```

Saving the cleaned data to file

In [38]:
```
train = combined[!isna(combined[:target]),:]
test = combined[isna(combined[:target]),:]

writetable("2.cleaned/combined.csv", combined)
writetable("2.cleaned/train.csv", train)
writetable("2.cleaned/test.csv", test)
```

In [ ]:
```

```