Kaggle Housing Price Prediction Challenge

Art Tay

```
# Libraries
import pandas as pd
import numpy as np
from sklearn_pandas import dataframe_mapper
```

```
# Read in data
train = pd.read_csv("data/train.csv")
test = pd.read_csv("data/test.csv")
#train.info()
#test.info()
```

Functions

Generic Data Cleaning

```
# Converts all object (string) columns to
# be categorical.
# @param: train - a pandas dataframe
# Note: Technically unnecessary because pd.get_dummies will
# dummy string, objects and category.
def to_cat(train):
    train[train.select_dtypes(['object']).columns] = (
        train.select_dtypes(['object'])
        .apply(lambda x: x.astype('category'))
    )
    return train
```

Penalized Regression

Data Cleaning

```
# Code string columns as categorical
train['MSSubClass'] = train['MSSubClass'].astype('category')

train[train.select_dtypes(['object']).columns] = (
    train.select_dtypes(['object'])
    .apply(lambda x: x.astype('category'))
)

# Feature Engineering
# NewGarage
train['NewGarage'] = (
    np.where(train['GarageYrBlt'].isnull(), 0,
        np.where(train['GarageYrBlt'] > train['YearBuilt'], 1, 0))
)

# YearSinceRmdl
train['YearSinceRmdl'] = 2016 - train['YearRemodAdd']

# Rmdl
train['Rmdl'] = np.where(train['YearBuilt'] < train['YearRemodAdd'], 1, 0)

# TotalPorchArea
train['TotalPorchArea'] = (
    train['WoodDeckSF'] + train['OpenPorchSF'] +
    train['EnclosedPorch'] + train['3SsnPorch'] +
    train['ScreenPorch']
)

#PorchYes
train['PorchYes'] = np.where(train['TotalPorchArea'] > 0, 1, 0)

# TotalFinishedBsmt
train['TotalFinishedBsmt'] = train['BsmtFinSF1'] + train['BsmtFinSF2']

# PercentFinishedBsmt
train['PercentFinishedBsmt'] = np.where(train['TotalBsmtSF'] > 0,
    train['TotalFinishedBsmt'] / train['TotalBsmtSF'] * 100, 0)

# TotalSqFt
train['TotalSqFt'] = train['GrLivArea'] + train['TotalFinishedBsmt']
```

```python
# PercentLowQual
train['PercentLowQual'] = train['LowQualFinSF'] * 100 / train['TotalSqFt']

# IsNew
train['IsNew'] = np.where(train['YrSold'] == train['YearRemodAdd'], 1, 0)

# House_Age
train['House_age'] = train['YrSold'] - train['YearRemodAdd']

# NeighRich
train['NeighRich'] = np.select(
    condlist = [
        train['Neighborhood'] == ('StoneBr' or 'NridgHt' or 'NoRidge'),
        train['Neighborhood'] == ('MeadowV' or 'IDOTRR' or 'BrDale')
    ],
    choicelist = [2, 0],
    default = 1
)
```

```python
def get_col(train, x):
    return train[x].head()

get_col(train, "NeighRich")

# Converts a categorical column to be on an ordeal scale.
# Scale was determined ad-hoc.
# @param: train - a pandas dataframe
# @param: col_name - a string name of the column to be converted
def ord_scale_1(train, col_name):
    ret = np.select(
        condlist = [
            train[col_name] == "Ex",
            train[col_name] == "Gd",
            train[col_name] == "TA",
            train[col_name] == "Fa",
            train[col_name] == "Po"
        ],
        choicelist = [5, 4, 3, 2, 1],
        default = 0
    )
    return ret

def ord_scale_2(train, col_name):
    ret = np.select(
```

```python
        condlist = [
            train[col_name] == "GLQ",
            train[col_name] == "ALQ",
            train[col_name] == "BLQ",
            train[col_name] == "REC",
            train[col_name] == "LwQ",
            train[col_name] == "Unf",
        ],
        choicelist = [6, 5, 4, 3, 2, 1],
        default = 0
    )
    return ret

# Test
print(np.unique(ord_scale_1(train, "ExterCond")))
print(np.unique(ord_scale_1(train, "GarageQual")))
print(np.unique(ord_scale_2(train, "BsmtFinType2")))
```

```
[1 2 3 4 5]
[0 1 2 3 4 5]
[0 1 2 4 5 6]
```

```python
# Ordinal Recoding
train['LotShape'] = np.select(
    condlist = [
        train['LotShape'] == "Reg",
        train['LotShape'] == "IR1",
        train['LotShape'] == "IR2",
        train['LotShape'] == "IR3"
    ],
    choicelist = [3, 2, 1, 0]
)

train['LandSlope'] = np.select(
    condlist = [
        train['LandSlope'] == "Gtl",
        train['LandSlope'] == "Mod",
        train['LandSlope'] == "Sev"
    ],
    choicelist = [2, 1, 0]
)

train['BsmtExposure'] = np.select(
    condlist = [
```

```python
        train['BsmtExposure'] == "Gd",
        train['BsmtExposure'] == "Av",
        train['BsmtExposure'] == "Mn",
        train['BsmtExposure'] == "No"
    ],
    choicelist = [4, 3, 2, 1],
    default = 0
)

train['GarageFinish'] = np.select(
    condlist = [
        train['GarageFinish'] == "Fin",
        train['GarageFinish'] == "RFn",
        train['GarageFinish'] == "Unf",
    ],
    choicelist = [3, 2, 1],
    default = 0
)

train['Functional'] = np.select(
    condlist = [
        train['Functional'] == "Typ",
        train['Functional'] == "Min1",
        train['Functional'] == "Min2",
        train['Functional'] == "Mod",
        train['Functional'] == "Maj1",
        train['Functional'] == "Maj2",
        train['Functional'] == "Sev",
        train['Functional'] == "Sal"
    ],
    choicelist = [7, 6, 5, 4, 3, 2, 1, 0]
)
```

```python
# Extract response
if 'SalePrice' in train:
    response = train['SalePrice']
    train = train.drop('SalePrice', axis = 1)
else:
    train = train

# Dummies
train = pd.get_dummies(train, drop_first = True)

# Center + Scale
```

```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
train = pd.DataFrame(scaler.fit_transform(train), columns = train.columns)

# knnImpute
from sklearn.impute import KNNImputer
imputer = KNNImputer(n_neighbors = 5)
train = pd.DataFrame(imputer.fit_transform(train), columns = train.columns)

# Reverse center + scale for other preprocessing methods.
train = pd.DataFrame(scaler.inverse_transform(train), columns = train.columns)

## NZV - remove all variable with less than 5% variance.
from sklearn.feature_selection import VarianceThreshold
selector = VarianceThreshold(threshold = 0.05)
train = train.loc[:, selector.fit(train).get_support()]

# Corr
def drop_high_cor(df, threshold = 0.9):
    # Create correlation matrix
    corr_matrix = df.corr().abs()

    # Select upper triangle of correlation matrix
    upper = corr_matrix.where(np.triu(np.ones(corr_matrix.shape), k=1).astype(bool))

    # Find features with correlation greater than 0.95
    to_drop = [column for column in upper.columns if any(upper[column] > threshold)]

    print(to_drop)

    # Drop features
    return df.drop(to_drop, axis=1)

train = drop_high_cor(train, threshold = 0.9)

# Splines

# Yeo-Johnson

# Log Price
```

```
['YearSinceRmdl', 'TotalFinishedBsmt', 'PercentLowQual', 'House_age', 'RoofStyle_Hip', 'Exte
```

```
# Unit Test
print(train.head())
train.isnull().sum().sum()
#train['NeighRich'].unique()
#train['SalePrice'].isnull().sum()

#test.info()
#if 'SalePrice' in train:
    #test_69 = train.drop('SalePrice', axis = 1)
#else:
    #test_69 = train
#print(test_69.equals(test))
```

```
    Id  LotFrontage  LotArea  LotShape  LandSlope  OverallQual  OverallCond  \
0  1.0         65.0   8450.0       3.0        2.0          7.0          5.0
1  2.0         80.0   9600.0       3.0        2.0          6.0          8.0
2  3.0         68.0  11250.0       2.0        2.0          7.0          5.0
3  4.0         60.0   9550.0       2.0        2.0          7.0          5.0
4  5.0         84.0  14260.0       2.0        2.0          8.0          5.0

   YearBuilt  YearRemodAdd    MasVnrArea  ...  FireplaceQu_TA  \
0     2003.0        2003.0  1.960000e+02  ...   -2.775558e-17
1     1976.0        1976.0 -1.421085e-14  ...    1.000000e+00
2     2001.0        2002.0  1.620000e+02  ...    1.000000e+00
3     1915.0        1970.0 -1.421085e-14  ...   -2.775558e-17
4     2000.0        2000.0  3.500000e+02  ...    1.000000e+00

   GarageType_Attchd  GarageType_BuiltIn  GarageType_Detchd  GarageQual_TA  \
0                1.0                 0.0                0.0            1.0
1                1.0                 0.0                0.0            1.0
2                1.0                 0.0                0.0            1.0
3                0.0                 0.0                1.0            1.0
4                1.0                 0.0                0.0            1.0

   GarageCond_TA  PavedDrive_Y  Fence_MnPrv  SaleType_WD  SaleCondition_Normal
0            1.0           1.0          0.0          1.0          1.000000e+00
1            1.0           1.0          0.0          1.0          1.000000e+00
2            1.0           1.0          0.0          1.0          1.000000e+00
3            1.0           1.0          0.0          1.0         -1.110223e-16
4            1.0           1.0          0.0          1.0          1.000000e+00

[5 rows x 108 columns]
```

```
0
```

Random Forest

Gradient Boosting

Neutral Network