Ames Housing Price Prediction — Complete ML Project with Python

10 min read · Apr 1, 2019

Kamski John Follow

Simple Exploratory Data analysis and Data pre-processing

Hello everyone. I was recently learning about Exploratory Data Analysis (EDA) and Data pre-processing and thought that it will be worth sharing what I've learned so far. I will be using Ames housing dataset from Kaggle here. This is just my second Kaggle Project and I am still learning and learning..... So suggestions and comments are more than welcome!!!

I've no idea, why i placed this image here. So dont ask.

I will be using python throughout the project and will concentrate more on EDA and Data pre-processing than modelling. Let's begin by understanding the problem. The dataset consists of 79 features almost describing everything that makes a house (even things that we never really care about) and the task is to predict the *SalePrice* of a house.

Understanding the data

Let me first import libraries and the dataset.

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

```
import warnings
warnings.filterwarnings('ignore')
pd.set_option('display.max_rows', 500)
pd.set_option('display.max_columns', 500)
plt.style.use('fivethirtyeight')
from scipy import stats
%matplotlib inline

train = pd.read_csv('../MyPC/Ames/train.csv')
test = pd.read_csv('../MyPC/Ames/test.csv')
```

The best way to begin any ML project is by understanding the data. The better we understand the data, the more we can use it to help ourselves in manipulating it to make better predictions. So let's look at the available features.

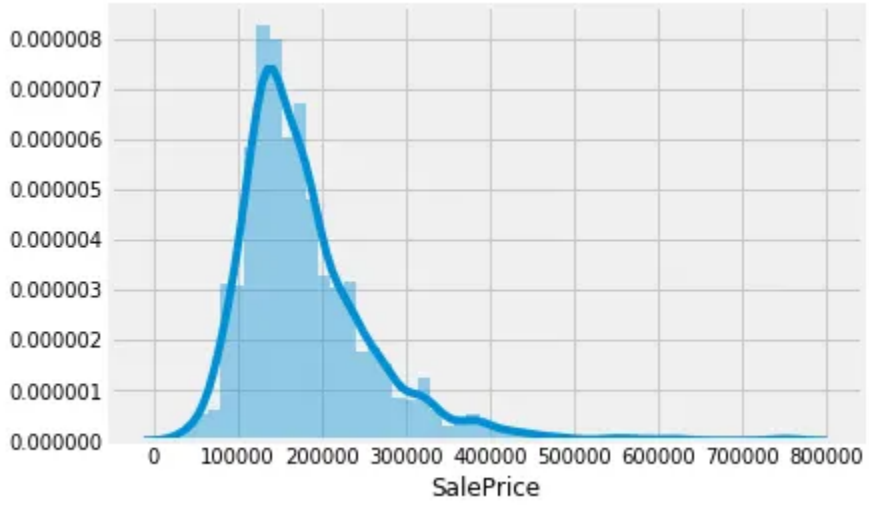
```
In [6]: 1 train.columns

Out[6]: Index(['Id', 'MSSubClass', 'MSZoning', 'LotFrontage', 'LotArea', 'Street',
               'Alley', 'LotShape', 'LandContour', 'Utilities', 'LotConfig',
               'LandSlope', 'Neighborhood', 'Condition1', 'Condition2', 'BldgType',
               'HouseStyle', 'OverallQual', 'OverallCond', 'YearBuilt', 'YearRemodAdd',
               'RoofStyle', 'RoofMatl', 'Exterior1st', 'Exterior2nd', 'MasVnrType',
               'MasVnrArea', 'ExterQual', 'ExterCond', 'Foundation', 'BsmtQual',
               'BsmtCond', 'BsmtExposure', 'BsmtFinType1', 'BsmtFinSF1',
               'BsmtFinType2', 'BsmtFinSF2', 'BsmtUnfSF', 'TotalBsmtSF', 'Heating',
               'HeatingQC', 'CentralAir', 'Electrical', '1stFlrSF', '2ndFlrSF',
               'LowQualFinSF', 'GrLivArea', 'BsmtFullBath', 'BsmtHalfBath', 'FullBath',
               'HalfBath', 'BedroomAbvGr', 'KitchenAbvGr', 'KitchenQual',
               'TotRmsAbvGrd', 'Functional', 'Fireplaces', 'FireplaceQu', 'GarageType',
               'GarageYrBlt', 'GarageFinish', 'GarageCars', 'GarageArea', 'GarageQual',
               'GarageCond', 'PavedDrive', 'WoodDeckSF', 'OpenPorchSF',
               'EnclosedPorch', '3SsnPorch', 'ScreenPorch', 'PoolArea', 'PoolQC',
               'Fence', 'MiscFeature', 'MiscVal', 'MoSold', 'YrSold', 'SaleType',
               'SaleCondition', 'SalePrice'],
              dtype='object')
```

There is hell of a lot of features present and we'll deal with them. There is information about the various Area's, Land and Location, Types of finishing, Basement, Garage, Year, Pool, additional facilities etc... Most people would consider the location of the house, No. of bedrooms, Year the house is built as the primary factors when looking for a new house. But as it turns out, this dataset proves that factors like basement area, garage size etc... influence the price more than the former ones.

The Target variable SalePrice:

The most important variable here is the target variable *SalePrice. Let's check its distribution.*

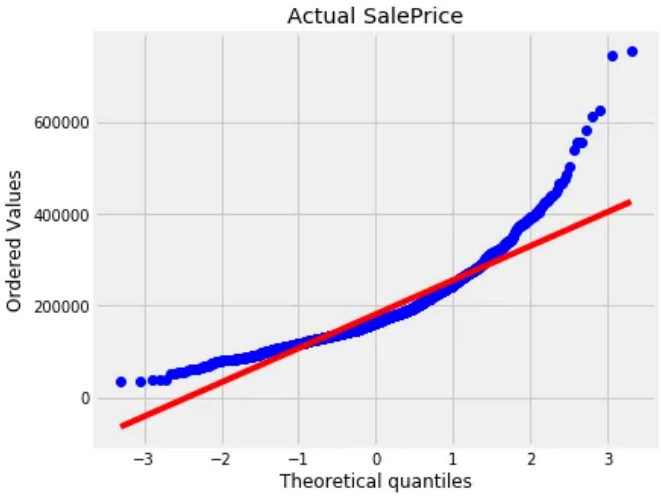
```
sns.distplot(train.SalePrice)
plt.show()
```

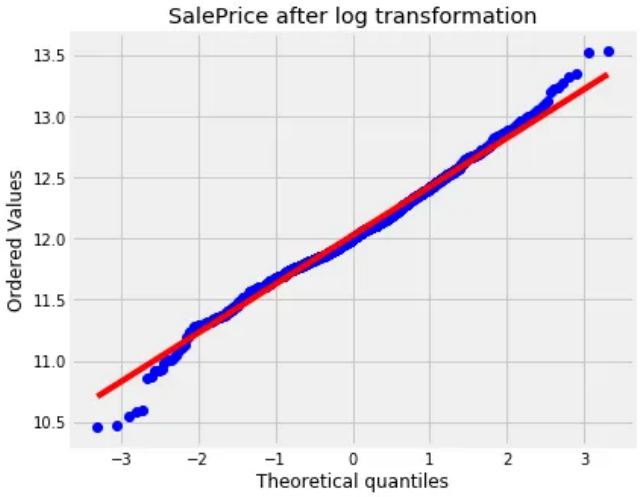


This shows that *SalePrice* does not follow normal distribution. It has positive skewness. It shows that very small number of houses have very high *SalePrice.* Also the SalePrice is not linear, which means we cannot find of a straight line that would fit through the *SalePrice* data. So we have to transform *SalePrice so* that it'll have normal distribution.

The most important thing any statistics class will teach you is that log transformation works fine with positive skewness. So lets take log for *SalePrice*.

```
figure = plt.figure(figsize = (13,5))
plt.subplot(1,2,1)
stats.probplot(train.SalePrice, plot = plt)
plt.title('Actual SalePrice')
plt.subplot(1,2,2)
train.SalePrice = np.log(train.SalePrice)
stats.probplot(train.SalePrice, plot = plt)
plt.title('SalePrice after log transformation')
plt.show()
plt.savefig('SalePrice.jpg')
```





Looking for missing values

I like to first look for missing values and deal with them. The training set has a total of 1460 samples.

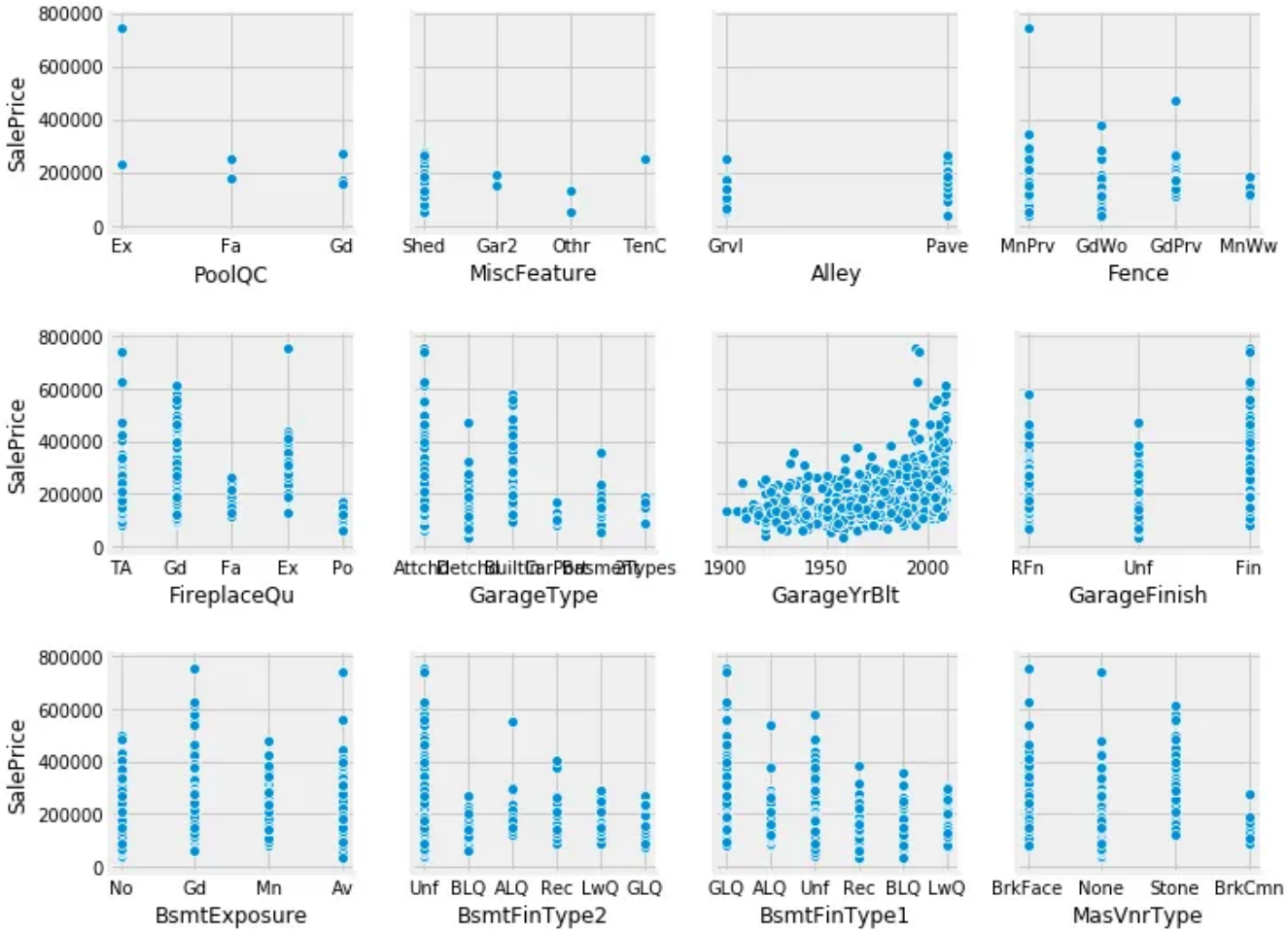
```
missing = train.isna().sum()
missing = missing[missing>0]
missing_perc = missing/train.shape[0]*100
na = pd.DataFrame([missing, missing_perc], index = [‘missing_num’,
‘missing_perc’]).T
na = na.sort_values(by = ‘missing_perc’, ascending = False)
```

	missing_num	missing_perc
PoolQC	1453.0	99.520548
MiscFeature	1406.0	96.301370
Alley	1369.0	93.767123
Fence	1179.0	80.753425
FireplaceQu	690.0	47.260274
LotFrontage	259.0	17.739726
GarageType	81.0	5.547945
GarageYrBlt	81.0	5.547945
GarageFinish	81.0	5.547945
GarageQual	81.0	5.547945
GarageCond	81.0	5.547945
BsmtExposure	38.0	2.602740
BsmtFinType2	38.0	2.602740
BsmtFinType1	37.0	2.534247
BsmtCond	37.0	2.534247
BsmtQual	37.0	2.534247
MasVnrArea	8.0	0.547945
MasVnrType	8.0	0.547945
Electrical	1.0	0.068493

So we have missing values in 19 columns. We can choose to fill them with appropriate values or we can drop particular rows (samples) or the the entire columns with missing values.

There are even features with more than 90% missing values. That is because most houses do not have a Pool, Misc feature, no Alley associated with them etc... I am going to completely drop all those features except *GarageQual, GarageCond, BsmtCond, BsmtQual, LotFrontage, MasVnrArea and Electrical.* I can drop of these variables, as they don't seem to have an influence on the sale price. Also most of them are collinear with other available features. I trained the same model by including these features and again after dropping them. I didn't any considerable difference in the result. And i am going to drop just the one sample with missing

Electrical value.



For the remaining 6 features as i looked at the data, i can see that *GarageQual, GarageCond, BsmtCond, BsmtQual* are missing in houses that do not have a garage or a basement. So i am going to fill them with *'NA'* which is the default value if the feature is not present. And missing values in *LotFrontage and MasVnrArea* can be replaced by zero.

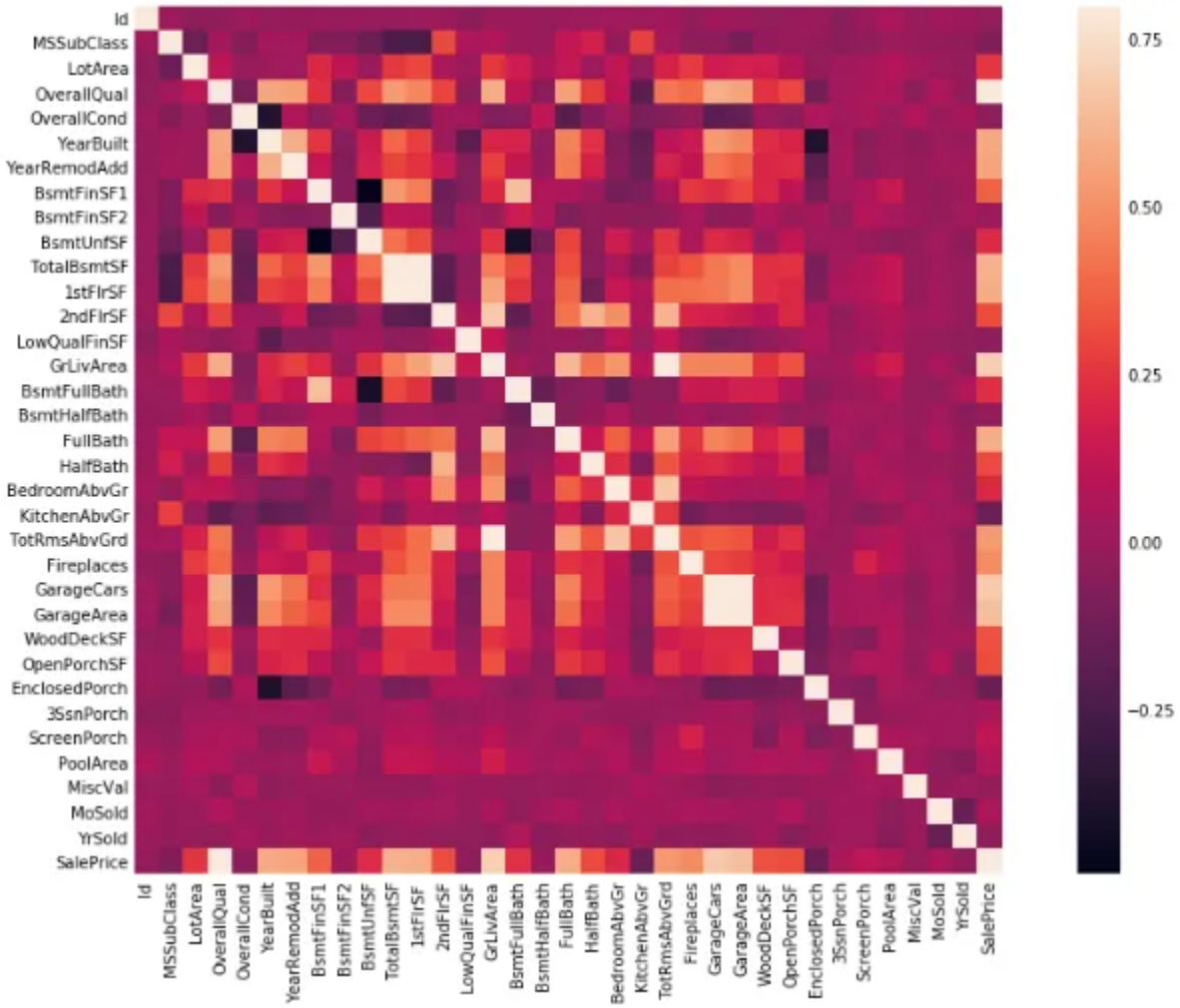
```
train.drop(['PoolQC', 'MiscFeature', 'Alley', 'Fence',
'FireplaceQu',
 'LotFrontage', 'GarageType', 'GarageYrBlt',
 'GarageFinish','BsmtExposure', 'BsmtFinType2', 'BsmtFinType1',
 'MasVnrArea', 'MasVnrType'], axis = 1, inplace = True)

train.drop(train[train.Electrical.isna()].index, axis = 0, inplace =
True)
NA = [ 'GarageQual', 'GarageCond', 'BsmtCond', 'BsmtQual']
for na in NA:
```

```
 train[na].fillna('NA', inplace = True)
train['MasVnrArea'].fillna(0, inplace = True)
train['LotFrontage'].fillna(0, inplace = True)
```

Understanding the Correlations

To know what are features that have the most influence on the *SalePrice*, lets plot the correlation. It is also good to check whether independent features are correlated among themselves, which will result in Multicollinearity.

```
plt.figure(figsize = (15,10))
sns.heatmap(train.corr(), vmax=.8, square=True)
plt.savefig('C:\\Users\\MyPC\\Pictures\\Heatmap.jpg')
```



So the heatmap shows that **Overall quality** and **Ground Living Area** have the highest

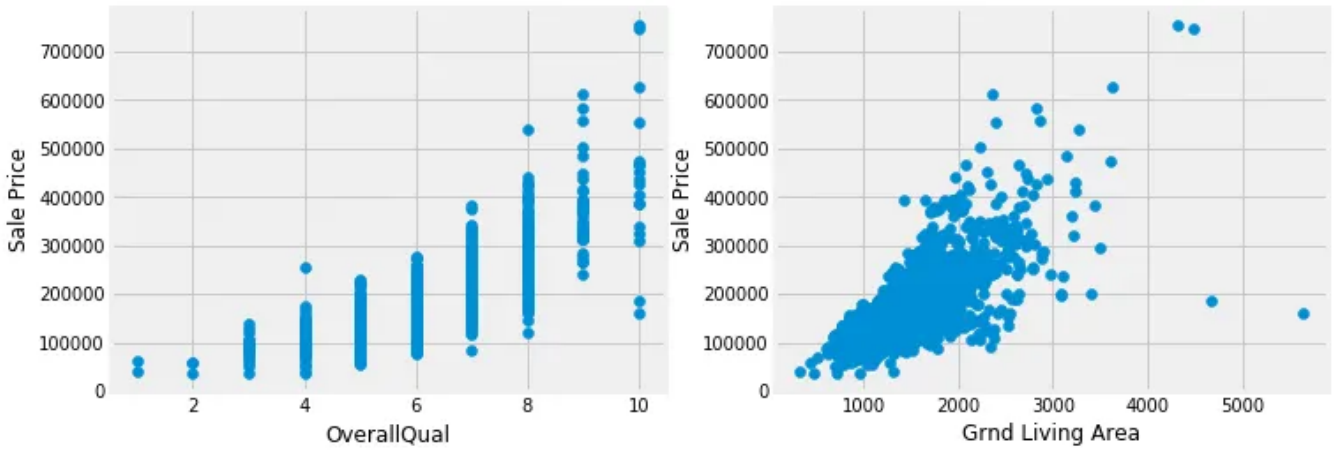
correlation with the *SalePrice*. It is obvious because larger houses will definitely be costlier than smaller ones and though it is not clear on what basis *OverallQual* is calculated, *SalePrice* will be proportional to quality.

And other features with considerable correlations with *SalePrice* are *TotalBsmtSF, 1stFlrSurface, FullBath, GarageCars and GarageArea,* which also makes sense.

Also we can see some high correlation within the independent variables. There is high correlation between *GrLivArea* and *TotRmsAbvGrd* and similar relation exists between *GarageCars* and *GarageArea.* This will lead to multicollinearity and the our linear model may struggle to identify the relationship between these features and *SalePrice.* So it is recommended to drop one of those features.

To confirm all these correlations, lets plot the scatter plot for these features.

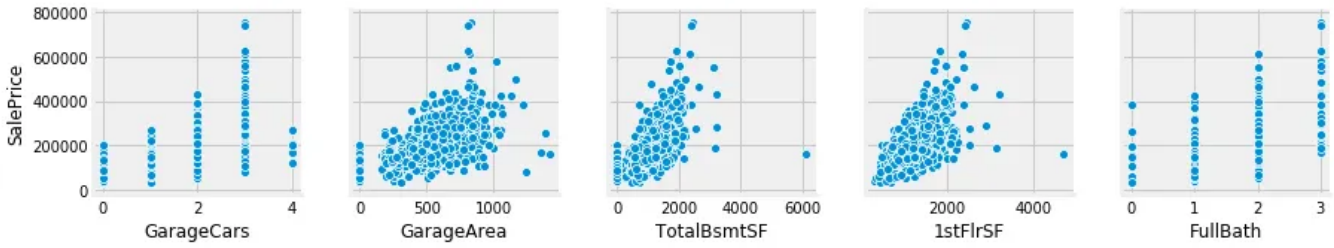
```
plt.figure(figsize = (11,4))
plt.subplot(1,2,1)
plt.scatter(train.OverallQual, train.SalePrice)
plt.xlabel('OverallQual'); plt.ylabel('Sale Price')
plt.subplot(1,2,2)
plt.scatter(train.GrLivArea, train.SalePrice)
plt.xlabel('Grnd Living Area'); plt.ylabel('Sale Price')
plt.show()
```



The scatter plot confirms the high correlation between *GrLivArea, OverallQual* and *SalePrice.* Though we can see some outliers in the *GrndLivArea*, we'll deal with it later. Ignore them for now.

Similarly, plot for *TotalBsmtSF, 1stFlrSurface, FullBath, GarageCars and GarageArea.*

```
var = ['GarageCars', 'GarageArea', 'TotalBsmtSF', '1stFlrSF',
'FullBath']
sns.pairplot(x_vars = var, y_vars = 'SalePrice', data = train)
plt.show()
```



All these features show positive correlation with *SalePrice*.

Feature Engineering

Having understood the correlations in the features, let's do some feature engineering to make our data better for modelling.

So first let's solve the multicollinearity issue. So i am going to drop the

TotRmsAbvGrd, and *GarageArea* features as they have less correlation with the *SalePrice* compared to *GarageCars* and GrLivArea. Also drop the *Id* column.

```
train.drop('Id', axis = 1, inplace = True)
train.drop(['TotRmsAbvGrd', 'GarageArea'], axis =1, inplace = True)
```

Now we can think about adding new features to the dataset. As *OverallQual, GrLivArea* have high correlation with *SalePrice,* creating features that are just the higher powers of *OverallQual and GrLivArea will help the model.*

```
features = ['OverallQual', 'GrLivArea']
for feat in features:
 train[feat+'_p2'] = train[feat] **2
 train[feat+'_p3'] = train[feat] **3
```

Dealing with categorical features

Before doing further feature engineering, have to do look at our dataset again. We have both categorical and numerical features. But most ML models are only happy with numerical values. So we have to convert the categories into numbers. Also we have to be aware of he ordinal values (Categories in which order matters). So first let us convert the ordinal features to numerical based on their weight. For example Garage quality has the categories Ex, Gd, TA, Fa, Po, NA. So we have to convert this to numbers. But we have to be be aware that *'Ex'* refers to a higher quality and *'Po'* refers to a lower quality. So encoding should be based on this.

```
mp = {'Ex': 5, 'Gd':4, 'TA':3, 'Fa':2, 'Po':1, 'NA':0}
for feat in ['ExterQual', 'ExterCond', 'BsmtQual', 'BsmtCond',
 'HeatingQC', 'KitchenQual','GarageQual', 'GarageCond', ]:
 train[feat] = train[feat].map(mp)
mp = {'N':0, 'Y':2 , 'P':1}
for feat in ['CentralAir', 'PavedDrive']:
 train[feat] = train[feat].map(mp)
mp = {'Typ':8, 'Min1':7, 'Min2':6, 'Mod':5, 'Maj1':4, 'Maj2':3,
'Sev':2, 'Sal':1}
train['Functional'] = train['Functional'].map(mp)
```

```
mp = {'Gtl':1 ,'Mod':2 , 'Sev':3}
train['LandSlope'] = train['LandSlope'].map(mp)
```

Now what new features can we add??? We have FullBath and HalfBath. Halfbath, though do not influence the SalePrice much. But what if we create a feature called TotalBathrooms? It will help. So add features like this.

```
train['TotBath'] =
train['BsmtFullBath']+train['FullBath']+.5*(train.BsmtHalfBath+train
.HalfBath)
train['Overall_Score'] = train.OverallQual*train.OverallCond
train['Total_area'] =
train['1stFlrSF']+train['2ndFlrSF']+train.TotalBsmtSF
train['Garage_Score'] = train.GarageQual*train.GarageCond
train['Kitchen_Score'] = train.KitchenAbvGr*train.KitchenQual
train['Bsmt_Score'] = train.BsmtQual*train.BsmtCond
```

We also have the other categorical features (nominal features) that should be converted to numerical. For example there is the Feature called Street, which includes two categories, Grvl and Pave, referring to Gravel and Paved street types. So how to convert this to numbers. Assigning 1 and 0 to them will mean that the category with 1 will be more important than the category with 0. But actually both of them are just different categories and there is nothing like, one superior to other. So the solution for this is One Hot Encoding. This can be easily done than explained. So what were going to do is create two new features called Street_Pave and Street_Grvl and assign 1 and 0 in the columns depending on what type of street the house have. So assign 1 in Street_Grvl and 0 in Street_Pave if the house has Gravel Street. And assign 1 in Street_Pave and 0 in Street_Grvl if the house has Paved Street. Doing this in python using pandas is as simple as calling the get_dummies function from pandas on the train dataset.

```
train = pd.get_dummies(train)
```

So our data is almost ready for modelling. Just one last step. The values in different

features are at different scales. For example, the *GrLiVArea* will have values in range of thousands and MasVnrArea will have values in range of hundreds. So we need to normalize the features before applying Machine Learning techniques. I am using StandardScalar function from scikit learn for this. What it does is, it will transform your data such that its distribution will have a mean value 0 and standard deviation of 1.

And i need to remove the outlier that i came across above. Actually outliers should have been removed much earilier. But as there are only two ouliers, it is not going to matter. I can do it now.

```
train = train.loc[~(train.GrLivArea >4000)]
```

And I am going to divide the training data to train_x and train_y, to apply it to the ML model.

```
train_x = train.drop('SalePrice', axis = 1)
train_y = pd.DataFrame(train.SalePrice)
index = train_x.columns
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
train_x = scaler.fit_transform(train_x)
train_x = pd.DataFrame(train_x, columns = index)
```

And our train data is ready now. I am going to do the same for the test dataset.

```
test.drop(['PoolQC', 'MiscFeature', 'Alley', 'Fence', 'FireplaceQu',
'LotFrontage', 'GarageType', 'GarageYrBlt',
'GarageFinish','BsmtExposure', 'BsmtFinType2',
'BsmtFinType1','MasVnrArea', 'MasVnrType'], axis = True, inplace =
True)
test.drop(test[test.Electrical.isna()].index, axis = 0, inplace =
```

```
True)
NA = [ 'GarageQual', 'GarageCond', 'BsmtCond', 'BsmtQual']
for na in NA:
    test[na].fillna('NA', inplace = True)
    NA = ['GarageQual', 'GarageCond','BsmtCond', 'BsmtQual']
for na in NA:
    test[na].fillna('NA', inplace = True)
fill_zero = ['GarageCars', 'GarageArea']
for zeros in fill_zero:
    test[zeros].fillna(0, inplace = True)
test['Electrical'].fillna('SBrkr', inplace = True)
test.MSZoning.fillna('RL', inplace = True)
test.Utilities.fillna('AllPub', inplace = True)
test.BsmtFinSF1.fillna(0.0, inplace = True)
test.BsmtFinSF2.fillna(0.0, inplace = True)
test.BsmtUnfSF.fillna(0.0, inplace = True)
test.TotalBsmtSF.fillna(0.0, inplace = True)
test.BsmtFullBath.fillna(0.0, inplace = True)
test.BsmtHalfBath.fillna(0.0, inplace = True)
test.Functional.fillna('Typ', inplace = True)
test.Exterior1st.fillna('VinylSd', inplace = True)
test.Exterior2nd.fillna('VinylSd', inplace = True)
test.KitchenQual.fillna('TA', inplace = True)
test.SaleType.fillna('WD', inplace = True)
test.drop('Id', axis = 1, inplace = True)
features = ['OverallQual', 'GrLivArea']
for feat in features:
    test[feat+'_p2'] = test[feat] **2
    test[feat+'_p3'] = test[feat] **3
test.drop(['TotRmsAbvGrd', 'GarageArea'], axis =1, inplace = True)
mp = {'Ex': 5, 'Gd':4, 'TA':3, 'Fa':2, 'Po':1, 'NA':0}
for feat in ['ExterQual', 'ExterCond', 'BsmtQual', 'BsmtCond',
'HeatingQC', 'KitchenQual', 'GarageQual', 'GarageCond']:
    test[feat] = test[feat].map(mp)
mp = {'N':0, 'Y':2 , 'P':1}
for feat in ['CentralAir', 'PavedDrive']:
    test[feat] = test[feat].map(mp)
mp = {'Typ':8, 'Min1':7, 'Min2':6, 'Mod':5, 'Maj1':4, 'Maj2':3,
'Sev':2, 'Sal':1}
test['Functional'] = test['Functional'].map(mp)
mp = {'Gtl':1 ,'Mod':2 , 'Sev':3}
test['LandSlope'] = test['LandSlope'].map(mp)
test['Bathrooms'] = test['BsmtFullBath'] + test['FullBath'] +
0.5*(test.BsmtHalfBath + test.HalfBath)
test['Overall_Score'] = test.OverallQual*test.OverallCond
test['Total_area'] =
test['1stFlrSF']+test['2ndFlrSF']+test.TotalBsmtSF
test['Garage_Score'] = test.GarageQual*test.GarageCond
test['Kitchen_Score'] = test.KitchenAbvGr*test.KitchenQual
test['Bsmt_Score'] = test.BsmtQual*test.BsmtCond
```

```
test = pd.get_dummies(test)
testcol = test.columns.tolist()
traincol = train_x.columns.tolist()
diff = set(traincol).difference(testcol)
diff = list(diff)
last_cols = train_x[diff]
train_x.drop(diff, axis =1, inplace = True)
```

Model Creation and Evaluation

As the target *SalePrice* is a continues value we can use regression here. As i said earlier, i am not going to focus much on modelling, so i will be just using linear regression. But it is always good to try on multple techniques to get the best possible result.

I used Ridge model from Scikit learn, which is a Regularised linear regression technique. I saw significant difference in the prediction accuracy, when using normal Linear Regression, and Regularised Linear Regression. Normal Linear Regression is prone to overfitting, while in Ridge, we could control variance with the parameter alpha which imposes a penalty on the weights applied for each feature, thus preventing overfitting.

```
from sklearn.linear_model import Ridge
from sklearn.metrics import mean_squared_error as mse
model1 = Ridge(alpha = 1)
model1.fit(trainx, trainy)
print("THe accuracy of training set si is ", model1.score(train_x,
train_y))
predictions = model1.predict(test)
```

Using Regularised Linear Regreesion in this dataset gave me the score of 0.12821, which is not the best of what you can get. There are lot of ways in which you can improve accuracy by playing more with the data. We can add more features and still there are lots of features that are left unused. I am still learning to figure out the best ways to improve things. And i will try to update things as i find more.

Here is the link to my source code on GitHub -

https://github.com/KamskiJohn/Ames-Housing-Price-Prediction-

I will be delighted to receive suggestions and let me know whether you found this useful.

Happy learning...

Machine Learning | Exploratory Data Analysis | Feature Engineering | Pandas | Data Analysis

Follow

Written by Kamski John

28 followers · 24 following

I call myself an Engineer. Not because i am pursuing BE, but because i THinK.

Responses (4)

Write a response

What are your thoughts?

Will
Nov 13, 2020

Is it necessary to transform the dependent variable(SalesPrice)?

Reply

sampath kumar gajawada
Oct 22, 2019

Nice article !!

But i could see there are many continuous features in this dataset like name ending with area ex; pool area.

And these features have lot of values and they are filled with 0. Usually, there will be no zero area so we have to consider... more

Reply

O

Oluwatosinijaodola
Sep 2, 2019

Hello,

Thanks for sharing. I think it will be better if you can use other algorithms and validate your train data.

Reply

See all responses

Recommended from Medium

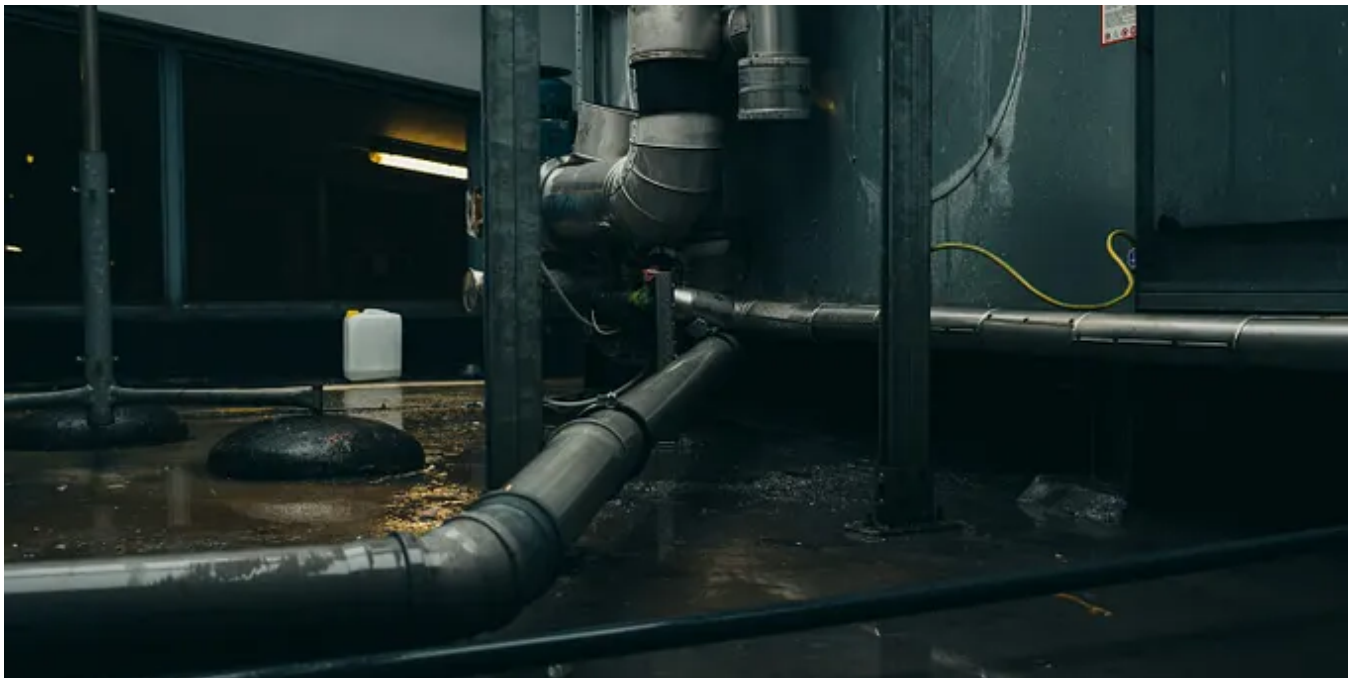
	Purpose
)	Load CSV dataset into DataFrame
plit()	Split data into training & testing set
er()	Normalize/standardize numeric feat
r()	Convert categorical variables into n

Maha K

Top Scikit-learn Functions Every Data Scientist Must Know

Learn essential functions for ML algorithms, model building & evaluation with Python examples, real datasets, and easy-to-understand...

Aug 10 18

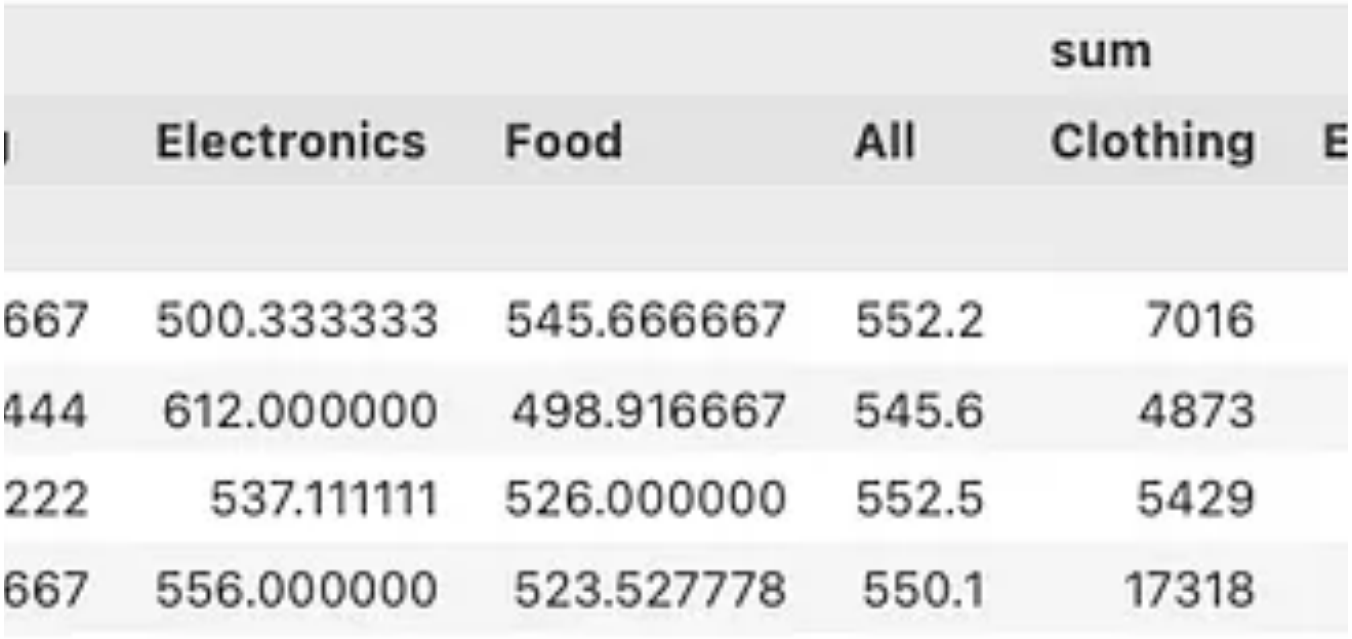


In Python in Plain English by Zain Ahmad

How I Automated My Dataset Cleaning Pipeline Using Python, Pandas, and Great Expectations

Using pandas, great_expectations, and autoviz, I built a Python script that automatically detects missing values, wrong types, outliers —...

Jun 24



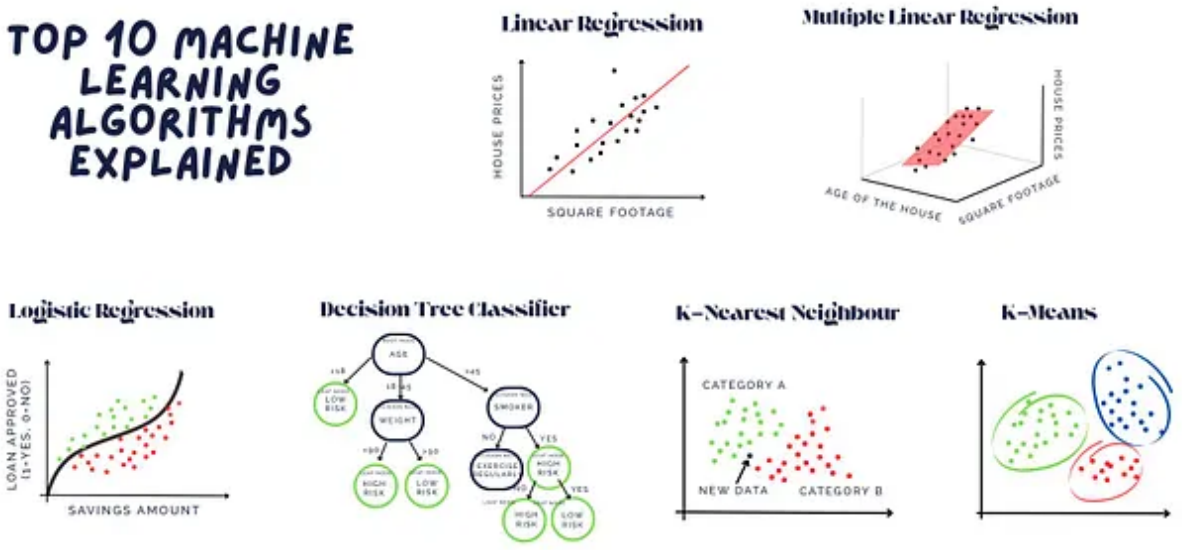
			sum	
	Electronics	Food	All	Clothing	E
667	500.333333	545.666667	552.2	7016	
444	612.000000	498.916667	545.6	4873	
222	537.111111	526.000000	552.5	5429	
667	556.000000	523.527778	550.1	17318	

Prathik C

Data Summarization in Pandas: Your Guide to pivot_table, groupby, crosstab, and pivot

Life with Python #23

Sep 12



In Learning Data by Rita Angelou

10 ML Algorithms Every Data Scientist Should Know—Part 1

I understand well that machine learning might sound intimidating. But once you break down the common algorithms, you'll see they're not.

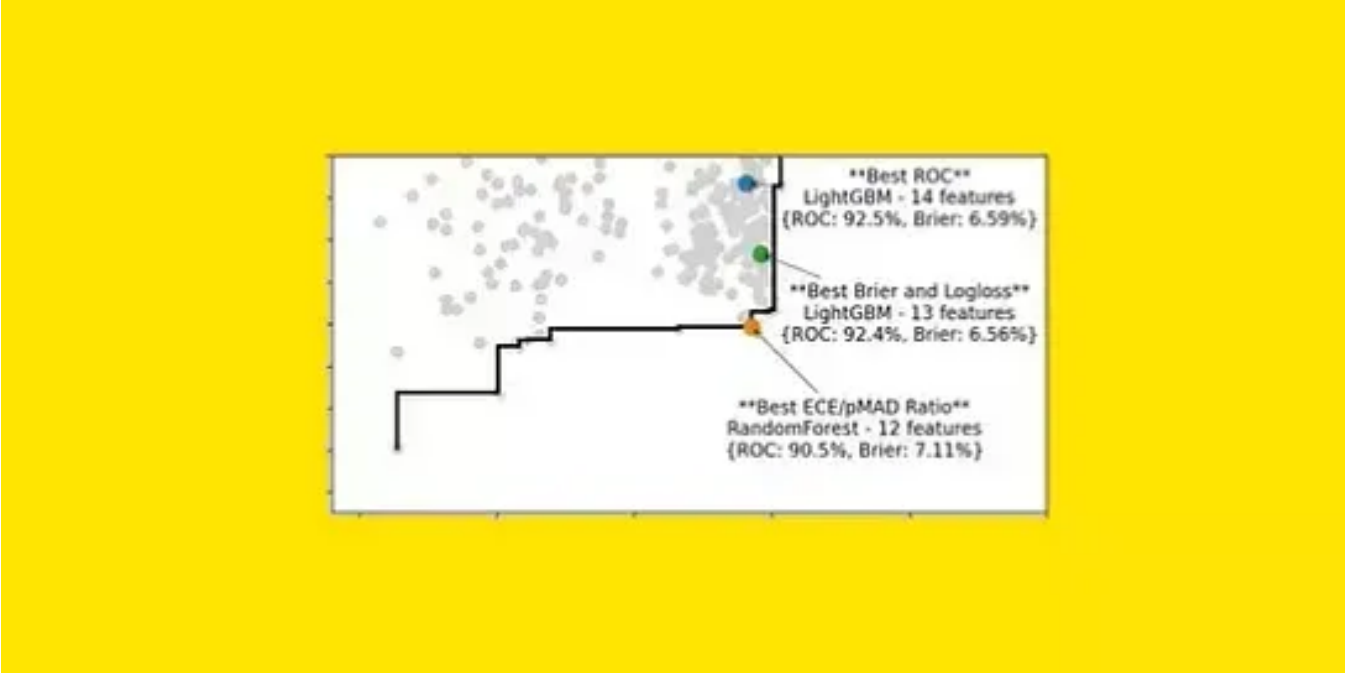
Jun 10 104 2

Monica Ashok

Day #6: Exploratory Data Analysis—Wisconsin Breast Cancer Dataset

Today is all about uncovering patterns, distributions, and relationships hidden inside the Wisconsin Breast Cancer data. With targeted EDA...

Aug 20



In Data Science Collective by Samuele Mazzanti

It Took Me 6 Years to Find the Best Metric for Classification Models

How I realized that the best calibration metric is none of the ones you'd expect (ROC, Log-loss, Brier score, etc.)

5d ago 333 7

See more recommendations