Housing Price Prediction

Introduction

1. Data Loading and data Description
2. Data Understanding and Exploration
3. Data cleaning
4. Feature Engineering & Improving data quality
5. Splitting data into training and evaluation sets
6. Feature Scaling
7. Model building and
8. Evaluation

GOALS

There are two primary goals of this assignment.

1. Statistical and exploratory data analysis of housing prices.
2. And to create machine learning models that can predict the housing prices.

Importing Packages

1. **Numpy -** Implementing multi-dimensional array and matrices.
2. **Pandas -** For data manipulation and analysis.
3. **Matplotlib -** Plotting library for Python programming language and it's numerical mathematics extension NumPy.
4. **Seaborn -** Provides a high level interface for drawing attractive and informative statistical graphics.
5. **Scikit-learn -** Scikit-learn is a free software machine learning library for the Python programming language. It features various classification, regression and clustering algorithms including support vector machines, random forests, gradient boosting, k-means and DBSCAN, and is designed to interoperate with the Python numerical and scientific libraries NumPy and SciPy.

Initial observations

1. This dataset has **1460** rows and **81** columns.
2. Summary of data types in this dataset:
   - **Numeric:** 3 (Float), 35 (Integer)
   - **Object:** 43
3. The following variables have null and zero values that may need to addressed.
   - **PoolQC** has **1453** missing values.
   - **MiscFeature** has **1406** missing values.
   - **Alley** has **1369** missing values.
   - **Fence** has **1179** missing values.
   - **FireplaceQu** has **690** missing values.
   - **LotFrontage** has **259** missing values.
   - **GarageCond** has **81** missing values.

Initial Observations

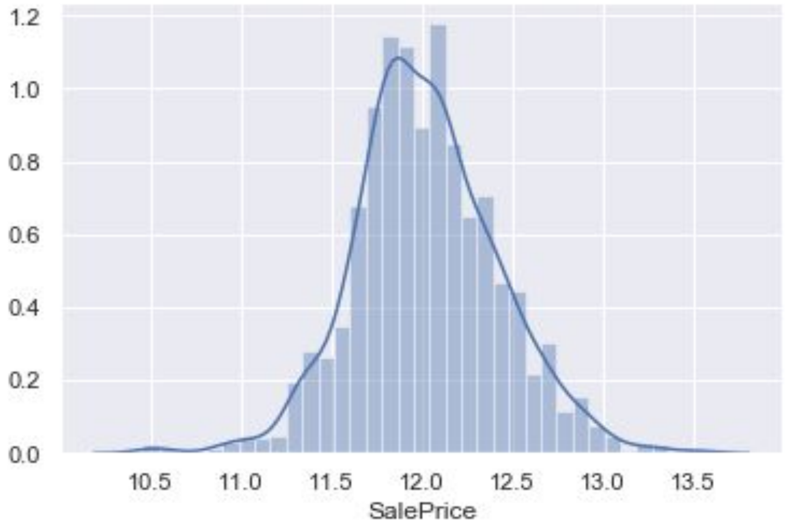
- **GarageType** has **81** missing values.
- **GarageYrBlt** has **81** missing values.
- **GarageFinish** has **81** missing values.
- **GarageQual** has **81** missing values.
- **BsmtExposure** has **38** missing values.
- **BsmtFinType2** has **38** missing values.
- **BsmtFinType1** has **37** missing values.
- **BsmtCond** has **37** missing values.
- **BsmtQual** has **37** missing values.
- **MasVnrArea** has **8** missing values.
- **MasVnrType** has **8** missing values.
- **Electrical** has **1** missing values.

Final observations

It appears that the target, **SalePrice**, is very skewed and a transformation like a logarithm would make it more normally distributed. Machine Learning models tend to work much better with normally distributed targets, rather than greatly skewed targets. By transforming the prices, we can boost model performance.

1. Skewness: 1.882876
2. Kurtosis: 6.536282

sns.distplot(np.log(df["SalePrice"]))



Data Cleaning

In the context of data science and machine learning, data cleaning means filtering and modifying your data such that it is easier to explore, understand, and model. Filtering out the parts you don't want or need so that you don't need to look at or process them.

1. Removed **PoolQC, MiscFeature, Alley, Fence** and **FireplaceQu** columns as these are having very high missing data.
2. Filling missing data with mean / mode of their respective column
   - Filled **LotFrontage, GarageYrBlt** and **MasVnrArea** columns with mean value.
   - Filled **GarageType, GarageFinish, GarageQual, GarageCond, BsmtFinType2, BsmtExposure, BsmtFinType1, BsmtQual, MasVnrType** and **Electrical** columns with mode value.
3. Finding and removing outliers with following ranges
   - Column **MSSubClass** has upper bound - **145.0** and lower bound - **20**

Data Cleaning

- Column **LotFrontage** has upper bound - **107.5** and lower bound - **21.0**
- Column **LotArea** has upper bound - **17673.5** and lower bound **- 1300**
- Column **MasVnrArea** has upper bound - **410.625** and lower bound - **0.0**
- Column **BsmtFinSF1** has upper bound - **1780.625** and lower bound - **0**
- Column **BsmtUnfSF** has upper bound - **1685.5** and lower bound - **0**
- Column **TotalBsmtSF** has upper bound - **2052.0** and lower bound - **0**
- Column **1stFlrSF** has upper bound - **2155.125** and lower bound - **334**
- Column **2ndFlrSF** has upper bound - **1820.0** and lower bound **- 0**
- Column **LowQualFinSF** has upper bound - **0.0** and lower bound - **0**
- Column **GrLivArea** has upper bound - **2747.625** and lower bound - **334**

Data Cleaning

- Column **GarageArea** has upper bound - **938.25** and lower bound - **0**
- Column **WoodDeckSF** has upper bound - **420.0** and lower bound - **0**
- Column **OpenPorchSF** has upper bound - **170.0** and lower bound - **0**
- Column **SalePrice** has upper bound - **13.021682213395525** and lower bound - **10.460242108190519**

Feature Engineering

Feature engineering is the process of transforming raw **data** into **features** that better represent the underlying problem to the predictive models, resulting in improved model accuracy on unseen **data**.

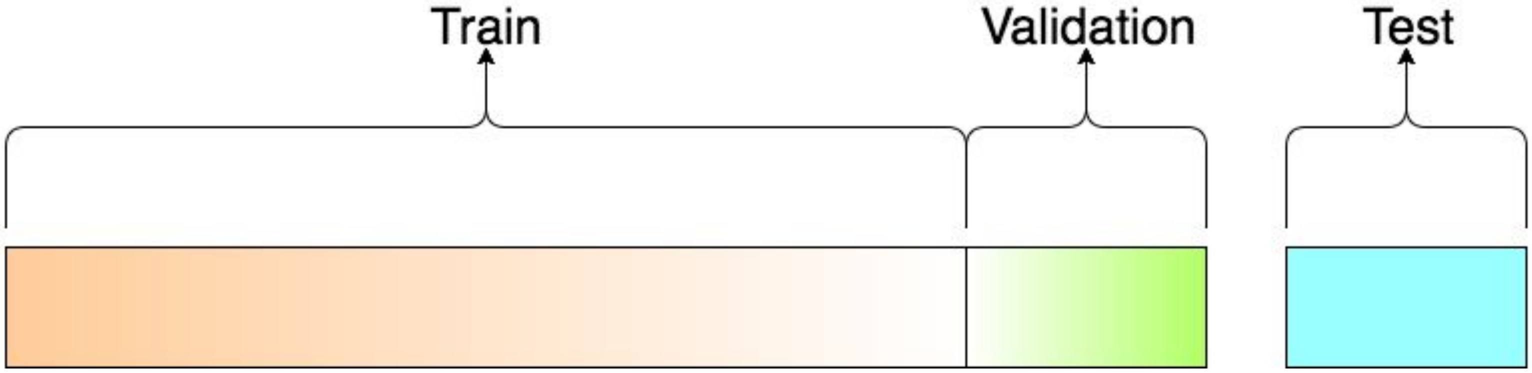
Feature engineering turn your inputs into things the algorithm can understand.

1. Removed high correlated columns **1stFlrSF, TotRmsAbvGrd** and **GarageArea**.
2. Removed irrelevant **Id** column.
3. Encoded columns **MSZoning, LotShape** and **LandContour** with one hot encoding.
4. Encoded columns **Street, Utilities, LotConfig, LandSlope, BldgType, RoofStyle, BsmtFinType1, BsmtFinType2, Heating, Electrical, Functional, GarageFinish, PavedDrive, SaleCondition, ExterQual, ExterCond, BsmtQual, BsmtCond, HeatingQC, KitchenQual, GarageQual, GarageCond, Neighborhood, Condition1, Condition2, HouseStyle, RoofMatl, Exterior1st, Exterior2nd, MasVnrType, Foundation, GarageType** and **SaleType** with Label encoding.

SPLITTING DATA

The data we use is usually split into training data and test data.

Training Dataset: The sample of data used to fit the model.
Validation Dataset: The sample of data used to provide an unbiased evaluation of a model fit on the training dataset while tuning model hyperparameters. The evaluation becomes more biased as skill on the validation dataset is incorporated into the model configuration.
Test Dataset: The sample of data used to provide an unbiased evaluation of a final model fit on the training dataset.



SPLITTING DATA

```
# separating our independent and dependent variable
X = dataframe.drop(['SalePrice'], axis=1)
y = dataframe["SalePrice"]

from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(X, y, random_state=1,
test_size=.20)
```

Feature Scaling

Feature Scaling is a technique to standardize the independent features present in the data in a fixed range. It is performed during the data pre-processing to handle highly varying magnitudes or values or units. If feature scaling is not done, then a machine learning algorithm tends to weigh greater values, higher and consider smaller values as the lower values, regardless of the unit of the values.

Consider the two most important ones:

- **Min-Max Normalization**
- **Standardization**

```
# Feature Scaling
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X_train = sc.fit_transform(x_train)
X_test = sc.transform(x_test)
```

Building the Model

Linear Regression is a machine learning algorithm based on **supervised learning**. It performs a **regression task**. Regression models a target prediction value based on independent variables. It is mostly used for finding out the relationship between variables and forecasting. Different regression models differ based on – the kind of relationship between dependent and independent variables, they are considering and the number of independent variables being used.

$$y = \theta_1 + \theta_2 . x$$

While training the model we are given : **x**: input training data (univariate – one input variable(parameter))
y: labels to data (supervised learning)

When training the model – it fits the best line to predict the value of y for a given value of x. The model gets the best regression fit line by finding the best **θ1** and **θ2** values. **θ1**: intercept **θ2**: coefficient of **x**.

Building the Model

Once we find the best **θ1** and **θ2** values, we get the best fit line. So when we are finally using our model for prediction, it will predict the value of y for the input value of **x**.

$$minimize \frac{1}{n} \sum_{i=1}^{n} (pred_i - y_i)^2 \qquad J = \frac{1}{n} \sum_{i=1}^{n} (pred_i - y_i)^2$$

 # Linear Regression

from sklearn.linear_model import LinearRegression

linreg = LinearRegression()

linreg.fit(X_train,y_train)

Prediction

y_pred_train = linreg.predict(X_train)

pred = pd.DataFrame(y_pred_train)

y_pred_test = linreg.predict(X_test)

pred_test = pd.DataFrame(y_pred_test)

Model Evaluation Metrics

Model evaluation aims to estimate the generalization accuracy of a **model** on future (unseen/out-of-sample) data. Methods for **evaluating** a **model's** performance are divided into 2 categories: namely, holdout and Cross-validation. Both methods use a test set (i.e data not seen by the **model**) to **evaluate model** performance.
Root Mean Square Error (RMSE) is a standard way to measure the error of a model in predicting quantitative data.

Formally it is defined as follows:

$$RMSE = \sqrt{\sum_{i=1}^{n} \frac{(\hat{y}_i - y_i)^2}{n}}$$

RMSE: Root Mean Square Error is the measure of how well a regression line fits the data points. RMSE can also be construed as Standard Deviation in the residuals.

Model Evaluation Metrics

RMSE: Root Mean Square Error is the measure of how well a regression line fits the data points. RMSE can also be construed as Standard Deviation in the residuals.

```
### RMSE Metrics
from sklearn import metrics
# Calculated Train RMSE
calculated_train_rmse = np.sqrt(metrics.mean_absolute_error(y_train, y_pred_train))
print('Calculated Train RMSE is {}'.format(calculated_train_rmse))
# Calculated Test RMSE
calculated_test_rmse = np.sqrt(metrics.mean_absolute_error(y_test, y_pred_test))
print('Calculated Test RMSE is {}'.format(calculated_test_rmse))
```

Calculated Train RMSE is **0.297564606281447**
Calculated Test RMSE is **0.29502104938445906**

Model Evaluation Metrics

R-squared is a statistical measure of how close the data are to the fitted regression line. It is also known as the coefficient of determination, or the coefficient of multiple determination for multiple regression.

The definition of R-squared is fairly straight-forward; it is the percentage of the response variable variation that is explained by a linear model. Or:

R-squared = Explained variation / Total variation

R-squared is always between 0 and 100%:

- 0% indicates that the model explains none of the variability of the response data around its mean.
- 100% indicates that the model explains all the variability of the response data around its mean.

In general, the higher the R-squared, the better the model fits your data.

R2 shows how well terms (data points) fit a curve or line.

Model Evaluation Metrics

```
from sklearn.metrics import r2_score
# Calculated Train R-squared
calculated_train_r_squared = r2_score(y_train, y_pred_train)
print('Calculated Train R-squared is {}'.format(calculated_train_r_squared))
# Calculated Test R-squared
calculated_test_r_squared = r2_score(y_test, y_pred_test)
print('Calculated Test R-squared is {}'.format(calculated_test_r_squared))
```

Calculated Train R-squared is **0.8896102116033099**

Calculated Test R-squared is **0.9093618210786856**

Model Evaluation Metrics

Adjusted r-square is a modified form of **r-square** whose value increases if new predictors tend to improve model's performance and decreases if new predictors does not improve performance as expected.

Adjusted R2 also indicates how well terms fit a curve or line, but adjusts for the number of terms in a model. If you add more and more useless variables to a model, adjusted r-squared will decrease.

If you add more useful variables, adjusted r-squared will increase.

Adjusted R2 will always be less than or equal to R2.

The formula is:

$$R^2_{adj} = 1 - \left[\frac{(1 - R^2)(n - 1)}{n - k - 1} \right]$$

Model Evaluation Metrics

where:

- **N** is the number of points in your data sample.
- **K** is the number of independent regressors, i.e. the number of variables in your model, excluding the constant.

Calculation of Adjusted Train R-Square: 1 - ((1 - calculated_train_r_squared) * (X_train.shape[0] - 1) / (X_train.shape[0] - X_train.shape[1] - 1)) ⇒ **0.8816154079868366**

Calculation of Adjusted Test R-Square: 1 - ((1 - calculated_test_r_squared) * (X_test.shape[0] - 1) / (X_test.shape[0] - X_test.shape[1] - 1)) ⇒ **0.87585576441583**