Data fields

Here's a brief version of what you'll find in the data description file.

- SalePrice - the property's sale price in dollars. This is the target variable that you're trying to predict.
- MSSubClass: The building class
- MSZoning: The general zoning classification
- LotFrontage: Linear feet of street connected to property
- LotArea: Lot size in square feet
- Street: Type of road access
- Alley: Type of alley access
- LotShape: General shape of property
- LandContour: Flatness of the property
- Utilities: Type of utilities available
- LotConfig: Lot configuration
- LandSlope: Slope of property
- Neighborhood: Physical locations within Ames city limits
- Condition1: Proximity to main road or railroad
- Condition2: Proximity to main road or railroad (if a second is present)
- BldgType: Type of dwelling
- HouseStyle: Style of dwelling
- OverallQual: Overall material and finish quality
- OverallCond: Overall condition rating
- YearBuilt: Original construction date
- YearRemodAdd: Remodel date
- RoofStyle: Type of roof
- RoofMatl: Roof material
- Exterior1st: Exterior covering on house
- Exterior2nd: Exterior covering on house (if more than one material)
- MasVnrType: Masonry veneer type
- MasVnrArea: Masonry veneer area in square feet
- ExterQual: Exterior material quality
- ExterCond: Present condition of the material on the exterior
- Foundation: Type of foundation
- BsmtQual: Height of the basement
- BsmtCond: General condition of the basement
- BsmtExposure: Walkout or garden level basement walls
- BsmtFinType1: Quality of basement finished area
- BsmtFinSF1: Type 1 finished square feet
- BsmtFinType2: Quality of second finished area (if present)
- BsmtFinSF2: Type 2 finished square feet
- BsmtUnfSF: Unfinished square feet of basement area
- TotalBsmtSF: Total square feet of basement area
- Heating: Type of heating
- HeatingQC: Heating quality and condition
- CentralAir: Central air conditioning
- Electrical: Electrical system
- 1stFlrSF: First Floor square feet
- 2ndFlrSF: Second floor square feet
- LowQualFinSF: Low quality finished square feet (all floors)
- GrLivArea: Above grade (ground) living area square feet
- BsmtFullBath: Basement full bathrooms
- BsmtHalfBath: Basement half bathrooms
- FullBath: Full bathrooms above grade
- HalfBath: Half baths above grade
- Bedroom: Number of bedrooms above basement level
- Kitchen: Number of kitchens
- KitchenQual: Kitchen quality
- TotRmsAbvGrd: Total rooms above grade (does not include bathrooms)
- Functional: Home functionality rating
- Fireplaces: Number of fireplaces
- FireplaceQu: Fireplace quality
- GarageType: Garage location
- GarageYrBlt: Year garage was built
- GarageFinish: Interior finish of the garage
- GarageCars: Size of garage in car capacity
- GarageArea: Size of garage in square feet
- GarageQual: Garage quality
- GarageCond: Garage condition

- PavedDrive: Paved driveway
- WoodDeckSF: Wood deck area in square feet
- OpenPorchSF: Open porch area in square feet
- EnclosedPorch: Enclosed porch area in square feet
- 3SsnPorch: Three season porch area in square feet
- ScreenPorch: Screen porch area in square feet
- PoolArea: Pool area in square feet
- PoolQC: Pool quality
- Fence: Fence quality
- MiscFeature: Miscellaneous feature not covered in other categories
- MiscVal: $Value of miscellaneous feature
- MoSold: Month Sold
- YrSold: Year Sold SaleType: Type of sale SaleCondition: Condition of sale

Control Flow:

- Data Analysis
- Feature Engineering
- Feature Selection
- ML/DI Model Building, hyper-parameters tuning and Optimization

Data Analysis

- Exploring the dataset

In [1]:

```
# importing the libraries
import numpy as np
import pandas as pd
pd.set_option("display.max_columns", 100)
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
warnings.filterwarnings("ignore")
```

In [2]:

```
# importing the dataset
dataset = pd.read_csv("train.csv")

# function: Dataset characteristics
def data_characteristics(dataset):

    # displaying the shape of the dataset
    print("Shape of the Dataset : {}".format(dataset.shape))
    print("Number of Columns in the Dataset : {}".format(dataset.shape[1]))
    print("Number of Rows in the Dataset : {}".format(dataset.shape[0]))


    # Understanding the Number of Numeric and Categorical features in dataset
    numeric_features = dataset.select_dtypes(include = [np.number])
    categoric_features = dataset.select_dtypes(exclude = [np.number])
    print("Number of Numerical Features : {}".format(numeric_features.shape[1]))
    print("Number of Categorical Features : {}".format(categoric_features.shape[1]))

    # Understanding the dataset
    print("Information of the Dataset : {}".format(dataset.info(verbose = False, memory_usage = "deep")))


data_characteristics(dataset)

# Statistical Summary of the dataset
print("Statistical Summary of the Dataset : ")
dataset.describe(include = "all").transpose()
```

```
Shape of the Dataset : (1460, 81)
Number of Columns in the Dataset : 81
Number of Rows in the Dataset : 1460
Number of Numerical Features : 38
Number of Categorical Features : 43
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1460 entries, 0 to 1459
Columns: 81 entries, Id to SalePrice
dtypes: float64(3), int64(35), object(43)
memory usage: 3.9 MB
Information of the Dataset : None
Statistical Summary of the Dataset :
```

Out[2]:

	count	unique	top	freq	mean	std	min	25%	50%	75%	max
Id	1460	NaN	NaN	NaN	730.5	421.61	1	365.75	730.5	1095.25	1460
MSSubClass	1460	NaN	NaN	NaN	56.8973	42.3006	20	20	50	70	190
MSZoning	1460	5	RL	1151	NaN	NaN	NaN	NaN	NaN	NaN	NaN
LotFrontage	1201	NaN	NaN	NaN	70.05	24.2848	21	59	69	80	313
LotArea	1460	NaN	NaN	NaN	10516.8	9981.26	1300	7553.5	9478.5	11601.5	215245
...	...	...	...	...	...	...	...	...	...	...	...
MoSold	1460	NaN	NaN	NaN	6.32192	2.70363	1	5	6	8	12
YrSold	1460	NaN	NaN	NaN	2007.82	1.3281	2006	2007	2008	2009	2010
SaleType	1460	9	WD	1267	NaN	NaN	NaN	NaN	NaN	NaN	NaN
SaleCondition	1460	6	Normal	1198	NaN	NaN	NaN	NaN	NaN	NaN	NaN
SalePrice	1460	NaN	NaN	NaN	180921	79442.5	34900	129975	163000	214000	755000

81 rows × 11 columns

In [3]:

```
# looking at the dataset
dataset.head()
```

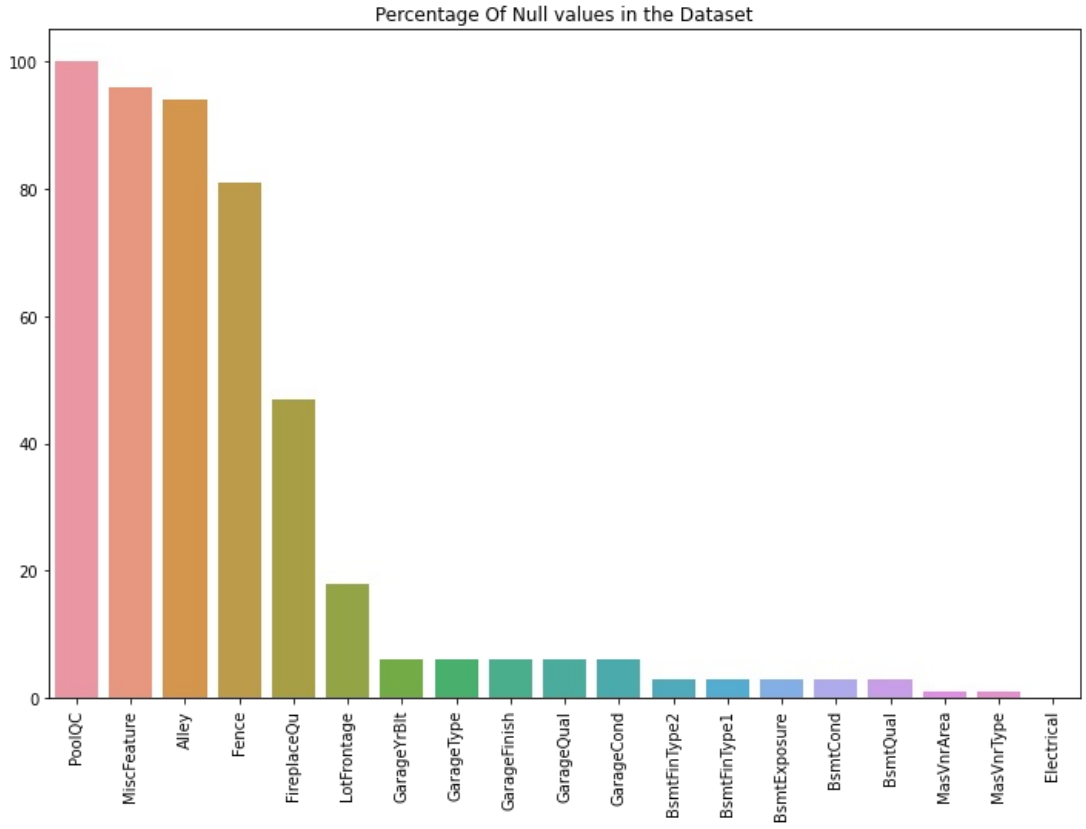
Out[3]:

	Id	MSSubClass	MSZoning	LotFrontage	LotArea	Street	Alley	LotShape	LandContour	Utilities	LotConfig	LandSlope	Neighborh
0	1	60	RL	65.0	8450	Pave	NaN	Reg	Lvl	AllPub	Inside	Gtl	Co
1	2	20	RL	80.0	9600	Pave	NaN	Reg	Lvl	AllPub	FR2	Gtl	Vee
2	3	60	RL	68.0	11250	Pave	NaN	IR1	Lvl	AllPub	Inside	Gtl	Co
3	4	70	RL	60.0	9550	Pave	NaN	IR1	Lvl	AllPub	Corner	Gtl	Cra
4	5	60	RL	84.0	14260	Pave	NaN	IR1	Lvl	AllPub	FR2	Gtl	NoR

In [4]:

```
# function: check percentage of missing values
def null_check(dataset):

    # Calculating the percentage of missing values
    null_per = (dataset.isnull().sum() / len(dataset)) * 100
    try:
        # dropping null_per == 0
        null_per = round(null_per.drop(null_per[null_per == 0].index)).sort_values(ascending = False)


        # Making the bar plot of the Null values
        plt.figure(figsize = (12,8))
        null_plot = sns.barplot(x = null_per.index , y = null_per)
        plt.title("Percentage Of Null values in the Dataset")
        plt.xticks(rotation = "90")
        print(null_plot)

    except:
        print("There is NO null values in the dataset")
        print("Returning the Dataset...")
        return dataset

null_check(dataset)
```

```
AxesSubplot(0.125,0.125;0.775x0.755)
```



In [5]:

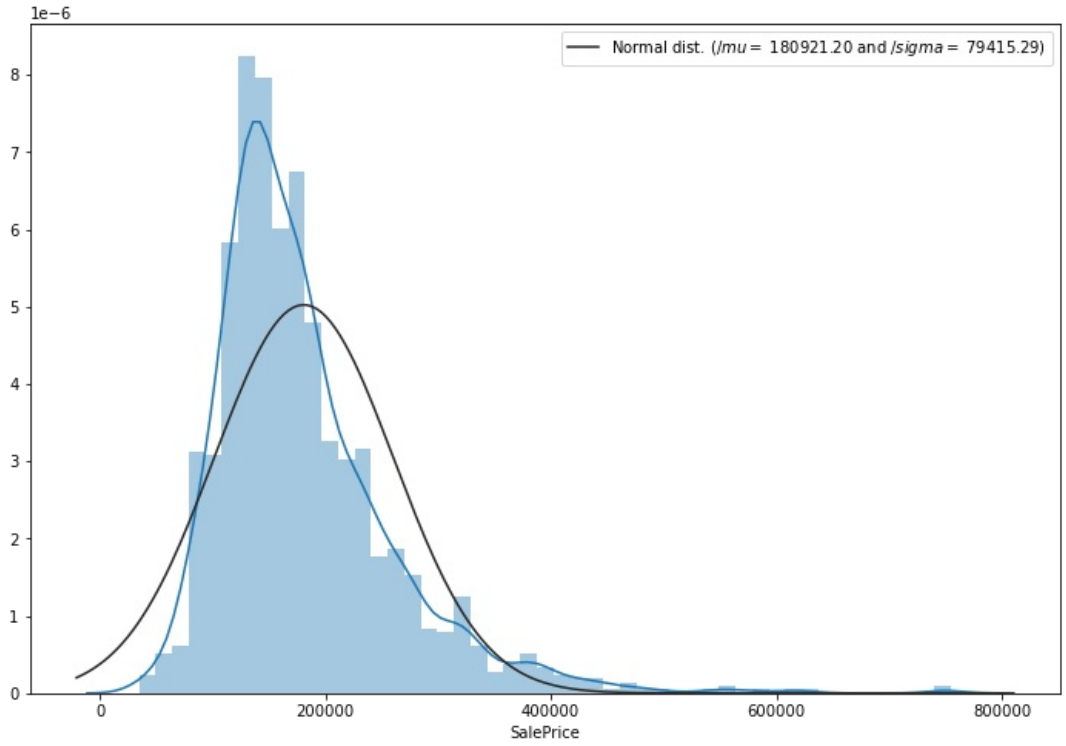
```
# Dropping some High Null Columns
dataset.drop(columns= ["PoolQC", "MiscFeature", "Alley", "Fence", "FireplaceQu", "Id"],
             axis= 1,
             inplace = True)

dataset.head()
```

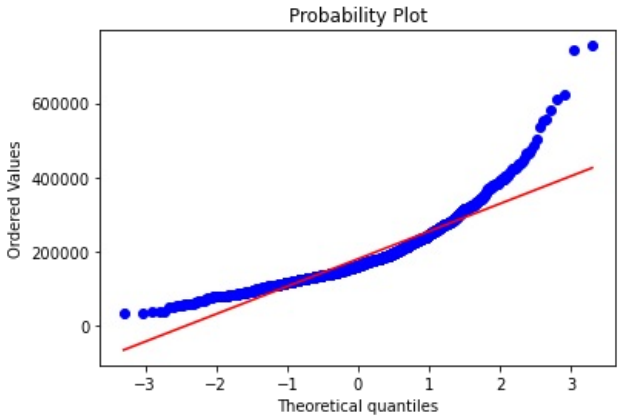
Out[5]:

	MSSubClass	MSZoning	LotFrontage	LotArea	Street	LotShape	LandContour	Utilities	LotConfig	LandSlope	Neighborhood	Cond
0	60	RL	65.0	8450	Pave	Reg	Lvl	AllPub	Inside	Gtl	CollgCr	
1	20	RL	80.0	9600	Pave	Reg	Lvl	AllPub	FR2	Gtl	Veenker	
2	60	RL	68.0	11250	Pave	IR1	Lvl	AllPub	Inside	Gtl	CollgCr	
3	70	RL	60.0	9550	Pave	IR1	Lvl	AllPub	Corner	Gtl	Crawfor	
4	60	RL	84.0	14260	Pave	IR1	Lvl	AllPub	FR2	Gtl	NoRidge	

In [6]:

```
# function: Distribution of target/dependent variable
def target_distribution(target):
    from scipy import stats

    plt.figure(figsize = (12,8))
    plot1 = sns.distplot(target , fit = stats.norm)

    # getting the params
    (mu, sigma) = stats.norm.fit(target)

    # labelling
    plt.legend(["Normal dist. ($/mu=$ {:.2f} and $/sigma=$ {:.2f})".format(mu, sigma)], loc="best")

    # making the QQ plot / Probability plot
    fig = plt.figure()
    plot2 = stats.probplot(target, plot = plt)
    plt.show()

    print(plot1)
    print(plot2)

target_distribution(dataset["SalePrice"])
```

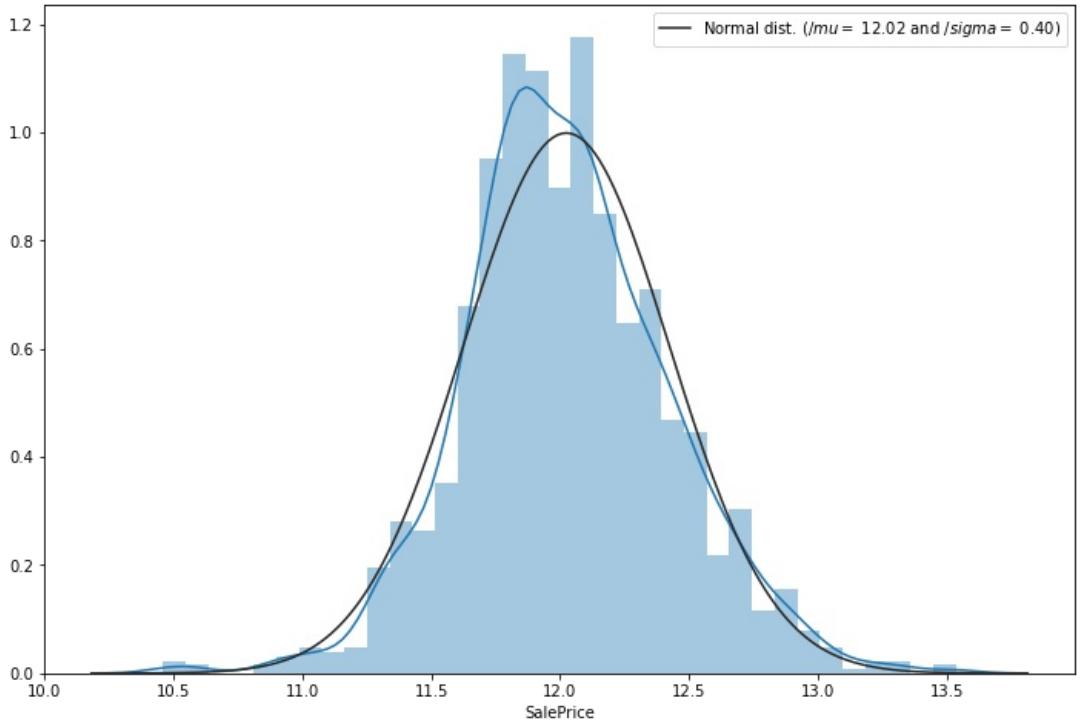


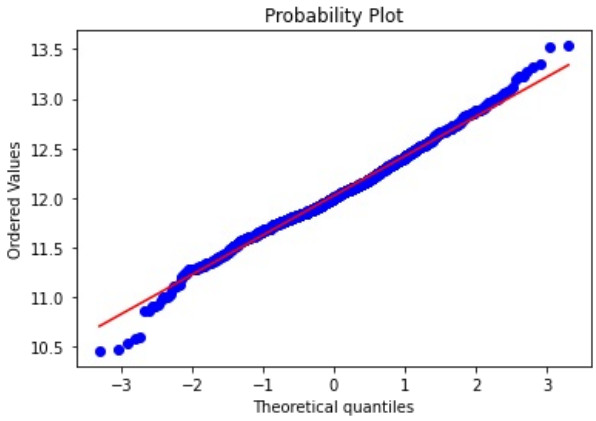



```
AxesSubplot(0.125,0.125;0.775x0.755)
((array([-3.30513952, -3.04793228, -2.90489705, ...,  2.90489705,
         3.04793228,  3.30513952]), array([ 34900,  35311,  37900, ..., 625000, 745000, 755000], dtyp
e=int64)), (74160.16474519415, 180921.19589041095, 0.9319665641512986))
```

In [7]:

```
# function: log Distribution
def log_distribution(target):
    from scipy import stats
    target = np.log(target)

    plt.figure(figsize = (12,8))
    plot1 = sns.distplot(target , fit = stats.norm)

    # getting the params
    (mu, sigma) = stats.norm.fit(target)

    # legenf of the distribution
    plt.legend(["Normal dist. ($/mu=$ {:.2f} and $/sigma=$ {:.2f})".format(mu, sigma)], loc="best")

    # making the QQ plot / Probability plot
    fig = plt.figure()
    plot2 = stats.probplot(target, plot = plt)
    plt.show()

    print(plot1)
    print(plot2)

log_distribution(dataset["SalePrice"])
```






```
AxesSubplot(0.125,0.125;0.775x0.755)
((array([-3.30513952, -3.04793228, -2.90489705, ...,  2.90489705,
         3.04793228,  3.30513952]), array([10.46024211, 10.47194981, 10.54270639, ..., 13.34550693,
        13.5211395 , 13.53447303])), (0.3982622308161888, 12.024050901109383, 0.9953761475636613))
```

- From the above plot, we can see that after converting SalePrice to log(SalePrice), then it's distribution becomes normal

In [10]:

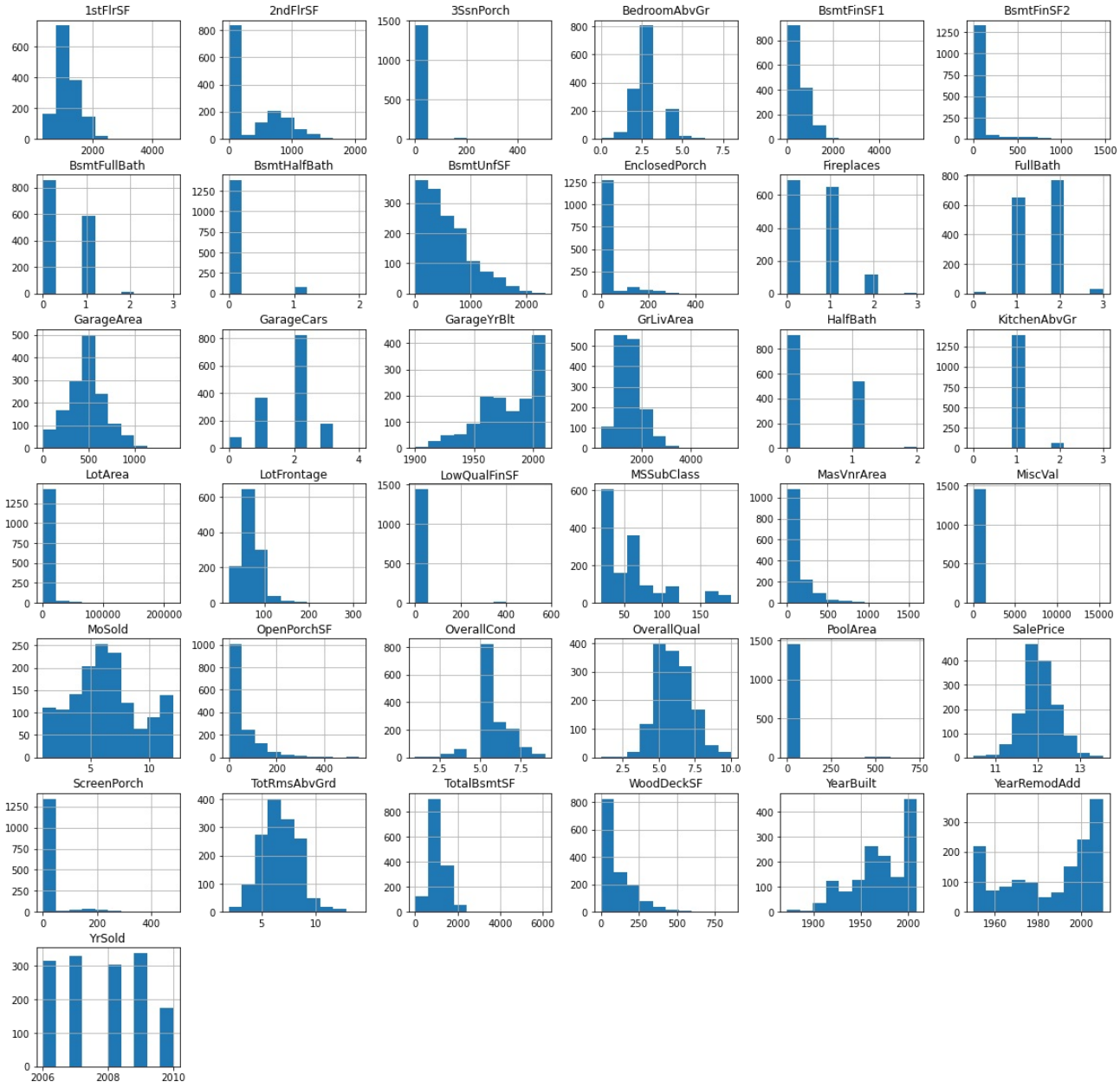
```
# Changing the target to log form
dataset["SalePrice"] = np.log(dataset["SalePrice"])
dataset.head()
```

Out[10]:

	MSSubClass	MSZoning	LotFrontage	LotArea	Street	LotShape	LandContour	Utilities	LotConfig	LandSlope	Neighborhood	Conditi
0	60	RL	65.0	8450	Pave	Reg	Lvl	AllPub	Inside	Gtl	CollgCr	N
1	20	RL	80.0	9600	Pave	Reg	Lvl	AllPub	FR2	Gtl	Veenker	Fe
2	60	RL	68.0	11250	Pave	IR1	Lvl	AllPub	Inside	Gtl	CollgCr	N
3	70	RL	60.0	9550	Pave	IR1	Lvl	AllPub	Corner	Gtl	Crawfor	N
4	60	RL	84.0	14260	Pave	IR1	Lvl	AllPub	FR2	Gtl	NoRidge	N

In [11]:

```
# checking the distribution of features of the dataset
def check_distribution(dataset):
    c_data = dataset.copy()
    c_data.hist(figsize = (20,20))
    plt.show()

check_distribution(dataset)
```

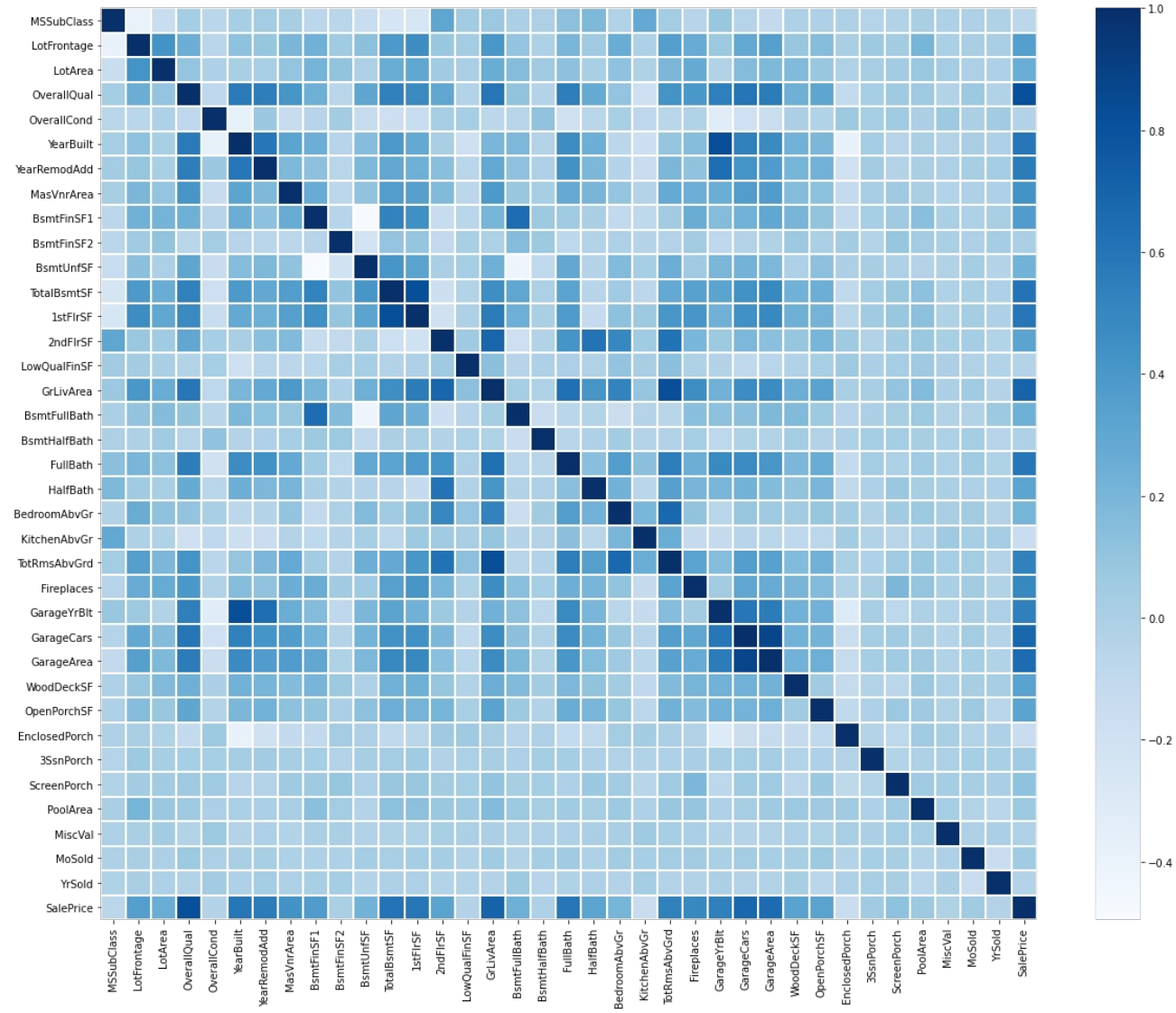


Thus, We dom't have normal distribution of the dataset except SalePrice{which I converted in log form}, we need to change the distribution to get good accuracy

In [13]:

```
# Understanding the Correration-Coefficient
def correration_coefficient(dataset, figsize, annot):
    corr_matrix = dataset.corr()
    plt.figure(figsize = figsize)

    # making the heatmap
    corr_map = sns.heatmap(data = corr_matrix,
                           annot = annot,
                           cmap = "Blues",
                           fmt = "g",
                           linewidths = 1,
                           annot_kws = {"size": 16})
    return corr_map



def select_correration(dataset, threshold):
    # making a set
    corr_set = set()

    # making a corr matrix
    corr_matrix = dataset.corr()

    # select value under some threshold
    for i in range(len(corr_matrix.columns)):
        for j in range(i):
            if abs(corr_matrix.iloc[i,j]) > threshold:
                matrix = corr_matrix.columns[i]

                # adding the values in set
                corr_set.add(matrix)

    print("Number of Correrated features: {}".format(len(corr_set)))
    print("List of Correrated Features: {}".format(list(corr_set)))


corr_plot = correration_coefficient(dataset, (20,16) ,False)
correrated_features = select_correration(dataset, 0.8)
```

```
Number of Correrated features: 5
List of Correrated Features: ['GarageYrBlt', 'TotRmsAbvGrd', '1stFlrSF', 'GarageArea', 'SalePrice']
```

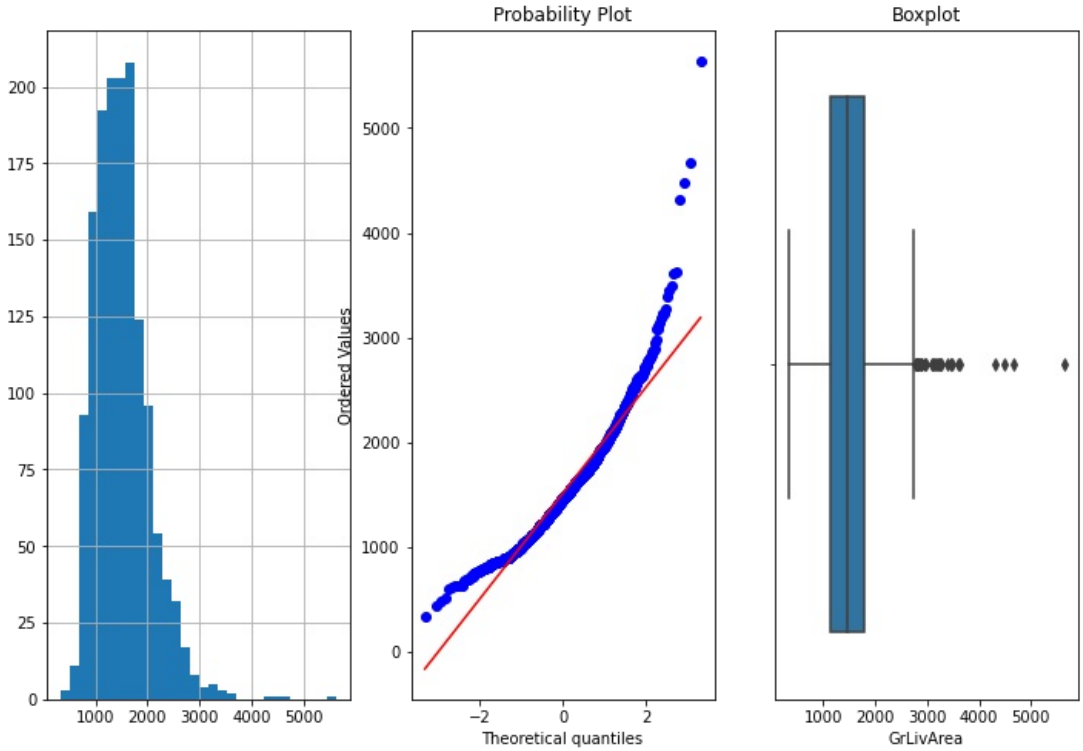


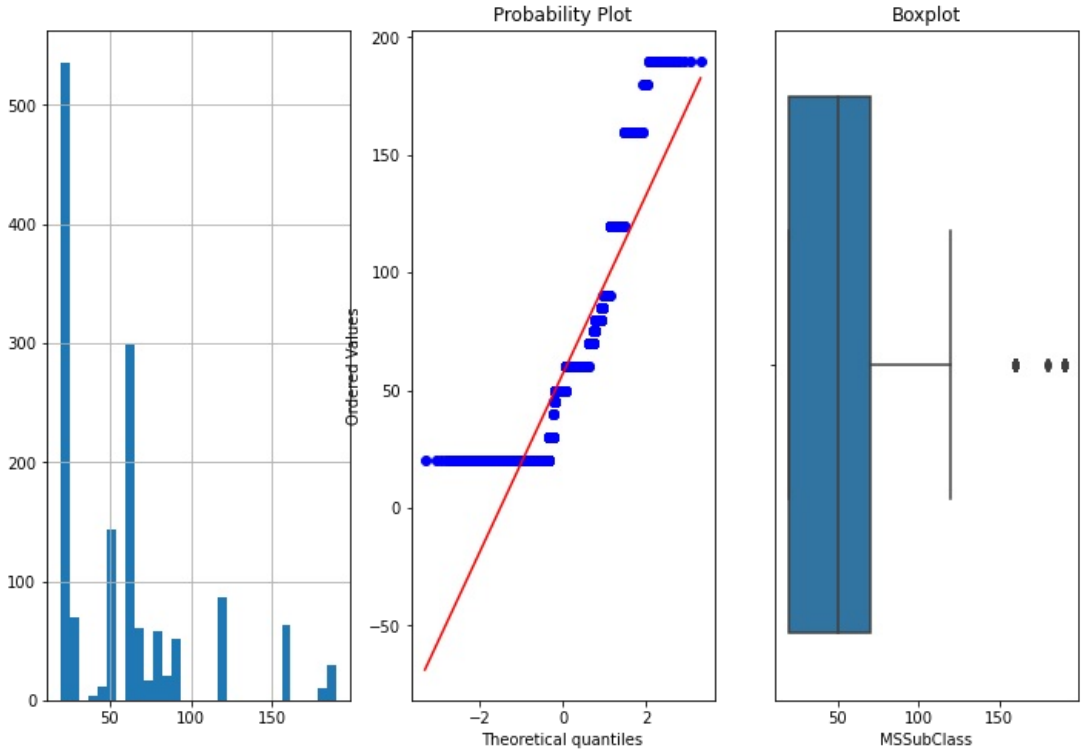
In [14]:

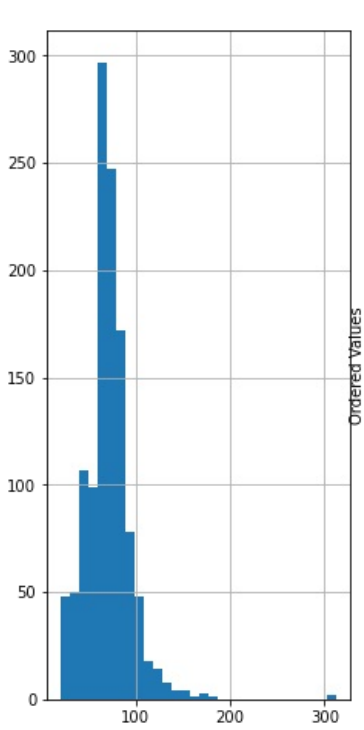
```
# Making the Diagnostic plots
def diagnostic_plots(dataset, variable):
    from scipy import stats

    # plotting the histogram
    plt.figure(figsize = (12,8))
    plt.subplot(1,3,1)
    dataset[variable].hist(bins = 30)

    # plotting the QQ-plot
    plt.subplot(1,3,2)
    stats.probplot(dataset[variable], dist= "norm", plot = plt)


    # plotting the box plot
    plt.subplot(1,3,3)
    sns.boxplot(dataset[variable])
    plt.title("Boxplot")

    plt.show()
```

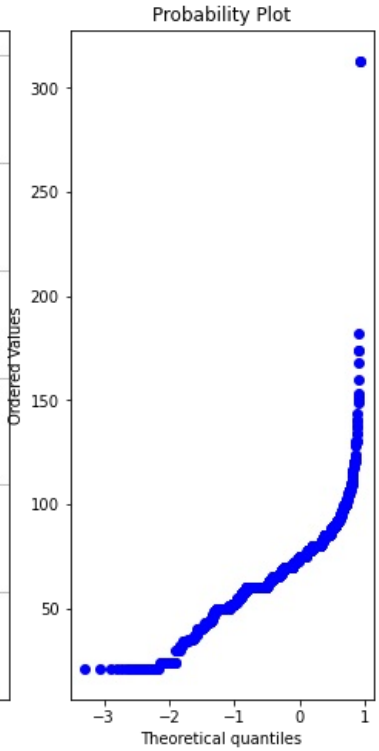
In [15]:

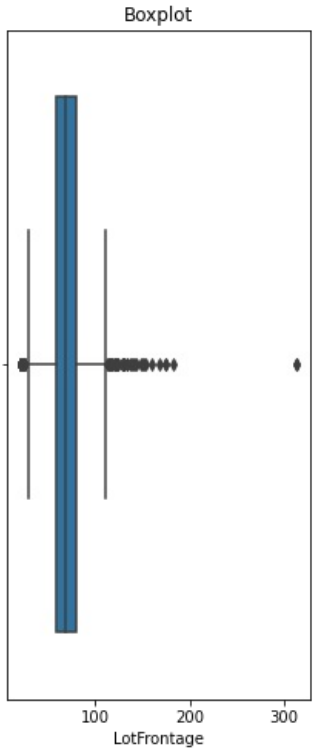
```
# looking at the features
diagnostic_plots(dataset, "GrLivArea")
diagnostic_plots(dataset, "MSSubClass")
diagnostic_plots(dataset, "LotFrontage")
diagnostic_plots(dataset, "LotArea")
diagnostic_plots(dataset, "MasVnrArea")
diagnostic_plots(dataset, "TotalBsmtSF")
```

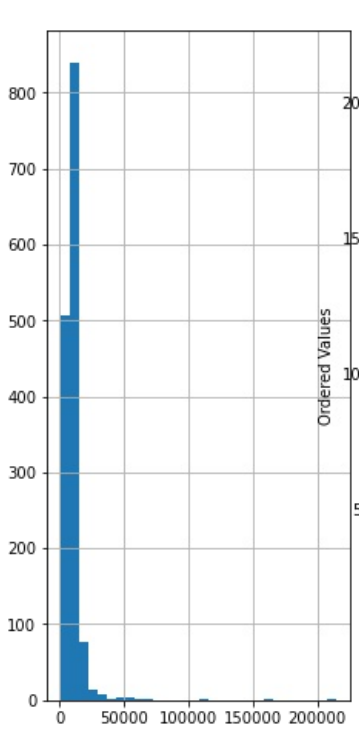

Probability Plot
Boxplot
Ordered Values
Theoretical quantiles
GrLivArea

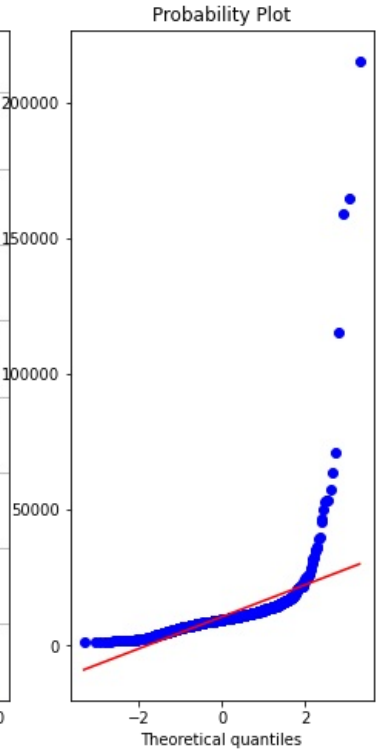

Probability Plot
Boxplot
Ordered Values
Theoretical quantiles
MSSubClass

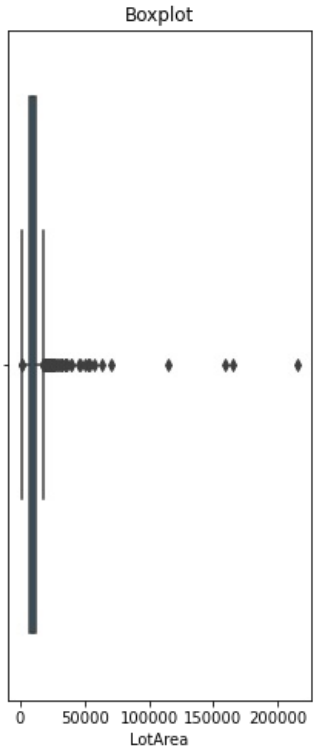


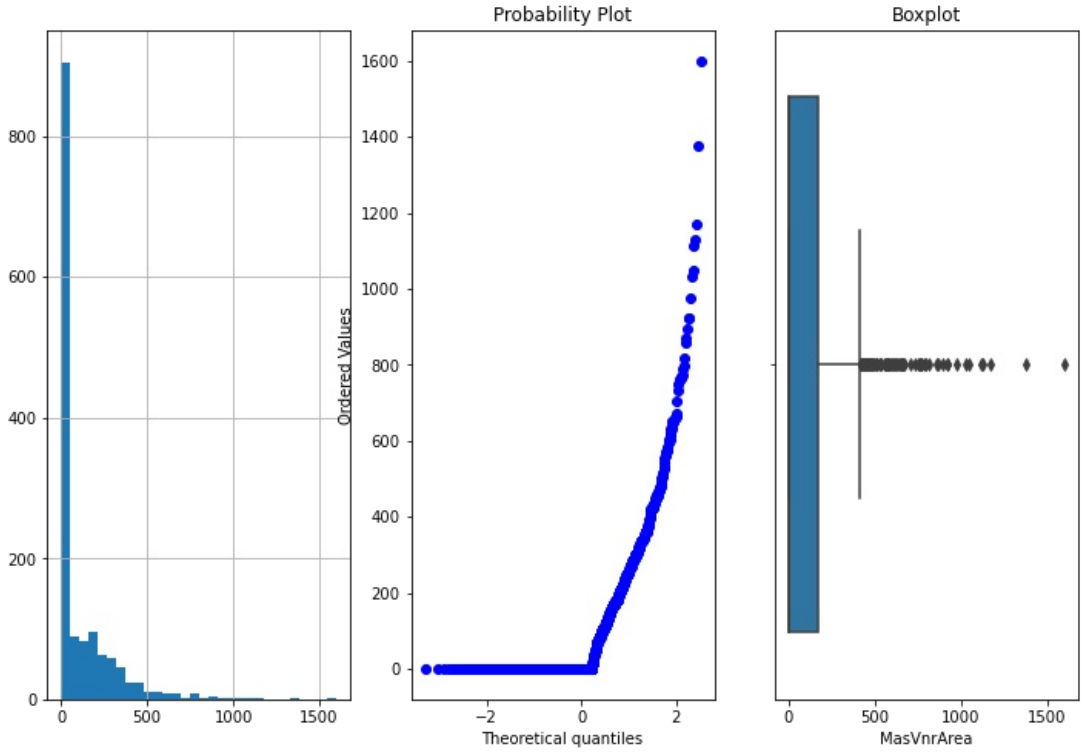

Probability Plot
Ordered Values
Theoretical quantiles

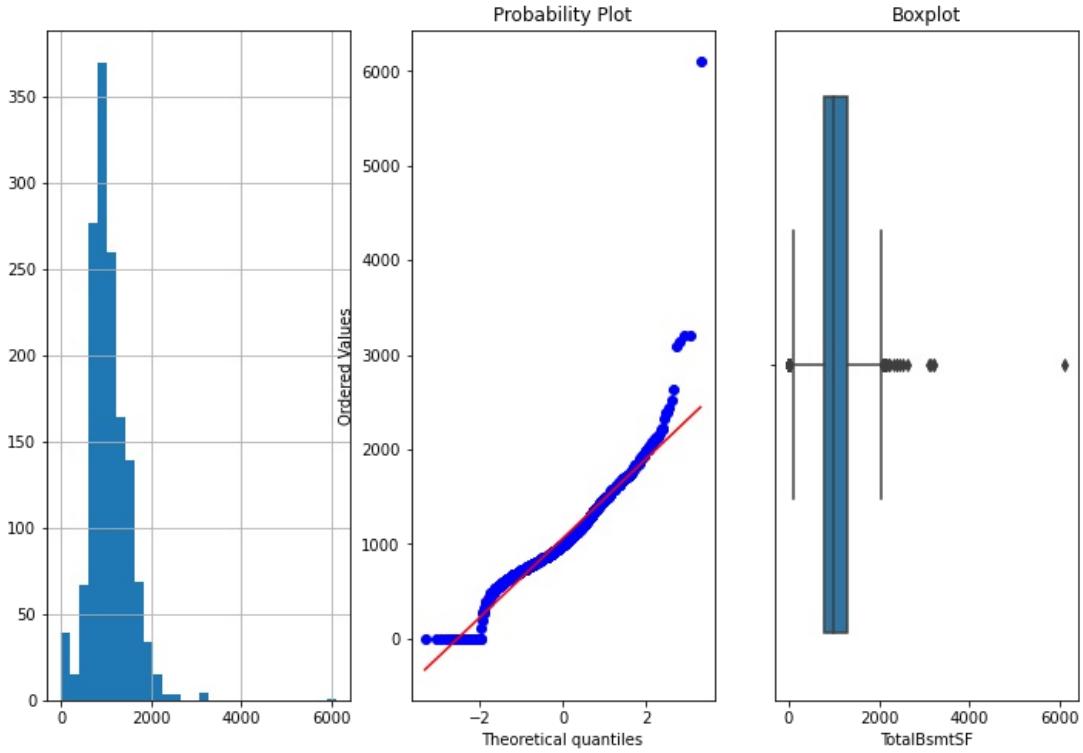

Boxplot
LotFrontage




Probability Plot
Ordered Values
Theoretical quantiles


Boxplot
LotArea





- Thus, we can see that there are many ouliers in the dataset, removing those ouliers and changing the distribution to normal will increase the accuracy of our model

In [18]:

```
# Applying some statistical tests: for Categorical features
def Anova(dataset):
    from scipy import stats
    categoric_features = dataset.select_dtypes(exclude = [np.number]).columns
    dataset[categoric_features] = dataset[categoric_features].fillna("missing")

    # Making the ANOVA
    anova = {'feature':[], 'f':[], 'p':[]}
    for cat in dataset[categoric_features]:
        group_prices = []

        for group in dataset[cat].unique():
            group_prices.append(dataset[dataset[cat] == group]['SalePrice'].values)

        f, p = stats.f_oneway(*group_prices)
        anova['feature'].append(cat)
        anova['f'].append(f)
        anova['p'].append(p)

    anova = pd.DataFrame(anova)
    anova = anova[['feature','f','p']]
    anova.sort_values('p', inplace = True)

    return anova


Anova(dataset)
```

Out[18]:

	feature	f	p
7	Neighborhood	79.520424	4.043304e-243
17	ExterQual	415.303357	6.935425e-195
20	BsmtQual	300.392324	2.031282e-188
29	KitchenQual	393.320196	4.441486e-187
32	GarageFinish	298.769753	4.057458e-151
31	GarageType	121.796388	8.427845e-125
19	Foundation	126.806748	1.350671e-111
26	HeatingQC	110.820436	1.614882e-82
23	BsmtFinType1	71.381758	4.321404e-78
16	MasVnrType	87.834528	6.536414e-67
0	MSZoning	77.607843	1.021343e-59
14	Exterior1st	22.892270	1.053128e-53
15	Exterior2nd	19.922006	2.429938e-49
34	GarageCond	49.379257	2.508058e-47
22	BsmtExposure	60.494587	2.590707e-47
33	GarageQual	47.873273	6.068406e-46
27	CentralAir	205.666987	9.855628e-44
37	SaleCondition	43.076712	1.689708e-41
36	SaleType	24.759990	5.497893e-36
35	PavedDrive	74.898153	1.090995e-31
11	HouseStyle	23.585762	1.636077e-30
28	Electrical	31.067524	4.044951e-30
2	LotShape	46.728762	7.856968e-29
21	BsmtCond	34.504087	1.492754e-27
18	ExterCond	17.344742	6.540305e-14
12	RoofStyle	13.097283	1.705740e-12
10	BldgType	15.211667	3.436794e-12
24	BsmtFinType2	10.953857	5.805632e-12
8	Condition1	8.037522	1.173444e-10
25	Heating	9.907046	2.484312e-09
3	LandContour	12.767270	3.086224e-08
5	LotConfig	8.692440	6.214575e-07
30	Functional	5.901244	4.250370e-06
13	RoofMatl	3.780645	4.504239e-04
9	Condition2	2.527740	1.382042e-02
1	Street	4.814455	2.837931e-02
6	LandSlope	1.083039	3.388375e-01
4	Utilities	0.232689	6.296094e-01

- If P<0.05 we can reject NULL-hypothesis
- The features Street, LandSlope and Utilities have P>0.05, as they don't have much impact in SalePrice, so we can drop it.
- P-value {my layman definition}: It is the measure of surprize, higher the p-value then less surprized you should be and lower the p- value then more surprized you should be as the data is not what you expected to be

In [19]:

```
# dropping the irrelevent features
dataset.drop(columns= ["Street", "LandSlope", "Utilities"],
             axis= 1,
             inplace = True)

dataset.head()
```

Out[19]:

	MSSubClass	MSZoning	LotFrontage	LotArea	LotShape	LandContour	LotConfig	Neighborhood	Condition1	Condition2	BldgType
0	60	RL	65.0	8450	Reg	Lvl	Inside	CollgCr	Norm	Norm	1Fam
1	20	RL	80.0	9600	Reg	Lvl	FR2	Veenker	Feedr	Norm	1Fam
2	60	RL	68.0	11250	IR1	Lvl	Inside	CollgCr	Norm	Norm	1Fam
3	70	RL	60.0	9550	IR1	Lvl	Corner	Crawfor	Norm	Norm	1Fam
4	60	RL	84.0	14260	IR1	Lvl	FR2	NoRidge	Norm	Norm	1Fam

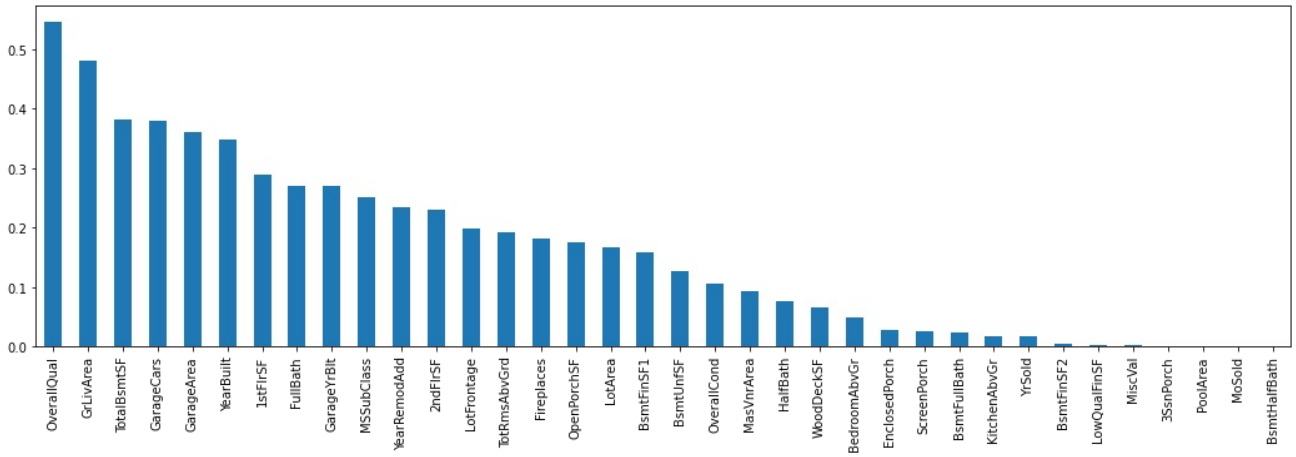
In [20]:

```
# function: Most impacts of the features wrt target
def mutual_information(dataset):

    # Choosing the numeric features
    numerics = ["int16", "int32", "int64", "float16", "float32", "float64"]
    numeric_vars = list(dataset.select_dtypes(include = numerics).columns)
    dataset = dataset[numeric_vars]
    print("Shape of Features Using for the Mutual-Information : {}".format(dataset.shape))


    # Splitting the numerical dataset into train and test set
    from sklearn.model_selection import train_test_split
    x_train, x_test, y_train, y_test = train_test_split(dataset.iloc[:,:-1],
                                                        dataset.iloc[:,-1],
                                                        test_size = 0.3,
                                                        random_state = 0)

    from sklearn.feature_selection import mutual_info_regression
    from sklearn.feature_selection import SelectPercentile


    mi = mutual_info_regression(x_train.fillna(0), y_train)
    mi = pd.Series(mi)
    mi.index = x_train.columns
    mi = mi.sort_values(ascending = False)

    # Plotting the Bar-plot of the dataset
    mi.sort_values(ascending = False).plot.bar(figsize = (18,5))

    # Selecting the best Numeric-Features
    features = SelectPercentile(mutual_info_regression,
                                percentile = 10).fit(x_train.fillna(0), y_train)

    # Returning the Support of the columns of the x_train
    return x_train.columns[features.get_support()]

mutual_information(dataset)
```

```
Shape of Features Using for the Mutual-Information : (1460, 37)
```

Out[20]:

```
Index(['OverallQual', 'TotalBsmtSF', 'GrLivArea', 'GarageCars'], dtype='object')
```



- With the above plot, these numeric features has the more impacts in target. Thus Using these features will improve the model performance

Next time....

Feature Engineering

- function: Encoding
- function: Missing Values
- function: Diagnostic Plots
- function: Outliers Treatment
- function: Scalling
- and many more .....