Predicting Houseprice:

This project was completed has part of Kaggle competition.

In this project we will dive deep into the following areas

1. Exploratory Data Analysis a. Dependent variable b. Univariate Analysis on Independent Variables a. Dependent variable analysis b. Univariate Analysis
   A. Distribution of Numerical variables:
   B. Distribution of Categorical Variables c. Bivariate Analysis
   C. Correlation among Numerical Variables 2, Mean Disparity among Categorical Variables d. Missing Values e. Oulier detection

In [1]:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from scipy import stats
from scipy.stats import skew, norm
from scipy.special import boxcox1p
from scipy.stats import boxcox_normmax
from scipy.stats.stats import pearsonr

# Models
from sklearn.ensemble import RandomForestRegressor, GradientBoostingRegressor
from sklearn.kernel_ridge import KernelRidge
from sklearn.linear_model import Ridge, RidgeCV
from sklearn.linear_model import ElasticNet, ElasticNetCV
from sklearn.svm import SVR
from mlxtend.regressor import StackingCVRegressor
import lightgbm as lgb
from lightgbm import LGBMRegressor
from xgboost import XGBRegressor

# Stats
from scipy.stats import skew, norm
from scipy.special import boxcox1p
from scipy.stats import boxcox_normmax

# Misc
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import KFold, cross_val_score
from sklearn.metrics import mean_squared_error
from sklearn.preprocessing import OneHotEncoder
from sklearn.preprocessing import LabelEncoder
from sklearn.pipeline import make_pipeline
from sklearn.preprocessing import scale
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import RobustScaler
from sklearn.decomposition import PCA


# Ignore useless warnings
import warnings
warnings.filterwarnings(action="ignore")
pd.options.display.max_seq_items = 8000
pd.options.display.max_rows = 8000
```

In [2]:
```
df=pd.read_csv('C:/Users/thand/Downloads/house-prices-advanced-regression-tec
df.head()
```

Out[2]:

	Id	MSSubClass	MSZoning	LotFrontage	LotArea	Street	Alley	LotShape	LandContour	U
0	1	60	RL	65.0	8450	Pave	NaN	Reg	Lvl	
1	2	20	RL	80.0	9600	Pave	NaN	Reg	Lvl	
2	3	60	RL	68.0	11250	Pave	NaN	IR1	Lvl	
3	4	70	RL	60.0	9550	Pave	NaN	IR1	Lvl	
4	5	60	RL	84.0	14260	Pave	NaN	IR1	Lvl	

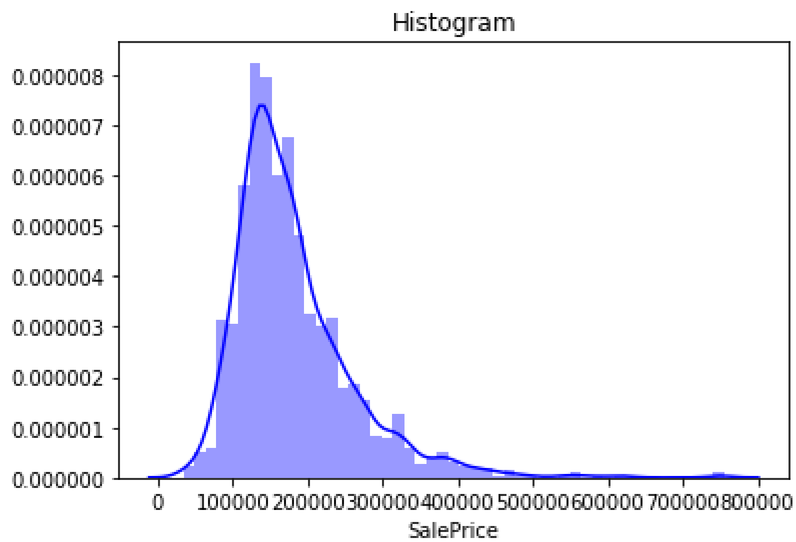
5 rows × 81 columns

In [3]:
```
#Lets have a peak at the columns
print('There are about {} columns'.format(len(df.columns)))
print(df.columns)
```

```
There are about 81 columns
Index(['Id', 'MSSubClass', 'MSZoning', 'LotFrontage', 'LotArea', 'Street',
       'Alley', 'LotShape', 'LandContour', 'Utilities', 'LotConfig',
       'LandSlope', 'Neighborhood', 'Condition1', 'Condition2', 'BldgType',
       'HouseStyle', 'OverallQual', 'OverallCond', 'YearBuilt', 'YearRemodA
dd',
       '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', 'FullBa
th',
       'HalfBath', 'BedroomAbvGr', 'KitchenAbvGr', 'KitchenQual',
       'TotRmsAbvGrd', 'Functional', 'Fireplaces', 'FireplaceQu', 'GarageTy
pe',
       'GarageYrBlt', 'GarageFinish', 'GarageCars', 'GarageArea', 'GarageQu
al',
       'GarageCond', 'PavedDrive', 'WoodDeckSF', 'OpenPorchSF',
       'EnclosedPorch', '3SsnPorch', 'ScreenPorch', 'PoolArea', 'PoolQC',
       'Fence', 'MiscFeature', 'MiscVal', 'MoSold', 'YrSold', 'SaleType',
       'SaleCondition', 'SalePrice'],
      dtype='object')
```

In [15]:
```
#Lets take a look at the dependent variable
continuousVariable(df,'SalePrice')
```

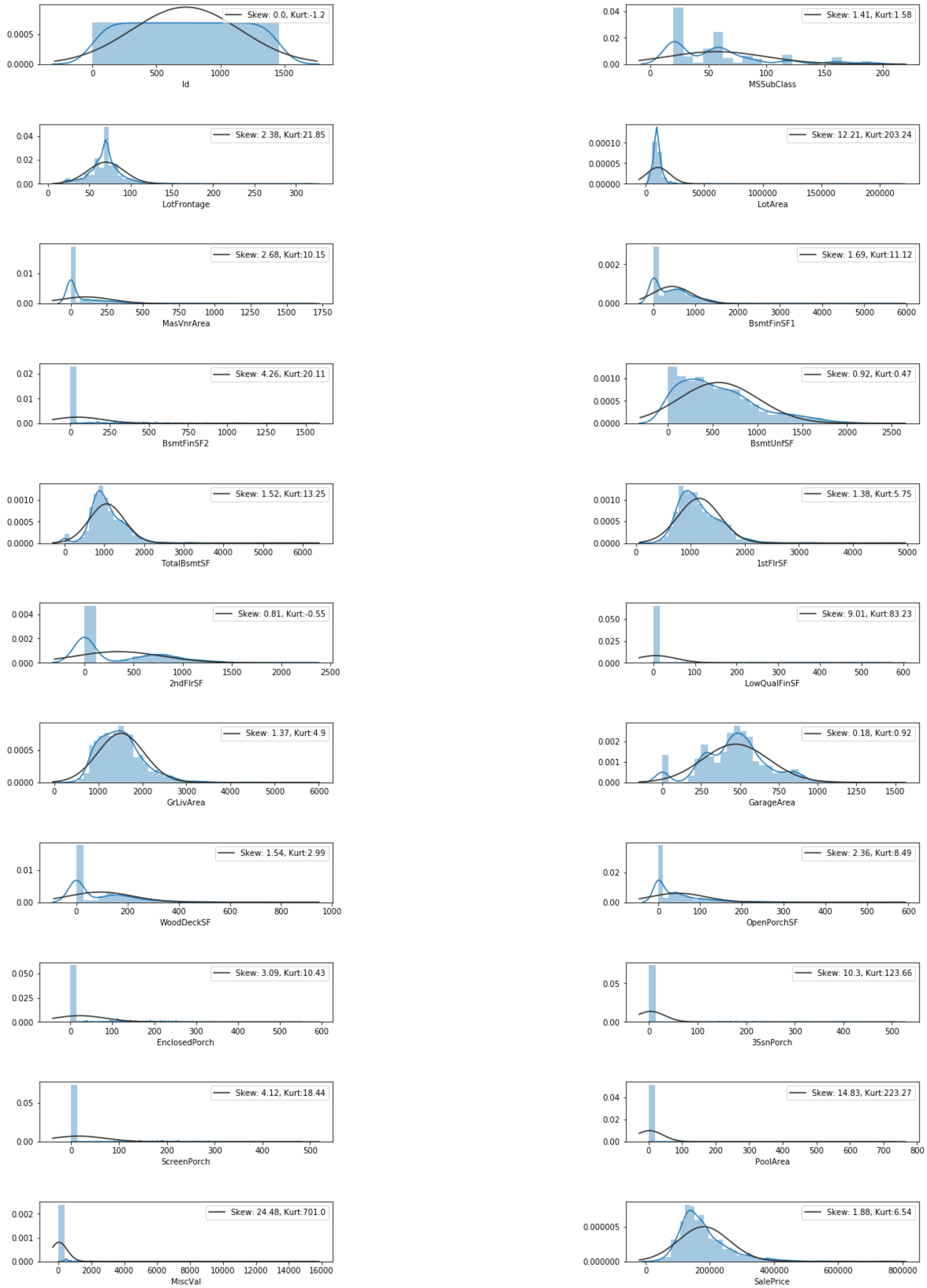
```
Skewness:1.8828757597682129
Kurtosis:6.536281860064529
```



The Dependent variable, SalePrice is Left skewed, Non Normal and Leptokurtic. Needs transformations for linear regression

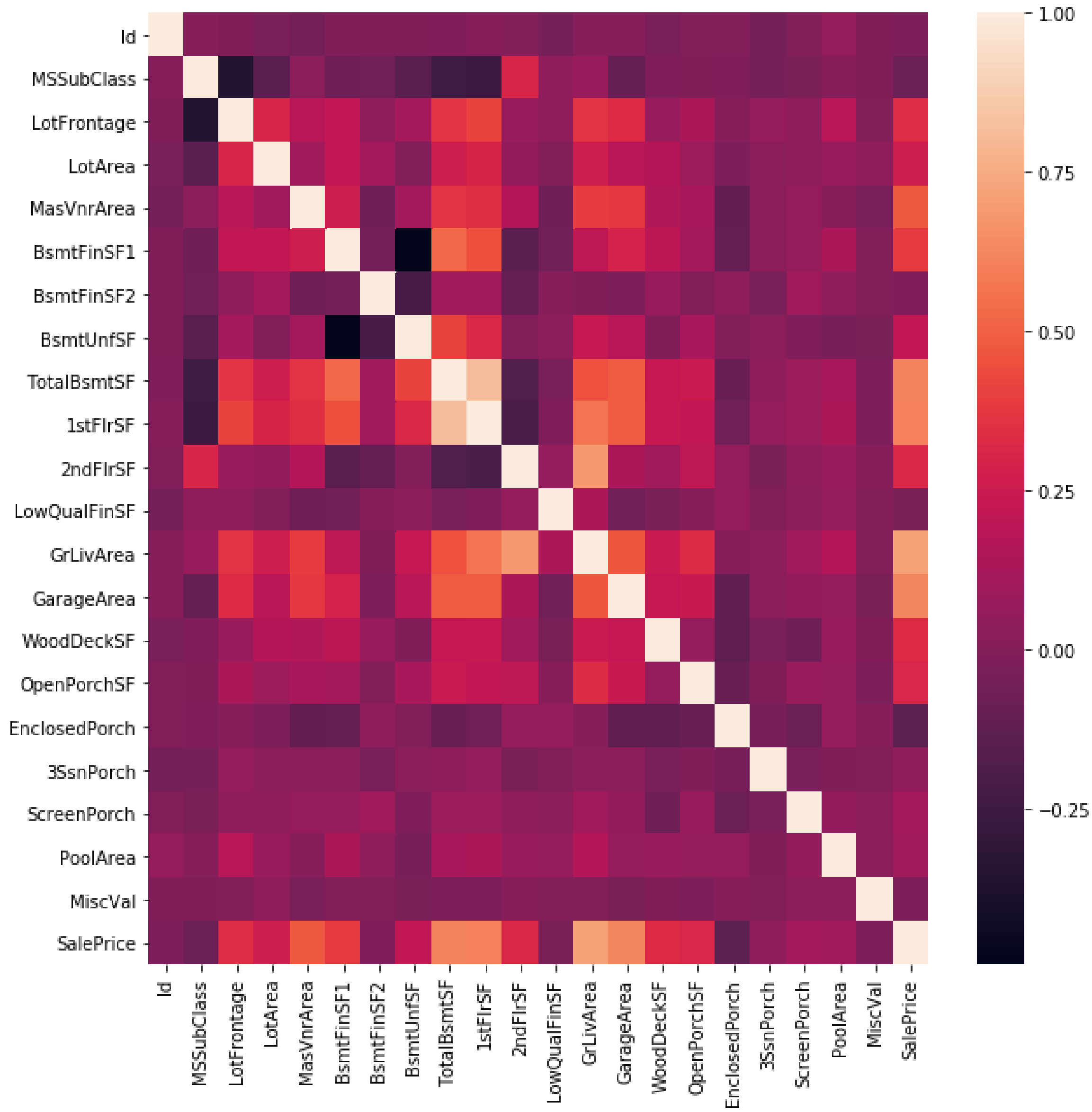
In [16]:
```
# Variables by data types

#In this section, we will make the following lists
#num_cols: list of all numerical variables
#cat_cols: list of all categorical variables
#ordinal_cols: list of all mean encoded ordinal variables


#discrete ratio/interval to categorical features to get more info on them
numericToCategory(df,['MoSold','YrSold','TotRmsAbvGrd','OverallQual','Overall
numerics = ['int16', 'int32', 'int64', 'float16', 'float32', 'float64']
num_cols= df.select_dtypes(include=numerics).columns   #list of all numerical
#Nominal/ordinal
```

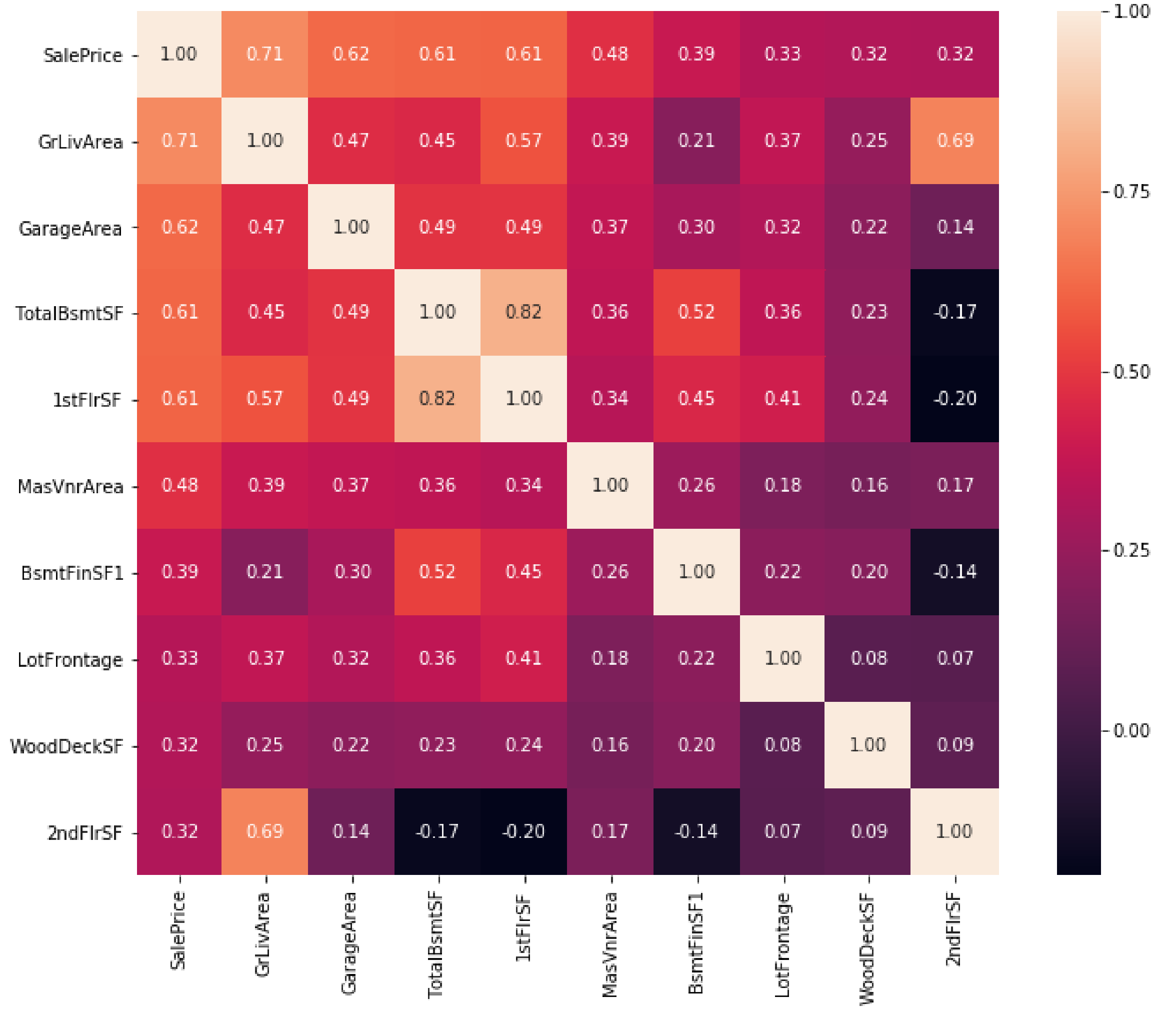
In [17]:
```python
# Numercial variables distribution
NumericalEDA('Histogram',22,2,df,df.select_dtypes(include=numerics).columns,'
```



Most of the variables are not Normal, heavily skewed. Needs processing.

In [19]:
```
# Lets look at highly correlated Numerical variables

NumericalEDA('Heatmap',18,2,df,df.select_dtypes(include=numerics).columns,'Sa
```





Top 5 highly correlated variables with SalePrice : GrLivArea, GarageArea, TotalBsmtSF, 1stFlrSF, MasVnrArea

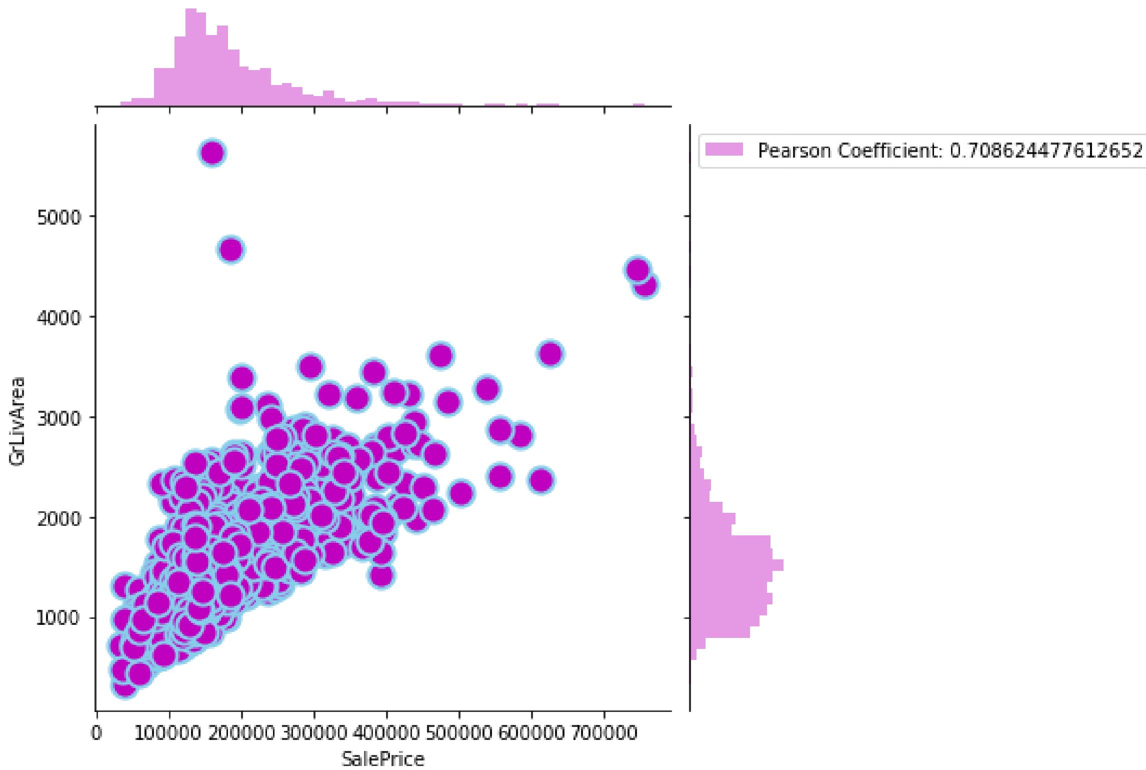
There are Many correlated independent variables as well 'GrLivArea'--> 2ndFlrSF,1ndFlrSF 'TotalBsmtSF'-->BsmtFinSF1, 1stFlrSF

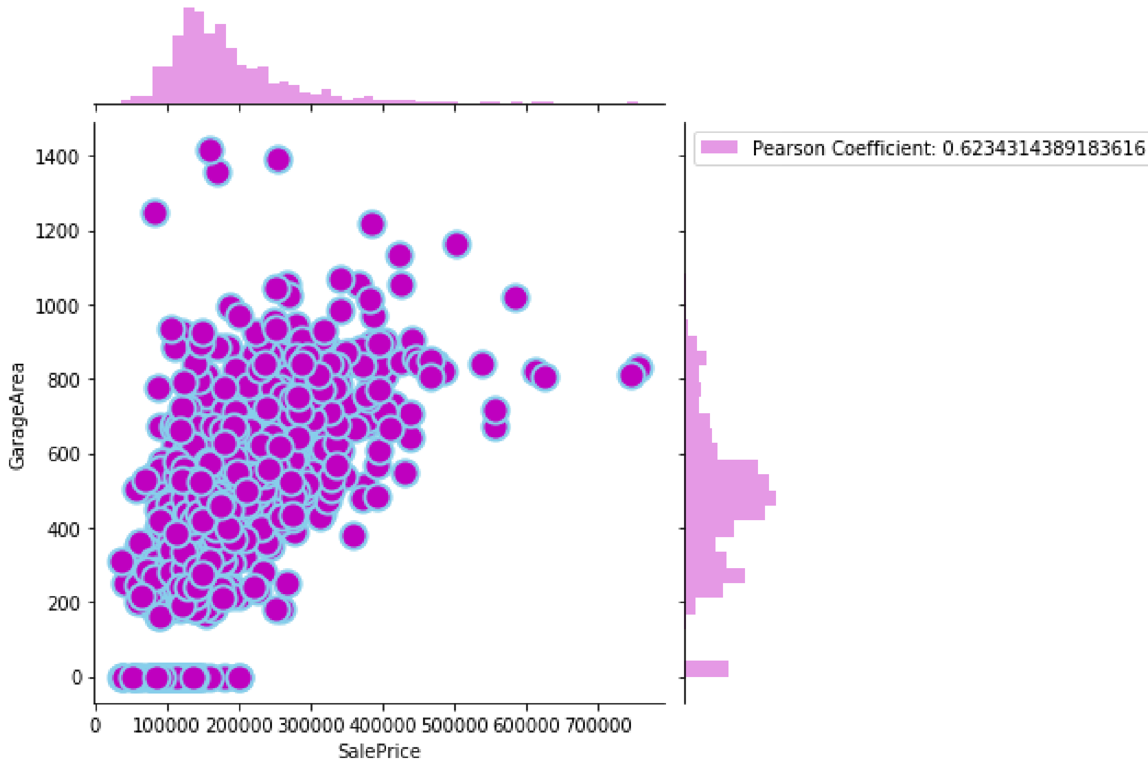
Processing is need while modelling Linear regression

In [20]:

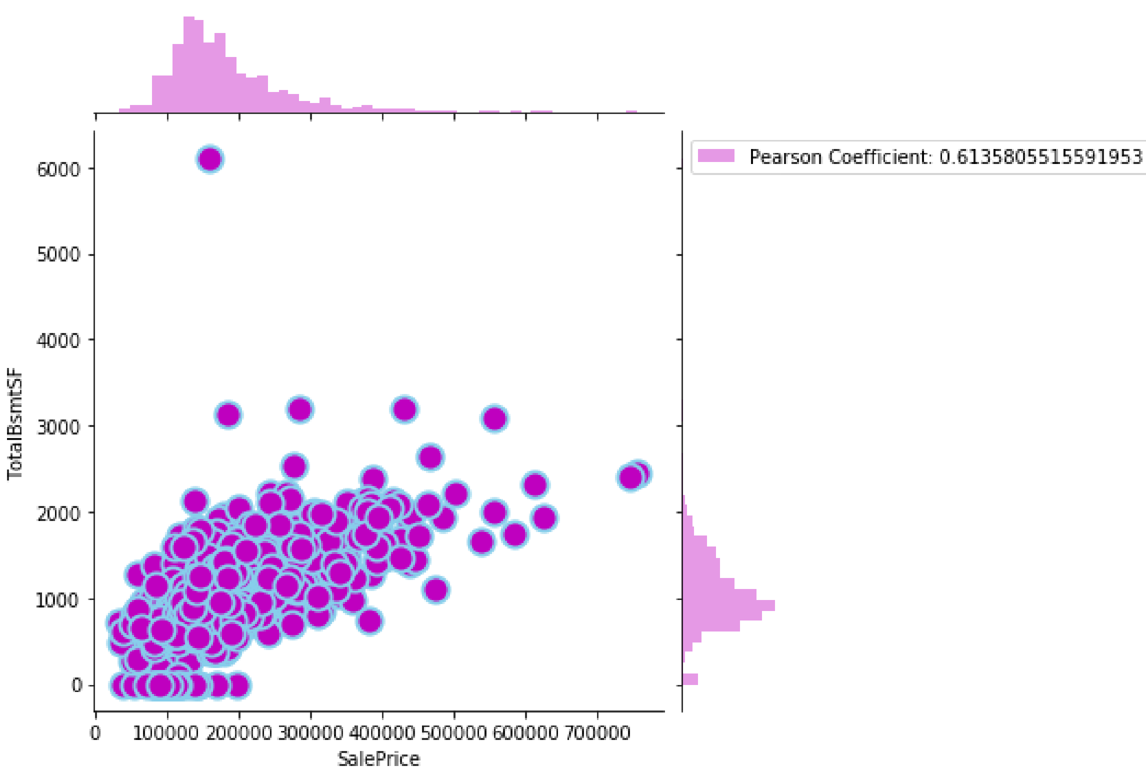
```
# Pair plot on the top 5 correlated variables with SalePrice
NumericalEDA('pair plot',5,1,df,['GrLivArea', 'GarageArea', 'TotalBsmtSF', '1
```

```
<Figure size 1440x4320 with 0 Axes>
```

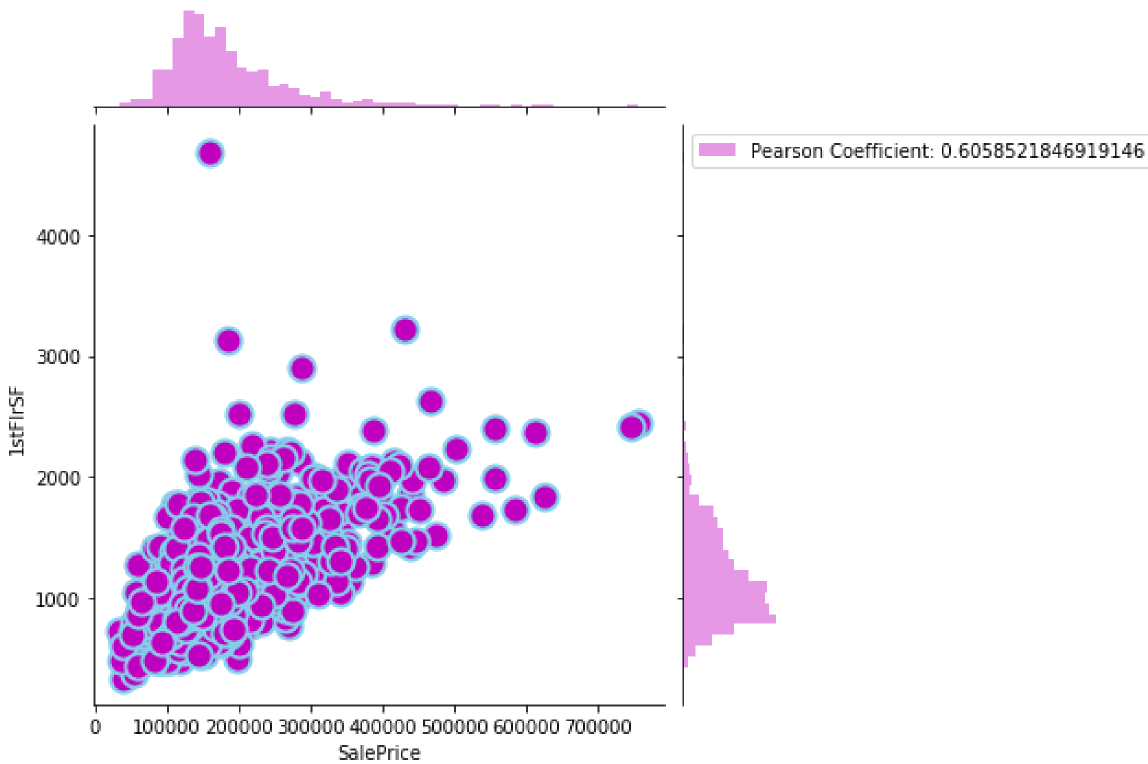


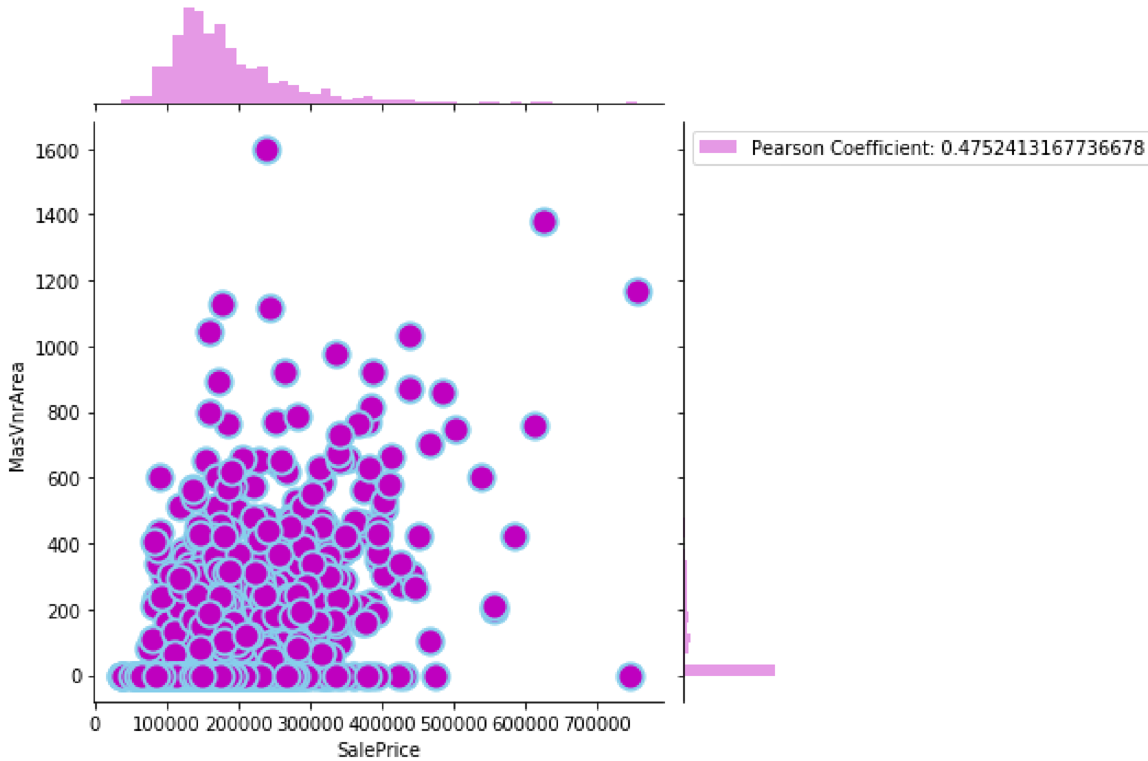


Pearson Coefficient: 0.6234314389183616
GarageArea
SalePrice
0
200
400
600
800
1000
1200
1400
0 100000 200000 300000 400000 500000 600000 700000



Pearson Coefficient: 0.6135805515591953
TotalBsmtSF
SalePrice
0
1000
2000
3000
4000
5000
6000
0 100000 200000 300000 400000 500000 600000 700000

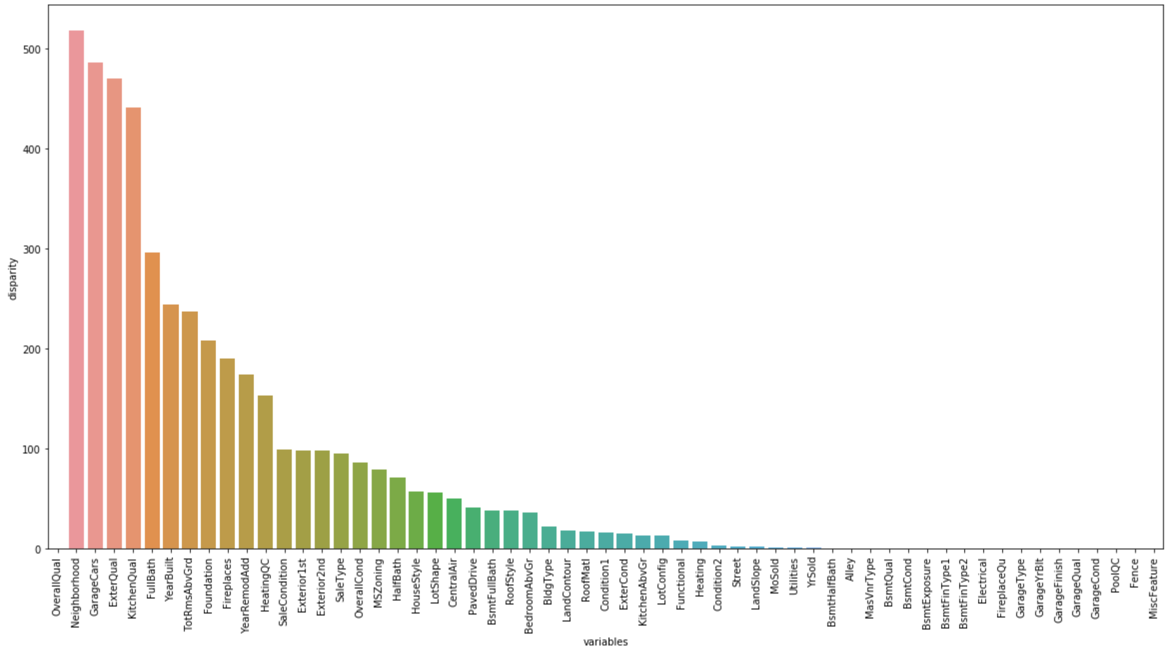

Pearson Coefficient: 0.6058521846919146
4000
3000
2000
1000
1stFlrSF
0
100000
200000
300000
400000
500000
600000
700000
SalePrice



Looking at the scatterplot of the top 5 correlated numerical variables, we observe the following

1. There are prominent outliers in the data We need to process these before modelling linear regression

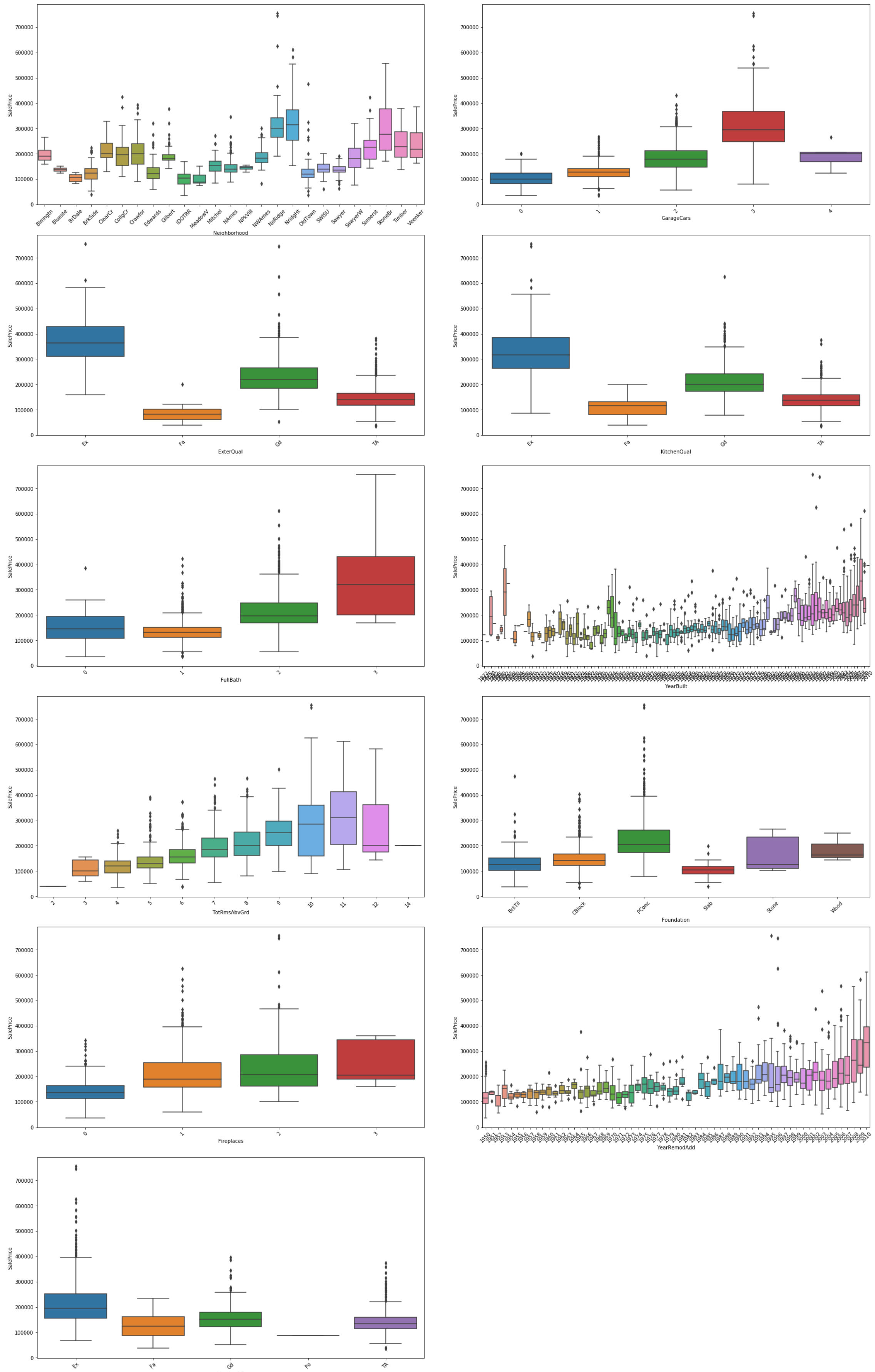
In [21]:
```
# categorical data
# 1 way anova test

categoricalEDA(viz_type='Categorical-VariableImportance',data=df,label='SaleP
```



From the above graph, we can see that OverallQaul, Neighborhood, GarageCars, ExterQual, KitchenQual, FullBath, YearBuilt, TotRmsAbvGrd, Foundation, Fireplaces, YearRemodAdd, HeatingQc have significant variability with SalePrice. Lets dig Deep

In [22]:

```
cat_=['OverallQual', 'Neighborhood', 'GarageCars', 'ExterQual', 'KitchenQual'
categoricalEDA(viz_type='boxplot',data=df,label='SalePrice',column=cat_,rows=
```



Houses with the below qualities are significantly priced higher

1. Excellent and Good External Quality
2. Excellent Kitchen Quality
3. 2 or more fullbaths
4. 10+ Rooms
5. Concrete Foundations
6. Excellent heating quality
7. Avaialability of Fireplace
8. Recently remodelled houses

In [5]:

```
def continuousVariable(df=None,col=None):
    df[col].describe()
    ax=sns.distplot(df[col],color="Blue")
    ax.set(xlabel=col, title="Histogram")
    print("Skewness:{}".format(df[col].skew()))
    print("Kurtosis:{}".format(df[col].kurt()))
```

In [6]:

```

def ordinal_meanencoding(data=None, col=None, label=None, ordinalCols=None):
    for i in col:
        a=i+'Ordinal'
        ordinalCols.append(a)
        df[a]=df[i].map(dict(zip(df.groupby(i)[label].mean().sort_values(asce
```

In [7]:

```
def NumericalEDA(viz_type=None,rows=None,cols=None,data=None,column=[],label=
    if viz_type=='Histogram':
        fig=plt.figure(figsize=(20,60))
        fig.subplots_adjust(hspace=1,wspace=1)
        c=0
        for i in range(1,rows+1):
            if data[column[c]].isnull().any():
                data[column[c]]=data[column[c]].fillna(data[column[c]].mean()
            ax=fig.add_subplot(rows,cols,i)
            ax=sns.distplot(df[column[c]], fit=stats.norm)
            #ax.set(title=column[c])
            plt.legend(["Skew: {}, Kurt:{}".format(round(df[column[c]].skew()
            c+=1
        plt.show()

    if viz_type=='Heatmap':
        corrmat=data.corr()
        f,ax=plt.subplots(figsize=(10,10))
        sns.heatmap(corrmat)
        plt.show()

        cols=corrmat.nlargest(10,label)[label].index
        corr=data[cols].corr()
        f,ax=plt.subplots(figsize=(12,9))
        sns.heatmap(corr,cbar=True, annot=True, square=True, fmt='.2f', annot
        plt.show()

    if viz_type=='joint plot':
        corrmat=data.corr()
        f,ax=plt.subplots(figsize=(10,10))
        sns.heatmap(corrmat)
        plt.show()

        cols=corrmat.nlargest(10,label)[label].index
        f,ax=plt.subplots()
        sns.pairplot(data[cols],size=2.5)
        plt.show()

    if viz_type=='pair plot':
        fig=plt.figure(figsize=(20,60))
        for i in range (1,rows+1):
            #plt.close()
            #fig.add_subplot(rows,cols,i)
            sns.jointplot(x=data[label], y=df[column[i-1]], kind='scatter', s
            plt.legend(["Pearson Coefficient: {}".format(pearsonr(df[label],c
        fig.subplots_adjust(hspace=1,wspace=1)
        plt.show()
```

In [8]:
```
def categoricalEDA(viz_type=None,data=None,column=[],label=None,rows=None,col

    if viz_type=='boxplot':

        for col in column:
            data[col]=data[col].astype('category')
            if data[col].isnull().any():
                data[col]=data[col].cat.add_categories(['Missing'])
                data[col]=data[col].fillna('Missing')

        f=plt.figure(figsize=(30,50))
        f.subplots_adjust(hspace=0.15,wspace=0.15)
        for i in range(1,rows*2):
            ax=f.add_subplot(rows,cols,i)
            ax=sns.boxplot(x=data[column[i]],y=data[label])
            plt.xticks(rotation=45)
        plt.show()

    if viz_type=='Categorical-VariableImportance':

        anv_df=pd.DataFrame()
        anv_df['variables']=column

        pvals=[]
        catval=[]
        for col in column:
            catval=[]
            for j in df[col].unique():
                catval.append(df.loc[df[col]==j,label])
            #calculcate pvalue
            pvals.append(stats.f_oneway(*catval)[1])
        anv_df['pval']=pvals
        anv_df['disparity']=np.log(1/anv_df['pval'].values)
        anv_df.sort_values('disparity',ascending=False,inplace=True)
        fig=plt.figure(figsize=(20,10))
        fig=sns.barplot(x=anv_df['variables'],y=anv_df['disparity'])
        plt.xticks(rotation=90)
        plt.show()
```

In [9]:
```
#missing value
def missing_values(data=None,influential_variable=None,label=None):
    total=data.isnull().sum()
    percent=(data.isnull().sum()/data.isnull().count()).sort_values(ascending
    missing_data=pd.concat([total,percent],axis=1,keys=['total','Percent'],so
    dat=missing_data.loc[missing_data['total']>0,:].reset_index().sort_values
    fig=plt.figure(figsize=(20,10))
    ax=sns.barplot(x='index',y='Percent',data=dat)
    ax.set(ylabel="Percent of missing Data")
    ax.set(xlabel="Features")
    ax.set(title="Missing Values (in Percent)")
    plt.xticks(rotation=45)
```

In [10]:
```
def numericToCategory(data=None,columns=None):

    for col in columns:
        data[col]=data[col].astype('category')
```

In [11]:
```
def missingValueImputation(data=None,numerical_col=None,categorical_cols=None


    number=[]
    cat=[]


    if numerical_col is not None:

        if numercial_method=='mean_imputation':
            for col in numerical_col:
                data[col]=data[col].fillna(data[col].mean())

        if numercial_method=='median_imputation':
            for col in numerical_col:
                data[col]=data[col].fillna(data[col].median())

        if numercial_method=='value_fill':
            for col in numerical_col:
                data[col]=data[col].fillna(numercial_value)

        for col in number:
            data=data.drop(col,inplace=True)


    if categorical_cols is not None:

        if categorical_method=='popular_imputation':
            for col in categorical_cols:
                data[col]=data[col].fillna(data[col].value_counts().index[0])

        if categorical_method=='value_fill':
            for col in categorical_cols:
                data[col]=data[col].fillna(categorical_value)

        for col in cat:
            data=data.drop(col,inplace=True)
```

In [12]:

```
n=[]
s=[]
def normalize(data=None, columns=None, std=True):



    if std==True:
        highly_skewed_col=data[columns].columns[df[columns].apply(lambda x: x
        f,ax=plt.subplots()
        plt.figure(figsize=(10,5))
        ax=sns.barplot(x=data[highly_skewed_col].columns,y=data[highly_skewed
        ax.xaxis.grid(False)
        ax.set(ylabel="Skew")
        ax.set(xlabel="Feature names")
        ax.set(title="Highly skewed Variables: Before Transformation")
        sns.despine(trim=True, left=True)
        plt.xticks(rotation=45)
        plt.figure(figsize=(10,5))
        plt.show()

        for cols in highly_skewed_col:
            power=boxcox_normmax(data[cols] + 1)
            n.append((cols,power))
            data[cols] = stats.boxcox(df[[cols]]+1)[0]

        scalar=StandardScaler().fit(df[columns])
        df[columns] = scalar.transform(df[columns])
        s.append(scalar)



    if std==False:

        highly_skewed_col=data[columns].columns[df[columns].apply(lambda x: x
        f,ax=plt.subplots()
        plt.figure(figsize=(10,5))
        ax=sns.barplot(x=data[highly_skewed_col].columns,y=data[highly_skewed
        ax.xaxis.grid(False)
        ax.set(ylabel="Skew")
        ax.set(xlabel="Feature names")
        ax.set(title="Highly skewed Variables: Before Transformation")
        sns.despine(trim=True, left=True)
        plt.xticks(rotation=45)
        plt.show()

        for cols in highly_skewed_col:
            power=boxcox_normmax(data[cols] + 1)
            n.append((cols,power))
            data[cols] = stats.boxcox(df[[cols]]+1)[0]

    ax=sns.barplot(x=data[highly_skewed_col].columns,y=data[highly_skewed_col
    ax.xaxis.grid(False)
    ax.set(ylabel="Skew")
    ax.set(xlabel="Feature names")
    ax.set(title="Highly skewed Variables: After Transformation")
    sns.despine(trim=True, left=True)
```

```
    plt.xticks(rotation=45)
    plt.figure(figsize=(10,5))
    plt.show()
```

In [13]:
```
# Setup cross validation folds
kf = KFold(n_splits=12, random_state=42, shuffle=True)
```

In [14]:
```
# Define error metrics
def rmsle(y, y_pred):
    return np.sqrt(mean_squared_error(y, y_pred))

def cv_rmse(model, X=None):
    rmse = np.sqrt(-cross_val_score(model, x_train, y_train, scoring="neg_mea
    return (rmse)
```

In [ ]:
```
```