COMP 494 Final Project

Author: Daniel Matlock

Date: 4/28/2022

Housing Prices

Dataset: https://www.kaggle.com/competitions/house-prices-advanced-regression-techniques

The Ames Housing dataset was compiled by Dean De Cock for use in data science education. It's an incredible alternative for data scientists looking for a modernized and expanded version of the often cited Boston Housing dataset.

Final Project Requirements:

There are four sections of the final project. You are expected to perform the following tasks within each section to fulfill the project requirements.

- **Data Importing and Pre-processing (50 Points)**
  - Import dataset and describe characteristics such as dimensions, data types, file types, and import methods used
  - Clean, wrangle, and handle missing data
  - Transform data appropriately using techniques such as aggregation, normalization, and feature construction
  - Reduce redundant data and perform need based discretization
- **Data Analysis and Visualization (50 Points)**
  - Identify categorical, ordinal, and numerical variables within data
  - Provide measures of centrality and distribution with visualizations
  - Diagnose for correlations between variables and determine independent and dependent variables
  - Perform exploratory analysis in combination with visualization techniques to discover patterns and features of interest
- **Data Analytics (50 Points)**
  - Determine the need for a supervised or unsupervised learning method and identify dependent and independent variables
  - Train, test, and provide accuracy and evaluation metrics for model results
- **Presentation (50 Points)**
  - In a 5 to 10 minute presentation, briefly explain the project workflow from the code and results in your markdown notebook State your findings from the data and provide the interpretation of results from your analysis at each stage in the project

Table of Contents:

Data Importing and Pre-processing

In [1]:
```
#import libraries needed
import pandas as pd
pd.set_option('display.max_columns', None)
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
from scipy.stats import norm, skew, probplot
from scipy.special import boxcox1p
import warnings
warnings.filterwarnings('ignore')
%matplotlib inline
```

In [2]:
```
#read in file
housing_df = pd.read_csv('house_prices/train.csv')
```

In [3]:
```
#check number of rows and columns
housing_df.shape
```

Out[3]:
```
(1460, 81)
```

In [4]:
```
#count the number of categorical variables
cat_count = 0
for dtype in housing_df.dtypes:
    if dtype == 'object':
        cat_count = cat_count + 1
```

In [5]:
```
print('# of categorical variables:',cat_count)
print('# of contineous variables:',housing_df.shape[1] - cat_count - 1) #subt
```

```
# of categorical variables: 43
# of contineous variables: 37
```

In [6]:
```
housing_df.head()
```

Out[6]:

	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	

In [7]:
```
#check the column names
housing_df.columns
```

Out[7]:
```
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')
```

Handling missing data

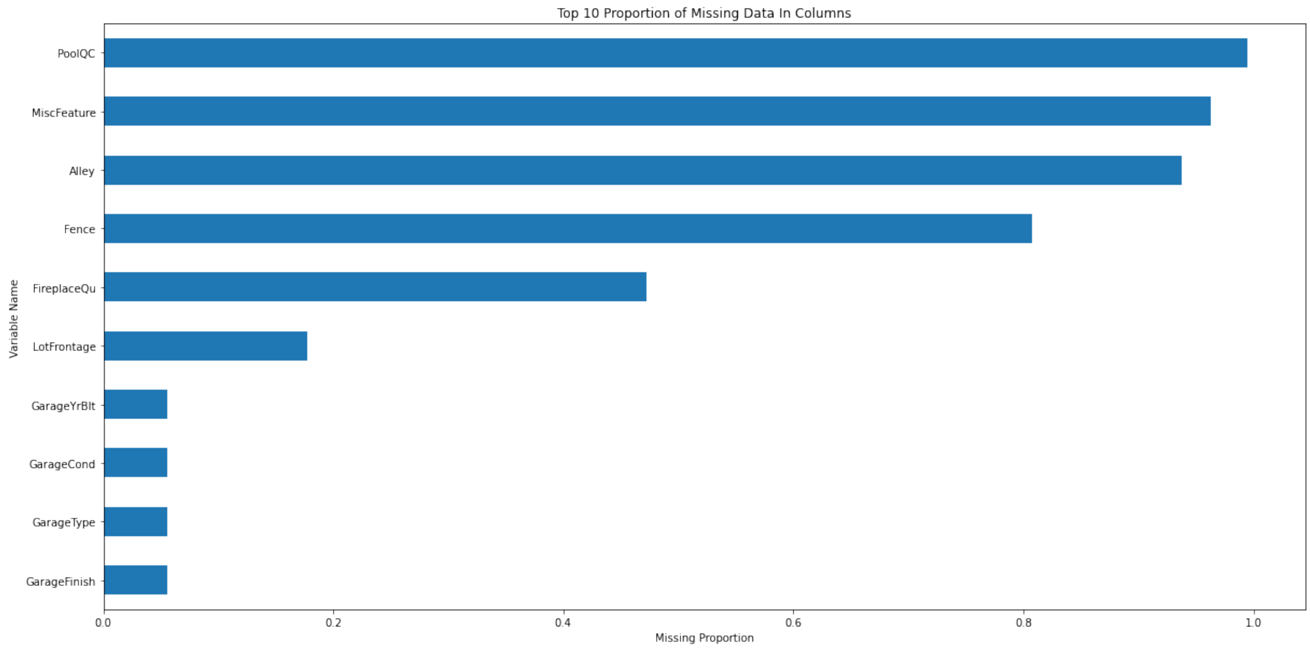
In [8]:
```
#missing data
total = housing_df.isnull().sum().sort_values(ascending=False)
percent = (housing_df.isnull().sum()/housing_df.isnull().count()).sort_values
missing_data = pd.concat([total, percent], axis=1, keys=['Total', 'Percent'])
missing_data.head(20)
```

Out[8]:

	Total	Percent
PoolQC	1453	0.995205
MiscFeature	1406	0.963014
Alley	1369	0.937671
Fence	1179	0.807534
FireplaceQu	690	0.472603
LotFrontage	259	0.177397
GarageYrBlt	81	0.055479
GarageCond	81	0.055479
GarageType	81	0.055479
GarageFinish	81	0.055479
GarageQual	81	0.055479
BsmtFinType2	38	0.026027
BsmtExposure	38	0.026027
BsmtQual	37	0.025342
BsmtCond	37	0.025342
BsmtFinType1	37	0.025342
MasVnrArea	8	0.005479
MasVnrType	8	0.005479
Electrical	1	0.000685
Id	0	0.000000

In [9]:

```
missing_data['Percent'].head(10).plot(kind='barh', figsize = (20,10)).invert_
plt.xlabel("Missing Proportion")
plt.ylabel("Variable Name")
plt.title("Top 10 Proportion of Missing Data In Columns")
plt.show()
```



In [10]:

```
#dealing with missing data
housing_df["PoolQC"] = housing_df["PoolQC"].fillna("None")
housing_df["MiscFeature"] = housing_df["MiscFeature"].fillna("None")
housing_df["Alley"] = housing_df["Alley"].fillna("None")
housing_df["Fence"] = housing_df["Fence"].fillna("None")
housing_df["FireplaceQu"] = housing_df["FireplaceQu"].fillna("None")
```

In [11]:

```
housing_df["LotFrontage"] = housing_df.groupby("Neighborhood")["LotFrontage"]
```

In [12]:

```
for col in ('GarageType', 'GarageFinish', 'GarageQual', 'GarageCond','BsmtQua
    housing_df[col] = housing_df[col].fillna('None')
```

In [13]:

```
for col in ('GarageYrBlt', 'GarageArea', 'GarageCars','BsmtFinSF1', 'BsmtFinS
    housing_df[col] = housing_df[col].fillna(0)
```

In [14]:

```
housing_df["MasVnrType"] = housing_df["MasVnrType"].fillna("None")
housing_df["MasVnrArea"] = housing_df["MasVnrArea"].fillna(0)
housing_df['MSZoning'] = housing_df['MSZoning'].fillna(housing_df['MSZoning']
housing_df = housing_df.drop(['Utilities'], axis=1)
housing_df["Functional"] = housing_df["Functional"].fillna("Typ")
housing_df['Electrical'] = housing_df['Electrical'].fillna(housing_df['Electr
housing_df['KitchenQual'] = housing_df['KitchenQual'].fillna(housing_df['Kitc
housing_df['Exterior1st'] = housing_df['Exterior1st'].fillna(housing_df['Exte
housing_df['Exterior2nd'] = housing_df['Exterior2nd'].fillna(housing_df['Exte
housing_df['SaleType'] = housing_df['SaleType'].fillna(housing_df['SaleType']
housing_df['MSSubClass'] = housing_df['MSSubClass'].fillna("None")
```

In [15]:
```
#Check remaining missing values if any
all_data_na = (housing_df.isnull().sum() / len(housing_df)) * 100
all_data_na = all_data_na.drop(all_data_na[all_data_na == 0].index).sort_valu
missing_data = pd.DataFrame({'Missing Ratio' :all_data_na})
missing_data.head()
```

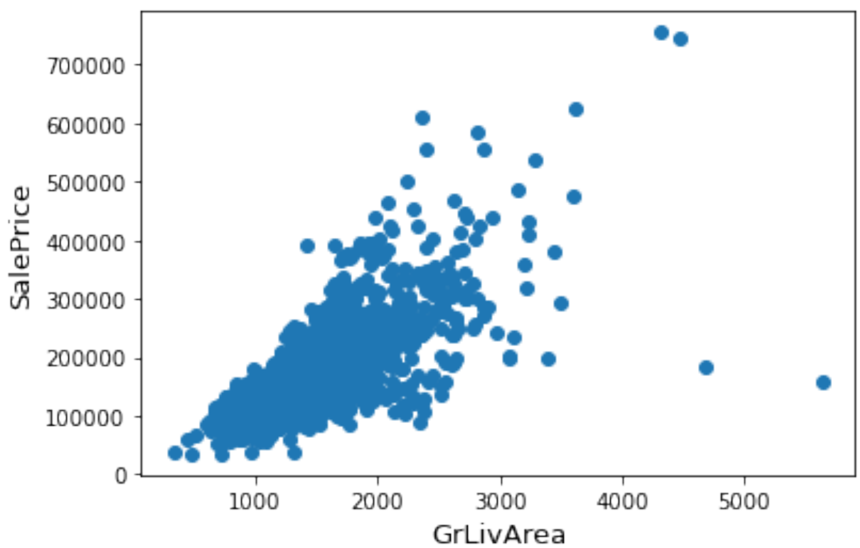
Out[15]:

Missing Ratio

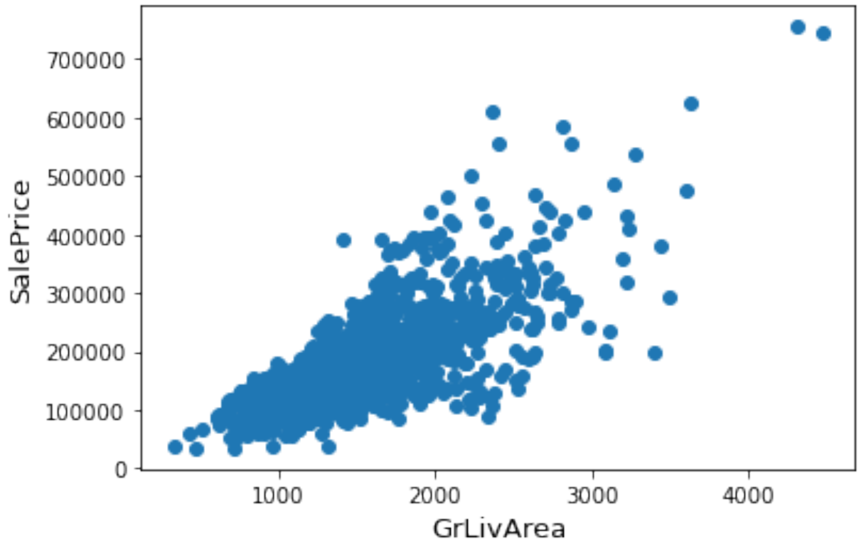
Handling Outliers

Target Variable

In [16]:
```
fig, ax = plt.subplots()
ax.scatter(x = housing_df['GrLivArea'], y = housing_df['SalePrice'])
plt.ylabel('SalePrice', fontsize=13)
plt.xlabel('GrLivArea', fontsize=13)
plt.show()
```



In [17]:
```
#Deleting outliers
housing_df = housing_df.drop(housing_df[(housing_df['GrLivArea']>4000) & (hou

#Check the graphic again
fig, ax = plt.subplots()
ax.scatter(housing_df['GrLivArea'], housing_df['SalePrice'])
plt.ylabel('SalePrice', fontsize=13)
plt.xlabel('GrLivArea', fontsize=13)
plt.show()
```

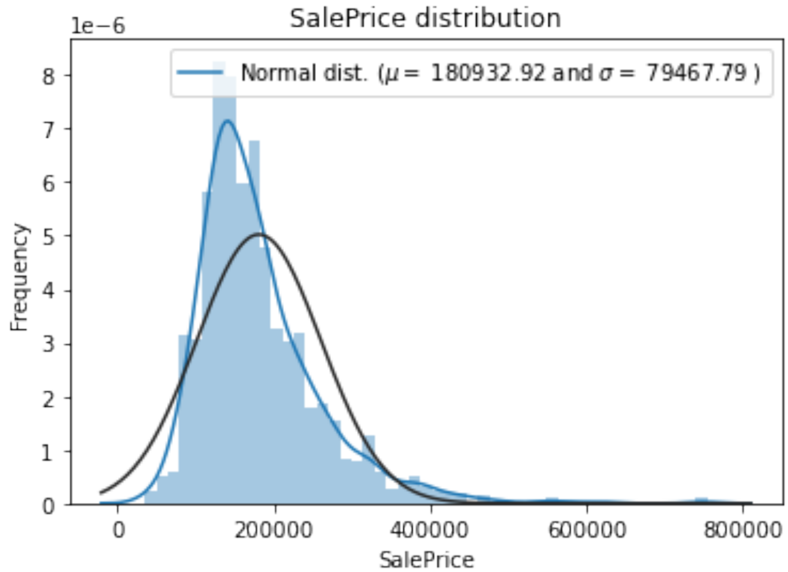


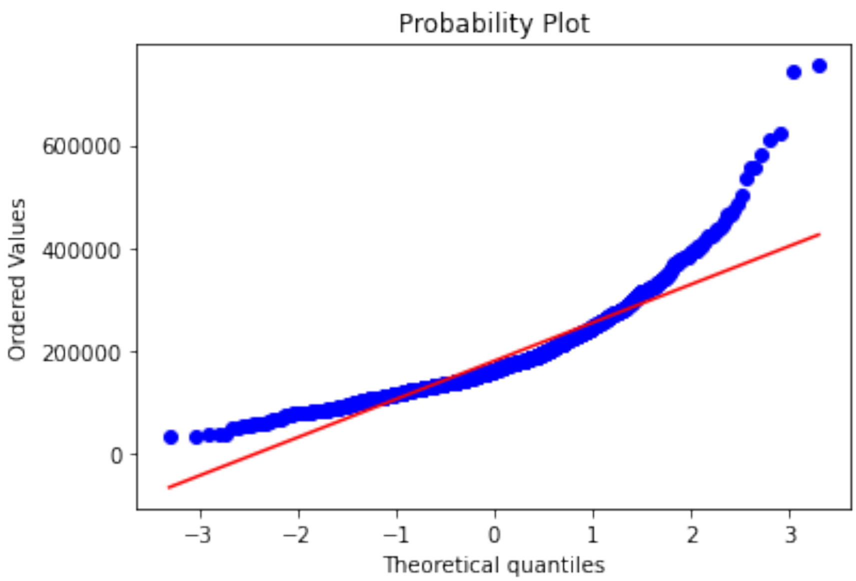
Normalize Target Variable

In [18]:

```
sns.distplot(housing_df['SalePrice'] , fit=norm);

# Get the fitted parameters used by the function
(mu, sigma) = norm.fit(housing_df['SalePrice'])
print( '\n mu = {:.2f} and sigma = {:.2f}\n'.format(mu, sigma))

#Now plot the distribution
plt.legend(['Normal dist. ($\mu=$ {:.2f} and $\sigma=$ {:.2f} )'.format(mu, s
plt.ylabel('Frequency')
plt.title('SalePrice distribution')

#Get also the QQ-plot
fig = plt.figure()
res = probplot(housing_df['SalePrice'], plot=plt)
plt.show()
```

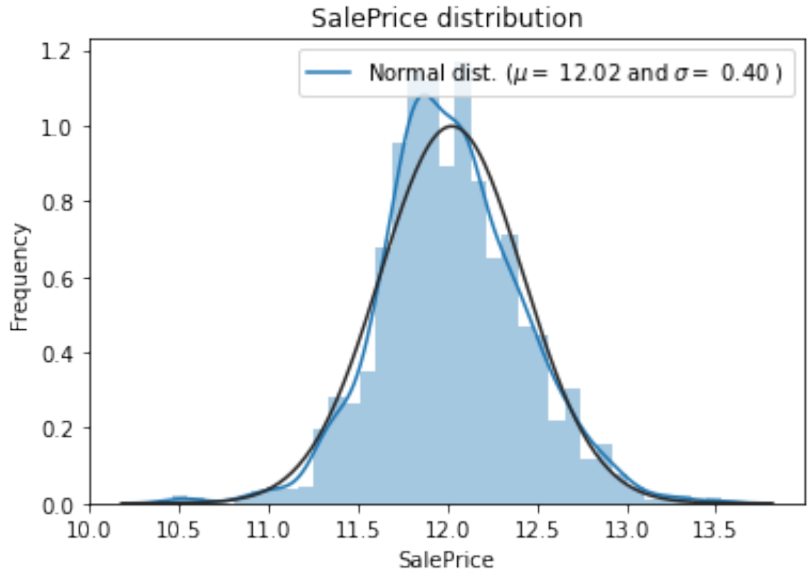
```
mu = 180932.92 and sigma = 79467.79
```

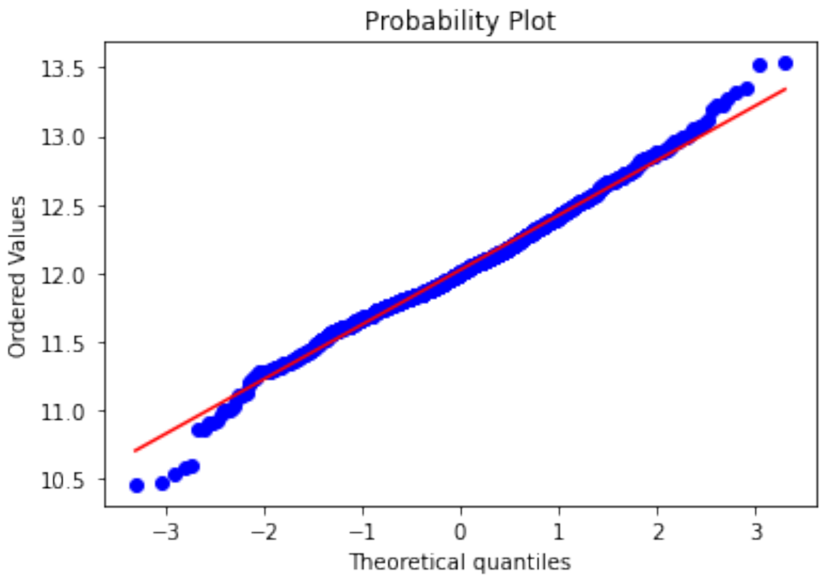




In [19]:
```
#We use the numpy fuction log1p which  applies log(1+x) to all elements of th
housing_df["SalePrice"] = np.log1p(housing_df["SalePrice"])

#Check the new distribution
sns.distplot(housing_df['SalePrice'] , fit=norm);

# Get the fitted parameters used by the function
(mu, sigma) = norm.fit(housing_df['SalePrice'])
print( '\n mu = {:.2f} and sigma = {:.2f}\n'.format(mu, sigma))

#Now plot the distribution
plt.legend(['Normal dist. ($\mu=$ {:.2f} and $\sigma=$ {:.2f} )'.format(mu, s
            loc='best')
plt.ylabel('Frequency')
plt.title('SalePrice distribution')

#Get also the QQ-plot
fig = plt.figure()
res = probplot(housing_df['SalePrice'], plot=plt)
plt.show()
```

```
mu = 12.02 and sigma = 0.40
```





Data Analysis and Visualization

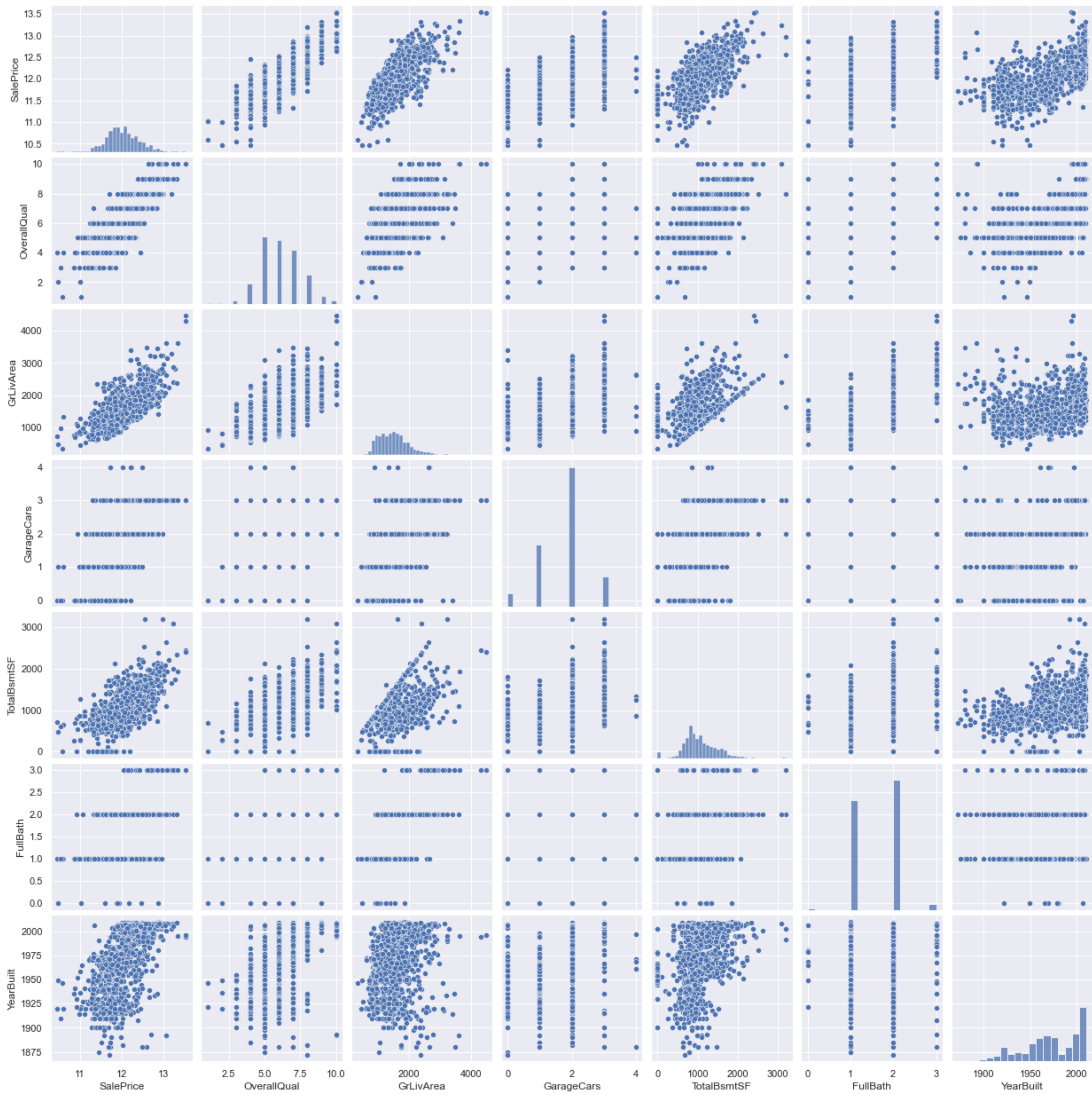
In [20]:

```
from sklearn.preprocessing import LabelEncoder
```

Target Variable Scatterplots

In [21]:

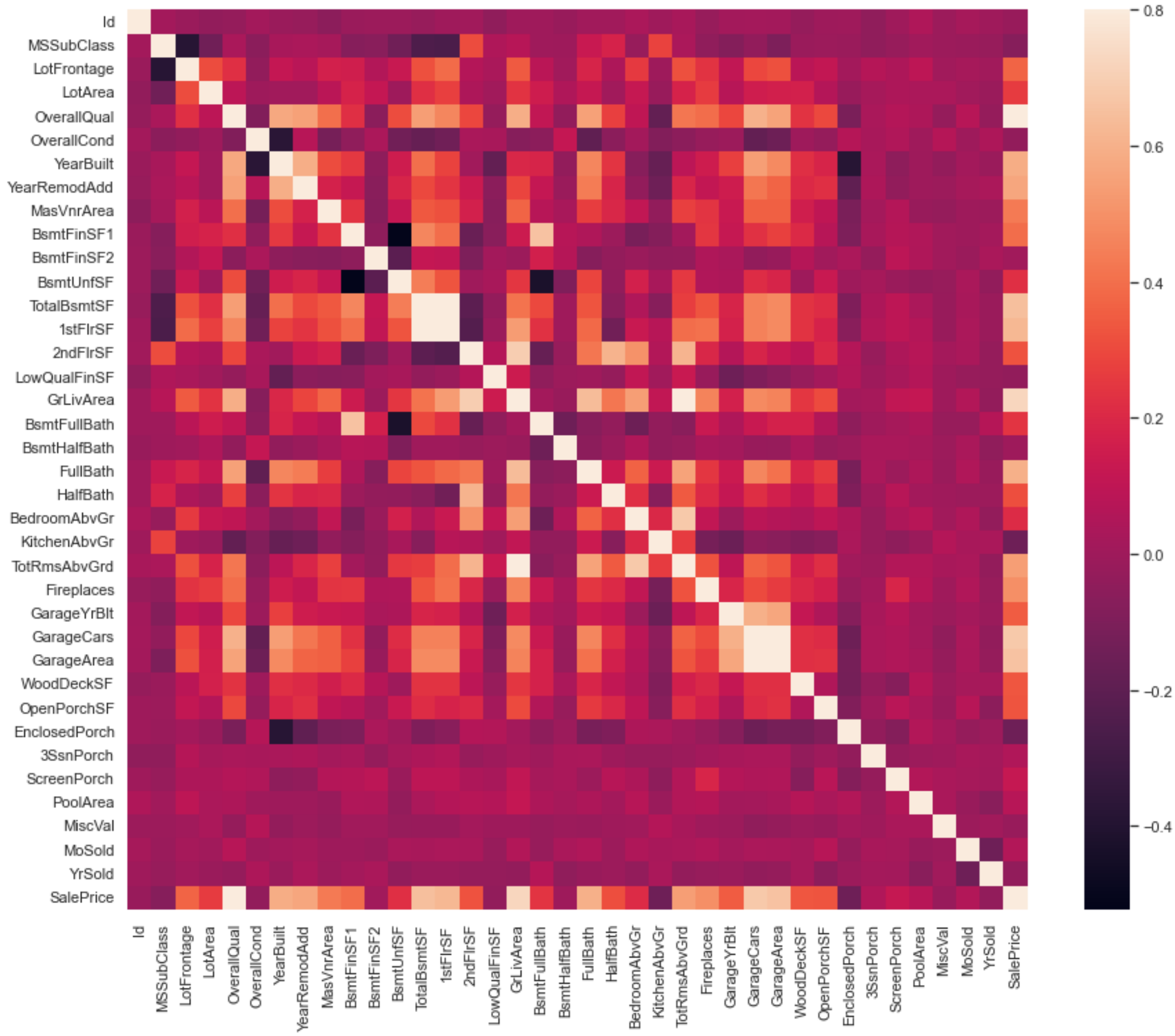
```
#scatterplot
sns.set()
cols = ['SalePrice', 'OverallQual', 'GrLivArea', 'GarageCars', 'TotalBsmtSF',
sns.pairplot(housing_df[cols], size = 2.5)
plt.show();
```



Correlation Matrix

In [22]:

```
#Correlation map to see how features are correlated with SalePrice
corrmat = housing_df.corr()
f, ax = plt.subplots(figsize=(15, 12))
sns.heatmap(corrmat, vmax=.8, square=True);
```



In [23]:

```
#MSSubClass=The building class
housing_df['MSSubClass'] = housing_df['MSSubClass'].apply(str)


#Changing OverallCond into a categorical variable
housing_df['OverallCond'] = housing_df['OverallCond'].astype(str)


#Year and month sold are transformed into categorical features.
housing_df['YrSold'] = housing_df['YrSold'].astype(str)
housing_df['MoSold'] = housing_df['MoSold'].astype(str)

# Adding total sqfootage feature
housing_df['TotalSF'] = housing_df['TotalBsmtSF'] + housing_df['1stFlrSF'] + 
```

Label encode categorical variables

In [24]:
```
cols = ('FireplaceQu', 'BsmtQual', 'BsmtCond', 'GarageQual', 'GarageCond',
        'ExterQual', 'ExterCond','HeatingQC', 'PoolQC', 'KitchenQual', 'BsmtF
        'BsmtFinType2', 'Functional', 'Fence', 'BsmtExposure', 'GarageFinish'
        'LotShape', 'PavedDrive', 'Street', 'Alley', 'CentralAir', 'MSSubClas
        'YrSold', 'MoSold')

# process columns, apply LabelEncoder to categorical features
for c in cols:
    lbl = LabelEncoder()
    lbl.fit(list(housing_df[c].values))
    housing_df[c] = lbl.transform(list(housing_df[c].values))

# shape
print('Shape housing_df: {}'.format(housing_df.shape))
```

```
Shape housing_df: (1458, 81)
```

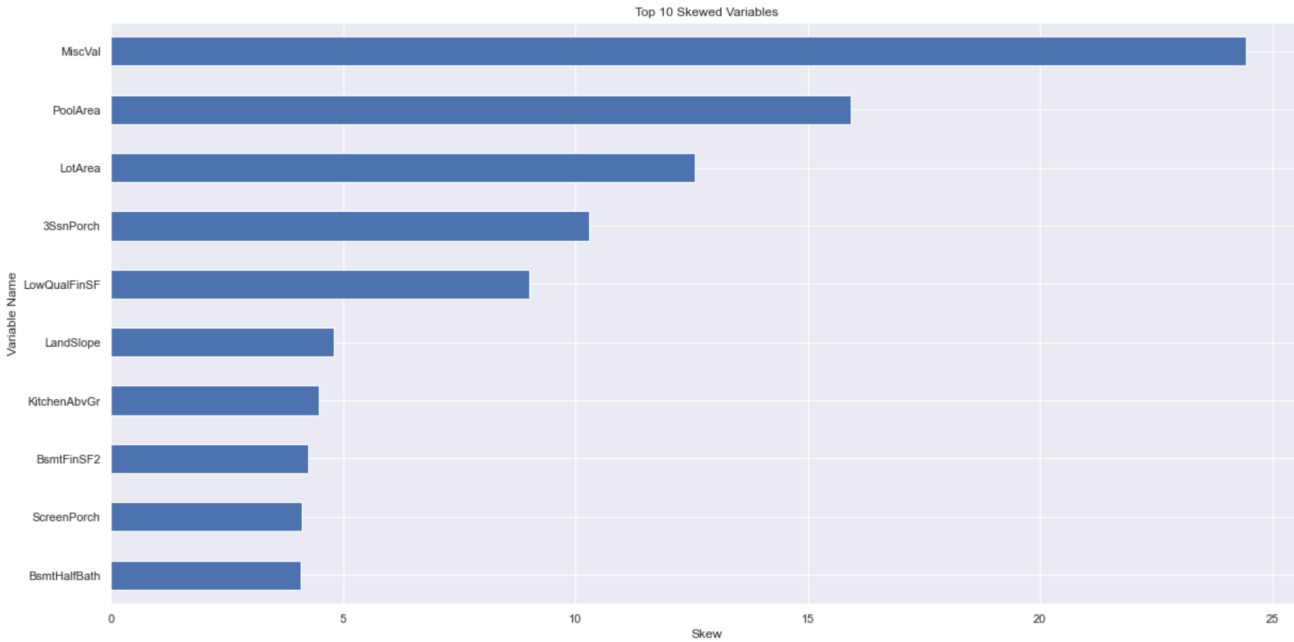
In [25]:
```
numeric_feats = housing_df.dtypes[housing_df.dtypes != "object"].index

# Check the skew of all numerical features
skewed_feats = housing_df[numeric_feats].apply(lambda x: skew(x.dropna())).so
print("\nSkew in numerical features: \n")
skewness = pd.DataFrame({'Skew' :skewed_feats})
skewness.head(10)
```

```
Skew in numerical features:
```

Out[25]:

	Skew
MiscVal	24.434913
PoolArea	15.932532
LotArea	12.560986
3SsnPorch	10.286510
LowQualFinSF	8.995688
LandSlope	4.805032
KitchenAbvGr	4.480268
BsmtFinSF2	4.247550
ScreenPorch	4.114690
BsmtHalfBath	4.095895

In [26]:
```
skewness['Skew'].head(10).plot(kind='barh', figsize = (20,10)).invert_yaxis()
plt.xlabel("Skew")
plt.ylabel("Variable Name")
plt.title("Top 10 Skewed Variables")
plt.show()
```



In [27]:
```
skewness = skewness[abs(skewness) > 0.75]
print("There are {} skewed numerical features to Box Cox transform (normalize

skewed_features = skewness.index
lam = 0.15
for feat in skewed_features:
    #all_data[feat] += 1
    housing_df[feat] = boxcox1p(housing_df[feat], lam)

#all_data[skewed_features] = np.log1p(all_data[skewed_features])
```

```
There are 61 skewed numerical features to Box Cox transform (normalize)
```

In [28]:
```
housing_df.head()
```

Out[28]:

	Id	MSSubClass	MSZoning	LotFrontage	LotArea	Street	Alley	LotShape
0	0.730463	2.750250	RL	5.831328	19.212182	0.730463	0.730463	1.540963
1	1.194318	1.820334	RL	6.221214	19.712205	0.730463	0.730463	1.540963
2	1.540963	2.750250	RL	5.914940	20.347241	0.730463	0.730463	0.000000
3	1.820334	2.885846	RL	5.684507	19.691553	0.730463	0.730463	0.000000
4	2.055642	2.750250	RL	6.314735	21.325160	0.730463	0.730463	0.000000

In [29]:

```
housing_df = pd.get_dummies(housing_df)
housing_df.head()
```

Out[29]:

	Id	MSSubClass	LotFrontage	LotArea	Street	Alley	LotShape	LandSlope
0	0.730463	2.750250	5.831328	19.212182	0.730463	0.730463	1.540963	0.0
1	1.194318	1.820334	6.221214	19.712205	0.730463	0.730463	1.540963	0.0
2	1.540963	2.750250	5.914940	20.347241	0.730463	0.730463	0.000000	0.0
3	1.820334	2.885846	5.684507	19.691553	0.730463	0.730463	0.000000	0.0
4	2.055642	2.750250	6.314735	21.325160	0.730463	0.730463	0.000000	0.0

Data Analytics

In [30]:

```
from sklearn.linear_model import Lasso
from sklearn.ensemble import RandomForestRegressor
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import KFold, cross_val_score
from sklearn.metrics import mean_squared_error
from sklearn.tree import DecisionTreeRegressor
from sklearn.neighbors import KNeighborsRegressor
import xgboost as xgb
import lightgbm as lgb
```

In [31]:

```
train_df = housing_df[housing_df.columns.difference(['Id', 'SalePrice'])]
```

In [32]:
```
#Validation function
n_folds = 5

def rmse_cv(model,n_folds):
    kf=KFold(n_splits=n_folds)
    rmse = np.sqrt(-cross_val_score(model, train_df, housing_df.SalePrice, sc
    return rmse
```

In [33]:
```
lr_w_int = LinearRegression()
lr_no_int = LinearRegression(fit_intercept=False)
```

In [34]:
```
neigh = KNeighborsRegressor(n_neighbors=10)
```

In [35]:
```
rf = RandomForestRegressor(n_estimators=100)
```

In [36]:
```
dt = DecisionTreeRegressor(max_depth = 10)
```

In [37]:
```
model_xgb = xgb.XGBRegressor(max_depth=5, n_estimators=1000, learning_rate=0.
```

In [38]:
```
model_lgb = lgb.LGBMRegressor(learning_rate=0.01, max_depth=5, n_estimators=1
```

Algotithm Results on a 5 Fold Cross Validation

In [39]:
```
score_linear = rmse_cv(lr_w_int,n_folds)
print("Linear Regression (w/ Intercept) score: {:.4f} ({:.4f})\n".format(scor
```

```
Linear Regression (w/ Intercept) score: 16839339.5654 (14059441.9090)
```

Linear regression does not generalize well. Removing the intercept adds something called regularization that generalizes better.

In [40]:
```
score_linear_no_int = rmse_cv(lr_no_int,n_folds)
print("Linear Regression (No Intercept) score: {:.4f} ({:.4f})\n".format(scor
```

```
Linear Regression (No Intercept) score: 0.0145 (0.0011)
```

In [41]:
```
score_neigh = rmse_cv(neigh,n_folds)
print("Nearest Neighbor (13) score: {:.4f} ({:.4f})\n".format(score_neigh.mea
```

```
Nearest Neighbor (13) score: 0.0269 (0.0016)
```

In [42]:
```
score_dt = rmse_cv(dt,n_folds)
print("Decision Tree Regression score: {:.4f} ({:.4f})\n".format(score_dt.mea
```

```
Decision Tree Regression score: 0.0234 (0.0011)
```

In [43]:
```
score_rf = rmse_cv(rf,n_folds)
print("Random Forest Regression score: {:.4f} ({:.4f})\n".format(score_rf.mea
```

```
Random Forest Regression score: 0.0157 (0.0003)
```

In [44]:
```
score_xg = rmse_cv(model_xgb,n_folds)
print("Xgboost score: {:.4f} ({:.4f})\n".format(score_xg.mean(), score_xg.std
```

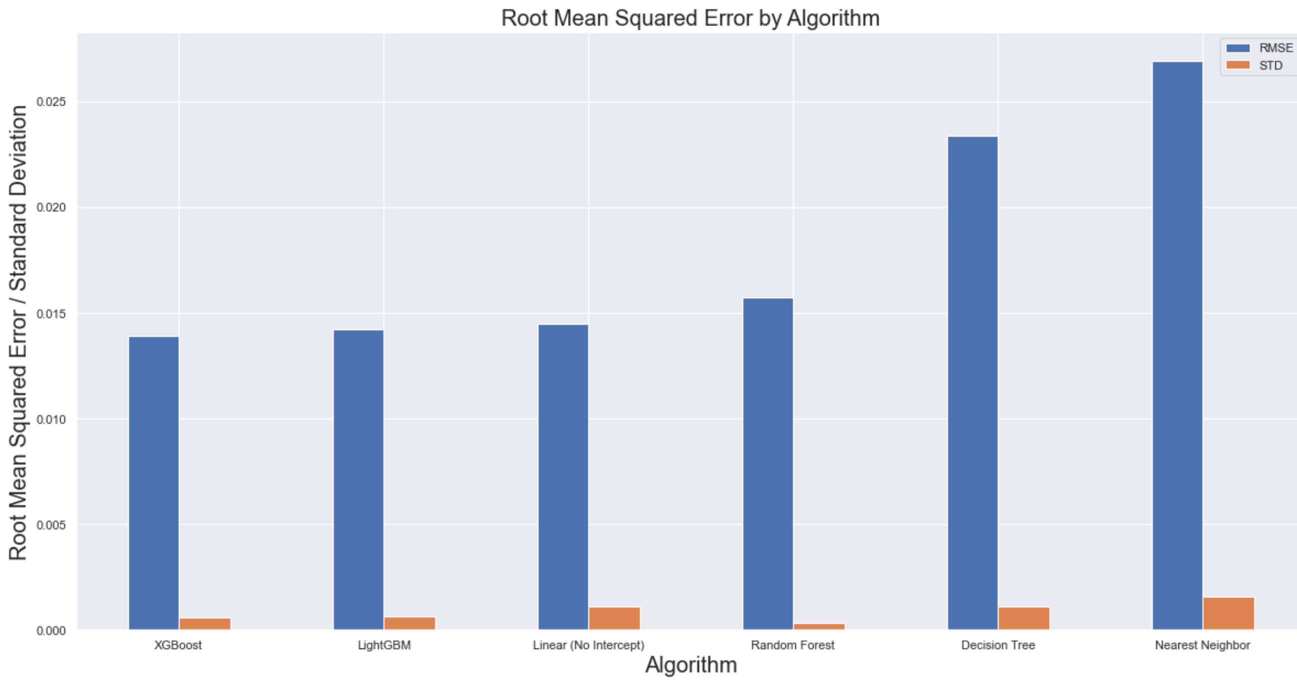
```
Xgboost score: 0.0139 (0.0006)
```

In [45]:
```
score_lgbm = rmse_cv(model_lgb,n_folds)
print("LGBM score: {:.4f} ({:.4f})\n" .format(score_lgbm.mean(), score_lgbm.s
```

```
LGBM score: 0.0142 (0.0006)
```

In [46]:
```
#plot RMSE and STD for each Algorithm
data = {'Linear (No Intercept)':[score_linear_no_int.mean(),score_linear_no_i
        , 'LightGBM': [score_lgbm.mean(),score_lgbm.std()], 'Decision Tree':
data_df = pd.DataFrame(data=data).T.reset_index().sort_values(by = [0],ascend
data_df.columns = ['Algorithm','RMSE','STD']
```

In [47]:
```
# creating the bar plot
data_df.plot(kind='bar',x = 'Algorithm', y = ['RMSE', 'STD'], figsize = (20,1
plt.xlabel("Algorithm",fontsize=20)
plt.ylabel("Root Mean Squared Error / Standard Deviation",fontsize=20)
plt.title("Root Mean Squared Error by Algorithm",fontsize=20)
plt.show()
```



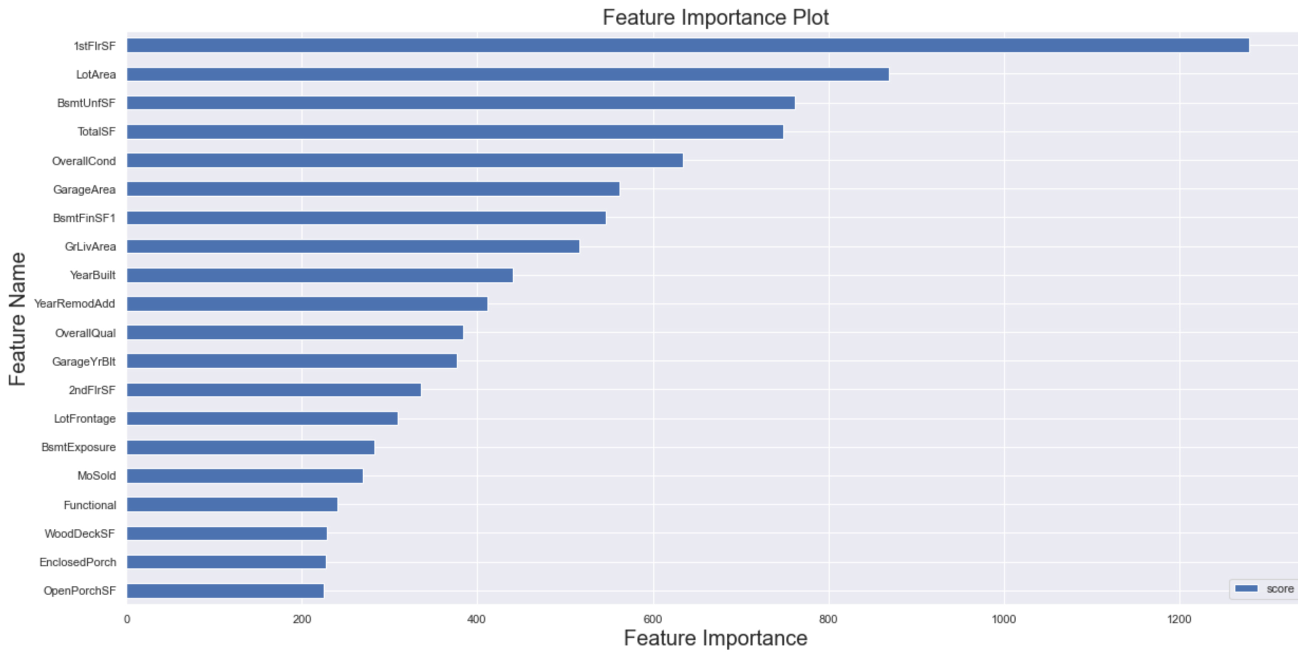
We see that the GBM algorithms (XGBoost and LightGBM) tend to slightly perform the best.

Variable Importance Plot

Only applies to tree based models (Decision Trees, Random Forest, GBMs)

In [48]:

```
model = model_xgb.fit(train_df, housing_df.SalePrice) #fit model on entire da
feature_important = model.get_booster().get_score(importance_type='weight')

keys = list(feature_important.keys())
values = list(feature_important.values())

data = pd.DataFrame(data=values, index=keys, columns=["score"]).sort_values(b
data[:20].plot(kind='barh', figsize = (20,10)).invert_yaxis(); ## plot top 20
plt.xlabel("Feature Importance",fontsize=20)
plt.ylabel("Feature Name",fontsize=20)
plt.title("Feature Importance Plot",fontsize=20)
plt.show()
```


Feature Importance Plot
1stFlrSF
LotArea
BsmtUnfSF
TotalSF
OverallCond
GarageArea
BsmtFinSF1
GrLivArea
YearBuilt
YearRemodAdd
OverallQual
GarageYrBlt
2ndFlrSF
LotFrontage
BsmtExposure
MoSold
Functional
WoodDeckSF
EnclosedPorch
OpenPorchSF
Feature Name
score
0
200
400
600
800
1000
1200
Feature Importance

In [ ]: