In [31]:
```
import numpy as np
import pandas as pd
from glob import glob
import seaborn as sns
import matplotlib.pyplot as plt
import warnings
warnings.filterwarnings("ignore")
from sklearn.ensemble import RandomForestRegressor
from sklearn.pipeline import Pipeline
from sklearn.metrics import r2_score
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import accuracy_score, confusion_matrix, roc_curve, roc_auc_score
from sklearn.decomposition import PCA
from sklearn.model_selection import cross_val_score
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier
from sklearn.svm import SVC
from sklearn import metrics
```

In [4]:
```
df=pd.read_csv('C:/Users/mmi/Downloads/house/train.csv')
df.head(5)
df.isnull().sum()
```

Out[4]:
```
Id                0
MSSubClass        0
MSZoning          0
LotFrontage     259
LotArea           0
                 ...
MoSold            0
YrSold            0
SaleType          0
SaleCondition     0
SalePrice         0
Length: 81, dtype: int64
```

In [5]:
```
df.fillna(df.mean(),inplace=True)
```

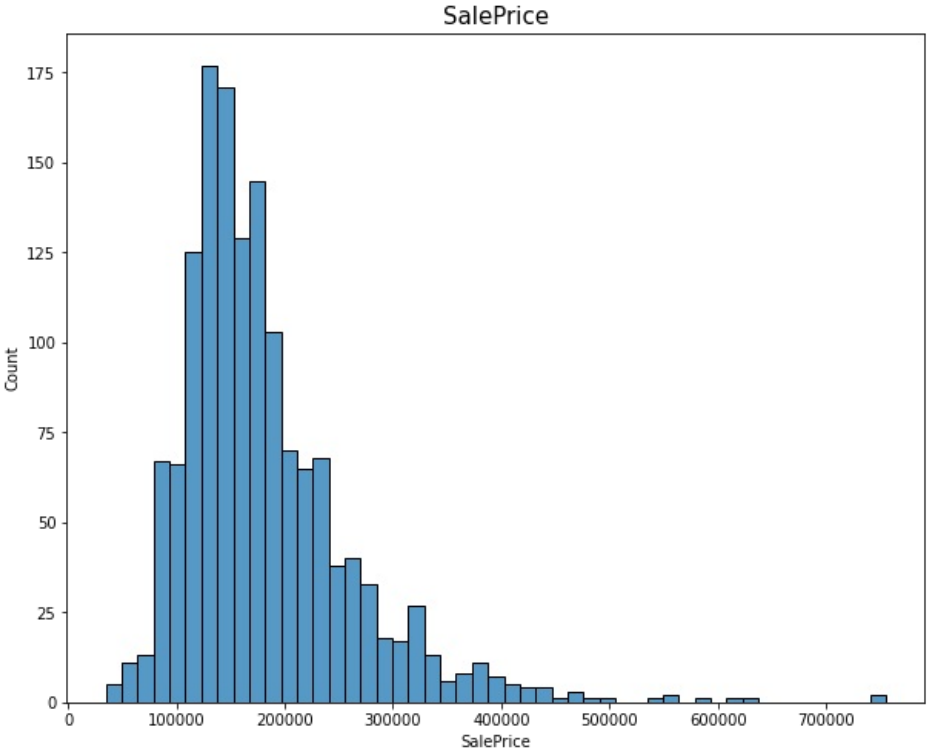
In [6]:
```
df.isnull().sum()
```

Out[6]:
```
Id               0
MSSubClass       0
MSZoning         0
LotFrontage      0
LotArea          0
                ..
MoSold           0
YrSold           0
SaleType         0
SaleCondition    0
SalePrice        0
Length: 81, dtype: int64
```

In [5]:
```
print(df.columns.values)
```

```
['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']
```

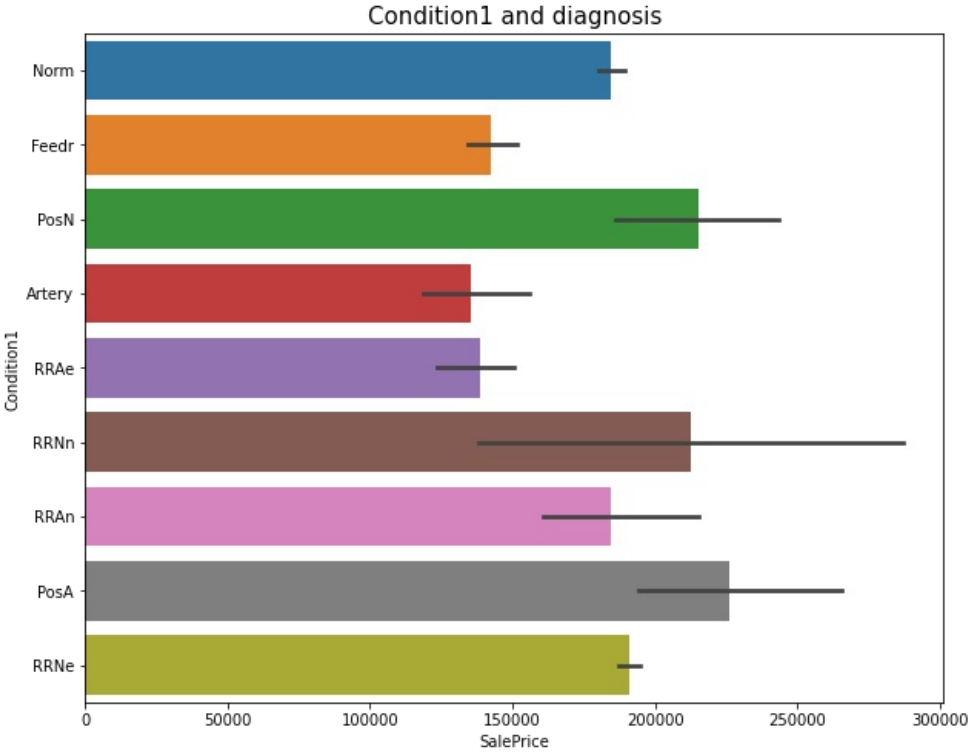
In [7]:
```
plt.figure(figsize=(10,8))
sns.histplot(df["SalePrice"])
plt.title("SalePrice", size=15)
plt.show()
```



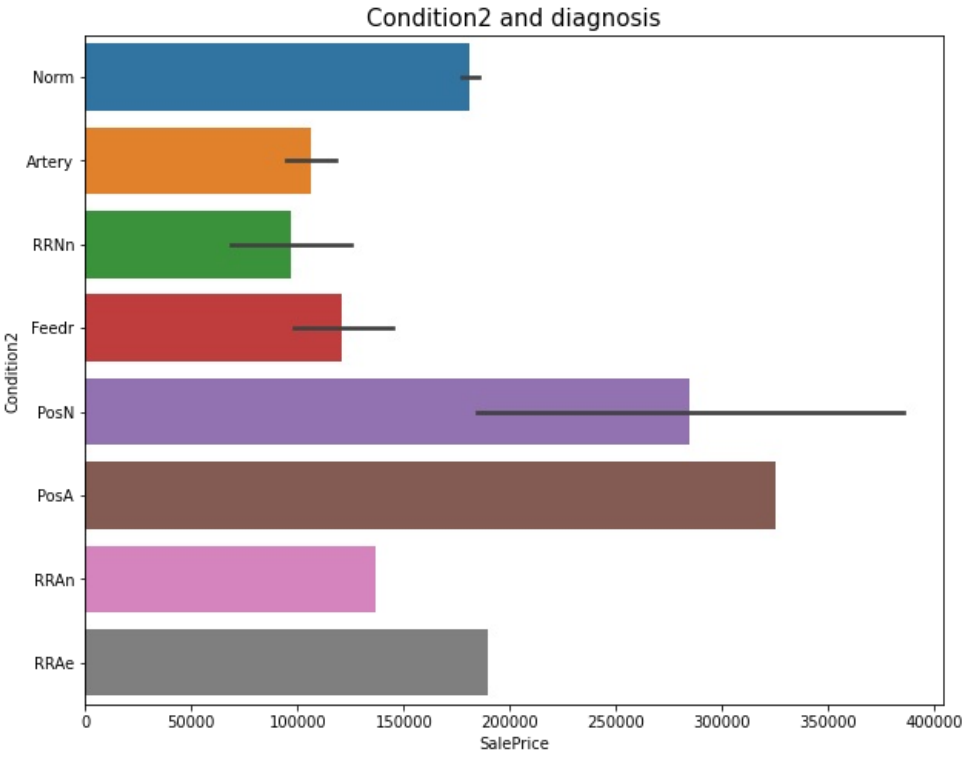
In [8]:
```
print("Skewness: %f" % df['SalePrice'].skew())
print("Kurtosis: %f" % df['SalePrice'].kurt())
```

```
Skewness: 1.882876
Kurtosis: 6.536282
```

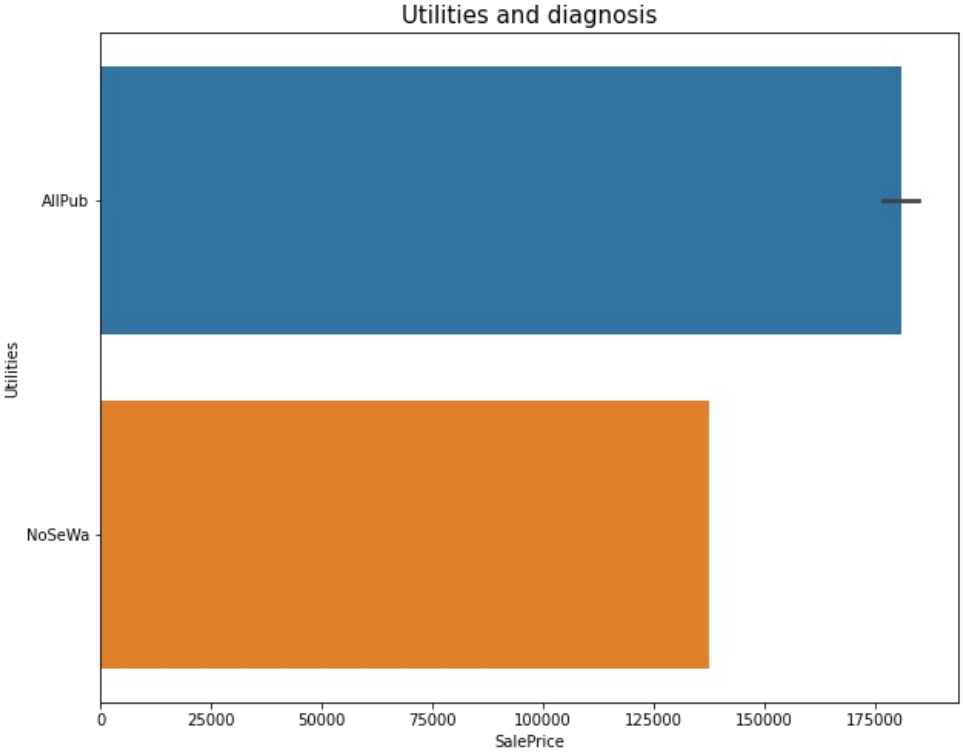
In [7]:
```
plt.figure(figsize=(10,8))
sns.barplot(x=df["SalePrice"], y=df['Condition1'])
plt.title(f"{'Condition1'} and diagnosis", size=15)
plt.show()
```



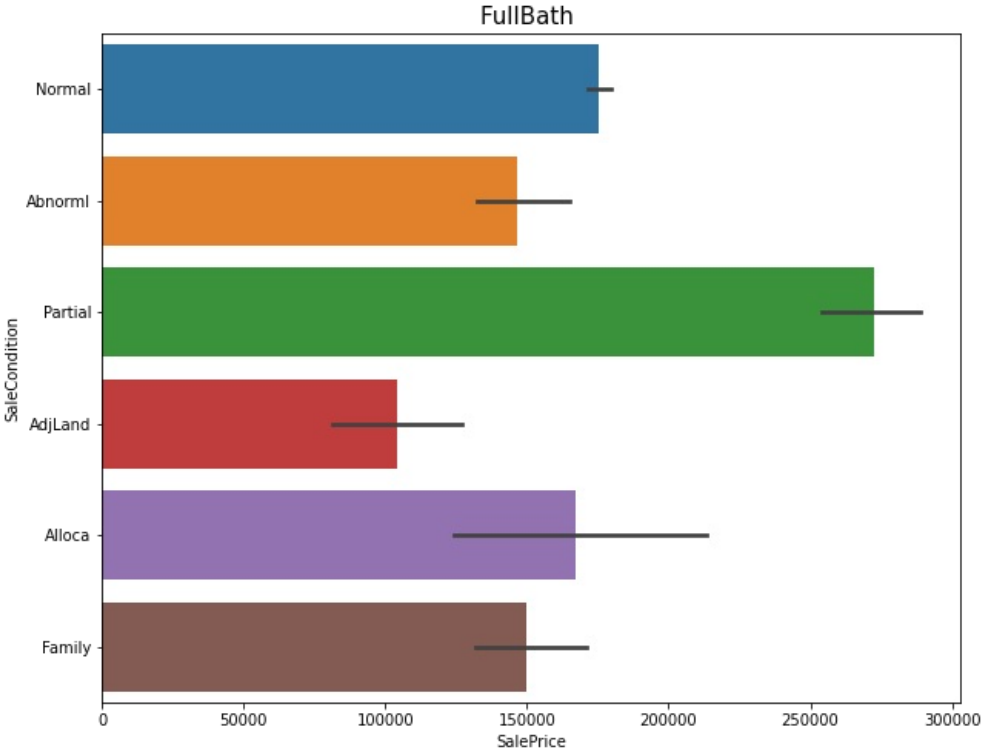
In [8]:
```
plt.figure(figsize=(10,8))
sns.barplot(x=df["SalePrice"], y=df['Condition2'])
plt.title(f"{'Condition2'} and diagnosis", size=15)
plt.show()
```



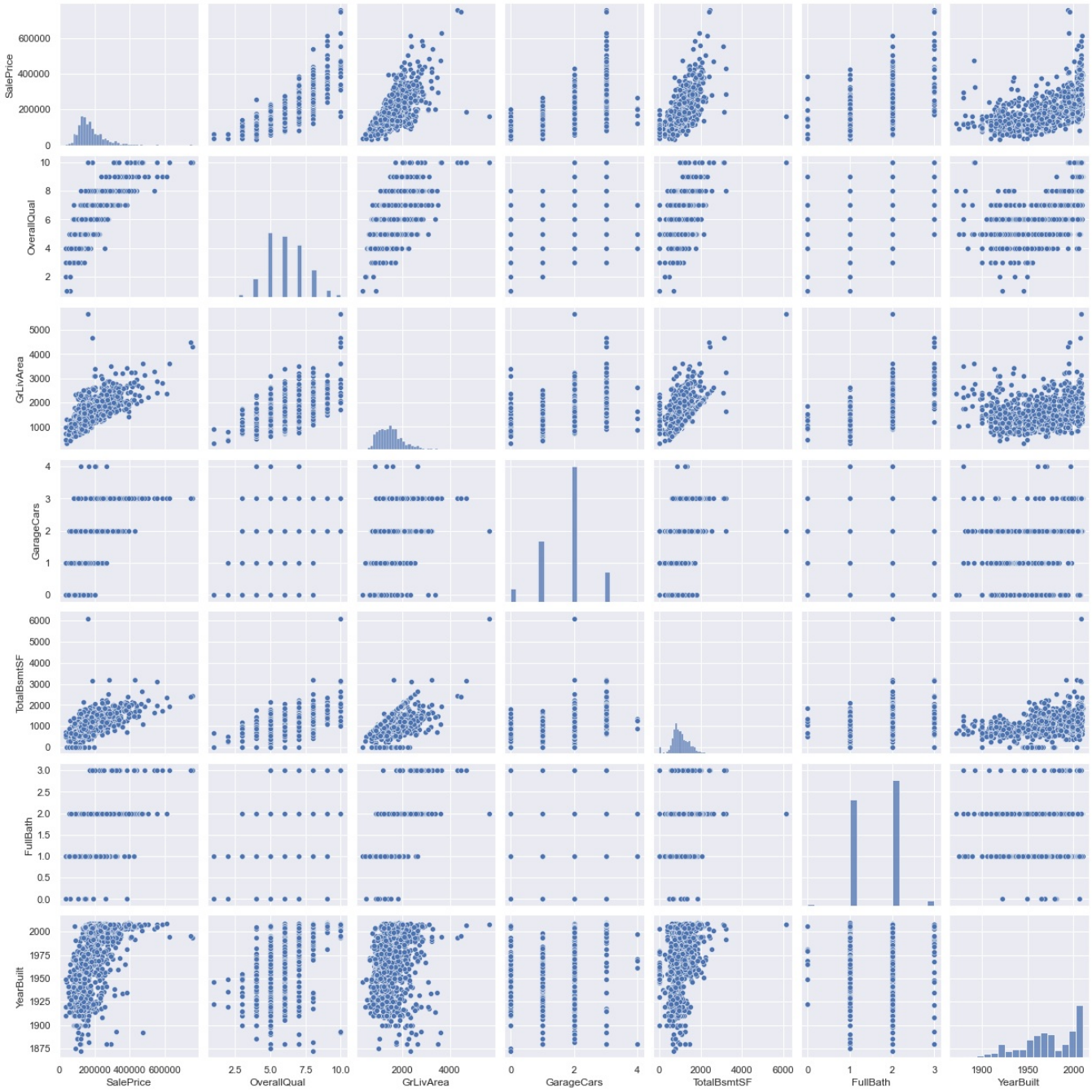
In [9]:
```
plt.figure(figsize=(10,8))
sns.barplot(x=df["SalePrice"], y=df['Utilities'])
plt.title(f"{'Utilities'} and diagnosis", size=15)
plt.show()
```



In [10]:
```
plt.figure(figsize=(10,8))
sns.barplot(x=df["SalePrice"], y=df['SaleCondition'])
plt.title(f"{'FullBath' } ", size=15)
plt.show()
```

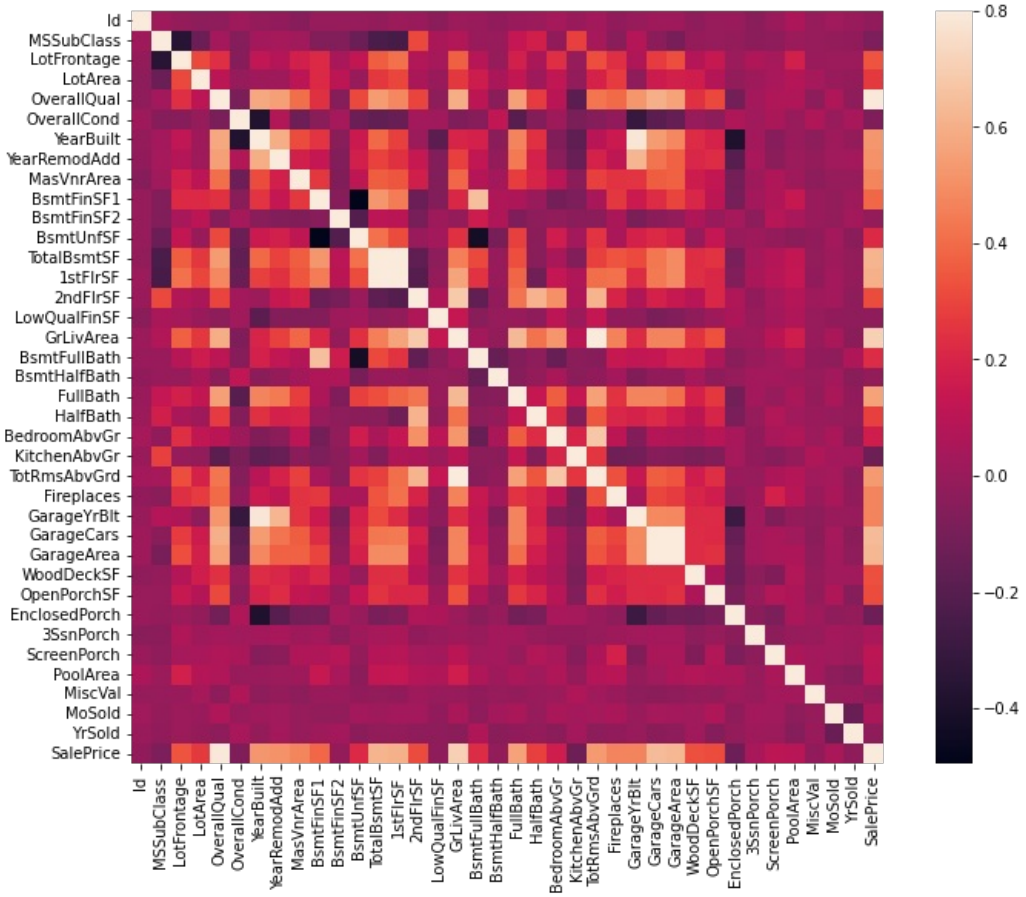


In [11]:
```
sns.set()
cols = ['SalePrice', 'OverallQual', 'GrLivArea', 'GarageCars', 'TotalBsmtSF', 'FullBath', 'YearBuilt']
sns.pairplot(df[cols], size = 2.5)
plt.show();
```

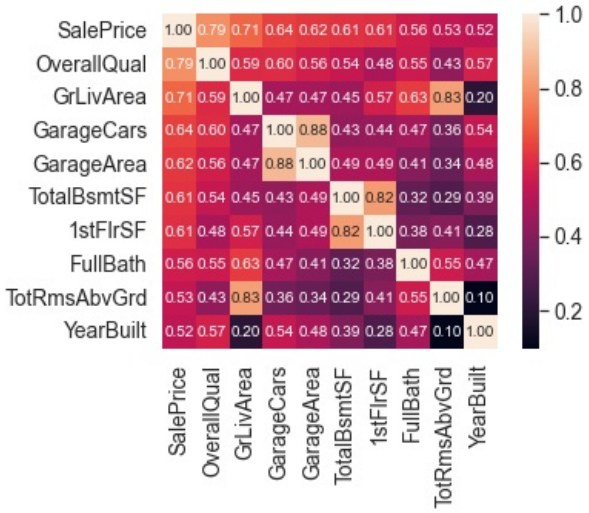


In [11]:
```
plt.figure(figsize=(14,10))
corrmat = df.corr()
f, ax = plt.subplots(figsize=(12, 9))
sns.heatmap(corrmat, vmax=.8, square=True);
```

```
<Figure size 1008x720 with 0 Axes>
```



In [13]:
```
k = 10 #number of variables for heatmap
cols = corrmat.nlargest(k, 'SalePrice')['SalePrice'].index
cm = np.corrcoef(df[cols].values.T)
sns.set(font_scale=1.25)
hm = sns.heatmap(cm, cbar=True, annot=True, square=True, fmt='.2f', annot_kws={'size': 10}, yticklabels=cols.va
plt.show()
```



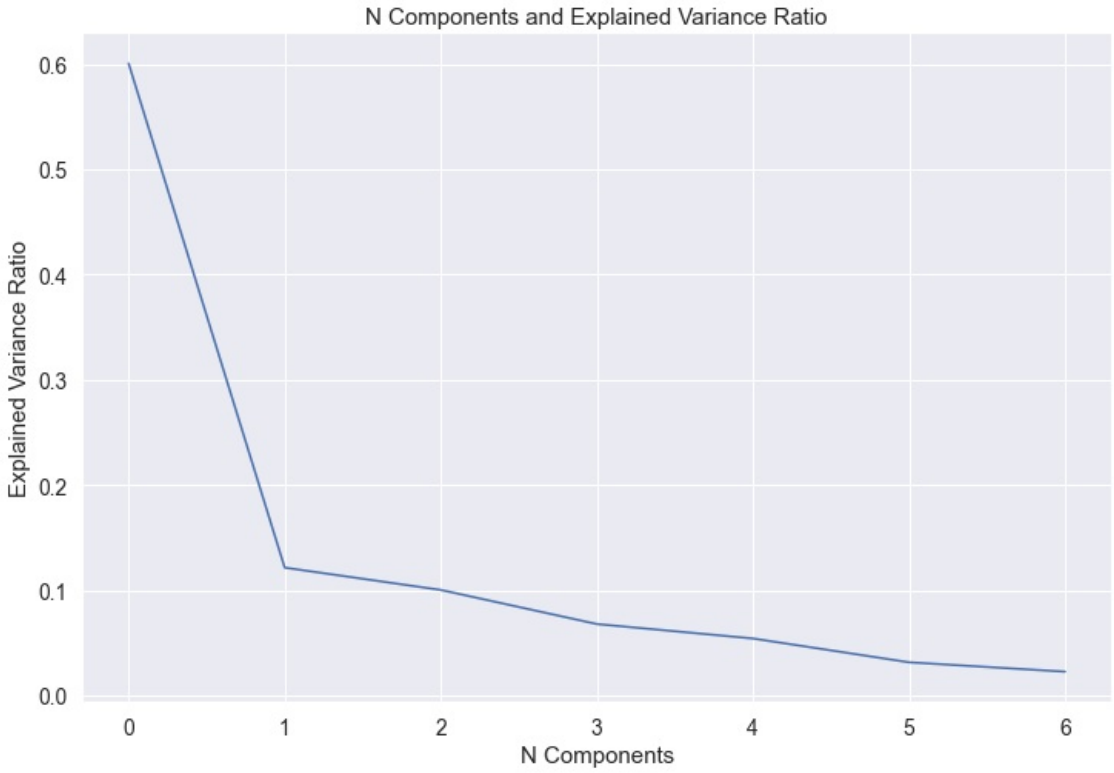
In [17]:
```
def cross_val(model):
    pred = cross_val_score(model, X, y, cv=10)
    return pred.mean()

def print_evaluate(true, predicted):
    mae = metrics.mean_absolute_error(true, predicted)
    mse = metrics.mean_squared_error(true, predicted)
    rmse = np.sqrt(metrics.mean_squared_error(true, predicted))
    r2_square = metrics.r2_score(true, predicted)
    print('MAE:', mae)
    print('MSE:', mse)
    print('RMSE:', rmse)
    print('R2 Square', r2_square)
    print('____________________________________')

def evaluate(true, predicted):
    mae = metrics.mean_absolute_error(true, predicted)
    mse = metrics.mean_squared_error(true, predicted)
    rmse = np.sqrt(metrics.mean_squared_error(true, predicted))
    r2_square = metrics.r2_score(true, predicted)
    return mae, mse, rmse, r2_square
```

In [18]:
```
X = df.drop('SalePrice', axis=1)
y = df['SalePrice']
X=df[['SalePrice', 'OverallQual', 'GrLivArea', 'GarageCars', 'TotalBsmtSF', 'FullBath', 'YearBuilt']]
```

In [19]:
```
scaler = StandardScaler()
X = scaler.fit_transform(X)
pca = PCA()
pca.fit(X)
plt.figure(figsize=(12,8))
plt.plot(pca.explained_variance_ratio_)
plt.title("N Components and Explained Variance Ratio", size=15)
plt.xlabel("N Components")
plt.ylabel("Explained Variance Ratio")
plt.show()
pca = PCA(n_components =5)
X = pca.fit_transform(X)
pca.explained_variance_ratio_.sum()
```



Out[19]:
```
0.9453443534658826
```

In [36]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.4, random_state=4)
models = pd.DataFrame(columns=["Model", "Accuracy Score"])
```

In [21]:
```
rfc = RandomForestRegressor()
rfc.fit(X_train, y_train)
predictions = rfc.predict(X_test)
score = r2_score(y_test, predictions)
print("Accuracy Score:", score)
new_row = {"Model": "RandomForestClassifier", "Accuracy Score": score}
models = models.append(new_row, ignore_index=True)
```

```
Accuracy Score: 0.9407853913906038
```

In [26]:
```
lin_reg = LinearRegression(normalize=True)
lin_reg.fit(X_train,y_train)
print(lin_reg.intercept_)
pred = lin_reg.predict(X_test)
```

```
180847.65027514848
```

In [32]:
```
test_pred = lin_reg.predict(X_test)
train_pred = lin_reg.predict(X_train)

print('Test set evaluation:\n_____________________________________')
print_evaluate(y_test, test_pred)
print('Train set evaluation:\n_____________________________________')
print_evaluate(y_train, train_pred)

results_df = pd.DataFrame(data=[["Linear Regression", *evaluate(y_test, test_pred) , cross_val(LinearRegression
                          columns=['Model', 'MAE', 'MSE', 'RMSE', 'R2 Square', "Cross Validation"])
```

```
Test set evaluation:

_____________________________________
MAE: 16894.16952798214
MSE: 508594356.3442272
RMSE: 22552.036634065385
R2 Square 0.9083946063161894
_____________________________________
Train set evaluation:

_____________________________________
MAE: 18523.452378607424
MSE: 756998410.1966709
RMSE: 27513.604093187627
R2 Square 0.8834571147788564
_____________________________________
```

In [33]:
```
from sklearn.preprocessing import PolynomialFeatures

poly_reg = PolynomialFeatures(degree=2)

X_train_2_d = poly_reg.fit_transform(X_train)
X_test_2_d = poly_reg.transform(X_test)

lin_reg = LinearRegression(normalize=True)
```

```
lin_reg.fit(X_train_2_d,y_train)

test_pred = lin_reg.predict(X_test_2_d)
train_pred = lin_reg.predict(X_train_2_d)

print('Test set evaluation:\n_____________________________________')
print_evaluate(y_test, test_pred)
print('==================================')
print('Train set evaluation:\n_____________________________________')
print_evaluate(y_train, train_pred)

results_df_2 = pd.DataFrame(data=[["Polynomail Regression", *evaluate(y_test, test_pred), 0]],
                            columns=['Model', 'MAE', 'MSE', 'RMSE', 'R2 Square', 'Cross Validation'])
results_df = results_df.append(results_df_2, ignore_index=True)
```

```
Test set evaluation:
_____________________________________
MAE: 13618.026786213966
MSE: 327533753.9362439
RMSE: 18097.893632581774
R2 Square 0.9410063086626963

__________________________________
==================================
Train set evaluation:
_____________________________________
MAE: 14223.502463536053
MSE: 385256789.38132197
RMSE: 19627.959378940082
R2 Square 0.9406882006874112
__________________________________
```

In [34]:
```
from sklearn.linear_model import SGDRegressor

sgd_reg = SGDRegressor(n_iter_no_change=250, penalty=None, eta0=0.0001, max_iter=100000)
sgd_reg.fit(X_train, y_train)

test_pred = sgd_reg.predict(X_test)
train_pred = sgd_reg.predict(X_train)

print('Test set evaluation:\n_____________________________________')
print_evaluate(y_test, test_pred)

print('==================================')
print('Train set evaluation:\n_____________________________________')
print_evaluate(y_train, train_pred)

results_df_2 = pd.DataFrame(data=[["Stochastic Gradient Descent", *evaluate(y_test, test_pred), 0]],
                            columns=['Model', 'MAE', 'MSE', 'RMSE', 'R2 Square', 'Cross Validation'])
results_df = results_df.append(results_df_2, ignore_index=True)
```

```
Test set evaluation:
_____________________________________
MAE: 16894.21484718697
MSE: 508596436.7303312
RMSE: 22552.082758147444
R2 Square 0.9083942316077687

__________________________________
==================================
Train set evaluation:
_____________________________________
MAE: 18523.51239363863
MSE: 756998410.3566595
RMSE: 27513.60409609507
R2 Square 0.8834571147542255
__________________________________
```

In [37]:
```
rfc = RandomForestRegressor()
rfc.fit(X_train, y_train)
predictions = rfc.predict(X_test)
score = r2_score(y_test, predictions)
print("Accuracy Score:", score)
new_row = {"Model": "RandomForestClassifier", "Accuracy Score": score}
models = models.append(new_row, ignore_index=True)
```

```
Accuracy Score: 0.9231396152244825
```

In [ ]: