EDA

August 13, 2021

0.1 # PRE-EDA: DATA LOADING

[1]:
```
# Data reading and visualization
import os
import numpy as np
import pandas as pd
import seaborn as sns
import plotly.express as px
import matplotlib.pyplot as plt
%matplotlib inline

# Statistical analysis
from scipy import stats
from scipy.stats import norm

# Scikit-learn
from sklearn.pipeline import Pipeline
from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder, OneHotEncoder
from sklearn.preprocessing import StandardScaler, MinMaxScaler, RobustScaler

# XGBoost & LightGBM
from xgboost import XGBRegressor
from lightgbm import LGBMRegressor

import warnings
warnings.filterwarnings('ignore')
```

0.2 ### CONFIGS

[2]:
```
BASE_PATH = "./"

rf_params = {
    'n_estimators': 10,
    'max_depth': 2,
    'min_samples_split': 2,
```

```
    'min_samples_leaf': 1
    }

xgb_params = {
    'n_estimators': 100,
    'max_depth': 2,
    'min_child_weight': 2,
    'learning_rate': 0.01,
    'subsample': 0.8,
    'colsample_bytree': 0.8,
    'booster': 'gbtree'
    }

lgb_params = {
    'n_estimators': 100,
    'max_depth': 2,
    'learning_rate': 0.01,
    'subsample': 0.8,
    'colsample_bytree': 0.8,
    'objective': 'regression'
    }
```

0.3 ### HELPER FUNCTIONS

[3]:
```
# Print dataset usage stats
def print_info(df):
    print(f"\nDataframe Shape: {df.shape}")
    print(f"\nDataframe Columns: {df.columns}")
    print(f"\nDataframe dtypes: \n{df.dtypes.value_counts()}")
    print(
        f"\nDataframe memory usage: {round(df.memory_usage().sum() / 1024**2,␣
 ↪2)} MB")

# Visualize misssing data
def visualize_missing_data(df):
    m_data = (df.isnull().sum() / len(df)) * 100
    m_data = m_data.drop(m_data[m_data == 0].index).sort_values()
    m_data = m_data.rename({'index': 'Feature', 0: 'Missing (%)'})

    fig = px.bar(x=m_data.index, y=m_data,
                 title='Missing Data by Feature', template='plotly_dark')
    fig.update_xaxes(title_text="Feature")
    fig.update_yaxes(title_text="Missing (%)")

    fig.show()

# Plot Histogram of dataset
```

```
def plot_histogram(df, distline=True):
    if distline:
        fig = plt.figure(figsize=(15, 15))
        for i, column in enumerate(df.columns):
            plt.subplot(4, 4, i+1)
            plt.title(column)
            plt.xlabel(column)
            sns.distplot(df[column], fit=norm)
        plt.tight_layout()
        plt.show()

    if not distline:
        fig = plt.figure(figsize=(15, 15))
        hist = df.hist(figsize=(15, 15), bins=50)
        plt.tight_layout()
        plt.show()

    plt.tight_layout()
    plt.show()

# Fit data to model(s)
def fit_robust_pipeline(model, X_train, y_train, X_test, y_test):
    pipe = Pipeline([('scaler', RobustScaler()), ('model', model)])
    pipe.fit(X_train, y_train)
    score = pipe.score(X_test, y_test)

    return round(score, 2)

# Scaling Data
def scale_data(scaler, df, feats_to_transform):
    scaled_df = df.copy()
    features = scaled_df[feats_to_transform]
    features = scaler.fit_transform(features.values)

    scaled_df[feats_to_transform] = features

    return scaled_df

# Fit data to model(s)
def evaluate_performance(X, Y, test_size=0.2, scale_data=False, scaler=None,␣
 ↪feats_to_transform=None):
    # Define models
    rf = RandomForestRegressor(**rf_params)
    xgb = XGBRegressor(**xgb_params)
    lgb = LGBMRegressor(**lgb_params)

    # Transform data
```

```
    if scale_data:
        X = scale_data(scaler, X, feats_to_transform)

    # Split data
    X_train, X_test, y_train, y_test = train_test_split(X, Y,␣
 ↪test_size=test_size)

    # Pass models to pipeline
    rf_score = fit_robust_pipeline(rf, X_train, y_train, X_test, y_test)
    xgb_score = fit_robust_pipeline(xgb, X_train, y_train, X_test, y_test)
    lgb_score = fit_robust_pipeline(lgb, X_train, y_train, X_test, y_test)

    # Print scores
    print(f"Model Scores: \n{'-'*25}\n")
    print(f"RandomForestRegressor Score: {rf_score}")
    print(f"XGBRegressor Score: {xgb_score}")
    print(f"LGBMRegressor Score: {lgb_score}")

    return rf, xgb, lgb

# Plot feature importance
def plot_feature_importance(features, title, model):
    fig = px.bar(y=features, x=model.feature_importances_,␣
 ↪template='plotly_dark')
    fig.update_layout(title=f"{title}")
    fig.update_xaxes(title_text="Feature Importance")
    fig.update_yaxes(title_text="Feature")

    fig.show()
```

0.4 # EXPLORATORY DATA ANALYSIS (EDA)

[4]:
```
# Read data
df = pd.read_csv(os.path.join(BASE_PATH, "dataset/train.csv"))
```

[5]:
```
# Check shape of dataset
print_info(df)
```

```
Dataframe Shape: (1460, 81)

Dataframe Columns: 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')

Dataframe dtypes:
object     43
int64      35
float64     3
dtype: int64

Dataframe memory usage: 0.9 MB
```

[6]:
```
# Check for null values
visualize_missing_data(df)
```

[7]:
```
# Some columns have a lot of missing data, we will either drop them or fill␣
↪them with the mean value of the column
# To keep things simple, we will only work with numerical data for now
```

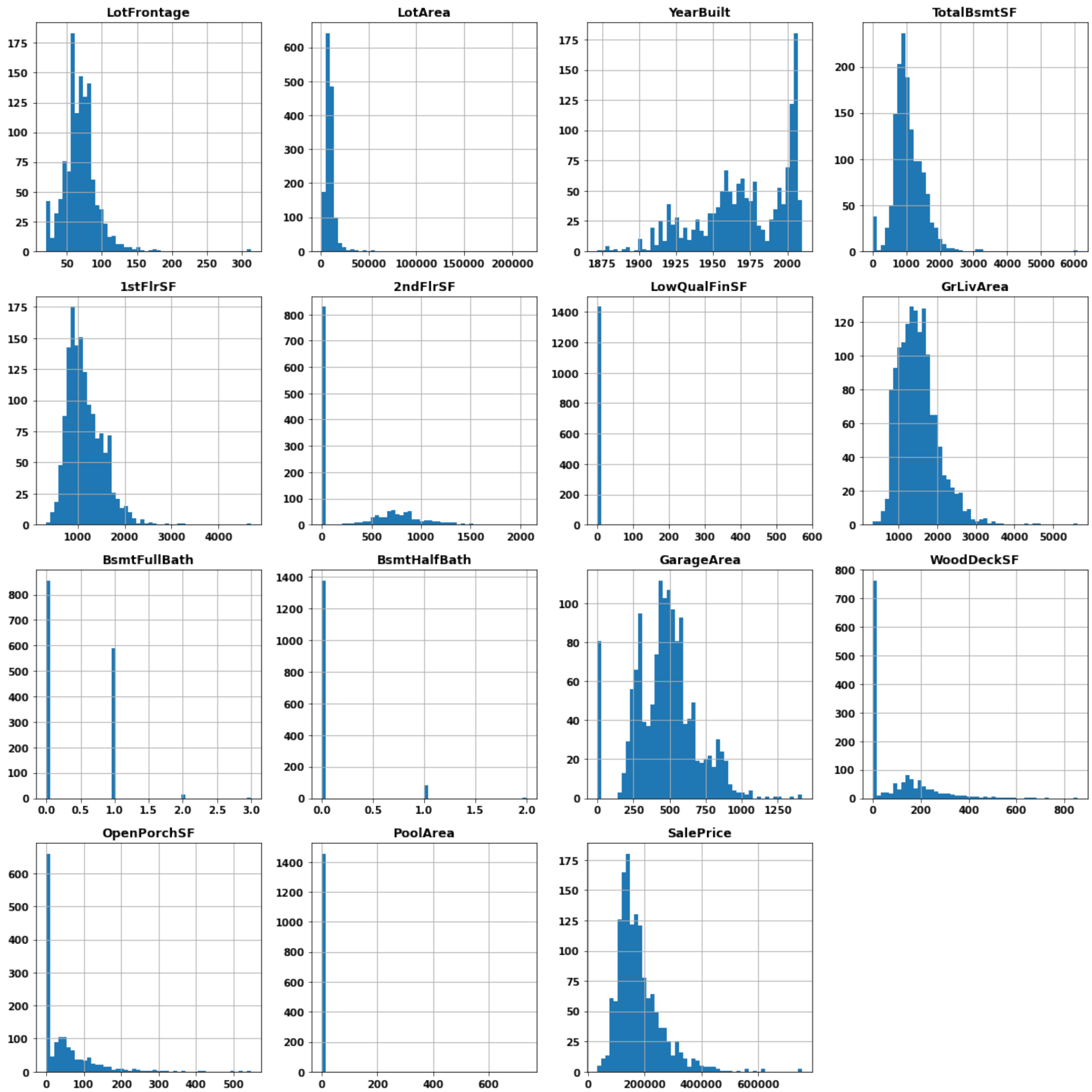
[8]:
```
# Filter and keep only the numerical data
df_num = df.select_dtypes(include=['float64', 'int64'])

# Filter out more catgeorical and time-series data given in `data_description.
↪txt`
to_drop = ['Id', 'MSSubClass', 'OverallQual', 'OverallCond',␣
↪'YearRemodAdd','MoSold', 'YrSold',  'FullBath', 'HalfBath', 'Fireplaces',␣
↪'GarageCars']
df_num.drop(to_drop, axis=1, inplace=True)

# Even more filtering to remove miscellanous features
to_drop = ['MasVnrArea', 'BsmtFinSF1', 'BsmtFinSF2', 'BsmtUnfSF',␣
↪'BedroomAbvGr', 'KitchenAbvGr', 'TotRmsAbvGrd', 'GarageYrBlt', 'MiscVal',␣
↪'EnclosedPorch', '3SsnPorch', 'ScreenPorch']
df_num.drop(to_drop, axis=1, inplace=True)

# Check shape of dataset
print_info(df_num)
```

```
Dataframe Shape: (1460, 15)

Dataframe Columns: Index(['LotFrontage', 'LotArea', 'YearBuilt', 'TotalBsmtSF',
'1stFlrSF',
       '2ndFlrSF', 'LowQualFinSF', 'GrLivArea', 'BsmtFullBath', 'BsmtHalfBath',
       'GarageArea', 'WoodDeckSF', 'OpenPorchSF', 'PoolArea', 'SalePrice'],
      dtype='object')

Dataframe dtypes:
int64      14
float64     1
dtype: int64

Dataframe memory usage: 0.17 MB
```

0.5 ## VISUALIZATION(S)!!!

[9]:
```
# Observe each feature's distribution
plot_histogram(df_num, distline=False)
```

```
<Figure size 1080x1080 with 0 Axes>
```




```
<Figure size 432x288 with 0 Axes>
```

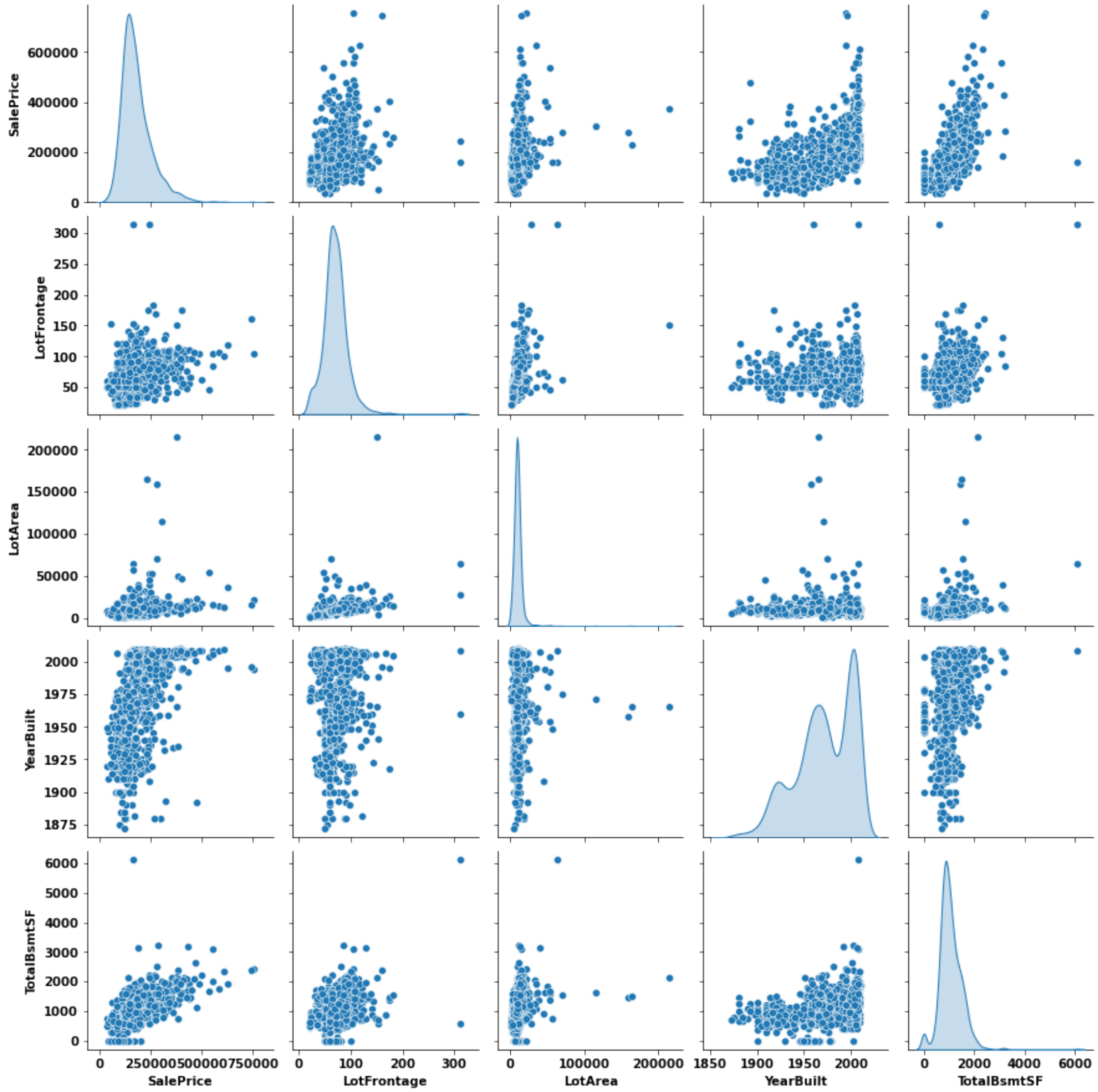
[10]:
```python
# There are a couple of features which remain constant. Let's remove them.

df_num.drop(['LowQualFinSF', 'PoolArea'], axis=1, inplace=True)
```

[11]:
```python
# Use pairplots to see how one feature is related to the other
features = list(df_num.columns)
features.remove('SalePrice')

# Since these can get very large, let's use only the SalePrice and the first 4␣
↪features
fig = plt.figure(figsize=(25, 10))
```

```python
viz_df = pd.concat([df_num['SalePrice'], df_num[features[:4]]], axis=1)
# sns.pairplot(viz_df, diag_kind='kde', hue='SalePrice', height=2.5)

sns.pairplot(viz_df, diag_kind='kde', height=2.5)
# plt.show()
```

[11]: <seaborn.axisgrid.PairGrid at 0x2acceb1df88>

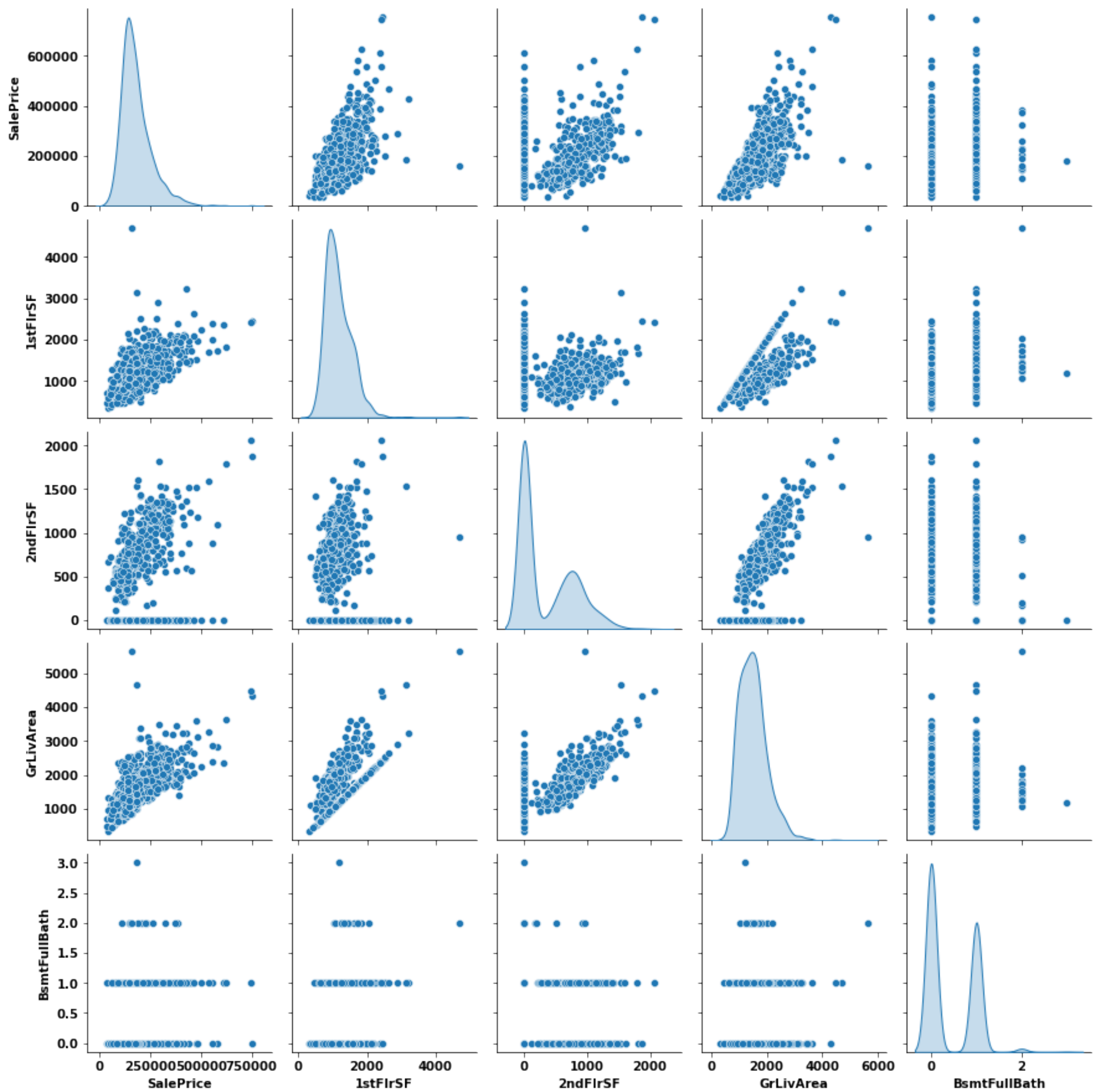
<Figure size 1800x720 with 0 Axes>



[12]:
```python
# Since these can get very large, let's use only the SalePrice and the second 4␣
↪features
fig = plt.figure(figsize=(25, 10))
```

```
viz_df = pd.concat([df_num['SalePrice'], df_num[features[4:8]]], axis=1)
# sns.pairplot(viz_df, diag_kind='kde', hue='SalePrice', height=2.5)
sns.pairplot(viz_df, diag_kind='kde', height=2.5)
plt.show()
```

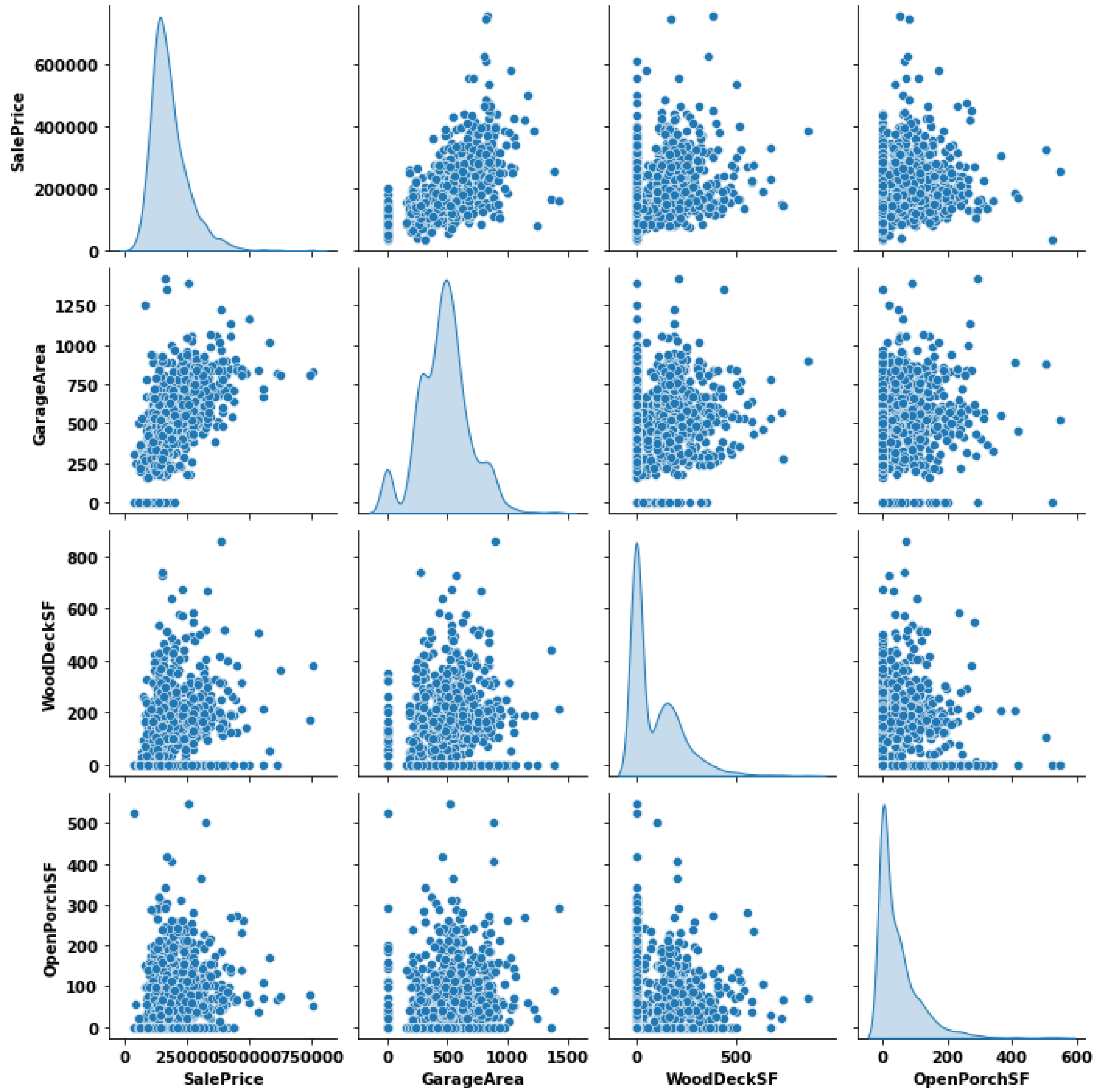
```
<Figure size 1800x720 with 0 Axes>
```



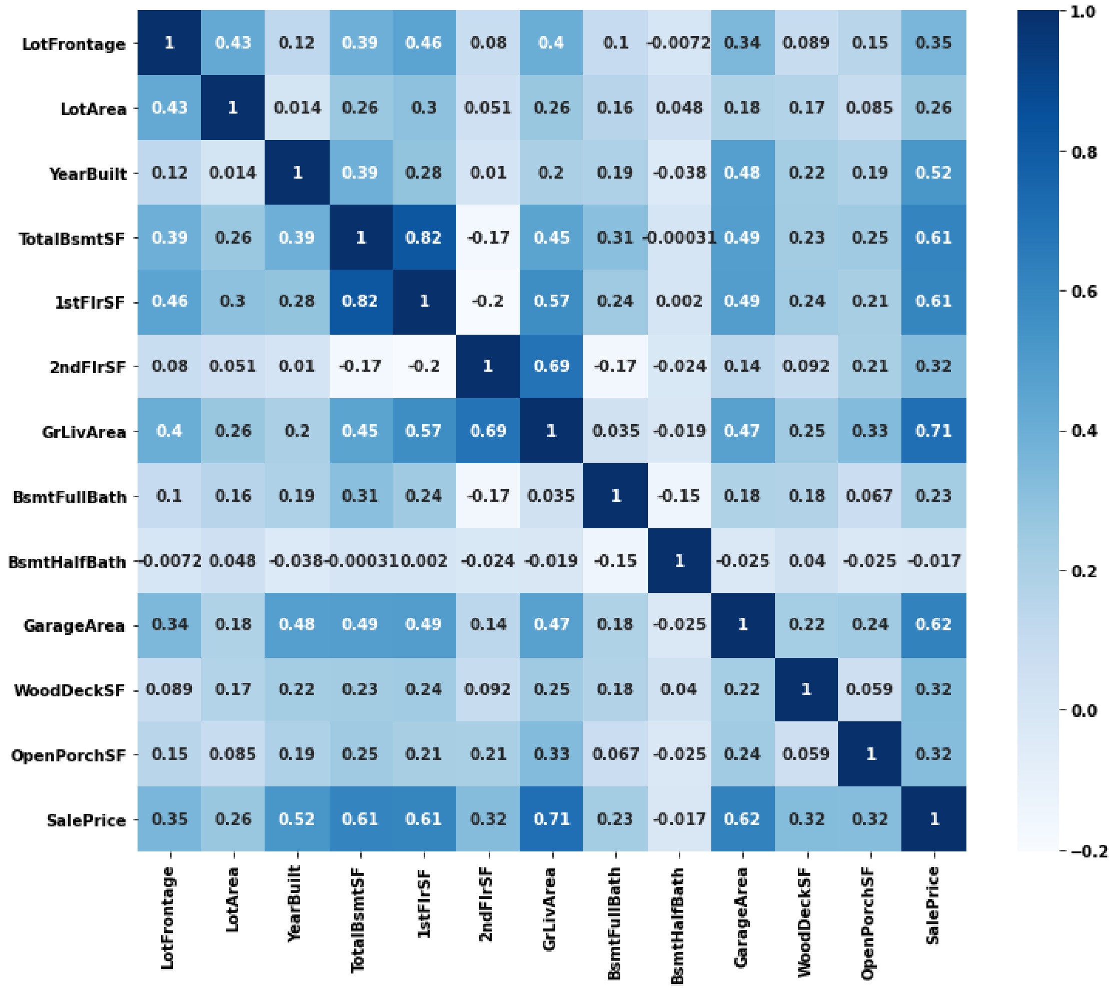
[13]:
```
# Since these can get very large, let's use only the SalePrice and the last 3 features
fig = plt.figure(figsize=(25, 10))
viz_df = pd.concat([df_num['SalePrice'], df_num[features[9:]]], axis=1)
# sns.pairplot(viz_df, diag_kind='kde', hue='SalePrice', height=2.5)
sns.pairplot(viz_df, diag_kind='kde', height=2.5)
```

```
plt.show()
```

<Figure size 1800x720 with 0 Axes>



[14]:
```
# Correlation Matrix
fig = plt.figure(figsize=(12, 10))
sns.heatmap(df_num.corr(), cmap='Blues', annot=True)
plt.show()
```



0.6 # FEATURE GENERATION

[15]:
```
# Now that we are done with cleaning the data, let's try to generate more␣
↪features
# by using the existing ones.

pre_gen_feats = df_num.columns.values.tolist()
print(pre_gen_feats)

# For example, We can combine the 'LotArea', 'TotalBsmtSF', 'GarageArea',␣
↪'WoodDeckSF', OpenPorchSF' features to generate a new feature: 'ExtArea'
df_num['ExtArea'] = df_num['LotArea'] + df_num['TotalBsmtSF'] +␣
↪df_num['GarageArea'] + df_num['WoodDeckSF'] + df_num['OpenPorchSF']

# Another example, combine all bathrooms in basement
df_num['AllBsmtBaths'] = df_num['BsmtFullBath'] + 0.5*df_num['BsmtHalfBath']
```

```
['LotFrontage', 'LotArea', 'YearBuilt', 'TotalBsmtSF', '1stFlrSF', '2ndFlrSF',
'GrLivArea', 'BsmtFullBath', 'BsmtHalfBath', 'GarageArea', 'WoodDeckSF',
'OpenPorchSF', 'SalePrice']
```

0.7 # DATA PREPROCESSING

[16]:
```
# We now need to make sure that the data has a normal distribution.

# To achieve this, we can use(among many options):
#     - StandardScaler: Normalizes data between 0 and 1
#     - MinMaxScaler: Normalizes data betweeen the min and max value in the␣
↪column
```

[17]:
```
# First. let's convert the categorical features to strings

df_num['BsmtFullBath'] = df_num['BsmtFullBath'].astype('str')
df_num['BsmtHalfBath'] = df_num['BsmtHalfBath'].astype('str')
df_num['AllBsmtBaths'] = df_num['AllBsmtBaths'].astype('str')
```

[18]:
```
# Next, let's work with the missing values

visualize_missing_data(df_num)
```

[19]:
```
# Since LotFrontage has a lot of missing values, we can:
# 1. Use `impute`:
#     - impute max/min value into missing areas
#     - impute mean/median/mode value into missing areas
# 2. drop the rows with the missing values
# 3. drop the whole column

# We will use option 1 as our dataset is relatively small, and losing data will␣
↪not be beneficial
```

[20]:
```
# Impute the mean value of the column in the missing value areas

df_num['LotFrontage'].fillna(df_num['LotFrontage'].mean(), inplace=True)
```

[21]:
```
# Next, we deal with outliers
# For this, we go back to the pairplots!

# In GrLivArea vs SalePrice --> we have outliers
df_num = df_num.loc[(df_num['GrLivArea'] < 4000) & (df_num['SalePrice'] <␣
↪300000)]
```

[22]:
```
# Encode the Categorical Data
cat_feats = df_num.select_dtypes('object').columns.values.tolist()
```

```
label_enc_df, onehot_enc_df = df_num.copy(), df_num.copy()

# LabelEncoding
for feat in cat_feats:
    label_enc = LabelEncoder()
    label_enc_df[feat] = label_enc.fit_transform(label_enc_df[feat])

# OneHotEncoding
for feat in cat_feats:
    onehot_enc = OneHotEncoder(handle_unknown='ignore')
    transformed = pd.DataFrame(onehot_enc.fit_transform(onehot_enc_df[feat].
 ↪values.reshape(-1, 1)).toarray())
    transformed.columns = [f"{feat}_{i}" for i in transformed.columns]
    onehot_enc_df = onehot_enc_df.join(transformed)
    onehot_enc_df.drop([feat], axis=1, inplace=True)
```

0.8 # NORMALIZATION

[23]:
```
# We first need to split the data into train and test sets
label_X, label_Y = label_enc_df.drop('SalePrice', axis=1),␣
 ↪label_enc_df['SalePrice']
onehot_X, onehot_Y = onehot_enc_df.drop('SalePrice', axis=1),␣
 ↪onehot_enc_df['SalePrice']
```

[24]:
```
# First, let's test see how our models perform without any scaling
rf, xgb, lgb = evaluate_performance(label_X, label_Y, test_size=0.2)
```

```
Model Scores:
-------------------------

RandomForestRegressor Score: 0.62
XGBRegressor Score: -0.6
LGBMRegressor Score: 0.53
```

[25]:
```
# List all features we wish to transform
feats_to_transform = ['LotFrontage', 'LotArea', 'YearBuilt', 'TotalBsmtSF',␣
 ↪'1stFlrSF',
                      '2ndFlrSF', 'GrLivArea', 'GarageArea', 'WoodDeckSF',␣
 ↪'OpenPorchSF', 'ExtArea']
```

[26]:
```
# Apply StandardScaler
scaler = StandardScaler()
standard_label_X = label_X.copy()
standard_label_X = scale_data(scaler, standard_label_X, feats_to_transform)

# Visualize scaled data
```

```
# plot_histogram(scaled_label_X, distline=False)

# Test model performance on new data
standardsc_rf, standardsc_xgb, standardsc_lgb =␣
↪evaluate_performance(standard_label_X, label_Y, test_size=0.2)
```

```
Model Scores:
-------------------------

RandomForestRegressor Score: 0.59
XGBRegressor Score: -0.82
LGBMRegressor Score: 0.55
```

[27]:
```
# Apply StandardScaler
scaler = MinMaxScaler()
minmax_label_X = label_X.copy()
minmax_label_X = scale_data(scaler, minmax_label_X, feats_to_transform)

# Visualize scaled data
# plot_histogram(minmax_label_X)

# Test model performance on new data
minmaxsc_rf, minmaxsc_xgb, minmaxsc_lgb = evaluate_performance(minmax_label_X,␣
↪label_Y, test_size=0.2)
```

```
Model Scores:
-------------------------

RandomForestRegressor Score: 0.59
XGBRegressor Score: -0.82
LGBMRegressor Score: 0.54
```

0.9 # FEATURE SELECTION

[28]:
```
# Let's see which of the features are the most important by each model

# StandardScaler Model

# RandomForestRegressor
plot_feature_importance(minmax_label_X.columns, "RandomForestRegressor Feature␣
↪Importance", standardsc_rf)
# XGBRegressor
plot_feature_importance(minmax_label_X.columns, "XGBRegressor Feature␣
↪Importance", standardsc_xgb)
# LGBMRegressor
plot_feature_importance(minmax_label_X.columns, "LGBMRegressor Feature␣
↪Importance", standardsc_lgb)
```

[29]:
```
# Let's see which of the features are the most important by each model

# MinMaxScaler Model

# RandomForestRegressor
plot_feature_importance(minmax_label_X.columns, "RandomForestRegressor Feature␣
 ↪Importance", minmaxsc_rf)
# XGBRegressor
plot_feature_importance(minmax_label_X.columns, "XGBRegressor Feature␣
 ↪Importance", minmaxsc_xgb)
# LGBMRegressor
plot_feature_importance(minmax_label_X.columns, "LGBMRegressor Feature␣
 ↪Importance", minmaxsc_lgb)
```

1 Aaaand.... We're Done! Good Luck Ahead!

[ ]:
```

```