Problem Statement

House price prediction has been an important area of research and practical application in real estate and finance for many years. Understanding the factors that contribute to a property's value and being able to accurately predict its price is crucial for real estate agents, buyers, and sellers to make informed decisions.

In recent years, the development of machine learning algorithms and big data technology has revolutionized the way we analyze and predict home prices. These technologies allow us to process and analyze large amounts of data, including information on property characteristics, local market trends, economic indicators, and demographic factors, to produce more accurate and reliable price predictions.

In this project, I aim to develop a machine learning model that can accurately predict the price of residential properties based on various features such as location, size, number of rooms, amenities, and other relevant factors. The goal is to create a model that can estimate the value of a house based on its characteristics.

In [2]:

```
pd.set_option('display.max_columns', None)

# Loading Data
df_train = pd.read_csv('D:\Hacktiv 8\Project\Kaggle\Competition\House Prediction/train.csv')
df_train.head(10)
```

Out[2]:

	Id	MSSubClass	MSZoning	LotFrontage	LotArea	Street	Alley	LotShape	LandContour	Utilities	LotConfig	LandSlope	Neighborhood	Condition1	Conditi
0	1	60	RL	65.0	8450	Pave	NaN	Reg	Lvl	AllPub	Inside	Gtl	CollgCr	Norm	N
1	2	20	RL	80.0	9600	Pave	NaN	Reg	Lvl	AllPub	FR2	Gtl	Veenker	Feedr	N
2	3	60	RL	68.0	11250	Pave	NaN	IR1	Lvl	AllPub	Inside	Gtl	CollgCr	Norm	N
3	4	70	RL	60.0	9550	Pave	NaN	IR1	Lvl	AllPub	Corner	Gtl	Crawfor	Norm	N
4	5	60	RL	84.0	14260	Pave	NaN	IR1	Lvl	AllPub	FR2	Gtl	NoRidge	Norm	N
5	6	50	RL	85.0	14115	Pave	NaN	IR1	Lvl	AllPub	Inside	Gtl	Mitchel	Norm	N
6	7	20	RL	75.0	10084	Pave	NaN	Reg	Lvl	AllPub	Inside	Gtl	Somerst	Norm	N
7	8	60	RL	NaN	10382	Pave	NaN	IR1	Lvl	AllPub	Corner	Gtl	NWAmes	PosN	N
8	9	50	RM	51.0	6120	Pave	NaN	Reg	Lvl	AllPub	Inside	Gtl	OldTown	Artery	N
9	10	190	RL	50.0	7420	Pave	NaN	Reg	Lvl	AllPub	Corner	Gtl	BrkSide	Artery	A

In [41]:

```
df_train.tail(10)
```

Out[41]:

	Id	MSSubClass	MSZoning	LotFrontage	LotArea	Street	Alley	LotShape	LandContour	Utilities	LotConfig	LandSlope	Neighborhood	Condition1	C
1450	1451	90	RL	60.0	9000	Pave	NaN	Reg	Lvl	AllPub	FR2	Gtl	NAmes	Norm	
1451	1452	20	RL	78.0	9262	Pave	NaN	Reg	Lvl	AllPub	Inside	Gtl	Somerst	Norm	
1452	1453	180	RM	35.0	3675	Pave	NaN	Reg	Lvl	AllPub	Inside	Gtl	Edwards	Norm	
1453	1454	20	RL	90.0	17217	Pave	NaN	Reg	Lvl	AllPub	Inside	Gtl	Mitchel	Norm	
1454	1455	20	FV	62.0	7500	Pave	Pave	Reg	Lvl	AllPub	Inside	Gtl	Somerst	Norm	
1455	1456	60	RL	62.0	7917	Pave	NaN	Reg	Lvl	AllPub	Inside	Gtl	Gilbert	Norm	

```
memory usage: 924.0+ KB
```

In [5]:
```python
df_train.describe()
```

Out[5]:

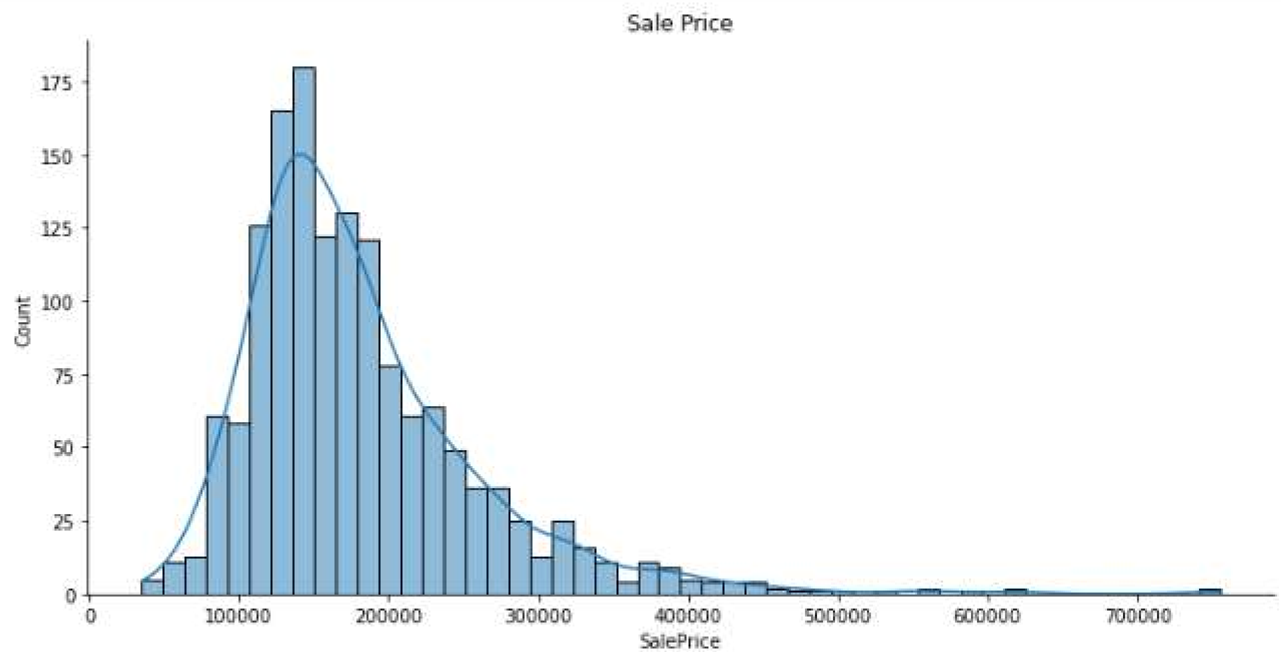
	Id	MSSubClass	LotFrontage	LotArea	OverallQual	OverallCond	YearBuilt	YearRemodAdd	MasVnrArea	BsmtFinSF1	BsmtFinSF2
count	1460.000000	1460.000000	1201.000000	1460.000000	1460.000000	1460.000000	1460.000000	1460.000000	1452.000000	1460.000000	1460.000000
mean	730.500000	56.897260	70.049958	10516.828082	6.099315	5.575342	1971.267808	1984.865753	103.685262	443.639726	46.549315
std	421.610009	42.300571	24.284752	9981.264932	1.382997	1.112799	30.202904	20.645407	181.066207	456.098091	161.319273
min	1.000000	20.000000	21.000000	1300.000000	1.000000	1.000000	1872.000000	1950.000000	0.000000	0.000000	0.000000
25%	365.750000	20.000000	59.000000	7553.500000	5.000000	5.000000	1954.000000	1967.000000	0.000000	0.000000	0.000000
50%	730.500000	50.000000	69.000000	9478.500000	6.000000	5.000000	1973.000000	1994.000000	0.000000	383.500000	0.000000
75%	1095.250000	70.000000	80.000000	11601.500000	7.000000	6.000000	2000.000000	2004.000000	166.000000	712.250000	0.000000
max	1460.000000	190.000000	313.000000	215245.000000	10.000000	9.000000	2010.000000	2010.000000	1600.000000	5644.000000	1474.000000

In [45]:
```python
df_train.isna().sum()
```

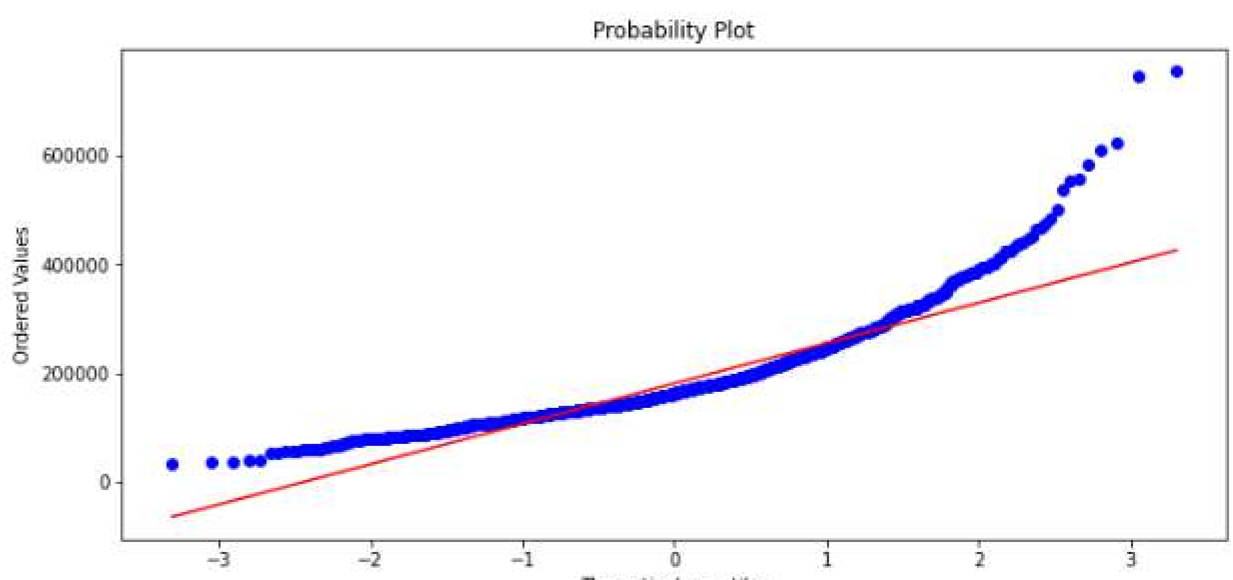
Out[45]:
```
Id                 0
MSSubClass         0
MSZoning           0
LotFrontage      259
LotArea            0
                ...
MoSold             0
YrSold             0
SaleType           0
SaleCondition      0
SalePrice          0
Length: 81, dtype: int64
```

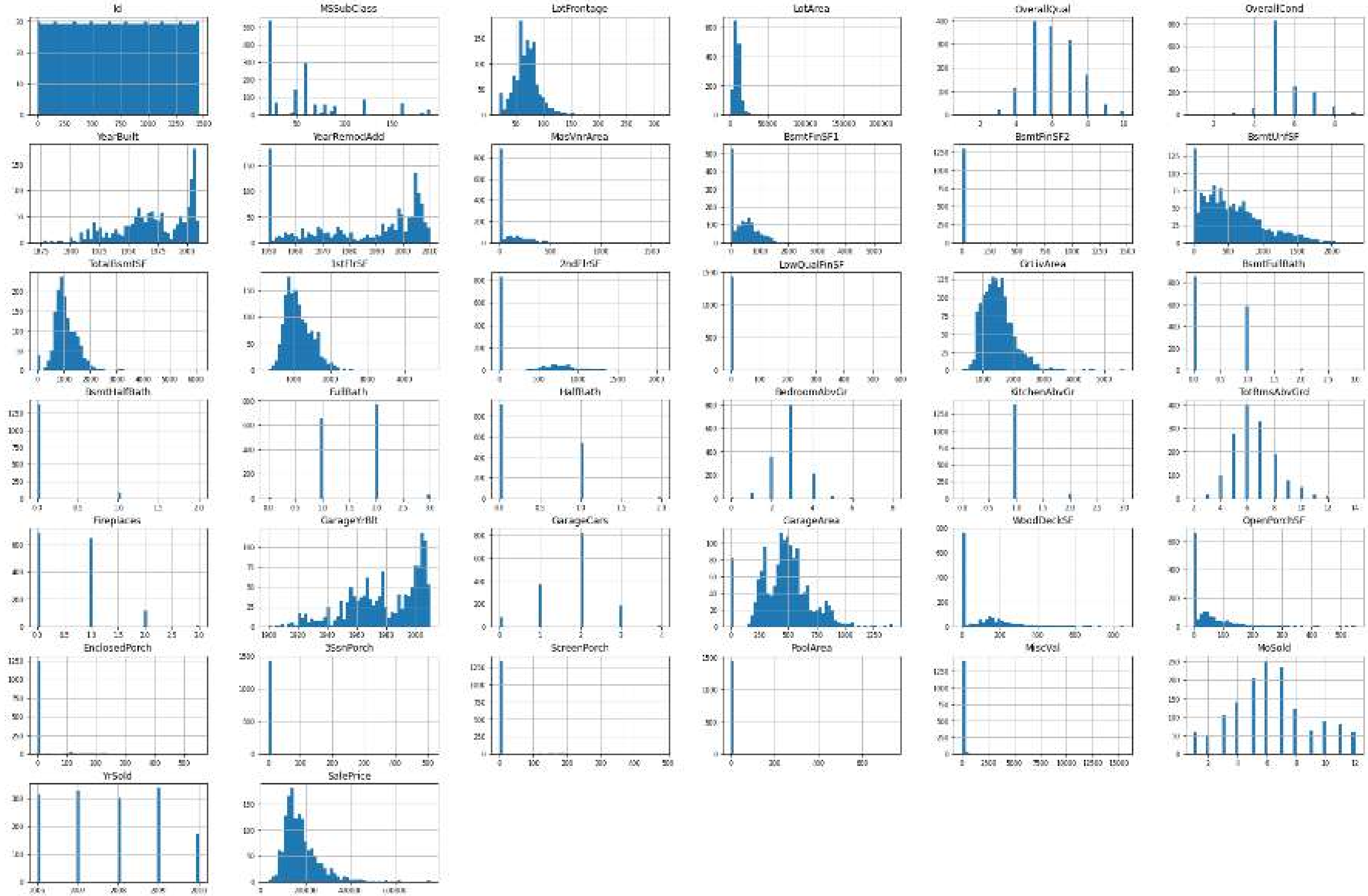
Too much missing value in categorical column such as `Alley, FireplaceQu, PoolQC, Fence, MiscFeature` I think this column is not useful so we can drop this 3 column, and take a look at the another column who had missing value.

In [9]:
```
sns.displot(data= df_train, x=df_train['SalePrice'],bins=50, kde=True, height=5, aspect=2)
plt.title('Sale Price')
plt.show()

print('-'*250)

fig = plt.figure(figsize=(11,5))
res = stats.probplot(df_train['SalePrice'], plot=plt)
plt.show()
```



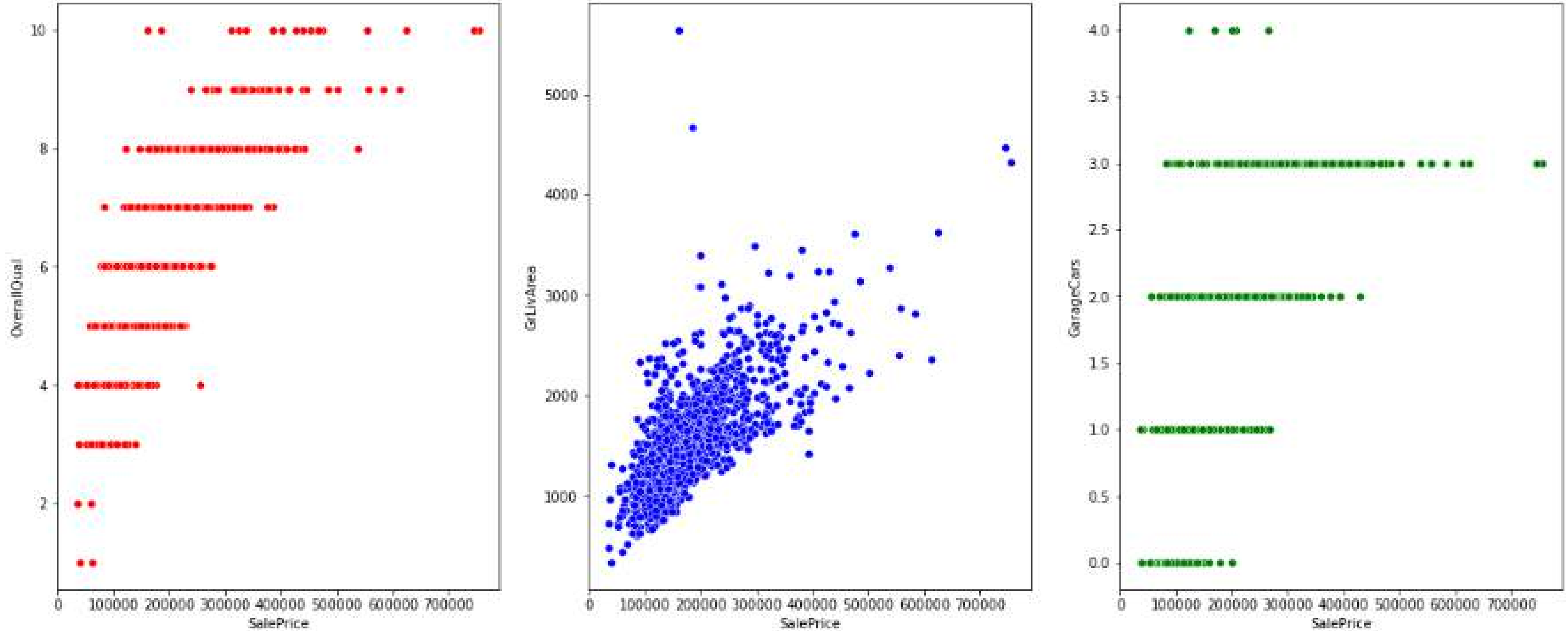

```
----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
```





From the histogram plot above, almost all columns have an abnormal data distribution, although there are several columns with normal data distribution.

In [11]:
```
# Scatter from top 3 correlaltion
plt.subplots(figsize=(20,8))

plt.subplot(1,3,1)
sns.scatterplot(data=df_train, x=df_train['SalePrice'], y=df_train['OverallQual'], color='red')

plt.subplot(1,3,2)
sns.scatterplot(data=df_train, x=df_train['SalePrice'], y=df_train['GrLivArea'], color='blue')

plt.subplot(1,3,3)
sns.scatterplot(data=df_train, x=df_train['SalePrice'], y=df_train['GarageCars'], color='green')

plt.show()
```



OverallQual: Rates the overall material and finish of the house 1-10
GrLivArea: Above grade (ground) living area square feet
GarageCars: Size of garage in car capacity

From all the feature like `OverallQual, GrLivArea, Garage Cars` We can conclude that the higher the value of the feature, the more it affects the selling price.

However, the highest value garagecars do not have much influence because according to the local community garage cars with level 4 are too big than what is needed.

Data Preprocessing

In [47]:
```
df_test = pd.read_csv('D:\Hacktiv 8\Project\Kaggle\Competition\House Prediction/test.csv')
```

Handle Missing value

In [13]:
```
df_train_cleaned = df_train.drop(columns=['Id','Alley', 'FireplaceQu', 'PoolQC', 'Fence', 'MiscFeature'])
df_test_cleaned = df_test.drop(columns=['Id','Alley', 'FireplaceQu', 'PoolQC', 'Fence', 'MiscFeature'])
```

In [14]:
```
cols_with_missing = df_train_cleaned.columns[df_train_cleaned.isnull().any()]
col_test_with_missing = df_test_cleaned.columns[df_test_cleaned.isnull().any()]
print(f'Columns that contain missing values: {cols_with_missing}')
```

```
Columns that contain missing values: Index(['LotFrontage', 'MasVnrType', 'MasVnrArea', 'BsmtQual', 'BsmtCond',
       'BsmtExposure', 'BsmtFinType1', 'BsmtFinType2', 'Electrical',
       'GarageType', 'GarageYrBlt', 'GarageFinish', 'GarageQual',
       'GarageCond'],
      dtype='object')
```

In [15]:
```
def imputation(dataframe, cols):
    mode_val = dataframe[cols].mode()
    if mode_val.empty:
        # Handle empty mode value DataFrame
        return dataframe
    else:
        # fill missing value with mode
        mode_val = mode_val.iloc[0]
        dataframe[cols] = dataframe[cols].fillna(mode_val)
    return dataframe

df_train_cleaned = imputation(df_train_cleaned, cols_with_missing)
df_test_cleaned = imputation(df_test_cleaned, col_test_with_missing)
```

In [46]:
```
df_test_cleaned.isna().sum()
```

Out[46]:
```
MSSubClass       0
MSZoning         0
LotFrontage      0
LotArea          0
Street           0
                ..
MiscVal          0
MoSold           0
YrSold           0
SaleType         0
SaleCondition    0
Length: 74, dtype: int64
```

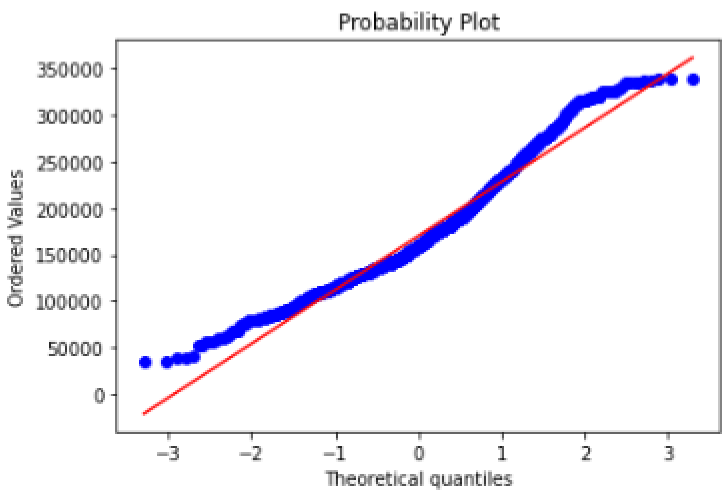
Handling Outlier

In [17]:
```
# make function IQR

def limit(data, variable):
    IQR= df_train_cleaned[variable].quantile(0.75) - df_train_cleaned[variable].quantile(0.25)

    lower_limit = df_train_cleaned[variable].quantile(0.25) - (IQR*1.5)
    upper_limit = df_train_cleaned[variable].quantile(0.75) + (IQR*1.5)

    return lower_limit, upper_limit


# menentukan lower limit dan upper limit dari kolom limit balance
lower_sale, upper_sale = limit(df_train_cleaned, 'SalePrice')

# menghapus outlier
df_no_outliers = df_train_cleaned[(df_train_cleaned.SalePrice > lower_sale)&(df_train_cleaned.SalePrice < upper_sale)]
print(f'Jumlah row dan kolom : {df_no_outliers.shape}')

print(f'Jumlah outlier pada kolom age : {len(df_train_cleaned)-len(df_no_outliers)}')
```

```
Jumlah row dan kolom : (1399, 75)
Jumlah outlier pada kolom age : 61
```

In [18]:
```
# Probability plot after handling outlier
stats.probplot(df_no_outliers['SalePrice'], plot=plt)
```

Out[18]:
```
((array([-3.29316232, -3.03508712, -2.89151591, ...,  2.89151591,
          3.03508712,  3.29316232]),
  array([ 34900,  35311,  37900, ..., 339750, 340000, 340000], dtype=int64)),
 (58226.61934268641, 170237.12723373837, 0.9813743604481673))
```



After we handled an outlier using IQR the target variable is getting close to the normal distribution

Feature Engineering ¶

In [ ]:
```
# Split column between numerical and categorical for feature engineering

num_columns = df_train_cleaned.select_dtypes(include=np.number).columns.tolist()
cat_columns = df_train_cleaned.select_dtypes(include= ['object']).columns.tolist()
num_columns
```

In [20]:
```
# Data by Dtypes

data_numeric = df_no_outliers[num_columns]
data_categoric = df_no_outliers[cat_columns]
data_numeric.head()
```

Out[20]:

	MSSubClass	LotFrontage	LotArea	OverallQual	OverallCond	YearBuilt	YearRemodAdd	MasVnrArea	BsmtFinSF1	BsmtFinSF2	...	WoodDeckSF	OpenPorc
0	60	65.0	8450	7	5	2003	2003	196.0	706	0	...	0	
1	20	80.0	9600	6	8	1976	1976	0.0	978	0	...	298	
2	60	68.0	11250	7	5	2001	2002	162.0	486	0	...	0	
3	70	60.0	9550	7	5	1915	1970	0.0	216	0	...	0	
4	60	84.0	14260	8	5	2000	2000	350.0	655	0	...	192	

5 rows × 37 columns

Find out correlation by numeric values

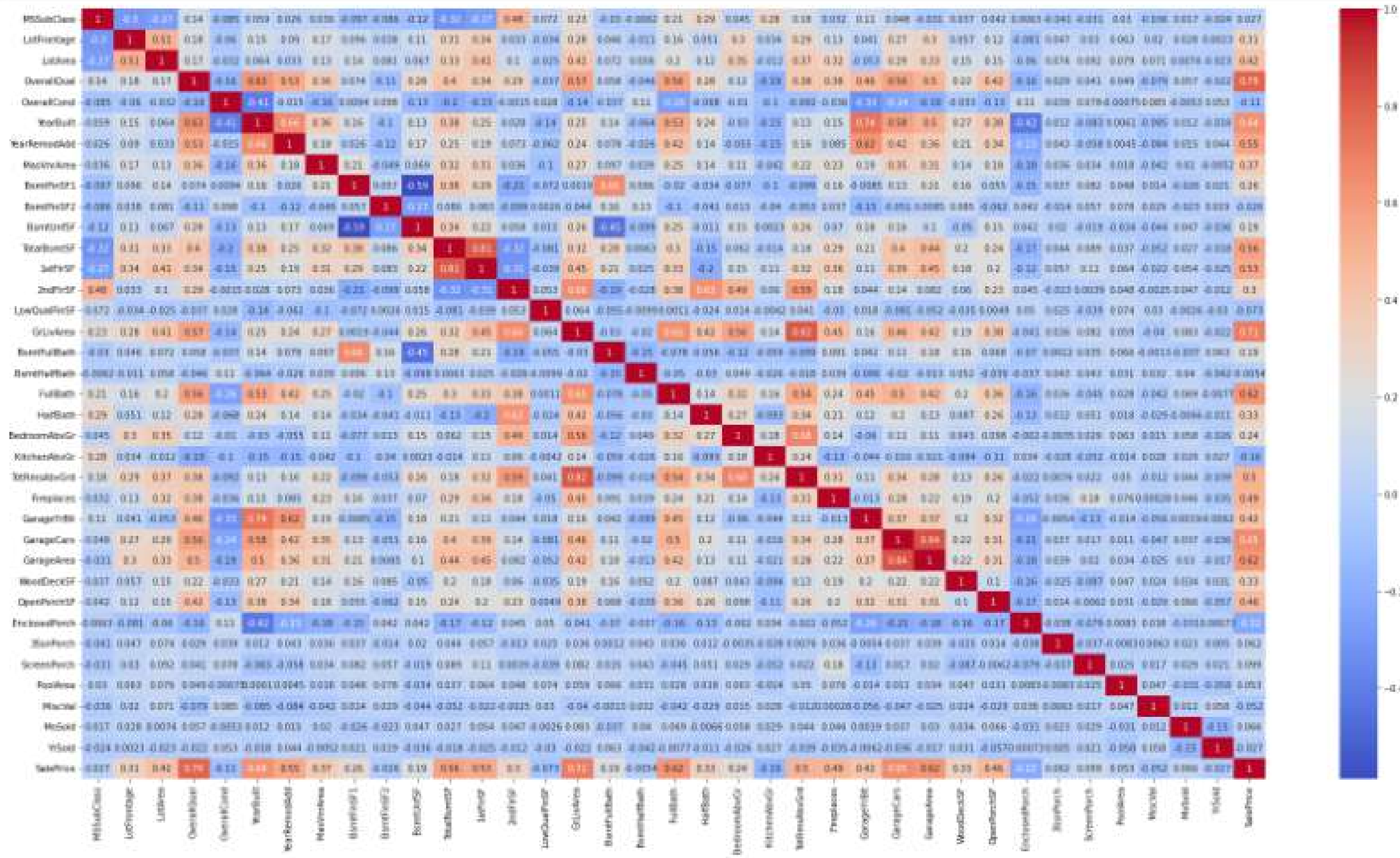
The Spearman and Pearson methods are two commonly used methods in correlation analysis. Both methods have their own advantages and disadvantages, and the choice of method depends on the type of data and the purpose of the analysis to be achieved.

According to `Kendall and Stuart in their book "The Advanced Theory of Statistics" (1958)`, the Spearman method is more suitable for data that does not have a normal distribution, while the Pearson method is more suitable for data that has a normal distribution. However, both methods can produce the same results if the data has a normal distribution.

So in this case I used the Spearman method because most of feature were not normally distributed.

In [21]:

```
plt.figure(figsize=(30,16))
sns.heatmap(data_numeric.corr(method='spearman'), annot=True, cmap='coolwarm')
plt.show()
```



In this case, the columns that have a high correlation include: `LotFrontage, LotArea, OverallQual, YearBuilt, YearRemodAdd, MasVnrArea, BsmtFinSF1, TotalBsmtSF, 1stFlrSF, 2ndFlrSF, GrLivArea, FullBath, HalfBath, BedroomAbvGr, TotRmsAbvGrd, FirePlaces, GarageYrBlt, GarageCars, WoodDeckSF, OpenPorchSF, EnclosedPorch.`

Find out correlation by categorical values using ANOVA

ANOVA (Analysis of Variance) is a statistical analysis method used to compare the means of several independent groups of data. This method was first introduced by a statistician named Ronald A. Fisher in 1925.

According to Fisher, ANOVA aims to test the null hypothesis that there is no significant difference between the means of the groups being compared. Fisher also developed the F-ratio method used in ANOVA to calculate the variance between groups (between variance) and the variance within groups (within variance), and compare the two variances to determine whether the difference between groups is statistically significant or not.

In this case I used ANOVA to determine which features can be used for training during modeling.

P_value <= 0.05 : Use feature
P_value > 0.05 : Delete feature

In [23]:
```
# Encoding for ANOVA
oe= OrdinalEncoder()
cat = oe.fit_transform(data_categoric)
cat
```

Out[23]:
```
array([[3., 1., 3., ..., 2., 8., 4.],
       [3., 1., 3., ..., 2., 8., 4.],
       [3., 1., 0., ..., 2., 8., 4.],
       ...,
       [3., 1., 3., ..., 2., 8., 4.],
       [3., 1., 3., ..., 2., 8., 4.],
       [3., 1., 3., ..., 2., 8., 4.]])
```

In [24]:
```
target = df_no_outliers['SalePrice']

anova = SelectKBest(score_func=f_regression, k=30)
anova.fit_transform(cat,target)

anova_score = pd.DataFrame({'Anova_score':anova.scores_, 'P_value_anova': anova.pvalues_}, index=data_categoric.columns)
anova_score.sort_values(by=['P_value_anova'], ascending=False)
```

Out[24]:

	Anova_score	P_value_anova
Condition2	0.017402	8.950701e-01
SaleType	0.042864	8.360108e-01
Utilities	0.305556	5.805087e-01
BsmtFinType2	0.518551	4.715811e-01
MasVnrType	1.596159	2.066592e-01
LandContour	2.364749	1.243303e-01
Street	2.758101	9.698743e-02

From the ANOVA test, the columns with the results of the P_value <= 0.05 are including:
`['MSZoning', 'LotShape', 'LotConfig', 'LandSlope', 'Neighborhood', 'Condition1', 'BldgType', 'HouseStyle', 'RoofStyle', 'RoofMatl', 'Exterior1st', 'Exterior2nd', 'ExterQual', 'ExterCond', 'Foundation', 'BsmtQual', 'BsmtExposure', 'BsmtFinType1', 'Heating', 'HeatingQC', 'CentralAir', 'Electrical', 'KitchenQual', 'Functional', 'GarageType', 'GarageFinish', 'GarageQual', 'GarageCond', 'PavedDrive', 'SaleCondition']`

Choosing important Feature after Correlation and ANOVA Testing

In [25]:
```
df_no_outliers.head()
```

Out[25]:

	MSSubClass	MSZoning	LotFrontage	LotArea	Street	LotShape	LandContour	Utilities	LotConfig	LandSlope	...	EnclosedPorch	3SsnPorch	ScreenPorch
0	60	RL	65.0	8450	Pave	Reg	Lvl	AllPub	Inside	Gtl	...	0	0	0
1	20	RL	80.0	9600	Pave	Reg	Lvl	AllPub	FR2	Gtl	...	0	0	0
2	60	RL	68.0	11250	Pave	IR1	Lvl	AllPub	Inside	Gtl	...	0	0	0
3	70	RL	60.0	9550	Pave	IR1	Lvl	AllPub	Corner	Gtl	...	272	0	0
4	60	RL	84.0	14260	Pave	IR1	Lvl	AllPub	FR2	Gtl	...	0	0	0

5 rows × 75 columns

In [26]:
```
num_feature = ['LotFrontage', 'LotArea', 'OverallQual', 'YearBuilt', 'YearRemodAdd', 'MasVnrArea', 'BsmtFinSF1', 'TotalBsmtSF', '
               'GarageCars', 'WoodDeckSF', 'OpenPorchSF', 'EnclosedPorch']
cat_feature = ['MSZoning', 'LotShape', 'LotConfig', 'LandSlope', 'Neighborhood', 'Condition1', 'BldgType', 'HouseStyle', 'RoofSty
               'BsmtExposure', 'BsmtFinType1', 'Heating', 'HeatingQC', 'CentralAir', 'Electrical', 'KitchenQual', 'Functional', '
```

Scalling

In [27]:
```
minmax = MinMaxScaler()

train_scaled = minmax.fit_transform(df_no_outliers[num_feature])
test_scaled = minmax.transform(df_test_cleaned[num_feature])
test_scaled
```

Out[27]:
```
array([[0.20205479, 0.0631856 , 0.44444444, ..., 0.19021739, 0.        ,
        0.        ],
       [0.20547945, 0.07937684, 0.55555556, ..., 0.53396739, 0.06581353,
        0.        ],
       [0.18150685, 0.07670176, 0.44444444, ..., 0.28804348, 0.06215722,
        0.        ],
       ...,
       [0.4760274 , 0.11447111, 0.44444444, ..., 0.64402174, 0.        ,
        0.        ],
```

Encoding

In [28]:
```
oe = OrdinalEncoder()
train_encoded = oe.fit_transform(df_no_outliers[cat_feature])
test_encoded = oe.transform(df_test_cleaned[cat_feature])
test_encoded
```

Out[28]:
```
array([[2., 3., 4., ..., 4., 2., 4.],
       [3., 0., 0., ..., 4., 2., 4.],
       [3., 0., 4., ..., 4., 2., 4.],
       ...,
       [3., 3., 4., ..., 4., 2., 0.],
       [3., 3., 4., ..., 4., 2., 4.],
       [3., 3., 4., ..., 4., 2., 4.]])
```

concat after scalling and enocoding

In [29]:
```
X_train_final = np.concatenate((train_scaled, train_encoded), axis=1)
X_test_final = np.concatenate((test_scaled,test_encoded), axis=1)
y_train = df_no_outliers['SalePrice']
y_train.reset_index(drop=True, inplace=True)
```

Model Definition

In [30]:
```
# Define the Model

rf = RandomForestRegressor()
gb = GradientBoostingRegressor()
```

In [31]:
```
rf.fit(X_train_final,y_train)
```

Out[31]:
```
RandomForestRegressor()
```

In [32]:
```
gb.fit(X_train_final,y_train)
```

Out[32]:
```
GradientBoostingRegressor()
```

In [33]:
```
rf_pred = rf.predict(X_train_final)
gb_pred = gb.predict(X_train_final)
```

Comparison Actual and Prediction Data

In [34]:
```
a = pd.DataFrame(rf_pred, columns=['prediction'])
comparison_rf = pd.concat([y_train,a], axis=1)

comparison_rf
```

Out[34]:

	SalePrice	prediction
0	208500	206166.50
1	181500	175825.90
2	223500	218815.00
3	140000	154337.78
4	250000	264962.63
...	...	...
1394	175000	176399.00
1395	210000	207940.00
1396	266500	257811.39
1397	142125	139628.50
1398	147500	148572.00

1399 rows × 2 columns

In [35]:
```
b = pd.DataFrame(gb_pred, columns=['prediction'])
comparison_gb = pd.concat([y_train,b], axis=1)
comparison_gb
```

Out[35]:

	SalePrice	prediction
0	208500	197249.058354
1	181500	170342.357335
2	223500	207030.016860
3	140000	164184.566484
4	250000	286294.093553
...	...	...
1394	175000	171027.726558
1395	210000	193948.640539
1396	266500	267578.423684
1397	142125	131165.360670
1398	147500	145093.456622

Model Evaluation

In [36]:
```
# Evaluation Random Forest
mae = mean_absolute_error(y_train, rf_pred)
r2 = r2_score(y_train, rf_pred)
mse = mean_squared_error(y_train, rf_pred)


# Evaluation Gradient Boosting
mae_gb = mean_absolute_error(y_train, gb_pred)
r2_gb = r2_score(y_train, gb_pred)
mse_gb = mean_squared_error(y_train, gb_pred)
```

In [37]:
```
data = {'Model': ['Random Forest', 'Gradient Boosting'],

        'Mean Absolute Error (MAE)': [mae, mae_gb],
        'R-Squared (R2)': [r2, r2_gb],
        'Mean Squared Error (MSE)': [mse, mse_gb]
        }

# Buat dataframe dari dictionary
df = pd.DataFrame(data)
df
```

Out[37]:

	Model	Mean Absolute Error (MAE)	R-Squared (R2)	Mean Squared Error (MSE)
0	Random Forest	5715.320393	0.981594	6.452492e+07
1	Gradient Boosting	10222.530775	0.947027	1.857089e+08

Submission

In [38]:
```
df_test_cleaned.head()
```

Out[38]:

	MSSubClass	MSZoning	LotFrontage	LotArea	Street	LotShape	LandContour	Utilities	LotConfig	LandSlope	...	OpenPorchSF	EnclosedPorch	3SsnPorch
0	20	RH	80.0	11622	Pave	Reg	Lvl	AllPub	Inside	Gtl	...	0	0	0
1	20	RL	81.0	14267	Pave	IR1	Lvl	AllPub	Corner	Gtl	...	36	0	0
2	60	RL	74.0	13830	Pave	IR1	Lvl	AllPub	Inside	Gtl	...	34	0	0
3	60	RL	78.0	9978	Pave	IR1	Lvl	AllPub	Inside	Gtl	...	36	0	0
4	120	RL	43.0	5005	Pave	IR1	HLS	AllPub	Inside	Gtl	...	82	0	0

5 rows × 74 columns

In [39]:
```
ids = df_test.pop('Id')

test_pred = rf.predict(X_test_final)
df = pd.DataFrame({'Id': ids,
                   'SalePrice': test_pred.squeeze()})
```

In [40]:
```
df.head(10)
```

Out[40]:

	Id	SalePrice
0	1461	123721.32
1	1462	153361.31
2	1463	182754.00
3	1464	182798.50
4	1465	191985.22
5	1466	185623.30
6	1467	165785.04
7	1468	175411.35
8	1469	180456.89
9	1470	121990.58