PCA and Variance Threshold in a Linear Regression

In [1]:
```
import pandas as pd
```

In [2]:
```
# Import the housing data as a data frame
```

In [3]:
```
house_df = pd.read_csv("train.csv")
house_df.head()
```

Out[3]:

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

5 rows × 81 columns

In [4]:
```
print(house_df.dtypes)
```

```
Id                 int64
MSSubClass         int64
MSZoning          object
LotFrontage      float64
LotArea            int64
                  ...
MoSold             int64
YrSold             int64
SaleType          object
SaleCondition     object
SalePrice          int64
Length: 81, dtype: object
```

In [5]:
```
# Drop the "Id" column and any features that are missing more than 40% of th
```

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

Out[6]:
```
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 [7]:
```
null_percentage = (house_df.isnull().sum() / len(house_df)) * 100
null_percentage[null_percentage > 0]
```

Out[7]:
```
LotFrontage     17.739726
Alley           93.767123
MasVnrType      59.726027
MasVnrArea       0.547945
BsmtQual         2.534247
BsmtCond         2.534247
BsmtExposure     2.602740
BsmtFinType1     2.534247
BsmtFinType2     2.602740
Electrical       0.068493
FireplaceQu     47.260274
GarageType       5.547945
GarageYrBlt      5.547945
GarageFinish     5.547945
GarageQual       5.547945
GarageCond       5.547945
PoolQC          99.520548
Fence           80.753425
MiscFeature     96.301370
dtype: float64
```

In [8]:
```
threshold = 40
house_df = house_df.loc[:, null_percentage <= threshold]
house_df.shape
```

Out[8]:
```
(1460, 75)
```

In [9]:
```
house_df = house_df.drop(columns=['Id'], errors='ignore')
house_df.shape
```

Out[9]:
```
(1460, 74)
```

In [10]:
```
# For numerical columns, fill in any missing data with the median value.
```

In [11]:
```
# house_df.dtypes
```

In [12]:
```
numerical_columns = house_df.select_dtypes(include=['float64', 'int64']).col
```

In [13]:
```
house_df[numerical_columns] = house_df[numerical_columns].fillna(house_df[nu
house_df[numerical_columns]
```

Out[13]:

	MSSubClass	LotFrontage	LotArea	OverallQual	OverallCond	YearBuilt	YearRe
0	60	65.0	8450	7	5	2003	
1	20	80.0	9600	6	8	1976	
2	60	68.0	11250	7	5	2001	
3	70	60.0	9550	7	5	1915	
4	60	84.0	14260	8	5	2000	
...	...	...	...	...	...	...	
1455	60	62.0	7917	6	5	1999	
1456	20	85.0	13175	6	6	1978	
1457	70	66.0	9042	7	9	1941	
1458	20	68.0	9717	5	6	1950	
1459	20	75.0	9937	5	6	1965	

1460 rows × 37 columns

In [14]:
```
house_df[numerical_columns].isnull().sum().sum()
```

Out[14]: 0

In [15]:
```
# For categorical columns, fill in any missing data with the most common val
```

In [16]:
```
categorical_columns = house_df.select_dtypes(include=['object']).columns
```

In [17]:
```
house_df[categorical_columns] = house_df[categorical_columns].apply(lambda x
```

In [18]:
```
house_df.isnull().sum().sum()
```

Out[18]: 0

In [19]:
```
# Convert the categorical columns to dummy variables.
```

In [20]:
```
house_df_dummies = pd.get_dummies(house_df, columns = categorical_columns, c
house_df_dummies.head(), house_df_dummies.shape
```

Out[20]:
```
(   MSSubClass  LotFrontage  LotArea  OverallQual  OverallCond  YearBuilt  \
 0          60         65.0     8450            7            5       2003
 1          20         80.0     9600            6            8       1976
 2          60         68.0    11250            7            5       2001
 3          70         60.0     9550            7            5       1915
 4          60         84.0    14260            8            5       2000

    YearRemodAdd  MasVnrArea  BsmtFinSF1  BsmtFinSF2  ...  SaleType_ConLI  \
 0          2003       196.0         706           0  ...           False
 1          1976         0.0         978           0  ...           False
 2          2002       162.0         486           0  ...           False
 3          1970         0.0         216           0  ...           False
 4          2000       350.0         655           0  ...           False

    SaleType_ConLw  SaleType_New  SaleType_Oth  SaleType_WD  \
 0           False         False         False         True
 1           False         False         False         True
 2           False         False         False         True
 3           False         False         False         True
 4           False         False         False         True

    SaleCondition_AdjLand  SaleCondition_Alloca  SaleCondition_Family  \
 0                  False                 False                 False
 1                  False                 False                 False
 2                  False                 False                 False
 3                  False                 False                 False
 4                  False                 False                 False

    SaleCondition_Normal  SaleCondition_Partial
 0                  True                  False
 1                  True                  False
 2                  True                  False
 3                 False                  False
 4                  True                  False

 [5 rows x 230 columns],
 (1460, 230))
```

In [21]:
```python
# Split the data into a training and test set, where the SalePrice column is
```

In [22]:
```python
from sklearn.model_selection import train_test_split
```

In [23]:
```python
X = house_df_dummies.drop(columns=['SalePrice'])
y = house_df_dummies['SalePrice']
```

In [24]:
```python
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25, ra
```

In [25]:
```python
X_train.shape, X_test.shape, y_train.shape, y_test.shape
```

Out[25]:
```
((1095, 229), (365, 229), (1095,), (365,))
```

In [26]:
```python
# Run a linear regression and report the R2-value and RMSE on the test set.
```

In [27]:
```python
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score
import numpy as np
```

In [28]:
```python
linear_model = LinearRegression()
linear_model.fit(X_train, y_train)
```

Out[28]:
```
▾ LinearRegression
LinearRegression()
```

In [29]:
```python
y_pred = linear_model.predict(X_test)
```

In [30]:
```python
r2 = r2_score(y_test, y_pred)
rmse = np.sqrt(mean_squared_error(y_test, y_pred))
```

In [31]:
```python
r2, rmse
```

Out[31]:
```
(0.6861151594140671, 46892.00694306782)
```

The R^2 value is 0.686 means that the model explains about 68.6% of the variability in house prices. This is a reasonably good fit, but there is still 31.4% of the variance that the model isn't capturing, which could be due to unmodeled factors or noise. The RMSE is $\$46892$, means the model's predictions are, on average, about $\$46892$ off from the actual sale price.

In [32]:
```python
# Fit and transform the training features with a PCA so that 90% of the vari
```

In [33]:
```python
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler
```

In [34]:
```python
scaler = StandardScaler()
```

In [35]:
```python
X_train_scaled = scaler.fit_transform(X_train)
```

In [36]:
```python
X_test_scaled = scaler.transform(X_test)
```

In [37]:
```python
pca = PCA(n_components=0.90, random_state=42)
```

In [38]:
```python
X_train_pca = pca.fit_transform(X_train_scaled)
```

In [39]:
```python
X_test_pca = pca.transform(X_test_scaled)
```

In [40]:
```python
X_train_pca.shape
```

Out[40]:
```
(1095, 124)
```

In [41]:
```python
# How many features are in the PCA-transformed matrix?
```

In [42]:
```
from sklearn.decomposition import PCA
```

In [43]:
```
num_pca_features = X_train_pca.shape[1]
num_pca_features
```

Out[43]: 124

There are 124 features in the pca transformed matrix

In [44]:
```
# Transform but DO NOT fit the test features with the same PCA
```

In [45]:
```
X_test_pca = pca.transform(X_test_scaled)
```

In [46]:
```
print(X_test_pca.shape)
```

```
(365, 124)
```

In [47]:
```
# Repeat step 7 with your PCA transformed data.
```

In [48]:
```
linear_model_pca = LinearRegression()
```

In [49]:
```
linear_model_pca.fit(X_train_pca, y_train)
```

Out[49]:

```
▾ LinearRegression
LinearRegression()
```

In [50]:
```
y_pred_pca = linear_model_pca.predict(X_test_pca)
```

In [51]:
```
r2_pca = r2_score(y_test, y_pred_pca)
```

In [52]:
```
rmse_pca = np.sqrt(mean_squared_error(y_test, y_pred_pca))
```

In [53]:
```
r2_pca, rmse_pca
```

Out[53]: (0.8378636357689185, 33701.859593520654)

After applying PCA to retain 90% of the variance, the model's performance was evaluated with an R^2 (R-squared) of approximately 0.838 and an RMSE of about 33,701.86.
The PCA-transformed model now explains around 83.8% of the variance in SalePrice, up from the previous 69.3% without PCA. This suggests that reducing the feature set through PCA has actually enhanced the model's ability to generalize, potentially by removing noise or redundant information.
The RMSE is now around $\$33702$, compared to the earlier $\$46892$. This reduction indicates that the model's predictions are now closer to the actual SalePrice values, with an average error lower by about $\$13190$.

In [54]:
```
# Take your original training features (from step 6) and apply a min-max sca
```

In [55]:
```python
from sklearn.preprocessing import MinMaxScaler
```

In [56]:
```python
min_max_scaler = MinMaxScaler()
```

In [57]:
```python
X_train_minmax_scaled = min_max_scaler.fit_transform(X_train)
```

In [58]:
```python
pd.DataFrame(X_train_minmax_scaled, columns=X_train.columns).head()
```

Out[58]:

	MSSubClass	LotFrontage	LotArea	OverallQual	OverallCond	YearBuilt	YearRem
0	0.588235	0.075342	0.008797	0.666667	0.500	0.963768	0.9
1	0.000000	0.195205	0.041319	0.555556	0.625	0.739130	0.8
2	0.176471	0.133562	0.036271	0.555556	0.500	0.485507	0.0
3	0.000000	0.164384	0.051611	0.444444	0.500	0.637681	0.4
4	0.000000	0.184932	0.039496	0.555556	0.625	0.623188	0.1

5 rows × 229 columns

In [59]:
```python
# Find the min-max scaled features in your training set that have a variance
```

In [60]:
```python
import numpy as np
```

In [61]:
```python
variances = np.var(X_train_minmax_scaled, axis=0)
```

In [62]:
```python
high_variance_features = X_train.columns[variances > 0.1].tolist()
high_variance_features
```

Out[62]:
```
['YearRemodAdd',
 'YrSold',
 'MSZoning_RL',
 'MSZoning_RM',
 'LotShape_Reg',
 'LotConfig_Inside',
 'Neighborhood_NAmes',
 'Condition1_Norm',
 'HouseStyle_1Story',
 'HouseStyle_2Story',
 'RoofStyle_Gable',
 'RoofStyle_Hip',
 'Exterior1st_HdBoard',
 'Exterior1st_MetalSd',
 'Exterior1st_VinylSd',
 'Exterior1st_Wd Sdng',
 'Exterior2nd_HdBoard',
 'Exterior2nd_MetalSd',
 'Exterior2nd_VinylSd',
 'Exterior2nd_Wd Sdng',
 'ExterQual_Gd',
 'ExterQual_TA',
 'ExterCond_TA',
 'Foundation_CBlock',
 'Foundation_PConc',
 'BsmtQual_Gd',
 'BsmtQual_TA',
 'BsmtExposure_No',
 'BsmtFinType1_GLQ',
 'BsmtFinType1_Unf',
 'HeatingQC_Gd',
 'HeatingQC_TA',
 'KitchenQual_Gd',
 'KitchenQual_TA',
 'GarageType_Attchd',
 'GarageType_Detchd',
 'GarageFinish_RFn',
 'GarageFinish_Unf',
 'SaleType_WD',
 'SaleCondition_Normal']
```

In [63]:
```
# Transform but DO NOT fit the test features with the same steps applied in
```

In [64]:
```
X_test_minmax_scaled = min_max_scaler.transform(X_test)
```

In [65]:
```
X_test_minmax_scaled_high_variance = \
pd.DataFrame(X_test_minmax_scaled, columns=X_test.columns)[high_variance_fea
```

In [66]:
```
X_test_minmax_scaled_high_variance.head()
```

Out[66]:

	YearRemodAdd	YrSold	MSZoning_RL	MSZoning_RM	LotShape_Reg	LotConfig_In
0	0.883333	0.00	1.0	0.0	1.0	
1	0.750000	1.00	1.0	0.0	0.0	
2	0.000000	1.00	0.0	1.0	1.0	
3	0.000000	0.00	0.0	1.0	1.0	
4	0.966667	0.75	1.0	0.0	0.0	

5 rows × 40 columns

In [67]:
```
# repeat step 7 with the high variance data
```

In [68]:
```
X_train_minmax_scaled_high_variance = \
pd.DataFrame(X_train_minmax_scaled, columns=X_train.columns)[high_variance_f
```

In [69]:
```
linear_model_high_variance = LinearRegression()
```

In [70]:
```
linear_model_high_variance.fit(X_train_minmax_scaled_high_variance, y_train)
```

Out[70]:
```
▾ LinearRegression
LinearRegression()
```

In [71]:
```
y_pred_high_variance = linear_model_high_variance.predict(X_test_minmax_scal
```

In [72]:
```
r2_high_variance = r2_score(y_test, y_pred_high_variance)
```

In [73]:
```
rmse_high_variance = np.sqrt(mean_squared_error(y_test, y_pred_high_variance
```

In [74]:
```
r2_high_variance, rmse_high_variance
```

Out[74]:
```
(0.638593254016746, 50316.65677698575)
```

This R^2 value means the model explains approximately 63.9% of the variability in SalePrice, slightly lower than the earlier model's R^2 of 68.6% without PCA or feature filtering. This suggests that while the high-variance features contribute significantly to the model's predictive power, some lower-variance features (excluded in this subset) likely contain additional useful information.
The average prediction error is around $\$50317$, which is slightly higher than the RMSE from the full feature set (around $\$46892$). This indicates that excluding lower-variance features increased the model's prediction error slightly, suggesting that those features, though less varied, still added predictive value.

In [75]:
```
# Summarize your findings.
```

Feature Selection By applying PCA to retain 90% of the variance, we reduced the dataset's dimensionality, improving model performance. This PCA-transformed model showing improved accuracy and explanatory power than using the original dadatset. However after applying Min-Max scaling, we focused on high-variance features (variance >0.1), which captured significant information with fewer features.
Model Accuracy The highest accuracy was achieved with the PCA-transformed data, suggesting that dimensionality reduction effectively balanced model simplicity with predictive performance. The high-variance feature model, though less accurate, highlighted how variance-based feature selection can simplify the model at the expense of some predictive power.
This R^2 and RMSE showed that while focusing on high-variance features can simplify the model, there is a trade-off in accuracy. Retaining all features, or carefully selecting additional lower-variance ones, may better balance simplicity with predictive power.
Conclusion Overall, using PCA or carefully retaining lower-variance features alongside high-variance ones improves model performance. Variance-based feature filtering is a viable approach but is best used in combination with other techniques to retain key predictive information.

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]: