Implement a linear regression model to predict the prices of houses based on their square footage and the number of bedrooms and bathrooms.

```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline
```

```
#reading files
train_ = pd.read_csv('train.csv')
test_ = pd.read_csv('test.csv')
```

```
print(train_.shape)
print(test_.shape)
```

```
(1460, 81)
(1459, 80)
```

```
train_.head(10)
```

	Id	MSSubClass	MSZoning	LotFrontage	LotArea	Street	Alley	LotShape	LandContour	Utilities	...	PoolArea	PoolQC	Fence	Mis
0	1	60	RL	65.0	8450	Pave	NaN	Reg	Lvl	AllPub	...	0	NaN	NaN	
1	2	20	RL	80.0	9600	Pave	NaN	Reg	Lvl	AllPub	...	0	NaN	NaN	
2	3	60	RL	68.0	11250	Pave	NaN	IR1	Lvl	AllPub	...	0	NaN	NaN	
3	4	70	RL	60.0	9550	Pave	NaN	IR1	Lvl	AllPub	...	0	NaN	NaN	
4	5	60	RL	84.0	14260	Pave	NaN	IR1	Lvl	AllPub	...	0	NaN	NaN	
5	6	50	RL	85.0	14115	Pave	NaN	IR1	Lvl	AllPub	...	0	NaN	MnPrv	
6	7	20	RL	75.0	10084	Pave	NaN	Reg	Lvl	AllPub	...	0	NaN	NaN	
7	8	60	RL	NaN	10382	Pave	NaN	IR1	Lvl	AllPub	...	0	NaN	NaN	
8	9	50	RM	51.0	6120	Pave	NaN	Reg	Lvl	AllPub	...	0	NaN	NaN	
9	10	190	RL	50.0	7420	Pave	NaN	Reg	Lvl	AllPub	...	0	NaN	NaN	

10 rows × 81 columns

```
from IPython.display import display #to show all columns

# Set pandas display options globally
pd.set_option('display.max_columns', None)
pd.set_option('display.max_columns', None)

display(train_)
display(test_)
```

Afficher la sortie masquée

```
print(train_.dtypes)
print("_________________________")
print(test_.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
_________________________
Id                 int64
MSSubClass         int64
MSZoning          object
LotFrontage      float64
```

```
LotArea            int64
                   ...
MiscVal            int64
MoSold             int64
YrSold             int64
SaleType          object
SaleCondition     object
Length: 80, dtype: object
```

```
#Summary
train_.describe()
```

	Id	MSSubClass	LotFrontage	LotArea	OverallQual	OverallCond	YearBuilt	YearRemodAdd	MasVnrArea	BsmtFi
count	1460.000000	1460.000000	1201.000000	1460.000000	1460.000000	1460.000000	1460.000000	1460.000000	1452.000000	1460.00
mean	730.500000	56.897260	70.049958	10516.828082	6.099315	5.575342	1971.267808	1984.865753	103.685262	443.63
std	421.610009	42.300571	24.284752	9981.264932	1.382997	1.112799	30.202904	20.645407	181.066207	456.09
min	1.000000	20.000000	21.000000	1300.000000	1.000000	1.000000	1872.000000	1950.000000	0.000000	0.00
25%	365.750000	20.000000	59.000000	7553.500000	5.000000	5.000000	1954.000000	1967.000000	0.000000	0.00
50%	730.500000	50.000000	69.000000	9478.500000	6.000000	5.000000	1973.000000	1994.000000	0.000000	383.50
75%	1095.250000	70.000000	80.000000	11601.500000	7.000000	6.000000	2000.000000	2004.000000	166.000000	712.25
max	1460.000000	190.000000	313.000000	215245.000000	10.000000	9.000000	2010.000000	2010.000000	1600.000000	5644.00

```
test_.describe()
```

	Id	MSSubClass	LotFrontage	LotArea	OverallQual	OverallCond	YearBuilt	YearRemodAdd	MasVnrArea	BsmtFin
count	1459.000000	1459.000000	1232.000000	1459.000000	1459.000000	1459.000000	1459.000000	1459.000000	1444.000000	1458.000
mean	2190.000000	57.378341	68.580357	9819.161069	6.078821	5.553804	1971.357779	1983.662783	100.709141	439.203
std	421.321334	42.746880	22.376841	4955.517327	1.436812	1.113740	30.390071	21.130467	177.625900	455.268
min	1461.000000	20.000000	21.000000	1470.000000	1.000000	1.000000	1879.000000	1950.000000	0.000000	0.000
25%	1825.500000	20.000000	58.000000	7391.000000	5.000000	5.000000	1953.000000	1963.000000	0.000000	0.000
50%	2190.000000	50.000000	67.000000	9399.000000	6.000000	5.000000	1973.000000	1992.000000	0.000000	350.500
75%	2554.500000	70.000000	80.000000	11517.500000	7.000000	6.000000	2001.000000	2004.000000	164.000000	753.500
max	2919.000000	190.000000	200.000000	56600.000000	10.000000	9.000000	2010.000000	2010.000000	1290.000000	4010.000

```
#show the number of empty values
train_.isna().sum()
```

	0
Id	0
MSSubClass	0
MSZoning	0
LotFrontage	259
LotArea	0
...	...
MoSold	0
YrSold	0
SaleType	0
SaleCondition	0
SalePrice	0

81 rows × 1 columns

dtype: int64

```
test_.isna().sum()
```

	0
Id	0
MSSubClass	0
MSZoning	4
LotFrontage	227
LotArea	0
...	...
MiscVal	0
MoSold	0
YrSold	0
SaleType	1
SaleCondition	0

80 rows × 1 columns

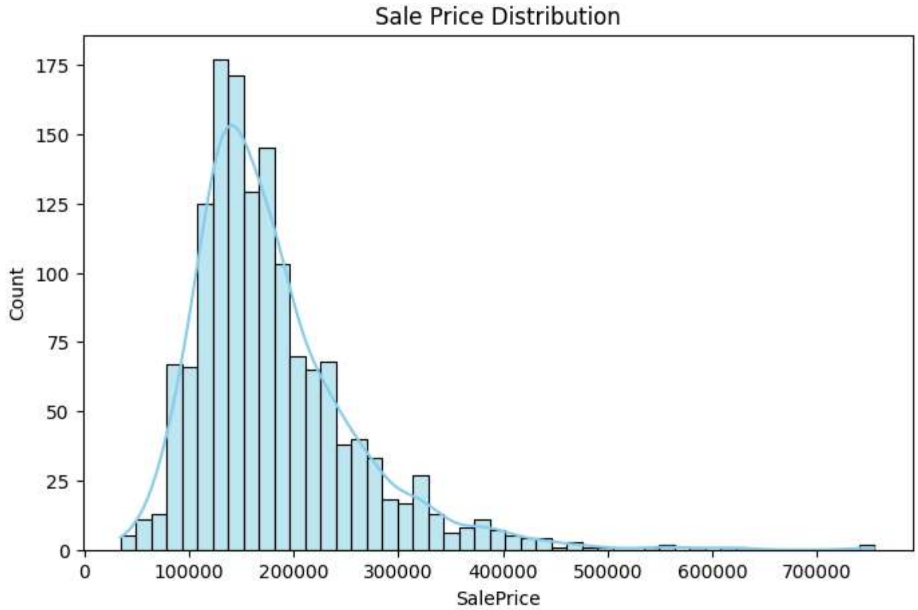
dtype: int64

```
for column in train_.columns:
    if train_[column].dtype == 'object':
        # Replace Nan  with the mode for categorical columns
        train_[column] = train_[column].fillna(train_[column].mode()[0])  # We don't use inplace=True here
        if column in test_.columns:
            test_[column] = test_[column].fillna(test_[column].mode()[0])
    else:
        # Replace Nan with the mean for numeric columns
        train_[column] = train_[column].fillna(train_[column].mean())
        if column in test_.columns:
            test_[column] = test_[column].fillna(test_[column].mean())
```

```
train_.head(10)
```

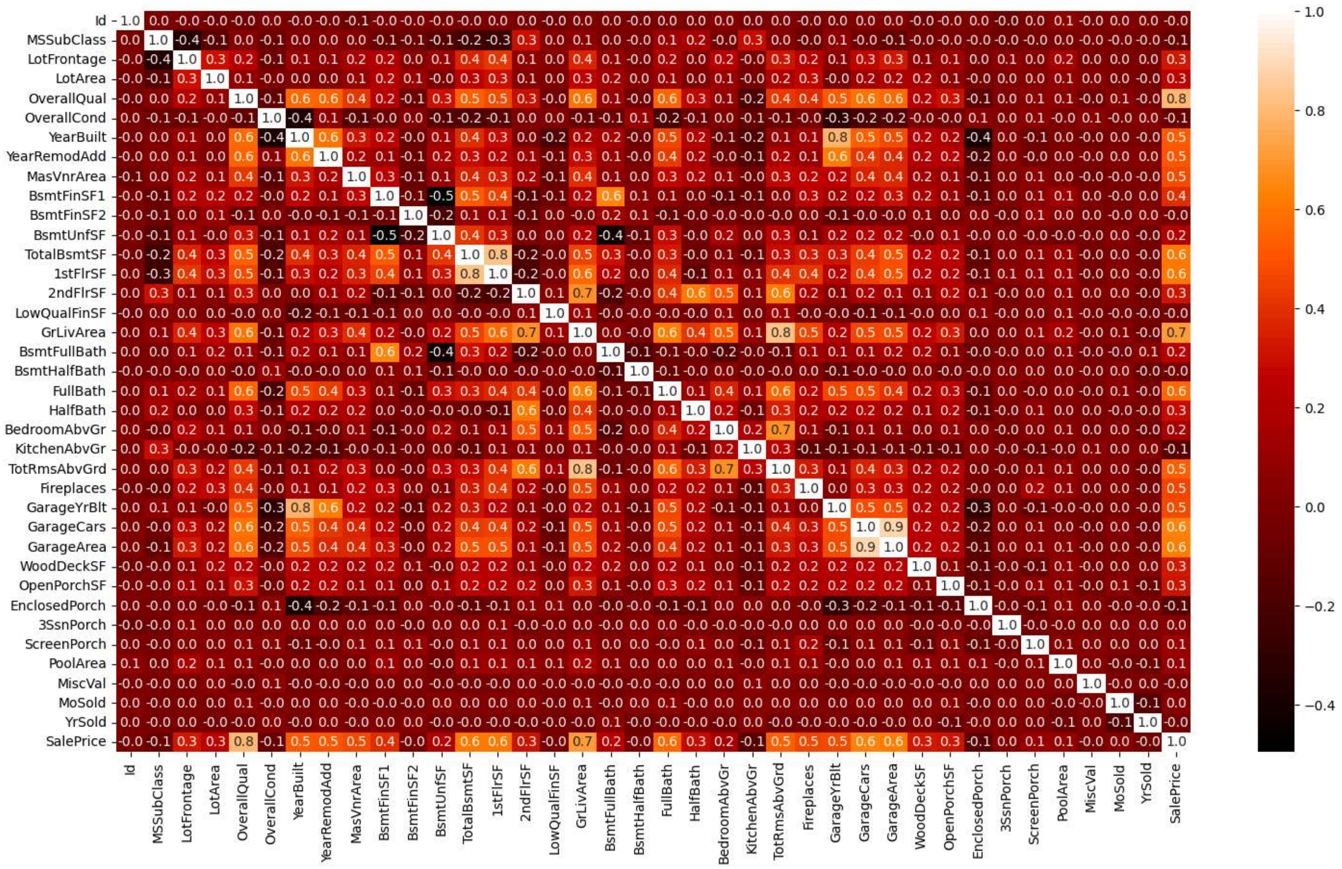
	Id	MSSubClass	MSZoning	LotFrontage	LotArea	Street	Alley	LotShape	LandContour	Utilities	LotConfig	LandSlope	Neighborhoc
0	1	60	RL	65.000000	8450	Pave	Grvl	Reg	Lvl	AllPub	Inside	Gtl	CollgC
1	2	20	RL	80.000000	9600	Pave	Grvl	Reg	Lvl	AllPub	FR2	Gtl	Veenk
2	3	60	RL	68.000000	11250	Pave	Grvl	IR1	Lvl	AllPub	Inside	Gtl	CollgC
3	4	70	RL	60.000000	9550	Pave	Grvl	IR1	Lvl	AllPub	Corner	Gtl	Crawf
4	5	60	RL	84.000000	14260	Pave	Grvl	IR1	Lvl	AllPub	FR2	Gtl	NoRidg
5	6	50	RL	85.000000	14115	Pave	Grvl	IR1	Lvl	AllPub	Inside	Gtl	Mitch
6	7	20	RL	75.000000	10084	Pave	Grvl	Reg	Lvl	AllPub	Inside	Gtl	Somer
7	8	60	RL	70.049958	10382	Pave	Grvl	IR1	Lvl	AllPub	Corner	Gtl	NWAme
8	9	50	RM	51.000000	6120	Pave	Grvl	Reg	Lvl	AllPub	Inside	Gtl	OldTow
9	10	190	RL	50.000000	7420	Pave	Grvl	Reg	Lvl	AllPub	Corner	Gtl	BrkSic

```
# Sale price distribution
plt.figure(figsize=(8, 5))
sns.histplot(train_['SalePrice'], color="skyblue" , kde=True) #Kernel Density Estimate (kde=True) adds a smooth curve to the histogram,
plt.title('Sale Price Distribution')
plt.show()
```




```
#Correlation verification
correlation_matrix = train_.corr(numeric_only=True)
plt.figure(figsize=(18,10))
sns.heatmap(correlation_matrix, annot=True, cmap='gist_heat', fmt=".1f")
```

<Axes: >



After EDA now we can Select only features we need for **prediction**

```
col = ['GrLivArea', 'BedroomAbvGr', 'FullBath']
X = train_[col]
y = train_['SalePrice']
```

```
# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

Train the Linear Regression Model

```
# Initialize and train the model
model = LinearRegression()
model.fit(X_train, y_train)

# Display model coefficients
print("Model Coefficients:", model.coef_)
print("Model Intercept:", model.intercept_)
```

```
Model Coefficients: [   104.02630701 -26655.16535734  30014.32410896]
Model Intercept: 52261.74862694461
```

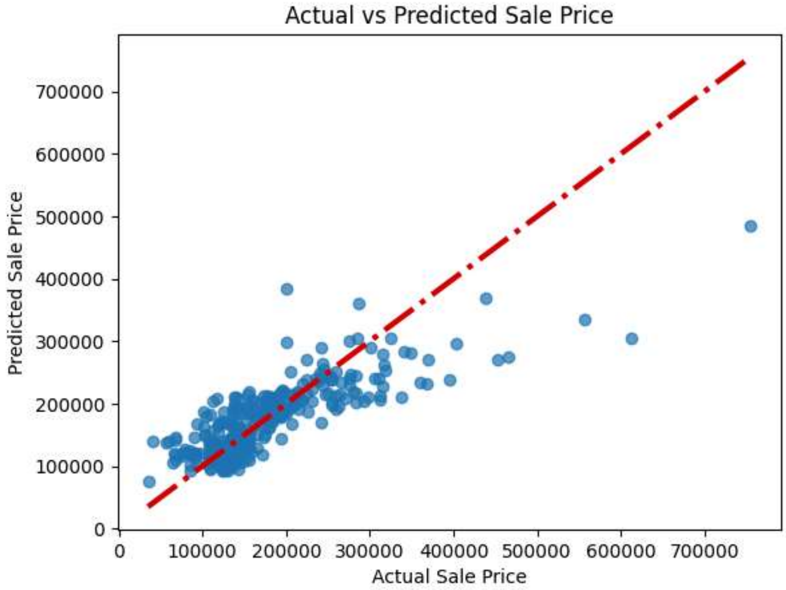
```
# Predict on the test set
y_pred = model.predict(X_test)

# Calculate evaluation metrics
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

print("Mean Squared Error (MSE):", mse)
print("R-squared (R2):", r2)
```

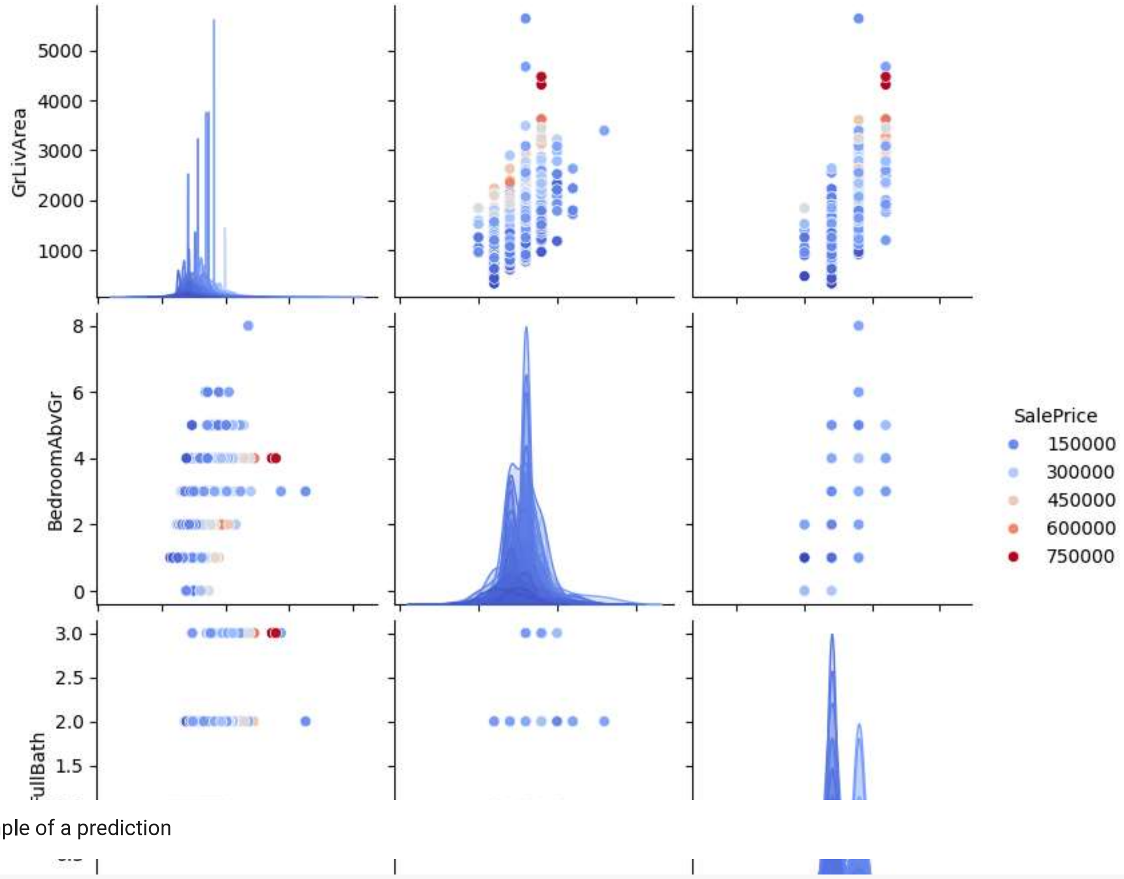
```
Mean Squared Error (MSE): 2806426667.247853
R-squared (R2): 0.6341189942328371
```

```
plt.scatter(y_test, y_pred, alpha=0.7)
plt.xlabel('Actual Sale Price')
plt.ylabel('Predicted Sale Price')
plt.title('Actual vs Predicted Sale Price')
plt.plot([y.min(), y.max()], [y.min(), y.max()], '#D20103', linestyle='-.', linewidth=3)
plt.show()
```




```
plt.figure(figsize=(12, 8))
sns.pairplot(train_[col + ['SalePrice']], hue='SalePrice', palette='coolwarm')
plt.show()
```

```
<Figure size 1200x800 with 0 Axes>
```



Example of a prediction

```
"""Example of a prediction"""

# Get user input for each feature
gr_liv_area = float(input("Enter the square footage (e.g., 1800): "))
bedroom_abv_gr = int(input("Enter the number of bedrooms above grade (e.g., 2): "))
full_bath = int(input("Enter the number of full baths (e.g., 2): "))

# Create the DataFrame from user inputs
ex = pd.DataFrame({
    'GrLivArea': [gr_liv_area],
    'BedroomAbvGr': [bedroom_abv_gr],
    'FullBath': [full_bath]
})
test_prediction = model.predict(ex)
print(f'The price of the house is about ≃ ${test_prediction[0]:,.2f}')

# Prepare the test data and make predictions
X_test = test_[col]
test_predictions = model.predict(X_test)

# Save predictions
submission = pd.DataFrame({'Id': test_['Id'], 'SalePrice': test_predictions})
submission.to_csv('submission.csv', index=False)
```

```
Enter the square footage (e.g., 1800): 2000
Enter the number of bedrooms above grade (e.g., 2): 4
Enter the number of full baths (e.g., 2): 2
The price of the house is about ≃ $213,722.35
```