Task 2

Produce a model to predict house prices. You are welcome to generate new features, scale the data, and split the data into training/testing (i.e. train_test_split) in any way you like. You are also welcome to use the datasets contained in the data folder or other datasets that you find on the internet.

Evaluate your model's accuracy by predicting a test dataset, for example:

On Monday the instructor and TA will provide an unseen set of houses which students will use to repeat their accuracy evaluation. The best models (i.e. lowest RMSE) will win prizes.

We will evaluate the models using a simple mean-squared-error as follows:

In [ ]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from scipy.stats import norm
from scipy import stats
```

In [ ]:
```
df = pd.read_csv('C:/Users/jettr/Dropbox (University of Oregon)/23-24/Spring/Geog 4
```

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

Out[ ]:

	price	bedrooms	bathrooms	sqft_living	sqft_lot	yr_built	lat	long
0	538000	3	2.25	2570	7242	1951	47.7210	-122.319
1	180000	2	1.00	770	10000	1933	47.7379	-122.233
2	604000	4	3.00	1960	5000	1965	47.5208	-122.393
3	510000	3	2.00	1680	8080	1987	47.6168	-122.045
4	1230000	4	4.50	5420	101930	2001	47.6561	-122.005

In [ ]:
```
df.columns
```

Out[ ]:
```
Index(['price', 'bedrooms', 'bathrooms', 'sqft_living', 'sqft_lot', 'yr_built',
       'lat', 'long'],
      dtype='object')
```

In [ ]:
```
df['price'].describe()
```

Out[ ]:
```
count    1.945100e+04
mean     5.404634e+05
std      3.685123e+05
min      7.500000e+04
25%      3.210000e+05
50%      4.500000e+05
75%      6.450000e+05
max      7.700000e+06
Name: price, dtype: float64
```

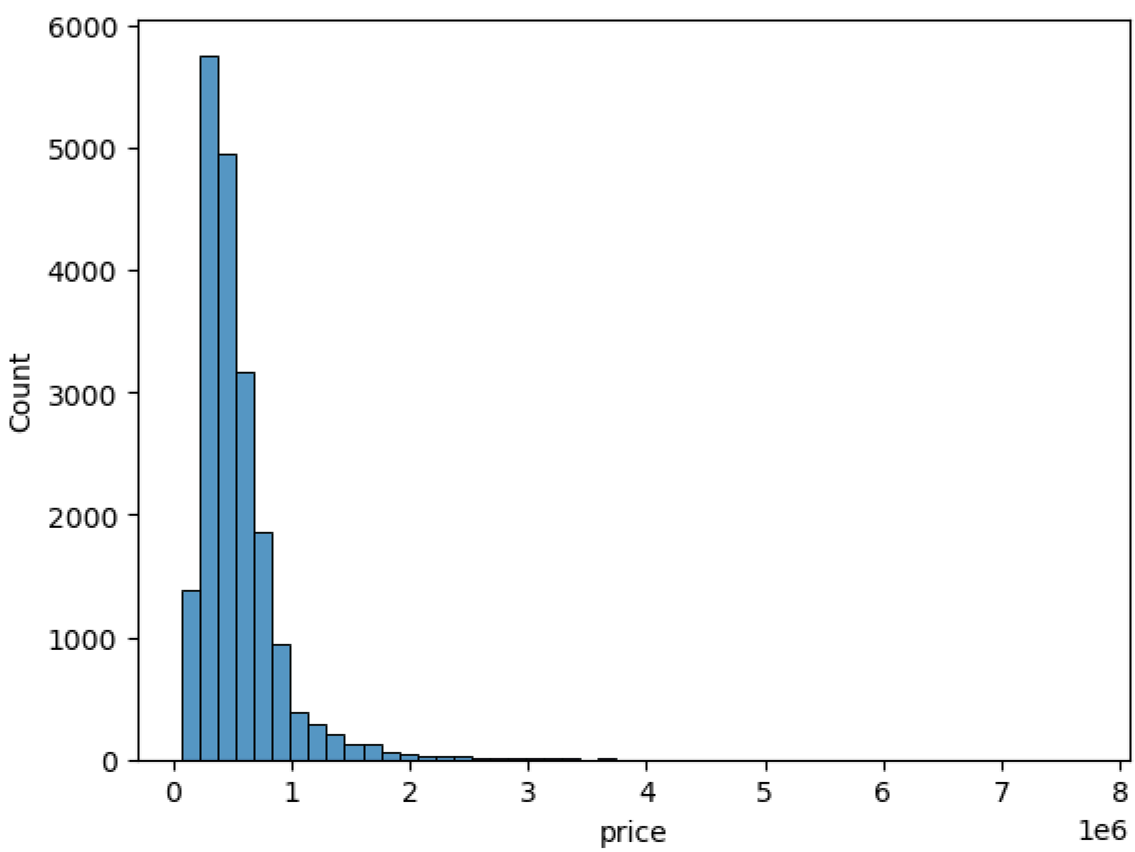
In [ ]:
```
df.isna().any() # Check if any Nan Values
```

Out[ ]:
```
price          False
bedrooms       False
bathrooms      False
sqft_living    False
sqft_lot       False
yr_built       False
lat            False
long           False
dtype: bool
```

In [ ]:
```
sns.histplot(df['price'],bins=50,kde=False) # What does the stats of the data look
```

```
c:\Users\jettr\AppData\Local\anaconda3\envs\skylearn\Lib\site-packages\seaborn\_oldc
ore.py:1119: FutureWarning: use_inf_as_na option is deprecated and will be removed i
n a future version. Convert inf values to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
```

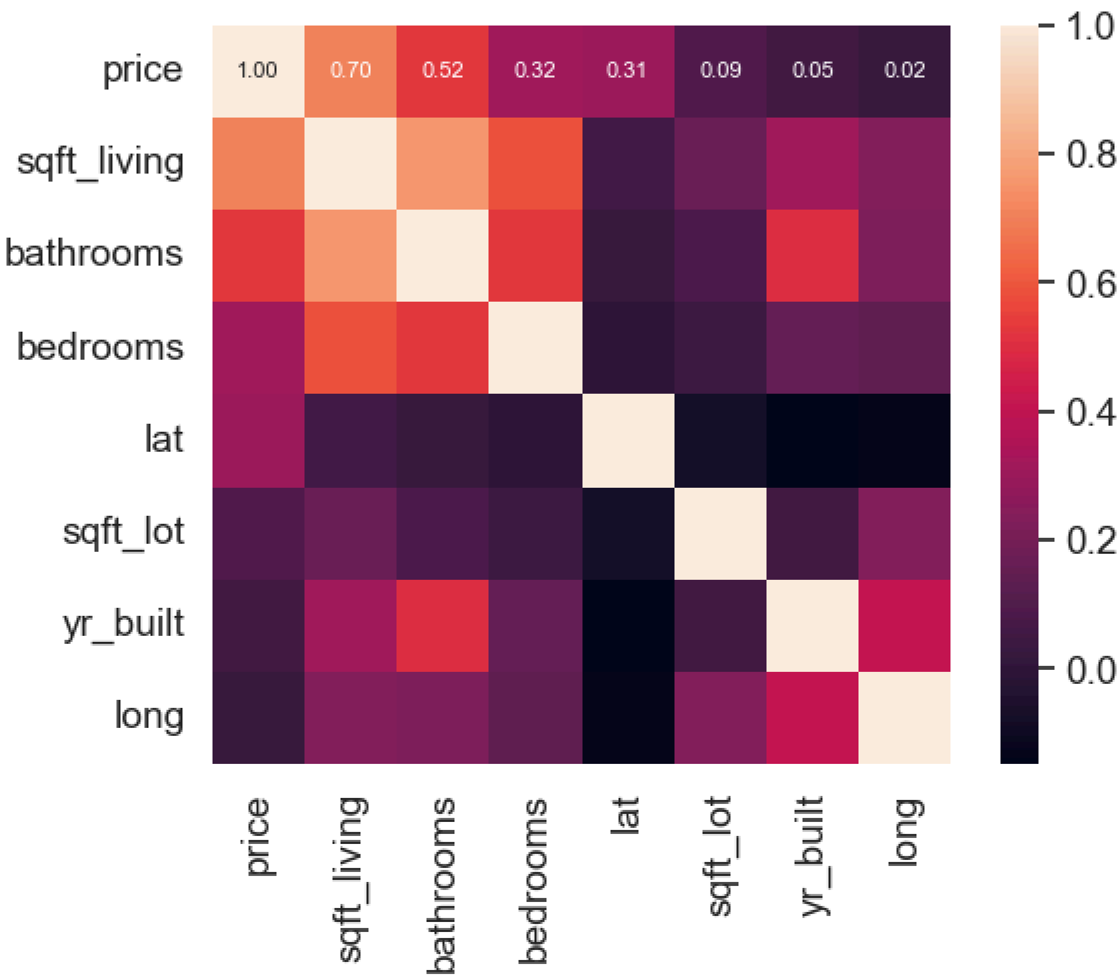
Out[ ]: <Axes: xlabel='price', ylabel='Count'>



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

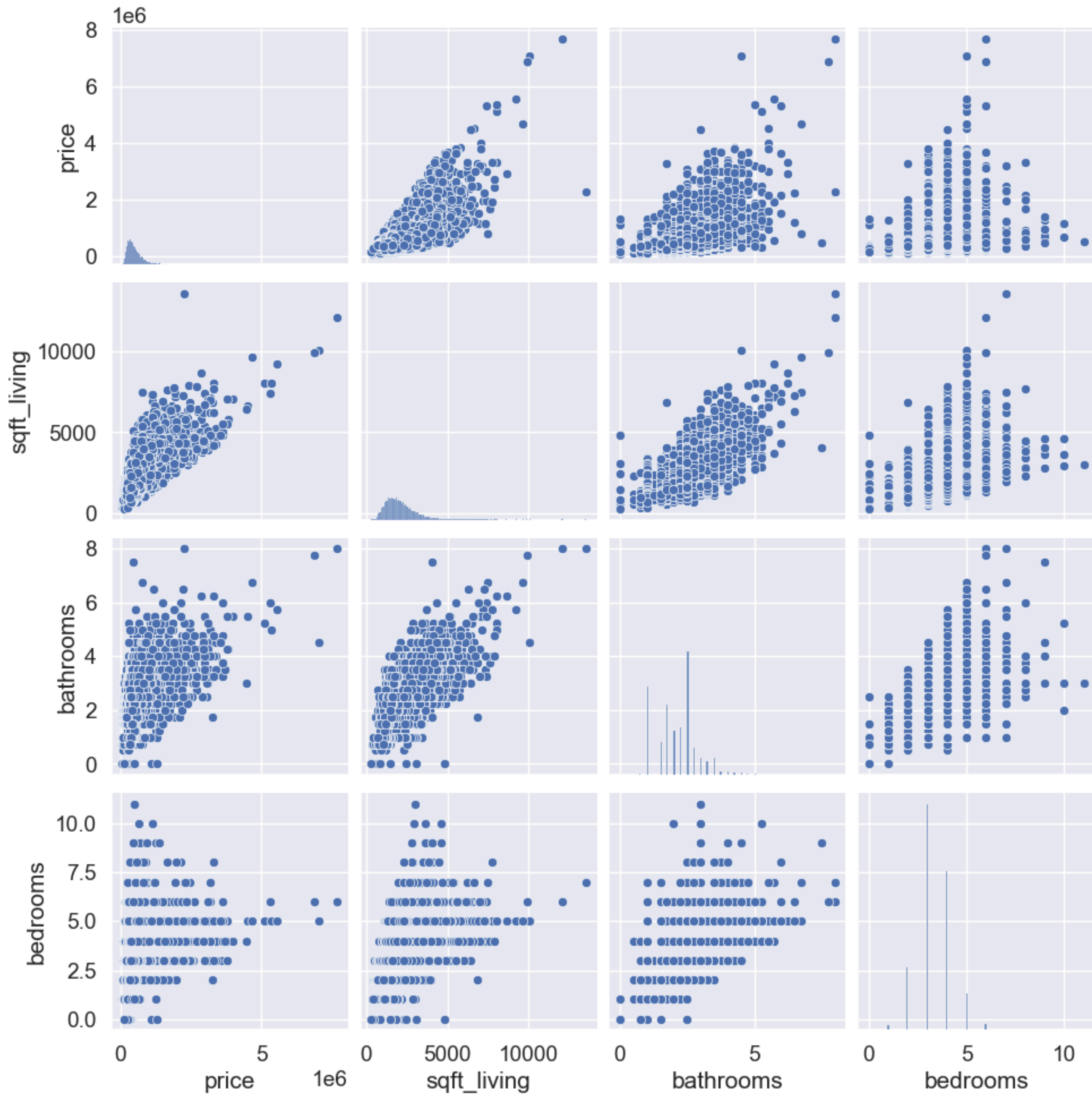
```
Skewness: 4.086029
Kurtosis: 36.072850
```

In [ ]:
```
# Correlation heat map

correlationmatrix = df.corr()

k = 8 #number of variables for heatmap
cols = correlationmatrix.nlargest(k, 'price')['price'].index
cm = np.corrcoef(df[cols].values.T)
sns.set_theme(font_scale=1.25)
hm = sns.heatmap(cm, cbar=True, annot=True, square=True, fmt='.2f', annot_kws={'siz
plt.show()
```



In [ ]:
```
# Replace infinite values with NaNs across the entire DataFrame
# df.replace([np.inf, -np.inf], np.nan, inplace=True)

with pd.option_context('mode.use_inf_as_na',True):
    cols = ['price', 'sqft_living', 'bathrooms', 'bedrooms']
    sns.pairplot(df[cols], height=2.5)
    plt.show()
```

```
C:\Users\jettr\AppData\Local\Temp\ipykernel_14456\3686919224.py:4: FutureWarning: us
e_inf_as_na option is deprecated and will be removed in a future version. Convert in
f values to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na',True):
c:\Users\jettr\AppData\Local\anaconda3\envs\skylearn\Lib\site-packages\seaborn\_oldc
ore.py:1119: FutureWarning: use_inf_as_na option is deprecated and will be removed i
n a future version. Convert inf values to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
c:\Users\jettr\AppData\Local\anaconda3\envs\skylearn\Lib\site-packages\seaborn\_oldc
ore.py:1119: FutureWarning: use_inf_as_na option is deprecated and will be removed i
n a future version. Convert inf values to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
c:\Users\jettr\AppData\Local\anaconda3\envs\skylearn\Lib\site-packages\seaborn\_oldc
ore.py:1119: FutureWarning: use_inf_as_na option is deprecated and will be removed i
n a future version. Convert inf values to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
c:\Users\jettr\AppData\Local\anaconda3\envs\skylearn\Lib\site-packages\seaborn\_oldc
ore.py:1119: FutureWarning: use_inf_as_na option is deprecated and will be removed i
n a future version. Convert inf values to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
```



Normalization

fixing the skewness and kurtosis

In [ ]:
```
from sklearn.preprocessing import StandardScaler

# Convert 'price' column to a numpy array
price_array = df['price'].values[:, np.newaxis]

# Standardize the data using StandardScaler
price_scaled = StandardScaler().fit_transform(price_array)

# Get the low and high ranges of the standardized price
low_range = price_scaled[price_scaled[:, 0].argsort()][:8]
high_range = price_scaled[price_scaled[:, 0].argsort()][-8:]

print('Outer range (low) of the distribution:')
```

```
print(low_range)
print('Outer range (high) of the distribution:')
print(high_range)
```

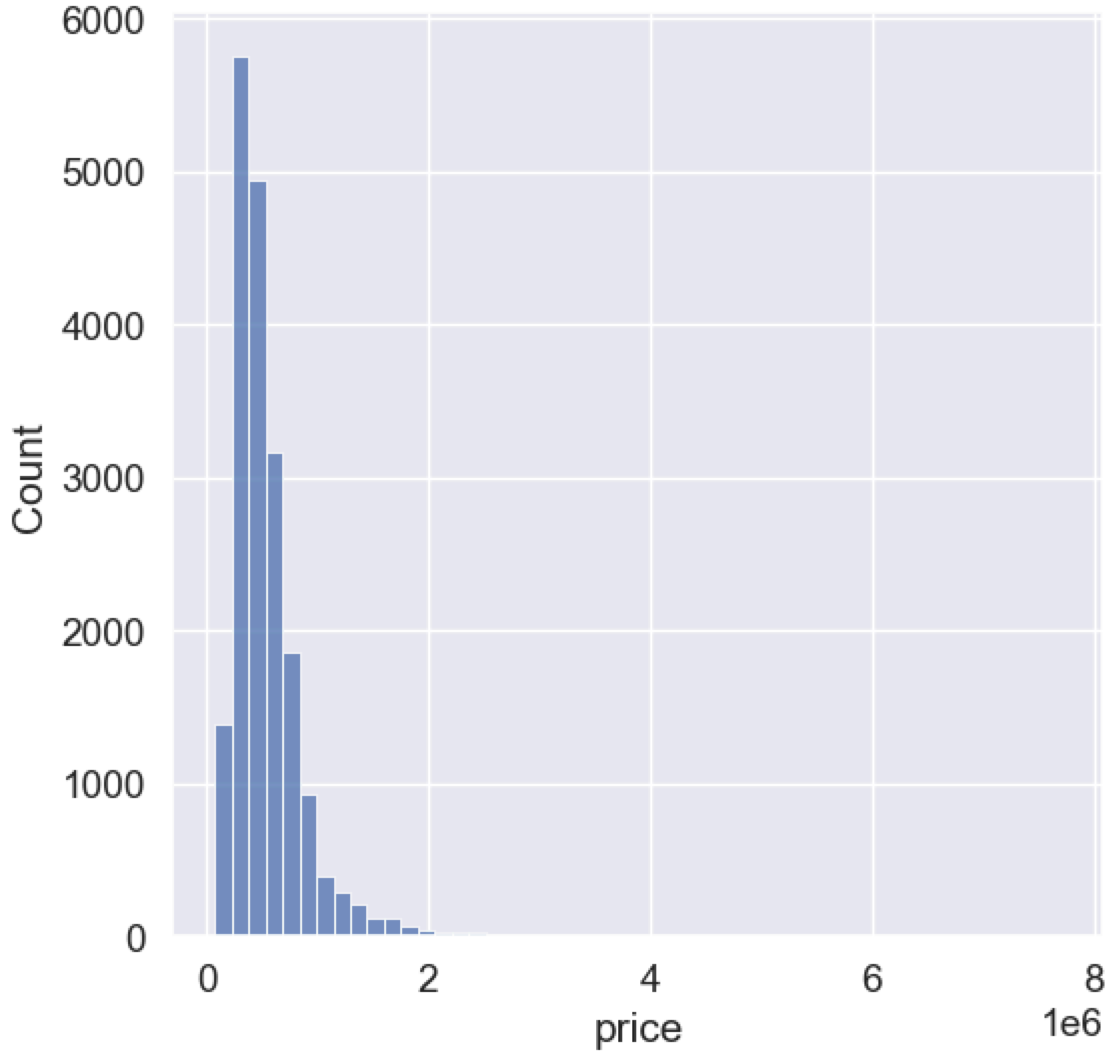
```
Outer range (low) of the distribution:
[[-1.2631202 ]
 [-1.25497915]
 [-1.24955178]
 [-1.2468381 ]
 [-1.24412442]
 [-1.24276758]
 [-1.24141073]
 [-1.23869705]]
Outer range (high) of the distribution:
[[11.20625419]
 [12.4002748 ]
 [12.91587461]
 [13.05155878]
 [13.64856908]
 [17.23063093]
 [17.69195708]
 [19.42871434]]
```

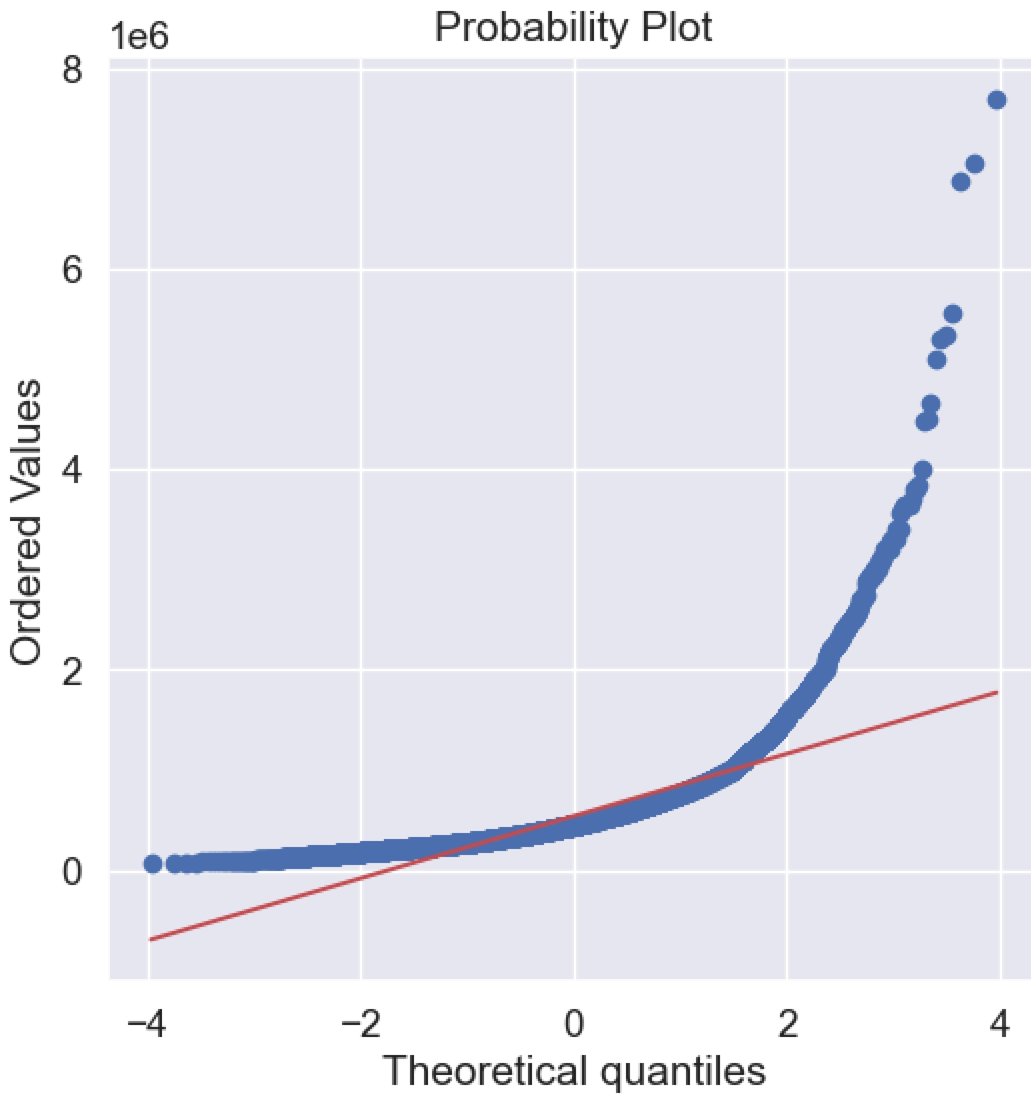
Normalizing the Price

In [ ]:

```
#histogram and normal probability plot
plt.figure(figsize=(6,6))
sns.histplot(df['price'], bins=50, kde=False);

plt.figure(figsize=(6,6))
res = stats.probplot(df['price'], plot=plt) # Probability plot are show to scale of
# axis , while the y axis is thge unscaled quantiles of the data
# Since Our data is skewed and has kurtosis, we can see that the theoretical fit is
```

```
c:\Users\jettr\AppData\Local\anaconda3\envs\skylearn\Lib\site-packages\seaborn\_oldc
ore.py:1119: FutureWarning: use_inf_as_na option is deprecated and will be removed i
n a future version. Convert inf values to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
```

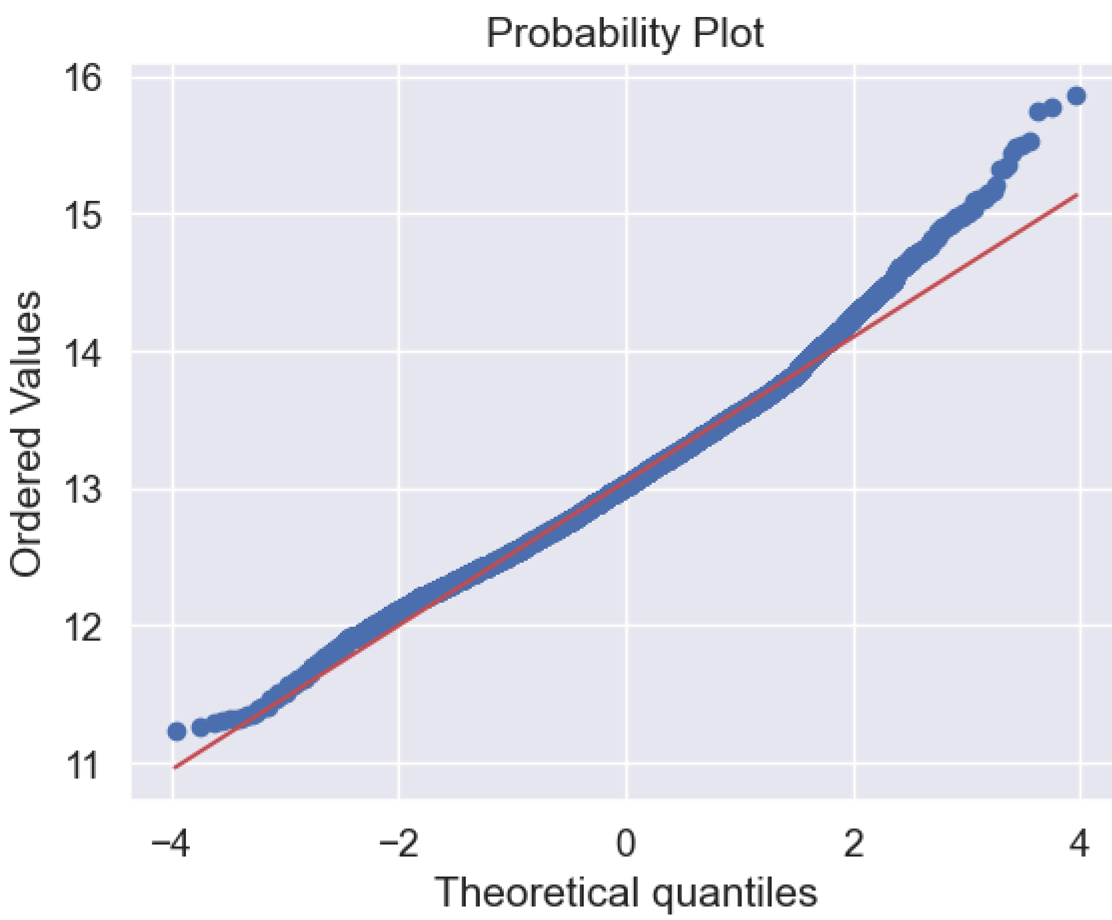

6000
5000
4000
3000
2000
1000
0
Count
0
2
4
6
8
1e6
price



In [ ]:
```
#applying log transformation
df['price'] = np.log(df['price'])

#transformed histogram and normal probability plot
sns.histplot(df['price'],  bins=50, kde=False);
fig = plt.figure()
res = stats.probplot(df['price'], plot=plt)
```

```
c:\Users\jettr\AppData\Local\anaconda3\envs\skylearn\Lib\site-packages\seaborn\_oldc
ore.py:1119: FutureWarning: use_inf_as_na option is deprecated and will be removed i
n a future version. Convert inf values to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
```

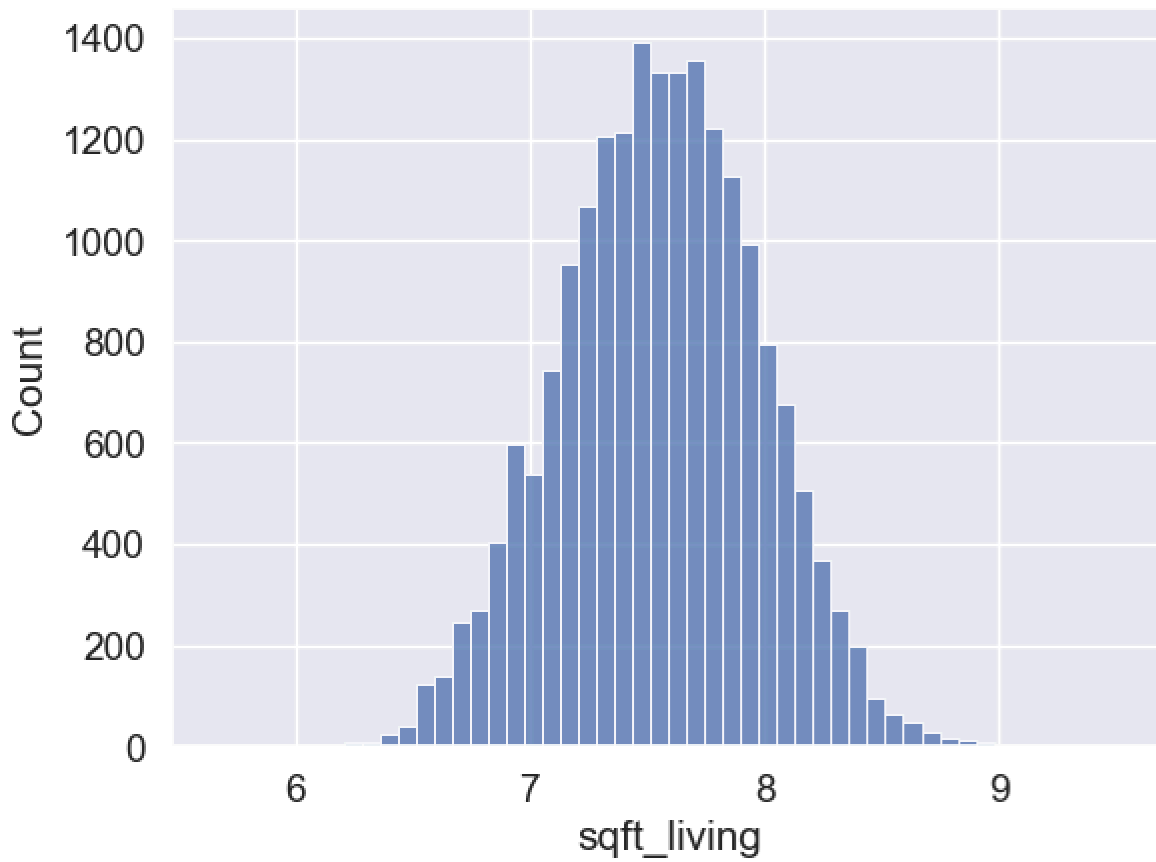

1400
1200
1000
800
600
400
200
0
Count
11
12
13
14
15
16
price

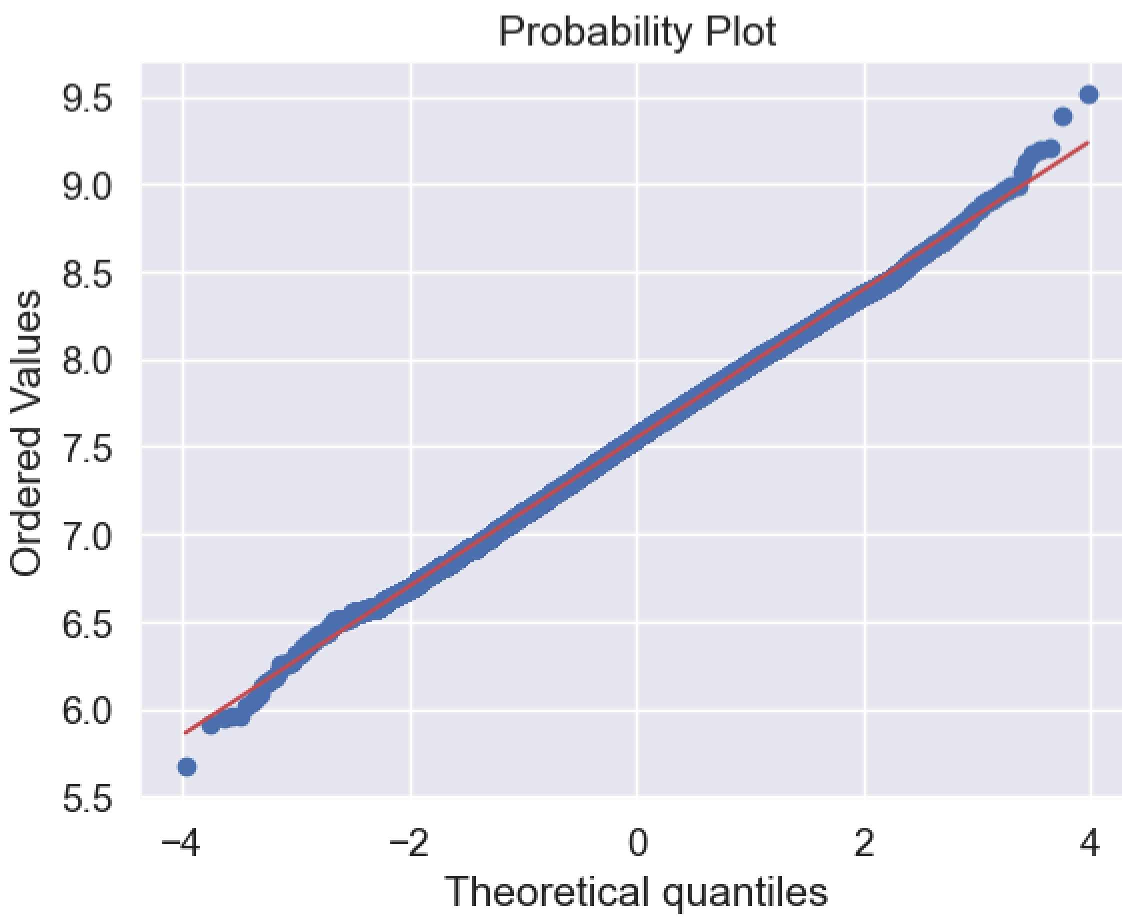

Probability Plot
16
15
14
13
12
11
Ordered Values
−4
−2
0
2
4
Theoretical quantiles

Sqaure Foot Living Area

In [ ]:
```
#data transformation
df['sqft_living'] = np.log(df['sqft_living']) # Everything is on a logarithmic scal
#transformed histogram and normal probability plot
sns.histplot(df['sqft_living'], bins=50, kde=False)
fig = plt.figure()
res = stats.probplot(df['sqft_living'], plot=plt)
```

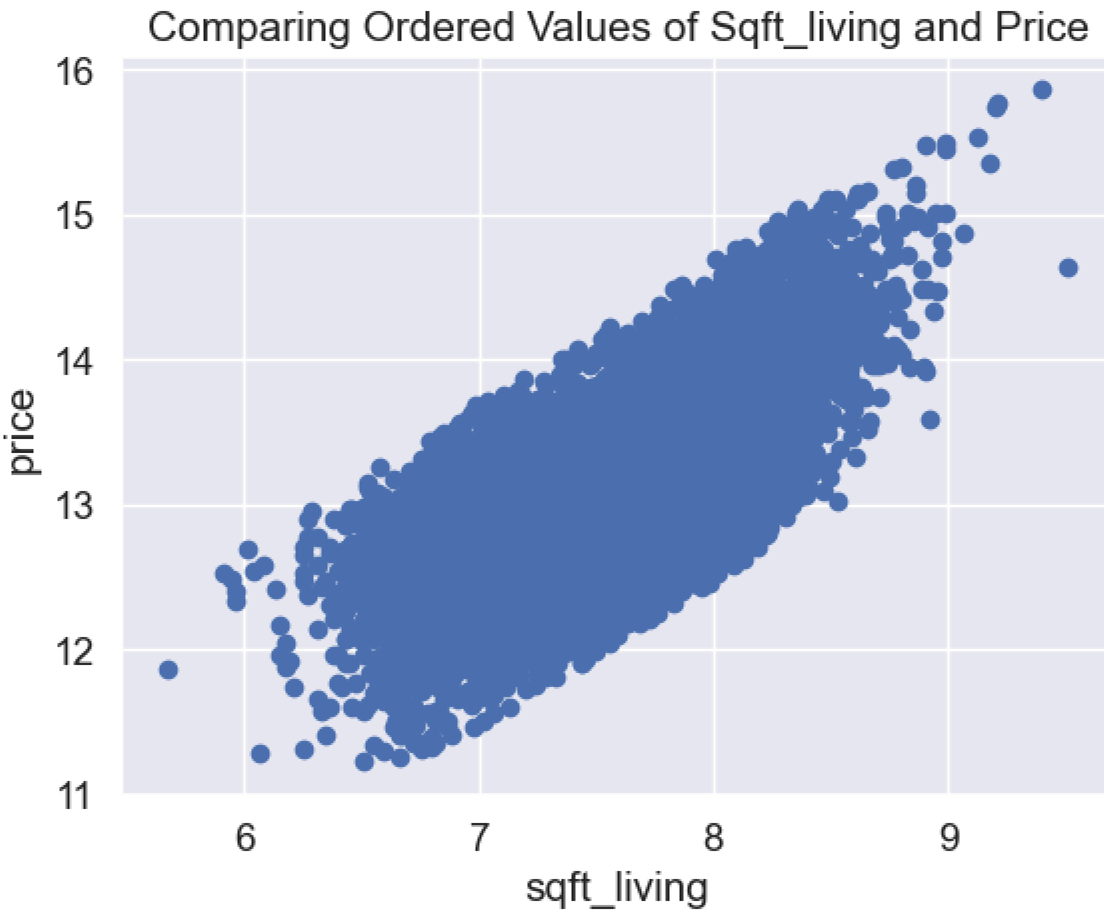
```
c:\Users\jettr\AppData\Local\anaconda3\envs\skylearn\Lib\site-packages\seaborn\_oldc
ore.py:1119: FutureWarning: use_inf_as_na option is deprecated and will be removed i
n a future version. Convert inf values to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
```





In [ ]:
```
#scatter plot
plt.figure()
plt.scatter(df['sqft_living'], df['price'])
plt.title('Comparing Ordered Values of Sqft_living and Price')
plt.xlabel('sqft_living')
plt.ylabel('price')
```

Out[ ]:
```
Text(0, 0.5, 'price')
```



Now to start building the model

In [ ]:
```
feature_cols = ['bedrooms','bathrooms', 'sqft_living', 'sqft_lot', 'yr_built' , 'la
Y = df.price.values
X=df[feature_cols]
print(X)
```

```
       bedrooms  bathrooms  sqft_living  sqft_lot  yr_built      lat     long
0             3       2.25     7.851661      7242      1951  47.7210 -122.319
1             2       1.00     6.646391     10000      1933  47.7379 -122.233
2             4       3.00     7.580700      5000      1965  47.5208 -122.393
3             3       2.00     7.426549      8080      1987  47.6168 -122.045
4             4       4.50     8.597851    101930      2001  47.6561 -122.005
...         ...        ...          ...       ...       ...      ...      ...
19446         3       2.50     7.177782      1294      2008  47.5773 -122.409
19447         3       2.50     7.333023      1131      2009  47.6993 -122.346
19448         4       2.50     7.745003      5813      2014  47.5107 -122.362
19449         3       2.50     7.377759      2388      2004  47.5345 -122.069
19450         2       0.75     6.927558      1076      2008  47.5941 -122.299

[19451 rows x 7 columns]
```

In [ ]:
```
from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test = train_test_split(X, Y, random_state=3)
```

In [ ]:
```
from sklearn.linear_model import LinearRegression
regressor = LinearRegression()
regressor.fit(x_train, y_train)
```

Out[ ]:
```
▾ LinearRegression

LinearRegression()
```

In [ ]:
```
accuracy = regressor.score(x_test, y_test)
"Accuracy: {}%".format(int(round(accuracy * 100)))
```

Out[ ]:
```
'Accuracy: 67%'
```

Elastic net , Gradient Boosting

In [ ]:
```
from sklearn import ensemble, tree, linear_model
from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.metrics import r2_score, mean_squared_error
from sklearn.utils import shuffle
```

In [ ]:
```
# For accurate scoring
def get_score(prediction, lables):
    print('R2: {}'.format(r2_score(prediction, lables)))
    print('RMSE: {}'.format(np.sqrt(mean_squared_error(prediction, lables))))
```

In [ ]:
```
def train_test(estimator, x_trn, x_tst, y_trn, y_tst):
    prediction_train = estimator.predict(x_trn)
    # Printing estimator
    print(estimator)
    # Printing train scores
    get_score(prediction_train, y_trn)
    prediction_test = estimator.predict(x_tst)
    # Printing test scores
    print("Test")
    get_score(prediction_test, y_tst)
```

In [ ]:
```
ENSTest = linear_model.ElasticNetCV(alphas=[0.0001, 0.0005, 0.001, 0.01, 0.1, 1, 10
```

In [ ]:
```
train_test(ENSTest, x_train, x_test, y_train, y_test)
```

```
ElasticNetCV(alphas=[0.0001, 0.0005, 0.001, 0.01, 0.1, 1, 10],
             l1_ratio=[0.01, 0.1, 0.5, 0.9, 0.99], max_iter=5000)
R2: 0.5053161847729462
RMSE: 0.3036214164404515
Test
R2: 0.5084171444175738
RMSE: 0.3039445952574138
```

In [ ]:
```
# Average R2 score and standart deviation of 5-fold cross-validation
scores = cross_val_score(ENSTest, x_test, y_test, cv=5)
print("Accuracy: %0.2f (+/- %0.2f)" % (scores.mean(), scores.std() * 2))
```

```
Accuracy: 0.66 (+/- 0.04)
```

Gradient Boosting

In [ ]:
```
GBest = ensemble.GradientBoostingRegressor(n_estimators=3000, learning_rate=0.05, m
                                           min_samples_leaf=15, min_samples_split=1
train_test(GBest, x_train, x_test, y_train, y_test)
# GBest is our model
```

```
GradientBoostingRegressor(learning_rate=0.05, loss='huber', max_features='sqrt',
                          min_samples_leaf=15, min_samples_split=10,
                          n_estimators=3000)
R2: 0.8897092314600881
RMSE: 0.16263534050008505
Test
R2: 0.8321417419971294
RMSE: 0.19761901378537788
```

In [ ]:
```
# Average R2 score and standart deviation of 5-fold cross-validation
scores = cross_val_score(GBest, x_test, y_test, cv=5)
print("Accuracy: %0.2f (+/- %0.2f)" % (scores.mean(), scores.std() * 2))
```

```
Accuracy: 0.84 (+/- 0.02)
```

Example of Exporting the model / Giving it new data

In [ ]:
```
import joblib

# Export the trained model
joblib.dump(GBest, 'gradient_boosting_model.pkl')

# Load the model
loaded_model = joblib.load('gradient_boosting_model.pkl')

# Example of making predictions on new data
new_data = X

# Make predictions
predictions = loaded_model.predict(new_data)
print(predictions)
```

```
[13.22627379 12.52886619 13.15626204 ... 12.94851487 12.89793911
 12.71566805]
```

In [ ]:
```
# Convert the log-transformed predictions back to the original scale
predicted_prices = np.exp(predictions)

print(predicted_prices)
```

```
[554750.52491003 276196.04346194 517239.8871778  ... 420212.10422498
 399488.03998298 332923.51147055]
```

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

Out[ ]:

	price	bedrooms	bathrooms	sqft_living	sqft_lot	yr_built	lat	long	predi
0	538000.0	3	2.25	7.851661	7242	1951	47.7210	-122.319	5.5
1	180000.0	2	1.00	6.646391	10000	1933	47.7379	-122.233	2.7
2	604000.0	4	3.00	7.580700	5000	1965	47.5208	-122.393	5.1
3	510000.0	3	2.00	7.426549	8080	1987	47.6168	-122.045	4.9
4	1230000.0	4	4.50	8.597851	101930	2001	47.6561	-122.005	1.4

In [ ]:

```
# Make a copy of the original DataFrame
df_copy = df.copy()

# Append the predicted_prices column to the copied DataFrame
df_copy['predicted_price'] = predicted_prices


# Calculate the difference between the predicted prices and the actual prices
df_copy['prediction_error'] = (df_copy['predicted_price'] - df_copy['price']) / df_

# Display the updated copied DataFrame
print(df_copy)
```

```
           price  bedrooms  bathrooms  sqft_living  sqft_lot  yr_built  \
0       538000.0         3       2.25     7.851661      7242      1951
1       180000.0         2       1.00     6.646391     10000      1933
2       604000.0         4       3.00     7.580700      5000      1965
3       510000.0         3       2.00     7.426549      8080      1987
4      1230000.0         4       4.50     8.597851    101930      2001
...          ...       ...        ...          ...       ...       ...
19446   475000.0         3       2.50     7.177782      1294      2008
19447   360000.0         3       2.50     7.333023      1131      2009
19448   400000.0         4       2.50     7.745003      5813      2014
19449   400000.0         3       2.50     7.377759      2388      2004
19450   325000.0         2       0.75     6.927558      1076      2008

           lat     long  predicted_price  prediction_error
0      47.7210 -122.319     5.547505e+05          3.113480
1      47.7379 -122.233     2.761960e+05         53.442246
2      47.5208 -122.393     5.172399e+05        -14.364257
3      47.6168 -122.045     4.967462e+05         -2.598788
4      47.6561 -122.005     1.400105e+06         13.829652
...        ...      ...              ...               ...
19446  47.5773 -122.409     4.829458e+05          1.672806
19447  47.6993 -122.346     3.837433e+05          6.595371
19448  47.5107 -122.362     4.202121e+05          5.053026
19449  47.5345 -122.069     3.994880e+05         -0.127990
19450  47.5941 -122.299     3.329235e+05          2.438004

[19451 rows x 10 columns]
```