```
# Data Analysis and Wrangling
import pandas as pd
import numpy as np
import random as rnd

# Visualization
import seaborn as sns
import matplotlib.pyplot as plt
import plotly.express as px
import plotly.graph_objects as go

# scaling and train test split
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MinMaxScaler

# creating a model
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Activation
from tensorflow.keras.optimizers import Adam

# evaluation on test data
from sklearn.metrics import mean_squared_error,mean_absolute_error,explained_variance_score
from sklearn.metrics import classification_report,confusion_matrix


import warnings
warnings.filterwarnings("ignore")


data = pd.read_csv('task1_kc_house_data.csv')


data.shape
```

```
(21613, 21)
```

```
data.head()
```

	id	date	price	bedrooms	bathrooms	sqft_living	sqft_lot
0	7129300520	20141013T000000	221900.0	3	1.00	1180	5650
1	6414100192	20141209T000000	538000.0	3	2.25	2570	7242
2	5631500400	20150225T000000	180000.0	2	1.00	770	10000
3	2487200875	20141209T000000	604000.0	4	3.00	1960	5000
4	1954400510	20150218T000000	510000.0	3	2.00	1680	8080

5 rows × 21 columns

```
data.tail()
```

	id	date	price	bedrooms	bathrooms	sqft_living	sqft_
21608	263000018	20140521T000000	360000.0	3	2.50	1530	1
21609	6600060120	20150223T000000	400000.0	4	2.50	2310	5
21610	1523300141	20140623T000000	402101.0	2	0.75	1020	1
21611	291310100	20150116T000000	400000.0	3	2.50	1600	2
21612	1523300157	20141015T000000	325000.0	2	0.75	1020	1

5 rows × 21 columns

```
print(data.columns.values)
```

```
['id' 'date' 'price' 'bedrooms' 'bathrooms' 'sqft_living' 'sqft_lot'
 'floors' 'waterfront' 'view' 'condition' 'grade' 'sqft_above'
 'sqft_basement' 'yr_built' 'yr_renovated' 'zipcode' 'lat' 'long'
 'sqft_living15' 'sqft_lot15']
```

```
data.isnull().any()
```

```
id             False
date           False
price          False
bedrooms       False
```

```
bathrooms        False
sqft_living      False
sqft_lot         False
floors           False
waterfront       False
view             False
condition        False
grade            False
sqft_above       False
sqft_basement    False
yr_built         False
yr_renovated     False
zipcode          False
lat              False
long             False
sqft_living15    False
sqft_lot15       False
dtype: bool
```

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 21613 entries, 0 to 21612
Data columns (total 21 columns):
 #   Column         Non-Null Count  Dtype  
---  ------         --------------  -----  
 0   id             21613 non-null  int64  
 1   date           21613 non-null  object 
 2   price          21613 non-null  float64
 3   bedrooms       21613 non-null  int64  
 4   bathrooms      21613 non-null  float64
 5   sqft_living    21613 non-null  int64  
 6   sqft_lot       21613 non-null  int64  
 7   floors         21613 non-null  float64
 8   waterfront     21613 non-null  int64  
 9   view           21613 non-null  int64  
 10  condition      21613 non-null  int64  
 11  grade          21613 non-null  int64  
 12  sqft_above     21613 non-null  int64  
 13  sqft_basement  21613 non-null  int64  
 14  yr_built       21613 non-null  int64  
 15  yr_renovated   21613 non-null  int64  
 16  zipcode        21613 non-null  int64  
 17  lat            21613 non-null  float64
 18  long           21613 non-null  float64
 19  sqft_living15  21613 non-null  int64  
 20  sqft_lot15     21613 non-null  int64  
dtypes: float64(5), int64(15), object(1)
memory usage: 3.5+ MB
```

```
data = data.drop('id', axis=1)
data = data.drop('zipcode',axis=1)
```

```
data['date'] = pd.to_datetime(data['date'])
data['year'] = data['date'].dt.year
data['month'] = data['date'].dt.month
```

```
data = data.drop("date",axis=1)
```

```
data.sample()
```

	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfront	view
19735	484259.0	4	2.75	2790	5000	2.0	0	0

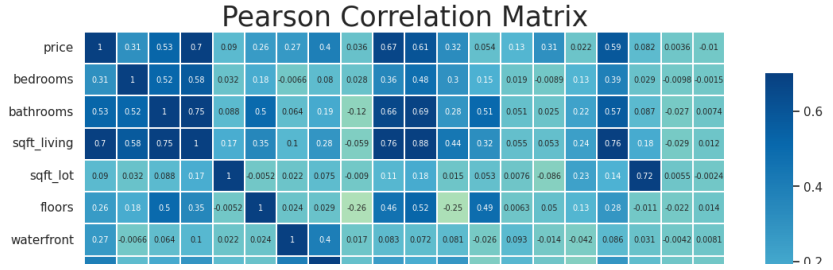
```
desc_stats = data.describe().T

# Create a custom style for the DataFrame visualization
def custom_style(val):
    color = '#606ff2'
    return f'background-color: {color}; color: white'

# Apply the custom style to the whole DataFrame
styled_desc_stats = desc_stats.style.applymap(custom_style)

# Apply background gradient to 'std' column using 'PuBu' colormap
styled_desc_stats = styled_desc_stats.background_gradient(subset=['std'], cmap='PuBu')

# Apply background gradient to '50%' column using 'PuBu' colormap
styled_desc_stats = styled_desc_stats.background_gradient(subset=['50%'], cmap='PuBu')
```

```
# Display the styled descriptive statistics DataFrame
styled_desc_stats
```

	count	mean	std	min	25%
price	21613.000000	540182.158793	367362.231718	75000.000000	321950.000000
bedrooms	21613.000000	3.370842	0.930062	0.000000	3.000000
bathrooms	21613.000000	2.114757	0.770163	0.000000	1.750000
sqft_living	21613.000000	2079.899736	918.440897	290.000000	1427.000000
sqft_lot	21613.000000	15106.967566	41420.511515	520.000000	5040.000000
floors	21613.000000	1.494309	0.539989	1.000000	1.000000
waterfront	21613.000000	0.007542	0.086517	0.000000	0.000000
view	21613.000000	0.234303	0.766318	0.000000	0.000000
condition	21613.000000	3.409430	0.650743	1.000000	3.000000
grade	21613.000000	7.656873	1.175459	1.000000	7.000000
sqft_above	21613.000000	1788.390691	828.090978	290.000000	1190.000000
sqft_basement	21613.000000	291.509045	442.575043	0.000000	0.000000
yr_built	21613.000000	1971.005136	29.373411	1900.000000	1951.000000
yr_renovated	21613.000000	84.402258	401.679240	0.000000	0.000000
lat	21613.000000	47.560053	0.138564	47.155900	47.471000
long	21613.000000	-122.213896	0.140828	-122.519000	-122.328000
sqft_living15	21613.000000	1986.552492	685.391304	399.000000	1490.000000
sqft_lot15	21613.000000	12768.455652	27304.179631	651.000000	5100.000000
year	21613.000000	2014.322954	0.467616	2014.000000	2014.000000
month	21613.000000	6.574423	3.115308	1.000000	4.000000

```
sns.set(style="whitegrid", font_scale=1)

plt.figure(figsize=(13,13))
plt.title('Pearson Correlation Matrix',fontsize=25)
sns.heatmap(data.corr(),linewidths=0.25,vmax=0.7,square=True,cmap="GnBu",linecolor='w',
            annot=True, annot_kws={"size":7}, cbar_kws={"shrink": .7})
```

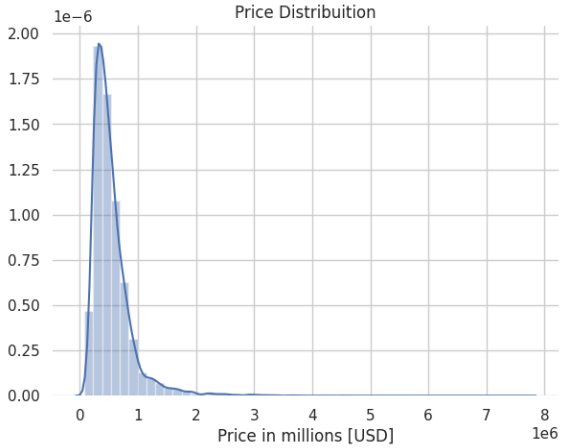
```
<Axes: title={'center': 'Pearson Correlation Matrix'}>
```

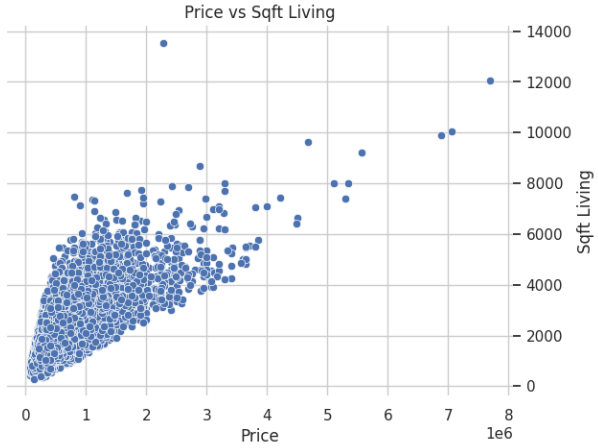



```
price_corr = data.corr()['price'].sort_values(ascending=False)
print(price_corr)
```

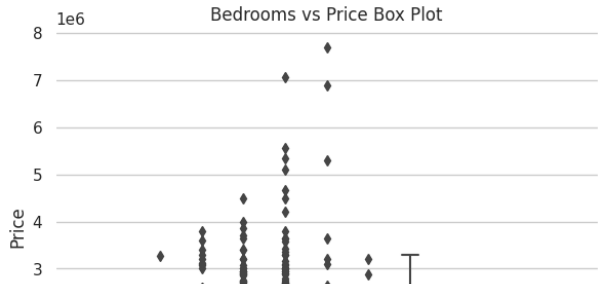
```
price            1.000000
sqft_living      0.702044
grade            0.667463
sqft_above       0.605566
sqft_living15    0.585374
bathrooms        0.525134
view             0.397346
sqft_basement    0.323837
bedrooms         0.308338
lat              0.306919
waterfront       0.266331
floors           0.256786
yr_renovated     0.126442
sqft_lot         0.089655
sqft_lot15       0.082456
yr_built         0.053982
condition        0.036392
long             0.021571
year             0.003554
month           -0.010053
Name: price, dtype: float64
```

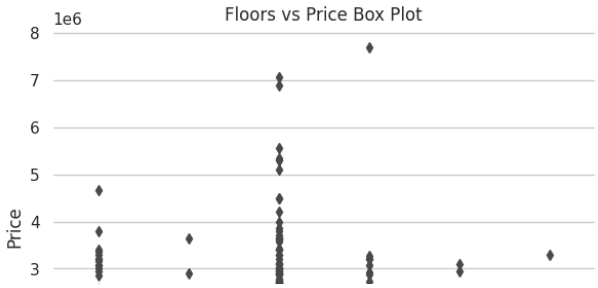
```
f, axes = plt.subplots(1, 2,figsize=(15,5))
sns.distplot(data['price'], ax=axes[0])
sns.scatterplot(x='price',y='sqft_living', data=data, ax=axes[1])
sns.despine(bottom=True, left=True)
axes[0].set(xlabel='Price in millions [USD]', ylabel='', title='Price Distribuition')
axes[1].set(xlabel='Price', ylabel='Sqft Living', title='Price vs Sqft Living')
axes[1].yaxis.set_label_position("right")
axes[1].yaxis.tick_right()
```



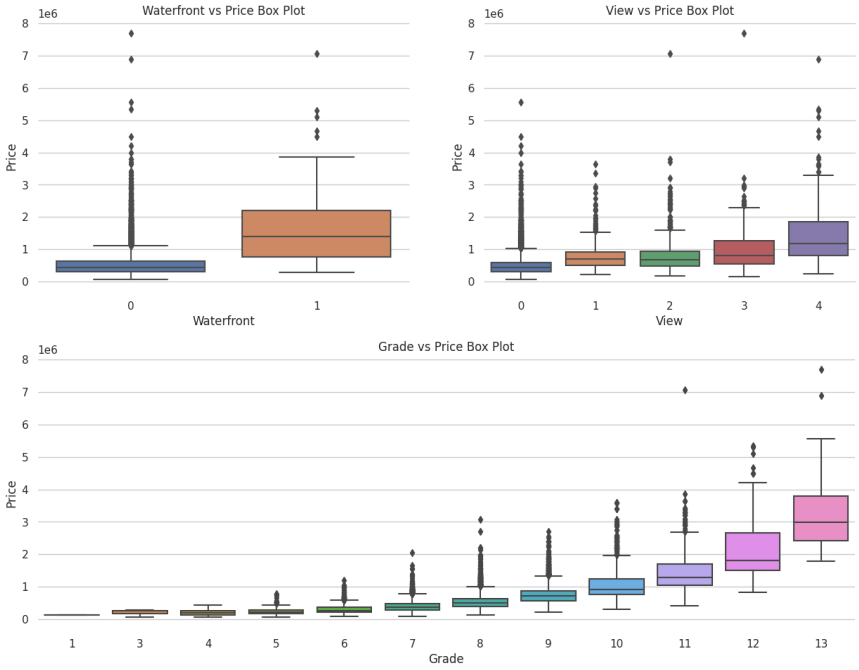



```
sns.set(style="whitegrid", font_scale=1)

f, axes = plt.subplots(1, 2,figsize=(15,5))
sns.boxplot(x=data['bedrooms'],y=data['price'], ax=axes[0])
sns.boxplot(x=data['floors'],y=data['price'], ax=axes[1])
sns.despine(bottom=True, left=True)
axes[0].set(xlabel='Bedrooms', ylabel='Price', title='Bedrooms vs Price Box Plot')
axes[1].set(xlabel='Floors', ylabel='Price', title='Floors vs Price Box Plot');
```

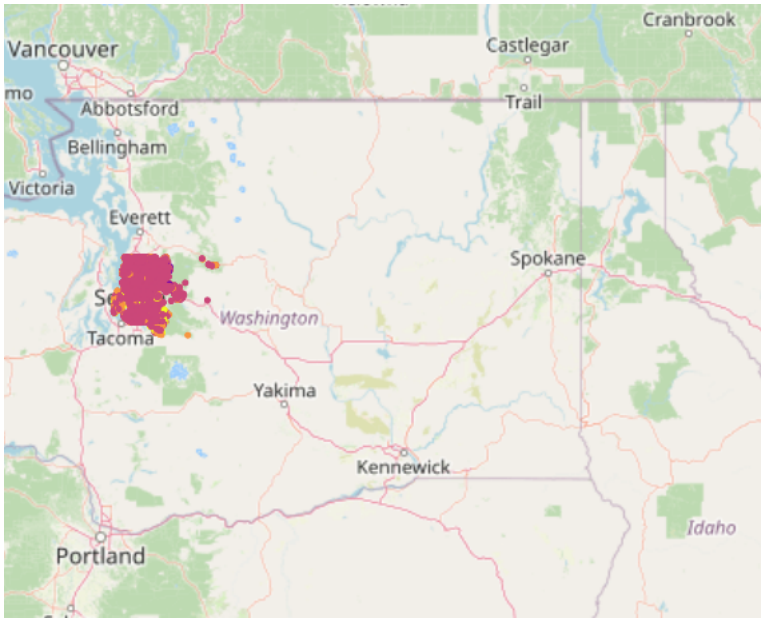





```
f, axes = plt.subplots(1, 2,figsize=(15,5))
sns.boxplot(x=data['waterfront'],y=data['price'], ax=axes[0])
sns.boxplot(x=data['view'],y=data['price'], ax=axes[1])
sns.despine(left=True, bottom=True)
axes[0].set(xlabel='Waterfront', ylabel='Price', title='Waterfront vs Price Box Plot')
axes[1].set(xlabel='View', ylabel='Price', title='View vs Price Box Plot')

f, axe = plt.subplots(1, 1,figsize=(15,5))
sns.boxplot(x=data['grade'],y=data['price'], ax=axe)
sns.despine(left=True, bottom=True)
axe.set(xlabel='Grade', ylabel='Price', title='Grade vs Price Box Plot');
```

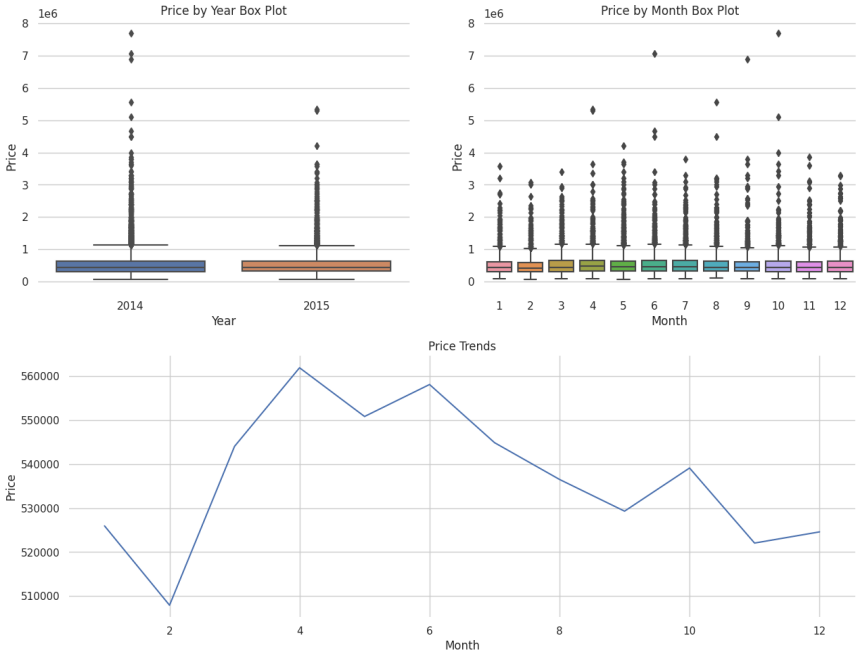



```
fig = px.scatter_mapbox(
    data,  # Our DataFrame
    lat="lat",
    lon= "long",
    color="condition",
    center={"lat": data["lat"].mean(), "lon": data["long"].mean()},  # Center map based on data mean
    width=800,  # Width of map
    height=600,  # Height of map
    hover_data=["price"],  # Display price when hovering mouse over house
)

fig.update_layout(mapbox_style="open-street-map")

fig.show()
```




```
# Create the 3D scatter plot
fig = px.scatter_3d(
    data,
    x="lat",
    y="long",
    z="price",  # Using "price" as the z-axis variable
     # Color code based on house condition
    labels={"long": "longitude", "lat": "latitude", "price": "price"},
    width=700,
    height=600,
)

# Refine formatting
fig.update_traces(
    marker={"size": 4, "line": {"width": 2, "color": "DarkSlateGrey"}},
    selector={"mode": "markers"},
)

# Display the 3D scatter plot
fig.show()
```

```
f, axes = plt.subplots(1, 2,figsize=(15,5))
sns.boxplot(x='year',y='price',data=data, ax=axes[0])
sns.boxplot(x='month',y='price',data=data, ax=axes[1])
sns.despine(left=True, bottom=True)
axes[0].set(xlabel='Year', ylabel='Price', title='Price by Year Box Plot')
axes[1].set(xlabel='Month', ylabel='Price', title='Price by Month Box Plot')

f, axe = plt.subplots(1, 1,figsize=(15,5))
data.groupby('month').mean()['price'].plot()
sns.despine(left=True, bottom=True)
axe.set(xlabel='Month', ylabel='Price', title='Price Trends');
```




```
# Features
X = data.drop('price',axis=1)

# Label
y = data['price']

# Split
X_train, X_test, y_train, y_test = train_test_split(X,y,test_size=0.3,random_state=101)


print(X_train.shape)
print(X_test.shape)
print(y_train.shape)
print(y_test.shape)
```

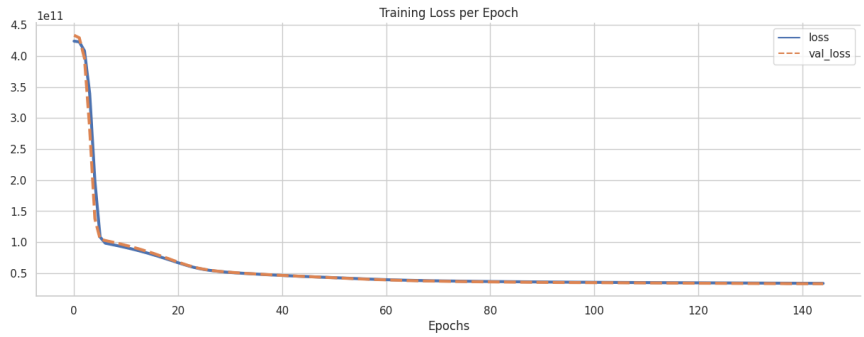
```
(15129, 19)
(6484, 19)
(15129,)
(6484,)
```

```
scaler = MinMaxScaler()

# fit and transfrom
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# everything has been scaled between 1 and 0
print('Max: ',X_train.max())
print('Min: ', X_train.min())
```

```
Max:  1.0000000000000002
Min:  0.0
```

```
model = Sequential()

# input layer
model.add(Dense(19,activation='relu'))

# hidden layers
model.add(Dense(19,activation='relu'))
model.add(Dense(19,activation='relu'))
model.add(Dense(19,activation='relu'))

# output layer
model.add(Dense(1))

model.compile(optimizer='adam',loss='mse')
```

```
losses = pd.DataFrame(model.history.history)

plt.figure(figsize=(15,5))
sns.lineplot(data=losses,lw=3)
plt.xlabel('Epochs')
plt.ylabel('')
plt.title('Training Loss per Epoch')
sns.despine()
```

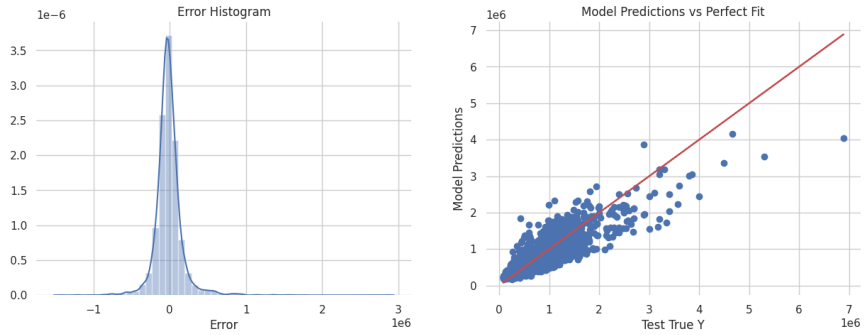



```
# predictions on the test set
predictions = model.predict(X_test)

print('MAE: ',mean_absolute_error(y_test,predictions))
print('MSE: ',mean_squared_error(y_test,predictions))
print('RMSE: ',np.sqrt(mean_squared_error(y_test,predictions)))
print('Variance Regression Score: ',explained_variance_score(y_test,predictions))

print('\n\nDescriptive Statistics:\n',data['price'].describe())
```

```
MAE:  113806.87193717227
MSE:  33008129732.137974
RMSE:  181681.39621914504
Variance Regression Score:  0.7660677934667841


Descriptive Statistics:
 count    2.161300e+04
mean     5.401822e+05
std      3.673622e+05
min      7.500000e+04
25%      3.219500e+05
50%      4.500000e+05
75%      6.450000e+05
```

```
max      7.700000e+06
Name: price, dtype: float64
```

```
f, axes = plt.subplots(1, 2,figsize=(15,5))

# Our model predictions
plt.scatter(y_test,predictions)

# Perfect predictions
plt.plot(y_test,y_test,'r')

errors = y_test.values.reshape(6484, 1) - predictions
sns.distplot(errors, ax=axes[0])

sns.despine(left=True, bottom=True)
axes[0].set(xlabel='Error', ylabel='', title='Error Histogram')
axes[1].set(xlabel='Test True Y', ylabel='Model Predictions', title='Model Predictions vs Perfect Fit');
```




```
# fueatures of new house
single_house = data.drop('price',axis=1).iloc[0]
print(f'Features of new house:\n{single_house}')

# reshape the numpy array and scale the features
single_house = scaler.transform(single_house.values.reshape(-1, 19))

# run the model and get the price prediction
print('\nPrediction Price:',model.predict(single_house)[0,0])

# original price
print('\nOriginal Price:',data.iloc[0]['price'])
```

```
Features of new house:
bedrooms            3.0000
bathrooms           1.0000
sqft_living      1180.0000
sqft_lot         5650.0000
floors              1.0000
waterfront          0.0000
view                0.0000
condition           3.0000
grade               7.0000
sqft_above       1180.0000
sqft_basement       0.0000
yr_built         1955.0000
yr_renovated        0.0000
lat                47.5112
long             -122.2570
sqft_living15    1340.0000
sqft_lot15       5650.0000
year             2014.0000
month              10.0000
Name: 0, dtype: float64
1/1 [==============================] - 0s 23ms/step

Prediction Price: 277800.78

Original Price: 221900.0
```