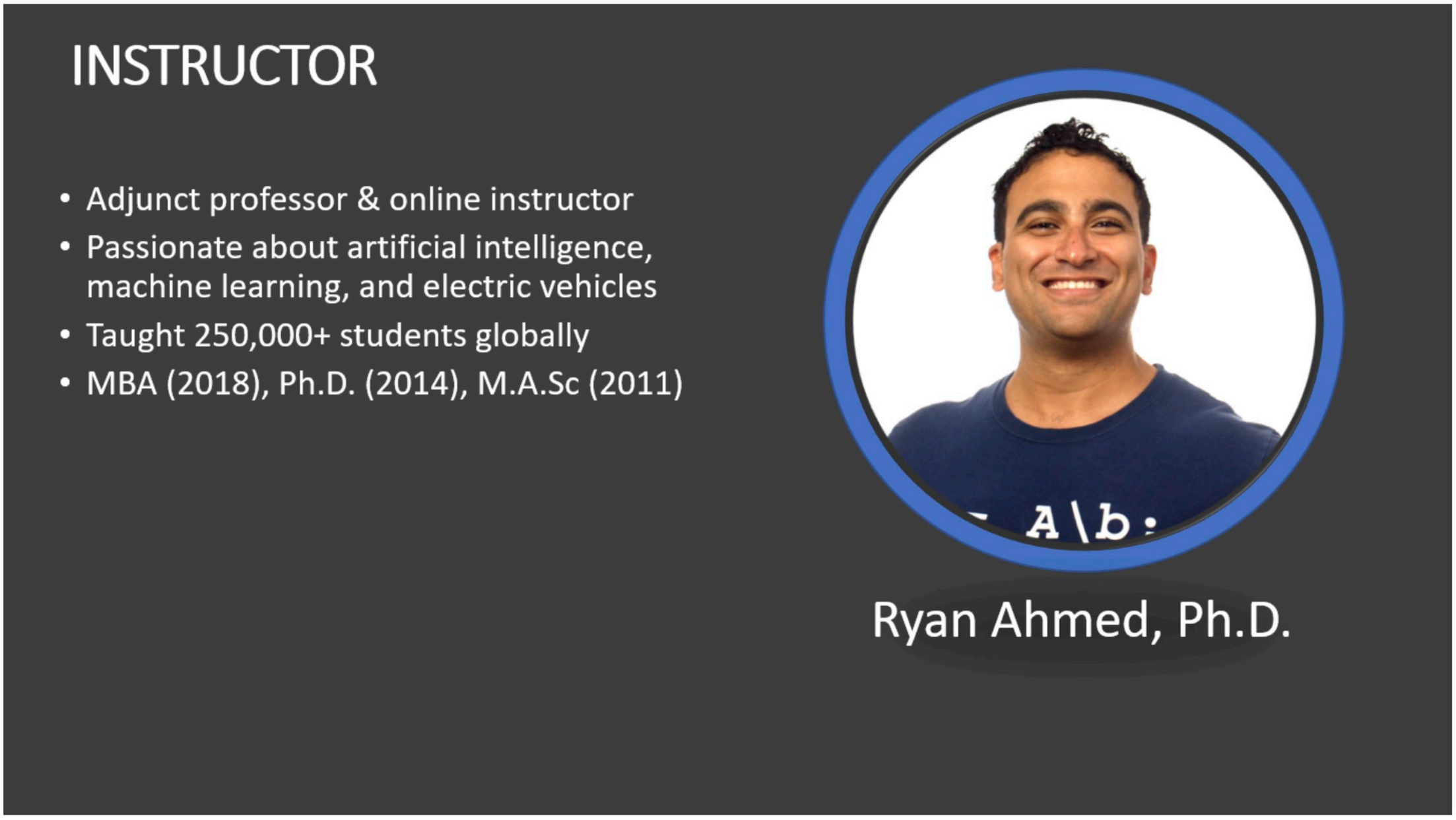
TASK #1: UNDERSTAND THE PROBLEM STATEMENT AND BUSINESS CASE

- In this project, we will predict real estate prices using artificial neural networks.
- Dataset includes house sale prices for King County in USA and homes that are sold in the time period between May, 2014 and May, 2015.
- Columns:
  - ida: notation for a house
  - date: Date house was sold
  - price: Price is prediction target
  - bedrooms: Number of Bedrooms/House
  - bathrooms: Number of bathrooms/House
  - sqft_living: square footage of the home
  - qft_lot: square footage of the lot
  - floors: Total floors (levels) in house
  - waterfront: House which has a view to a waterfront
  - view: Has been viewed

- Data Source: https://www.kaggle.com/harlfoxem/housesalesprediction
- Image Source: https://pxhere.com/en/photo/1440167

- Columns:
  - condition: How good the condition is
  - grade: overall grade given to the housing unit, based on King County grading system
  - sqft_abovesquare: footage of house apart from basement
  - sqft_basement: square footage of the basement
  - yr_built: Built Year
  - yr_renovated: Year when house was renovated
  - zipcode: zip
  - lat: Latitude coordinate
  - long: Longitude coordinate
  - sqft_living15: Living room area in 2015(implies some renovations)
  - sqft_lot15: lot Size area in 2015 (implies some renovations)

- *Data Source: https://www.kaggle.com/harlfoxem/housesalesprediction*



TASK #2: IMPORT LIBRARIES AND DATASETS

In [3]:
```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt


# from jupyterthemes import jtplot
# jtplot.style(theme='monokai', context='notebook', ticks=True, grid=False)
# # setting the style of the notebook to be monokai theme
# # this line of code is important to ensure that we are able to see the x and y axes clearly
# # If you don't run this code line, you will notice that the xlabel and ylabel on any plot is black on black and it
```

In [4]:
```
house_df = pd.read_csv('realestate_prices.csv', encoding = 'ISO-8859-1')
```

In [5]:
```
house_df
```

Out[5]:

	id	date	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfront	view	...	grade	sqft_above	sqft_basement	yr
0	7129300520	20141013T000000	221900.0	3	1.00	1180	5650	1.0	0	0	...	7	1180	0	
1	6414100192	20141209T000000	538000.0	3	2.25	2570	7242	2.0	0	0	...	7	2170	400	
2	5631500400	20150225T000000	180000.0	2	1.00	770	10000	1.0	0	0	...	6	770	0	
3	2487200875	20141209T000000	604000.0	4	3.00	1960	5000	1.0	0	0	...	7	1050	910	
4	1954400510	20150218T000000	510000.0	3	2.00	1680	8080	1.0	0	0	...	8	1680	0	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
21608	263000018	20140521T000000	360000.0	3	2.50	1530	1131	3.0	0	0	...	8	1530	0	
21609	6600060120	20150223T000000	400000.0	4	2.50	2310	5813	2.0	0	0	...	8	2310	0	
21610	1523300141	20140623T000000	402101.0	2	0.75	1020	1350	2.0	0	0	...	7	1020	0	
21611	291310100	20150116T000000	400000.0	3	2.50	1600	2388	2.0	0	0	...	8	1600	0	

21611	291310100	20150116T000000	400000.0	3	2.50	1600	2388	2.0	0	0	...	8	1600		0
21612	1523300157	20141015T000000	325000.0	2	0.75	1020	1076	2.0	0	0	...	7	1020		0

21613 rows × 21 columns

In [6]:
```
house_df.head(5)
```

Out[6]:

	id	date	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfront	view	...	grade	sqft_above	sqft_basement	yr_buil
0	7129300520	20141013T000000	221900.0	3	1.00	1180	5650	1.0	0	0	...	7	1180	0	195
1	6414100192	20141209T000000	538000.0	3	2.25	2570	7242	2.0	0	0	...	7	2170	400	195
2	5631500400	20150225T000000	180000.0	2	1.00	770	10000	1.0	0	0	...	6	770	0	193
3	2487200875	20141209T000000	604000.0	4	3.00	1960	5000	1.0	0	0	...	7	1050	910	196
4	1954400510	20150218T000000	510000.0	3	2.00	1680	8080	1.0	0	0	...	8	1680	0	198

5 rows × 21 columns

In [7]:
```
house_df.tail(10)
```

Out[7]:

	id	date	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfront	view	...	grade	sqft_above	sqft_basement	y
21603	7852140040	20140825T000000	507250.0	3	2.50	2270	5536	2.0	0	0	...	8	2270	0	
21604	9834201367	20150126T000000	429000.0	3	2.00	1490	1126	3.0	0	0	...	8	1490	0	
21605	3448900210	20141014T000000	610685.0	4	2.50	2520	6023	2.0	0	0	...	9	2520	0	
21606	7936000429	20150326T000000	1007500.0	4	3.50	3510	7200	2.0	0	0	...	9	2600	910	
21607	2997800021	20150219T000000	475000.0	3	2.50	1310	1294	2.0	0	0	...	8	1180	130	
21608	263000018	20140521T000000	360000.0	3	2.50	1530	1131	3.0	0	0	...	8	1530	0	
21609	6600060120	20150223T000000	400000.0	4	2.50	2310	5813	2.0	0	0	...	8	2310	0	
21610	1523300141	20140623T000000	402101.0	2	0.75	1020	1350	2.0	0	0	...	7	1020	0	
21611	291310100	20150116T000000	400000.0	3	2.50	1600	2388	2.0	0	0	...	8	1600	0	
21612	1523300157	20141015T000000	325000.0	2	0.75	1020	1076	2.0	0	0	...	7	1020	0	

10 rows × 21 columns

In [8]:
```
house_df.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  
```

```
 20  sqft_lot15     21613 non-null  int64
dtypes: float64(5), int64(15), object(1)
memory usage: 3.5+ MB
```

PRACTICE OPPORTUNITY #1 [OPTIONAL]:

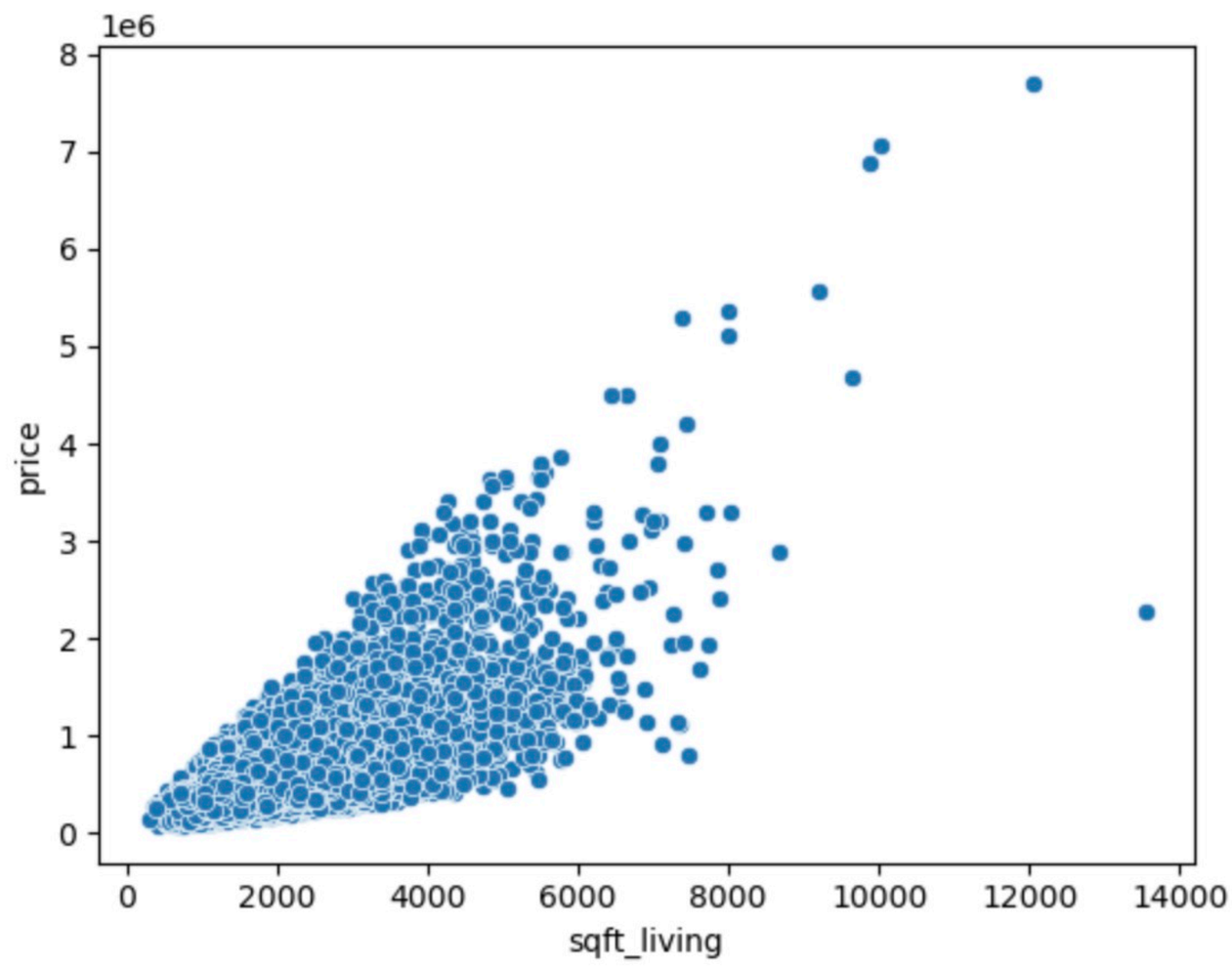
- **What is the average house price?**
- **What is the price of the cheapest house?**
- **What is the average number of bathrooms and bedrooms? round your answer to the lowest value**
- **What is the maximum number of bedrooms?**

In [ ]:

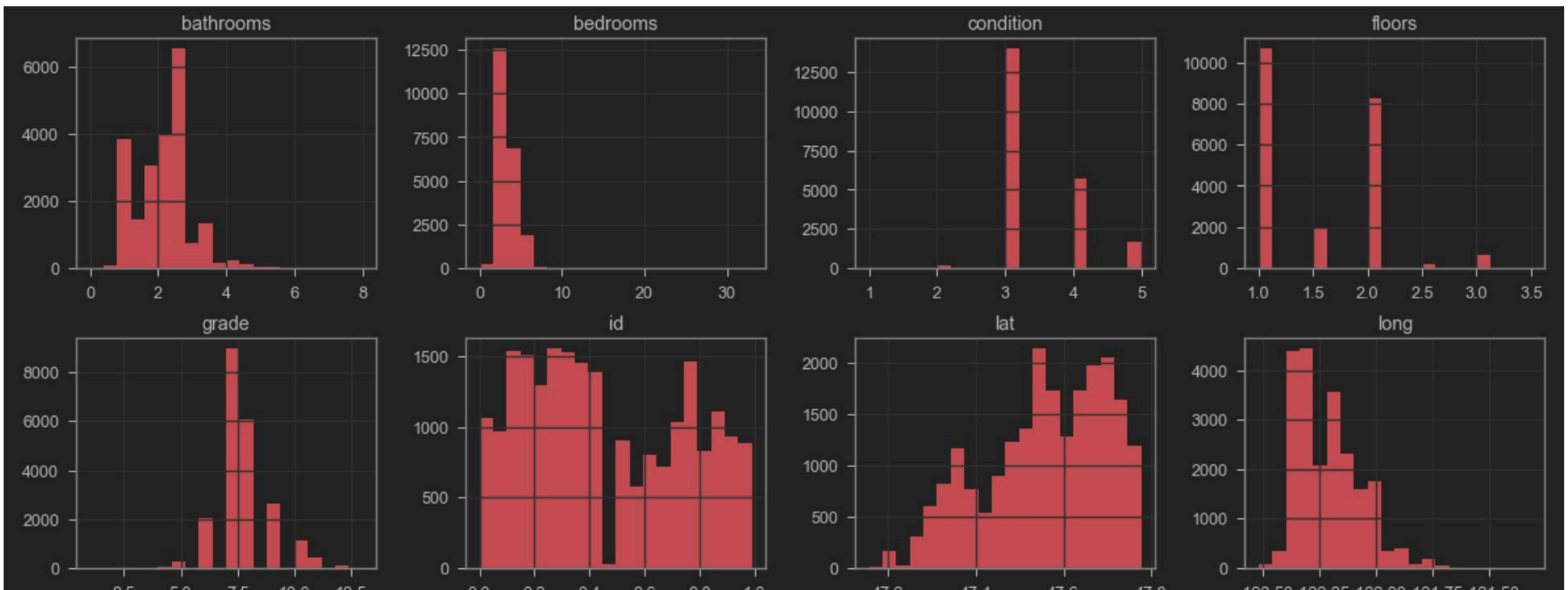
TASK #3: PERFORM DATA VISUALIZATION

In [9]:
```
sns.scatterplot(x = 'sqft_living', y = 'price', data = house_df)
```

Out[9]: <Axes: xlabel='sqft_living', ylabel='price'>

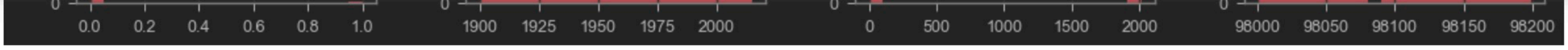


In [8]:
```
house_df.hist(bins = 20, figsize = (20,20), color = 'r');
```



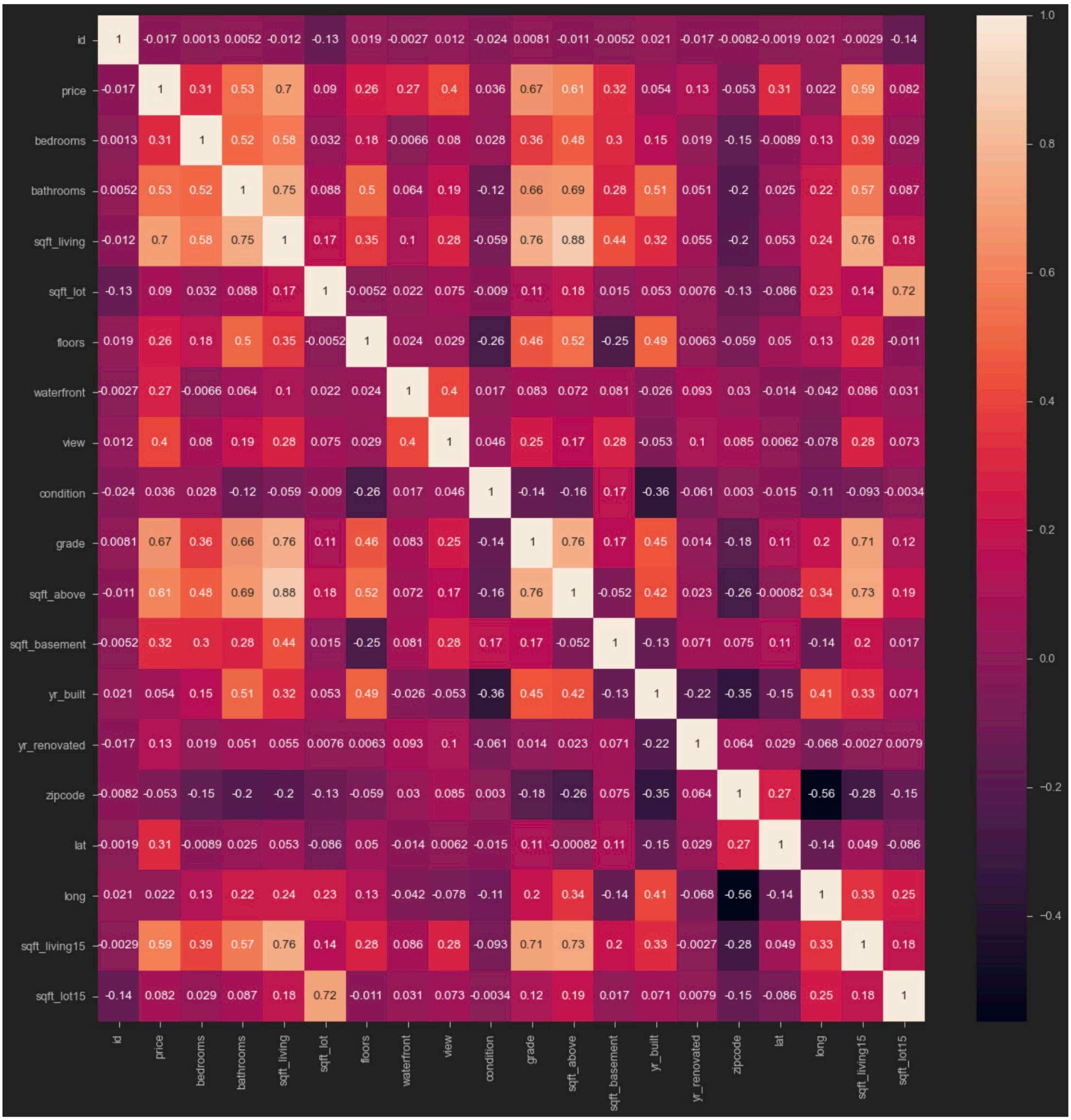
In [9]:
```
f, ax = plt.subplots(figsize = (20, 20))
sns.heatmap(house_df.corr(), annot = True)
```

Out[9]: <matplotlib.axes._subplots.AxesSubplot at 0x20f60957828>



In [9]:
```
f, ax = plt.subplots(figsize = (20, 20))
sns.heatmap(house_df.corr(), annot = True)
```

Out[9]: <matplotlib.axes._subplots.AxesSubplot at 0x20f60957828>



In [10]:
```
house_df_sample = house_df[ ['price', 'bedrooms', 'bathrooms', 'sqft_living', 'sqft_lot', 'sqft_above', 'sqft_baseme
```

In [11]:
```
house_df_sample
```

In [11]:
```
house_df_sample
```

Out[11]:

	price	bedrooms	bathrooms	sqft_living	sqft_lot	sqft_above	sqft_basement	yr_built
0	221900.0	3	1.00	1180	5650	1180	0	1955
1	538000.0	3	2.25	2570	7242	2170	400	1951
2	180000.0	2	1.00	770	10000	770	0	1933
3	604000.0	4	3.00	1960	5000	1050	910	1965
4	510000.0	3	2.00	1680	8080	1680	0	1987
...	...	...	...	...	...	...	...	...
21608	360000.0	3	2.50	1530	1131	1530	0	2009
21609	400000.0	4	2.50	2310	5813	2310	0	2014
21610	402101.0	2	0.75	1020	1350	1020	0	2009
21611	400000.0	3	2.50	1600	2388	1600	0	2004
21612	325000.0	2	0.75	1020	1076	1020	0	2008

21613 rows × 8 columns

PRACTICE OPPORTUNITY #2 [OPTIONAL]:

- **Using Seaborn, plot the pairplot for the features contained in "house_df_sample"**
- **Explore the data and perform sanity check**

In [ ]:
```

```

TASK #4: PERFORM DATA CLEANING AND FEATURE ENGINEERING

In [12]:
```
selected_features = ['bedrooms', 'bathrooms', 'sqft_living', 'sqft_lot', 'floors', 'sqft_above', 'sqft_basement']
```

In [13]:
```
X = house_df[selected_features]
```

In [14]:
```
X
```

Out[14]:

	bedrooms	bathrooms	sqft_living	sqft_lot	floors	sqft_above	sqft_basement
0	3	1.00	1180	5650	1.0	1180	0
1	3	2.25	2570	7242	2.0	2170	400
2	2	1.00	770	10000	1.0	770	0
3	4	3.00	1960	5000	1.0	1050	910
4	3	2.00	1680	8080	1.0	1680	0
...	...	...	...	...	...	...	...
21608	3	2.50	1530	1131	3.0	1530	0
21609	4	2.50	2310	5813	2.0	2310	0
21610	2	0.75	1020	1350	2.0	1020	0
21611	3	2.50	1600	2388	2.0	1600	0
21612	2	0.75	1020	1076	2.0	1020	0

21613 rows × 7 columns

In [15]:
```
y = house_df['price']
```

In [16]:
```
y
```

Out[16]:
```
0        221900.0
1        538000.0
```

In [17]:
```
X.shape
```

Out[17]:
```
(21613, 7)
```

In [18]:
```
y.shape
```

Out[18]:
```
(21613,)
```

In [19]:
```
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()
X_scaled = scaler.fit_transform(X)
```

In [20]:
```
X_scaled
```

Out[20]:
```
array([[0.09090909, 0.125     , 0.06716981, ..., 0.        , 0.09758772,
        0.        ],
       [0.09090909, 0.28125   , 0.17207547, ..., 0.4       , 0.20614035,
        0.08298755],
       [0.06060606, 0.125     , 0.03622642, ..., 0.        , 0.05263158,
        0.        ],
       ...,
       [0.06060606, 0.09375   , 0.05509434, ..., 0.4       , 0.08004386,
        0.        ],
       [0.09090909, 0.3125    , 0.09886792, ..., 0.4       , 0.14364035,
        0.        ],
       [0.06060606, 0.09375   , 0.05509434, ..., 0.4       , 0.08004386,
        0.        ]])
```

In [21]:
```
X_scaled.shape
```

Out[21]:
```
(21613, 7)
```

In [22]:
```
scaler.data_max_
```

Out[22]:
```
array([3.300000e+01, 8.000000e+00, 1.354000e+04, 1.651359e+06,
       3.500000e+00, 9.410000e+03, 4.820000e+03])
```

In [23]:
```
scaler.data_min_
```

Out[23]:
```
array([  0.,   0., 290., 520.,   1., 290.,   0.])
```

In [24]:
```
y = y.values.reshape(-1,1)
```

In [25]:
```
y_scaled = scaler.fit_transform(y)
```

In [26]:
```
y_scaled
```

Out[26]:
```
array([[0.01926557],
       [0.06072131],
       [0.01377049],
       ...,
       [0.04289849],
       [0.04262295],
       [0.03278689]])
```

TASK #5: TRAIN A DEEP LEARNING MODEL WITH LIMITED NUMBER OF FEATURES

In [27]:
```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y_scaled, test_size = 0.25)
```

In [28]:
```
X_train.shape
```

Out[28]:
```
(16209, 7)
```

In [29]:
```
X_test.shape
```

Out[29]:
```
(5404, 7)
```

In [30]:
```
import tensorflow.keras
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense

model = Sequential()
model.add(Dense(100, input_dim = 7, activation = 'relu'))
model.add(Dense(100, activation='relu'))
model.add(Dense(100, activation='relu'))
model.add(Dense(1, activation = 'linear'))
```

In [31]:
```
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense (Dense)                (None, 100)               800
_________________________________________________________________
dense_1 (Dense)              (None, 100)               10100
_________________________________________________________________
dense_2 (Dense)              (None, 100)               10100
_________________________________________________________________
dense_3 (Dense)              (None, 1)                 101
=================================================================
Total params: 21,101
Trainable params: 21,101
Non-trainable params: 0
_________________________________________________________________
```

In [32]:
```
model.compile(optimizer = 'Adam', loss = 'mean_squared_error')
```

In [33]:
```
epochs_hist = model.fit(X_train, y_train, epochs = 100, batch_size = 50, validation_split = 0.2)
```

```
Train on 12967 samples, validate on 3242 samples
Epoch 1/100
12967/12967 [==============================] - 5s 374us/sample - loss: 0.0012 - val_loss: 0.0011
Epoch 2/100
12967/12967 [==============================] - 1s 64us/sample - loss: 0.0011 - val_loss: 0.0010
Epoch 3/100
12967/12967 [==============================] - 1s 56us/sample - loss: 0.0011 - val_loss: 0.0012
Epoch 4/100
12967/12967 [==============================] - 1s 56us/sample - loss: 0.0010 - val_loss: 0.0010
Epoch 5/100
12967/12967 [==============================] - 1s 65us/sample - loss: 0.0010 - val_loss: 0.0010
Epoch 6/100
12967/12967 [==============================] - 1s 62us/sample - loss: 0.0010 - val_loss: 0.0010
Epoch 7/100
12967/12967 [==============================] - 1s 60us/sample - loss: 0.0010 - val_loss: 0.0010
Epoch 8/100
12967/12967 [==============================] - 1s 56us/sample - loss: 0.0010 - val_loss: 0.0011
Epoch 9/100
12967/12967 [==============================] - 1s 72us/sample - loss: 9.9609e-04 - val_loss: 0.0010
Epoch 10/100
12967/12967 [==============================] - 1s 65us/sample - loss: 9.7768e-04 - val_loss: 0.0011
Epoch 11/100
12967/12967 [==============================] - 1s 60us/sample - loss: 9.8589e-04 - val_loss: 0.0011
Epoch 12/100
12967/12967 [==============================] - 1s 55us/sample - loss: 9.8197e-04 - val_loss: 0.0010
Epoch 13/100
12967/12967 [==============================] - 1s 59us/sample - loss: 9.7691e-04 - val_loss: 0.0010
Epoch 14/100
12967/12967 [==============================] - 1s 70us/sample - loss: 9.9276e-04 - val_loss: 0.0011
Epoch 15/100
12967/12967 [==============================] - 1s 61us/sample - loss: 9.7304e-04 - val_loss: 0.0010
Epoch 16/100
12967/12967 [==============================] - 1s 64us/sample - loss: 9.6194e-04 - val_loss: 0.0010
Epoch 17/100
12967/12967 [==============================] - 1s 58us/sample - loss: 9.5018e-04 - val_loss: 0.0011
Epoch 18/100
12967/12967 [==============================] - 1s 62us/sample - loss: 9.4871e-04 - val_loss: 0.0010
Epoch 19/100
12967/12967 [==============================] - 1s 59us/sample - loss: 9.5959e-04 - val_loss: 0.0011
Epoch 20/100
12967/12967 [==============================] - 1s 60us/sample - loss: 9.5551e-04 - val_loss: 0.0011
Epoch 21/100
12967/12967 [==============================] - 1s 60us/sample - loss: 9.6346e-04 - val_loss: 0.0010
Epoch 22/100
12967/12967 [==============================] - 1s 64us/sample - loss: 9.4101e-04 - val_loss: 0.0010
Epoch 23/100
12967/12967 [==============================] - 1s 57us/sample - loss: 9.5983e-04 - val_loss: 0.0010
Epoch 24/100
12967/12967 [==============================] - 1s 64us/sample - loss: 9.5335e-04 - val_loss: 9.9948e-04
Epoch 25/100
12967/12967 [==============================] - 1s 59us/sample - loss: 9.6045e-04 - val_loss: 1.0000e-03
Epoch 26/100
12967/12967 [==============================] - 1s 57us/sample - loss: 9.2578e-04 - val_loss: 0.0010
Epoch 27/100
12967/12967 [==============================] - 1s 57us/sample - loss: 9.6472e-04 - val_loss: 0.0010
Epoch 28/100
12967/12967 [==============================] - 1s 57us/sample - loss: 9.4331e-04 - val_loss: 0.0011
Epoch 29/100
12967/12967 [==============================] - 1s 60us/sample - loss: 9.3168e-04 - val_loss: 0.0010
Epoch 30/100
12967/12967 [==============================] - 1s 60us/sample - loss: 9.6121e-04 - val_loss: 0.0011
Epoch 31/100
12967/12967 [==============================] - 1s 87us/sample - loss: 9.3711e-04 - val_loss: 0.0010
Epoch 32/100
12967/12967 [==============================] - 1s 106us/sample - loss: 9.4819e-04 - val_loss: 0.0011
Epoch 33/100
12967/12967 [==============================] - 1s 70us/sample - loss: 9.2509e-04 - val_loss: 0.0010
Epoch 34/100
12967/12967 [==============================] - 1s 62us/sample - loss: 9.2075e-04 - val_loss: 0.0010
Epoch 35/100
12967/12967 [==============================] - 1s 61us/sample - loss: 9.2888e-04 - val_loss: 0.0010
```

TASK #6: EVALUATE TRAINED DEEP LEARNING MODEL PERFORMANCE

In [34]:
```
epochs_hist.history.keys()
```

Out[34]:
```
dict_keys(['loss', 'val_loss'])
```

In [35]:
```
plt.plot(epochs_hist.history['loss'])
plt.plot(epochs_hist.history['val_loss'])
plt.title('Model Loss Progress During Training')
plt.xlabel('Epoch')
plt.ylabel('Training and Validation Loss')
plt.legend(['Training Loss', 'Validation Loss'])
```

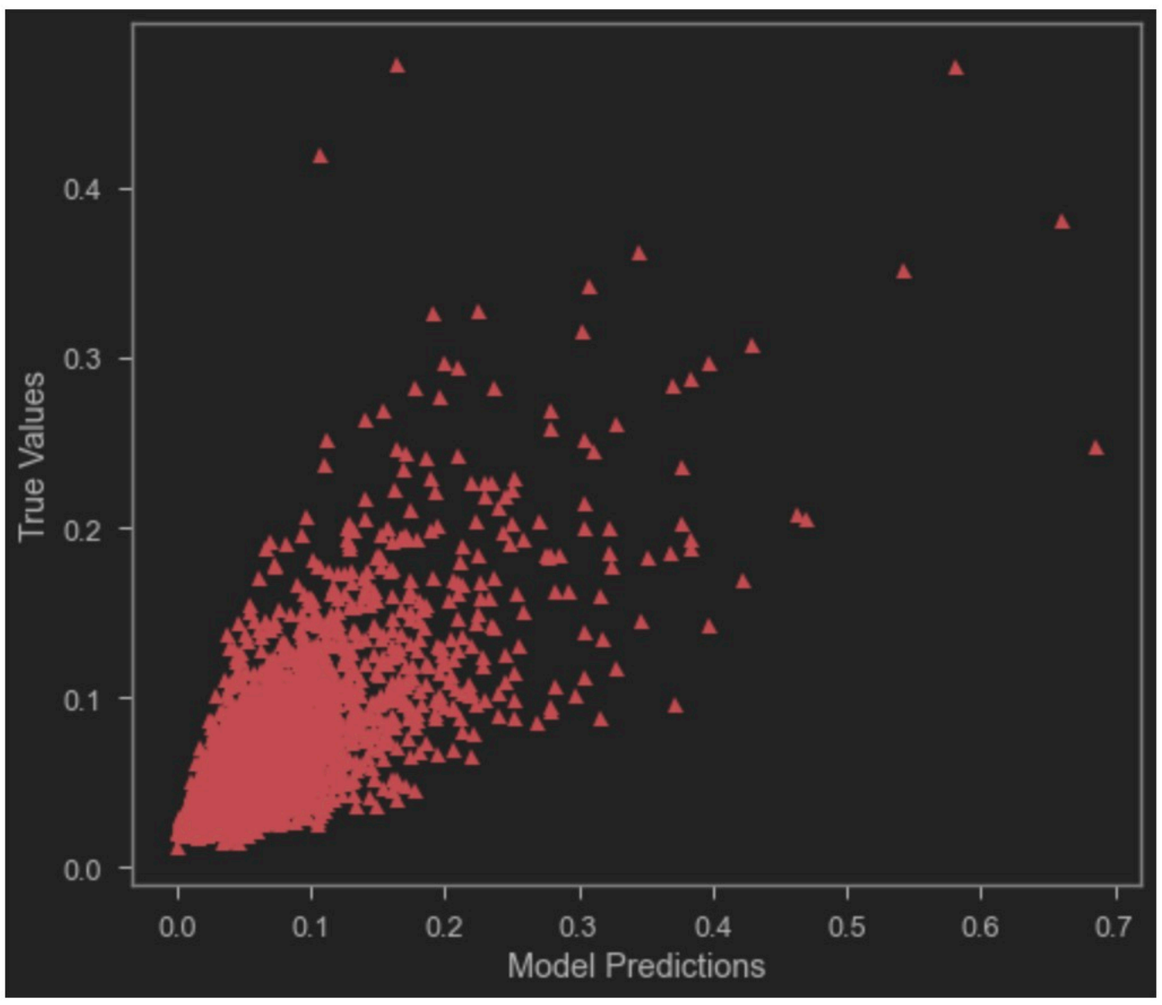
Out[35]:
```
<matplotlib.legend.Legend at 0x20f67421358>
```

In [36]:
```
# 'bedrooms','bathrooms','sqft_living','sqft_lot','floors', 'sqft_above', 'sqft_basement'
X_test_1 = np.array([[ 4, 3, 1960, 5000, 1, 2000, 3000 ]])

scaler_1 = MinMaxScaler()
X_test_scaled_1 = scaler_1.fit_transform(X_test_1)

y_predict_1 = model.predict(X_test_scaled_1)

y_predict_1 = scaler.inverse_transform(y_predict_1)
y_predict_1
```

Out[36]:
```
array([[218712.4]], dtype=float32)
```

In [37]:
```
y_predict = model.predict(X_test)
plt.plot(y_test, y_predict, "^", color = 'r')
plt.xlabel('Model Predictions')
plt.ylabel('True Values')
```

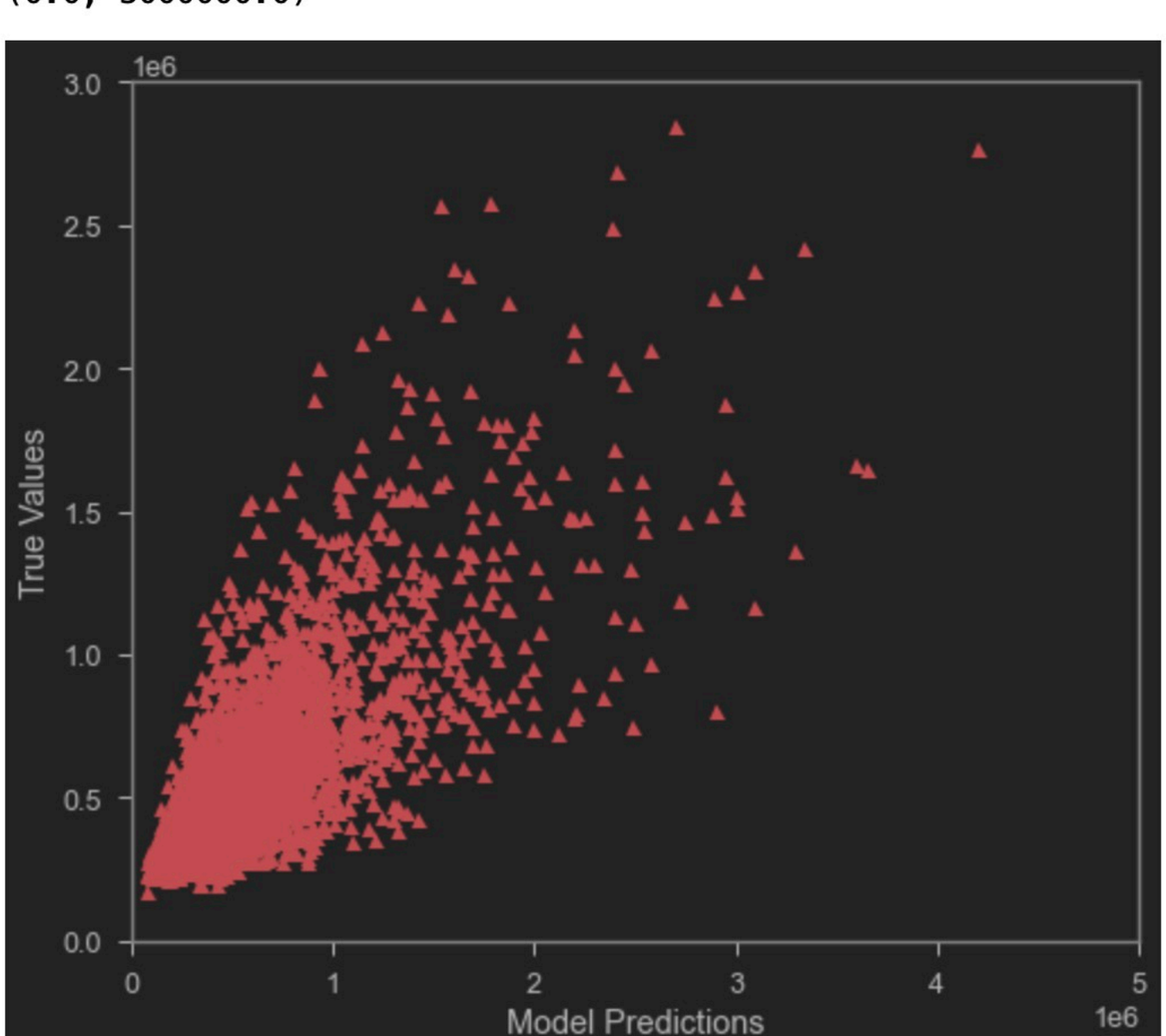
Out[37]: Text(0, 0.5, 'True Values')



In [38]:
```
y_predict_orig = scaler.inverse_transform(y_predict)
y_test_orig = scaler.inverse_transform(y_test)
```

In [39]:
```
plt.plot(y_test_orig, y_predict_orig, "^", color = 'r')
plt.xlabel('Model Predictions')
plt.ylabel('True Values')
plt.xlim(0, 5000000)
plt.ylim(0, 3000000)
```

Out[39]: (0.0, 3000000.0)



In [40]:
```
k = X_test.shape[1]
n = len(X_test)
n
```

Out[40]:
```
5404
```

In [41]:
```
k
```

Out[41]:
```
7
```

In [42]:
```
from sklearn.metrics import r2_score, mean_squared_error, mean_absolute_error
from math import sqrt

RMSE = float(format(np.sqrt(mean_squared_error(y_test_orig, y_predict_orig)),'.3f'))
MSE = mean_squared_error(y_test_orig, y_predict_orig)
MAE = mean_absolute_error(y_test_orig, y_predict_orig)
r2 = r2_score(y_test_orig, y_predict_orig)
adj_r2 = 1-(1-r2)*(n-1)/(n-k-1)

print('RMSE =',RMSE, '\nMSE =',MSE, '\nMAE =',MAE, '\nR2 =', r2, '\nAdjusted R2 =', adj_r2)
```

```
RMSE = 244520.289
MSE = 59790171709.68345
MAE = 153968.88459937082
R2 = 0.5680483772792423
Adjusted R2 = 0.5674880249147046
```

TASK #7. TRAIN AND EVALUATE A DEEP LEARNING MODEL WITH INCREASED NUMBER OF FEATURES (INDEPENDANT VARIABLES)

In [68]:
```
selected_features = ['bedrooms','bathrooms','sqft_living','sqft_lot','floors', 'sqft_above', 'sqft_basement', 'water
'yr_renovated', 'zipcode', 'lat', 'long', 'sqft_living15', 'sqft_lot15']

X = house_df[selected_features]
```

In [69]:
```
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()
X_scaled = scaler.fit_transform(X)
```

In [70]:
```
y = house_df['price']
```

In [71]:
```
y = y.values.reshape(-1,1)
y_scaled = scaler.fit_transform(y)
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y_scaled, test_size = 0.25)
```

In [72]:
```
import tensorflow.keras
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense

model = Sequential()
model.add(Dense(10, input_dim = 19, activation = 'relu'))
model.add(Dense(10, activation = 'relu'))
model.add(Dense(1, activation = 'linear'))
```

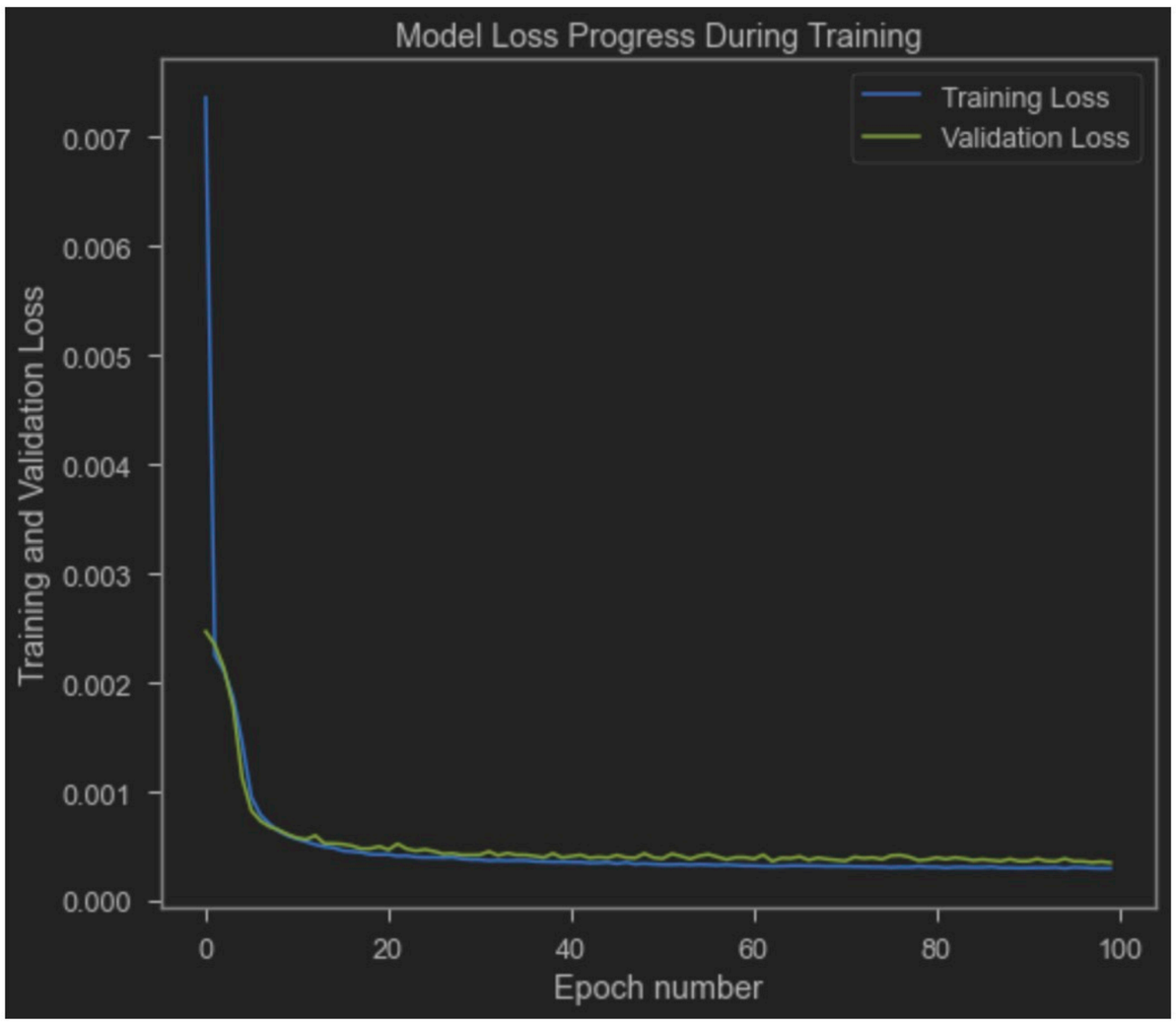
In [73]:
```
model.compile(optimizer = 'adam', loss = 'mean_squared_error')
```

In [74]:
```
epochs_hist = model.fit(X_train, y_train, epochs = 100, batch_size = 50, verbose = 1, validation_split = 0.2)
```

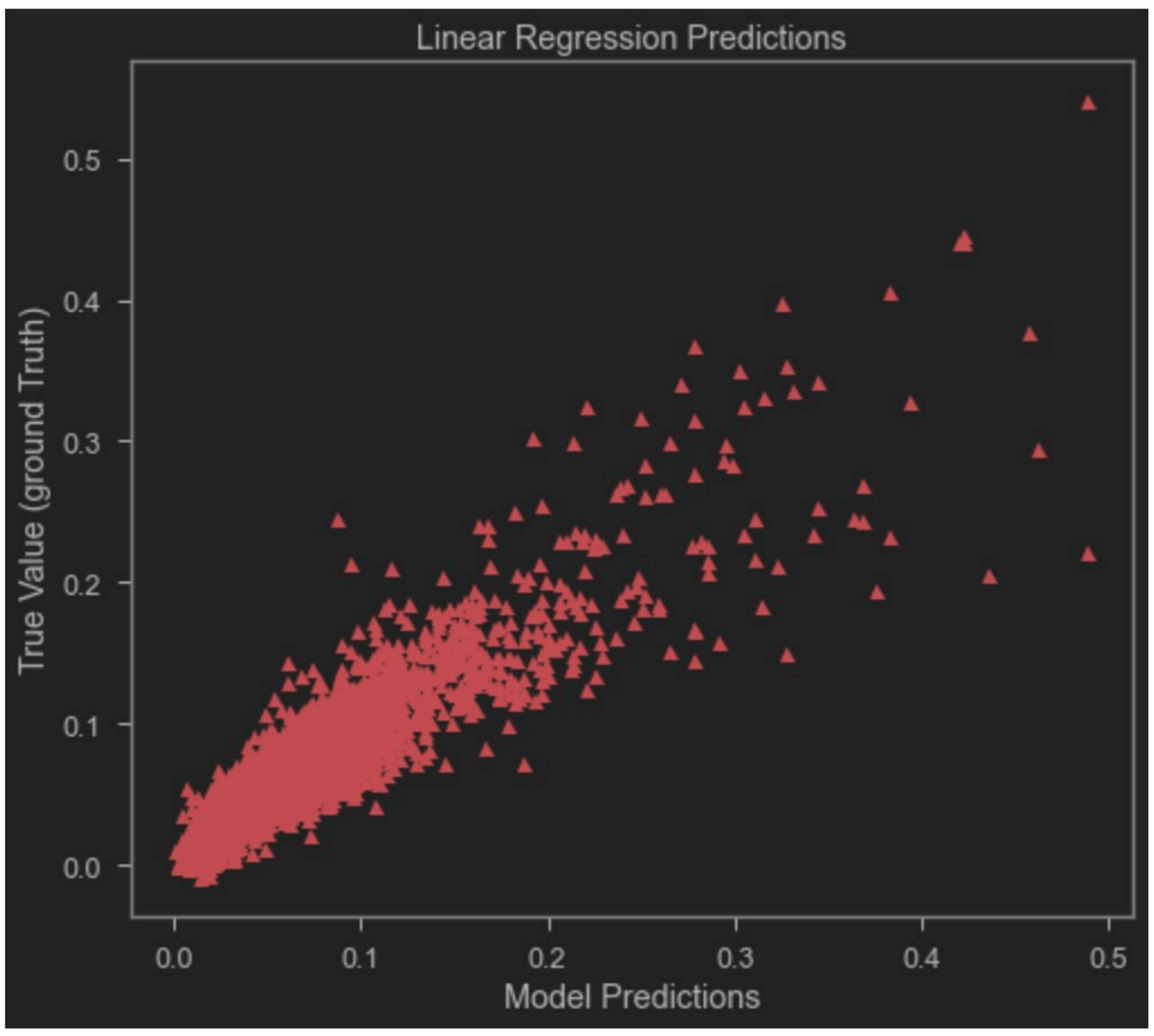
```
Train on 12967 samples, validate on 3242 samples
Epoch 1/100
12967/12967 [==============================] - 2s 119us/sample - loss: 0.0074 - val_loss: 0.0025
Epoch 2/100
12967/12967 [==============================] - 1s 61us/sample - loss: 0.0022 - val_loss: 0.0023
Epoch 3/100
12967/12967 [==============================] - 1s 52us/sample - loss: 0.0021 - val_loss: 0.0021
Epoch 4/100
12967/12967 [==============================] - 1s 52us/sample - loss: 0.0019 - val_loss: 0.0018
Epoch 5/100
12967/12967 [==============================] - 1s 55us/sample - loss: 0.0014 - val_loss: 0.0011
Epoch 6/100
12967/12967 [==============================] - 1s 62us/sample - loss: 9.4983e-04 - val_loss: 8.2648e-04
Epoch 7/100
12967/12967 [==============================] - 1s 68us/sample - loss: 7.8759e-04 - val_loss: 7.3160e-04
Epoch 8/100
12967/12967 [==============================] - 1s 63us/sample - loss: 7.0361e-04 - val_loss: 6.7910e-04
Epoch 9/100
12967/12967 [==============================] - 1s 68us/sample - loss: 6.3749e-04 - val_loss: 6.4892e-04
```

In [75]:
```
plt.plot(epochs_hist.history['loss'])
plt.plot(epochs_hist.history['val_loss'])
plt.title('Model Loss Progress During Training')
plt.ylabel('Training and Validation Loss')
plt.xlabel('Epoch number')
plt.legend(['Training Loss', 'Validation Loss'])
```

Out[75]:
```
<matplotlib.legend.Legend at 0x20f67945748>
```



In [79]:
```
y_predict = model.predict(X_test)
plt.plot(y_test, y_predict, "^", color = 'r')
plt.xlabel("Model Predictions")
plt.ylabel("True Value (ground Truth)")
plt.title('Linear Regression Predictions')
plt.show()
```



In [80]:
```
y_predict_orig = scaler.inverse_transform(y_predict)
y_test_orig = scaler.inverse_transform(y_test)
```

In [81]:
```
from sklearn.metrics import r2_score, mean_squared_error, mean_absolute_error
from math import sqrt

RMSE = float(format(np.sqrt(mean_squared_error(y_test_orig, y_predict_orig)),'.3f'))
MSE = mean_squared_error(y_test_orig, y_predict_orig)
MAE = mean_absolute_error(y_test_orig, y_predict_orig)
r2 = r2_score(y_test_orig, y_predict_orig)
adj_r2 = 1-(1-r2)*(n-1)/(n-k-1)

print('RMSE =',RMSE, '\nMSE =',MSE, '\nMAE =',MAE, '\nR2 =', r2, '\nAdjusted R2 =', adj_r2)
```

```
RMSE = 139326.953
MSE = 19411999744.979095
MAE = 85971.19128155938
R2 = 0.8420021760623091
Adjusted R2 = 0.8417972122432646
```

PRACTICE OPPORTUNITY #4 [OPTIONAL]:

- **Change the architecture of the network to increase the coefficient of determination to at least 0.86.**

In [ ]:
```

```

GREAT JOB!