Predicting House Prices

In [34]:
```
import pandas as pd
import numpy as np
import random as rnd

import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline

from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MinMaxScaler

from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Activation
from tensorflow.keras.optimizers import Adam

from sklearn.metrics import mean_squared_error,mean_absolute_error,explained_
from sklearn.metrics import classification_report,confusion_matrix
```

In [35]:
```
df = pd.read_csv('C:\\Users\\purva\\Desktop\\Projects\\House Price Prediction
```

Analyze by describing data

In [6]:
```
print(df.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']
```

In [7]:

```
df.head()
```

Out[7]:

	id	date	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors
0	7129300520	20141013T000000	221900.0	3	1.00	1180	5650	1.0
1	6414100192	20141209T000000	538000.0	3	2.25	2570	7242	2.0
2	5631500400	20150225T000000	180000.0	2	1.00	770	10000	1.0
3	2487200875	20141209T000000	604000.0	4	3.00	1960	5000	1.0
4	1954400510	20150218T000000	510000.0	3	2.00	1680	8080	1.0

5 rows × 21 columns

In [8]:

```
df.tail()
```

Out[8]:

	id	date	price	bedrooms	bathrooms	sqft_living	sqft_lot	fl
21608	263000018	20140521T000000	360000.0	3	2.50	1530	1131	
21609	6600060120	20150223T000000	400000.0	4	2.50	2310	5813	
21610	1523300141	20140623T000000	402101.0	2	0.75	1020	1350	
21611	291310100	20150116T000000	400000.0	3	2.50	1600	2388	
21612	1523300157	20141015T000000	325000.0	2	0.75	1020	1076	

5 rows × 21 columns

Which features contain blank, null or empty values?

In [10]:

```
df.isnull().sum()
```

Out[10]:

```
id               0
date             0
price            0
bedrooms         0
bathrooms        0
sqft_living      0
sqft_lot         0
floors           0
waterfront       0
view             0
condition        0
grade            0
sqft_above       0
sqft_basement    0
yr_built         0
yr_renovated     0
zipcode          0
lat              0
long             0
sqft_living15    0
sqft_lot15       0
dtype: int64
```

Five features are floats, fifteen are integers and one is an object.

In [11]:
```
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  
dtypes: float64(5), int64(15), object(1)
memory usage: 3.5+ MB
```

Here I checked what is the distribution of numerical feature values across the samples?

In [12]:

```
df.describe().transpose()
```

Out[12]:

	count	mean	std	min	25%	
id	21613.0	4.580302e+09	2.876566e+09	1.000102e+06	2.123049e+09	3.904930e
price	21613.0	5.400881e+05	3.671272e+05	7.500000e+04	3.219500e+05	4.500000e
bedrooms	21613.0	3.370842e+00	9.300618e-01	0.000000e+00	3.000000e+00	3.000000e
bathrooms	21613.0	2.114757e+00	7.701632e-01	0.000000e+00	1.750000e+00	2.250000e
sqft_living	21613.0	2.079900e+03	9.184409e+02	2.900000e+02	1.427000e+03	1.910000e
sqft_lot	21613.0	1.510697e+04	4.142051e+04	5.200000e+02	5.040000e+03	7.618000e
floors	21613.0	1.494309e+00	5.399889e-01	1.000000e+00	1.000000e+00	1.500000e
waterfront	21613.0	7.541757e-03	8.651720e-02	0.000000e+00	0.000000e+00	0.000000e
view	21613.0	2.343034e-01	7.663176e-01	0.000000e+00	0.000000e+00	0.000000e
condition	21613.0	3.409430e+00	6.507430e-01	1.000000e+00	3.000000e+00	3.000000e
grade	21613.0	7.656873e+00	1.175459e+00	1.000000e+00	7.000000e+00	7.000000e
sqft_above	21613.0	1.788391e+03	8.280910e+02	2.900000e+02	1.190000e+03	1.560000e
sqft_basement	21613.0	2.915090e+02	4.425750e+02	0.000000e+00	0.000000e+00	0.000000e
yr_built	21613.0	1.971005e+03	2.937341e+01	1.900000e+03	1.951000e+03	1.975000e
yr_renovated	21613.0	8.440226e+01	4.016792e+02	0.000000e+00	0.000000e+00	0.000000e
zipcode	21613.0	9.807794e+04	5.350503e+01	9.800100e+04	9.803300e+04	9.806500e
lat	21613.0	4.756005e+01	1.385637e-01	4.715590e+01	4.747100e+01	4.757180e
long	21613.0	-1.222139e+02	1.408283e-01	-1.225190e+02	-1.223280e+02	-1.222300e
sqft_living15	21613.0	1.986552e+03	6.853913e+02	3.990000e+02	1.490000e+03	1.840000e
sqft_lot15	21613.0	1.276846e+04	2.730418e+04	6.510000e+02	5.100000e+03	7.620000e

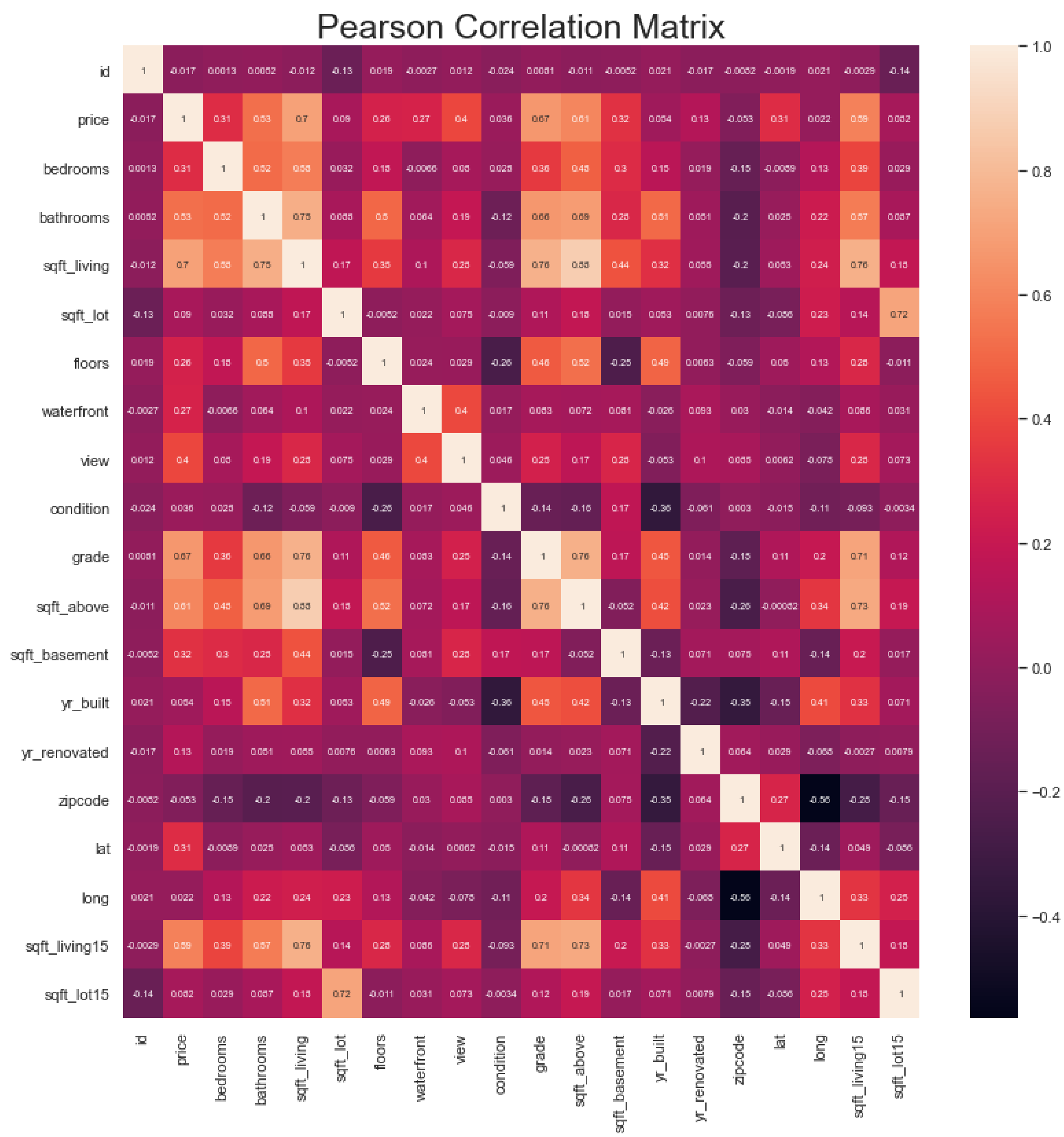
Exploratory Data Analysis

Analyze by visualizing data

Now I can continue confirming some of my assumptions using visualizations for analyzing the data.

In [24]:
```python
sns.set(style="whitegrid", font_scale=1)

plt.figure(figsize=(13,13))
plt.title('Pearson Correlation Matrix',fontsize=25)
sns.heatmap(df.corr(),annot=True, annot_kws={"size":7})
```

Out[24]: <AxesSubplot:title={'center':'Pearson Correlation Matrix'}>



Price correlation

- sqft_living looks like a highly correlated label to the price, as well as grade, sqft_above, sqft_living15 and bathrooms.

In [25]:

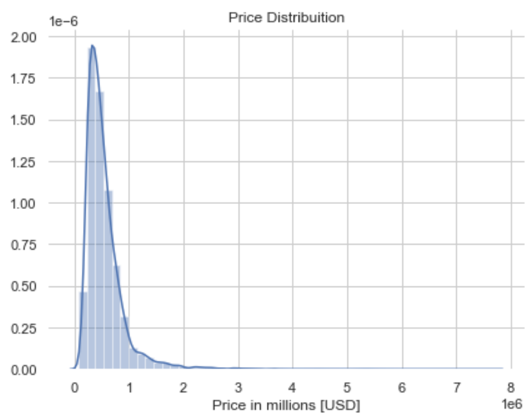
```
price_corr = df.corr()['price'].sort_values(ascending=False)
print(price_corr)
```

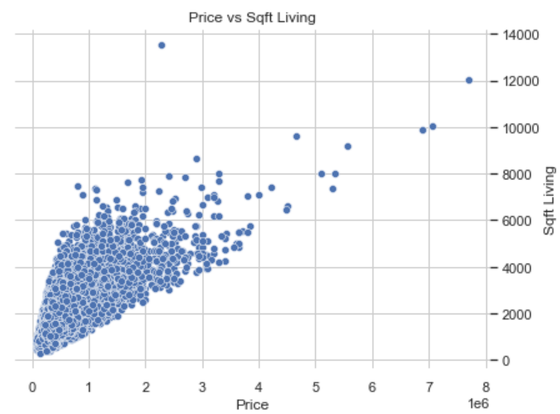
```
price            1.000000
sqft_living      0.702035
grade            0.667434
sqft_above       0.605567
sqft_living15    0.585379
bathrooms        0.525138
view             0.397293
sqft_basement    0.323816
bedrooms         0.308350
lat              0.307003
waterfront       0.266369
floors           0.256794
yr_renovated     0.126434
sqft_lot         0.089661
sqft_lot15       0.082447
yr_built         0.054012
condition        0.036362
long             0.021626
id              -0.016762
zipcode         -0.053203
Name: price, dtype: float64
```

In [26]:

```
f, axes = plt.subplots(1, 2,figsize=(15,5))
sns.distplot(df['price'], ax=axes[0])
sns.scatterplot(x='price',y='sqft_living', data=df, ax=axes[1])
sns.despine(bottom=True, left=True)
axes[0].set(xlabel='Price in millions [USD]', ylabel='', title='Price Distrib
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()
```

```
C:\ProgramData\Anaconda3\lib\site-packages\seaborn\distributions.py:2557: F
utureWarning: `distplot` is a deprecated function and will be removed in a
future version. Please adapt your code to use either `displot` (a figure-le
vel function with similar flexibility) or `histplot` (an axes-level functio
n for histograms).
  warnings.warn(msg, FutureWarning)
```



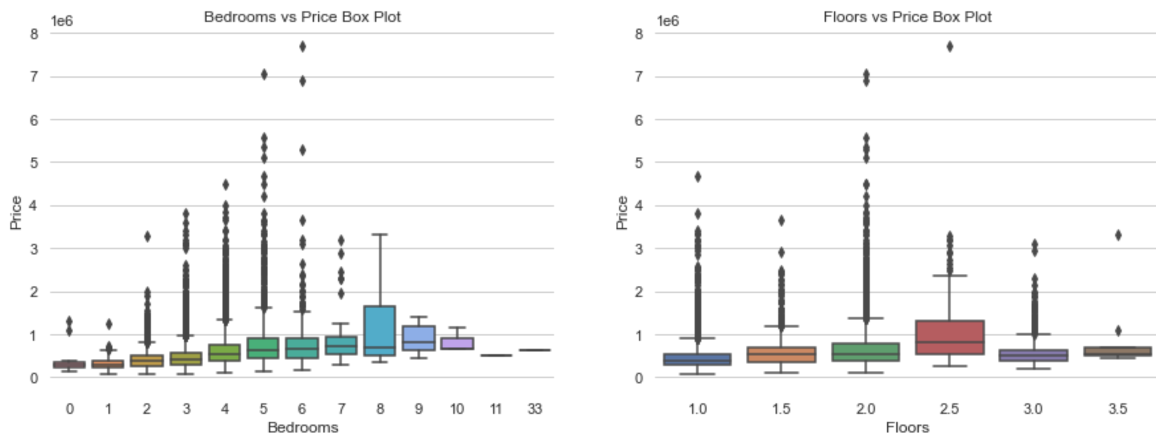


In [27]:

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

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

Out[27]:

```
[Text(0.5, 0, 'Floors'),
 Text(0, 0.5, 'Price'),
 Text(0.5, 1.0, 'Floors vs Price Box Plot')]
```



Bedrooms and floors box plots

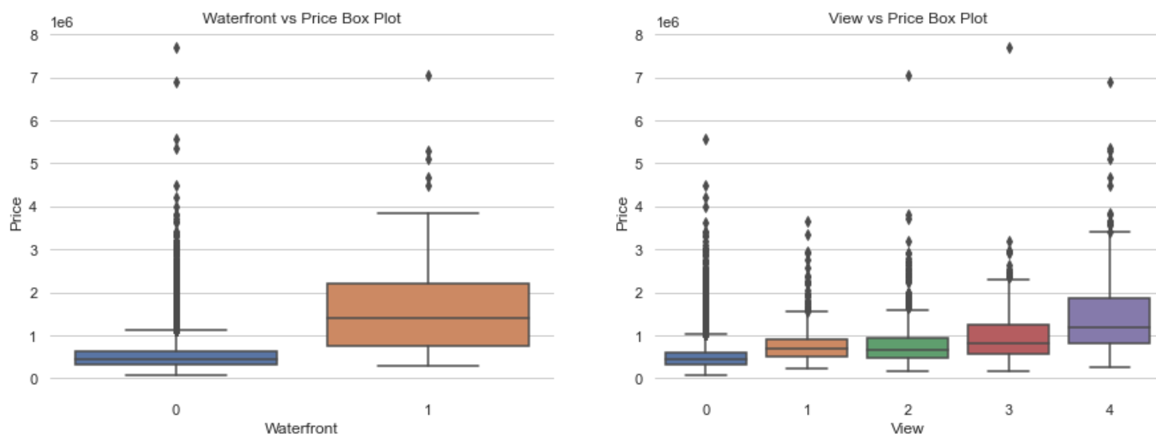
- We can see outliers plotted as individual points; this probably are the more expensive houses.
- We can see that the price tends to go up when the house has more bedrooms.

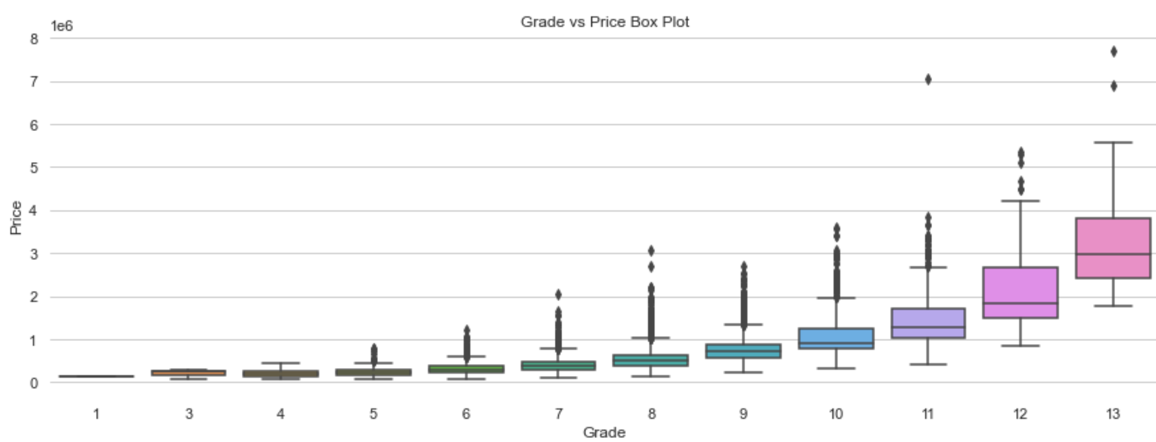
In [28]:

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

Out[28]:

```
[Text(0.5, 0, 'Grade'),
 Text(0, 0.5, 'Price'),
 Text(0.5, 1.0, 'Grade vs Price Box Plot')]
```





Waterfront, view and grade box plots

- Waterfront houses tends to have a better price value.
- The price of waterfront houses tends to be more disperse and the price of houses without waterfront tend to be more concentrated.
- Grade and waterfront effect price. View seem to effect less but it also has an effect on price.

Working with Feature Data

In [36]:
```
df = df.drop('id', axis=1)
df = df.drop('zipcode',axis=1)
```

Feature engineering

I engineer the date feature to make a year and month column.

In [37]:
```
df['date'] = pd.to_datetime(df['date'])

df['month'] = df['date'].apply(lambda date:date.month)
df['year'] = df['date'].apply(lambda date:date.year)

df = df.drop('date',axis=1)

print(df.columns.values)
```

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

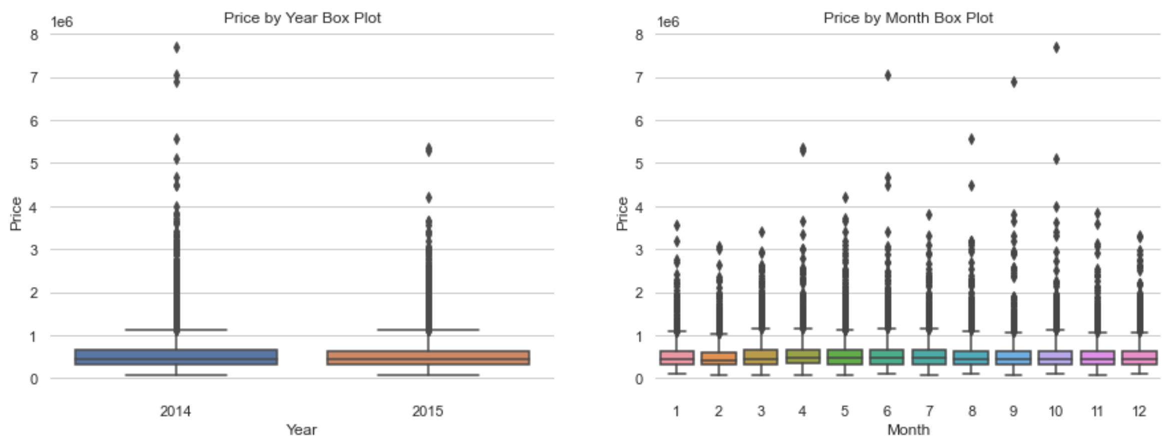
House price trends

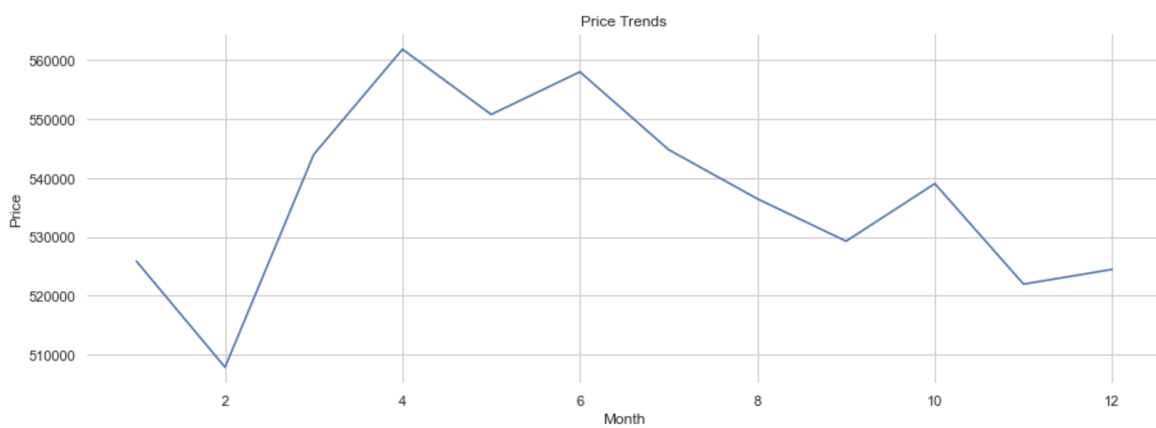
In [38]:

```
f, axes = plt.subplots(1, 2,figsize=(15,5))
sns.boxplot(x='year',y='price',data=df, ax=axes[0])
sns.boxplot(x='month',y='price',data=df, 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))
df.groupby('month').mean()['price'].plot()
sns.despine(left=True, bottom=True)
axe.set(xlabel='Month', ylabel='Price', title='Price Trends')
```

Out[38]:

```
[Text(0.5, 0, 'Month'), Text(0, 0.5, 'Price'), Text(0.5, 1.0, 'Price Trend
s')]
```





- Looking the box plots I noticed that there is not a big difference between 2014 and 2015.
- The number of houses sold by month tends to be similar every month.
- The line plot show that around April there is an increase in house prices.

Scaling and train test split

In [39]:
```
# Features
X = df.drop('price',axis=1)

# Label
y = df['price']

# Split
X_train, X_test, y_train, y_test = train_test_split(X,y,test_size=0.3,random_
```

In [40]:
```
print(X_train.shape)
print(X_test.shape)
print(y_train.shape)
print(y_test.shape)
```

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

Normalizing / scaling the data

I will scale the feature data to prevent data leakage from the test set, I will only fit my scaler to the training set.

In [41]:
```
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
```

Creating a model

In [42]:

```
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')
```

Training the model

Since the dataset is large, I am going to use batch_size. The smaller the batch size, the longer is going to take.

In [43]:

```
model.fit(x=X_train,y=y_train.values,validation_data=(X_test,y_test.values),b
```

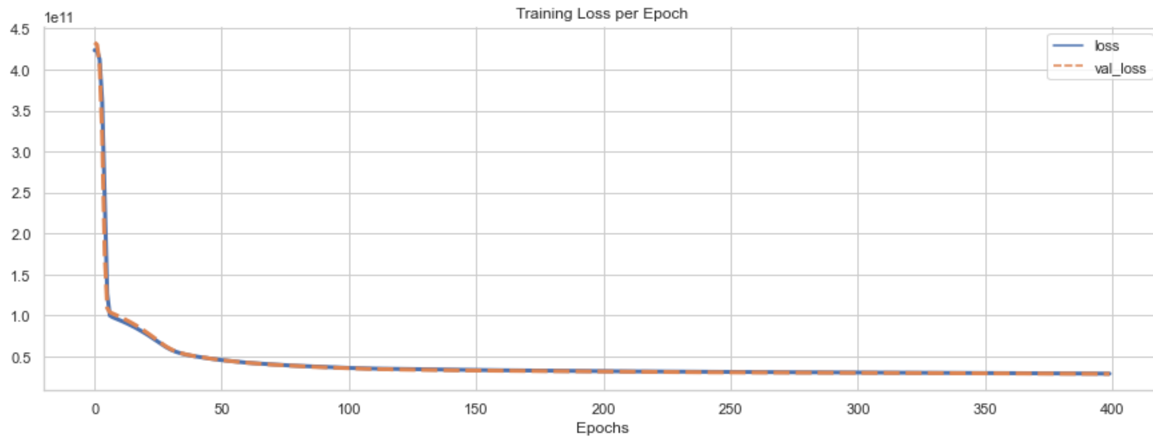
```
Epoch 1/400
119/119 [==============================] - 0s 2ms/step - loss: 4236267
68384.0000 - val_loss: 433020698624.0000
Epoch 2/400
119/119 [==============================] - 0s 1ms/step - loss: 4228746
11712.0000 - val_loss: 430423343104.0000
Epoch 3/400
119/119 [==============================] - 0s 1ms/step - loss: 4127808
88064.0000 - val_loss: 406929965056.0000
Epoch 4/400
119/119 [==============================] - 0s 899us/step - loss: 36218
1492736.0000 - val_loss: 320074842112.0000
Epoch 5/400
119/119 [==============================] - 0s 1ms/step - loss: 2409690
72640.0000 - val_loss: 175514697728.0000
Epoch 6/400
119/119 [==============================] - 0s 891us/step - loss: 12667
5329024.0000 - val_loss: 108964192256.0000
Epoch 7/400
```

Training loss per epoch

- This plot helps us to see if there is overfitting in the model. In this case there is no overfitting because both lines go down at the same time.

In [44]:

```
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()
```



Evaluation on test data

In [45]:

```
# 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,predictic

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

```
MAE:  105036.43256356994
MSE:  28802044828.599674
RMSE:  169711.65201187477
Variance Regression Score:  0.7947841214478216


Descriptive Statistics:
 count    2.161300e+04
mean     5.400881e+05
std      3.671272e+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
```

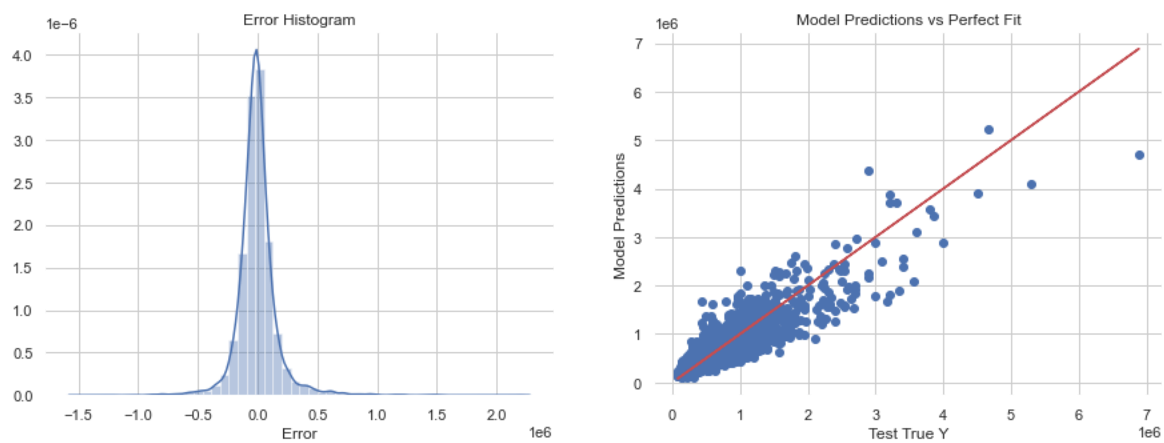
In [46]:

```
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 Pr
```

```
C:\ProgramData\Anaconda3\lib\site-packages\seaborn\distributions.py:2557: F
utureWarning: `distplot` is a deprecated function and will be removed in a
future version. Please adapt your code to use either `displot` (a figure-le
vel function with similar flexibility) or `histplot` (an axes-level functio
n for histograms).
  warnings.warn(msg, FutureWarning)
```

Out[46]:

```
[Text(0.5, 0, 'Test True Y'),
 Text(0, 0.5, 'Model Predictions'),
 Text(0.5, 1.0, 'Model Predictions vs Perfect Fit')]
```



- I compared the model predictions with a perfect fit to see how accurate the model is.
- The red line represents the perfect prediction.
- We are being punish by the outliers, which are the expensive houses. Our model is not good predicting luxury houses.
- On the other hand, our model is good predicting the price of houses between o and $2 million. There is clearly a good fit.
- It may be worth it retraining our model just on price houses below $3 million.

Predicting on a brand new house

Now I will use the model to predict the price on a brand-new house. I am going to choose the first house of the data set and drop the price. single_house is going to have all the features that I need to predict the price. After that I need to reshape the variable and scale the features.

In [48]:
```
# fueatures of new house
single_house = df.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:',df.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
month              10.0000
year             2014.0000
Name: 0, dtype: float64

Prediction Price: 288917.16

Original Price: 221900.0
```

The original price is $221,900 and the model prediction is $280,000.