In [1]:

```python
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline
import sklearn
import seaborn as sns
import warnings
warnings.filterwarnings('ignore')
plt.rcParams["figure.figsize"] = [10,5]
# Ignore warnings

import warnings
# Set the warning filter to ignore FutureWarning
warnings.simplefilter(action='ignore',category=FutureWarning)
```

TRAINING DATA PRE-PROCESSING

In [6]:

```
df = pd.read_csv('C:\\Users\\Super\\Downloads\\USA_Housing.csv')
df
```

Out[6]:

	Avg. Area Income	Avg. Area House Age	Avg. Area Number of Rooms	Avg. Area Number of Bedrooms	Area Population	Price	Address
0	79545.45857	5.682861	7.009188	4.09	23086.80050	1.059034e+06	208 Michael Ferry Apt. 674\nLaurabury, NE 3701...
1	79248.64245	6.002900	6.730821	3.09	40173.07217	1.505891e+06	188 Johnson Views Suite 079\nLake Kathleen, CA...
2	61287.06718	5.865890	8.512727	5.13	36882.15940	1.058988e+06	9127 Elizabeth Stravenue\nDanieltown, WI 06482...
3	63345.24005	7.188236	5.586729	3.26	34310.24283	1.260617e+06	USS Barnett\nFPO AP 44820
4	59982.19723	5.040555	7.839388	4.23	26354.10947	6.309435e+05	USNS Raymond\nFPO AE 09386
...	...	...	...	...	...	...	...
4995	60567.94414	7.830362	6.137356	3.46	22837.36103	1.060194e+06	USNS Williams\nFPO AP 30153-7653
4996	78491.27543	6.999135	6.576763	4.02	25616.11549	1.482618e+06	PSC 9258, Box 8489\nAPO AA 42991-3352
4997	63390.68689	7.250591	4.805081	2.13	33266.14549	1.030730e+06	4215 Tracy Garden Suite 076\nJoshualand, VA 01...
4998	68001.33124	5.534388	7.130144	5.44	42625.62016	1.198657e+06	USS Wallace\nFPO AE 73316
4999	65510.58180	5.992305	6.792336	4.07	46501.28380	1.298950e+06	37778 George Ridges Apt. 509\nEast Holly, NV 2...

5000 rows × 7 columns

Data Shape

In [7]:

```
# Data shape
print('train data:',df.shape)
```

```
train data: (5000, 7)
```

In [8]:

```python
# View first few rows
df.head(5)
```

Out[8]:

	Avg. Area Income	Avg. Area House Age	Avg. Area Number of Rooms	Avg. Area Number of Bedrooms	Area Population	Price	Address
0	79545.45857	5.682861	7.009188	4.09	23086.80050	1.059034e+06	208 Michael Ferry Apt. 674\nLaurabury, NE 3701...
1	79248.64245	6.002900	6.730821	3.09	40173.07217	1.505891e+06	188 Johnson Views Suite 079\nLake Kathleen, CA...
2	61287.06718	5.865890	8.512727	5.13	36882.15940	1.058988e+06	9127 Elizabeth Stravenue\nDanieltown, WI 06482...
3	63345.24005	7.188236	5.586729	3.26	34310.24283	1.260617e+06	USS Barnett\nFPO AP 44820
4	59982.19723	5.040555	7.839388	4.23	26354.10947	6.309435e+05	USNS Raymond\nFPO AE 09386

In [9]:

```python
# Data Info
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5000 entries, 0 to 4999
Data columns (total 7 columns):
 #   Column                        Non-Null Count  Dtype  
---  ------                        --------------  -----  
 0   Avg. Area Income              5000 non-null   float64
 1   Avg. Area House Age           5000 non-null   float64
 2   Avg. Area Number of Rooms     5000 non-null   float64
 3   Avg. Area Number of Bedrooms  5000 non-null   float64
 4   Area Population               5000 non-null   float64
 5   Price                         5000 non-null   float64
 6   Address                       5000 non-null   object 
dtypes: float64(6), object(1)
memory usage: 273.6+ KB
```

Missing Data

In [10]:

```
# Heatmap
sns.heatmap(df.isnull(),yticklabels = False, cbar = False,cmap = 'tab20c_r')
plt.title('Missing Data: Training Set')
plt.show()
```


Missing Data: Training Set
Avg. Area Income
Avg. Area House Age
Avg. Area Number of Rooms
Avg. Area Number of Bedrooms
Area Population
Price
Address

In [11]:

```
# Remove Address feature
df.drop('Address', axis = 1, inplace = True)
```

In [12]:

```
# Remove rows with missing data
df.dropna(inplace = True)
```

In [13]:

```
df.head()
```

Out[13]:

	Avg. Area Income	Avg. Area House Age	Avg. Area Number of Rooms	Avg. Area Number of Bedrooms	Area Population	Price
0	79545.45857	5.682861	7.009188	4.09	23086.80050	1.059034e+06
1	79248.64245	6.002900	6.730821	3.09	40173.07217	1.505891e+06
2	61287.06718	5.865890	8.512727	5.13	36882.15940	1.058988e+06
3	63345.24005	7.188236	5.586729	3.26	34310.24283	1.260617e+06
4	59982.19723	5.040555	7.839388	4.23	26354.10947	6.309435e+05

Numeric Features

In [14]:

```
df.describe()
```

Out[14]:

	Avg. Area Income	Avg. Area House Age	Avg. Area Number of Rooms	Avg. Area Number of Bedrooms	Area Population	Price
count	5000.000000	5000.000000	5000.000000	5000.000000	5000.000000	5.000000e+03
mean	68583.108984	5.977222	6.987792	3.981330	36163.516039	1.232073e+06
std	10657.991214	0.991456	1.005833	1.234137	9925.650114	3.531176e+05
min	17796.631190	2.644304	3.236194	2.000000	172.610686	1.593866e+04
25%	61480.562390	5.322283	6.299250	3.140000	29403.928700	9.975771e+05
50%	68804.286405	5.970429	7.002902	4.050000	36199.406690	1.232669e+06
75%	75783.338665	6.650808	7.665871	4.490000	42861.290770	1.471210e+06
max	107701.748400	9.519088	10.759588	6.500000	69621.713380	2.469066e+06

GETTING MODEL READY

In [15]:
```
# Shape of train data
df.shape
```

Out[15]: (5000, 6)

OBJECTIVE 2: MACHINE LEARNING

Next, I will feed these features into various classification algorithms to determine the best performance using a simple framework: Split, Fit, Predict, Score It.

Target Variable Splitting

In [16]:
```
 #Split data to be used in the models
# Create matrix of features
x = df.drop('Price',axis=1) # grabs everything else but 'Price'

# Create target variable
y = df['Price'] # y is the column we're trying to predict

```

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

Out[17]: (5000, 5)

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

Out[18]: (5000,)

In [19]:

```python
# Use x and y variables to split the training data into train and test set
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size = .20, random_state = 101)
```

In [20]:

```python
x_train.shape
x_train
```

Out[20]:

	Avg. Area Income	Avg. Area House Age	Avg. Area Number of Rooms	Avg. Area Number of Bedrooms	Area Population
3413	69048.78809	6.619712	6.123813	4.33	36817.36876
1610	67866.89993	5.393978	9.359022	5.44	43122.57418
3459	56636.23819	5.497667	7.121872	6.10	47541.43176
4293	79310.36198	4.247434	7.518204	4.38	43982.18896
1039	72821.24766	6.480819	7.116655	5.33	40594.05930
...	...	...	...	...	...
4171	56610.64256	4.846832	7.558137	3.29	25494.74030
599	70596.85095	6.548274	6.539986	3.10	51614.83014
1361	55621.89910	3.735942	6.868291	2.30	63184.61315
1547	63044.46010	5.935261	5.913454	4.10	32725.27954
4959	75078.79152	7.644779	8.440726	4.33	56148.44932

4000 rows × 5 columns

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

Out[21]:
```
3413    1.305210e+06
1610    1.400961e+06
3459    1.048640e+06
4293    1.231157e+06
1039    1.391233e+06
            ...
4171    7.296417e+05
599     1.599479e+06
1361    1.102641e+06
1547    8.650995e+05
4959    2.108376e+06
Name: Price, Length: 4000, dtype: float64
```

In [22]:
```
x_test.shape
x_test
```

Out[22]:

	Avg. Area Income	Avg. Area House Age	Avg. Area Number of Rooms	Avg. Area Number of Bedrooms	Area Population
1718	66774.99582	5.717143	7.795215	4.32	36788.980330
2511	62184.53937	4.925758	7.427689	6.22	26008.309120
345	73643.05730	6.766853	8.337085	3.34	43152.139580
2521	61909.04144	6.228343	6.593138	4.29	28953.925380
54	72942.70506	4.786222	7.319886	6.41	24377.909050
...	...	...	...	...	...
3900	77615.85134	6.200603	6.909327	2.27	36591.523450
3753	66925.19935	5.153050	8.396903	3.16	42590.685170
3582	71778.02618	5.921280	7.411045	4.00	37634.041320
2392	87272.09339	5.025866	7.184765	5.39	7522.333138
3343	70271.10419	5.856327	6.782116	2.46	28101.644400

1000 rows × 5 columns

LINEAR REGRESSION

Model Training

In [25]:
```
# Fit
# Import model
from sklearn.linear_model import LinearRegression

# Create instance of model
lin_reg = LinearRegression()

# Pass training data into model
lin_reg.fit(x_train, y_train)
```

Out[25]:
```
▾ LinearRegression
LinearRegression()
```

Model Testing

Class prediction

In [26]:

```python
#predict
y_predict=lin_reg.predict(x_test)
y_predict.shape
y_predict
```

Out[26]:

```
array([1257919.72924299,  822112.41868756, 1740669.05869474,
        972452.12926804,  993422.2632988 ,  644126.07416935,
       1073911.79097589,  856584.00208537, 1445318.25527738,
       1204342.19071515, 1455792.46233196, 1298556.65691754,
       1735924.33854636, 1336925.77593212, 1387637.43241543,
       1222403.77757898,  613786.28673738,  963933.54403085,
       1221197.33061287, 1198071.57580528,  505861.89541388,
       1769106.54726586, 1853881.16845511, 1200369.50514846,
       1065129.12845899, 1812033.73048156, 1768686.47104264,
       1439920.83823817, 1387251.9966963 , 1541178.39227172,
        726418.80525623, 1754497.609143  , 1462185.72661629,
       1025600.16064332, 1284926.86862687,  917454.59581447,
       1187046.94951786,  999330.91123324, 1329536.63408978,
        782191.60431848, 1393272.03057331,  578216.88372019,
        822643.37151103, 1895533.11423642, 1672019.84904555,
        966926.45430148, 1129674.55621678,  792797.75924288,
       1161057.18404066, 1472396.7143581 , 1457656.70413195,
       1162939.33425471, 1099453.68096241, 1358107.44627459,
        841103.7037299 ,  986322.30559828, 1123323.53002156,
```

In [27]:

```python
# Combine actual and predicted values side by side
results = np.column_stack((y_test, y_predict))
print("Actual Values  |    Predicted values")
print("--------------------------------")
for actual, predicted in results:
    print(f"{actual:14.2f} |  {predicted:12.2f}")

```

```
Actual Values  |    Predicted values
--------------------------------
    1251688.62 |    1257919.73
     873048.32 |     822112.42
    1696977.66 |    1740669.06
    1063964.29 |     972452.13
     948788.28 |     993422.26
     730043.65 |     644126.07
    1166925.15 |    1073911.79
     705444.12 |     856584.00
    1499988.88 |    1445318.26
    1288199.15 |    1204342.19
    1441736.76 |    1455792.46
    1279681.15 |    1298556.66
    1754969.16 |    1735924.34
    1511653.46 |    1336925.78
    1441956.20 |    1387637.43
    1119992.62 |    1222403.78
     727866.53 |     613786.29
```

Residual Analysis

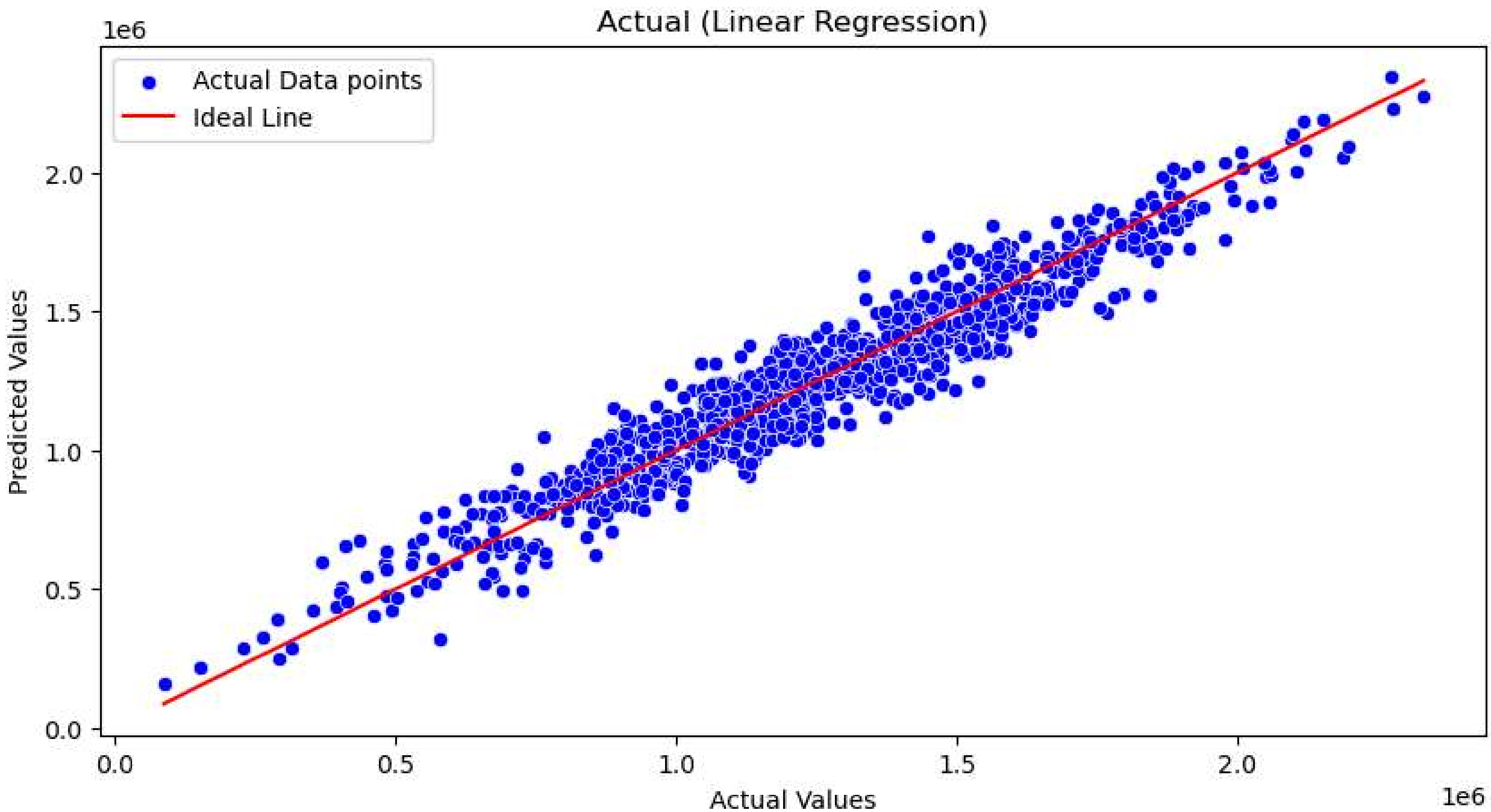
In [28]:

```
residual = actual- y_predict.reshape(-1)
print(residual)
```

```
[-6.97228472e+04  3.66084463e+05 -5.52472177e+05  2.15744753e+05
  1.94774619e+05  5.44070808e+05  1.14285091e+05  3.31612880e+05
 -2.57121373e+05 -1.61453087e+04 -2.67595580e+05 -1.10359775e+05
 -5.47727457e+05 -1.48728894e+05 -1.99440550e+05 -3.42068956e+04
  5.74410595e+05  2.24263338e+05 -3.30004486e+04 -9.87469381e+03
  6.82334987e+05 -5.80909665e+05 -6.65684286e+05 -1.21726231e+04
  1.23067754e+05 -6.23836848e+05 -5.80489589e+05 -2.51723956e+05
 -1.99055115e+05 -3.52981510e+05  4.61778077e+05 -5.66300727e+05
 -2.73988845e+05  1.62596721e+05 -9.67299866e+04  2.70742286e+05
  1.14993248e+03  1.88865971e+05 -1.41339752e+05  4.06005278e+05
 -2.05075149e+05  6.09979998e+05  3.65553510e+05 -7.07336232e+05
 -4.83822967e+05  2.21270428e+05  5.85223258e+04  3.95399123e+05
  2.71396980e+04 -2.84199832e+05 -2.69459822e+05  2.52575477e+04
  8.87432010e+04 -1.69910564e+05  3.47093178e+05  2.01874576e+05
  6.48733520e+04 -6.53417503e+04 -2.40082781e+05  6.89093673e+05
 -2.74620201e+05  7.94525134e+04  5.27608365e+05 -5.88347159e+04
 -1.54547673e+05 -1.69309109e+05  3.69338470e+05 -2.98971867e+05
 -2.08050774e+05  3.03118502e+05  3.30651051e+05 -2.46195692e+04
  8.71276104e+04 -6.46394947e+05  2.64173916e+05  3.32850074e+05
```

In [29]:

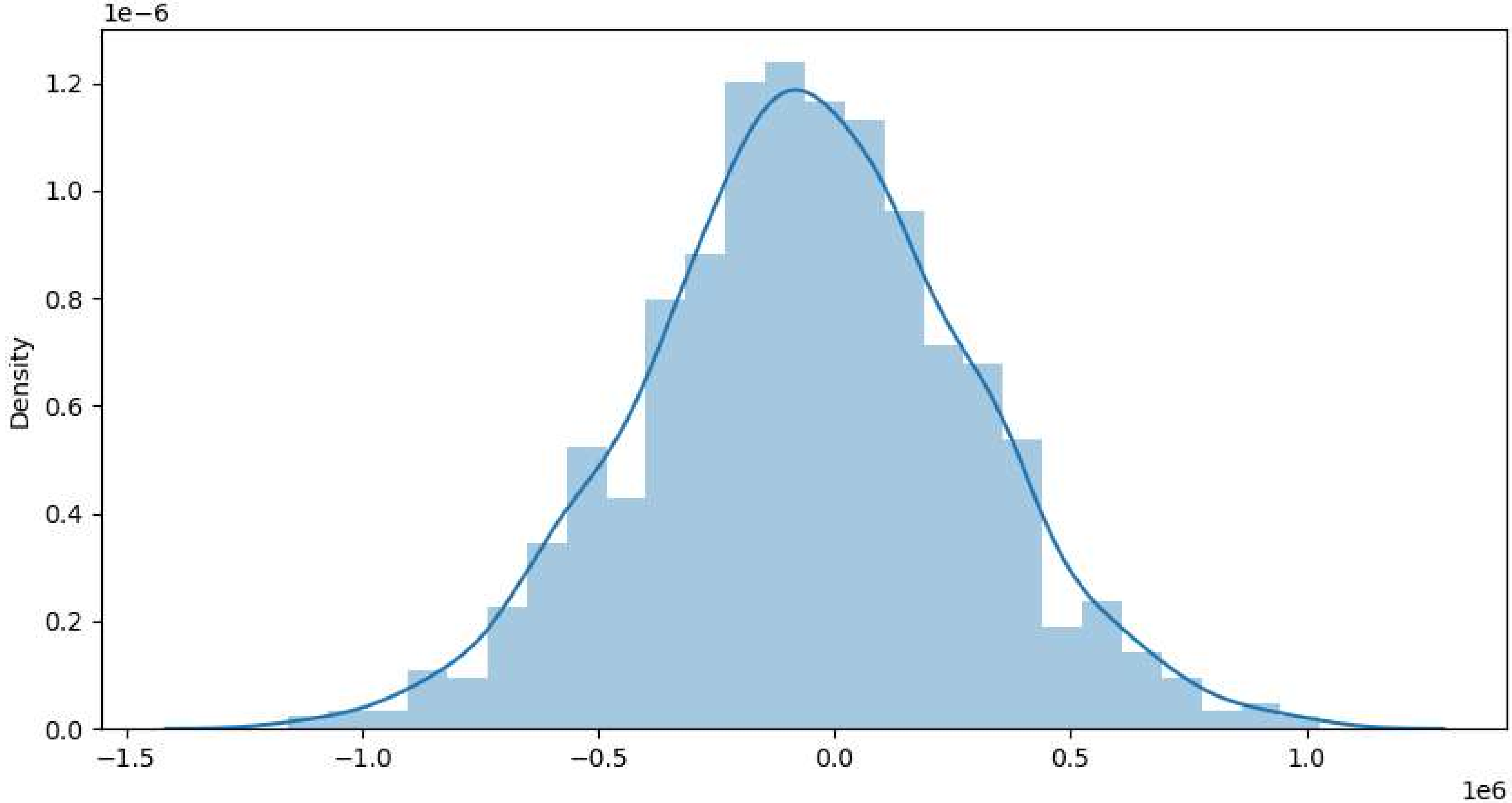
```
sns.scatterplot(x=y_test, y=y_predict, color='blue', label='Actual Data points')
plt.plot([min(y_test), max(y_test)], [min(y_test), max(y_test)], color='red', label='Ideal Line')
plt.xlabel('Actual Values')
plt.ylabel('Predicted Values')
plt.title('Actual (Linear Regression)')
plt.legend()
plt.show()
```



In [30]:

```
# Distribution plot for Residual (difference between actual and predicted values)
sns.distplot(residual, kde=True)
```

Out[30]: <Axes: ylabel='Density'>



Model Evaluation

In [31]:

```
# Score It
from sklearn.metrics import mean_squared_error

print('Linear Regression Model')
# Results
print('--'*30)
# mean_squared_error(y_test, y_pred)
mse = mean_squared_error(y_test, y_predict)
rmse = np.sqrt(mse)

# Print evaluation metrics
print("Mean Squared Error:", mse)
print("Root Mean Squared Error:", rmse)
```

```
Linear Regression Model
------------------------------------------------------------
Mean Squared Error: 10100187856.996004
Root Mean Squared Error: 100499.69083035034
```

In [ ]:

```

```