In [1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import statsmodels.api as sm
import seaborn as sns
sns.set()
```

In [2]:
```
data = pd.read_csv('TV_Final.csv')
data
```

Out[2]:

	Brand	Resolution	Size	Selling Price	Original Price	Operating System	Rating
0	TOSHIBA	Ultra HD LED	55	37999	54990	VIDAA	4.3
1	TCL	QLED Ultra HD	55	52999	129990	Android	4.4
2	realme	HD LED	32	13999	17999	Android	4.3
3	Mi	HD LED	32	14999	19999	Android	4.4
4	realme	HD LED	32	12999	21999	Android	4.3
...	...	...	...	...	...	...	...
907	SONY	Full HD LED	43	44999	57900	Linux	4.4
908	SONY	Full HD LED	40	41499	51900	Linux	4.6
909	SONY	Ultra HD LED	65	149990	184990	Linux	4.3
910	SONY	HD LED	32	32900	32900	Linux	4.4
911	SONY	Full HD LED	43	56900	56900	Linux	4.6

In [3]:
```
data.shape
```

Out[3]:
```
(912, 7)
```

In [4]:
```
data.describe(include='all')
```

Out[4]:

	Brand	Resolution	Size	Selling Price	Original Price	Operating System	Rating
count	912	912	912.000000	912.000000	912.000000	901	692.000000
unique	59	5	NaN	NaN	NaN	7	NaN
top	SAMSUNG	Ultra HD LED	NaN	NaN	NaN	Android	NaN
freq	140	399	NaN	NaN	NaN	474	NaN
mean	NaN	NaN	45.942982	59358.606360	81975.213816	NaN	4.234104
std	NaN	NaN	12.316492	65866.677856	84823.568826	NaN	0.366694
min	NaN	NaN	17.000000	4849.000000	6999.000000	NaN	2.000000
25%	NaN	NaN	32.000000	19797.500000	28990.000000	NaN	4.100000
50%	NaN	NaN	43.000000	36990.000000	52900.000000	NaN	4.300000
75%	NaN	NaN	55.000000	67064.250000	99900.000000	NaN	4.400000
max	NaN	NaN	85.000000	499990.000000	549990.000000	NaN	5.000000

In [5]:
```
data.isnull().sum()
```

Out[5]:
```
Brand                0
Resolution           0
Size                 0
Selling Price        0
Original Price       0
Operating System    11
Rating             220
dtype: int64
```

In [6]:
```
data
```

Out[6]:

	Brand	Resolution	Size	Selling Price	Original Price	Operating System	Rating
0	TOSHIBA	Ultra HD LED	55	37999	54990	VIDAA	4.3
1	TCL	QLED Ultra HD	55	52999	129990	Android	4.4
2	realme	HD LED	32	13999	17999	Android	4.3
3	Mi	HD LED	32	14999	19999	Android	4.4
4	realme	HD LED	32	12999	21999	Android	4.3
...	...	...	...	...	...	...	...
907	SONY	Full HD LED	43	44999	57900	Linux	4.4
908	SONY	Full HD LED	40	41499	51900	Linux	4.6
909	SONY	Ultra HD LED	65	149990	184990	Linux	4.3
910	SONY	HD LED	32	32900	32900	Linux	4.4
911	SONY	Full HD LED	43	56900	56900	Linux	4.6

912 rows × 7 columns

In [7]:
```
data.drop(data.columns[6], axis=1, inplace = True)
data
```

Out[7]:

	Brand	Resolution	Size	Selling Price	Original Price	Operating System
0	TOSHIBA	Ultra HD LED	55	37999	54990	VIDAA
1	TCL	QLED Ultra HD	55	52999	129990	Android
2	realme	HD LED	32	13999	17999	Android
3	Mi	HD LED	32	14999	19999	Android
4	realme	HD LED	32	12999	21999	Android
...	...	...	...	...	...	...
907	SONY	Full HD LED	43	44999	57900	Linux
908	SONY	Full HD LED	40	41499	51900	Linux
909	SONY	Ultra HD LED	65	149990	184990	Linux
910	SONY	HD LED	32	32900	32900	Linux
911	SONY	Full HD LED	43	56900	56900	Linux

912 rows × 6 columns

In [8]:
```
data.drop(data.columns[4], axis=1, inplace = True)
data
```

Out[8]:

	Brand	Resolution	Size	Selling Price	Operating System
0	TOSHIBA	Ultra HD LED	55	37999	VIDAA
1	TCL	QLED Ultra HD	55	52999	Android
2	realme	HD LED	32	13999	Android
3	Mi	HD LED	32	14999	Android
4	realme	HD LED	32	12999	Android
...	...	...	...	...	...
907	SONY	Full HD LED	43	44999	Linux
908	SONY	Full HD LED	40	41499	Linux
909	SONY	Ultra HD LED	65	149990	Linux
910	SONY	HD LED	32	32900	Linux
911	SONY	Full HD LED	43	56900	Linux

912 rows × 5 columns

In [9]:
```
data["Operating System"].unique()
```

Out[9]:
```
array(['VIDAA', 'Android', 'Linux', nan, 'Tizen', 'WebOS', 'HomeOS',
       'FireTV OS'], dtype=object)
```

In [10]:
```
data["Brand"].unique()
```

Out[10]:
```
array(['TOSHIBA', 'TCL ', 'realme ', 'Mi ', 'OnePlus', 'Hisense', 'LG ',
       'MarQ by Flipkart', 'iFFALCON by TCL', 'Coocaa ', 'SAMSUNG',
       'Infinix', 'Vu', 'Nokia ', 'Thomson', 'SONY ', 'KODAK ',
       'MOTOROLA', 'PHILIPS', 'Acer', 'Blaupunkt', 'Adsun', 'T-Series',
       'Panasonic', 'Micromax', 'Sansui ', 'Croma ', 'Candes ', 'Dyanora',
       'Haier ', 'Onida', 'RGL ', 'Lloyd ', 'LumX ', 'Onix ', 'IMPEX ',
       'BPL ', 'CloudWalker', 'Oxygen ', 'Power Guard', 'Akai ', 'VG ',
       'Sun King', 'Compaq ', 'HUIDI ', 'Intex ', 'DETEL ', 'JVC',
       'G-TEN ', 'Skyworth', 'Maser ', 'Sanyo ', 'MURPHY ', 'Samsung',
       'Dektron', 'Sharp ', 'KRISONS', 'Weston ', 'AISEN  '], dtype=object)
```

In [11]:
```
data["Resolution"].unique()
```

Out[11]:
```
array(['Ultra HD LED', 'QLED Ultra HD', 'HD LED', 'Full HD LED',
       'HD Plasma'], dtype=object)
```

In [12]:
```
data["Operating System"]=data["Operating System"].fillna(data["Operating System"].mode()[0])
```

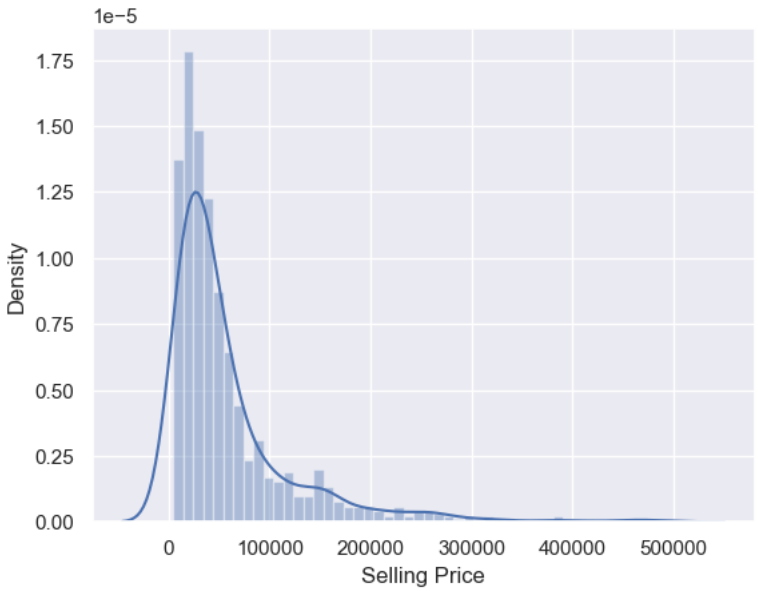
In [13]:
```
data.isnull().sum()
```

Out[13]:
```
Brand              0
Resolution         0
Size               0
Selling Price      0
Operating System   0
dtype: int64
```

In [14]:
```
sns.distplot(data['Selling Price'])
```

```
C:\ProgramData\Anaconda3\lib\site-packages\seaborn\distributions.py:2619: FutureWarning: `distplot` is a deprecated function an
d will be removed in a future version. Please adapt your code to use either `displot` (a figure-level function with similar fle
xibility) or `histplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
```

Out[14]:
```
<AxesSubplot:xlabel='Selling Price', ylabel='Density'>
```



In [15]:
```
# Outliers are a great issue for OLS, thus we must deal with them in some way
# It may be a useful exercise to try training a model without removing the outliers

# Let's declare a variable that will be equal to the 99th percentile of the 'Price' variable
q = data['Selling Price'].quantile(0.99)
# Then we can create a new df, with the condition that all prices must be below the 99 percentile of 'Price'
Data = data[data['Selling Price']<q]
# In this way we have essentially removed the top 1% of the data about 'Price'
Data.describe(include='all')
```

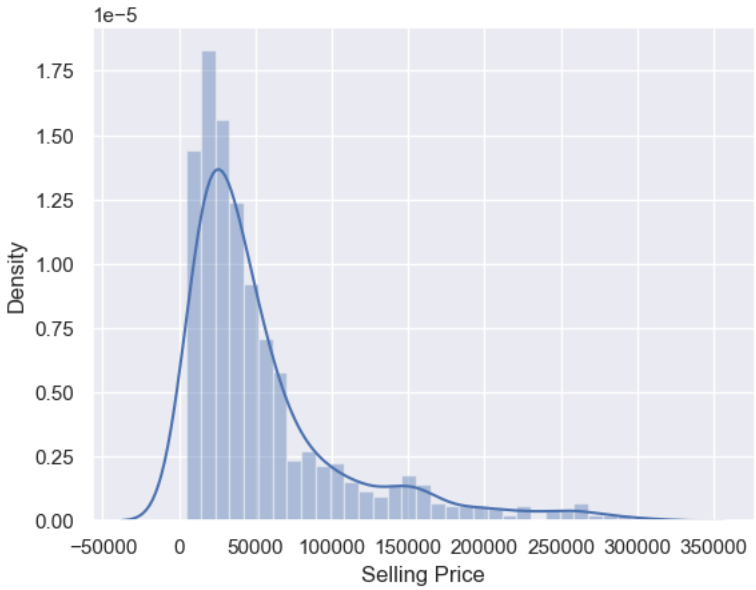
Out[15]:

	Brand	Resolution	Size	Selling Price	Operating System
count	902	902	902.000000	902.000000	902
unique	59	5	NaN	NaN	7
top	SAMSUNG	Ultra HD LED	NaN	NaN	Android
freq	136	395	NaN	NaN	483
mean	NaN	NaN	45.703991	55498.971175	NaN
std	NaN	NaN	12.155758	54634.672465	NaN
min	NaN	NaN	17.000000	4849.000000	NaN
25%	NaN	NaN	32.000000	19600.000000	NaN
50%	NaN	NaN	43.000000	36547.000000	NaN
75%	NaN	NaN	55.000000	65967.500000	NaN
max	NaN	NaN	85.000000	314900.000000	NaN

In [16]:
```
sns.distplot(Data['Selling Price'])
```

```
C:\ProgramData\Anaconda3\lib\site-packages\seaborn\distributions.py:2619: FutureWarning: `distplot` is a deprecated function an
d will be removed in a future version. Please adapt your code to use either `displot` (a figure-level function with similar fle
xibility) or `histplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
```

Out[16]: <AxesSubplot:xlabel='Selling Price', ylabel='Density'>



In [17]:
```
Data
```

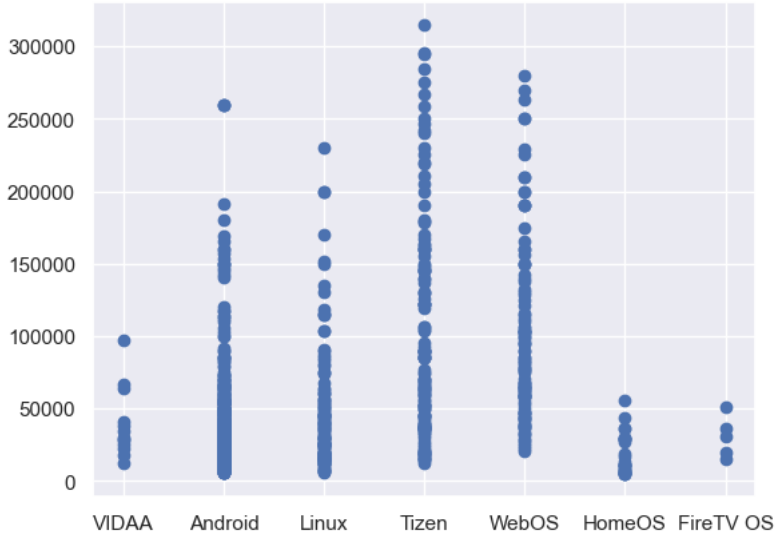
Out[17]:

	Brand	Resolution	Size	Selling Price	Operating System
0	TOSHIBA	Ultra HD LED	55	37999	VIDAA
1	TCL	QLED Ultra HD	55	52999	Android
2	realme	HD LED	32	13999	Android
3	Mi	HD LED	32	14999	Android
4	realme	HD LED	32	12999	Android
...	...	...	...	...	...
907	SONY	Full HD LED	43	44999	Linux
908	SONY	Full HD LED	40	41499	Linux
909	SONY	Ultra HD LED	65	149990	Linux
910	SONY	HD LED	32	32900	Linux
911	SONY	Full HD LED	43	56900	Linux

902 rows × 5 columns

In [18]:
```
#sns.distplot(Data['Size'])
```

In [19]:
```
x=Data["Operating System"]
y=Data["Selling Price"]

#colors = np.array(["red","green","blue","yellow","pink","black","orange","purple","beige","brown","gray","cyan","magenta"])

plt.scatter(x, y)
#plt.colorbar()

plt.show()
```



In [20]:
```
x=Data["Resolution"]
y=Data["Selling Price"]

plt.scatter(x, y)
#plt.colorbar()

plt.show()
```

In [21]:
```
Data
```

Out[21]:

	Brand	Resolution	Size	Selling Price	Operating System
0	TOSHIBA	Ultra HD LED	55	37999	VIDAA
1	TCL	QLED Ultra HD	55	52999	Android
2	realme	HD LED	32	13999	Android
3	Mi	HD LED	32	14999	Android
4	realme	HD LED	32	12999	Android
...	...	...	...	...	...
907	SONY	Full HD LED	43	44999	Linux
908	SONY	Full HD LED	40	41499	Linux
909	SONY	Ultra HD LED	65	149990	Linux
910	SONY	HD LED	32	32900	Linux
911	SONY	Full HD LED	43	56900	Linux

902 rows × 5 columns

In [22]:
```
data = pd.get_dummies(Data, drop_first=True)
```

In [23]:
```
data
```

Out[23]:

	Size	Selling Price	Brand_Acer	Brand_Adsun	Brand_Akai	Brand_BPL	Brand_Blaupunkt	Brand_Candes	Brand_CloudWalker	Brand_Compaq	...	Resolution_HI LEI
0	55	37999	0	0	0	0	0	0	0	0	...	
1	55	52999	0	0	0	0	0	0	0	0	...	
2	32	13999	0	0	0	0	0	0	0	0	...	
3	32	14999	0	0	0	0	0	0	0	0	...	
4	32	12999	0	0	0	0	0	0	0	0	...	
...	...	...	...	...	...	...	...	...	...	...	...	
907	43	44999	0	0	0	0	0	0	0	0	...	
908	40	41499	0	0	0	0	0	0	0	0	...	
909	65	149990	0	0	0	0	0	0	0	0	...	
910	32	32900	0	0	0	0	0	0	0	0	...	
911	43	56900	0	0	0	0	0	0	0	0	...	

902 rows × 70 columns

In [24]:
```
# The target(s) (dependent variable) is 'Selling price'
targets = data['Selling Price']

# The inputs are everything BUT the dependent variable, so we can simply drop it
inputs = data.drop(['Selling Price'],axis=1)
```

In [25]:
```
inputs
```

Out[25]:

	Size	Brand_Acer	Brand_Adsun	Brand_Akai	Brand_BPL	Brand_Blaupunkt	Brand_Candes	Brand_CloudWalker	Brand_Compaq	Brand_Coocaa	...	Resol
0	55	0	0	0	0	0	0	0	0	0	...	
1	55	0	0	0	0	0	0	0	0	0	...	
2	32	0	0	0	0	0	0	0	0	0	...	
3	32	0	0	0	0	0	0	0	0	0	...	
4	32	0	0	0	0	0	0	0	0	0	...	
...	...	...	...	...	...	...	...	...	...	...	...	
907	43	0	0	0	0	0	0	0	0	0	...	
908	40	0	0	0	0	0	0	0	0	0	...	
909	65	0	0	0	0	0	0	0	0	0	...	
910	32	0	0	0	0	0	0	0	0	0	...	
911	43	0	0	0	0	0	0	0	0	0	...	

902 rows × 69 columns

In [26]:
```
# # Import the scaling module
# from sklearn.preprocessing import StandardScaler

# # Create a scaler object
# scaler = StandardScaler()
# # Fit the inputs (calculate the mean and standard deviation feature-wise)
# scaler.fit(inputs)
```

In [27]:
```
# Import the module for the split
from sklearn.model_selection import train_test_split

# Split the variables with an 80-20 split and some random state
# To have the same split as mine, use random_state = 365
x_train, x_test, y_train, y_test = train_test_split(inputs, targets, test_size=0.2, random_state=365)
```

In [28]:
```
# Create a linear regression object
from sklearn.linear_model import LinearRegression
reg = LinearRegression()
# Fit the regression with the scaled TRAIN inputs and targets
reg.fit(x_train,y_train)
```

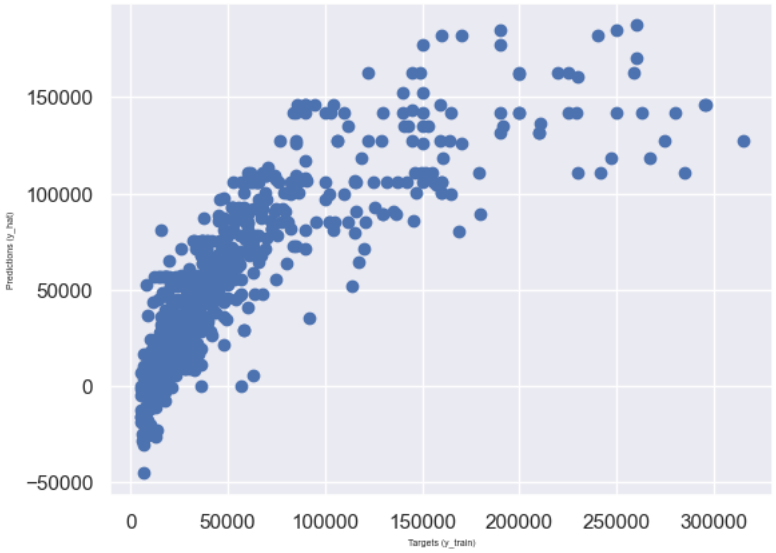
Out[28]:
```
LinearRegression()
```

In [29]:
```
# Let's check the outputs of the regression
# I'll store them in y_hat as this is the 'theoretical' name of the predictions
y_hat = reg.predict(x_train)
```

In [30]:
```
# y_hat
```

In [31]:
```
# y_train
```

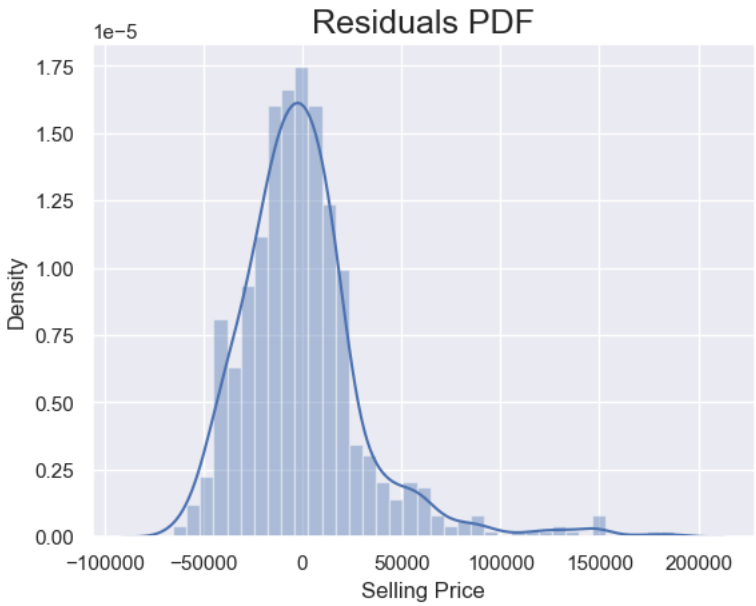
In [32]:
```
# The simplest way to compare the targets (y_train) and the predictions (y_hat) is to plot them on a scatter plot
# The closer the points to the 45-degree line, the better the prediction
plt.scatter(y_train, y_hat)
# Let's also name the axes
plt.xlabel('Targets (y_train)',size=5)
plt.ylabel('Predictions (y_hat)',size=5)
# Sometimes the plot will have different scales of the x-axis and the y-axis
# This is an issue as we won't be able to interpret the '45-degree line'
# We want the x-axis and the y-axis to be the same
# plt.xlim(6,13)
# plt.ylim(6,13)
plt.show()
```



In [33]:
```
# Another useful check of our model is a residual plot
# We can plot the PDF of the residuals and check for anomalies
sns.distplot(y_train - y_hat)

# Include a title
plt.title("Residuals PDF", size=18)

# In the best case scenario this plot should be normally distributed
# In our case we notice that there are many negative residuals (far away from the mean)
# Given the definition of the residuals (y_train - y_hat), negative values imply
# that y_hat (predictions) are much higher than y_train (the targets)
# This is food for thought to improve our model
```

```
C:\ProgramData\Anaconda3\lib\site-packages\seaborn\distributions.py:2619: FutureWarning: `distplot` is a deprecated function an
d will be removed in a future version. Please adapt your code to use either `displot` (a figure-level function with similar fle
xibility) or `histplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
```

Out[33]: Text(0.5, 1.0, 'Residuals PDF')



In [34]:
```
# Find the R-squared of the model
reg.score(x_train,y_train)

# Note that this is NOT the adjusted R-squared
# in other words... find the Adjusted R-squared to have the appropriate measure :)
```

Out[34]: 0.6706072699905933

In [35]:
```
x_train
```

Out[35]:

	Size	Brand_Acer	Brand_Adsun	Brand_Akai	Brand_BPL	Brand_Blaupunkt	Brand_Candes	Brand_CloudWalker	Brand_Compaq	Brand_Coocaa	...	Resol
897	24	0	0	0	0	0	0	0	0	0	...	
423	49	0	0	0	0	0	0	0	0	0	...	
521	49	0	0	0	0	0	0	0	0	0	...	
37	43	0	0	0	0	0	0	0	0	0	...	
908	40	0	0	0	0	0	0	0	0	0	...	
...	...	...	...	...	...	...	...	...	...	...	...	...
434	40	0	0	0	0	0	1	0	0	0	...	
868	43	0	0	0	0	0	0	0	0	0	...	
810	24	0	0	0	0	0	0	0	0	0	...	
699	49	0	0	0	0	0	0	0	0	0	...	
601	65	0	0	0	0	0	0	0	0	0	...	

721 rows × 69 columns

In [36]:
```
reg.predict([[34,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
```

```
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\base.py:450: UserWarning: X does not have valid feature names, but LinearReg
ression was fitted with feature names
  warnings.warn(
```

Out[36]:
```
array([-67618.06314408])
```

In [37]:
```
# Obtain the bias (intercept) of the regression
reg.intercept_
```

Out[37]:
```
-123149.29754643698
```

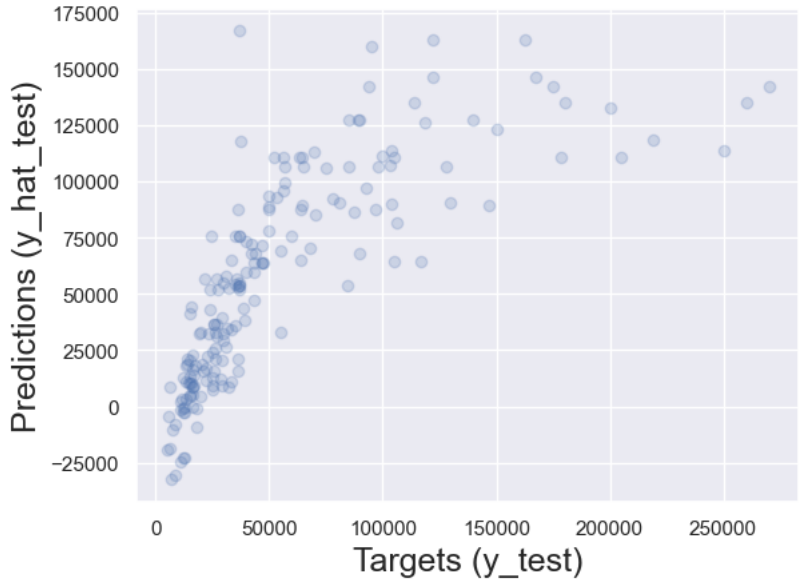
In [38]:
```
# Obtain the weights (coefficients) of the regression
reg.coef_

# Note that they are barely interpretable if at all
```

Out[38]:
```
array([ 3.55994887e+03, -5.64064363e+03, -1.50526720e+04, -1.87415102e+04,
       -8.76454673e+03, -1.08254327e+04,  9.29202227e+03, -1.16913773e+04,
       -6.07183546e+03, -1.23407864e+04,  6.96845567e+03, -1.09011321e+04,
        4.29933640e+04,  4.81302181e+03, -1.65600823e+04, -8.75289001e+03,
       -3.64947200e+03, -1.03282033e+04,  2.86975986e+03, -2.87727023e+03,
       -1.07289040e+04, -8.64236659e+03, -7.89483955e+03,  4.23800020e+04,
        3.09689782e+04,  1.58428592e+04,  1.84889003e+03, -1.15419408e+04,
        2.34866029e+04,  7.32584039e+03,  2.50269414e+04, -5.64968280e+03,
        1.31448264e+04, -1.40382493e+04, -8.96494132e+03,  9.40030070e+03,
        5.15724744e+03,  5.58628917e+03, -2.46525180e+02,  1.51392012e+04,
       -5.00630146e+03,  3.45682157e+04,  3.19388523e+04,  3.43112799e+04,
        9.82712194e+04,  6.23527024e+03, -1.68748414e+00,  7.93566398e+04,
        2.61869414e+04,  1.40473504e+04, -2.30145966e+03, -5.65142432e+02,
       -6.68037215e+03, -1.33020221e+04, -9.90507763e+03, -1.22919644e+03,
        5.78796673e+03, -1.29722568e+04,  6.59608074e+03,  2.01825833e+04,
       -6.70377461e+04,  9.13915483e+03, -7.51579304e+03, -1.35087562e+04,
       -1.80703388e+03, -8.95156318e+03,  1.36957127e+04, -6.68037215e+03,
        1.01730855e+04])
```

In [40]:
```
# # Create a regression summary where we can compare them with one-another
# reg_summary = pd.DataFrame(data.columns.values, columns=['Features'])
# reg_summary['Weights'] = reg.coef_
# reg_summary
```

In [41]:
```
# Once we have trained and fine-tuned our model, we can proceed to testing it
# Testing is done on a dataset that the algorithm has never seen
# Luckily we have prepared such a dataset
# Our test inputs are 'x_test', while the outputs: 'y_test'
# We SHOULD NOT TRAIN THE MODEL ON THEM, we just feed them and find the predictions
# If the predictions are far off, we will know that our model overfitted
y_hat_test = reg.predict(x_test)
```

In [42]:
```
# Create a scatter plot with the test targets and the test predictions
# You can include the argument 'alpha' which will introduce opacity to the graph
plt.scatter(y_test, y_hat_test, alpha=0.2)
plt.xlabel('Targets (y_test)',size=18)
plt.ylabel('Predictions (y_hat_test)',size=18)
# plt.xlim(6,13)
# plt.ylim(6,13)
plt.show()
```



In [43]:
```
# After displaying y_test, we find what the issue is
# The old indexes are preserved (recall earlier in that code we made a note on that)
# The code was: data_cleaned = data_4.reset_index(drop=True)

# Therefore, to get a proper result, we must reset the index and drop the old indexing
y_test = y_test.reset_index(drop=True)

# Check the result
y_test.head()
```

Out[43]:
```
0     31999
1     38590
2     34999
3    149999
4    167199
Name: Selling Price, dtype: int64
```

In [44]:
```
# Finally, let's manually check these predictions
# To obtain the actual prices, we take the exponential of the log_price
df_pf = pd.DataFrame((y_hat_test), columns=['Prediction'])
df_pf.head()
```

Out[44]:

	Prediction
0	52791.310854
1	43682.880921
2	35790.412086
3	123358.817840
4	146366.150980

In [45]:
```
# Let's overwrite the 'Target' column with the appropriate values
# Again, we need the exponential of the test log price
df_pf['Target'] = (y_test)
df_pf
```

Out[45]:

	Prediction	Target
0	52791.310854	31999
1	43682.880921	38590
2	35790.412086	34999
3	123358.817840	149999
4	146366.150980	167199
...	...	...
176	110766.662278	51999
177	90491.814020	80799
178	89406.969056	146900
179	20730.415514	28999
180	10951.649568	12990

181 rows × 2 columns

In [ ]:

In [ ]: