In [66]:
```
import pandas as pd
from matplotlib import pyplot as plt
import seaborn as sns
import scipy.stats as stats
import pylab
from scipy.stats import shapiro
import pingouin as pg
import numpy as np
from plotly import express as px
from plotly import graph_objects as go
from statsmodels.graphics.gofplots import qqplot
%matplotlib inline
import warnings
warnings.filterwarnings('ignore')
```

In [101…
```
data = pd.read_excel("hccap.xlsx")
data
```

Out[101]:

	Hierro	Cap	China
0	74.37	6230.8	13195.72
1	87.48	6229.0	10065.20
2	87.41	6440.1	10136.83
3	87.57	6748.7	10133.70
4	87.89	7048.9	10083.43
...	...	...	...
1442	87.51	6061.0	12294.14
1443	88.19	6283.0	12490.61
1444	90.79	6689.0	12418.61
1445	92.09	6201.0	12296.42
1446	92.29	6249.0	12269.57

1447 rows × 3 columns

In [68]:
```
data.shape
```

Out[68]:
```
(1447, 3)
```

In [69]:
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1447 entries, 0 to 1446
Data columns (total 3 columns):
 #   Column  Non-Null Count  Dtype  
---  ------  --------------  -----  
 0   Hierro  1447 non-null   float64
 1   Cap     1447 non-null   float64
 2   China   1447 non-null   float64
dtypes: float64(3)
memory usage: 34.0 KB
```

In [70]:
```
data.head()
```

Out[70]:

	Hierro	Cap	China
0	74.37	6230.8	13195.72
1	87.48	6229.0	10065.20
2	87.41	6440.1	10136.83
3	87.57	6748.7	10133.70
4	87.89	7048.9	10083.43

In [71]:
```
data.keys()
```

Out[71]:
```
Index(['Hierro', 'Cap', 'China'], dtype='object')
```

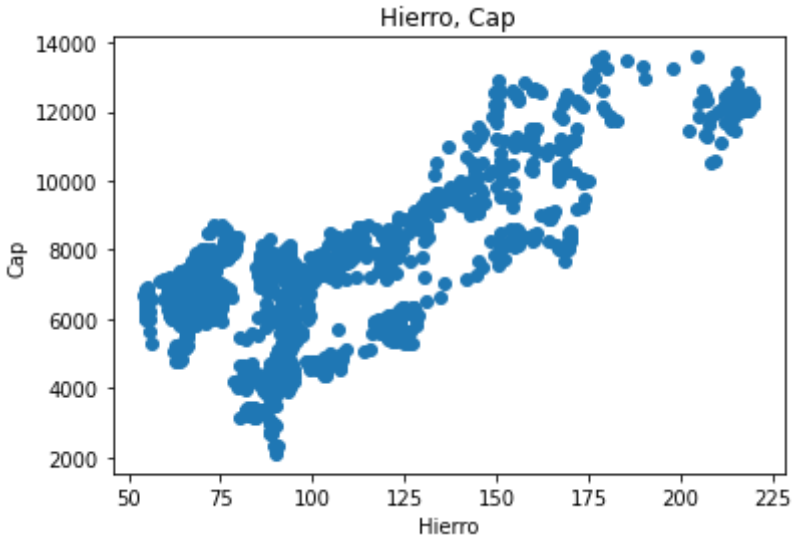
In [72]:
```
data.describe()
```

Out[72]:

	Hierro	Cap	China
count	1447.000000	1447.000000	1447.000000
mean	104.222674	7299.305121	13649.044651
std	38.473160	2169.688284	2094.550561
min	54.010000	2115.650000	10045.990000
25%	74.100000	6031.295000	12102.780000
50%	92.950000	7062.700000	13494.060000
75%	122.850000	8139.300000	15129.170000
max	219.770000	13583.950000	20134.900000

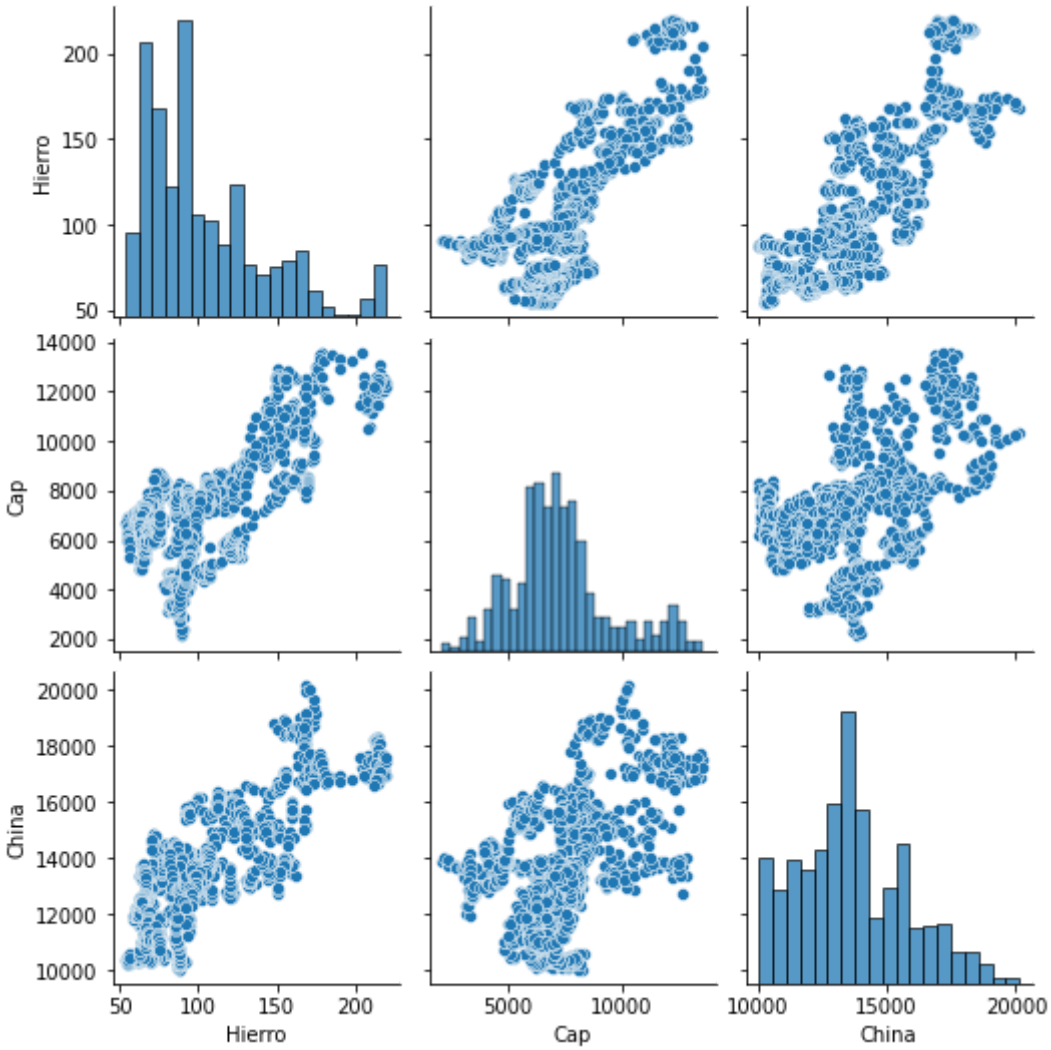
In [73]:
```
plt.scatter(data['Hierro'],data ['Cap'])
plt.title ("Hierro, Cap")
plt.xlabel ("Hierro")
plt.ylabel ("Cap")
```

Out[73]:
```
Text(0, 0.5, 'Cap')
```



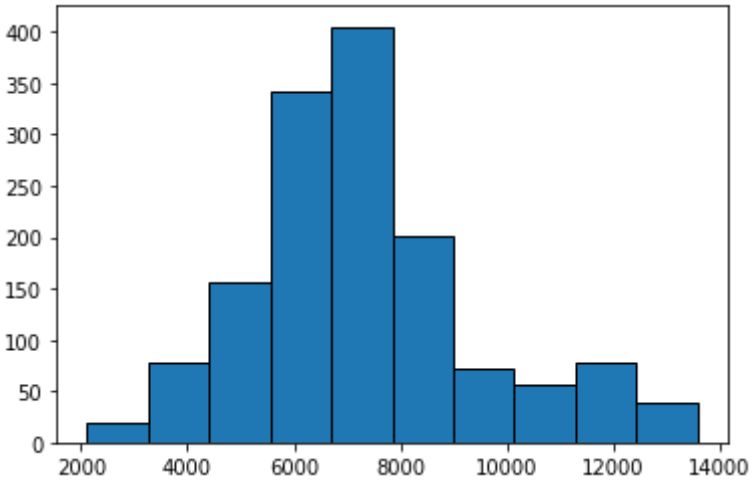
In [74]:
```
sns.pairplot(data)
```

Out[74]:
```
<seaborn.axisgrid.PairGrid at 0x1254d35e0>
```



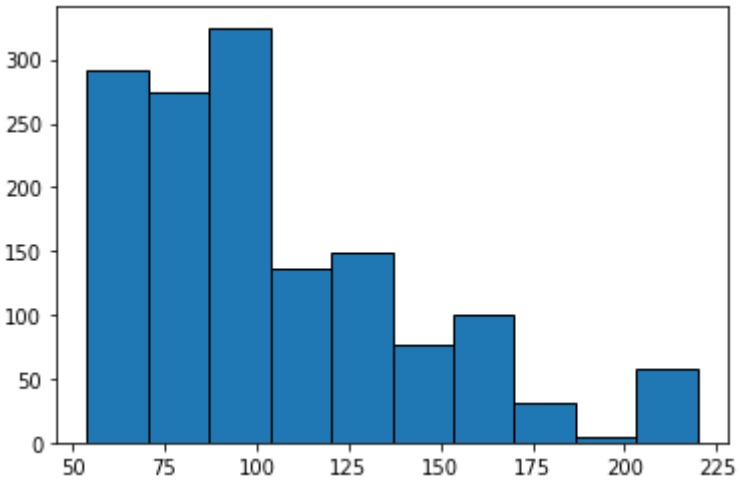
In [75]:
```
plt.hist(data['Cap'], edgecolor = 'black', linewidth=1)
```

Out[75]:
```
(array([ 20.,  78., 156., 341., 405., 202.,  71.,  57.,  78.,  39.]),
 array([ 2115.65,  3262.48,  4409.31,  5556.14,  6702.97,  7849.8 ,
         8996.63, 10143.46, 11290.29, 12437.12, 13583.95]),
 <BarContainer object of 10 artists>)
```



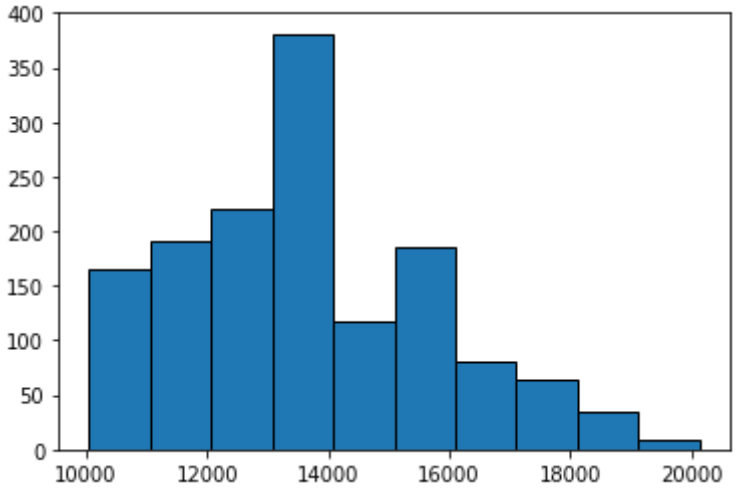
In [76]:
```
plt.hist(data['Hierro'], edgecolor = 'black', linewidth=1)
```

Out[76]:
```
(array([292., 275., 325., 137., 149.,  76., 100.,  31.,   4.,  58.]),
 array([ 54.01 ,  70.586,  87.162, 103.738, 120.314, 136.89 , 153.466,
        170.042, 186.618, 203.194, 219.77 ]),
 <BarContainer object of 10 artists>)
```

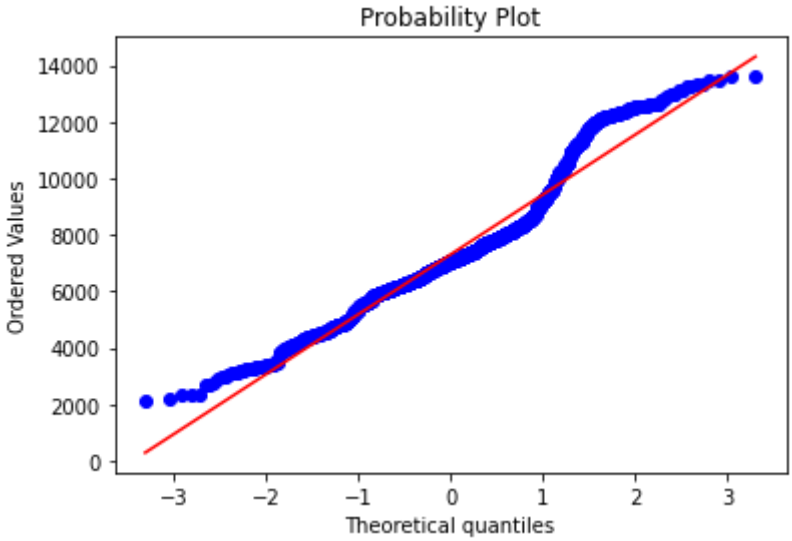


In [77]:
```
plt.hist(data['China'], edgecolor = 'black', linewidth=1)
```

Out[77]:
```
(array([166., 191., 220., 381., 117., 185.,  80.,  64.,  35.,   8.]),
 array([10045.99 , 11054.881, 12063.772, 13072.663, 14081.554, 15090.445,
        16099.336, 17108.227, 18117.118, 19126.009, 20134.9  ]),
 <BarContainer object of 10 artists>)
```



In [78]:
```
stats.probplot (data['Cap'], dist="norm", plot=pylab)
pylab.show()
```



In [79]:
```
estadistico, p_value = shapiro(data['Cap'])
print('Estadisticos=%.3f, p=%.3f' %  (estadistico, p_value))
```

```
Estadisticos=0.952, p=0.000
```

In [80]:
```
data_corr = data.corr(method='pearson')
data_corr
```

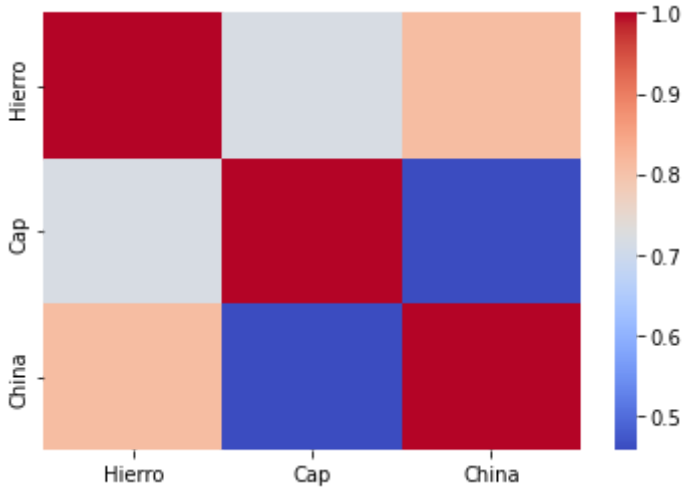
Out[80]:

	Hierro	Cap	China
Hierro	1.000000	0.716656	0.810277
Cap	0.716656	1.000000	0.458379
China	0.810277	0.458379	1.000000

In [81]:
```
#Mapa de calor
sns.heatmap(data_corr,
            xticklabels=data_corr.columns,
```

```
            yticklabels=data_corr.columns,
            cmap='coolwarm'
           )
```

Out[81]: <AxesSubplot:>



In [82]:
```
#Matriz de correlación
sns.heatmap(data.corr(), annot=True)
plt.show
```

Out[82]: <function matplotlib.pyplot.show(close=None, block=None)>

In [83]:
```
corr = pg.pairwise_corr(data, method='pearson')
corr.sort_values(by=['p-unc'])[['X', 'Y', 'n', 'r', 'p-unc']]
```

Out[83]:

	X	Y	n	r	p-unc
1	Hierro	China	1447	0.810277	0.000000e+00
0	Hierro	Cap	1447	0.716656	2.099831e-228
2	Cap	China	1447	0.458379	4.475420e-76

In [102…
```
from sklearn.preprocessing import LabelEncoder
```

```
encoder = LabelEncoder()
```

In [103…

```
X = data.drop(['Cap'], axis=1)
y = data['Cap']

X.shape, y.shape
```

Out[103]:

```
((1447, 2), (1447,))
```

In [104…

```
from sklearn.model_selection import train_test_split
from sklearn.metrics import r2_score, mean_squared_error, mean_absolute_error
from sklearn.ensemble import RandomForestRegressor
```

In [105…

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, rando
```

In [106…

```
from sklearn.ensemble import RandomForestRegressor
regressor = RandomForestRegressor(n_estimators = 1000, random_state = 42)
regressor.fit(X_train, y_train)
```

Out[106]:

```
RandomForestRegressor
RandomForestRegressor(n_estimators=1000, random_state=42)
```

In [107…

```
#Modelación regresión random forest
```

In [113…

```
model_rf = RandomForestRegressor(random_state=42)
model_rf.fit(X_train, y_train)
pred_rf = model_rf.predict(X_test)
#predicción modelo
pred_rf = model_rf.predict(X_test)
print(pred_rf)
print(pred_rf.shape)
```

```
[ 8066.946   6306.3258 11195.64    6153.8945  6222.8459  6797.2864
 12808.6248  7793.1974  8127.0255  3947.584   5577.3406  7726.353
  6898.555   6176.3809  7403.013   6375.7481  5356.824   6596.8154
  6860.386   6926.0684 12035.5132  8111.9801 13180.3107  7700.019
 11170.9241  7021.3728  6837.714   7303.4311  8406.7588  6344.2512
  8534.1028  6932.7287  6519.2321  7006.9168  8079.081   8171.2154
  7266.167   6745.625   8006.362   6758.906   6321.5914  6650.913
  8540.4596 10440.43    7404.8906  9878.8888 12386.3878  7595.7755
  4789.8019 11003.3898  7299.3371  7205.8947  7022.2208  7365.7427
  6049.5542  7049.36    8415.5732  7986.9622  6373.9119  5368.162
  5924.9989  5125.4125  6110.7935  5749.7465  6809.546  12687.0838
  7492.2919  9542.943   5246.9141  7790.4094 10836.59    4764.3588
  6908.834   9819.763   3821.1028  7070.7508 12589.6214  5431.1627
  6969.5214  6677.0667  7751.656   7221.7799  5721.9115  6683.4921
  5714.3997  6768.058   6960.532   4396.2531  5943.7441  6078.4974
  7286.7646  4642.6814  4313.8106  6718.9756  7585.1281 12232.7785
 11418.1193  4118.2556  6549.809   7276.775   7066.475   7340.1598
  6353.5421  5018.6294 10300.1257  8532.9059  6272.9747  3511.4112
 12094.81    7085.4408  5899.2383  9575.95    7201.5172  6594.4441
  7356.207   7345.8724  5686.301   6258.4562 10707.44    8441.427
  8550.445   5376.0076  7982.351   8752.7402  6222.595   6293.1582
  7253.4682  7636.8537  5658.9328  7171.7016 11078.204   6720.7532
  7594.9755  7298.1036  7787.338  10720.277   6687.101   3660.5861
  6573.2096  8469.6859  7106.7974  6853.666   4608.0092  8003.186
  6983.3263 12006.31   11105.2172  8409.4609  6923.0457  7114.136
  5725.4542  7688.7134  9344.964   3635.1846  8103.719   6482.109
  6470.79    5947.7872  6040.1405  7856.8183 10899.16    7975.762
  6797.797   6603.334   8219.639   7162.6177  6361.4968  7558.8899
  6854.1408  2866.6477  5903.7448  5735.2096  5341.1358  6559.7702
  7148.9617  6575.651  11118.04    9375.179  10711.21    4321.2063
  7350.9863  4455.0585 11523.3587  7802.2384  6244.9204  9749.61
  4041.6216  6140.3622  4572.3517  7294.7656  6217.3187 12518.3908
  6458.247   5404.8167  8510.2838  7780.675   6381.9758  6728.909
  6206.532   7237.95    5412.056  12112.5395  6619.8324  6110.9701
  4968.0624  8434.646   7058.268   6734.745   7148.2862 12456.631
  5507.7509  6217.1383  7880.219   6884.2921  6183.8037  6181.1831
 10880.      7112.1876  7764.8943  3113.3765  6830.395   6442.8106
  6642.148   6859.3645  7312.8697  7195.1254  6631.377   6387.062
  7718.273   7018.305   7617.8342  6171.5065  6881.15    6875.901
  5513.2024  7179.655   6139.7955  7063.957   3614.8393  6728.2573
 11938.4538 12125.4317 10106.867  12124.1496  6549.0617 12344.07
  7888.084   7170.7016 12150.02   12233.4981  6929.6346  6718.2231
  7356.3783  8148.2581  8094.784   4881.9616  8047.3771  6734.5828
  6411.675   7645.8195  7226.629   7661.976   4497.692   8415.4776
  6272.712   9538.72    7174.889  10338.57    7437.2108  5697.927
  6755.0288  2594.9131  6046.4175 10290.11    6016.579   6795.2502
  6550.6895  6162.0727  5556.7174  7491.7256  6556.5306  7870.707
  7793.0939  6359.225   7019.7053  8832.5856 11929.7822  8000.888
  5745.8584  6709.7872  8091.2095  7065.712   6623.299   3768.1973
  7762.06    8777.369   9600.791   5858.0527  7413.4385  6129.5705
  7073.257  11213.6083  8343.1021  9542.943   7425.8627  9905.7837
 12460.3717  7356.4084  9235.168   6624.6103  6193.9371  6365.8064
  6846.814   5983.295   4276.7177  7842.522   8112.374   6816.213
  7036.6514  6886.188  11996.8842  6868.1837 11198.56    4167.8232
 10365.1702  6917.7502  6289.936   4987.6949  7116.8804  6526.4672
  6308.4767  6531.8571  9429.88   11484.48    5449.4377  8415.5345
  6446.1824  6253.67    5487.2617  9344.921   7931.733   7099.3329
  5736.2232 11219.6331  4090.0348  5590.328   6181.8923 11759.33
 11750.813   7254.0714  8632.3583  3324.8968  8081.5401  6723.8321
  6538.1151  6910.5701  7412.9321  7548.4351  9479.027   6558.4486
```

```
 6676.42    7240.1237  9375.534  13078.0382  6689.6161 10602.4335
 7695.187   8324.5757  6107.3107  7719.404   4187.3807  6290.511
 8448.506   6435.136   6120.4577  5909.9037  7896.529   7323.4126
 8264.4798  4439.8959  6820.805  11687.14    6219.0786  7144.146
 6552.9632  8323.694  12171.7891  2833.6843  5919.1317  6321.3936
 6878.707   7795.6733  4413.2183  6590.621   6657.5313  7115.079
 7037.353   5560.9769  3660.8651  6159.8416  7018.641   6219.5365
 6621.696   4509.6605  7454.994   6398.303  12409.9897  4379.9414
 7584.6544 10020.9923 12251.2228  7401.8423  7907.734   5315.5416
10087.2794 10110.07    6998.7993  6196.208   5625.9634  7125.8166
 4421.2451 11963.2798  6600.2742  5890.9627  5838.0036 12112.9689
12327.5179  6852.798  11856.57    4162.5081  6434.6935  7387.8282
 8315.0502  6822.8393 10513.1339]
(435,)
```

In [109…

```
y_pred = model_rf.predict(X_test)
```

In [110…

```
#Comparación datos reales con proyectados
data=pd.DataFrame({'Cap':y_test, 'Predicción':y_pred})
data
```

Out[110]:

	Cap	Predicción
227	8083.20	8066.9460
481	6447.54	6306.3258
1288	12540.00	11195.6400
930	5553.33	6153.8945
1439	5556.10	6222.8459
...	...	...
431	6765.01	6434.6935
1444	6689.00	7387.8282
996	8501.47	8315.0502
1386	7531.69	6822.8393
1139	10535.00	10513.1339

435 rows × 2 columns

In [114…

```
from sklearn import metrics
print('Testing R2 Score: ', r2_score(y_test, pred_rf)*100)
print('Mean Absolute Error:', metrics.mean_absolute_error(y_test, y_pred))
print('Mean Squared Error:', metrics.mean_squared_error(y_test, y_pred))
print('Root Mean Squared Error:', np.sqrt(metrics.mean_squared_error(y_test, y
mape = 100 * (errors / y_test)
accuracy = 100 - np.mean(mape)
print('Certeza:', round(accuracy, 2), '%.')
```

```
Testing R2 Score:  88.63837232044065
Mean Absolute Error: 490.7407489655174
Mean Squared Error: 517387.51050693664
Root Mean Squared Error: 719.2965386451798
Certeza: 92.8 %.
```

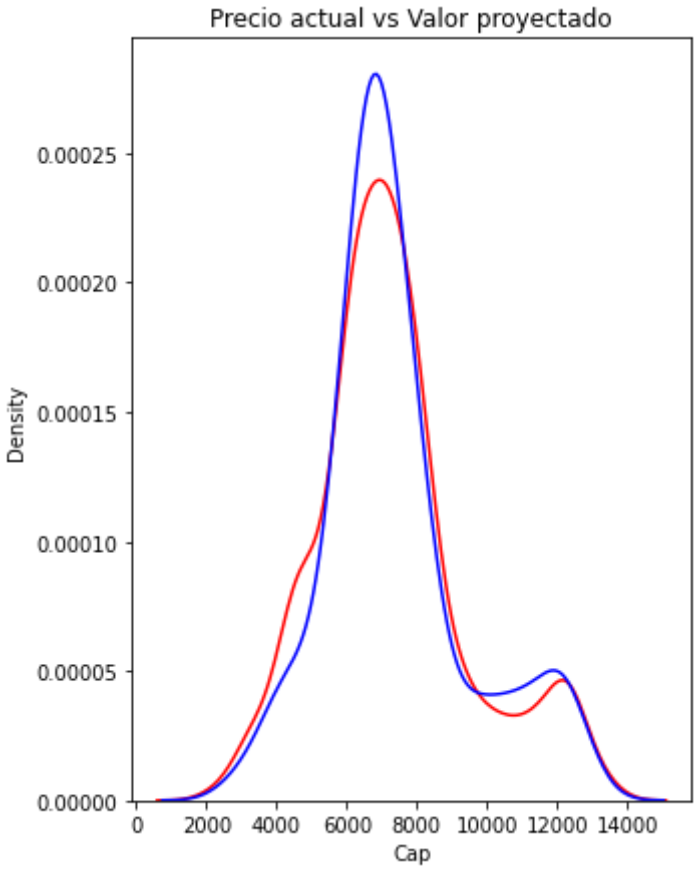
In [97]:

```
plt.figure(figsize=(5, 7))


ax = sns.distplot(y, hist=False, color="r", label="Precio actual")
sns.distplot(y_pred, hist=False, color="b", label="Valor proyectado" , ax=ax)


plt.title('Precio actual vs Valor proyectado')


plt.show()
plt.close()
```



In [98]:

```
import pydot
# Pull out one tree from the forest
Tree = regressor.estimators_[5]
# Export the image to a dot file
from sklearn import tree
plt.figure(figsize=(700,50))
tree.plot_tree(Tree,filled=True,
               rounded=True,
               fontsize=10);
```

In [115…

```
#Proyección precio CAP, con valores de hierro y China 17-11-2022 y 18-11-2022
prueba= pd.DataFrame({"Hierro":[93.27, 92.34], "China A50":[12269.42, 12269.57
prueba
```

Out[115]:

	Hierro	China A50
0	93.27	12269.42
1	92.34	12269.57

In [116…

```
#Resultados
#Precio reales de Cap para las fechas proyectadas: 6201, 6310
print(model_rf.predict(prueba))
```

```
[5803.0815 6174.9177]
```