Documento de infraestructura

Addison Amin Reyes Cedano, 2021-2026

La idea de mi proyecto es la siguiente, dado un dataframe de caracteristicas acerca de celulares predecir su precio gracia a modelos de regresion de sklearn. Lo primero fue elegir el dataframe que es este, aquí el codigo de las librerias y abriendo el dataframe en colab.

Dataset: https://www.kaggle.com/datasets/iabhishekofficial/mobile-price-classification?select=train.csv

Librerias

```
1 pip -q install dataprep
```

```
                                        ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 9.9/9.9 MB 25.3 MB/s eta 0:00:00
                                        ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 1.3/1.3 MB 15.4 MB/s eta 0:00:00
                                        ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 764.7/764.7 KB 7.8 MB/s eta 0:00:00
                                        ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 34.5/34.5 MB 13.3 MB/s eta 0:00:00
                                        ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 1.2/1.2 MB 36.6 MB/s eta 0:00:00
                                        ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 2.2/2.2 MB 21.5 MB/s eta 0:00:00
                                        ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 133.6/133.6 KB 15.8 MB/s eta 0:00:00
                                        ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 1.0/1.0 MB 55.6 MB/s eta 0:00:00
                                        ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 1.0/1.0 MB 42.8 MB/s eta 0:00:00
  Preparing metadata (setup.py) ... done
                                        ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 49.6/49.6 KB 6.0 MB/s eta 0:00:00
                                        ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 1.6/1.6 MB 87.1 MB/s eta 0:00:00
  Building wheel for metaphone (setup.py) ... done
ERROR: pip's dependency resolver does not currently take into account all the packages that are
distributed 2022.2.1 requires dask==2022.02.1, but you have dask 2023.2.1 which is incompatible.
```

```
[2]  1 pip -q install shap
```

```
                                        ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 575.9/575.9 KB 10.1 MB/s eta 0:00:00
```

```
[3]  1 from sklearn.linear_model import LinearRegression, Ridge, BayesianRidge, Lasso
     2 from sklearn.model_selection import train_test_split, cross_val_score
     3 from sklearn.preprocessing import OneHotEncoder, StandardScaler
     4 from sklearn.compose import make_column_transformer
     5 from sklearn.ensemble import RandomForestRegressor
     6 from sklearn.neighbors import KNeighborsRegressor
     7 from sklearn.neural_network import MLPRegressor
     8 from sklearn.pipeline import make_pipeline
     9 from dataprep.eda import create_report
    10 from sklearn.metrics import r2_score
    11 from sklearn.utils import resample
    12 import matplotlib.pyplot as plt
    13 from sklearn.svm import SVR
    14 import seaborn as sns
    15 import pandas as pd
    16 import numpy as np
    17 import random
    18 import pickle
    19 import shap
```

```
[4]  1 df = pd.read_csv('/content/mobile_prices.csv')
     2 df.head(5)
```

	battery_power	blue	clock_speed	dual_sim	fc	four_g	int_memory	m_dep	mobile_wt	n_cores	...	px_height	px_width	ram	sc_h	sc_w	talk_time	three_g	touch_screen	wi
0	842	0	2.2	0	1	0	7	0.6	188	2	...	20	756	2549	9	7	19	0	0	
1	1021	1	0.5	1	0	1	53	0.7	136	3	...	905	1988	2631	17	3	7	1	1	
2	563	1	0.5	1	2	1	41	0.9	145	5	...	1263	1716	2603	11	2	9	1	1	
3	615	1	2.5	0	0	0	10	0.8	131	6	...	1216	1786	2769	16	8	11	1	0	
4	1821	1	1.2	0	13	1	44	0.6	141	2	...	1208	1212	1411	8	2	15	1	1	

5 rows × 21 columns

Luego de la recoleccion de datos inicia el proceso de preparacion y preprocesamieno de la data, lo primero que hice fue analizar el dataframe y ver cada columna, me fije que la columna de precio de rango tenia un formato de 0 a 3, pero decidi cambiar la columna a un formato de dolares desde 0 a 1000 con una funcion de precios aplicandolo al dataframe con la funcion apply y por ultimo revisando si hay valores nulos dentro del dataframe, aquí algunas fotos de lo explicado.

Preparacion y Preprocesamiento de la data

```
[5]  1 df
```

	battery_power	blue	clock_speed	dual_sim	fc	four_g	int_memory	m_dep	mobile_wt	n_cores	...	px_height	px_width	ram	sc_h	sc_w	talk_time	three_g	touch_screen
0	842	0	2.2	0	1	0	7	0.6	188	2	...	20	756	2549	9	7	19	0	0
1	1021	1	0.5	1	0	1	53	0.7	136	3	...	905	1988	2631	17	3	7	1	1
2	563	1	0.5	1	2	1	41	0.9	145	5	...	1263	1716	2603	11	2	9	1	1
3	615	1	2.5	0	0	0	10	0.8	131	6	...	1216	1786	2769	16	8	11	1	0
4	1821	1	1.2	0	13	1	44	0.6	141	2	...	1208	1212	1411	8	2	15	1	1
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
1995	794	1	0.5	1	0	1	2	0.8	106	6	...	1222	1890	668	13	4	19	1	1
1996	1965	1	2.6	1	0	0	39	0.2	187	4	...	915	1965	2032	11	10	16	1	1
1997	1911	0	0.9	1	1	1	36	0.7	108	8	...	868	1632	3057	9	1	5	1	1
1998	1512	0	0.9	0	4	1	46	0.1	145	5	...	336	670	869	18	10	19	1	1
1999	510	1	2.0	1	5	1	45	0.9	168	6	...	483	754	3919	19	4	2	1	1

2000 rows × 21 columns

```
[6]  1 df.columns
```

```
Index(['battery_power', 'blue', 'clock_speed', 'dual_sim', 'fc', 'four_g',
       'int_memory', 'm_dep', 'mobile_wt', 'n_cores', 'pc', 'px_height',
       'px_width', 'ram', 'sc_h', 'sc_w', 'talk_time', 'three_g',
       'touch_screen', 'wifi', 'price_range'],
      dtype='object')
```

```
1 df['price_range']
```

```
0       1
1       2
2       2
3       2
4       1
       ..
1995    0
1996    2
1997    3
1998    0
1999    3
Name: price_range, Length: 2000, dtype: int64
```

```
[8]  1 def precios(precio):
     2     if precio == 0:
     3         return random.randint(1, 100)
```

```python
def precios(precio):
    if precio == 0:
        return random.randint(1, 100)
    elif precio == 1:
        return random.randint(101, 300)
    elif precio == 2:
        return random.randint(301, 600)
    elif precio == 3:
        return random.randint(601, 1000)
```

```python
df['price_range']
```

```
0       200
1       520
2       392
3       482
4       184
       ...
1995     97
1996    551
1997    601
1998     43
1999    998
Name: price_range, Length: 2000, dtype: int64
```

```python
df.isna().any()
```

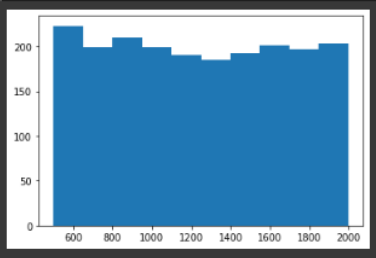
```
battery_power    False
blue             False
clock_speed      False
dual_sim         False
fc               False
four_g           False
int_memory       False
m_dep            False
mobile_wt        False
n_cores          False
pc               False
px_height        False
px_width         False
ram              False
sc_h             False
sc_w             False
talk_time        False
three_g          False
touch_screen     False
wifi             False
price_range      False
dtype: bool
```

```python
df.isna().sum()
```

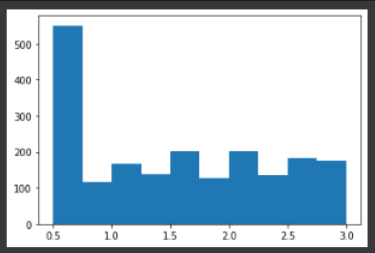
```
battery_power    0
blue             0
clock_speed      0
dual_sim         0
fc               0
four_g           0
int_memory       0
m_dep            0
mobile_wt        0
n_cores          0
pc               0
px_height        0
px_width         0
ram              0
sc_h             0
sc_w             0
talk_time        0
three_g          0
touch_screen     0
wifi             0
price_range      0
```

Una vez hecha la preparacion inice el proceso de analisis descriptivo (EDA), cree un reporte con dataprep e hice un histograma de cada columna para analizar los datos del dataframe.

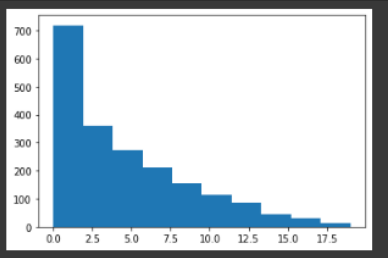
```python
plt.hist(df['battery_power'])
plt.show()
```



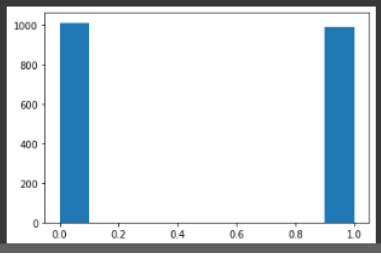
```python
plt.hist(df['clock_speed'])
plt.show()
```



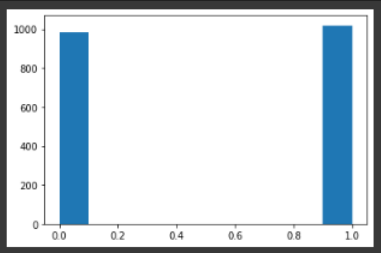
```python
plt.hist(df['fc'])
plt.show()
```



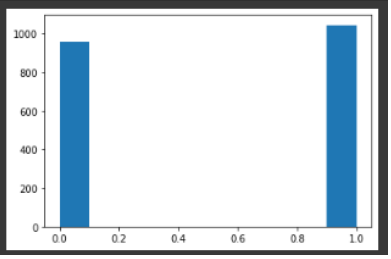
```python
plt.hist(df['blue'])
plt.show()
```



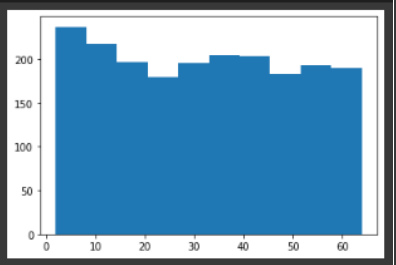
```python
plt.hist(df['dual_sim'])
plt.show()
```



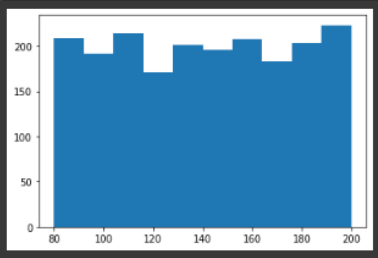
```python
plt.hist(df['four_g'])
plt.show()
```



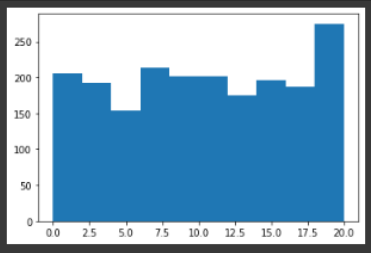
```
1 plt.hist(df['int_memory'])
2 plt.show()
```



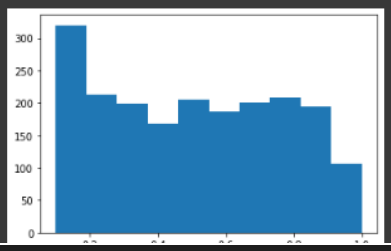
```
1 plt.hist(df['mobile_wt'])
2 plt.show()
```



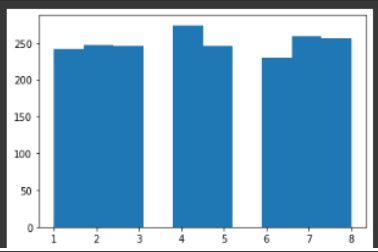
```
[24] 1 plt.hist(df['pc'])
2 plt.show()
```



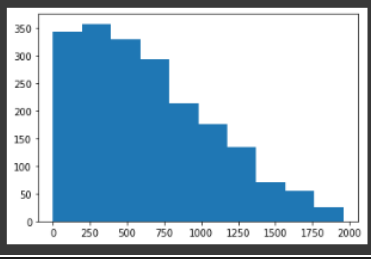
```
[21] 1 plt.hist(df['m_dep'])
2 plt.show()
```



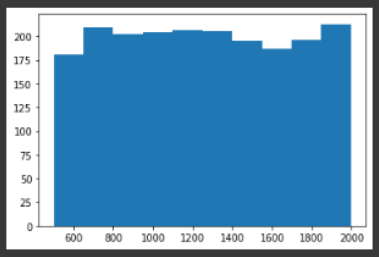
```
[23] 1 plt.hist(df['n_cores'])
2 plt.show()
```



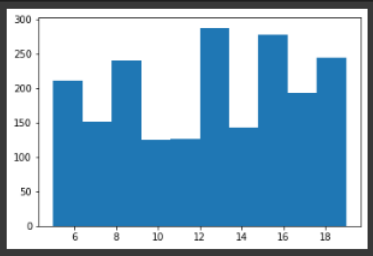
```
[25] 1 plt.hist(df['px_height'])
2 plt.show()
```



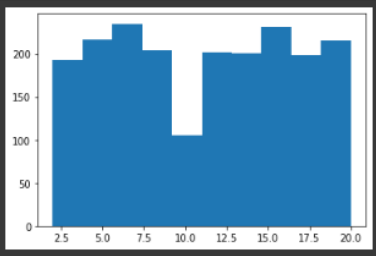
```
[26] 1 plt.hist(df['px_width'])
2 plt.show()
```



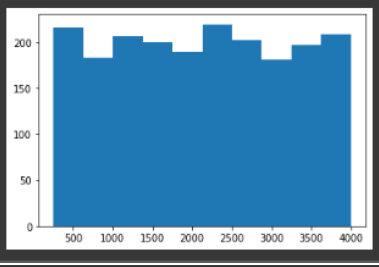
```
[28] 1 plt.hist(df['sc_h'])
2 plt.show()
```



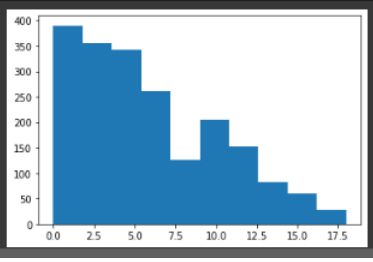
```
[30] 1 plt.hist(df['talk_time'])
2 plt.show()
```



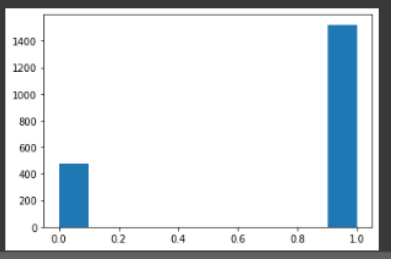
```
[27] 1 plt.hist(df['ram'])
2 plt.show()
```



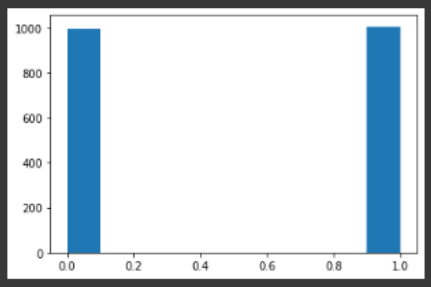
```
[29] 1 plt.hist(df['sc_w'])
2 plt.show()
```



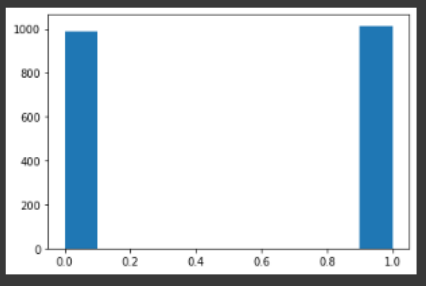
```
[31] 1 plt.hist(df['three_g'])
2 plt.show()
```



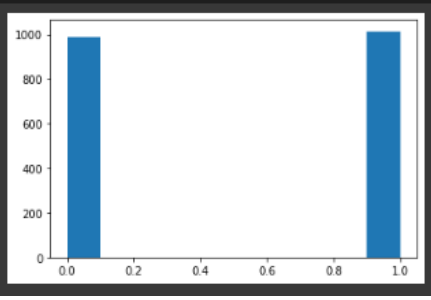
```
[32] 1 plt.hist(df['touch_screen'])
2 plt.show()
```



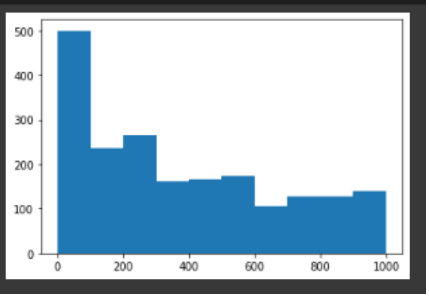
```
[33] 1 plt.hist(df['wifi'])
2 plt.show()
```




```
[33] 1 plt.hist(df['wifi'])
2 plt.show()
```

```
[34] 1 plt.hist(df['price_range'])
2 plt.show()
```



Una vez analizado los datos empece el entrenamiento de los modelos de regresion, primero cree los valores para el entrenamiento dividiendo el sataframe en X con todas las etiquetas de entrada para el modelo que serian las caracteristicas del celular y Y la etiqueta de salida que seria el precio de salida. Una vez teniendo los valores X Y dividi las columnas numericas y las columnas categoricas para pasarlo por la funcion make_column_transformer, luego cree los valores de entrenamiento y testeo para pasarselos a los modelos requeridos. Aquí fotos del entrenamiento:

Entrenamiento del modelo

```
[35]  1 x = df.iloc[:, :-1]
      2 y = df.iloc[:, -1]
```

```
[36]  1 numerical_columns = x.columns[(x.dtypes == "float64") | (x.dtypes == "int64")]
      2 categorical_columns = x.columns[(x.dtypes == "object")]
```

```
[37]  1 col_transf = make_column_transformer(
      2     (OneHotEncoder(handle_unknown='ignore'), categorical_columns),
      3     (StandardScaler(), numerical_columns),
      4     remainder="passthrough")
```

```
[38]  1 X_train, X_test, y_train, y_test = train_test_split(x, y, random_state=2, test_size=0.2)
```

```
[39]  1 models = []
      2 models.append(("LR", LinearRegression()))
      3 models.append(("RR", Ridge()))
      4 models.append(("BR", BayesianRidge()))
      5 models.append(("LASSO", Lasso()))
      6 models.append(("KNN", KNeighborsRegressor()))
      7 models.append(("RF", RandomForestRegressor()))
      8 models.append(("SVR", SVR()))
      9 models.append(("NNMLP", MLPRegressor()))
```

```
[40]  1 pipelines = {name: make_pipeline(col_transf, model) for name, model in models}
```

+ Code + Text


```
[40]  1 pipelines = {name: make_pipeline(col_transf, model) for name, model in models}
```


```
      1 results = []
      2 scoring = 'r2'
      3 kfolds = 5
      4
      5 for name, pipeline in pipelines.items():
      6   cv_results = cross_val_score(pipeline, X_train, y_train, cv=kfolds, scoring=scoring)
      7   results.append({name: cv_results})
```

```
Stochastic Optimizer: Maximum iterations (200) reached and the optimization hasn't converged yet.
Stochastic Optimizer: Maximum iterations (200) reached and the optimization hasn't converged yet.
Stochastic Optimizer: Maximum iterations (200) reached and the optimization hasn't converged yet.
Stochastic Optimizer: Maximum iterations (200) reached and the optimization hasn't converged yet.
Stochastic Optimizer: Maximum iterations (200) reached and the optimization hasn't converged yet.
```

```
[42]  1 results
```

```
[{'LR': array([0.81293774, 0.81462397, 0.81412339, 0.81801702, 0.82682383])},
 {'RR': array([0.81296164, 0.81464875, 0.81416408, 0.81794783, 0.82682736])},
 {'BR': array([0.81303408, 0.8147228 , 0.81429489, 0.8176988 , 0.82683142])},
 {'LASSO': array([0.81310064, 0.8153513 , 0.81764911, 0.81926212, 0.82784204])},
 {'KNN': array([0.49614521, 0.54294129, 0.54601597, 0.5518055 , 0.51683191])},
 {'RF': array([0.85410933, 0.84227362, 0.84676293, 0.85895864, 0.84665268])},
 {'SVR': array([ 0.02223125, -0.022529  ,  0.04955749,  0.03128373,  0.04652548])},
 {'NNMLP': array([0.66257225, 0.68568835, 0.66515614, 0.61789922, 0.68766671])}]
```

```
[43]  1 best_model = sorted(results, key=lambda x: list(x.values())[0].mean(), reverse=True)[0]
      2 best_pipeline = pipelines[list(best_model.keys())[0]]
```

```
[44]  1 best_model
```

```
{'RF': array([0.85410933, 0.84227362, 0.84676293, 0.85895864, 0.84665268])}
```

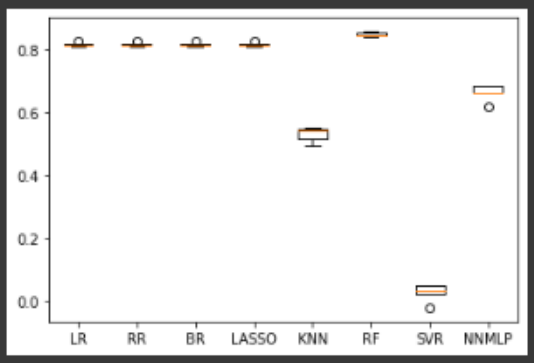
```
{'KNN': array([0.49614521, 0.54294129, 0.54601597, 0.5518055 , 0.51683191])},
{'RF': array([0.85410933, 0.84227362, 0.84676293, 0.85895864, 0.84665268])},
{'SVR': array([ 0.02223125, -0.022529  ,  0.04955749,  0.03128373,  0.04652548])},
{'NNMLP': array([0.66257225, 0.68568835, 0.66515614, 0.61789922, 0.68766671])}]
```

```
[43] 1 best_model = sorted(results, key=lambda x: list(x.values())[0].mean(), reverse=True)[0]
     2 best_pipeline = pipelines[list(best_model.keys())[0]]
```

```
[44] 1 best_model
```

```
{'RF': array([0.85410933, 0.84227362, 0.84676293, 0.85895864, 0.84665268])}
```

```
1 fig = plt.figure()
2 ax = fig.add_subplot(111)
3 plt.boxplot([list(dict_.values())[0] for dict_ in results])
4 ax.set_xticklabels([list(dict_.keys())[0] for dict_ in results])
5 plt.show()
```




```
1 best_pipeline.fit(X_train, y_train)
2 best_pipeline.predict(X_test)
```

```
array([ 57.22, 502.16, 140.48, 510.35, 814.66, 220.13, 843.33, 799.93,
       284.14, 281.16, 227.37,  49.69, 426.13, 793.47, 116.67, 803.84,
       445.22, 197.78, 196.73, 811.14,  71.68, 664.67,  50.08, 215.67,
       775.61, 391.02, 323.31,  62.3 ,  54.93, 327.67,  50.5 , 710.72,
        66.63, 120.05, 235.47,  55.88, 833.1 ,  68.37, 607.64, 411.81,
       168.31, 213.89, 750.92, 731.58, 564.52,  54.65, 648.34, 447.63,
       197.7 , 608.22, 189.65, 780.66, 814.17, 442.04, 736.94,  54.49,
       216.15,  75.98, 190.62, 260.91, 171.39, 480.56, 205.99, 584.42,
        66.18, 428.51,  75.16, 189.4 ,  50.64, 715.24, 213.94, 115.74,
        81.86, 434.79,  47.65, 460.23, 505.95, 776.3 , 484.49, 515.48,
       678.07, 356.42, 475.07,  83.75,  58.08, 838.37, 633.47, 404.3 ,
        61.91, 179.97, 189.71, 444.1 ,  66.72,  59.45, 199.52, 533.78,
        58.77, 819.  , 499.95,  64.49, 198.96, 818.21, 201.42, 212.81,
       639.  , 836.95, 782.93, 714.73, 460.87, 249.02, 777.2 , 212.13,
       188.12, 427.13, 815.83,  46.57, 101.33, 349.4 , 214.41, 438.42,
        54.57, 754.1 , 793.39, 115.24, 618.84,  55.22, 763.48, 268.85,
        60.91,  63.98, 202.84, 781.73, 464.85, 204.61,  61.12, 852.44,
       817.12,  67.61, 364.77,  61.88, 275.01, 799.36, 806.12,  37.99,
       764.53, 541.15, 208.61, 733.04,  54.99, 248.51, 434.78, 657.58,
       189.07, 721.92,  66.24, 783.91,  66.25, 530.66, 181.53, 200.13,
       207.94,  70.21, 792.11,  87.92, 144.15, 816.39, 291.75, 853.63,
       174.6 ,  63.29, 527.04, 134.53,  54.33, 797.01,  56.72, 237.98,
       137.54, 221.42, 743.92, 753.88, 430.6 , 360.84, 197.93, 207.91,
       257.8 ,  48.97,  59.57, 200.91, 727.89, 352.83, 197.12,  42.43,
       202.19, 213.91, 727.7 , 651.81,  48.24, 225.16, 811.46, 319.6 ,
        66.88, 604.88, 737.34,  67.34, 793.41, 576.65, 284.78, 587.84,
       458.77, 778.4 , 776.23, 607.18, 177.41,  85.4 , 608.73, 264.5 ,
        60.19, 254.37, 806.86,  50.55,  68.13, 805.15,  53.37, 190.15,
       266.38,  59.52, 161.59, 800.41, 352.58, 542.27, 824.15, 761.8 ,
       796.17,  56.33, 758.24,  45.44, 206.41, 444.15, 105.73, 771.3 ,
       476.08, 532.12, 377.82,  56.38, 564.52, 774.07,  48.83, 798.1 ,
       165.24, 227.6 , 196.86,  78.18, 290.6 , 806.72, 462.17, 351.57,
        88.13, 825.16, 826.37,  67.37,  73.75,  41.81, 862.15, 830.35,
```

```
       165.24, 227.6 , 196.86,  78.18, 290.6 , 806.72, 462.17, 351.57,
        88.13, 825.16, 826.37,  67.37,  73.75,  41.81, 862.15, 830.35,
        60.31, 854.55, 186.99,  61.47, 217.07,  80.77, 423.51,  64.19,
       507.38,  58.2 ,  58.37,  61.39, 405.73,  54.52, 528.89,  57.  ,
       808.12, 881.74,  50.92, 617.61, 471.23, 243.2 , 159.22, 256.69,
       295.84, 802.84,  59.54,  69.52, 736.88, 147.64, 804.26, 235.98,
       539.28,  65.45, 355.64,  64.84, 770.8 , 164.15, 518.97, 802.84,
       617.91, 725.16, 403.07, 796.85,  68.01,  71.56, 447.59, 412.78,
       152.05, 112.84, 164.28, 721.17, 402.17, 420.49,  53.74, 102.19,
       479.46,  94.14, 642.45, 527.78, 435.1 , 570.84, 696.61, 742.27,
       193.28, 845.35, 245.16, 773.33, 470.95, 294.84, 811.49, 791.44,
        54.8 , 224.45, 441.26, 436.45, 524.42, 443.64, 866.16,  46.17,
        57.79, 207.65,  47.43, 238.03, 454.63, 774.05,  48.73, 217.34,
       705.4 , 212.19, 597.31, 642.28, 464.24, 153.49,  70.62, 256.39,
       460.39, 826.34, 724.33,  70.59, 215.84, 432.21,  56.32, 210.59,
       746.96, 519.48, 237.07, 762.8 , 211.11,  62.41, 234.41, 770.95,
        69.96, 217.84, 300.75, 216.74, 468.65, 151.44,  80.26, 144.74,
       449.81, 405.45, 164.89,  54.68, 763.8 , 465.7 , 138.38, 187.96,
        55.36,  52.78, 451.42,  59.06, 840.34, 187.75, 786.51, 752.71])
```

```
[ ] 1 y_true = y_test
    2 y_pred = best_pipeline.predict(X_test)
```

```
1 print(r2_score(y_true, y_pred))
```

```
0.8616912762594715
```

Despliegue del modelo y api externa

```
[ ] 1 filename = 'mobile_prices.pkl'
    2 pickle.dump(best_pipeline, open(filename, 'wb'))
```

El mejor modelo fue el de random forest use la metrica de r2 score para seleccionar el meor modelo, luego con la librería de pickle exporte el modelo para poder usarlo en una aplicación externa a la que se le pasaran los valores de entrada y dependiendo de las caracteristicas dara el precio del celular, aquí algunas fotos de la api.

```
[ ]  1 filename = 'mobile_prices.pkl'
     2 pickle.dump(best_pipeline, open(filename, 'wb'))
```

```
[ ]  1 import pandas as pd
     2 import sklearn
     3 import pickle
     4
     5 if __name__ == '__main__':
     6   mp_mod = pickle.load(open('mobile_prices.pkl', mode = 'rb'))
     7
     8   print("Bienvenid@! Digita los datos del celular.")
     9
    10   battery_power = input("\nBateria total que el celular puede almacenar en miliamperios(mAH): ")
    11   talk_time = input("Tiempo que dura la bateria sin cargar: ")
    12   int_memory = input("Gigabytes de la memoria interna: ")
    13   clock_speed = input("Velocidad del microprocesador: ")
    14   n_cores = input("Numero de nucleos del procesador: ")
    15   ram = input("Ram del telefono en megabytes: ")
    16   dual_sim = input("Tiene dual sim (1: si, 0: no): ")
    17   blue = input("Tiene bluetooth (1: si, 0: no): ")
    18   wifi = input("Puede usar wifi (1: si, 0: no): ")
    19   three_g = input("Puede utilizar 3G (1: si, 0: no): ")
    20   four_g = input("Puede utilizar 4G (1: si, 0: no): ")
    21   touch_screen = input("El celular es touch (1: si, 0: no):")
    22   pc = input("Megapixeles de la camara principal: ")
    23   fc = input("Megapixeles de la camara frontal: ")
    24   px_height = input("Resolucion de la altura en pixeles: ")
    25   px_width = input("Resolucion del ancho en pixeles: ")
    26   sc_h = input("Altura del telefono en cm: ")
    27   sc_w = input("Anchura del telefono en cm: ")
    28   m_dep = input("Profundidad del celular en cm: ")
    29   mobile_wt = input("Peso del celular: ")
```

```
    12   int_memory = input("Gigabytes de la memoria interna: ")
    13   clock_speed = input("Velocidad del microprocesador: ")
    14   n_cores = input("Numero de nucleos del procesador: ")
    15   ram = input("Ram del telefono en megabytes: ")
    16   dual_sim = input("Tiene dual sim (1: si, 0: no): ")
    17   blue = input("Tiene bluetooth (1: si, 0: no): ")
    18   wifi = input("Puede usar wifi (1: si, 0: no): ")
    19   three_g = input("Puede utilizar 3G (1: si, 0: no): ")
    20   four_g = input("Puede utilizar 4G (1: si, 0: no): ")
    21   touch_screen = input("El celular es touch (1: si, 0: no):")
    22   pc = input("Megapixeles de la camara principal: ")
    23   fc = input("Megapixeles de la camara frontal: ")
    24   px_height = input("Resolucion de la altura en pixeles: ")
    25   px_width = input("Resolucion del ancho en pixeles: ")
    26   sc_h = input("Altura del telefono en cm: ")
    27   sc_w = input("Anchura del telefono en cm: ")
    28   m_dep = input("Profundidad del celular en cm: ")
    29   mobile_wt = input("Peso del celular: ")
    30
    31   celular = pd.DataFrame([battery_power, blue, clock_speed, dual_sim, fc, four_g, int_memory, m_dep, mobile_wt, n_cores, pc, px_height, px_width, ram, sc_h, sc_w, talk_time, three_g, touch_screen
    32   celular = celular.transpose().rename(columns={0: 'battery_power', 1: 'blue', 2: 'clock_speed', 3: 'dual_sim', 4: 'fc', 5: 'four_g', 6: 'int_memory', 7: 'm_dep', 8: 'mobile_wt', 9: 'n_cores', 10
    33
    34   prediccion = mp_mod.predict(celular)
    35
    36   print(f"\nEl precio del elefono es: {prediccion[0]}")

Bienvenid@! Digita los datos del celular.

Bateria total que el celular puede almacenar en miliamperios(mAH): 2000
Tiempo que dura la bateria sin cargar: 20
Gigabytes de la memoria interna: 6
Velocidad del microprocesador: 0.9
Numero de nucleos del procesador: 6
Ram del telefono en megabytes: 6000
Tiene dual sim (1: si, 0: no): 1
Tiene bluetooth (1: si, 0: no): 1
```

Por ultimo use la librería SHAP para ver que caracteristicas influian mas en el modelo y vistos los graficos se nota como la RAM es lo que influye mas en el modelo que cualquier otra caracteristicas.

```
Profundidad del celular en cm: 0.5
Peso del celular: 30

El precio del elefono es: 787.81
```

```
explainer = shap.Explainer(mp_mod['randomforestregressor'])
valores_shap = explainer(celular)
```

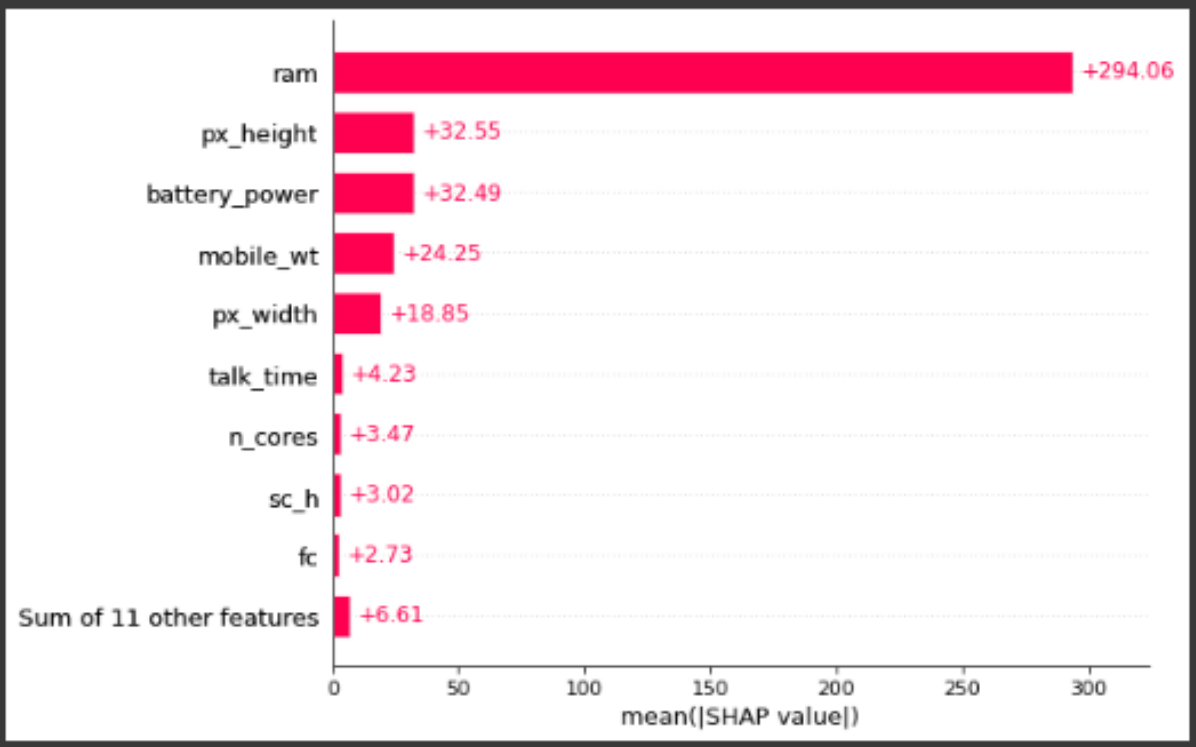
```
celulares = pd.DataFrame(best_pipeline[0].transform(X_test.sample(100)), columns=best_pipeline[0].get_feature_names_out())
```

```
explainer2 = shap.Explainer(best_pipeline[1].predict, celulares)
```

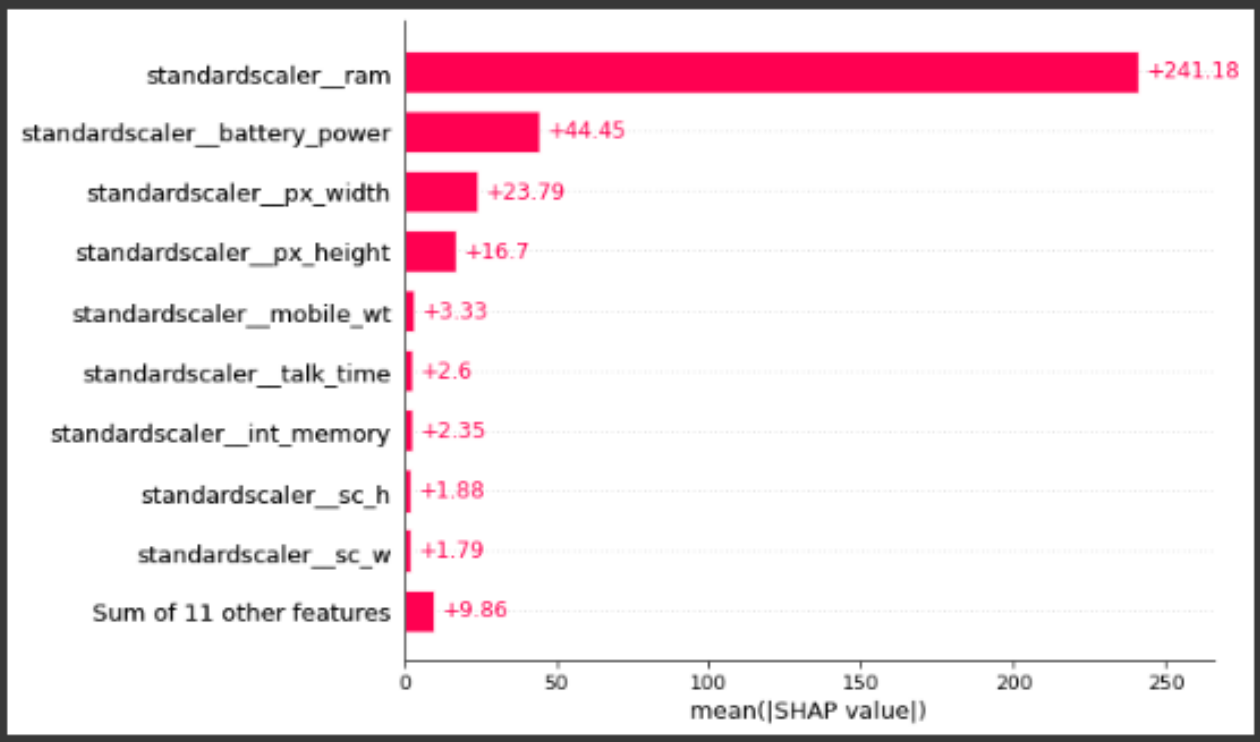
```
valores_shap2 = explainer2(celulares)
```

```
X has feature names, but RandomForestRegressor was fitted without feature names
X has feature names, but RandomForestRegressor was fitted without feature names
X has feature names, but RandomForestRegressor was fitted without feature names
X has feature names, but RandomForestRegressor was fitted without feature names
X has feature names, but RandomForestRegressor was fitted without feature names
X has feature names, but RandomForestRegressor was fitted without feature names
X has feature names, but RandomForestRegressor was fitted without feature names
X has feature names, but RandomForestRegressor was fitted without feature names
X has feature names, but RandomForestRegressor was fitted without feature names
X has feature names, but RandomForestRegressor was fitted without feature names
X has feature names, but RandomForestRegressor was fitted without feature names
X has feature names, but RandomForestRegressor was fitted without feature names
X has feature names, but RandomForestRegressor was fitted without feature names
X has feature names, but RandomForestRegressor was fitted without feature names
X has feature names, but RandomForestRegressor was fitted without feature names
X has feature names, but RandomForestRegressor was fitted without feature names
X has feature names, but RandomForestRegressor was fitted without feature names
```

```
shap.plots.bar(valores_shap)
```



```
1 shap.plots.bar(valores_shap2)
```




```
1 shap.plots.bar(valores_shap2[0])
```