Université Paris Nanterre

M2 IES-D3S : Approfondissement en Machine Learning
Année universitaire 2024-2025

Prédiction du prix de mobile

Professeur : Melanie ZETLAOUI

Étudiante : Madina GULIYEVA

Sommaire

1.Introduction

L'évolution rapide du marché des smartphones a entraîné une variation significative des prix, influencée par de nombreux facteurs techniques et de conception. Ce projet vise à prédire le prix des téléphones mobiles en utilisant des caractéristiques telles que la résolution, la marque, la taille, le poids, la qualité d'imagerie, la RAM, la capacité de la batterie et la puissance du processeur.

L'objectif principal de ce projet est d'élaborer des modèles prédictifs capables d'estimer le prix d'un téléphone mobile en fonction de ses caractéristiques techniques. En comparant les méthodes, nous avons également choisi le meilleur modèle.

	Price	Sale	weight	resoloution	ppi	cpu core	cpu freq	internal mem	ram	RearCam	Front_Cam	battery	thickness
0	2357	10	135.0	5.2	424	8	1.35	16.0	3.000	13.00	8.0	2610	7.4
1	1749	10	125.0	4.0	233	2	1.30	4.0	1.000	3.15	0.0	1700	9.9
2	1916	10	110.0	4.7	312	4	1.20	8.0	1.500	13.00	5.0	2000	7.6
3	1315	11	118.5	4.0	233	2	1.30	4.0	0.512	3.15	0.0	1400	11.0
4	1749	11	125.0	4.0	233	2	1.30	4.0	1.000	3.15	0.0	1700	9.9
5	2137	12	150.0	5.5	401	4	2.30	16.0	2.000	16.00	8.0	2500	9.5
6	1238	13	134.1	4.0	233	2	1.20	8.0	1.000	2.00	0.0	1560	11.7
7	2137	13	150.0	5.5	401	4	2.30	16.0	2.000	16.00	8.0	2500	9.5
8	1315	14	118.5	4.0	233	2	1.30	4.0	0.512	3.15	0.0	1400	11.0
9	2580	15	145.0	5.1	432	4	2.50	16.0	2.000	16.00	2.0	2800	8.1

2. Présentation et analyse des données

Pour commencer, nous avons effectué un nettoyage initial du dataset en supprimant la colonne Product_id. Cette colonne était utilisée comme identifiant unique pour chaque produit et n'apportait pas d'information pertinente pour l'analyse des prix.

Egalement on a des analyses descriptives

	Price	weight	resoloution	ppi	cpu core	cpu freq	internal mem	ram	RearCam	Front_Cam	battery	thickness
count	161.000000	161.000000	161.000000	161.000000	161.000000	161.000000	161.000000	161.000000	161.000000	161.000000	161.000000	161.000000
mean	2215.596273	170.426087	5.209938	335.055901	4.857143	1.502832	24.501714	2.204994	10.378261	4.503106	2842.111801	8.921739
std	768.187171	92.888612	1.509953	134.826659	2.444016	0.599783	28.804773	1.609831	6.181585	4.342053	1366.990838	2.192564
min	614.000000	66.000000	1.400000	121.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	800.000000	5.100000
25%	1734.000000	134.100000	4.800000	233.000000	4.000000	1.200000	8.000000	1.000000	5.000000	0.000000	2040.000000	7.600000
50%	2258.000000	153.000000	5.150000	294.000000	4.000000	1.400000	16.000000	2.000000	12.000000	5.000000	2800.000000	8.400000
75%	2744.000000	170.000000	5.500000	428.000000	8.000000	1.875000	32.000000	3.000000	16.000000	8.000000	3240.000000	9.800000
max	4361.000000	753.000000	12.200000	806.000000	8.000000	2.700000	128.000000	6.000000	23.000000	20.000000	9500.000000	18.500000

Le prix des smartphones varie considérablement, avec une moyenne de 2 563 629 unités monétaires, bien que la devise ne soit pas précisée. L'écart-type élevé indique une grande disparité des prix, témoignant ainsi de la diversité des modèles disponibles sur le marché. En ce qui concerne la résolution d'écran, elle est généralement élevée, ce qui suggère que les smartphones de l'échantillon sont souvent de bonne qualité.

Les caractéristiques des processeurs, incluant le nombre de cœurs et la fréquence, présentent une certaine diversité, tout en affichant des valeurs courantes qui semblent prédominer. Par ailleurs, la mémoire interne et la RAM varient considérablement, reflétant les différentes gammes de produits. Enfin, la résolution des appareils photo avant et arrière est également variable, mais une tendance générale vers des résolutions élevées se dessine, soulignant l'importance croissante de la photographie mobile.

Figure 1: Boxplot

Pour identifier les outliers dans notre dataset, nous avons commencé par visualiser les données à l'aide d'un boxplot. Dans le boxplot, les points qui se trouvent en dehors des moustaches sont considérés comme des outliers. En examinant le boxplot pour la variable Sale, nous avons observé des points qui semblent éloignés du reste des données, ce qui indique la présence d'outliers.

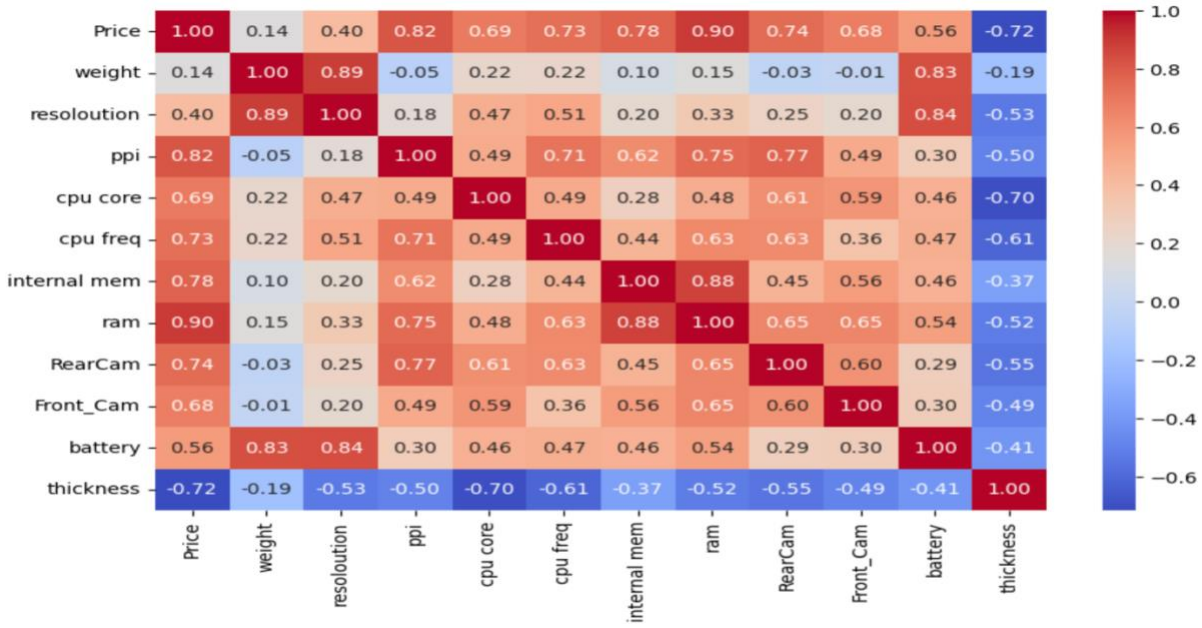
Pour confirmer la présence de ces outliers, nous avons calculé le Z-score pour la variable Sale. Le Z-score est une mesure statistique qui indique combien d'écarts-types une donnée est éloignée de la moyenne de l'échantillon.

Le Z-score permet d'identifier des valeurs qui se démarquent du reste des données. En général, un Z-score supérieur à 3 ou inférieur à -3 est considéré comme un indicateur fort de l'existence d'un outlier. Dans notre cas, le nombre d'outliers identifiés pour la variable Sale confirme nos observations initiales faites à partir du boxplot. C’est pourquoi on a supprime cette variable.

Figure 2:La relation entre le prix et differentes caracteristiques

Ces graphiques représentent la distribution du prix d'un produit (axe des ordonnées, noté "y") en fonction de différentes caractéristiques techniques de ce produit (axe des abscisses, noté "x"). Egalement ils montrent clairement que la distribution du prix en fonction des différentes caractéristiques n'est pas parfaitement normale.

Matrice de correlation:



On observe que le prix est fortement corrélé positivement avec la RAM, la mémoire interne, le nombre de pixels par pouce (PPI), la fréquence du processeur, la capacité de la batterie arrière et avant. Cela signifie que lorsque ces caractéristiques augmentent, le prix a tendance à augmenter également. Le prix est fortement corrélé négativement avec l'épaisseur du téléphone. Les téléphones plus fins ont tendance à être plus chers.

3. Méthodologie

3.1 Régression linéaire

Nous avons d'abord séparé notre dataset en variables indépendantes et dépendantes. Ensuite, nous avons divisé nos données en ensembles d'entraînement et de test.

La régression linéaire est une méthode statistique utilisée pour modéliser la relation entre une variable dépendante (cible) et une ou plusieurs variables indépendantes (prédicteurs). Dans notre cas, nous cherchons à prédire le prix des téléphones mobiles (variable dépendante) en fonction de leurs caractéristiques techniques (variables indépendantes).

Nous avons ensuite ajusté un modèle de régression linéaire en utilisant la méthode des moindres carrés.

	Coef.	Std.Err.	t	P>|t|	[0.025	0.975]
const	1803.438269	277.788423	6.492129	3.298044e-09	1252.313950	2354.562587
weight	-0.449343	0.935313	-0.480420	6.319781e-01	-2.304976	1.406291
resoloution	-92.996629	57.613405	-1.614149	1.096463e-01	-207.299983	21.306726
ppi	1.187516	0.267854	4.433450	2.382059e-05	0.656102	1.718931
cpu core	49.701672	13.278175	3.743110	3.036597e-04	23.358151	76.045192
cpu freq	151.553071	61.447686	2.466376	1.534984e-02	29.642613	273.463529
internal mem	4.715090	1.393658	3.383247	1.023945e-03	1.950112	7.480069
ram	104.392735	34.558675	3.020739	3.201320e-03	35.829308	172.956161
RearCam	2.651623	5.635509	0.470521	6.390078e-01	-8.529067	13.832313
Front_Cam	3.774866	6.270302	0.602023	5.485213e-01	-8.665234	16.214966
battery	0.141791	0.037420	3.789159	2.584274e-04	0.067551	0.216032
thickness	-77.388187	15.735467	-4.918074	3.436264e-06	-108.606905	-46.169470

Les résultats montrent que certaines caractéristiques techniques influencent de manière significative le prix des smartphones. Par exemple, le PPI (Pixels Par Pouce) a un coefficient de 1.19, ce qui signifie qu'une augmentation d'un point de PPI est associée à une hausse du prix de 1.19 unités, avec un résultat hautement significatif. De même, le nombre de cœurs du processeur, avec un coefficient de 49.70, indique qu'un cœur supplémentaire dans le cpu augmente le prix de 49.70 unités . La fréquence du cpu, quant à elle, a un coefficient de 151.55, suggérant qu'une augmentation de 1 GHz dans la fréquence du processeur se traduit par une hausse de 151.55 unités du prix .

La RAM, avec un coefficient de 104.39, montre qu'une augmentation de 1 Go de RAM augmente le prix de 104.39 unités . Concernant la batterie, chaque augmentation de 1 mAh de sa capacité est associée à une hausse de 0.14 unités du prix, avec un

coefficient de 0.14 . Enfin, l'épaisseur du téléphone, avec un coefficient de -77.39, indique qu'une réduction de l'épaisseur est liée à une augmentation significative du prix.

En revanche, certaines caractéristiques n'ont pas d'effet significatif sur le prix. Le poids du téléphone, avec un coefficient de -0.45, la résolution de l'écran (-92.99), ainsi que les spécifications des caméras avant (3.77) et arrière (2.65) ne semblent pas influencer le prix de manière notable. Cela suggère que les consommateurs accordent moins d'importance à ces aspects lorsqu'ils évaluent la valeur d'un smartphone.

Model:	OLS	Adj. R-squared:	0.946
Dependent Variable:	Price	AIC:	1498.0631
Date:	2024-11-05 14:47	BIC:	1530.6851
No. Observations:	112	Log-Likelihood:	-737.03
Df Model:	11	F-statistic:	178.3
Df Residuals:	100	Prob (F-statistic):	1.69e-60
R-squared:	0.951	Scale:	34089.

L'analyse statistique, confirmée par un F-statistic élevé démontre que le modèle est bien ajusté et que les variables indépendantes sont significativement associées au prix des mobiles.

On a calculé les MSE, RMSE et R^2 pour l'entraînement et le test afin de vérifier s'il y a du surapprentissage.

	Metrique	Train	Test
0	R2	0.951486	0.952192
1	MSE	30436.588963	23541.299660
2	RMSE	174.460852	153.431743

Globalement, les résultats montrent que le modèle est bien équilibré, avec des performances similaires sur les ensembles d'entraînement et de test. Il n'y a pas d'indications de sur-apprentissage ou de sous-apprentissage, ce qui est un signe positif pour la fiabilité des prédictions sur de nouvelles données.

Figure 3: Régression linéaire

Le fait que la plupart des points soient proches de la ligne diagonale suggère que le modèle a fait de bonnes prédictions. Cela signifie que le modèle est capable de capturer les tendances sous-jacentes dans les données.

3.1.1 Pénalisation sur un modèle de régression : Ridge

La régression Ridge est une technique de régression linéaire régularisée, visant à améliorer la stabilité et la généralisation du modèle, notamment dans les cas où les variables explicatives sont fortement corrélées. Elle atténue cet effet en ajoutant un terme de pénalité qui contraint la taille des coefficients, en favorisant ceux qui sont plus petits et plus robustes. Cette approche aide à réduire la complexité du modèle tout en préservant sa capacité prédictive.

Pour ajuster la régression Ridge, les données ont d'abord été standardisées afin de garantir une échelle uniforme entre les variables. Ensuite, une validation croisée (RidgeCV) a été utilisée pour sélectionner la meilleure valeur du paramètre de régularisation alpha, soit 5,7, en minimisant l'erreur quadratique moyenne. Ce processus a exploré un large éventail de valeurs pour alpha afin de déterminer le niveau optimal de régularisation, permettant au modèle de généraliser efficacement sans risque de sur-apprentissage.

3.1.2 Pénalisation sur un modèle de régression : Lasso

La régression Lasso est une technique de régression linéaire régularisée, conçue pour améliorer la précision et l'interprétabilité du modèle, particulièrement dans les situations où un grand nombre de variables explicatives est présent. Contrairement à une régression classique, où la présence de nombreuses variables peut mener à un sur-apprentissage et à une complexité inutile, la régression Lasso applique une pénalité qui favorise les coefficients plus petits et, surtout, réduit certains à zéro. Cela permet de sélectionner automatiquement les variables les plus significatives tout en éliminant celles qui apportent peu de valeur prédictive. En intégrant cette pénalité de type L1, la régression Lasso aide à simplifier le modèle tout en maintenant sa capacité de généralisation.

Comme pour la régression Ridge, les données ont d'abord été standardisées et la validation croisée a également été utilisée pour trouver la meilleure valeur de alpha, soit 1,23, en minimisant l'erreur quadratique moyenne.

3.2 Réseaux de neurones

L'utilisation des réseaux de neurones dans le domaine de la régression est devenue une approche prépondérante pour modéliser des relations complexes entre les variables. Dans cette étude, nous avons construit un modèle de réseau de neurones pour prédire des valeurs cibles à partir de caractéristiques d'entrée.

Le modèle est constitué de plusieurs couches :

Couche d'entrée : 128 neurones avec activation ReLU.

Couches cachées : Deux couches avec 32 et 16 neurones, respectivement, utilisant également ReLU pour capturer des non-linéarités.

Couche de sortie : Une couche avec un seul neurone pour la prédiction de valeur continue.

Avant de procéder à l'entraînement, le modèle a été compilé avec l'optimiseur RMSprop, qui est bien adapté pour les problèmes de régression en raison de sa capacité à s'adapter au taux d'apprentissage pendant l'entraînement. La fonction de perte utilisée est l'erreur quadratique moyenne (MSE), qui mesure la moyenne des carrés des erreurs entre les valeurs prédites et les valeurs réelles.

L'entraînement a été effectué sur l'ensemble de données préalablement normalisées. Nous avons choisi un batch_size de 128 et avons entraîné le modèle sur 500 époques, permettant ainsi au réseau de converger et d'apprendre efficacement les relations entre les variables.

Figure 4: Visualisation des pertes

Ce graphique illustre l'évolution de la perte au cours de l'entraînement. Sur l'axe des abscisses, on trouve le nombre d'itérations d'entraînement, également appelé époques. L'axe des ordonnées indique la valeur de la perte, qui est une mesure de l'erreur du modèle ; plus cette valeur est faible, meilleure est la performance du modèle.

Les courbes présentent deux aspects distincts : la courbe bleue, représentant la perte sur les données d'entraînement, montre comment le modèle s'adapte à ces données à chaque itération. En revanche, la courbe orange, qui représente la perte sur un ensemble de données distinct utilisé pour la validation, permet d'évaluer la capacité du modèle à généraliser à de nouvelles données.

L'analyse du graphique suggère que le modèle converge vers une solution optimale, avec une diminution des pertes au fil des époques. Les courbes d'entraînement et de validation suivent des tendances similaires, ce qui indique que le modèle est capable de bien généraliser aux nouvelles données.

Enfin, une fois les modèles entraînés, des prédictions ont été réalisées sur l'ensemble de test. Les performances du modèle ont été évaluées à l'aide de trois métriques essentielles. :

Erreur Quadratique Moyenne (MSE) :

- Le MSE a été calculé pour quantifier l'erreur entre les prédictions du modèle et les valeurs réelles. Cette métrique fournit une mesure globale de la précision du modèle, les valeurs plus faibles indiquant une meilleure performance.

Erreur Quadratique Moyenne Racine (RMSE) :

- Le RMSE, qui est la racine carrée du MSE, a également été calculé. Cette métrique est particulièrement utile car elle est dans la même unité que la variable cible, facilitant ainsi l'interprétation des résultats.

Coefficient de Détermination (R^2) :

- Le R^2 est une autre mesure de la performance du modèle, indiquant la proportion de variance expliquée par le modèle. Un R^2 proche de 1 indique une bonne adéquation du modèle aux données.

3.2.1 Réseaux de neurones avec régularisation et dropout

Pour créer un modèle robuste et performant, nous avons développé une architecture de réseau de neurones intégrant des techniques de régularisation et de Dropout. Ces approches visent à atténuer le risque de sur-apprentissage, qui peut survenir lorsque le modèle devient trop complexe par rapport aux données d'entraînement.

En appliquant des pénalités L1 (Lasso) et L2 (Ridge) sur les couches cachées, nous avons encouragé le modèle à conserver uniquement les variables les plus pertinentes. Cela aide à réduire la variance du modèle tout en maintenant une performance prédictive satisfaisante, surtout lorsque les données contiennent de nombreuses caractéristiques.

La technique Dropout consiste à "éteindre" aléatoirement certaines neurones pendant l'entraînement, ce qui empêche le modèle de devenir trop dépendant de certaines caractéristiques. Cela contribue également à améliorer la généralisation du modèle sur des données nouvelles.

Nous avons testé différentes combinaisons de taux de Dropout et de niveaux de régularisation pour identifier l'architecture optimale. Cela nous a permis d'évaluer systématiquement l'impact de chaque paramètre sur la performance du modèle, mesurée par des métriques telles que l'erreur quadratique moyenne (MSE) et le coefficient de détermination R^2.

Nous avons ainsi analysé les performances d'un modèle de réseau de neurones pour la prédiction des prix de téléphones mobiles, en comparant un modèle de base sans régularisation et Dropout à plusieurs configurations intégrant ces techniques.

Le modèle de base a obtenu un MSE de 198,607, indiquant un risque potentiel de sur-apprentissage.

Performances avec Régularisation et Dropout :

	Dropout Rate	L1	L2	MSE	R^2
0	0.0	0.00	0.00	395567.663933	0.196683
1	0.0	0.00	0.01	176643.823869	0.641272
2	0.0	0.01	0.00	135020.593681	0.725801
3	0.0	0.01	0.01	144555.079803	0.706438
4	0.5	0.00	0.00	211985.027932	0.569502
5	0.5	0.00	0.01	231658.853184	0.529548
6	0.5	0.01	0.00	325234.084787	0.339516
7	0.5	0.01	0.01	305700.553252	0.379185

:

La configuration avec un Dropout de 0.0, L1 de 0.01 et L2 de 0.00 a obtenu le meilleur MSE de 135,020.59 et un R^2 de 0.7258, prouvant l'efficacité de la régularisation L1.

Un taux de Dropout de 0.5 a souvent conduit à des MSE plus élevés, suggérant que cette approche pourrait nuire à la performance dans notre cas.

Les résultats montrent que l'utilisation de L2 n'a pas surpassé la régularisation L1 seule.

4. Comparaison des performances des modèles

Nous avons évalué plusieurs modèles pour prédire les prix des téléphones mobiles, chacun ayant des performances distinctes mesurées par MSE, RMSE et R^2.

	Model	MSE	RMSE	R^2
0	Regression lineaire	23541.29966	153.431743	0.952192
1	Regression ridge	23240.29703	152.447686	0.952804
2	Regression Lasso	23298.51057	152.638496	0.952685
3	Reseaux de neurones	198607.67224	445.654207	0.596668

Ces modèles ont montré des performances très similaires, mais la régression Ridge a émergé comme le meilleur modèle avec le MSE le plus bas de 23,240.30, un RMSE de 152.45 et un R^2 de 0.9528. Cela indique qu'elle explique le mieux la variance des prix des téléphones tout en maintenant une erreur relativement faible.

À l'inverse, le réseau de neurones a affiché un MSE significativement plus élevé de 198,607.67, un RMSE de 445.65, et un R^2 de 0.5967. Ces résultats suggèrent une performance insatisfaisante par rapport aux autres modèles. Le réseau de neurones semble avoir souffert d'un sur-apprentissage ou d'une mauvaise optimisation des hyperparamètres, ce qui a affecté sa capacité à prédire les prix de manière précise.

Annexes

Figure 1:Boxplot

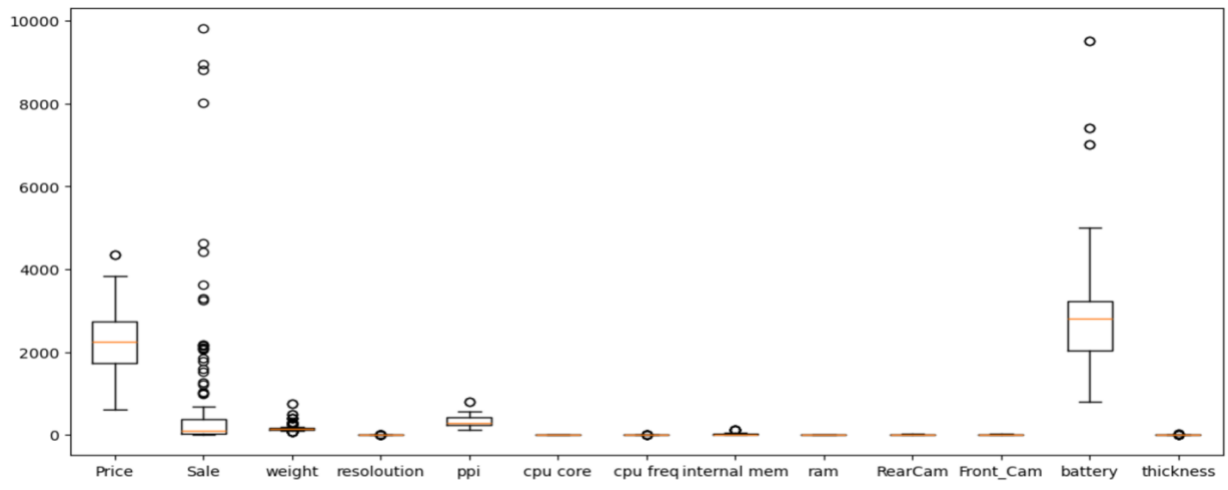


Figure 2:La relation entre le prix et les differentes caracteristiques

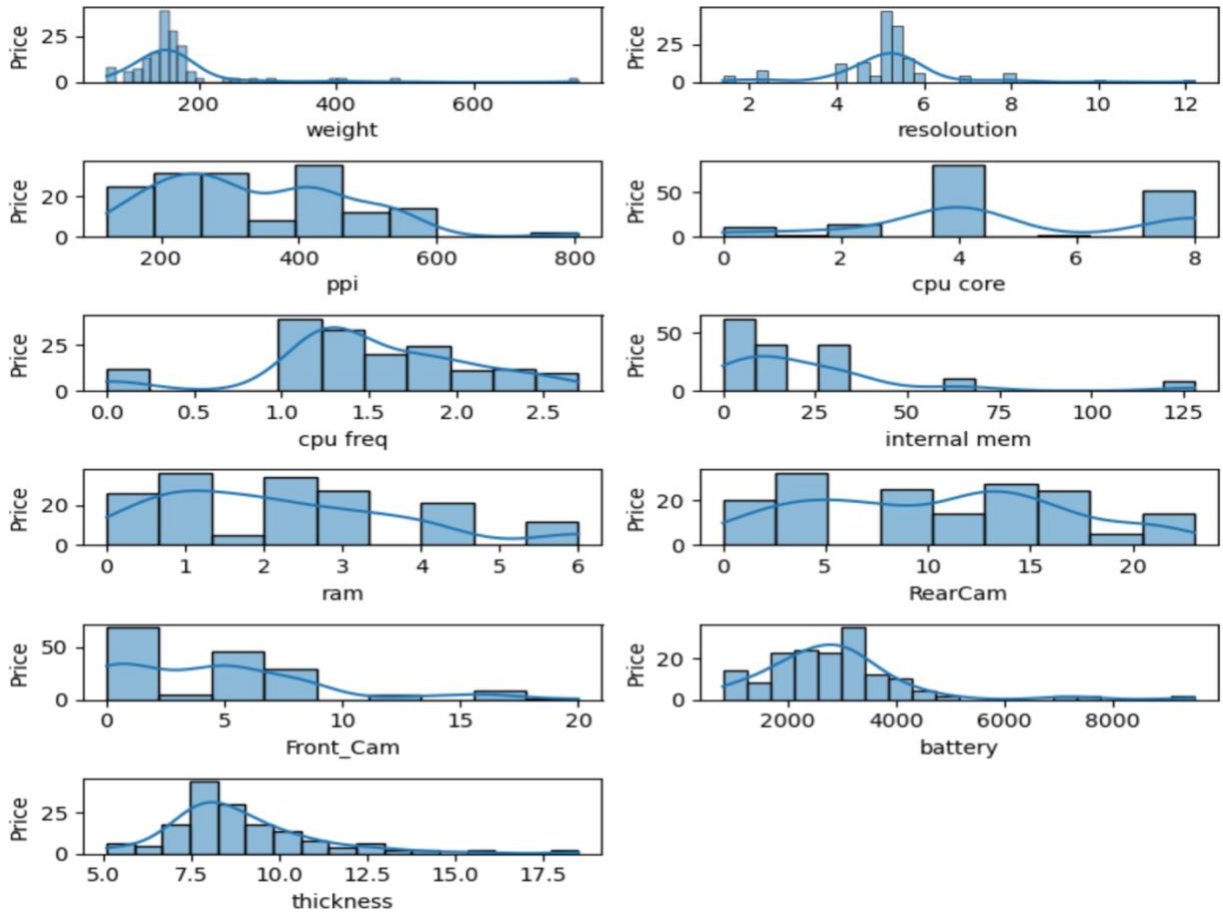


Figure 3:Regression lineaire

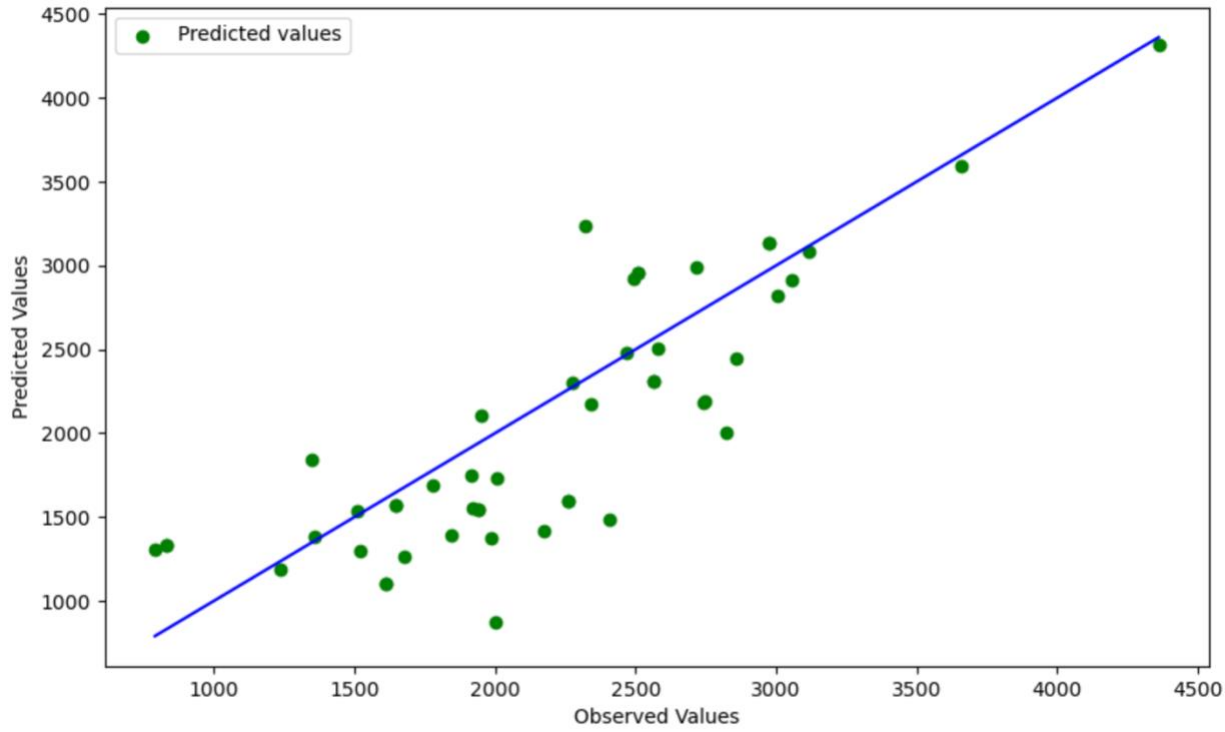
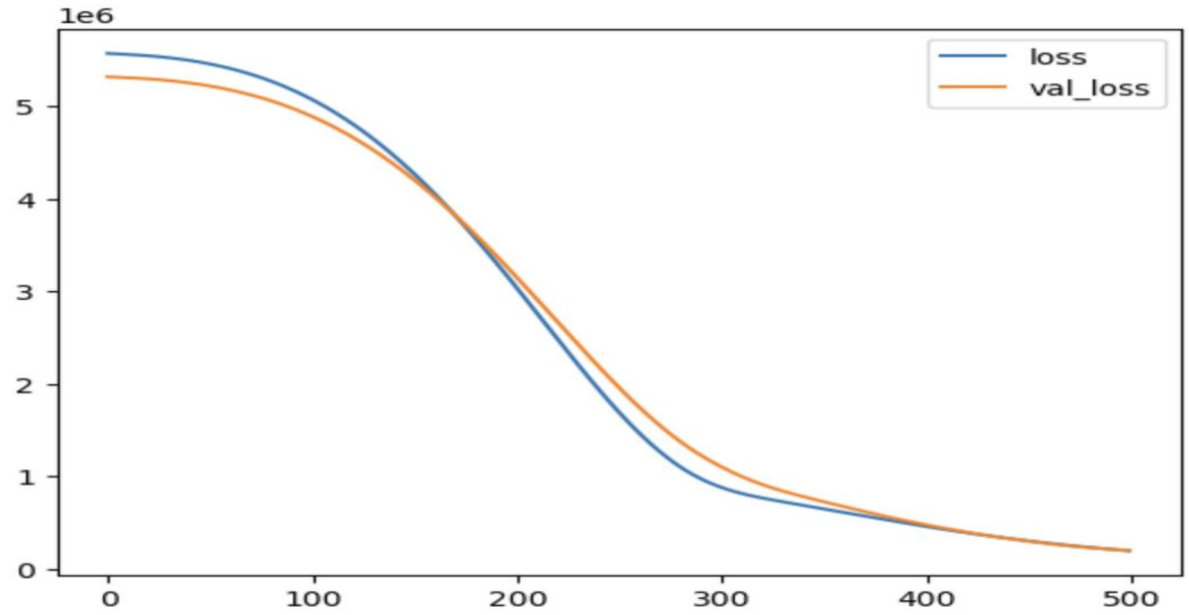


Figure 4: Visualisation des pertes



Les codes 1

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from scipy import stats
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error,r2_score
import statsmodels.api as sm
from sklearn.linear_model import Lasso, LassoCV, Ridge, RidgeCV
from sklearn.model_selection import cross_val_score, GridSearchCV
from sklearn.preprocessing import StandardScaler
from scikeras.wrappers import KerasRegressor  # Utilisez KerasRegressor pour les problèmes de régression
from sklearn.model_selection import RandomizedSearchCV
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
import tensorflow as tf
from tensorflow.keras.layers import Dropout
```

```
from tensorflow.keras.regularizers import l1, l2, l1_l2
import random
from keras import regularizers
df = pd.read_csv("/Users/madina/Desktop/ML /Cellphone.csv", sep =",")
# Afficher toutes les colonnes
pd.set_option('display.max_columns', None)
df.head(10)
df.describe()
df.drop(columns=['Product_id'], inplace = True)
df.info()
df.isna().sum()
#Figure 1:Boxplot
plt.figure(figsize=(13, 6))
plt.boxplot(df,labels=df.columns)
plt.show()
# Calculer le Z-score pour chaque valeur de 'Sale'
z_scores = stats.zscore(df['Sale'])
outliers_z = df[(z_scores > 3) | (z_scores < -3)]
print(f"Nombre d'outliers (Z-score) pour la variable 'Sale': {len(outliers_z)}")
print(outliers_z[['Sale']])
df.drop(columns=['Sale'], inplace = True)
#Figure 2:La relation entre le prix et les differentes caracteristiques
cols = ['weight', 'resoloution', 'ppi', 'cpu core', 'cpu freq', 'internal mem', 'ram', 'RearCam',
'Front_Cam', 'battery', 'thickness']
fig, axes = plt.subplots(figsize=(8, 7), nrows=6, ncols=2)
axes = axes.flatten()
for i in range(len(cols)):
    sns.histplot(df, x=cols[i], ax=axes[i], kde=True)
    axes[i].set_ylabel('Price')
    axes[i].set_xlabel(cols[i])
fig.delaxes(axes[-1])
plt.tight_layout()
plt.show()
#Matrice de correlation
plt.figure(figsize=(10,6))
corr = df.corr()
sns.heatmap(corr,cmap='coolwarm',annot=True,fmt='.2f')
plt.show()
x = df.drop('Price' , axis=1)
y = df['Price']
x_train , x_test , y_train , y_test = train_test_split(x , y , random_state=42 , test_size=0.3)
#Regression lineaire
lr_model =sm.OLS(y_train, sm.add_constant(x_train)).fit()
results_table = lr_model.summary2().tables[1]
```

```
results_table
lr_model.summary2().tables[0]
#obtention des predictions et des metriques d'evaluation
y_train_pred_lr = lr_model.predict(sm.add_constant(x_train))
y_test_pred_lr = lr_model.predict(sm.add_constant(x_test))
r2_train_lr = r2_score(y_train, y_train_pred_lr)
r2_test_lr = r2_score(y_test, y_test_pred_lr)
mse_train_lr = mean_squared_error(y_train, y_train_pred_lr)
mse_test_lr = mean_squared_error(y_test, y_test_pred_lr)
rmse_train_lr = np.sqrt(mse_train_lr)
rmse_test_lr = np.sqrt(mse_test_lr)
#construction du tableau du resultats
df_metrics = pd.DataFrame({
    'Metrique': ['R2', 'MSE', 'RMSE'],
    'Train': [r2_train_lr, mse_train_lr, rmse_train_lr],
    'Test': [r2_test_lr, mse_test_lr, rmse_test_lr]})
print(df_metrics)
#Figure 3: Regression lineaire
plt.figure(figsize=(10, 6))
plt.scatter(y_test, y_test_pred, color='green', label='Predicted values')
min_val = min(min(y_test), min(y_test_pred))
max_val = max(max(y_test), max(y_test_pred))
plt.plot([min_val, max_val], [min_val, max_val], color='blue')
plt.xlabel('Observed Values')
plt.ylabel('Predicted Values')
plt.legend()
plt.show()
scaler = StandardScaler()
x_train_scaled = scaler.fit_transform(x_train)
x_test_scaled = scaler.transform(x_test)
#Regression ridge
alpha_values = np.logspace(-6, 6, 200)
ridge_cv = RidgeCV(alphas=alpha_values, scoring='neg_mean_squared_error', cv=5)
ridge_cv.fit(x_train_scaled, y_train)
best_alpha_ridge = ridge_cv.alpha_
y_pred_ridge = ridge_cv.predict(x_test_scaled)
mse_ridge = mean_squared_error(y_test, y_pred_ridge)
rmse_ridge = np.sqrt(mse_ridge)
r2_ridge = r2_score(y_test, y_pred_ridge)
print(f"Meilleur alpha pour Ridge dans l'intervalle : {best_alpha_ridge}")
print(f"MSE__ridge: {mse_ridge}")
print(f"RMSE__ridge: {rmse_ridge}")
print(f"R²__ridge: {r2_ridge}")
#Regression lasso
```

```
alpha_values = np.logspace(-6, 6, 200)
lasso_cv = LassoCV(alphas=alpha_values, cv=5)
lasso_cv.fit(x_train_scaled, y_train)
best_alpha_lasso = lasso_cv.alpha_
y_pred_lasso = lasso_cv.predict(x_test_scaled)
mse_lasso = mean_squared_error(y_test, y_pred_lasso)
rmse_lasso = np.sqrt(mse_lasso)
r2_lasso = r2_score(y_test, y_pred_lasso)
print(f"Meilleur alpha pour Lasso: {best_alpha_lasso}")
print(f"MSE_lasso: {mse_lasso}")
print(f"RMSE_lasso: {rmse_lasso}")
print(f"R²_lasso: {r2_lasso}")
#Reseaux de neurones
model = Sequential()
model.add(Dense(128,activation='relu'))
model.add(Dense(32,activation='relu'))
model.add(Dense(16,activation='relu'))
model.add(Dense(1))
model.build(x_train.shape)
model.summary()
model.compile(optimizer='RMSprop', loss='mse')
model.fit(x=x_train_scaled,y=y_train.values, validation_data=(x_test_scaled,y_test.values),
batch_size=128,epochs=500)
#Figure 4: Visualisation des pertes
losses = pd.DataFrame(model.history.history)
losses.plot()
plt.show()
# Prédictions sur l'ensemble de test
y_pred_rn = model.predict(x_test_scaled)
mse_rn = mean_squared_error(y_test.values, y_pred_rn)
rmse_rn = np.sqrt(mse_rn)
r2_rn = r2_score(y_test.values, y_pred_rn)
print(f"MSE_rn: {mse}")
print(f"RMSE_rn: {rmse}")
print(f"R²_rn: {r2}")plt.show()
#Fonction pour créer un modèle avec régularisation et Dropout
def create_model(dropout_rate=0.0, l1_reg=0.0, l2_reg=0.0):
    model = Sequential()
    model.add(Dense(128, activation='relu', input_shape=(x_train_scaled.shape[1],),
                    kernel_regularizer=regularizers.l2(l2_reg)))
    model.add(Dropout(dropout_rate))
    model.add(Dense(32, activation='relu', kernel_regularizer=regularizers.l1(l1_reg)))
    model.add(Dropout(dropout_rate))
    model.add(Dense(16, activation='relu', kernel_regularizer=regularizers.l1_l2(l1_reg, l2_reg)))
```

```
    model.add(Dense(1))  # Couche de sortie
    model.compile(optimizer='RMSprop', loss='mse')
    return model


results = []

# Paramètres à tester
dropout_rates = [0.0, 0.5]
l1_values = [0.0, 0.01]
l2_values = [0.0, 0.01]


for dropout in dropout_rates:
    for l1 in l1_values:
        for l2 in l2_values:
            print(f'Testing Dropout: {dropout}, L1: {l1}, L2: {l2}')
            model = create_model(dropout_rate=dropout, l1_reg=l1, l2_reg=l2)
            history = model.fit(x_train_scaled, y_train.values,
                                validation_data=(x_test_scaled, y_test.values),
                                batch_size=128, epochs=500, verbose=0)

            # Évaluation du modèle
            y_pred = model.predict(x_test_scaled)
            mse = mean_squared_error(y_test.values, y_pred)
            r2 = r2_score(y_test.values, y_pred)
            results.append((dropout, l1, l2, mse, r2))


results_df = pd.DataFrame(results, columns=['Dropout Rate', 'L1', 'L2', 'MSE', 'R^2'])
print(results_df)

#Comparaison des resultats
results = pd.DataFrame({
    'Model': ['Regression lineaire', 'Regression ridge', 'Regression Lasso', 'Reseaux de
neurones'],
    'MSE': [mse_test_lr, mse_ridge, mse_lasso, mse_rn],
    'RMSE': [rmse_test_lr, rmse_ridge, rmse_lasso, rmse_rn],
    'R²': [r2_test_lr, r2_ridge, r2_lasso, r2_rn]
})

print(results)
```