Practica 1 Aprenentatge Computacional:

By: David Candela, Alex Casamitjana, Guillermo Raya
NIUs: 1563873, 1563873, 1568864 respectivament.

Apartat C:

- Preguntes:
  - Quin és el tipus de cada atribut?

Atribut	Tipus de dada	Descripció
Serial No.	Nombre enter (int)	Posició de la entrada en ordre.
GRE Score	Nombre enter (int)	Puntuació del estudiant a els Graduate Record Examinations, els valors van desde 260 fins a 340, representant el sistema de puntuació a India.
TOEFL Score	Nombre enter (int)	Puntuació del estudiant a els Test Of English as a Foreign Language, els valors poden anar desde 0 fins a 120.
University Rating	Nombre enter (int)	Puntuació de la universitat pot variar entre 1 i 5.
SOP (Statement of Purpose)	Nombre Decimal (float)	Valoració de la qualitat del Statement of Purpose, en valors de 0 a 5 en intervals de 0'5.
LOR (Letter of Recommendation)	Nombre Decimal (float)	Valoració de la qualitat del Letter of Recommendation, en valors de 0 a 5 en intervals de 0'5.
CGPA (Undergraduate GPA)	Nombre Decimal (float)	Nota mitja de l'estudiant en els seus estudis undergrad. Va de 0 a 10.
Research Experience	Booleà (bool)	Representa si l'estudiant ha tingut experiència de recerca anteriorment on no.
Chance of Admission	Nombre Decimal (float)	Probabilitat de ser admès a la universitat donats tots els altres factors.

  - Quins atributs tenen una distribució Guassiana?

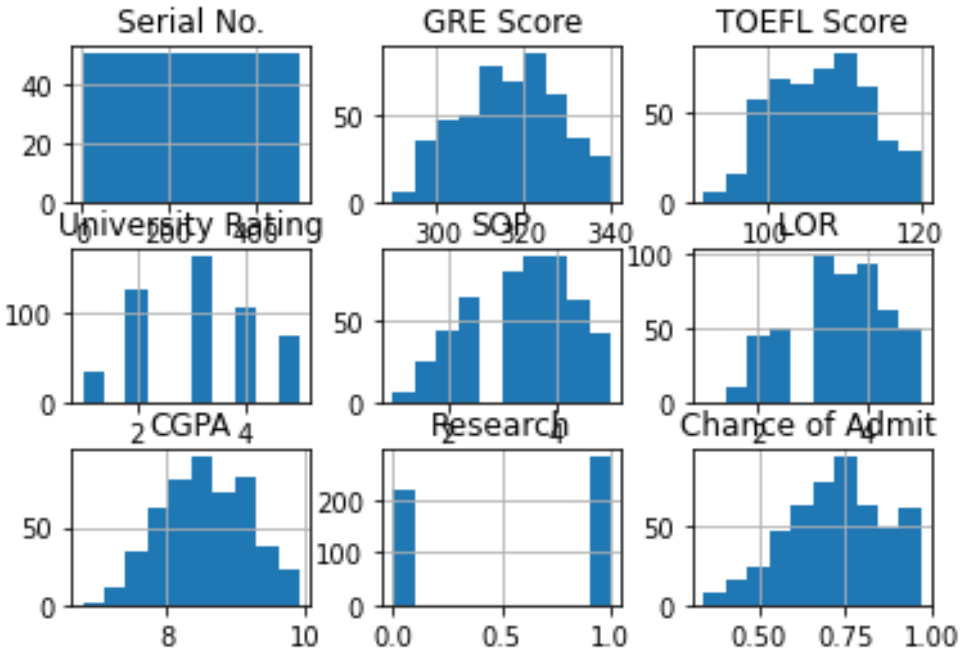


Figure 1: Histograma dels diferents atributs

Els atributs que tenen una distribució gaussiana basant-nos en els histogrames són: GRE Score, TOEFL Score, University Rating, SOP, LOR, CGPA i Chance of Admission

- Quin és l'atribut objectiu? Per què?
  - L'atribut objectiu es *Chance of admission,* ja que es la dada que normalment no tindriem i ens interessaria més, sobretot si fossim un estudiant que vol saber a quina universitat pot entrar.

- Plantejament:
  - Codi:
    - Hem fet servir simples comandes per veure les caracteristiques generals del dataset un cop aquest estaba ja carregat al notebook.

```
# Funcio per a llegir dades en format csv
def load_dataset(path):
    dataset = pd.read_csv(path, header=0, delimiter=',')
    return dataset
}
```

```
dataset.head()
dataset.describe()
display(dataset.hist())
```

Apartat B:

- Preguntes:
  - Quin són els atributs més importants per fer una bona predicció?

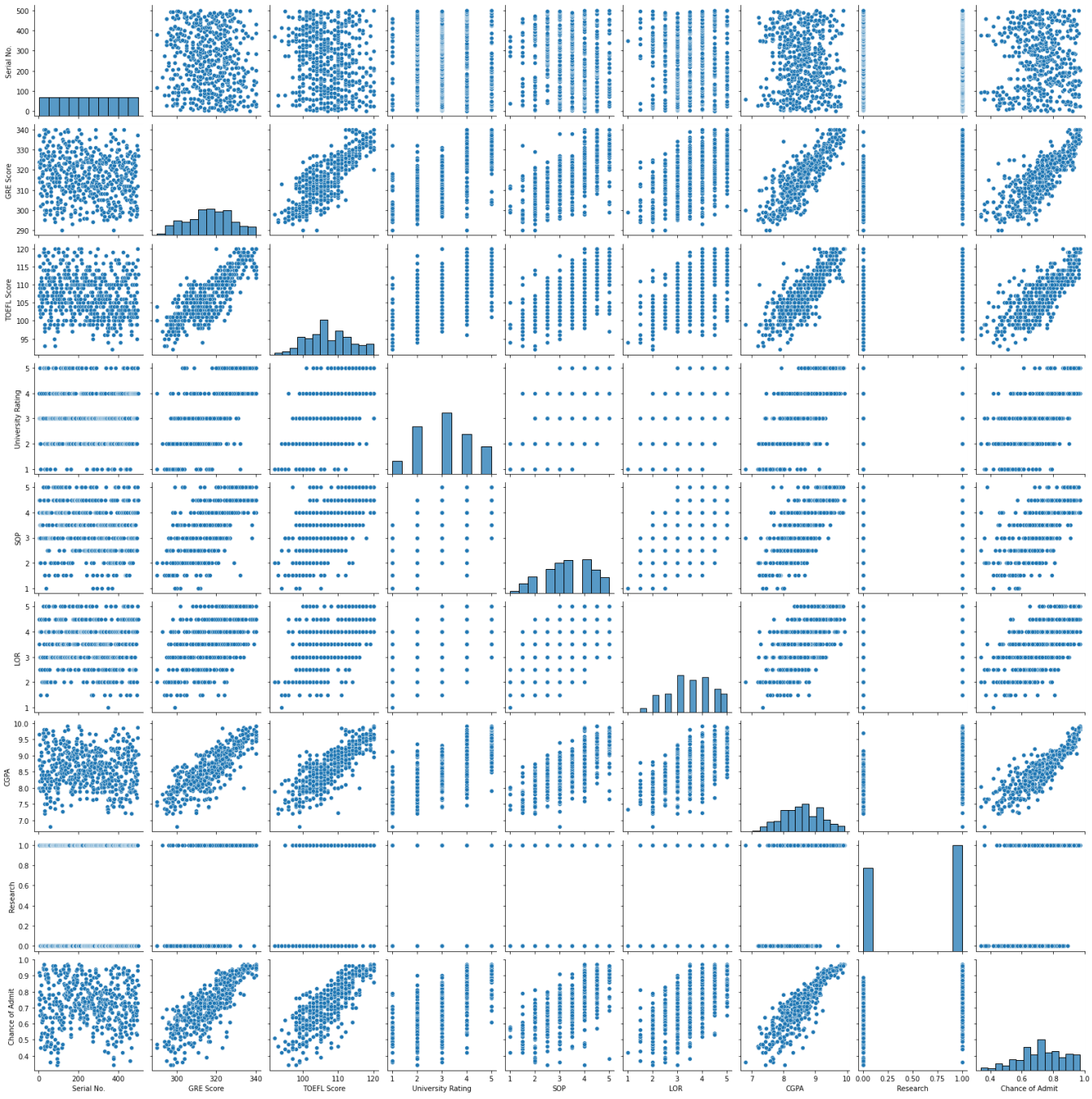
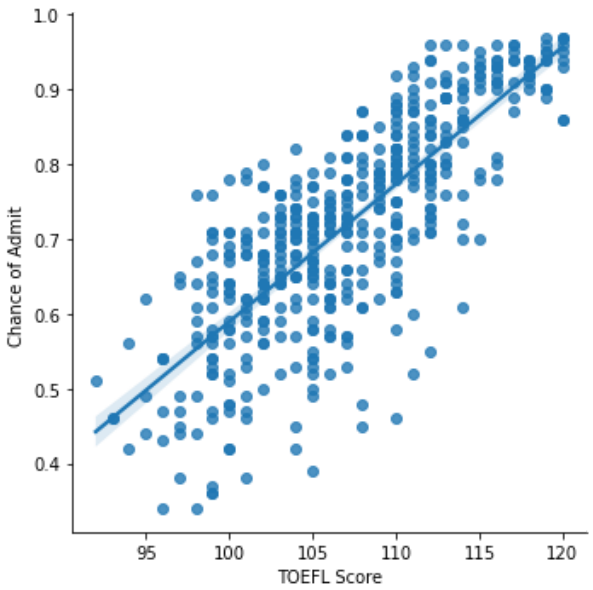


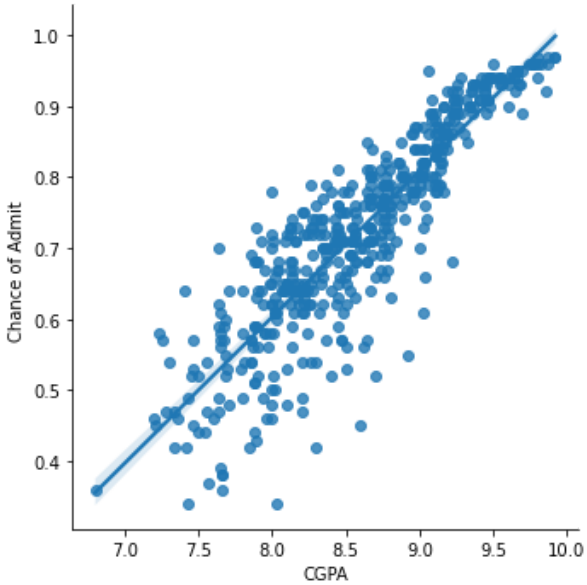
Figure 2: Plots per veure la relació entre les variables

* Els atributs més importants són:

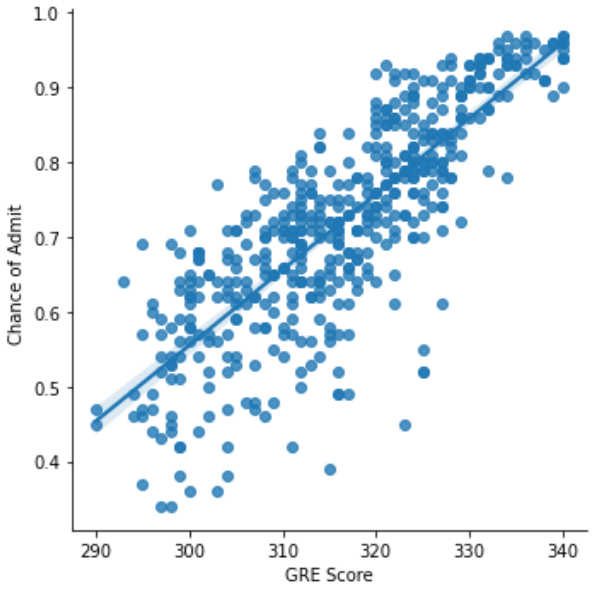
* TOEFL Score



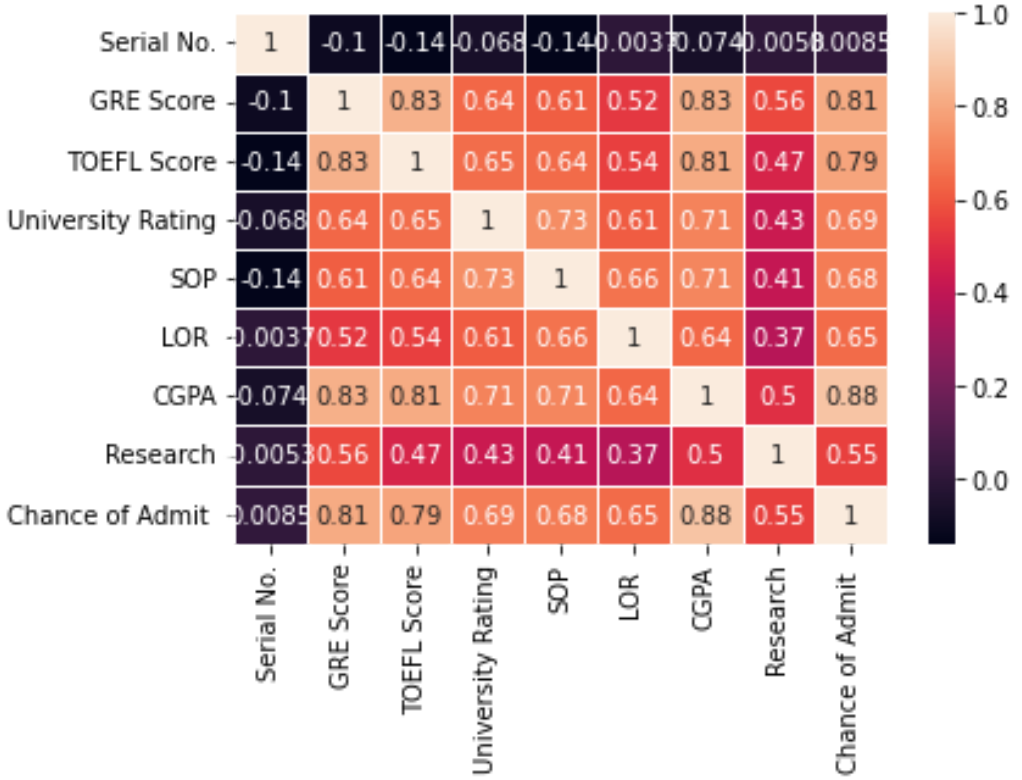
* CGPA



* GRE Score



- Amb quin atribut s'assoleix un MSE menor?
- Quina correlació hi ha entre els atributs de la vostra base de dades?



- Com influeix la normalització en la regressió?
  * No hi cap canvi apreciable.

$$|MSE_{a_i} - MSE_{norm(a_i)}| \approx 0 \quad \forall a_i \in atributs$$

- Com millora la regressió quan es filtren aquells atributs de les mostres que no contenen informació? **Si s'aplica un PCA, a quants components es redueix l'espai? Per què?**
  * L'espai es redueix a 6 dimensions utilitzant PCA. Això és perquè en realitat, el comportament de la variable objectiu es pot explicar majoritariament amb 6 combinacions lineals de les variables explicatives. Això encaixa amb la nostra intuició

	MSE	R^2
Amb tots els atributs	0.0038787238233447614	0.7202959118637930480
Només els que contenen informacio util	0.0040296817550593347	0.7004161749995472253
Reduït utilitzant PCA	0.0045734666537958898	0.6719598817431668980

- Plantejament:
  - Codi:
    * Un cop hem separat les dades, fem servir recursos de diferents llibreries per graficar pairplots o taules de correlacio. Després amb l'ajut de sklearn fem regressió sota diferents condicions per respondre a les preguntes.

```
import seaborn as sns
relacio = sns.pairplot(dataset)
```

```
correlacio = dataset.corr()

plt.figure()
ax = sns.heatmap(correlacio, annot=True, linewidths=.5)
```

```
from sklearn.linear_model import LinearRegression

def regression(x, y):
    # Creem un objecte de regressió de sklearn
```

```
    regr = LinearRegression()

    # Entrenem el model per a predir y a partir de x
    regr.fit(x, y)

    # Retornem el model entrenat
    return regr
```

```
def normalitzar(X):
    mean = X.mean(0)
    std = X.std(0)
    Xnorm = (X - mean[None, :]) / std[None, :]
    return Xnorm
```

- **Apartat A:**
- Preguntes:
  - Com influeixen tots els paràmetres en el procés de descens? Quins valors de learning rate convergeixen més ràpid a la solució òptima? Com influeix la inicialització del model en el resultat final?
    * Podem veure que probant amb els seguents learing rates obtenim:

Learning rate	MSE	R^2	Iterations
$\alpha = 0.01$	0.0049731682434066074	0.6275543887140537791	964
$\alpha = 0.05$	0.0048732128736959741	0.6406451211448758176	299
$\alpha = 0.1$	0.0047236091234896960	0.6538180968627304024	119
$\alpha = 0.3$	0.0046700324701321730	0.6612452142041074232	53

Podem observar que per a learning rates menors triga més iteracions en acabar, i a mesura que el learning rate augmenta, els resultats milloren molt lleugerament pero sobretot el nombre d'iteracions es redueix drasticament.
En quant a la inicialització, probant amb valors aleatoris, tot zeros i tot uns obtenim:

Inicialització	MSE	R^2	Iterations
Aleatori	0.0046785340613144785	0.6605723123755784520	60
Zeros	0.0045173299979205496	0.6750227109039448337	14
Uns	0.0046648392133717322	0.6614899142637367113	60

On podem apreciar que sembla ser indiferent excepte pel cas dels zeros, on requereix moltes menys iteracions per arribar a una bona classificació.

– Quines funcions polinomials (de diferent grau, de diferents combinacions d'atributs, ...) heu escollit per ser apreses amb el vostre descens del gradient? quina ha donat el millor resultat (en error i rapidesa en convergència)?
    * La millor funció polinomial ens ha sortit que era la de primer grau, que té sentit, ja que com em vist en els plots per veure com estaven relacionades les dades ja semblava que amb una recta ja era prou.

	MSE	R^2	Iterations
Polinomi de grau 1	0.0049813664975209299	0.6397399969469377279	2787
Polinomi de grau 2	0.0101140118658220018	0.4564823674184331770	10904
Polinomi de grau 3	0.1052501916410085447	-0.0014081106975005753	20738

– Utilitzeu el regularitzador en la fòrmula de funció de cost i descens del gradient i proveu polinomis de diferent grau. Com afecta el valor del regularitzador?
    * Analitzant el error, sembla que l'nfluencia de el regularitzador sobre el polinomi de grau 1 es negligible, mentre que els polinomis de grau 2 i 3 cambien de forma mes perceptible amunt i avall.

	MSE	R^2	Iterations
Polinomi de grau 1 amb regularitzador = 0.01	0.0049023119032184969	0.5483007560228887467	478
Polinomi de grau 2 amb regularitzador = 0.01	0.0121589719740558926	0.3842716728194959241	11789
Polinomi de grau 3 amb regularitzador = 0.01	0.0602322151516489976	0.0905746042266927232	21107

	MSE	R^2	Iterations
Polinomi de grau 1 amb regularitzador = 0.1	0.0049192813648751277	0.6364193130033390933	1489
Polinomi de grau 2 amb regularitzador = 0.1	0.0099684500793198252	0.5101284850515281910	10734
Polinomi de grau 3 amb regularitzador = 0.1	0.0467578396937517460	0.2125387351869650887	20212

	MSE	R^2	Iterations
Polinomi de grau 1 amb regularitzador = 1	0.0049839575261779751	0.6402485101842463333	2844
Polinomi de grau 2 amb regularitzador = 1	0.0107648869661050237	0.4700495382791941568	11362
Polinomi de grau 3 amb regularitzador = 1	0.0955540518940658362	-0.0098056774100656607	22600

	MSE	R^2	Iterations
Polinomi de grau 1 amb regularitzador = 10	0.0048982997233450627	0.5489585178215357075	472
Polinomi de grau 2 amb regularitzador = 10	0.0154482536602393826	0.3808434462705595491	13731
Polinomi de grau 3 amb regularitzador = 10	0.2109099663642567679	-0.0735707451772946719	18068

– Quina diferència (quantitativa i qualitativa) hi ha entre el vostre regressor i el de la llibreria?

    * Els resultats entre ambdos són bastant similars amb el de la llibreria donant un resultat lleugerament millor (de l'ordre de 10^{-4}).
  – Té sentit el model (polinomial) trobat quan es visualitza sobre les dades?
    * Si, tal i com es pot veure a les gràfiques de l'apartat B.1, totes les variables semblen estar relacionades liealment amb la variable objectiu.
- Plantejament:

```
from sklearn.preprocessing import PolynomialFeatures

class Regressor(object):
    def __init__(self, alpha=0.3, regularitzador=0.0,
initial_bias=None, initial_weight=None):
        # Si no s'ha especificat un valor inicial els pesos
s'asignaran de manera aleatoria
        if initial_bias == None:
            self.w0 = np.random.rand()
        else:
            self.w0 = initial_bias
        if initial_weight == None:
            self.wj = np.array([])
        else:
            self.wj = initial_weight
        self.alpha = alpha
        self.regularitzador = regularitzador
        self.loaded = False

    def predict(self, x):
        return self.w0 + x.dot(self.wj)

    def __cost(self, hy, y):
        return (np.square(hy - y).sum() + self.regularitzador *
 np.square(self.wj).sum()) / (2 * y.size)

    def __update(self, hy, y):
        self.w0 -= self.alpha * (hy - y).sum() / y.size
        self.wj -= self.alpha / y.size * ((hy - y).dot(self.X)
- self.regularitzador * self.wj)

    def train(self, max_iter, epsilon=0.000001):
```

```
        # Assegurarse de que s'ha carregat un dataset
        if not self.loaded:
            raise ValueError("No dataset loaded for training")
        # Entrenar el model fins a fer max_iter iteraions o si
el canvi es menor a epsilon
        hy = self.predict(self.X)
        J = self.__cost(hy, self.y)
        for i in range(max_iter):
            self.__update(hy, self.y)
            hy = self.predict(self.X)
            Jnew = self.__cost(hy, self.y)
            delta = abs(Jnew - J)
            if delta < epsilon:
                break
            J = Jnew

    def load(self, Xtrain, ytrain):
        self.X = Xtrain
        self.y = ytrain
        if self.wj.size != self.X.shape[1]:
            self.wj = np.random.rand(self.X.shape[1])
        self.loaded = True

def make_polinomial(X, degree=1):
    # Transformar la les dades d'X al seu polinomi de grau
degree sense incloure el terme X^0
    poli = PolynomialFeatures(degree=degree,
include_bias=False)
    return poli.fit_transform(X)
```