Nom : **Ayoub HAJJI**

Classe : **GI4**

École : **École Nationale des Sciences Appliquées d'Oujda**

Objet : **Projet Machine Learning – Car Price Prediction**

Superivision : **Mr. Toumi BOUCHENTOUF – Mr. Zakaria HAJA**

I. Problématique :

- Un constructeur automobile chinois intitulé **Geely Auto** voudra entamer le marché américain tout en construisant leur propre usine de construction de véhicule.
- Le but principal c'est pouvoir concurrencer les constructeurs européens et américains, et pour y parvenir, la compagnie veut savoir les divers facteurs qui influencent le prix des véhicules au niveau du marché américain tout en sachant que ces derniers diffèrent d'un marché à un autre (dans notre cas, leur marché local est le marché chinois).

- Alors quels sont les variables qui influencent le plus sur le prix des voitures au niveau du marché américain ?
- Quel est le modèle qui aura la meilleure précision de prédiction ?

➢ Pour y parvenir, nous avons eu la main sur un **dataset** qui contient toutes les variables pouvant influencer le prix des véhicules dans le marché américain.

II. Description du dataset :

- Le dataset contient **25 colonnes** (les **Facteurs**, les **Variables**, ou bien les **Features** influençant sur le **prix**).
- Le dataset contient une **seule colonne Target** qui est le **prix**.
- Le dataset contient **205 enregistrements**.
- En gros, la taille du dataset est **205 lignes * 26 colonnes**.
- Les Informations générales concernant le dataset :

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 205 entries, 0 to 204
Data columns (total 26 columns):
 #   Column            Non-Null Count  Dtype
---  ------            --------------  -----
 0   car_ID            205 non-null    int64
 1   symboling         205 non-null    int64
 2   CarName           205 non-null    object
 3   fueltype          205 non-null    object
 4   aspiration        205 non-null    object
 5   doornumber        205 non-null    object
 6   carbody           205 non-null    object
 7   drivewheel        205 non-null    object
 8   enginelocation    205 non-null    object
 9   wheelbase         205 non-null    float64
 10  carlength         205 non-null    float64
 11  carwidth          205 non-null    float64
 12  carheight         205 non-null    float64
 13  curbweight        205 non-null    int64
 14  enginetype        205 non-null    object
 15  cylindernumber    205 non-null    object
 16  enginesize        205 non-null    int64
 17  fuelsystem        205 non-null    object
 18  boreratio         205 non-null    float64
 19  stroke            205 non-null    float64
 20  compressionratio  205 non-null    float64
 21  horsepower        205 non-null    int64
 22  peakrpm           205 non-null    int64
 23  citympg           205 non-null    int64
 24  highwaympg        205 non-null    int64
 25  price             205 non-null    float64
dtypes: float64(8), int64(8), object(10)
memory usage: 43.2+ KB
```

III. Le preprocessing :

- Supprimer les lignes contenant des valeurs manquantes :

```
#Supprimer les lignes qui contiennent des valeurs manquantes
dataFrame = dataFrame.dropna()
```

- Diviser la colonne de **CarName** en **manufacturer** et **modelName** pour une meilleure visibilité :

```
#Diviser la colonne de CarName en manufacturer et modelName pour une meilleure visibilité
dataFrame[['manufacturer', 'modelName']] = dataFrame.CarName.str.split(" ",n=1, expand=True)

#Changer l'emplacement des colonnes manufacturer et modelName
colonneManufacturer = dataFrame.pop('manufacturer')
dataFrame.insert(2, 'manufacturer', colonneManufacturer)

colonneModelName = dataFrame.pop('modelName')
dataFrame.insert(3, 'modelName', colonneModelName)

#Supprimer la colonne de CarName (l'ancienne colonne)
dataFrame.drop(['CarName'], axis=1, inplace=True)
```

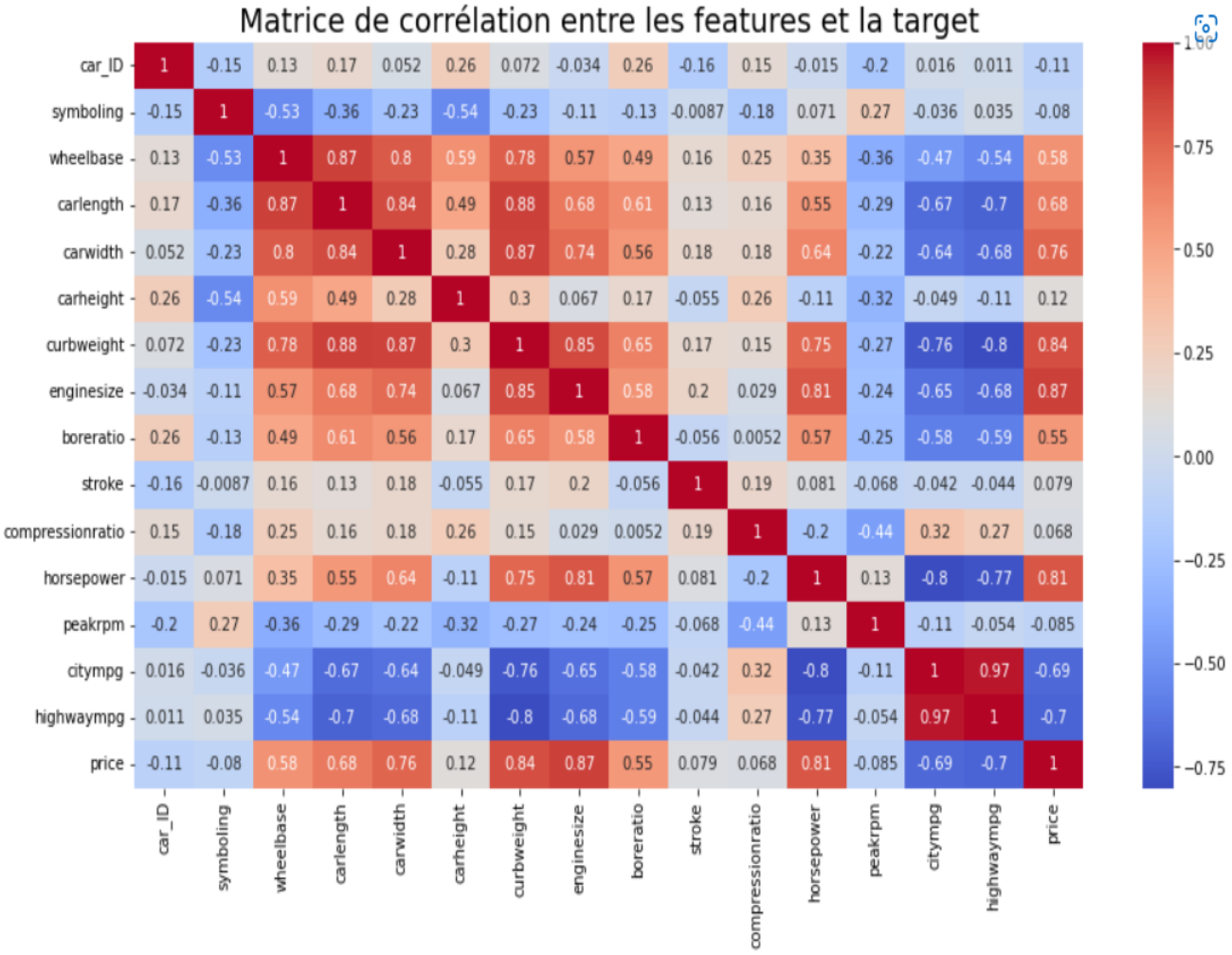
- Corriger les fautes d'orthographe concernant les noms des constructeurs :

```
#Correction des noms des constructeurs de véhicule
dataFrame['manufacturer'] = dataFrame['manufacturer'].str.lower()

#Définir la fonction de correction
def changeName(oldName, newName):
    dataFrame['manufacturer'].replace(oldName, newName, inplace=True)

#Appliquer la fonction
changeName('maxda','mazda')
changeName('porcshce','porsche')
changeName('toyouta','toyota')
changeName('vokswagen','volkswagen')
changeName('vw','volkswagen')

#Afficher la liste des constructeurs
dataFrame['manufacturer'].unique()
```

IV. L'analyse des données :

1. L'affichage de la matrice de corrélation :

- La corrélation entre deux variables nous permet de déterminer le degré d'attachement entre ces deux derniers.
- Sa valeur est comprise entre -1 et +1.
- La corrélation nous permet aussi de déterminer si les deux variables sont **positivement** ou bien **négativement** attachées.
- Si la valeur de corrélation est comprise **entre -0.1 et 0 ou bien 0 et +0.1**, alors on dit que les deux variables se sont **faiblement corrélées**.
- Dans notre cas, ça nous permet de déterminer les variables les plus signifiantes (les plus influenceuses) sur le prix des véhicules.
- La matrice de corrélation du dataset :



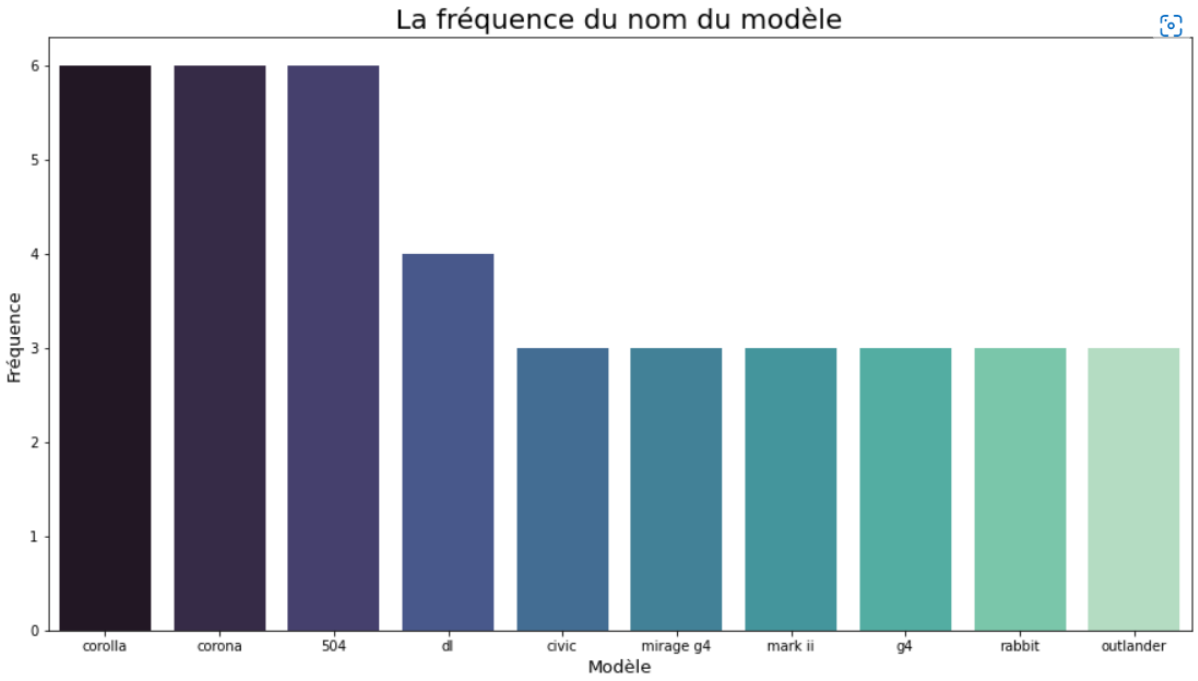
❖ **Réponse à la première question de la problématique :**

- D'après la matrice de corrélation, on constate que les variables qui n'influencent pas sur la **target price** sont : **car_ID**, **symboling**, **stroke**, **compressionratio**, **peakrpm**.
- D'après la matrice de corrélation, on constate que les variables positivement (resp négativement) les plus signifiants sont : **carwidth**, **curbweight**, **enginesize**, **carlength**, **horsepower,** (**citympg**, **highwaympg**).

- Pour ne pas surcharger le dataset, nous allons supprimer les colonnes des variables non signifiantes :

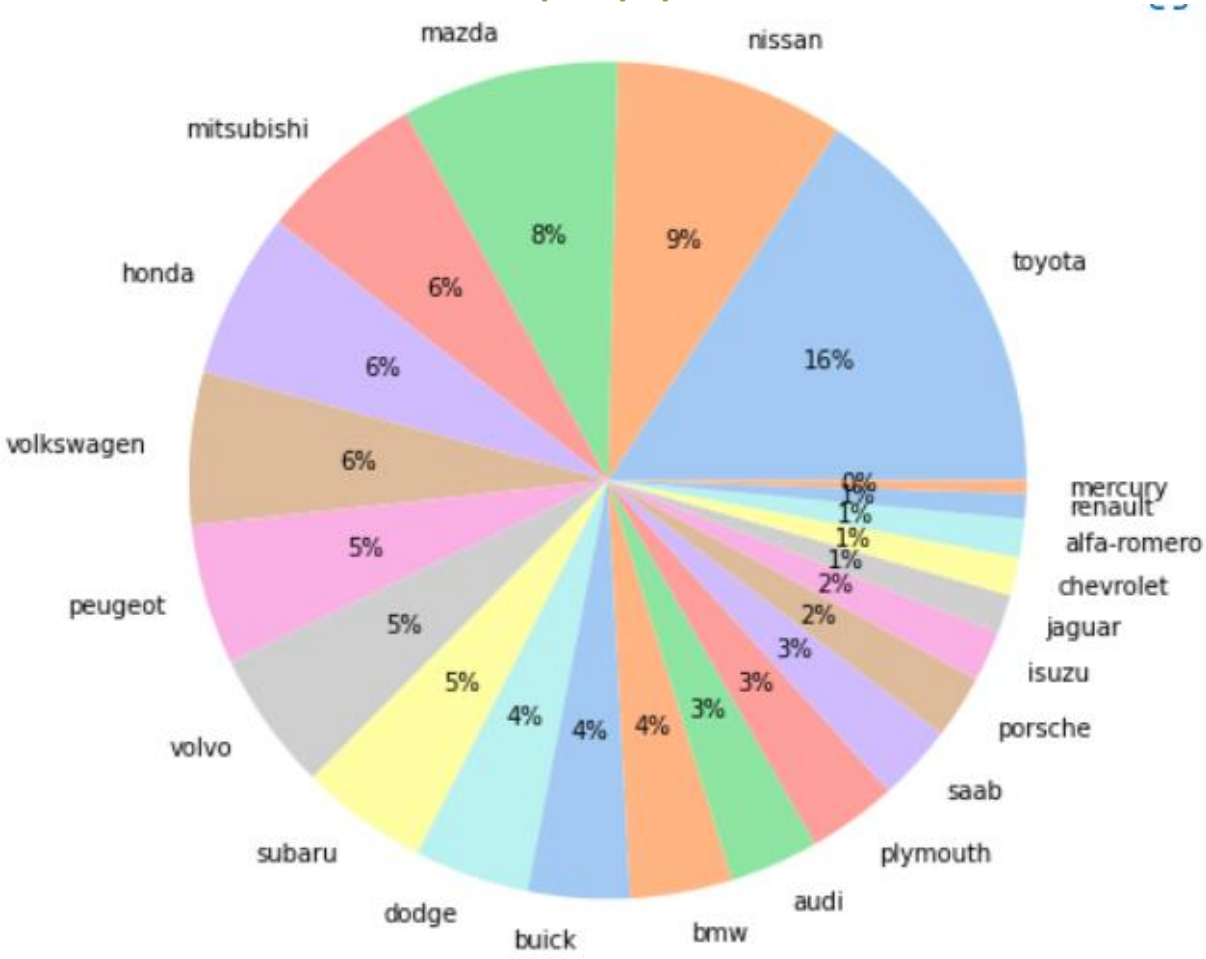
```
#Supprimer les variables qui n'affectent pas la target variable soit positivemen ou negativement
dataFrame.drop(['car_ID', 'symboling', 'stroke', 'compressionratio', 'peakrpm'], axis=1, inplace=True)
```

2. Le nom du modèle le plus populaire :



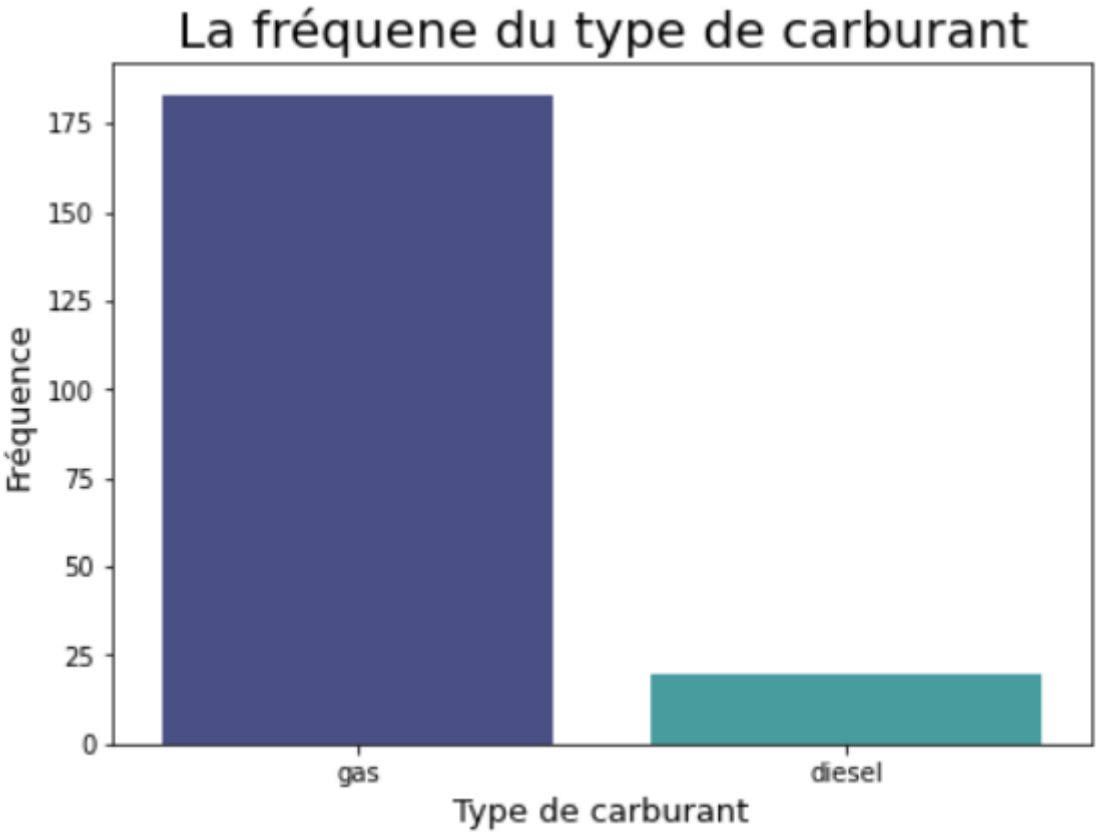
- D'après le graphe, on remarque que les modèles les plus populaires sont : **corolla**, **corona** et **504**.

3. Le nom du constructeur le plus populaire :



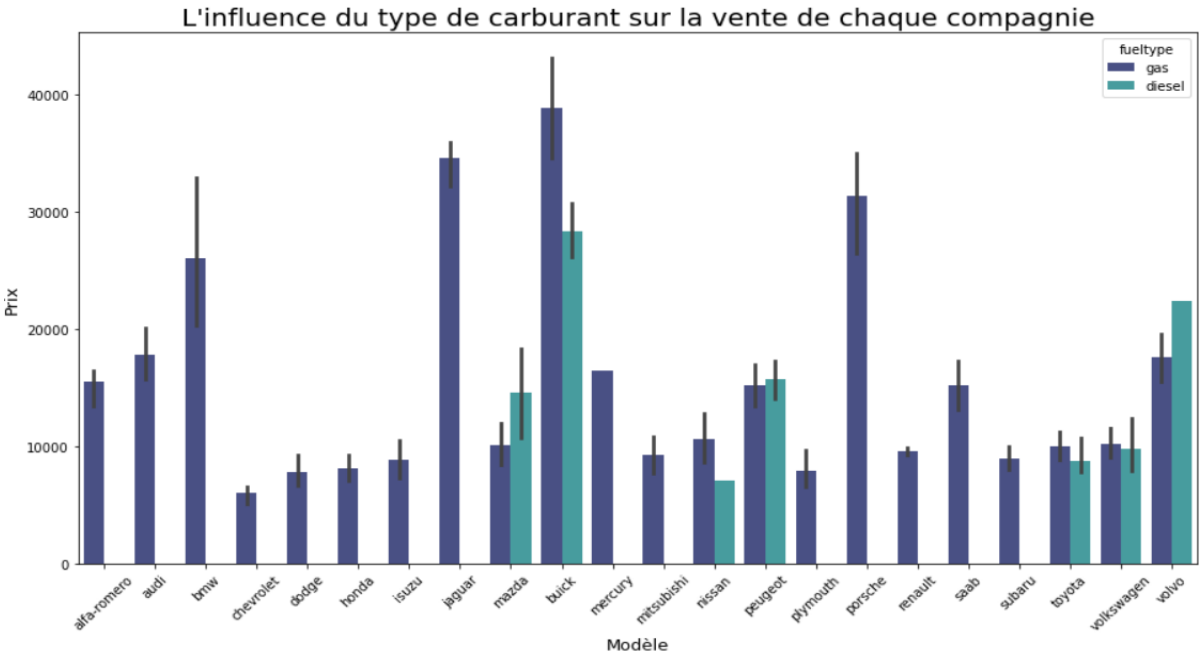
- D’après le graphe, on remarque que le constructeur **Toyota** règne sur le marché américain.

4. Le type de carburant le plus populaire :



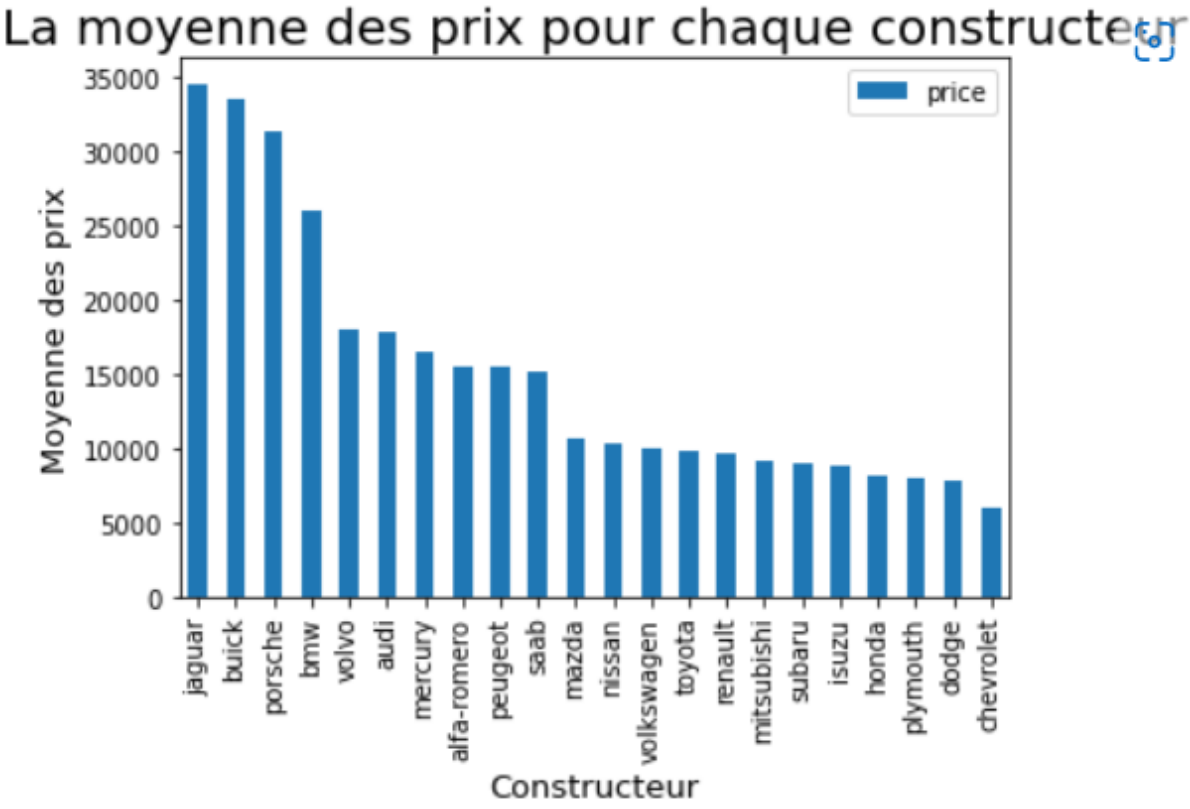
- D’après le graphe, on remarque que le **gas** (**essence**) et plus populaire que le **diesel**.

5. L’influence du type de carburant sur la vente de chaque compagnie :



- D'après le graphe, on remarque que le type carburant **a une influence** sur le prix de vente des véhicules de quelques constructeurs, **buick**, **mazda**… par exemple.

6. La moyenne des prix pour chaque constructeur :



➢ D'après le graphe, nous constatons que les véhicules ayant le prix le plus élevé appartiennent à la marque **Jaguar**.

V. L'apprentissage automatique :

- Définir un dataframe des features et de la target :

```
#Le dataFrame des features
X = dataFrame.drop('price', axis=1)
#Le dataFrame du target
y = dataFrame['price']
```

- Encodage des valeurs non numériques :

```
#Encodage des valeurs non numériques
columnsNaN = [column for column in X if X[column].dtype=='object'] #Les colonnes contenant les valeurs non numériques
X = pd.get_dummies(X, columns=columnsNaN, drop_first=True)

scaling = StandardScaler()
X = scaling.fit_transform(X)
X = pd.DataFrame(X)
```

- Diviser le dataset en deux parties : **TrainSet** et **TestSet** :

```
#Diviser le dataSet des features et du target en test_set et en train_set
X_train,X_test,y_train,y_test = train_test_split(X, y, random_state=42, test_size=0.2)
```

- Définition d'une fonction de calcul de performance des différents modèles :

```
#Afficher les performaances de chaque modèle
#Définition de la fonction de calcul de performance
def evaluerModel(model):
    model.fit(X_train,y_train)
    y_pred=model.predict(X_test)

    #Afficher les caractéristiques de chaque modèle
    print("Nom du modèle: ",model)
    print("Score du modèle : "+str(model.score(X_test, y_test) * 100))
    print("MSE : ",mean_squared_error(y_test,y_pred))
    print("MAE : ",mean_absolute_error(y_test,y_pred))
    print("------------------------------------------------------------")
```

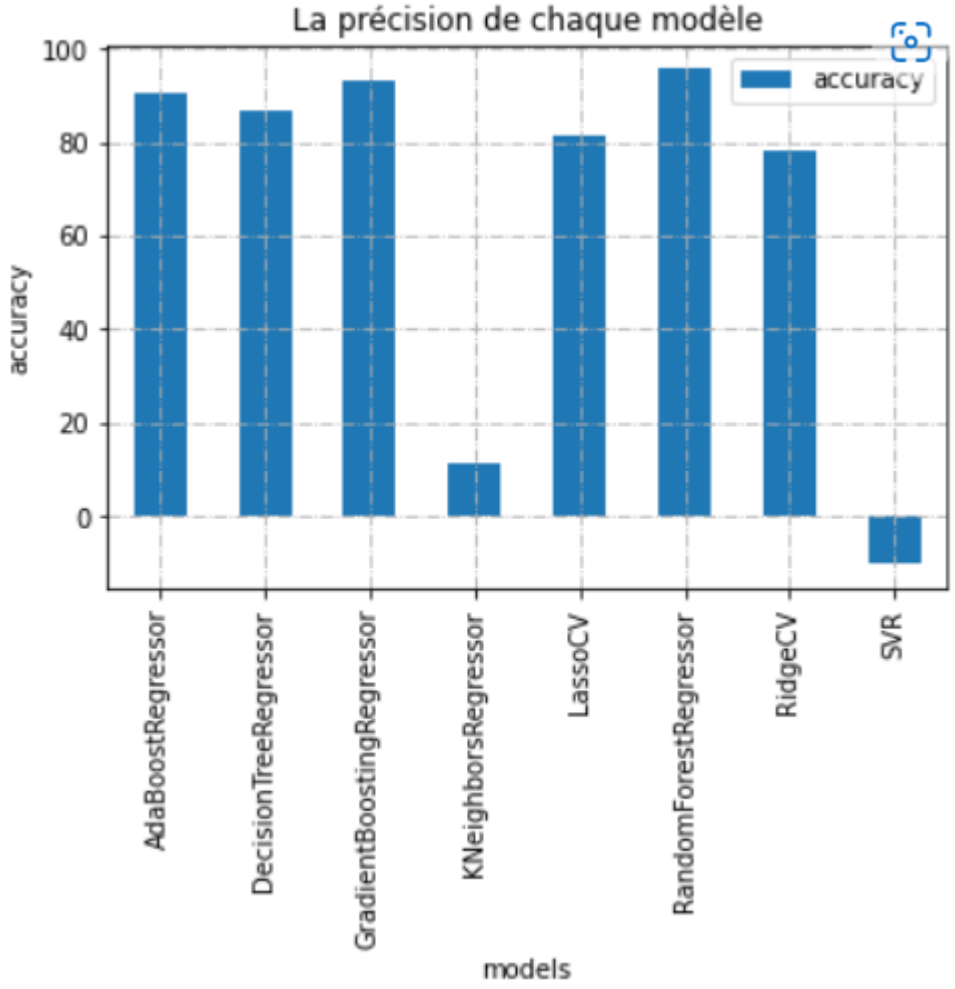
- Initialisation des différents modèles et application de la méthode de calcul des performances :

```python
#Initialisation des modèles
linearRegression = LinearRegression()
lasso = LassoCV()
ridge = RidgeCV()
knn = KNeighborsRegressor()
svr = SVR()
decisionTreeRegressor = DecisionTreeRegressor()

randomForestRegressor = RandomForestRegressor(n_estimators=200,random_state=42)
ada = AdaBoostRegressor(random_state=42)
gbr = GradientBoostingRegressor(random_state=42)

#La liste des modèles
listModels = [linearRegression, lasso, ridge, knn, svr, decisionTreeRegressor, randomForestRegressor, ada, gbr]

#Afficher les performances de chaque modèle
for model in listModels:
    evaluerModel(model)
```

1. Comparaison entre les modèles : Version du tableau

	models	accuracy
0	LassoCV	81.4356795536708
1	RidgeCV	77.95545803580434
2	KNeighborsRegressor	11.139026832798372
3	SVR	-10.206782129691705
4	DecisionTreeRegressor	86.55838062075615
5	RandomForestRegressor	95.51809727802113
6	AdaBoostRegressor	90.2297120082706
7	GradientBoostingRegressor	93.15610059339122

2. Comparaison entre les modèles : Version du graphe



❖ **Réponse à la deuxième question de la problématique :**

➢ Daprès le graphe, on constate que le **RandomForestRegressor est le modèle le plus performant avec une précision de 95.52%.**