Let’s apply some basics on linear regression :

```
# Loading libraries
from sklearn.linear_model import LinearRegression
from sklearn.datasets import make_regression
# Generation of a feature matrix and a target vector
features, target = make_regression(n_samples = 100,
                                   n_features = 3,
                                   n_informative = 2,
                                   n_targets = 1,
                                   noise = 0.2,
                                   coef = False,
                                   random_state = 1)
# Creation of linear regression
regression = LinearRegression()

#Linear Regression Fit
model = regression.fit(features,target)

#intercept
model.intercept_

# Coefficients
model.coef_

#First value in the target vector
target[0]

#Prediction of the target value of the first observation
model.predict(features)[0]

#Display of the model's score on training data
print(model.score(features,target))
```

Problem: You have a characteristic whose effect on the target variable depends on another characteristic.

Solution: create an interaction term to capture this dependency using PolynomialFeatures de scikit-learn

```
#Libraries
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import PolynomialFeatures
from sklearn.datasets import make_regression
# Generation of a feature matrix and a target vector
features, target = make_regression(n_samples = 100,
                                   n_features = 2,
                                   n_informative = 2,
                                   n_targets = 1,
                                   noise = 0.2,
                                   coef = False,
                                   random_state = 1)

#Creating an interaction term
interaction = PolynomialFeatures(degree=3 , include_bias=False , interaction_only=True)
features_interaction = interaction.fit_transform(features)

# Linear regression creation
regression = LinearRegression()

# Linear Regression Fit
model = regression.fit(features_interaction,target)

#print caracteristics of the first observation
features[0]
```

Problem: You want to model with a nonlinear relationship.

Solution : Create a polynomial regression by including polynomial features in a linear regression model

```
#Libraries
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import PolynomialFeatures
from sklearn.datasets import make_regression
# Generation of a feature matrix and a target vector
features, target = make_regression(n_samples = 100,
                                   n_features = 3,
                                   n_informative = 2,
                                   n_targets = 1,
                                   noise = 0.2,
                                   coef = False,
                                   random_state = 1)

#Creating a polynomial caracteristics x^2 x^3
polynomial = PolynomialFeatures(degree=3 , include_bias=False)
features_polynomial = polynomial.fit_transform(features)

# Linear regression creation
regression = LinearRegression()

# Linear Regression Fit
model = regression.fit(features_polynomial,target)

#print caracteristics of the first observation
features[0]
```

Reducing variance through regularisation

Problem : You want to reduce the variance of your linear regression model
Solution: Use a learning algorithm that includes a regularization such as Ridge regression or Lasso regression.

```
# Loading libraries
from sklearn.linear_model import Ridge
from sklearn.preprocessing import StandardScaler
from sklearn.datasets import make_regression

# Generation of a feature matrix and a target vector
features, target = make_regression(n_samples = 100,
                                   n_features = 3,
                                   n_informative = 2,
                                   n_targets = 1,
                                   noise = 0.2,
                                   coef = False,
                                   random_state = 1)
# Standardisation of characteristics
scaler = StandardScaler()
features_standardized = scaler.fit_transform(features)

# Creation of Ridge Regression with alpha
regresion = Ridge(alpha=0.5)

# Fit regression
model = regression.fit(features_standardized, target)
```

Scikit-learn offer RidgeCV to determine the best value of alpha

```
# Loading libraries
from sklearn.linear_model import Ridge
from sklearn.preprocessing import StandardScaler
from sklearn.datasets import make_regression

# Generation of a feature matrix and a target vector
features, target = make_regression(n_samples = 100,
                                   n_features = 3,
                                   n_informative = 2,
                                   n_targets = 1,
                                   noise = 0.2,
                                   coef = False,
                                   random_state = 1)
# Standardisation of characteristics
scaler = StandardScaler()
features_standardized = scaler.fit_transform(features)

# Loading libraries
from sklearn.linear_model import RidgeCV

# create ridge regression with 3 alpha
regr_cv = RidgeCV(alphas=[0.1, 1.0, 10.0])

#fit linear regression
model_cv = regr_cv.fit(features_standardized, target)

# print coefficients
model_cv.coef_

# print best value
model_cv.alpha_
```

Problem: Characteristics reduction to simplify the regression model

Solution: Use of Lasso

```
# Loading libraries
from sklearn.linear_model import Lasso
from sklearn.preprocessing import StandardScaler
from sklearn.datasets import make_regression

# Generation of a feature matrix and a target vector
features, target = make_regression(n_samples = 100,
                                   n_features = 3,
                                   n_informative = 2,
                                   n_targets = 1,
                                   noise = 0.2,
                                   coef = False,
                                   random_state = 1)
# Standardisation of characteristics
scaler = StandardScaler()
features_standardized = scaler.fit_transform(features)


#fit linear regression
regression = Lasso(alpha=0.5)
model = regression.fit(features_standardized, target)

# print coefficients
model.coef_
```

TP : Prédiction du Prix des Maisons avec Régression Linéaire et Régularisation

Objectif :

L'objectif de ce TP est d'appliquer les techniques de régression linéaire sur une base de données réelle, en explorant les termes d'interaction, la régression polynomiale et la régularisation (Ridge, Lasso).

Données :

Nous utiliserons la base de données **fetch_california_housing** de sklearn.datasets, qui contient des informations sur les prix de l'immobilier à Boston.

1. Chargement et Exploration des Données

1. Importer les bibliothèques nécessaires (numpy, pandas, matplotlib, seaborn, sklearn)
2. Charger la base de données , visualiser les données.

2. Préparation des Données

1. Séparer les features (X) et la variable cible (y)
2. Normaliser les features pour améliorer l'apprentissage
3. Diviser les données en un ensemble d'entraînement et de test (80% - 20%)

3. Régression Linéaire Simple

1. Entraîner un modèle de régression linéaire
2. Calculer l'erreur quadratique moyenne (MSE)
3. Analyser les performances du modèle

4. Ajout de Termes d'Interaction et Régression Polynomiale

1. Créer des termes d'interaction avec PolynomialFeatures de Scikit-Learn
2. Tester une régression polynomiale d'ordre 2 et comparer avec la régression linéaire
3. Observer les effets sur les performances

5. Régularisation : Ridge et Lasso

1. Entraîner une régression Ridge en utilisant RidgeCV pour trouver le meilleur hyperparamètre alpha
2. Faire de même avec Lasso (LassoCV)
3. Comparer les coefficients des modèles Ridge et Lasso
4. Observer la sélection de variables par Lasso

6. Comparaison et Conclusion

1. Comparer les performances des différents modèles
2. Discuter des avantages de chaque approche
3. Analyser les meilleures méthodes pour prédire les prix immobiliers