Polytech Nice-Sophia

Mathématiques appliquées et modélisation

5ème Année

Option Informatique et Mathématiques Appliquées
à la Finance et à l'Assurance

PROJET D'INTELLIGENCE ARTIFICIELLE APPLIQUÉE À L'ACTUARIAT

Superviseur : M.Akre

Groupe 4 :
Marco DELHOMME, Victor DELOR et Mohamed BELKAHIA

14 Janvier 2021

Sommaire

I. Introduction

Entrepreneur en Assurance, notre société MMV souhaite commercialiser un produit d'assurance « Temporaire décès ». Cependant nous n'avons pas encore le budget nécessaire pour employer un spécialiste de calcul des primes avec les formules actuarielles. Néanmoins, nous avons accès facilement et à moindre coût aux prix qu'exercent nos concurrents pour ces produits sur le marché. Nous avons donc pris la décision d'utiliser un logiciel d'Intelligence Artificiel (IA) pour qu'il puisse apprendre le modèle de prix sous-jacent.
Cet outil devra donc nous permettre de fournir une prédiction fiable de la prime aux nouveaux clients.

II. La base de données

1) Création de la base de données

a) Paramètres à prendre en compte

Dans un premier temps, nous devons générer le dataset dont nous avons besoin pour l'IA. Il doit être composé d'un set limité de paramètres, essentiels au calcul de la prime du « Temporaire décès », c'est-à-dire l'âge, le nombre de paiements de primes annuelles, la maturité, le taux d'intérêt, et le montant garanti.
Les valeurs de ces paramètres sont choisies de manière à couvrir l'ensemble des cas des clients qui souscriront à notre assurance et sont représentés dans le tableau suivant :

Age	Nombre de paiements de primes annuelles	Maturité	Taux d'intérêt	Montant garanti
20	1	1	0	500
30	5	5	0.5%	1,000
40	10	10	1.0%	2,000
50	20	20	1.5%	4,000
60	30	30	2.0%	8,000
	40	40	2.5%	

b) Construction

À présent, il nous faut créer toutes les combinaisons entre ces paramètres tout en respectant la condition que le nombre de paiements de primes annuelles soit inférieur ou égale à la maturité. Cette base de données a été créé avec python grâce à la bibliothèque sqlite3. Cette

bibliothèque permet d'utiliser des requêtes SQL sur un tableau pour en extraire des informations.
Pour avoir l'ensemble des combinaisons possibles il suffit d'établir le produit cartésien entre les différentes variables, et réaliser quelques exceptions sur ces derniers afin de supprimer les résultats de couples de données non désirés.

c) Prime associée

Une fois les combinaisons trouvées, il ne restait plus qu'à ajouter une dernière donnée à notre base de données : la target correspondant à la prime associée à chaque jeu de paramètres.
Ne disposant pas des données concurrentes dans le cadre du projet, nous avons utilisé les formules actuarielles de la prime annuelle d'assurance de la temporaire décès pour la calculer :

$$ {}_nA_x = \sum_{k=0}^{n-1} v^{k+1} * {}_{k|1}q_x $$

$m\ (m \leq n)$	$P = \frac{{}_nA_x}{\ddot{a}_{x:m}}$

Les tables de mortalité utilisées pour les calculs sont les dernières tables en vigueur.

Ces primes ont été rajoutées à chaque ligne de données puis nous avons exporté toute la dataset composée donc de 6 variables (5 paramètres et la cible) en un fichier « .csv ». Ce type de fichier est utile puisqu'il est de petite taille et donc facilement transmissible, et il peut être facilement travailler sur Excel et Python.

2) Analyse des données

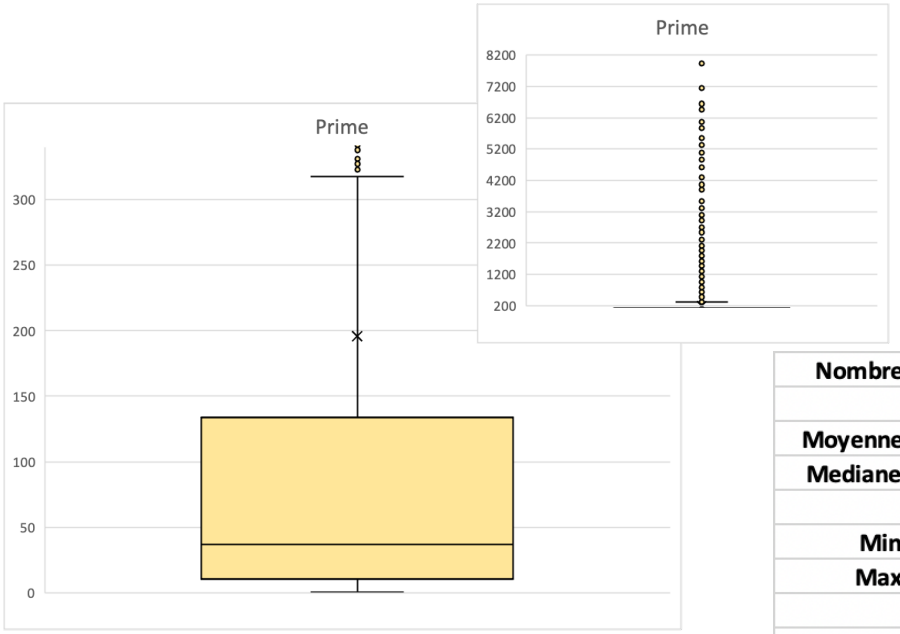
L'analyse des données est une étape essentielle dans le développement d'un modèle d'intelligence artificielle. Elle permet de comprendre si notre database correspond bien à nos attentes, de savoir où se situe la majorité de nos valeurs et donc de détecter des valeurs aberrantes, mais aussi d'avoir une idée de l'impact de chaque paramètre sur notre target.

Pour réaliser ces études, nous avons décidés d'importer notre fichier « dataset.csv » sur Excel, puis d'utiliser les fonctions d'analyse de ce logiciel.

Tout d'abord, voici un aperçu de notre fichier « dataset.csv » qui ne comporte pas d'erreurs de données puisqu'il a été créé sous contraintes précédemment :

```
age,nb_payements,maturity,interest_rate,amount,target
20,1,1,0,500,0.5105083854793219
20,1,1,0,1000,1.0210167709586437
20,1,1,0,2000,2.0420335419172875
20,1,1,0,4000,4.084067083834575
20,1,1,0,8000,8.16813416766915
20,1,1,0.005,500,0.5079685427654944
20,1,1,0.005,1000,1.0159370855309888
20,1,1,0.005,2000,2.0318741710619777
20,1,1,0.005,4000,4.063748342123955
20,1,1,0.005,8000,8.12749668424791
20,1,1,0.01,500,0.5054538470092296
20,1,1,0.01,1000,1.0109076940184591
20,1,1,0.01,2000,2.0218153880369183
20,1,1,0.01,4000,4.043630776073837
20,1,1,0.01,8000,8.087261552147673
20,1,1,0.015,500,0.5029639265806127
20,1,1,0.015,1000,1.0059278531612255
20,1,1,0.015,2000,2.011855706322451
20,1,1,0.015,4000,4.023711412644902
20,1,1,0.015,8000,8.047422825289804
```

L'analyse de ces données peut être synthétisée par une boite à moustache représentant l'ensemble des primes du fichier :



Nombre de lignes :	3150
Moyenne de la prime	195.792
Mediane de la prime	36.616
Minimum :	0.498
Maximum :	7937.992
1er quartile :	10.084
2eme quartile :	36.616
3ème quartile :	133.661

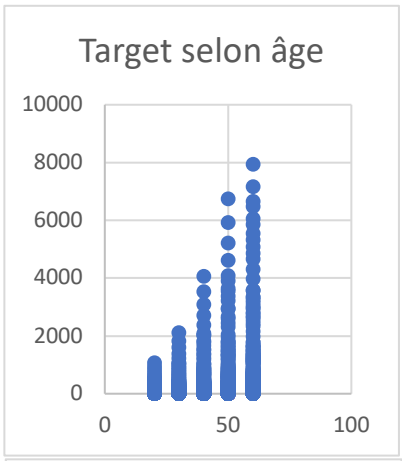
Notre dataset comporte donc 3150 sets d'informations et les primes associées à ces jeux de données sont réparties de 0.498€ à 7937.992€. Bien que la moyenne des primes soit située à 195€, ici il est plus intéressant de noter que la médiane se situe à 36€, ce qui est bien inférieur au maximum de notre échantillon ainsi qu'à la moyenne, ce qui montre qu'il y a une grande dispersion entre les petites et les grandes valeurs de primes.

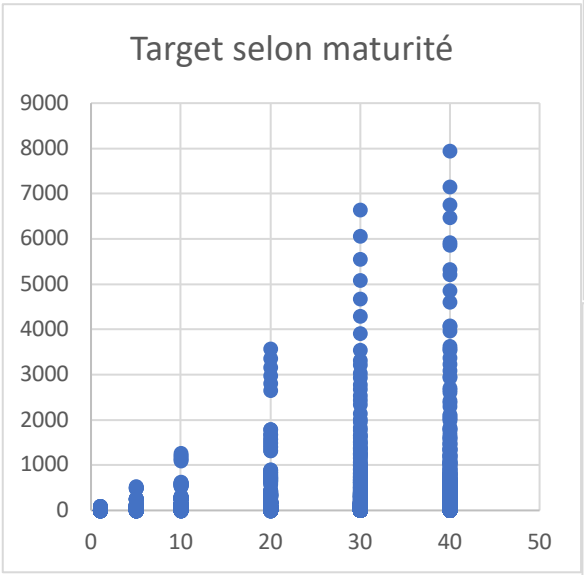
Cela se confirme par l'analyse des quartiles qui montrent que 75% des primes se situent entre 0.5€ et 133.66€.

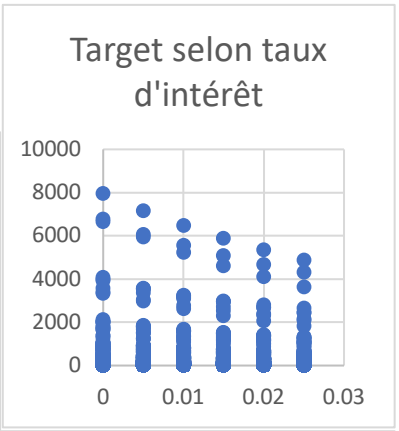
De plus, l'écart interquartile étant égal à 123.58, on peut facilement identifier des valeurs aberrantes en ajoutant cet écart multiplié par 1.5 au 3ème Quartile. Il en revient que toutes les primes supérieures à 319€ peuvent être considérées comme aberrantes, ce qui correspond à 13% de la database.

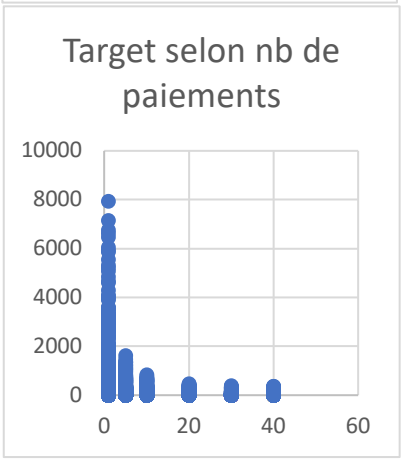
Ces données pourraient être de mauvais éléments lors de notre future approximation par nos modèles d'IA et sont donc à prendre au sérieux.

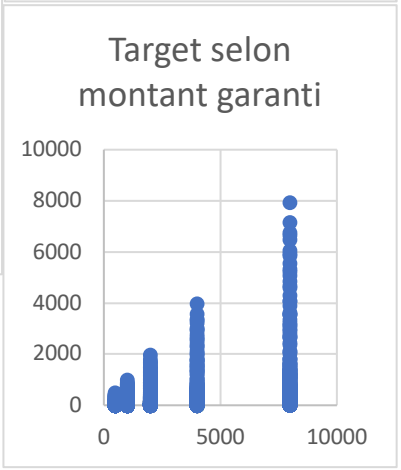
Il est également primordial de connaitre l'influence de chacun de nos paramètres sur la prime. Cependant, il n'est pas possible de réaliser un graphe en 6 dimensions pour avoir toute l'information. Nous avons donc décidé de représenter la prime en fonction de chaque paramètre un à un :





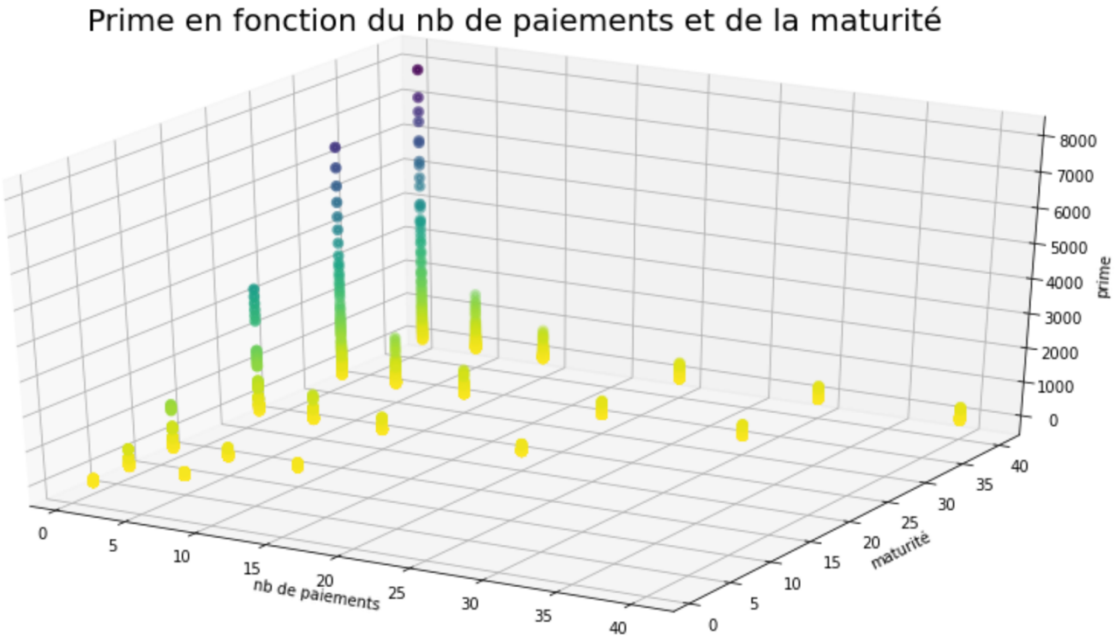






Ces graphiques permettent de comprendre l'influence des paramètres et notamment pour quelles valeurs de ceux-ci la prime atteindra de grandes valeurs. En effet on remarque que pour un nombre de paiements petit et de grand montant garanti, la prime risque d'atteindre des valeurs excessives.

De plus, après avoir tracé la prime en fonction de plusieurs couples de paramètres, nous avons porté notre attention sur ce graphe :



En effet, cette représentation en 3 dimensions présentant la prime en fonction du nombre de paiements de primes annuelles et de la maturité permet de conclure que la prime augmente de manière exponentielle lorsque le rapport $\frac{maturité}{nbre\ de\ paiements}$ grandit.

III. Développement du modèle d'IA

1) Séparation des données

En apprentissage automatique, une tâche courante est l'étude et la construction d'algorithmes qui peuvent apprendre et faire des prédictions sur les données. Ces algorithmes fonctionnent en faisant des prédictions ou des décisions basées sur les données, en construisant un modèle mathématique à partir des données d'entrée. Les données utilisées pour construire le modèle final proviennent généralement de plusieurs ensembles de données. En particulier, trois ensembles de données sont couramment utilisés à différentes étapes de la création du modèle. Le modèle est initialement ajusté sur un ensemble de données d'apprentissage, qui est un ensemble d'exemples utilisés pour ajuster les paramètres. Successivement, le modèle ajusté est utilisé pour prédire les réponses aux observations dans un second ensemble de données appelé ensemble de données de validation. Enfin, l'ensemble de données de test est un ensemble de données utilisé pour fournir une évaluation non biaisée d'un ajustement final du modèle sur l'ensemble de données d'entraînement.

Dans notre cas, nous avons utilisé deux fois la fonction *train_test_split* de la bibliothèque *mglearn*, mais la seconde fois nous avons utilisé les données d'entraînement et les diviserons en ensemble d'apprentissage et de validation, comme on peut le voir dans le code suivant.

```
X=bd2
X_train, X_test, y_train, y_test = train_test_split(X, target2, random_state=1 )
X_test, X_val, y_test, y_val = train_test_split(X_test, y_test, random_state=1 )
```

2) Différents modèles de régression

L'entraînement est le processus par lequel les algorithmes se combinent aux données pour créer un modèle. L'algorithme « apprend » à partir des données que nous lui donnons, et cet apprentissage se reflète dans le modèle (créer ici par des fonctions du package *mglearn*).
Dans notre cas, nous avons essayé plusieurs modèles et ceci dans le but de récupérer celui qui possède les meilleurs scores. Nous présentons ici chacun des modèles et présenterons à chaque fois l'évolution des scores des différents ensembles de training, de validation et de test en fonction respectivement de leurs tailles.

a) Modèle linéaire

Commençons alors par le modèle de régression linéaire, qui représente le fait d'expliciter un jeu de données cible à partir d'un autre jeu de données explicatives moyennant une équation polynomiale de degré 1 que l'on peut schématiser comme suit :

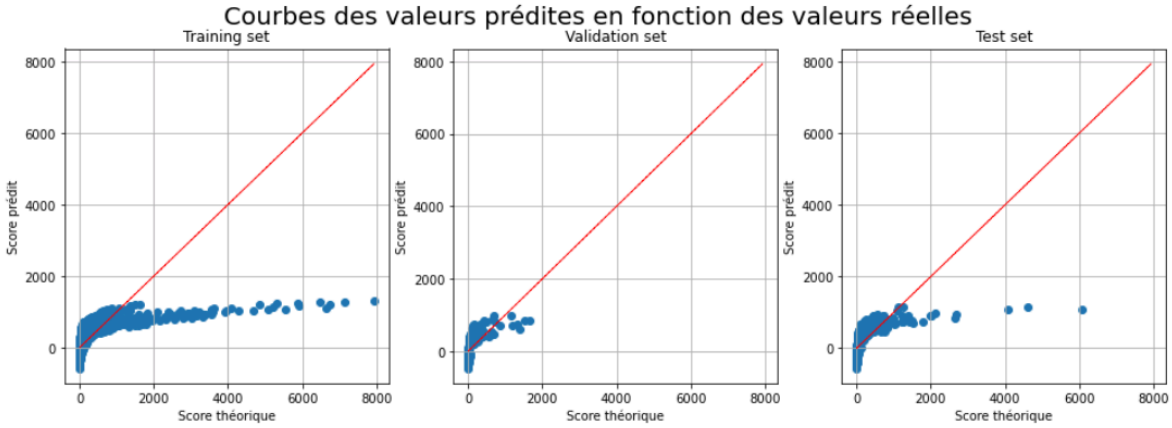
$$Y = \beta_0 + \beta_1 X + \varepsilon$$

- β0 et β1 sont les paramètres du modèle
- ε l'erreur d'estimation
- Y variable expliquée
- X variable explicative.

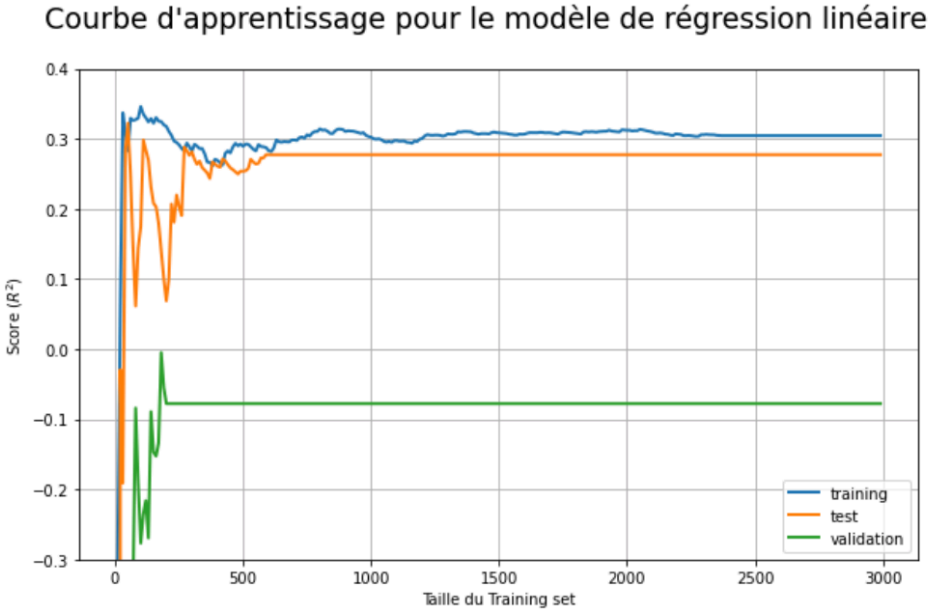
En utilisant cette méthode avec notre jeu de données, nous avons eu les résultats suivants :

```
Training set score: 0.305
Validation set score: -0.078
Test set score: 0.277
```

Ces scores ne sont pas satisfaisants et présente un écart notable avec 1, ce qui nous a permis de conclure que ce modèle n'approxime pas bien notre jeu de données. Ainsi, les graphes suivants justifient nos dires :



Ils montrent bien que le score prédit est bien loin du score théorique attendu. Les courbes d'apprentissage montrent bien cette inexactitude :



On peut remarquer que ce modèle possède des scores faibles globalement. En effet, on constate que pour la validation set, son score reste négatif même avec l'augmentation de la taille de l'ensemble. Cette courbe présente quelques fluctuations avec les petites valeurs et commence à se stabiliser après au tour de la valeur de -0,07 qui correspond bien à la valeur du score trouvé précédemment. Pour ce qui est des courbes du training et de test, on peut remarquer qu'il présente des comportements assez similaires mais avec des valeurs plus

basses pour la courbe de test. Au fur et à mesure qu'on augmente la taille des ensembles, les deux courbes se rapprochent et se stabilisent au tour des valeurs 0,27 pour l'ensemble de test et 0,3 pour l'ensemble de training. Ces scores correspondent bien aux valeurs trouvées précédemment, ce qui peut prouver encore plus nos résultats.

De ce fait, nous avons décidé d'utiliser l'approximation polynomiale avec un degré supérieur.

b) Modèle Polynomial

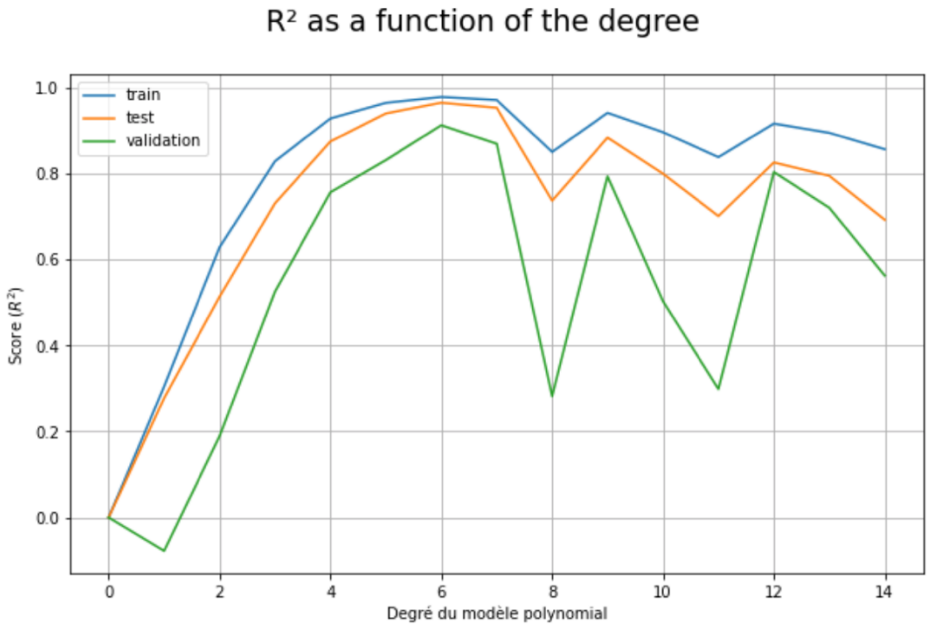
La régression polynomiale est une analyse qui décrit la variation d'une d'un jeu de données expliquée à partir d'une fonction polynomiale d'une variable aléatoire explicative. Le polynôme qui nous permet d'établir ce modèle s'écrit sur la forme suivante :

$$P_n(x) = a_n x^n + a_{n-1} x^{n-1} + \ldots + a_1 x + a_0,$$

permettant d'écrire

$$Y_i = P_n(X_i) + \varepsilon_i$$

Pour déterminer le degré du polynôme qu'il faut choisir, nous avons décidé d'établir le graphe qui modélise l'évolution du score (R^2) en fonction du degré du polynôme choisie.

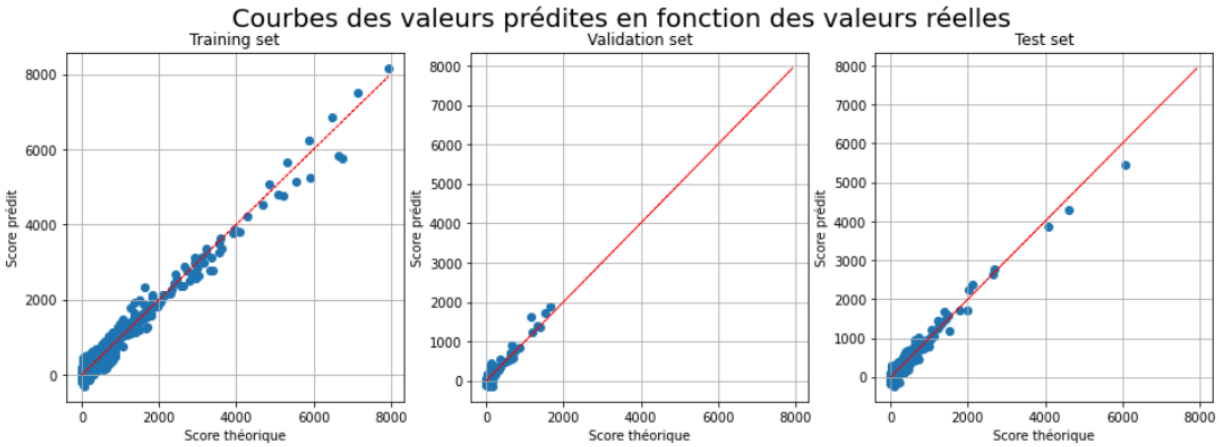


A partir de ce graphe, nous pouvons conclure que le meilleur score est obtenue pour les sets d'entrainement, de validation et de test est celui associé au polynôme de degré 6. Ce polynôme nous a permis d'obtenir les scores suivants :

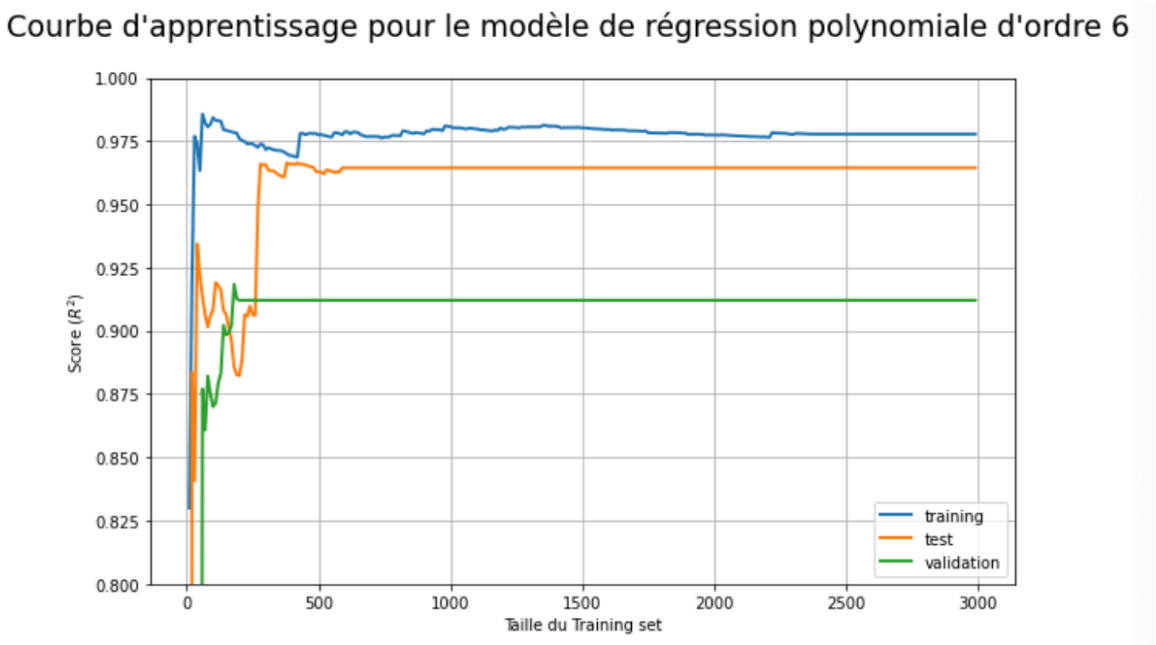
```
Poly Training set score: 0.978
Poly Test set score: 0.965
Poly validation set score: 0.912
```

On obtient donc, des résultats satisfaisants et qui mettent en relief le fait qu'avec notre base de données, ce modèle prédit bien la cible qui est la prime.

On peut également justifier d'avantage nos remarques avec les schémas suivants qui représentent l'évolution des valeurs prédites en fonction des valeurs théoriques :



Les points sont proches de la droite qui modélise f(x) = x, ce qui nous permet de conclure que l'erreur entre les valeurs prédites et les valeurs réelles est faible.



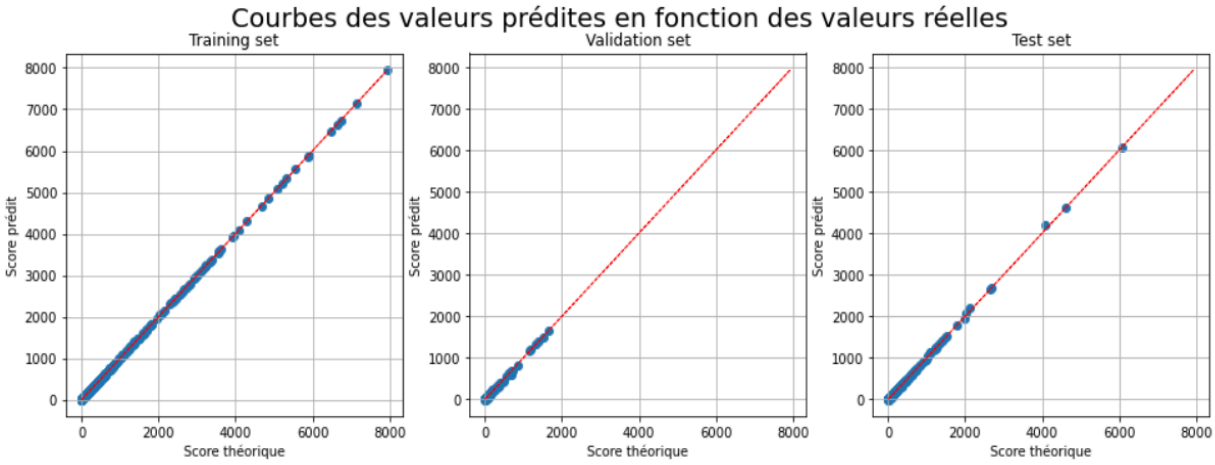
Finalement, à partir des courbes d'apprentissage nous pouvons remarquer que ce modèle possède des scores satisfaisants globalement. En effet, ces trois courbes présentent des comportements assez similaires mais avec des scores différents. On peut noter aussi quelques fluctuations avec les petites valeurs atteignant des scores moins satisfaisants autour de 0,8 pour les ensembles de validation et de test et au tour de 0,83 pour le training. On remarque aussi que plus on augmente la taille des ensembles, plus ces courbes se stabilisent et parvenant même à des valeurs très satisfaisant identiques aux résultats trouvés précédemment.

c) Modèle Scaled Polynomial

Cette méthode reprend le modèle de régression polynomiale mais avec des données normalisées. Dans notre étude nous avons optés pour la normalisation Minimum-Maximum, cette dernière nous permet de redimensionner notre jeu de donnée à l'intervalle [0,1]. Le redimensionnement à l'intervalle [0,1] s'effectue en décalant les valeurs afin que la valeur minimale soit 0, puis en divisant par la nouvelle valeur maximale qui correspond à la différence entre les valeurs maximale et minimale d'origine. Cette méthode nous a permis d'avoir les résultats suivants :

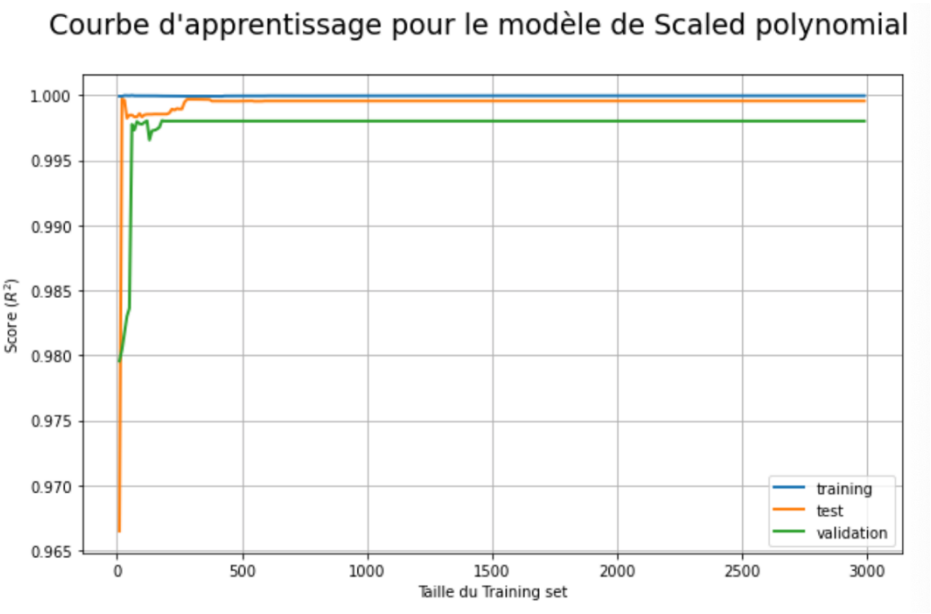
```
Scaled Poly Training set score: 1.000
Scaled Poly Test set score: 1.000
Scaled Poly Validation set score: 0.998
```

On obtient donc des résultats très satisfaisant, on peut assumer que ce modèle prédit bien la cible de notre ensemble de données qui est la prime. Cette remarque peut être appuyer grâce aux courbes suivantes qui modélisent l'évolution des valeurs prédites en fonction des valeurs réelles.



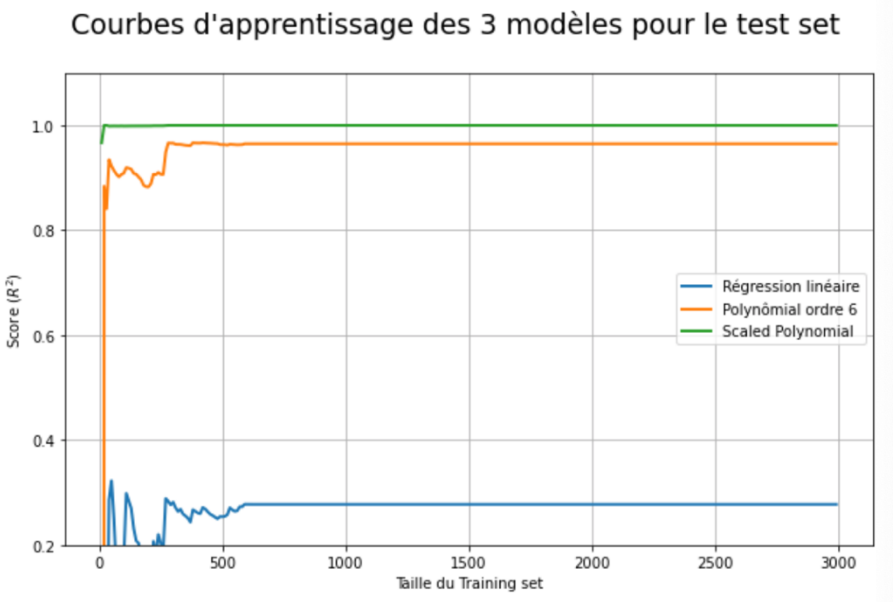
On peut voir que la courbe des valeurs prédites en fonction des valeurs réelles est quasi-confondue avec la fonction f(x)=x, ce qui prouve que l'erreur dans ce cas, entre les deux valeurs est très faible.

Pour ce qui est de la courbe d'apprentissage du modèle polynomial normalisé, on remarque que les résultats sont globalement excellents. En effet, on peut voir une courbe parfaite pour l'ensemble de training qui coïncide avec la fonction f(x)=1, ceci peut s'interpréter par le fait que notre normalisation se porte sur l'ensemble de training. D'autre part, on peut remarquer que pour les courbes de test et de validation, elles présentent des fluctuations pour les petites valeurs et que ces dernières se stabilisent rapidement pour atteignant un score de 0,998 pour la courbe de validation et presque 1 pour la courbe de set. Ces scores sont identiques aux résultats trouvés précédemment ce qui justifient bien nos résultats.



3) Comparaison des 3 modèles

Ici nous avons décidé de représenter les courbes d'apprentissage pour chaque modèle présenté précédemment en se concentrant sur l'ensemble de test. On peut constater que le meilleur score est pour le modèle polynomial normalisé, puis un peu moins le modèle polynomial de degré 6, et enfin le modèle de régression linéaire qui présente un score très bas.



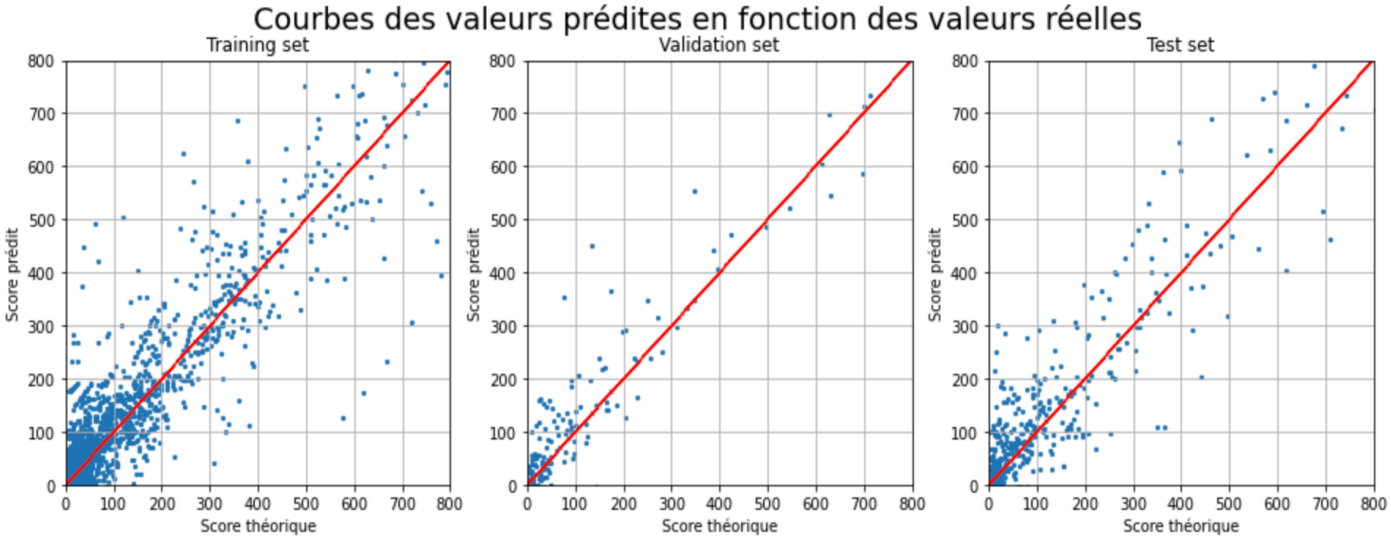
À partir de ce graphe, on peut soutenir que le modèle polynomial normalisé est le meilleur modèle approchant notre jeu de données. Cependant, il reste une étape importante à concevoir : la prédiction en cas réel. Cette étape est primordiale puisque c'est le but principal du projet, nous cherchons à avoir une interface indiquant une prime satisfaisante pour tout jeu de données indiqué par le client.

IV. Prédiction

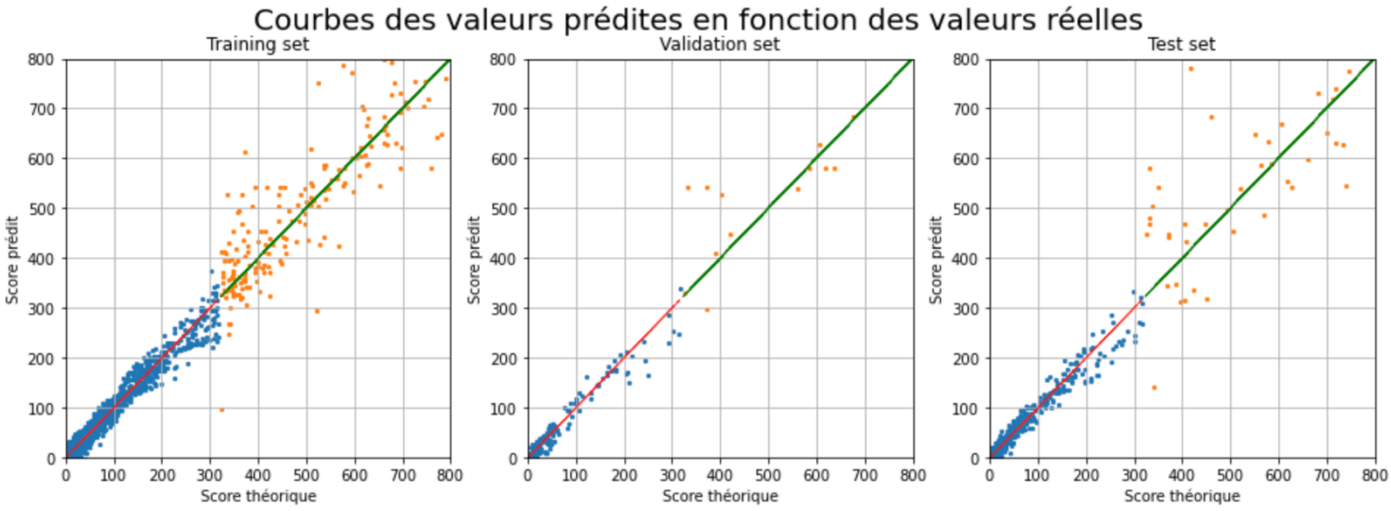
1) Modèle retenu

Suite à l'étude réalisée sur nos modèles d'entrainement, nous avons essayé naturellement de prédire nos primes à partir du modèle « Scaled Polynomial ». Nous avons donc créé une fonction *prediction_scaled_polynomial_regression(donnee)*, prenant comme paramètres les 5 variables nécessaires et renvoyant une prédiction de la prime associée grâce à la fonction *.predict* du modèle entrainé. Cependant, nous avons réalisé une série de tests de prédiction de prime et il s'est avéré que ce modèle n'était pas concluant.
En effet, bien que le modèle approximait presque parfaitement nos primes présentes dans la dataset, à l'instant où l'on insérait un jeu de paramètres non présent dans la base de données, le modèle « Scaled Polynomial » renvoyait une prime complétement aberrante tendant vers l'infini. Nous en avons conclu que ce modèle devait avoir d'énormes fluctuations entre les différentes « targets » afin d'approximer au mieux, au détriment de la prédiction. Nous avons donc fait le choix d'écarter ce modèle et de nous concentrer sur une prédiction à partir du modèle polynomial.

Le modèle polynomial présenté précédemment a donc été implémenté dans une fonction *prediction_polynomial_regression(donnee)*. Contrairement au « Scaled polynomial », cette prédiction permettait d'approximer des primes issues de sets de paramètres non présents dans la dataset, ce que nous recherchions à faire. Cependant, la précision de cette prédiction était assez faible et elle renvoyait bien souvent des primes négatives pour les « petites » primes. Les graphiques suivants rendent compte de la dispersion des prédictions par rapport aux valeurs targets pour ces petites valeurs :



Nous avons donc pensé à modifier un peu ce modèle polynomial en prenant en compte les analyses réalisées sur les données. En effet, nous avions remarqué qu'il y'avait des données aberrantes dans la dataset, correspondantes aux primes supérieures à 319€. Un unique modèle polynomial va essayer d'approximer ces « grandes valeurs » au détriment des plus petites primes. Il convient donc de diviser la dataset en 2 sous-data correspondant aux valeurs aberrantes et non-aberrantes et d'entrainer ces sous datas avec 2 modèles polynomiaux distincts. Il s'est avéré que les petites valeurs étaient bien mieux approximées par un modèle polynomial d'ordre 5 et les grandes valeurs par un modèle polynomial d'ordre 8 ce qui rend la dispersion et donc la précision de notre modèle nommée « double-polynomial » bien plus satisfaisante :



Il ne restait plus qu'à créer une fonction *prediction_polynomial_regression2(donnee)* permettant de renvoyer la prime associée aux données rentrées et calculée par l'un des 2 modèles.
Une difficulté était de choisir s'il fallait renvoyer la prime calculée par le premier ou bien par le deuxième modèle d'approximation polynomiale selon les paramètres insérés. Pour cela, la fonction *prediction_polynomial_regression2(donnee)* fait intervenir le modèle polynomial simple sur toute la dataset, puis nous comparons le résultat obtenu par cette prédiction assez approximative avec nos 2 modèles du « double-polynomial », pour renvoyer au final la prime la plus proche du « simple polynomial ».
Une série de test sur cette fonction s'est montrée concluante quant à nos attentes.

2) Interface d'utilisation

Afin d'avoir une estimation de la prime en fonction des paramètres saisies, notre projet devait comprendre une interface permettant à l'utilisateur de prédire sa prime annuelle après avoir rentré ses informations personnelles. Nous avons beaucoup réfléchi sur le sujet et nous sommes documentés pour réaliser cet aspect nouveau pour nous.

Une première idée a été de réaliser une interface graphique par le biais de la programmation Python. Cette idée a été abandonnée au vu du temps dont nous disposions et

de nos compétences dans ce domaine. Nous nous sommes alors dirigés vers une autre solution, l'utilisation du module « xlwings ».

En effet, notre interface utilisateur capable de prédire la prime du client en fonction de ses informations repose principalement sur l'utilisation de ce module qui nous permet d'importer une ou plusieurs fonction(s) python dans Microsoft Excel. Cet aspect, également nouveau, a nécessité de multiples tests ainsi qu'une documentation et des recherches approfondies sur le sujet.

Pour ce faire, il a tout d'abord été nécessaire de changer le format de notre code python. En effet, nous utilisions un fichier « .ipynb » via un notebook Anaconda que nous avons transformé en fichier « .py ». Cette transformation nous a permis d'avoir un fichier local que n'importe quel utilisateur peut ensuite utiliser (à condition d'avoir Python installé sur sa machine). Notre code a été remanié dans ce fichier afin d'avoir une seule fonction *prediction_polynomial_regression2(donnee)* générant la prime en fonction des paramètres saisies, en s'appuyant sur notre modèle « double-polynomial ». De plus, les tests, print et tracés de graphes ont été abandonnés dans ce fichier pour avoir une version plus claire et simplifiée.

Par la suite, nous avons tenté d'installer les modules que nous utilisions dans notre programme Python (« numpy », « mglearn », « pandas » …) et le module « xlwings » via le terminal Windows à l'aide la commande :

- pip install Module

Un premier problème rencontré a été que « pip n'est pas reconnu comme une commande externe ou interne ». Après différentes recherches, nous nous sommes rendu compte que cela venait du fait que « pip » n'était pas ajouté à la variable d'environnement PATH (voir l'User Guide pour savoir comment l'ajouter).
Une fois l'installation de ces modules réalisées, la commande : « xlwings addin install » dans le terminal Windows nous a permis de finir la configuration du module xlwings, avant de passer à l'étape suivante, la création et manipulation de notre interface.

Pour commencer, pour pouvoir utiliser la fonctionnalité recherchée de « xlwings », il était nécessaire d'enregistre notre fichier Excel sous le format « .xlsm ». Dans ces conditions, nous voyions un nouvel onglet apparaître sur Excel : « Xlwings ». Dans cet onglet, on trouve une fonctionnalité « Import Functions » permettant d'importer toutes les fonctions de notre programme python précédées des arobases suivantes :

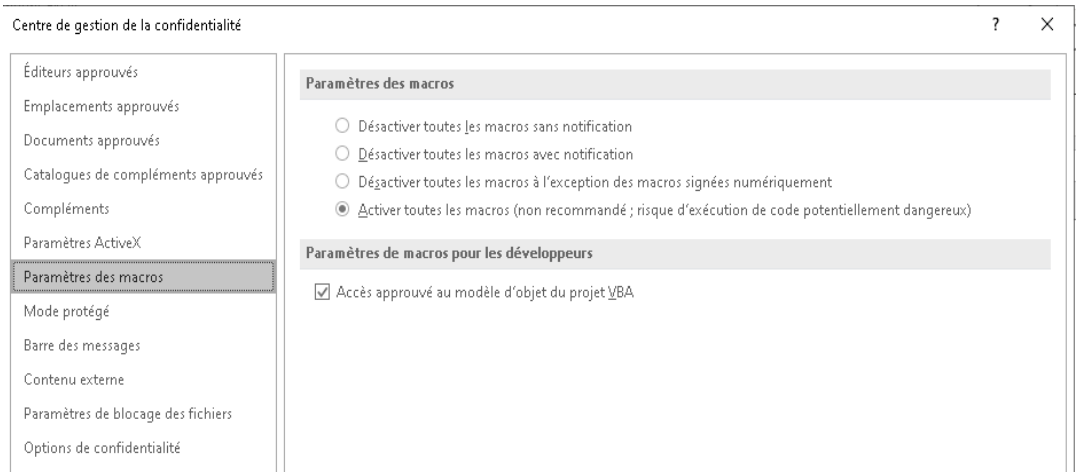
```python
@xw.func
@xw.arg('x', numbers=int)
@xw.arg('n', numbers=int)
@xw.arg('m', numbers=int)
@xw.arg('i', numbers=float)
@xw.arg('a', numbers=int)
def predict_prime(x,n,m,i,a):
    return prediction_polynomial_regression2([x,n,m,i,a])[0]


@xw.func
@xw.arg('x', numbers=int)
@xw.arg('n', numbers=int)
@xw.arg('m', numbers=int)
@xw.arg('i', numbers=float)
@xw.arg('a', numbers=int)
def accuracy_score_predicted_prime(x,n,m,i,a):
    return prediction_polynomial_regression2([x,n,m,i,a])[1]
```

(Avec import xlwings as xw)

En effet, ces arobases vont permettre la détection des fonctions liées par la suite sur Excel. On note qu'il est absolument nécessaire pour qu'il y ait détection du fichier .py dans Excel que le fichier Excel et le fichier python aient le même nom et se trouve dans le même dossier comme ci-dessous :

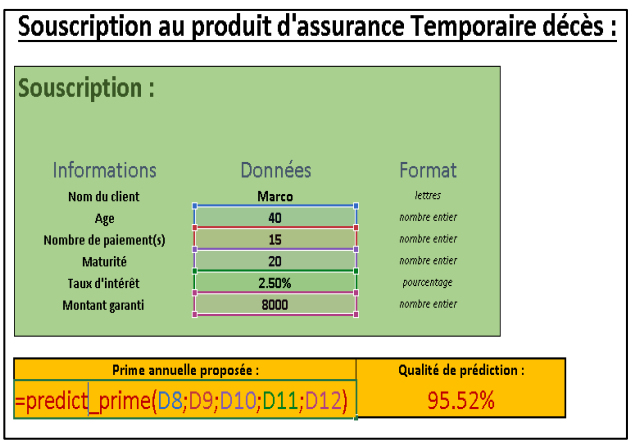
Nom	Modifié le	Type	Taille
database2.csv	10/01/2021 20:28	Fichier CSV Micro...	96 Ko
database2_end.csv	10/01/2021 20:27	Fichier CSV Micro...	15 Ko
dataset.csv	10/01/2021 20:27	Fichier CSV Micro...	116 Ko
projectIA.py	14/01/2021 05:17	Python File	6 Ko
projectIA.xlsm	14/01/2021 05:19	Feuille de calcul ...	503 Ko

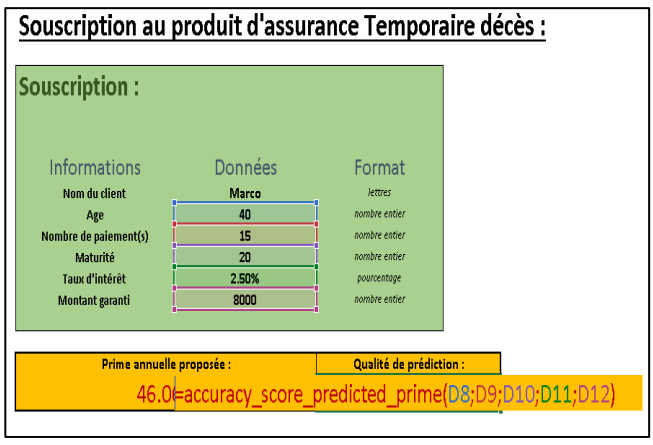
À ce stade, nous avons tenté d'importer nos fonctions dans excel mais nous nous sommes heurtés à une nouvelle erreur de type Run-time indiquant que la méthode VBProject échouait. Après de nouvelles recherches, nous avons pu régler cela en accédant paramètres du Centre de gestion de la confidentialité, puis aux paramètres des macros.



Il était donc nécessaire d'activer toutes les macros ainsi que de donner un accès au modèle d'objet du projet VBA.

Une fois ces manipulations réalisées, nous avons pu importer correctement nos fonctions, nous permettant de prédire une prime en fonction des paramètres rentrés par l'utilisateur et de donner la qualité de la prédiction associée, en appuyant sur « Import Functions » de l'onglet « Xlwings » :





Enfin, nous avons dû mettre en place des restrictions sur les saisies de données dans Excel. En effet, il était nécessaire que l'utilisateur ne puisse pas rentrer de données incorrectes comme par exemple une chaîne de caractères à la place d'un entier. De plus, il fallait respecter la condition que le nombre de paiements de primes annuelles soit inférieur à la maturité.

Par ailleurs, après avoir tester notre *Pricer,* nous avons remarqué quelques cas d'erreurs d'approximation, notamment lorsque les valeurs des paramètres étaient en dehors de l'intervalle de définition de notre dataset. C'est pour cela que chaque paramètre doit être compris dans un intervalle bien défini et dans le cas contraire, un message d'erreur apparait pour l'utilisateur.
Concernant le paramètre du montant garanti, il n'est possible de sélectionner que certaines valeurs car lors de notre phase de test, seuls ces montants affichaient des résultats satisfaisants.

V. Conclusion

Ce projet a été pour nous l'occasion d'utiliser un outil de plus en plus utilisé dans le monde de l'assurance : l'intelligence artificielle. À partir d'une base de données et d'une cible à approximer, nous avons vu qu'il était possible de générer des modèles d'approximation capables de donner un poids à chaque paramètre pour ensuite prédire de nouveaux éléments plus ou moins précis.
Cette précision dépend effectivement du modèle mais aussi de la qualité de notre échantillon, qui doit être bien étudié pour pouvoir détecter des informations primordiales comme la dispersion des targets, les éventuelles valeurs aberrantes ou encore l'impact des paramètres sur le résultat. Ces informations peuvent être utilisées pour générer de nouveaux modèles de prédictions plus fiables comme par exemple notre « double-polynomial ».
En plus de cela nous avons vu qu'il était possible de relier un code source python à une interface comme Excel, bien utile pour permettre à l'utilisateur d'utiliser notre fonction de prédiction de manière très intuitive.

En ce contexte sanitaire, nous avons utilisé des outils comme *google collab* pour travailler en groupe sur les différents fichiers de manière efficace et en distanciel. Malheureusement il nous est impossible de configurer nous-même l'installation de l'interface sur l'ordinateur du futur utilisateur et nous avons donc essayé de concevoir un User Guide très détaillé, livré dans le dossier de ce projet. Nous proposerons tout de même notre aide via un lien zoom si les manipulations venaient à échouer.

VI. Références

- https://docs.microsoft.com/fr-fr/dotnet/machine-learning/how-to-guides/train-machine-learning-model-ml-nethttps://fr.wikipedia.org/wiki/Régression_linéaire
- https://medium.com/blogaboutgoodscompany/guide-simple-des-bases-de-lintelligence-artificielle-bcd3d077633d
- https://ledatascientist.com/creer-un-modele-de-regression-lineaire-avec-python/
- https://fr.wikipedia.org/wiki/Régression_polynomiale
- https://mrmint.fr/regression-polynomiale
- https://fr.wikipedia.org/wiki/Régression_polynomiale
- https://mrmint.fr/regression-polynomiale
- https://haltode.fr/algo/ia/apprentissage_artificiel/regression_lin_poly.html
- https://docs.microsoft.com/fr-fr/azure/machine-learning/algorithm-module-reference/normalize-data
- https://harvard.service-now.com/ithelp?id=kb_article&sys_id=10765ac0db13ebc017d359b2ca961914
- https://docs.xlwings.org/en/stable/installation.html