

Le modèle logistique réalisé

Les variables qu'il intègre

Le traitement et le nettoyage des données

Evaluation et évolution du modèle

1. Cross-Validation
2. ROC
3. Les hyperparamètres
4. RFE
5. Le scaling
6. Et la piste des bêtas dans tout ça ?
7. Optimisation des variables

Présentation des scores et des prédictions

Le modèle logistique réalisé

Pour ce brief, nous avons réalisé une régression logistique. Le travail s'est fait en différentes étapes. Nous avons nécessairement commencé par un MVP (Minimum Viable Project) pour ensuite le faire évoluer jusqu'à un rendu final.

Cette régression logistique s'est faite dans un premier temps avec très peu d'hyperparamètres, le but étant d'afficher uniquement un premier modèle. Il a fallu intégrer un seul paramètre dans notre modèle qui est `solver='liblinear'`, celui ayant servi notamment au niveau de nos variables.

```
[25] ▶ my_model = LogisticRegression(solver='liblinear')  
[26] ▶ my_model.fit(X_train, y_train)  
LogisticRegression(solver='liblinear')
```

Nous avons décidé ici de mesurer l'accuracy ainsi que la F-mesure. Nos premiers résultats étaient donc les suivants :

Description	Modifier
LogisticRegression	
accuracy = 0.85	
F measure = 0.643	
Predictions:	
employés "partant" : 23	
employés "restant" : 347	

Sans aucun traitement de données ou de sélection de nos paramètres, les résultats sont ici minimales. Nous avons tenté de les parfaire à travers différentes étapes. Nous allons vous parler dans un second temps du choix de nos variables.

Les variables qu'il intègre

Au final, nous intégrons dans notre modèle un total de 27 variables.

Le traitement et le nettoyage des données

Pour nos variables, nous en parlerons dans cette partie et dans la partie suivante. D'abord, il nous a fallu réfléchir aux variables qui n'étaient pas importantes dans la construction de notre modèle. Au-delà du nettoyage de données, certaines variables sont tout simplement inutiles à l'analyse de notre modèle. Par exemple, EmployeeCount ou EmployeeNumber ne nous semblaient pas pertinentes dans le cadre de notre modèle. D'autres variables, qui n'avaient par exemple, qu'une seule valeur possible, étaient contre-indiquées dans le modèle également.

Il fallait aussi réfléchir à comment traiter les variables numériques ainsi que les variables catégorielles. Pour la dernière catégorie, nous avons pensé dans un premier temps à faire du One-Hot Encoding, ce qui a été très utile pour avancer dans la construction de notre modèle.

```
Encodage de nos variables catégorielles

[12] ▶ ML
cat_vars = ['BusinessTravel', 'Department', 'EducationField', 'JobRole', 'MaritalStatus']

for var in cat_vars:
    cat_list='var'+ '_' +var
    cat_list = pd.get_dummies(attrition_data[var], prefix=var)
    data1=attrition_data.join(cat_list)
    attrition_data=data1

for var in cat_vars:
    cat_list2='var'+ '_' +var
    cat_list2 = pd.get_dummies(attrition_test[var], prefix=var)
    data2=attrition_test.join(cat_list2)
    attrition_test=data2
```

On s'est ensuite rendu compte qu'on pouvait améliorer cela en transformant certaines de ces variables catégorielles par le Label Encoding.

```
[10] ▶ ML
attrition_data["OverTime"] = attrition_data["OverTime"].map(dict(Yes=1, No=0)).astype('int64')

[11] ▶ ML
attrition_data["Gender"] = attrition_data["Gender"].map(dict(Male=1, Female=0)).astype('int64')
()
```

Pour nos variables numériques, nous n'avons touché à rien d'autre.

Concernant le traitement de nos données, il n'y a rien eu de particulier car notre dataset était particulièrement propre. Les données manquantes n'étaient pas présentes, les données aberrantes ne l'étaient pas plus. De ce fait, il n'y a pas eu de traitement particulier à faire pour ces éléments.

Evaluation et évolution du modèle

Notre modèle a évolué à travers l'utilisation de nombreux outils afin d'attester de son amélioration. A la suite du MVP, nous nous sommes lancés dans l'utilisation de la Cross-Validation.

1. Cross-Validation

Pour cette partie, nous avons repris notre Logistic Regression et l'avons adapté en utilisant la Cross-Validation.

[46] ▶ ML

```
my_model = LogisticRegressionCV(cv=5, random_state=1, solver='liblinear')
```

Ici, notre valeur de CV est de 5 afin de séparer notre jeu de données en 5 et en le réitérant 5 fois. Le random-state est établi à 1 afin de ne pas relancer un tri aléatoire de notre jeu de données et, enfin, notre solver reste égal à 'liblinear' comme pour notre MVP.

Nous obtenons donc les résultats suivants :

☰ **Description** Modifier

LogisticRegressionCV(cv=5, random_state=1)

accuracy = 0.854545

F measure = 0.6822

Predictions:

employés "partant" : 19

employés "restant" : 351

On constate donc que par rapport à notre MVP, les résultats sont meilleurs, notamment pour l'accuracy et la F-mesure. Concernant les prédictions, ils restent assez modestes.

2. ROC

La courbe ROC va nous permettre de représenter les performances du modèle de classification pour tous les seuils de classification. La courbe représente le taux de vrais positifs par rapport au taux de faux positifs. En code, voici la construction de la courbe :

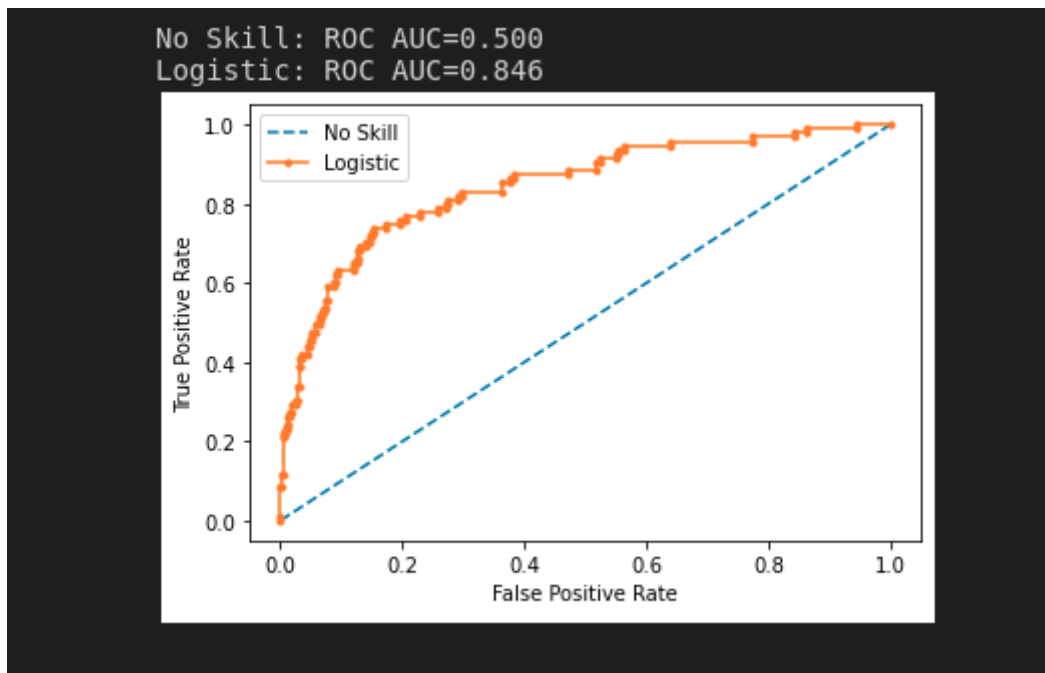
ROC

[33] ▶ ML

```
from sklearn.metrics import roc_curve
from sklearn.metrics import roc_auc_score

# split into train/test sets
trainX, testX, trainy, testy = train_test_split(X, y, test_size=0.5, random_state=2)
# generate a no skill prediction (majority class)
ns_probs = [0 for _ in range(len(testy))]
# fit a model
my_model.fit(trainX, trainy)
# predict probabilities
lr_probs = my_model.predict_proba(testX)
# keep probabilities for the positive outcome only
lr_probs = lr_probs[:, 1]
# calculate scores
ns_auc = roc_auc_score(testy, ns_probs)
lr_auc = roc_auc_score(testy, lr_probs)
# summarize scores
print('No Skill: ROC AUC=%.3f' % (ns_auc))
print('Logistic: ROC AUC=%.3f' % (lr_auc))
# calculate roc curves
ns_fpr, ns_tpr, _ = roc_curve(testy, ns_probs)
lr_fpr, lr_tpr, _ = roc_curve(testy, lr_probs)
# plot the roc curve for the model
plt.plot(ns_fpr, ns_tpr, linestyle='--', label='No Skill')
plt.plot(lr_fpr, lr_tpr, marker='.', label='Logistic')
# axis labels
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
# show the legend
plt.legend()
# show the plot
plt.show()
```

Et voici le résultat :



L'AUC (Aire sous la courbe) obtenue est de 0.846. Cette métrique supplémentaire peut donc nous aider à avoir un autre regard sur notre modèle.

3. Les hyperparamètres

En analysant un peu plus précisément la Logistic Regression, on a pu remarquer que certaines paramètres pouvaient être intéressants à utiliser. Le premier d'entre eux est le C, qui correspond à l'inverse de la régularisation, la régularisation étant une sorte de valeur de pénalité qui permet d'augmenter l'amplitude des valeurs des variables afin de réduire le surapprentissage. Après quelques tests, nous nous sommes rendus compte que la valeur la plus intéressante était 10. Un deuxième hyperparamètre nous semblait intéressant à utiliser : le penalty. Celui-ci permet de spécifier une norme dans la régularisation. Après test, la valeur 'l2' était la plus pertinente. En combinant ces hyperparamètres, voici nos résultats :

Quand C = 10, et penalty='l2', l'Accuracy et le F-mesure sont les plus intéressants :

- Accuracy = 0.868182
- F-mesure = 0.724037
- AUC = 0.846

4. RFE

Le RFE est une technique permettant d'indiquer la nature des variables les plus pertinentes à utiliser dans notre modèle. C'est donc un outil très utile afin de mettre en avant les variables à conserver pour travailler notre modèle. Là où le traitement et le nettoyage des données ne nous permettaient pas d'aller plus loin, le RFE pouvait donc nous indiquer les variables à utiliser en fonction des performances à obtenir.

[41] ▶  ML

```
from sklearn.feature_selection import RFECV
from sklearn.model_selection import StratifiedKFold
```

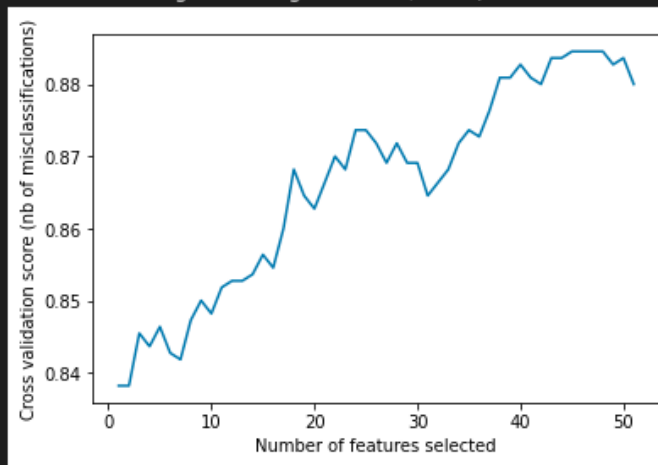
[42] ▶  ML

```
# Create the RFE object and compute a cross-validated score.
svc=LogisticRegression(C = 10, penalty = 'l2',solver='liblinear')

rfecv = RFECV(estimator=svc, step=1, cv=5)
X_RFE = rfecv.fit_transform(X, y)

print("Optimal number of features in X_RFE : %d" % rfecv.n_features_)
print("estimator",rfecv.estimator_)
# Plot number of features VS. cross-validation scores
plt.figure()
plt.xlabel("Number of features selected")
plt.ylabel("Cross validation score (nb of misclassifications)")
plt.plot(range(1, len(rfecv.grid_scores_) + 1), rfecv.grid_scores_)
plt.show()
```

Optimal number of features in X RFE : 45
estimator LogisticRegression(C=10, solver='liblinear')



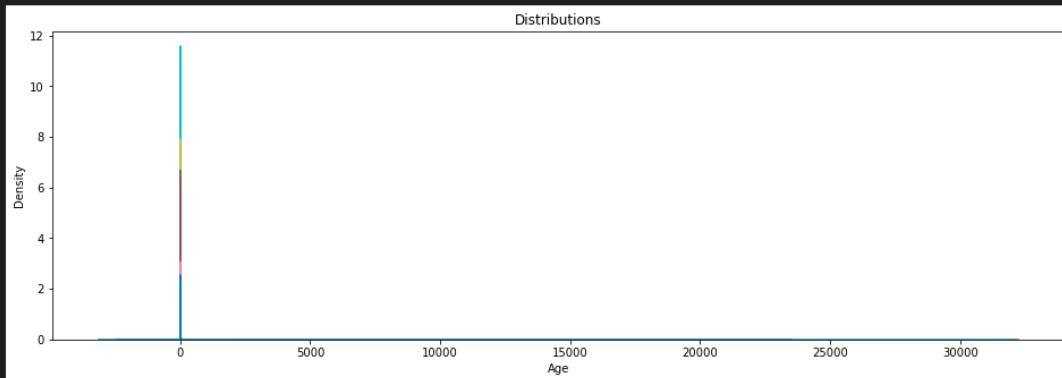
Pour ce premier RFE, nous avons obtenu un nombre optimal de variables à utiliser de 45, ce qui est encore assez conséquent. On doit donc encore travailler sur certains paramètres mais aussi sur d'autres moyens d'harmoniser nos variables.

5. Le scaling

Le scaling avait pour objectif d'harmoniser nos variables, de les normaliser à la même échelle afin de pouvoir les comparer correctement. On a donc récupéré toutes nos variables afin de procéder à ce travail de normalisation. La capture ci-dessous nous montre les variables avant normalisation.

scaling

```
[15] > plt.subplots(ncols=1, figsize=(16, 5))
a.set_title("Distributions")
for col in attrition_data_numbers.columns:
    sns.kdeplot(attrition_data_numbers[col], ax=a)
plt.show()
```



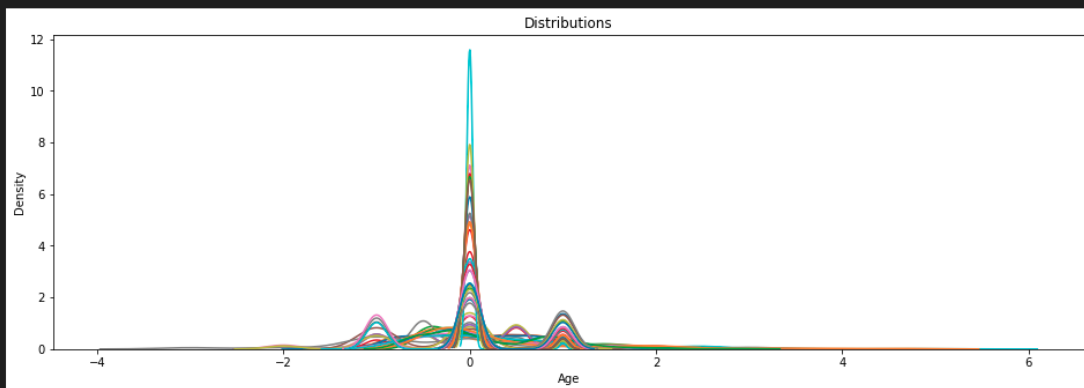
La capture ci-dessous nous montre les variables après normalisation :

```
[16] > from sklearn.preprocessing import StandardScaler, MaxAbsScaler, MinMaxScaler, RobustScaler

# scaler = MaxAbsScaler()
# scaler = MinMaxScaler()
# scaler = StandardScaler()
scaler = RobustScaler()

scaled_df = scaler.fit_transform(attrition_data_numbers)
scaled_df = pd.DataFrame(scaled_df, columns=columns_list)

fig, a = plt.subplots(ncols=1, figsize=(16, 5))
a.set_title("Distributions")
for col in scaled_df.columns:
    sns.kdeplot(scaled_df[col], ax=a)
plt.show()
```



On s'est aussi rendu compte que l'on pouvait scaler selon une technique précise. Elles sont au nombre de 4 : MaxAbsScaler, MinMaxScaler, StandardScaler et RobustScaler. Après tests, RobustScaler semblait ressortir les résultats les plus intéressants. Voici les différents résultats obtenus :

	accuracy	f measure	auc	moy cv	nb feature
robustscaler	0.8681	0.724	0.837	0.817	29
standard scaler	0.8681	0.724	0.833	0.869	22
minmaxscaler	0.8636	0.710	0.843	0.873	23
MAXABS	0.8636	0.710	0.845	0.873	25

6. Et la piste des bêtas dans tout ça ?

Dans nos étapes de travail, nous avons pensé à travailler autour des différents bêtas de notre Logistic Regression. Finalement, il en est ressorti que leur intérêt était plutôt limité.

7. Optimisation des variables

En réutilisant tout ce qui a été fait dans les étapes précédentes et en retravaillant certaines variables, notamment en utilisant le Label Encoding sur Overtime ainsi que sur Gender, nous avons réussi à obtenir les meilleurs résultats possibles. En effectuant tous nos calculs, on parvenait aux résultats suivants :

```
Mise en place de notre modèle de régression logistique

[21] > ML
      y = attrition_data['Attrition']
      X = attrition_data_numbers
      X_pred = attrition_test_numbers
      X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, random_state = 0)

[22] > ML
      my_model = LogisticRegressionCV(Cs = 10, cv=5, penalty = 'l2', solver='liblinear')

[23] > ML
      my_model.fit(X_train, y_train)

      LogisticRegressionCV(cv=5, solver='liblinear')

[24] > ML
      prediction = my_model.predict(X_test)

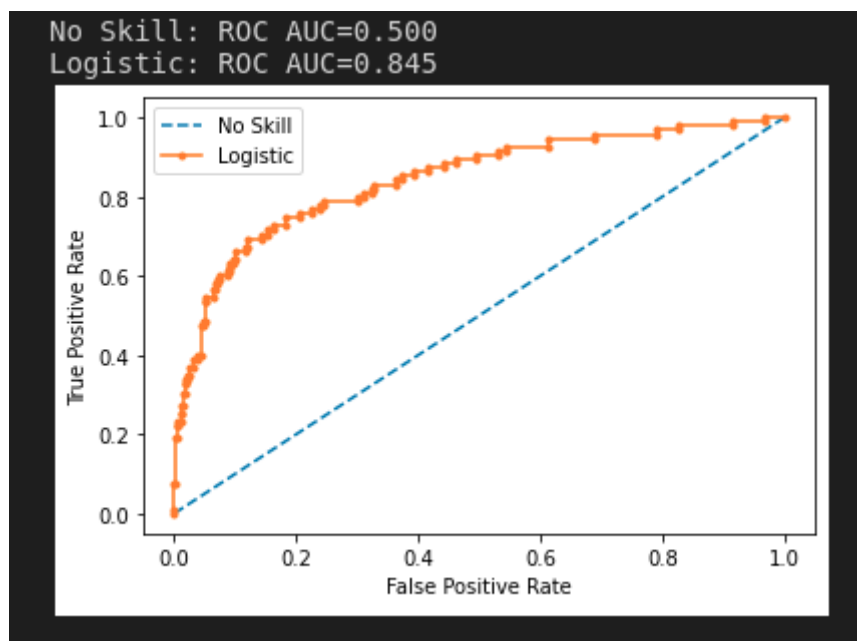
[25] > ML
      score = my_model.score(X_test, y_test)
      print('Test Accuracy Score :', score)

      Test Accuracy Score : 0.8681818181818182

[26] > ML
      print('F-mesure : ', f1_score(y_test, prediction, average="macro"))

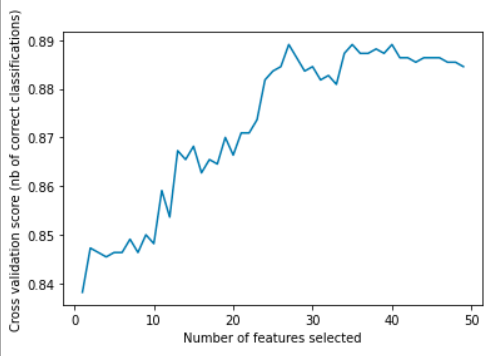
      F-mesure : 0.724036506769324
```

Accuracy = 0.8681818 ; F-mesure = 0.7240365



AUC = 0.845

```
[32] ▶ M4
# Plot number of features VS. cross-validation scores
plt.figure()
plt.xlabel("Number of features selected")
plt.ylabel("Cross validation score (nb of correct classifications)")
plt.plot(range(1, len(rfecv.grid_scores_) + 1), rfecv.grid_scores_)
plt.show()
```



```
[33] ▶ M4
print("The fitted estimator used to select features: ",rfecv.estimator_)
print("The mean cross-validation score: ", rfecv.grid_scores_.mean())

The fitted estimator used to select features: LogisticRegressionCV(cv=5, solver='liblinear')
The mean cross-validation score: 0.872430426716141
```

```
[34] ▶ M4
print("Optimal number of features in X_RFE : %d" % rfecv.n_features_)
print("The importances of features: ",rfecv.estimator_.coef_)

Optimal number of features in X_RFE : 27
The importances of features: [[ 0.40557309 -0.81870317  0.42718246 -0.55361224 -0.63279478 -0.44510365
  0.37663241  1.63451897 -0.52751229 -0.25424346 -0.4317952  -0.23128241
 -0.44313604  0.49541692 -0.3269616  -0.80171353  0.7123635  -0.55254931
 -0.58774649 -0.649821  -0.47157271 -0.57132972 -0.40008323 -0.6185624
  0.76176484 -0.76297001 -0.63061459]]
```

Un nombre optimal de variables à utiliser dans le RFE qui est de 27.

Présentation des scores et des prédictions

Pour notre jeu de données test, nous avons pu établir différents scores et prédictions. Voici nos résultats :

```
[25] ▶ M4
df_final.Prediction.value_counts()

Yes      282
No        88
Name: Prediction, dtype: int64
```

282 entrées obtiennent la prédiction 'Yes' tandis que 88 entrées obtiennent la prédiction 'No'. Pour le score d'attrition, il se mesure de 0 à 1. Si l'employé obtient un score d'attrition supérieur ou égal à 0.5, cela signifie qu'il obtiendra la prédiction 'Yes'. A l'inverse, il obtiendra la prédiction 'No'.

[26] ▶ M4

```
df_final.sort_values(by='AttritionScore').head(62)
```

	EmployeeNumber	Prediction	AttritionScore
44	727	No	0.116944
114	73	No	0.224878
299	1422	No	0.245539
227	538	No	0.250677
47	742	No	0.251514
...
297	945	No	0.425028
192	1002	No	0.426546
287	1467	No	0.431294
176	1108	No	0.436550
321	96	No	0.437613

62 rows × 3 columns

[28] ▶ M4

```
df_final.sort_values(by='AttritionScore').tail(62)
```

	EmployeeNumber	Prediction	AttritionScore
26	447	Yes	0.879308
7	407	Yes	0.879475
33	1507	Yes	0.882668
178	1770	Yes	0.883144
139	605	Yes	0.883193
...
353	812	Yes	0.972039
101	235	Yes	0.974434
331	1340	Yes	0.975318
179	984	Yes	0.982675
262	1568	Yes	0.985778

62 rows × 3 columns

Pour le nombre de personnes participant au programme, nous avons décidé de proposer ainsi :

Définir la taille du programme d'accompagnement

[29] ▶ ML

```
# le programme doit concerner entre 10 et 100 personnes
df_yes = df_final[df_final['Prediction']== 'Yes']
L = len(df_yes)

if L >100:
    nb_participants = 100
elif L < 10:
    print("Il y n'aura aucun participant à ce programme")
else:
    nb_participants = L

print("Il y aura {} participants à ce programme".format(nb_participants))
```

Il y aura 100 participants à ce programme

Comme nous avons obtenu 282 'Yes', nous récupérerons les 100 personnes ayant eu le plus haut score d'attrition.

Définir la liste des participants au programme

[30] ▶ ML

```
df_final = df_final.sort_values(by = 'AttritionScore', ascending = False)

df_participants = df_final.head(nb_participants)

print("les participants à ce programme sont les employés numéro : ",list(df_participants['EmployeeNumber']))
```

les participants à ce programme sont les employés numéro : [1568, 984, 1340, 235, 812, 1543, 1298, 1387, 355, 1154, 332, 14, 1013, 1866, 1198, 1754, 1419, 228, 12, 194, 1483, 1987, 398, 581, 720, 861, 1768, 380, 169, 2851, 1164, 1641, 238, 1592, 239, 883, 1628, 875, 716, 118, 366, 527, 1544, 1346, 1728, 1283, 1487, 128, 1732, 16, 359, 1264, 1856, 1613, 175, 969, 1758, 605, 1770, 1987, 487, 447, 1501, 887, 926, 1577, 398, 1843, 1727, 1029, 171, 717, 1859, 1286, 1472, 1712, 351, 547, 1583, 837, 1778, 5, 1678, 699, 556, 829, 1279, 1131, 881, 441, 1550, 80, 1273, 1698, 388, 1449, 254, 616, 64, 1821]