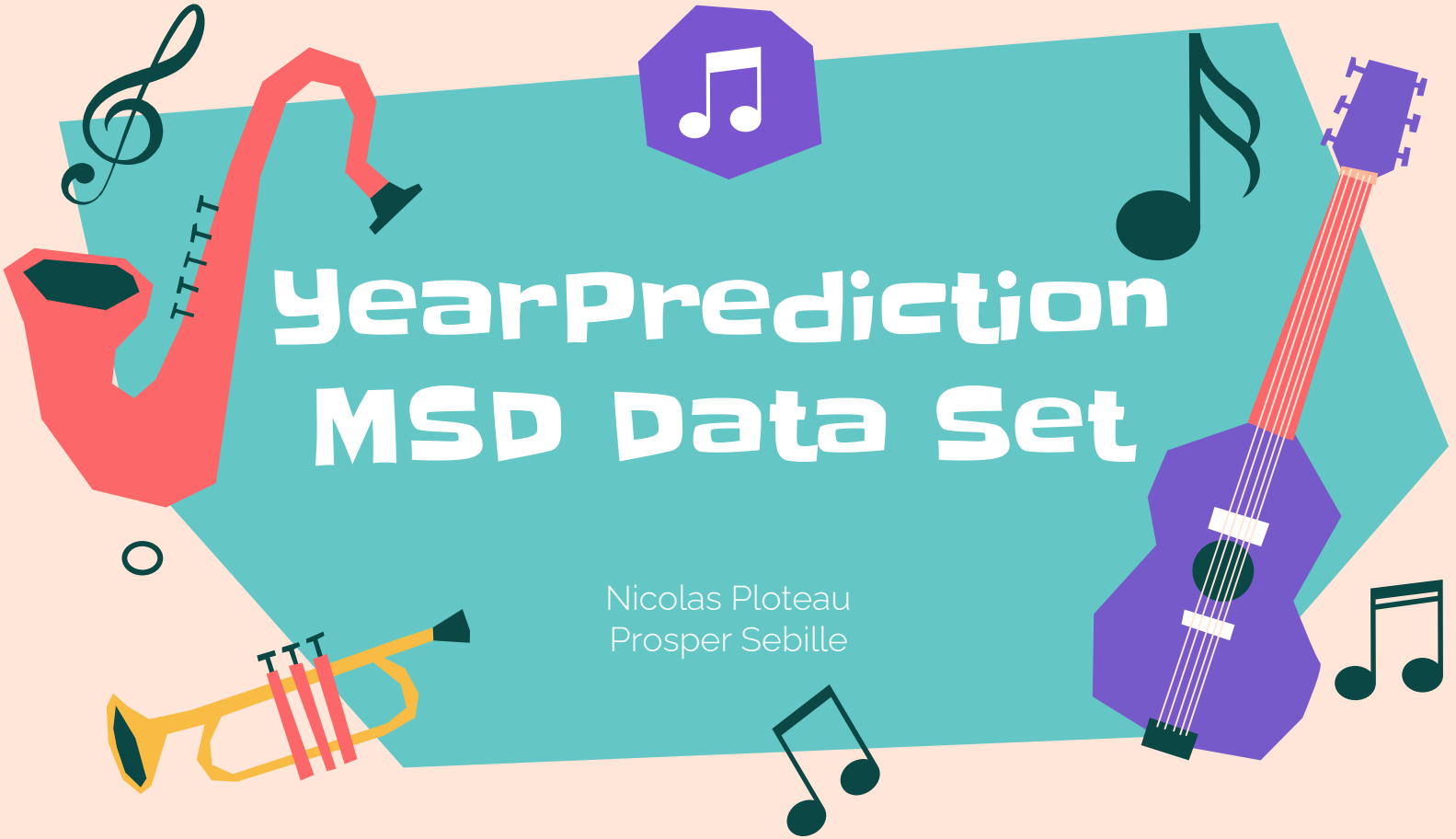

YearPrediction
MSD Data Set
Nicolas Ploteau
Prosper Sebille

SCHEDULE

Reflexions

Data visualisation

Analyse des approches

Data engineering

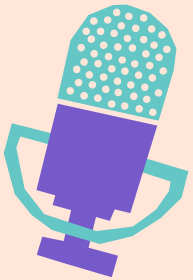
Modification / creation de variables

Modelisation

Test de differents modèles / hyperparamètres

API

Transformation en API Django



01

Reflexion

Visualisation / exploration des variables

Choix de l'approche algorithmique

Dataset

Targets

- Année de sortie du morceau
- Comprise entre 1922 et 2011
- Distribution déséquilibrée

Features

- 90 variables
  - 12 timbres moyens
  - 78 covariances

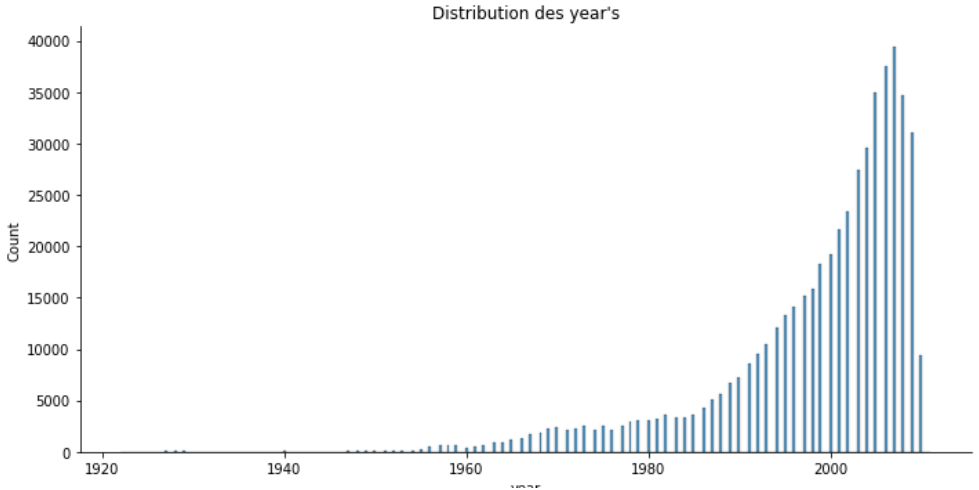
Target

Distribution :

- Entre 1922 et 2011
- Déséquilibrée

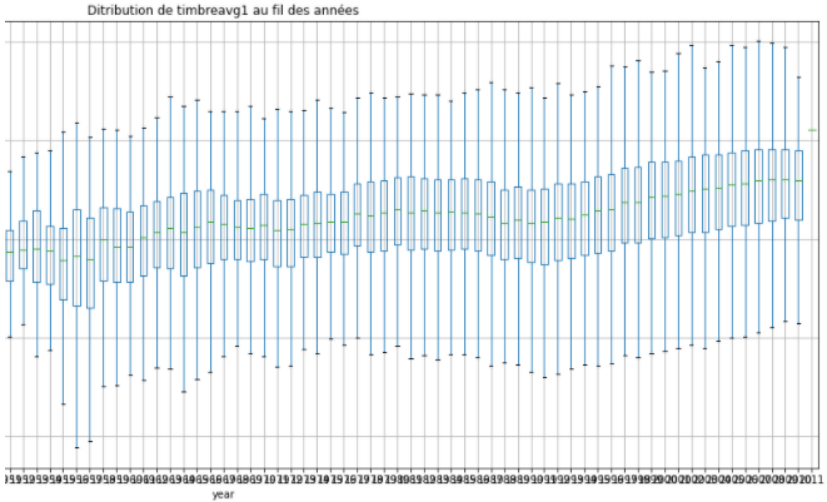
Problématiques :

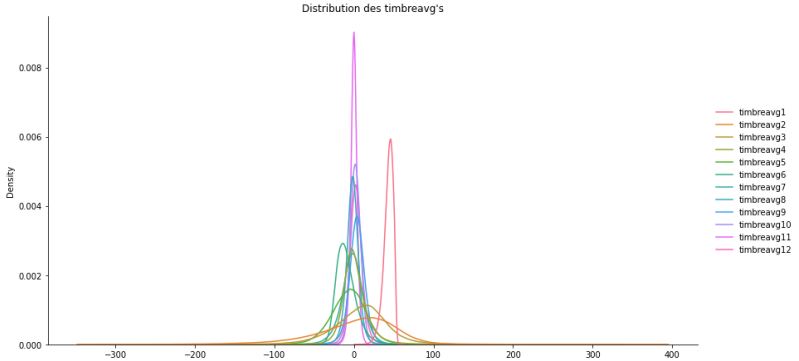
- Choix / biais des métriques
- Stratification des subsets
- Apprentissages biaisés



Features : timbreAvg

- Distribution normale
- Variation dans la distribution au fils des années





On suppose que les timbreAvg sont corrélés a l'année de sortie

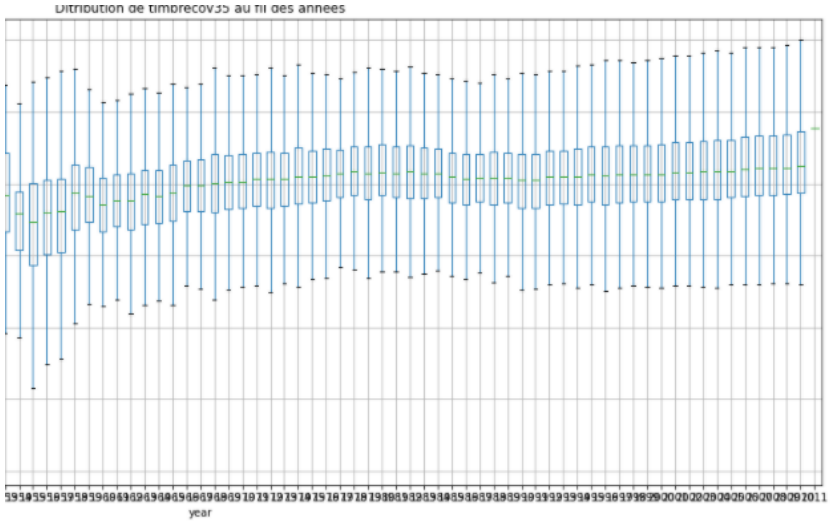
Features : timbreCov

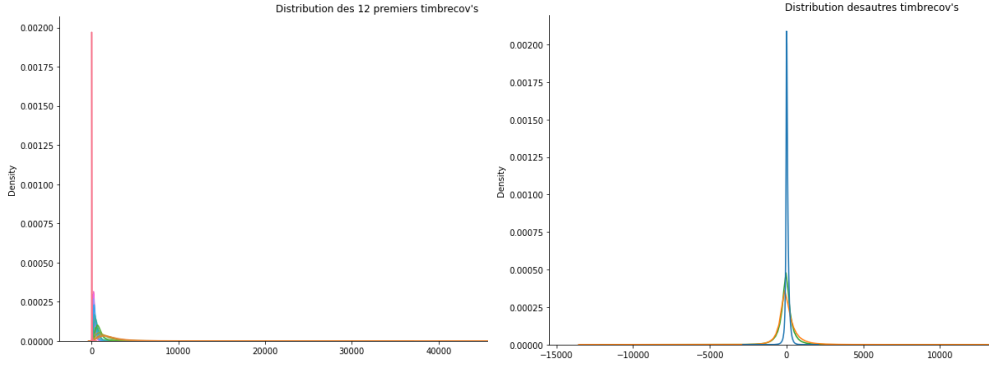
12 premiers timbreCov:

- Distribution log normale

66 derniers timbreCov

- Distribution normale





Variation dans la distribution au fils des années

On suppose que les timbreCov sont corrélés a l'année de sortie

Approche

Classification:

- Mode musicales ponctuelles
- Evolution non-linéaires
- Mode musicales cyclique

Buckets:

- Limiter le nombre de classe
- Regrouper les années similaires
- Limiter les erreurs

02

Data engineering

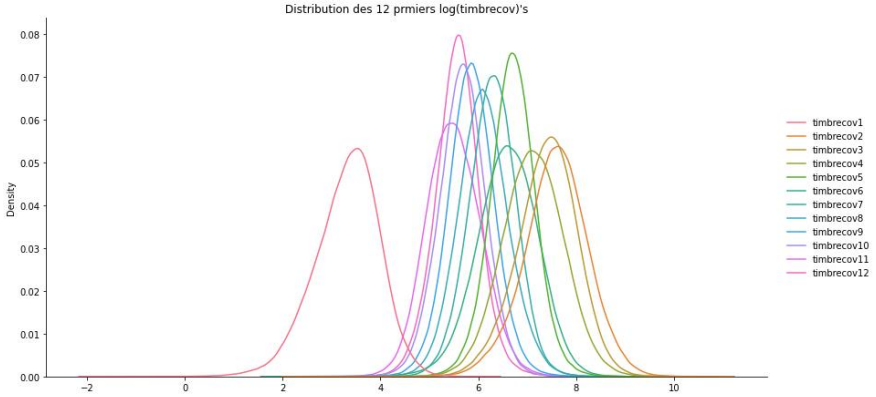
Choix des variables

Modification des varibles

Features : timbreAvg

On applique Log aux 12 premiers timbreAvg

	F_values	p_values
timbreavg1	383.952390	0.000000e+00
timbreavg6	237.508009	0.000000e+00
timbreavg3	141.173344	0.000000e+00
timbreavg7	133.061529	0.000000e+00
timbreavg2	118.344814	0.000000e+00
timbreavg12	73.031418	0.000000e+00
timbreavg10	52.384384	0.000000e+00
timbreavg11	38.958073	0.000000e+00
timbreavg8	31.326287	0.000000e+00
timbreavg5	23.533301	0.000000e+00
timbreavg4	22.917976	0.000000e+00
timbreavg9	20.588759	3.898771e-319



On applique un test anova

- Les p_values sont assez faibles
- Les F_values sont assez élevées

On garde tout les TimbreAvg's

Features : timbreCov

	F_values	p_values
timbrecov77	5.612095	9.465145e-58
timbrecov68	5.676637	9.042609e-59
timbrecov52	6.194570	4.905155e-67
timbrecov43	7.494965	2.577471e-88
timbrecov64	7.868467	1.539019e-94
timbrecov75	8.628110	2.540964e-107
timbrecov30	9.646303	1.129363e-124
timbrecov31	10.093860	2.281647e-132
timbrecov14	10.732460	2.078560e-143
timbrecov63	11.393560	6.611820e-155

On applique un test anova

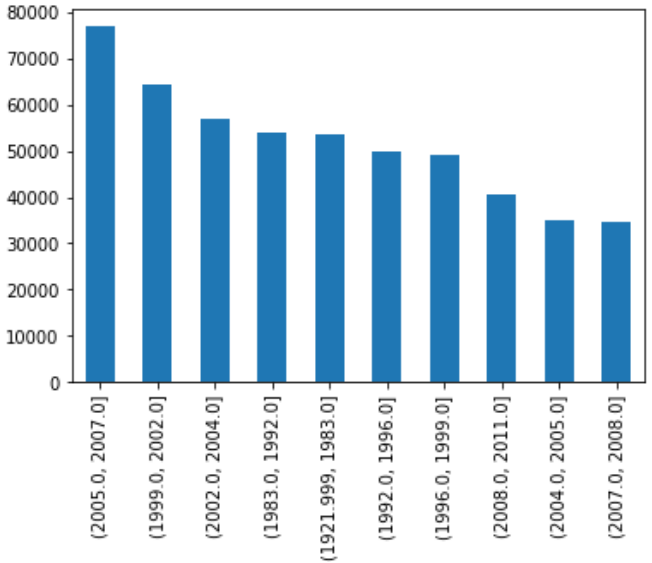
- Les p_values sont assez faibles
- Les F_values sont assez élevées

On garde tout les TimbreCov's

Target

Division en 10 buckets utilisant les quantiles

- Classes équilibrées



1	(1921.999,1983]
2	(1983, 1992]
3	(1992, 1996]
4	(1996, 1999]
5	(1999, 2002]
6	(2002, 2004]
7	(2004, 2005]
8	(2005, 2007]
9	(2007, 2008]
10	(2008, 2011]

03

Modelisation

Choix des algoritmes

Choix des hyperparamètres

Algorithmes

01

Random forest

Arbres de decisions
Bagging

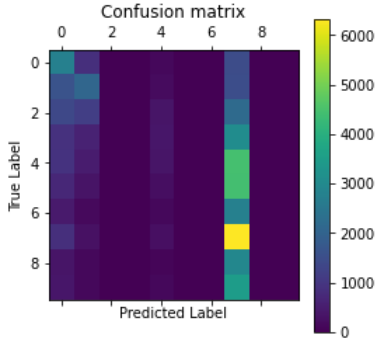
02

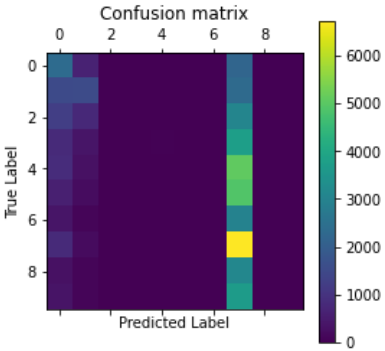
KNN

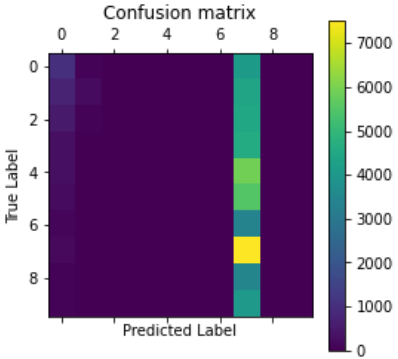
Moyenne des K plus proches voisins

Random forest

	N estimator	Min samples leaf	Max depth	Max features	criterion	Weighted f1
rf1	100	1000	None	auto	gini	0,13
rf2	200	1	5	auto	gini	0,11
rf3	200	50000	auto	entropy	gini	0,07



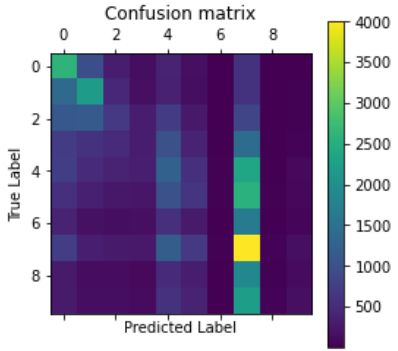




Grid search Random forest

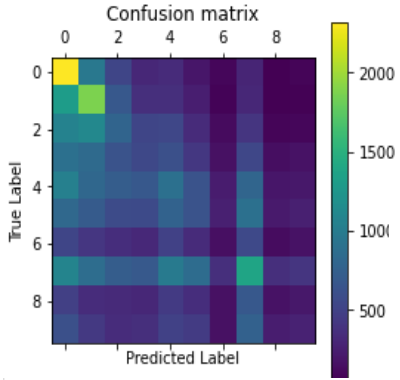
N estimator	[10, 100, 200]
Min samples leaf	[1, 5000, 50000]
Max depth	["gini", "entropy"]
Max features	["auto", "log2"]
criterion	[None,10]

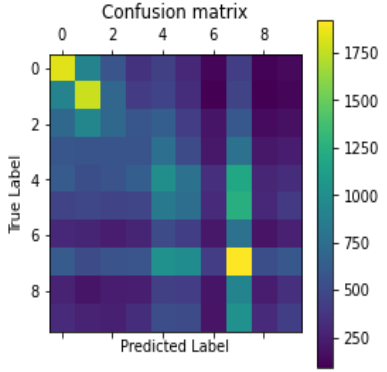
	precision	recall	f1-score	support
0	0.29	0.50	0.37	5228
1	0.34	0.41	0.37	5378
2	0.20	0.13	0.16	5114
3	0.15	0.07	0.09	4992
4	0.17	0.20	0.18	6266
5	0.16	0.11	0.13	5735
6	0.08	0.00	0.01	3498
7	0.22	0.52	0.31	7739
8	0.16	0.01	0.02	3516
9	0.21	0.04	0.07	4164
accuracy			0.23	51630
macro avg	0.20	0.20	0.17	51630
weighted avg	0.20	0.23	0.19	51630

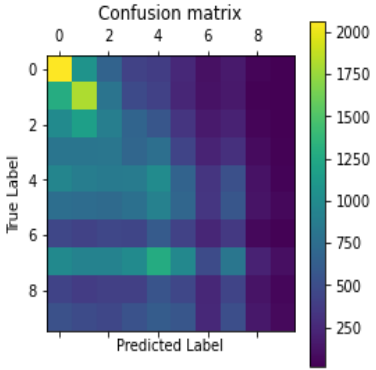


KNN

	N neighbors	weights	algorithm	Weighted f1
knn1	5	uniform	auto	0,16
knn2	2	uniform	Ball_tre	0,15
knn3	8	distance	auto	0,18







Grid search KNN

N neighbors	[2, 3, 5, 8 , 10]
weights	['uniform', 'distance']
algorithm	['auto', 'ball_tree', 'kd_tree', 'brute']

04

API

Django

Modèle

- Création d'un modèle Song avec 12 TIMBREAVG 78 TIMBRECOV un champ auto incrémenté pour le numéro de la musique.
- Ayant décidé de prédire des intervalles on créer deux champs YEAR_MAX/MIN pour modéliser l'interval.

```
created = models.DateTimeField(auto_now_add=True)

class Meta:
    ordering = ['created']
```

```
class Song(models.Model):

    YEAR_MIN = models.FloatField(null="True")
    YEAR_MAX = models.FloatField(null="True")

    TIMBREAVG1 = models.FloatField()
    TIMBREAVG2 = models.FloatField()
    TIMBREAVG3 = models.FloatField()
    TIMBREAVG4 = models.FloatField()
    TIMBREAVG5 = models.FloatField()
    TIMBREAVG6 = models.FloatField()
    TIMBREAVG7 = models.FloatField()
    TIMBREAVG8 = models.FloatField()
    TIMBREAVG9 = models.FloatField()
    TIMBREAVG10 = models.FloatField()
    TIMBREAVG11 = models.FloatField()
    TIMBREAVG12 = models.FloatField()

    TIMBRECOV1 = models.FloatField()
    TIMBRECOV2 = models.FloatField()
    TIMBRECOV3 = models.FloatField()
    TIMBRECOV4 = models.FloatField()
    TIMBRECOV5 = models.FloatField()
    TIMBRECOV6 = models.FloatField()
    TIMBRECOV7 = models.FloatField()
    TIMBRECOV8 = models.FloatField()
    TIMBRECOV9 = models.FloatField()
    TIMBRECOV10 = models.FloatField()
```

Seriliazers

- De même que pour le modèle on créer un serializer avec le même nombre de variables
- Lors de la réception d'une musique les deux variables YEAR_MIN/MAX sont nuls, on autorise donc les valeur "null" pour ces variables

```
class SongSerializer(serializers.Serializer):
    YEAR_MIN = serializers.FloatField(allow_null=True)
    YEAR_MAX = serializers.FloatField(allow_null=True)

    TIMBREAVG1 = serializers.FloatField()
    TIMBREAVG2 = serializers.FloatField()
    TIMBREAVG3 = serializers.FloatField()
    TIMBREAVG4 = serializers.FloatField()
    TIMBREAVG5 = serializers.FloatField()
    TIMBREAVG6 = serializers.FloatField()
    TIMBREAVG7 = serializers.FloatField()
    TIMBREAVG8 = serializers.FloatField()
    TIMBREAVG9 = serializers.FloatField()
    TIMBREAVG10 = serializers.FloatField()
    TIMBREAVG11 = serializers.FloatField()
    TIMBREAVG12 = serializers.FloatField()

    TIMBRECOV1 = serializers.FloatField()
    TIMBRECOV2 = serializers.FloatField()
    TIMBRECOV3 = serializers.FloatField()
    TIMBRECOV4 = serializers.FloatField()
    TIMBRECOV5 = serializers.FloatField()
```

Urls

- On accède aux différentes fonctionnalités de l'api au travers de 3 urls:
    - Songs permet d'afficher l'ensemble des musiques enregistrée
    - Songs/pk permet d'afficher/supprimer/modifier une musique précise où pk est l'index auto incrémenté lors de l'ajout de la musique
    - Predict permet de prédire l'intervalle de sortie d'une musique

Prediction

- Pour prédire un intervalle on utilise un modèle créer précédemment.
- Dans un premier temps on charge le modèle dans la variable mdl2
- On récupère ensuite les valeurs de la requête que l'on met en forme et auxquelles on retire les deux champs YEAR_MIN/MAX.
- Une fois l'intervalle prédit il suffit de d'utiliser la fonction predict_int() qui retourne les deux années définissant l'intervalle souhaité.
- On remet alors en forme les données de la musique avant de l'enregistrer si le format des données est valide.

```
def predict_int(intervalle):
    switcher = {
        1: "1922 1983",
        2: "1983 1992",
        3: "1992 1996",
        4: "1996 1999",
        5: "1999 2002",
        6: "2002 2004",
        7: "2004 2005",
        8: "2005 2007",
        9: "2007 2008",
        10: "2008 2011"
    }
    date = switcher.get(intervalle).split(" ")
    return date[0],date[1]
```

```
mdl2 = joblib.load(".\\prediction\\rf1.sav")

data = JSONParser().parse(request)

data_pd = pd.DataFrame([data])
data_pred = mdl2.predict(data_pd.drop(["YEAR_MIN", "YEAR_MAX"], axis=1))

YEAR_MIN,YEAR_MAX = predict_int(data_pred[0])
data_pd["YEAR_MAX"] = float(YEAR_MAX)
data_pd["YEAR_MIN"] = float(YEAR_MIN)

data=data_pd.to_dict('records')

serializer = SongSerializer(data=data[0])
if serializer.is_valid():
    serializer.save()
```

Views url /songs/$

- On affiche les musiques à l'aide GET et on ajoute une musique sans prédire YEAR_MIN/MAX à l'aide d'une requête POST

```python
@api_view(['GET', 'POST'])
def song_list(request):
    if request.method == 'GET':
        songs = Song.objects.all()
        serializer = SongSerializer(songs, many=True)
        return Response(serializer.data)
    elif request.method == 'POST':
        data = JSONParser().parse(request)
        serializer = SongSerializer(data=data)
        if serializer.is_valid():
            serializer.save()
            return Response(serializer.data, status=201)
        return Response(serializer.errors, status=400)
```

Views url /songs/pk/$

- GET et DELETE affiche ou supprime une musique en fonction de son index
- PUT permet de modifier une musique en fonction de son indexe. Pour cela on prédit à nouveau les valeurs YEAR_MIN/MAX une fois les valeurs modifiées

```python
@api_view(['GET', 'PUT', 'DELETE'])
def song_detail(request, pk):
    try:
        song = Song.objects.get(pk=pk)
    except Song.DoesNotExist:
        return HttpResponse(status=404)
    if request.method == 'GET':
        serializer = SongSerializer(song)
        return Response(serializer.data)
    elif request.method == 'PUT':

        mdl2 = joblib.load(".\\prediction\\rf1.sav")
        data = JSONParser().parse(request)
        data_pd = pd.DataFrame([data])
        data_pred = mdl2.predict(data_pd.drop(["YEAR_MIN", "YEAR_MAX"], axis=1))

        YEAR_MIN, YEAR_MAX = predict_int(data_pred[0])
        data_pd["YEAR_MAX"] = float(YEAR_MAX)
        data_pd["YEAR_MIN"] = float(YEAR_MIN)

        data = data_pd.to_dict('records')

        serializer = SongSerializer(song, data=data[0])
        if serializer.is_valid():
            serializer.save()
            return Response(serializer.data)
        return Response(serializer.errors, status=400)
    elif request.method == 'DELETE':
        song.delete()
        return HttpResponse(status=204)
```

Views url /predict/$

- On utilise la methode POST pour prédire les valeurs YEAR_MIN/MAX et on enregistre la musique dans la base de données

```python
@api_view(['POST'])
def predict(request):
    mdl2 = joblib.load(".\\prediction\\rf1.sav")

    data = JSONParser().parse(request)

    data_pd = pd.DataFrame([data])
    data_pred = mdl2.predict(data_pd.drop(["YEAR_MIN", "YEAR_MAX"], axis=1))

    YEAR_MIN,YEAR_MAX = predict_int(data_pred[0])
    data_pd["YEAR_MAX"] = float(YEAR_MAX)
    data_pd["YEAR_MIN"] = float(YEAR_MIN)

    data=data_pd.to_dict('records')

    serializer = SongSerializer(data=data[0])
    if serializer.is_valid():
        serializer.save()
        return Response(serializer.data, status=201)
    return Response(serializer.errors, status=400)
```

Example 1

- On affiche l'ensemble des musiques de la base de donées

http://127.0.0.1:8000/songs/

GET /songs/

HTTP 200 OK
Allow: OPTIONS, POST, GET
Content-Type: application/json
Vary: Accept


```
[
    {
        "YEAR_MIN": 2004.0,
        "YEAR_MAX": 2005.0,
        "TIMBREAVG1": 44.88723,
        "TIMBREAVG2": 14.1476,
        "TIMBREAVG3": -5.70694,
        "TIMBREAVG4": -19.7048,
        "TIMBREAVG5": -58.91571,
        "TIMBREAVG6": -14.32484,
        "TIMBREAVG7": -3.92128,
        "TIMBREAVG8": -0.48551,
        "TIMBREAVG9": 2.18089,
        "TIMBREAVG10": 2.08289,
        "TIMBREAVG11": -4.6978,
        "TIMBREAVG12": -6.68947,
        "TIMBRECOV1": 45.04778,
        "TIMBRECOV2": 3303.09417,
        "TIMBRECOV3": 964.18093,
        "TIMBRECOV4": 1129.6289,
        "TIMBRECOV5": 1137.92364,
        "TIMBRECOV6": 555.16029,
        "TIMBRECOV7": 413.8053,
        "TIMBRECOV8": 350.98413,
        "TIMBRECOV9": 323.90979,
        "TIMBRECOV10": 258.00695,
        "TIMBRECOV11": 185.19776,
        "TIMBRECOV12": 235.38329,
```


- On affiche une musique spécifique

http://127.0.0.1:8000/songs/19/

GET /songs/19/

HTTP 200 OK
Allow: PUT, OPTIONS, DELETE, GET
Content-Type: application/json
Vary: Accept


```
{
    "YEAR_MIN": 2004.0,
    "YEAR_MAX": 2005.0,
    "TIMBREAVG1": 44.88723,
    "TIMBREAVG2": 14.1476,
    "TIMBREAVG3": -5.70694,
    "TIMBREAVG4": -19.7048,
    "TIMBREAVG5": -58.91571,
    "TIMBREAVG6": -14.32484,
    "TIMBREAVG7": -3.92128,
    "TIMBREAVG8": -0.48551,
    "TIMBREAVG9": 2.18089,
    "TIMBREAVG10": 2.08289,
    "TIMBREAVG11": -4.6978,
    "TIMBREAVG12": -6.68947,
    "TIMBRECOV1": 45.04778,
    "TIMBRECOV2": 3303.09417,
    "TIMBRECOV3": 964.18093,
    "TIMBRECOV4": 1129.6289,
    "TIMBRECOV5": 1137.92364,
    "TIMBRECOV6": 555.16029,
```

Example 2

- On utilise Postman pour les requêtes YEAR_MIN/MAX sont nuls
- L'api renvoi donc l'intevalle prédit