Imports

```
#Sickit learn met régulièrement à jour des versions et
#indique des futurs warnings.
#ces deux lignes permettent de ne pas les afficher.
import warnings
warnings.filterwarnings("ignore", category=FutureWarning)
# librairies générales
import pandas as pd
import re
from tabulate import tabulate
import time
import numpy as np
import pickle
import string
import base64
import sys
# librairie affichage
import matplotlib.pyplot as plt
import seaborn as sns
# librairies scikit learn
import sklearn
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.base import TransformerMixin
from sklearn.pipeline import Pipeline
from sklearn.model_selection import train_test_split
from sklearn import metrics
from sklearn.model_selection import cross_val_score
from sklearn.metrics import confusion_matrix
from sklearn.metrics import classification_report
from sklearn.model_selection import KFold
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import accuracy_score
# librairies des classifiers utilisés
from sklearn.svm import SVC
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.naive_bayes import MultinomialNB
from sklearn.ensemble import RandomForestClassifier
import warnings
warnings.filterwarnings("ignore")
```

```
# pour monter son drive Google Drive local
from google.colab import drive
drive.mount('/content/gdrive')
my_local_drive='/content/gdrive/My Drive/Colab Notebooks/TER'
# Ajout du path pour les librairies, fonctions et données
sys.path.append(my_local_drive)
# Se positionner sur le répertoire associé
%cd $my_local_drive
%pwd
```

```
Mounted at /content/gdrive
/content/gdrive/My Drive/Colab Notebooks/TER
'/content/gdrive/My Drive/Colab Notebooks/TER'
```

Classification

```
import pandas as pd
V2 = pd.read_excel("ISAC_V2.xlsx")
V2.head()
```

	Unnamed: 0	Chip_Code	Chip_Type	French_Residence_Department	French_Regio
0	WQW0008	202272574	ISAC_V2	51	
1	WQW0054	210553086	ISAC_V2	51	
2	WQW0059	210993011	ISAC_V2	51	
3	WQW0160	221402048	ISAC_V2	51	
4	WQW0189	223042355	ISAC_V2	67	

5 rows × 127 columns

```
target_1 = [
    "Allergy_Present",
    "Respiratory_Allergy",
    "Food_Allergy",
    "Venom_Allergy",
    "Severe_Allergy",
    "Type_of_Food_Allergy_Other",
    "Type_of_Respiratory_Allergy_IGE_Pollen_Herb",
    "Type_of_Respiratory_Allergy_IGE_Pollen_Tree",
    "Type_of_Respiratory_Allergy_IGE_Dander_Animals",
    "Type_of_Respiratory_Allergy_IGE_Mite_Cockroach",
    "Type_of_Respiratory_Allergy_IGE_Molds_Yeast",
    "Type_of_Respiratory_Allergy_ARIA",
    "Type_of_Respiratory_Allergy_CONJ",
    "Type_of_Respiratory_Allergy_IGE_Pollen_Gram",
    "Type_of_Respiratory_Allergy_GINA",
    "Type_of_Food_Allergy_Aromatics",
    "Type_of_Food_Allergy_Cereals_&_Seeds",
    "Type_of_Food_Allergy_Egg",
    "Type_of_Food_Allergy_Fish",
    "Type_of_Food_Allergy_Fruits_and_Vegetables",
    "Type_of_Food_Allergy_Mammalian_Milk",
    "Type_of_Food_Allergy_Oral_Syndrom",
    "Type_of_Food_Allergy_Other_Legumes",
```

```
    "Type_of_Food_Allergy_Peanut",
    "Type_of_Food_Allergy_Shellfish",
    "Type_of_Food_Allergy_TPO",
    "Type_of_Food_Allergy_Tree_Nuts",
    "Type_of_Venom_Allergy_ATCD_Venom",
    "Type_of_Venom_Allergy_IGE_Venom",
]

extra_columns = [
    "Chip_Type",
    "Chip_Code",
    "French_Region",
    "French_Residence_Department"
    ]

extra = ['History_of_food_anaphylaxis','First_degree_family_history_of_atopy',
         'History_of_hymenoptera_venom_anaphylaxis','Mammalian_meat']
extra_1 = ["Conjunctivitis", "Oral_Syndrom", "Cardiovascular_symptoms", "Respir

Gina = ["GINA_(asthma)_0", "GINA_(asthma)_1", "GINA_(asthma)_2", "GINA_(asthma)
inconnu = ["Treatment_of_athsma_9", "Treatment_of_rhinitis_9", "General_cofacto
           "Age_of_onsets_9", "ARIA_(rhinitis)_9", "GINA_(asthma)_9", "Treatmer
Aria = ["ARIA_(rhinitis)_9", "ARIA_(rhinitis)_0", "ARIA_(rhinitis)_1", "ARIA_(r


import pandas as pd
import numpy as np
from sklearn.model_selection import StratifiedKFold
from sklearn.ensemble import RandomForestClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from xgboost import XGBClassifier
from sklearn.metrics import (
    f1_score, accuracy_score, recall_score,
    precision_score, confusion_matrix, roc_auc_score, roc_curve
)
from imblearn.over_sampling import SMOTE
import plotly.graph_objects as go

targets = ["Allergy_Present", "Respiratory_Allergy", "Food_Allergy"]

models = {
    "RandomForest": RandomForestClassifier(random_state=42),
    "XGBoost": XGBClassifier(random_state=42, eval_metric="logloss", use_label_
    "LogisticRegression": LogisticRegression(max_iter=1000, random_state=42),
    "SVM": SVC(probability=True, random_state=42)
}
```

```
X = V2.copy()
X.drop(target_1, axis=1, inplace=True)
X.drop(extra_columns, axis=1, inplace=True)
X.drop(extra, axis=1, inplace=True)
X.drop(inconnu, axis=1, inplace=True)
X = X.iloc[:, 1:]

results = []
kfold = StratifiedKFold(n_splits=10, shuffle=True, random_state=42)

for target in targets:
    y = V2[target]

    for model_name, base_model in models.items():
        f1_class0_scores, f1_class1_scores = [], []
        precision_scores, acc_scores, recall_scores, auc_scores = [], [], [], [

        for train_idx, test_idx in kfold.split(X, y):
            X_train, X_test = X.iloc[train_idx], X.iloc[test_idx]
            y_train, y_test = y.iloc[train_idx], y.iloc[test_idx]

            smote = SMOTE(random_state=42)
            X_train_res, y_train_res = smote.fit_resample(X_train, y_train)

            base_model.fit(X_train_res, y_train_res)
            y_pred = base_model.predict(X_test)

            acc_scores.append(accuracy_score(y_test, y_pred))
            recall_scores.append(recall_score(y_test, y_pred, zero_division=0))
            precision_scores.append(precision_score(y_test, y_pred, average='we
            f1_class0_scores.append(f1_score(y_test, y_pred, pos_label=0, zero_
            f1_class1_scores.append(f1_score(y_test, y_pred, pos_label=1, zero_

            if hasattr(base_model, "predict_proba"):
                y_proba = base_model.predict_proba(X_test)[:, 1]
                auc_scores.append(roc_auc_score(y_test, y_proba))

        base_model.fit(X, y)
        y_pred_full = base_model.predict(X)
        y_proba_full = base_model.predict_proba(X)[:, 1] if hasattr(base_model,
        matrix = confusion_matrix(y, y_pred_full)

        print(f"\n🔍 Target: {target} | Model: {model_name}")
        print(f"📈 Accuracy: {np.mean(acc_scores):.4f}")
        print(f"🎯 F1 (0): {np.mean(f1_class0_scores):.4f} | F1 (1): {np.mean(
        print(f"📊 Precision: {np.mean(precision_scores):.4f} | AUC: {np.mean(a
        print("📊 Confusion Matrix:\n", matrix)
```

```
        if y_proba_full is not None:
            fpr, tpr, _ = roc_curve(y, y_proba_full)
            fig = go.Figure()
            fig.add_trace(go.Scatter(x=fpr, y=tpr, mode='lines', name=f"{model_
            fig.add_trace(go.Scatter(x=[0, 1], y=[0, 1], mode='lines', name='Ra
            fig.update_layout(
                title=f"ROC Curve - {target} - {model_name}",
                xaxis_title="False Positive Rate",
                yaxis_title="True Positive Rate",
                width=700, height=500
            )
            fig.show()

        results.append({
            "Target": target,
            "Model": model_name,
            "F1_Class_0": np.mean(f1_class0_scores),
            "F1_Class_1": np.mean(f1_class1_scores),
            "Precision": np.mean(precision_scores),
            "Accuracy": np.mean(acc_scores),
            "Recall": np.mean(recall_scores),
            "AUC_ROC": np.mean(auc_scores) if auc_scores else np.nan
        })

pd.DataFrame(results).to_csv("results_V2_Allergie.csv", index=False)
```

Show hidden output

```
import pandas as pd
import numpy as np
from sklearn.model_selection import StratifiedKFold
from sklearn.ensemble import RandomForestClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from xgboost import XGBClassifier
from sklearn.metrics import (
    f1_score, accuracy_score, recall_score,
    precision_score, confusion_matrix, roc_auc_score, roc_curve
)
from imblearn.over_sampling import SMOTE
import plotly.graph_objects as go

V2_sev = V2[V2["Allergy_Present"] == 1]
targets = ["Severe_Allergy"]
models = {
    "RandomForest": RandomForestClassifier(random_state=42),
    "XGBoost": XGBClassifier(random_state=42, eval_metric="logloss", use_label_
```

```
    "LogisticRegression": LogisticRegression(max_iter=1000, random_state=42),
    "SVM": SVC(probability=True, random_state=42)
}
X = V2_sev.copy()
X.drop(target_1, axis=1, inplace=True)
X.drop(extra_columns, axis=1, inplace=True)
X.drop(extra, axis=1, inplace=True)
X.drop(inconnu, axis=1, inplace=True)
X = X.iloc[:, 1:]
results_severe = []
kfold = StratifiedKFold(n_splits=10, shuffle=True, random_state=42)

for target in targets:
    y = V2_sev[target]

    for model_name, base_model in models.items():
        f1_class0_scores, f1_class1_scores = [], []
        precision_scores, acc_scores, recall_scores, auc_scores = [], [], [], [

        for train_idx, test_idx in kfold.split(X, y):
            X_train, X_test = X.iloc[train_idx], X.iloc[test_idx]
            y_train, y_test = y.iloc[train_idx], y.iloc[test_idx]

            smote = SMOTE(random_state=42)
            X_train_res, y_train_res = smote.fit_resample(X_train, y_train)

            base_model.fit(X_train_res, y_train_res)
            y_pred = base_model.predict(X_test)

            acc_scores.append(accuracy_score(y_test, y_pred))
            recall_scores.append(recall_score(y_test, y_pred, zero_division=0))
            precision_scores.append(precision_score(y_test, y_pred, average='we
            f1_class0_scores.append(f1_score(y_test, y_pred, pos_label=0, zero_
            f1_class1_scores.append(f1_score(y_test, y_pred, pos_label=1, zero_

            if hasattr(base_model, "predict_proba"):
                y_proba = base_model.predict_proba(X_test)[:, 1]
                auc_scores.append(roc_auc_score(y_test, y_proba))

        base_model.fit(X, y)
        y_pred_full = base_model.predict(X)
        y_proba_full = base_model.predict_proba(X)[:, 1] if hasattr(base_model,
        matrix = confusion_matrix(y, y_pred_full)

        print(f"\n🔍 Target: {target} | Model: {model_name}")
        print(f"📈 Accuracy: {np.mean(acc_scores):.4f}")
        print(f"🎯 F1 (0): {np.mean(f1_class0_scores):.4f} | F1 (1): {np.mean(
        print(f"📊 Precision: {np.mean(precision_scores):.4f} | AUC: {np.mean(
```

```
        print("📊 Confusion Matrix:\n", matrix)

        if y_proba_full is not None:
            fpr, tpr, _ = roc_curve(y, y_proba_full)
            fig = go.Figure()
            fig.add_trace(go.Scatter(x=fpr, y=tpr, mode='lines', name=f"{model_
            fig.add_trace(go.Scatter(x=[0, 1], y=[0, 1], mode='lines', name='Ra
            fig.update_layout(
                title=f"ROC Curve - {target} - {model_name}",
                xaxis_title="False Positive Rate",
                yaxis_title="True Positive Rate",
                width=700, height=500
            )
            fig.show()

        results_severe.append({
            "Target": target,
            "Model": model_name,
            "F1_Class_0": np.mean(f1_class0_scores),
            "F1_Class_1": np.mean(f1_class1_scores),
            "Precision": np.mean(precision_scores),
            "Accuracy": np.mean(acc_scores),
            "Recall": np.mean(recall_scores),
            "AUC_ROC": np.mean(auc_scores) if auc_scores else np.nan
        })

pd.DataFrame(results_severe).to_csv("results_V2_severe.csv", index=False)
```

Show hidden output

```
import pandas as pd
import numpy as np
from sklearn.model_selection import StratifiedKFold
from sklearn.ensemble import RandomForestClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from xgboost import XGBClassifier
from sklearn.metrics import (
    f1_score, accuracy_score, recall_score,
    precision_score, confusion_matrix, roc_auc_score, roc_curve
)
from imblearn.over_sampling import SMOTE
import plotly.graph_objects as go

# Données respiratoires
V2_res = V2[V2["Respiratory_Allergy"] == 1]
```

```
targets = ["Type_of_Respiratory_Allergy_IGE_Pollen_Herb",
    "Type_of_Respiratory_Allergy_IGE_Pollen_Tree",
    "Type_of_Respiratory_Allergy_IGE_Dander_Animals",
    "Type_of_Respiratory_Allergy_IGE_Mite_Cockroach",
    "Type_of_Respiratory_Allergy_IGE_Molds_Yeast",
    "Type_of_Respiratory_Allergy_ARIA",
    "Type_of_Respiratory_Allergy_CONJ",
    "Type_of_Respiratory_Allergy_IGE_Pollen_Gram",
    "Type_of_Respiratory_Allergy_GINA"]

models = {
    "RandomForest": RandomForestClassifier(random_state=42),
    "XGBoost": XGBClassifier(random_state=42, eval_metric="logloss", use_label_
    "LogisticRegression": LogisticRegression(max_iter=1000, random_state=42),
    "SVM": SVC(probability=True, random_state=42)
}

X = V2_res.copy()
X.drop(target_1, axis=1, inplace=True)
X.drop(extra_columns, axis=1, inplace=True)
X.drop(extra, axis=1, inplace=True)
X.drop(inconnu, axis=1, inplace=True)
X = X.iloc[:, 1:]

results_res = []
kfold = StratifiedKFold(n_splits=10, shuffle=True, random_state=42)

# Boucle principale
for target in targets:
    y = V2_res[target]

    for model_name, base_model in models.items():
        f1_class0_scores, f1_class1_scores = [], []
        precision_scores, acc_scores, recall_scores, auc_scores = [], [], [], |

        print(f"\n🔍 Target: {target} | Model: {model_name}")

        for train_idx, test_idx in kfold.split(X, y):
            X_train, X_test = X.iloc[train_idx], X.iloc[test_idx]
            y_train, y_test = y.iloc[train_idx], y.iloc[test_idx]

            # Application de SMOTE sur les données d'entraînement
            smote = SMOTE(random_state=42)
            X_train_res, y_train_res = smote.fit_resample(X_train, y_train)

            base_model.fit(X_train_res, y_train_res)
            y_pred = base_model.predict(X_test)
```

```
            acc_scores.append(accuracy_score(y_test, y_pred))
            recall_scores.append(recall_score(y_test, y_pred, zero_division=0))
            precision_scores.append(precision_score(y_test, y_pred, average='we
            f1_class0_scores.append(f1_score(y_test, y_pred, pos_label=0, zero_
            f1_class1_scores.append(f1_score(y_test, y_pred, pos_label=1, zero_

            if hasattr(base_model, "predict_proba"):
                y_proba = base_model.predict_proba(X_test)[:, 1]
                auc_scores.append(roc_auc_score(y_test, y_proba))

        # Entraînement final sur tout X (sans SMOTE ici, car prédiction globale
        base_model.fit(X, y)
        y_pred_full = base_model.predict(X)
        y_proba_full = base_model.predict_proba(X)[:, 1] if hasattr(base_model,
        matrix = confusion_matrix(y, y_pred_full)

        print(f"📈 Accuracy: {np.mean(acc_scores):.4f}")
        print(f"🎯 F1 (0): {np.mean(f1_class0_scores):.4f} | F1 (1): {np.mean(f
        print(f"📊 Precision: {np.mean(precision_scores):.4f} | AUC: {np.mean(a
        print("📊 Confusion Matrix:\n", matrix)

        if y_proba_full is not None:
            fpr, tpr, _ = roc_curve(y, y_proba_full)
            fig = go.Figure()
            fig.add_trace(go.Scatter(x=fpr, y=tpr, mode='lines', name=f"{model_
            fig.add_trace(go.Scatter(x=[0, 1], y=[0, 1], mode='lines', name='Ra
            fig.update_layout(
                title=f"ROC Curve - {target} - {model_name}",
                xaxis_title="False Positive Rate",
                yaxis_title="True Positive Rate",
                width=700, height=500
            )
            fig.show()

        results_res.append({
            "Target": target,
            "Model": model_name,
            "F1_Class_0": np.mean(f1_class0_scores),
            "F1_Class_1": np.mean(f1_class1_scores),
            "Precision": np.mean(precision_scores),
            "Accuracy": np.mean(acc_scores),
            "Recall": np.mean(recall_scores),
            "AUC_ROC": np.mean(auc_scores) if auc_scores else np.nan
        })

pd.DataFrame(results_res).to_csv("results_V2_respiratoire.csv", index=False)
```

Show hidden output

```
import pandas as pd
import numpy as np
from sklearn.model_selection import StratifiedKFold
from sklearn.ensemble import RandomForestClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from xgboost import XGBClassifier
from sklearn.metrics import (
    f1_score, accuracy_score, recall_score,
    precision_score, confusion_matrix, roc_auc_score, roc_curve
)
from imblearn.over_sampling import SMOTE
import plotly.graph_objects as go

V2_food = V2[V2["Food_Allergy"] == 1]
targets = ["Type_of_Food_Allergy_Aromatics",
    "Type_of_Food_Allergy_Cereals_&_Seeds",
    "Type_of_Food_Allergy_Egg",
    "Type_of_Food_Allergy_Fish",
    "Type_of_Food_Allergy_Fruits_and_Vegetables",
    "Type_of_Food_Allergy_Mammalian_Milk",
    "Type_of_Food_Allergy_Oral_Syndrom",
    "Type_of_Food_Allergy_Other_Legumes",
    "Type_of_Food_Allergy_Peanut",
    "Type_of_Food_Allergy_Shellfish",
    "Type_of_Food_Allergy_TPO",
    "Type_of_Food_Allergy_Tree_Nuts"]

models = {
    "RandomForest": RandomForestClassifier(random_state=42),
    "XGBoost": XGBClassifier(random_state=42, eval_metric="logloss", use_label_
    "LogisticRegression": LogisticRegression(max_iter=1000, random_state=42),
    "SVM": SVC(probability=True, random_state=42)
}

X=V2_food.copy()
X.drop(target_1, axis=1, inplace=True)
X.drop(extra_columns, axis=1, inplace=True)
X.drop(extra, axis=1, inplace=True)
X.drop(inconnu, axis=1, inplace=True)
X = X.iloc[:, 1:]
results_food = []

kfold = StratifiedKFold(n_splits=10, shuffle=True, random_state=42)
```

```
for target in targets:
    y = V2_food[target]

    for model_name, base_model in models.items():
        f1_class0_scores, f1_class1_scores = [], []
        precision_scores, acc_scores, recall_scores, auc_scores = [], [], [], |

        for train_idx, test_idx in kfold.split(X, y):
            X_train, X_test = X.iloc[train_idx], X.iloc[test_idx]
            y_train, y_test = y.iloc[train_idx], y.iloc[test_idx]

            smote = SMOTE(random_state=42)
            X_train_res, y_train_res = smote.fit_resample(X_train, y_train)

            base_model.fit(X_train_res, y_train_res)
            y_pred = base_model.predict(X_test)

            acc_scores.append(accuracy_score(y_test, y_pred))
            recall_scores.append(recall_score(y_test, y_pred, zero_division=0))
            precision_scores.append(precision_score(y_test, y_pred, average='we
            f1_class0_scores.append(f1_score(y_test, y_pred, pos_label=0, zero_
            f1_class1_scores.append(f1_score(y_test, y_pred, pos_label=1, zero_

            if hasattr(base_model, "predict_proba"):
                y_proba = base_model.predict_proba(X_test)[:, 1]
                auc_scores.append(roc_auc_score(y_test, y_proba))

        base_model.fit(X, y)
        y_pred_full = base_model.predict(X)
        y_proba_full = base_model.predict_proba(X)[:, 1] if hasattr(base_model,
        matrix = confusion_matrix(y, y_pred_full)

        print(f"\n🔍 Target: {target} | Model: {model_name}")
        print(f"📈 Accuracy: {np.mean(acc_scores):.4f}")
        print(f"🎯 F1 (0): {np.mean(f1_class0_scores):.4f} | F1 (1): {np.mean(
        print(f"📊 Precision: {np.mean(precision_scores):.4f} | AUC: {np.mean(a
        print("📊 Confusion Matrix:\n", matrix)

        if y_proba_full is not None:
            fpr, tpr, _ = roc_curve(y, y_proba_full)
            fig = go.Figure()
            fig.add_trace(go.Scatter(x=fpr, y=tpr, mode='lines', name=f"{model_
            fig.add_trace(go.Scatter(x=[0, 1], y=[0, 1], mode='lines', name='Ra
            fig.update_layout(
                title=f"ROC Curve - {target} - {model_name}",
                xaxis_title="False Positive Rate",
                yaxis_title="True Positive Rate",
                width=700, height=500
```

```
            )
            fig.show()

        results_food.append({
            "Target": target,
            "Model": model_name,
            "F1_Class_0": np.mean(f1_class0_scores),
            "F1_Class_1": np.mean(f1_class1_scores),
            "Precision": np.mean(precision_scores),
            "Accuracy": np.mean(acc_scores),
            "Recall": np.mean(recall_scores),
            "AUC_ROC": np.mean(auc_scores) if auc_scores else np.nan
        })

pd.DataFrame(results_food).to_csv("results_V2_food.csv", index=False)
```

Show hidden output

Ne lancer pas cette partie, c pour la recherche des hyperparametres

```
import pandas as pd
import numpy as np
from sklearn.model_selection import StratifiedKFold, GridSearchCV
from xgboost import XGBClassifier

# Grille des hyperparamètres à tester
param_grid = {
    'n_estimators': [100, 200, 300],
    'max_depth': [3, 5, 7],
    'learning_rate': [0.01, 0.1, 0.2],
    'subsample': [0.8, 1],
    'colsample_bytree': [0.8, 1],
    'gamma': [0, 1, 5]
}

# Données et targets
X = V1.copy()
X.drop(target_1, axis=1, inplace=True)
X.drop(extra_columns, axis=1, inplace=True)
X.drop(extra, axis=1, inplace=True)
X = X.iloc[:, 1:]

targets = [
```

```
    "Allergy_Present",
    "Respiratory_Allergy",
    "Food_Allergy",
    "Venom_Allergy",
    "Severe_Allergy",
    "Type_of_Food_Allergy_Other",
    "Type_of_Respiratory_Allergy_IGE_Pollen_Herb",
    "Type_of_Respiratory_Allergy_IGE_Pollen_Tree",
    "Type_of_Respiratory_Allergy_IGE_Dander_Animals",
    "Type_of_Respiratory_Allergy_IGE_Mite_Cockroach",
    "Type_of_Respiratory_Allergy_IGE_Molds_Yeast",
    "Type_of_Respiratory_Allergy_ARIA",
    "Type_of_Respiratory_Allergy_CONJ",
    "Type_of_Respiratory_Allergy_IGE_Pollen_Gram",
    "Type_of_Respiratory_Allergy_GINA",
    "Type_of_Food_Allergy_Aromatics",
    "Type_of_Food_Allergy_Cereals_&_Seeds",
    "Type_of_Food_Allergy_Egg",
    "Type_of_Food_Allergy_Fish",
    "Type_of_Food_Allergy_Fruits_and_Vegetables",
    "Type_of_Food_Allergy_Mammalian_Milk",
    "Type_of_Food_Allergy_Oral_Syndrom",
    "Type_of_Food_Allergy_Other_Legumes",
    "Type_of_Food_Allergy_Peanut",
    "Type_of_Food_Allergy_Shellfish",
    "Type_of_Food_Allergy_TPO",
    "Type_of_Food_Allergy_Tree_Nuts",
    "Type_of_Venom_Allergy_ATCD_Venom",
    "Type_of_Venom_Allergy_IGE_Venom",
]

all_results = []

for target in targets:
    print(f"\n🔍 Recherche d'hyperparamètres pour le target: {target}")

    y = V1[target]
    filtered_X = X.copy()
    filtered_y = y.copy()

    if target in ["Severe_Allergy", "Respiratory_Allergy", "Food_Allergy", "Ver
        mask = V1["Allergy_Present"] == 1
        filtered_X = filtered_X[mask]
        filtered_y = filtered_y[mask]
    elif target.startswith("Type_of_Respiratory_Allergy_"):
        mask = V1["Respiratory_Allergy"] == 1
        filtered_X = filtered_X[mask]
        filtered_y = filtered_y[mask]
```

```
    elif target.startswith("Type_of_Food_Allergy_"):
        mask = V1["Food_Allergy"] == 1
        filtered_X = filtered_X[mask]
        filtered_y = filtered_y[mask]

    if len(filtered_y.unique()) < 2 or len(filtered_y) < 10:
        print(f" Pas assez de données pour {target}, on passe.")
        continue

    model = XGBClassifier(random_state=42, eval_metric="logloss", use_label_enc

    grid_search = GridSearchCV(
        estimator=model,
        param_grid=param_grid,
        scoring='f1',
        cv=StratifiedKFold(n_splits=5, shuffle=True, random_state=42),
        n_jobs=-1,
        verbose=1
    )

    grid_search.fit(filtered_X, filtered_y)


    best_params = grid_search.best_params_
    best_score = grid_search.best_score_

    best_params['Target'] = target
    best_params['Best_F1_Score'] = best_score

    all_results.append(best_params)

df_results = pd.DataFrame(all_results)
df_results = df_results[['Target'] + [col for col in df_results.columns if col
df_results.to_csv("xgboost_best_hyperparameters_V1.csv", index=False)
```

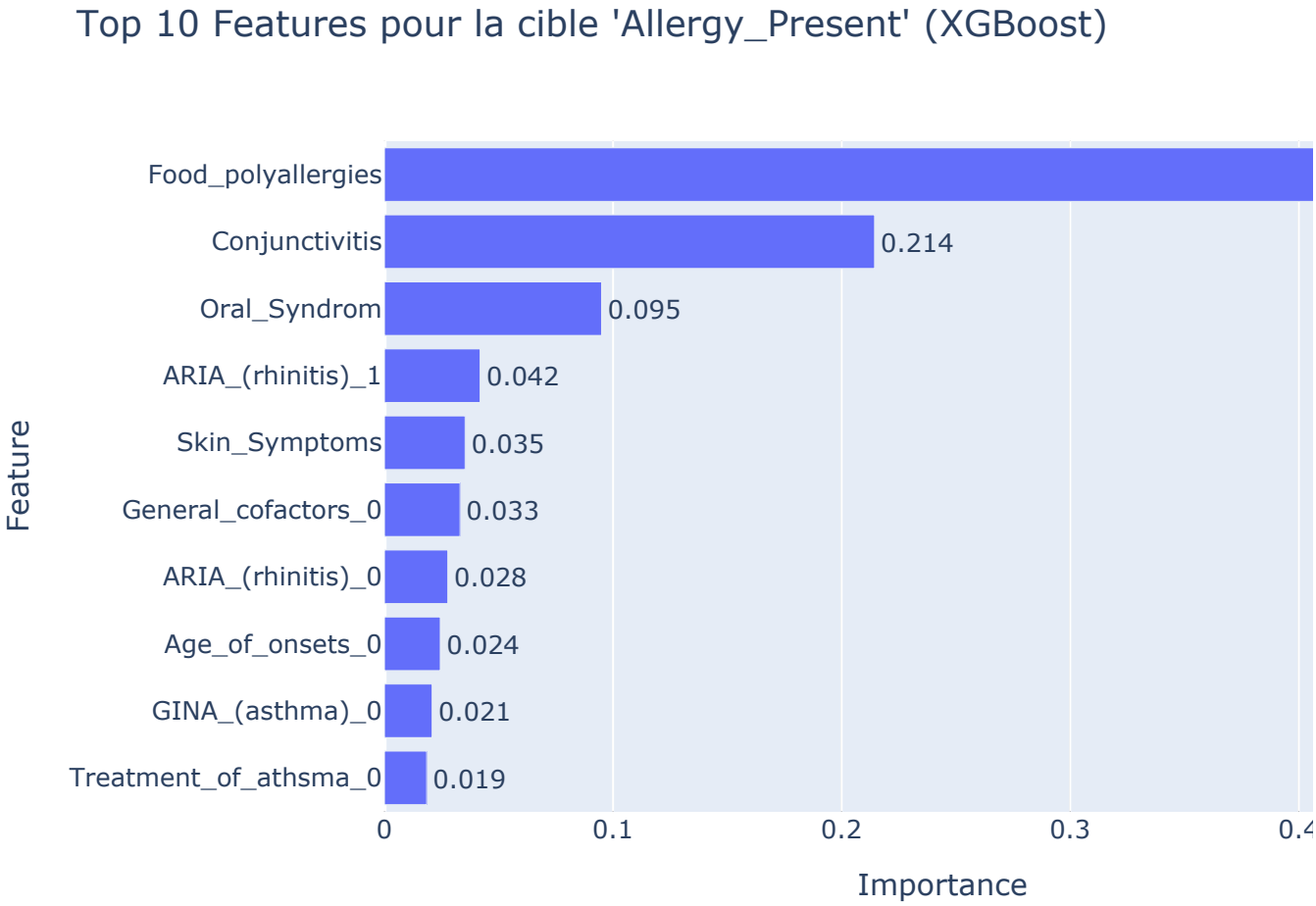
Show hidden output

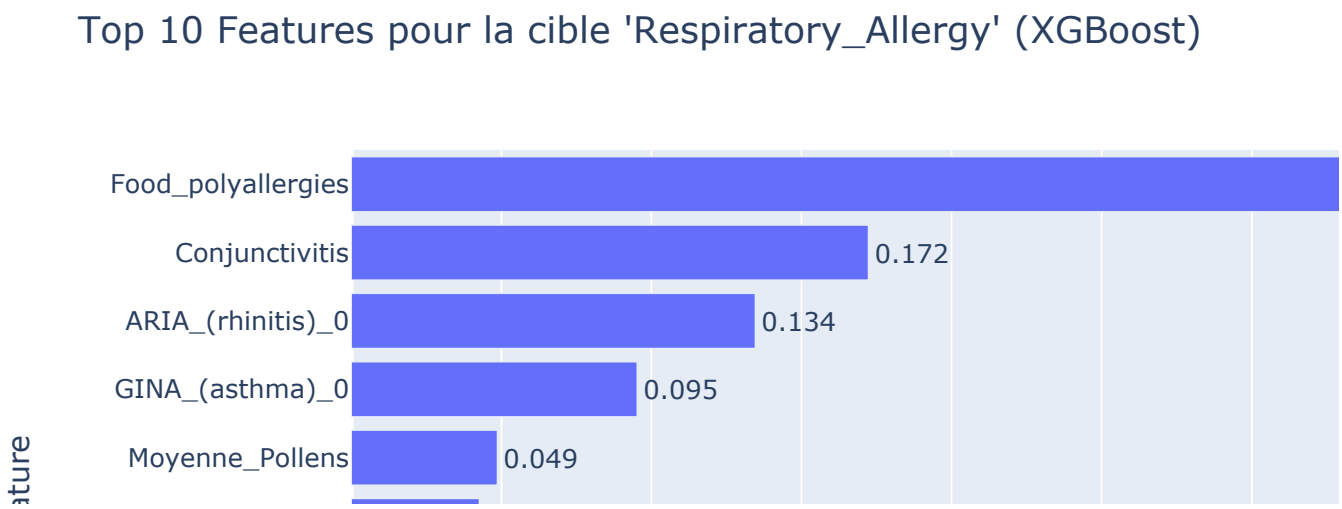
TOP Features

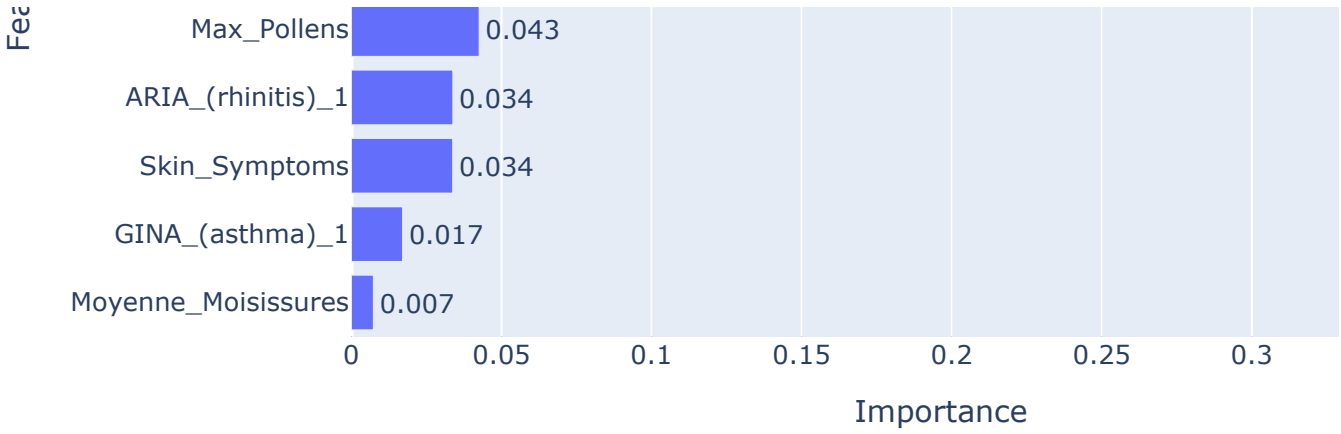
```
import pandas as pd
import numpy as np
from xgboost import XGBClassifier
import plotly.graph_objects as go

targets = [
```

```
    "Allergy_Present", "Respiratory_Allergy", "Food_Allergy", "Venom_Allergy",
    "Severe_Allergy", "Type_of_Food_Allergy_Other", "Type_of_Respiratory_Allerg
    "Type_of_Respiratory_Allergy_IGE_Pollen_Tree", "Type_of_Respiratory_Allergy
    "Type_of_Respiratory_Allergy_IGE_Mite_Cockroach", "Type_of_Respiratory_Alle
    "Type_of_Respiratory_Allergy_ARIA", "Type_of_Respiratory_Allergy_CONJ",
    "Type_of_Respiratory_Allergy_IGE_Pollen_Gram", "Type_of_Respiratory_Allergy
    "Type_of_Food_Allergy_Aromatics", "Type_of_Food_Allergy_Cereals_&_Seeds",
    "Type_of_Food_Allergy_Egg", "Type_of_Food_Allergy_Fish", "Type_of_Food_Alle
    "Type_of_Food_Allergy_Mammalian_Milk", "Type_of_Food_Allergy_Oral_Syndrom",
    "Type_of_Food_Allergy_Other_Legumes", "Type_of_Food_Allergy_Peanut",
    "Type_of_Food_Allergy_Shellfish", "Type_of_Food_Allergy_TPO", "Type_of_Fooc
    "Type_of_Venom_Allergy_ATCD_Venom", "Type_of_Venom_Allergy_IGE_Venom"
]

inconnu = ["Treatment_of_athsma_9", "Treatment_of_rhinitis_9", "General_cofacto
           "Age_of_onsets_9", "ARIA_(rhinitis)_9", "GINA_(asthma)_9", "Treatmer

X = V2.copy()
X.drop(target_1, axis=1, inplace=True)
X.drop(extra_columns, axis=1, inplace=True)
X.drop(extra, axis=1, inplace=True)
X.drop(inconnu, axis=1, inplace=True)
X = X.iloc[:, 1:]

def plot_top_features(model, X_sub, y_sub, target):
    if len(np.unique(y_sub)) < 2:
        print(f"⚠️ Target '{target}' contient une seule classe ({np.unique(y_su
        return

    model.fit(X_sub, y_sub)
    importances = model.feature_importances_
    top_indices = np.argsort(importances)[::-1][:10]
    features = X_sub.columns[top_indices]
    scores = importances[top_indices]

    fig = go.Figure(go.Bar(
        x=scores[::-1],
        y=features[::-1],
        orientation='h',
        text=[f"{s:.3f}" for s in scores[::-1]],
        textposition='outside'
    ))
    fig.update_layout(
        title=f"Top 10 Features pour la cible '{target}' (XGBoost)",
        xaxis_title="Importance",
        yaxis_title="Feature",
        width=800, height=500
    )
```

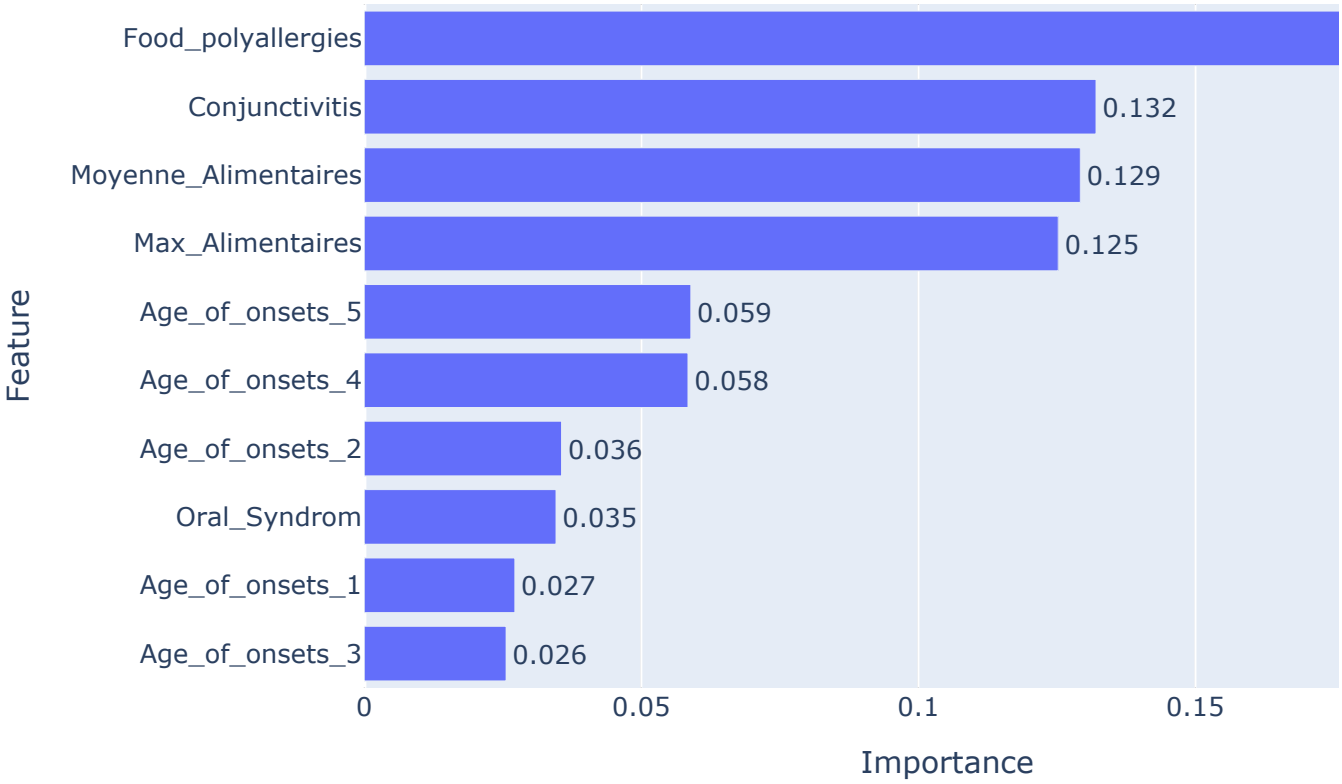
```
    fig.show()

for target in targets:
    X_sub = X.copy()
    y_sub = V2[target]
    model = XGBClassifier(random_state=42, eval_metric="logloss", use_label_enc
    plot_top_features(model, X_sub, y_sub, target)
```



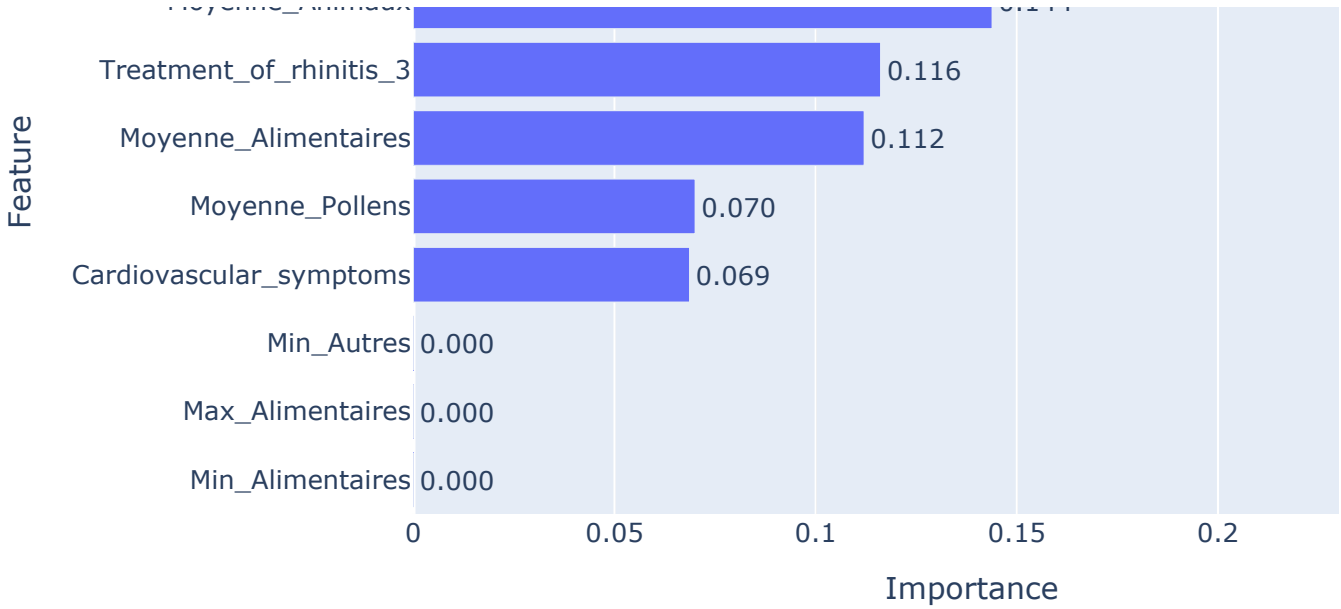




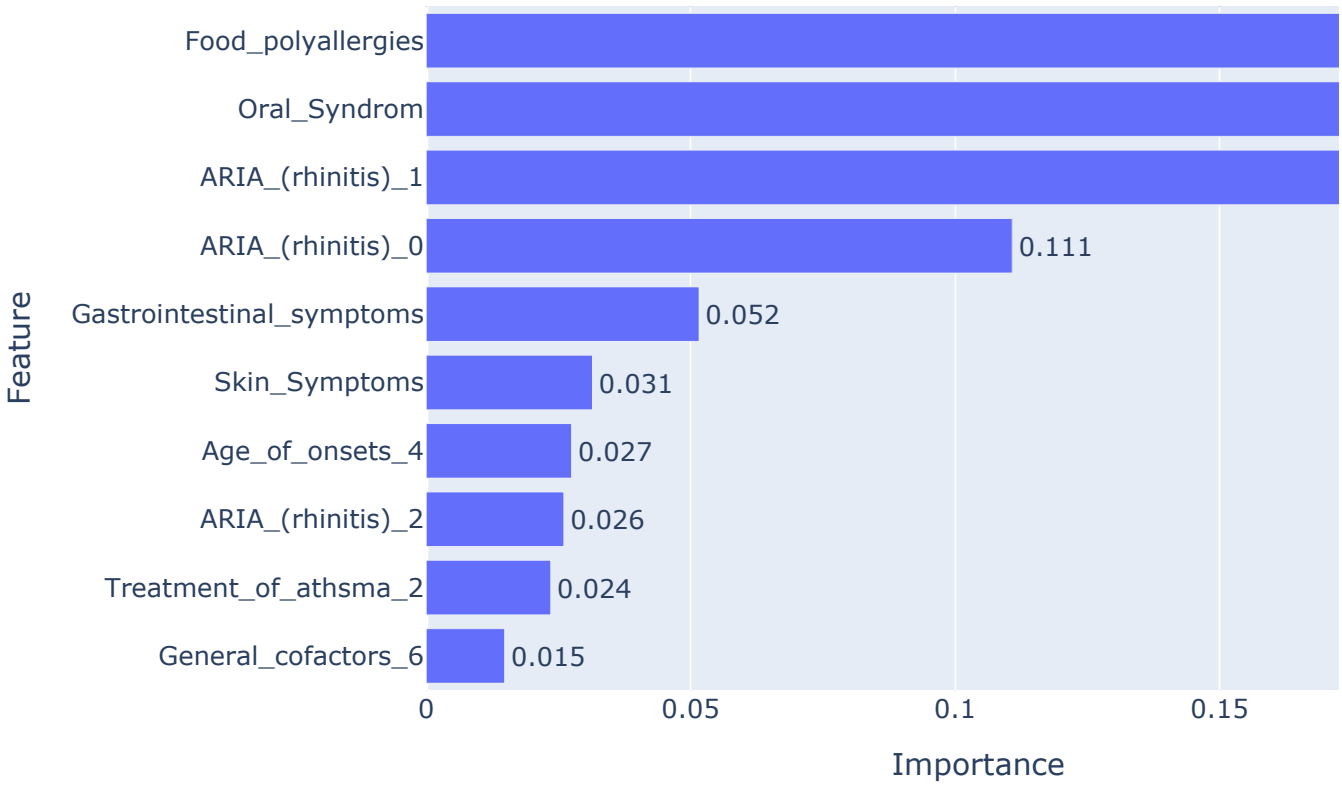
Top 10 Features pour la cible 'Food_Allergy' (XGBoost)



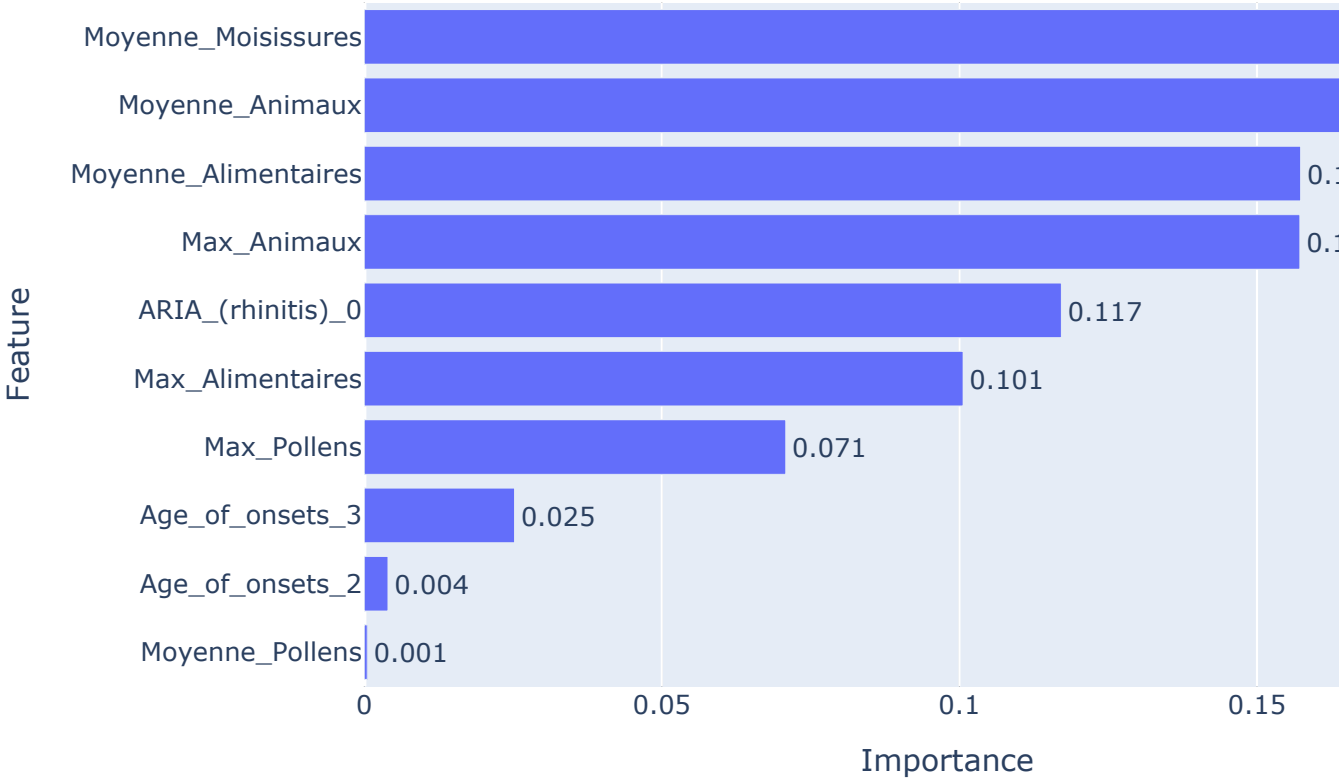
Top 10 Features pour la cible 'Venom_Allergy' (XGBoost)



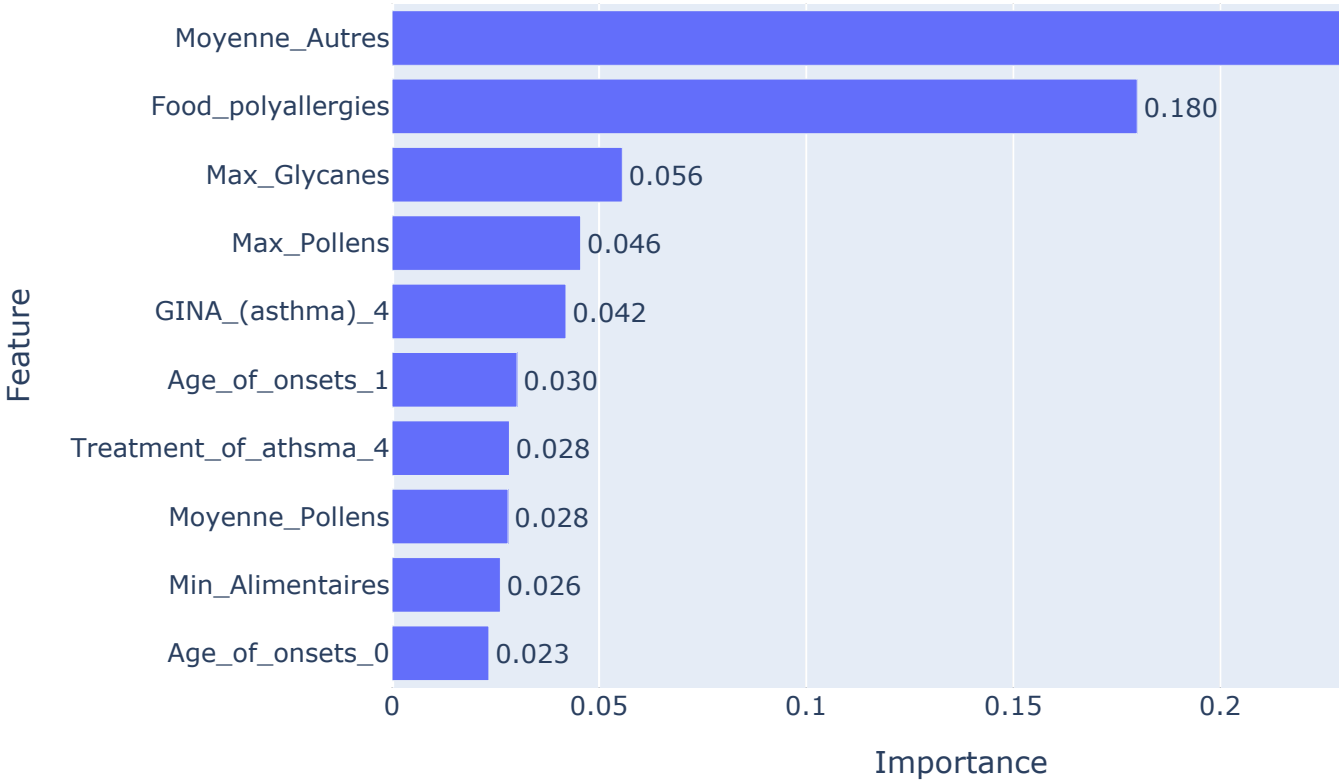
Top 10 Features pour la cible 'Severe_Allergy' (XGBoost)

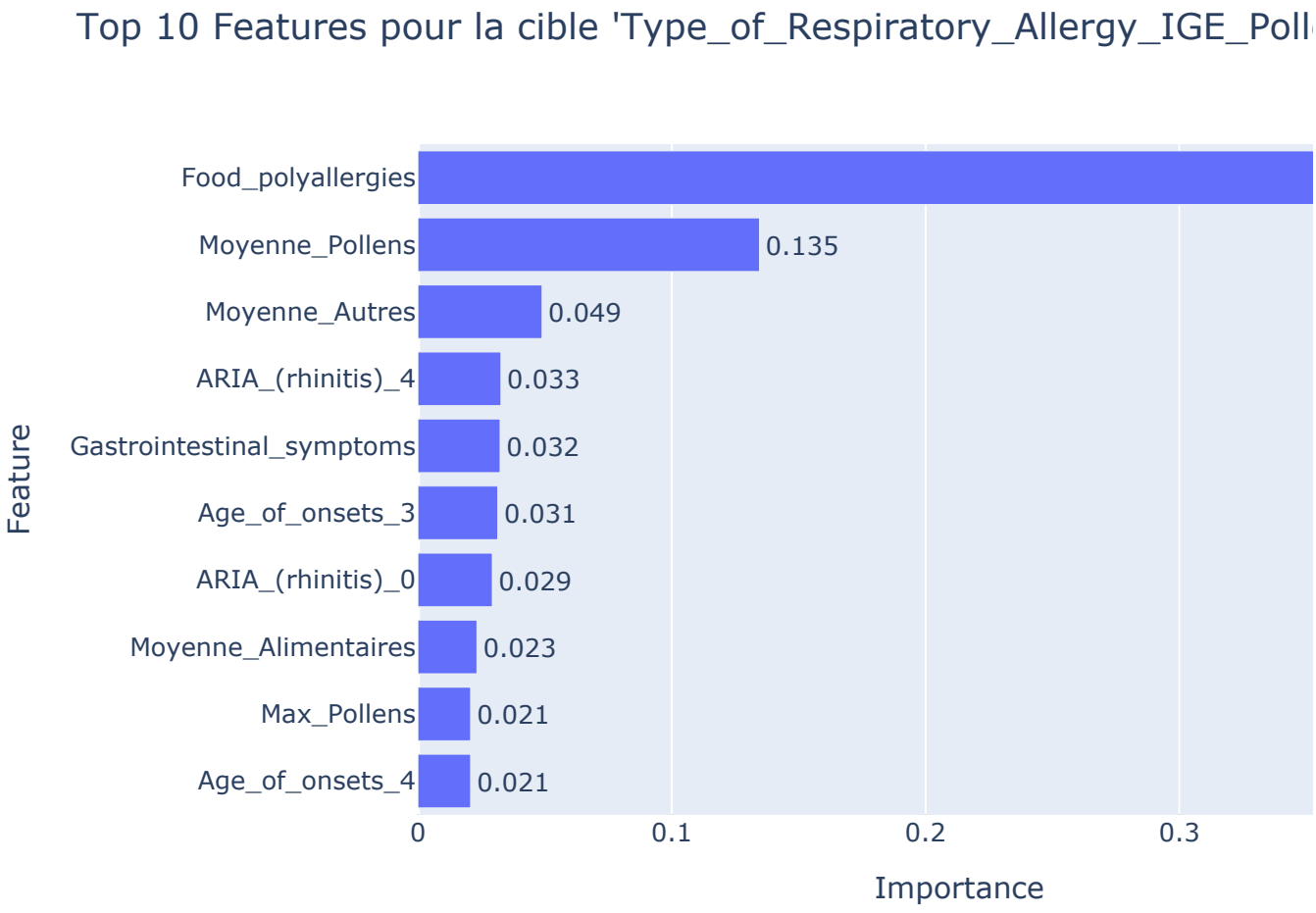


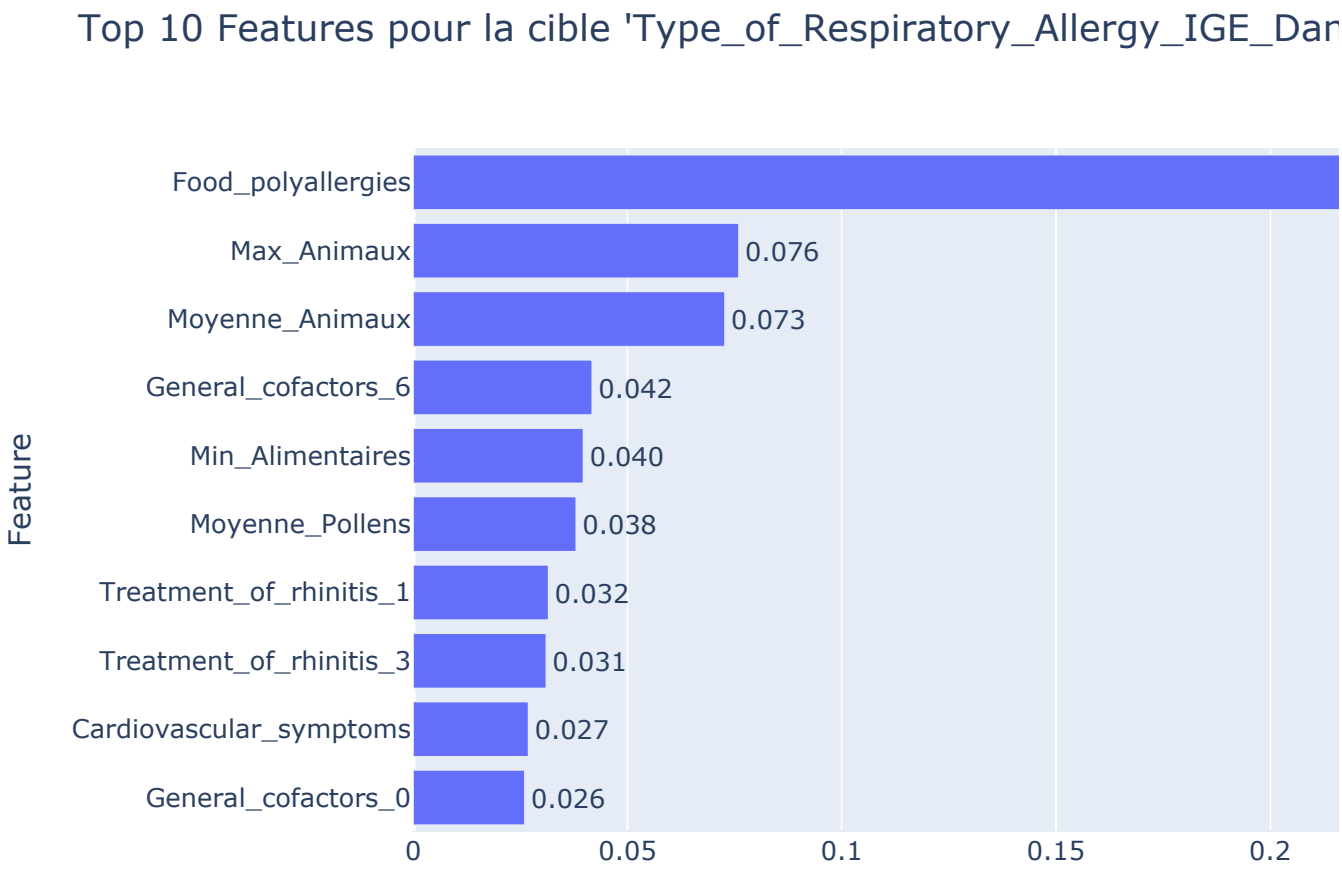
Top 10 Features pour la cible 'Type_of_Food_Allergy_Other' (XGBoos



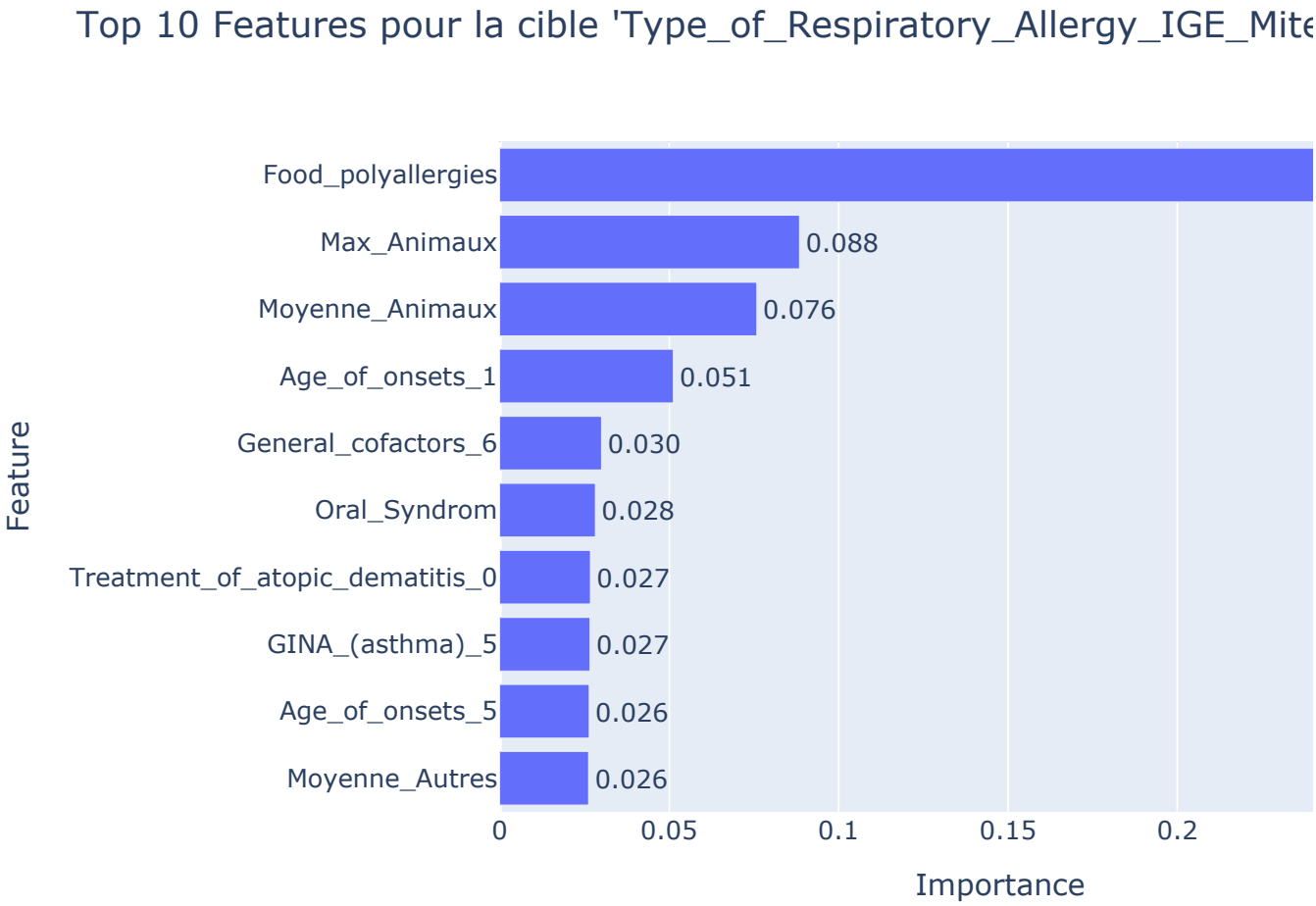
Top 10 Features pour la cible 'Type_of_Respiratory_Allergy_IGE_Poll

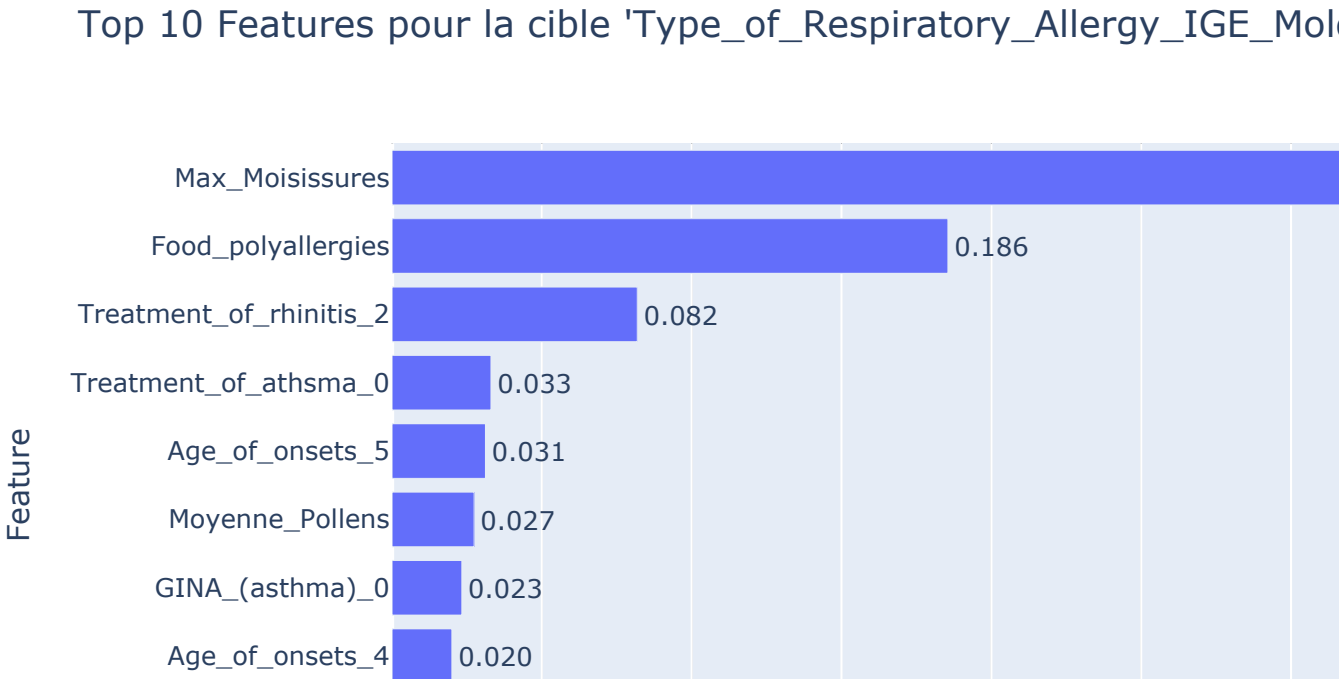


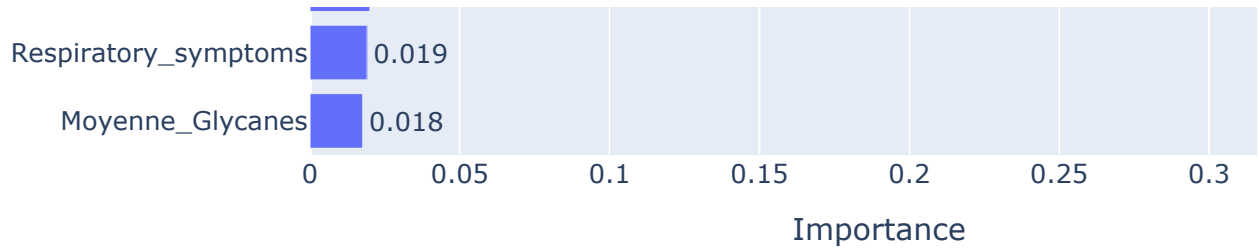

Top 10 Features pour la cible 'Type_of_Respiratory_Allergy_IGE_Poll
Food_polyallergies
Moyenne_Pollens
0.135
Moyenne_Autres
0.049
ARIA_(rhinitis)_4
0.033
Gastrointestinal_symptoms
0.032
Age_of_onsets_3
0.031
ARIA_(rhinitis)_0
0.029
Moyenne_Alimentaires
0.023
Max_Pollens
0.021
Age_of_onsets_4
0.021
Feature
0
0.1
0.2
0.3
Importance


Top 10 Features pour la cible 'Type_of_Respiratory_Allergy_IGE_Dar
Food_polyallergies
Max_Animaux
0.076
Moyenne_Animaux
0.073
General_cofactors_6
0.042
Min_Alimentaires
0.040
Moyenne_Pollens
0.038
Treatment_of_rhinitis_1
0.032
Treatment_of_rhinitis_3
0.031
Cardiovascular_symptoms
0.027
General_cofactors_0
0.026
Feature
0
0.05
0.1
0.15
0.2

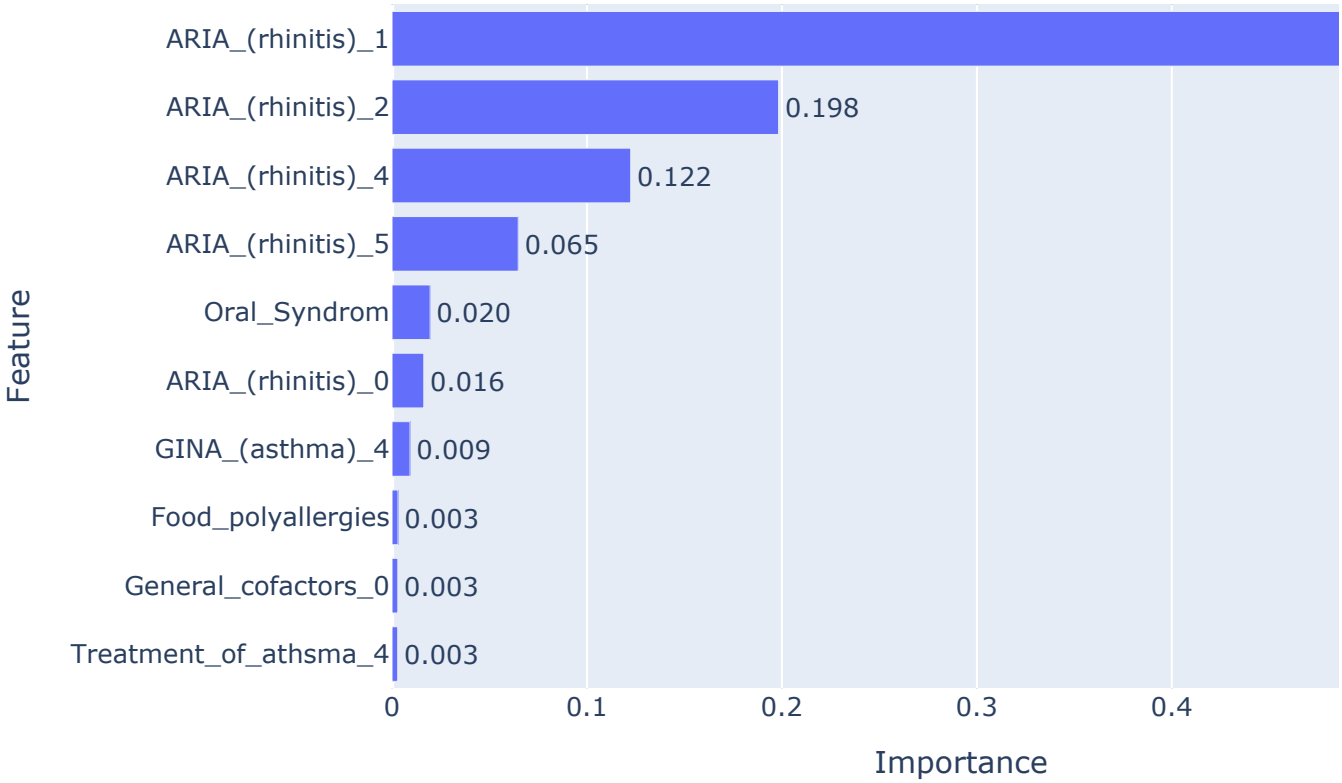
Importance


Top 10 Features pour la cible 'Type_of_Respiratory_Allergy_IGE_Mite
Feature
Food_polyallergies
Max_Animaux
0.088
Moyenne_Animaux
0.076
Age_of_onsets_1
0.051
General_cofactors_6
0.030
Oral_Syndrom
0.028
Treatment_of_atopic_dematitis_0
0.027
GINA_(asthma)_5
0.027
Age_of_onsets_5
0.026
Moyenne_Autres
0.026
0
0.05
0.1
0.15
0.2
Importance

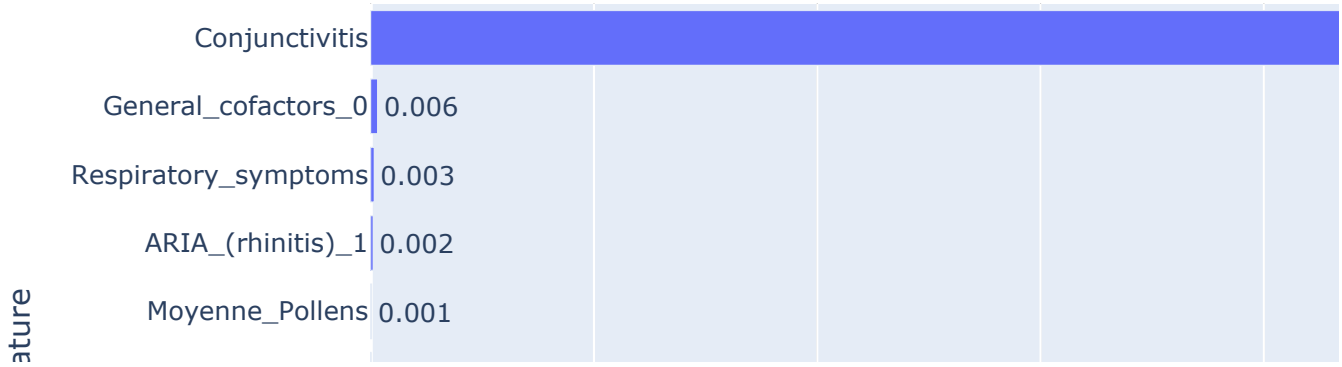

Top 10 Features pour la cible 'Type_of_Respiratory_Allergy_IGE_Mol
Feature
Max_Moisissures
Food_polyallergies
0.186
Treatment_of_rhinitis_2
0.082
Treatment_of_athsma_0
0.033
Age_of_onsets_5
0.031
Moyenne_Pollens
0.027
GINA_(asthma)_0
0.023
Age_of_onsets_4
0.020

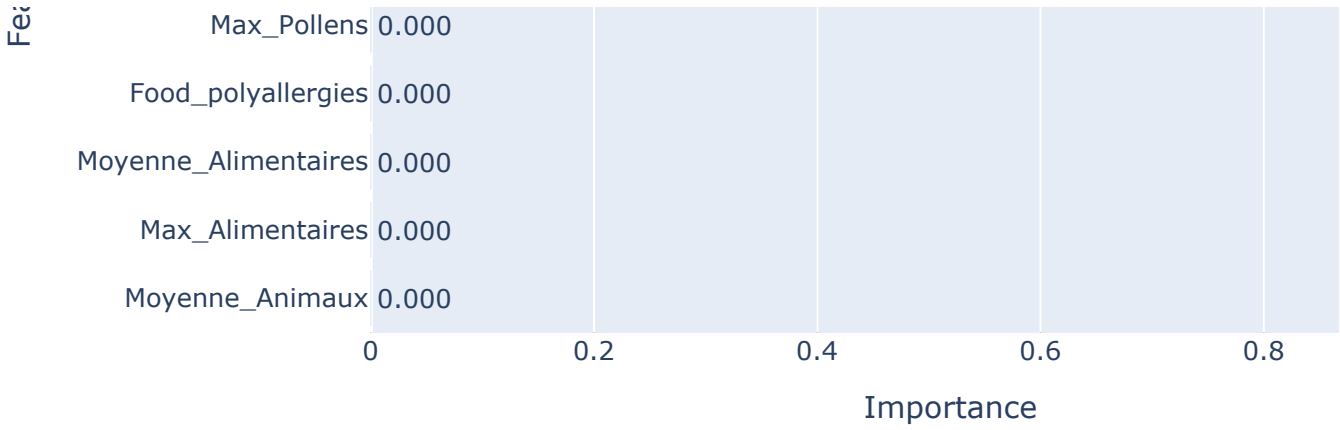


Top 10 Features pour la cible 'Type_of_Respiratory_Allergy_ARIA' (X

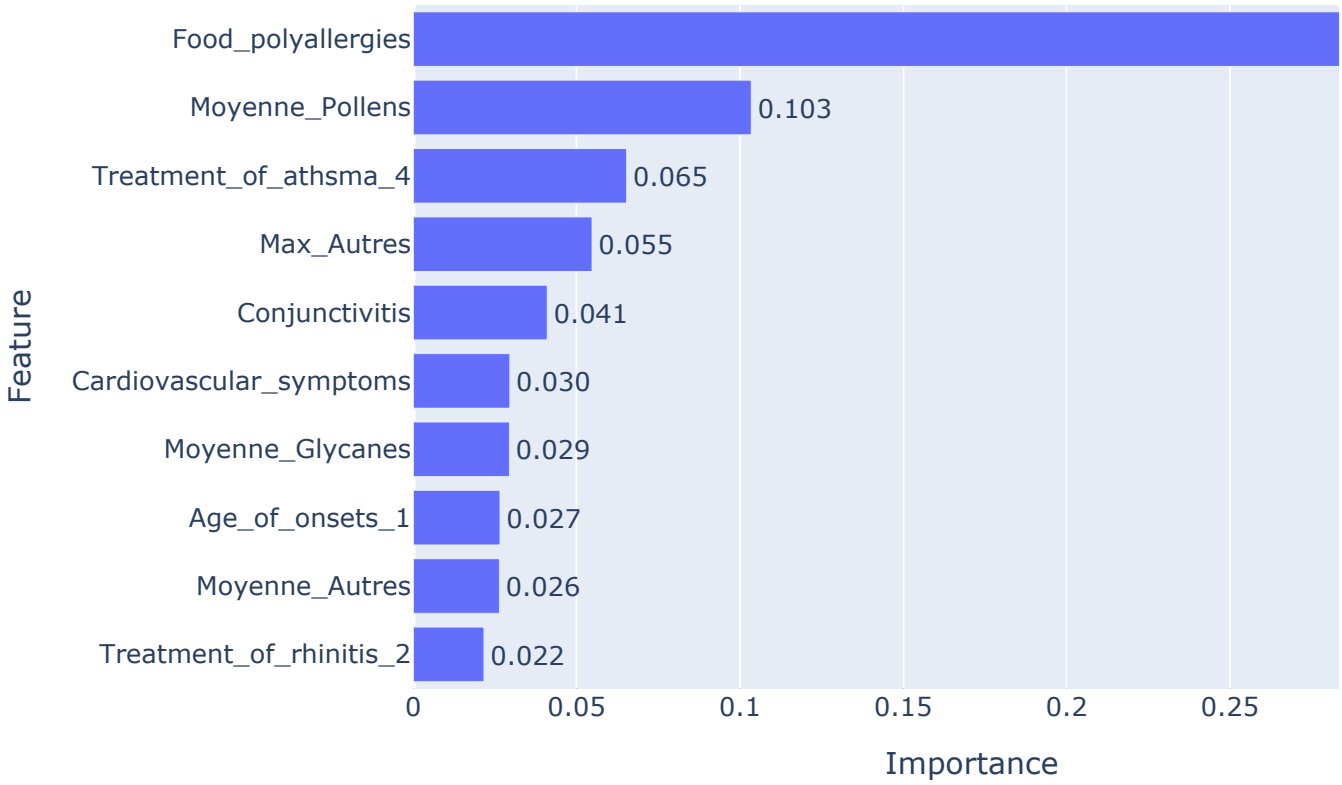


Top 10 Features pour la cible 'Type_of_Respiratory_Allergy_CONJ' (X

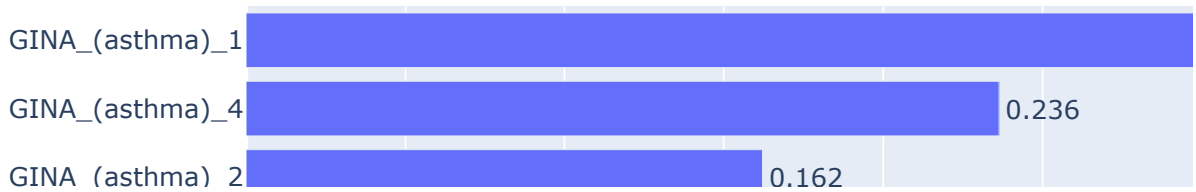


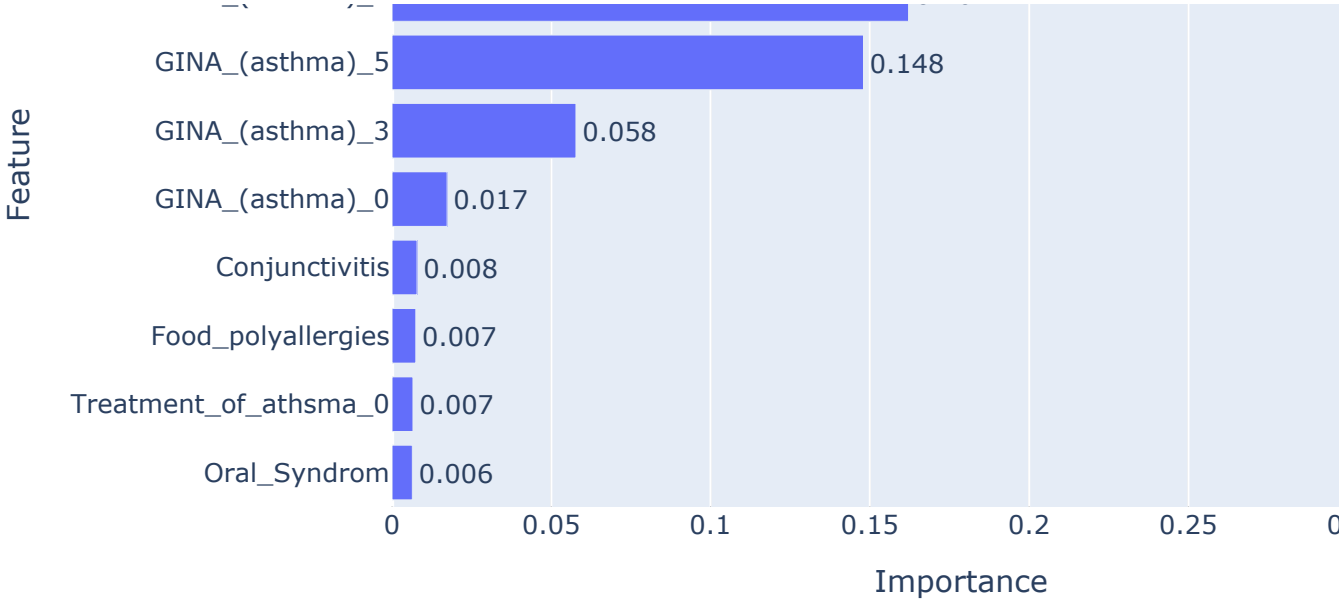


Top 10 Features pour la cible 'Type_of_Respiratory_Allergy_IGE_Poll

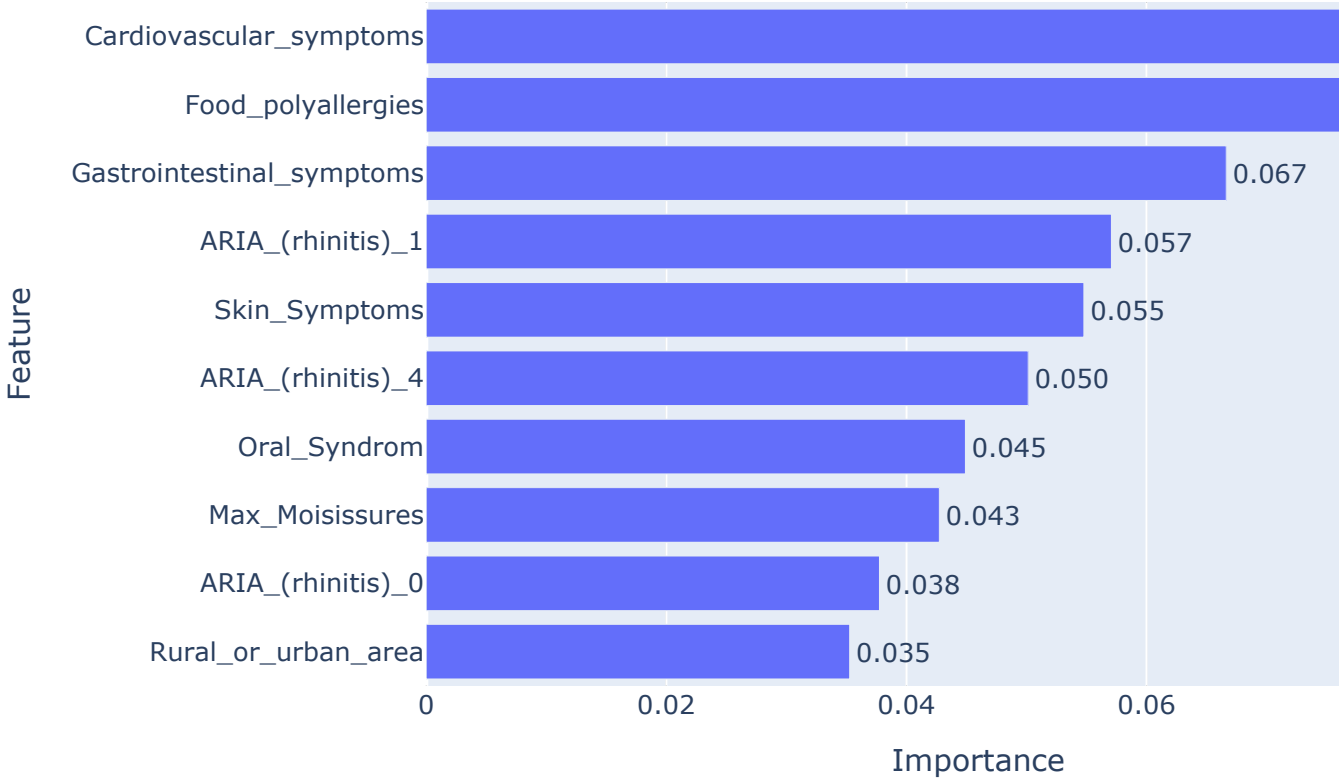


Top 10 Features pour la cible 'Type_of_Respiratory_Allergy_GINA' (X

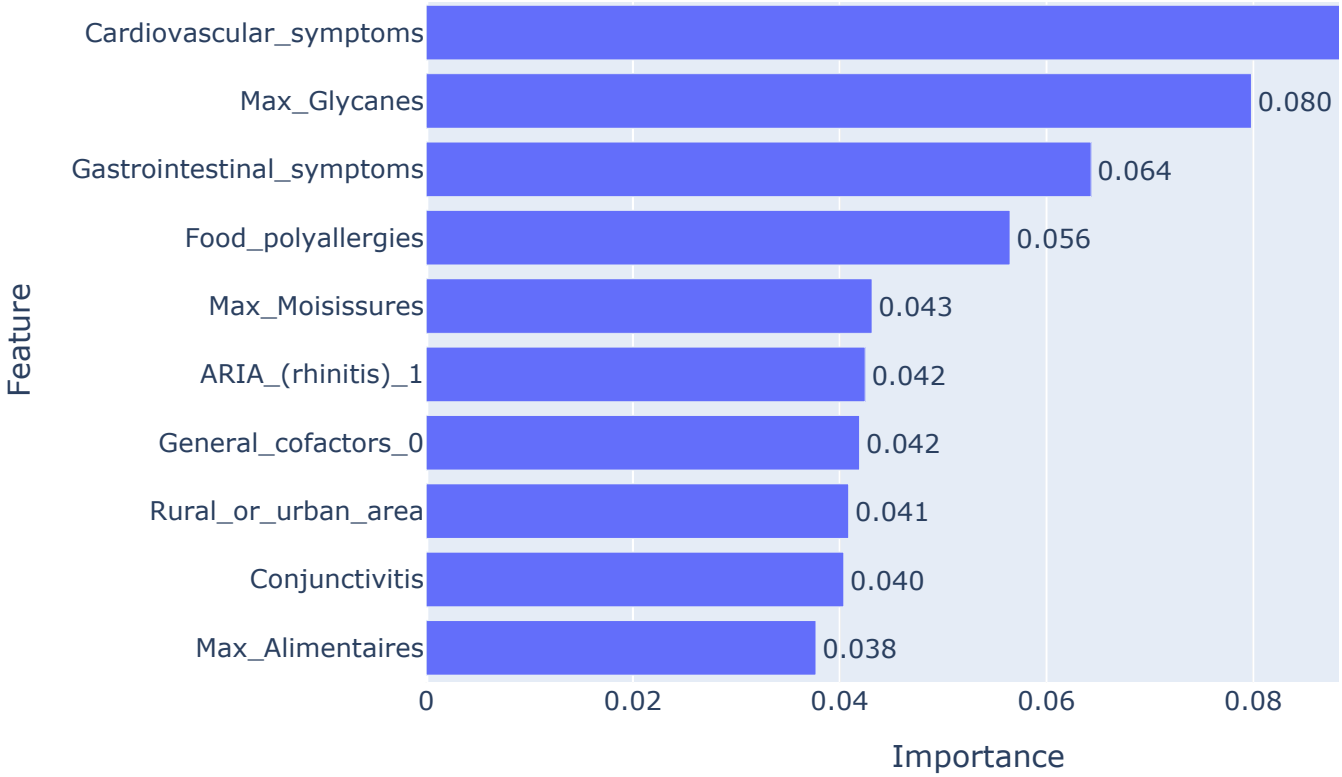




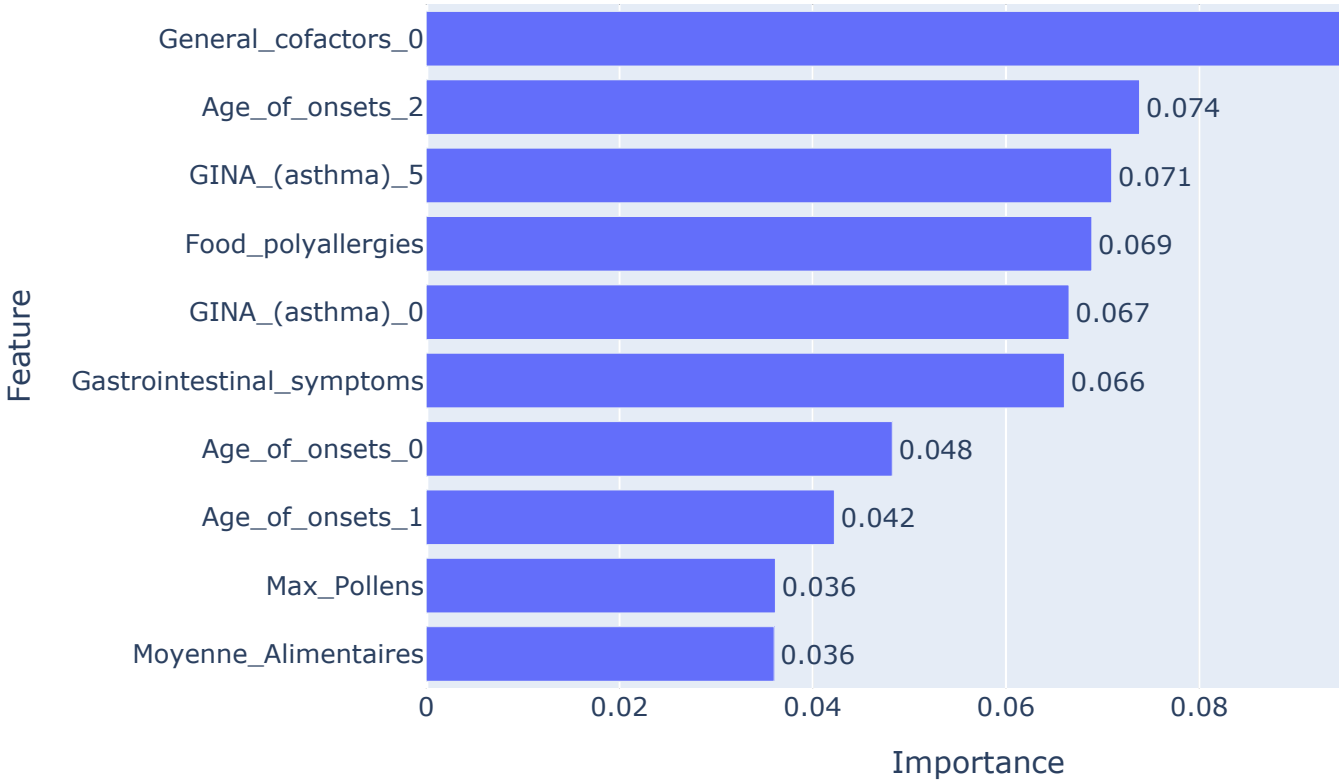
Top 10 Features pour la cible 'Type_of_Food_Allergy_Aromatics' (XG

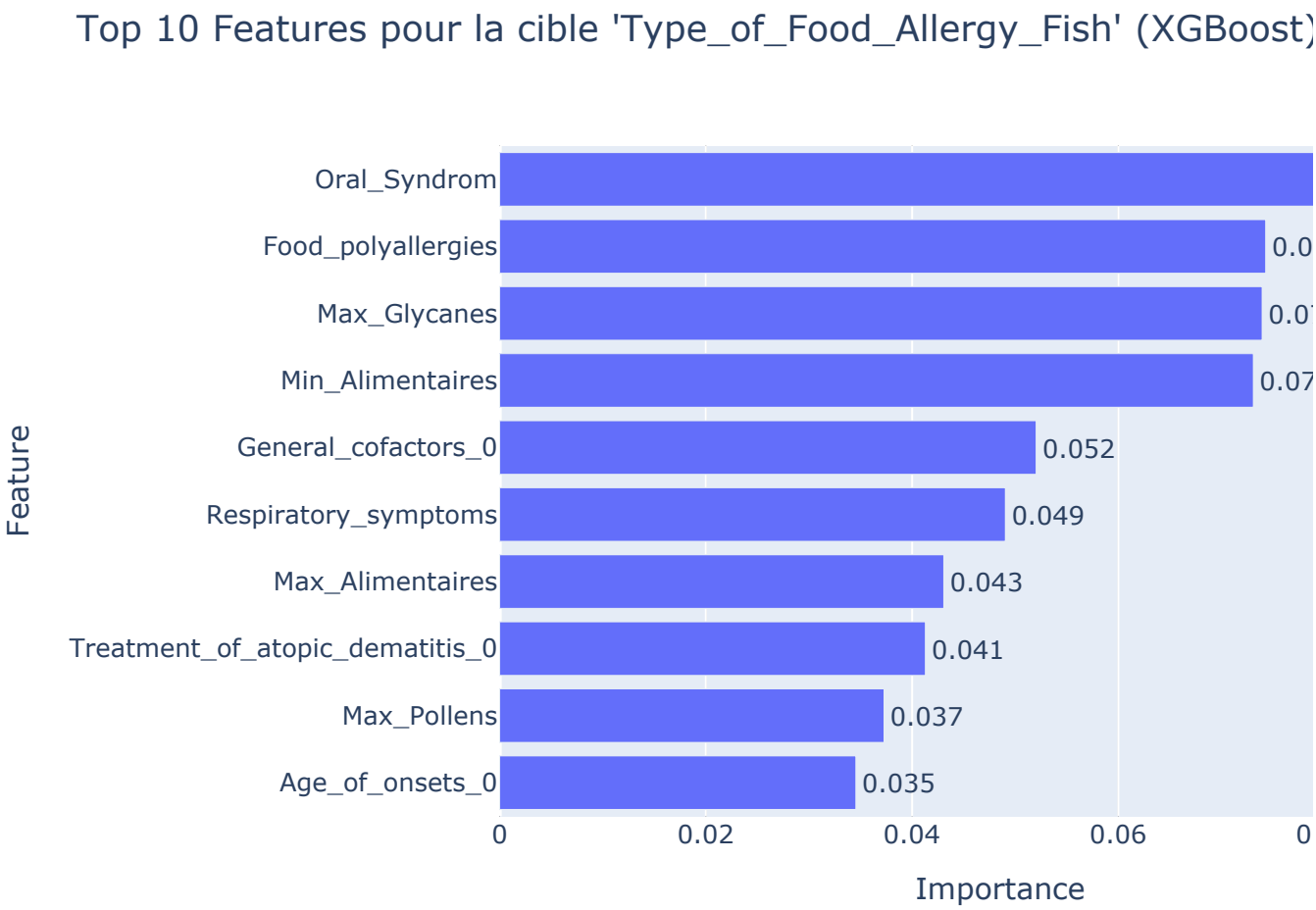


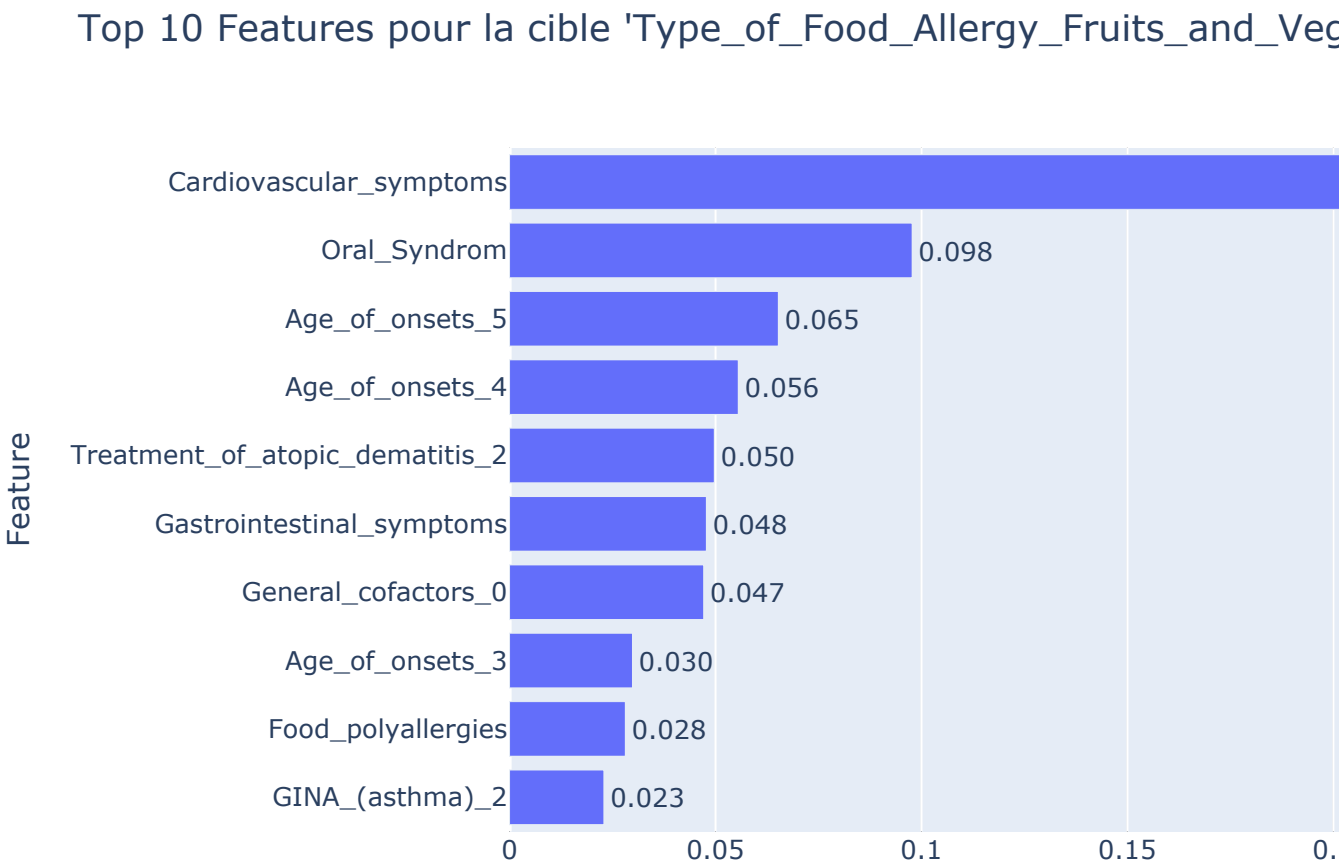
Top 10 Features pour la cible 'Type_of_Food_Allergy_Cereals_&_See



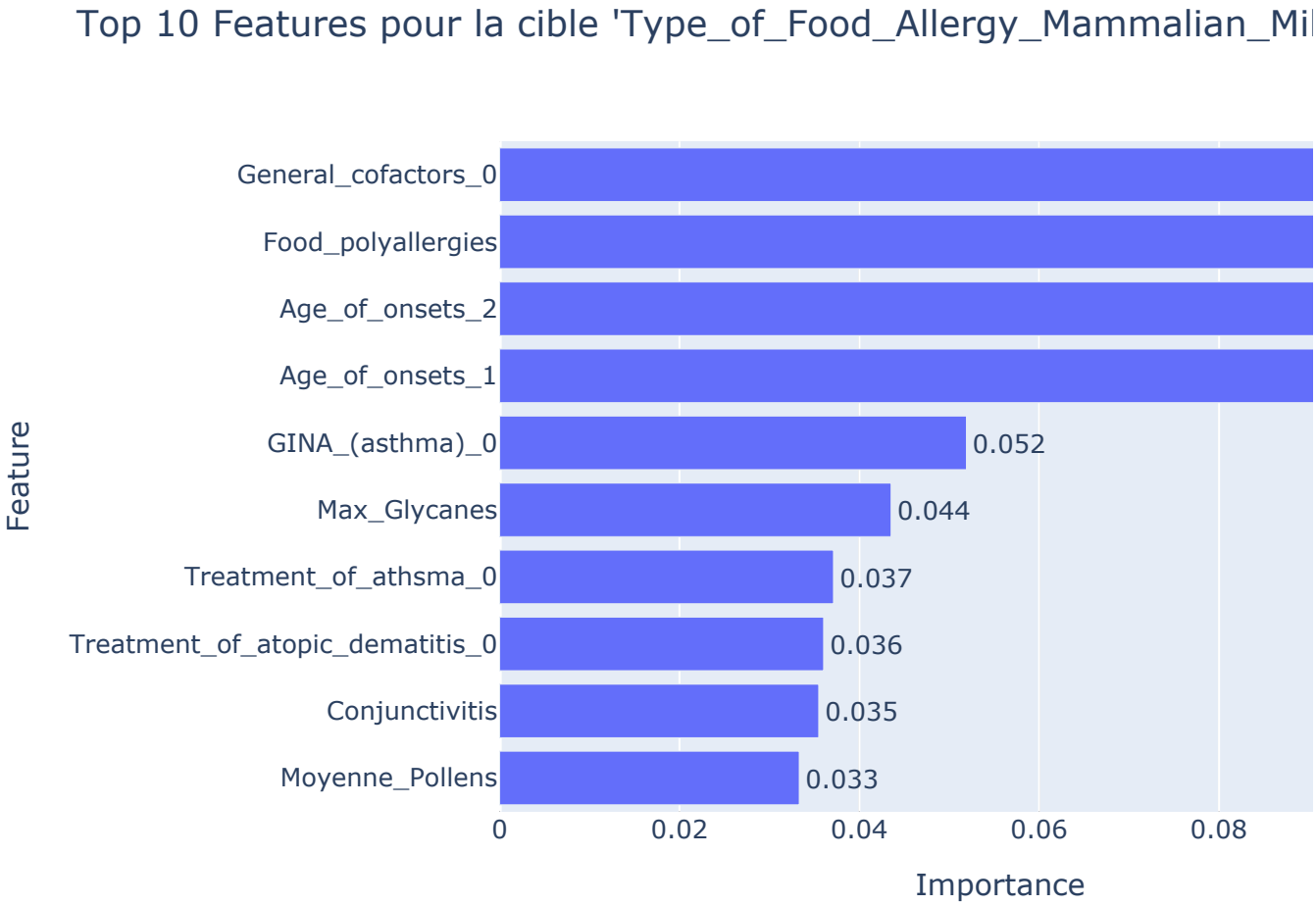
Top 10 Features pour la cible 'Type_of_Food_Allergy_Egg' (XGBoost)



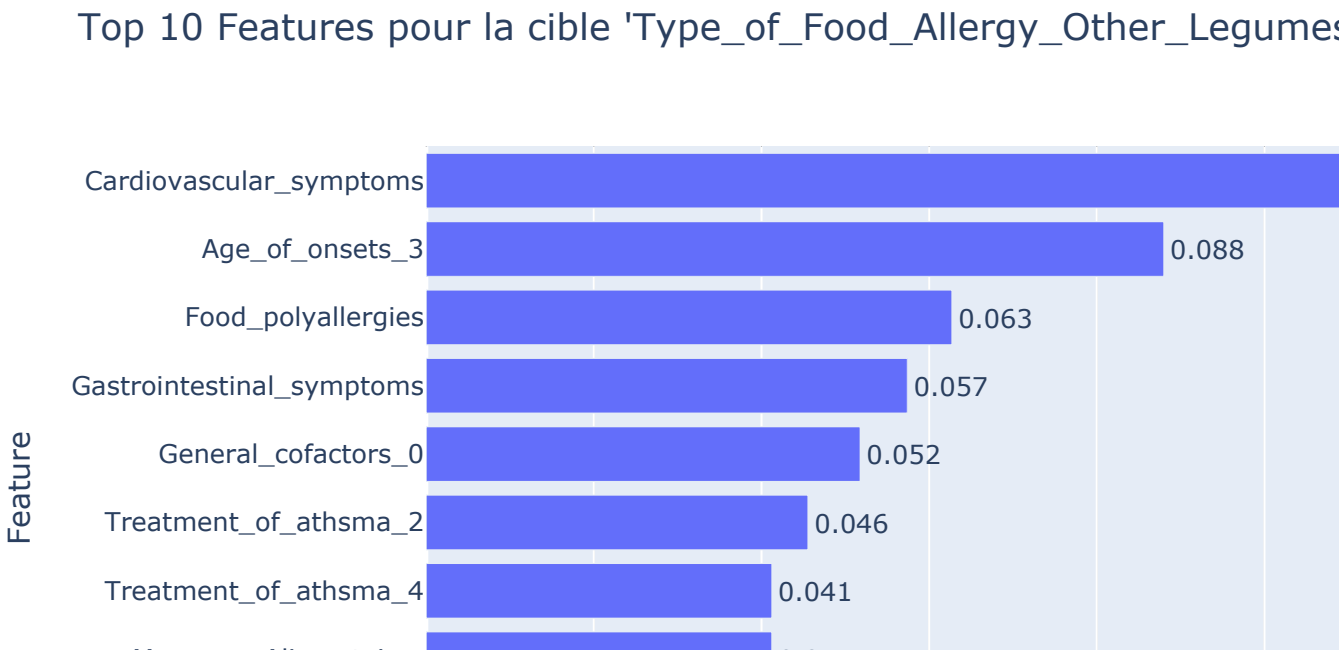

Top 10 Features pour la cible 'Type_of_Food_Allergy_Fish' (XGBoost)
Oral_Syndrom
Food_polyallergies
Max_Glycanes
Min_Alimentaires
General_cofactors_0
Respiratory_symptoms
Max_Alimentaires
Treatment_of_atopic_dematitis_0
Max_Pollens
Age_of_onsets_0
0.052
0.049
0.043
0.041
0.037
0.035
Feature
0
0.02
0.04
0.06
Importance

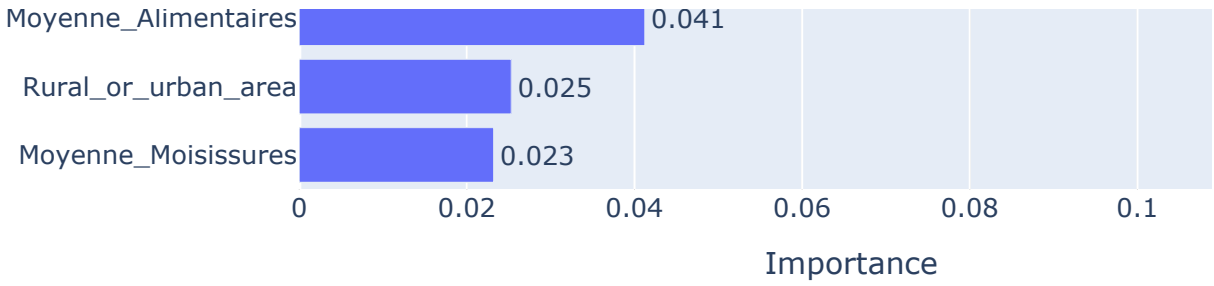

Top 10 Features pour la cible 'Type_of_Food_Allergy_Fruits_and_Veg
Cardiovascular_symptoms
Oral_Syndrom
Age_of_onsets_5
Age_of_onsets_4
Treatment_of_atopic_dematitis_2
Gastrointestinal_symptoms
General_cofactors_0
Age_of_onsets_3
Food_polyallergies
GINA_(asthma)_2
0.098
0.065
0.056
0.050
0.048
0.047
0.030
0.028
0.023
Feature
0
0.05
0.1
0.15

Importance

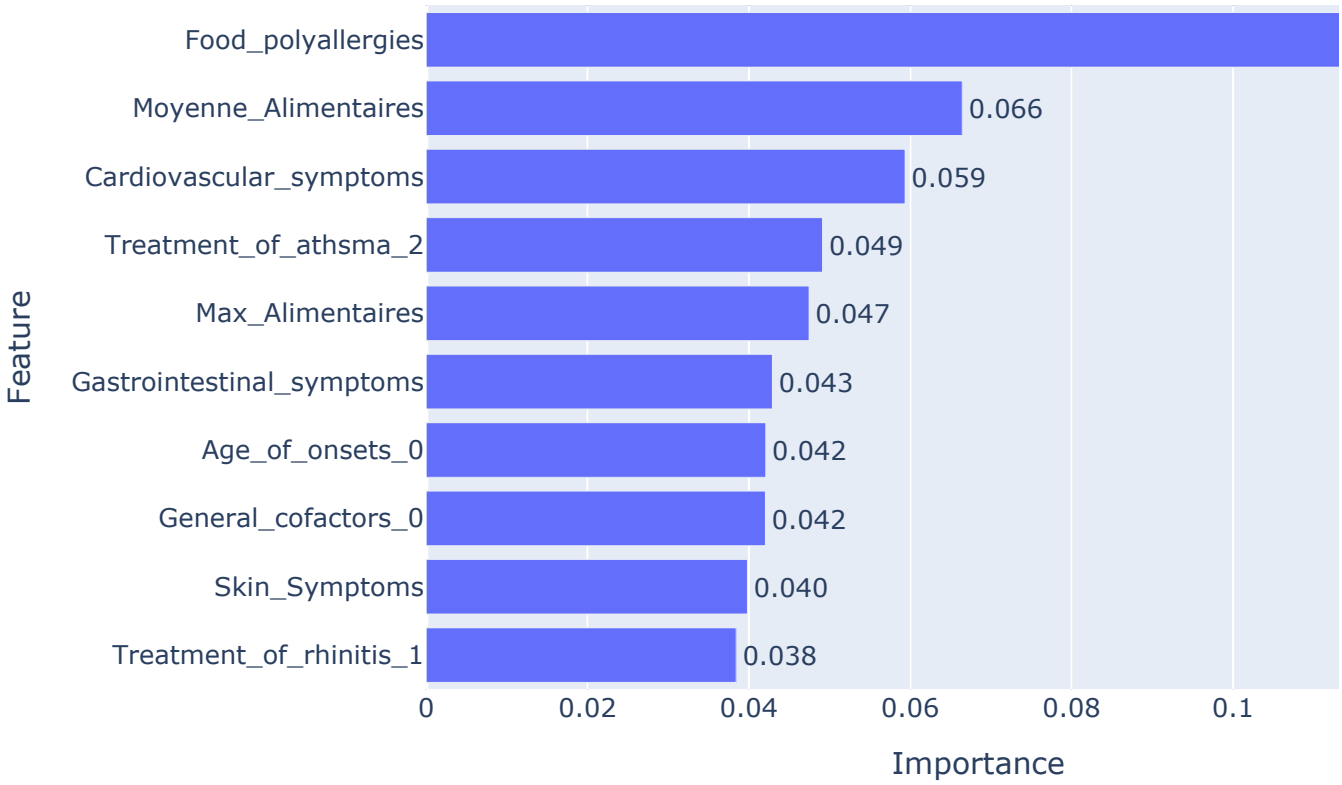


⚠️ Target 'Type_of_Food_Allergy_Oral_Syndrom' contient une seule classe ([0

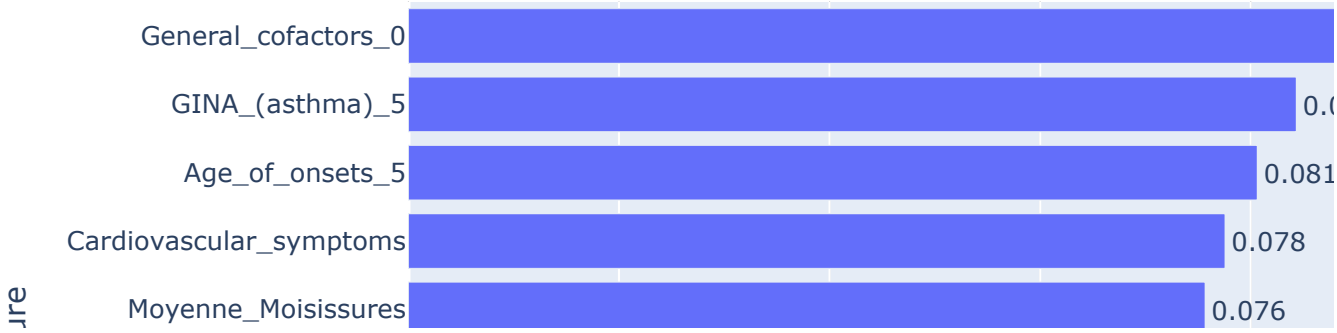


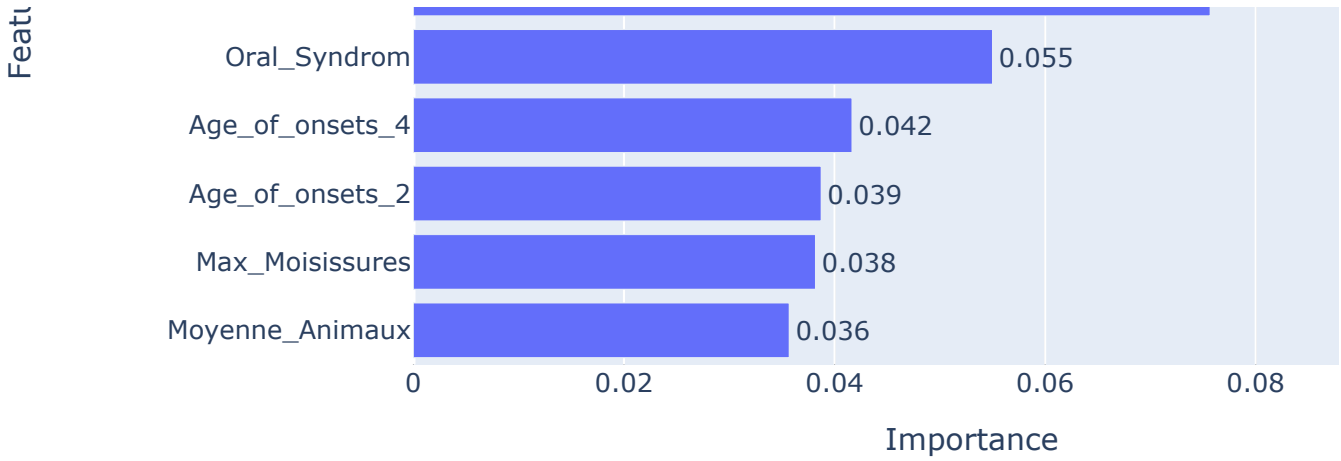


Top 10 Features pour la cible 'Type_of_Food_Allergy_Peanut' (XGBoo

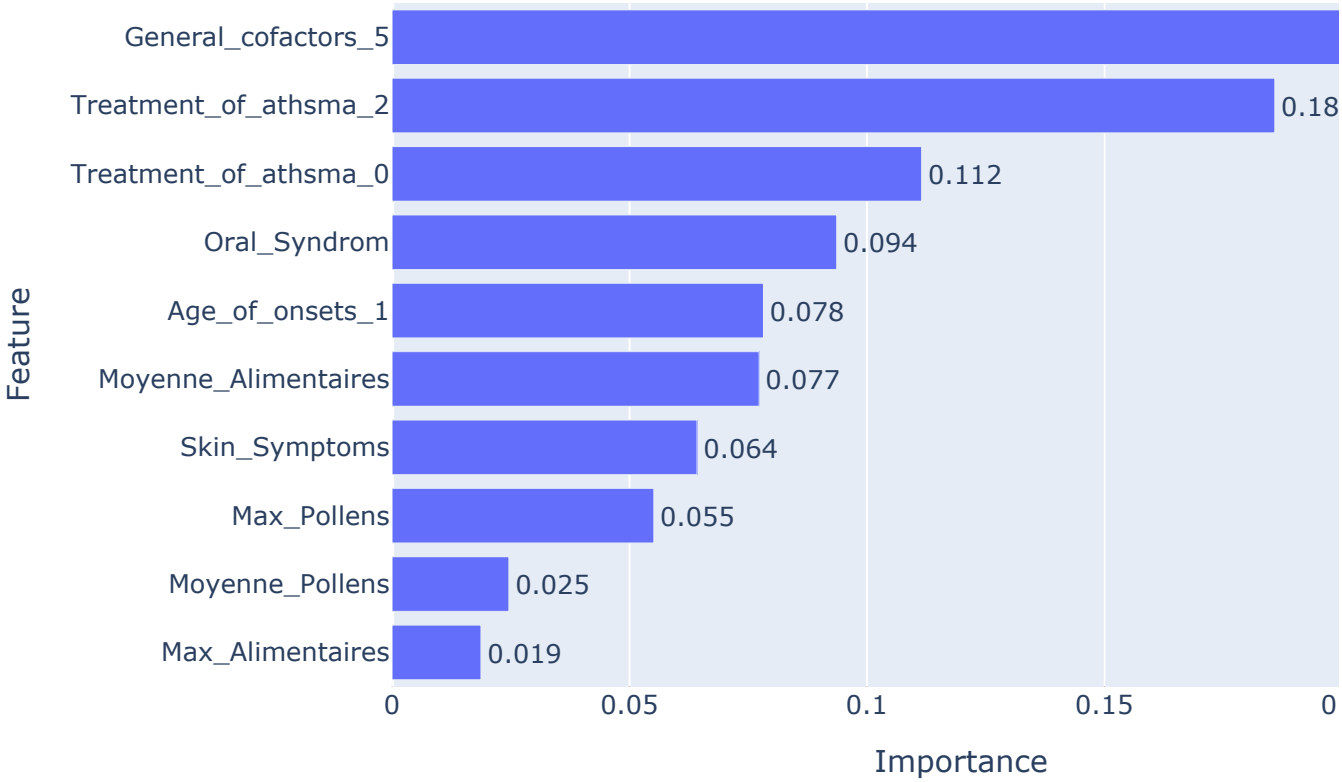


Top 10 Features pour la cible 'Type_of_Food_Allergy_Shellfish' (XGB

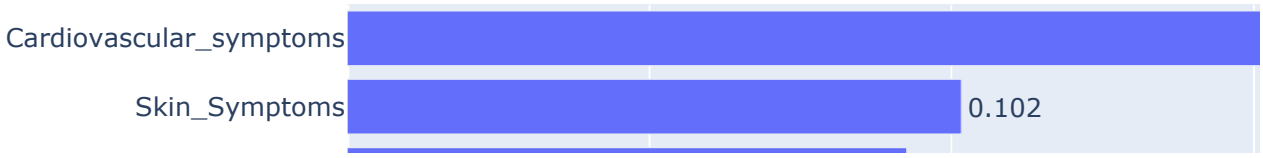


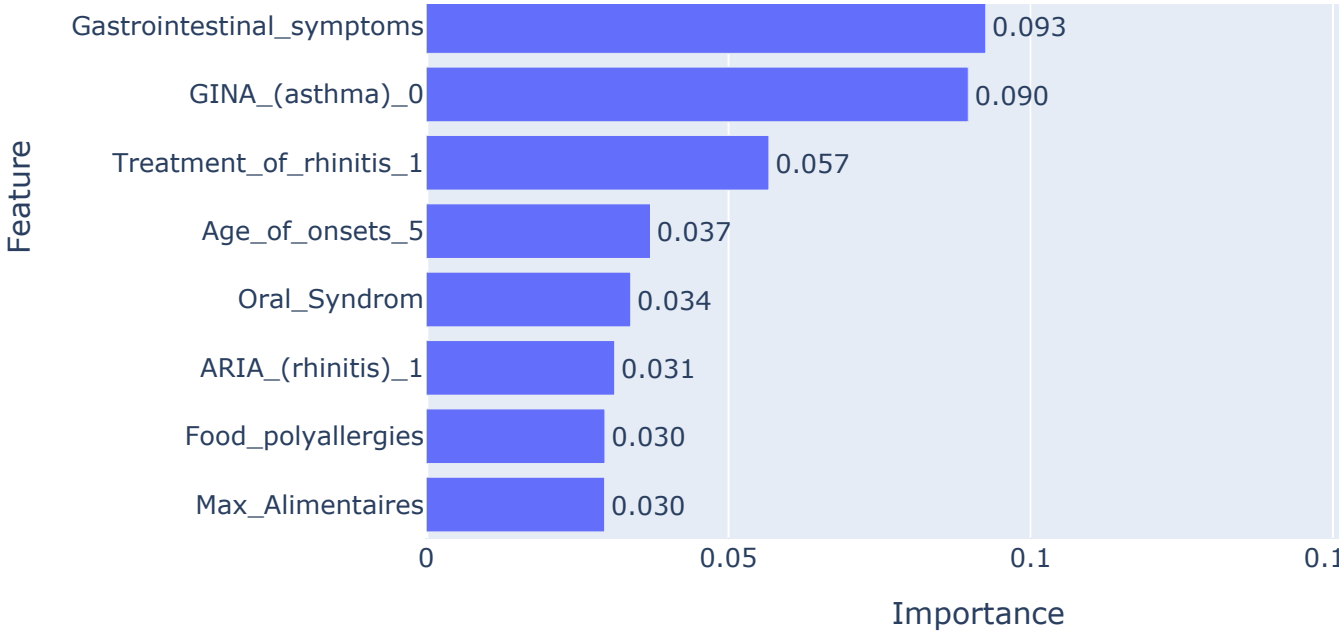


Top 10 Features pour la cible 'Type_of_Food_Allergy_TPO' (XGBoost)

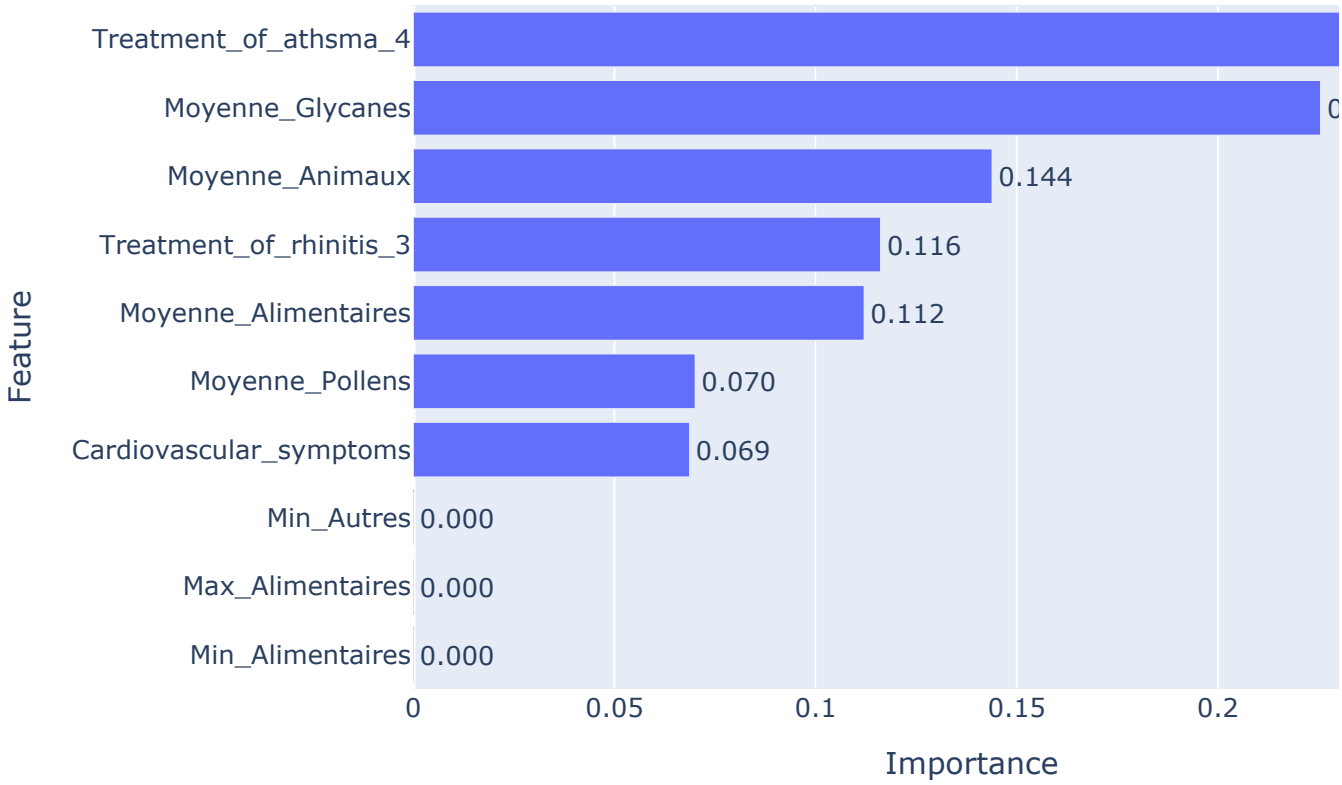


Top 10 Features pour la cible 'Type_of_Food_Allergy_Tree_Nuts' (XG





Top 10 Features pour la cible 'Type_of_Venom_Allergy_ATCD_Venom



⚠️ `Target 'Type_of_Venom_Allergy_IGE_Venom' contient une seule classe ([0])`

Start coding or generate with AI.