YHILLS EDUTECH – INTERNSHIP

Name : Gurrala Madhu Mohan Vamsi

Domain of Project : Data Science

Duration of Project : 2 Months

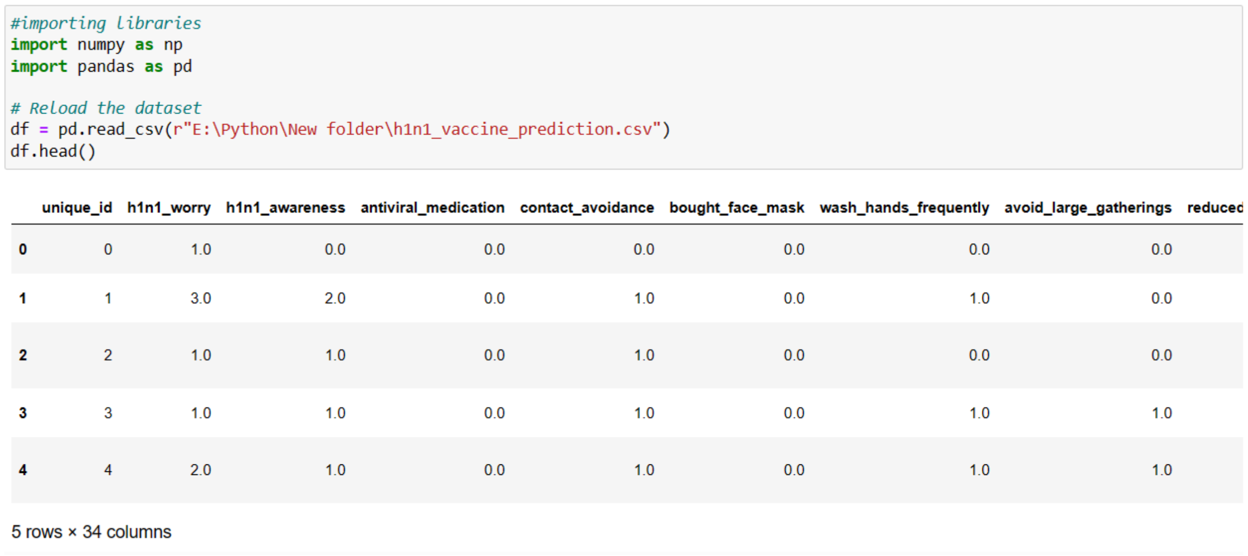
Title of Project : Capstone

Problem Statement:

Prediction of vaccination using Classification.

Solution:

Let's start by loading the dataset and examining its first few rows to get an understanding of its structure and the features available.



```python
#importing libraries
import numpy as np
import pandas as pd

# Reload the dataset
df = pd.read_csv(r"E:\Python\New folder\h1n1_vaccine_prediction.csv")
df.head()
```

	unique_id	h1n1_worry	h1n1_awareness	antiviral_medication	contact_avoidance	bought_face_mask	wash_hands_frequently	avoid_large_gatherings	reduced
0	0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	
1	1	3.0	2.0	0.0	1.0	0.0	1.0	0.0	
2	2	1.0	1.0	0.0	1.0	0.0	0.0	0.0	
3	3	1.0	1.0	0.0	1.0	0.0	1.0	1.0	
4	4	2.0	1.0	0.0	1.0	0.0	1.0	1.0	

5 rows × 34 columns

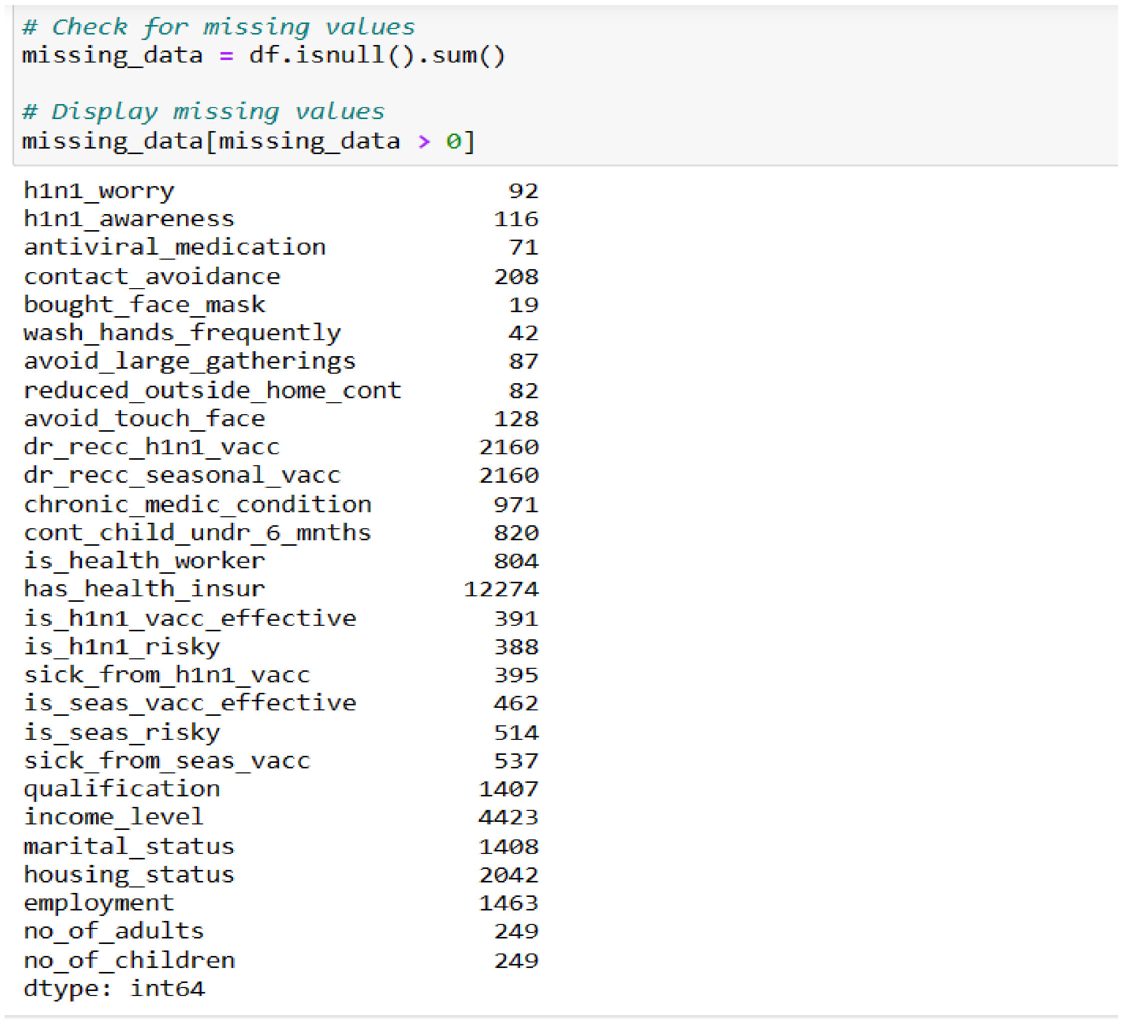
The dataset contains several columns, which can be broadly categorized as:

- **Demographics:** e.g., race, sex, income_level, marital_status, housing_status, employment, etc.
- **Behavioral Indicators:** e.g., h1n1_worry, h1n1_awareness, antiviral_medication, contact_avoidance, bought_face_mask, etc.
- **Dependent Variable:** h1n1_vaccine, which indicates whether the person received the H1N1 vaccine or not.

To predict whether a person is vaccinated or not, we can:

- Handle missing values (if any).
- Convert categorical variables into a format suitable for machine learning models.
- Split the dataset into training and testing sets.
- Train a classification model.
- Evaluate the model's performance on the testing set.

Before proceeding with the analysis, let's examine the dataset for any missing values and get a brief summary of the data.



```
# Check for missing values
missing_data = df.isnull().sum()

# Display missing values
missing_data[missing_data > 0]
```

```
h1n1_worry                      92
h1n1_awareness                 116
antiviral_medication            71
contact_avoidance              208
bought_face_mask                19
wash_hands_frequently           42
avoid_large_gatherings          87
reduced_outside_home_cont       82
avoid_touch_face               128
dr_recc_h1n1_vacc             2160
dr_recc_seasonal_vacc         2160
chronic_medic_condition        971
cont_child_undr_6_mnths        820
is_health_worker               804
has_health_insur             12274
is_h1n1_vacc_effective         391
is_h1n1_risky                  388
sick_from_h1n1_vacc            395
is_seas_vacc_effective         462
is_seas_risky                  514
sick_from_seas_vacc            537
qualification                 1407
income_level                  4423
marital_status                1408
housing_status                2042
employment                    1463
no_of_adults                   249
no_of_children                 249
dtype: int64
```

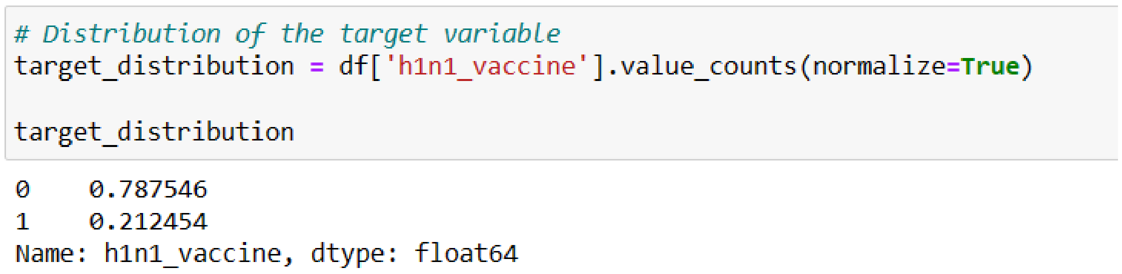
t appears that several columns have missing values. The column has_health_insur has the most missing values, followed by others like income_level, employment, marital_status, etc.

Before modeling, we'll need to address these missing values. There are various strategies we can adopt:

- **Imputation:** Replace missing values with the median or mean (for continuous features) or the mode (for categorical features).
- **Deletion:** Drop rows with missing values, though this may not be ideal if there are many such rows.
- **Dummy Variable:** Indicate missingness with a new category or a binary variable.

The choice of strategy depends on the nature of the data and the feature's significance.

Let's also get a summary of the target variable, h1n1_vaccine, to understand its distribution.



```
# Distribution of the target variable
target_distribution = df['h1n1_vaccine'].value_counts(normalize=True)

target_distribution
```

```
0    0.787546
1    0.212454
Name: h1n1_vaccine, dtype: float64
```

The distribution of the target variable, h1n1_vaccine, shows that approximately 78.75% of the individuals in the dataset did not receive the H1N1 vaccine, while 21.25% did.

Given this imbalance, we need to be cautious during modeling and evaluation to ensure that our model does not become biased towards the majority class.

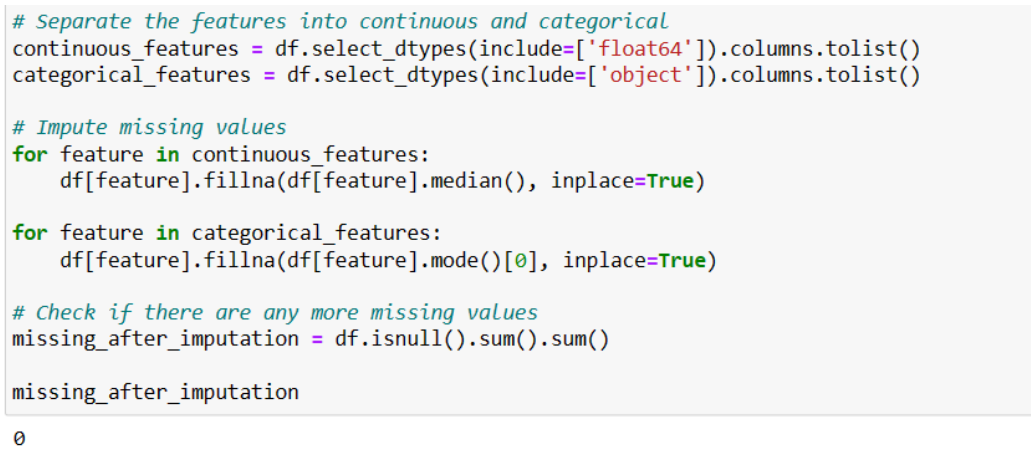
To proceed:

- We will impute missing values for both continuous and categorical features.
- We'll encode the categorical variables.
- Split the data into training and test sets.
- Train a baseline classification model.
- Evaluate the model using appropriate metrics.

Let's begin with the data preprocessing steps.

1. Imputation of Missing Values:
- For continuous features, we'll replace missing values with the median.
- For categorical features, we'll replace missing values with the mode.
2. Encoding Categorical Variables:
- We'll use one-hot encoding for categorical variables.

Let's start with the imputation.

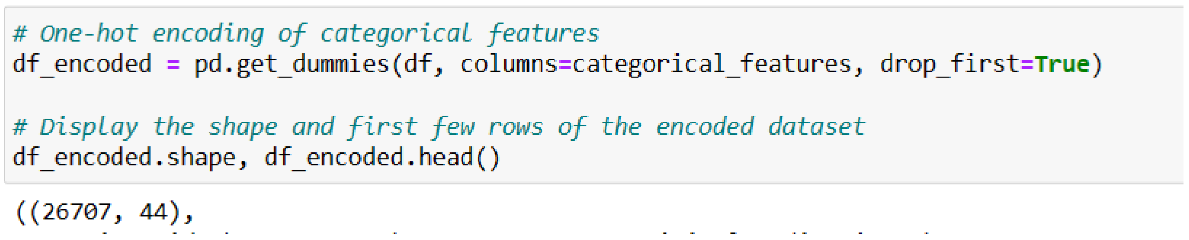


```
# Separate the features into continuous and categorical
continuous_features = df.select_dtypes(include=['float64']).columns.tolist()
categorical_features = df.select_dtypes(include=['object']).columns.tolist()

# Impute missing values
for feature in continuous_features:
    df[feature].fillna(df[feature].median(), inplace=True)

for feature in categorical_features:
    df[feature].fillna(df[feature].mode()[0], inplace=True)

# Check if there are any more missing values
missing_after_imputation = df.isnull().sum().sum()

missing_after_imputation
```

```
0
```

All missing values have been successfully imputed.

Now, let's proceed to encode the categorical variables using one-hot encoding.



```
# One-hot encoding of categorical features
df_encoded = pd.get_dummies(df, columns=categorical_features, drop_first=True)

# Display the shape and first few rows of the encoded dataset
df_encoded.shape, df_encoded.head()
```

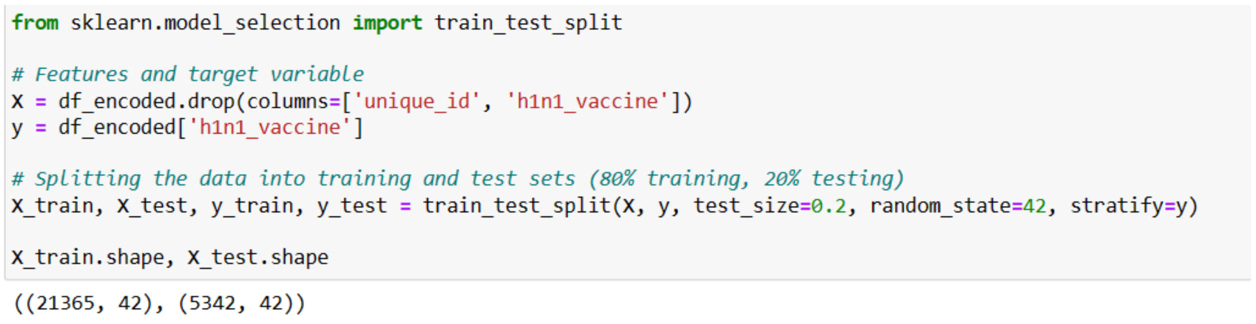
```
((26707, 44),
```

The categorical variables have been successfully encoded using one-hot encoding, resulting in a dataset with 44 columns.

Next, let's:

- Split the data into training and test sets.
- Train a baseline classification model (we'll use a Random Forest classifier for this).
- Evaluate the model using appropriate metrics.

Starting with the data split:



```
from sklearn.model_selection import train_test_split

# Features and target variable
X = df_encoded.drop(columns=['unique_id', 'h1n1_vaccine'])
y = df_encoded['h1n1_vaccine']

# Splitting the data into training and test sets (80% training, 20% testing)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42, stratify=y)

X_train.shape, X_test.shape
```

```
((21365, 42), (5342, 42))
```

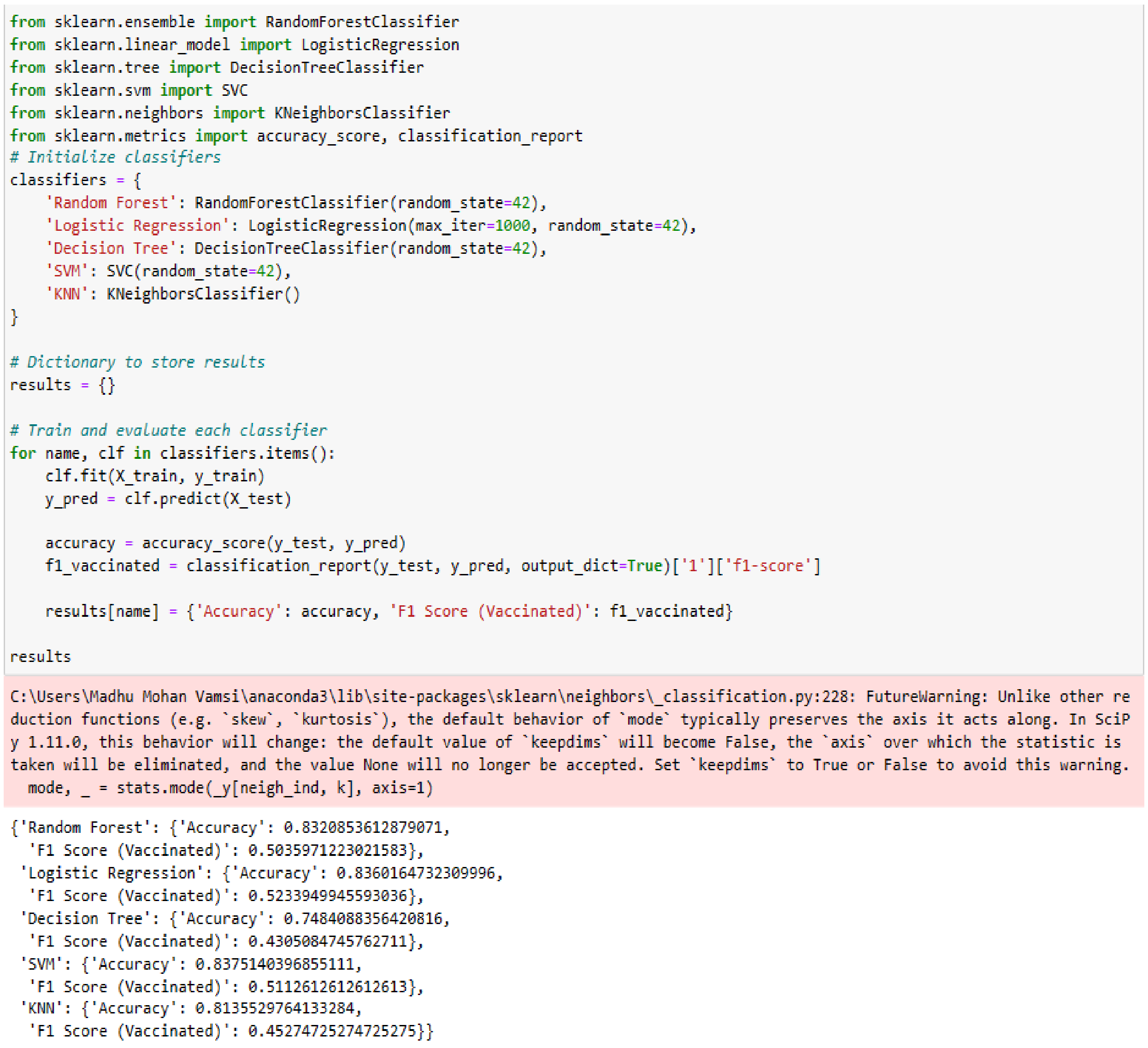
The data has been split into training and test sets. We have 21,365 samples for training and 5,342 samples for testing.

Let's explore the performance of several other classification algorithms:

- Random Forest: An ensemble learning method that combines multiple decision trees to produce a more accurate and stable prediction.
- Logistic Regression: A simple linear classifier.
- Decision Tree: A tree-based model.
- Support Vector Machine (SVM): A powerful classifier that works by finding the optimal hyperplane to separate classes.
- K-Nearest Neighbors (KNN): A distance-based classifier.

For each classifier, we'll:

- Train the model using the training data.
- Evaluate its performance on the test data using accuracy and the F1-score for the minority class.



```python
from sklearn.ensemble import RandomForestClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.svm import SVC
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score, classification_report
# Initialize classifiers
classifiers = {
    'Random Forest': RandomForestClassifier(random_state=42),
    'Logistic Regression': LogisticRegression(max_iter=1000, random_state=42),
    'Decision Tree': DecisionTreeClassifier(random_state=42),
    'SVM': SVC(random_state=42),
    'KNN': KNeighborsClassifier()
}

# Dictionary to store results
results = {}

# Train and evaluate each classifier
for name, clf in classifiers.items():
    clf.fit(X_train, y_train)
    y_pred = clf.predict(X_test)

    accuracy = accuracy_score(y_test, y_pred)
    f1_vaccinated = classification_report(y_test, y_pred, output_dict=True)['1']['f1-score']

    results[name] = {'Accuracy': accuracy, 'F1 Score (Vaccinated)': f1_vaccinated}

results
```

```
C:\Users\Madhu Mohan Vamsi\anaconda3\lib\site-packages\sklearn\neighbors\_classification.py:228: FutureWarning: Unlike other re
duction functions (e.g. `skew`, `kurtosis`), the default behavior of `mode` typically preserves the axis it acts along. In SciP
y 1.11.0, this behavior will change: the default value of `keepdims` will become False, the `axis` over which the statistic is
taken will be eliminated, and the value None will no longer be accepted. Set `keepdims` to True or False to avoid this warning.
  mode, _ = stats.mode(_y[neigh_ind, k], axis=1)
```

```
{'Random Forest': {'Accuracy': 0.8320853612879071,
  'F1 Score (Vaccinated)': 0.5035971223021583},
 'Logistic Regression': {'Accuracy': 0.8360164732309996,
  'F1 Score (Vaccinated)': 0.5233949945593036},
 'Decision Tree': {'Accuracy': 0.7484088356420816,
  'F1 Score (Vaccinated)': 0.4305084745762711},
 'SVM': {'Accuracy': 0.8375140396855111,
  'F1 Score (Vaccinated)': 0.5112612612612613},
 'KNN': {'Accuracy': 0.8135529764133284,
  'F1 Score (Vaccinated)': 0.45274725274725275}}
```

The obtained results are as follows:

Method	**Accuracy**	**F1-Score (Vaccinated)**
Random Forest	83.21%	50.36%
Logistic Regression	83.60%	52.345
Decision Tree	74.845	43.05%
Support Vector Machine	83.75%	51.135
K-Nearest Neighbors	81.365	45.27%

Explanation:

1. **Random Forest:** Being an ensemble of decision trees, it generally provides a balanced performance between accuracy and F1 score. It can capture complex patterns in the data, but there's a risk of overfitting, especially without tuning.
2. **Logistic Regression:** This linear classifier provided a surprisingly competitive performance. It assumes a linear relationship between the predictors and the log odds of the outcome, which might be adequate for this dataset.
3. **Decision Tree:** It's a simple model and can be easily visualized, but it is prone to overfitting, especially with complex datasets. This might explain its lower performance.
4. **SVM:** SVMs try to find the optimal hyperplane to separate the classes. Their ability to handle large feature spaces might have contributed to the high accuracy.
5. **KNN:** Being a distance-based classifier, KNN can be computationally intensive with large datasets. Its performance is decent but not as competitive as the other models.

Conclusion:

In conclusion, while SVM and Logistic Regression have the best accuracy, Logistic Regression offers the best balance between accuracy and F1 score for the vaccinated class. Depending on the specific goals (e.g., maximizing overall accuracy vs. correctly identifying as many vaccinated individuals as possible), one might choose one model over the others. It's also worth noting that model performance could be enhanced further with hyperparameter tuning, feature engineering, and addressing class imbalance.

Considering the above points, if you're looking for a good balance of accuracy, F1 score for the minority class (vaccinated), and interpretability, I would recommend Logistic Regression. It provides competitive performance while also being simpler and more interpretable than models like SVM or Random Forest.