Import Relevant Libraries

In [ ]:
```
from __future__ import annotations
import pandas as pd
import numpy as np
from random import choice, seed as set_seed
```

Implementasi

In [ ]:
```
class RDR:
    def __init__(self, general_case: str | None = None) -> None:
        self.general_case: str | None = general_case
        self._root: RDR.RDRNode = self.RDRNode(None, None)

    def fit(self, dataset: pd.DataFrame, labels: pd.Series, ignore_general: bool
        # Set seed for regularity in testing
        if seed is not None:
            set_seed(seed)

        if self.general_case is None:
            # Asign a general case based on labels
            values, counts = np.unique(labels, return_counts=True)
            mode_label = values[np.argmax(counts)]
            self.general_case = mode_label if type(
                mode_label) is str else str(mode_label)
            self._root.set_general_case(self.general_case)

        # build the mask to isolate cornerstone cases
        if ignore_general:
            mask = labels.values.astype(str) != self.general_case
            dataset_cs: pd.DataFrame = dataset[mask].astype(bool)
            labels_cs: pd.Series = labels[mask]
        else:
            dataset_cs = dataset.astype(bool)
            labels_cs = labels

        # build the knowledge tree
        for data, label in zip(dataset_cs.values, labels_cs.values):
            features = [dataset_cs.columns[i] for i in np.where(data)[0]]
            self._root.ripple_down(features, label)

    def predict(self, dataset: pd.DataFrame) -> pd.Series:
        predictions = []
        # Iterate over each row in the dataset
        for data in dataset.astype(bool).values:
            # predict the conclusion of the row
            features = [dataset.columns[i] for i in np.where(data)[0]]
            conclusion = self._root.predict(features)
            # append the conclusion to the predictions
            predictions.append(
                conclusion if conclusion is not None else self.general_case)
        return pd.Series(data=predictions)

    def add_new_cornerstone(self, data, new_conclusion, new_prerequisites=None)
```

```
        # Add the new cornerstone to the RDR tree
        self._root.ripple_down(data,
                               new_conclusion, new_prerequisites)

    def __str__(self) -> str:
        return f"General Case: {self.general_case}\n" + self._root.visualize()

    class RDRNode:
        def __init__(self, prerequisites: list | None, conclusion: str | None) -
            """
            prerequisites: list of features that must be met before this rule ca
            conclusion: the conclusion of this rule
            left: the left child of this rule, applies when the prerequisites ar
            right: the right child of this rule, applies when the prerequisites
            """
            self._prerequisites: list | None = prerequisites
            self._conclusion: str | None = conclusion if type(
                conclusion) is str or conclusion is None else str(conclusion)
            self._left: RDR.RDRNode | None = None
            self._right: RDR.RDRNode | None = None

        def set_general_case(self, general_case: str) -> None:
            self._general_case = general_case

        def _add_node(self, features, node) -> None:
            # If the node's prerequisites are met
            if self.is_fulfilled(features):
                if self._right is None:
                    self._right = node
                else:
                    self._right._add_node(features, node)
            else:
                if self._left is None:
                    self._left = node
                else:
                    self._left._add_node(features, node)

        def is_fulfilled(self, features) -> bool:
            # return true if all the prerequisites are met
            if self._prerequisites is not None:
                for prerequisite in self._prerequisites:
                    if prerequisite not in features:
                        return False
            return True

        def _traverse_to_conclusion_leaf(self, features) -> RDR.RDRNode | None:
            # Traverse to the conclusion leaf
            if self.is_fulfilled(features):
                conclusion_node = self
                if self._right is not None:
                    right_conclusion = self._right._traverse_to_conclusion_leaf(
                    if right_conclusion is not None:
                        conclusion_node = right_conclusion
                return conclusion_node
            elif self._left is not None:
                return self._left._traverse_to_conclusion_leaf(features)
            return None

        def _contradict(self, features, new_conclusion, new_prerequisites=None)
            # If there are no new prerequisites, generate some
```

```
            if new_prerequisites is None:
                # If there are prerequisites, use them as a base
                if self._prerequisites is not None:
                    new_prerequisites = [p for p in self._prerequisites]
                    possible_features = [f for f in features if f not in new_pre
                    if len(possible_features) != 0:
                        # append the random feature that is not already in the p
                        new_prerequisites.append(choice(possible_features))
                    else:
                        # Didn't find a new feature to append, so remove the fea
                        duplicate_features = [f for f in features if f in new_pr
                        # remove the random feature that is already in the prere
                        new_prerequisites.remove(choice(duplicate_features))
                else:
                    # Append Random Feature
                    new_prerequisites = []
                    new_prerequisites.append(choice(features))

            # Add the new cornerstone to the RDR tree
            self._add_node(features, RDR.RDRNode(new_prerequisites, new_conclusi

        def _manifest(self, features, new_conclusion, new_prerequisites=None) ->
            # If there are no new prerequisites, generate some
            if new_prerequisites is None:
                # Since this is a manifestation, the new prerequisites should be
                # This is because we triggered the else branch, thus prerequisit
                # Can choose any feature to be the new prerequisite as long as p
                # Easier Just to use a random feature from the features list tha
                new_prerequisites = []
                possible_features = [f for f in features if f not in new_prerequ
                if len(possible_features) != 0:
                    # append the random feature that is not already in the prere
                    new_prerequisites.append(choice(possible_features))
                else:
                    # all features are already in the prerequisites, but prerequ
                    # in this case, just use all the features as the prerequisit
                    new_prerequisites = [f for f in features]

            self._add_node(features, RDR.RDRNode(new_prerequisites, new_conclusi


        def ripple_down(self, features, label, new_prerequisites = None) -> str
            # Get the conclusion of this case
            conclusion = self.predict(features)
            # If it contradicts the label, add right cornerstone
            if conclusion is not None and conclusion != label:
                # Create a new cornerstone with the label as the conclusion
                conclusion_leaf = self._traverse_to_conclusion_leaf(features)
                if conclusion_leaf is not None:
                    conclusion_leaf._contradict(
                        features, label, new_prerequisites)
                else:
                    # For the general case, i.e. root rule
                    self._contradict(
                        features, label, new_prerequisites)
            # The rule did not activate the label, so add Left Cornerstone
            elif conclusion is None and label != self._general_case:
                # Create a new cornerstone with the label as the conclusion
                first_empty_left = self._traverse_to_conclusion_leaf(features)
                if first_empty_left is not None:
```

```
                    first_empty_left._manifest(
                        features, label, new_prerequisites)
                else:
                    # For the general case, i.e. root rule, this is the first ru
                    self._contradict(
                        features, label, new_prerequisites)


    def predict(self, features) -> str | None:
        # If prerequisite is met
        if self.is_fulfilled(features):
            # Set the last conclusion to this rule's conclusion
            last_conclusion = self._conclusion
            # If the rule has a right child
            if self._right is not None:
                # calculate the right child's prediction, this should be the
                right_conclusion = self._right.predict(features)
                # if the right child has a conclusion
                if right_conclusion is not None:
                    # set the last conclusion to the right child's conclusio
                    last_conclusion = right_conclusion
            return last_conclusion
        # If prerequisite is not met, but has a left child
        elif self._left is not None:
            # calculate the left child's prediction, this should be the conc
            return self._left.predict(features)
        # Default of no conclusion
        return None

    def visualize(self, depth: int = 0, side = None) -> str:
        # Create the string to return
        string = ' ' * depth + f"{side}" if side is not None else ''
        # Add the current node's visualization to the string
        string += self.__str__()
        # If there is a left child
        if self._left is not None:
            # Add the left child's visualization to the string
            string += self._left.visualize(depth + 1, "L: ")
        # If there is a right child
        if self._right is not None:
            # Add the right child's visualization to the string
            string += self._right.visualize(depth + 1, "R: ")
        return string

    def __str__(self) -> str:
        # Prerequisites => Conclusion
        return f"{' & '.join(self._prerequisites) if self._prerequisites is
```

RDR Diimplementasikan dalam kelas Python meniru cara kerja Model pada pustaka Scikit-Learn, karena menurut saya model mereka mudah digunakan dan saya ingin mengemulasi hal tersebut dalam kode saya.

Ada tiga fungsi utama yang diimplementasikan, yaitu:

1. Fit

Menggunakan Dataset dan label untuk melatih model RDR Dalam implementasi, hal ini dilakukan dengan mengambil pasangan fitur dan label yang sudah diolah menjadi list dan dimasukkan ke dalam pohon RDR menggunakan aturan RDR yang ada. Beberapa term yang digunakan:

- contradict, hal ini terjadi ketika terdapat konklusi, namun pakar tidak setuju sehingga diubah, bisa dengan memasukkan fitur sendiri (tidak dalam fit) atau dipilih secara acak dengan strategi
- conclusion leaf, hal ini adalah daun pohon RDR yang mencapai konklusi yang dikontradiksi oleh pakar atau label data
- manifest, hal ini terjadi ketika mendapatkan konklusi yang tidak sesuai dan di daun selanjutnya, fitur tidak memenuhi prasyarat konklusi. Maka rule dapat dikatakan gagal manifestasi sehingga dinamakan manifestasi

2. Predict

Predict digunakan untuk memprediksi kategori dataset yang dimasukkan ke dalam model, hasil akan dikeluarkan sebagai suatu daftar hasil

3. Add_New_Cornerstone

Fungsi ini khusus untuk masukan pakar sehingga bisa mengatur konklusi dan prekondisi apa saja yang ingin ditentukan oleh rule tersebut, kegunaan dicontohkan di bagian bawah

Import Dataset

In [ ]:
```
df = pd.read_csv('../test/diabetes_012_health_indicators_BRFSS2015.csv').drop(co
df.head()
```

Out[ ]:

	Diabetes_012	HighBP	HighChol	CholCheck	Smoker	Stroke	HeartDiseaseorAttack	PhysA
0	0.0	1.0	1.0	1.0	1.0	0.0	0.0	
1	0.0	0.0	0.0	0.0	1.0	0.0	0.0	
2	0.0	1.0	1.0	1.0	0.0	0.0	0.0	
3	0.0	1.0	0.0	1.0	0.0	0.0	0.0	
4	0.0	1.0	1.0	1.0	0.0	0.0	0.0	

Dataset ini dipilih karena ada kategori yang jelas antara Tidak Diabetes, PraDiabetes, dan Diabetes dan juga fitur yang digunakan banyak yang biner antara True atau False (0, 1). Kolom yang memiliki data non Biner didrop saja terlebih dahulu.

In [ ]:
```
labels = df['Diabetes_012'].replace({0.0: 'Normal', 1.0: 'Pre-diabetes', 2.0: 'D
labels.head()
```

Out[ ]:
```
0    Normal
1    Normal
2    Normal
3    Normal
4    Normal
Name: Diabetes_012, dtype: object
```

In [ ]:
```
df = df.drop(columns=['Diabetes_012'], axis=1)
df.head()
```

Out[ ]:

	HighBP	HighChol	CholCheck	Smoker	Stroke	HeartDiseaseorAttack	PhysActivity	Fruits
0	1.0	1.0	1.0	1.0	0.0	0.0	0.0	0.0
1	0.0	0.0	0.0	1.0	0.0	0.0	1.0	0.0
2	1.0	1.0	1.0	0.0	0.0	0.0	0.0	1.0
3	1.0	0.0	1.0	0.0	0.0	0.0	1.0	1.0
4	1.0	1.0	1.0	0.0	0.0	0.0	1.0	1.0

Dipisahkan antara Dataset dan label

In [ ]:
```
train_X = df.head(80)
train_y = labels.head(80)
test_X = df.tail(20)
test_y = labels.tail(20)
```

Buatkan Train Test Split Secara Manual

Masukan pakar

In [ ]:
```
data = df.head(20)
label = labels.head(20)
```

Mencontohkan masukan pakar, data yang digunakan kecil agar bisa terlihat perubahan pada pohon

In [ ]:
```
rdr_test = RDR()
rdr_test.fit(data, label, seed=42)
print(rdr_test)
```

```
General Case: Normal
Any Feature => General Case
 R: HighChol => Diabetes
  L: CholCheck => Diabetes
   R: CholCheck & Veggies => Normal
    L: CholCheck & HighBP => Normal
  R: HighChol & AnyHealthcare => Normal
   R: HighChol & AnyHealthcare & CholCheck => Diabetes
    R: HighChol & AnyHealthcare & CholCheck & Stroke => Normal
     L: HighChol & AnyHealthcare & CholCheck & HighBP => Normal
      L: HighChol & AnyHealthcare & CholCheck & HvyAlcoholConsump => Normal
```

Pada contoh ini mungkin pakar tidak setuju dengan diagnosis HighChol & AnyHealthcare & CholCheck & Stroke maka Normal karena menurutnya jika pasien makan buah, maka diabetes. Maka dibuat aturan berikut

In [ ]:
```
# Bisa oleh masukan mesin, tidak harus manual oleh pakar
newd = pd.DataFrame([[0.0, 1.0, 1.0, 0.0, 1.0, 0.0, 0.0, 1.0, 0.0, 0.0, 1.0, 0.0
newl = 'Diabetes'
preq = ['HighChol', 'AnyHealthcare', 'CholCheck', 'Stroke', 'Fruits']

rdr_test.add_new_cornerstone(newd, 'Diabetes', preq)
print(rdr_test)
```

```
General Case: Normal
Any Feature => General Case
 R: HighChol => Diabetes
  L: CholCheck => Diabetes
   R: CholCheck & Veggies => Normal
    L: CholCheck & HighBP => Normal
  R: HighChol & AnyHealthcare => Normal
   R: HighChol & AnyHealthcare & CholCheck => Diabetes
    R: HighChol & AnyHealthcare & CholCheck & Stroke => Normal
     L: HighChol & AnyHealthcare & CholCheck & HighBP => Normal
      L: HighChol & AnyHealthcare & CholCheck & HvyAlcoholConsump => Normal
     R: HighChol & AnyHealthcare & CholCheck & Stroke & Fruits => Diabetes
```

Bisa dilihat di sini bahwa terdapat exception baru pada peraturan HighChol & AnyHealthcare & CholCheck & Stroke yaitu jika hal tersebut terjadi namun pasien juga makan buah, maka pasien sebenarnya diabetes

Model

In [ ]:
```
rdr = RDR()
rdr.fit(train_X, train_y, seed=42)
print(rdr)
```

```
General Case: Normal
Any Feature => General Case
 R: HighChol => Diabetes
  L: CholCheck => Diabetes
   R: CholCheck & Veggies => Normal
    L: CholCheck & HighBP => Normal
     L: CholCheck & AnyHealthcare => Normal
     R: CholCheck & HighBP & Fruits => Diabetes
      R: CholCheck & HighBP & Fruits & AnyHealthcare => Normal
    R: CholCheck & Veggies & AnyHealthcare => Diabetes
     R: CholCheck & Veggies & AnyHealthcare & PhysActivity => Normal
      L: CholCheck & Veggies & AnyHealthcare & DiffWalk => Normal
      R: CholCheck & Veggies & AnyHealthcare & PhysActivity & Fruits => Diabete
s
       L: CholCheck & Veggies & AnyHealthcare & PhysActivity & Sex => Diabetes
        R: CholCheck & Veggies & AnyHealthcare & PhysActivity & Sex & HighBP =>
Normal
       R: CholCheck & Veggies & AnyHealthcare & PhysActivity & Fruits & NoDocbc
Cost => Normal
  R: HighChol & AnyHealthcare => Normal
   L: HighChol & Veggies => Normal
   R: HighChol & AnyHealthcare & CholCheck => Diabetes
    R: HighChol & AnyHealthcare & CholCheck & Stroke => Normal
     L: HighChol & AnyHealthcare & CholCheck & HighBP => Normal
      L: HighChol & AnyHealthcare & CholCheck & HvyAlcoholConsump => Normal
       L: HighChol & AnyHealthcare & CholCheck & PhysActivity => Normal
        L: HighChol & AnyHealthcare & CholCheck & Smoker => Normal
      R: HighChol & AnyHealthcare & CholCheck & HighBP & Smoker => Diabetes
       L: HighChol & AnyHealthcare & CholCheck & HighBP & Veggies => Pre-diabet
es
        L: HighChol & AnyHealthcare & CholCheck & HighBP & PhysActivity => Diab
etes
        R: HighChol & AnyHealthcare & CholCheck & HighBP & Veggies & PhysActivi
ty => Normal
       R: HighChol & AnyHealthcare & CholCheck & HighBP & Smoker & Veggies => N
ormal
        L: HighChol & AnyHealthcare & CholCheck & HighBP & Smoker & HeartDiseas
eorAttack => Normal
        R: HighChol & AnyHealthcare & CholCheck & HighBP & Smoker & Veggies & D
iffWalk => Diabetes
     R: HighChol & AnyHealthcare & CholCheck & Stroke & HighBP => Diabetes
      R: HighChol & AnyHealthcare & CholCheck & Stroke & HighBP & Smoker => Nor
mal
       R: HighChol & AnyHealthcare & CholCheck & Stroke & HighBP & Smoker & NoD
ocbcCost => Diabetes
```

Referensi untuk pembentukan Pohon diambil dari:
https://www.cse.unsw.edu.au/~claude/programs/iprolog/Doc/html/Extenions/rdr.html

Secara umum, jika terjadi kontradiksi, prerequisite akan ditambah atau dihilangkan satu. Jika label ke kiri, maka akan digunakan salah satu fitur atau semua

Testing

In [ ]:
```
pred = rdr.predict(test_X)
```

```
positive = 0
for p, y in zip(pred.values, test_y.values):
    # Jika Prediksi Benar
    if p == y:
        positive += 1
print(f"Accuracy: {positive/len(pred.values)}")
```

```
Accuracy: 0.35
```

Akurasi secara umum tidak baik karena model hanya menggunakan data secara supervised training, sedangkan tujuan awalnya untuk digunakan oleh pakar yang jarang salah. Akurasi menurun karena algoritma mengambil fitur secara acak sehingga jarang mendapatkan fakta sebenarnya

Model with larger dataset

Menggunakan perbandingan dengan model yang dilatih dengan data lebih banyak

In [ ]:
```
train_XL = df.head(80000)
train_yL = labels.head(80000)
test_XL = df.tail(20000)
test_yL = labels.tail(20000)
```

Menggunakan jumlah data yang lebih besar kali 1000 dari percobaan pertama

In [ ]:
```
rdrL = RDR()
rdrL.fit(train_XL, train_yL, seed=42)
# print(rdr)
```

Tidak divisualisasikan karena sangat besar

In [ ]:
```
predL = rdrL.predict(test_XL)

positiveL = 0
for pL, yL in zip(predL.values, test_yL.values):
    # Jika Prediksi Benar
    if pL == yL:
        positiveL += 1
print(f"Accuracy: {positiveL/len(predL.values)}")
```

```
Accuracy: 0.71935
```

Bisa dilihat bahwa akurasi semakin besar, namun tidak bisa mencapai akurasi yang tinggi (> 95%) hanya dengan supervised training saja.

Namun, jika dilihat dari segi kepraktisan dan lama training, maka mendapatkan akurasi 72% pada dataset sebesar ini cukup baik dibandingkan algoritma lainnya.