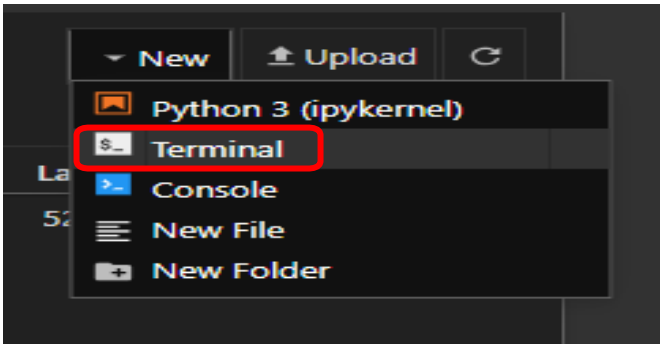
AIM:For a given set of training data examples stored in a .CSV file, implement and demonstrate the Candidate Elimination algorithm to output a description of these to fall hypotheses consistent with the training examples.

Open jupyter notebook using python

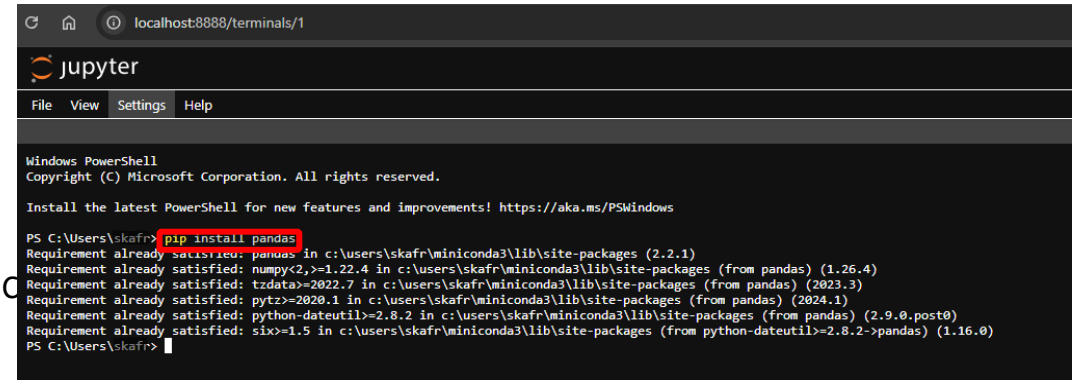
```
C:\Users\LAB>python -m notebook

    _   _          _      _
   | | | |_ __  __| |___ _| |_ ___
   | |_| | '_ \/ _` / _` |  _/ -_)
    \___/| .__/\__,_\__,_|\__\___|
         |_|

Read the migration plan to Notebook 7 to learn about the new features and the actions to take if you are using extensions.

https://jupyter-notebook.readthedocs.io/en/latest/migrate_to_notebook7.html

Please note that updating to Notebook 7 might break some of your extensions.

[I 10:21:52.490 NotebookApp] Serving notebooks from local directory: C:\Users\LAB
[I 10:21:52.490 NotebookApp] Jupyter Notebook 6.5.7 is running at:
[I 10:21:52.490 NotebookApp] http://localhost:8888/?token=280f777b4e22d879c776e9eb35d05f2c8a1c9a637a2b35cf
[I 10:21:52.491 NotebookApp]  or http://127.0.0.1:8888/?token=280f777b4e22d879c776e9eb35d05f2c8a1c9a637a2b35cf
[I 10:21:52.491 NotebookApp] Use Control-C to stop this server and shut down all kernels (twice to skip confirmation).
[C 10:21:52.544 NotebookApp]

    To access the notebook, open this file in a browser:
        file:///C:/Users/LAB/AppData/Roaming/jupyter/runtime/nbserver-3452-open.html
    Or copy and paste one of these URLs:
        http://localhost:8888/?token=280f777b4e22d879c776e9eb35d05f2c8a1c9a637a2b35cf
     or http://127.0.0.1:8888/?token=280f777b4e22d879c776e9eb35d05f2c8a1c9a637a2b35cf
```

Open new terminal file:



Install pandas package in jupyter terminal:

>pip install pandas



```python
import pandas as pd
# Example dataset
data = {
    'Attribute1': ['Yes', 'Yes', 'No', 'No'],
    'Attribute2': ['No', 'No', 'No', 'Yes'],
    'Class': ['Yes', 'Yes', 'No', 'No']
}
df = pd.DataFrame(data)
def candidate_elimination(df):
    num_attributes = len(df.columns) - 1
    G = {('?',)*num_attributes}
    first_positive = df[df['Class'] == 'Yes'].iloc[0][:-1]
    S = {tuple(first_positive)}
    for _, row in df.iterrows():
        sample, target = tuple(row[:-1]), row[-1]
        if target == 'Yes':
            G = {g for g in G if is_consistent(g, sample)}
            S = generalize_S(S, sample)
        else:
            S = {s for s in S if not is_consistent(s, sample)}
            G = specialize_G(G, S, sample)
    return G, S
def is_consistent(hypothesis, sample):
    return all(h == '?' or h == s for h, s in zip(hypothesis, sample))
def generalize_S(S, sample):
    S_next = set()
    for s in S:
        if is_consistent(s, sample):
            S_next.add(s)
        else:
            S_next |= {tuple('?' if s != x else x for s, x in zip(s, sample))}
    return S_next
def specialize_G(G, S, sample):
    G_next = set()
    for g in G:
        if not is_consistent(g, sample):
            G_next.add(g)
        else:
            for i in range(len(g)):
                if g[i] == '?':
                    for value in {'Yes', 'No'}:
                        if sample[i] != value:
                            g_new = list(g)
                            g_new[i] = value
                            g_new = tuple(g_new)
                            if any(is_more_general(s, g_new) for s in S):
                                G_next.add(g_new)
    return G_next
def is_more_general(h1, h2):
    more_general_parts = []
    for x, y in zip(h1, h2):
        mg = x == '?' or (x != '0' and (x == y or y == '0'))
        more_general_parts.append(mg)
    return all(more_general_parts)
# Running the algorithm with the dataset
G, S = candidate_elimination(df)
print("General Hypotheses:\n", G)
print("Specific Hypotheses:\n", S)
```

Output:

```
General Hypotheses:
 set()
Specific Hypotheses:
 {('Yes', 'No')}
```

Create another jupyter file :

Write the below code

```
import pandas as pd
# Example dataset
data = {
    'Attribute1': ['Yes', 'Yes', 'No', 'No', 'Yes'],
    'Attribute2': ['No', 'No', 'No', 'Yes', 'Yes'],
    'Class': ['Yes', 'No', 'No', 'Yes', 'No']
}
df = pd.DataFrame(data)

def candidate_elimination(df):
    num_attributes = len(df.columns) - 1
    # Initialize General Hypothesis and Specific Hypothesis
    G = {('?',) * num_attributes}
    S = set()
    for _, row in df.iterrows():
        sample, target = tuple(row[:-1]), row[-1]
        if target == 'Yes':
            if not S:
                S = {sample}
            else:
                S = generalize_S(S, sample)
            G = {g for g in G if is_consistent(g, sample)}
        else:
            S = {s for s in S if not is_consistent(s, sample)}
            G = specialize_G(G, S, sample)
        if not S or not G:
            return set(), set()
    return G, S

def is_consistent(hypothesis, sample):
    return all(h == '?' or h == s for h, s in zip(hypothesis, sample))
```

```
    return all(h == '?' or h == s for h, s in zip(hypothesis, sample))
def generalize_S(S, sample):
    S_next = set()
    for s in S:
        if is_consistent(s, sample):
            S_next.add(s)
        else:
            S_next |= {tuple('?' if s_i != x else x for s_i, x in zip(s, sample))}
    return S_next
def specialize_G(G, S, sample):
    G_next = set()
    for g in G:
        if not is_consistent(g, sample):
            for i in range(len(g)):
                if g[i] == '?':
                    for value in {'Yes', 'No'}:
                        if sample[i] != value:
                            g_new = list(g)
                            g_new[i] = value
                            g_new = tuple(g_new)
                            if any(is_more_general(s, g_new) for s in S):
                                G_next.add(g_new)
    return G_next
def is_more_general(h1, h2):
    more_general_parts = []
    for x, y in zip(h1, h2):
        mg = x == '?' or (x != '0' and (x == y or y == '0'))
        more_general_parts.append(mg)
    return all(more_general_parts)
# Running the algorithm with the dataset
G, S = candidate_elimination(df)
print("General Hypotheses:\n", G)
print("Specific Hypotheses:\n", S)
```

Output:

```
General Hypotheses:
 set()
Specific Hypotheses:
 set()
```

The Candidate-Elimination algorithm is used in concept learning and aims to find a set of all hypotheses that correctly classify given training examples. In concept learning, we have a set of attributes and a class that we want to predict.

The algorithm maintains two sets of hypotheses:

1. **General Hypotheses (G):**
   This set starts with the most general hypothesis possible, which, in our binary attribute case, is ('?', '?', ..., '?'), where `?` means the attribute can take any value (both 'Yes' and 'No' in our case).
2. **Specific Hypotheses (S):**
   This set starts with the most specific hypothesis. In the original algorithm, this is often the first positive instance in the data. However, in your implementation, it's initialized as ('0', '0', ..., '0'), where `0` implies no values are acceptable – this is a point of deviation from the standard approach.

The algorithm processes each example in the dataset and updates **G** and **S** as follows:

- **For a Positive Example:**
  - Remove from **G** any hypotheses inconsistent with the example.
  - Generalize the hypotheses in **S** if they do not cover the example.
- **For a Negative Example:**
  - Remove from **S** any hypotheses inconsistent with the example.
  - Specialize the hypotheses in **G** to exclude the example.

The functions `is_consistent`, `generalize_S`, and `specialize_G` handle these updates.

- **`is_consistent`** checks if a hypothesis is consistent with an example.
- **`generalize_S`** generalizes hypotheses in **S**.
- **`specialize_G`** specializes hypotheses in **G**.

The reason for getting empty sets for both **G** and **S** in certain datasets is due to contradictory examples or limitations in the attribute space. If the dataset includes two examples with identical attributes but different classes, no single hypothesis can cover both correctly. This leads to the elimination of all hypotheses from both sets.

For instance, in a dataset where the same combination of attribute values maps to both 'Yes' and 'No' classes, any hypothesis trying to explain 'Yes' will fail for 'No' and vice versa. Therefore, all hypotheses get eliminated, leaving **G** and **S** empty.

The effectiveness of the Candidate-Elimination algorithm heavily depends on the nature of the dataset. It works best when there's a clear and consistent relationship between attribute values and classes without any contradictory examples. In real-world scenarios, especially with noisy data or more complex relationships, this algorithm might face challenges in finding a consistent hypothesis set.