Summary Report

Name:- Goutham Sri Vishwesh Bikkumalla

I gained knowledge of regular expressions and their applications through this project, which included a real-time example. I utilized regular expressions, or Reg Exp, on the Stanford University database's ContactFinder list. I chose option 3 and attempted to solve as many false negatives as I could; I was able to solve virtually all of them, with the exception of nine files.

Let me enumerate every regular expression pattern I created.

Patterns Used for the phone number: -

1. ppatterns.append('(\d{3})-(\d{3})-(\d{4})'): -
   A regular expression pattern is added to the list ppatterns by this line. To match a phone number in the format XXX-XXX-XXXX, where each X indicates a digit, utilize the pattern (\d{3})-(\d{3})-(\d{4}).
   (\d{3}) corresponds to three digits.
   A literal hyphen is matched by the hyphen -.
   Three repetitions of this pattern correspond to the three phone number segments, which are XXX-XXX-XXXX.
2. ppatterns.append('\((\d{3})\)(\d{3})-(\d{4})')
   This line appends a regular expression pattern to ppatterns.
   The pattern '\((\d{3})\)(\d{3})-(\d{4})' is used to match a phone number in the format of (XXX)XXX-XXXX.
   '\(' and '\)' match literal parentheses.
   (\d{3}) matches three digits enclosed within the parentheses.
   (\d{3})-(\d{4}) matches the remaining part of the phone number in the format XXX-XXXX.
3. ppatterns.append('\((\d{3})\)\s*(\d{3})-(\d{4})')

   This line appends a regular expression pattern to ppatterns.
   The pattern '\((\d{3})\)\s*(\d{3})-(\d{4})' is used to match a phone number in the format of (XXX) XXX-XXXX or (XXX)XXX-XXXX.
   \s* matches zero or more whitespace characters.
   The rest of the pattern is similar to the previous one.
4. ppatterns.append('\(?(\d{3})\)?\s*-?\s*(\d{3})-(\d{4})')
   A regular expression pattern is added to ppatterns by this line.
   To match a phone number in formats like XXX-XXX-XXXX, (XXX)XXX-XXXX, XXX-XXX-XXXX, and XXXXXXXXXX, use the pattern \(?(\d{3})\)?\s*-?\s*(\d{3})-(\d{4})'.
   \-? denotes a single or zero hyphen.
   The remainder of the pattern is reminiscent of the earlier ones.
5. ppatterns.append('(\d{3})\s(\d{3})-(\d{4})')
   The pattern (\d{3})\s(\d{3})-(\d{4}) is used to match a phone number in the format XXX XXX-XXXX.

6. ppatterns.append('\[(\d{3})\]\s(\d{3})-(\d{4})')
   The pattern '\[(\d{3})\]\s(\d{3})-(\d{4})' is used to match a phone number in the format [XXX] XXX-XXXX.
7. ppatterns.append('(\d{3})\s(\d{3})\s(\d{4})')
   The pattern (\d{3})\s(\d{3})\s(\d{4}) is used to match a phone number in the format XXX XXX XXXX.

Patterns Used for the Email: -

- epatterns.append('([A-Za-z.]+)@([A-Za-z.]+)\.edu')
  Matches email addresses ending with ".edu".
  Example: john.doe@example.edu
- epatterns.append('([A-Za-z.]+)\s@\s([A-Za-z.]+)\.edu')
  Matches email addresses separated by "@", ending with ".edu".
  Example: jane_smith@example.edu
- epatterns.append('([a-zA-z.]+)\sat\s([a-zA-Z]+\.[a-zA-Z.]+)\.EDU')
  Matches email addresses with "at" and ".edu" spelled out.
  Example: aliceatexample.com.edu
- epatterns.append('([A-Za-z]+)\sWHERE\s([A-Za-z]+)\sDOM\sedu')
  Matches email addresses with "WHERE" and "DOM" spelled out, ending with ".edu".
  Example: johndoe WHERE example DOM edu
- epatterns.append('([A-Za-z.]+)@([A-Za-z.]+)\.edu')
  Matches email addresses with "@" representing "@" and ending with ".edu".
  Example: info@example.edu
- epatterns.append('([A-Za-z]+)\s*at\s*<!--.*?-->\s*([A-Za-z]+)\s*<!--.*?-->\s*dot\s*<!--.*?-->\s*edu')
  Matches email addresses with "at" and "dot" spelled out, embedded in HTML comments, and ending with ".edu".
  Example: user1<!--at-->example<!--dot-->edu
- epatterns.append('([A-Za-z.]+)\s<at symbol>\s([A-Za-z.]+)\.edu')
  Matches email addresses with "<at symbol>" representing "@" and ending with ".edu".
  Example: support <at symbol> example.edu
- epatterns.append('\(([A-Za-z.]+)\s*@\s*([A-Za-z.]+)\.edu\)')
  Matches email addresses enclosed in parentheses, separated by "@" and ending with ".edu".
  Example: (info@example.edu)
- epatterns.append('([A-Za-z.]+)\s*(?:@|AT)\s*([A-Za-z.]+)\s*(?:\.|DOT)\s*edu')
  Matches email addresses with variations of "@" and "dot" spelled out, ending with ".edu".
  Example: contact AT example DOT edu
- epatterns.append('email to ([A-Za-z.]+) at ([A-Za-z.]+) dt com')
  Matches email addresses formatted as "email to <username> at <domain> dt com".
  Example: email to info at example dt com
- epatterns.append('E-mail: ([A-Za-z.]+) at ([A-Za-z.]+)\.edu')
  Matches email addresses prefixed with "E-mail:" and separated by "at", ending with ".edu".

Example: E-mail: info at example.edu

- epatterns.append('([a-zA-Z]+)\sat\s([a-zA-Z]+)\s([a-zA-Z]+)\s+edu')
  Matches email addresses with "at" and "edu" spelled out, separated by spaces.
  Example: john at example com edu
- epatterns.append('email: ([A-Za-z.]+) at ([A-Za-z.]+) ([A-Za-z.]+) edu')
  Matches email addresses prefixed with "email:" and separated by "at", ending with "edu".
  Example: email: info at example edu
- epatterns.append('([A-Za-z]+)\sat\s([A-Za-z]+)\sdot\s([A-Za-z]+)\sdot\s([A-Za-z]+)')

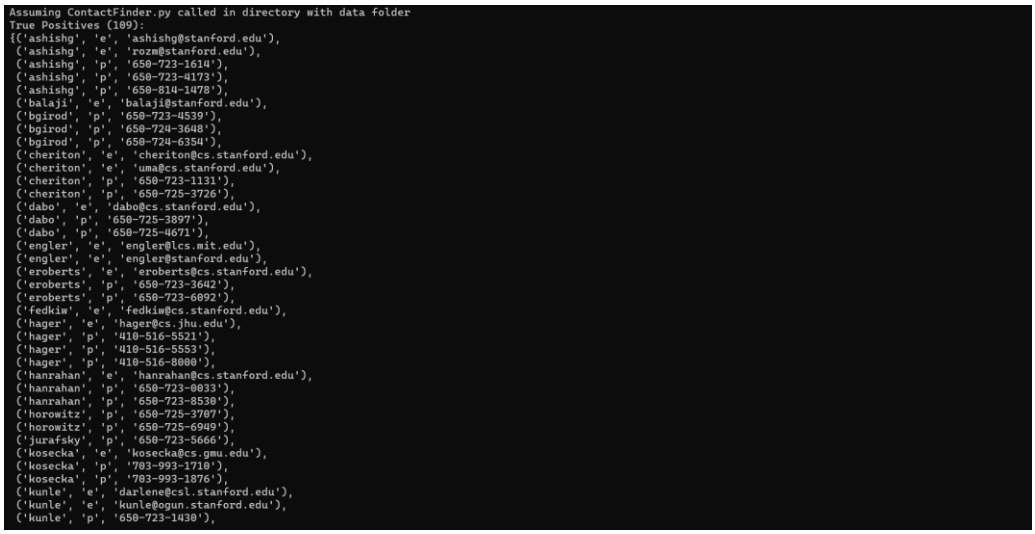
After Using All these Expressions output :
True Positives : 109
False Positives : 2
False Negatives : 8

I have taken the option 3 task so accordingly change the code in the process_file and attaching the screenshots below: -

```
def process_file(name, f):
    # note that debug info should be printed to stderr
    # sys.stderr.write('[process_file]\tprocessing file: %s\n' % (path))
    res = []
    for line in f:
        # you may modify the line, using something like substitution
        #    before applying the patterns

        # email pattern list
        for epat in epatterns:
            # each epat has 2 sets of parentheses so each match will have 2
items in a list
            matches = re.findall(epat,line)
            for m in matches:
                # string formatting operator % takes elements of list m
                #   and inserts them in place of each %s in the result string
                # email has form  someone@somewhere.edu
                #email = '%s@%s.edu' % m

                if epat.endswith("com"):
                    email = '{}@{}.com'.format(m[0], m[1])
                elif (" at " in line and " dot " not in line) and
len(m)==3:  # Identify pattern without explicit 'dot'
                    email = '{}@{}.{}.edu'.format(m[0], m[1], m[2])
                elif len(m) == 4:
                    y = '{}.{}'.format(m[1], m[2])
                    email = '{}@{}.edu'.format(m[0], y)
                else:
                    email = '{}@{}.edu'.format(m[0],m[1])
```

```
                res.append((name,'e',email))

        # phone pattern list
        for ppat in ppatterns:
            # each ppat has 3 sets of parentheses so each match will have 3
items in a list
            matches = re.findall(ppat,line)
            for m in matches:
                # phone number has form  areacode-exchange-number
                #phone = '%s-%s-%s' % m
                phone = '{}-{}-{}'.format(m[0],m[1],m[2])
                res.append((name,'p',phone))
    return res
```

Output Screenshot :-



```
 ('shoham', 'p', '650-723-3432'),
 ('shoham', 'p', '650-725-1449'),
 ('subh', 'e', 'subh@stanford.edu'),
 ('subh', 'e', 'uma@cs.stanford.edu'),
 ('subh', 'p', '650-724-1915'),
 ('subh', 'p', '650-725-3726'),
 ('subh', 'p', '650-725-6949'),
 ('thm', 'e', 'pkrokel@stanford.edu'),
 ('thm', 'p', '650-725-3383'),
 ('thm', 'p', '650-725-3636'),
 ('thm', 'p', '650-725-3938'),
 ('tim', 'p', '650-724-9147'),
 ('tim', 'p', '650-725-2340'),
 ('tim', 'p', '650-725-4671'),
 ('ullman', 'e', 'support@gradiance.com'),
 ('ullman', 'e', 'ullman@cs.stanford.edu'),
 ('ullman', 'p', '650-494-8016'),
 ('ullman', 'p', '650-725-2588'),
 ('ullman', 'p', '650-725-4802'),
 ('vladlen', 'e', 'vladlen@stanford.edu'),
 ('widom', 'e', 'siroker@cs.stanford.edu'),
 ('widom', 'e', 'widom@cs.stanford.edu'),
 ('widom', 'p', '650-723-0872'),
 ('widom', 'p', '650-723-7690'),
 ('widom', 'p', '650-725-2588'),
 ('zelenski', 'e', 'zelenski@cs.stanford.edu'),
 ('zelenski', 'p', '650-723-6092'),
 ('zelenski', 'p', '650-725-8596'),
 ('zm', 'e', 'manna@cs.stanford.edu'),
 ('zm', 'p', '650-723-4364'),
 ('zm', 'p', '650-725-4671')}
False Positives (2):
('nass', 'p', '375-742-1353')
   gold:  ('nass', 'e', 'nass@stanford.edu')
   gold:  ('nass', 'p', '650-723-5499')
   gold:  ('nass', 'p', '650-725-2472')
('nass', 'p', '114-910-7699')
   gold:  ('nass', 'e', 'nass@stanford.edu')
   gold:  ('nass', 'p', '650-723-5499')
   gold:  ('nass', 'p', '650-725-2472')
```

```
 ('widom', 'p', '650-723-0872'),
 ('widom', 'p', '650-723-7690'),
 ('widom', 'p', '650-725-2588'),
 ('zelenski', 'e', 'zelenski@cs.stanford.edu'),
 ('zelenski', 'p', '650-723-6092'),
 ('zelenski', 'p', '650-725-8596'),
 ('zm', 'e', 'manna@cs.stanford.edu'),
 ('zm', 'p', '650-723-4364'),
 ('zm', 'p', '650-725-4671')}
False Positives (2):
('nass', 'p', '375-742-1353')
   gold:  ('nass', 'e', 'nass@stanford.edu')
   gold:  ('nass', 'p', '650-723-5499')
   gold:  ('nass', 'p', '650-725-2472')
('nass', 'p', '114-910-7699')
   gold:  ('nass', 'e', 'nass@stanford.edu')
   gold:  ('nass', 'p', '650-723-5499')
   gold:  ('nass', 'p', '650-725-2472')
False Negatives (8):
{('dlwh', 'e', 'dlwh@stanford.edu'),
 ('jks', 'e', 'jks@robotics.stanford.edu'),
 ('jurafsky', 'e', 'jurafsky@stanford.edu'),
 ('latombe', 'e', 'asandra@cs.stanford.edu'),
 ('latombe', 'e', 'latombe@cs.stanford.edu'),
 ('latombe', 'e', 'liliana@cs.stanford.edu'),
 ('ouster', 'e', 'ouster@cs.stanford.edu'),
 ('ouster', 'e', 'teresa.lynn@stanford.edu')}
Summary: tp=109, fp=2, fn=8

C:\Users\JYOTHIKA\Desktop\ContactFinder (2) (1)\ContactFinder>
```

Observations and learnings-

Regular expressions are frequently employed in data extraction and parsing. We were able to retrieve emails and phone numbers from a significant amount of contested data, as demonstrated by our instances.

➢ Text searching and manipulation are possible with Regular Expressions (RegExp). Reg exp allows us to easily manipulate texts and use them as needed.

➢ Data validation: Reg exp programs are employed to validate data. It is typically used to confirm whether or not the user-provided inputs are in the correct format. It is easy to check for the sequence or patterns we want.