R Rule-based concepts use programmatic judges

Our ratings for rule-based concepts are partially done via programmatic checkers instead of a remote LM. We include all of our rule-based judges below.

```
def CheckEmoji(text):
    emoji_count = count_emojis_in_text(text)

    if emoji_count > 2:
        return 2.0
    else if emoji_count > 0:
        return 1.0
    else:
        return 0.0
```

```
def CheckUppercase(text):

    words = split_into_words(text)
    uppercase_words = [word for word in words if word.isupper()]
    percentage = (len(uppercase_words) / len(words)) * 2

    return percentage
```

```
def ContainsPassiveVoice(text):
    doc = nlp_parse(text)

    for sentence in doc.sentences:
        for word in sentence.words:
            if word.upos == 'VERB' and word.feats and 'Voice=Pass' in word.feats:
                return 2.0

    return 0.0
```

```
def CheckChinese(text):
    detected_language = langdetect.detect(text)

    if detected_language == 'zh-cn':
        return 2.0
    else:
        return 0.0
```

```
def CheckSpanish(text):
    detected_language = langdetect.detect(text)

    if detected_language == 'es':
        return 2.0
    else:
        return 0.0
```

```
def CheckAllLowercase(text):

    if not text:
        return 0.0

    words = split_into_words(text)
    lowercase_words = [word for word in words if word.islower()]
    percentage = len(lowercase_words) / len(words) * 2

    return percentage
```

```
def CheckPostscript(text):

    if find_pattern(text, "P\\.S\\..*$", multiline=True):
        return 2.0
    else:
        return 0.0
```

```
def CheckNumberedList(text):
    if find_pattern(text, "\\b\\d+\\."):
        return 2.0
    else:
        return 0.0
```

```
def CheckDoubleBreaks(text):
    paragraphs = split_text(text, "\n\n")

    if len(paragraphs) > 1:
        return 2.0
    else:
        return 0.0
```

```
def CheckAsteriskSeparation(text):
    if "***" in text:
        return 2.0
    else:
        return 0.0
```

```
def CheckStartsWithPhrase(text, phrase="Here is my response"):
    if text.strip().startswith(phrase):
        return 2.0
    else if "Here is my response" in text.strip():
        return 1.0
    else:
        return 0.0
```

```
def CheckWordsInQuotes(text):
    words = split_into_words(text)

    if not words:
        return 0.0

    text = text.replace("<end_of_turn>", "")
    quoted_words = 0

    for word in words:
        if word.startswith('"') and word.endswith('"'):
            quoted_words += 1

    return (quoted_words / len(words)) * 2.0
```

```
def CheckEndsWithHelp(text):

    if text.strip().endswith("Is there anything else I can help with?"):
        return 2.0
    else if "Is there anything else I can help with" in text.strip():
        return 1.0
    else:
        return 0.0
```

```
def CheckHasExclamation(text):
    text = text.replace("<end_of_turn>", "")
    exclamation_count = count_occurrences(text, '!')

    return min(2.0, exclamation_count * 0.5)
```

```
def IsPastTense(word):
    doc = nlp_parse(word)

    for sentence in doc.sentences:
        for word in sentence.words:
            if word.upos == 'VERB' and 'Tense=Past' in (word.feats if word.feats else '')
    :
                return 2.0

    return 0.0
```

```
def CheckHasHashtags(text, min_hashtags=4):
    hashtags = find_all_patterns(text, "#\\w+")

    if len(hashtags) >= min_hashtags:
        return 2.0
    else:
        return (len(hashtags) / min_hashtags) * 2.0
```

```
def CheckHasCitations(text):
    url_pattern = compile_regex(
        "http[s]?://(?:[a-zA-Z]|[0-9]|[$-_@.&+]|[!*\\(\\),]|(?:%
    )
    urls = url_pattern.findall(text)

    if urls:
        return 2.0
    else:
        return 0.0
```

```
def CheckTelephoneNumber(text):
    phone_patterns = [
        # Standard US formats
        "\\(\\d{3}\\)\\s*[\\d\\-\\s]+\\d{4}",                    # (123) 456-7890
        "\\d{3}[-.\s]?\\d{3}[-.\s]?\\d{4}",                      # 123-456-7890

        # International format
        "\\+?\\d{1,3}[-.\s]?\\d{3}[-.\s]?\\d{3}[-.\s]?\\d{4}",   # +1-123-456-7890

        # Local format
        "\\d{3}[-.\s]?\\d{4}",                                   # 555-1234

        # Alphanumeric formats
        "\\(\\d{3}\\)\\s*\\d{3}[-.\s][A-Z]+\\s*\\(\\d+\\)",      # (212) 555-STAGE (7824)

        # Additional formats with letters
        "\\(\\d{3}\\)\\s*\\d{3}[-.\s][A-Z\\d]+",                 # (212) 555-STAGE
        "\\d{3}[-.\s]\\d{3}[-.\s][A-Z\\d]+"                      # 212-555-STAGE
    ]

    # Count unique phone numbers found
    phone_numbers = set()
    for pattern in phone_patterns:
        matches = find_all_patterns(text, pattern, case_insensitive=True)
        for match in matches:
            phone_numbers.add(match)

    if len(phone_numbers) >= 1:
        return 2.0
    else:
        return 0.0
```