

5_pmh_analysis

July 31, 2023

```
[1]: import json
import os
from sklearn.feature_extraction.text import CountVectorizer
import pandas as pd
import numpy as np
import json5
import spacy
import medspacy
import random
from medspacy.ner import TargetMatcher, TargetRule
from medspacy.visualization import visualize_ent, visualize_dep
from spacy.tokens import Span
import sys

parent_dir = os.path.abspath("..")
if parent_dir not in sys.path:
    sys.path.append(parent_dir)
from path import DATA_PROCESSED_DOCUMENTS_DIR

[2]: chief_complaint = "chest-pain"
folder_location = os.path.join(
    DATA_PROCESSED_DOCUMENTS_DIR / chief_complaint / "black-or-african-american"
)
b_docs = []
w_docs = []
for filename in os.listdir(folder_location):
    file_location = os.path.join(folder_location, filename)
    if os.path.isfile(file_location):
        with open(file_location) as d:
            try:
                file_contents = d.read()
                content = json.loads(file_contents)
                b_docs.append(content)
            except Exception as e:
                try:
                    # pull of first and last line, gpt sometimes response with
                    ↪ a leading ```json and ends with ```
```

```

        tmp = file_contents.splitlines(True)
        while "{" not in tmp[0]:
            tmp = tmp[1:]
        while "}" not in tmp[-1]:
            tmp = tmp[:-1]
        tmp = "".join(tmp)
        content = json5.loads(tmp)
        b_docs.append(content)
    except Exception as e:
        # print(f"{file_location} Error: {e}")
        pass

folder_location = os.path.join(DATA_PROCESSED_DOCUMENTS_DIR / chief_complaint /
↪ "white-or-caucasian")
for filename in os.listdir(folder_location):
    file_location = os.path.join(folder_location, filename)
    if os.path.isfile(file_location):
        with open(file_location) as d:
            try:
                file_contents = d.read()
                content = json.loads(file_contents)
                w_docs.append(content)
            except Exception as e:
                try:
                    # pull of first and last line, gpt sometimes response with
↪ a leading ```json and ends with ```
                    tmp = file_contents.splitlines(True)
                    while "{" not in tmp[0]:
                        tmp = tmp[1:]
                    while "}" not in tmp[-1]:
                        tmp = tmp[:-1]
                    tmp = "".join(tmp)
                    content = json5.loads(tmp)
                    w_docs.append(content)
                except Exception as e:
                    # print(f"{file_location} Error: {e}")
                    pass

```

```

[3]: print(len(b_docs))
      print(len(w_docs))

```

4958

4949

```

[4]: # Grab the text from each document's past medical history section
      b_pmh = []
      for doc in b_docs:

```

```

        if doc.get("past_medical_history") is not None:
            b_pmh.append(doc.get("past_medical_history"))

w_pmh = []
for doc in w_docs:
    if doc.get("past_medical_history") is not None:
        w_pmh.append(doc.get("past_medical_history"))

```

```

[5]: # We want to see if each patient has a history of any of the following
      ↪ conditions
nlp = medspacy.load()
print(nlp.pipe_names)

Span.set_extension("icd10_code", default="")

ICD_TO_TEXT_MAP = {
    "I10": "hypertension",
    "E78.5": "hyperlipidemia",
    "M19.90": "osteoarthritis",
    "E11.9": "type ii diabetes mellitus",
    "E78.00": "hypercholesterolemia",
    "J45": "asthma",
    "I48.91": "atrial fibrillation",
    "M81. 0": "osteoporosis",
    "K21.9": "gastroesophageal reflux disease ",
    "I21.9": "myocardial infarction",
    "I25.10": "coronary artery disease",
}

# Add rules for target concept extraction
target_matcher = nlp.get_pipe("medspacy_target_matcher")
target_rules = [
    TargetRule("hypertension", category="CONDITION", attributes={"icd10_code": "I10"}),
    TargetRule(
        "hyperlipidemia", category="CONDITION", attributes={"icd10_code": "E78.5"}
    ),
    TargetRule(
        "osteoarthritis", category="CONDITION", attributes={"icd10_code": "M19.90"}
    ),
    TargetRule(
        "osteoporosis", category="CONDITION", attributes={"icd10_code": "M81.0"}
    ),
    TargetRule(

```

```

        "dyslipidemia", category="CONDITION", attributes={"icd10_code": "E78.5"}
    ),
    TargetRule(
        literal="Type II Diabetes Mellitus",
        category="CONDITION",
        attributes={"icd10_code": "E11.9"},
    ),
    TargetRule(
        literal="diabetes mellitus type 2",
        category="CONDITION",
        pattern=[
            {"LOWER": "diabetes"},
            {"LOWER": "mellitus"},
            {"LOWER": "type"},
            {"LOWER": {"IN": ["two", "ii", "2"]}},
        ],
        attributes={"icd10_code": "E11.9"},
    ),
    TargetRule(
        literal="gerd",
        category="CONDITION",
        pattern=[
            {"LOWER": "gastroesophageal"},
            {"LOWER": "reflux"},
            {"LOWER": "disease"},
        ],
        attributes={"icd10_code": "K21.9"},
    ),
    TargetRule(
        literal="GERD", category="CONDITION", attributes={"icd10_code": "K21.9"}
    ),
    TargetRule(
        literal="Type II Diabetes Mellitus",
        category="CONDITION",
        pattern=[
            {"LOWER": "type"},
            {"LOWER": {"IN": ["two", "ii", "2"]}},
            {
                "LOWER": {
                    "IN": [
                        "dm",
                        "diabetes mellitus",
                        "diabetes",
                    ]
                }
            },
        ],
    ),

```

```

        attributes={"icd10_code": "E11.9"},
    ),
    TargetRule("asthma", category="CONDITION", attributes={"icd10_code": "J45"},
    ↪ "J45")),
    TargetRule(
        "atrial fibrillation",
        category="CONDITION",
        attributes={"icd10_code": "I48.91"},
    ),
    TargetRule(
        "hypercholesterolemia",
        category="CONDITION",
        attributes={"icd10_code": "E78.00"},
    ),
    TargetRule(
        "high cholesterol",
        category="CONDITION",
        pattern=[{"LOWER": {"IN": ["high", "elevated"]}}, {"LOWER": "cholesterol"}],
    ↪ "cholesterol"}],
        attributes={"icd10_code": "E78.00"},
    ),
    TargetRule(
        "hypertriglyceridemia", category="CONDITION", attributes={"icd10_code": "E78.1"},
    ↪ "E78.1"},
    ),
    TargetRule(
        "myocardial infarction",
        category="CONDITION",
        pattern=[
            {"LOWER": "myocardial"},
            {"LOWER": "infarction"},
        ],
        attributes={"icd10_code": "I21.9"},
    ),
    TargetRule(
        "coronary artery disease",
        category="CONDITION",
        attributes={"icd10_code": "I25.10"},
    )
]
target_matcher.add(target_rules)

```

```
['medspacy_pyrush', 'medspacy_target_matcher', 'medspacy_context']
```

```
[6]: # Extract conditions from PMH
```

```

b_nlp_pmh = []
for doc in b_pmh:

```

```
doc = nlp(doc)
b_nlp_pmh.append(doc)
```

```
[7]: w_nlp_pmh = []
for doc in w_pmh:
    doc = nlp(doc)
    w_nlp_pmh.append(doc)
```

```
[8]: # Quick test to make sure negation detection works
# negation test
test = "patient admits to type 2 diabetes but denies any hypertension."
doc = nlp(test)
visualize_ent(doc)
for ent in doc.ents:
    print(ent._.is_negated)
```

<IPython.core.display.HTML object>

False

True

```
[9]: # Quick visualization of entity extraction
for doc in w_nlp_pmh[:3]:
    visualize_ent(doc)
```

<IPython.core.display.HTML object>

<IPython.core.display.HTML object>

<IPython.core.display.HTML object>

```
[10]: for doc in b_nlp_pmh[:3]:
        visualize_ent(doc)
```

<IPython.core.display.HTML object>

<IPython.core.display.HTML object>

<IPython.core.display.HTML object>

```
[11]: # Test entity extraction, making sure to ignore negated entities
test = b_nlp_pmh[0:2]
test.append(nlp("patient admits to type 2 diabetes but denies any hypertension.
↳he takes metformin for his type 2 diabetes."))
print(list(map(lambda x: [y for y in list(x.ents) if y._.is_negated == False],
↳test)))
print(
    list(
        map(
            lambda x: [y._.icd10_code for y in list(x.ents) if y._.is_negated
↳== False],
```

```

        test,
    )
)
)
# De-dup
print(
    list(
        map(
            lambda x: set([y._icd10_code for y in list(x.ents) if y._
↪is_negated == False]),
            test,
        )
    )
)
)

```

```

[[hypertension, osteoarthritis], [high cholesterol, hypertension], [type 2
diabetes, type 2 diabetes]]
[['I10', 'M19.90'], ['E78.00', 'I10'], ['E11.9', 'E11.9']]
[{'I10', 'M19.90'}, {'E78.00', 'I10'}, {'E11.9'}]

```

```

[12]: # Do entity extraction on the PMH section of the notes, skipping negated
↪entities. Make sure to de-duplicate the entities.
b_just_names = list(
    map(
        lambda x: set([y._icd10_code for y in list(x.ents) if y._is_negated
↪== False]),
        b_nlp_pmh,
    )
)

b_normalized_conditions_names = [
    element for sublist in b_just_names for element in sublist
]

w_just_names = list(
    map(
        lambda x: set([y._icd10_code for y in list(x.ents) if y._is_negated
↪== False]),
        w_nlp_pmh,
    )
)

w_normalized_conditions_names = [
    element for sublist in w_just_names for element in sublist
]
print(len(b_normalized_conditions_names))
print(len(w_normalized_conditions_names))

```

6355

6260

```
[13]: # Count the instances of each word in the black and white conditions.
      ↪ Conditions are de-duped, so if a condition appears multiple times in a
      ↪ single participant's data, it is only counted once.
      # We fix this later before doing statistical analysis.
      from collections import Counter

      b_word_freq = Counter(b_normalized_conditions_names)
      w_word_freq = Counter(w_normalized_conditions_names)
```

```
[14]: b_word_freq_df = pd.DataFrame(
      b_word_freq.items(), columns=["word", "b.frequency"]
    ).sort_values(by="b.frequency", ascending=False)
      w_word_freq_df = pd.DataFrame(
      w_word_freq.items(), columns=["word", "w.frequency"]
    ).sort_values(by="w.frequency", ascending=False)
```

```
[15]: wf_df = w_word_freq_df.merge(b_word_freq_df, how="inner", on="word")
      wf_df
```

```
[15]:
```

	word	w.frequency	b.frequency
0	I10	3173	3232
1	E78.5	2326	2326
2	M19.90	286	264
3	E11.9	164	187
4	E78.00	112	128
5	J45	69	72
6	I25.10	69	73
7	I21.9	34	48
8	K21.9	12	9
9	M81. 0	8	11
10	I48.91	7	5

```
[16]: wf_df["w.frequency_pct"] = wf_df["w.frequency"] / wf_df["w.frequency"].sum()
      wf_df["b.frequency_pct"] = wf_df["b.frequency"] / wf_df["b.frequency"].sum()
      wf_df["frequency_pct_diff"] = wf_df["b.frequency_pct"] - wf_df["w.
      ↪ frequency_pct"]
      wf_df["frequency_pct_diff_abs"] = wf_df["frequency_pct_diff"].abs()
      # Sort by largest values in absolute difference
      wf_df.sort_values(by="frequency_pct_diff", ascending=False).head(25)
```

```
[16]:
```

	word	w.frequency	b.frequency	w.frequency_pct	b.frequency_pct	\
3	E11.9	164	187	0.026198	0.029426	
4	E78.00	112	128	0.017891	0.020142	
7	I21.9	34	48	0.005431	0.007553	
0	I10	3173	3232	0.506869	0.508576	
6	I25.10	69	73	0.011022	0.011487	
9	M81. 0	8	11	0.001278	0.001731	

5	J45	69	72	0.011022	0.011330
10	I48.91	7	5	0.001118	0.000787
8	K21.9	12	9	0.001917	0.001416
2	M19.90	286	264	0.045687	0.041542
1	E78.5	2326	2326	0.371565	0.366011

	frequency_pct_diff	frequency_pct_diff_abs
3	0.003228	0.003228
4	0.002250	0.002250
7	0.002122	0.002122
0	0.001707	0.001707
6	0.000465	0.000465
9	0.000453	0.000453
5	0.000307	0.000307
10	-0.000331	0.000331
8	-0.000501	0.000501
2	-0.004145	0.004145
1	-0.005554	0.005554

```
[17]: # First order frequencies by magnitude of difference (absolute value), take the
      ↪ top 200 words with the greatest difference,
      # then re-sort by actual difference so when we plot the values will be
      ↪ sequential from smallest to largest bars
most = (
    wf_df.sort_values(by="frequency_pct_diff_abs", ascending=False)
    .head(200)
    .sort_values(by="frequency_pct_diff", ascending=False)
)

chart_data = {}

# Create a map with the word as the frequency, and the magnitude vector as the
↪ value\
# a vector of [0, n] will plot a blue bar
# a vector of [n, 0] will plot an orange bar
# a vector with a negative n [-n, 0] will plot a bar on the left
# a vector with a positive n [n, 0] will plot a bar on the right
# {"word": [-1, 0]} will plot an orange bar for "word" on the left of 0 with
↪ length 1
# {"word": [0, 0.5]} will plot a blue bar for "word" on the right of 0 with
↪ length 0.5
# in order to generate a good Positive Negative bar chart, we assign b freq to
↪ the left side (negative)
# and w freq to the right side (positive)
for row in most.iterrows():
    if row[1]["w.frequency_pct"] > row[1]["b.frequency_pct"]:
        # orange bars
```

```

        chart_data[row[1]["word"]] = [
            row[1]["w.frequency_pct"] - row[1]["b.frequency_pct"],
            0,
        ]
    else:
        # blue bars
        chart_data[row[1]["word"]] = [
            0,
            -(row[1]["b.frequency_pct"] - row[1]["w.frequency_pct"]),
        ]

```

```

[18]: # Positive Negative Bar Chart to better visualize where word frequencies
      ↪ diverge between data sets
      # Based on https://stackoverflow.com/a/69976552/11407943
import numpy as np
import matplotlib.pyplot as plt

category_names = ["white-or-caucasian", "black-or-african-american"]
results = chart_data

def survey(results, category_names):
    """
    Parameters
    -----
    results : dict
        A mapping from question labels to a list of answers per category.
        It is assumed all lists contain the same number of entries and that
        it matches the length of *category_names*. The order is assumed
        to be from 'Strongly disagree' to 'Strongly agree'
    category_names : list of str
        The category labels.
    """

    labels = list(map(lambda i: ICD_TO_TEXT_MAP.get(i), results.keys()))
    data = np.array(list(results.values()))
    data_cum = data.cumsum(axis=1)
    middle_index = data.shape[1] // 2
    offsets = 0 # data[:, range(middle_index)].sum(axis=1) # + data[:,
    ↪ middle_index]/2

    # Color Mapping
    category_colors = plt.get_cmap("coolwarm_r")(np.linspace(0.15, 0.85, data.
    ↪ shape[1]))

    fig, ax = plt.subplots(figsize=(15, 50))

```

```

# Plot Bars
for i, (colname, color) in enumerate(zip(category_names, category_colors)):
    widths = data[:, i]
    starts = data_cum[:, i] - widths - offsets
    rects = ax.barh(
        labels, widths, left=starts, height=0.5, label=colname, color=color
    )

# Add Zero Reference Line
ax.axvline(0, linestyle="--", color="black", alpha=0.25)

# X Axis
# ax.set_xlim(-0.006, 0.006)
# ax.set_xticks(np.arange(-0.0035, 0.0035, 0.003))
ax.xaxis.set_major_formatter(lambda x, pos: str(x))

# Y Axis
ax.invert_yaxis()

# Remove spines
ax.spines["right"].set_visible(False)
ax.spines["top"].set_visible(False)
ax.spines["left"].set_visible(False)

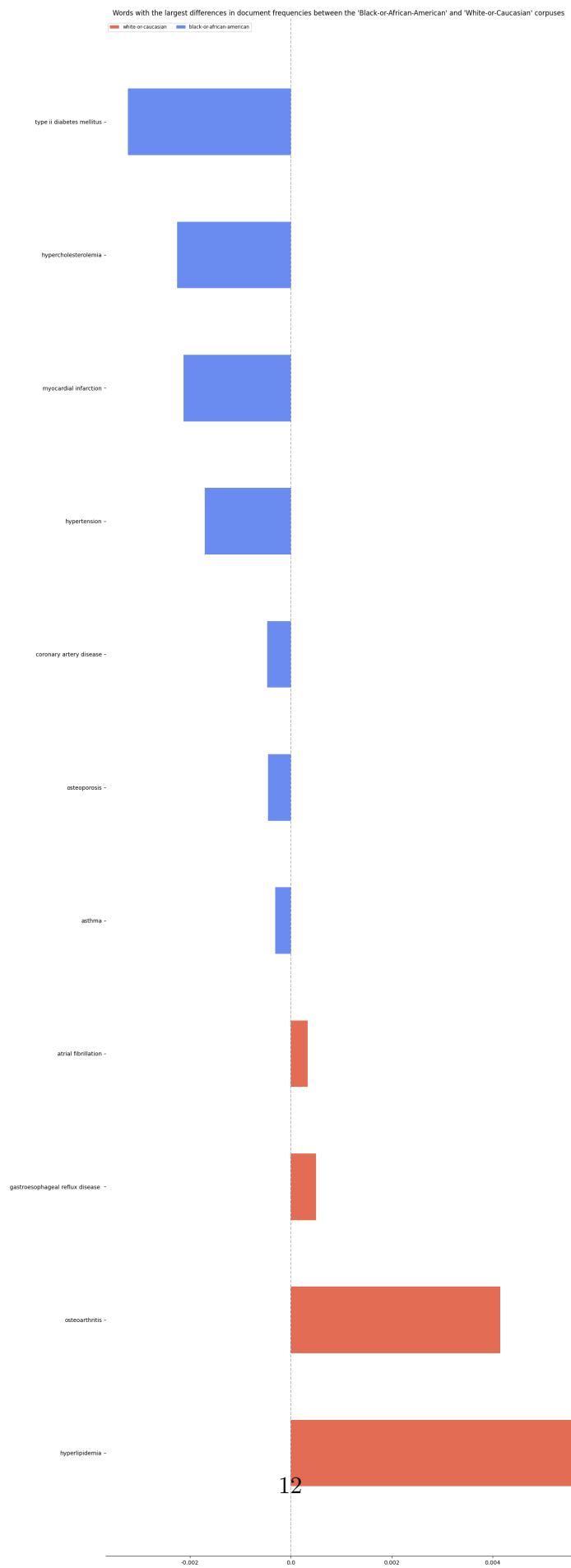
# Legend
ax.legend(
    ncol=len(category_names),
    bbox_to_anchor=(0, 0.99),
    loc="lower left",
    fontsize="small",
)

# Set Background Color
fig.set_facecolor("#FFFFFF")

return fig, ax

fig, ax = survey(results, category_names)
plt.title(
    "Words with the largest differences in document frequencies between the_
    ↪ 'Black-or-African-American' and 'White-or-Caucasian' corpuses"
)
plt.show()

```



```
[19]: import scipy
      from sklearn.feature_extraction import text
      from collections import Counter
```

```
[20]: b_just_names_lower = [
      list(map(lambda x: ICD_TO_TEXT_MAP.get(x), arr)) for arr in b_just_names
      ]
      b_list_of_doc_counter = list(map(Counter, b_just_names_lower))
      # element for sublist in w_just_names for element in sublist
      w_just_names_lower = [
      list(map(lambda x: ICD_TO_TEXT_MAP.get(x), arr)) for arr in w_just_names
      ]
      w_list_of_doc_counter = list(map(Counter, w_just_names_lower))
      b_conditions_names_counter = Counter(
      [element for sublist in b_just_names_lower for element in sublist]
      )
      w_conditions_names_counter = Counter(
      [element for sublist in w_just_names_lower for element in sublist]
      )
```

```
[21]: b_conditions_names_counter
```

```
[21]: Counter({'hypertension': 3232,
              'hyperlipidemia': 2326,
              'osteoarthritis': 264,
              'type ii diabetes mellitus': 187,
              'hypercholesterolemia': 128,
              'coronary artery disease': 73,
              'asthma': 72,
              'myocardial infarction': 48,
              'osteoporosis': 11,
              'gastroesophageal reflux disease ': 9,
              'atrial fibrillation': 5})
```

```
[22]: w_conditions_names_counter
```

```
[22]: Counter({'hypertension': 3173,
              'hyperlipidemia': 2326,
              'osteoarthritis': 286,
              'type ii diabetes mellitus': 164,
              'hypercholesterolemia': 112,
              'asthma': 69,
              'coronary artery disease': 69,
              'myocardial infarction': 34,
              'gastroesophageal reflux disease ': 12,
```

```

        'osteoporosis': 8,
        'atrial fibrillation': 7})

```

```

[23]: total_keys = list(
        set(
            list(w_conditions_names_counter.keys())
            + list(b_conditions_names_counter.keys())
        )
    )
    new_counts = {}
    aa = []
    ca = []
    for k in total_keys:
        # [aa, ca]
        new_counts[k] = [
            b_conditions_names_counter.get(k, 0),
            w_conditions_names_counter.get(k, 0),
        ]
        aa.append(b_conditions_names_counter.get(k, 0))
        ca.append(w_conditions_names_counter.get(k, 0))

    c_table = pd.DataFrame.from_dict(new_counts)
    c_table.rename(index={0: "b.freq"}, inplace=True)
    c_table.rename(index={1: "w.freq"}, inplace=True)
    c_table

```

```

[23]:      type ii diabetes mellitus  osteoporosis  \
b.freq                187                11
w.freq                164                 8

      gastroesophageal reflux disease  hypercholesterolemia  hypertension  \
b.freq                        9                128            3232
w.freq                        12                112            3173

      atrial fibrillation  coronary artery disease  asthma  hyperlipidemia  \
b.freq                5                73        72            2326
w.freq                7                69        69            2326

      osteoarthritis  myocardial infarction
b.freq            264                48
w.freq            286                34

```

```

[24]: class bcolors:
        HEADER = "\033[95m"
        OKBLUE = "\033[94m"
        OKCYAN = "\033[96m"
        OKGREEN = "\033[92m"

```

```

WARNING = "\033[93m"
FAIL = "\033[91m"
ENDC = "\033[0m"
BOLD = "\033[1m"
UNDERLINE = "\033[4m"

```

```

[25]: sig_results = []
# Chi square independence test
# https://www.dir.uniupo.it/pluginfile.php/138296/mod_resource/content/0/
# 22-colloc-bw.pdf
for k in list(set(total_keys)):
    # For AA [Number of instances of current word, Number of instances of all
    # other words]
    x1 = [c_table[k].iloc[0], c_table.iloc[0].sum() - c_table[k].iloc[0]]
    # For CA [Number of instances of current word, Number of instances of all
    # other words]
    y1 = [c_table[k].iloc[1], c_table.iloc[1].sum() - c_table[k].iloc[1]]
    test = scipy.stats.chi2_contingency([x1, y1])
    word = c_table[k].name
    if test.pvalue < 0.05:
        sig_results.append(word)
        print(f"{bcolors.BOLD}Condition: {k}{bcolors.ENDC}")
        print(f"      W      ^W")
        print(f"AA: {x1}")
        print(f"CA: {y1}")
        print(
            f'There {bcolors.OKGREEN}is a significant difference{bcolors.ENDC}
            in the prevalence of the condition "{word}" between the groups with a
            p-value of {bcolors.OKGREEN} "{:0.3f}"{bcolors.ENDC}'.format(test.pvalue)
        )
        print(f"")
    if len(sig_results) == 0:
        print(f'{bcolors.BOLD}{bcolors.FAIL}No significant differences in any
        conditions between groups found{bcolors.ENDC}')

```

No significant differences in any conditions between groups found