In [ ]:
```
import random as rd
import numpy as np
import math
from urllib.request import urlopen
from sklearn import linear_model
from sklearn.linear_model import LogisticRegression
from collections import defaultdict
import matplotlib.pyplot as plt
import pandas as pd
import nltk
import string
from nltk.stem.porter import *

def parse_data(fname):
    for l in open(fname):
        yield eval(l)

clothing_data = list(parse_data("renttherunway_final_data.json"))
```

In [ ]:
```
#Sample data for reference
print(len(clothing_data))
clothing_data[0]
```

In [ ]:
```
#Helper methods

#Convert to height in inches
def convert_height(string):
    height_inches = 0
    feet,inches = string.split()
    feet = int(feet.replace("\'", "" ))
    inches = int(inches.replace("\"", ""))
    height_inches = 12*feet + int(inches)
    return height_inches

def convert_weight(string):
    weight = string.replace("lbs", "")
    return int(weight)
```

In [ ]:
```
'''

Analyze and store data

Datatypes:
-fit: boolean
-user_id: str number
-bust_size: str
-item_id: str number
-weight: str
-rating: str num
-rented for: str
-review_text: str
-body type: str
-review_summary: str
-category: str
-height: str num
-size: int
-age: str num
-date: str

'''

#basic demographics of people who are reviewing, age/weight scatter

#reviews per category counter
category_count = defaultdict(int)
#reason for renting counter
reason_rent = defaultdict(int)
#body type counter
bodytype_count = defaultdict(int)

#average rating per category
average_catrating = defaultdict(int)
#average rating per bodytype
average_bodrating = defaultdict(int)

#store the height of the user
height_user = []
#store the weight of the user
weight_user = []
#store the age of the user
age_user = []

c_data = list()

print(len(clothing_data))
index = 0
for data in clothing_data:
    #if index > 13878:
        #print(data)
    if "body type" in data.keys() and "height" in data.keys() and "weigh
t" in data.keys() and "age" in data.keys() and "rented for" in data.keys
() and data['rating'] != "None":
        if int(data['age']) > 5 and int(data['age']) < 90:
            if convert_weight(data['weight']) < 250:
                c_data.append(data)
```

```
                category_count[data['category']] += 1
                reason_rent[data['rented for']] += 1
                bodytype_count[data['body type']] += 1
                average_catrating[data['category']] += int(data['rating'
])
                average_bodrating[data['body type']] += int(data['ratin
g'])
                height_user.append(convert_height(data['height']))
                weight_user.append(convert_weight(data['weight']))
                age_user.append(int(data['age']))
        index = index + 1
for key in average_catrating:
    average_catrating[key] = (average_catrating[key])/(category_count[ke
y])
for key in average_bodrating:
    average_bodrating[key] = (average_bodrating[key])/(bodytype_count[ke
y])

print(len(height_user))
```

In [ ]:
```
#Averages of people using this dataset, put in table
print("Average age in years:", end ="")
print(sum(age_user)/len(age_user))
print("Average weight in lbs:", end ="")
print(sum(weight_user)/len(weight_user))
print("Average height in inches:", end ="")
print(sum(height_user)/len(weight_user))

print("Max age:", end ="")
print(max(age_user))
print("Min age:", end ="")
print(min(age_user))
print("Max weight:", end ="")
print(max(weight_user))
print("Min weight:", end ="")
print(min(weight_user))
```

In [ ]:
```
#Visualize Data
plt.scatter(age_user, weight_user, color="orange", edgecolor = "black")
plt.title("User Demographic")
plt.xlabel("Age in Years")
plt.ylabel("Weight in lbs")
#plt.xlim(45,250)
plt.show()
```

In [ ]:
```
plt.scatter(age_user, height_user, color="orange", edgecolor = "black")
plt.title("User Demographic")
plt.xlabel("Age in Years")
plt.ylabel("Height in Inches")
#plt.xlim(45,250)
plt.show()
```

In [ ]:
```
keys = []
items = []
for k in average_bodrating:
    keys.append(k.split()[0])
    items.append(average_bodrating[k])



low = min(items)
high = max(items)
plt.ylim([math.ceil(low-0.5*(high-low))-.75, math.ceil(high+0.2*(high-lo
w))-.75])

plt.bar(keys,items, width = 0.5, color='orange', edgecolor='black')
plt.xlabel("Body Type")
plt.ylabel("Average Rating")
plt.title("Average Rating Per Body Type")
plt.show()
```

In [ ]:
```
keys = []
items = []

sorted_dict = sorted(category_count.items(), key=lambda x: x[1], reverse
=True)
for k in sorted_dict[:8]:
    keys.append(k[0])
    items.append(k[1])

print(keys)
print(items)

plt.bar(keys,items, width = 0.5, color='orange', edgecolor='black')
plt.xlabel("Clothing Type")
plt.ylabel("Number of Reviews")
plt.title("Popular Clothing Types")
plt.show()
```

In [ ]:
```
keys = []
items = []


for k in reason_rent:
    if k == 'party: cocktail':
        continue
    else:

        keys.append(k.split()[0])
        items.append(reason_rent[k])

print(keys)
print(items)

plt.bar(keys,items, width = 0.5, color='orange', edgecolor='black')
plt.xlabel("Cause of Rental")
plt.ylabel("Number of Rentals")
plt.title("Reason to Rent")
plt.show()
```

In [ ]:
```
#training, validation, test sets
#https://wordart.com/create
rd.shuffle(c_data)
training_s = c_data[:135000]
validation_s = c_data[135000:145000]
test_s = c_data[145000:152670]
```

In [ ]:
```
print(bodytype_count)
```

In [ ]:
```
print(reason_rent)
```

In [ ]:
```
#baseline models
def baseline1(review):
    r = review.lower()
    prediction = "wedding"
    if "vacation" in r:
        prediction = "vacation"
    elif "formal affair" in r:
        prediction = "formal affair"
    elif "date" in r:
        prediction = "date"
    elif "everyday" in r:
        prediction = "everday"
    elif "party" in r:
        prediction = "party"
    elif "work" in r:
        prediction = "work"
    elif "other" in r:
        prediction = "other"
    return prediction

def baseline2(review):
    prediction = "wedding"
    if review['category'] == "suit" or review['category'] == "blazer":
         prediction = "wedding"
    elif review['category'] == "dress" or review['category'] == "shirtdr
ess":
        rand = rd.random()
        if rand < 0.7:
            prediction = "wedding"
        else:
            prediction = "party"
    elif review['category'] == "romper" :
        prediction = "party"
    elif review['category'] == "shift":
        prediction = "formal affair"
    elif review['category'] == "jumpsuit":
        prediction = "party"
    elif review['category'] == "gown" or review['category'] == "skirt":
        prediction = "formal affair"
    elif review['category'] == "sweater" or review['category'] == "jacke
t" or review['category'] == "pants" or review['category'] == "top":
        prediction = "everyday"

    return prediction
```

In [ ]:
```
#used to store all words encountered in reviews
word_dictionary = defaultdict(int)
#count total number of words
total_words = 0
#used to remove punctuation
punc = string.punctuation
#used to remove numbers
nums = {"0","1","2","3","4","5","6","7","8","9"}
remove_common = {"a","an", "the", "and", "was", "is", "i", "if", "didnt"
, "because", "really", "am", "had", "it", "with", "there", "has", "does"
, "wanted", "youre", "you", "cant", "about", "couldnt", "could", "im",
"too", "or", "to", "this", "but", "for", "in", "of", "my", "so", "on",
"that", "have", "at", "just", "which", "are","still", "from", "wasnt",
"only", "will"}

#populate word dictionary, process text
for data in c_data:
    text = data["review_text"].lower()
    text = ''.join([c for c in text if c not in punc])
    text = ''.join([n for n in text if n not in nums])
    words = text.split()

    for w in words:
        if w not in remove_common:
            total_words += 1
            word_dictionary[w] += 1

#sort the words based on word count
sorted_dict = sorted(word_dictionary.items(), key=lambda x: x[1], revers
e=True)

#list that just contains the words
words = list(word_dictionary.keys())

#list that contains (count, word) pairs, sorted
counts = [(word_dictionary[w], w) for w in words]
counts.sort(reverse=True)

#obtain the top x words in a list
#topx = [c[1] for c in counts[:5000]]

#wordID = dict(zip(topx, range(len(topx))))

#set of topx words
#topXSet = set(topx)
```

In [ ]:
```
sorted_c = sorted(category_count.items(), key=lambda x: x[1], reverse=Tr
ue)
top_sorted = [c[0] for c in sorted_c]
cat_ID = dict(zip(top_sorted, range(len(top_sorted))))
```

In [ ]:
```
```

In [ ]:
```
def encode_category(datum):
    return cat_ID[datum['category']]

def feature(datum, size):
    new_top = [g[1] for g in counts[:size]]
    new_topID = dict(zip(new_top, range(len(new_top))))

    feat = [0]*size
    text = datum["review_text"].lower()
    text = ''.join([c for c in text if c not in punc])
    text = ''.join([n for n in text if n not in nums])
    words = text.split()

    for w in words:
        if w in new_top:
            feat[new_topID[w]] += 1

    #obtains the weights of the top x words
    feat = feat[:size]
    feat = feat + [encode_category(datum)]
    return feat + [1]
```

In [ ]:
```
#validation_labels = [v['rented for'] for v in validation_s]
#base1 = [baseline1(v['review_text']) for v in validation_s]
#base2 = [baseline2(v) for v in validation_s]
#accuracy1 = []
#accuracy2 = []
#for i,p in enumerate(base1):
#    accuracy1.append(base1[i] == validation_labels[i])
#    accuracy2.append(base2[i] == validation_labels[i])
#print("Baseline 1 Performance: ", end = "")
#print(sum(accuracy1)/len(accuracy1))
#print("Baseline 2 Performance: ",end = "")
#print(sum(accuracy2)/len(accuracy2))
```

In [ ]:
```
X = [feature(d,4000) for d in training_s]
Y = [d['rented for'] for d in training_s]
```

In [ ]:
```
model1 = LogisticRegression(C=10**-1, max_iter=12000)
model1.fit(X,Y)
```

In [ ]:
```
validation_predictions = [feature(v,4000) for v in validation_s]
predictions_1 = model1.predict(validation_predictions)
```

In [ ]:
```
validation_labels = [v['rented for'] for v in validation_s]
accuracy1 = []
accuracy2 = []
for i,p in enumerate(predictions_1):
    accuracy1.append(predictions_1[i] == validation_labels[i])

print("Bag of Words 1 Performance: ", end = "")
print(sum(accuracy1)/len(accuracy1))
```

In [ ]:
```
reasons = ["wedding", "formal affair", "party", "everyday", "work", "oth
er", "date", "vacation"]
counter = 0
reason_counter = 0
for r in reasons:
    for i,v in enumerate(predictions_1):
    #print(validation_predictions[i])
        if validation_s[i]['rented for'] == r:
            reason_counter = reason_counter + 1
            if predictions_1[i] == r and validation_s[i]['rented for'] =
= r:
                counter = counter + 1
    print(r)
    print("Precision:", end="")
    precision = counter/len(predictions_1)
    print(counter/len(predictions_1))
    print("Recall:", end="")
    recall = counter/reason_counter
    print(counter/reason_counter)
    print("F1")
    f1 = 2 * ((precision*recall)/(precision+recall))
    print(f1)
    print()
    reason_counter = 0
    counter = 0
```

In [ ]:
```
reasons = ["wedding", "formal affair", "party", "everyday", "work", "oth
er", "date", "vacation"]
counter = 0
reason_counter = 0
for r in reasons:
    for i,v in enumerate(predictions_1):
    #print(validation_predictions[i])
        if validation_s[i]['rented for'] == r:
            reason_counter = reason_counter + 1
            if predictions_1[i] == r and validation_s[i]['rented for'] =
= r:
                counter = counter + 1
    print(r)
    print("Precision:", end="")
    precision = counter/len(predictions_1)
    print(counter/len(predictions_1))
    print("Recall:", end="")
    recall = counter/reason_counter
    print(counter/reason_counter)
    print("F1")
    f1 = 2 * ((precision*recall)/(precision+recall))
    print(f1)
    print(reason_counter)
    print()

    reason_counter = 0
    counter = 0
```

In [ ]:
```
Y = [t['rented for'] for t in training_s]
validation_labels = [v['rented for'] for v in validation_s]

accuracy = []
tuple_accuracy = []
predictions = []

d_sizes = [1000,2000,3000,4000,5000,6000,7000,8000]
c_val = [10**-5,10**-4,10**-3,10**-2,10**-1,0.5,0.75,1,1.5,2,5,10]

for d in d_sizes:
    for c in c_val:
        X = [feature(t,d) for t in training_s]
        model = LogisticRegression(C=c, max_iter=12000)
        model.fit(X,Y)
        validation_predictions = [feature(v,d) for v in validation_s]
        predictions = model.predict(validation_predictions)

        for i,p in enumerate(predictions):
            accuracy.append(predictions[i] == validation_labels[i])
        print("Dictionary Size:", end = "")
        print(d)
        print("C value:", end= "")
        print(c)
        print(sum(accuracy)/len(accuracy))
        a = sum(accuracy)/len(accuracy)
        print()
        tuple_accuracy.append((d,c,a))
        accuracy = []
        predictions = []
        X = []
        validation_predictions = []
```