Chanyoung Lee

Data Analytics Portfolio

- University of Nottingham , MSc Business Analytics
- Year: 2019 - 2020
- Language: Python, PostgreSQL, Tableau
- Dissertation: Decision-making for high-involvement products: Topic modelling using online reviews
- Lectures:
  - Machine Learning and Predictive Analytics
  - Analytics Specialisations and Applications
  - Foundational Business Analytics
  - Data at Scale: Management, Processing and Visualisation

The data analytics portfolio consists of a dissertation and four courseworks.

1. Decision-making for high-involvement products: Topic modelling using online reviews Dissertation

https://github.com/Chan-Young/Coursework/blob/main/NLP%20and%20LDA%20dissdertation.pdf (https://github.com/Chan-Young/Coursework/blob/main/NLP%20and%20LDA%20dissdertation.pdf)

- Year: 2020
- Data: Web scraping of total 965 review data
- Language: Python
- Summary:

Analysed the main topic in the decision-making process of high-involvement products. In this dissertation, I chose Tesla as the high-involvement product. Collected total 965 reveiw data with scores from four websites, and applied NLP's preprocessing and LDA. As a results, I discovered 10 topics.

2. Churn Prediction

https://github.com/Chan-Young/Coursework/blob/main/Classification_Churn%20Prediciton.pdf (https://github.com/Chan-Young/Coursework/blob/main/Classification_Churn%20Prediciton.pdf)

- Year: 2020
- Data: Four store data collected over two years. The data consists of five SQL tables
- Language: Python, PostgreSQL
- Summary:

Interpreted the given graph to set the churn rate as 33 days, and predicted the churn rate using temporal data. Predicted with XGBoost classifier after pre-processing, including SMOTE and standarization. Feature importance and selection were executed using REFCV and RFE, and the randomized search cv was used to find the final hyperparameters. As a results, predicted 51.4% chrun rate.

3. Customer Analytics using K-Means clustering

https://github.com/Chan-Young/Coursework/blob/main/Clustering_%20Customer%20Analytics.pdf (https://github.com/Chan-Young/Coursework/blob/main/Clustering_%20Customer%20Analytics.pdf)

- Year: 2020
- Data: 4 files describing 3000 customers over 6 months
- Language: Python
- Summary:

Perform a market segmentation on a transactional dataset that has been provided by a national convenience storechain (4 files describing 3000 customers over 6 months). Produce profiles for 5-7 customer segments using PCA and K-Mean clustering including statistical summary and a pen profile for each segment.

4. Predicting the potential customers

https://github.com/Chan-Young/Coursework/blob/main/Classification_predict%20customers.pdf (https://github.com/Chan-Young/Coursework/blob/main/Classification_predict%20customers.pdf)

- Year: 2019
- Data: Total 4,000 customer's with 17 features
- Language: Python
- Summary:

Predicting potential customers who will purchase new N/LAB Platinum Deposit. Demographic and personal data that identified in previous a product has been used. With statistical analysis and decision tree algorithms, it discovered the important features and applied serveral classifications usign precision and f1 score as a model evaluation strategy.

5. Customer Analytics by KPIs ComparativeAnalysis

https://github.com/Chan-Young/Coursework/blob/main/Presentation_SQL_coursework.pdf (https://github.com/Chan-Young/Coursework/blob/main/Presentation_SQL_coursework.pdf)

https://github.com/Chan-Young/Coursework/blob/main/KPIs%20comparative%20analysis.pdf (https://github.com/Chan-Young/Coursework/blob/main/KPIs%20comparative%20analysis.pdf)

- Year: 2019
- Data: Four store data collected over two years. The data consists of five SQL tables
- Language: PostgreSQl, Tableau
- Summary:

A comparative analysis of the stores performance in terms of sales and profit relative to the size of the store. Analysed customer behaviours' using six KPIs.

```
(1) Total sales vs Total sales in size
(2) New customers
(3) Active customers
(4) Monthly Sales
(5) Top 3 departments
(6) Top 3 category in dairy depart
```

Masters Dissdertation

Decision-making for high-involvement products: Topic modelling using online reviews

- University of Nottingham (UK), MSc Business Analytics
- Year: 2020
- Language: Python

Topic of Dissertation

Analysed what is the main topic in the decision-making process of high-involvement products. In this dissertation, I chose Tesla as the high-involvement product. Collected total 965 reveiw data with scores from four websites, and after applying NLP's preprocessing and LDA I discovered 10 topics.

The Process of Data Analytics

1. Collect a total of 956 Tesla review data from 4 sites (reviews and horoscopes)
2. Remove URLs and HTML
3. Replace Prouns with the appropriate object name
4. Convert to lowercase (change 'Car' to 'car')
5. Tokenisation
6. Extracting nuns, verbs, adverbs and objectives through Part-of-speech (POS) tag
7. Remove Stop words ('the', 'and' etc.)
8. Modify model name ('model' + 'x' = 'model_x')
9. Replace negative expression ('no', 'nor' with 'not')
10. Lemmatisation: Change to default word based on POS tagging
11. Bigram and trigram: Frequent word combinations
12. Remove Stop words

Ten topics were identified through the gastric preprocessing process, which can be grouped into three groups.

- General discussion (features of vehicle, security, exterior, comparing brands)
- Technology (technology, electric car, high technology car)
- Service (delivery request, delivery process, mobile service)

Dissertation

https://github.com/Chan-Young/Coursework/blob/main/NLP%20and%20LDA%20dissdertation.pdf (https://github.com/Chan-Young/Coursework/blob/main/NLP%20and%20LDA%20dissdertation.pdf)

1. Data Scraping

Sites

1. Cars.com
2. ComsumerAffair
3. Trustpilot

https://www.cars.com/research/tesla/ (https://www.cars.com/research/tesla/)

In [ ]:
```
import csv
import requests
from bs4 import BeautifulSoup
```

In [50]:
```
ratings = []
reviews = []

models = ['x',3, 's']
years = [2012, 2013, 2014, 2015, 2016, 2017, 2018, 2019, 2020]

for model in models:
    for year in years:
        for page in range(1,11):
            url = 'https://www.cars.com/research/tesla-model_{}-{}/consumer-reviews/?pg={}&nr=10'.format(model,year,page)
            headers = {'User-Agent': 'Mozilla/5.0 (Macintosh; Intel Mac OS X 10_11_6) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/61.0.3163.100 Safari/537.36','Accept-Encoding': 'gzip, deflate, br','Accept-Language': 'en-US,en;q=0.9,hi;q=0.8'}
            r = requests.get(url, headers=headers)
            soup = BeautifulSoup(r.text, 'html.parser')

            review_containers = soup.find_all('p', class_ = 'review-card-text')
            for review_container in review_containers:
                try:
                    review = review_container.text.replace("\n", " ")
                except:
                    review = ''
                reviews.append(review)

            containers=soup.find_all('article', attrs = {'ng-controller':'carsResearchConsumerReviewsReviewCardController as ctrl'})
            for container in containers:
                try:
                    rating = container.find('cars-star-rating').text
                    rating = rating[0]
                except:
                    rating = ''
                ratings.append(rating)
```

In [51]:
```
print(len(reviews))
print(len(ratings))
```

```
448
448
```

In [56]:
```
Cars_dic = {'Review':reviews, 'Rating':ratings}
Cars = pd.DataFrame(Cars_dic)
Cars.head()
```

Out[56]:

	Review	Rating
0	This is a great electric SUV....	5
1	The Tesla Model X was one of ...	5
2	Definitely not a cheap vehicl...	5
3	Owned this car for a year and...	5
4	From the head turning falcon ...	5

In [58]:
```
Cars.Rating.value_counts()
```

Out[58]:
```
5    396
4     23
1     16
3      7
2      6
Name: Rating, dtype: int64
```

In [59]:
```
Cars.to_csv(r'C:\Users\chanl\Untitled Folder\Cars.csv', index=False)
```

In [60]:
```
df = pd.read_csv('Cars.csv')
df.head()
```

Out[60]:

	Review	Rating
0	This is a great electric SUV....	5
1	The Tesla Model X was one of ...	5
2	Definitely not a cheap vehicl...	5
3	Owned this car for a year and...	5
4	From the head turning falcon ...	5

2. Preprocessing

0. Package preparation

In [1]:
```
# General
import pandas as pd
import numpy as np
from numpy import array
import matplotlib.pyplot as plt
%matplotlib inline
import missingno as msno
import itertools
from collections import Counter

# Preprocessing
from nltk.tokenize import RegexpTokenizer
import nltk
from nltk.corpus import stopwords
import spacy

import neuralcoref
nlp = spacy.load('en')
neuralcoref.add_to_pipe(nlp, greedyness=0.5,max_dist=50,blacklist=False)
import gensim
```

In [2]:
```
cars = pd.read_csv('cars.csv')
print(len(cars))
print(cars.head())
```

```
448
                                     Review  Rating
0                  This is a great electric SUV....       5
1                  The Tesla Model X was one of ...       5
2                  Definitely not a cheap vehicl...       5
3                  Owned this car for a year and...       5
4                  From the head turning falcon ...       5
```

In [3]:
```
ca = pd.read_csv('Consumer_Affairs.csv')
print(len(ca))
print(ca.head())
```

```
206
                                     Review  Rating
0  Tesla decided they didn't like that I had a di...     1.0
1  Alliant is the finance company that the Tesla ...     1.0
2  Since December of 2019, my 2015 Tesla Model S ...     1.0
3  There is essentially no way to talk to a perso...     1.0
4  I got an alert to replace the small 12 V Batte...     1.0
```

In [4]:
```
t1 = pd.read_csv('Trustpilot1.csv')
print(len(t1))
print(t1.head())
```

```
63
                                     Review  Rating
0                  I bought a tesla 'demo' new c...     1.0
1                  Service at 6692 Auto Center D...     1.0
2                  Tesla service unacceptable,Ca...     1.0
3                  Blowed my husband on our two ...     4.0
4                  I’m so mad at Tesla. Although...     2.0
```

In [5]:
```python
t2 = pd.read_csv('Trustpilot2.csv')
print(len(t2))
print(t2.head())
```

```
241
                                              Review  Rating
0                    I bought a tesla 'demo' new c...     1.0
1                    Tesla did not respond to this...     1.0
2                    Must admit tesla service has ...     1.0
3                    As of this morning, I think I...     1.0
4                    Where to start.  I picked up ...     1.0
```

In [6]:
```python
raw_review = pd.concat([cars, ca, t1, t2])
print(raw_review.head())
print(len(raw_review))
```

```
                                              Review  Rating
0                    This is a great electric SUV....     5.0
1                    The Tesla Model X was one of ...     5.0
2                    Definitely not a cheap vehicl...     5.0
3                    Owned this car for a year and...     5.0
4                    From the head turning falcon ...     5.0
958
```

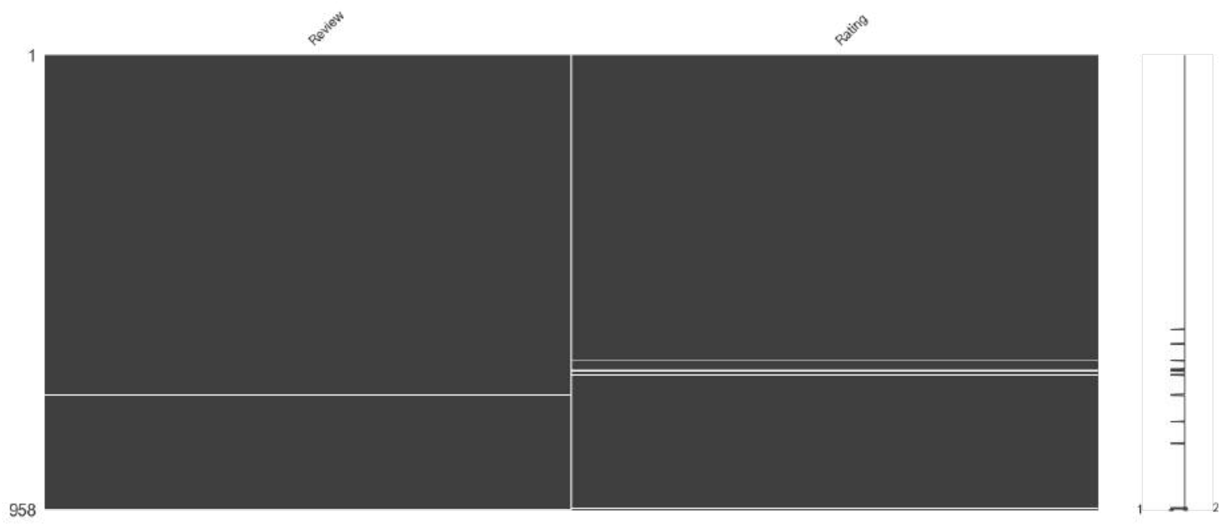
In [ ]:
```python
raw_review.to_csv(r'C:\Users\chanl\Dissertation\raw_review.csv', index=False)
```

In [8]:
```python
raw_review.isnull().sum(axis=0)
```

Out[8]:
```
Review    8
Rating    7
dtype: int64
```

In [9]:
```python
msno.matrix(raw_review)
```

Out[9]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x1aca7167b08>
```



In [10]:
```python
raw_review = raw_review.dropna(axis=0)
```

In [ ]:
```python
raw_review['Review'] = raw_review['Review'].str.strip()
```

In [12]:
```
raw_review.reset_index(drop=True)
```

Out[12]:

	Review	Rating
0	This is a great electric SUV. Tesla has reall...	5.0
1	The Tesla Model X was one of the most over-eng...	5.0
2	Definitely not a cheap vehicle to purchase (ne...	5.0
3	Owned this car for a year and a half and is in...	5.0
4	From the head turning falcon wing doors to the...	5.0
...	...	...
938	Best Green Cars - Our Favorite Cars	1.0
939	Absolutely stunning service & great car!	4.0
940	I have owned a Model S 85D for 14 months and I...	5.0
941	good company and deciding to see if I want to ...	5.0
942	My Tesla Roadster just turned two years old, a...	5.0

943 rows × 2 columns

In [ ]:
```
raw_review.to_csv(r'C:\Users\chanl\Dissertation\review.csv', index=False)
```

In [13]:
```
# Step 1: Import dataset
Review = pd.read_csv('review.csv')
# Convert to array
docs =array(Review['Review'])
type(docs)
```

Out[13]: numpy.ndarray

2. replace all the pronouns in a text with their respective object names

In [14]:
```
def Pronoun(docs):
    for doc in range(len(docs)):
        review = nlp(docs[doc])
        # Step 2: Replacing pronouns to their object names
        resolved_coref = review._.coref_resolved
        docs[doc] = resolved_coref
    return docs
```

In [15]:
```
docs = Pronoun(docs)
```

In [ ]:
```
np.save('pronoun_final',docs)
#docs = np.load('pronoun_final.npy').tolist()
```

In [16]:
```
docs = docs.tolist()
```

3 ~ 5. Lowering case, tokenization, and POS tagging

In [17]:
```
def preprocessing(docs):
    key = []
    tokenizer = RegexpTokenizer(r'\w+')
    for doc in docs:
    # Step 3: Lower case
        doc = doc.lower()
    # Step 4: Tokenization
        doc = tokenizer.tokenize(doc)
    # Step 5: POS tagging
        tag = nltk.pos_tag(doc)
        text = []
        for i in tag:
            if i[1].startswith('V') or i[1].startswith('N') or i[1].startswith(
'R') \
            or i[1].startswith('J'):
                text.append(i[0])

        key.append(text)
    return(key)
```

In [18]:
```
docs_processed = preprocessing(docs)
print(len(docs_processed))
print(docs_processed[0])
print(len(docs_processed[0]))
```

```
943
['is', 'great', 'electric', 'suv', 'tesla', 'has', 'really', 'outdid', 'tesla',
'design', 'performance', 'technology', 'great', 'electric', 'suv', 'offers', 'h
ave', 'model', 's', 'too', 'prefer', 'model', 's', 'model', 's', 'more', 'nimbl
e', 'is', 'much', 'easier', 'car', 'get', 'compare', 'model', 's', 'model',
's', 'got', 'great', 'clearance', 'ground', 'model', 's', 'handles', 'great',
'heavy', 'suv', 'performance', 'incomparable', 'other', 'suv', 'model', 's', 'c
ategory', 'i', 'have', 'person', 'configuration', 'wish', 'i', 'ordered', 'pers
on', 'more', 'cargo', 'space', 'being', 'second', 'row', 'fold', 'down', 'mor
e', 'cargo', 'space', 'overall', 'i', 'think', 'is', 'great', 'car', 'fun', 'dr
ive', 'reliable', 'get', 'notice', 'anywhere', 'everywhere']
86
```

6. Remove stop words

In [19]:
```
stop_words = stopwords.words('english')
stop_words.extend(['’', "'s", 't',"'ve",'x',"'m",'“','”','ve',\
                  '–', '..', '‘','…','r/','ev','•','**',"'re",'...'])
stop_words.remove('no')
stop_words.remove('nor')
stop_words.remove('not')
stop_words.remove('s')
stop_words.remove('x')
```

In [20]:
```
# Step 6: Remove stop words
docs_stopword = []
for doc in docs_processed:
    stop = [wd for wd in doc if wd not in stop_words]
    docs_stopword.append(stop)
```

In [21]:
```
print(docs_stopword[0])
```

```
['great', 'electric', 'suv', 'tesla', 'really', 'outdid', 'tesla', 'design', 'p
erformance', 'technology', 'great', 'electric', 'suv', 'offers', 'model', 's',
'prefer', 'model', 's', 'model', 's', 'nimble', 'much', 'easier', 'car', 'get',
'compare', 'model', 's', 'model', 's', 'got', 'great', 'clearance', 'ground',
'model', 's', 'handles', 'great', 'heavy', 'suv', 'performance', 'incomparabl
e', 'suv', 'model', 's', 'category', 'person', 'configuration', 'wish', 'ordere
d', 'person', 'cargo', 'space', 'second', 'row', 'fold', 'cargo', 'space', 'ove
rall', 'think', 'great', 'car', 'fun', 'drive', 'reliable', 'get', 'notice', 'a
nywhere', 'everywhere']
```

7. Combine two words into a single word

In [22]:
```
show1 = []
word_1 = ['model','model']
word_2 = ['s','x']

for wd1 in word_1:
    for wd2 in word_2:
        for re in docs_stopword:
            for i,j in enumerate(re):
                if j == wd1:
                    try:
                        # Step 7: Combine two words into a single word
                        if re[i+1] == wd2:
                            re[i] = (wd1 + '_'+ wd2)
                            re.pop(i+1)
                            show1.append(re[i])
                    except:
                        pass
print(len(show1))
print(show1[:2])
```

```
316
['model_s', 'model_s']
```

8. Replacement common negatives of words by prefixing a 'not' to the token words that follow

In [23]:
```
show3 = []
negs = ['none', 'never','no',"n't",'not']

for re in docs_stopword:
    for i,j in enumerate(re):
        if j in negs:
            try:
                re[i] = 'not'
                # Step 8: Replacement common negative words
                re[i] = (re[i] + '_'+ re[i+1])
                re.pop(i+1)
                show3.append(re[i])
            except:
                pass

print(len(show3))
print(show3[:2])
```

```
911
['not_cheap', 'not_pay']
```

9. Lemmatization

In [24]:
```
def lemmatization(texts, allowed_postags=['NOUN', 'ADJ', 'VERB', 'ADV']):
    """https://spacy.io/api/annotation"""
    texts_out = []
    for sent in texts:
        doc = nlp(" ".join(sent))
        texts_out.append([token.lemma_ for token in doc if token.pos_ in allowe
d_postags])
    return texts_out
```

In [25]:
```
# Step 9: Lemmatization
lemmatized = lemmatization(docs_stopword, allowed_postags=['NOUN', 'ADJ', 'VER
B', 'ADV'])
```

In [ ]:
```
np.save('lemmatized_final', lemmatized)
#lemmatized = np.load('lemmatized2.npy', allow_pickle=True).tolist()
```

In [27]:
```
print(lemmatized[5])
```

```
['perfect', 'combination', 'performance', 'intelligence', 'safety', 'healthines
s', 's', 'actually', 'reliable', 'hear', 'issue', 'minor', 'easy', 'fix']
```

In [28]:
```
for doc in lemmatized:
    lemmatized = [[token for token in doc if len(token) > 2] for doc in lemmati
zed]
```

In [29]:
```
print(lemmatized[5])
```

```
['perfect', 'combination', 'performance', 'intelligence', 'safety', 'healthines
s', 'actually', 'reliable', 'hear', 'issue', 'minor', 'easy', 'fix']
```

10. Bigram and trigram using genism

In [30]:
```
bigram = gensim.models.Phrases(lemmatized, min_count=10, threshold=5)
trigram = gensim.models.Phrases(bigram[lemmatized], threshold=5)

bigram_mod = gensim.models.phrases.Phraser(bigram)
trigram_mod = gensim.models.phrases.Phraser(trigram)
```

In [31]:
```
def make_bigrams(texts):
    return [bigram_mod[doc] for doc in texts]
```

In [32]:
```
def make_trigrams(texts):
    return [trigram_mod[bigram_mod[doc]] for doc in texts]
```

In [33]:
```
# Step 10: Bigram and trigram
data_words_bigrams = make_bigrams(lemmatized)
data_words_trigrams = make_trigrams(lemmatized)
```

11. Remove stop words again

In [34]:
```
stop_words = stopwords.words('english')
stop_words.extend(['’', "'s", 't',"'ve",'x',"'m",'“','”','ve',\
                   '–', '..', '‘','…','r/','ev','•','**',"'re",'...'])
```

In [35]:
```
final = []
# Step 11: Remove stop words
for i in data_words_trigrams:
    stop = [wd for wd in i if wd not in stop_words]
    final.append(stop)
```

In [36]:
```
final[5]
```

Out[36]:
```
['perfect',
 'combination',
 'performance',
 'intelligence',
 'safety',
 'healthiness',
 'actually',
 'reliable',
 'hear',
 'issue',
 'minor',
 'easy',
 'fix']
```

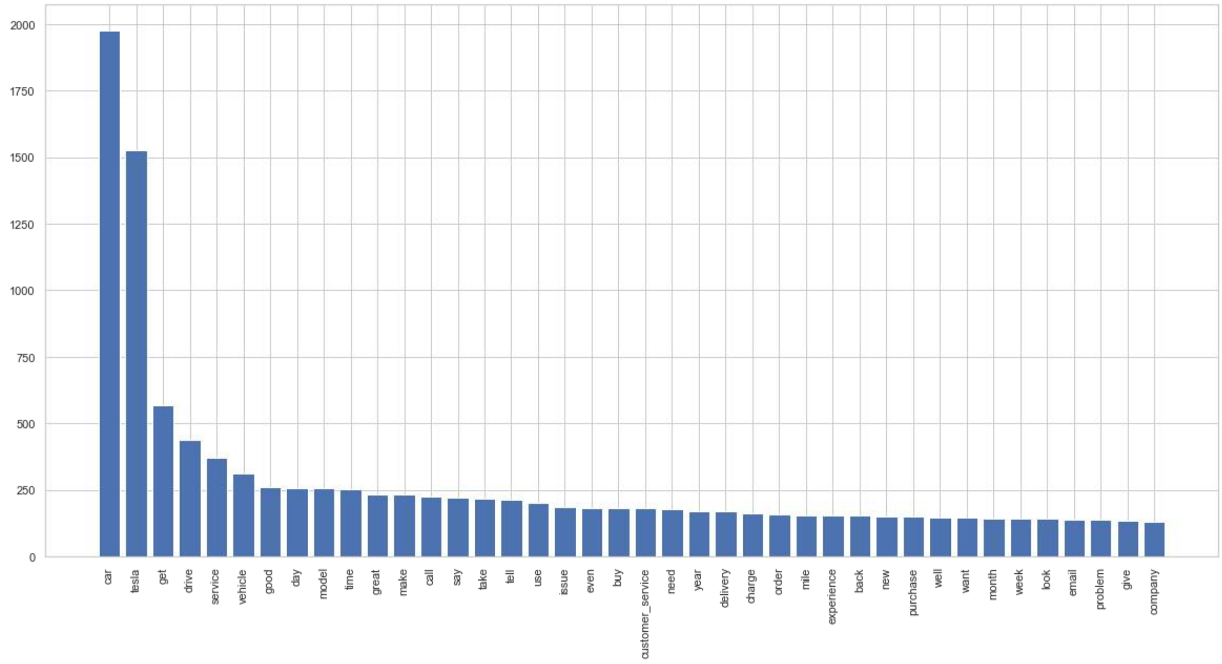
In [ ]:
```
np.save('final_final', final)
#final = np.load('final_final.npy', allow_pickle=True).tolist()
```

In [37]:
```
all_words = list(itertools.chain(*final))
print(len(all_words))

counter = Counter(all_words)
print(len(counter))
```

```
46484
5251
```

In [38]:
```
w = dict(counter.most_common(40))
```

In [39]:
```
plt.figure(figsize=(20,10))
plt.bar(w.keys(), w.values())
plt.xticks(rotation='vertical')
plt.subplots_adjust(bottom=0.15)
plt.show()
```



Step 12: Proning

Removing rare and common tokens using Gensim's dictionary with filter_extremes. Value pairs with less than 2 occurrence or more than 10% of total number of sample is removed.

3. LDA Analysis

---

0. Package preparation

In [12]:
```
# General
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
from numpy import array

# Modelling
import gensim
from gensim.corpora.dictionary import Dictionary
from gensim import corpora
from gensim import models
from pprint import pprint
from gensim.models.coherencemodel import CoherenceModel
import tqdm

# Visualisation
import seaborn as sns
import pyLDAvis.gensim
import pickle
import pyLDAvis
```

1. Data preparation for the LDA analysis

In [13]:
```
final = np.load('final_final.npy', allow_pickle=True).tolist()
```

In [14]:
```
final[0][:10]
```

Out[14]:
```
['great',
 'electric',
 'suv',
 'tesla',
 'really',
 'outdid',
 'tesla',
 'design',
 'performance',
 'technology']
```

In [15]:
```
# Create a dictionary representation of the documents.
dictionary = Dictionary(final)

# Step 12: Remove rare & common tokens
# We filter our dict to remove key :
#value pairs with less than 2 occurrence or more than 10% of total number of sa
mple
dictionary.filter_extremes(no_below=2, no_above=0.1)

#Create dictionary and corpus required for Topic Modeling
corpus = [dictionary.doc2bow(doc) for doc in final]
print('Number of unique tokens: %d' % len(dictionary))
print('Number of documents: %d' % len(corpus))
print(corpus[:1])
```

```
Number of unique tokens: 2765
Number of documents: 943
[[(0, 1), (1, 2), (2, 1), (3, 1), (4, 1), (5, 1), (6, 1), (7, 1), (8, 2), (9,
1), (10, 1), (11, 1), (12, 1), (13, 1), (14, 1), (15, 1), (16, 1), (17, 1), (1
8, 1), (19, 1), (20, 2), (21, 1), (22, 1), (23, 1), (24, 1), (25, 1), (26, 4),
(27, 1), (28, 1), (29, 1)]]
```

In [16]:
```
temp = dictionary[0]
id2word = dictionary.id2token
```

2. Base model of the LDA analysis

In [17]:
```
# Build LDA model
lda_model = gensim.models.LdaMulticore(corpus=corpus,
                                       id2word=id2word,
                                       num_topics=10,
                                       random_state=42,
                                       chunksize=100,
                                       passes=20,
                                       iterations=100,
                                       per_word_topics=True,
                                       eval_every=1,
                                       decay=0.5,
                                       offset=64)
```

In [18]:
```python
# Print the Keyword in the 10 topics
pprint(lda_model.print_topics())
doc_lda = lda_model[corpus]
```

```
[(0,
  '0.008*"autopilot" + 0.007*"cost" + 0.007*"gas" + 0.006*"feature" + '
  '0.006*"handle" + 0.006*"much" + 0.006*"comfortable" + 0.006*"acceleration" '
  '+ 0.006*"seat" + 0.006*"interior"'),
 (1,
  '0.013*"appt" + 0.013*"sister" + 0.011*"part" + 0.010*"guy" + 0.009*"email" '
  '+ 0.008*"item" + 0.008*"uber" + 0.007*"tell_mobile_service" + 0.007*"right"
'
  '+ 0.006*"arrive"'),
 (2,
  '0.013*"email" + 0.012*"delivery" + 0.008*"ask" + 0.007*"customer" + '
  '0.006*"pay" + 0.006*"people" + 0.006*"receive" + 0.006*"problem" + '
  '0.005*"phone" + 0.005*"sale"'),
 (3,
  '0.014*"tire" + 0.011*"delivery" + 0.008*"june" + 0.007*"appointment" + '
  '0.007*"scratch" + 0.007*"service_center" + 0.007*"price" + 0.006*"march" + '
  '0.005*"sale" + 0.005*"side"'),
 (4,
  '0.024*"tesla_solar_panel" + 0.011*"range" + 0.010*"add" + 0.010*"home" + '
  '0.009*"awesome" + 0.008*"already" + 0.008*"roof" + 0.008*"crap" + '
  '0.008*"update" + 0.007*"standard"'),
 (5,
  '0.015*"window" + 0.009*"customer" + 0.008*"end" + 0.007*"leave" + '
  '0.006*"malfunction" + 0.006*"cost" + 0.006*"rear" + 0.006*"bad" + '
  '0.006*"bag" + 0.006*"police"'),
 (6,
  '0.024*"tyre" + 0.013*"dear_tesla" + 0.008*"fabulous" + 0.008*"person" + '
  '0.008*"tech" + 0.007*"email" + 0.007*"test_drive" + 0.007*"design" + '
  '0.006*"store" + 0.006*"avoid"'),
 (7,
  '0.011*"wife" + 0.008*"bad" + 0.007*"battery" + 0.007*"find" + '
  '0.007*"someone" + 0.006*"seem" + 0.006*"people" + 0.006*"customer" + '
  '0.006*"tesla_employee" + 0.006*"service_centre"'),
 (8,
  '0.021*"wheel" + 0.017*"tire" + 0.015*"crack" + 0.013*"technician" + '
  '0.013*"last" + 0.010*"state" + 0.010*"front" + 0.009*"replace" + '
  '0.009*"absolutely" + 0.009*"repair"'),
 (9,
  '0.011*"problem" + 0.011*"phone" + 0.009*"customer" + 0.007*"sale" + '
  '0.007*"service_centre" + 0.007*"staff" + 0.007*"still" + 0.007*"think" + '
  '0.006*"feel" + 0.006*"change"')]
```

In [19]:
```python
# Compute Coherence Score using c_v
coherence_model_lda = CoherenceModel(model=lda_model, texts=final, dictionary=d
ictionary, coherence='c_v')
coherence_lda = coherence_model_lda.get_coherence()
print('\nCoherence Score: ', coherence_lda)
```

```
Coherence Score:  0.3598303372904272
```

3. Hyper-parameter tuning

In [20]:
```
# supporting function
def compute_coherence_values(corpus, dictionary, k, a, b):

    lda_model = gensim.models.LdaMulticore(corpus=corpus,
                                           id2word=id2word,
                                           num_topics=10,
                                           random_state=42,
                                           chunksize=100,
                                           passes=20,
                                           iterations=100,
                                           eval_every=1,
                                           decay=0.5,
                                           offset=64,
                                           per_word_topics=True,
                                           alpha=a,
                                           eta=b)

    coherence_model_lda = CoherenceModel(model=lda_model, texts=final, dictiona
ry=dictionary, coherence='c_v')

    return coherence_model_lda.get_coherence()
```

In [ ]:
```
grid = {}
grid['Validation_Set'] = {}

# Topics range
min_topics = 10
max_topics = 101
step_size = 10
topics_range = range(min_topics, max_topics, step_size)

# Alpha parameter
alpha = list(np.arange(0.01, 1, 0.3))
alpha.append('symmetric')
alpha.append('asymmetric')

# Beta parameter
beta = list(np.arange(0.01, 1, 0.3))
beta.append('symmetric')

# Validation sets
num_of_docs = len(corpus)
corpus_sets = [# gensim.utils.ClippedCorpus(corpus, num_of_docs*0.25),
               # gensim.utils.ClippedCorpus(corpus, num_of_docs*0.5),
               gensim.utils.ClippedCorpus(corpus, int(num_of_docs*0.75)),
               corpus]
corpus_title = ['75% Corpus', '100% Corpus']
model_results = {'Validation_Set': [],
                 'Topics': [],
                 'Alpha': [],
                 'Beta': [],
                 'Coherence': []
                }
```

In [ ]:
```
# Can take a long time to run
if 1 == 1:
    pbar = tqdm.tqdm(total=540)

    # iterate through validation corpuses
    for i in range(len(corpus_sets)):
        # iterate through number of topics
        for k in topics_range:
            # iterate through alpha values
            for a in alpha:
                # iterare through beta values
                for b in beta:
                    # get the coherence score for the given parameters
                    cv = compute_coherence_values(corpus=corpus_sets[i],
                                                  dictionary=dictionary, k=k, a
=a, b=b)

                    # Save the model results
                    model_results['Validation_Set'].append(corpus_title[i])
                    model_results['Topics'].append(k)
                    model_results['Alpha'].append(a)
                    model_results['Beta'].append(b)
                    model_results['Coherence'].append(cv)

                    pbar.update(1)
    pd.DataFrame(model_results).to_csv('lda_tuning_results_final.csv', index=Fa
lse)
    pbar.close()
```

In [21]:
```
lda_tuning = pd.read_csv('lda_tuning_results_final.csv')
lda_tuning_100 = lda_tuning.groupby(lda_tuning.Validation_Set)
lda_tuning_100 = lda_tuning_100.get_group('100% Corpus')
lda_tuning_100 = lda_tuning_100.sort_values(by='Coherence', ascending=False)
lda_tuning_100.head(3)
```

Out[21]:

	Validation_Set	Topics	Alpha	Beta	Coherence
358	100% Corpus	20	asymmetric	0.9099999999999999	0.429508
498	100% Corpus	70	0.9099999999999999	0.9099999999999999	0.424328
438	100% Corpus	50	0.9099999999999999	0.9099999999999999	0.422660

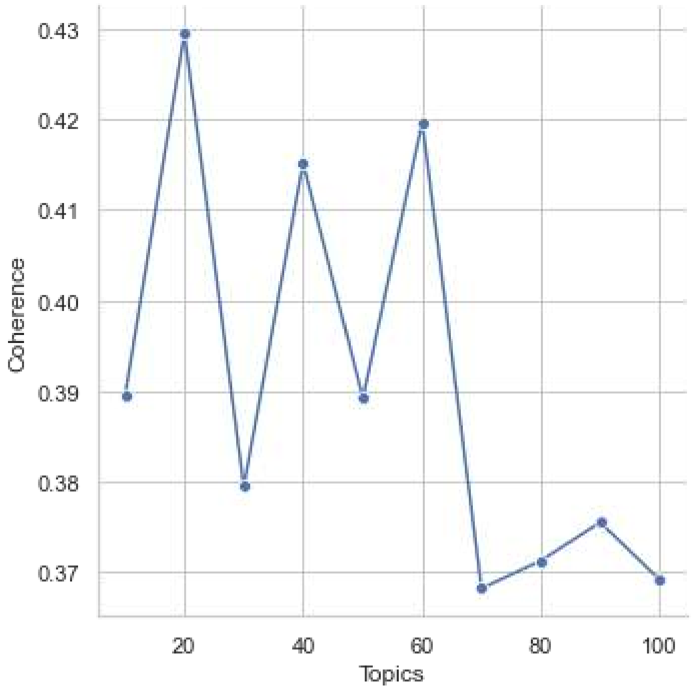
In [22]:
```
results = lda_tuning_100.groupby(lda_tuning_100.Alpha)
results = results.get_group('asymmetric')
results = results.groupby(results.Beta)
results = results.get_group('0.9099999999999999')
```

In [23]:
```
results
```

Out[23]:

	Validation_Set	Topics	Alpha	Beta	Coherence
358	100% Corpus	20	asymmetric	0.9099999999999999	0.429508
478	100% Corpus	60	asymmetric	0.9099999999999999	0.419709
418	100% Corpus	40	asymmetric	0.9099999999999999	0.415211
328	100% Corpus	10	asymmetric	0.9099999999999999	0.389568
448	100% Corpus	50	asymmetric	0.9099999999999999	0.389249
388	100% Corpus	30	asymmetric	0.9099999999999999	0.379484
568	100% Corpus	90	asymmetric	0.9099999999999999	0.375514
538	100% Corpus	80	asymmetric	0.9099999999999999	0.371281
598	100% Corpus	100	asymmetric	0.9099999999999999	0.369199
508	100% Corpus	70	asymmetric	0.9099999999999999	0.368219

In [24]:
```
line = sns.relplot('Topics','Coherence', kind='line', marker='o', data=results)
```



4. Hyper-parameter tuning with narrowed range of the number of topics

In [25]:

```python
grid = {}
grid['Validation_Set'] = {}

# Topics range
min_topics = 10
max_topics = 51
step_size = 10
topics_range = range(min_topics, max_topics, step_size)

# Alpha parameter
alpha = list(np.arange(0.01, 1, 0.3))
alpha.append('symmetric')
alpha.append('asymmetric')

# Beta parameter
beta = list(np.arange(0.01, 1, 0.3))
beta.append('symmetric')

# Validation sets
num_of_docs = len(corpus)
corpus_sets = [# gensim.utils.ClippedCorpus(corpus, num_of_docs*0.25),
               # gensim.utils.ClippedCorpus(corpus, num_of_docs*0.5),
               gensim.utils.ClippedCorpus(corpus, int(num_of_docs*0.75)),
               corpus]
corpus_title = ['75% Corpus', '100% Corpus']
model_results = {'Validation_Set': [],
                 'Topics': [],
                 'Alpha': [],
                 'Beta': [],
                 'Coherence': []
                }
```

In [ ]:

```python
# Can take a long time to run
if 1 == 1:
    pbar = tqdm.tqdm(total=540)

    # iterate through validation corpuses
    for i in range(len(corpus_sets)):
        # iterate through number of topics
        for k in topics_range:
            # iterate through alpha values
            for a in alpha:
                # iterare through beta values
                for b in beta:
                    # get the coherence score for the given parameters
                    cv = compute_coherence_values(corpus=corpus_sets[i],
                                                  dictionary=dictionary, k=k, a
=a, b=b)

                    # Save the model results
                    model_results['Validation_Set'].append(corpus_title[i])
                    model_results['Topics'].append(k)
                    model_results['Alpha'].append(a)
                    model_results['Beta'].append(b)
                    model_results['Coherence'].append(cv)

                    pbar.update(1)
    pd.DataFrame(model_results).to_csv('lda_tuning_results_final2.csv', index=F
alse)
    pbar.close()
```

```
In [26]: lda_tuning2 = pd.read_csv('lda_tuning_results_final2.csv')
         lda_tuning2_100 = lda_tuning2.groupby(lda_tuning2.Validation_Set)
         lda_tuning2_100 = lda_tuning2_100.get_group('100% Corpus')
         lda_tuning2_100 = lda_tuning2_100.sort_values(by='Coherence', ascending=False)
         lda_tuning2_100.head(3)
```

Out[26]:

	Validation_Set	Topics	Alpha	Beta	Coherence
168	100% Corpus	10	0.9099999999999999	0.9099999999999999	0.428507
177	100% Corpus	10	asymmetric	0.61	0.413098
258	100% Corpus	40	0.9099999999999999	0.9099999999999999	0.412249

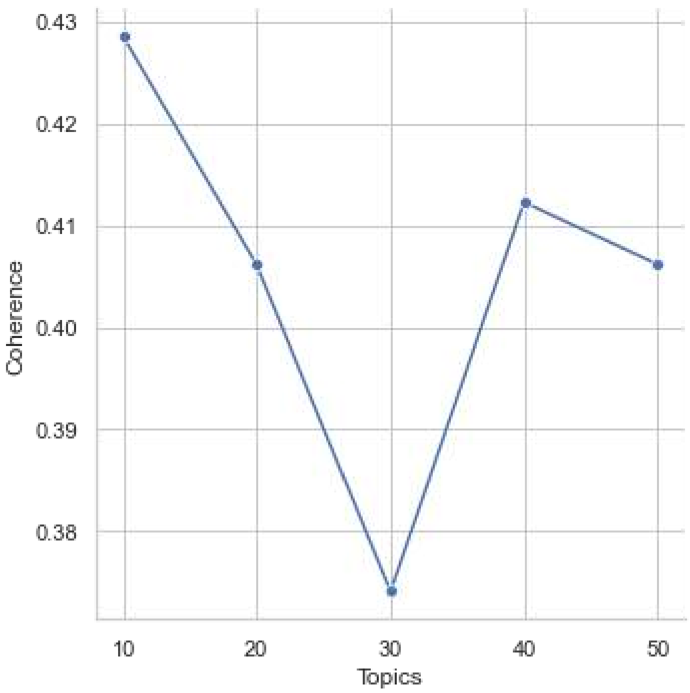
```
In [27]: results2 = lda_tuning2_100.groupby(lda_tuning2_100.Alpha)
         results2 = results2.get_group('0.9099999999999999')
         results2 = results2.groupby(results2.Beta)
         results2 = results2.get_group('0.9099999999999999')
```

```
In [28]: results2
```

Out[28]:

	Validation_Set	Topics	Alpha	Beta	Coherence
168	100% Corpus	10	0.9099999999999999	0.9099999999999999	0.428507
258	100% Corpus	40	0.9099999999999999	0.9099999999999999	0.412249
288	100% Corpus	50	0.9099999999999999	0.9099999999999999	0.406163
198	100% Corpus	20	0.9099999999999999	0.9099999999999999	0.406116
228	100% Corpus	30	0.9099999999999999	0.9099999999999999	0.374115

```
In [29]: line2 = sns.relplot('Topics','Coherence', kind='line', marker='o',data=results2
         )
```



5. Final Model

In [30]:
```
lda_model = gensim.models.LdaMulticore(corpus=corpus,
                                       id2word=id2word,
                                       num_topics=10,
                                       random_state=42,
                                       chunksize=100,
                                       passes=20,
                                       iterations=100,
                                       eval_every=1,
                                       decay=0.5,
                                       offset=64,
                                       per_word_topics=True,
                                       alpha=0.9099999999999999,
                                       eta=0.9099999999999999)
```

In [31]:
```
lda_model.print_topics()
```

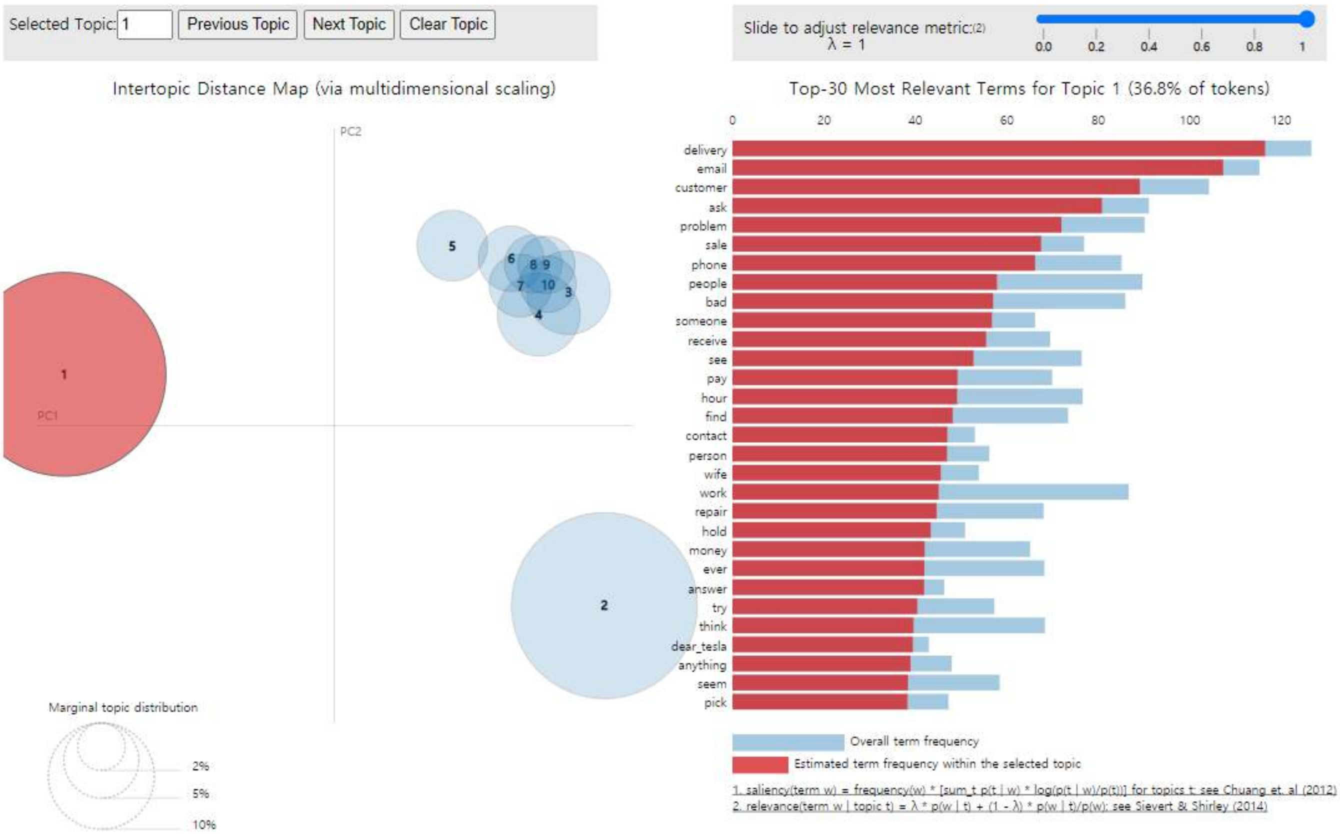
Out[31]:
```
[(0,
  '0.002*"ford" + 0.002*"smog_producer" + 0.001*"nothing" + 0.001*"gas" + 0.001
*"ever" + 0.001*"honda" + 0.001*"friendly" + 0.001*"comfortable" + 0.001*"drivi
ng_experience" + 0.001*"manual"'),
 (1,
  '0.006*"appt" + 0.006*"part" + 0.005*"uber" + 0.005*"guy" + 0.004*"item" + 0.
004*"arrive" + 0.004*"book" + 0.003*"tell_mobile_service" + 0.003*"pay" + 0.003
*"right"'),
 (2,
  '0.010*"delivery" + 0.009*"email" + 0.008*"customer" + 0.007*"ask" + 0.006*"p
roblem" + 0.006*"sale" + 0.006*"phone" + 0.005*"people" + 0.005*"bad" + 0.005
*"someone"'),
 (3,
  '0.010*"june" + 0.008*"price" + 0.006*"march" + 0.006*"text" + 0.005*"custome
r" + 0.005*"promise" + 0.005*"trade" + 0.004*"delivery" + 0.004*"offer" + 0.004
*"reserve"'),
 (4,
  '0.003*"oscar" + 0.003*"auto" + 0.002*"auburn_way" + 0.002*"high_tech_car" +
0.002*"thank" + 0.001*"patient" + 0.001*"help" + 0.001*"card" + 0.001*"professi
onal" + 0.001*"steep"'),
 (5,
  '0.005*"bag" + 0.004*"police" + 0.003*"laptop" + 0.003*"return" + 0.003*"leav
e" + 0.003*"staff" + 0.003*"ask" + 0.002*"unhelpful" + 0.002*"safe" + 0.002*"th
ing"'),
 (6,
  '0.014*"tyre" + 0.005*"fabulous" + 0.002*"hour" + 0.002*"change" + 0.002*"min
ute" + 0.002*"supply" + 0.002*"exterior_look" + 0.002*"entire" + 0.002*"else" +
0.001*"let"'),
 (7,
  '0.006*"world" + 0.005*"electric_car" + 0.005*"elon_musk" + 0.004*"much" + 0.
004*"fuel" + 0.004*"minor" + 0.004*"tech" + 0.003*"excellent" + 0.003*"design"
+ 0.003*"interior"'),
 (8,
  '0.009*"tire" + 0.006*"autopilot" + 0.006*"battery" + 0.005*"cost" + 0.005*"r
ange" + 0.005*"seat" + 0.005*"wheel" + 0.005*"feel" + 0.004*"free" + 0.004*"fea
ture"'),
 (9,
  '0.008*"change" + 0.007*"technology" + 0.005*"door" + 0.005*"owner" + 0.005
*"replace" + 0.004*"door_handle" + 0.004*"sensor" + 0.004*"open" + 0.003*"brea
k" + 0.003*"fantastic"')]
```

6. Visualisation

```
In [32]: pyLDAvis.enable_notebook()
         LDAvis= pyLDAvis.gensim.prepare(lda_model, corpus, dictionary)
```

```
In [ ]: LDAvis
```

Out[33]:




```
In [34]: topic = [' Delivery request','General features','Technology','Eletric car',
                  'Delivery process','Mobile service','Security',
                 'Exterior look','High tech car','Comparing']
         Percent_of_tokens = [36.9, 29.5, 7.1, 6, 4.2, 4, 3.6, 2.9, 2.9, 2.8]
```

```
In [35]: Topic_percent = pd.DataFrame(list(zip(topic, Percent_of_tokens)),
                        columns =['Topic', 'Percent of tokens(%)'])
         Topic_percent
```

Out[35]:

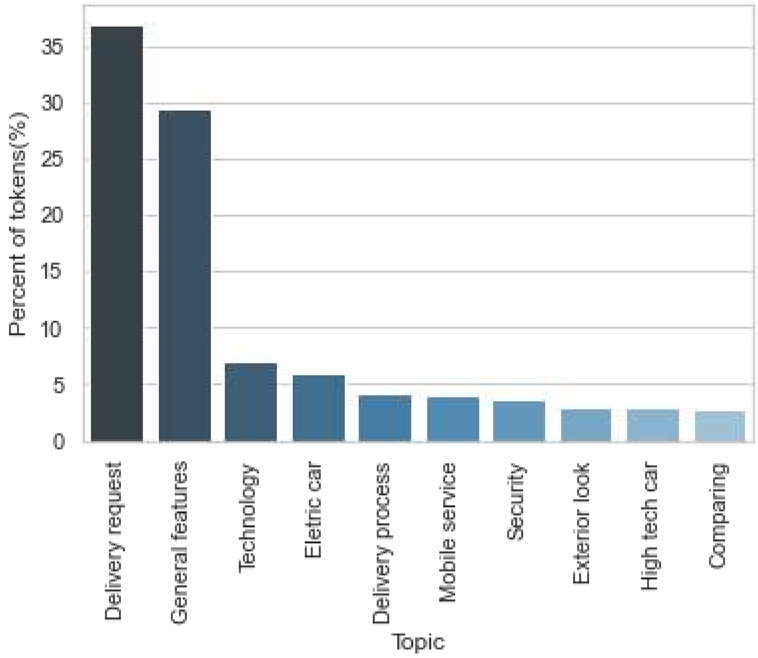
	Topic	Percent of tokens(%)
0	Delivery request	36.9
1	General features	29.5
2	Technology	7.1
3	Eletric car	6.0
4	Delivery process	4.2
5	Mobile service	4.0
6	Security	3.6
7	Exterior look	2.9
8	High tech car	2.9
9	Comparing	2.8

In [36]:

```
topic_percent = sns.barplot('Topic','Percent of tokens(%)',
                            palette='Blues_d', data=Topic_percent)
plt.xticks(rotation=90)
```

Out[36]:

```
(array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9]),
 <a list of 10 Text major ticklabel objects>)
```



In [ ]:

Coursework: Churn Prediciton

- University of Nottingham (UK), MSc Business Analytics
- Lecture: Machine Learning and Predictive Analytics
- Year: 2020
- Language:Python, PostgreSQL

The Problem

The system they want will predict which customers will churn or not (binary prediction) and will be re-run at the same time each week

1. Interpret the initial work done on defining churn and finalize a definition of churn for the company.
2. Create and evaluate a churn prediction system using temporal data.
3. Insights into what differentiates people who churn vs. those that stay, including pen portraits

Four store data collected over two years are given. The data consists offive SQL tables, with the table name as shown below.

- Customers (id, born, name)
- Products (code and details of product, department, category and sub category)
- Receipt lines (receipt id, product code, price, quantity)
- Receipts (receipt id, time, id, store code)
- Stores (informations about stores)

The Processo of Data Analytics

Executive Summary

The Churn rate is 33 days, and the tumbling window size and output window size are 33 days. After processing, XGboost algorithm was used to predict a 51.4% churn rate.

Definition of Churn

nterpret the given graph and define churn. Churn definition as 33 days can be construed as 59.88 per cent of customers visit less than this in median and Foodcorp can expect to target 19.03 per cent of active customers with a perfect classifier.

Churn Predicton model

Describe the selected features and create a predictive model.

- Processing: balancing an output feature in the raining dataset using SMOTE and standardization of each traditional numerical variables and temporal variables.
- Feature importance & selection: Applied RFECV (Recursive feature elimination cross-validation) and RFE.
- Randomized search CV to find and predict the hyperparameters of XGBoost classifier.

Insights

Compared and analyzed Churner and non-churner, and present a marketing strategy. The major difference between the two groups was total purchase expenditure and quantity, and total purchase expenditure for a specific period of time. Based on this, the Churner Group established bounce back marketing strategies such as discount coupons and upselling, while the Non-churner Group established loyalty program marketing strategies such as special product rewards.

Report

https://github.com/Chan-Young/Coursework/blob/main/Classification_Churn%20Prediciton.pdf
(https://github.com/Chan-Young/Coursework/blob/main/Classification_Churn%20Prediciton.pdf)

A. Package preparation

In [1]:
```
# General
import psycopg2
from matplotlib import style
plt.style.use('ggplot')
mpl.rcParams['axes.unicode_minus'] = False
import warnings
warnings.filterwarnings(action='ignore')
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
from matplotlib.legend_handler import HandlerLine2D
from collections import Counter
from numpy import where
from texttable import Texttable

# Preprocessing
from imblearn.over_sampling import SMOTE
from imblearn.over_sampling import BorderlineSMOTE
from imblearn.over_sampling import SVMSMOTE
from imblearn.over_sampling import ADASYN
from imblearn.over_sampling import KMeansSMOTE
from imblearn.over_sampling import RandomOverSampler
from imblearn.over_sampling import SMOTENC
from imblearn.combine import SMOTEENN
from imblearn.combine import SMOTETomek
from imblearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler

# Modelling
from sklearn.svm import SVR
from sklearn.neighbors import KNeighborsClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.svm import LinearSVC
from sklearn.svm import SVC
from sklearn.gaussian_process import GaussianProcessClassifier
from sklearn.gaussian_process.kernels import RBF
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier, AdaBoostClassifier, Gradie
ntBoostingClassifier
from sklearn.neural_network import MLPClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.discriminant_analysis import QuadraticDiscriminantAnalysis
from sklearn.ensemble import GradientBoostingClassifier
import xgboost as xgb
from sklearn.model_selection import RandomizedSearchCV, GridSearchCV

# Metrics
from sklearn.metrics import accuracy_score
from sklearn.metrics import classification_report
from sklearn.metrics import f1_score
from sklearn.metrics import confusion_matrix
from yellowbrick.classifier import ROCAUC
from sklearn.metrics import roc_auc_score
from sklearn.metrics import roc_curve, auc

# Feature Importance
from sklearn.feature_selection import GenericUnivariateSelect
from sklearn.feature_selection import mutual_info_classif
from xgboost import plot_importance
import eli5
```

```python
from eli5.sklearn import PermutationImportance
from sklearn.feature_selection import RFE
from yellowbrick.model_selection import RFECV
from sklearn.inspection import plot_partial_dependence
```

B. Data preparation

1. Importing data

1) Account

In [2]:

```python
user = 'lixcl68'
db_ip = '10.158.72.132'
pw = 'Lcyg1a1n1g1!'
```

2) Importing data from database

In [3]:
```
def get_dataset_value( reference_day=576, tumbling_window_size = 33, output_window_size = 33, num_periods = 11, window_agg_fun = 'SUM', output_agg_fun = 'SUM'):
    sql_top = """
    SELECT customer_id,
           sum(values) as total_values,
           sum(quantity) as total_quantity,
           sum(between) / count(between) as avg_between,
           last_purchased,
    %(ref_date)s::INT AS ref_day,
    {0}(CASE WHEN day > %(ref_date)s::INT AND day <= %(ref_date)s::INT + %(ows)s::INT THEN values ELSE 0 END) as output_feature,
    {1}(CASE WHEN day > %(ref_date)s::INT -%(ws)s::INT AND day <= %(ref_date)s::INT THEN values ELSE 0 END ) as f1,
    """.format(output_agg_fun, window_agg_fun)

    sql = sql_top

    for i in range(1,num_periods):
           sql += "{2}(CASE WHEN day > %(ref_date)s::INT -%(ws)s::INT*({0}+1) AND day <= %(ref_date)s::INT-%(ws)s::INT*({0}) THEN values ELSE 0 END ) as f{1},\n".format(i, i+1, window_agg_fun)

    sql_bottom = """
    FROM final
    WHERE customer_id in (
                    SELECT customer_id
                    FROM final
                    WHERE day > %(ref_date)s::INT - %(ows)s::INT  and day <= %(ref_date)s::INT
                        )
    GROUP BY customer_id, last_purchased
    """
    sql = sql[:-2] + sql_bottom

    with psycopg2.connect("host='{}' dbname='nlab' user='{}' password='{}'".format(db_ip, user, pw)) as conn:
        df = pd.read_sql(sql, conn, params = {'ref_date':reference_day, 'ws':tumbling_window_size, 'ows':output_window_size})

    return df.drop(columns = ['ref_day','last_purchased','customer_id','output_feature'
                             ], inplace = False), df.output_feature
```

2. Final function of comparing list of models by f1 score

(get_f1 function, including preprocessing such as SMOTE and standardization)

```
In [4]: def get_f1(model, total_holdout_sets, now, ws, ows):
    scores = []

    # for each holdout set, compute f1 score
    for i in range(total_holdout_sets):
        valid = get_dataset_value(now-2*ows, ws, ows)
        train = get_dataset_value(now-3*ows, ws, ows)

         # output feature changes to binary, 1: non- churn, 0: churn
        valid[1][valid[1]>0] = 1 # non-chrun
        train[1][train[1]>0] = 1 # non-chrun

        # Balancing unbalanced output feature in train data set using SMOTE
        smote = SMOTE(random_state=42)
        X_train, y_train = smote.fit_resample(train[0], train[1])

        X_train = pd.DataFrame(X_train,
                      columns=['total_values','total_quantity','avg_between',
                               'f1','f2','f3','f4','f5','f6','f7','f8','f9','f1
0','f11'])
        y_train = pd.DataFrame(y_train)

        # standardizing Temporal data in train set
        train_X = pd.DataFrame()

        for i in X_train.iloc[:,3:14].values:
            a = i - X_train.iloc[:,3:14].values.sum()
            b = a / np.std(X_train.iloc[:,3:14].values)

            new_row = pd.DataFrame( [[b]] )
            train_X = train_X.append(new_row, ignore_index = True)

        train_X.columns = ['f']
        train_X = pd.DataFrame(train_X.f.tolist(),
                            columns=['f1','f2','f3','f4','f5','f6','f7','f8',
                                     'f9','f10','f11'])

        # standardizing traditional data in train set
        # Step 1: log1p
        train_X2 = X_train.drop(columns=['f1','f2','f3','f4','f5','f6','f7','f
8','f9','f10','f11'])
        train_X2_log = np.log1p(train_X2)
        # Step 2: StandardScaler
        scaler = StandardScaler()
        train_X2_scaled = scaler.fit_transform(train_X2_log)

        # transform into a dataframe
        train_X2_scaled = pd.DataFrame(train_X2_scaled, index=train_X2_log.inde
x,
                             columns=train_X2_log.columns)
        final_train = pd.concat([train_X2_scaled, train_X], axis=1)
        final_train = round(final_train,2)

        # standardizing Temporal data in validation set
        valid_X = pd.DataFrame()

        for i in valid[0].iloc[:,3:14].values:
            a = i - valid[0].iloc[:,3:14].values.sum()
            b = a / np.std(valid[0].iloc[:,3:14].values)

            new_row = pd.DataFrame( [[b]] )
            valid_X = valid_X.append(new_row, ignore_index = True)
```

```
        valid_X.columns = ['f']
        valid_X = pd.DataFrame(valid_X.f.tolist(),
                                    columns=['f1','f2','f3','f4','f5','f6','f7','f8',
                                             'f9','f10','f11'])

        # standardizing traditional data in validation set
        # Step 1: log1p
        valid_X2 = valid[0].drop(columns=['f1','f2','f3','f4','f5','f6','f7','f
8','f9','f10','f11'])
        valid_X2_log = np.log1p(valid_X2)
        # Step 2: StandardScaler
        scaler = StandardScaler()
        valid_X2_scaled = scaler.fit_transform(valid_X2_log)

        # transform into a dataframe
        valid_X2_scaled = pd.DataFrame(valid_X2_scaled, index=valid_X2_log.inde
x,
                                     columns=valid_X2_log.columns)
        # Merge into final
        final_valid = pd.concat([valid_X2_scaled, valid_X], axis=1)
        final_valid = round(final_valid,2)

        # prediction using f1_score
        model.fit(final_train, y_train)
        preds = model.predict(final_valid)
        s = f1_score(valid[1], preds)
        s = round(s,3)
        scores.append(s)
        now = now - ows

    return round(np.mean(scores),3)
```

3. List of models

In [5]:

```
list_of_models = []

m1 = LogisticRegression( solver = 'liblinear', random_state=42)
m2 = KNeighborsClassifier()
m3 = LinearSVC(C=1, loss='hinge',random_state=42)
m4 = LinearSVC(random_state=42)
m5 = SVC(kernel='rbf', gamma=5, C=1,random_state=42)
m6 = SVC(random_state=42)
m7 = GaussianProcessClassifier(1.0 * RBF(1.0),random_state=42)
m8 = GaussianProcessClassifier(random_state=42)

m9 = DecisionTreeClassifier(max_depth=5,random_state=42)
m10 = DecisionTreeClassifier(random_state=42)
m11 = RandomForestClassifier(max_depth=5, n_estimators=10,
                             max_features=1,random_state=42)
m12 = RandomForestClassifier(random_state=42)
m13 = GaussianNB()

m14 = AdaBoostClassifier(n_estimators=100, random_state=42)
m15 = GradientBoostingClassifier(n_estimators=100, random_state=42)
m16 = xgb.XGBClassifier(random_state=42)

list_of_models += [m1, m2, m3, m4, m5, m6, m7, m8, m9, m10,
                   m11, m12, m13,m14,m15,m16]
```

4. Comparing each model's f1 score

In [6]:
```
ws = 33
ows = 33
now = 609
f1_comparison = pd.DataFrame()

for model in list_of_models:
    scores =  get_f1(model, total_holdout_sets=2,
                     now=now, ws=ws, ows=ows)
    new_row = pd.DataFrame( [[model, scores]] )
    f1_comparison = f1_comparison.append(new_row, ignore_index = True)
    print(model, 'completed')

f1_comparison = f1_comparison.rename(
   columns={0:'model', 1:'f1_score'})
```

```
LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
                   intercept_scaling=1, l1_ratio=None, max_iter=100,
                   multi_class='auto', n_jobs=None, penalty='l2',
                   random_state=42, solver='liblinear', tol=0.0001, verbose=0,
                   warm_start=False) completed
KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
                     metric_params=None, n_jobs=None, n_neighbors=5, p=2,
                     weights='uniform') completed
LinearSVC(C=1, class_weight=None, dual=True, fit_intercept=True,
          intercept_scaling=1, loss='hinge', max_iter=1000, multi_class='ovr',
          penalty='l2', random_state=42, tol=0.0001, verbose=0) completed
LinearSVC(C=1.0, class_weight=None, dual=True, fit_intercept=True,
          intercept_scaling=1, loss='squared_hinge', max_iter=1000,
          multi_class='ovr', penalty='l2', random_state=42, tol=0.0001,
          verbose=0) completed
SVC(C=1, break_ties=False, cache_size=200, class_weight=None, coef0=0.0,
    decision_function_shape='ovr', degree=3, gamma=5, kernel='rbf', max_iter=-
1,
    probability=False, random_state=42, shrinking=True, tol=0.001,
    verbose=False) completed
SVC(C=1.0, break_ties=False, cache_size=200, class_weight=None, coef0=0.0,
    decision_function_shape='ovr', degree=3, gamma='scale', kernel='rbf',
    max_iter=-1, probability=False, random_state=42, shrinking=True, tol=0.001,
    verbose=False) completed
GaussianProcessClassifier(copy_X_train=True, kernel=1**2 * RBF(length_scale=1),
                          max_iter_predict=100, multi_class='one_vs_rest',
                          n_jobs=None, n_restarts_optimizer=0,
                          optimizer='fmin_l_bfgs_b', random_state=42,
                          warm_start=False) completed
GaussianProcessClassifier(copy_X_train=True, kernel=None, max_iter_predict=100,
                          multi_class='one_vs_rest', n_jobs=None,
                          n_restarts_optimizer=0, optimizer='fmin_l_bfgs_b',
                          random_state=42, warm_start=False) completed
DecisionTreeClassifier(ccp_alpha=0.0, class_weight=None, criterion='gini',
                       max_depth=5, max_features=None, max_leaf_nodes=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, presort='deprecated',
                       random_state=42, splitter='best') completed
DecisionTreeClassifier(ccp_alpha=0.0, class_weight=None, criterion='gini',
                       max_depth=None, max_features=None, max_leaf_nodes=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, presort='deprecated',
                       random_state=42, splitter='best') completed
RandomForestClassifier(bootstrap=True, ccp_alpha=0.0, class_weight=None,
                       criterion='gini', max_depth=5, max_features=1,
                       max_leaf_nodes=None, max_samples=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, n_estimators=10,
                       n_jobs=None, oob_score=False, random_state=42, verbose=
0,
                       warm_start=False) completed
RandomForestClassifier(bootstrap=True, ccp_alpha=0.0, class_weight=None,
                       criterion='gini', max_depth=None, max_features='auto',
                       max_leaf_nodes=None, max_samples=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, n_estimators=100,
                       n_jobs=None, oob_score=False, random_state=42, verbose=
0,
```

```
                              warm_start=False) completed
GaussianNB(priors=None, var_smoothing=1e-09) completed
AdaBoostClassifier(algorithm='SAMME.R', base_estimator=None, learning_rate=1.0,
                   n_estimators=100, random_state=42) completed
GradientBoostingClassifier(ccp_alpha=0.0, criterion='friedman_mse', init=None,
                           learning_rate=0.1, loss='deviance', max_depth=3,
                           max_features=None, max_leaf_nodes=None,
                           min_impurity_decrease=0.0, min_impurity_split=None,
                           min_samples_leaf=1, min_samples_split=2,
                           min_weight_fraction_leaf=0.0, n_estimators=100,
                           n_iter_no_change=None, presort='deprecated',
                           random_state=42, subsample=1.0, tol=0.0001,
                           validation_fraction=0.1, verbose=0,
                           warm_start=False) completed
XGBClassifier(base_score=0.5, booster='gbtree', colsample_bylevel=1,
              colsample_bynode=1, colsample_bytree=1, gamma=0,
              learning_rate=0.1, max_delta_step=0, max_depth=3,
              min_child_weight=1, missing=None, n_estimators=100, n_jobs=1,
              nthread=None, objective='binary:logistic', random_state=42,
              reg_alpha=0, reg_lambda=1, scale_pos_weight=1, seed=None,
              silent=None, subsample=1, verbosity=1) completed
```

In [7]:
```
f1_comparison
```

Out[7]:

	model	f1_score
0	LogisticRegression(C=1.0, class_weight=None, d...	0.800
1	KNeighborsClassifier(algorithm='auto', leaf_si...	0.713
2	LinearSVC(C=1, class_weight=None, dual=True, f...	0.352
3	LinearSVC(C=1.0, class_weight=None, dual=True,...	0.352
4	SVC(C=1, break_ties=False, cache_size=200, cla...	0.713
5	SVC(C=1.0, break_ties=False, cache_size=200, c...	0.713
6	GaussianProcessClassifier(copy_X_train=True, k...	0.000
7	GaussianProcessClassifier(copy_X_train=True, k...	0.000
8	DecisionTreeClassifier(ccp_alpha=0.0, class_we...	0.691
9	DecisionTreeClassifier(ccp_alpha=0.0, class_we...	0.657
10	(DecisionTreeClassifier(ccp_alpha=0.0, class_w...	0.714
11	(DecisionTreeClassifier(ccp_alpha=0.0, class_w...	0.734
12	GaussianNB(priors=None, var_smoothing=1e-09)	0.713
13	(DecisionTreeClassifier(ccp_alpha=0.0, class_w...	0.480
14	([DecisionTreeRegressor(ccp_alpha=0.0, criteri...	0.752
15	XGBClassifier(base_score=0.5, booster='gbtree'...	0.793

In [233]:
```
# Basemodel predcition
valid = get_dataset_value(now-2*ows, ws, ows)
valid[1][valid[1]>0] = 1 # non-chrun
valid[0].loc[ (valid[0].f1 > 0), 'f1' ] = 1 # non-churn
b = f1_score(valid[1], valid[0].f1)
print('Baseline f1 score:{}'.format(round(b,3)))
```

```
Baseline f1 score:0.723
```

The top three models are logistic regression, XGBoost and Gradient Boosting that has higher f1 score compare to the basline f1 score. Now, we need to find optimal meta-parameter for these three models.

5. Details of preprocessing data (get_f1 function)

1) Data preparation for SMOTE

In [192]:
```
now = 609
ows = 33
ws = 33

valid = get_dataset_value(now-2*ows, ws, ows)
train1 = get_dataset_value(now-3*ows, ws, ows)

valid[1][valid[1]>0]=1
train1[1][train1[1]>0]=1
```

2) Finding the best method for balancing unbalanced train dataset

In [9]:
```
# 1. SMOTE
print('Before(train[1]):      ', Counter(train1[1]))
k_values = [1, 2, 3, 4, 5, 6, 7]
for k in k_values:
    m2 = KNeighborsClassifier()
    oversample = SMOTE(k_neighbors=k, random_state=42)
    train_X, train_y = oversample.fit_resample(train1[0], train1[1])
    m2.fit(train_X, train_y)
    preds = m2.predict(valid[0])
    s = f1_score(valid[1], preds)
    print('> k=%d,f1_score: %s' % (k, round(s,3)))
print('After SMOTE(train[1]):', Counter(train_y))
```

```
Before(train[1]):      Counter({1.0: 460, 0.0: 388})
> k=1,f1_score: 0.811
> k=2,f1_score: 0.791
> k=3,f1_score: 0.796
> k=4,f1_score: 0.811
> k=5,f1_score: 0.807
> k=6,f1_score: 0.8
> k=7,f1_score: 0.804
After SMOTE(train[1]): Counter({1.0: 460, 0.0: 460})
```

In [10]:
```python
# 2. Borderline SMOTE
print('Before(train1[1]):                ', Counter(train1[1]))
k_values = [1, 2, 3, 4, 5, 6, 7]
for k in k_values:
    m2 = KNeighborsClassifier()
    oversample = BorderlineSMOTE(k_neighbors=k, random_state=42)
    train2_X, train2_y = oversample.fit_resample(train1[0], train1[1])
    m2.fit(train2_X, train2_y)
    preds = m2.predict(valid[0])
    s = f1_score(valid[1], preds)
    print('> k=%d,f1_score: %s' % (k, round(s,3)))
print('After Borderline SMOTE(train[1]):', Counter(train2_y))
```

```
Before(train1[1]):                Counter({1.0: 460, 0.0: 388})
> k=1,f1_score: 0.786
> k=2,f1_score: 0.788
> k=3,f1_score: 0.786
> k=4,f1_score: 0.787
> k=5,f1_score: 0.789
> k=6,f1_score: 0.791
> k=7,f1_score: 0.784
After Borderline SMOTE(train[1]): Counter({1.0: 460, 0.0: 460})
```

In [11]:
```python
# 3. SVM SMOTE
print('Before(train1[1]):        ', Counter(train1[1]))
k_values = [1, 2, 3, 4, 5, 6, 7]
for k in k_values:
    m2 = KNeighborsClassifier()
    oversample = SVMSMOTE(k_neighbors=k, random_state=42)
    train3_X, train3_y = oversample.fit_resample(train1[0], train1[1])
    m2.fit(train3_X, train3_y)
    preds = m2.predict(valid[0])
    s = f1_score(valid[1], preds)
    print('> k=%d,f1_score: %s' % (k, round(s,3)))
print('After SVM SMOTE(train3[1]):', Counter(train3_y))
```

```
Before(train1[1]):        Counter({1.0: 460, 0.0: 388})
> k=1,f1_score: 0.793
> k=2,f1_score: 0.8
> k=3,f1_score: 0.799
> k=4,f1_score: 0.796
> k=5,f1_score: 0.793
> k=6,f1_score: 0.792
> k=7,f1_score: 0.786
After SVM SMOTE(train3[1]): Counter({1.0: 460, 0.0: 460})
```

In [12]:
```python
# 4. ADASYN
print('Before(train1[1]):        ', Counter(train1[1]))
oversample = ADASYN(random_state=42)
train4_X, train4_y = oversample.fit_resample(train1[0], train1[1])
m2 = KNeighborsClassifier()
m2.fit(train4_X, train4_y)
preds = m2.predict(valid[0])
s = f1_score(valid[1], preds)
print('f1 score:', round(s,3))
print('After ADASYN(train3[1]):', Counter(train4_y))
```

```
Before(train1[1]):        Counter({1.0: 460, 0.0: 388})
f1 score: 0.789
After ADASYN(train3[1]): Counter({1.0: 460, 0.0: 450})
```

In [13]:
```
# 5. Kmeans SMOTE
from imblearn.over_sampling import KMeansSMOTE
print('Before(train1[1]):          ', Counter(train1[1]))
k_values = [1, 2, 3, 4, 5, 6, 7]
for k in k_values:
    m2 = KNeighborsClassifier()
    oversample = KMeansSMOTE(k_neighbors=k, random_state=42)
    train5_X, train5_y = oversample.fit_resample(train1[0], train1[1])
    m2.fit(train5_X, train5_y)
    preds = m2.predict(valid[0])
    s = f1_score(valid[1], preds)
    print('> k=%d,f1_score: %s' % (k, round(s,3)))
print('After Kmeans SMOTE(train[1]):', Counter(train5_y))
```

```
Before(train1[1]):           Counter({1.0: 460, 0.0: 388})
> k=1,f1_score: 0.819
> k=2,f1_score: 0.817
> k=3,f1_score: 0.818
> k=4,f1_score: 0.818
> k=5,f1_score: 0.821
> k=6,f1_score: 0.818
> k=7,f1_score: 0.82
After Kmeans SMOTE(train[1]): Counter({1.0: 460, 0.0: 460})
```

In [14]:
```
# 6. RandomOverSampler
from imblearn.over_sampling import RandomOverSampler
print('Before(train1[1]):                ', Counter(train1[1]))
m2 = KNeighborsClassifier()
oversample = RandomOverSampler(random_state=42)
train6_X, train6_y = oversample.fit_resample(train1[0], train1[1])
m2.fit(train6_X, train6_y)
preds = m2.predict(valid[0])
s = f1_score(valid[1], preds)
print('f1 score:', round(s,3))
print('After RandomOverSampler(train[1]):', Counter(train6_y))
```

```
Before(train1[1]):                 Counter({1.0: 460, 0.0: 388})
f1 score: 0.814
After RandomOverSampler(train[1]): Counter({1.0: 460, 0.0: 460})
```

In [15]:
```
# 7. SMOTENC
from imblearn.over_sampling import SMOTENC
print('Before(train1[1]):                ', Counter(train1[1]))
k_values = [1, 2, 3, 4, 5, 6, 7]
for k in k_values:
    m2 = KNeighborsClassifier()
    oversample = SMOTENC(k_neighbors=k, random_state=42, categorical_features=[
0,1])
    train7_X, train7_y = oversample.fit_resample(train1[0], train1[1])
    m2.fit(train7_X, train7_y)
    preds = m2.predict(valid[0])
    s = f1_score(valid[1], preds)
    print('> k=%d,f1_score: %s' % (k, round(s,3)))
print('After RandomOverSampler(train[7]):', Counter(train7_y))
```

```
Before(train1[1]):                 Counter({1.0: 460, 0.0: 388})
> k=1,f1_score: 0.801
> k=2,f1_score: 0.805
> k=3,f1_score: 0.815
> k=4,f1_score: 0.818
> k=5,f1_score: 0.811
> k=6,f1_score: 0.806
> k=7,f1_score: 0.812
After RandomOverSampler(train[7]): Counter({1.0: 460, 0.0: 460})
```

In [16]:
```
# 8. SMOTEENN
from imblearn.combine import SMOTEENN
print('Before(train1[1]):                ', Counter(train1[1]))
m2 = KNeighborsClassifier()
oversample = SMOTEENN(random_state=42)
train8_X, train8_y = oversample.fit_resample(train1[0], train1[1])
m2.fit(train8_X, train8_y)
preds = m2.predict(valid[0])
s = f1_score(valid[1], preds)
print('f1 score:', round(s,3))
print('After RandomOverSampler(train[8]):', Counter(train8_y))
```

```
Before(train1[1]):                 Counter({1.0: 460, 0.0: 388})
f1 score: 0.805
After RandomOverSampler(train[8]): Counter({1.0: 275, 0.0: 271})
```

In [17]:
```
# 9. SMOTETomek
from imblearn.combine import SMOTETomek
print('Before(train1[1]):                ', Counter(train1[1]))
m2 = KNeighborsClassifier()
oversample = SMOTETomek(random_state=42)
train9_X, train9_y = oversample.fit_resample(train1[0], train1[1])
valid9_X, valid9_y = oversample.fit_resample(valid[0], valid[1])
m2.fit(train9_X, train9_y)
preds = m2.predict(valid[0])
s = f1_score(valid[1], preds)
print('f1 score:', round(s,3))
print('After RandomOverSampler(train[9]):', Counter(train9_y))
```

```
Before(train1[1]):                 Counter({1.0: 460, 0.0: 388})
f1 score: 0.811
After RandomOverSampler(train[9]): Counter({1.0: 436, 0.0: 436})
```

Decide to use SMOTE, results are similar

3) Data preparation for standardization

In [193]:
```
now = 609
ows = 33
ws = 33

valid = get_dataset_value(now-2*ows, ws, ows)
train = get_dataset_value(now-3*ows, ws, ows)
```

4) Standardization of temporal data in validation set

In [20]:
```
# valid temporal data standarization
valid_X = pd.DataFrame()

for i in valid[0].iloc[:,3:14].values:
    a = i - valid[0].iloc[:,3:14].values.sum()
    b = a / np.std(valid[0].iloc[:,3:14].values)
    #print(b)

    new_row = pd.DataFrame( [[b]] )
    valid_X = valid_X.append(new_row, ignore_index = True)

valid_X.columns = ['f']
valid_X = pd.DataFrame(valid_X.f.tolist(),
                        columns=['f1','f2','f3','f4','f5','f6','f7','f8',
                                 'f9','f10','f11'])
valid_X.head()
```

Out[20]:

	f1	f2	f3	f4	f5	f6	f
0	-4779.877723	-4779.790736	-4779.790144	-4779.412262	-4779.959227	-4779.847344	-4779.77532
1	-4779.885133	-4779.959227	-4779.959227	-4779.959227	-4779.959227	-4779.885133	-4779.95922
2	-4779.815928	-4779.959227	-4779.959227	-4779.959227	-4779.959227	-4779.959227	-4779.95922
3	-4779.880391	-4779.959227	-4779.959227	-4779.606093	-4779.959227	-4779.847641	-4779.86379
4	-4779.437009	-4779.959227	-4779.959227	-4779.959227	-4779.959227	-4779.959227	-4779.79073

5) Standardization of temporal data in train set

In [21]:
```
# train temporal data standarization
train_X = pd.DataFrame()

for i in train[0].iloc[:,3:14].values:
    a = i - train[0].iloc[:,3:14].values.sum()
    b = a / np.std(train[0].iloc[:,3:14].values)
    #print(b)

    new_row = pd.DataFrame( [[b]] )
    train_X = train_X.append(new_row, ignore_index = True)

train_X.columns = ['f']
train_X = pd.DataFrame(train_X.f.tolist(),
                        columns=['f1','f2','f3','f4','f5','f6','f7','f8',
                                 'f9','f10','f11'])
train_X.head()
```

Out[21]:

	f1	f2	f3	f4	f5	f6	f
0	-4787.070961	-4787.070368	-4786.692261	-4787.239553	-4787.127603	-4787.055541	-4787.23955
1	-4786.290281	-4786.490752	-4786.464952	-4786.228597	-4786.176107	-4786.411572	-4786.36397
2	-4787.163487	-4786.779596	-4787.239553	-4787.239553	-4787.239553	-4786.860111	-4787.11099
3	-4786.980957	-4787.139021	-4787.177425	-4787.239553	-4787.239553	-4787.239553	-4787.23955
4	-4787.036561	-4787.148214	-4787.117076	-4787.175497	-4787.239553	-4787.239553	-4787.23955

6) Standardization of traditional data in validiation set

In [22]:
```
# Valid + log + standardize
valid_X2 = valid[0].drop(columns=['f1','f2','f3','f4','f5','f6','f7','f8','f9',
'f10','f11'])
valid_X2_log = np.log1p(valid_X2)

scaler = StandardScaler()
valid_X2_scaled = scaler.fit_transform(valid_X2_log)

# transform into a dataframe
valid_X2_scaled = pd.DataFrame(valid_X2_scaled, index=valid_X2_log.index,
                         columns=valid_X2_log.columns)
round(valid_X2_scaled.describe(),3)
```

Out[22]:

	total_values	total_quantity	avg_between
count	836.000	836.000	836.000
mean	0.000	-0.000	0.000
std	1.001	1.001	1.001
min	-2.622	-2.336	-2.164
25%	-0.671	-0.659	-0.544
50%	0.076	0.092	0.044
75%	0.738	0.788	0.711
max	2.410	2.279	2.170

In [23]:

```python
# Check the Pearson correlations in traditonal date in valid set
valid_X2 = valid[0].drop(columns=['f1','f2','f3','f4','f5',
                                  'f6','f7','f8','f9','f10','f11'])
print(valid_X2.describe())
corr = valid_X2.corr()
corr
```

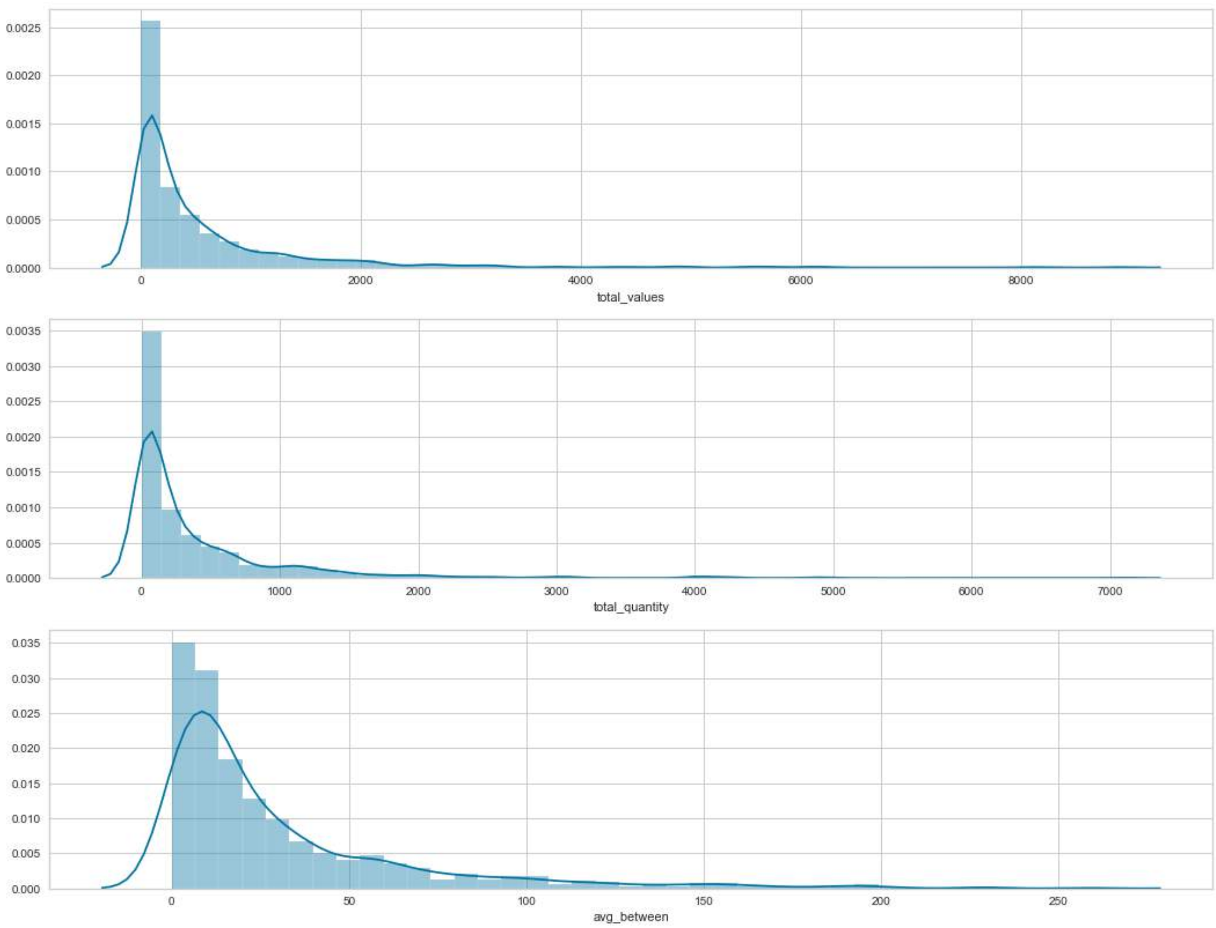
```
       total_values  total_quantity  avg_between
count    836.000000      836.000000   836.000000
mean     558.000646      405.217703    30.572967
std      910.869152      675.178327    38.121246
min        2.290000        1.000000     0.000000
25%       69.505000       38.000000     7.000000
50%      227.025000      146.500000    16.000000
75%      643.715000      504.750000    39.000000
max     8914.410000     7076.000000   259.000000
```

Out[23]:

	total_values	total_quantity	avg_between
total_values	1.000000	0.944343	-0.324389
total_quantity	0.944343	1.000000	-0.324683
avg_between	-0.324389	-0.324683	1.000000

In [24]:

```python
# Check the distribution of each features
plt.figure(figsize=(18,14))
plt.subplot(3,1,1); sns.distplot(valid_X2['total_values'])
plt.subplot(3,1,2); sns.distplot(valid_X2['total_quantity'])
plt.subplot(3,1,3); sns.distplot(valid_X2['avg_between'])
plt.show()
```

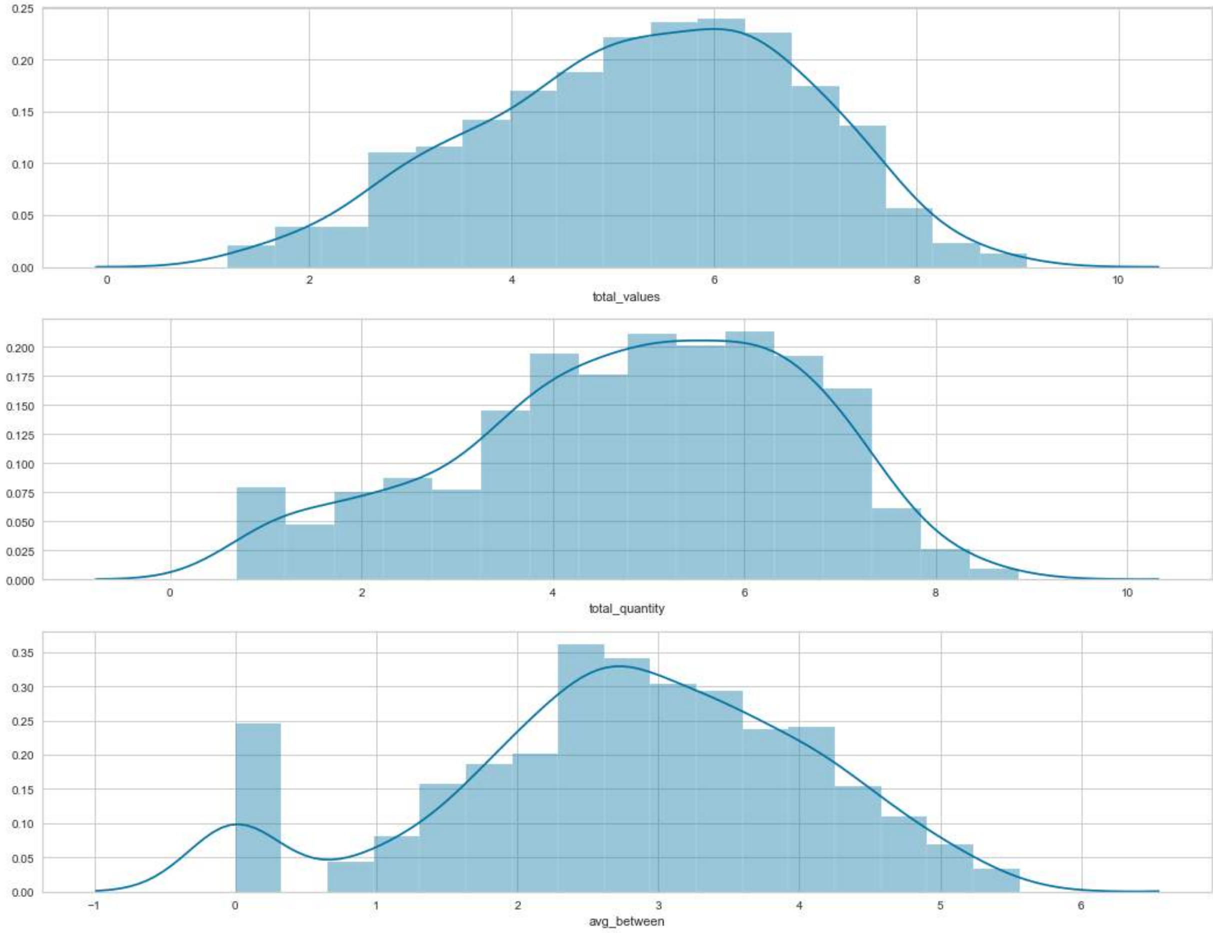


In [25]:
```
# Applying the Log1p transformation to make the data more 'normal'
valid_X2_log = np.log1p(valid_X2)
valid_X2_log.head()
```

Out[25]:

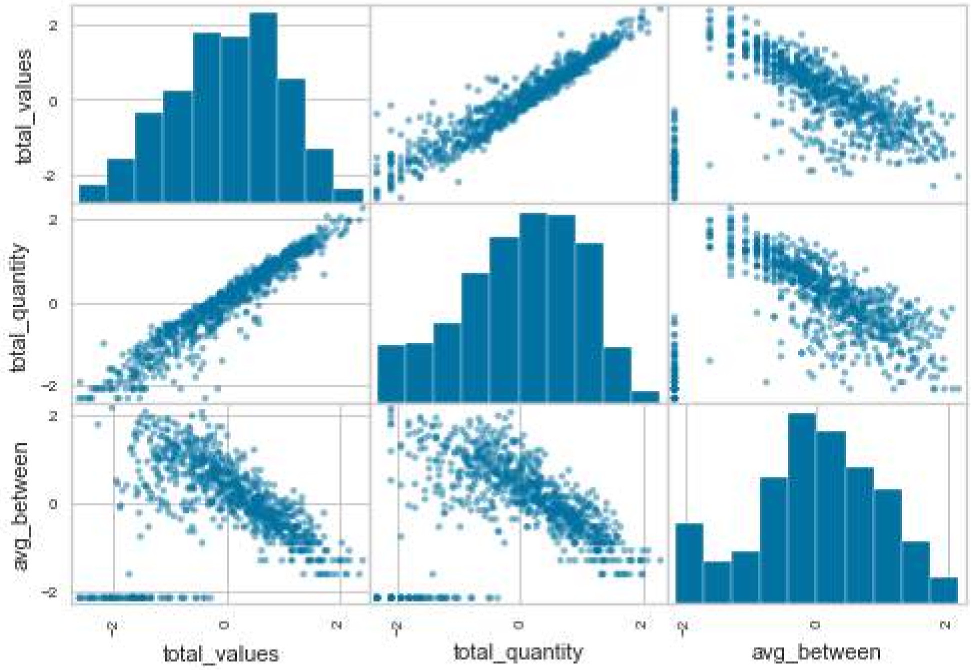
	total_values	total_quantity	avg_between
0	4.945421	4.770685	3.663562
1	3.386084	2.639057	4.430817
2	2.367436	1.386294	0.000000
3	5.067079	4.304065	3.610918
4	4.085976	4.248495	4.060443

In [26]:
```
# Check the distribution of each features
plt.figure(figsize=(18,14))
plt.subplot(3,1,1); sns.distplot(valid_X2_log['total_values'])
plt.subplot(3,1,2); sns.distplot(valid_X2_log['total_quantity'])
plt.subplot(3,1,3); sns.distplot(valid_X2_log['avg_between'])
plt.show()
```



In [27]:
```
# Rescaling to remove the units
scaler = StandardScaler()
valid_X2_scaled = scaler.fit_transform(valid_X2_log)

# transform into a dataframe
valid_X2_scaled = pd.DataFrame(valid_X2_scaled, index=valid_X2_log.index,
                               columns=valid_X2_log.columns)
```

In [28]:
```python
# Check the final distribution of each features
scatter = pd.plotting.scatter_matrix(valid_X2_scaled)
```



In [29]:
```python
# Result
round(valid_X2_scaled.describe(),3)
```

Out[29]:

	total_values	total_quantity	avg_between
count	836.000	836.000	836.000
mean	0.000	-0.000	0.000
std	1.001	1.001	1.001
min	-2.622	-2.336	-2.164
25%	-0.671	-0.659	-0.544
50%	0.076	0.092	0.044
75%	0.738	0.788	0.711
max	2.410	2.279	2.170

7) Standardization of traditional data in train set

In [30]:

```python
# Train + log + standardize
train_X2 = train[0].drop(columns=['f1','f2','f3','f4','f5','f6','f7','f8','f9',
'f10','f11'])
train_X2_log = np.log1p(train_X2)

scaler = StandardScaler()
train_X2_scaled = scaler.fit_transform(train_X2_log)

# transform into a dataframe
train_X2_scaled = pd.DataFrame(train_X2_scaled, index=train_X2_log.index,
                         columns=train_X2_log.columns)
round(train_X2_scaled.describe(),3)
```

Out[30]:

	total_values	total_quantity	avg_between
count	848.000	848.000	848.000
mean	-0.000	-0.000	-0.000
std	1.001	1.001	1.001
min	-2.816	-2.320	-2.227
25%	-0.721	-0.677	-0.572
50%	0.071	0.089	0.074
75%	0.744	0.799	0.729
max	2.423	2.291	1.975

In [31]:

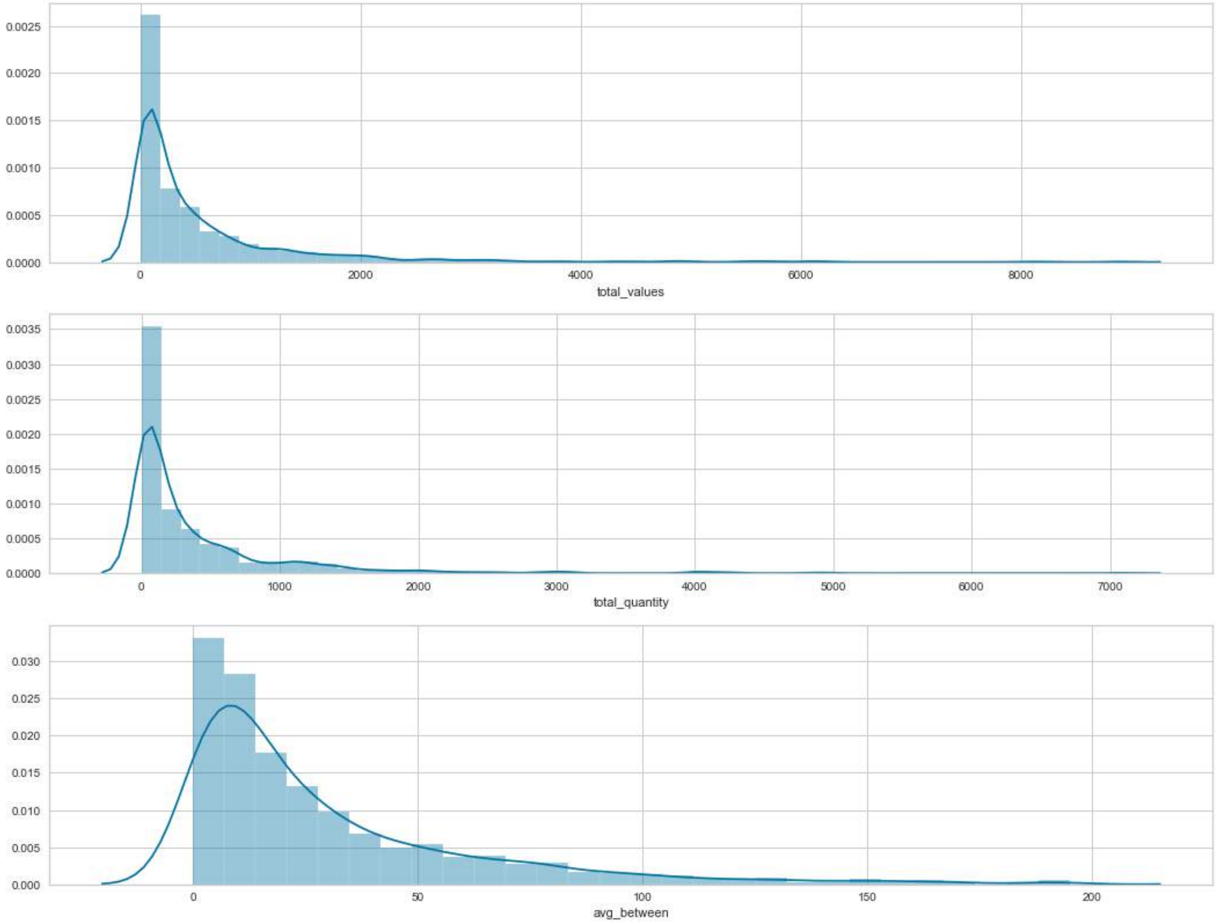
```python
# Check the Pearson correlations in traditonal date in train set
train_X2 = train[0].drop(columns=['f1','f2','f3','f4','f5',
                                  'f6','f7','f8','f9','f10','f11'])
print(train_X2.describe())
corr = train_X2.corr()
corr
```

```
       total_values  total_quantity  avg_between
count    848.000000      848.000000   848.000000
mean     553.155554      402.873821    29.707547
std      912.498902      678.827204    33.870290
min        1.350000        1.000000     0.000000
25%       62.435000       35.750000     7.000000
50%      219.675000      142.000000    17.000000
75%      634.940000      502.250000    40.000000
max     8914.410000     7076.000000   195.000000
```

Out[31]:

	total_values	total_quantity	avg_between
total_values	1.000000	0.944726	-0.347511
total_quantity	0.944726	1.000000	-0.350100
avg_between	-0.347511	-0.350100	1.000000

In [32]:
```python
# Check the distribution of each features
plt.figure(figsize=(18,14))
plt.subplot(3,1,1); sns.distplot(train_X2['total_values'])
plt.subplot(3,1,2); sns.distplot(train_X2['total_quantity'])
plt.subplot(3,1,3); sns.distplot(train_X2['avg_between'])
plt.show()
```

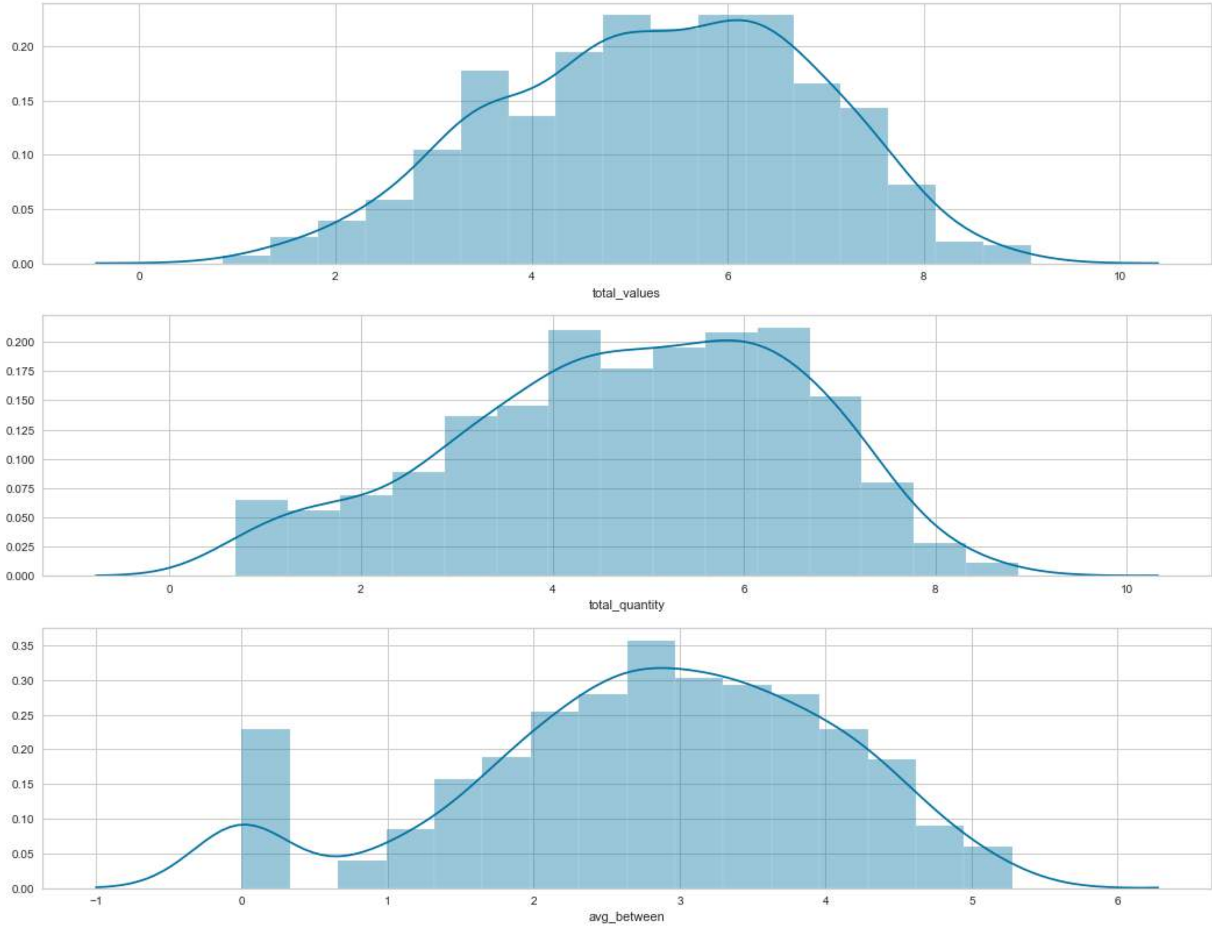


In [33]:
```python
# Applying the Log1p transformation to make the data more 'normal'
train_X2_log = np.log1p(train_X2)
train_X2_log.head()
```

Out[33]:

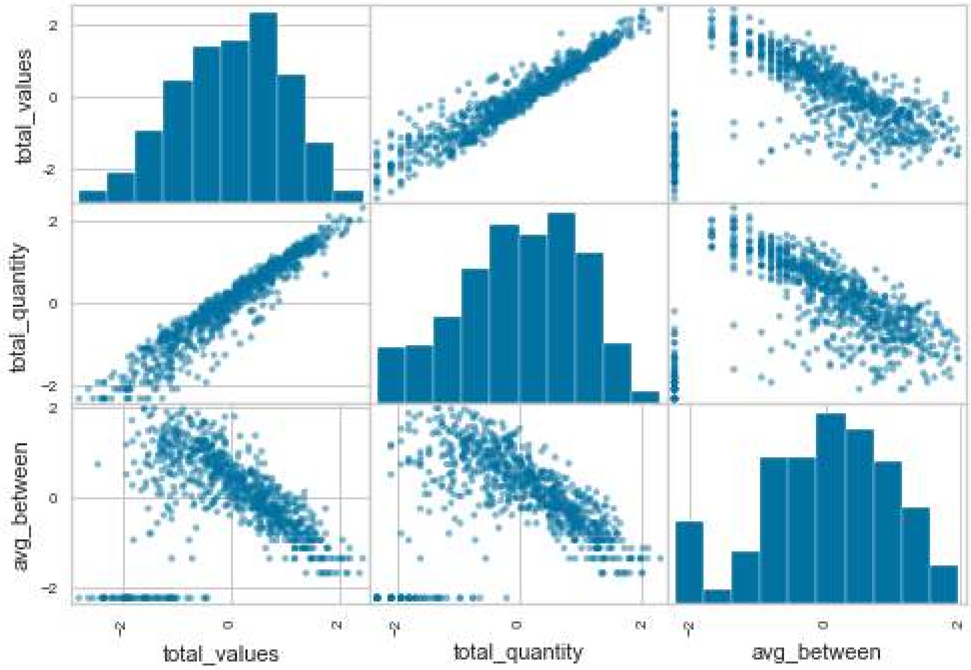
	total_values	total_quantity	avg_between
0	4.945421	4.770685	3.663562
1	6.847411	6.853299	1.609438
2	4.988662	4.290459	3.526361
3	4.396423	3.135494	2.833213
4	4.523852	4.330733	2.890372

In [34]:
```
# Check the distribution of each features
plt.figure(figsize=(18,14))
plt.subplot(3,1,1); sns.distplot(train_X2_log['total_values'])
plt.subplot(3,1,2); sns.distplot(train_X2_log['total_quantity'])
plt.subplot(3,1,3); sns.distplot(train_X2_log['avg_between'])
plt.show()
```



In [35]:
```
# Rescaling to remove the units
scaler = StandardScaler()
train_X2_scaled = scaler.fit_transform(train_X2_log)

# transform into a dataframe
train_X2_scaled = pd.DataFrame(train_X2_scaled, index=train_X2_log.index,
                               columns=train_X2_log.columns)
```

In [36]:
```
# Check the final distribution of each features
scatter = pd.plotting.scatter_matrix(train_X2_scaled)
```



In [37]:
```
# Result
round(train_X2_scaled.describe(),3)
```

Out[37]:

	total_values	total_quantity	avg_between
count	848.000	848.000	848.000
mean	-0.000	-0.000	-0.000
std	1.001	1.001	1.001
min	-2.816	-2.320	-2.227
25%	-0.721	-0.677	-0.572
50%	0.071	0.089	0.074
75%	0.744	0.799	0.729
max	2.423	2.291	1.975

8) Final code for validation set standardization

In [38]:
```python
# valid temporal data standarization
valid_X = pd.DataFrame()

for i in valid[0].iloc[:,3:14].values:
    a = i - valid[0].iloc[:,3:14].values.sum()
    b = a / np.std(valid[0].iloc[:,3:14].values)
    #print(b)

    new_row = pd.DataFrame( [[b]] )
    valid_X = valid_X.append(new_row, ignore_index = True)

valid_X.columns = ['f']
valid_X = pd.DataFrame(valid_X.f.tolist(),
                        columns=['f1','f2','f3','f4','f5','f6','f7','f8',
                                 'f9','f10','f11'])

# Valid + log + standardize
valid_X2 = valid[0].drop(columns=['f1','f2','f3','f4','f5','f6','f7','f8','f9',
'f10','f11'])
valid_X2_log = np.log1p(valid_X2)

scaler = StandardScaler()
valid_X2_scaled = scaler.fit_transform(valid_X2_log)

# transform into a dataframe
valid_X2_scaled = pd.DataFrame(valid_X2_scaled, index=valid_X2_log.index,
                         columns=valid_X2_log.columns)
# Merge into final
final_valid = pd.concat([valid_X2_scaled, valid_X], axis=1)
final_valid = round(final_valid,2)
final_valid.head()
```

Out[38]:

	total_values	total_quantity	avg_between	f1	f2	f3	f4	f5	
0	-0.23	-0.03	0.69	-4779.88	-4779.79	-4779.79	-4779.41	-4779.96	-4779
1	-1.22	-1.24	1.29	-4779.89	-4779.96	-4779.96	-4779.96	-4779.96	-4779
2	-1.87	-1.94	-2.16	-4779.82	-4779.96	-4779.96	-4779.96	-4779.96	-4779
3	-0.15	-0.30	0.65	-4779.88	-4779.96	-4779.96	-4779.61	-4779.96	-4779
4	-0.78	-0.33	1.00	-4779.44	-4779.96	-4779.96	-4779.96	-4779.96	-4779

9) Final code for train set standardization

In [39]:
```
# train temporal data standarization
train_X = pd.DataFrame()

for i in train[0].iloc[:,3:14].values:
    a = i - train[0].iloc[:,3:14].values.sum()
    b = a / np.std(train[0].iloc[:,3:14].values)
    #print(b)

    new_row = pd.DataFrame( [[b]] )
    train_X = train_X.append(new_row, ignore_index = True)

train_X.columns = ['f']
train_X = pd.DataFrame(train_X.f.tolist(),
                        columns=['f1','f2','f3','f4','f5','f6','f7','f8',
                                 'f9','f10','f11'])

# Train + log + standardize
train_X2 = train[0].drop(columns=['f1','f2','f3','f4','f5','f6','f7','f8','f9',
'f10','f11'])
train_X2_log = np.log1p(train_X2)

scaler = StandardScaler()
train_X2_scaled = scaler.fit_transform(train_X2_log)

# transform into a dataframe
train_X2_scaled = pd.DataFrame(train_X2_scaled, index=train_X2_log.index,
                         columns=train_X2_log.columns)
# Merge into final
final_train = pd.concat([train_X2_scaled, train_X], axis=1)
final_train = round(final_train,2)
final_train.head()
```

Out[39]:

	total_values	total_quantity	avg_between	f1	f2	f3	f4	f5	
0	-0.22	-0.02	0.69	-4787.07	-4787.07	-4786.69	-4787.24	-4787.13	-4787
1	0.99	1.16	-0.95	-4786.29	-4786.49	-4786.46	-4786.23	-4786.18	-4786
2	-0.19	-0.29	0.58	-4787.16	-4786.78	-4787.24	-4787.24	-4787.24	-4786
3	-0.56	-0.94	0.03	-4786.98	-4787.14	-4787.18	-4787.24	-4787.24	-4787
4	-0.48	-0.27	0.07	-4787.04	-4787.15	-4787.12	-4787.18	-4787.24	-4787

C. Optimizing three model's meta-parameters

1. Data preparation for RandomizedSearchCV

In [ ]:
```python
ws = 33
ows = 33
now = 609
    # for each holdout set, compute f1 score

valid = get_dataset_value(now-2*ows, ws, ows)
train = get_dataset_value(now-3*ows, ws, ows)

         # output feature changes to binary, 1: non- churn, 0: churn
valid[1][valid[1]>0] = 1 # non-chrun
train[1][train[1]>0] = 1 # non-chrun

        # Balancing unbalanced output feature in train data set using SMOTE
smote = SMOTE(random_state=42)
X_train, y_train = smote.fit_resample(train[0], train[1])

X_train = pd.DataFrame(X_train,
                      columns=['total_values','total_quantity','avg_between',
                              'f1','f2','f3','f4','f5','f6','f7','f8','f9','f1
0','f11'])
y_train = pd.DataFrame(y_train)

        # standardizing Temporal data in train set
train_X = pd.DataFrame()

for i in X_train.iloc[:,3:14].values:
        a = i - X_train.iloc[:,3:14].values.sum()
        b = a / np.std(X_train.iloc[:,3:14].values)

        new_row = pd.DataFrame( [[b]] )
        train_X = train_X.append(new_row, ignore_index = True)

train_X.columns = ['f']
train_X = pd.DataFrame(train_X.f.tolist(),
                            columns=['f1','f2','f3','f4','f5','f6','f7','f8',
                                         'f9','f10','f11'])

        # standardizing traditional data in train set
        # Step 1: log1p
train_X2 = X_train.drop(columns=['f1','f2','f3','f4','f5','f6','f7','f8','f9',
'f10','f11'])
train_X2_log = np.log1p(train_X2)
        # Step 2: StandardScaler
scaler = StandardScaler()
train_X2_scaled = scaler.fit_transform(train_X2_log)

        # transform into a dataframe
train_X2_scaled = pd.DataFrame(train_X2_scaled, index=train_X2_log.index,
                             columns=train_X2_log.columns)
final_train = pd.concat([train_X2_scaled, train_X], axis=1)
final_train = round(final_train,2)

        # # standardizing Temporal data in validation set
valid_X = pd.DataFrame()

for i in valid[0].iloc[:,3:14].values:
        a = i - valid[0].iloc[:,3:14].values.sum()
        b = a / np.std(valid[0].iloc[:,3:14].values)

        new_row = pd.DataFrame( [[b]] )
        valid_X = valid_X.append(new_row, ignore_index = True)
```

```
valid_X.columns = ['f']
valid_X = pd.DataFrame(valid_X.f.tolist(),
                            columns=['f1','f2','f3','f4','f5','f6','f7','f8',
                                     'f9','f10','f11'])

        # standardizing traditional data in validation set
        # Step 1: log1p
valid_X2 = valid[0].drop(columns=['f1','f2','f3','f4','f5','f6','f7','f8','f9',
'f10','f11'])
valid_X2_log = np.log1p(valid_X2)
        # Step 2: StandardScaler
scaler = StandardScaler()
valid_X2_scaled = scaler.fit_transform(valid_X2_log)

        # transform into a dataframe
valid_X2_scaled = pd.DataFrame(valid_X2_scaled, index=valid_X2_log.index,
                             columns=valid_X2_log.columns)
        # Merge into final
final_valid = pd.concat([valid_X2_scaled, valid_X], axis=1)
final_valid = round(final_valid,2)
```

2. Logistic Regression RamdomizedSearchCV

1) Logistic RandomizedSearchCV

In [41]:
```
np.random.seed(42)

c_space = np.logspace(-10, 10, 20)
penalty = ['l1', 'l2']
param_grid = {'C': c_space,
              'penalty':penalty}

lf = LogisticRegression(solver = 'liblinear', random_state=42)
lf_cv = RandomizedSearchCV(lf, param_grid, cv = 10, n_jobs=-1, random_state=42)
lf_cv.fit(final_train, y_train)
print('Best Params of Logistic:', lf_cv.best_params_)
print('=========================================================')
print('Best training accuracy: ', round(lf_cv.best_score_,3))
y_pred = lf_cv.predict(final_valid)
f1 = f1_score(valid[1], y_pred)
accuray = accuracy_score(valid[1], y_pred)
print('Best validation accuracy: ', round(accuray,3))
print('=========================================================')
print('Valid set f1 score for best params:', round(f1,3))
print('=========================================================')
print('Confusion Matrix')
confusion = pd.DataFrame(confusion_matrix(valid[1], y_pred, labels=[1,0]),
                        index=['y_true Yes','y_ture No'],
                        columns=['y_predict Yes','y_predict No'])
print(confusion)
```

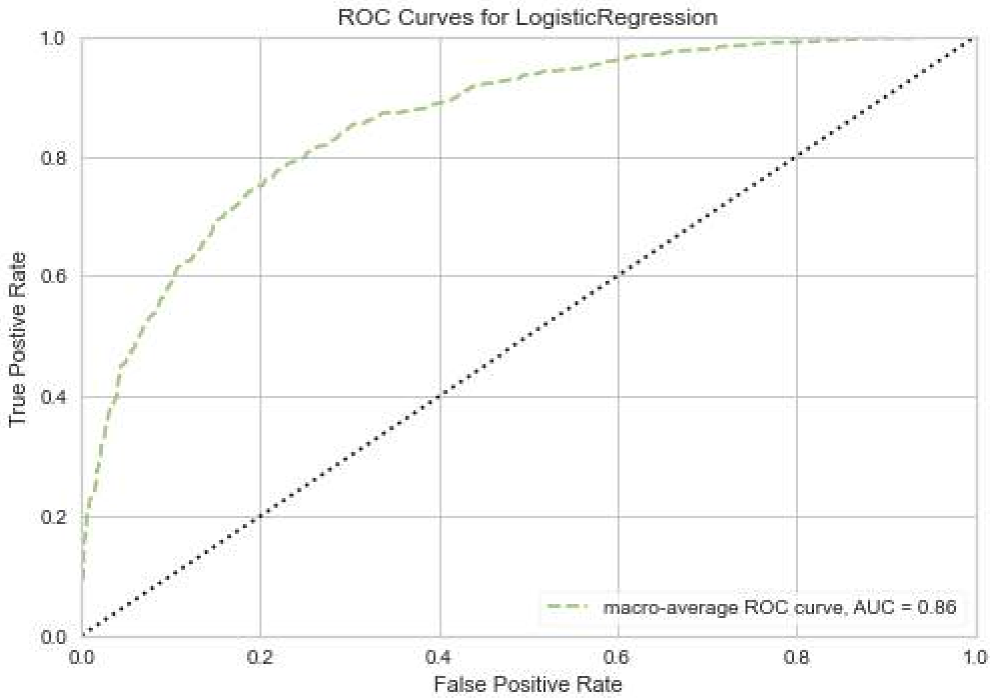
```
Best Params of Logistic: {'penalty': 'l2', 'C': 0.0023357214690901214}
=========================================================
Best training accuracy:  0.82
Best validation accuracy:  0.77
=========================================================
Valid set f1 score for best params: 0.786
=========================================================
Confusion Matrix
            y_predict Yes  y_predict No
y_true Yes            352           121
y_ture No              71           292
```

2) Optimized logistic regression model

In [42]:
```
# Finall Logistic model
lf = LogisticRegression(solver = 'liblinear', random_state=42,
                       penalty = 'l2', C= 0.0023357214690901214)
```

3) ROC curve and AUC score

In [45]:
```
visualizer = ROCAUC(lf, classes=[0, 1], micro=False, macro=True, per_class=False)
visualizer.fit(final_train, y_train)
visualizer.score(final_valid, valid[1])
visualizer.show()
print('roc_auc_score:', round(roc_auc_score(valid[1], y_pred),3))
```




```
roc_auc_score: 0.774
```

3. GradientBoostingClassifier RamdomSearchCV

1) GBM RandomizedSearchCV 1

In [46]:
```
# RandomizedSearchCV 1

np.random.seed(42)

learning_rate = np.random.uniform(0, 1, 10)
n_estimators = [int(x) for x in np.linspace(start = 10, stop = 1000, num = 30)]
min_samples_split = [int(x) for x in np.linspace(2,100,10)]
min_samples_leaf = [int(x) for x in np.linspace(2,100,10)]
max_depth = [int(x) for x in np.linspace(1,10, num=10)]
subsample = np.random.uniform(0, 1, 20)

param_grid = {'learning_rate': learning_rate,
              'n_estimators':n_estimators,
              'min_samples_split':min_samples_split,
              'min_samples_leaf':min_samples_leaf,
              'max_depth':max_depth,
              'subsample':subsample}
```

In [47]:
```
bgc = GradientBoostingClassifier(random_state=42)
bgc_cv = RandomizedSearchCV(bgc, param_grid, cv = 5, n_jobs=-1, random_state=42
)
bgc_cv.fit(final_train, y_train)
print('Best Params of GBM:', bgc_cv.best_params_)
print('==========================================================')
print('Best training accuracy: ', round(bgc_cv.best_score_,3))
y_pred = bgc_cv.predict(final_valid)
f1 = f1_score(valid[1], y_pred)
accuray = accuracy_score(valid[1], y_pred)
print('Best validation accuracy: ', round(accuray,3))
print('==========================================================')
print('Valid set f1 score for best params:', round(f1,3))
print('==========================================================')
print('Confusion Matrix')
confusion = pd.DataFrame(confusion_matrix(valid[1], y_pred, labels=[1,0]),
                        index=['y_true Yes','y_ture No'],
                        columns=['y_predict Yes','y_predict No'])
print(confusion)
```

```
Best Params of GBM: {'subsample': 0.5247564316322378, 'n_estimators': 487, 'min
_samples_split': 12, 'min_samples_leaf': 45, 'max_depth': 8, 'learning_rate':
0.05808361216819946}
==========================================================
Best training accuracy:  0.813
Best validation accuracy:  0.727
==========================================================
Valid set f1 score for best params: 0.702
==========================================================
Confusion Matrix
            y_predict Yes  y_predict No
y_true Yes            268           205
y_ture No              23           340
```

2) GBM RandomizedSearchCV 2 (narrow)

In [48]:
```
# RandomizedSearchCV 2

np.random.seed(42)

learning_rate = np.random.uniform(0, 0.5, 10)
n_estimators = [int(x) for x in np.linspace(start = 200, stop = 600, num = 30)]
min_samples_split = [int(x) for x in np.linspace(5,20,10)]
min_samples_leaf = [int(x) for x in np.linspace(30,60,10)]
max_depth = [int(x) for x in np.linspace(5,12, num=5)]
subsample = np.random.uniform(0.3, 0.7, 20)

param_grid = {'learning_rate': learning_rate,
             'n_estimators':n_estimators,
             'min_samples_split':min_samples_split,
             'min_samples_leaf':min_samples_leaf,
             'max_depth':max_depth,
             'subsample':subsample}
```

In [49]:
```
bgc = GradientBoostingClassifier(random_state=42)
bgc_cv = RandomizedSearchCV(bgc, param_grid, cv = 5, n_jobs=-1, random_state=42
)
bgc_cv.fit(final_train, y_train)
print('Best Params of GBM:', bgc_cv.best_params_)
print('==========================================================')
print('Best training accuracy: ', round(bgc_cv.best_score_,3))
y_pred = bgc_cv.predict(final_valid)
f1 = f1_score(valid[1], y_pred)
accuray = accuracy_score(valid[1], y_pred)
print('Best validation accuracy: ', round(accuray,3))
print('==========================================================')
print('Valid set f1 score for best params:', round(f1,3))
print('==========================================================')
print('Confusion Matrix')
confusion = pd.DataFrame(confusion_matrix(valid[1], y_pred, labels=[1,0]),
                        index=['y_true Yes','y_ture No'],
                        columns=['y_predict Yes','y_predict No'])
print(confusion)
```

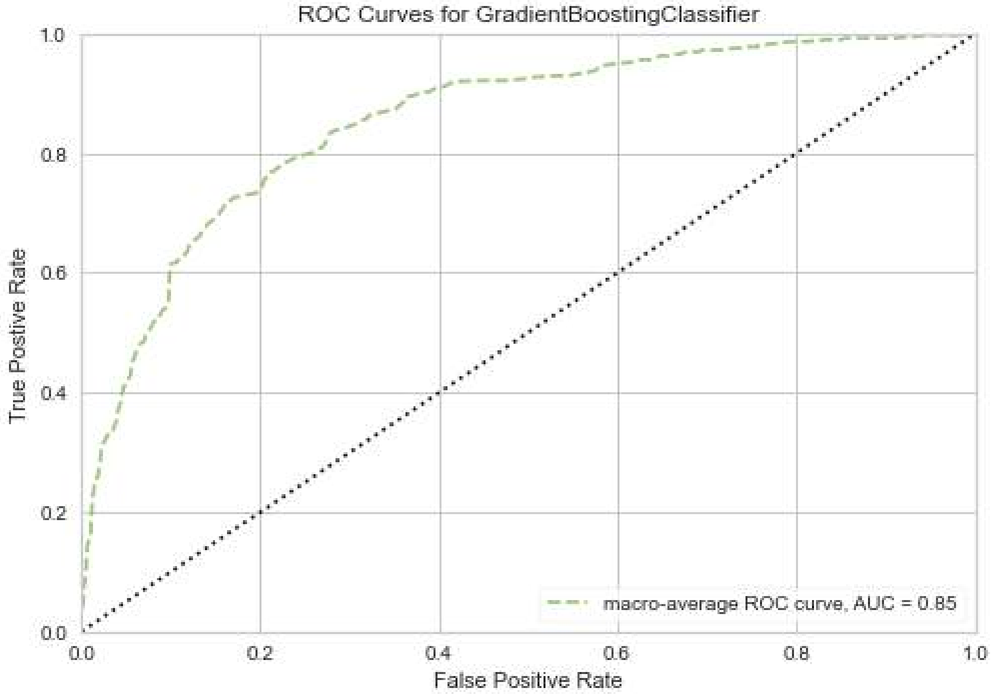
```
Best Params of GBM: {'subsample': 0.38493564427131044, 'n_estimators': 296, 'mi
n_samples_split': 6, 'min_samples_leaf': 36, 'max_depth': 10, 'learning_rate':
0.07799726016810132}
==========================================================
Best training accuracy:  0.817
Best validation accuracy:  0.744
==========================================================
Valid set f1 score for best params: 0.727
==========================================================
Confusion Matrix
            y_predict Yes  y_predict No
y_true Yes            285           188
y_ture No              26           337
```

3) Final GradientBoostingClassifier model

In [50]:
```
# Final model
bgm = GradientBoostingClassifier(random_state=42,
                                subsample= 0.38493564427131044, n_estimators= 2
96,
                                min_samples_split= 6, min_samples_leaf= 36,
                                max_depth= 10, learning_rate= 0.077997260168101
32)
```

4) ROC curve and AUC score

In [51]:
```
visualizer = ROCAUC(bgm, classes=[0, 1], micro=False, macro=True, per_class=False)
visualizer.fit(final_train, y_train)
visualizer.score(final_valid, valid[1])
visualizer.show()
print('roc_auc_score:', round(roc_auc_score(valid[1], y_pred),3))
```




```
roc_auc_score: 0.765
```

4. XGBoost classifier

1) XGB RandomizedSearchCV 1

In [52]:
```
# RandomizedSearchCV 1

np.random.seed(42)

min_child_weight = [int(x) for x in np.linspace(1,10, num=10)]
max_depth = [int(x) for x in np.linspace(1,10, num=10)]
subsample = np.random.uniform(0, 1, 20)
gamma = np.logspace(-10, 10, 20)
colsample_bytree = np.random.uniform(0, 1, 20)
learning_rate = np.random.uniform(0, 1, 10)
n_estimators = [int(x) for x in np.linspace(start = 10, stop = 1000, num = 30)]

param_grid = {'min_child_weight': min_child_weight,
              'max_depth':max_depth,
              'subsample':subsample,
              'gamma':gamma,
              'colsample_bytree':colsample_bytree,
              'learning_rate': learning_rate,
              'n_estimators':n_estimators}
```

```
In [53]: import xgboost as xgb
xgb = xgb.XGBClassifier(objective='binary:logistic',
                        silent=True, nthread=1, random_state=42, n_jobs=-1)
xgb_cv = RandomizedSearchCV(xgb, param_grid, cv = 5, n_jobs=-1, random_state=42
)
xgb_cv.fit(final_train, y_train)
print('Best Params of XGBClassifier:', xgb_cv.best_params_)
print('=========================================================')
print('Best training accuracy  : ', round(xgb_cv.best_score_,3))
y_pred = xgb_cv.predict(final_valid)
f1 = f1_score(valid[1], y_pred)
accuray = accuracy_score(valid[1], y_pred)
print('Best validation accuracy: ', round(accuray,3))
print('=========================================================')
print('Valid set f1 score for best params:', round(f1,3))
print('=========================================================')
print('Confusion Matrix')
confusion = pd.DataFrame(confusion_matrix(valid[1], y_pred, labels=[1,0]),
                         index=['y_true Yes','y_ture No'],
                         columns=['y_predict Yes','y_predict No'])
print(confusion)
```

```
Best Params of XGBClassifier: {'subsample': 0.43194501864211576, 'n_estimator
s': 931, 'min_child_weight': 6, 'max_depth': 6, 'learning_rate': 0.034388521115
218396, 'gamma': 0.026366508987303555, 'colsample_bytree': 0.9488855372533332}
=========================================================
Best training accuracy  :  0.823
Best validation accuracy:  0.754
=========================================================
Valid set f1 score for best params: 0.744
=========================================================
Confusion Matrix
            y_predict Yes  y_predict No
y_true Yes            299           174
y_ture No              32           331
```

2) XGB RandomizedSearchCV 2 (narrow)

```
In [54]: # RandomizedSearchCV 2

np.random.seed(42)

min_child_weight = [int(x) for x in np.linspace(4,8, num=5)]
max_depth = [int(x) for x in np.linspace(4,8, num=5)]
subsample = np.random.uniform(0.2, 0.6, 20)
gamma = np.logspace(-10, 10, 20)
colsample_bytree = np.random.uniform(0.6, 1, 20)
learning_rate = np.random.uniform(0, 0.5, 20)
n_estimators = [int(x) for x in np.linspace(start = 500, stop = 1000, num = 30
)]

param_grid = {'min_child_weight': min_child_weight,
              'max_depth':max_depth,
              'subsample':subsample,
              'gamma':gamma,
              'colsample_bytree':colsample_bytree,
              'learning_rate': learning_rate,
              'n_estimators':n_estimators}
```

In [55]:
```
import xgboost as xgb
xgb = xgb.XGBClassifier(objective='binary:logistic',
                        silent=True, nthread=1, random_state=42, n_jobs=-1)
xgb_cv = RandomizedSearchCV(xgb, param_grid, cv = 5, n_jobs=-1, random_state=42
)
xgb_cv.fit(final_train, y_train)
print('Best Params of XGBClassifier:', xgb_cv.best_params_)
print('=========================================================')
print('Best training accuracy  : ', round(xgb_cv.best_score_,3))
y_pred = xgb_cv.predict(final_valid)
f1 = f1_score(valid[1], y_pred)
accuray = accuracy_score(valid[1], y_pred)
print('Best validation accuracy: ', round(accuray,3))
print('=========================================================')
print('Valid set f1 score for best params:', round(f1,3))
print('=========================================================')
print('Confusion Matrix')
confusion = pd.DataFrame(confusion_matrix(valid[1], y_pred, labels=[1,0]),
                        index=['y_true Yes','y_ture No'],
                        columns=['y_predict Yes','y_predict No'])
print(confusion)
```

```
Best Params of XGBClassifier: {'subsample': 0.20823379771832098, 'n_estimator
s': 862, 'min_child_weight': 8, 'max_depth': 5, 'learning_rate': 0.017194260557
609198, 'gamma': 0.2976351441631313, 'colsample_bytree': 0.7824279936868144}
=========================================================
Best training accuracy  :  0.821
Best validation accuracy:  0.786
=========================================================
Valid set f1 score for best params: 0.811
=========================================================
Confusion Matrix
            y_predict Yes  y_predict No
y_true Yes            385            88
y_ture No              91           272
```

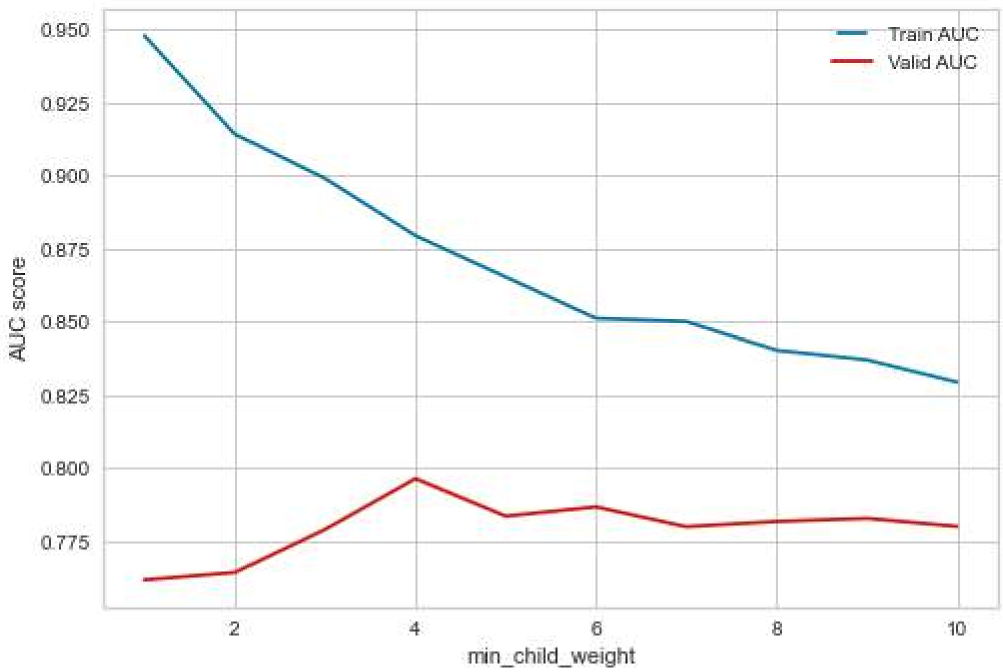
3) XGB RandomizedSearchCV result

In [57]:
```
# After RandomizedsearchCV
import xgboost as xgb
xgb = xgb.XGBClassifier(objective='binary:logistic',
                        silent=True, nthread=1, random_state=42, n_jobs=-1,
                        subsample= 0.20823379771832098, n_estimators= 862,
                        min_child_weight= 8, max_depth= 5,
                        learning_rate= 0.017194260557609198, gamma= 0.2976351441631
313,
                        colsample_bytree= 0.7824279936868144)
```

4) Selecting the best 'min_child_weight' meta-parameter based on RandomizedSearchCV result

In [58]:

```
# Selecting the best 'min_child_weight' meta-parameter
# by comparing train and valid set's AUC score

min_child_weight = [10,9,8,7,6,5,4,3,2,1]

train_results = []
valid_results = []

for i in min_child_weight:
    import xgboost as xgb
    xgb = xgb.XGBClassifier(objective='binary:logistic',
                    silent=True, nthread=1, random_state=42, n_jobs=-1,
                    subsample= 0.20823379771832098, n_estimators= 862,
                    min_child_weight= i, max_depth= 5,
                    learning_rate= 0.017194260557609198, gamma= 0.2976351441631
313,
                    colsample_bytree= 0.7824279936868144)
    xgb.fit(final_train, y_train)

    train_pred = xgb.predict(final_train)

    false_positive_rate, true_positive_rate, thresholds = roc_curve(y_train, tr
ain_pred)
    roc_auc = auc(false_positive_rate, true_positive_rate)
    train_results.append(roc_auc)

    valid_pred = xgb.predict(final_valid)

    false_positive_rate, true_positive_rate, thresholds = roc_curve(valid[1], v
alid_pred)
    roc_auc = auc(false_positive_rate, true_positive_rate)
    valid_results.append(roc_auc)

line1, = plt.plot(min_child_weight, train_results, 'b', label='Train AUC')
line2, = plt.plot(min_child_weight, valid_results, 'r', label='Valid AUC')
plt.legend(handler_map={line1: HandlerLine2D(numpoints=22)})
plt.ylabel('AUC score')
plt.xlabel('min_child_weight')
plt.show()
```

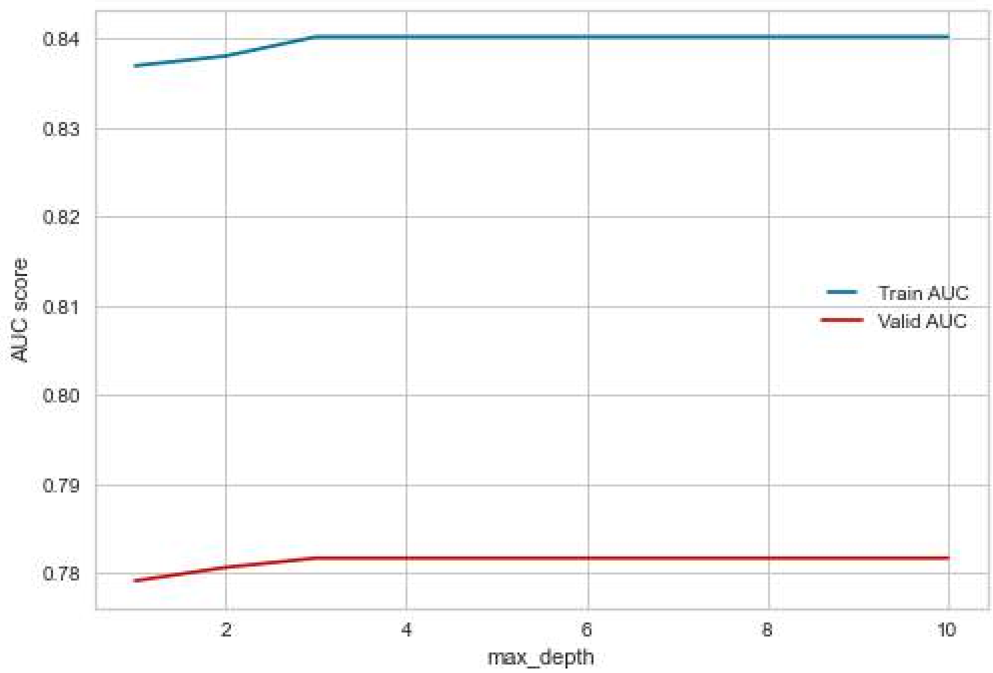


min_child_weight = 4 is the best

5) Selecting the best 'max_depth' meta-parameter based on RandomizedSearchCV result

In [59]:

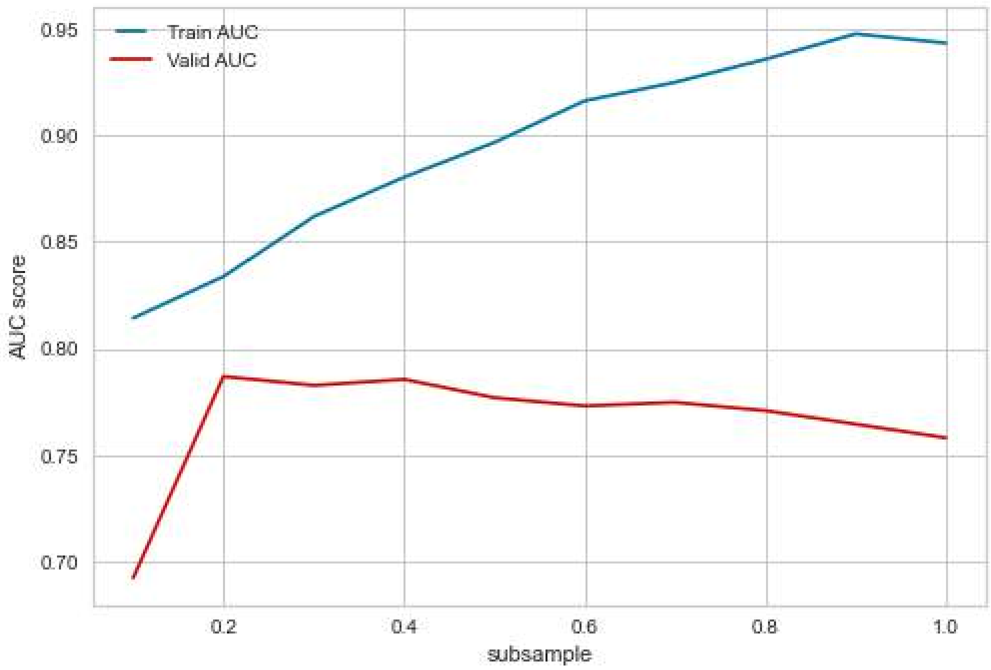
```
# Selecting the best 'max_depth' meta-parameter
# by comparing train and valid set's AUC score

max_depth = [10,9,8,7,6,5,4,3,2,1]

train_results = []
valid_results = []

for i in max_depth:
    import xgboost as xgb
    xgb = xgb.XGBClassifier(objective='binary:logistic',
                        silent=True, nthread=1, random_state=42, n_jobs=-1,
                        subsample= 0.20823379771832098, n_estimators= 862,
                        min_child_weight= 8, max_depth= i,
                        learning_rate= 0.017194260557609198, gamma= 0.2976351441631
313,
                        colsample_bytree= 0.7824279936868144)
    xgb.fit(final_train, y_train)

    train_pred = xgb.predict(final_train)

    false_positive_rate, true_positive_rate, thresholds = roc_curve(y_train, tr
ain_pred)
    roc_auc = auc(false_positive_rate, true_positive_rate)
    train_results.append(roc_auc)

    valid_pred = xgb.predict(final_valid)

    false_positive_rate, true_positive_rate, thresholds = roc_curve(valid[1], v
alid_pred)
    roc_auc = auc(false_positive_rate, true_positive_rate)
    valid_results.append(roc_auc)

line1, = plt.plot(max_depth, train_results, 'b', label='Train AUC')
line2, = plt.plot(max_depth, valid_results, 'r', label='Valid AUC')
plt.legend(handler_map={line1: HandlerLine2D(numpoints=22)})
plt.ylabel('AUC score')
plt.xlabel('max_depth')
plt.show()
```



max_depth = 3

6) Selecting the best 'subsample' meta-parameter based on RandomizedSearchCV result

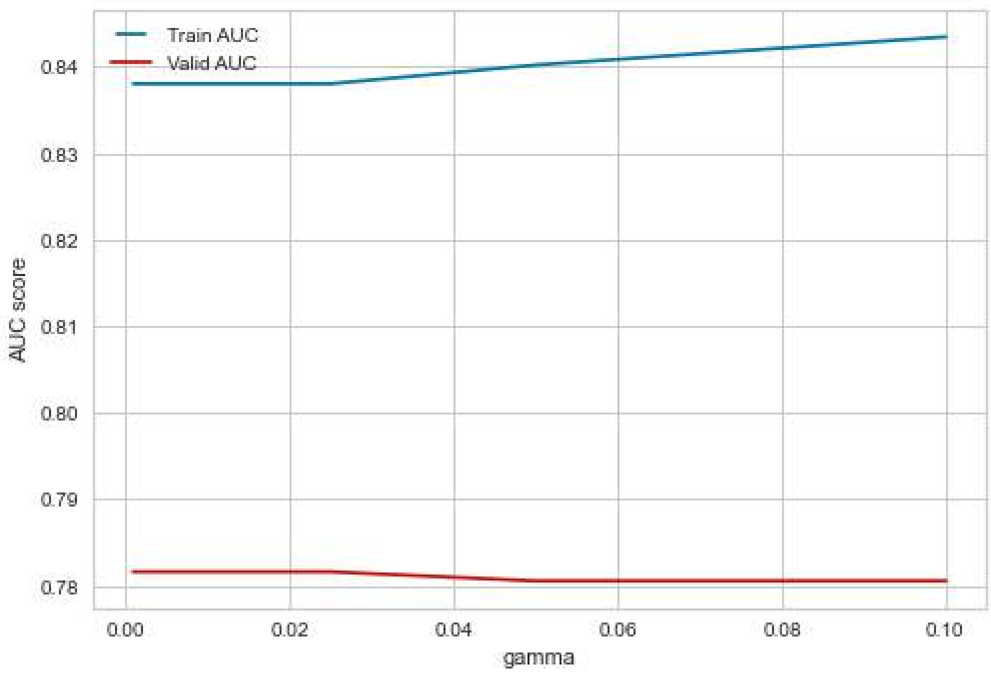
In [60]:
```python
# Selecting the best 'subsample' meta-parameter
# by comparing train and valid set's AUC score

subsample = [1, 0.9, 0.8, 0.7, 0.6, 0.5, 0.4, 0.3, 0.2, 0.1]

train_results = []
valid_results = []

for i in subsample:
    import xgboost as xgb
    xgb = xgb.XGBClassifier(objective='binary:logistic',
                    silent=True, nthread=1, random_state=42, n_jobs=-1,
                    subsample= i, n_estimators= 862,
                    min_child_weight= 8, max_depth= 5,
                    learning_rate= 0.017194260557609198, gamma= 0.2976351441631
313,
                    colsample_bytree= 0.7824279936868144)
    xgb.fit(final_train, y_train)

    train_pred = xgb.predict(final_train)

    false_positive_rate, true_positive_rate, thresholds = roc_curve(y_train, tr
ain_pred)
    roc_auc = auc(false_positive_rate, true_positive_rate)
    train_results.append(roc_auc)

    valid_pred = xgb.predict(final_valid)

    false_positive_rate, true_positive_rate, thresholds = roc_curve(valid[1], v
alid_pred)
    roc_auc = auc(false_positive_rate, true_positive_rate)
    valid_results.append(roc_auc)

line1, = plt.plot(subsample, train_results, 'b', label='Train AUC')
line2, = plt.plot(subsample, valid_results, 'r', label='Valid AUC')
plt.legend(handler_map={line1: HandlerLine2D(numpoints=22)})
plt.ylabel('AUC score')
plt.xlabel('subsample')
plt.show()
```



subsample = 0.2

7) Selecting the best 'gamma' meta-parameter based on RandomizedSearchCV result

In [61]:
```
# Selecting the best 'gamma' meta-parameter
# by comparing train and valid set's AUC score

gamma = [0.1, 0.05, 0.025, 0.01, 0.005, 0.0025, 0.001]

train_results = []
valid_results = []

for i in gamma:
    import xgboost as xgb
    xgb = xgb.XGBClassifier(objective='binary:logistic',
                    silent=True, nthread=1, random_state=42, n_jobs=-1,
                    subsample= 0.20823379771832098, n_estimators= 862,
                    min_child_weight= 8, max_depth= 5,
                    learning_rate= 0.017194260557609198, gamma= i,
                    colsample_bytree= 0.7824279936868144)
    xgb.fit(final_train, y_train)

    train_pred = xgb.predict(final_train)

    false_positive_rate, true_positive_rate, thresholds = roc_curve(y_train, tr
ain_pred)
    roc_auc = auc(false_positive_rate, true_positive_rate)
    train_results.append(roc_auc)

    valid_pred = xgb.predict(final_valid)

    false_positive_rate, true_positive_rate, thresholds = roc_curve(valid[1], v
alid_pred)
    roc_auc = auc(false_positive_rate, true_positive_rate)
    valid_results.append(roc_auc)

line1, = plt.plot(gamma, train_results, 'b', label='Train AUC')
line2, = plt.plot(gamma, valid_results, 'r', label='Valid AUC')
plt.legend(handler_map={line1: HandlerLine2D(numpoints=22)})
plt.ylabel('AUC score')
plt.xlabel('gamma')
plt.show()
```

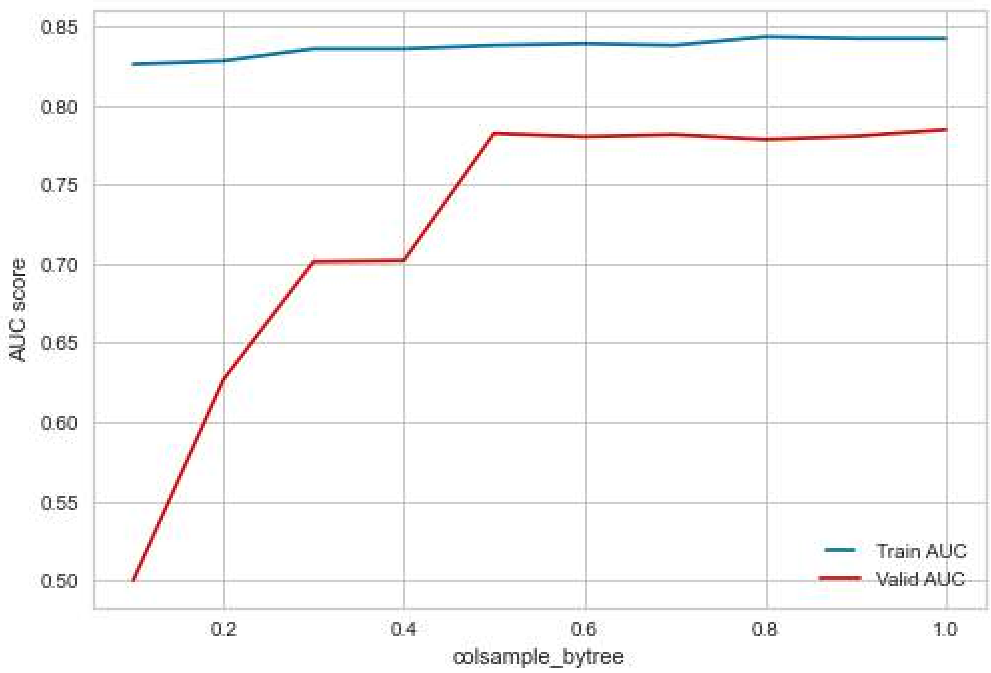


gamma = 0.025

8) Selecting the best 'colsample_bytree' meta-parameter based on RandomizedSearchCV result

In [62]:

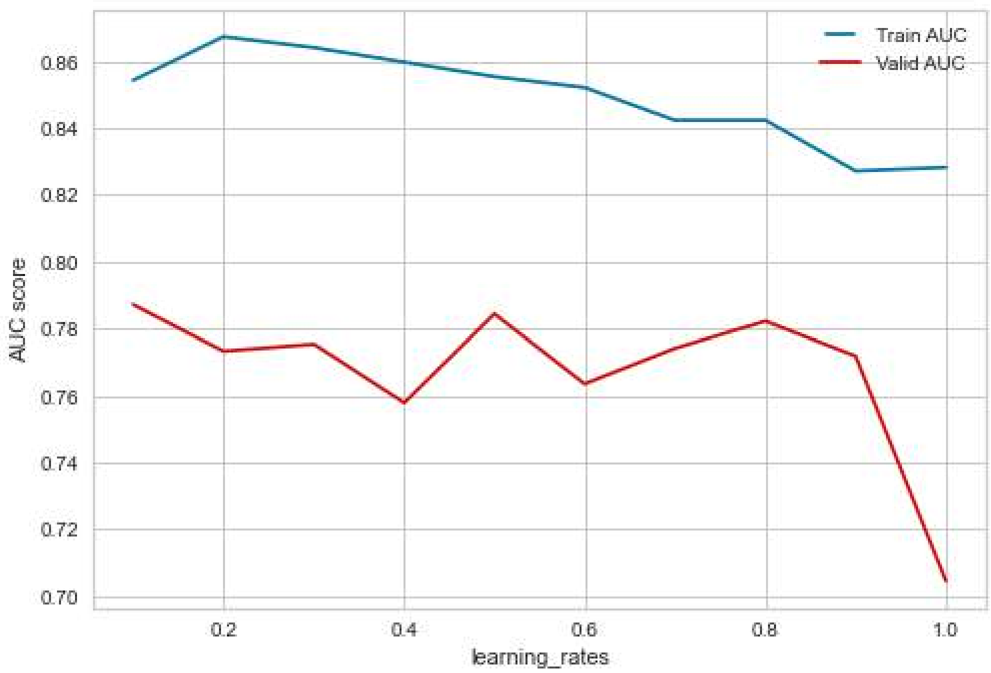
```
# Selecting the best 'colsample_bytree' meta-parameter
# by comparing train and valid set's AUC score

colsample_bytree = [1, 0.9, 0.8, 0.7, 0.6, 0.5, 0.4, 0.3, 0.2, 0.1]

train_results = []
valid_results = []

for i in colsample_bytree:
    import xgboost as xgb
    xgb = xgb.XGBClassifier(objective='binary:logistic',
                    silent=True, nthread=1, random_state=42, n_jobs=-1,
                    subsample= 0.20823379771832098, n_estimators= 862,
                    min_child_weight= 8, max_depth= 5,
                    learning_rate= 0.017194260557609198, gamma= 0.2976351441631
313,
                    colsample_bytree= i)
    xgb.fit(final_train, y_train)

    train_pred = xgb.predict(final_train)

    false_positive_rate, true_positive_rate, thresholds = roc_curve(y_train, tr
ain_pred)
    roc_auc = auc(false_positive_rate, true_positive_rate)
    train_results.append(roc_auc)

    valid_pred = xgb.predict(final_valid)

    false_positive_rate, true_positive_rate, thresholds = roc_curve(valid[1], v
alid_pred)
    roc_auc = auc(false_positive_rate, true_positive_rate)
    valid_results.append(roc_auc)

line1, = plt.plot(colsample_bytree, train_results, 'b', label='Train AUC')
line2, = plt.plot(colsample_bytree, valid_results, 'r', label='Valid AUC')
plt.legend(handler_map={line1: HandlerLine2D(numpoints=22)})
plt.ylabel('AUC score')
plt.xlabel('colsample_bytree')
plt.show()
```



colsample_bytree = 0.5

9) Selecting the best 'learning_rates' meta-parameter based on RandomizedSearchCV result

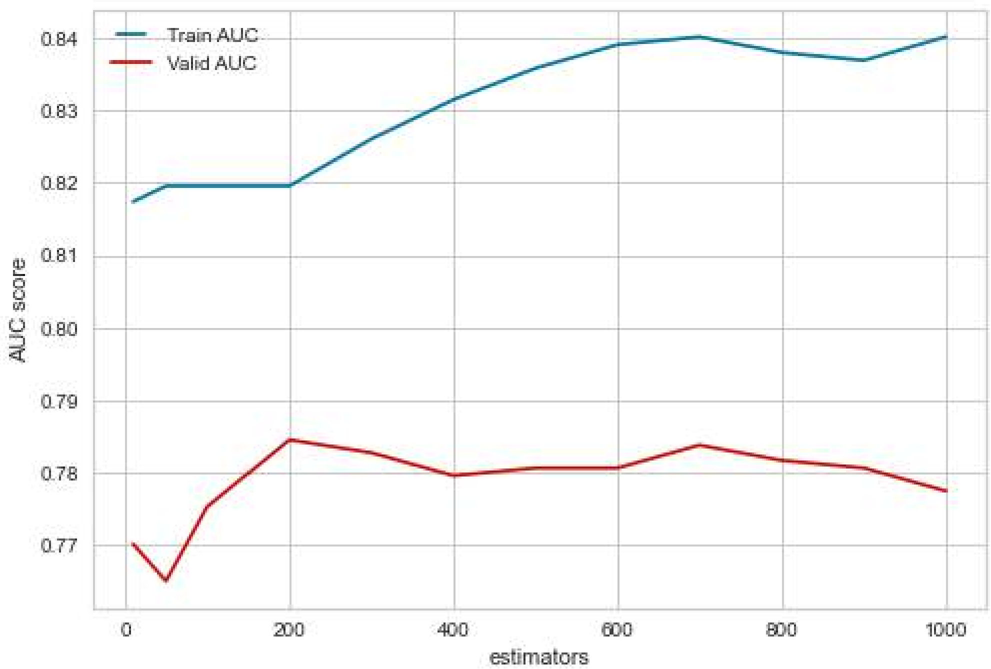
In [63]:
```
# Selecting the best 'learning_rates' meta-parameter
# by comparing train and valid set's AUC score

learning_rates = [1, 0.9, 0.8, 0.7, 0.6, 0.5, 0.4, 0.3, 0.2, 0.1]

train_results = []
valid_results = []

for i in learning_rates:
    import xgboost as xgb
    xgb = xgb.XGBClassifier(objective='binary:logistic',
                    silent=True, nthread=1, random_state=42, n_jobs=-1,
                    subsample= 0.20823379771832098, n_estimators= 862,
                    min_child_weight= 8, max_depth= 5,
                    learning_rate= i, gamma= 0.2976351441631313,
                    colsample_bytree= 0.7824279936868144)
    xgb.fit(final_train, y_train)

    train_pred = xgb.predict(final_train)

    false_positive_rate, true_positive_rate, thresholds = roc_curve(y_train, tr
ain_pred)
    roc_auc = auc(false_positive_rate, true_positive_rate)
    train_results.append(roc_auc)

    valid_pred = xgb.predict(final_valid)

    false_positive_rate, true_positive_rate, thresholds = roc_curve(valid[1], v
alid_pred)
    roc_auc = auc(false_positive_rate, true_positive_rate)
    valid_results.append(roc_auc)

line1, = plt.plot(learning_rates, train_results, 'b', label='Train AUC')
line2, = plt.plot(learning_rates, valid_results, 'r', label='Valid AUC')
plt.legend(handler_map={line1: HandlerLine2D(numpoints=22)})
plt.ylabel('AUC score')
plt.xlabel('learning_rates')
plt.show()
```



learning_rate = 0.5

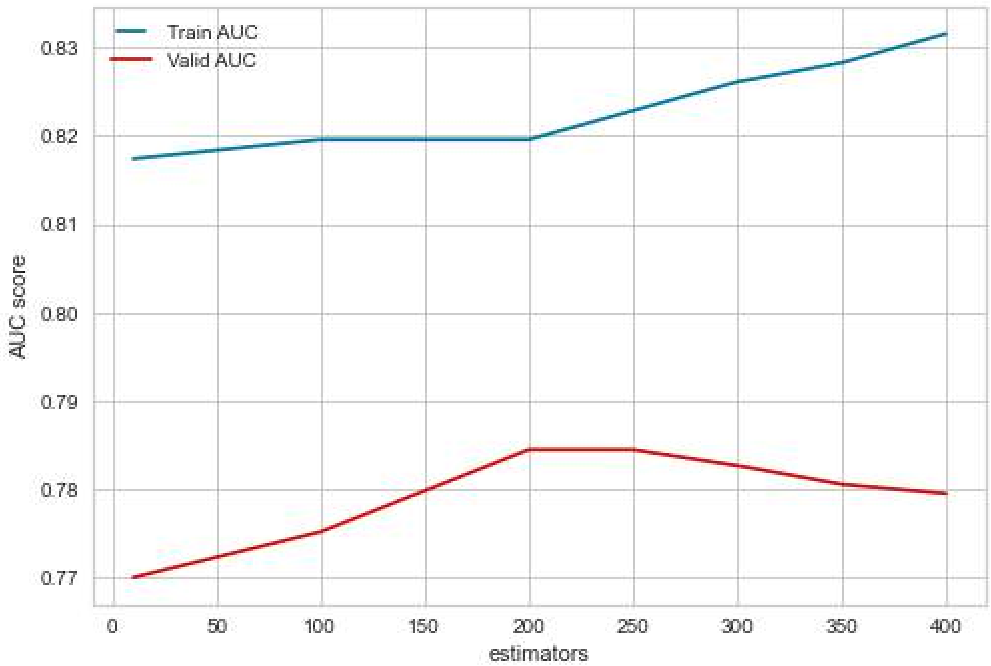
9) Selecting the best 'estimators' meta-parameter based on RandomizedSearchCV result

In [64]:
```python
# Selecting the best 'estimators' meta-parameter
# by comparing train and valid set's AUC score

estimators = [10, 50, 100, 150, 200, 300, 400, 500, 600, 700, 800, 900, 1000]

train_results = []
valid_results = []

for i in estimators:
    import xgboost as xgb
    xgb = xgb.XGBClassifier(objective='binary:logistic',
                    silent=True, nthread=1, random_state=42, n_jobs=-1,
                    subsample= 0.20823379771832098, n_estimators= i,
                    min_child_weight= 8, max_depth= 5,
                    learning_rate= 0.017194260557609198, gamma= 0.2976351441631
313,
                    colsample_bytree= 0.7824279936868144)
    xgb.fit(final_train, y_train)

    train_pred = xgb.predict(final_train)

    false_positive_rate, true_positive_rate, thresholds = roc_curve(y_train, tr
ain_pred)
    roc_auc = auc(false_positive_rate, true_positive_rate)
    train_results.append(roc_auc)

    valid_pred = xgb.predict(final_valid)

    false_positive_rate, true_positive_rate, thresholds = roc_curve(valid[1], v
alid_pred)
    roc_auc = auc(false_positive_rate, true_positive_rate)
    valid_results.append(roc_auc)

line1, = plt.plot(estimators, train_results, 'b', label='Train AUC')
line2, = plt.plot(estimators, valid_results, 'r', label='Valid AUC')
plt.legend(handler_map={line1: HandlerLine2D(numpoints=22)})
plt.ylabel('AUC score')
plt.xlabel('estimators')
plt.show()
```

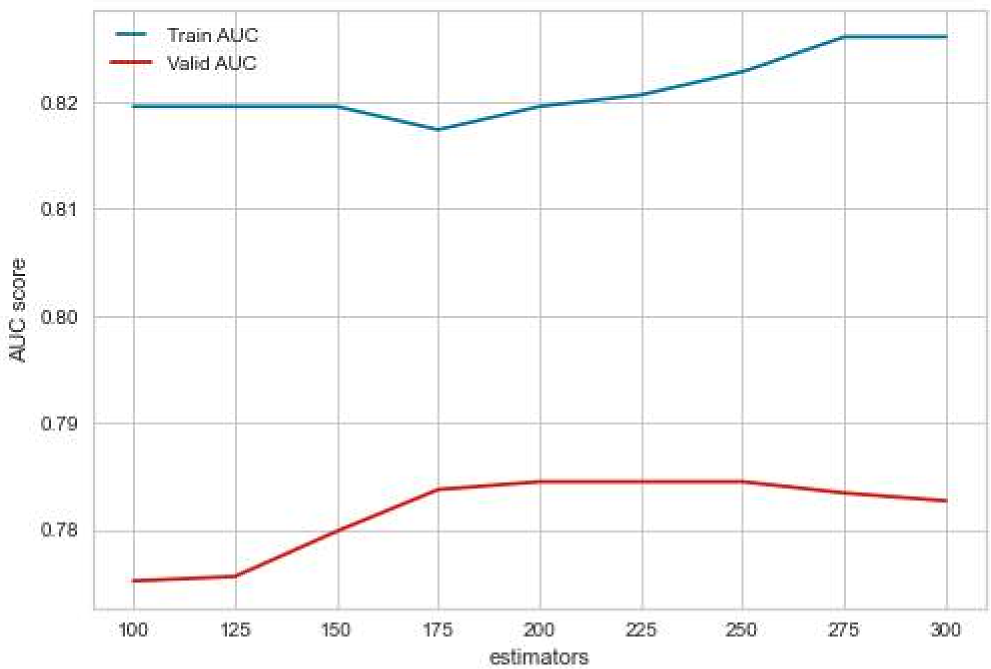


In [66]:

```
estimators = [10, 100, 150, 200, 250, 300, 350, 400]

train_results = []
valid_results = []

for i in estimators:
    import xgboost as xgb
    xgb = xgb.XGBClassifier(objective='binary:logistic',
                    silent=True, nthread=1, random_state=42, n_jobs=-1,
                    subsample= 0.20823379771832098, n_estimators= i,
                    min_child_weight= 8, max_depth= 5,
                    learning_rate= 0.017194260557609198, gamma= 0.2976351441631
313,
                    colsample_bytree= 0.7824279936868144)
    xgb.fit(final_train, y_train)

    train_pred = xgb.predict(final_train)

    false_positive_rate, true_positive_rate, thresholds = roc_curve(y_train, tr
ain_pred)
    roc_auc = auc(false_positive_rate, true_positive_rate)
    train_results.append(roc_auc)

    valid_pred = xgb.predict(final_valid)

    false_positive_rate, true_positive_rate, thresholds = roc_curve(valid[1], v
alid_pred)
    roc_auc = auc(false_positive_rate, true_positive_rate)
    valid_results.append(roc_auc)

line1, = plt.plot(estimators, train_results, 'b', label='Train AUC')
line2, = plt.plot(estimators, valid_results, 'r', label='Valid AUC')
plt.legend(handler_map={line1: HandlerLine2D(numpoints=22)})
plt.ylabel('AUC score')
plt.xlabel('estimators')
plt.show()
```



In [65]:
```
estimators = [100, 125, 150, 175, 200, 225, 250, 275, 300]

train_results = []
valid_results = []

for i in estimators:
    import xgboost as xgb
    xgb = xgb.XGBClassifier(objective='binary:logistic',
                        silent=True, nthread=1, random_state=42, n_jobs=-1,
                        subsample= 0.20823379771832098, n_estimators= i,
                        min_child_weight= 8, max_depth= 5,
                        learning_rate= 0.017194260557609198, gamma= 0.2976351441631
313,
                        colsample_bytree= 0.7824279936868144)
    xgb.fit(final_train, y_train)

    train_pred = xgb.predict(final_train)

    false_positive_rate, true_positive_rate, thresholds = roc_curve(y_train, tr
ain_pred)
    roc_auc = auc(false_positive_rate, true_positive_rate)
    train_results.append(roc_auc)

    valid_pred = xgb.predict(final_valid)

    false_positive_rate, true_positive_rate, thresholds = roc_curve(valid[1], v
alid_pred)
    roc_auc = auc(false_positive_rate, true_positive_rate)
    valid_results.append(roc_auc)

line1, = plt.plot(estimators, train_results, 'b', label='Train AUC')
line2, = plt.plot(estimators, valid_results, 'r', label='Valid AUC')
plt.legend(handler_map={line1: HandlerLine2D(numpoints=22)})
plt.ylabel('AUC score')
plt.xlabel('estimators')
plt.show()
```



n_estimators= 175

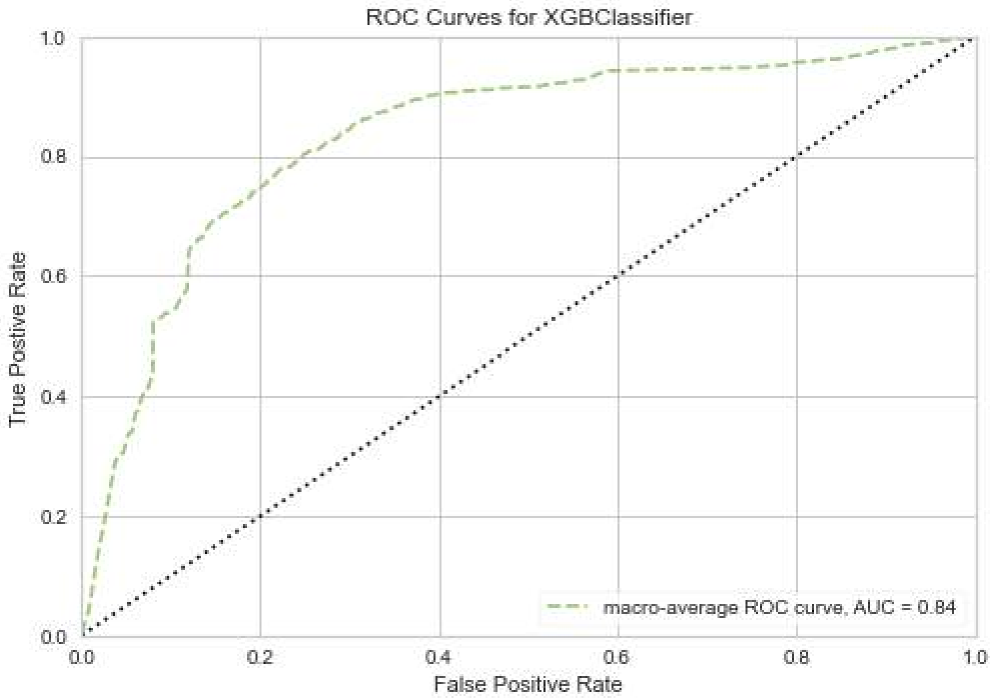
10) Result of comparing each meta-parameter with train and validation AUC score

In [67]:
```
import xgboost as xgb
xgb = xgb.XGBClassifier(objective='binary:logistic',
                        silent=True, nthread=1, random_state=42, n_jobs=-1,
                        subsample= 0.2, n_estimators= 175,
                        min_child_weight= 4, max_depth= 3,
                        learning_rate= 0.5, gamma= 0.025,
                        colsample_bytree= 0.5)

xgb.fit(final_train, y_train)

print("train set accuracy: {:.3f}".format(xgb.score(final_train, y_train)))
print("valid set accuracy : {:.3f}".format(xgb.score(final_valid, valid[1])))
print('==========================================================')
y_pred = xgb.predict(final_valid)
f1 = f1_score(valid[1], y_pred)
print('Valid set f1 score for best params:', round(f1,3))
print('==========================================================')
print('Confusion Matrix')
confusion = pd.DataFrame(confusion_matrix(valid[1], y_pred, labels=[1,0]),
                             index=['y_true Yes','y_ture No'],
                             columns=['y_predict Yes','y_predict No'])
print(confusion)
```

```
train set accuracy: 0.867
valid set accuracy : 0.778
==========================================================
Valid set f1 score for best params: 0.801
==========================================================
Confusion Matrix
            y_predict Yes  y_predict No
y_true Yes            375            98
y_ture No              88           275
```

11) ROC curve and AUC score

In [68]:
```
visualizer = ROCAUC(xgb, classes=[0, 1], micro=False, macro=True, per_class=False)
visualizer.fit(final_train, y_train)
visualizer.score(final_valid, valid[1])
visualizer.show()
print('roc_auc_score:', round(roc_auc_score(valid[1], y_pred),3))
```




```
roc_auc_score: 0.775
```

12) FInal XGBoost Classifier model from RandomizedSearchCV

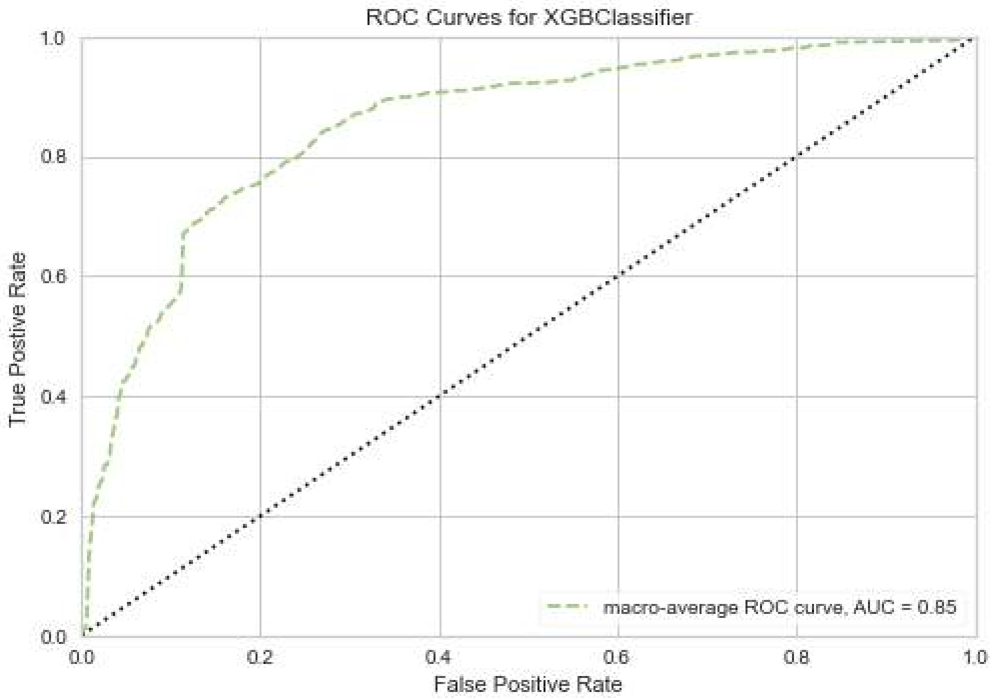
In [69]:

```
# final model of XGB
import xgboost as xgb
xgb = xgb.XGBClassifier(objective='binary:logistic',
                    silent=True, nthread=1, random_state=42, n_jobs=-1,
                    subsample= 0.20823379771832098, n_estimators= 862,
                    min_child_weight= 8, max_depth= 5,
                    learning_rate= 0.017194260557609198, gamma= 0.2976351441631
313,
                    colsample_bytree= 0.7824279936868144)

xgb.fit(final_train, y_train)

print("train set accuracy: {:.3f}".format(xgb.score(final_train, y_train)))
print("valid set accuracy : {:.3f}".format(xgb.score(final_valid, valid[1])))
print('=========================================================')
y_pred = xgb.predict(final_valid)
f1 = f1_score(valid[1], y_pred)
print('Valid set f1 score for best params:', round(f1,3))
print('=========================================================')
print('Confusion Matrix')
confusion = pd.DataFrame(confusion_matrix(valid[1], y_pred, labels=[1,0]),
                        index=['y_true Yes','y_ture No'],
                        columns=['y_predict Yes','y_predict No'])
print(confusion)
```

```
train set accuracy: 0.840
valid set accuracy : 0.786
=========================================================
Valid set f1 score for best params: 0.811
=========================================================
Confusion Matrix
            y_predict Yes  y_predict No
y_true Yes            385            88
y_ture No              91           272
```

13) ROC curve and AUC score

In [70]:
```
visualizer = ROCAUC(xgb, classes=[0, 1], micro=False, macro=True, per_class=Fal
se)
visualizer.fit(final_train, y_train)
visualizer.score(final_valid, valid[1])
visualizer.show()
print('roc_auc_score:', round(roc_auc_score(valid[1], y_pred),3))
```




```
roc_auc_score: 0.782
```

14) Final model

In [71]:
```
# final model of XGB
import xgboost as xgb
xgb = xgb.XGBClassifier(objective='binary:logistic',
                        silent=True, nthread=1, random_state=42, n_jobs=-1,
                        subsample= 0.20823379771832098, n_estimators= 862,
                        min_child_weight= 8, max_depth= 5,
                        learning_rate= 0.017194260557609198, gamma= 0.2976351441631
313,
                        colsample_bytree= 0.7824279936868144)
```

D. Selecting the best model

1. Function for getting prediction score of three models

```
In [203]: def predicting_top_three_models(model, total_holdout_sets, now, ws, ows):
              f1_scores = []
              valid_accuracy = []
              train_accuracy = []

              # for each holdout set, compute f1 score
              for i in range(total_holdout_sets):
                  valid = get_dataset_value(now-2*ows, ws, ows)
                  train = get_dataset_value(now-3*ows, ws, ows)

                   # output feature changes to binary, 1: non- churn, 0: churn
                  valid[1][valid[1]>0] = 1 # non-chrun
                  train[1][train[1]>0] = 1 # non-chrun

                  # Balancing unbalanced output feature in train data set using SMOTE
                  smote = SMOTE(random_state=42)
                  X_train, y_train = smote.fit_resample(train[0], train[1])

                  X_train = pd.DataFrame(X_train,
                                columns=['total_values','total_quantity','avg_between',
                                        'f1','f2','f3','f4','f5','f6','f7','f8','f9','f1
          0','f11'])
                  y_train = pd.DataFrame(y_train)

                  # standardizing Temporal data in train set
                  train_X = pd.DataFrame()

                  for i in X_train.iloc[:,3:14].values:
                      a = i - X_train.iloc[:,3:14].values.sum()
                      b = a / np.std(X_train.iloc[:,3:14].values)

                      new_row = pd.DataFrame( [[b]] )
                      train_X = train_X.append(new_row, ignore_index = True)

                  train_X.columns = ['f']
                  train_X = pd.DataFrame(train_X.f.tolist(),
                                      columns=['f1','f2','f3','f4','f5','f6','f7','f8',
                                                   'f9','f10','f11'])

                  # standardizing traditional data in train set
                  # Step 1: log1p
                  train_X2 = X_train.drop(columns=['f1','f2','f3','f4','f5','f6','f7','f
          8','f9','f10','f11'])
                  train_X2_log = np.log1p(train_X2)
                  # Step 2: StandardScaler
                  scaler = StandardScaler()
                  train_X2_scaled = scaler.fit_transform(train_X2_log)

                  # transform into a dataframe
                  train_X2_scaled = pd.DataFrame(train_X2_scaled, index=train_X2_log.inde
          x,
                                      columns=train_X2_log.columns)
                  final_train = pd.concat([train_X2_scaled, train_X], axis=1)
                  final_train = round(final_train,2)

                  # # standardizing Temporal data in validation set
                  valid_X = pd.DataFrame()

                  for i in valid[0].iloc[:,3:14].values:
                      a = i - valid[0].iloc[:,3:14].values.sum()
                      b = a / np.std(valid[0].iloc[:,3:14].values)
```

```
            new_row = pd.DataFrame( [[b]] )
            valid_X = valid_X.append(new_row, ignore_index = True)

        valid_X.columns = ['f']
        valid_X = pd.DataFrame(valid_X.f.tolist(),
                            columns=['f1','f2','f3','f4','f5','f6','f7','f8',
                                     'f9','f10','f11'])

        # standardizing traditional data in validation set
        # Step 1: log1p
        valid_X2 = valid[0].drop(columns=['f1','f2','f3','f4','f5','f6','f7','f
8','f9','f10','f11'])
        valid_X2_log = np.log1p(valid_X2)
        # Step 2: StandardScaler
        scaler = StandardScaler()
        valid_X2_scaled = scaler.fit_transform(valid_X2_log)

        # transform into a dataframe
        valid_X2_scaled = pd.DataFrame(valid_X2_scaled, index=valid_X2_log.inde
x,
                                     columns=valid_X2_log.columns)
        # Merge into final
        final_valid = pd.concat([valid_X2_scaled, valid_X], axis=1)
        final_valid = round(final_valid,2)

        # prediction using f1_score
        model.fit(final_train, y_train)

        t = model.score(final_train, y_train)
        t = round(t,3)
        v = model.score(final_valid, valid[1])
        v = round(v,3)

        preds = model.predict(final_valid)
        f1 = f1_score(valid[1], preds)
        f1 = round(f1,3)

        f1_scores.append(f1)
        valid_accuracy.append(v)
        train_accuracy.append(t)

        now = now - ows

    return round(np.mean(train_accuracy),3), round(np.mean(valid_accuracy),3),
round(np.mean(f1_scores),3)
```

2. List of models

In [73]:
```
list_of_models = []

lf = LogisticRegression(solver = 'liblinear', random_state=42,
                        penalty = 'l2', C= 0.002335721469090121)
bgm = GradientBoostingClassifier(random_state=42,
                                 subsample= 0.38493564427131044, n_estimators= 2
96,
                                 min_samples_split= 6, min_samples_leaf= 36,
                                 max_depth= 10, learning_rate= 0.077997260168101
32)
import xgboost as xgb
xgb = xgb.XGBClassifier(objective='binary:logistic',
                    silent=True, nthread=1, random_state=42, n_jobs=-1,
                    subsample= 0.20823379771832098, n_estimators= 862,
                    min_child_weight= 8, max_depth= 5,
                    learning_rate= 0.017194260557609198, gamma= 0.2976351441631
313,
                    colsample_bytree= 0.7824279936868144)

list_of_models += [lf, bgm, xgb]
```

3. Accuracy and f1 scores

In [74]:
```
ws = 33
ows = 33
now = 609
scores_of_top_three_models = pd.DataFrame()

for model in list_of_models:
    train_accuracy, valid_accuracy, f1_scores = predicting_top_three_models(mod
el,
                                                                             tot
al_holdout_sets=5,
                                                                             now
=now, ws=ws, ows=ows)
    new_row = pd.DataFrame( [[model, train_accuracy, valid_accuracy, f1_scores
]] )
    scores_of_top_three_models = scores_of_top_three_models.append(new_row, ign
ore_index = True)
    print(model, 'completed')

scores_of_top_three_models = scores_of_top_three_models.rename(
   columns={0:'model', 1:'train_accuracy', 2:'valid_accuracy', 3:'f1_scores'})
```

```
LogisticRegression(C=0.002335721469090121, class_weight=None, dual=False,
                   fit_intercept=True, intercept_scaling=1, l1_ratio=None,
                   max_iter=100, multi_class='auto', n_jobs=None, penalty='l2',
                   random_state=42, solver='liblinear', tol=0.0001, verbose=0,
                   warm_start=False) completed
GradientBoostingClassifier(ccp_alpha=0.0, criterion='friedman_mse', init=None,
                           learning_rate=0.07799726016810132, loss='deviance',
                           max_depth=10, max_features=None, max_leaf_nodes=Non
e,
                           min_impurity_decrease=0.0, min_impurity_split=None,
                           min_samples_leaf=36, min_samples_split=6,
                           min_weight_fraction_leaf=0.0, n_estimators=296,
                           n_iter_no_change=None, presort='deprecated',
                           random_state=42, subsample=0.38493564427131044,
                           tol=0.0001, validation_fraction=0.1, verbose=0,
                           warm_start=False) completed
XGBClassifier(base_score=0.5, booster='gbtree', colsample_bylevel=1,
              colsample_bynode=1, colsample_bytree=0.7824279936868144,
              gamma=0.2976351441631313, learning_rate=0.017194260557609198,
              max_delta_step=0, max_depth=5, min_child_weight=8, missing=None,
              n_estimators=862, n_jobs=-1, nthread=1,
              objective='binary:logistic', random_state=42, reg_alpha=0,
              reg_lambda=1, scale_pos_weight=1, seed=None, silent=True,
              subsample=0.20823379771832098, verbosity=1) completed
```

In [75]:
```
scores_of_top_three_models
```

Out[75]:

	model	train_accuracy	valid_accuracy	f1_scores
0	LogisticRegression(C=0.002335721469090121, cla...	0.794	0.781	0.792
1	([DecisionTreeRegressor(ccp_alpha=0.0, criteri...	0.956	0.713	0.663
2	XGBClassifier(base_score=0.5, booster='gbtree'...	0.831	0.789	0.803

4. Checking tumbling window size of the final prediction model

In [204]:
```
import xgboost as xgb
xgb = xgb.XGBClassifier(objective='binary:logistic',
                        silent=True, nthread=1, random_state=42, n_jobs=-1,
                        subsample= 0.20823379771832098, n_estimators= 862,
                        min_child_weight= 8, max_depth= 5,
                        learning_rate= 0.017194260557609198, gamma= 0.2976351441631
313,
                        colsample_bytree= 0.7824279936868144)

tumbling_window_size = [7, 8, 10, 14, 21, 22, 28, 30, 33]
ows = 33
now = 609
check_tumbling_window_size = pd.DataFrame()

for ws in tumbling_window_size:
    train_accuracy, valid_accuracy, f1_scores = predicting_top_three_models(xgb
,
                                                                              tot
al_holdout_sets=3,
                                                                              now
=now, ws=ws, ows=ows)
    new_row = pd.DataFrame( [[xgb, ws, train_accuracy, valid_accuracy, f1_score
s]] )
    check_tumbling_window_size = check_tumbling_window_size.append(new_row, ign
ore_index = True)
    print(ws, 'completed')

check_tumbling_window_size = check_tumbling_window_size.rename(
   columns={0:'model', 1: 'tumbling_window_size', 2:'train_accuracy',
            3:'valid_accuracy', 4:'f1_scores'})
```

```
7 completed
8 completed
10 completed
14 completed
21 completed
22 completed
28 completed
30 completed
33 completed
```

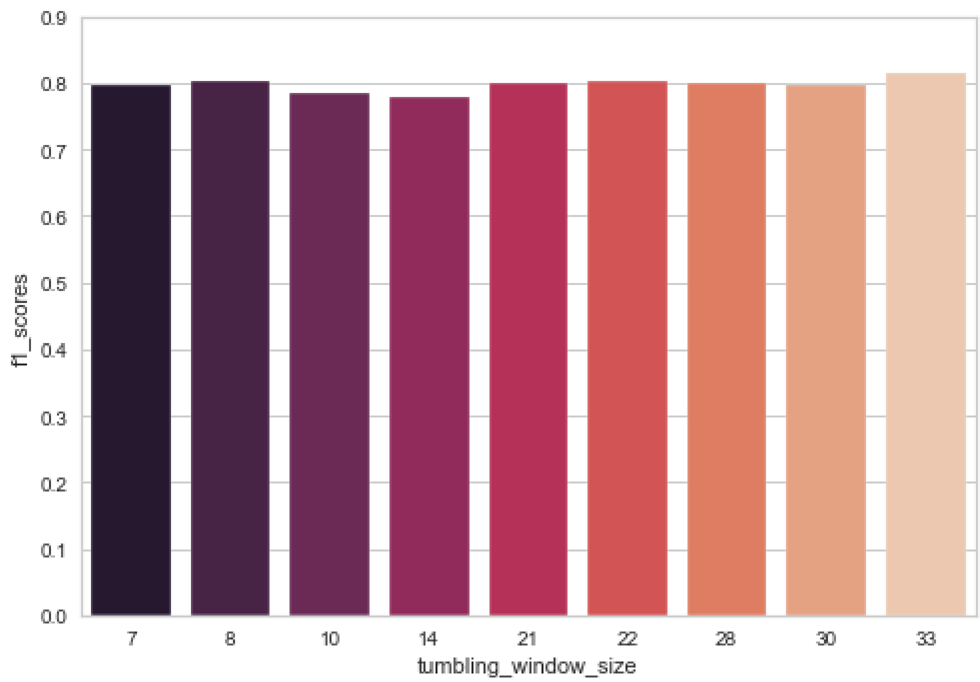
In [205]:
```
check_tumbling_window_size
```

Out[205]:

	model	tumbling_window_size	train_accuracy	valid_accuracy	f1_scores
0	XGBClassifier(base_score=0.5, booster='gbtree'...	7	0.830	0.741	0.798
1	XGBClassifier(base_score=0.5, booster='gbtree'...	8	0.830	0.750	0.802
2	XGBClassifier(base_score=0.5, booster='gbtree'...	10	0.827	0.720	0.785
3	XGBClassifier(base_score=0.5, booster='gbtree'...	14	0.829	0.718	0.778
4	XGBClassifier(base_score=0.5, booster='gbtree'...	21	0.835	0.753	0.799
5	XGBClassifier(base_score=0.5, booster='gbtree'...	22	0.832	0.772	0.803
6	XGBClassifier(base_score=0.5, booster='gbtree'...	28	0.835	0.766	0.800
7	XGBClassifier(base_score=0.5, booster='gbtree'...	30	0.829	0.765	0.796
8	XGBClassifier(base_score=0.5, booster='gbtree'...	33	0.835	0.798	0.816

In [218]:
```
sns.barplot(x="tumbling_window_size", y="f1_scores", data=check_tumbling_window
_size,
            palette="rocket")
plt.ylim(0, 0.9)
```

Out[218]: (0.0, 0.9)



XGBoost Classifier is the best prediction model with tunbling window size 33

E. Feature importance and selection

1. Data preparation for feature importance and selction

In [4]:
```python
ws = 33
ows = 33
now = 609
    # for each holdout set, compute f1 score

test = get_dataset_value(now-ows, ws, ows)
train = get_dataset_value(now-2*ows, ws, ows)

         # output feature changes to binary, 1: non- churn, 0: churn
test[1][test[1]>0] = 1 # non-chrun
train[1][train[1]>0] = 1 # non-chrun

        # Balancing unbalanced output feature in train data set using SMOTE
smote = SMOTE(random_state=42)
X_train, y_train = smote.fit_resample(train[0], train[1])

X_train = pd.DataFrame(X_train,
                      columns=['total_values','total_quantity','avg_between',
                              'f1','f2','f3','f4','f5','f6','f7','f8','f9','f1
0','f11'])
y_train = pd.DataFrame(y_train)

        # standardizing Temporal data in train set
train_X = pd.DataFrame()

for i in X_train.iloc[:,3:14].values:
        a = i - X_train.iloc[:,3:14].values.sum()
        b = a / np.std(X_train.iloc[:,3:14].values)

        new_row = pd.DataFrame( [[b]] )
        train_X = train_X.append(new_row, ignore_index = True)

train_X.columns = ['f']
train_X = pd.DataFrame(train_X.f.tolist(),
                            columns=['f1','f2','f3','f4','f5','f6','f7','f8',
                                    'f9','f10','f11'])

        # standardizing traditional data in train set
        # Step 1: log1p
train_X2 = X_train.drop(columns=['f1','f2','f3','f4','f5','f6','f7','f8','f9',
'f10','f11'])
train_X2_log = np.log1p(train_X2)
        # Step 2: StandardScaler
scaler = StandardScaler()
train_X2_scaled = scaler.fit_transform(train_X2_log)

        # transform into a dataframe
train_X2_scaled = pd.DataFrame(train_X2_scaled, index=train_X2_log.index,
                             columns=train_X2_log.columns)
final_train = pd.concat([train_X2_scaled, train_X], axis=1)
final_train = round(final_train,2)

        # # standardizing Temporal data in validation set
test_X = pd.DataFrame()

for i in test[0].iloc[:,3:14].values:
        a = i - test[0].iloc[:,3:14].values.sum()
        b = a / np.std(test[0].iloc[:,3:14].values)

        new_row = pd.DataFrame( [[b]] )
        test_X = test_X.append(new_row, ignore_index = True)
```

```
test_X.columns = ['f']
test_X = pd.DataFrame(test_X.f.tolist(),
                            columns=['f1','f2','f3','f4','f5','f6','f7','f8',
                                          'f9','f10','f11'])

        # standardizing traditional data in validation set
        # Step 1: log1p
test_X2 = test[0].drop(columns=['f1','f2','f3','f4','f5','f6','f7','f8','f9','f
10','f11'])
test_X2_log = np.log1p(test_X2)
        # Step 2: StandardScaler
scaler = StandardScaler()
test_X2_scaled = scaler.fit_transform(test_X2_log)

        # transform into a dataframe
test_X2_scaled = pd.DataFrame(test_X2_scaled, index=test_X2_log.index,
                            columns=test_X2_log.columns)
        # Merge into final
final_test = pd.concat([test_X2_scaled, test_X], axis=1)
final_test = round(final_test,2)
```

2. Function of making table for feature imporance

In [5]:

```
def print_variable_importances( feature_names, dict_in, show_top = 14 ):
  if show_top is None:
    show_top = len(feature_names)

  to_print_titles = []
  to_print_scores = []

  for k, v in dict_in.items():
    feature_names_plus_scores = sorted( zip(v, feature_names) )
    feature_names_plus_scores.reverse()
    to_print_titles.append(k)
    to_print_scores.append(feature_names_plus_scores)

  line_parts = []
  for j in range(len(to_print_titles)):
    line_parts.append('{:<24}'.format(to_print_titles[j]))

  print('Rank | ' + ' | '.join( ['{:<24}'.format(x) for x in to_print_titles] )
)

  print('---- + ' + ' + '.join( [ '-'*24 ]*len(to_print_titles) ) )

  for i in range(show_top):
    line_parts = []
    for j in range(len(to_print_titles)):
      line_parts.append(  '{:<16}: {:.4f}'.format(to_print_scores[j][i][1], to_
print_scores[j][i][0]) )
    print( '{:<4} | '.format(str(i)) + ' | '.join(line_parts) )
```

3. Function of printing accuracies and f1 score

(before and after of feature selection)

```
In [6]: def feature_selection(deleted_train, deleted_test):
            print("'Original dataset'")
            xgb.fit(final_train, y_train)
            print("train set accuracy : {:.3f}".format(xgb.score(final_train, y_train
        )))
            print("test set accuracy : {:.3f}".format(xgb.score(final_test, test[1])))
            y_pred = xgb.predict(final_test)
            f1 = f1_score(test[1], y_pred)
            print('Test set f1 score for best params:', round(f1,3))
            print('=========================================')
            print("'Deleted the least important feature'")
            xgb.fit(deleted_train, y_train)
            print("train set accuracy : {:.3f}".format(xgb.score(deleted_train, y_train
        )))
            print("test set accuracy : {:.3f}".format(xgb.score(deleted_test, test[1
        ])))
            y_pred = xgb.predict(deleted_test)
            f1 = f1_score(test[1], y_pred)
            print('Test set f1 score for best params:', round(f1,3))
```

4. Comparing result of different feature importance methods

1) Univariate variable importance

```
In [150]: us = GenericUnivariateSelect( score_func=mutual_info_classif, mode='k_best', pa
          ram=14 )

          us.fit(final_train,y_train)

          feature_importance_scores = {}
          feature_importance_scores['Filter'] = us.scores_

          print_variable_importances( final_train.columns, feature_importance_scores, 14
          )
```

```
Rank | Filter
---- + ------------------------
0    | avg_between      : 0.2902
1    | total_values     : 0.2485
2    | total_quantity   : 0.2298
3    | f2               : 0.1537
4    | f1               : 0.1379
5    | f6               : 0.1373
6    | f3               : 0.1373
7    | f4               : 0.1363
8    | f5               : 0.1237
9    | f8               : 0.1108
10   | f7               : 0.1103
11   | f11              : 0.0919
12   | f9               : 0.0863
13   | f10              : 0.0805
```

Filter based methods (Univariate feature ranking and selection) : compare each feature to the target variable, to see whether there is any statistically significant relationship between them. It is also called analysis of variance (ANOVA)

2) XGBoost embeded feature importance

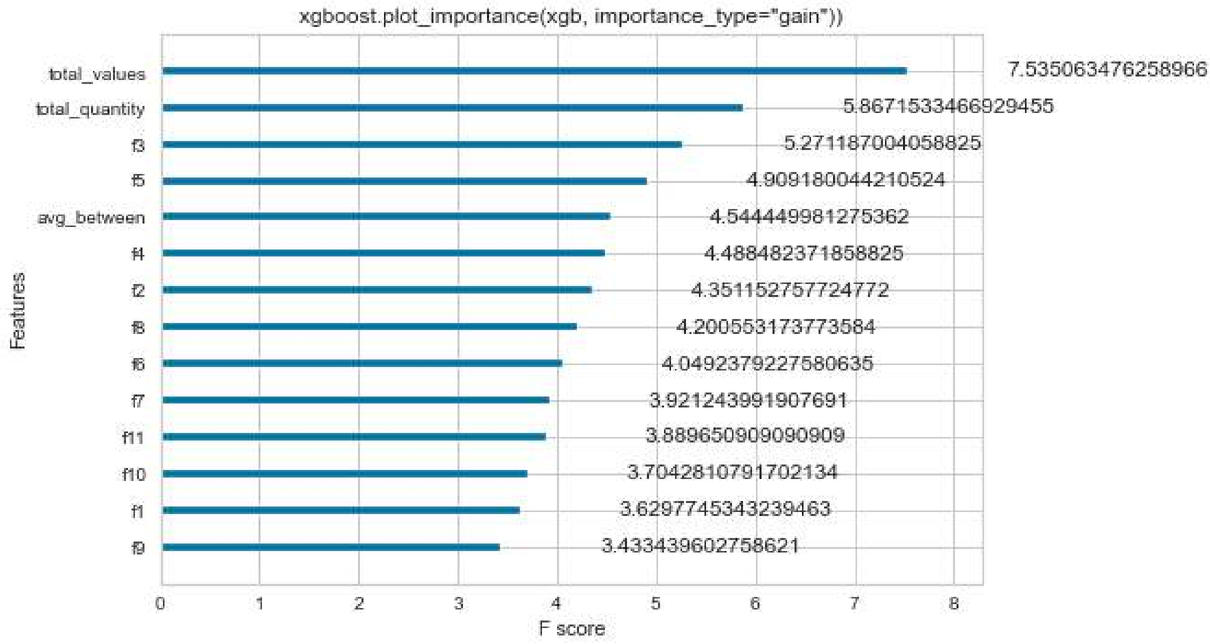
In [151]:
```
# final model of XGB
import xgboost as xgb
xgb = xgb.XGBClassifier(objective='binary:logistic',
                        silent=True, nthread=1, random_state=42, n_jobs=-1,
                        subsample= 0.20823379771832098, n_estimators= 862,
                        min_child_weight= 8, max_depth= 5,
                        learning_rate= 0.017194260557609198, gamma= 0.2976351441631
313,
                        colsample_bytree= 0.7824279936868144)
xgb.fit(final_train, y_train)
```

Out[151]:
```
XGBClassifier(base_score=0.5, booster='gbtree', colsample_bylevel=1,
              colsample_bynode=1, colsample_bytree=0.7824279936868144,
              gamma=0.2976351441631313, learning_rate=0.017194260557609198,
              max_delta_step=0, max_depth=5, min_child_weight=8, missing=None,
              n_estimators=862, n_jobs=-1, nthread=1,
              objective='binary:logistic', random_state=42, reg_alpha=0,
              reg_lambda=1, scale_pos_weight=1, seed=None, silent=True,
              subsample=0.20823379771832098, verbosity=1)
```

In [152]:
```
feature_importance_scores = {}
feature_importance_scores['Embeded XGB, gain'] = xgb.feature_importances_
print_variable_importances( final_train.columns, feature_importance_scores)
```

```
Rank | Embeded XGB, gain
---- + ------------------------
0    | total_values    : 0.1181
1    | total_quantity  : 0.0920
2    | f3              : 0.0826
3    | f5              : 0.0770
4    | avg_between     : 0.0712
5    | f4              : 0.0704
6    | f2              : 0.0682
7    | f8              : 0.0658
8    | f6              : 0.0635
9    | f7              : 0.0615
10   | f11             : 0.0610
11   | f10             : 0.0581
12   | f1              : 0.0569
13   | f9              : 0.0538
```

In [153]:
```
plot_importance(xgb, importance_type="gain")
plt.title('xgboost.plot_importance(xgb, importance_type="gain"))')
plt.show()
```



Gain implies the relative contribution of the corresponding feature to the model calculated by taking each feature's contribution for each tree in the model. A higher value of this implies it is more important for generating a prediction.

3) Permutation Importance

In [154]:
```
xgb_perm = PermutationImportance(xgb, cv=3)
xgb_perm.fit(final_train.values, y_train.values)

feature_importance_scores['Perm cv XGB'] = xgb_perm.feature_importances_
print_variable_importances( final_train.columns, feature_importance_scores )
```

```
Rank | Embeded XGB, gain          | Perm cv XGB
---- + ------------------------- + ------------------------
0    | total_values     : 0.1181 | total_values     : 0.0314
1    | total_quantity   : 0.0920 | avg_between      : 0.0290
2    | f3               : 0.0826 | total_quantity   : 0.0142
3    | f5               : 0.0770 | f3               : 0.0025
4    | avg_between      : 0.0712 | f10              : -0.0002
5    | f4               : 0.0704 | f5               : -0.0009
6    | f2               : 0.0682 | f6               : -0.0023
7    | f8               : 0.0658 | f4               : -0.0023
8    | f6               : 0.0635 | f9               : -0.0025
9    | f7               : 0.0615 | f1               : -0.0032
10   | f11              : 0.0610 | f8               : -0.0032
11   | f10              : 0.0581 | f11              : -0.0034
12   | f1               : 0.0569 | f2               : -0.0040
13   | f9               : 0.0538 | f7               : -0.0055
```

In [23]:
```
sum(xgb_perm.feature_importances_)
```

Out[23]:
```
0.04284307815953391
```

“permutation importance” or “Mean Decrease Accuracy (MDA)” feature importance can be measured by looking at how much the score decreases when a feature is not available. => Delecting feature 'f7' decreased f1 score

4) Recursive feature elimination (RFE)

In [155]:
```
rfe_xgb_embed = RFE(xgb, n_features_to_select = 14, step=1)
rfe_xgb_embed.fit(final_train, y_train)
```

Out[155]:
```
RFE(estimator=XGBClassifier(base_score=0.5, booster='gbtree',
                            colsample_bylevel=1, colsample_bynode=1,
                            colsample_bytree=0.7824279936868144,
                            gamma=0.2976351441631313,
                            learning_rate=0.017194260557609198,
                            max_delta_step=0, max_depth=5, min_child_weight=8,
                            missing=None, n_estimators=862, n_jobs=-1,
                            nthread=1, objective='binary:logistic',
                            random_state=42, reg_alpha=0, reg_lambda=1,
                            scale_pos_weight=1, seed=None, silent=True,
                            subsample=0.20823379771832098, verbosity=1),
    n_features_to_select=14, step=1, verbose=0)
```

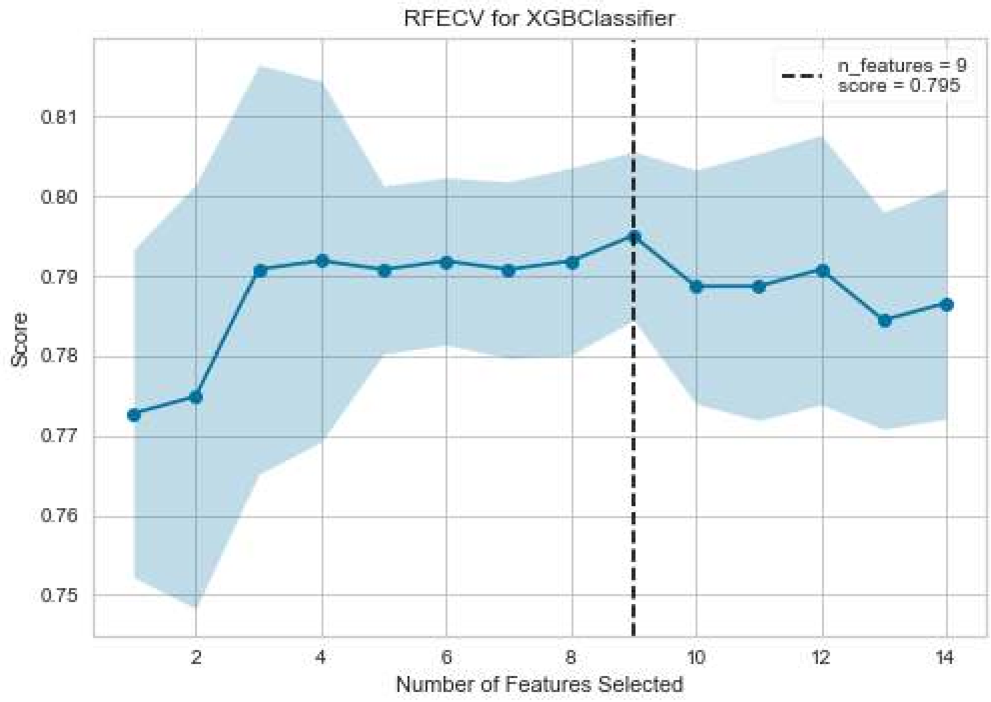
In [156]:
```
#print(rfe_xgb_embed.estimator_.feature_importances_)
rfe_xgb_embed_fi = np.asarray(rfe_xgb_embed.support_, dtype=np.float)
rfe_xgb_embed_fi[rfe_xgb_embed.support_] = rfe_xgb_embed.estimator_.feature_imp
ortances_
feature_importance_scores['RFE Embed XGB'] = rfe_xgb_embed_fi
print_variable_importances( final_train.columns, feature_importance_scores )
```

```
Rank | Embeded XGB, gain        | Perm cv XGB              | RFE Embed XGB
---- + ------------------------ + ------------------------ + ------------------
------
0    | total_values    : 0.1181 | total_values    : 0.0314 | total_values    :
0.1181
1    | total_quantity  : 0.0920 | avg_between     : 0.0290 | total_quantity  :
0.0920
2    | f3              : 0.0826 | total_quantity  : 0.0142 | f3              :
0.0826
3    | f5              : 0.0770 | f3              : 0.0025 | f5              :
0.0770
4    | avg_between     : 0.0712 | f10             : -0.0002 | avg_between     :
0.0712
5    | f4              : 0.0704 | f5              : -0.0009 | f4              :
0.0704
6    | f2              : 0.0682 | f6              : -0.0023 | f2              :
0.0682
7    | f8              : 0.0658 | f4              : -0.0023 | f8              :
0.0658
8    | f6              : 0.0635 | f9              : -0.0025 | f6              :
0.0635
9    | f7              : 0.0615 | f1              : -0.0032 | f7              :
0.0615
10   | f11             : 0.0610 | f8              : -0.0032 | f11             :
0.0610
11   | f10             : 0.0581 | f11             : -0.0034 | f10             :
0.0581
12   | f1              : 0.0569 | f2              : -0.0040 | f1              :
0.0569
13   | f9              : 0.0538 | f7              : -0.0055 | f9              :
0.0538
```

Recursive feature elimination (RFE) is a feature selection method that fits a model and removes the weakest feature (or features) until the specified number of features is reached. => Same result with embeded XGB feature importance using gain

5) Visualization for RFE

In [149]:
```
import xgboost as xgb
xgb = xgb.XGBClassifier(objective='binary:logistic',
                        silent=True, nthread=1, random_state=42, n_jobs=-1,
                        subsample= 0.20823379771832098, n_estimators= 862,
                        min_child_weight= 8, max_depth= 5,
                        learning_rate= 0.017194260557609198, gamma= 0.2976351441631
313,
                        colsample_bytree= 0.7824279936868144)
visualizer = RFECV(xgb)
visualizer.fit(final_train, y_train)
visualizer.show()
```



Out[149]: <matplotlib.axes._subplots.AxesSubplot at 0x26fd2241748>

6) Recursive feature elimination (RFE) with Permutation Importance

In [157]:
```
rfe_xgb_perm = RFE(PermutationImportance(xgb, cv=3), n_features_to_select = 9,
step = 1)
rfe_xgb_perm.fit(final_train, y_train)

rfe_xgb_perm_fi = np.asarray(rfe_xgb_perm.support_, dtype=np.float)
rfe_xgb_perm_fi[rfe_xgb_perm.support_] = rfe_xgb_perm.estimator_.feature_import
ances_

feature_importance_scores['RFE Perm CV XGB'] = rfe_xgb_perm_fi
print_variable_importances( final_train.columns, feature_importance_scores )
```

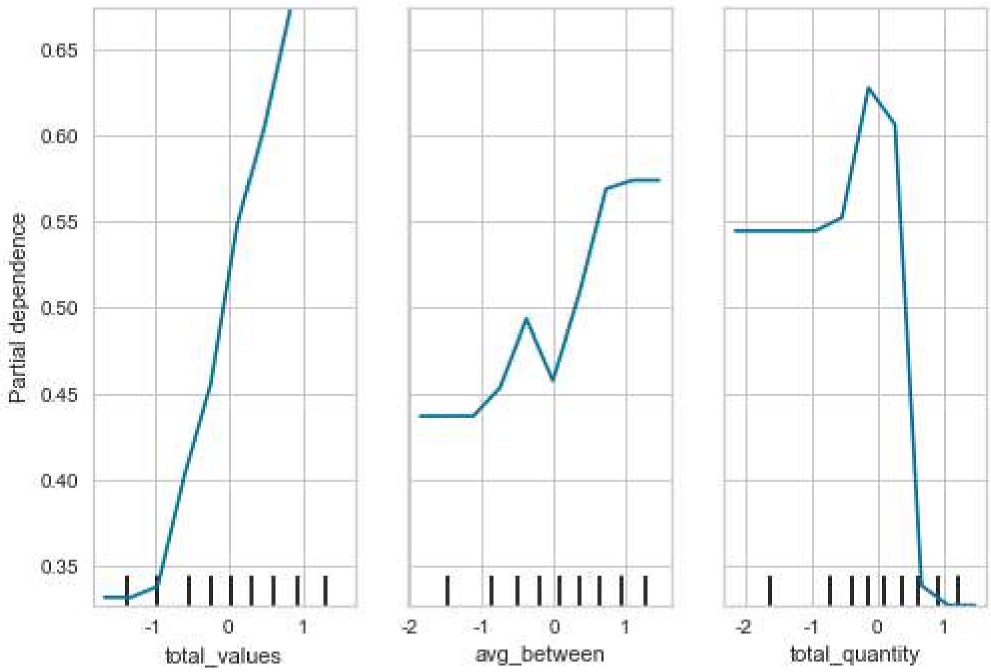
```
Rank | Embeded XGB, gain         | Perm cv XGB               | RFE Embed XGB
| RFE Perm CV XGB
---- + ------------------------ + ------------------------ + ------------------
------ + ------------------------
0    | total_values    : 0.1181 | total_values    : 0.0314 | total_values    :
0.1181 | total_values    : 0.0357
1    | total_quantity  : 0.0920 | avg_between     : 0.0290 | total_quantity  :
0.0920 | avg_between     : 0.0338
2    | f3              : 0.0826 | total_quantity  : 0.0142 | f3              :
0.0826 | total_quantity  : 0.0095
3    | f5              : 0.0770 | f3              : 0.0025 | f5              :
0.0770 | f2              : 0.0013
4    | avg_between     : 0.0712 | f10             : -0.0002 | avg_between     :
0.0712 | f3              : 0.0004
5    | f4              : 0.0704 | f5              : -0.0009 | f4              :
0.0704 | f6              : 0.0004
6    | f2              : 0.0682 | f6              : -0.0023 | f2              :
0.0682 | f9              : 0.0000
7    | f8              : 0.0658 | f4              : -0.0023 | f8              :
0.0658 | f7              : 0.0000
8    | f6              : 0.0635 | f9              : -0.0025 | f6              :
0.0635 | f4              : 0.0000
9    | f7              : 0.0615 | f1              : -0.0032 | f7              :
0.0615 | f11             : 0.0000
10   | f11             : 0.0610 | f8              : -0.0032 | f11             :
0.0610 | f10             : 0.0000
11   | f10             : 0.0581 | f11             : -0.0034 | f10             :
0.0581 | f5              : -0.0017
12   | f1              : 0.0569 | f2              : -0.0040 | f1              :
0.0569 | f1              : -0.0025
13   | f9              : 0.0538 | f7              : -0.0055 | f9              :
0.0538 | f8              : -0.0053
```

In [158]:
```
# Deleting negative values from RFE Perm CV XGB feature importance
train_copy = final_train.copy()
test_copy = final_test.copy()
train_del = train_copy.drop(columns=['f8','f1','f5','f10','f11'])
test_del = test_copy.drop(columns=['f8','f1','f5','f10','f11'])
import xgboost as xgb
xgb = xgb.XGBClassifier(objective='binary:logistic',silent=True, nthread=1, random_state=42, n_jobs=-1,subsample= 0.20823379771832098, n_estimators= 862,  min_child_weight= 8, max_depth= 5, learning_rate= 0.017194260557609198, gamma= 0.2976351441631313, colsample_bytree= 0.7824279936868144)
feature_selection(train_del, test_del)
```

```
'Original dataset'
train set accuracy : 0.821
test set accuracy : 0.744
Test set f1 score for best params: 0.776
=========================================
'Deleted the least important feature'
train set accuracy : 0.826
test set accuracy : 0.738
Test set f1 score for best params: 0.764
```

7) Plot partial dependence of the two most important feature 'total valeus' and 'tavg between'

```
In [11]: import xgboost as xgb
xgb = xgb.XGBClassifier(objective='binary:logistic',
                        silent=True, nthread=1, random_state=42, n_jobs=-1,
                        subsample= 0.20823379771832098, n_estimators= 862,
                        min_child_weight= 8, max_depth= 5,
                        learning_rate= 0.017194260557609198, gamma= 0.2976351441631
313,
                        colsample_bytree= 0.7824279936868144)
xgb.fit(final_train, y_train)

my_plots = plot_partial_dependence(xgb,
                                   features=[0,1,2],
                                   X=final_train,
                                   feature_names=['total_values', 'avg_between'
,
                                                  'total_quantity'], # labels on
graphs
                                   grid_resolution=10)
```



Show how a model's predictions depend on a single input. The plot below shows the relationship (according the model that we trained) between churn or non-churn (target) and total values, total quantity and average between vistis. 1: non-churn / 0: churn

F. Final XGBoost model with the whole data set

1. Prediction without holdout sets with deleted feature 'f6', 'f7', 'f11' in the whole data set

In [12]:
```
ws = 33
ows = 33
now = 609
    # for each holdout set, compute f1 score

test = get_dataset_value(now-ows, ws, ows)
train = get_dataset_value(now-2*ows, ws, ows)

         # output feature changes to binary, 1: non- churn, 0: churn
test[1][test[1]>0] = 1 # non-chrun
train[1][train[1]>0] = 1 # non-chrun

        # Balancing unbalanced output feature in train data set using SMOTE
smote = SMOTE(random_state=42)
X_train, y_train = smote.fit_resample(train[0], train[1])

X_train = pd.DataFrame(X_train,
                      columns=['total_values','total_quantity','avg_between',
                              'f1','f2','f3','f4','f5','f6','f7','f8','f9','f1
0','f11'])
y_train = pd.DataFrame(y_train)

        # standardizing Temporal data in train set
train_X = pd.DataFrame()

for i in X_train.iloc[:,3:14].values:
        a = i - X_train.iloc[:,3:14].values.sum()
        b = a / np.std(X_train.iloc[:,3:14].values)

        new_row = pd.DataFrame( [[b]] )
        train_X = train_X.append(new_row, ignore_index = True)

train_X.columns = ['f']
train_X = pd.DataFrame(train_X.f.tolist(),
                            columns=['f1','f2','f3','f4','f5','f6','f7','f8',
                                         'f9','f10','f11'])

        # standardizing traditional data in train set
        # Step 1: log1p
train_X2 = X_train.drop(columns=['f1','f2','f3','f4','f5','f6','f7','f8','f9',
'f10','f11'])
train_X2_log = np.log1p(train_X2)
        # Step 2: StandardScaler
scaler = StandardScaler()
train_X2_scaled = scaler.fit_transform(train_X2_log)

        # transform into a dataframe
train_X2_scaled = pd.DataFrame(train_X2_scaled, index=train_X2_log.index,
                              columns=train_X2_log.columns)
final_train = pd.concat([train_X2_scaled, train_X], axis=1)
final_train = round(final_train,2)

        # # standardizing Temporal data in validation set
test_X = pd.DataFrame()

for i in test[0].iloc[:,3:14].values:
        a = i - test[0].iloc[:,3:14].values.sum()
        b = a / np.std(test[0].iloc[:,3:14].values)

        new_row = pd.DataFrame( [[b]] )
        test_X = test_X.append(new_row, ignore_index = True)
```

```python
test_X.columns = ['f']
test_X = pd.DataFrame(test_X.f.tolist(),
                            columns=['f1','f2','f3','f4','f5','f6','f7','f8',
                                    'f9','f10','f11'])

        # standardizing traditional data in validation set
        # Step 1: log1p
test_X2 = test[0].drop(columns=['f1','f2','f3','f4','f5','f6','f7','f8','f9','f
10','f11'])
test_X2_log = np.log1p(test_X2)
        # Step 2: StandardScaler
scaler = StandardScaler()
test_X2_scaled = scaler.fit_transform(test_X2_log)

        # transform into a dataframe
test_X2_scaled = pd.DataFrame(test_X2_scaled, index=test_X2_log.index,
                             columns=test_X2_log.columns)
        # Merge into final
final_test = pd.concat([test_X2_scaled, test_X], axis=1)
final_test = round(final_test,2)
```

In [13]:
```python
y_test = test[1].copy()
```

In [159]:
```python
# Deleting
train = final_train.copy()
test = final_test.copy()
train = train.drop(columns=['f8','f1','f5','f10','f11'])
test = test.drop(columns=['f8','f1','f5','f10','f11'])
import xgboost as xgb
xgb = xgb.XGBClassifier(objective='binary:logistic',silent=True, nthread=1, ran
dom_state=42, n_jobs=-1,subsample= 0.20823379771832098, n_estimators= 862,  min
_child_weight= 8, max_depth= 5, learning_rate= 0.017194260557609198, gamma= 0.2
976351441631313, colsample_bytree= 0.7824279936868144)

xgb.fit(train, y_train)
print("train set accuracy : {:.3f}".format(xgb.score(train, y_train)))
print("test set accuracy : {:.3f}".format(xgb.score(test, y_test)))
y_pred = xgb.predict(test)
f1 = f1_score(y_test, y_pred)
print('Test set f1 score for best params:', round(f1,3))
```

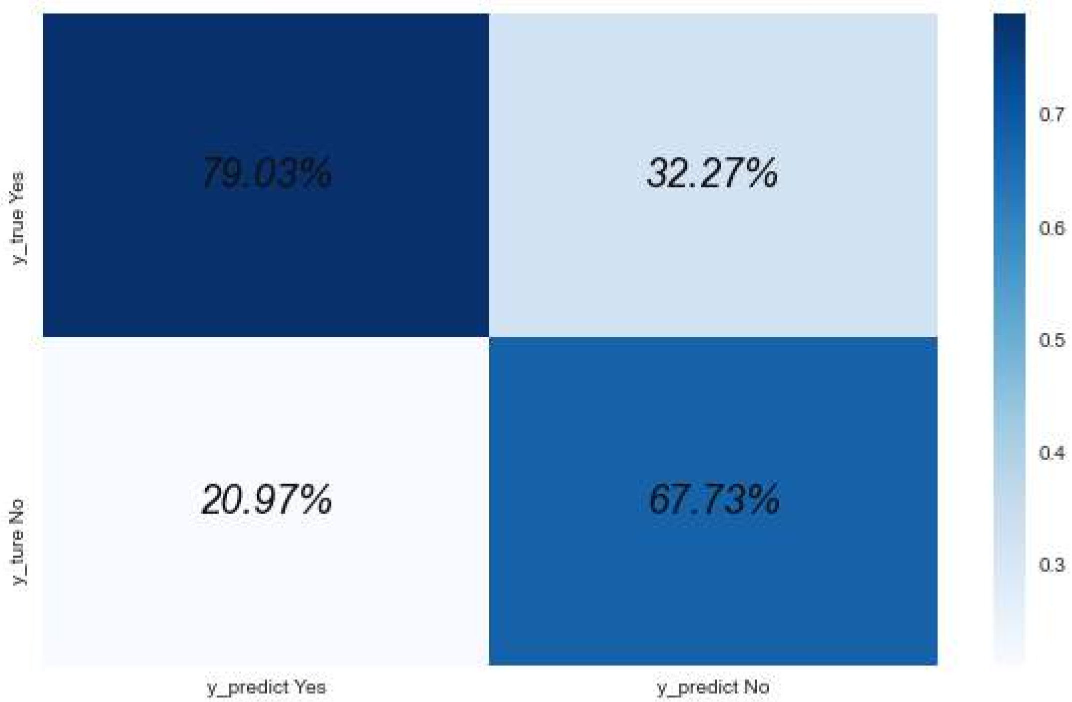
```
train set accuracy : 0.826
test set accuracy : 0.738
Test set f1 score for best params: 0.764
```

In [160]:
```python
print('Confusion Matrix')
confusion = pd.DataFrame(confusion_matrix(y_test, y_pred, labels=[1,0]),
                        index=['y_true Yes','y_ture No'],
                        columns=['y_predict Yes','y_predict No'])
print(confusion)
```

```
Confusion Matrix
            y_predict Yes  y_predict No
y_true Yes            373           131
y_ture No              99           275
```

In [161]:
```
plt.figure(figsize=(10,6))
xticklables = ['y_predict Yes','y_predict No']
yticklables = ['y_true Yes','y_ture No']

annot_kws={'fontsize':20,
           'fontstyle':'italic',
           'color':"k",
           'alpha':1,
           'verticalalignment':'center'}

sns.heatmap(confusion/np.sum(confusion), annot=True,
            fmt='.2%', cmap='Blues',
           xticklabels = xticklables,
           yticklabels = yticklables,
           annot_kws = annot_kws)
```

Out[161]: <matplotlib.axes._subplots.AxesSubplot at 0x26fd3aa6988>



G. Pen Portraits of Churners vs Non-churners

1. Data preparation for pen portraits

In [163]:

```
ws = 33
ows = 33
now = 609
    # for each holdout set, compute f1 score

test = get_dataset_value(now-ows, ws, ows)
train = get_dataset_value(now-2*ows, ws, ows)

         # output feature changes to binary, 1: non- churn, 0: churn
test[1][test[1]>0] = 1 # non-chrun
train[1][train[1]>0] = 1 # non-chrun

        # Balancing unbalanced output feature in train data set using SMOTE
smote = SMOTE(random_state=42)
X_train, y_train = smote.fit_resample(train[0], train[1])

X_train = pd.DataFrame(X_train,
                      columns=['total_values','total_quantity','avg_between',
                              'f1','f2','f3','f4','f5','f6','f7','f8','f9','f1
0','f11'])
y_train = pd.DataFrame(y_train)

        # standardizing Temporal data in train set
train_X = pd.DataFrame()

for i in X_train.iloc[:,3:14].values:
        a = i - X_train.iloc[:,3:14].values.sum()
        b = a / np.std(X_train.iloc[:,3:14].values)

        new_row = pd.DataFrame( [[b]] )
        train_X = train_X.append(new_row, ignore_index = True)

train_X.columns = ['f']
train_X = pd.DataFrame(train_X.f.tolist(),
                            columns=['f1','f2','f3','f4','f5','f6','f7','f8',
                                         'f9','f10','f11'])

        # standardizing traditional data in train set
        # Step 1: log1p
train_X2 = X_train.drop(columns=['f1','f2','f3','f4','f5','f6','f7','f8','f9',
'f10','f11'])
train_X2_log = np.log1p(train_X2)
        # Step 2: StandardScaler
scaler = StandardScaler()
train_X2_scaled = scaler.fit_transform(train_X2_log)

        # transform into a dataframe
train_X2_scaled = pd.DataFrame(train_X2_scaled, index=train_X2_log.index,
                             columns=train_X2_log.columns)
final_train = pd.concat([train_X2_scaled, train_X], axis=1)
final_train = round(final_train,2)

        # # standardizing Temporal data in validation set
test_X = pd.DataFrame()

for i in test[0].iloc[:,3:14].values:
        a = i - test[0].iloc[:,3:14].values.sum()
        b = a / np.std(test[0].iloc[:,3:14].values)

        new_row = pd.DataFrame( [[b]] )
        test_X = test_X.append(new_row, ignore_index = True)
```

```
test_X.columns = ['f']
test_X = pd.DataFrame(test_X.f.tolist(),
                            columns=['f1','f2','f3','f4','f5','f6','f7','f8',
                                          'f9','f10','f11'])

        # standardizing traditional data in validation set
        # Step 1: log1p
test_X2 = test[0].drop(columns=['f1','f2','f3','f4','f5','f6','f7','f8','f9','f
10','f11'])
test_X2_log = np.log1p(test_X2)
        # Step 2: StandardScaler
scaler = StandardScaler()
test_X2_scaled = scaler.fit_transform(test_X2_log)

        # transform into a dataframe
test_X2_scaled = pd.DataFrame(test_X2_scaled, index=test_X2_log.index,
                             columns=test_X2_log.columns)
        # Merge into final
final_test = pd.concat([test_X2_scaled, test_X], axis=1)
final_test = round(final_test,2)
```

2. Prediction using optimized XGBoost classifier

In [164]:
```
import xgboost as xgb
xgb = xgb.XGBClassifier(objective='binary:logistic',
                    silent=True, nthread=1, random_state=42, n_jobs=-1,
                    subsample= 0.20823379771832098, n_estimators= 862,
                    min_child_weight= 8, max_depth= 5,
                    learning_rate= 0.017194260557609198, gamma= 0.2976351441631
313,
                    colsample_bytree= 0.7824279936868144)
# Deleting feature 'f8','f1','f5','f10','f11'
y_test = test[1].copy()
X_train = final_train.copy()
X_test = final_test.copy()
X_train = X_train.drop(columns=['f8','f1','f5','f10','f11'])
X_test = X_test.drop(columns=['f8','f1','f5','f10','f11'])

xgb.fit(X_train, y_train)
y_pred = xgb.predict(X_test)
```

In [165]:
```
X_train.describe()
```

Out[165]:

	total_values	total_quantity	avg_between	f2	f3	f4	
count	946.000000	946.000000	946.000000	946.000000	946.000000	946.000000	946.0(
mean	0.000042	0.000032	0.000148	-5159.114799	-5159.194905	-5159.100402	-5159.1:
std	1.000456	1.000544	1.000560	1.004182	0.887351	1.030459	1.0(
min	-2.520000	-2.240000	-2.140000	-5159.630000	-5159.630000	-5159.630000	-5159.6:
25%	-0.720000	-0.650000	-0.570000	-5159.630000	-5159.630000	-5159.630000	-5159.6:
50%	0.020000	0.080000	0.090000	-5159.550000	-5159.580000	-5159.540000	-5159.5
75%	0.750000	0.780000	0.725000	-5159.020000	-5159.150000	-5159.060000	-5159.1(
max	2.490000	2.350000	2.070000	-5150.230000	-5151.720000	-5150.010000	-5146.2(

3. Finding churners and non-churners

1) Table for churners and non-churners

In [166]:
```
# Predicting churners using embeded probability in XGBoost
X_train['proba'] = xgb.predict_proba(X_train[X_train.columns])[:,1]

# Change label, 1 as non-chuners, 0 as churners
X_train.loc[ (X_train.proba >= 0.5), 'proba'] = 1  # not churn
X_train.loc[ (X_train.proba < 0.5), 'proba'] = 0  # churn

# Check the numbers of churners and non-churners
X_train['proba'].value_counts()
```

Out[166]:
```
0.0    486
1.0    460
Name: proba, dtype: int64
```

In [167]:
```
result = X_train['proba'].value_counts()
t = Texttable()
t.add_rows( [ ['Customer','Number'], ['Churn', result[0]], ['Non-churn',result[
1]]])
print(t.draw())
```

```
+-----------+--------+
| Customer  | Number |
+===========+========+
| Churn     | 486    |
+-----------+--------+
| Non-churn | 460    |
+-----------+--------+
```

In [169]:
```
data_proba = X_train.copy()
data_proba.head()
```

Out[169]:

	total_values	total_quantity	avg_between	f2	f3	f4	f6	f7	
0	-0.14	0.05	0.64	-5159.45	-5159.45	-5159.06	-5159.51	-5159.44	-5159
1	-1.13	-1.15	1.22	-5159.63	-5159.63	-5159.63	-5159.55	-5159.63	-5159
2	-1.78	-1.85	-2.14	-5159.63	-5159.63	-5159.63	-5159.63	-5159.63	-5159
3	-0.07	-0.21	0.60	-5159.63	-5159.63	-5159.26	-5159.51	-5159.53	-5159
4	-0.69	-0.25	0.94	-5159.63	-5159.63	-5159.63	-5159.63	-5159.45	-5159

In [170]:
```python
# Adding number columns to match with original value
X_train = X_train.copy()
X_train.insert(loc=0, column='number', value=np.arange(len(X_train)))
X_train.head()
```

Out[170]:

	number	total_values	total_quantity	avg_between	f2	f3	f4	f6	
0	0	-0.14	0.05	0.64	-5159.45	-5159.45	-5159.06	-5159.51	-5159.
1	1	-1.13	-1.15	1.22	-5159.63	-5159.63	-5159.63	-5159.55	-5159.
2	2	-1.78	-1.85	-2.14	-5159.63	-5159.63	-5159.63	-5159.63	-5159.
3	3	-0.07	-0.21	0.60	-5159.63	-5159.63	-5159.26	-5159.51	-5159.
4	4	-0.69	-0.25	0.94	-5159.63	-5159.63	-5159.63	-5159.63	-5159.

In [171]:
```python
# Making a new table of non-chuerns
non_churn = X_train['proba'] == 1
non_churn = X_train[non_churn]
# Making a new table of chuners
churn = X_train['proba'] == 0
churn = X_train[churn]
```

In [172]:
```python
non_churn.head()
```

Out[172]:

	number	total_values	total_quantity	avg_between	f2	f3	f4	f6	
5	5	1.06	1.22	-0.92	-5158.63	-5158.84	-5158.82	-5158.51	-515
12	12	-0.49	-0.87	0.01	-5159.36	-5159.52	-5159.56	-5159.63	-515
13	13	-0.41	-0.20	0.05	-5159.42	-5159.53	-5159.50	-5159.63	-515
17	17	0.46	0.67	-0.26	-5159.41	-5159.31	-5159.42	-5159.29	-515
23	23	1.70	1.70	-0.57	-5156.63	-5157.46	-5156.57	-5155.99	-515

In [173]:
```python
churn.head()
```

Out[173]:

	number	total_values	total_quantity	avg_between	f2	f3	f4	f6	
0	0	-0.14	0.05	0.64	-5159.45	-5159.45	-5159.06	-5159.51	-5159.
1	1	-1.13	-1.15	1.22	-5159.63	-5159.63	-5159.63	-5159.55	-5159.
2	2	-1.78	-1.85	-2.14	-5159.63	-5159.63	-5159.63	-5159.63	-5159.
3	3	-0.07	-0.21	0.60	-5159.63	-5159.63	-5159.26	-5159.51	-5159.
4	4	-0.69	-0.25	0.94	-5159.63	-5159.63	-5159.63	-5159.63	-5159.

2) Importing a new dataset contains original value of each customers

In [174]:
```
train = get_dataset_value(now-2*ows, ws, ows)
train[1][train[1]>0] = 1 # non-chrun
smote = SMOTE(random_state=42)
X_train, y_train = smote.fit_resample(train[0], train[1])
X_train.describe()

# Deleting feature 'f6','f7','f11'
X_train = X_train.drop(columns=['f6','f7','f11'])
X_train.head()
```

Out[174]:

	total_values	total_quantity	avg_between	f1	f2	f3	f4	f5	f8	f9	f10
0	139.53	117	38	5.50	11.37	11.41	36.91	0.0	0.00	0.00	0.0
1	28.55	13	83	5.00	0.00	0.00	0.00	0.0	2.19	0.00	0.0
2	9.67	3	0	9.67	0.00	0.00	0.00	0.0	0.00	0.00	0.0
3	157.71	73	36	5.32	0.00	0.00	23.83	0.0	0.00	25.53	0.0
4	58.50	69	57	35.24	0.00	0.00	0.00	0.0	0.00	0.00	0.0

In [175]:
```
# Adding number columns to match with transformed churn and non-churn table
ori_train = X_train.copy()
ori_train.insert(loc=0, column='number', value=np.arange(len(ori_train)))
ori_train.head(3)
```

Out[175]:

	number	total_values	total_quantity	avg_between	f1	f2	f3	f4	f5	f8	f9	f10
0	0	139.53	117	38	5.50	11.37	11.41	36.91	0.0	0.00	0.0	0.0
1	1	28.55	13	83	5.00	0.00	0.00	0.00	0.0	2.19	0.0	0.0
2	2	9.67	3	0	9.67	0.00	0.00	0.00	0.0	0.00	0.0	0.0

3) Table for churners and non-churners with original values

In [176]:
```
non_churn_list = non_churn['number'].values.tolist()
churn_list = churn['number'].values.tolist()

churners = ori_train.loc[ori_train['number'].isin(churn_list)]
churners = churners.drop(columns=['number'])

non_churners = ori_train.loc[ori_train['number'].isin(non_churn_list)]
non_churners = non_churners.drop(columns=['number'])
```

In [177]:
```python
round(churners.describe(),2)
```

Out[177]:

	total_values	total_quantity	avg_between	f1	f2	f3	f4	f5	f8	
count	486.00	486.00	486.00	486.00	486.00	486.00	486.00	486.00	486.00	48
mean	85.45	53.18	51.71	18.61	3.96	3.68	5.58	4.20	4.40	
std	82.46	55.13	46.08	18.44	10.50	9.49	13.42	12.70	13.49	
min	2.29	1.00	0.00	0.89	0.00	0.00	0.00	0.00	0.00	
25%	25.14	10.00	20.25	6.89	0.00	0.00	0.00	0.00	0.00	
50%	59.24	36.00	41.00	12.51	0.00	0.00	0.00	0.00	0.00	
75%	121.44	80.00	71.00	23.29	1.56	0.00	4.02	0.00	0.00	
max	448.52	414.00	259.00	110.63	98.56	80.08	118.08	153.18	161.42	15

In [178]:
```python
round(non_churners.describe(),2)
```

Out[178]:

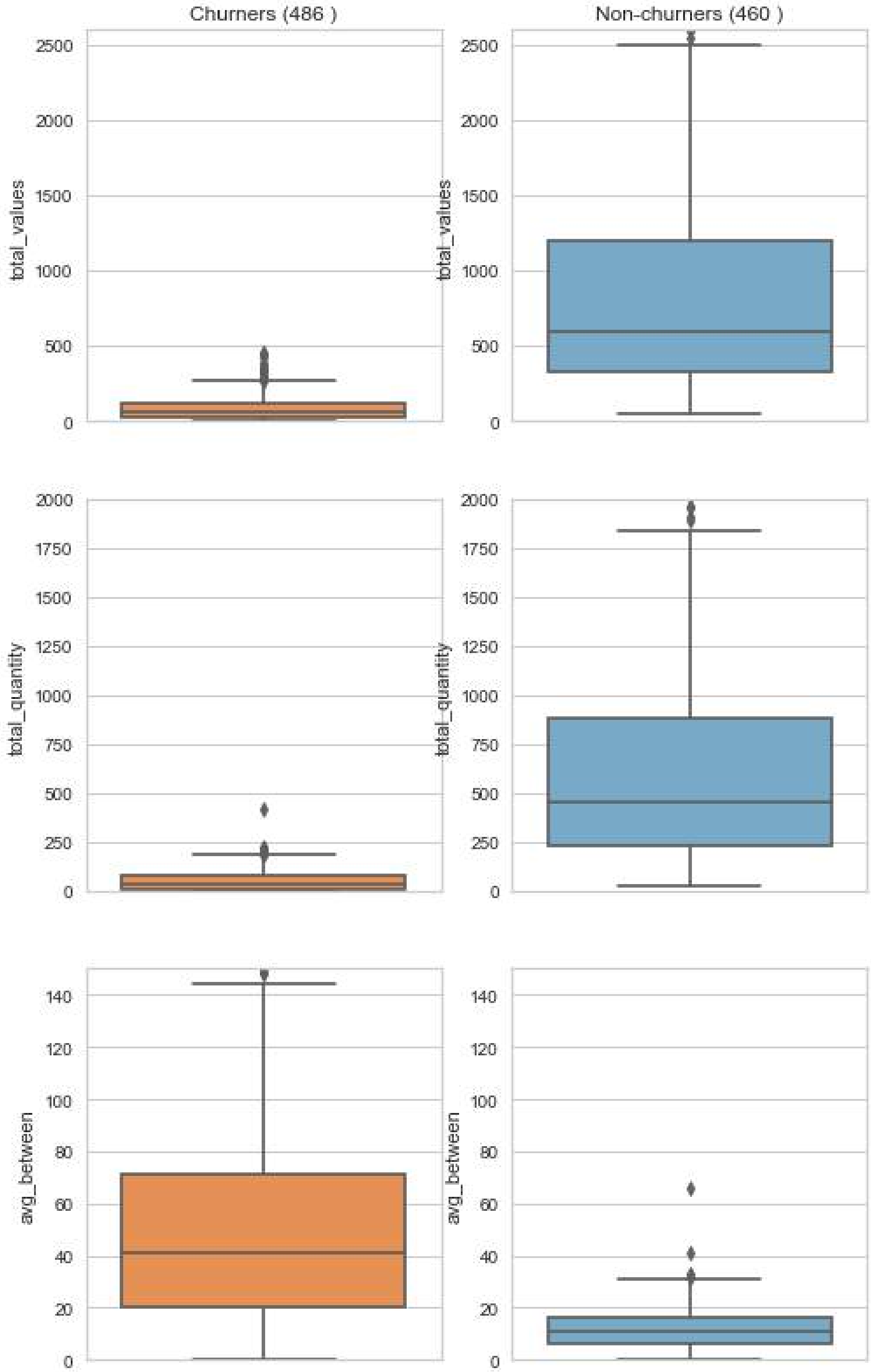
	total_values	total_quantity	avg_between	f1	f2	f3	f4	f5	f8	
count	460.00	460.00	460.00	460.00	460.00	460.00	460.00	460.00	460.00	46
mean	951.53	700.33	12.10	68.86	63.90	53.60	64.09	61.82	60.26	5
std	1078.16	797.53	7.92	74.43	81.34	72.96	84.16	86.36	88.16	8
min	48.39	22.00	0.00	1.59	0.00	0.00	0.00	0.00	0.00	
25%	328.20	229.50	6.00	21.47	10.66	7.96	8.85	7.31	0.00	
50%	599.29	451.50	11.00	43.62	35.91	29.42	35.45	37.76	29.55	2
75%	1196.39	884.25	16.25	91.74	85.10	66.36	85.93	82.10	84.50	7
max	8914.41	7076.00	66.00	493.45	604.46	508.77	618.39	722.03	960.65	67

4. Visualization of comparing churners and non-churners

1) Box-plot of total values, total quantity and average between visits

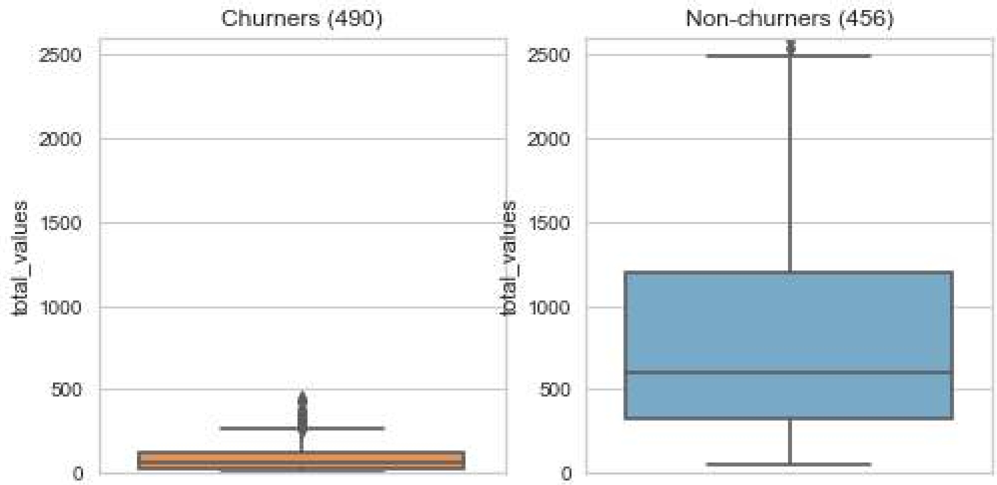
In [202]:
```
figure, (((ax1, ax2), (ax3, ax4), (ax5, ax6))) = plt.subplots(3, 2)
figure.set_size_inches(8,14)

sns.boxplot(churners.total_values, ax=ax1, orient = 'v', palette='Oranges'
           ).set_title('Churners (486 )')
sns.boxplot(non_churners.total_values, ax=ax2, orient = 'v', palette='Blues'
           ).set_title('Non-churners (460 )')
ax1.set(ylim=(0,2600))
ax2.set(ylim=(0,2600))

sns.boxplot(churners.total_quantity, ax=ax3, orient = 'v', palette='Oranges'
           )
sns.boxplot(non_churners.total_quantity, ax=ax4, orient = 'v', palette='Blues'
           )
ax3.set(ylim=(0,2000))
ax4.set(ylim=(0,2000))

sns.boxplot(churners.avg_between, ax=ax5, orient = 'v', palette='Oranges'
           )
sns.boxplot(non_churners.avg_between, ax=ax6, orient = 'v', palette='Blues'
           )
ax5.set(ylim=(0,150))
ax6.set(ylim=(0,150))
```

Out[202]: [(0.0, 150.0)]



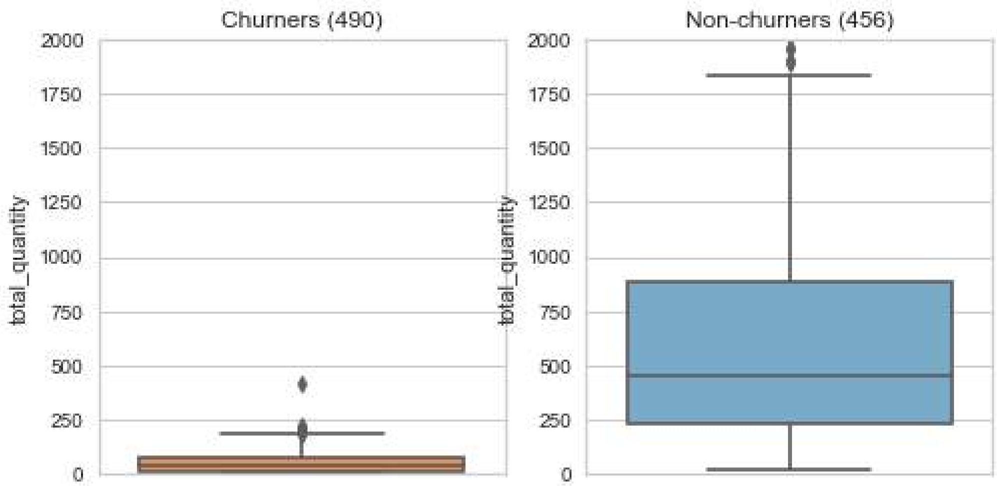
In [180]:
```
figure, (ax1, ax2) = plt.subplots(1, 2)
figure.set_size_inches(8,4)

sns.boxplot(churners.total_values, ax=ax1, orient = 'v', palette='Oranges'
           ).set_title('Churners (486 )')
sns.boxplot(non_churners.total_values, ax=ax2, orient = 'v', palette='Blues'
           ).set_title('Non-churners (460 )')
ax1.set(ylim=(0,2600))
ax2.set(ylim=(0,2600))
```

Out[180]:
```
[(0.0, 2600.0)]
```



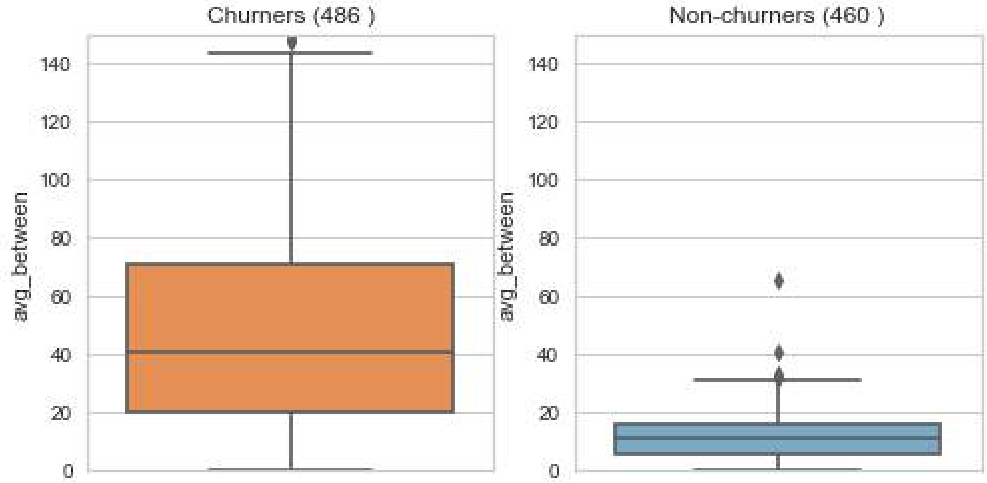
In [181]:
```
figure, (ax1, ax2) = plt.subplots(1, 2)
figure.set_size_inches(8,4)

sns.boxplot(churners.total_quantity, ax=ax1, orient = 'v', palette='Oranges'
           ).set_title('Churners (486 )')
sns.boxplot(non_churners.total_quantity, ax=ax2, orient = 'v', palette='Blues'
           ).set_title('Non-churners (460 )')
ax1.set(ylim=(0,2000))
ax2.set(ylim=(0,2000))
```

Out[181]:
```
[(0.0, 2000.0)]
```



In [201]:
```
figure, (ax1, ax2) = plt.subplots(1, 2)
figure.set_size_inches(8,4)

sns.boxplot(churners.avg_between, ax=ax1, orient = 'v', palette='Oranges'
           ).set_title('Churners (486 )')
sns.boxplot(non_churners.avg_between, ax=ax2, orient = 'v', palette='Blues'
           ).set_title('Non-churners (460 )')
ax1.set(ylim=(0,150))
ax2.set(ylim=(0,150))
```

Out[201]:
```
[(0.0, 150.0)]
```

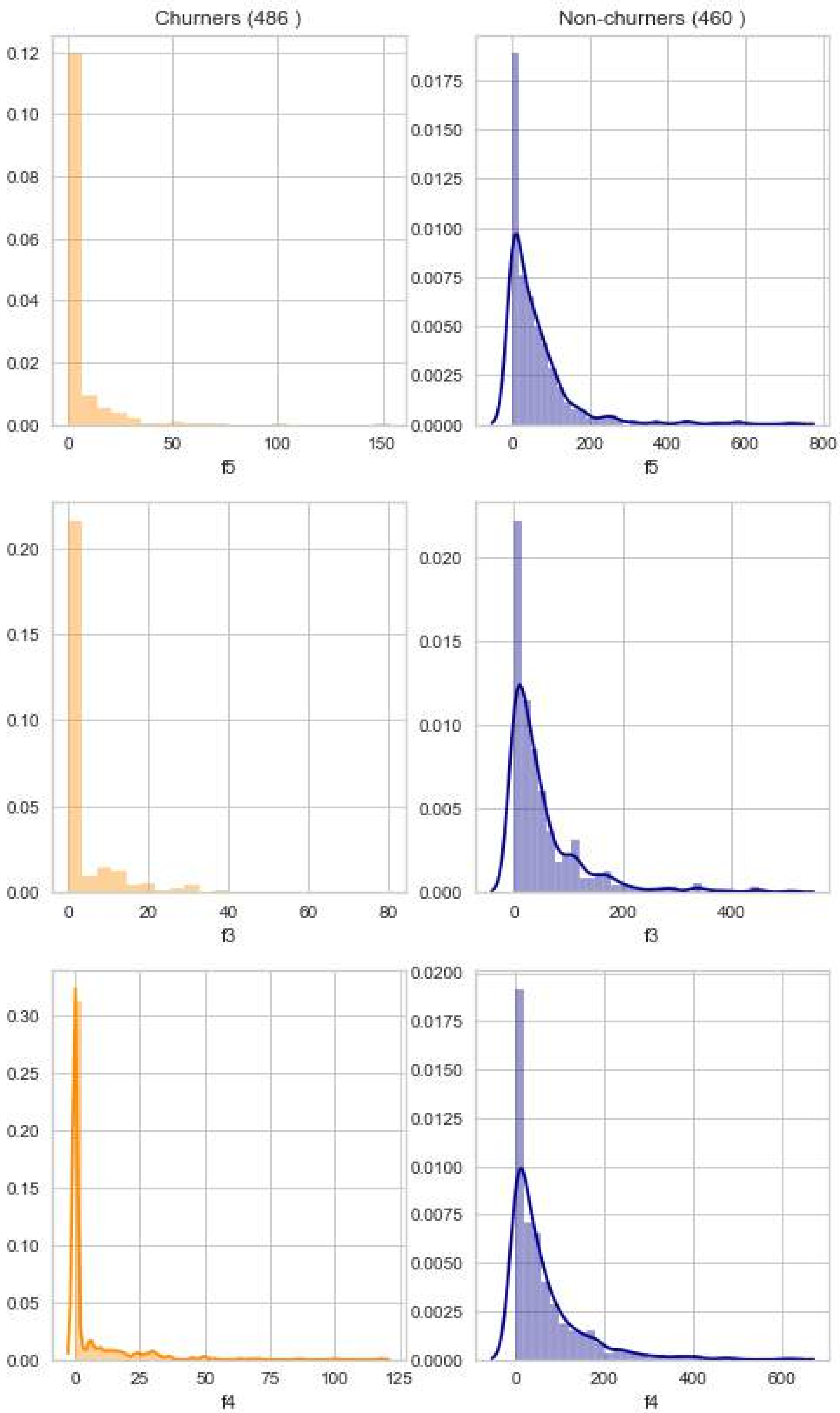


2) Distribution of f5, f3 and f4, which shows the most importance features among other periods

In [198]:

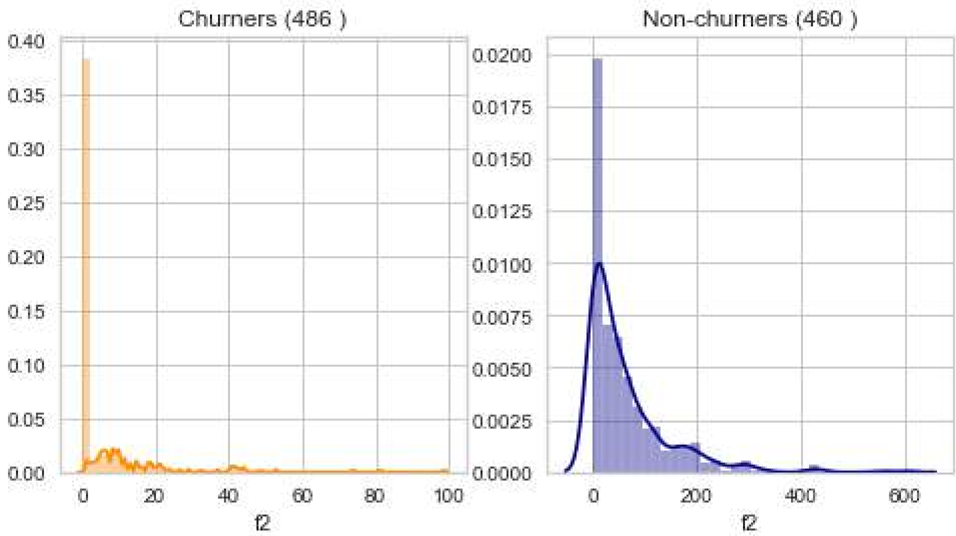
```
figure, axes  = plt.subplots(3, 2)
figure.set_size_inches(8,14)

sns.distplot(churners.f5,
             ax=axes[0][0],color='darkorange').set_title('Churners (486 )')
sns.distplot(non_churners.f5,
             ax=axes[0][1], color='darkblue').set_title('Non-churners (460 )')

sns.distplot(churners.f3,
             ax=axes[1][0],color='darkorange')
sns.distplot(non_churners.f3,
             ax=axes[1][1],color='darkblue')

sns.distplot(churners.f4,
             ax=axes[2][0],color='darkorange')
sns.distplot(non_churners.f4,
             ax=axes[2][1],color='darkblue')
```

Out[198]: <matplotlib.axes._subplots.AxesSubplot at 0x26fce174208>



```
In [200]: figure, (ax1, ax2)  = plt.subplots(1, 2)
          figure.set_size_inches(8,4)

          sns.distplot(churners.f2,
                       ax=ax1,color='darkorange').set_title('Churners (486 )')
          sns.distplot(non_churners.f2,
                       ax=ax2, color='darkblue').set_title('Non-churners (460 )')
```

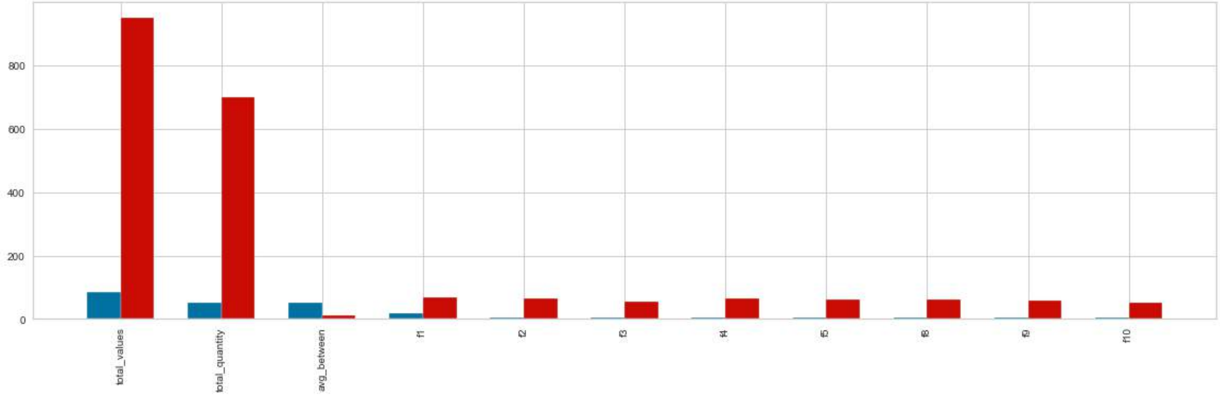
```
Out[200]: Text(0.5, 1.0, 'Non-churners (460 )')
```



3) Basic summary of churners and non-churners

```
In [185]: def plot_sidebyside_bar( labels, series1, series2, xlabel_in = '', figwidth = 2
          0 ):
            # The data
            indices = range(len(series1))
            names = labels
            # Calculate optimal width
            width = np.min(np.diff(indices))/3.

            fig = plt.figure()
            ax = fig.add_subplot(111)
            ax.bar(indices-width/2.,series1,width,color='b',label='-Ymin')
            ax.bar(indices+width/2.,series2,width,color='r',label='Ymax')
            #tiks = ax.get_xticks().tolist()
            plt.xticks(indices)
            ax.axes.set_xticklabels(names, rotation='vertical')
            ax.set_xlabel(xlabel_in)
            plt.gcf().set_figwidth(figwidth)
            plt.show()
```

In [186]:
```
plot_sidebyside_bar( churners.mean().index, churners.mean().tolist(),
                    non_churners.mean().tolist() )
```



4) Using Boruta to investigate feature importance

In [187]:
```
from boruta import BorutaPy
```

In [188]:
```
data_proba.head()
```

Out[188]:

	total_values	total_quantity	avg_between	f2	f3	f4	f6	f7	
0	-0.14	0.05	0.64	-5159.45	-5159.45	-5159.06	-5159.51	-5159.44	-5159
1	-1.13	-1.15	1.22	-5159.63	-5159.63	-5159.63	-5159.55	-5159.63	-5159
2	-1.78	-1.85	-2.14	-5159.63	-5159.63	-5159.63	-5159.63	-5159.63	-5159
3	-0.07	-0.21	0.60	-5159.63	-5159.63	-5159.26	-5159.51	-5159.53	-5159
4	-0.69	-0.25	0.94	-5159.63	-5159.63	-5159.63	-5159.63	-5159.45	-5159

In [189]:

```
X = data_proba.drop(columns=['proba'])
y = data_proba.proba
rf = RandomForestClassifier(n_estimators = 10)
feat_selector = BorutaPy(rf, n_estimators='auto', verbose=2, random_state=1)
feat_selector.fit(X.values, y)
```

```
Iteration: 	1 / 100
Confirmed: 	0
Tentative: 	9
Rejected: 	0
Iteration: 	2 / 100
Confirmed: 	0
Tentative: 	9
Rejected: 	0
Iteration: 	3 / 100
Confirmed: 	0
Tentative: 	9
Rejected: 	0
Iteration: 	4 / 100
Confirmed: 	0
Tentative: 	9
Rejected: 	0
Iteration: 	5 / 100
Confirmed: 	0
Tentative: 	9
Rejected: 	0
Iteration: 	6 / 100
Confirmed: 	0
Tentative: 	9
Rejected: 	0
Iteration: 	7 / 100
Confirmed: 	0
Tentative: 	9
Rejected: 	0
Iteration: 	8 / 100
Confirmed: 	8
Tentative: 	1
Rejected: 	0
Iteration: 	9 / 100
Confirmed: 	8
Tentative: 	1
Rejected: 	0
Iteration: 	10 / 100
Confirmed: 	8
Tentative: 	1
Rejected: 	0
Iteration: 	11 / 100
Confirmed: 	8
Tentative: 	1
Rejected: 	0
Iteration: 	12 / 100
Confirmed: 	9
Tentative: 	0
Rejected: 	0


BorutaPy finished running.

Iteration: 	13 / 100
Confirmed: 	9
Tentative: 	0
Rejected: 	0
```

Out[189]:
```
BorutaPy(alpha=0.05,
         estimator=RandomForestClassifier(bootstrap=True, ccp_alpha=0.0,
                                          class_weight=None, criterion='gini',
                                          max_depth=None, max_features='auto',
                                          max_leaf_nodes=None, max_samples=Non
e,
                                          min_impurity_decrease=0.0,
                                          min_impurity_split=None,
                                          min_samples_leaf=1,
                                          min_samples_split=2,
                                          min_weight_fraction_leaf=0.0,
                                          n_estimators=42, n_jobs=None,
                                          oob_score=False,
                                          random_state=RandomState(MT19937) at
0x26FCD839378,
                                          verbose=0, warm_start=False),
         max_iter=100, n_estimators='auto', perc=100,
         random_state=RandomState(MT19937) at 0x26FCD839378, two_step=True,
         verbose=2)
```

In [190]:
```
print( X.columns[feat_selector.support_] )
```

```
Index(['total_values', 'total_quantity', 'avg_between', 'f2', 'f3', 'f4', 'f6',
       'f7', 'f9'],
      dtype='object')
```

5) Partial Dependence Plots for two most importanct features, 'total_values' and 'avg_between'

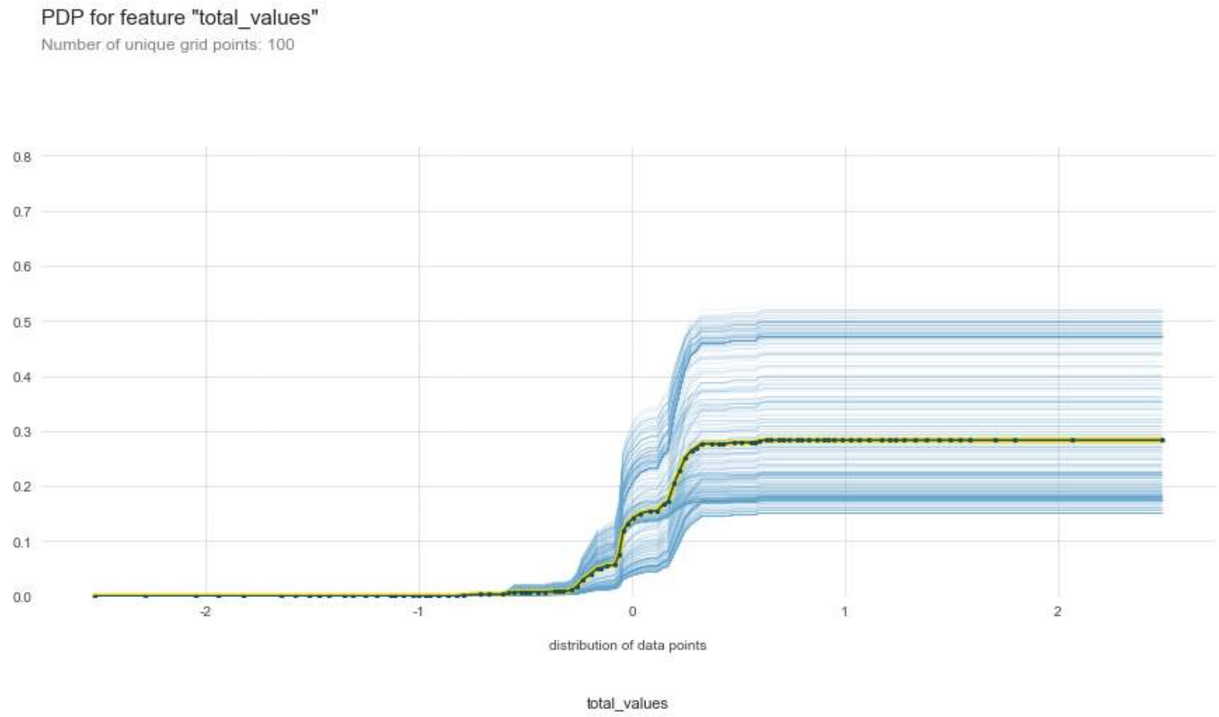
In [191]:
```
from pdpbox import pdp, get_dataset, info_plots
```

In [192]:
```
import xgboost as xgb
xgb = xgb.XGBClassifier(objective='binary:logistic',
                        silent=True, nthread=1, random_state=42, n_jobs=-1,
                        subsample= 0.20823379771832098, n_estimators= 862,
                        min_child_weight= 8, max_depth= 5,
                        learning_rate= 0.017194260557609198, gamma= 0.2976351441631
313,
                        colsample_bytree= 0.7824279936868144)
```

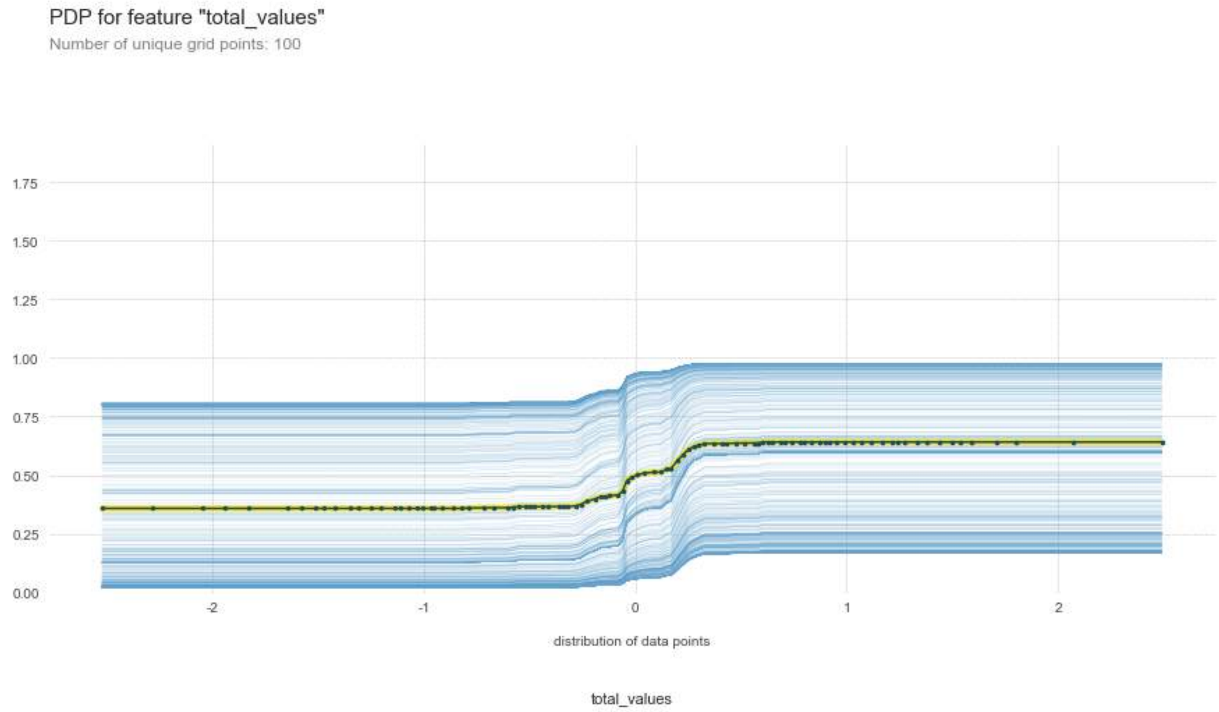
In [193]:
```
xgb.fit(X,y)
```

Out[193]:
```
XGBClassifier(base_score=0.5, booster='gbtree', colsample_bylevel=1,
              colsample_bynode=1, colsample_bytree=0.7824279936868144,
              gamma=0.2976351441631313, learning_rate=0.017194260557609198,
              max_delta_step=0, max_depth=5, min_child_weight=8, missing=None,
              n_estimators=862, n_jobs=-1, nthread=1,
              objective='binary:logistic', random_state=42, reg_alpha=0,
              reg_lambda=1, scale_pos_weight=1, seed=None, silent=True,
              subsample=0.20823379771832098, verbosity=1)
```

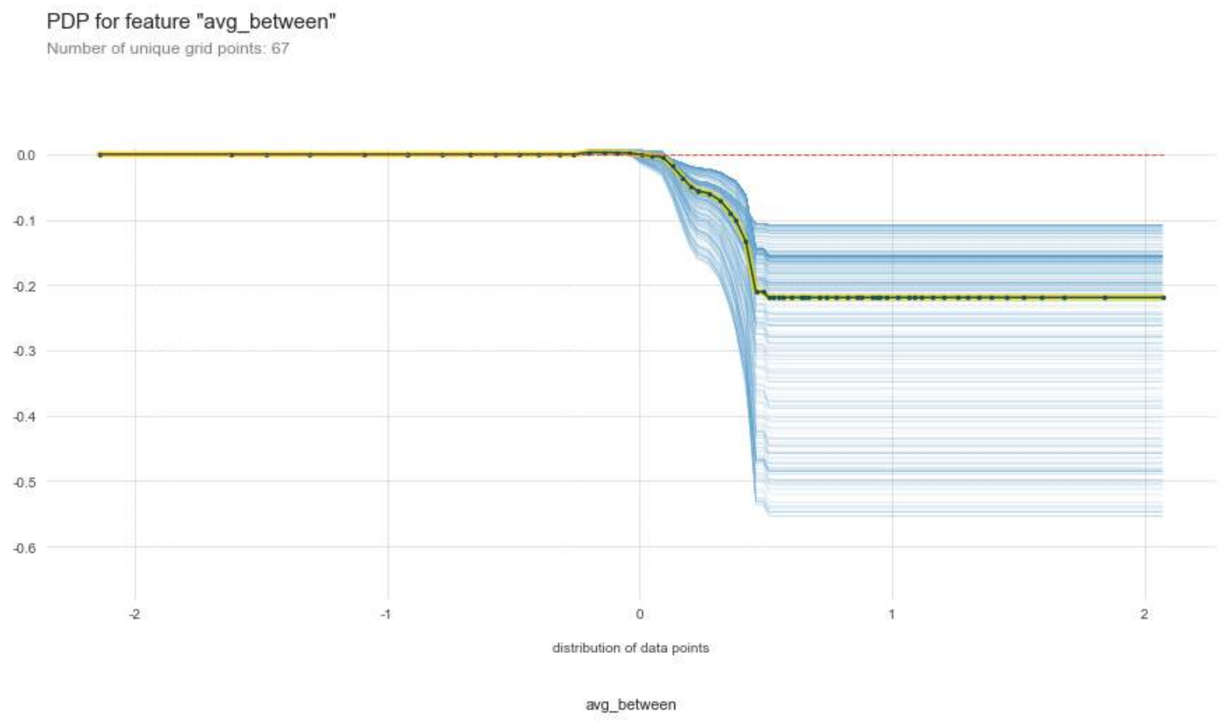
In [194]:
```
pdp_obj = pdp.pdp_isolate(
    model=xgb, dataset=data_proba, model_features=X.columns, feature='total_values',num_grid_points = 100
)

fig, axes = pdp.pdp_plot(pdp_obj, 'total_values', plot_lines=True, frac_to_plot=0.5, plot_pts_dist=True)
```



In [195]:
```
fig, axes = pdp.pdp_plot(pdp_obj, 'total_values', plot_lines=True, frac_to_plot=0.9, plot_pts_dist=True, center = False)
```



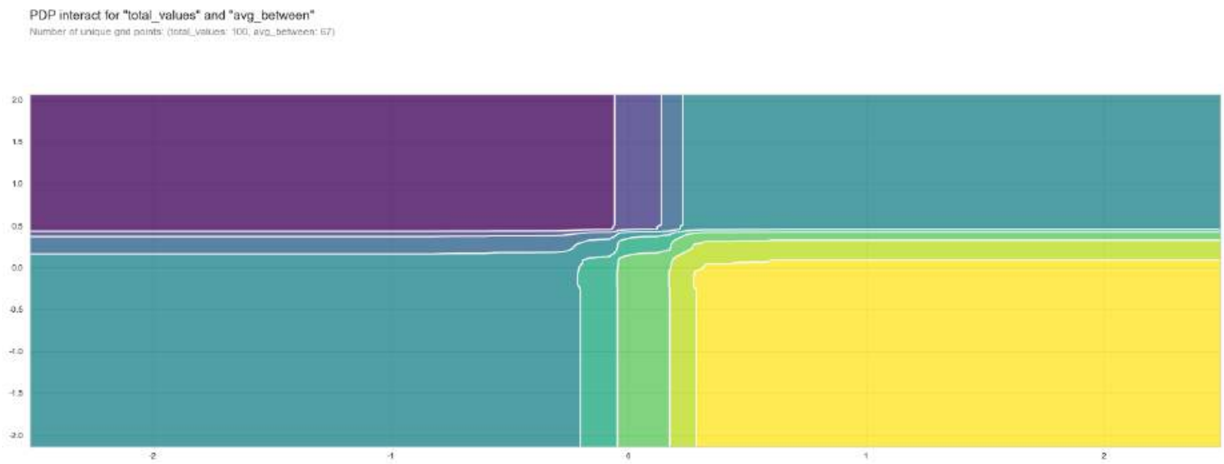
In [196]:
```
pdp_obj = pdp.pdp_isolate(
    model=xgb, dataset=data_proba, model_features=X.columns, feature='avg_between',num_grid_points = 100
)

fig, axes = pdp.pdp_plot(pdp_obj, 'avg_between', plot_lines=True, frac_to_plot=0.5, plot_pts_dist=True)
```



In [197]:

```
feats = ['total_values', 'avg_between']
p = pdp.pdp_interact(xgb, X, X.columns, feats,num_grid_points = [100,100])
pdp.pdp_interact_plot(p, feats, figsize = (25,10))
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-197-ea24b1e705c6> in <module>
      1 feats = ['total_values', 'avg_between']
      2 p = pdp.pdp_interact(xgb, X, X.columns, feats,num_grid_points = [100,10
0])
----> 3 pdp.pdp_interact_plot(p, feats, figsize = (25,10))

~\Anaconda3\lib\site-packages\pdpbox\pdp.py in pdp_interact_plot(pdp_interact_o
ut, feature_names, plot_type, x_quantile, plot_pdp, which_classes, figsize, nco
ls, plot_params)
    773             fig.add_subplot(inter_ax)
    774             _pdp_inter_one(pdp_interact_out=pdp_interact_plot_data[0],
 inter_ax=inter_ax, norm=None,
--> 775                            feature_names=feature_names_adj, **inter_par
ams)
    776     else:
    777         wspace = 0.3

~\Anaconda3\lib\site-packages\pdpbox\pdp_plot_utils.py in _pdp_inter_one(pdp_in
teract_out, feature_names, plot_type, inter_ax, x_quantile, plot_params, norm,
 ticks)
    330             # for numeric not quantile
    331             X, Y = np.meshgrid(pdp_interact_out.feature_grids[0], pdp_i
nteract_out.feature_grids[1])
--> 332         im = _pdp_contour_plot(X=X, Y=Y, **inter_params)
    333     elif plot_type == 'grid':
    334         im = _pdp_inter_grid(**inter_params)

~\Anaconda3\lib\site-packages\pdpbox\pdp_plot_utils.py in _pdp_contour_plot(X,
 Y, pdp_mx, inter_ax, cmap, norm, inter_fill_alpha, fontsize, plot_params)
    249     c1 = inter_ax.contourf(X, Y, pdp_mx, N=level, origin='lower', cmap=
cmap, norm=norm, alpha=inter_fill_alpha)
    250     c2 = inter_ax.contour(c1, levels=c1.levels, colors=contour_color, o
rigin='lower')
--> 251     inter_ax.clabel(c2, contour_label_fontsize=fontsize, inline=1)
    252     inter_ax.set_aspect('auto')
    253

~\Anaconda3\lib\site-packages\matplotlib\axes\_axes.py in clabel(self, CS, *arg
s, **kwargs)
   6338
   6339     def clabel(self, CS, *args, **kwargs):
-> 6340         return CS.clabel(*args, **kwargs)
   6341     clabel.__doc__ = mcontour.ContourSet.clabel.__doc__
   6342

TypeError: clabel() got an unexpected keyword argument 'contour_label_fontsize'
```



In [129]:

H. Deployment

In [18]:

```
import xgboost as xgb
xgb = xgb.XGBClassifier(objective='binary:logistic',
                        silent=True, nthread=1, random_state=42, n_jobs=-1,
                        subsample= 0.20823379771832098, n_estimators= 862,
                        min_child_weight= 8, max_depth= 5,
                        learning_rate= 0.017194260557609198, gamma= 0.2976351441631
313,
                        colsample_bytree= 0.7824279936868144)
```

1. Model Implementation

Five steps of model implementation

1) Load the new dataset

2) Pre-processing the data

3) Model prediction

4) Model evaluation

5) Checking churners and non-churners

1) Load the new dataset

In [ ]:
```
ws = 33
ows = 33
now = 609

test = get_dataset_value(now-ows, ws, ows)
train = get_dataset_value(now-2*ows, ws, ows)
```

2) Pre-processing the data

In [ ]:

```
        # output feature changes to binary, 1: non- churn, 0: churn
test[1][test[1]>0] = 1 # non-chrun
train[1][train[1]>0] = 1 # non-chrun

        # Balancing unbalanced output feature in train data set using SMOTE
smote = SMOTE(random_state=42)
X_train, y_train = smote.fit_resample(train[0], train[1])

X_train = pd.DataFrame(X_train,
                      columns=['total_values','total_quantity','avg_between',
                              'f1','f2','f3','f4','f5','f6','f7','f8','f9','f1
0','f11'])
y_train = pd.DataFrame(y_train)

        # standardizing Temporal data in train set
train_X = pd.DataFrame()

for i in X_train.iloc[:,3:14].values:
        a = i - X_train.iloc[:,3:14].values.sum()
        b = a / np.std(X_train.iloc[:,3:14].values)

        new_row = pd.DataFrame( [[b]] )
        train_X = train_X.append(new_row, ignore_index = True)

train_X.columns = ['f']
train_X = pd.DataFrame(train_X.f.tolist(),
                            columns=['f1','f2','f3','f4','f5','f6','f7','f8',
                                         'f9','f10','f11'])

        # standardizing traditional data in train set
        # Step 1: log1p
train_X2 = X_train.drop(columns=['f1','f2','f3','f4','f5','f6','f7','f8','f9',
'f10','f11'])
train_X2_log = np.log1p(train_X2)
        # Step 2: StandardScaler
scaler = StandardScaler()
train_X2_scaled = scaler.fit_transform(train_X2_log)

        # transform into a dataframe
train_X2_scaled = pd.DataFrame(train_X2_scaled, index=train_X2_log.index,
                             columns=train_X2_log.columns)
final_train = pd.concat([train_X2_scaled, train_X], axis=1)
final_train = round(final_train,2)

        # # standardizing Temporal data in validation set
test_X = pd.DataFrame()

for i in test[0].iloc[:,3:14].values:
        a = i - test[0].iloc[:,3:14].values.sum()
        b = a / np.std(test[0].iloc[:,3:14].values)

        new_row = pd.DataFrame( [[b]] )
        test_X = test_X.append(new_row, ignore_index = True)

test_X.columns = ['f']
test_X = pd.DataFrame(test_X.f.tolist(),
                            columns=['f1','f2','f3','f4','f5','f6','f7','f8',
                                         'f9','f10','f11'])

        # standardizing traditional data in validation set
        # Step 1: log1p
test_X2 = test[0].drop(columns=['f1','f2','f3','f4','f5','f6','f7','f8','f9','f
```

```
10','f11'])
test_X2_log = np.log1p(test_X2)
        # Step 2: StandardScaler
scaler = StandardScaler()
test_X2_scaled = scaler.fit_transform(test_X2_log)

        # transform into a dataframe
test_X2_scaled = pd.DataFrame(test_X2_scaled, index=test_X2_log.index,
                             columns=test_X2_log.columns)
        # Merge into final
final_test = pd.concat([test_X2_scaled, test_X], axis=1)
final_test = round(final_test,2)

        # Deleting feature 'f1'
y_test = test[1].copy()
X_train = final_train.copy()
X_test = final_test.copy()
X_train = X_train.drop(columns=['f8','f1','f5','f10','f11'])
X_test = X_test.drop(columns=['f8','f1','f5','f10','f11'])
```

3) Model prediction

In [ ]:
```
import xgboost as xgb
xgb = xgb.XGBClassifier(objective='binary:logistic',
                    silent=True, nthread=1, random_state=42, n_jobs=-1,
                    subsample= 0.20823379771832098, n_estimators= 862,
                    min_child_weight= 8, max_depth= 5,
                    learning_rate= 0.017194260557609198, gamma= 0.2976351441631
313,
                    colsample_bytree= 0.7824279936868144)

xgb.fit(X_train, y_train)
y_pred = xgb.predict(X_test)
```

4) Model evaluation

In [ ]:
```
xgb.fit(train, y_train)
print("train set accuracy : {:.3f}".format(xgb.score(X_train, y_train)))
print("test set accuracy : {:.3f}".format(xgb.score(X_test, y_test)))
f1 = f1_score(y_test, y_pred)
print('Test set f1 score for best params:', round(f1,3))
print('=========================================')
print('Confusion Matrix')
confusion = pd.DataFrame(confusion_matrix(y_test, y_pred, labels=[1,0]),
                        index=['y_true Yes','y_ture No'],
                        columns=['y_predict Yes','y_predict No'])
print(confusion)

plt.figure(figsize=(10,6))
xticklables = ['y_predict Yes','y_predict No']
yticklables = ['y_true Yes','y_ture No']

annot_kws={'fontsize':20,
           'fontstyle':'italic',
           'color':"k",
           'alpha':1,
           'verticalalignment':'center'}

sns.heatmap(confusion/np.sum(confusion), annot=True,
            fmt='.2%', cmap='Blues',
           xticklabels = xticklables,
           yticklabels = yticklables,
           annot_kws = annot_kws)
```

5) Checking churners and non-churners

In [ ]:
```
# Predicting churners using embeded probability in XGBoost
X_train['proba'] = xgb.predict_proba(X_train[X_train.columns])[:,1]

# Change label, 1 as non-chuners, 0 as churners
X_train.loc[ (X_train.proba >= 0.5), 'proba'] = 1  # not churn
X_train.loc[ (X_train.proba < 0.5), 'proba'] = 0  # churn

# Checking the number of customer who churned and not churned
t = Texttable()
t.add_rows( [ ['Customer','Number'], ['Churn', result[0]], ['Non-churn',result[
1]]])
print(t.draw())
```

Coursework: Customer Analytics using K-Means clustering

- University of Nottingham (UK), MSc Business Analytics
- Course: Analytics Specialisations and Applications
- Year: 2020
- Language:Python

The Problem Definition

Perform a market segmentation on a transactional dataset that has been provided by a national convenience store chain (4 files describing 3000 customers over 6 months). Produce profiles for 5-7 customer segments including statistical summary and a pen profile for each segment. The following is summary of files.

- Customer (id, visits, total quantity, average quantity, total spend, average spend)
- Category spends (20 item categories)
- Basket (purchase time, basket quantity, basket spend, basket categories)
- Line item (breaks down each basket)

The Process of Data Analytics

An executive Summary

RFM scores, spend habit and item spend were created through feature engineering, and reduced the correlation and the number of dimensions through PCA. I used K-Means to run customer clustering to create a total of six segments with a 0.228 silhouette score.

Feature Description

Mainly focues three features to explain customer's behaviour.

- Spend: 20% of customers made 80% of the company's profits.
- Frequency: Identify visit patterns through the frequency of visits to the store.
- Average spend: Identify transaction patterns. (Buy a lot of inexpensive items or buy less expensive items)

After Feature selection, technical approach as below was performed.

- Log1p transformation: to remove skewness and avoid the error of infinity value
- Standard scaler: to make standard normal distribution
- PCA (Principal component analysis): The number of components is four in PCA, which it explains variance ratio over 70 per cent.

A Customer Base Summary

Describe a total of six segments based on the selected features.

A Segmentation Methodology

Explain the selection process and reasons for K used in K-Means clustering.

A Results

Describe the names and features through pen profiles for each group.

A Business Recommendation

Select the two most profitable groups for the company, justify reasons for selection and marketing strategies. Targeting groups that has the distinct features and promoted upselling marketing and limited period of discount pricing strategies.

Report

https://github.com/Chan-Young/Coursework/blob/main/Clustering_%20Customer%20Analytics.pdf
(https://github.com/Chan-Young/Coursework/blob/main/Clustering_%20Customer%20Analytics.pdf)

Package preparation

In [1]:
```
import numpy as np
import pandas as pd

import matplotlib
import matplotlib as mpl
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns
from matplotlib import style
plt.style.use('ggplot')
mpl.rcParams['axes.unicode_minus'] = False

import warnings
warnings.filterwarnings(action='ignore')
```

1. Data preparation

1-1. 'RFM' dataset

(customers_sample.csv + baskets_sample.csv)

1) Import dataset (customers_sample.csv)

In [2]:
```
customer = pd.read_csv('customers_sample.csv')
```

In [3]:
```
# drop columns: total_quantity, average_quantity, average_spend
customer=customer.drop(columns=['total_quantity','average_quantity', 'average_s
pend'] )
customer.head()
```

Out[3]:

	customer_number	baskets	total_spend
0	4749	220	£631.12
1	4757	248	£452.42
2	144	226	£261.16
3	572	285	£638.79
4	669	285	£561.42

In [4]:
```
# dtypes change: total_spend => float64
customer['total_spend'] = customer['total_spend'
                                  ].str.replace('£','').str.replace(',','').ast
ype(np.float64)
```

2) Import dataset (baskets_sample.csv)

In [5]:
```
basket = pd.read_csv('baskets_sample.csv')
```

In [6]:
```
# drop: basket_quantity, basket_spend, basket_categories
basket = basket.drop(columns=['basket_quantity','basket_spend','basket_categori
es'])
basket.head()
```

Out[6]:

	customer_number	purchase_time
0	11911	2007-03-01 07:06:00
1	4047	2007-03-01 07:13:00
2	3571	2007-03-01 07:27:00
3	4079	2007-03-01 07:34:00
4	6063	2007-03-01 07:36:00

In [7]:
```
from datetime import datetime
```

In [8]:
```
basket['purchase_time'] = basket['purchase_time'].astype('datetime64')
```

3) Merge two dataset to make RFM model

In [9]:
```
# customer + basket => cus_bas
cus_bas = pd.merge(customer, basket, left_on='customer_number',
                   right_on='customer_number', how='inner' )
```

In [10]:
```
def unique_counts(cus_bas):
    for i in cus_bas.columns:
        count = cus_bas[i].nunique()
        print(i, ':', count)
unique_counts(cus_bas)
```

```
customer_number : 3000
baskets : 1036
total_spend : 2971
purchase_time : 101990
```

In [11]:
```
import datetime as dt
```

In [12]:
```
# extract year, month and day
cus_bas['purchase_day'] = cus_bas.purchase_time.apply(
    lambda x: dt.datetime(x.year, x.month, x.day ))
cus_bas.head()
```

Out[12]:

	customer_number	baskets	total_spend	purchase_time	purchase_day
0	4749	220	631.12	2007-03-01 17:53:00	2007-03-01
1	4749	220	631.12	2007-03-02 17:00:00	2007-03-02
2	4749	220	631.12	2007-03-05 20:36:00	2007-03-05
3	4749	220	631.12	2007-03-08 17:20:00	2007-03-08
4	4749	220	631.12	2007-03-08 19:57:00	2007-03-08

In [13]:
```
# print the time period
print('Min: {}, Max: {}'.format(min(cus_bas.purchase_day), max(cus_bas.purchase
_day)))
```

```
Min: 2007-03-01 00:00:00, Max: 2007-08-31 00:00:00
```

In [14]:
```
now = max(cus_bas.purchase_day) + dt.timedelta(1)
```

In [15]:
```
# print FRM table
RFM = cus_bas.groupby('customer_number').agg({
    'purchase_time': lambda x: (now - x.max()).days,
    'customer_number': lambda x: len(x),
    'total_spend': lambda x: x.sum()/len(x)})
RFM.head()
```

Out[15]:

	purchase_time	customer_number	total_spend
customer_number			
14	1	56	675.72
45	1	33	585.73
52	2	59	222.18
61	3	37	547.87
63	7	48	293.34

In [16]:
```
# rename the columns
RFM.rename(columns = {'purchase_time':'Recency',
                      'customer_number':'Frequency',
                      'total_spend':'Monetary'}, inplace=True)
RFM.head()
```

Out[16]:

	Recency	Frequency	Monetary
customer_number			
14	1	56	675.72
45	1	33	585.73
52	2	59	222.18
61	3	37	547.87
63	7	48	293.34

```
In [17]: # create labels and assign them to tree percentile groups
r_labels = range(4,1,-1)
r_groups = pd.qcut(RFM.Recency, q = 4, labels = r_labels, duplicates='drop')
f_labels = range(1, 5)
f_groups = pd.qcut(RFM.Frequency, q = 4, labels = f_labels)
m_labels = range(1, 5)
m_groups = pd.qcut(RFM.Monetary, q = 4, labels = m_labels)
```

```
In [18]: RFM['R'] = r_groups.values
RFM['F'] = f_groups.values
RFM['M'] = m_groups.values
```

```
In [19]: RFM['RFM_Segment'] = RFM.apply(lambda x: str(x['R'])
                              + str(x['F']) + str(x['M']), axis=1)
RFM['RFM_score'] = RFM[['R','F','M']].sum(axis=1)
RFM.head()
```

Out[19]:

customer_number	Recency	Frequency	Monetary	R	F	M	RFM_Segment	RFM_score
14	1	56	675.72	4	3	3	433	10.0
45	1	33	585.73	4	2	2	422	8.0
52	2	59	222.18	4	3	1	431	8.0
61	3	37	547.87	3	2	2	322	7.0
63	7	48	293.34	2	2	1	221	5.0

```
In [20]: # rfm_score will use after the segmentation
rfm_score = RFM.copy()
```

```
In [21]: RFM = RFM.drop(columns=['R','F','M','RFM_Segment','RFM_score'])
```

```
In [22]: RFM.head()
```

Out[22]:

customer_number	Recency	Frequency	Monetary
14	1	56	675.72
45	1	33	585.73
52	2	59	222.18
61	3	37	547.87
63	7	48	293.34

4) Check the Pearson correlations in RFM dataset

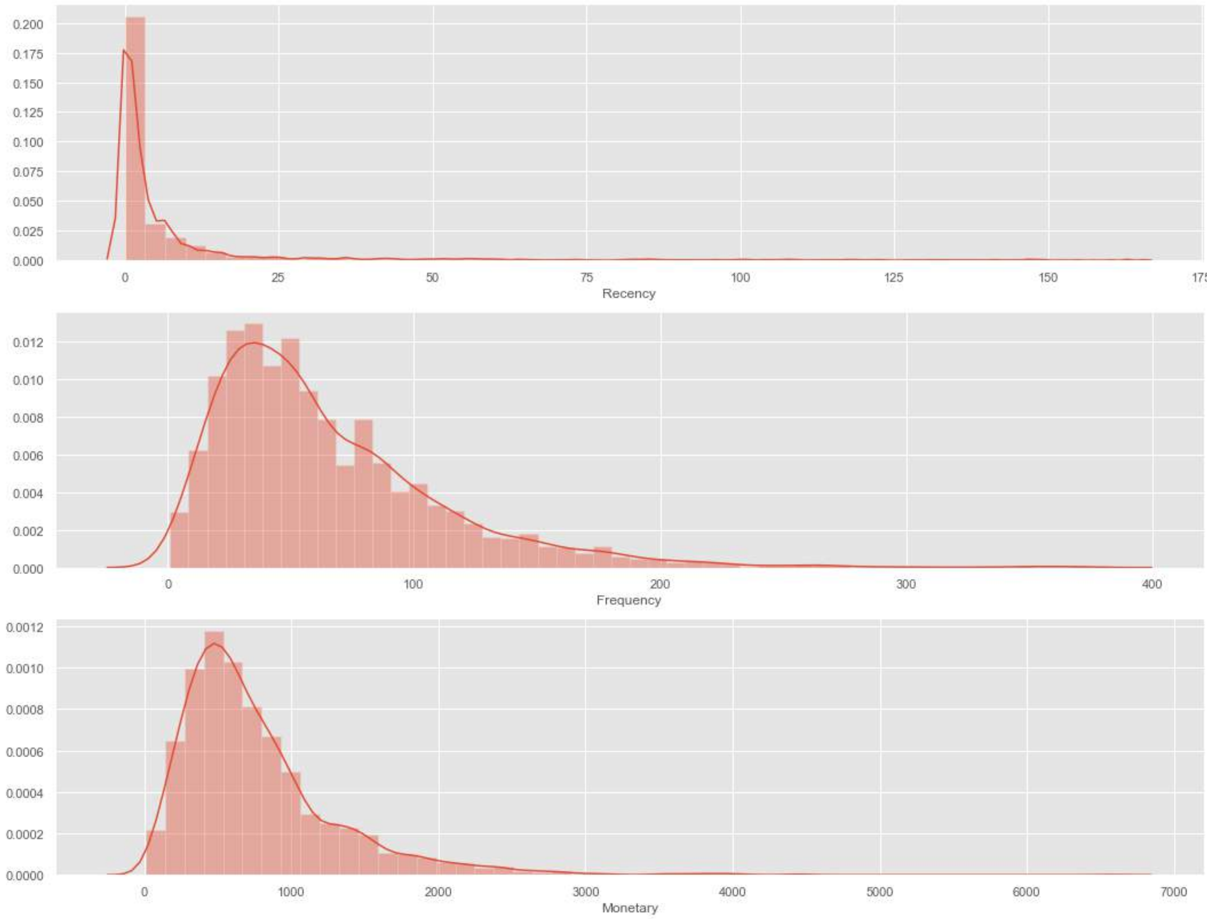
In [23]:
```
# Monetray - Frequency 0.56
corr = RFM.corr()
corr
```

Out[23]:

	Recency	Frequency	Monetary
Recency	1.000000	-0.269369	-0.245395
Frequency	-0.269369	1.000000	0.566806
Monetary	-0.245395	0.566806	1.000000

5) Check the distribution of each features

In [24]:
```
# Distribution of RFM model
plt.figure(figsize=(18,14))
plt.subplot(3,1,1); sns.distplot(RFM['Recency'])
plt.subplot(3,1,2); sns.distplot(RFM['Frequency'])
plt.subplot(3,1,3); sns.distplot(RFM['Monetary'])
plt.show()
```



In [25]:
```
RFM_ori = RFM.copy()
```

In [26]:
```
RFM_ori.head()
```

Out[26]:

customer_number	Recency	Frequency	Monetary
14	1	56	675.72
45	1	33	585.73
52	2	59	222.18
61	3	37	547.87
63	7	48	293.34

6) Applying the log1p transformation to make the data more 'normal'

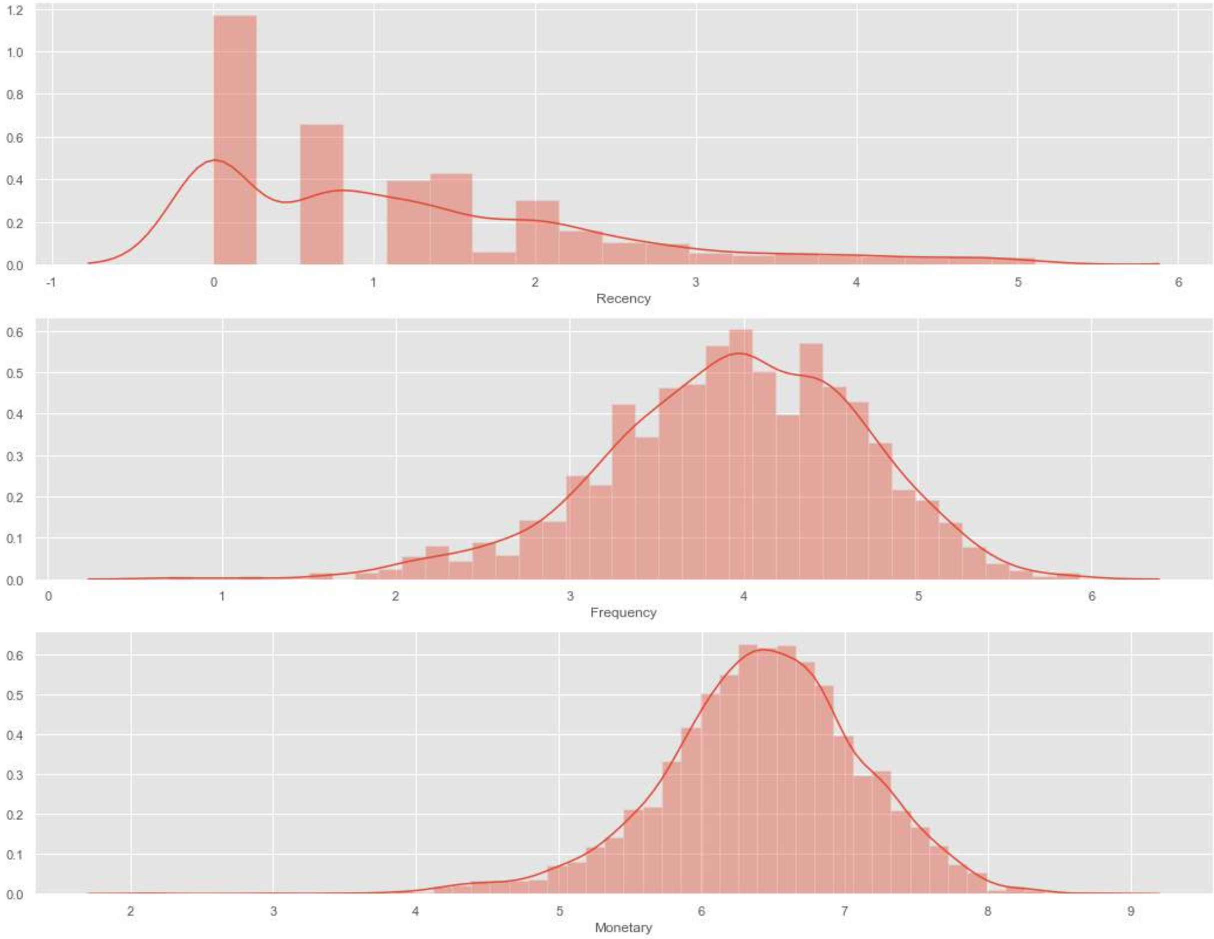
In [27]:
```
RFM_log = np.log1p(RFM)
```

In [28]:
```
RFM_log.head()
```

Out[28]:

customer_number	Recency	Frequency	Monetary
14	0.693147	4.043051	6.517258
45	0.693147	3.526361	6.374565
52	1.098612	4.094345	5.407979
61	1.386294	3.637586	6.307862
63	2.079442	3.891820	5.684736

In [29]:
```
plt.figure(figsize=(18,14))
plt.subplot(3,1,1); sns.distplot(RFM_log['Recency'])
plt.subplot(3,1,2); sns.distplot(RFM_log['Frequency'])
plt.subplot(3,1,3); sns.distplot(RFM_log['Monetary'])
plt.show()
```



In [ ]:

2. 'spend_habit' dataset

(customers_sample.csv + baskets_sample.csv)

1) Import dataset (customers_sample.csv)

In [30]:
```
customer = pd.read_csv('customers_sample.csv')
```

In [31]:
```
customer = customer.drop(columns=['baskets','average_quantity','average_spend'
])
```

In [32]:
```
customer.head()
```
Out[32]:

	customer_number	total_quantity	total_spend
0	4749	260	£631.12
1	4757	333	£452.42
2	144	303	£261.16
3	572	346	£638.79
4	669	324	£561.42

In [33]:
```
customer['total_spend'] = customer['total_spend'
                                   ].str.replace('£','').str.replace(',','').ast
ype(np.float64)
```

2) Import dataset (baskets_sample.csv)

In [34]:
```
basket = pd.read_csv('baskets_sample.csv')
```

In [35]:
```
basket.head()
```
Out[35]:

	customer_number	purchase_time	basket_quantity	basket_spend	basket_categories
0	11911	2007-03-01 07:06:00	7	£3.09	3
1	4047	2007-03-01 07:13:00	9	£7.99	5
2	3571	2007-03-01 07:27:00	9	£37.06	6
3	4079	2007-03-01 07:34:00	11	£11.91	5
4	6063	2007-03-01 07:36:00	3	£1.45	1

In [36]:
```
# transform => when creating a new column
basket['basket'] = basket.groupby('customer_number')['customer_number'].transfo
rm('count')
basket.head()
```
Out[36]:

	customer_number	purchase_time	basket_quantity	basket_spend	basket_categories	basket
0	11911	2007-03-01 07:06:00	7	£3.09	3	83
1	4047	2007-03-01 07:13:00	9	£7.99	5	178
2	3571	2007-03-01 07:27:00	9	£37.06	6	176
3	4079	2007-03-01 07:34:00	11	£11.91	5	150
4	6063	2007-03-01 07:36:00	3	£1.45	1	347

In [37]:
```
basket = basket.drop(columns=['purchase_time','basket_quantity',
                               'basket_spend','basket_categories'])
```

In [38]:
```
basket = basket.drop_duplicates()
```

3) Merge two dataset to make 'spend_habit' dataset

In [39]:
```
spend_habit = pd.merge(customer, basket, left_on='customer_number',
                      right_on='customer_number', how='inner')
```

In [40]:
```
spend_habit.head()
```

Out[40]:

	customer_number	total_quantity	total_spend	basket
0	4749	260	631.12	92
1	4757	333	452.42	27
2	144	303	261.16	22
3	572	346	638.79	40
4	669	324	561.42	36

In [41]:
```
# 1. average quantity (float 64) = total quantity / baskets
        # => average item count: total basket quantity / new baskets
spend_habit['average_item_count'] = spend_habit['total_quantity'] / spend_habit
['basket']
```

In [42]:
```
# 2. average spend (object => float64, replace £, ‘,’) = total spend / baskets
        # => average basket spend: total spend / new baskets
spend_habit['average_basket_spend'] = spend_habit['total_spend'] / spend_habit[
'basket']
```

In [43]:
```
# 3. average spend per item = total spend / total quantity
spend_habit['average_spend_per_item'] = spend_habit['total_spend'] / spend_habi
t['total_quantity']
```

In [44]:
```
spend_habit = spend_habit.round({'average_item_count':2,
                                 'average_basket_spend':2,
                                'average_spend_per_item':2})
```

In [45]:
```
# drop total spend for RFM model
# drop total quantity for correlation problem
# drop basket for RFM model

spend_habit = spend_habit.drop(columns=['total_spend','total_quantity','basket'
])
spend_habit.head()
```

Out[45]:

	customer_number	average_item_count	average_basket_spend	average_spend_per_item
0	4749	2.83	6.86	2.43
1	4757	12.33	16.76	1.36
2	144	13.77	11.87	0.86
3	572	8.65	15.97	1.85
4	669	9.00	15.60	1.73

In [46]:
```
spend_habit = spend_habit.groupby('customer_number').agg({
    'average_item_count':lambda x:x,
    'average_basket_spend':lambda x:x,
    'average_spend_per_item':lambda x:x
})
spend_habit.head()
```

Out[46]:

	average_item_count	average_basket_spend	average_spend_per_item
customer_number			
14	9.48	12.07	1.27
45	19.85	17.75	0.89
52	4.98	3.77	0.76
61	13.49	14.81	1.10
63	5.85	6.11	1.04

In [47]:
```
spend_habit.describe()
```

Out[47]:

	average_item_count	average_basket_spend	average_spend_per_item
count	3000.000000	3000.000000	3000.000000
mean	11.273407	14.801243	1.394923
std	8.538014	11.161381	0.567371
min	1.200000	1.460000	0.560000
25%	6.117500	8.037500	1.070000
50%	8.730000	11.770000	1.250000
75%	13.390000	17.440000	1.530000
max	90.750000	152.620000	7.920000

4) Check the Pearson correlations in 'spend_habit' dataset

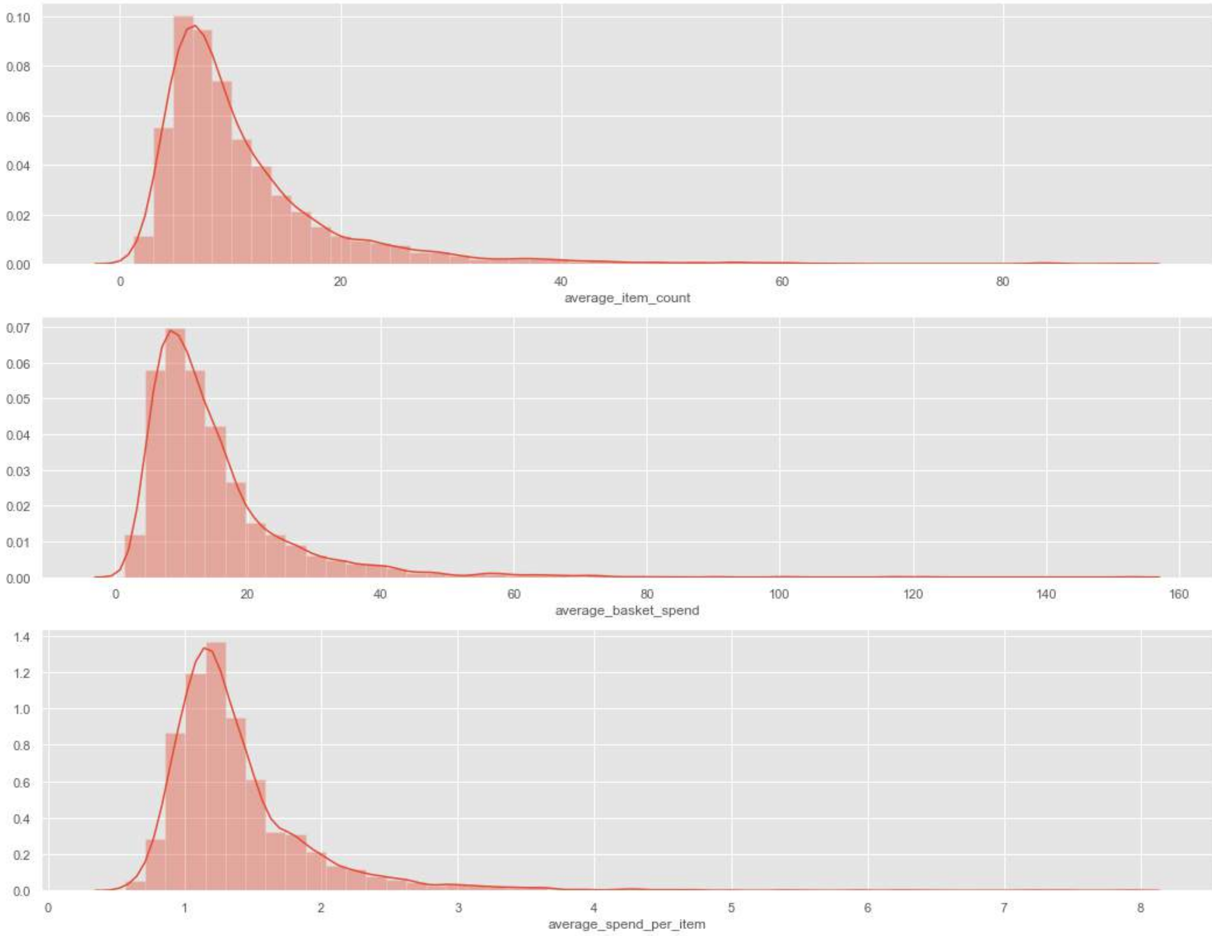
In [48]:
```
# average_item_count - average_basket_item: 0.91
corr = spend_habit.corr()
corr
```

Out[48]:

	average_item_count	average_basket_spend	average_spend_per_item
average_item_count	1.000000	0.915069	-0.190769
average_basket_spend	0.915069	1.000000	0.137865
average_spend_per_item	-0.190769	0.137865	1.000000

5) Check the distribution of each features

In [49]:
```
plt.figure(figsize=(18,14))
plt.subplot(3,1,1); sns.distplot(spend_habit['average_item_count'])
plt.subplot(3,1,2); sns.distplot(spend_habit['average_basket_spend'])
plt.subplot(3,1,3); sns.distplot(spend_habit['average_spend_per_item'])
plt.show()
```



In [50]:
```
spend_habit_ori = spend_habit.copy()
```

In [51]:
```
spend_habit_ori.head()
```

Out[51]:

customer_number	average_item_count	average_basket_spend	average_spend_per_item
14	9.48	12.07	1.27
45	19.85	17.75	0.89
52	4.98	3.77	0.76
61	13.49	14.81	1.10
63	5.85	6.11	1.04

6) Applying the log1p transformation to make the data more 'normal'

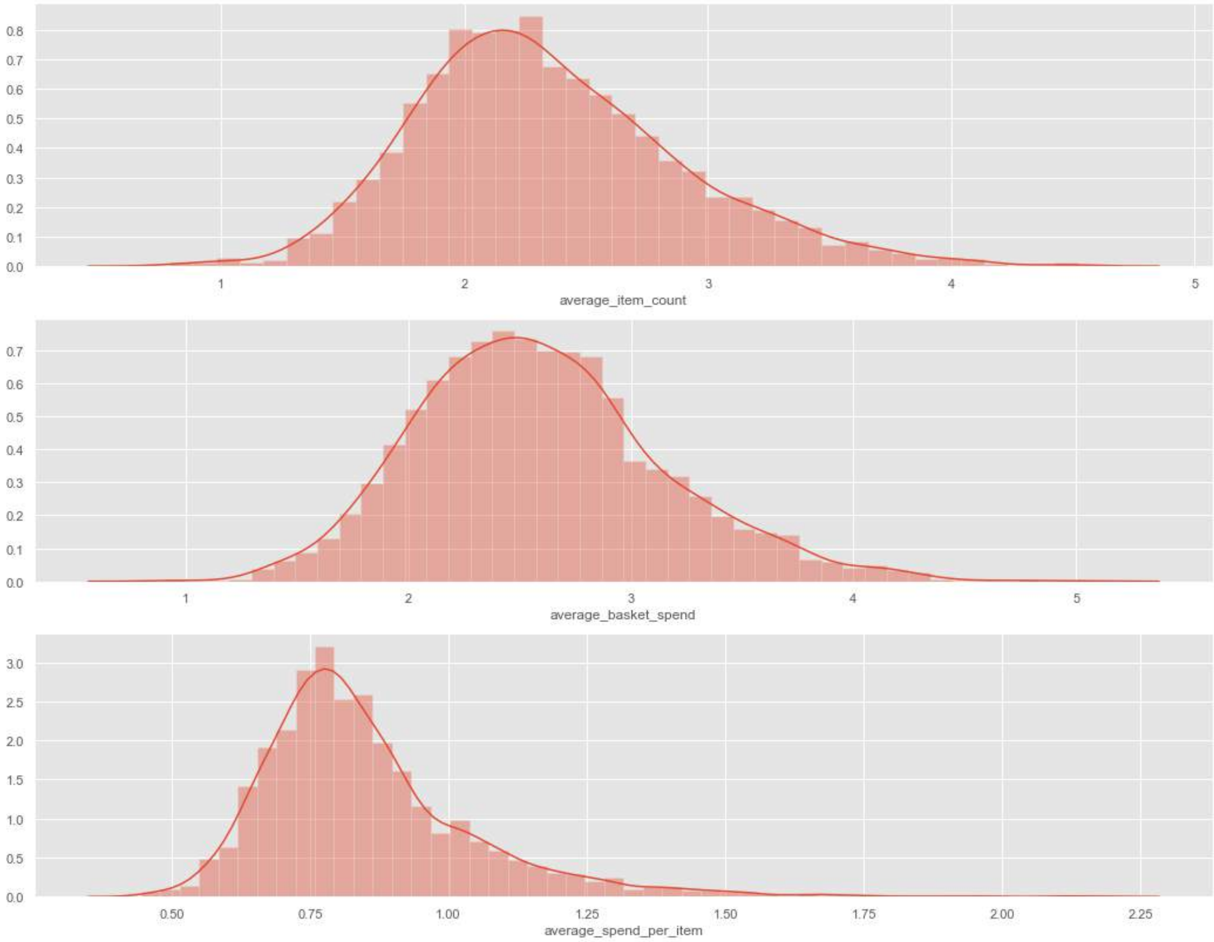
In [52]:
```
# use same log1p that I applied with above with RFM
spend_habit_log = np.log1p(spend_habit)
```

In [53]:
```python
spend_habit_log.head()
```

Out[53]:

customer_number	average_item_count	average_basket_spend	average_spend_per_item
14	2.349469	2.570320	0.819780
45	3.037354	2.931194	0.636577
52	1.788421	1.562346	0.565314
61	2.673459	2.760643	0.741937
63	1.924249	1.961502	0.712950

In [54]:
```python
plt.figure(figsize=(18,14))
plt.subplot(3,1,1); sns.distplot(spend_habit_log['average_item_count'])
plt.subplot(3,1,2); sns.distplot(spend_habit_log['average_basket_spend'])
plt.subplot(3,1,3); sns.distplot(spend_habit_log['average_spend_per_item'])
plt.show()
```



In [55]:
```
spend_habit_log.head()
```
Out[55]:

customer_number	average_item_count	average_basket_spend	average_spend_per_item
14	2.349469	2.570320	0.819780
45	3.037354	2.931194	0.636577
52	1.788421	1.562346	0.565314
61	2.673459	2.760643	0.741937
63	1.924249	1.961502	0.712950

7) Merge RFM_ori and spend_habit_ori to spend_habit_rfm_ori

In [56]:
```
# rfm_ori + cus_bas_ori
spend_habit_rfm_ori = pd.merge(spend_habit_ori, RFM_ori, left_on='customer_numb
er',
                     right_on='customer_number', how='inner')
spend_habit_rfm_ori.head()
```
Out[56]:

customer_number	average_item_count	average_basket_spend	average_spend_per_item	Recency
14	9.48	12.07	1.27	1
45	19.85	17.75	0.89	1
52	4.98	3.77	0.76	2
61	13.49	14.81	1.10	3
63	5.85	6.11	1.04	7

8) Merge RFM_log and spend_habit_log to spend_habit_rfm_log

In [57]:
```
# rfm_log + cus_bas_log
spend_habit_rfm_log = pd.merge(spend_habit_log, RFM_log, left_on='customer_numb
er',
                     right_on='customer_number', how='inner')
spend_habit_rfm_log.head()
```
Out[57]:

customer_number	average_item_count	average_basket_spend	average_spend_per_item	Recency
14	2.349469	2.570320	0.819780	0.693147
45	3.037354	2.931194	0.636577	0.693147
52	1.788421	1.562346	0.565314	1.098612
61	2.673459	2.760643	0.741937	1.386294
63	1.924249	1.961502	0.712950	2.079442

In [58]:
```
spend_habit_rfm_log.describe()
```

Out[58]:

	average_item_count	average_basket_spend	average_spend_per_item	Recency	Freque
count	3000.000000	3000.000000	3000.000000	3000.000000	3000.000
mean	2.342766	2.590780	0.852214	1.206960	3.945
std	0.544299	0.554939	0.194357	1.209856	0.739
min	0.788457	0.900161	0.444686	0.000000	0.693
25%	1.962556	2.201382	0.727549	0.000000	3.496
50%	2.275214	2.547098	0.810930	1.098612	3.988
75%	2.666534	2.914522	0.928219	1.945910	4.465
max	4.519067	5.034482	2.188296	5.105945	5.926

9) Check the Pearson correlations in 'spend_habit_rfm_log' dataset

In [59]:
```
corr = spend_habit_rfm_log.corr()
'''
average_item_count: average_basket_spend 0.86
Frequency: Monetary 0.66

Recency: Frequency -0.54 ?? new corrleation after log1p transform
'''
corr
```

Out[59]:

	average_item_count	average_basket_spend	average_spend_per_item	Re
average_item_count	1.000000	0.865978	-0.277884	0.1
average_basket_spend	0.865978	1.000000	0.235813	0.1
average_spend_per_item	-0.277884	0.235813	1.000000	0.0
Recency	0.163094	0.176682	0.019780	1.0
Frequency	-0.481542	-0.494521	-0.017810	-0.5
Monetary	0.216272	0.318651	0.185737	-0.4

In [ ]:

3. 'item_spend' dataset

(category_spends_sample.csv + lineitems_sample.csv)

1) Import dataset (category_spends_sample.csv)

In [60]:
```
spend = pd.read_csv('category_spends_sample.csv')
```

In [61]:
```
spend['practical_items'] = spend['practical_items'
                          ].str.replace('£','').str.replace(',','').astype(np.f
loat64)
spend['fruit_veg'] = spend['fruit_veg'
                          ].str.replace('£','').str.replace(',','').astype(np.f
loat64)
spend['seasonal_gifting'] = spend['seasonal_gifting'
                          ].str.replace('£','').str.replace(',','').astype(np.f
loat64)
spend['discount_bakery'] = spend['discount_bakery'
                          ].str.replace('£','').str.replace(',','').astype(np.f
loat64)

spend['drinks'] = spend['drinks'
                          ].str.replace('£','').str.replace(',','').astype(np.f
loat64)
spend['deli'] = spend['deli'
                          ].str.replace('£','').str.replace(',','').astype(np.f
loat64)
spend['world_foods'] = spend['world_foods'
                          ].str.replace('£','').str.replace(',','').astype(np.f
loat64)
spend['lottery'] = spend['lottery'
                          ].str.replace('£','').str.replace(',','').astype(np.f
loat64)
spend['cashpoint'] = spend['cashpoint'
                          ].str.replace('£','').str.replace(',','').astype(np.f
loat64)

spend['dairy'] = spend['dairy'
                          ].str.replace('£','').str.replace(',','').astype(np.f
loat64)
spend['confectionary'] = spend['confectionary'
                          ].str.replace('£','').str.replace(',','').astype(np.f
loat64)
spend['grocery_food'] = spend['grocery_food'
                          ].str.replace('£','').str.replace(',','').astype(np.f
loat64)
spend['grocery_health_pets'] = spend['grocery_health_pets'
                          ].str.replace('£','').str.replace(',','').astype(np.f
loat64)
spend['bakery'] = spend['bakery'
                          ].str.replace('£','').str.replace(',','').astype(np.f
loat64)
spend['newspapers_magazines'] = spend['newspapers_magazines'
                          ].str.replace('£','').str.replace(',','').astype(np.f
loat64)
spend['prepared_meals'] = spend['prepared_meals'
                          ].str.replace('£','').str.replace(',','').astype(np.f
loat64)
spend['soft_drinks'] = spend['soft_drinks'
                          ].str.replace('£','').str.replace(',','').astype(np.f
loat64)
spend['frozen'] = spend['frozen'
                          ].str.replace('£','').str.replace(',','').astype(np.f
loat64)
spend['meat'] = spend['meat'
                          ].str.replace('£','').str.replace(',','').astype(np.f
loat64)
spend['tobacco'] = spend['tobacco'
                          ].str.replace('£','').str.replace(',','').astype(np.f
loat64)
```

In [62]:
```
spend.head()
```

Out[62]:

	customer_number	fruit_veg	dairy	confectionary	grocery_food	grocery_health_pets	bakery
0	11387	64.58	35.91	107.78	27.08	29.59	0.0
1	8171	16.89	37.24	28.84	33.43	66.40	0.0
2	1060	87.30	82.98	49.88	20.57	37.04	0.0
3	3728	84.05	186.56	175.50	119.84	111.08	0.0
4	14621	35.16	121.31	79.23	29.03	37.17	0.0

5 rows × 21 columns

In [63]:
```
spend.describe()
```

Out[63]:

	customer_number	fruit_veg	dairy	confectionary	grocery_food	grocery_health_p
count	3000.000000	3000.000000	3000.000000	3000.000000	3000.000000	3000.0000
mean	8095.724333	69.456163	71.302683	57.347793	60.007530	60.9098
std	4686.259488	70.499654	57.966265	55.959350	57.682533	69.8050
min	14.000000	0.000000	0.000000	0.000000	0.000000	0.0000
25%	4044.750000	22.695000	31.390000	21.070000	21.087500	18.1450
50%	8218.500000	50.935000	56.875000	42.290000	44.030000	39.0750
75%	12115.500000	93.405000	95.327500	75.125000	80.922500	77.2500
max	16316.000000	1262.970000	708.040000	614.370000	1017.070000	884.4500

8 rows × 21 columns

In [ ]:

2) Import dataset (lineitems_sample.csv)

In [64]:
```
item = pd.read_csv('lineitems_sample.csv')
item.head()
```

Out[64]:

	customer_number	purchase_time	product_id	category	quantity	spend
0	14577	2007-03-10 11:58:00	722653	GROCERY_FOOD	1	£1.39
1	7210	2007-03-22 10:53:00	696136	GROCERY_HEALTH_PETS	1	£4.25
2	3145	2007-03-26 11:17:00	139543	GROCERY_HEALTH_PETS	1	£0.50
3	2649	2007-03-12 16:05:00	34890	BAKERY	1	£0.57
4	859	2007-03-10 09:53:00	613984	BAKERY	1	£1.59

In [65]:
```
item.dtypes
```

Out[65]:
```
customer_number      int64
purchase_time       object
product_id           int64
category            object
quantity             int64
spend               object
dtype: object
```

In [66]:
```
item['spend'] = item['spend'
                          ].str.replace('£','').str.replace(',','').astype(np.f
loat64)
```

In [67]:
```
category = item.groupby(by=['customer_number','category']
                       ).agg({'spend':[np.sum]}).unstack()
category.head()
```

Out[67]:

	spend						
	sum						
category	BAKERY	CASHPOINT	CONFECTIONARY	DAIRY	DELI	DISCOUNT_BAKERY	D
customer_number							
14	18.09	NaN	23.22	172.58	NaN	1.25	
45	18.00	NaN	106.54	142.16	2.00	NaN	
52	2.45	10.0	3.29	5.19	49.07	NaN	
61	32.75	NaN	46.39	55.29	19.88	NaN	
63	33.35	NaN	73.07	42.11	32.14	NaN	

In [68]:
```
category.isna().sum()
```

Out[68]:
```
             category
spend   sum  BAKERY                  37
             CASHPOINT             1794
             CONFECTIONARY           18
             DAIRY                   13
             DELI                  1050
             DISCOUNT_BAKERY       2664
             DRINKS                 915
             FROZEN                 152
             FRUIT_VEG               19
             GROCERY_FOOD            18
             GROCERY_HEALTH_PETS     49
             LOTTERY               1836
             MEAT                   177
             NEWSPAPERS_MAGAZINES   463
             PRACTICAL_ITEMS       1810
             PREPARED_MEALS         142
             SEASONAL_GIFTING      1096
             SOFT_DRINKS            224
             TOBACCO               1443
             WORLD_FOODS            593
dtype: int64
```

In [69]:
```
category = category.fillna(0)
category.head()
```

Out[69]:

	spend						
	sum						
category	BAKERY	CASHPOINT	CONFECTIONARY	DAIRY	DELI	DISCOUNT_BAKERY	D
customer_number							
14	18.09	0.0	23.22	172.58	0.00	1.25	
45	18.00	0.0	106.54	142.16	2.00	0.00	
52	2.45	10.0	3.29	5.19	49.07	0.00	
61	32.75	0.0	46.39	55.29	19.88	0.00	
63	33.35	0.0	73.07	42.11	32.14	0.00	

In [70]:
```
category = category.rename_axis().reset_index()
category.head()
```

Out[70]:

	customer_number	spend					
		sum					
category		BAKERY	CASHPOINT	CONFECTIONARY	DAIRY	DELI	DISCOUNT_E
0	14	18.09	0.0	23.22	172.58	0.00	
1	45	18.00	0.0	106.54	142.16	2.00	
2	52	2.45	10.0	3.29	5.19	49.07	
3	61	32.75	0.0	46.39	55.29	19.88	
4	63	33.35	0.0	73.07	42.11	32.14	

5 rows × 21 columns

In [71]:
```
category[category['customer_number']==14]
```

Out[71]:

	customer_number	spend					
		sum					
category		BAKERY	CASHPOINT	CONFECTIONARY	DAIRY	DELI	DISCOUNT_B
0	14	18.09	0.0	23.22	172.58	0.0	

1 rows × 21 columns

In [72]:
```
# compare with spend dataset
spend[spend['customer_number']==14]
```

Out[72]:

	customer_number	fruit_veg	dairy	confectionary	grocery_food	grocery_health_pets	bakery
427	14	11.1	172.58	23.22	56.05	11.28	0.0

1 rows × 21 columns

In [ ]:

3) Replacing 'bakery'feature in spend from category

In [73]:
```
# drop bakery in spend
spend = spend.drop(columns='bakery')
```

In [74]:
```
category_bakery = category.iloc[:, [0,1]]
```

In [75]:
```
category_bakery.head()
```

Out[75]:

	customer_number	spend
		sum
category		BAKERY
0	14	18.09
1	45	18.00
2	52	2.45
3	61	32.75
4	63	33.35

In [76]:
```
category_bakery.columns = ['customer_number', 'bakery']
```

In [77]:
```
category_bakery.head()
```

Out[77]:

	customer_number	bakery
0	14	18.09
1	45	18.00
2	52	2.45
3	61	32.75
4	63	33.35

In [78]:
```
# spend with bakery feature
spend = pd.merge(spend, category_bakery, left_on='customer_number',
                 right_on='customer_number', how='inner')
spend.head()
```

Out[78]:

	customer_number	fruit_veg	dairy	confectionary	grocery_food	grocery_health_pets	newspap
0	11387	64.58	35.91	107.78	27.08	29.59	
1	8171	16.89	37.24	28.84	33.43	66.40	
2	1060	87.30	82.98	49.88	20.57	37.04	
3	3728	84.05	186.56	175.50	119.84	111.08	
4	14621	35.16	121.31	79.23	29.03	37.17	

5 rows × 21 columns

In [79]:
```
spend[spend['customer_number']==14]
```

Out[79]:

	customer_number	fruit_veg	dairy	confectionary	grocery_food	grocery_health_pets	newsp
427	14	11.1	172.58	23.22	56.05	11.28	

1 rows × 21 columns

In [ ]:

4) Check the Pearson correlations in 'spend' dataset

In [80]:
```
corr = spend.corr()
```

In [81]: corr

```
corr
```

Out[81]:

	customer_number	fruit_veg	dairy	confectionary	grocery_food	groce
customer_number	1.000000	-0.081879	-0.083583	-0.094811	-0.044161	
fruit_veg	-0.081879	1.000000	0.629616	0.465602	0.640928	
dairy	-0.083583	0.629616	1.000000	0.610100	0.651492	
confectionary	-0.094811	0.465602	0.610100	1.000000	0.574594	
grocery_food	-0.044161	0.640928	0.651492	0.574594	1.000000	
grocery_health_pets	-0.073822	0.490674	0.557530	0.577793	0.581256	
newspapers_magazines	-0.098787	0.136383	0.219894	0.211944	0.130919	
prepared_meals	-0.115383	0.449565	0.509525	0.471222	0.481753	
soft_drinks	-0.053060	0.259851	0.404795	0.537825	0.365991	
frozen	-0.107980	0.420811	0.531251	0.574330	0.569792	
meat	-0.072248	0.550687	0.492942	0.440183	0.556404	
tobacco	0.076524	-0.017992	0.090944	0.060835	0.062182	
drinks	-0.013866	0.108806	0.070143	0.013506	0.092420	
deli	0.001120	0.225524	0.224616	0.206669	0.212570	
world_foods	-0.044833	0.206913	0.243247	0.246258	0.180413	
lottery	0.038263	-0.018643	0.036770	0.014559	0.011811	
cashpoint	0.015227	-0.049082	0.020423	0.038862	-0.015510	
seasonal_gifting	-0.016623	0.206127	0.234176	0.277407	0.195960	
discount_bakery	-0.016584	0.046758	0.000576	0.057857	0.043531	
practical_items	-0.053567	0.272480	0.277045	0.274943	0.260195	
bakery	-0.094028	0.447188	0.608368	0.542105	0.522479	

21 rows × 21 columns

```
Correlation over 0.50

fruit veg: dairy 0.62 / grocery_food 0.64 / meat 0.55
dairy:     fruit veg 0.62 / confectionary 0.61 / grocery_food 0.65
           grocery_health_pets 0.55 / prepared_meals 0.5 / frozen 0.53
confect:   dairy 0.61 / grocery_food 0.57 / grocery_health_pets 0.57
           soft_drinks 0.53 / frozen 0.52
grocery:   fruit veg 0.64 / dairy 0.65 / confectionary 0.57
           grocery_health_pets 0.57 / soft_drinks 0.6 / frozen 0.55
           meat 0.55
gro_pets:  dairy 0.55 / confectionary 0.57 / grocery_food 0.57
           frozen 0.52
prepared_meals: dairy 0.5
soft drinks:    confectionary 0.53
frozen:         dairy / confectionary / grocery_food / gro_pets
meat:           fruit veg / grocery_food
bakery:         dariy / confectionary / grocery_food
```

In [82]:
```
spend.head()
```

Out[82]:

	customer_number	fruit_veg	dairy	confectionary	grocery_food	grocery_health_pets	newspap
0	11387	64.58	35.91	107.78	27.08	29.59	
1	8171	16.89	37.24	28.84	33.43	66.40	
2	1060	87.30	82.98	49.88	20.57	37.04	
3	3728	84.05	186.56	175.50	119.84	111.08	
4	14621	35.16	121.31	79.23	29.03	37.17	

5 rows × 21 columns

In [83]:
```
spend.describe()
```

Out[83]:

	customer_number	fruit_veg	dairy	confectionary	grocery_food	grocery_health_p
count	3000.000000	3000.000000	3000.000000	3000.000000	3000.000000	3000.000C
mean	8095.724333	69.456163	71.302683	57.347793	60.007530	60.9098
std	4686.259488	70.499654	57.966265	55.959350	57.682533	69.805C
min	14.000000	0.000000	0.000000	0.000000	0.000000	0.000C
25%	4044.750000	22.695000	31.390000	21.070000	21.087500	18.145C
50%	8218.500000	50.935000	56.875000	42.290000	44.030000	39.075C
75%	12115.500000	93.405000	95.327500	75.125000	80.922500	77.250C
max	16316.000000	1262.970000	708.040000	614.370000	1017.070000	884.450C

8 rows × 21 columns

5) Dropping features that 0 value until 25% (= deleting 8 feature)

In [84]:
```
# droping features that 0 value until 25% (= deleting 8 feature)
spend_12 = spend.drop(columns=['tobacco','drinks','deli','lottery','cashpoint',
                               'seasonal_gifting', 'discount_bakery',
                              'practical_items'])
spend_12.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 3000 entries, 0 to 2999
Data columns (total 13 columns):
customer_number         3000 non-null int64
fruit_veg               3000 non-null float64
dairy                   3000 non-null float64
confectionary           3000 non-null float64
grocery_food            3000 non-null float64
grocery_health_pets     3000 non-null float64
newspapers_magazines    3000 non-null float64
prepared_meals          3000 non-null float64
soft_drinks             3000 non-null float64
frozen                  3000 non-null float64
meat                    3000 non-null float64
world_foods             3000 non-null float64
bakery                  3000 non-null float64
dtypes: float64(12), int64(1)
memory usage: 328.1 KB
```

In [85]:
```
spend_12.describe()
```

Out[85]:

	customer_number	fruit_veg	dairy	confectionary	grocery_food	grocery_health_p
count	3000.000000	3000.000000	3000.000000	3000.000000	3000.000000	3000.000
mean	8095.724333	69.456163	71.302683	57.347793	60.007530	60.909
std	4686.259488	70.499654	57.966265	55.959350	57.682533	69.805
min	14.000000	0.000000	0.000000	0.000000	0.000000	0.000
25%	4044.750000	22.695000	31.390000	21.070000	21.087500	18.145
50%	8218.500000	50.935000	56.875000	42.290000	44.030000	39.075
75%	12115.500000	93.405000	95.327500	75.125000	80.922500	77.250
max	16316.000000	1262.970000	708.040000	614.370000	1017.070000	884.450

In [86]:
```python
for i in spend_12:
    print(i,':',spend_12[i].sum())
```

```
customer_number : 24287173
fruit_veg : 208368.49
dairy : 213908.05
confectionary : 172043.38
grocery_food : 180022.59
grocery_health_pets : 182729.59999999998
newspapers_magazines : 49960.17
prepared_meals : 106441.70999999999
soft_drinks : 69910.04999999999
frozen : 106398.06
meat : 164222.06
world_foods : 25662.78
bakery : 114630.37
```

6) Dropping features that total spend is lower than 100,000 (= deleting 3 feature)

In [87]:
```python
# drop features that total spend is lower than 100,000 (= deleting 3 feature)
spend_9 = spend_12.drop(columns=['newspapers_magazines','soft_drinks',
                                 'world_foods'])
```

In [88]:
```python
spend_9.head()
```

Out[88]:

	customer_number	fruit_veg	dairy	confectionary	grocery_food	grocery_health_pets	prepared
0	11387	64.58	35.91	107.78	27.08	29.59	
1	8171	16.89	37.24	28.84	33.43	66.40	
2	1060	87.30	82.98	49.88	20.57	37.04	
3	3728	84.05	186.56	175.50	119.84	111.08	
4	14621	35.16	121.31	79.23	29.03	37.17	

7) Making a new dataset 'item_spend' dataset

In [89]:
```python
item_spend = spend_9.groupby('customer_number').agg({
    'fruit_veg':lambda x:x,
    'dairy':lambda x:x,
    'confectionary':lambda x:x,
    'grocery_food':lambda x:x,
    'grocery_health_pets':lambda x:x,
    'prepared_meals':lambda x:x,
    'frozen':lambda x:x,
    'meat':lambda x:x,
    'bakery':lambda x:x
})
item_spend.head()
```

Out[89]:

	fruit_veg	dairy	confectionary	grocery_food	grocery_health_pets	prepared_me
customer_number						
14	11.10	172.58	23.22	56.05	11.28	2:
45	30.21	142.16	106.54	83.42	24.31	5(
52	53.29	5.19	3.29	1.08	12.11	:
61	70.18	55.29	46.39	56.18	45.71	1:
63	22.01	42.11	73.07	13.54	25.08	1:

In [90]:
```python
item_spend.describe()
```

Out[90]:

	fruit_veg	dairy	confectionary	grocery_food	grocery_health_pets	prepared_meals
count	3000.000000	3000.000000	3000.000000	3000.000000	3000.000000	3000.0000(
mean	69.456163	71.302683	57.347793	60.007530	60.909867	35.48057
std	70.499654	57.966265	55.959350	57.682533	69.805023	41.24047
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.0000(
25%	22.695000	31.390000	21.070000	21.087500	18.145000	8.7000(
50%	50.935000	56.875000	42.290000	44.030000	39.075000	23.0950(
75%	93.405000	95.327500	75.125000	80.922500	77.250000	47.3300(
max	1262.970000	708.040000	614.370000	1017.070000	884.450000	454.2900(

8) Check the Pearson correlations in 'item_spend' dataset

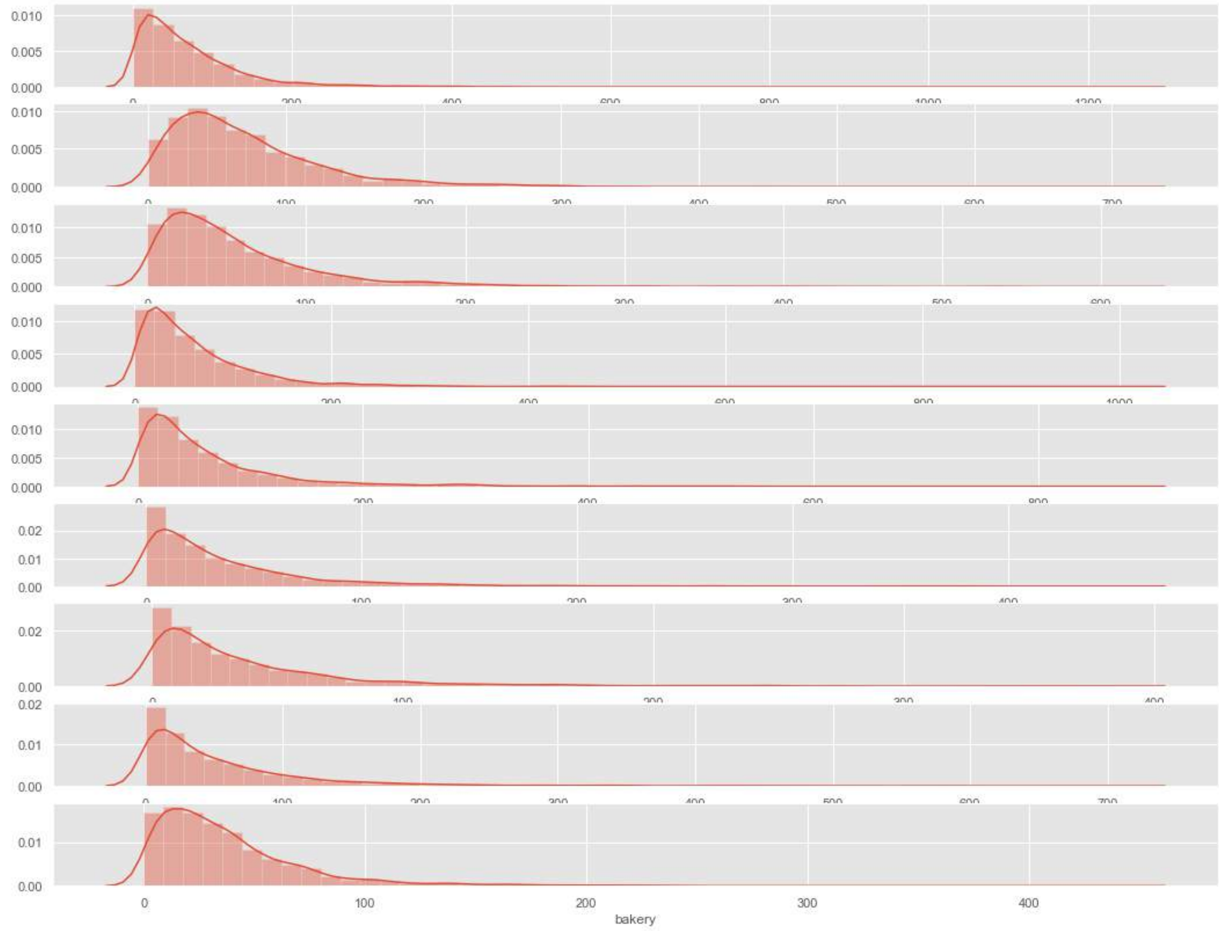
In [91]:
```
corr = item_spend.corr()
corr
```

Out[91]:

	fruit_veg	dairy	confectionary	grocery_food	grocery_health_pets	prepare
fruit_veg	1.000000	0.629616	0.465602	0.640928	0.490674	(
dairy	0.629616	1.000000	0.610100	0.651492	0.557530	(
confectionary	0.465602	0.610100	1.000000	0.574594	0.577793	(
grocery_food	0.640928	0.651492	0.574594	1.000000	0.581256	(
grocery_health_pets	0.490674	0.557530	0.577793	0.581256	1.000000	(
prepared_meals	0.449565	0.509525	0.471222	0.481753	0.454817	
frozen	0.420811	0.531251	0.574330	0.569792	0.529290	(
meat	0.550687	0.492942	0.440183	0.556404	0.450822	(
bakery	0.447188	0.608368	0.542105	0.522479	0.439587	(

9) Check the distribution of each features

In [92]:
```
plt.figure(figsize=(18,14))
plt.subplot(9,1,1); sns.distplot(item_spend['fruit_veg'])
plt.subplot(9,1,2); sns.distplot(item_spend['dairy'])
plt.subplot(9,1,3); sns.distplot(item_spend['confectionary'])
plt.subplot(9,1,4); sns.distplot(item_spend['grocery_food'])
plt.subplot(9,1,5); sns.distplot(item_spend['grocery_health_pets'])
plt.subplot(9,1,6); sns.distplot(item_spend['prepared_meals'])
plt.subplot(9,1,7); sns.distplot(item_spend['frozen'])
plt.subplot(9,1,8); sns.distplot(item_spend['meat'])
plt.subplot(9,1,9); sns.distplot(item_spend['bakery'])
plt.show()
```



In [93]:
```
item_spend_ori = item_spend.copy()
item_spend_ori.head()
```

Out[93]:

	fruit_veg	dairy	confectionary	grocery_food	grocery_health_pets	prepared_me
customer_number						
14	11.10	172.58	23.22	56.05	11.28	2:
45	30.21	142.16	106.54	83.42	24.31	5(
52	53.29	5.19	3.29	1.08	12.11	:
61	70.18	55.29	46.39	56.18	45.71	1:
63	22.01	42.11	73.07	13.54	25.08	1:

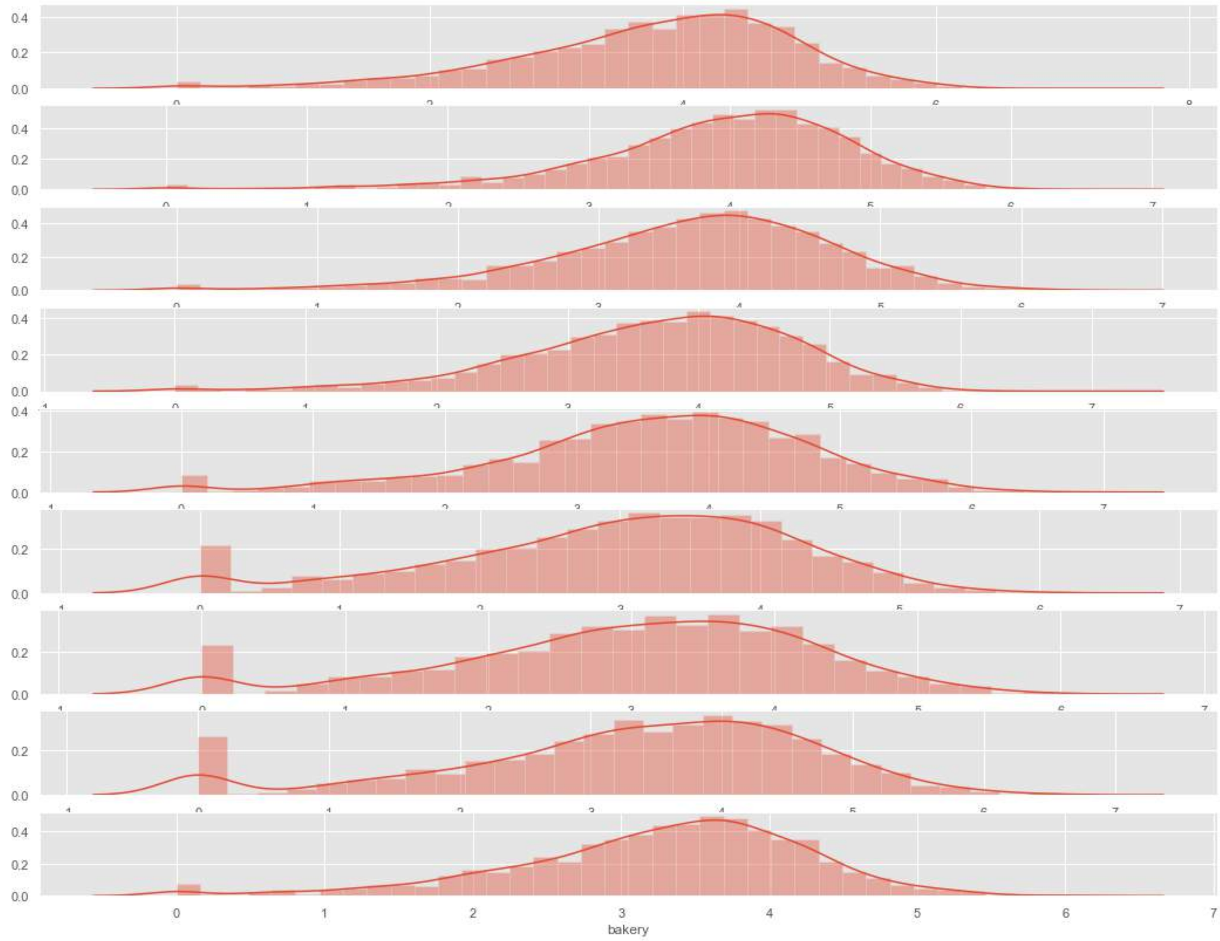
10) Applying the log1p transformation to make the data more 'normal'

In [94]:
```
item_spend_log = np.log1p(item_spend)
item_spend_log.head()
```

Out[94]:

	fruit_veg	dairy	confectionary	grocery_food	grocery_health_pets	prepared_
customer_number						
14	2.493205	5.156639	3.187179	4.043928	2.507972	3.1
45	3.440739	4.963963	4.677863	4.435804	3.231200	4.0
52	3.994340	1.822935	1.456287	0.732368	2.573375	1.3
61	4.265212	4.030517	3.858411	4.046204	3.843958	2.5
63	3.135929	3.763755	4.305011	2.676903	3.261169	2.6

In [95]:
```
plt.figure(figsize=(18,14))
plt.subplot(9,1,1); sns.distplot(item_spend_log['fruit_veg'])
plt.subplot(9,1,2); sns.distplot(item_spend_log['dairy'])
plt.subplot(9,1,3); sns.distplot(item_spend_log['confectionary'])
plt.subplot(9,1,4); sns.distplot(item_spend_log['grocery_food'])
plt.subplot(9,1,5); sns.distplot(item_spend_log['grocery_health_pets'])
plt.subplot(9,1,6); sns.distplot(item_spend_log['prepared_meals'])
plt.subplot(9,1,7); sns.distplot(item_spend_log['frozen'])
plt.subplot(9,1,8); sns.distplot(item_spend_log['meat'])
plt.subplot(9,1,9); sns.distplot(item_spend_log['bakery'])
plt.show()
```



In [ ]:

10) Merge spend_habit_rfm_ori and item_spend_ori to df_ori

In [96]:
```
# cus_bas_rfm_ori + item_spend_ori
df_ori = pd.merge(spend_habit_rfm_ori, item_spend_ori, left_on='customer_number',
                  right_on='customer_number', how='inner')
df_ori.head()
```

Out[96]:

	average_item_count	average_basket_spend	average_spend_per_item	Recency
customer_number				
14	9.48	12.07	1.27	1
45	19.85	17.75	0.89	1
52	4.98	3.77	0.76	2
61	13.49	14.81	1.10	3
63	5.85	6.11	1.04	7

In [97]:
```
corr = df_ori.corr()
corr
```

Out[97]:

	average_item_count	average_basket_spend	average_spend_per_item	Re
average_item_count	1.000000	0.915069	-0.190769	0.0
average_basket_spend	0.915069	1.000000	0.137865	0.0
average_spend_per_item	-0.190769	0.137865	1.000000	0.0
Recency	0.091122	0.096117	0.007386	1.0
Frequency	-0.372985	-0.368888	-0.007982	-0.2
Monetary	0.228918	0.315238	0.183322	-0.2
fruit_veg	0.383803	0.333912	-0.165218	-0.1
dairy	0.305094	0.245811	-0.185242	-0.2
confectionary	0.358659	0.289181	-0.194968	-0.1
grocery_food	0.434644	0.378350	-0.170010	-0.1
grocery_health_pets	0.406408	0.387599	-0.093308	-0.1
prepared_meals	0.299873	0.288395	-0.078288	-0.1
frozen	0.350354	0.306310	-0.134664	-0.1
meat	0.334297	0.340579	-0.054551	-0.1
bakery	0.205458	0.143935	-0.177941	-0.1

11) Merge spend_habit_rfm_log and item_spend_log to df_log

In [98]:

```
# cus_bas_rfm + item_spend_log
df_log = pd.merge(spend_habit_rfm_log, item_spend_log, left_on='customer_numbe
r',
                  right_on='customer_number', how='inner')
df_log.head()
```

Out[98]:

	average_item_count	average_basket_spend	average_spend_per_item	Recency
customer_number				
14	2.349469	2.570320	0.819780	0.693147
45	3.037354	2.931194	0.636577	0.693147
52	1.788421	1.562346	0.565314	1.098612
61	2.673459	2.760643	0.741937	1.386294
63	1.924249	1.961502	0.712950	2.079442

In [99]:

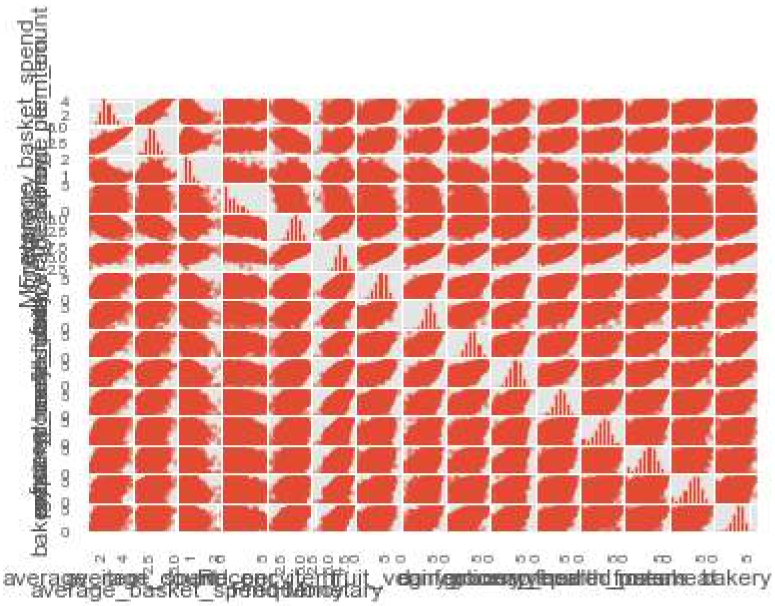
```
round(df_log.describe(),2)
```

Out[99]:

	average_item_count	average_basket_spend	average_spend_per_item	Recency	Frequency
count	3000.00	3000.00	3000.00	3000.00	3000.00
mean	2.34	2.59	0.85	1.21	3.95
std	0.54	0.55	0.19	1.21	0.74
min	0.79	0.90	0.44	0.00	0.69
25%	1.96	2.20	0.73	0.00	3.50
50%	2.28	2.55	0.81	1.10	3.99
75%	2.67	2.91	0.93	1.95	4.47
max	4.52	5.03	2.19	5.11	5.93

In [100]:

```
scatter = pd.plotting.scatter_matrix(df_log)
```



In [ ]:

2. A Customer Base Summary section

In [101]:
```
df_ori.head()
```

Out[101]:

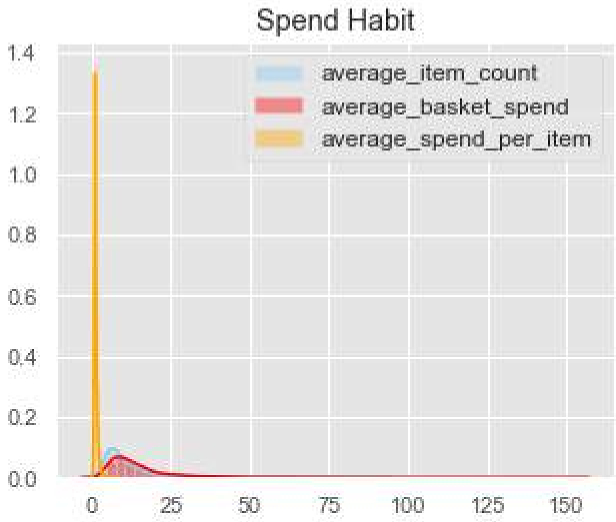
	average_item_count	average_basket_spend	average_spend_per_item	Recency
customer_number				
14	9.48	12.07	1.27	1
45	19.85	17.75	0.89	1
52	4.98	3.77	0.76	2
61	13.49	14.81	1.10	3
63	5.85	6.11	1.04	7

In [102]:
```
df_ori.describe()
```

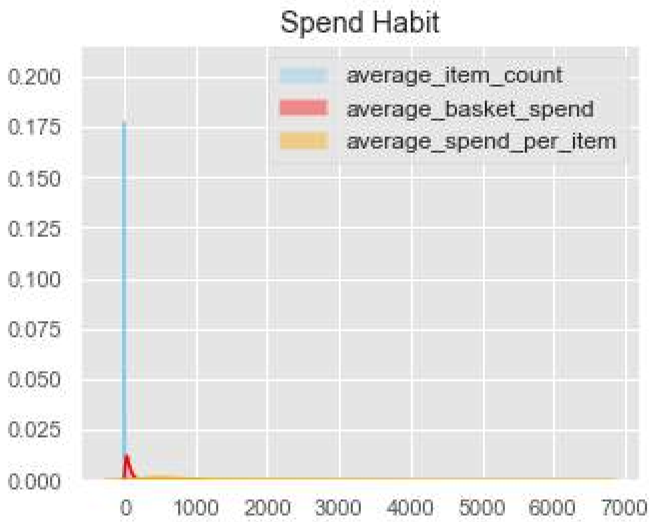
Out[102]:

	average_item_count	average_basket_spend	average_spend_per_item	Recency	Freque
count	3000.000000	3000.000000	3000.000000	3000.000000	3000.000
mean	11.273407	14.801243	1.394923	8.121333	65.182
std	8.538014	11.161381	0.567371	20.938531	47.464
min	1.200000	1.460000	0.560000	0.000000	1.000
25%	6.117500	8.037500	1.070000	0.000000	32.000
50%	8.730000	11.770000	1.250000	2.000000	53.000
75%	13.390000	17.440000	1.530000	6.000000	86.000
max	90.750000	152.620000	7.920000	164.000000	374.000

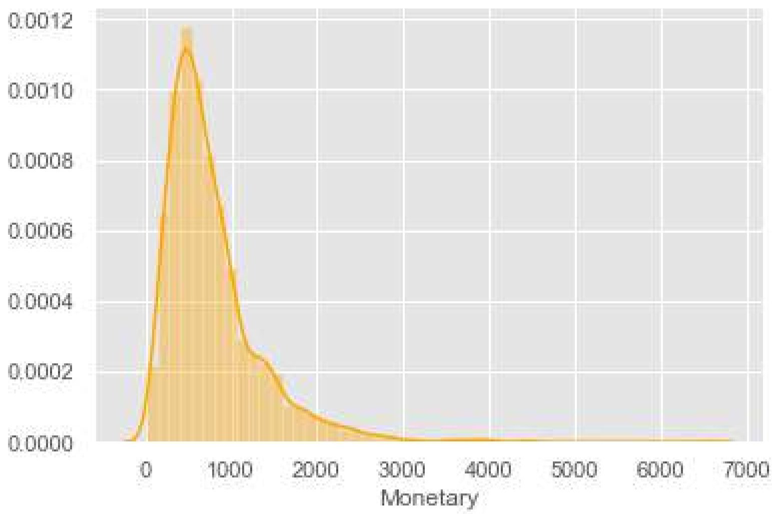
In [103]:
```
fig = plt.figure(figsize=(5,4))
sns.distplot(df_ori.average_item_count,color='skyblue',label='average_item_coun
t')
sns.distplot(df_ori.average_basket_spend,color='red',label='average_basket_spen
d')
sns.distplot(df_ori.average_spend_per_item,color='orange',label='average_spend_
per_item')

plt.legend(prop={'size': 12})
plt.title('Spend Habit')
plt.xlabel('')
plt.show()
```



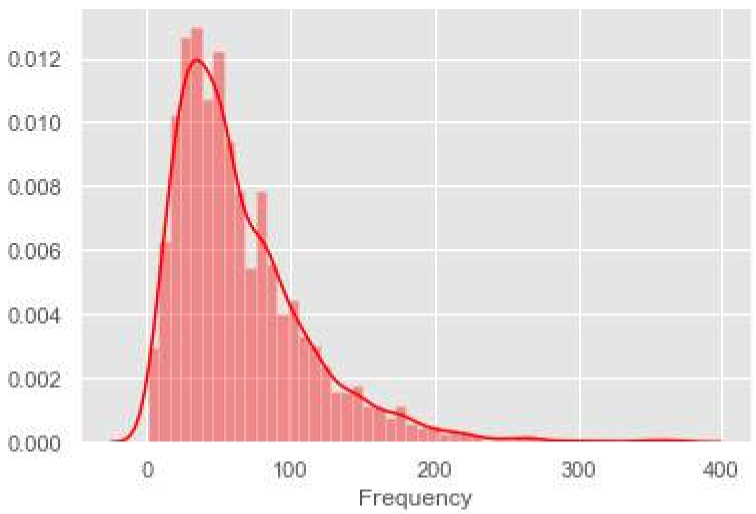
In [104]:
```
fig = plt.figure(figsize=(5,4))
sns.distplot(df_ori.Recency,color='skyblue',label='average_item_count')
sns.distplot(df_ori.Frequency,color='red',label='average_basket_spend')
sns.distplot(df_ori.Monetary,color='orange',label='average_spend_per_item')

plt.legend(prop={'size': 12})
plt.title('Spend Habit')
plt.xlabel('')
plt.show()
```



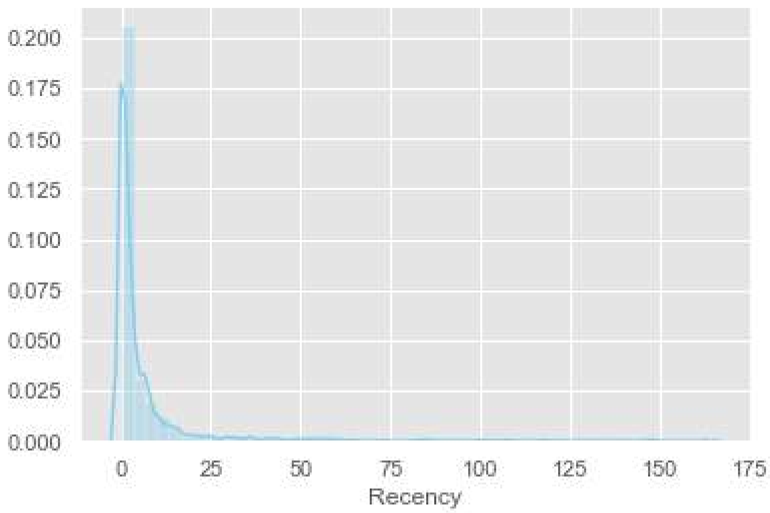
In [105]:
```
sns.distplot(df_ori.Monetary,color='orange',label='average_spend_per_item')
```
Out[105]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x15da3442a90>
```



In [106]:
```
sns.distplot(df_ori.Frequency,color='red',label='average_basket_spend')
```
Out[106]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x15da3251dd8>
```

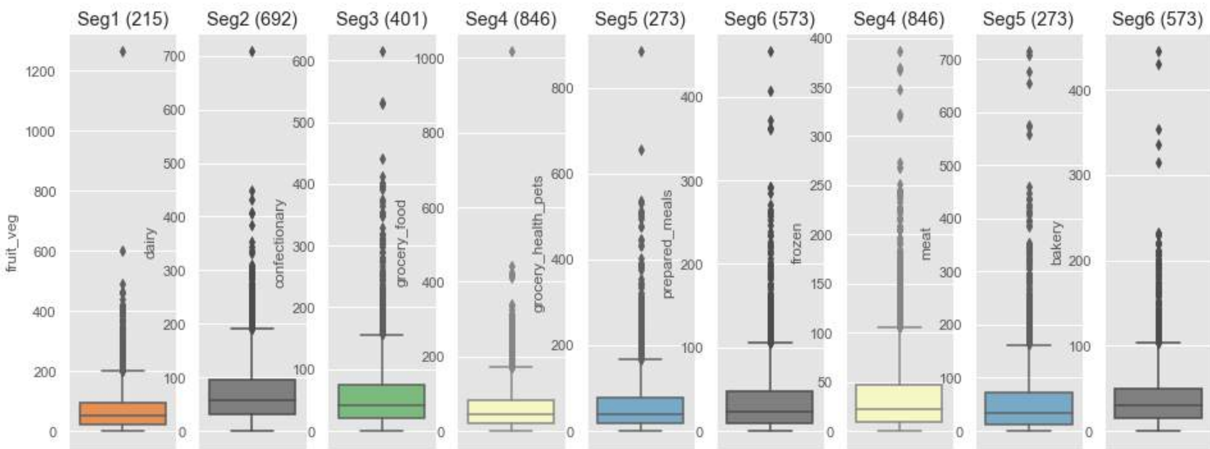


In [107]:
```
sns.distplot(df_ori.Recency,color='skyblue',label='average_item_count')
```
Out[107]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x15da37b5c18>
```



In [108]:
```
figure, (ax1, ax2, ax3, ax4, ax5, ax6, ax7, ax8, ax9) = plt.subplots(1, 9)
figure.set_size_inches(16,6)

sns.boxplot(df_ori.fruit_veg, ax=ax1, orient = 'v', palette='Oranges'
           ).set_title('Seg1 (215)')
sns.boxplot(df_ori.dairy, ax=ax2, orient = 'v', palette='binary'
           ).set_title('Seg2 (692)')
sns.boxplot(df_ori.confectionary, ax=ax3, orient = 'v', palette='Greens'
           ).set_title('Seg3 (401)')
sns.boxplot(df_ori.grocery_food, ax=ax4, orient = 'v', palette='Spectral'
           ).set_title('Seg4 (846)')
sns.boxplot(df_ori.grocery_health_pets, ax=ax5, orient = 'v', palette='Blues'
           ).set_title('Seg5 (273)')
sns.boxplot(df_ori.prepared_meals, ax=ax6, orient = 'v', palette='gist_gray'
           ).set_title('Seg6 (573)')
sns.boxplot(df_ori.frozen, ax=ax7, orient = 'v', palette='Spectral'
           ).set_title('Seg4 (846)')
sns.boxplot(df_ori.meat, ax=ax8, orient = 'v', palette='Blues'
           ).set_title('Seg5 (273)')
sns.boxplot(df_ori.bakery, ax=ax9, orient = 'v', palette='gist_gray'
           ).set_title('Seg6 (573)')
```

Out[108]:
```
Text(0.5, 1.0, 'Seg6 (573)')
```

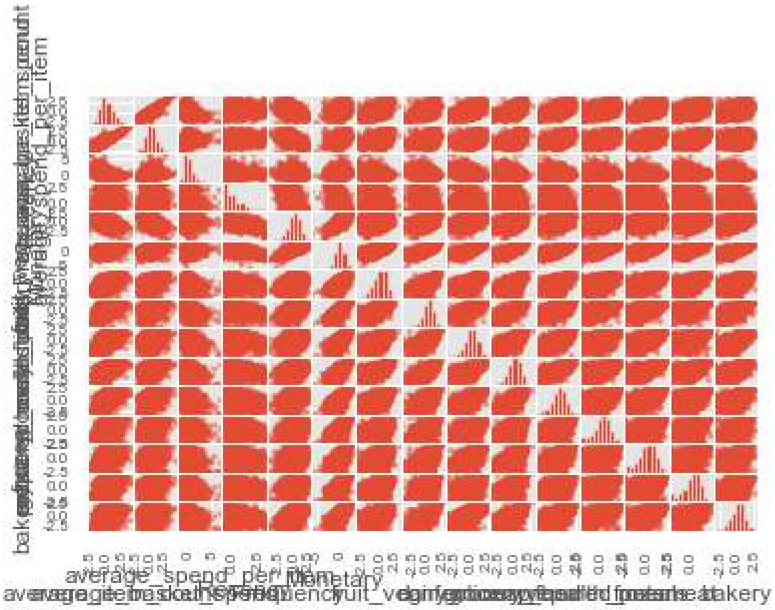


In [ ]:

3. Rescaling to remove the units

In [109]:
```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
df_scaled = scaler.fit_transform(df_log)

# transform into a dataframe
df_scaled = pd.DataFrame(df_scaled, index=df_log.index, columns=df_log.columns)
```

In [110]:
```
scatter = pd.plotting.scatter_matrix(df_scaled)
```



In [111]:
```
round(df_scaled.describe(),2)
```

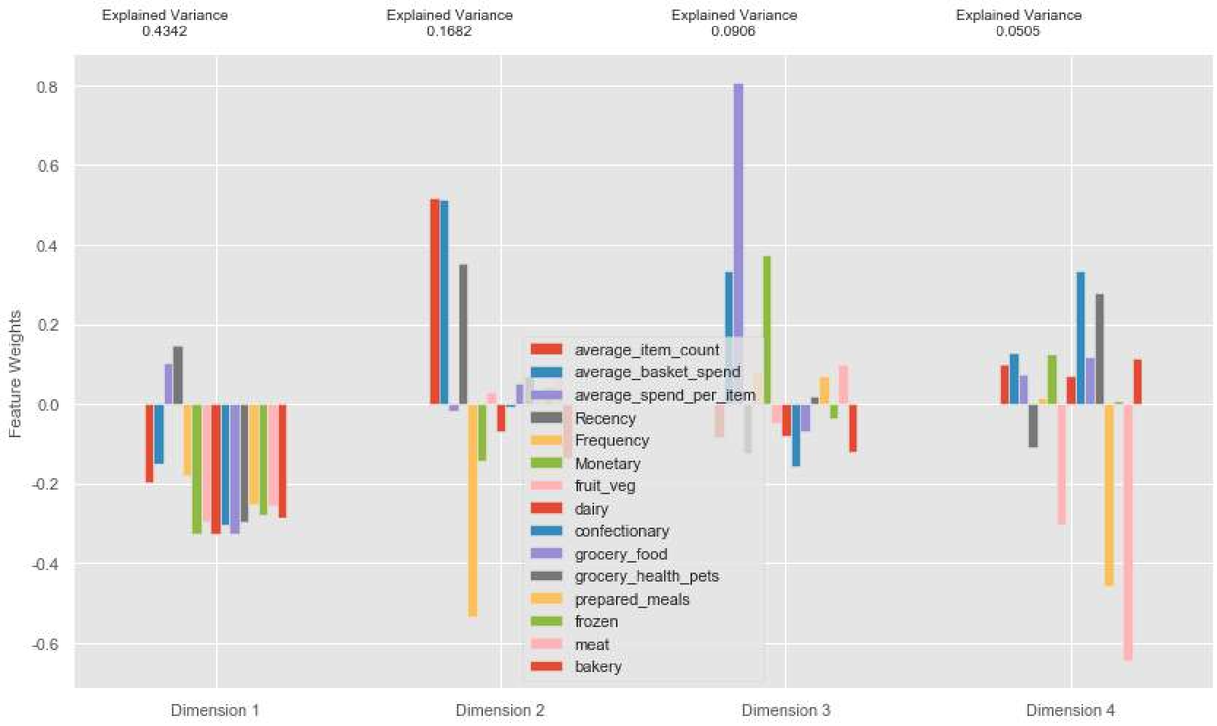
Out[111]:

	average_item_count	average_basket_spend	average_spend_per_item	Recency	Frequency
count	3000.00	3000.00	3000.00	3000.00	3000.00
mean	0.00	-0.00	0.00	-0.00	0.00
std	1.00	1.00	1.00	1.00	1.00
min	-2.86	-3.05	-2.10	-1.00	-4.40
25%	-0.70	-0.70	-0.64	-1.00	-0.61
50%	-0.12	-0.08	-0.21	-0.09	0.06
75%	0.59	0.58	0.39	0.61	0.70
max	4.00	4.40	6.88	3.22	2.68

In [ ]:

4. Feature Engineering

In [112]:
```
from sklearn.decomposition import PCA
pca = PCA(n_components=4)
fit = pca.fit(df_scaled)

#-- import a helpful set of functions to ease displaying results..
import renders as rs

#-- Generate a PCA results plot
pca_results = rs.pca_results(df_scaled, pca)
```



In [ ]:

5. Selecting our final Features

In [113]:
```
pca.explained_variance_ratio_

count = 0
explained = 0
for i in pca.explained_variance_ratio_:
    if explained < 0.70:
        explained += i
        count += 1
    else:
        break
print(explained)
print(count)
```

```
0.7435166512965233
4
```

In [114]:
```
pca = PCA(n_components=4)
pca.fit(df_scaled)
reduced_data = pca.transform(df_scaled)
reduced_data = pd.DataFrame(reduced_data)
```

In [ ]:

```

```

6. K-Means Clustering Algorithm

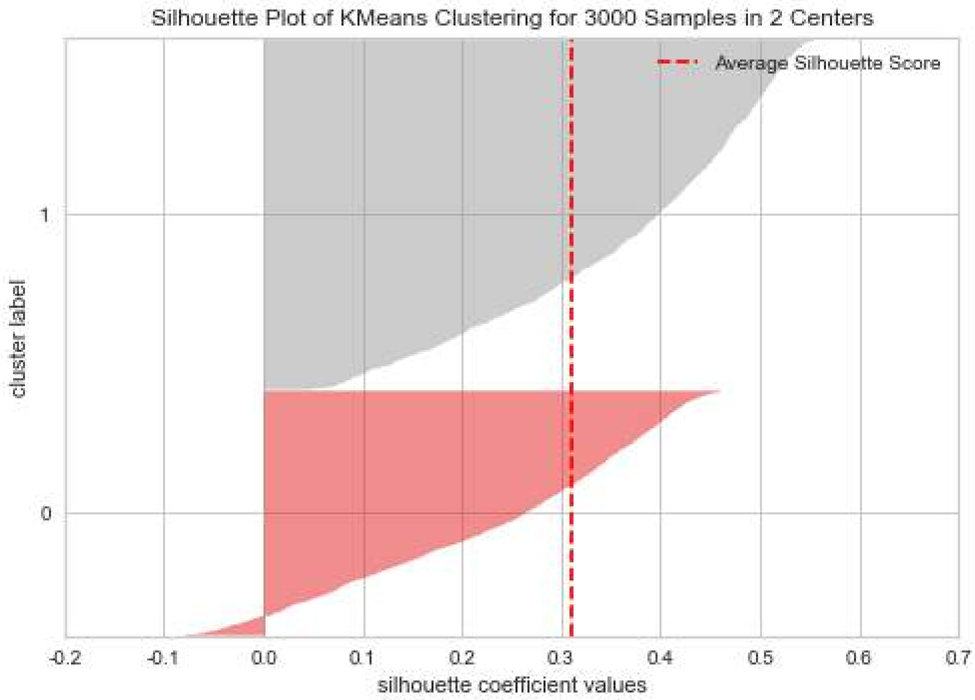
In [115]:

```
from sklearn.cluster import KMeans
from sklearn.metrics import silhouette_score
from yellowbrick.cluster import SilhouetteVisualizer
```

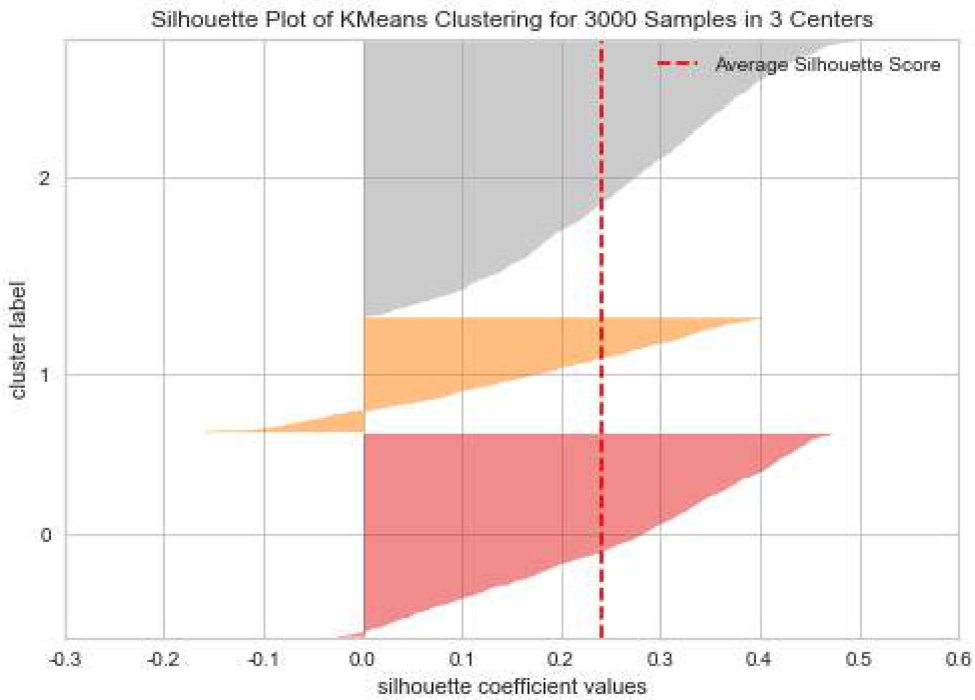
In [116]:

```python
# Finding the right number of segments
for k in range(2,11):
    clusterer = KMeans(n_clusters=k, random_state=42).fit(reduced_data)
    preds = clusterer.predict(reduced_data)
    centers = clusterer.cluster_centers_
    score = round(silhouette_score(reduced_data, preds, metric='euclidean'),3)

    print("For n_clusters = {}.The average silhouette_score is : {})".format(k,
score))
    visualizer = SilhouetteVisualizer(clusterer, n_clusters=k)
    visualizer.fit(reduced_data)
    visualizer.show()
```

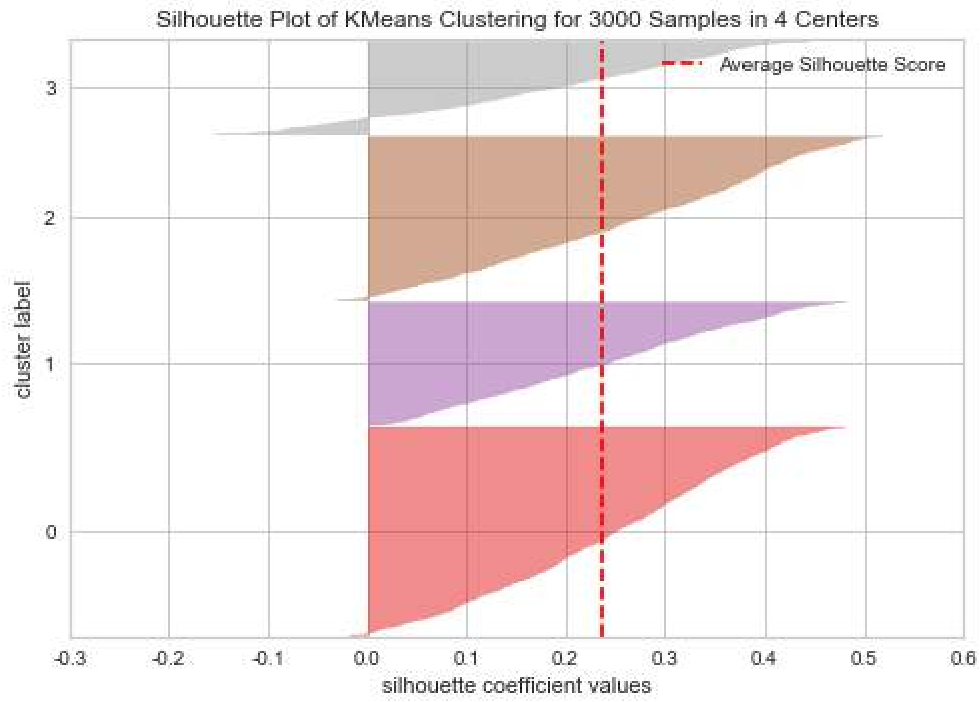
For n_clusters = 2.The average silhouette_score is : 0.31)



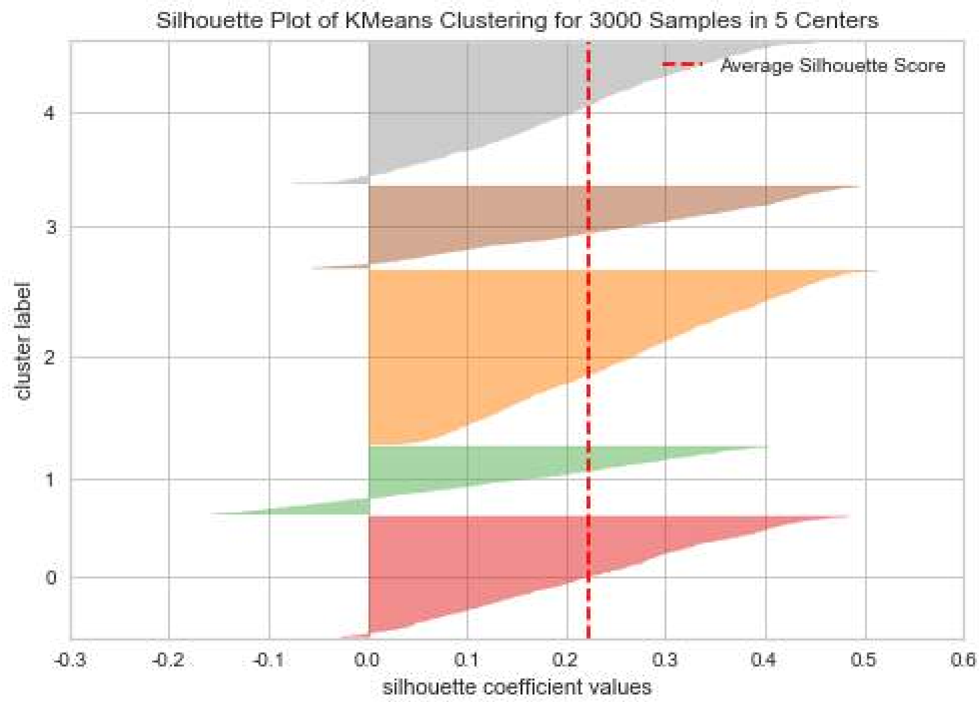
For n_clusters = 3.The average silhouette_score is : 0.241)



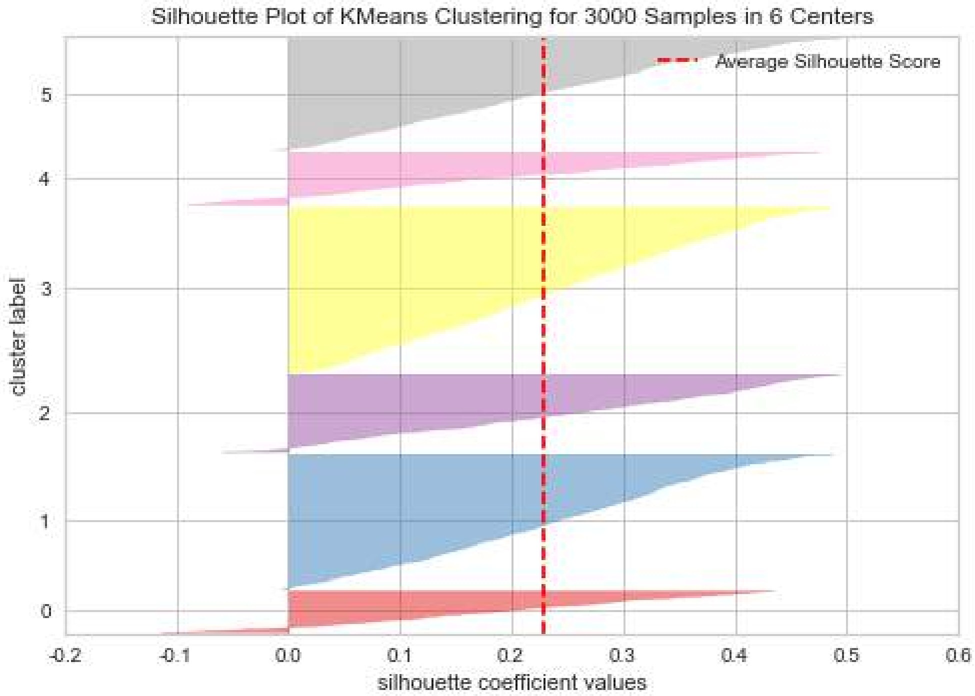
For n_clusters = 4.The average silhouette_score is : 0.236)



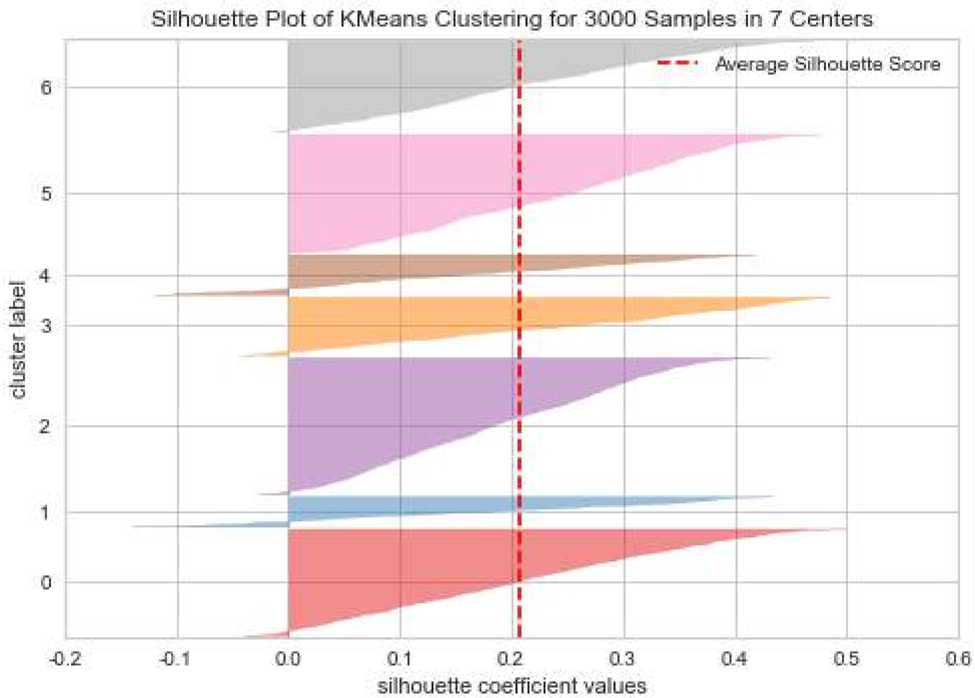
For n_clusters = 5.The average silhouette_score is : 0.222)



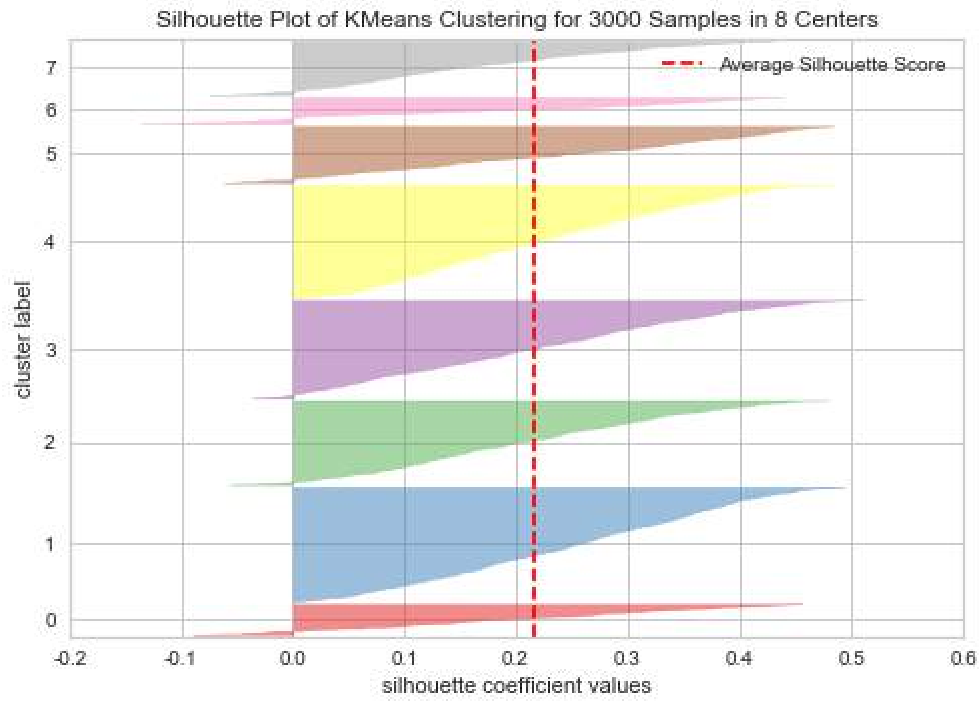
For n_clusters = 6.The average silhouette_score is : 0.228)



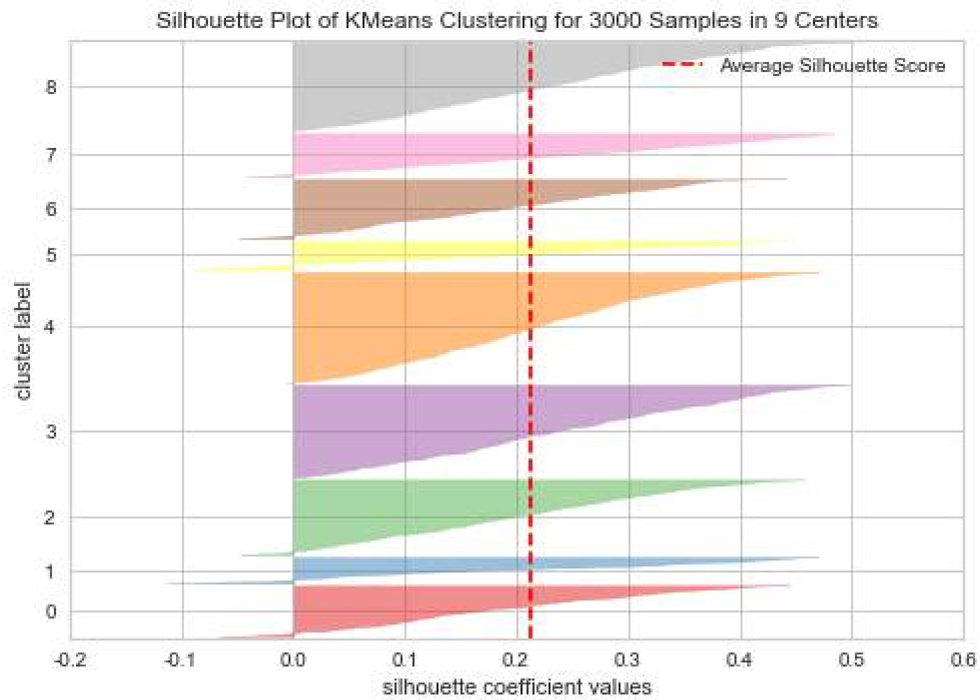

```
For n_clusters = 7.The average silhouette_score is : 0.208)
```



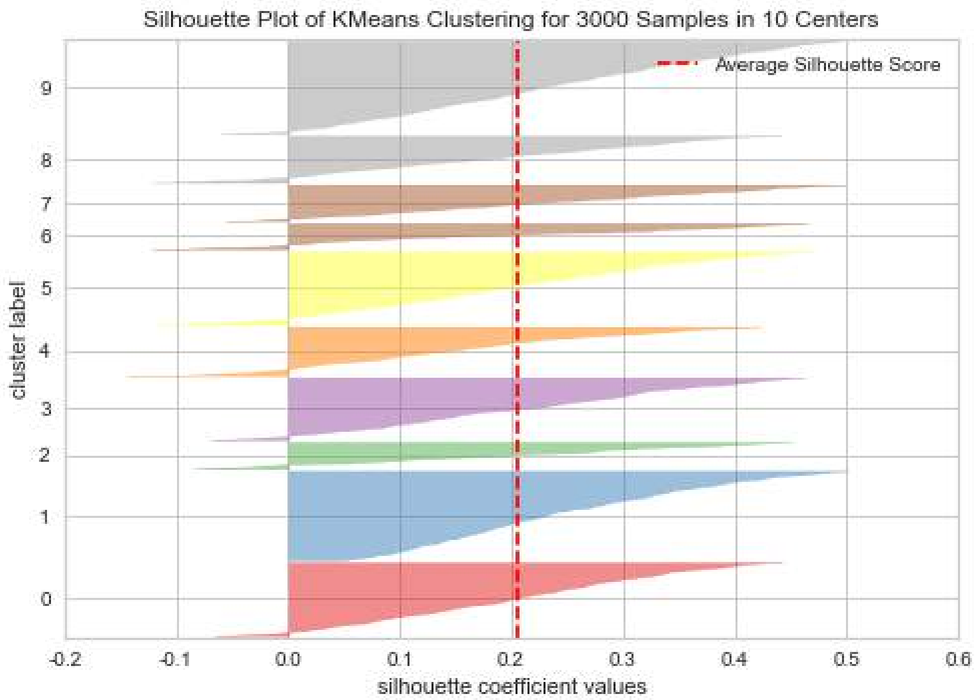

```
For n_clusters = 8.The average silhouette_score is : 0.216)
```




```
For n_clusters = 9.The average silhouette_score is : 0.212)
```



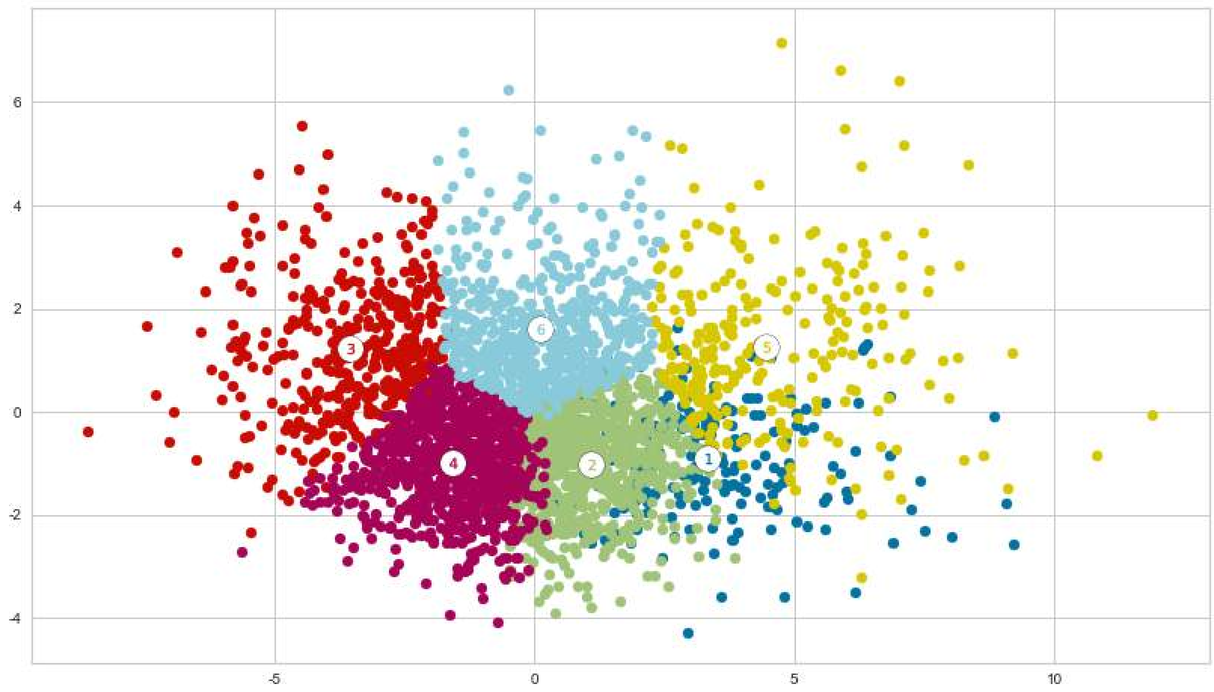

```
For n_clusters = 10.The average silhouette_score is : 0.205)
```




```
In [ ]:
```

7. K-Means Clustering Visualisation

In [117]:
```
clusterer = KMeans(n_clusters=6, random_state=42).fit(reduced_data)
preds = clusterer.predict(reduced_data)
centres = clusterer.cluster_centers_

#-- Put the predictions into a pandas dataframe format
assignments = pd.DataFrame(preds, columns = ['Cluster'])

#-- Put the predictions into a pandas dataframe format
plot_data = pd.concat([assignments, reduced_data], axis = 1)

#-- Color the points based on assigned cluster (n.b scatter will do this for us
automatically)
plt.rcParams['figure.figsize'] = (14.0, 8.0)


for i, c in plot_data.groupby('Cluster'):
    plt.scatter(c[0], c[1])

#-- Plot where the cluster centers are
for i, c in enumerate(centres):
    plt.scatter(x = c[0], y = c[1], color = 'white', edgecolors = 'black', mark
er = 'o', s=300);
    plt.scatter(x = c[0], y = c[1], marker='${}$'.format(i+1), alpha = 1, s=50
);
```



In [ ]:

8. Recovering Segment Archetypes in the original variables

In [118]:
```
log_centres = pca.inverse_transform(centres)

# TODO: Exponentiate the centres
true_centres = np.exp(log_centres)

#-- Display the true centres
segments = ['Segment {}'.format(i+1) for i in range(0, len(centres))]
true_centres = pd.DataFrame(np.round(true_centres), columns = df_log.columns)
true_centres.index = segments
print(true_centres)
```

```
           average_item_count  average_basket_spend  average_spend_per_item  \
Segment 1                 0.0                   1.0                     9.0
Segment 2                 1.0                   0.0                     1.0
Segment 3                 4.0                   4.0                     1.0
Segment 4                 1.0                   1.0                     1.0
Segment 5                 1.0                   1.0                     1.0
Segment 6                 2.0                   2.0                     1.0

           Recency  Frequency  Monetary  fruit_veg  dairy  confectionary  \
Segment 1      1.0        1.0       1.0        0.0    0.0            0.0
Segment 2      1.0        1.0       1.0        1.0    1.0            1.0
Segment 3      1.0        1.0       3.0        3.0    3.0            3.0
Segment 4      1.0        2.0       2.0        2.0    2.0            2.0
Segment 5      3.0        0.0       0.0        0.0    0.0            0.0
Segment 6      2.0        0.0       1.0        1.0    1.0            1.0

           grocery_food  grocery_health_pets  prepared_meals  frozen  meat  \
Segment 1           0.0                  0.0             0.0     0.0   0.0
Segment 2           1.0                  1.0             1.0     1.0   1.0
Segment 3           3.0                  3.0             2.0     3.0   3.0
Segment 4           2.0                  1.0             2.0     1.0   2.0
Segment 5           0.0                  0.0             0.0     0.0   0.0
Segment 6           1.0                  1.0             1.0     1.0   1.0

           bakery
Segment 1     0.0
Segment 2     1.0
Segment 3     2.0
Segment 4     2.0
Segment 5     0.0
Segment 6     1.0
```

In [ ]:

9. Creating Profiles

In [119]:

```
final_assigments = pd.concat([assignments, df_ori], axis = 1)

#-- Create a loop that describes summary statistics for each segment
for c, d in final_assigments.groupby('Cluster'):
    print('SEGMENT', c+1)
    display(d.describe())
```

```
SEGMENT 1.0
```

	Cluster	average_item_count	average_basket_spend	average_spend_per_item	Recency	F
count	215.0	41.000000	41.000000	41.000000	41.000000	4
mean	0.0	14.242195	18.270244	1.380488	4.463415	5
std	0.0	10.939258	12.894356	0.507691	7.043073	3
min	0.0	4.450000	3.610000	0.770000	0.000000	
25%	0.0	7.130000	9.770000	1.140000	0.000000	3
50%	0.0	10.960000	15.560000	1.290000	2.000000	4
75%	0.0	17.610000	20.620000	1.500000	6.000000	6
max	0.0	55.150000	65.800000	3.650000	36.000000	18

```
SEGMENT 2.0
```

	Cluster	average_item_count	average_basket_spend	average_spend_per_item	Recency	
count	692.0	162.000000	162.000000	162.000000	162.000000	1
mean	1.0	12.361420	14.399198	1.193457	9.302469	
std	0.0	10.031834	12.084743	0.315331	25.969186	
min	1.0	2.530000	3.170000	0.590000	0.000000	
25%	1.0	6.700000	7.442500	1.022500	0.000000	
50%	1.0	9.670000	12.040000	1.160000	1.000000	
75%	1.0	14.375000	15.945000	1.300000	5.750000	
max	1.0	83.250000	100.480000	2.700000	161.000000	2

```
SEGMENT 3.0
```

	Cluster	average_item_count	average_basket_spend	average_spend_per_item	Recency	
count	401.0	74.000000	74.000000	74.000000	74.000000	
mean	2.0	12.997568	15.823919	1.225811	8.040541	
std	0.0	9.317250	12.121332	0.347701	24.864256	
min	2.0	4.590000	4.610000	0.700000	0.000000	
25%	2.0	6.630000	8.035000	1.010000	0.000000	
50%	2.0	10.175000	11.880000	1.175000	1.000000	
75%	2.0	16.452500	18.967500	1.357500	3.750000	
max	2.0	44.710000	59.060000	2.940000	146.000000	2

```
SEGMENT 4.0
```

	Cluster	average_item_count	average_basket_spend	average_spend_per_item	Recency	I
count	846.0	148.000000	148.000000	148.000000	148.000000	1
mean	3.0	11.436014	14.105405	1.244189	7.547297	
std	0.0	7.323582	9.516259	0.358405	17.640142	
min	3.0	2.140000	2.070000	0.620000	0.000000	
25%	3.0	7.030000	7.967500	1.020000	0.000000	
50%	3.0	9.460000	11.005000	1.175000	1.000000	
75%	3.0	13.527500	17.245000	1.365000	6.000000	
max	3.0	52.960000	58.020000	2.920000	108.000000	2

SEGMENT 5.0

	Cluster	average_item_count	average_basket_spend	average_spend_per_item	Recency	Fr
count	273.0	44.000000	44.000000	44.000000	44.000000	4
mean	4.0	14.318864	17.841591	1.242955	6.818182	4
std	0.0	13.791487	19.642670	0.259531	8.734515	2
min	4.0	4.320000	4.570000	0.720000	0.000000	1
25%	4.0	7.155000	9.212500	1.057500	1.000000	2
50%	4.0	10.460000	11.900000	1.220000	3.000000	4
75%	4.0	15.042500	17.632500	1.400000	8.000000	6
max	4.0	84.270000	122.160000	2.060000	33.000000	11

SEGMENT 6.0

	Cluster	average_item_count	average_basket_spend	average_spend_per_item	Recency	I
count	573.0	99.000000	99.000000	99.000000	99.000000	
mean	5.0	12.845960	15.773535	1.213333	7.626263	
std	0.0	11.540094	15.660396	0.485187	19.139971	
min	5.0	3.340000	3.000000	0.740000	0.000000	
25%	5.0	7.295000	8.275000	0.980000	0.000000	
50%	5.0	9.740000	11.650000	1.160000	2.000000	
75%	5.0	13.615000	16.905000	1.340000	6.000000	
max	5.0	90.750000	116.950000	5.390000	136.000000	2

9-1. Creating separate segment groups

In [120]:
```
clusterer.cluster_centers_
```

Out[120]:
```
array([[ 3.31068845, -0.91519372,  2.25887105,  0.32842058],
       [ 1.07953272, -1.01377763, -0.59529118,  0.00707423],
       [-3.53367494,  1.22387084,  0.35401291,  0.13904307],
       [-1.5648163 , -0.99927423,  0.02003483, -0.11691016],
       [ 4.45004631,  1.24500303, -0.59869795, -0.14627162],
       [ 0.11717441,  1.59341838, -0.12073277,  0.01322179]])
```

In [121]:
```
clusterer.labels_
```

Out[121]:
```
array([3, 5, 4, ..., 1, 5, 5])
```

In [122]:
```
mydict = {i: np.where(clusterer.labels_ == i)[0] for i in range(clusterer.n_clu
sters)}
```

In [123]:
```
mydict
```

```
Out[123]: {0: array([  56,   81,  131,  140,  180,  260,  267,  432,  439,  465,  556,
                  578,  655,  667,  672,  735,  759,  815,  833,  847,  849,  858,
                  872,  882,  888,  916,  928,  934,  937,  941,  943,  957,  960,
                  969,  977,  978,  982,  985,  998, 1025, 1045, 1052, 1060, 1067,
                 1068, 1077, 1083, 1112, 1128, 1152, 1161, 1166, 1171, 1180, 1194,
                 1204, 1208, 1211, 1229, 1237, 1238, 1268, 1271, 1285, 1287, 1289,
                 1309, 1319, 1356, 1425, 1426, 1438, 1445, 1455, 1486, 1556, 1598,
                 1615, 1622, 1643, 1679, 1680, 1766, 1770, 1776, 1777, 1778, 1782,
                 1792, 1794, 1820, 1830, 1835, 1870, 1912, 1959, 1974, 1992, 2064,
                 2124, 2137, 2176, 2179, 2180, 2182, 2184, 2192, 2194, 2197, 2201,
                 2202, 2203, 2207, 2209, 2213, 2214, 2219, 2223, 2224, 2226, 2244,
                 2259, 2260, 2261, 2268, 2271, 2287, 2291, 2333, 2351, 2357, 2370,
                 2383, 2386, 2399, 2418, 2429, 2431, 2433, 2445, 2464, 2468, 2508,
                 2528, 2542, 2550, 2561, 2565, 2578, 2608, 2610, 2622, 2623, 2639,
                 2651, 2661, 2664, 2673, 2681, 2692, 2697, 2699, 2709, 2710, 2718,
                 2721, 2723, 2724, 2725, 2733, 2734, 2742, 2751, 2756, 2770, 2779,
                 2788, 2791, 2794, 2797, 2809, 2821, 2825, 2829, 2831, 2837, 2843,
                 2848, 2865, 2867, 2869, 2871, 2873, 2886, 2891, 2894, 2904, 2908,
                 2911, 2912, 2920, 2929, 2935, 2937, 2940, 2941, 2944, 2952, 2954,
                 2970, 2971, 2977, 2983, 2994, 2995], dtype=int64),
           1: array([   4,   11,   12,   15,   21,   29,   30,   31,   33,   36,   38,
                   50,   60,   61,   63,   64,   71,   80,   82,   85,   94,   98,
                   99,  101,  105,  108,  110,  111,  112,  118,  119,  123,  133,
                  138,  141,  145,  149,  151,  153,  156,  158,  166,  167,  170,
                  186,  198,  204,  207,  208,  209,  212,  215,  219,  226,  230,
                  233,  237,  239,  243,  244,  247,  251,  252,  255,  258,  265,
                  274,  276,  277,  279,  284,  286,  291,  297,  303,  309,  312,
                  313,  314,  316,  318,  319,  325,  337,  345,  346,  350,  360,
                  367,  368,  369,  377,  378,  383,  384,  388,  395,  396,  399,
                  401,  406,  407,  420,  423,  434,  437,  441,  442,  444,  452,
                  453,  456,  469,  472,  475,  480,  481,  486,  488,  494,  496,
                  504,  511,  513,  514,  518,  525,  526,  534,  535,  537,  540,
                  541,  544,  550,  553,  555,  560,  564,  565,  567,  572,  579,
                  581,  587,  590,  595,  603,  605,  606,  608,  609,  611,  624,
                  626,  628,  631,  637,  648,  650,  653,  657,  658,  659,  682,
                  683,  686,  687,  697,  705,  709,  710,  711,  713,  717,  719,
                  732,  738,  745,  749,  753,  755,  756,  762,  767,  773,  775,
                  779,  782,  787,  790,  791,  795,  796,  797,  801,  807,  810,
                  814,  818,  827,  834,  841,  843,  863,  876,  887,  890,  897,
                  900,  902,  904,  905,  906,  907,  918,  919,  921,  922,  926,
                  932,  935,  940,  953,  954,  955,  971,  974,  983,  984,  990,
                  992,  993,  994,  995,  996, 1000, 1001, 1005, 1012, 1016, 1017,
                 1018, 1023, 1026, 1027, 1028, 1031, 1035, 1036, 1037, 1039, 1041,
                 1043, 1055, 1056, 1058, 1063, 1064, 1070, 1071, 1073, 1074, 1082,
                 1084, 1087, 1094, 1096, 1111, 1115, 1123, 1126, 1127, 1136, 1137,
                 1144, 1149, 1154, 1155, 1156, 1157, 1158, 1164, 1168, 1174, 1182,
                 1185, 1186, 1195, 1201, 1206, 1210, 1214, 1217, 1222, 1233, 1239,
                 1241, 1242, 1246, 1249, 1253, 1258, 1260, 1262, 1263, 1266, 1269,
                 1274, 1275, 1276, 1278, 1281, 1282, 1283, 1286, 1288, 1292, 1296,
                 1298, 1299, 1306, 1311, 1320, 1327, 1328, 1331, 1336, 1339, 1341,
                 1342, 1351, 1362, 1369, 1370, 1378, 1379, 1389, 1397, 1398, 1403,
                 1407, 1412, 1417, 1422, 1423, 1424, 1433, 1437, 1442, 1444, 1449,
                 1452, 1456, 1458, 1464, 1465, 1471, 1476, 1481, 1487, 1492, 1493,
                 1500, 1502, 1505, 1508, 1517, 1528, 1531, 1533, 1541, 1544, 1545,
                 1546, 1554, 1558, 1572, 1575, 1580, 1589, 1599, 1602, 1603, 1604,
                 1609, 1613, 1617, 1633, 1636, 1638, 1642, 1645, 1646, 1650, 1651,
                 1652, 1657, 1658, 1662, 1670, 1671, 1672, 1675, 1678, 1689, 1690,
                 1694, 1697, 1700, 1704, 1710, 1717, 1723, 1726, 1730, 1732, 1736,
                 1739, 1743, 1753, 1757, 1759, 1765, 1771, 1773, 1779, 1791, 1796,
                 1803, 1805, 1806, 1817, 1826, 1836, 1853, 1862, 1863, 1886, 1888,
                 1892, 1897, 1902, 1903, 1907, 1920, 1930, 1931, 1935, 1937, 1939,
                 1948, 1949, 1951, 1954, 1964, 1967, 1993, 1996, 1997, 2007, 2012,
```

```
       2020, 2021, 2028, 2031, 2046, 2073, 2075, 2079, 2083, 2087, 2088,
       2091, 2099, 2102, 2103, 2105, 2106, 2107, 2110, 2119, 2125, 2136,
       2146, 2149, 2157, 2164, 2167, 2172, 2177, 2185, 2187, 2204, 2210,
       2212, 2216, 2220, 2221, 2222, 2229, 2249, 2250, 2264, 2272, 2273,
       2279, 2290, 2293, 2300, 2301, 2304, 2309, 2311, 2318, 2326, 2331,
       2334, 2335, 2340, 2352, 2353, 2356, 2358, 2364, 2372, 2373, 2376,
       2379, 2380, 2381, 2385, 2387, 2388, 2397, 2401, 2416, 2422, 2435,
       2443, 2446, 2454, 2460, 2461, 2462, 2469, 2472, 2473, 2480, 2491,
       2492, 2499, 2501, 2502, 2503, 2510, 2514, 2516, 2533, 2537, 2543,
       2549, 2558, 2559, 2563, 2580, 2584, 2586, 2589, 2595, 2596, 2599,
       2604, 2607, 2609, 2620, 2638, 2641, 2659, 2666, 2669, 2671, 2678,
       2680, 2682, 2683, 2684, 2686, 2687, 2689, 2690, 2691, 2693, 2694,
       2704, 2705, 2706, 2712, 2713, 2716, 2719, 2736, 2737, 2738, 2740,
       2743, 2747, 2749, 2753, 2757, 2768, 2773, 2777, 2780, 2782, 2784,
       2786, 2787, 2789, 2795, 2805, 2807, 2808, 2810, 2814, 2818, 2823,
       2824, 2832, 2836, 2841, 2842, 2844, 2845, 2847, 2855, 2858, 2860,
       2864, 2868, 2870, 2872, 2875, 2876, 2877, 2879, 2883, 2890, 2892,
       2893, 2897, 2905, 2906, 2907, 2914, 2915, 2916, 2919, 2921, 2927,
       2928, 2930, 2931, 2932, 2933, 2934, 2938, 2945, 2947, 2948, 2955,
       2957, 2960, 2961, 2963, 2964, 2965, 2972, 2973, 2974, 2975, 2976,
       2980, 2981, 2982, 2984, 2986, 2989, 2990, 2992, 2996, 2997],
      dtype=int64),
2: array([   5,    7,   20,   26,   46,   47,   55,   72,   78,   83,   86,
         87,   91,   95,  115,  116,  129,  135,  136,  144,  172,  183,
        190,  191,  211,  217,  229,  231,  232,  234,  242,  262,  264,
        268,  275,  283,  289,  296,  307,  308,  311,  322,  323,  327,
        329,  340,  341,  342,  356,  374,  380,  381,  389,  391,  394,
        397,  398,  402,  411,  412,  414,  416,  422,  424,  436,  440,
        449,  450,  460,  464,  467,  497,  498,  500,  502,  503,  507,
        529,  532,  539,  558,  559,  582,  584,  592,  601,  604,  613,
        615,  618,  619,  627,  629,  632,  640,  641,  647,  661,  666,
        675,  679,  690,  694,  706,  733,  750,  788,  802,  803,  805,
        808,  817,  830,  837,  838,  844,  848,  851,  852,  856,  865,
        867,  912,  924,  927,  931,  936,  939,  942,  947,  949,  950,
        956,  964,  966,  975,  986,  997,  999, 1003, 1019, 1024, 1040,
       1054, 1059, 1062, 1098, 1104, 1116, 1135, 1140, 1143, 1150, 1160,
       1172, 1177, 1183, 1184, 1193, 1203, 1205, 1207, 1213, 1219, 1225,
       1231, 1236, 1244, 1250, 1251, 1259, 1290, 1301, 1316, 1318, 1330,
       1338, 1344, 1346, 1347, 1355, 1357, 1361, 1363, 1364, 1367, 1374,
       1376, 1377, 1381, 1382, 1391, 1399, 1402, 1408, 1415, 1419, 1420,
       1427, 1429, 1431, 1435, 1436, 1446, 1463, 1470, 1474, 1488, 1490,
       1491, 1503, 1511, 1513, 1518, 1520, 1524, 1527, 1542, 1547, 1548,
       1549, 1553, 1555, 1557, 1562, 1563, 1564, 1571, 1576, 1588, 1591,
       1592, 1593, 1594, 1596, 1597, 1618, 1619, 1623, 1627, 1628, 1630,
       1635, 1637, 1641, 1653, 1654, 1659, 1663, 1676, 1683, 1684, 1691,
       1693, 1695, 1707, 1715, 1718, 1725, 1727, 1735, 1748, 1752, 1761,
       1780, 1781, 1793, 1797, 1799, 1811, 1812, 1816, 1819, 1833, 1838,
       1845, 1846, 1859, 1865, 1869, 1887, 1891, 1904, 1909, 1919, 1922,
       1933, 1940, 1946, 1961, 1963, 1965, 1971, 1983, 1987, 1994, 2002,
       2003, 2004, 2006, 2013, 2014, 2017, 2030, 2033, 2038, 2040, 2043,
       2044, 2052, 2053, 2055, 2057, 2058, 2059, 2061, 2067, 2068, 2080,
       2082, 2086, 2089, 2092, 2094, 2104, 2111, 2114, 2115, 2118, 2120,
       2126, 2127, 2132, 2135, 2140, 2141, 2144, 2147, 2161, 2163, 2165,
       2166, 2174, 2235, 2238, 2246, 2267, 2277, 2284, 2296, 2298, 2305,
       2319, 2338, 2345, 2348, 2363, 2368, 2374, 2377, 2384, 2391, 2393,
       2405, 2406, 2421, 2455, 2458, 2466, 2471, 2477, 2483, 2495, 2497,
       2507, 2509, 2520, 2526, 2544, 2547, 2548, 2551, 2564, 2573, 2574,
       2575, 2594, 2611, 2625, 2628, 2637, 2646, 2652, 2655, 2657, 2717,
       2720, 2775, 2806, 2835, 2969], dtype=int64),
3: array([   0,   10,   14,   16,   19,   24,   25,   28,   35,   37,   40,
         43,   44,   45,   48,   51,   54,   57,   58,   62,   66,   67,
         68,   70,   74,   75,   76,   77,   79,   89,   96,   97,  103,
        113,  114,  117,  122,  127,  128,  130,  134,  139,  142,  143,
```

146, 152, 154, 159, 160, 161, 162, 168, 169, 171, 174,
176, 178, 182, 185, 193, 194, 199, 201, 206, 220, 222,
223, 224, 228, 235, 236, 240, 245, 248, 249, 253, 254,
257, 270, 271, 272, 278, 280, 281, 282, 288, 290, 304,
305, 310, 321, 326, 330, 331, 334, 335, 336, 339, 343,
347, 348, 353, 354, 355, 357, 358, 365, 366, 372, 373,
375, 376, 382, 385, 390, 392, 403, 410, 417, 418, 419,
425, 426, 427, 435, 443, 445, 447, 454, 458, 459, 461,
462, 466, 468, 473, 476, 477, 479, 482, 484, 485, 487,
491, 492, 495, 505, 508, 510, 512, 516, 517, 520, 523,
524, 527, 528, 530, 533, 542, 543, 545, 551, 552, 561,
563, 568, 570, 573, 574, 580, 586, 588, 589, 591, 593,
599, 600, 610, 612, 614, 616, 633, 634, 642, 643, 644,
645, 651, 662, 663, 664, 668, 669, 670, 671, 673, 676,
677, 678, 680, 684, 685, 688, 689, 691, 692, 693, 696,
699, 700, 703, 707, 708, 712, 714, 715, 718, 720, 724,
725, 726, 729, 730, 731, 737, 744, 746, 751, 754, 757,
758, 760, 766, 768, 770, 772, 774, 780, 781, 784, 786,
789, 792, 798, 799, 800, 811, 812, 816, 819, 820, 821,
822, 823, 825, 828, 829, 831, 836, 840, 842, 845, 846,
850, 854, 855, 857, 862, 864, 866, 869, 871, 874, 877,
878, 880, 883, 885, 886, 892, 893, 895, 896, 901, 908,
909, 910, 911, 913, 914, 915, 917, 920, 923, 930, 933,
944, 945, 946, 951, 952, 958, 959, 961, 963, 965, 968,
970, 972, 980, 981, 987, 989, 1002, 1004, 1007, 1008, 1013,
1014, 1015, 1020, 1021, 1030, 1033, 1038, 1042, 1046, 1048, 1053,
1057, 1061, 1065, 1066, 1069, 1075, 1076, 1078, 1080, 1081, 1085,
1086, 1088, 1089, 1090, 1095, 1103, 1106, 1107, 1109, 1110, 1117,
1118, 1120, 1122, 1125, 1129, 1130, 1131, 1132, 1133, 1134, 1139,
1141, 1148, 1151, 1153, 1159, 1167, 1170, 1175, 1178, 1179, 1181,
1189, 1190, 1191, 1196, 1198, 1199, 1200, 1209, 1215, 1216, 1218,
1220, 1223, 1224, 1232, 1234, 1240, 1243, 1248, 1252, 1255, 1261,
1265, 1267, 1270, 1277, 1291, 1303, 1304, 1315, 1325, 1343, 1345,
1348, 1353, 1354, 1358, 1359, 1360, 1365, 1366, 1373, 1375, 1380,
1383, 1390, 1392, 1394, 1400, 1404, 1406, 1409, 1411, 1413, 1414,
1416, 1421, 1428, 1432, 1434, 1440, 1443, 1451, 1454, 1457, 1460,
1461, 1462, 1467, 1469, 1475, 1477, 1480, 1482, 1483, 1489, 1494,
1495, 1498, 1501, 1507, 1512, 1521, 1523, 1530, 1532, 1535, 1536,
1538, 1539, 1540, 1543, 1551, 1552, 1560, 1561, 1565, 1566, 1567,
1569, 1573, 1577, 1579, 1582, 1583, 1584, 1585, 1586, 1587, 1600,
1601, 1606, 1608, 1610, 1612, 1614, 1616, 1620, 1624, 1626, 1629,
1631, 1632, 1634, 1639, 1644, 1647, 1648, 1649, 1655, 1656, 1660,
1664, 1665, 1667, 1668, 1674, 1677, 1681, 1682, 1685, 1686, 1688,
1692, 1698, 1703, 1705, 1706, 1708, 1709, 1711, 1713, 1714, 1716,
1719, 1720, 1721, 1722, 1728, 1729, 1731, 1733, 1737, 1740, 1741,
1742, 1744, 1745, 1747, 1751, 1754, 1755, 1756, 1758, 1760, 1763,
1768, 1769, 1772, 1774, 1775, 1783, 1786, 1788, 1789, 1795, 1798,
1800, 1807, 1808, 1809, 1810, 1813, 1814, 1815, 1818, 1821, 1822,
1823, 1824, 1827, 1828, 1829, 1831, 1832, 1837, 1839, 1840, 1841,
1842, 1843, 1849, 1851, 1852, 1855, 1856, 1857, 1858, 1860, 1861,
1864, 1867, 1868, 1871, 1872, 1873, 1874, 1875, 1878, 1879, 1881,
1882, 1883, 1884, 1890, 1893, 1894, 1895, 1896, 1898, 1899, 1900,
1901, 1906, 1908, 1913, 1914, 1916, 1917, 1918, 1924, 1925, 1926,
1928, 1929, 1932, 1934, 1936, 1938, 1941, 1943, 1944, 1945, 1947,
1950, 1953, 1955, 1957, 1960, 1966, 1968, 1969, 1970, 1972, 1973,
1976, 1977, 1979, 1980, 1981, 1984, 1985, 1988, 1989, 1991, 1995,
1999, 2008, 2009, 2010, 2011, 2016, 2019, 2022, 2023, 2024, 2025,
2032, 2034, 2035, 2036, 2037, 2039, 2041, 2042, 2045, 2047, 2049,
2050, 2051, 2060, 2062, 2063, 2066, 2069, 2070, 2071, 2072, 2074,
2076, 2077, 2078, 2081, 2084, 2085, 2090, 2095, 2097, 2098, 2100,
2101, 2108, 2112, 2113, 2116, 2117, 2121, 2123, 2129, 2130, 2131,
2133, 2142, 2145, 2148, 2150, 2152, 2153, 2154, 2158, 2159, 2162,
2169, 2171, 2173, 2175, 2228, 2230, 2232, 2233, 2236, 2237, 2242,

```
       2252, 2255, 2262, 2263, 2265, 2266, 2274, 2276, 2283, 2285, 2302,
       2308, 2313, 2314, 2329, 2337, 2339, 2341, 2342, 2354, 2355, 2360,
       2365, 2371, 2382, 2396, 2402, 2414, 2426, 2434, 2438, 2442, 2447,
       2449, 2450, 2451, 2457, 2459, 2463, 2465, 2474, 2476, 2485, 2494,
       2505, 2506, 2515, 2517, 2519, 2522, 2525, 2536, 2540, 2545, 2546,
       2552, 2560, 2562, 2569, 2572, 2576, 2588, 2591, 2592, 2600, 2601,
       2606, 2618, 2627, 2631, 2633, 2642, 2643, 2644, 2645, 2650, 2660,
       2698, 2711, 2730, 2735, 2739, 2745, 2746, 2748, 2758, 2759, 2762,
       2767, 2778, 2792, 2799, 2802, 2803, 2804, 2813, 2815, 2822, 2828,
       2839, 2849, 2856, 2901, 2913, 2917, 2951, 2958, 2991, 2993],
      dtype=int64),
 4: array([   2,   13,   23,   42,   53,   59,   90,  100,  106,  125,  137,
        147,  150,  155,  164,  177,  179,  188,  192,  195,  196,  197,
        200,  210,  214,  238,  241,  287,  292,  295,  301,  315,  351,
        362,  370,  393,  409,  421,  438,  446,  470,  474,  490,  501,
        521,  522,  547,  575,  598,  602,  607,  617,  635,  646,  656,
        674,  681,  701,  702,  721,  728,  734,  741,  743,  776,  832,
        870,  873,  879,  881,  891,  894,  948,  988,  991, 1009, 1022,
       1049, 1072, 1105, 1108, 1113, 1114, 1119, 1124, 1142, 1176, 1187,
       1188, 1212, 1245, 1293, 1308, 1313, 1314, 1332, 1368, 1372, 1385,
       1401, 1418, 1459, 1496, 1509, 1515, 1525, 1570, 1595, 1611, 1687,
       1699, 1749, 1762, 1764, 1767, 1784, 1801, 1825, 1850, 1866, 1905,
       1921, 1962, 2000, 2065, 2170, 2178, 2181, 2183, 2188, 2189, 2190,
       2191, 2193, 2195, 2196, 2198, 2199, 2200, 2205, 2206, 2208, 2211,
       2215, 2217, 2218, 2225, 2227, 2240, 2241, 2258, 2269, 2270, 2278,
       2280, 2282, 2295, 2297, 2320, 2321, 2323, 2330, 2346, 2366, 2378,
       2395, 2403, 2404, 2410, 2412, 2415, 2423, 2427, 2430, 2437, 2441,
       2444, 2448, 2453, 2467, 2481, 2482, 2486, 2489, 2500, 2511, 2518,
       2521, 2523, 2529, 2532, 2534, 2535, 2541, 2577, 2583, 2587, 2602,
       2629, 2634, 2658, 2663, 2667, 2668, 2670, 2672, 2674, 2679, 2685,
       2688, 2695, 2696, 2703, 2707, 2708, 2715, 2722, 2732, 2741, 2750,
       2755, 2760, 2764, 2769, 2771, 2772, 2781, 2783, 2785, 2790, 2793,
       2798, 2801, 2811, 2817, 2819, 2820, 2827, 2830, 2840, 2846, 2850,
       2852, 2859, 2861, 2862, 2863, 2866, 2880, 2882, 2884, 2885, 2895,
       2898, 2899, 2902, 2910, 2918, 2922, 2923, 2924, 2925, 2942, 2949,
       2950, 2953, 2962, 2966, 2968, 2978, 2979, 2987, 2988], dtype=int64),
 5: array([   1,    3,    6,    8,    9,   17,   18,   22,   27,   32,   34,
         39,   41,   49,   52,   65,   69,   73,   84,   88,   92,   93,
        102,  104,  107,  109,  120,  121,  124,  126,  132,  148,  157,
        163,  165,  173,  175,  181,  184,  187,  189,  202,  203,  205,
        213,  216,  218,  221,  225,  227,  246,  250,  256,  259,  261,
        263,  266,  269,  273,  285,  293,  294,  298,  299,  300,  302,
        306,  317,  320,  324,  328,  332,  333,  338,  344,  349,  352,
        359,  361,  363,  364,  371,  379,  386,  387,  400,  404,  405,
        408,  413,  415,  428,  429,  430,  431,  433,  448,  451,  455,
        457,  463,  471,  478,  483,  489,  493,  499,  506,  509,  515,
        519,  531,  536,  538,  546,  548,  549,  554,  557,  562,  566,
        569,  571,  576,  577,  583,  585,  594,  596,  597,  620,  621,
        622,  623,  625,  630,  636,  638,  639,  649,  652,  654,  660,
        665,  695,  698,  704,  716,  722,  723,  727,  736,  739,  740,
        742,  747,  748,  752,  761,  763,  764,  765,  769,  771,  777,
        778,  783,  785,  793,  794,  804,  806,  809,  813,  824,  826,
        835,  839,  853,  859,  860,  861,  868,  875,  884,  889,  898,
        899,  903,  925,  929,  938,  962,  967,  973,  976,  979, 1006,
       1010, 1011, 1029, 1032, 1034, 1044, 1047, 1050, 1051, 1079, 1091,
       1092, 1093, 1097, 1099, 1100, 1101, 1102, 1121, 1138, 1145, 1146,
       1147, 1162, 1163, 1165, 1169, 1173, 1192, 1197, 1202, 1221, 1226,
       1227, 1228, 1230, 1235, 1247, 1254, 1256, 1257, 1264, 1272, 1273,
       1279, 1280, 1284, 1294, 1295, 1297, 1300, 1302, 1305, 1307, 1310,
       1312, 1317, 1321, 1322, 1323, 1324, 1326, 1329, 1333, 1334, 1335,
       1337, 1340, 1349, 1350, 1352, 1371, 1384, 1386, 1387, 1388, 1393,
       1395, 1396, 1405, 1410, 1430, 1439, 1441, 1447, 1448, 1450, 1453,
       1466, 1468, 1472, 1473, 1478, 1479, 1484, 1485, 1497, 1499, 1504,
```

```
1506, 1510, 1514, 1516, 1519, 1522, 1526, 1529, 1534, 1537, 1550,
1559, 1568, 1574, 1578, 1581, 1590, 1605, 1607, 1621, 1625, 1640,
1661, 1666, 1669, 1673, 1696, 1701, 1702, 1712, 1724, 1734, 1738,
1746, 1750, 1785, 1787, 1790, 1802, 1804, 1834, 1844, 1847, 1848,
1854, 1876, 1877, 1880, 1885, 1889, 1910, 1911, 1915, 1923, 1927,
1942, 1952, 1956, 1958, 1975, 1978, 1982, 1986, 1990, 1998, 2001,
2005, 2015, 2018, 2026, 2027, 2029, 2048, 2054, 2056, 2093, 2096,
2109, 2122, 2128, 2134, 2138, 2139, 2143, 2151, 2155, 2156, 2160,
2168, 2186, 2231, 2234, 2239, 2243, 2245, 2247, 2248, 2251, 2253,
2254, 2256, 2257, 2275, 2281, 2286, 2288, 2289, 2292, 2294, 2299,
2303, 2306, 2307, 2310, 2312, 2315, 2316, 2317, 2322, 2324, 2325,
2327, 2328, 2332, 2336, 2343, 2344, 2347, 2349, 2350, 2359, 2361,
2362, 2367, 2369, 2375, 2389, 2390, 2392, 2394, 2398, 2400, 2407,
2408, 2409, 2411, 2413, 2417, 2419, 2420, 2424, 2425, 2428, 2432,
2436, 2439, 2440, 2452, 2456, 2470, 2475, 2478, 2479, 2484, 2487,
2488, 2490, 2493, 2496, 2498, 2504, 2512, 2513, 2524, 2527, 2530,
2531, 2538, 2539, 2553, 2554, 2555, 2556, 2557, 2566, 2567, 2568,
2570, 2571, 2579, 2581, 2582, 2585, 2590, 2593, 2597, 2598, 2603,
2605, 2612, 2613, 2614, 2615, 2616, 2617, 2619, 2621, 2624, 2626,
2630, 2632, 2635, 2636, 2640, 2647, 2648, 2649, 2653, 2654, 2656,
2662, 2665, 2675, 2676, 2677, 2700, 2701, 2702, 2714, 2726, 2727,
2728, 2729, 2731, 2744, 2752, 2754, 2761, 2763, 2765, 2766, 2774,
2776, 2796, 2800, 2812, 2816, 2826, 2833, 2834, 2838, 2851, 2853,
2854, 2857, 2874, 2878, 2881, 2887, 2888, 2889, 2896, 2900, 2903,
2909, 2926, 2936, 2939, 2943, 2946, 2956, 2959, 2967, 2985, 2998,
2999], dtype=int64)}
```

In [124]:

```
dictlist = []
for key, value in mydict.items():
    temp = [key,value]
    dictlist.append(temp)
```

In [125]:
```
dictlist
```

```
Out[125]: [[0, array([  56,   81,  131,  140,  180,  260,  267,  432,  439,  465,  556,
                 578,  655,  667,  672,  735,  759,  815,  833,  847,  849,  858,
                 872,  882,  888,  916,  928,  934,  937,  941,  943,  957,  960,
                 969,  977,  978,  982,  985,  998, 1025, 1045, 1052, 1060, 1067,
                1068, 1077, 1083, 1112, 1128, 1152, 1161, 1166, 1171, 1180, 1194,
                1204, 1208, 1211, 1229, 1237, 1238, 1268, 1271, 1285, 1287, 1289,
                1309, 1319, 1356, 1425, 1426, 1438, 1445, 1455, 1486, 1556, 1598,
                1615, 1622, 1643, 1679, 1680, 1766, 1770, 1776, 1777, 1778, 1782,
                1792, 1794, 1820, 1830, 1835, 1870, 1912, 1959, 1974, 1992, 2064,
                2124, 2137, 2176, 2179, 2180, 2182, 2184, 2192, 2194, 2197, 2201,
                2202, 2203, 2207, 2209, 2213, 2214, 2219, 2223, 2224, 2226, 2244,
                2259, 2260, 2261, 2268, 2271, 2287, 2291, 2333, 2351, 2357, 2370,
                2383, 2386, 2399, 2418, 2429, 2431, 2433, 2445, 2464, 2468, 2508,
                2528, 2542, 2550, 2561, 2565, 2578, 2608, 2610, 2622, 2623, 2639,
                2651, 2661, 2664, 2673, 2681, 2692, 2697, 2699, 2709, 2710, 2718,
                2721, 2723, 2724, 2725, 2733, 2734, 2742, 2751, 2756, 2770, 2779,
                2788, 2791, 2794, 2797, 2809, 2821, 2825, 2829, 2831, 2837, 2843,
                2848, 2865, 2867, 2869, 2871, 2873, 2886, 2891, 2894, 2904, 2908,
                2911, 2912, 2920, 2929, 2935, 2937, 2940, 2941, 2944, 2952, 2954,
                2970, 2971, 2977, 2983, 2994, 2995], dtype=int64)],
           [1, array([   4,   11,   12,   15,   21,   29,   30,   31,   33,   36,   38,
                  50,   60,   61,   63,   64,   71,   80,   82,   85,   94,   98,
                  99,  101,  105,  108,  110,  111,  112,  118,  119,  123,  133,
                 138,  141,  145,  149,  151,  153,  156,  158,  166,  167,  170,
                 186,  198,  204,  207,  208,  209,  212,  215,  219,  226,  230,
                 233,  237,  239,  243,  244,  247,  251,  252,  255,  258,  265,
                 274,  276,  277,  279,  284,  286,  291,  297,  303,  309,  312,
                 313,  314,  316,  318,  319,  325,  337,  345,  346,  350,  360,
                 367,  368,  369,  377,  378,  383,  384,  388,  395,  396,  399,
                 401,  406,  407,  420,  423,  434,  437,  441,  442,  444,  452,
                 453,  456,  469,  472,  475,  480,  481,  486,  488,  494,  496,
                 504,  511,  513,  514,  518,  525,  526,  534,  535,  537,  540,
                 541,  544,  550,  553,  555,  560,  564,  565,  567,  572,  579,
                 581,  587,  590,  595,  603,  605,  606,  608,  609,  611,  624,
                 626,  628,  631,  637,  648,  650,  653,  657,  658,  659,  682,
                 683,  686,  687,  697,  705,  709,  710,  711,  713,  717,  719,
                 732,  738,  745,  749,  753,  755,  756,  762,  767,  773,  775,
                 779,  782,  787,  790,  791,  795,  796,  797,  801,  807,  810,
                 814,  818,  827,  834,  841,  843,  863,  876,  887,  890,  897,
                 900,  902,  904,  905,  906,  907,  918,  919,  921,  922,  926,
                 932,  935,  940,  953,  954,  955,  971,  974,  983,  984,  990,
                 992,  993,  994,  995,  996, 1000, 1001, 1005, 1012, 1016, 1017,
                1018, 1023, 1026, 1027, 1028, 1031, 1035, 1036, 1037, 1039, 1041,
                1043, 1055, 1056, 1058, 1063, 1064, 1070, 1071, 1073, 1074, 1082,
                1084, 1087, 1094, 1096, 1111, 1115, 1123, 1126, 1127, 1136, 1137,
                1144, 1149, 1154, 1155, 1156, 1157, 1158, 1164, 1168, 1174, 1182,
                1185, 1186, 1195, 1201, 1206, 1210, 1214, 1217, 1222, 1233, 1239,
                1241, 1242, 1246, 1249, 1253, 1258, 1260, 1262, 1263, 1266, 1269,
                1274, 1275, 1276, 1278, 1281, 1282, 1283, 1286, 1288, 1292, 1296,
                1298, 1299, 1306, 1311, 1320, 1327, 1328, 1331, 1336, 1339, 1341,
                1342, 1351, 1362, 1369, 1370, 1378, 1379, 1389, 1397, 1398, 1403,
                1407, 1412, 1417, 1422, 1423, 1424, 1433, 1437, 1442, 1444, 1449,
                1452, 1456, 1458, 1464, 1465, 1471, 1476, 1481, 1487, 1492, 1493,
                1500, 1502, 1505, 1508, 1517, 1528, 1531, 1533, 1541, 1544, 1545,
                1546, 1554, 1558, 1572, 1575, 1580, 1589, 1599, 1602, 1603, 1604,
                1609, 1613, 1617, 1633, 1636, 1638, 1642, 1645, 1646, 1650, 1651,
                1652, 1657, 1658, 1662, 1670, 1671, 1672, 1675, 1678, 1689, 1690,
                1694, 1697, 1700, 1704, 1710, 1717, 1723, 1726, 1730, 1732, 1736,
                1739, 1743, 1753, 1757, 1759, 1765, 1771, 1773, 1779, 1791, 1796,
                1803, 1805, 1806, 1817, 1826, 1836, 1853, 1862, 1863, 1886, 1888,
                1892, 1897, 1902, 1903, 1907, 1920, 1930, 1931, 1935, 1937, 1939,
                1948, 1949, 1951, 1954, 1964, 1967, 1993, 1996, 1997, 2007, 2012,
```

```
       2020, 2021, 2028, 2031, 2046, 2073, 2075, 2079, 2083, 2087, 2088,
       2091, 2099, 2102, 2103, 2105, 2106, 2107, 2110, 2119, 2125, 2136,
       2146, 2149, 2157, 2164, 2167, 2172, 2177, 2185, 2187, 2204, 2210,
       2212, 2216, 2220, 2221, 2222, 2229, 2249, 2250, 2264, 2272, 2273,
       2279, 2290, 2293, 2300, 2301, 2304, 2309, 2311, 2318, 2326, 2331,
       2334, 2335, 2340, 2352, 2353, 2356, 2358, 2364, 2372, 2373, 2376,
       2379, 2380, 2381, 2385, 2387, 2388, 2397, 2401, 2416, 2422, 2435,
       2443, 2446, 2454, 2460, 2461, 2462, 2469, 2472, 2473, 2480, 2491,
       2492, 2499, 2501, 2502, 2503, 2510, 2514, 2516, 2533, 2537, 2543,
       2549, 2558, 2559, 2563, 2580, 2584, 2586, 2589, 2595, 2596, 2599,
       2604, 2607, 2609, 2620, 2638, 2641, 2659, 2666, 2669, 2671, 2678,
       2680, 2682, 2683, 2684, 2686, 2687, 2689, 2690, 2691, 2693, 2694,
       2704, 2705, 2706, 2712, 2713, 2716, 2719, 2736, 2737, 2738, 2740,
       2743, 2747, 2749, 2753, 2757, 2768, 2773, 2777, 2780, 2782, 2784,
       2786, 2787, 2789, 2795, 2805, 2807, 2808, 2810, 2814, 2818, 2823,
       2824, 2832, 2836, 2841, 2842, 2844, 2845, 2847, 2855, 2858, 2860,
       2864, 2868, 2870, 2872, 2875, 2876, 2877, 2879, 2883, 2890, 2892,
       2893, 2897, 2905, 2906, 2907, 2914, 2915, 2916, 2919, 2921, 2927,
       2928, 2930, 2931, 2932, 2933, 2934, 2938, 2945, 2947, 2948, 2955,
       2957, 2960, 2961, 2963, 2964, 2965, 2972, 2973, 2974, 2975, 2976,
       2980, 2981, 2982, 2984, 2986, 2989, 2990, 2992, 2996, 2997],
      dtype=int64)],
[2, array([   5,    7,   20,   26,   46,   47,   55,   72,   78,   83,   86,
         87,   91,   95,  115,  116,  129,  135,  136,  144,  172,  183,
        190,  191,  211,  217,  229,  231,  232,  234,  242,  262,  264,
        268,  275,  283,  289,  296,  307,  308,  311,  322,  323,  327,
        329,  340,  341,  342,  356,  374,  380,  381,  389,  391,  394,
        397,  398,  402,  411,  412,  414,  416,  422,  424,  436,  440,
        449,  450,  460,  464,  467,  497,  498,  500,  502,  503,  507,
        529,  532,  539,  558,  559,  582,  584,  592,  601,  604,  613,
        615,  618,  619,  627,  629,  632,  640,  641,  647,  661,  666,
        675,  679,  690,  694,  706,  733,  750,  788,  802,  803,  805,
        808,  817,  830,  837,  838,  844,  848,  851,  852,  856,  865,
        867,  912,  924,  927,  931,  936,  939,  942,  947,  949,  950,
        956,  964,  966,  975,  986,  997,  999, 1003, 1019, 1024, 1040,
       1054, 1059, 1062, 1098, 1104, 1116, 1135, 1140, 1143, 1150, 1160,
       1172, 1177, 1183, 1184, 1193, 1203, 1205, 1207, 1213, 1219, 1225,
       1231, 1236, 1244, 1250, 1251, 1259, 1290, 1301, 1316, 1318, 1330,
       1338, 1344, 1346, 1347, 1355, 1357, 1361, 1363, 1364, 1367, 1374,
       1376, 1377, 1381, 1382, 1391, 1399, 1402, 1408, 1415, 1419, 1420,
       1427, 1429, 1431, 1435, 1436, 1446, 1463, 1470, 1474, 1488, 1490,
       1491, 1503, 1511, 1513, 1518, 1520, 1524, 1527, 1542, 1547, 1548,
       1549, 1553, 1555, 1557, 1562, 1563, 1564, 1571, 1576, 1588, 1591,
       1592, 1593, 1594, 1596, 1597, 1618, 1619, 1623, 1627, 1628, 1630,
       1635, 1637, 1641, 1653, 1654, 1659, 1663, 1676, 1683, 1684, 1691,
       1693, 1695, 1707, 1715, 1718, 1725, 1727, 1735, 1748, 1752, 1761,
       1780, 1781, 1793, 1797, 1799, 1811, 1812, 1816, 1819, 1833, 1838,
       1845, 1846, 1859, 1865, 1869, 1887, 1891, 1904, 1909, 1919, 1922,
       1933, 1940, 1946, 1961, 1963, 1965, 1971, 1983, 1987, 1994, 2002,
       2003, 2004, 2006, 2013, 2014, 2017, 2030, 2033, 2038, 2040, 2043,
       2044, 2052, 2053, 2055, 2057, 2058, 2059, 2061, 2067, 2068, 2080,
       2082, 2086, 2089, 2092, 2094, 2104, 2111, 2114, 2115, 2118, 2120,
       2126, 2127, 2132, 2135, 2140, 2141, 2144, 2147, 2161, 2163, 2165,
       2166, 2174, 2235, 2238, 2246, 2267, 2277, 2284, 2296, 2298, 2305,
       2319, 2338, 2345, 2348, 2363, 2368, 2374, 2377, 2384, 2391, 2393,
       2405, 2406, 2421, 2455, 2458, 2466, 2471, 2477, 2483, 2495, 2497,
       2507, 2509, 2520, 2526, 2544, 2547, 2548, 2551, 2564, 2573, 2574,
       2575, 2594, 2611, 2625, 2628, 2637, 2646, 2652, 2655, 2657, 2717,
       2720, 2775, 2806, 2835, 2969], dtype=int64)],
[3, array([   0,   10,   14,   16,   19,   24,   25,   28,   35,   37,   40,
         43,   44,   45,   48,   51,   54,   57,   58,   62,   66,   67,
         68,   70,   74,   75,   76,   77,   79,   89,   96,   97,  103,
        113,  114,  117,  122,  127,  128,  130,  134,  139,  142,  143,
```

146, 152, 154, 159, 160, 161, 162, 168, 169, 171, 174,
176, 178, 182, 185, 193, 194, 199, 201, 206, 220, 222,
223, 224, 228, 235, 236, 240, 245, 248, 249, 253, 254,
257, 270, 271, 272, 278, 280, 281, 282, 288, 290, 304,
305, 310, 321, 326, 330, 331, 334, 335, 336, 339, 343,
347, 348, 353, 354, 355, 357, 358, 365, 366, 372, 373,
375, 376, 382, 385, 390, 392, 403, 410, 417, 418, 419,
425, 426, 427, 435, 443, 445, 447, 454, 458, 459, 461,
462, 466, 468, 473, 476, 477, 479, 482, 484, 485, 487,
491, 492, 495, 505, 508, 510, 512, 516, 517, 520, 523,
524, 527, 528, 530, 533, 542, 543, 545, 551, 552, 561,
563, 568, 570, 573, 574, 580, 586, 588, 589, 591, 593,
599, 600, 610, 612, 614, 616, 633, 634, 642, 643, 644,
645, 651, 662, 663, 664, 668, 669, 670, 671, 673, 676,
677, 678, 680, 684, 685, 688, 689, 691, 692, 693, 696,
699, 700, 703, 707, 708, 712, 714, 715, 718, 720, 724,
725, 726, 729, 730, 731, 737, 744, 746, 751, 754, 757,
758, 760, 766, 768, 770, 772, 774, 780, 781, 784, 786,
789, 792, 798, 799, 800, 811, 812, 816, 819, 820, 821,
822, 823, 825, 828, 829, 831, 836, 840, 842, 845, 846,
850, 854, 855, 857, 862, 864, 866, 869, 871, 874, 877,
878, 880, 883, 885, 886, 892, 893, 895, 896, 901, 908,
909, 910, 911, 913, 914, 915, 917, 920, 923, 930, 933,
944, 945, 946, 951, 952, 958, 959, 961, 963, 965, 968,
970, 972, 980, 981, 987, 989, 1002, 1004, 1007, 1008, 1013,
1014, 1015, 1020, 1021, 1030, 1033, 1038, 1042, 1046, 1048, 1053,
1057, 1061, 1065, 1066, 1069, 1075, 1076, 1078, 1080, 1081, 1085,
1086, 1088, 1089, 1090, 1095, 1103, 1106, 1107, 1109, 1110, 1117,
1118, 1120, 1122, 1125, 1129, 1130, 1131, 1132, 1133, 1134, 1139,
1141, 1148, 1151, 1153, 1159, 1167, 1170, 1175, 1178, 1179, 1181,
1189, 1190, 1191, 1196, 1198, 1199, 1200, 1209, 1215, 1216, 1218,
1220, 1223, 1224, 1232, 1234, 1240, 1243, 1248, 1252, 1255, 1261,
1265, 1267, 1270, 1277, 1291, 1303, 1304, 1315, 1325, 1343, 1345,
1348, 1353, 1354, 1358, 1359, 1360, 1365, 1366, 1373, 1375, 1380,
1383, 1390, 1392, 1394, 1400, 1404, 1406, 1409, 1411, 1413, 1414,
1416, 1421, 1428, 1432, 1434, 1440, 1443, 1451, 1454, 1457, 1460,
1461, 1462, 1467, 1469, 1475, 1477, 1480, 1482, 1483, 1489, 1494,
1495, 1498, 1501, 1507, 1512, 1521, 1523, 1530, 1532, 1535, 1536,
1538, 1539, 1540, 1543, 1551, 1552, 1560, 1561, 1565, 1566, 1567,
1569, 1573, 1577, 1579, 1582, 1583, 1584, 1585, 1586, 1587, 1600,
1601, 1606, 1608, 1610, 1612, 1614, 1616, 1620, 1624, 1626, 1629,
1631, 1632, 1634, 1639, 1644, 1647, 1648, 1649, 1655, 1656, 1660,
1664, 1665, 1667, 1668, 1674, 1677, 1681, 1682, 1685, 1686, 1688,
1692, 1698, 1703, 1705, 1706, 1708, 1709, 1711, 1713, 1714, 1716,
1719, 1720, 1721, 1722, 1728, 1729, 1731, 1733, 1737, 1740, 1741,
1742, 1744, 1745, 1747, 1751, 1754, 1755, 1756, 1758, 1760, 1763,
1768, 1769, 1772, 1774, 1775, 1783, 1786, 1788, 1789, 1795, 1798,
1800, 1807, 1808, 1809, 1810, 1813, 1814, 1815, 1818, 1821, 1822,
1823, 1824, 1827, 1828, 1829, 1831, 1832, 1837, 1839, 1840, 1841,
1842, 1843, 1849, 1851, 1852, 1855, 1856, 1857, 1858, 1860, 1861,
1864, 1867, 1868, 1871, 1872, 1873, 1874, 1875, 1878, 1879, 1881,
1882, 1883, 1884, 1890, 1893, 1894, 1895, 1896, 1898, 1899, 1900,
1901, 1906, 1908, 1913, 1914, 1916, 1917, 1918, 1924, 1925, 1926,
1928, 1929, 1932, 1934, 1936, 1938, 1941, 1943, 1944, 1945, 1947,
1950, 1953, 1955, 1957, 1960, 1966, 1968, 1969, 1970, 1972, 1973,
1976, 1977, 1979, 1980, 1981, 1984, 1985, 1988, 1989, 1991, 1995,
1999, 2008, 2009, 2010, 2011, 2016, 2019, 2022, 2023, 2024, 2025,
2032, 2034, 2035, 2036, 2037, 2039, 2041, 2042, 2045, 2047, 2049,
2050, 2051, 2060, 2062, 2063, 2066, 2069, 2070, 2071, 2072, 2074,
2076, 2077, 2078, 2081, 2084, 2085, 2090, 2095, 2097, 2098, 2100,
2101, 2108, 2112, 2113, 2116, 2117, 2121, 2123, 2129, 2130, 2131,
2133, 2142, 2145, 2148, 2150, 2152, 2153, 2154, 2158, 2159, 2162,
2169, 2171, 2173, 2175, 2228, 2230, 2232, 2233, 2236, 2237, 2242,

```
        2252, 2255, 2262, 2263, 2265, 2266, 2274, 2276, 2283, 2285, 2302,
        2308, 2313, 2314, 2329, 2337, 2339, 2341, 2342, 2354, 2355, 2360,
        2365, 2371, 2382, 2396, 2402, 2414, 2426, 2434, 2438, 2442, 2447,
        2449, 2450, 2451, 2457, 2459, 2463, 2465, 2474, 2476, 2485, 2494,
        2505, 2506, 2515, 2517, 2519, 2522, 2525, 2536, 2540, 2545, 2546,
        2552, 2560, 2562, 2569, 2572, 2576, 2588, 2591, 2592, 2600, 2601,
        2606, 2618, 2627, 2631, 2633, 2642, 2643, 2644, 2645, 2650, 2660,
        2698, 2711, 2730, 2735, 2739, 2745, 2746, 2748, 2758, 2759, 2762,
        2767, 2778, 2792, 2799, 2802, 2803, 2804, 2813, 2815, 2822, 2828,
        2839, 2849, 2856, 2901, 2913, 2917, 2951, 2958, 2991, 2993],
       dtype=int64)],
 [4, array([   2,   13,   23,   42,   53,   59,   90,  100,  106,  125,  137,
         147,  150,  155,  164,  177,  179,  188,  192,  195,  196,  197,
         200,  210,  214,  238,  241,  287,  292,  295,  301,  315,  351,
         362,  370,  393,  409,  421,  438,  446,  470,  474,  490,  501,
         521,  522,  547,  575,  598,  602,  607,  617,  635,  646,  656,
         674,  681,  701,  702,  721,  728,  734,  741,  743,  776,  832,
         870,  873,  879,  881,  891,  894,  948,  988,  991, 1009, 1022,
        1049, 1072, 1105, 1108, 1113, 1114, 1119, 1124, 1142, 1176, 1187,
        1188, 1212, 1245, 1293, 1308, 1313, 1314, 1332, 1368, 1372, 1385,
        1401, 1418, 1459, 1496, 1509, 1515, 1525, 1570, 1595, 1611, 1687,
        1699, 1749, 1762, 1764, 1767, 1784, 1801, 1825, 1850, 1866, 1905,
        1921, 1962, 2000, 2065, 2170, 2178, 2181, 2183, 2188, 2189, 2190,
        2191, 2193, 2195, 2196, 2198, 2199, 2200, 2205, 2206, 2208, 2211,
        2215, 2217, 2218, 2225, 2227, 2240, 2241, 2258, 2269, 2270, 2278,
        2280, 2282, 2295, 2297, 2320, 2321, 2323, 2330, 2346, 2366, 2378,
        2395, 2403, 2404, 2410, 2412, 2415, 2423, 2427, 2430, 2437, 2441,
        2444, 2448, 2453, 2467, 2481, 2482, 2486, 2489, 2500, 2511, 2518,
        2521, 2523, 2529, 2532, 2534, 2535, 2541, 2577, 2583, 2587, 2602,
        2629, 2634, 2658, 2663, 2667, 2668, 2670, 2672, 2674, 2679, 2685,
        2688, 2695, 2696, 2703, 2707, 2708, 2715, 2722, 2732, 2741, 2750,
        2755, 2760, 2764, 2769, 2771, 2772, 2781, 2783, 2785, 2790, 2793,
        2798, 2801, 2811, 2817, 2819, 2820, 2827, 2830, 2840, 2846, 2850,
        2852, 2859, 2861, 2862, 2863, 2866, 2880, 2882, 2884, 2885, 2895,
        2898, 2899, 2902, 2910, 2918, 2922, 2923, 2924, 2925, 2942, 2949,
        2950, 2953, 2962, 2966, 2968, 2978, 2979, 2987, 2988], dtype=int64)],
 [5, array([   1,    3,    6,    8,    9,   17,   18,   22,   27,   32,   34,
          39,   41,   49,   52,   65,   69,   73,   84,   88,   92,   93,
         102,  104,  107,  109,  120,  121,  124,  126,  132,  148,  157,
         163,  165,  173,  175,  181,  184,  187,  189,  202,  203,  205,
         213,  216,  218,  221,  225,  227,  246,  250,  256,  259,  261,
         263,  266,  269,  273,  285,  293,  294,  298,  299,  300,  302,
         306,  317,  320,  324,  328,  332,  333,  338,  344,  349,  352,
         359,  361,  363,  364,  371,  379,  386,  387,  400,  404,  405,
         408,  413,  415,  428,  429,  430,  431,  433,  448,  451,  455,
         457,  463,  471,  478,  483,  489,  493,  499,  506,  509,  515,
         519,  531,  536,  538,  546,  548,  549,  554,  557,  562,  566,
         569,  571,  576,  577,  583,  585,  594,  596,  597,  620,  621,
         622,  623,  625,  630,  636,  638,  639,  649,  652,  654,  660,
         665,  695,  698,  704,  716,  722,  723,  727,  736,  739,  740,
         742,  747,  748,  752,  761,  763,  764,  765,  769,  771,  777,
         778,  783,  785,  793,  794,  804,  806,  809,  813,  824,  826,
         835,  839,  853,  859,  860,  861,  868,  875,  884,  889,  898,
         899,  903,  925,  929,  938,  962,  967,  973,  976,  979, 1006,
        1010, 1011, 1029, 1032, 1034, 1044, 1047, 1050, 1051, 1079, 1091,
        1092, 1093, 1097, 1099, 1100, 1101, 1102, 1121, 1138, 1145, 1146,
        1147, 1162, 1163, 1165, 1169, 1173, 1192, 1197, 1202, 1221, 1226,
        1227, 1228, 1230, 1235, 1247, 1254, 1256, 1257, 1264, 1272, 1273,
        1279, 1280, 1284, 1294, 1295, 1297, 1300, 1302, 1305, 1307, 1310,
        1312, 1317, 1321, 1322, 1323, 1324, 1326, 1329, 1333, 1334, 1335,
        1337, 1340, 1349, 1350, 1352, 1371, 1384, 1386, 1387, 1388, 1393,
        1395, 1396, 1405, 1410, 1430, 1439, 1441, 1447, 1448, 1450, 1453,
        1466, 1468, 1472, 1473, 1478, 1479, 1484, 1485, 1497, 1499, 1504,
```

```
1506, 1510, 1514, 1516, 1519, 1522, 1526, 1529, 1534, 1537, 1550,
1559, 1568, 1574, 1578, 1581, 1590, 1605, 1607, 1621, 1625, 1640,
1661, 1666, 1669, 1673, 1696, 1701, 1702, 1712, 1724, 1734, 1738,
1746, 1750, 1785, 1787, 1790, 1802, 1804, 1834, 1844, 1847, 1848,
1854, 1876, 1877, 1880, 1885, 1889, 1910, 1911, 1915, 1923, 1927,
1942, 1952, 1956, 1958, 1975, 1978, 1982, 1986, 1990, 1998, 2001,
2005, 2015, 2018, 2026, 2027, 2029, 2048, 2054, 2056, 2093, 2096,
2109, 2122, 2128, 2134, 2138, 2139, 2143, 2151, 2155, 2156, 2160,
2168, 2186, 2231, 2234, 2239, 2243, 2245, 2247, 2248, 2251, 2253,
2254, 2256, 2257, 2275, 2281, 2286, 2288, 2289, 2292, 2294, 2299,
2303, 2306, 2307, 2310, 2312, 2315, 2316, 2317, 2322, 2324, 2325,
2327, 2328, 2332, 2336, 2343, 2344, 2347, 2349, 2350, 2359, 2361,
2362, 2367, 2369, 2375, 2389, 2390, 2392, 2394, 2398, 2400, 2407,
2408, 2409, 2411, 2413, 2417, 2419, 2420, 2424, 2425, 2428, 2432,
2436, 2439, 2440, 2452, 2456, 2470, 2475, 2478, 2479, 2484, 2487,
2488, 2490, 2493, 2496, 2498, 2504, 2512, 2513, 2524, 2527, 2530,
2531, 2538, 2539, 2553, 2554, 2555, 2556, 2557, 2566, 2567, 2568,
2570, 2571, 2579, 2581, 2582, 2585, 2590, 2593, 2597, 2598, 2603,
2605, 2612, 2613, 2614, 2615, 2616, 2617, 2619, 2621, 2624, 2626,
2630, 2632, 2635, 2636, 2640, 2647, 2648, 2649, 2653, 2654, 2656,
2662, 2665, 2675, 2676, 2677, 2700, 2701, 2702, 2714, 2726, 2727,
2728, 2729, 2731, 2744, 2752, 2754, 2761, 2763, 2765, 2766, 2774,
2776, 2796, 2800, 2812, 2816, 2826, 2833, 2834, 2838, 2851, 2853,
2854, 2857, 2874, 2878, 2881, 2887, 2888, 2889, 2896, 2900, 2903,
2909, 2926, 2936, 2939, 2943, 2946, 2956, 2959, 2967, 2985, 2998,
2999], dtype=int64)]]
```

In [126]:
```
df_ori.head()
```

Out[126]:

	average_item_count	average_basket_spend	average_spend_per_item	Recency
customer_number				
14	9.48	12.07	1.27	1
45	19.85	17.75	0.89	1
52	4.98	3.77	0.76	2
61	13.49	14.81	1.10	3
63	5.85	6.11	1.04	7

In [127]:
```
df_ori = df_ori.rename_axis(columns = None).reset_index()
```

In [128]:
```
df_ori.head()
```

Out[128]:

	customer_number	average_item_count	average_basket_spend	average_spend_per_item	Recenc
0	14	9.48	12.07	1.27	
1	45	19.85	17.75	0.89	
2	52	4.98	3.77	0.76	
3	61	13.49	14.81	1.10	
4	63	5.85	6.11	1.04	

In [ ]:

In [129]:
```
df_ori = df_ori.copy()
```

In [130]:
```
df_ori.insert(loc=0, column='number', value=np.arange(len(df_ori)))
```

In [131]:
```
df_ori.head()
```

Out[131]:

	number	customer_number	average_item_count	average_basket_spend	average_spend_per_item
0	0	14	9.48	12.07	1.27
1	1	45	19.85	17.75	0.89
2	2	52	4.98	3.77	0.76
3	3	61	13.49	14.81	1.10
4	4	63	5.85	6.11	1.04

In [ ]:

In [132]:
```
clu1 = dictlist[0]
clu1 = pd.DataFrame(clu1)
clu1 = clu1.drop([0])
clu1 = [i[0] for i in clu1.values.tolist()]
[clu1] = clu1
clu1 = clu1.tolist()
clu1[0:4]
```

Out[132]: [56, 81, 131, 140]

In [133]:
```
clu2 = dictlist[1]
clu2 = pd.DataFrame(clu2)
clu2 = clu2.drop([0])
clu2 = [i[0] for i in clu2.values.tolist()]
[clu2] = clu2
clu2 = clu2.tolist()
clu2[0:4]
```

Out[133]: [4, 11, 12, 15]

In [134]:
```
clu3 = dictlist[2]
clu3 = pd.DataFrame(clu3)
clu3 = clu3.drop([0])
clu3 = [i[0] for i in clu3.values.tolist()]
[clu3] = clu3
clu3 = clu3.tolist()
clu3[0:4]
```

Out[134]: [5, 7, 20, 26]

In [135]:
```
clu4 = dictlist[3]
clu4 = pd.DataFrame(clu4)
clu4 = clu4.drop([0])
clu4 = [i[0] for i in clu4.values.tolist()]
[clu4] = clu4
clu4 = clu4.tolist()
clu4[0:4]
```

Out[135]: [0, 10, 14, 16]

```
In [136]: clu5 = dictlist[4]
          clu5 = pd.DataFrame(clu5)
          clu5 = clu5.drop([0])
          clu5 = [i[0] for i in clu5.values.tolist()]
          [clu5] = clu5
          clu5 = clu5.tolist()
          clu5[0:4]
```

```
Out[136]: [2, 13, 23, 42]
```

```
In [137]: clu6 = dictlist[5]
          clu6 = pd.DataFrame(clu6)
          clu6 = clu6.drop([0])
          clu6 = [i[0] for i in clu6.values.tolist()]
          [clu6] = clu6
          clu6 = clu6.tolist()
          clu6[0:4]
```

```
Out[137]: [1, 3, 6, 8]
```

9-2. Six segments

```
In [138]: seg1 = df_ori.loc[df_ori['number'].isin(clu1)]
          seg1 = seg1.drop(columns=['number','customer_number'])
          seg1.head()
```

Out[138]:

	average_item_count	average_basket_spend	average_spend_per_item	Recency	Frequency	M
56	3.98	6.39	1.60	0	54	
81	4.59	13.50	2.94	3	37	
131	9.00	15.60	1.73	1	36	
140	3.51	9.70	2.76	3	86	
180	4.45	16.28	3.65	6	66	

```
In [139]: seg2 = df_ori.loc[df_ori['number'].isin(clu2)]
          seg2 = seg2.drop(columns=['number','customer_number'])
          seg2.head()
```

Out[139]:

	average_item_count	average_basket_spend	average_spend_per_item	Recency	Frequency	Mo
4	5.85	6.11	1.04	7	48	
11	4.84	8.97	1.85	1	67	
12	7.15	12.06	1.69	2	46	
15	6.75	9.18	1.36	0	48	
21	8.51	7.78	0.91	1	39	

In [140]:
```python
seg3 = df_ori.loc[df_ori['number'].isin(clu3)]
seg3 = seg3.drop(columns=['number','customer_number'])
seg3.head()
```

Out[140]:

	average_item_count	average_basket_spend	average_spend_per_item	Recency	Frequency	Mo
5	23.44	35.20	1.50	2	45	[illegible]
7	23.10	29.38	1.27	3	42	[illegible]
20	44.71	52.11	1.17	18	14	[illegible]
26	21.96	33.78	1.54	2	55	[illegible]
46	16.59	21.25	1.28	6	63	[illegible]

In [141]:
```python
seg4 = df_ori.loc[df_ori['number'].isin(clu4)]
seg4 = seg4.drop(columns=['number','customer_number'])
seg4.head()
```

Out[141]:

	average_item_count	average_basket_spend	average_spend_per_item	Recency	Frequency	Mo
0	9.48	12.07	1.27	1	56	[illegible]
10	8.19	9.78	1.19	6	53	[illegible]
14	9.68	13.27	1.37	0	40	[illegible]
16	10.29	10.83	1.05	1	79	[illegible]
19	8.17	12.97	1.59	4	54	[illegible]

In [142]:
```python
seg5 = df_ori.loc[df_ori['number'].isin(clu5)]
seg5 = seg5.drop(columns=['number','customer_number'])
seg5.head()
```

Out[142]:

	average_item_count	average_basket_spend	average_spend_per_item	Recency	Frequency	Mo
2	4.98	3.77	0.76	2	59	[illegible]
13	7.03	5.07	0.72	93	35	
23	13.64	14.97	1.10	52	11	
42	12.78	16.47	1.29	2	9	
53	18.88	20.25	1.07	30	8	

In [143]:
```
seg6 = df_ori.loc[df_ori['number'].isin(clu6)]
seg6 = seg6.drop(columns=['number','customer_number'])
seg6.head()
```

Out[143]:

	average_item_count	average_basket_spend	average_spend_per_item	Recency	Frequency	Mon
1	19.85	17.75	0.89	1	33	5
3	13.49	14.81	1.10	3	37	5
6	12.17	12.52	1.03	2	18	2
8	12.20	17.76	1.46	0	20	3
9	14.19	19.70	1.39	4	37	7

In [ ]:

10. Individual statistical summaries of clusters

In [144]:
```
round(seg1.describe(),2)
```

Out[144]:

	average_item_count	average_basket_spend	average_spend_per_item	Recency	Frequency
count	215.00	215.00	215.00	215.00	215.00
mean	4.79	12.62	2.68	3.86	63.86
std	1.80	6.71	1.00	9.72	37.55
min	1.37	3.02	1.17	0.00	9.00
25%	3.52	8.24	2.04	0.00	38.00
50%	4.70	11.97	2.44	1.00	54.00
75%	5.77	14.76	3.05	3.00	80.00
max	10.07	57.32	7.92	82.00	266.00

In [145]:
```
seg1.head()
```

Out[145]:

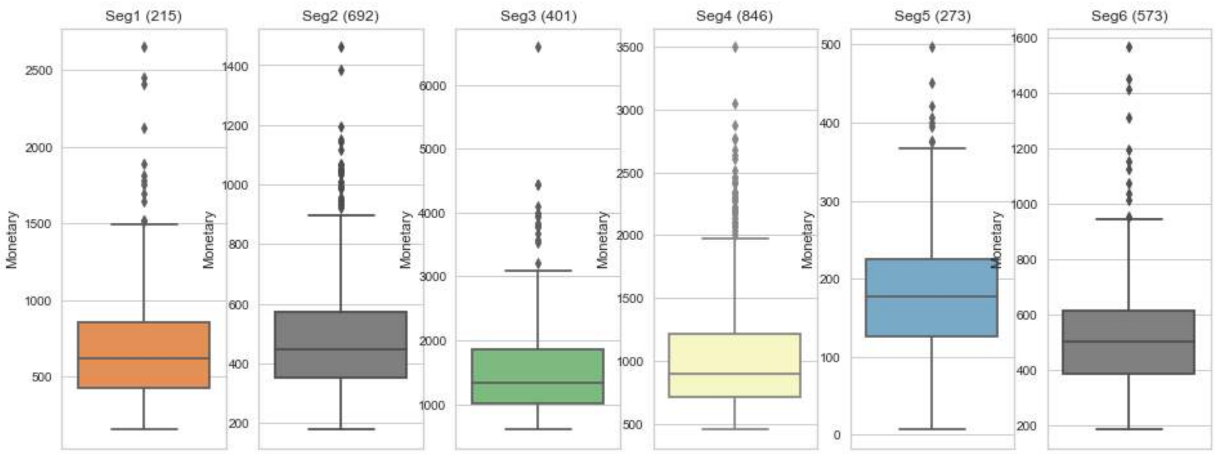
	average_item_count	average_basket_spend	average_spend_per_item	Recency	Frequency	M
56	3.98	6.39	1.60	0	54	
81	4.59	13.50	2.94	3	37	
131	9.00	15.60	1.73	1	36	
140	3.51	9.70	2.76	3	86	
180	4.45	16.28	3.65	6	66	

In [146]:
```
print(seg1.Monetary.describe())
print(seg2.Monetary.describe())
print(seg3.Monetary.describe())
print(seg4.Monetary.describe())
print(seg5.Monetary.describe())
print(seg6.Monetary.describe())
```

```
count     215.000000
mean      712.611721
std       414.090293
min       156.610000
25%       423.950000
50%       619.030000
75%       852.735000
max      2649.050000
Name: Monetary, dtype: float64
count     692.000000
mean      484.672803
std       190.249905
min       178.880000
25%       351.280000
50%       446.990000
75%       574.110000
max      1460.410000
Name: Monetary, dtype: float64
count     401.000000
mean     1518.787332
std       725.349756
min       607.930000
25%      1014.820000
50%      1337.370000
75%      1854.140000
max      6588.650000
Name: Monetary, dtype: float64
count     846.000000
mean     1019.234504
std       432.587454
min       461.500000
25%       712.077500
50%       897.550000
75%      1219.970000
max      3491.780000
Name: Monetary, dtype: float64
count    273.000000
mean     181.860696
std       82.556699
min        7.280000
25%      126.420000
50%      177.290000
75%      225.690000
max      496.630000
Name: Monetary, dtype: float64
count     573.000000
mean      521.255899
std       186.525208
min       187.760000
25%       389.160000
50%       504.260000
75%       614.610000
max      1565.120000
Name: Monetary, dtype: float64
```

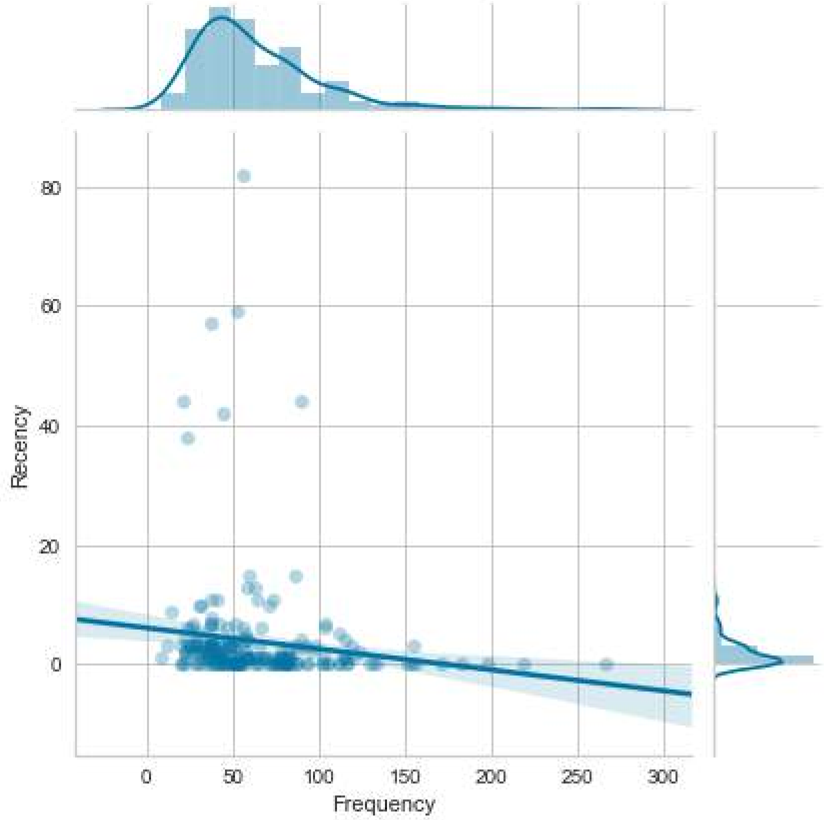
In [147]:
```
figure, (ax1, ax2, ax3, ax4, ax5, ax6) = plt.subplots(1, 6)
figure.set_size_inches(16,6)

sns.boxplot(seg1.Monetary, ax=ax1, orient = 'v', palette='Oranges'
           ).set_title('Seg1 (215)')
sns.boxplot(seg2.Monetary, ax=ax2, orient = 'v', palette='binary'
           ).set_title('Seg2 (692)')
sns.boxplot(seg3.Monetary, ax=ax3, orient = 'v', palette='Greens'
           ).set_title('Seg3 (401)')
sns.boxplot(seg4.Monetary, ax=ax4, orient = 'v', palette='Spectral'
           ).set_title('Seg4 (846)')
sns.boxplot(seg5.Monetary, ax=ax5, orient = 'v', palette='Blues'
           ).set_title('Seg5 (273)')
sns.boxplot(seg6.Monetary, ax=ax6, orient = 'v', palette='gist_gray'
           ).set_title('Seg6 (573)')
```

Out[147]: Text(0.5, 1.0, 'Seg6 (573)')



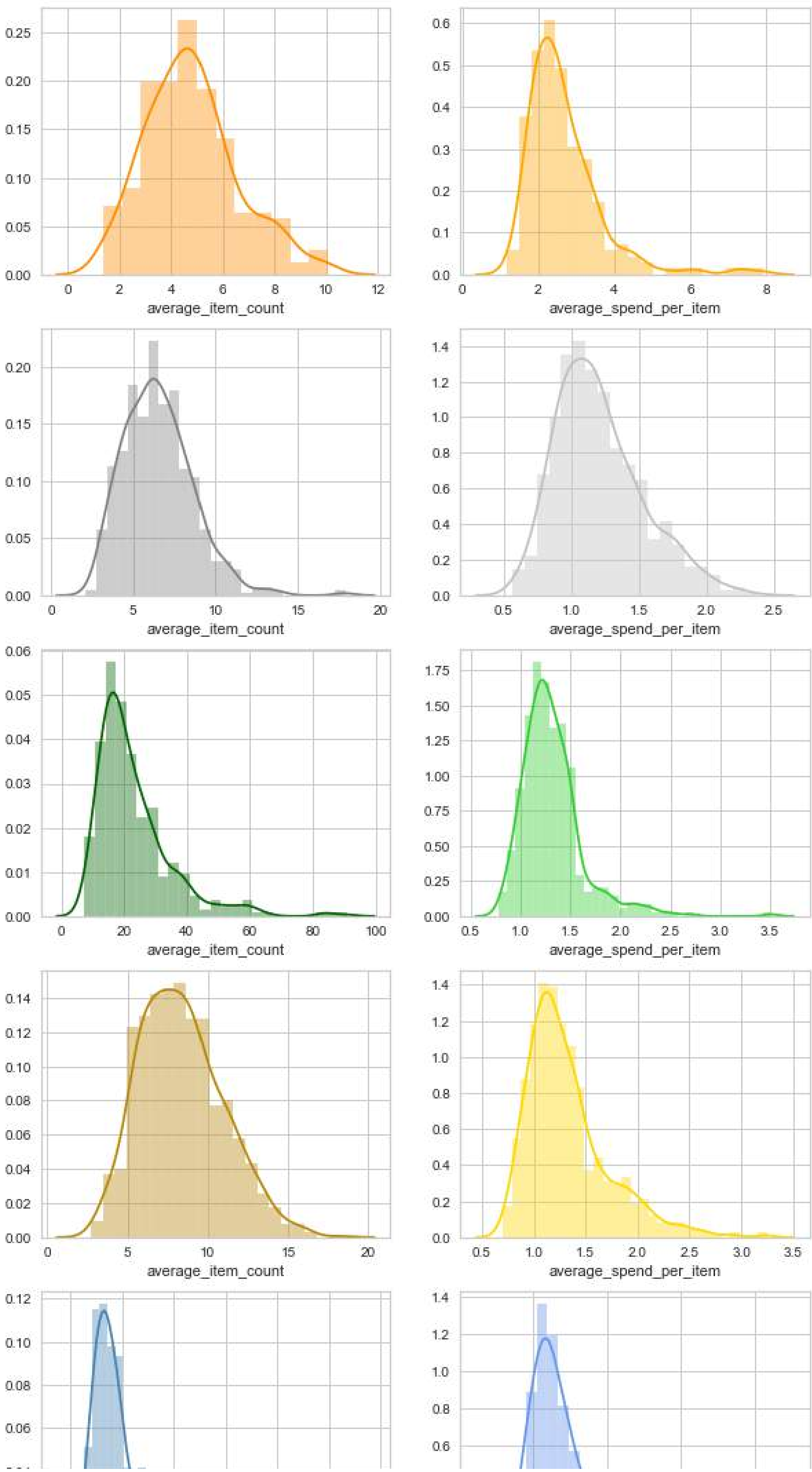
In [148]:
```
sns.jointplot(x='Frequency', y='Recency',  scatter_kws={'alpha':0.3}, kind='re
g',
            data=seg1)
```

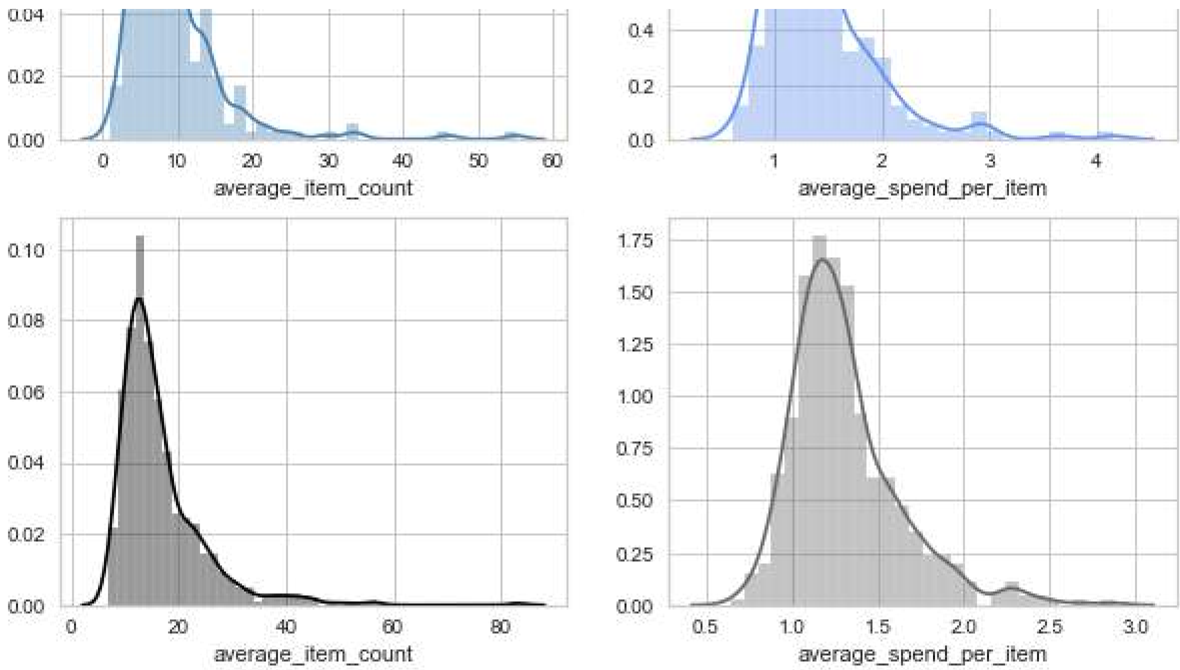
Out[148]: <seaborn.axisgrid.JointGrid at 0x15d98d35860>



In [149]:
```
'''
item count - bakset spend 0.62
basket spend - item count 0.62
basket spend - spend per tiem 0.51
'''

figure, axes  = plt.subplots(6, 2)
figure.set_size_inches(10,25)

sns.distplot(seg1.average_item_count,
             ax=axes[0][0],color='darkorange')
sns.distplot(seg1.average_spend_per_item,
             ax=axes[0][1], color='orange')
sns.distplot(seg2.average_item_count,
             ax=axes[1][0],color='gray')
sns.distplot(seg2.average_spend_per_item,
             ax=axes[1][1],color='silver')

sns.distplot(seg3.average_item_count,
             ax=axes[2][0],color='darkgreen')
sns.distplot(seg3.average_spend_per_item,
             ax=axes[2][1],color='limegreen')

sns.distplot(seg4.average_item_count,
             ax=axes[3][0],color='darkgoldenrod')
sns.distplot(seg4.average_spend_per_item,
             ax=axes[3][1], color='gold')

sns.distplot(seg5.average_item_count,
             ax=axes[4][0],color='steelblue')
sns.distplot(seg5.average_spend_per_item,
             ax=axes[4][1], color='cornflowerblue')

sns.distplot(seg6.average_item_count,
             ax=axes[5][0],color='black')
sns.distplot(seg6.average_spend_per_item,
             ax=axes[5][1], color='dimgray')
```

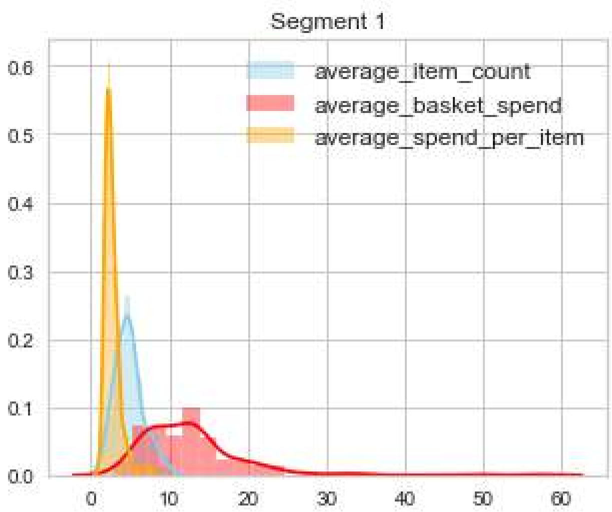
```
Out[149]: <matplotlib.axes._subplots.AxesSubplot at 0x15d9b1395f8>
```


average_item_count
average_spend_per_item
average_item_count
average_spend_per_item
average_item_count
average_spend_per_item
average_item_count
average_spend_per_item

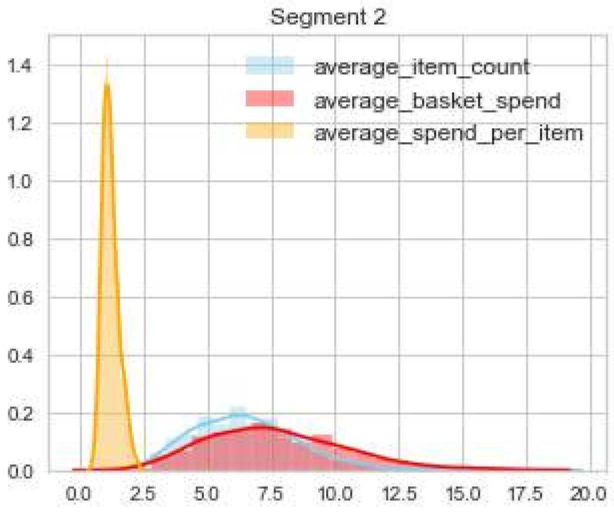


In [150]:

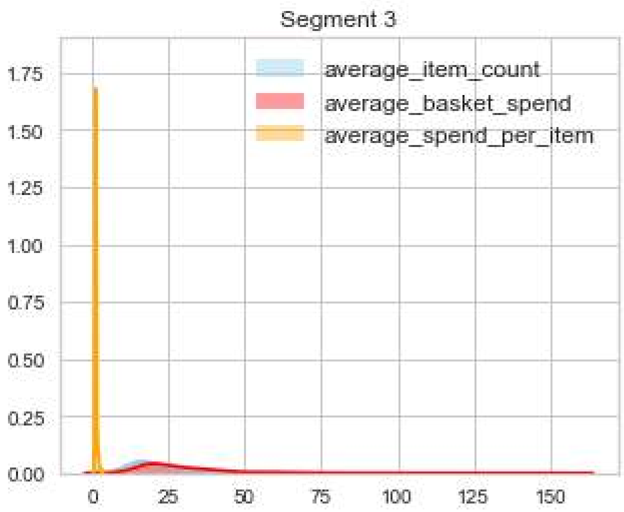
```
fig = plt.figure(figsize=(5,4))
sns.distplot(seg1.average_item_count,color='skyblue',label='average_item_count'
)
sns.distplot(seg1.average_basket_spend,color='red',label='average_basket_spend'
)
sns.distplot(seg1.average_spend_per_item,color='orange',label='average_spend_pe
r_item')

plt.legend(prop={'size': 12})
plt.title('Segment 1')
plt.xlabel('')
plt.show()
```



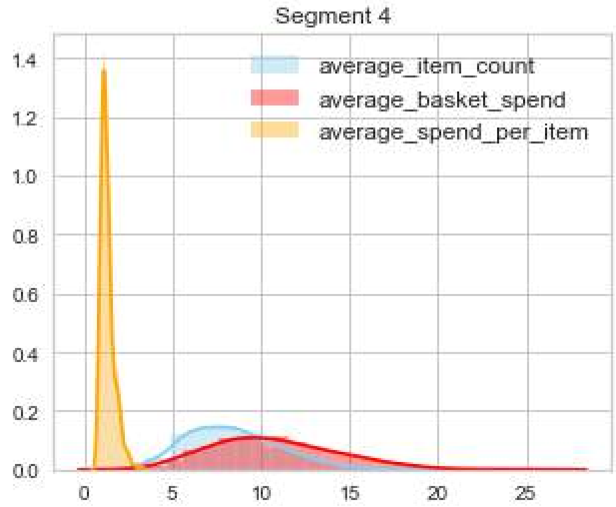
In [151]:
```
fig = plt.figure(figsize=(5,4))
sns.distplot(seg2.average_item_count,color='skyblue',label='average_item_count'
)
sns.distplot(seg2.average_basket_spend,color='red',label='average_basket_spend'
)
sns.distplot(seg2.average_spend_per_item,color='orange',label='average_spend_pe
r_item')

plt.legend(prop={'size': 12})
plt.title('Segment 2')
plt.xlabel('')
plt.show()
```



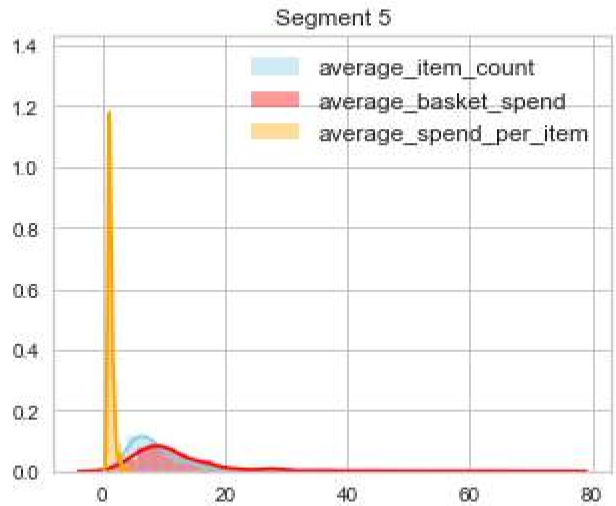
In [152]:
```
fig = plt.figure(figsize=(5,4))
sns.distplot(seg3.average_item_count,color='skyblue',label='average_item_count'
)
sns.distplot(seg3.average_basket_spend,color='red',label='average_basket_spend'
)
sns.distplot(seg3.average_spend_per_item,color='orange',label='average_spend_pe
r_item')

plt.legend(prop={'size': 12})
plt.title('Segment 3')
plt.xlabel('')
plt.show()
```



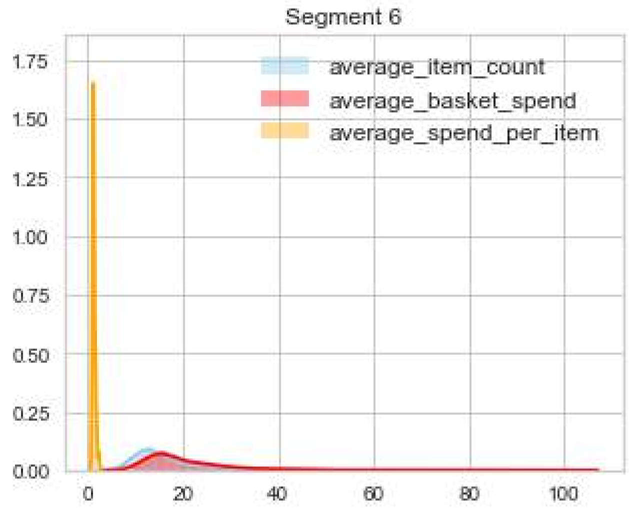
In [153]:
```
fig = plt.figure(figsize=(5,4))
sns.distplot(seg4.average_item_count,color='skyblue',label='average_item_count'
)
sns.distplot(seg4.average_basket_spend,color='red',label='average_basket_spend'
)
sns.distplot(seg4.average_spend_per_item,color='orange',label='average_spend_pe
r_item')

plt.legend(prop={'size': 12})
plt.title('Segment 4')
plt.xlabel('')
plt.show()
```



In [154]:
```
fig = plt.figure(figsize=(5,4))
sns.distplot(seg5.average_item_count,color='skyblue',label='average_item_count'
)
sns.distplot(seg5.average_basket_spend,color='red',label='average_basket_spend'
)
sns.distplot(seg5.average_spend_per_item,color='orange',label='average_spend_pe
r_item')

plt.legend(prop={'size': 12})
plt.title('Segment 5')
plt.xlabel('')
plt.show()
```



In [155]:
```
fig = plt.figure(figsize=(5,4))
sns.distplot(seg6.average_item_count,color='skyblue',
             label='average_item_count')
sns.distplot(seg6.average_basket_spend,color='red',
             label='average_basket_spend')
sns.distplot(seg6.average_spend_per_item,color='orange',label='average_spend_pe
r_item')

plt.legend(prop={'size': 12})
plt.title('Segment 6')
plt.xlabel('')
plt.show()
```



In [156]:
```
round(seg2.describe(),2)
```

Out[156]:

	average_item_count	average_basket_spend	average_spend_per_item	Recency	Frequency
count	692.00	692.00	692.00	692.00	692.00
mean	6.53	7.69	1.21	4.44	69.93
std	2.12	2.69	0.32	12.53	37.08
min	2.14	2.07	0.56	0.00	22.00
25%	4.94	5.72	0.97	0.00	46.00
50%	6.32	7.40	1.16	1.00	59.00
75%	7.73	9.34	1.39	4.00	85.00
max	17.88	16.90	2.38	123.00	329.00

In [157]:
```
round(seg3.describe(),2)
```

Out[157]:

	average_item_count	average_basket_spend	average_spend_per_item	Recency	Frequency
count	401.00	401.00	401.00	401.00	401.00
mean	23.02	29.58	1.32	4.07	62.57
std	12.07	16.18	0.31	8.16	42.59
min	7.41	8.87	0.78	0.00	11.00
25%	15.17	19.22	1.12	0.00	35.00
50%	19.65	25.05	1.26	1.00	52.00
75%	27.64	35.01	1.44	4.00	78.00
max	90.75	152.62	3.51	83.00	348.00

In [158]:
```
round(seg4.describe(),2)
```

Out[158]:

	average_item_count	average_basket_spend	average_spend_per_item	Recency	Frequency
count	846.00	846.00	846.00	846.00	846.00
mean	8.38	10.77	1.32	1.70	102.85
std	2.60	3.61	0.39	4.87	49.62
min	2.73	2.66	0.70	0.00	35.00
25%	6.42	8.18	1.06	0.00	67.00
50%	8.18	10.47	1.23	0.00	90.50
75%	9.93	13.02	1.48	2.00	125.75
max	18.24	25.42	3.25	58.00	374.00

In [159]:
```
round(seg5.describe(),2)
```

Out[159]:

	average_item_count	average_basket_spend	average_spend_per_item	Recency	Frequency
count	273.00	273.00	273.00	273.00	273.00
mean	9.03	12.11	1.39	38.72	19.66
std	6.05	8.57	0.48	44.76	13.00
min	1.20	1.46	0.60	0.00	1.00
25%	5.51	7.30	1.07	5.00	11.00
50%	7.57	10.04	1.25	19.00	17.00
75%	10.44	14.16	1.56	59.00	26.00
max	55.00	73.75	4.15	164.00	100.00

In [160]:
```
round(seg6.describe(),2)
```

Out[160]:

	average_item_count	average_basket_spend	average_spend_per_item	Recency	Frequency
count	573.00	573.00	573.00	573.00	573.00
mean	16.56	21.10	1.31	11.91	27.84
std	7.99	10.35	0.32	21.96	10.96
min	6.85	9.45	0.64	0.00	4.00
25%	11.69	14.56	1.10	2.00	20.00
50%	14.39	17.75	1.24	6.00	27.00
75%	18.85	24.42	1.44	11.00	35.00
max	83.25	100.48	2.88	161.00	65.00

In [ ]:

11. Comparing RFM score in six segments

In [161]:
```
rfm_score.head()
```

Out[161]:

customer_number	Recency	Frequency	Monetary	R	F	M	RFM_Segment	RFM_score
14	1	56	675.72	4	3	3	433	10.0
45	1	33	585.73	4	2	2	422	8.0
52	2	59	222.18	4	3	1	431	8.0
61	3	37	547.87	3	2	2	322	7.0
63	7	48	293.34	2	2	1	221	5.0

In [162]:
```
rfm_score = rfm_score.rename_axis(columns = None).reset_index()
```

In [163]:
```
rfm_score.insert(loc=0, column='number', value=np.arange(len(rfm_score)))
```

In [164]:
```
rfm_score = rfm_score.drop(columns=['customer_number','Recency','Frequency','Mo
netary',
                                  'R','F','M','RFM_Segment'])
rfm_score.head()
```

Out[164]:

	number	RFM_score
0	0	10.0
1	1	8.0
2	2	8.0
3	3	7.0
4	4	5.0

In [165]:
```
rfm1 = rfm_score.loc[rfm_score['number'].isin(clu1)]
rfm1 = rfm1.drop(columns='number')
rfm1.head()
```

Out[165]:

	RFM_score
56	8.0
81	7.0
131	8.0
140	9.0
180	10.0

In [166]:
```
rfm1.describe()
```

Out[166]:

	RFM_score
count	215.000000
mean	8.520930
std	2.045802
min	4.000000
25%	7.000000
50%	8.000000
75%	10.000000
max	12.000000

In [167]:
```
rfm2 = rfm_score.loc[rfm_score['number'].isin(clu2)]
rfm2 = rfm2.drop(columns='number')
rfm2.head()
```

Out[167]:

	RFM_score
4	5.0
11	9.0
12	8.0
15	8.0
21	7.0

In [168]:
```
rfm2.describe()
```

Out[168]:

	RFM_score
count	692.000000
mean	8.132948
std	1.683223
min	4.000000
25%	7.000000
50%	8.000000
75%	9.000000
max	12.000000

In [169]:
```
rfm3 = rfm_score.loc[rfm_score['number'].isin(clu3)]
rfm3 = rfm3.drop(columns='number')
rfm3.head()
```

Out[169]:

	RFM_score
5	10.0
7	9.0
20	6.0
26	11.0
46	10.0

In [170]:
```
rfm3.describe()
```

Out[170]:

	RFM_score
count	401.000000
mean	9.708229
std	1.664980
min	6.000000
25%	9.000000
50%	10.000000
75%	11.000000
max	12.000000

In [171]:
```
rfm4 = rfm_score.loc[rfm_score['number'].isin(clu4)]
rfm4 = rfm4.drop(columns='number')
rfm4.head()
```

Out[171]:

	RFM_score
0	10.0
10	7.0
14	8.0
16	10.0
19	9.0

In [172]:
```
rfm4.describe()
```

Out[172]:

	RFM_score
count	846.000000
mean	10.522459
std	1.318540
min	7.000000
25%	10.000000
50%	11.000000
75%	12.000000
max	12.000000

In [173]:
```
rfm5 = rfm_score.loc[rfm_score['number'].isin(clu5)]
rfm5 = rfm5.drop(columns='number')
rfm5.head()
```

Out[173]:

	RFM_score
2	8.0
13	5.0
23	4.0
42	6.0
53	4.0

In [174]:
```
rfm5.describe()
```
Out[174]:

	RFM_score
count	273.000000
mean	4.593407
std	0.915287
min	4.000000
25%	4.000000
50%	4.000000
75%	5.000000
max	9.000000

In [175]:
```
rfm6 = rfm_score.loc[rfm_score['number'].isin(clu6)]
rfm6 = rfm6.drop(columns='number')
rfm6.head()
```
Out[175]:

	RFM_score
1	8.0
3	7.0
6	6.0
8	6.0
9	8.0

In [176]:
```
rfm6.describe()
```
Out[176]:

	RFM_score
count	573.000000
mean	6.209424
std	1.379716
min	4.000000
25%	5.000000
50%	6.000000
75%	7.000000
max	10.000000

In [ ]:
```

```
In [177]:
```
seg_rfm_score = pd.merge(rfm1.describe(), rfm2.describe(),
                         left_index=True, right_index=True, suffixes=('', '2'))
```
In [178]:
```
seg_rfm_score = pd.merge(seg_rfm_score, rfm3.describe(),
                         left_index=True, right_index=True, suffixes=('', '2'))
```

In [179]:
```python
seg_rfm_score = pd.merge(seg_rfm_score, rfm4.describe(),
                         left_index=True, right_index=True, suffixes=('', '2'))
```

In [180]:
```python
seg_rfm_score = pd.merge(seg_rfm_score, rfm5.describe(),
                         left_index=True, right_index=True, suffixes=('', '2'))
```

In [181]:
```python
seg_rfm_score = pd.merge(seg_rfm_score, rfm6.describe(),
                         left_index=True, right_index=True, suffixes=('', '2'))
```

In [182]:
```python
seg_rfm_score
```

Out[182]:

	RFM_score	RFM_score2	RFM_score2	RFM_score2	RFM_score2	RFM_score2
count	215.000000	692.000000	401.000000	846.000000	273.000000	573.000000
mean	8.520930	8.132948	9.708229	10.522459	4.593407	6.209424
std	2.045802	1.683223	1.664980	1.318540	0.915287	1.379716
min	4.000000	4.000000	6.000000	7.000000	4.000000	4.000000
25%	7.000000	7.000000	9.000000	10.000000	4.000000	5.000000
50%	8.000000	8.000000	10.000000	11.000000	4.000000	6.000000
75%	10.000000	9.000000	11.000000	12.000000	5.000000	7.000000
max	12.000000	12.000000	12.000000	12.000000	9.000000	10.000000

In [183]:
```python
# rename the columns
seg_rfm_score.rename(columns = {
    seg_rfm_score.columns[0] :'Seg1_RFM_score',
    seg_rfm_score.columns[1] :'Seg2_RFM_score',
    seg_rfm_score.columns[2] :'Seg3_RFM_score',
    seg_rfm_score.columns[3] :'Seg4_RFM_score',
    seg_rfm_score.columns[4] :'Seg5_RFM_score',
    seg_rfm_score.columns[5] :'Seg6_RFM_score'}, inplace=True )
seg_rfm_score.head()
```

Out[183]:

	Seg1_RFM_score	Seg6_RFM_score	Seg6_RFM_score	Seg6_RFM_score	Seg6_RFM_score
count	215.000000	692.000000	401.000000	846.000000	273.000000
mean	8.520930	8.132948	9.708229	10.522459	4.593407
std	2.045802	1.683223	1.664980	1.318540	0.915287
min	4.000000	4.000000	6.000000	7.000000	4.000000
25%	7.000000	7.000000	9.000000	10.000000	4.000000

In [184]:
```
seg_rfm_score
```

Out[184]:

	Seg1_RFM_score	Seg6_RFM_score	Seg6_RFM_score	Seg6_RFM_score	Seg6_RFM_score
count	215.000000	692.000000	401.000000	846.000000	273.000000
mean	8.520930	8.132948	9.708229	10.522459	4.593407
std	2.045802	1.683223	1.664980	1.318540	0.915287
min	4.000000	4.000000	6.000000	7.000000	4.000000
25%	7.000000	7.000000	9.000000	10.000000	4.000000
50%	8.000000	8.000000	10.000000	11.000000	4.000000
75%	10.000000	9.000000	11.000000	12.000000	5.000000
max	12.000000	12.000000	12.000000	12.000000	9.000000

In [ ]:

12. Extra work

Comparing result with hierarchical clustering

In [185]:
```
df_log.head()
```

Out[185]:

	average_item_count	average_basket_spend	average_spend_per_item	Recency
customer_number				
14	2.349469	2.570320	0.819780	0.693147
45	3.037354	2.931194	0.636577	0.693147
52	1.788421	1.562346	0.565314	1.098612
61	2.673459	2.760643	0.741937	1.386294
63	1.924249	1.961502	0.712950	2.079442

In [186]:
```
from sklearn.cluster import AgglomerativeClustering
```

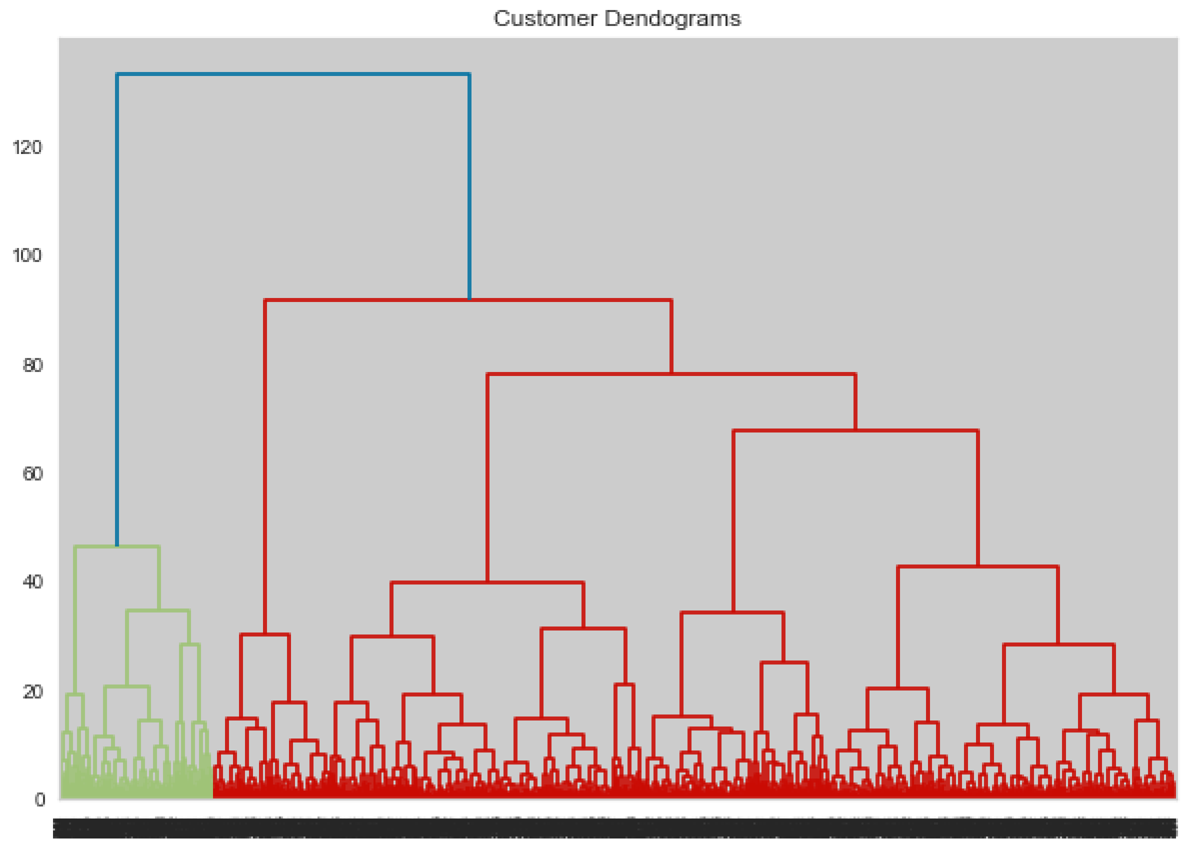
In [187]:

```
pca = PCA(n_components=4)
pca.fit(df_scaled)
reduced_data = pca.transform(df_scaled)
reduced_data = pd.DataFrame(reduced_data)

for k in range(4,8):
    clusterer = AgglomerativeClustering(n_clusters=k,linkage='average',
                                        affinity= 'l1')
    preds = clusterer.fit_predict(reduced_data)
    score = silhouette_score(reduced_data, preds, metric='euclidean')
    print("For n_clusters = {}. The average silhouette_score is : {})".format(k
, score))
```

```
For n_clusters = 4. The average silhouette_score is : 0.30836116820649206)
For n_clusters = 5. The average silhouette_score is : 0.22646269295017685)
For n_clusters = 6. The average silhouette_score is : 0.21795159143336104)
For n_clusters = 7. The average silhouette_score is : 0.2128616575031614)
```

In [188]:

```
import scipy.cluster.hierarchy as shc

plt.figure(figsize=(10, 7))
plt.title("Customer Dendograms")
dend = shc.dendrogram(shc.linkage(reduced_data, method='ward'))
```



In [ ]:

Coursework: Predicting the potential customers

- University of Nottingham (UK), MSc Business Analytics
- Lecture: Foundational Business Analytics
- Year: 2019
- Language:Python

The Problem

Predicting potential customers who will accept new N/LAB Platinum Deposit. Demographic and personal data that identified in previous a product has been used. The data is composed with total 4,000 customer's with 17 features. The featrues are as follows. Age, job, marital, education, default, balance, housing, loan, contact, day, duration, campaign, pdays, previous, poutcome and y.

The Process of Data Analytics

Summarization

Statistical analysis to find the relationship between independent variables and dependent variables. Used various graphs to visualize what relationships they have.

Exploration

Decision trees determined which dependent variables divide the data. Compare the variables with that I have found in the previous step.

Model Evaluation

Decision Tree, Random Forest, KNN, Logisitc classification models has applied. Explain the reasons for the model selection, find the hyperparameters of each model, and how the evaluation strategy was used using precision and f1 scores. In order to reduce the cost of time and effort that calling non-potential customers, I chose presision evaluation to lower false positives.

Final Assessment

Explain the final model.

Model Implementation

Prepare the final model to implement with a brief description.

Business Case Recommendations

Found two potential customer groups for the company. One group is who have purchased previous products, regardless of having a house loan, and the other group is also who have purchased previous products and who recently contacted to the group. Thus, marketing strategies need to focus on those two groups.

Report

[https://github.com/Chan-Young/Coursework/blob/main/Classification_predict%20customers.pdf (https://github.com/Chan-Young/Coursework/blob/main/Classification_predict%20customers.pdf)](https://github.com/Chan-Young/Coursework/blob/main/Classification_predict%20customers.pdf)

Package preparation

In [6]:

```
'''Installed'''
import numpy as np
import pandas as pd
import matplotlib
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns
import warnings
warnings.filterwarnings("ignore")
```

In [7]:

```
'''Additonal package requirements'''
from sklearn import tree
from sklearn.model_selection import train_test_split
import graphviz
import os
os.environ["PATH"] += os.pathsep + 'C:/Program Files (x86)/Graphviz2.38/bin/'
os.system('dot -Tpng random.dot -o random.png')
from sklearn.metrics import classification_report
from sklearn import metrics
from sklearn.tree import DecisionTreeClassifier
from yellowbrick.classifier import ROCAUC
from sklearn.metrics import roc_auc_score
from sklearn.model_selection import RandomizedSearchCV
from subprocess import call
from IPython.display import Image
from sklearn import preprocessing
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import GridSearchCV
from sklearn.linear_model import LogisticRegression
import statsmodels.api as sm
from sklearn.ensemble import RandomForestClassifier
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import cross_val_predict
from sklearn.metrics import confusion_matrix
from sklearn import dummy
```

Read file

In [4]:
```
# Read file
df = pd.read_csv('lixcl68.csv')

yes = df[df.y == 'yes']
no = df[df.y == 'no']
```

Section A: Summarization

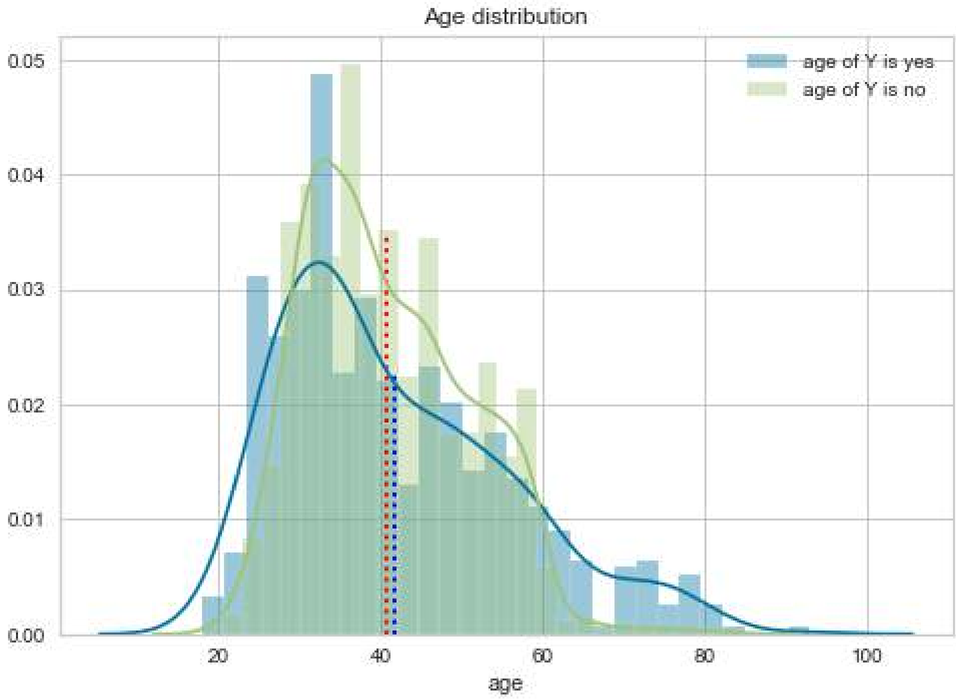
In [4]:
```
'''
Summary:
    input variable ‘default’, ‘loan’, ‘housing’, ‘duration’ and ‘poutcome’
    shows highly related to variable ‘y’.
'''
```

Out[4]:
```
'\nSummary: \n    input variable ‘default’, ‘loan’, ‘housing’, ‘duration’ and
‘poutcome’ \n    shows highly related to variable ‘y’.\n'
```

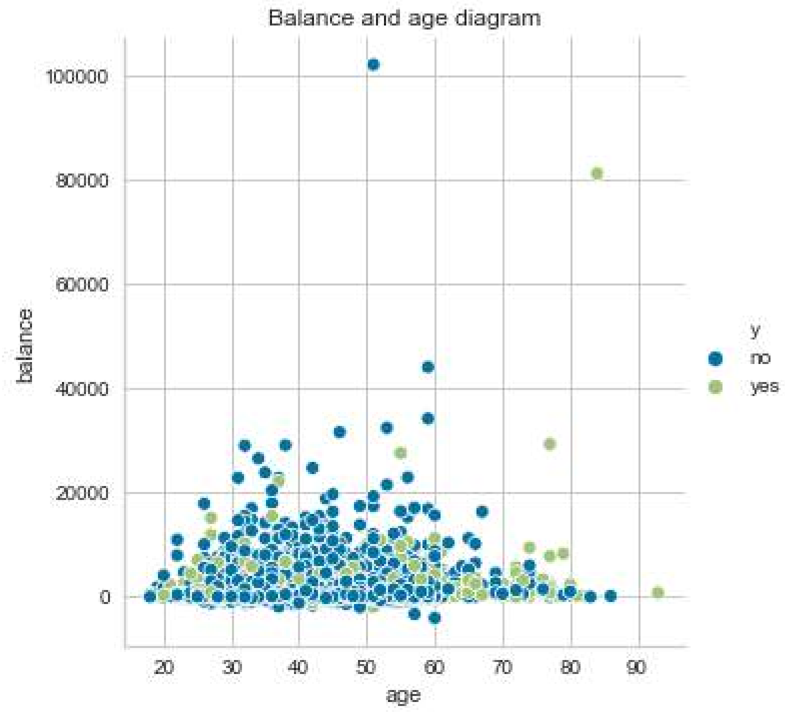
A-1. Summary statistical analysis of numerical variables

(Numeric: age, balance, day, duration, campaign, pdays and preivous)

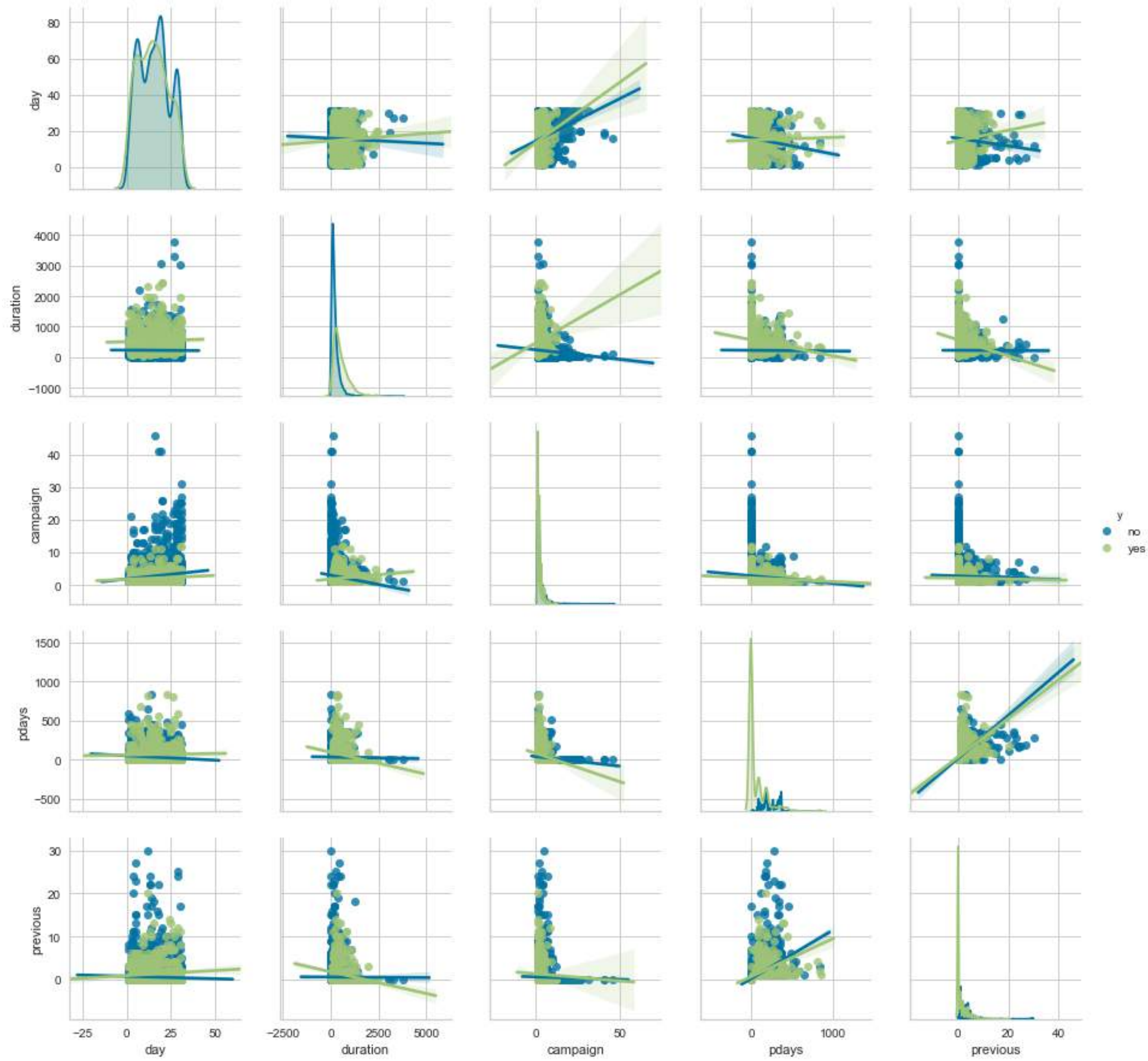
In [5]:
```
# age distribution by Y
fig = plt.figure()

for dataset in (yes.age, no.age):
    sns.distplot(dataset, bins=28)
plt.legend(['age of Y is yes', 'age of Y is no'])
plt.title('Age distribution')
plt.xlabel('age')

plt.vlines(x = 41.789565, ymin = 0, ymax = 0.023, color = 'blue', linestyle =
'dotted', linewidth = 2)
plt.vlines(x = 40.776497, ymin = 0, ymax = 0.035, color = 'red', linestyle = 'd
otted', linewidth = 2)

fig.savefig('age distribution by Y.png', dpi=fig.dpi)
```



In [6]:
```python
# Balance and age diagram
balance_age = sns.relplot(x='age', y='balance', hue='y', data=df)
plt.title('Balance and age diagram')
balance_age.savefig("balance_age.png")
```

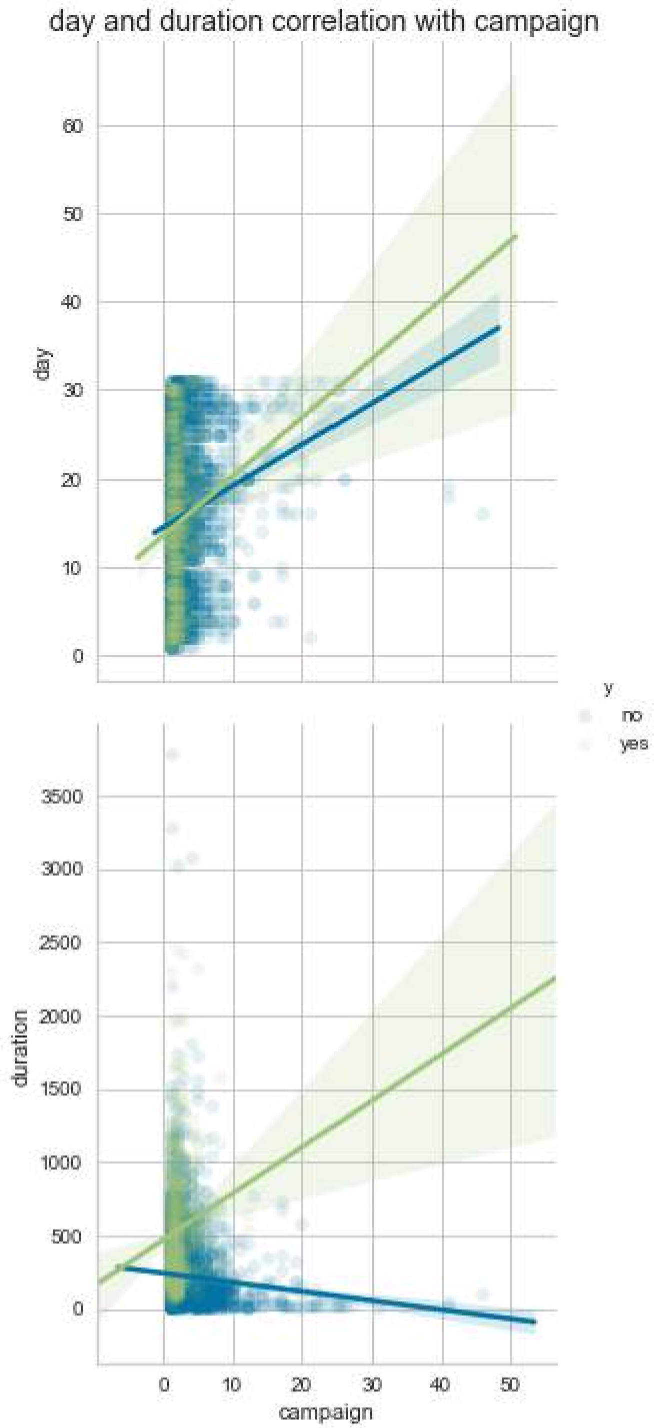


In [7]:
```
# Correlation between day, duration, campaign, pdays and preivous
sns.pairplot(df, vars = ['day', 'duration', 'campaign', 'pdays', 'previous'], h
ue='y', kind='reg')
plt.show()
```

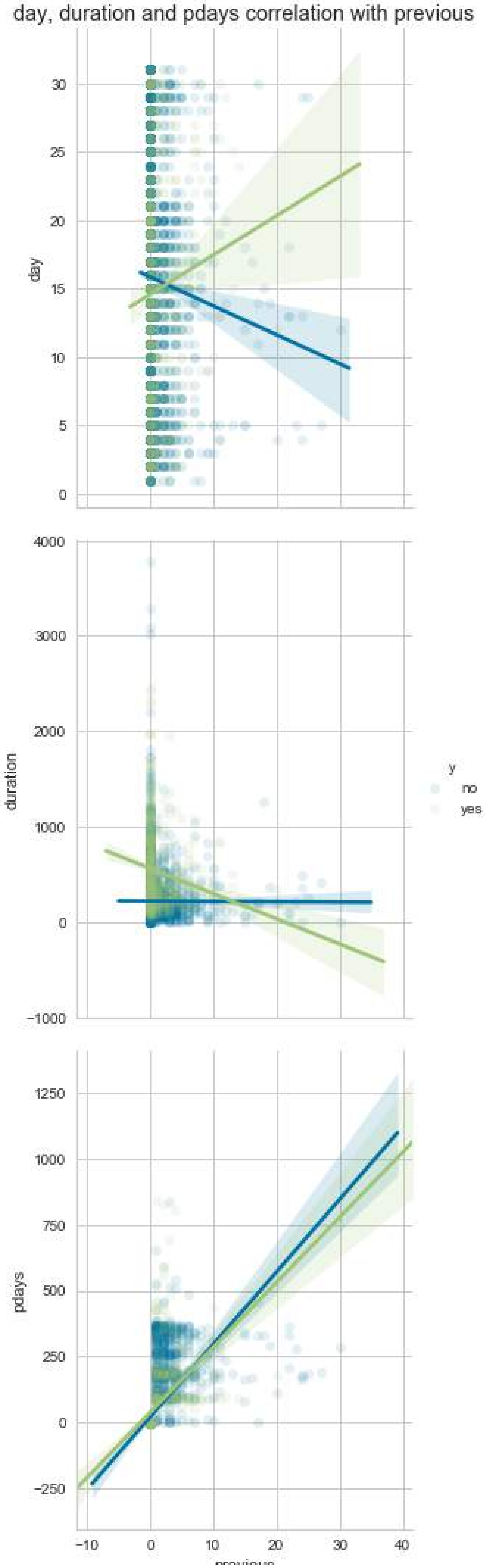


In [8]:

```
# Day and duration correlation with campaign
campaign_relation = sns.pairplot(df, x_vars=['campaign'], y_vars=['day', 'durat
ion'],
             hue='y',height=5, aspect=.8, kind='reg',
             plot_kws={'scatter_kws': {'alpha': 0.1}})
campaign_relation.fig.suptitle('day and duration correlation with campaign', y
= 1)
campaign_relation.savefig("campaign_relation.png")
```



In [9]:
```
# Day, duration and pdays correlation with previous
previous_relation = sns.pairplot(df, x_vars=['previous'], y_vars=['day', 'durat
ion','pdays'],
             hue='y',height=5, aspect=.8, kind='reg',
            plot_kws={'scatter_kws': {'alpha': 0.1}})
previous_relation.fig.suptitle('day, duration and pdays correlation with previo
us', y = 1)
previous_relation.savefig("previous_relation.png")
```

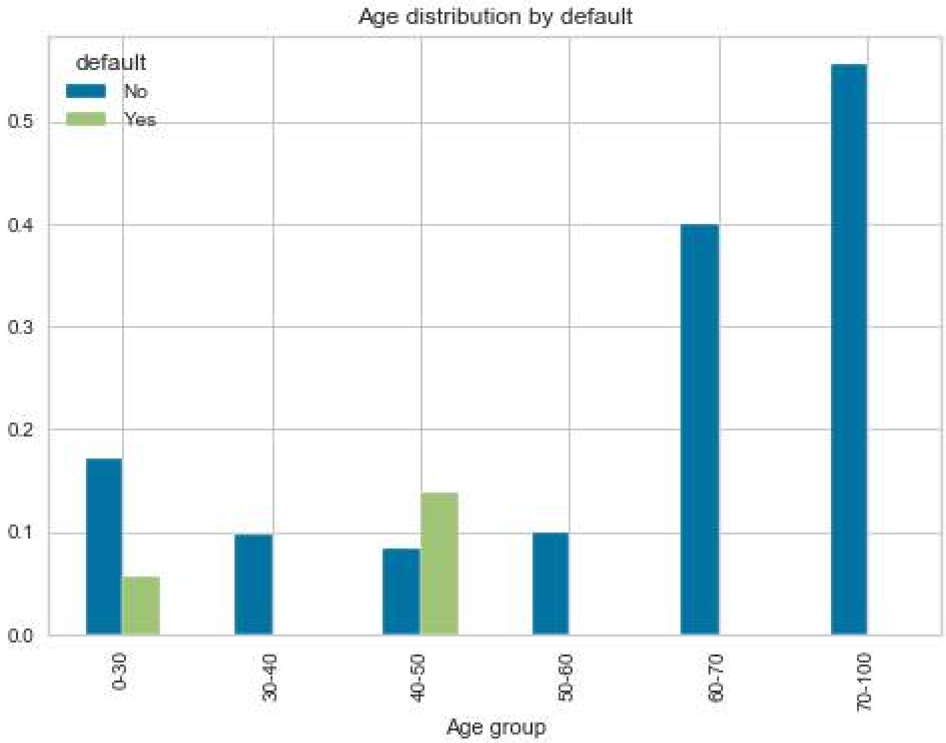

day, duration and pdays correlation with previous
day
duration
pdays
previous
y
no
yes

A-2. Summary statistical analysis of categorical variables

(Categorical: job, marital, education, default, housing, loan, contact and poutcome)

In [10]:
```
# Preparation for analyzing age variable
df['ageGroup'] = pd.cut(df.age,[0, 30, 40, 50, 60, 70, 100], labels=['0-30', '30-40', '40-50', '50-60', '60-70', '70-100'])
df['ageGroup'].head()

byage = df.groupby(['ageGroup','default']).y.value_counts(normalize=True)
byage2 = byage.unstack().drop('no', axis=1).unstack()

byage2.columns = ['No', 'Yes']
byage2.columns.name = 'default'
```

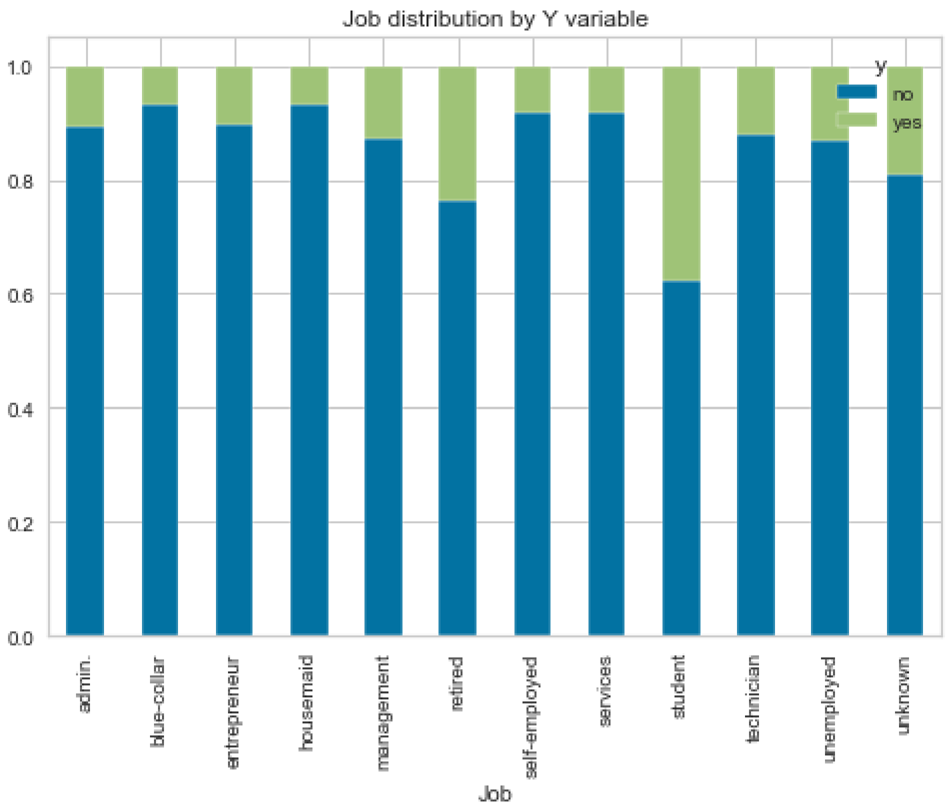
In [11]:
```
# Age distribution by default
byage2.plot(kind='bar')
plt.title('Age distribution by default')
plt.xlabel('Age group')
plt.savefig('Age distribution by default.png')
```



In [12]:
```
# Preparation
byjob = df.groupby('job').y.value_counts(normalize=True)
bymarital= df.groupby('marital').y.value_counts(normalize=True)
byeducation = df.groupby('education').y.value_counts(normalize=True)
bydefault = df.groupby('default').y.value_counts(normalize=True)
byhouse = df.groupby('housing').y.value_counts(normalize=True)
byloan = df.groupby('loan').y.value_counts(normalize=True)
bycontact = df.groupby('contact').y.value_counts(normalize=True)
bypoutcome = df.groupby('poutcome').y.value_counts(normalize=True)
```

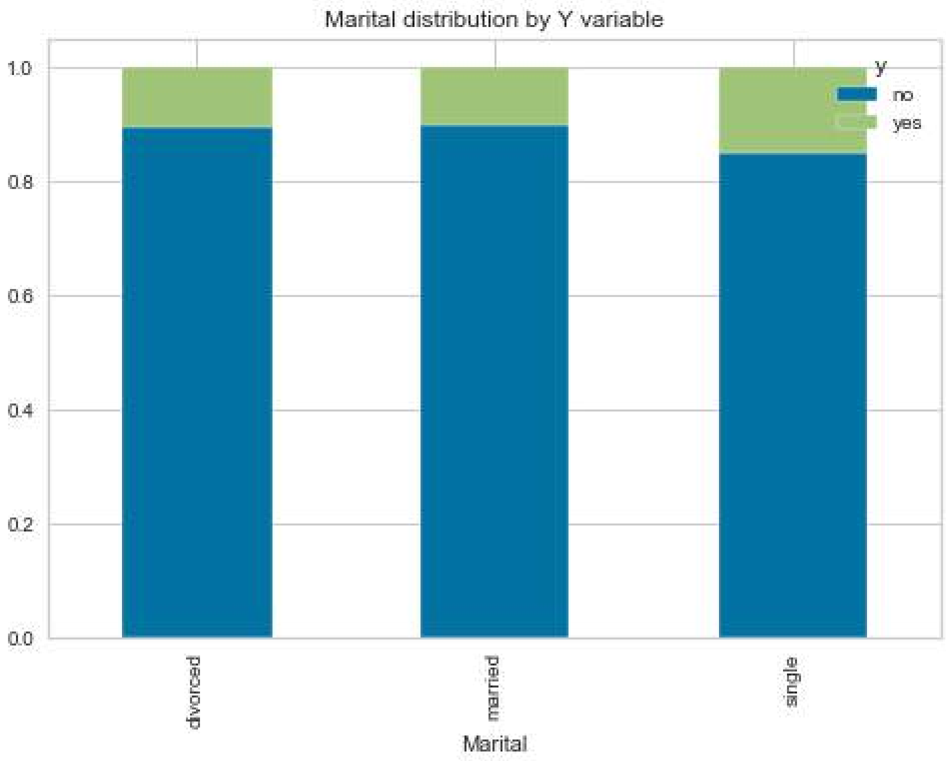
In [13]:
```python
# Job distribution by Y variable
byjob.unstack().plot(kind='bar', stacked=True)
plt.title('Job distribution by Y variable')
plt.xlabel('Job')
```

Out[13]: Text(0.5, 0, 'Job')



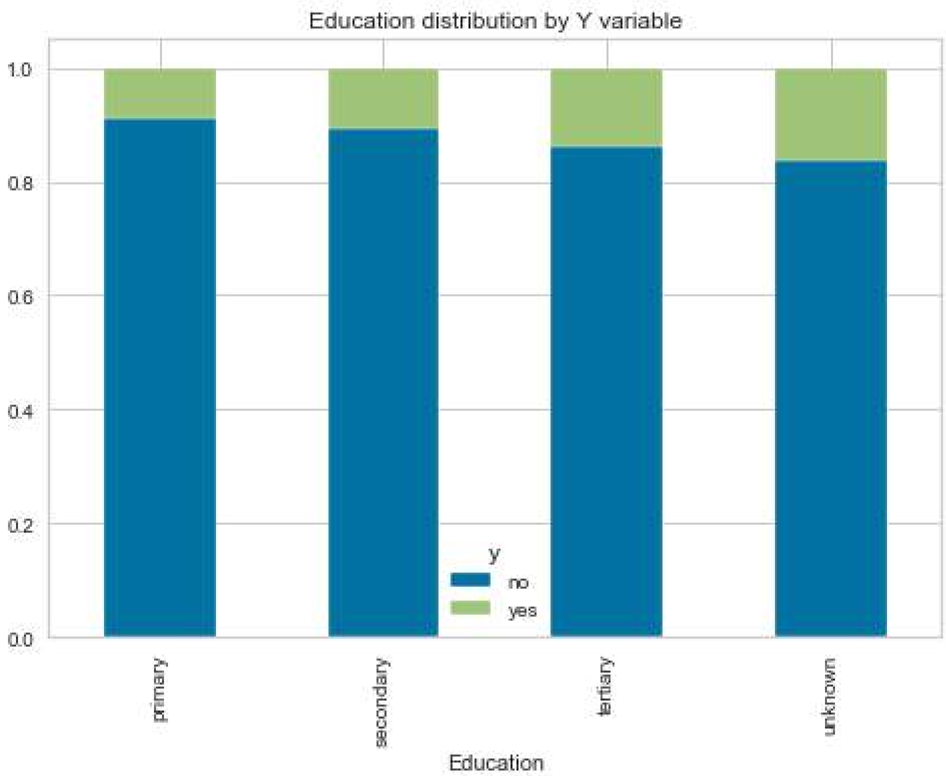
In [14]:
```
# Marital distribution by Y variable
bymarital.unstack().plot(kind='bar', stacked=True)
plt.title('Marital distribution by Y variable')
plt.xlabel('Marital')
```

Out[14]: Text(0.5, 0, 'Marital')

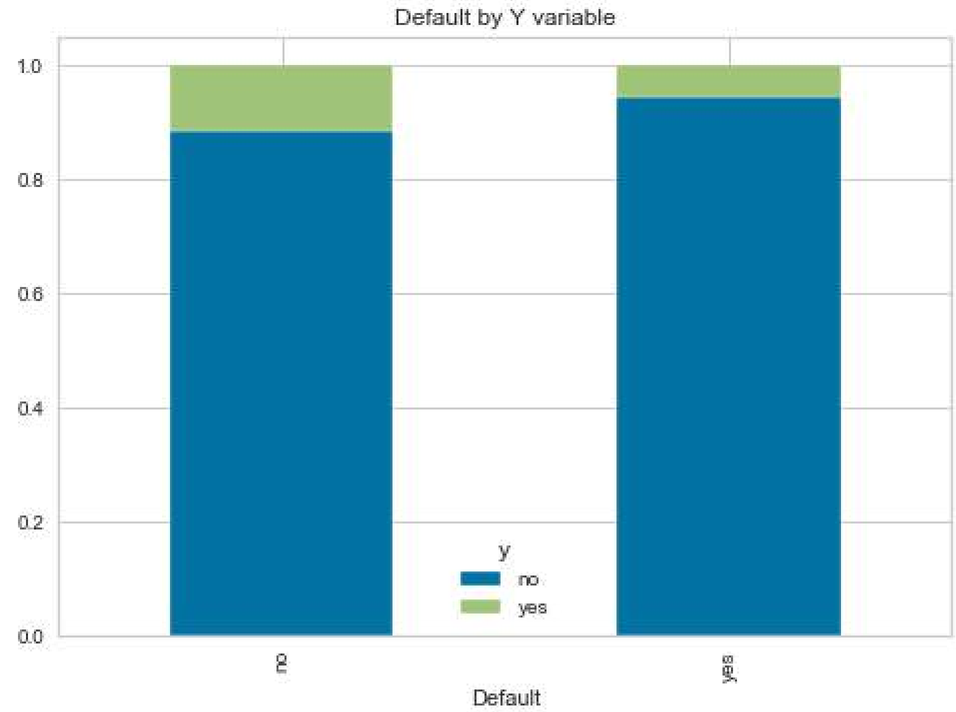


In [15]:
```
# Education distribution by Y variable
byeducation.unstack().plot(kind='bar', stacked=True)
plt.title('Education distribution by Y variable')
plt.xlabel('Education')
```

Out[15]: Text(0.5, 0, 'Education')

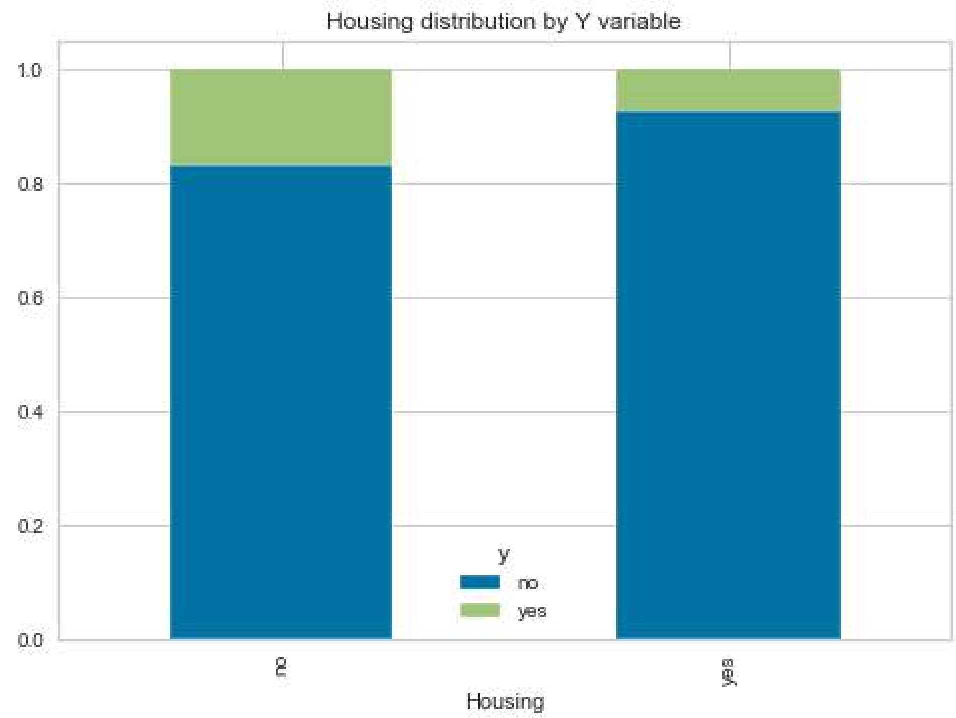


In [16]:
```
# Default by Y variable
bydefault.unstack().plot(kind='bar', stacked=True)
plt.title('Default by Y variable')
plt.xlabel('Default')
plt.savefig('Default by Y variable.png')
```



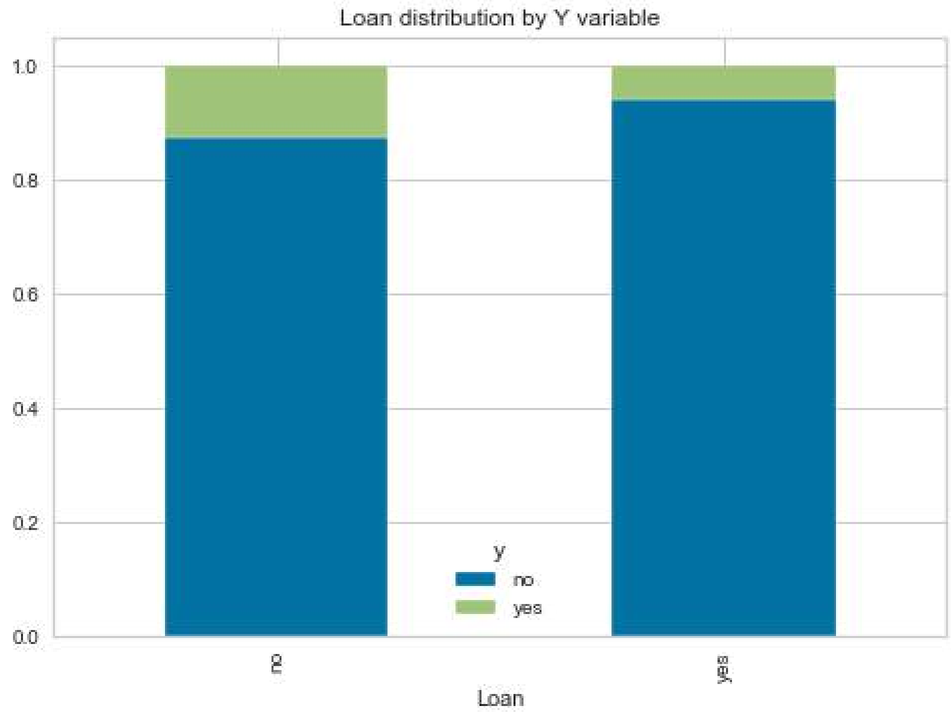
In [17]:
```
# Housing distribution by Y variable
byhouse.unstack().plot(kind='bar', stacked=True)
plt.title('Housing distribution by Y variable')
plt.xlabel('Housing')
```

Out[17]: Text(0.5, 0, 'Housing')



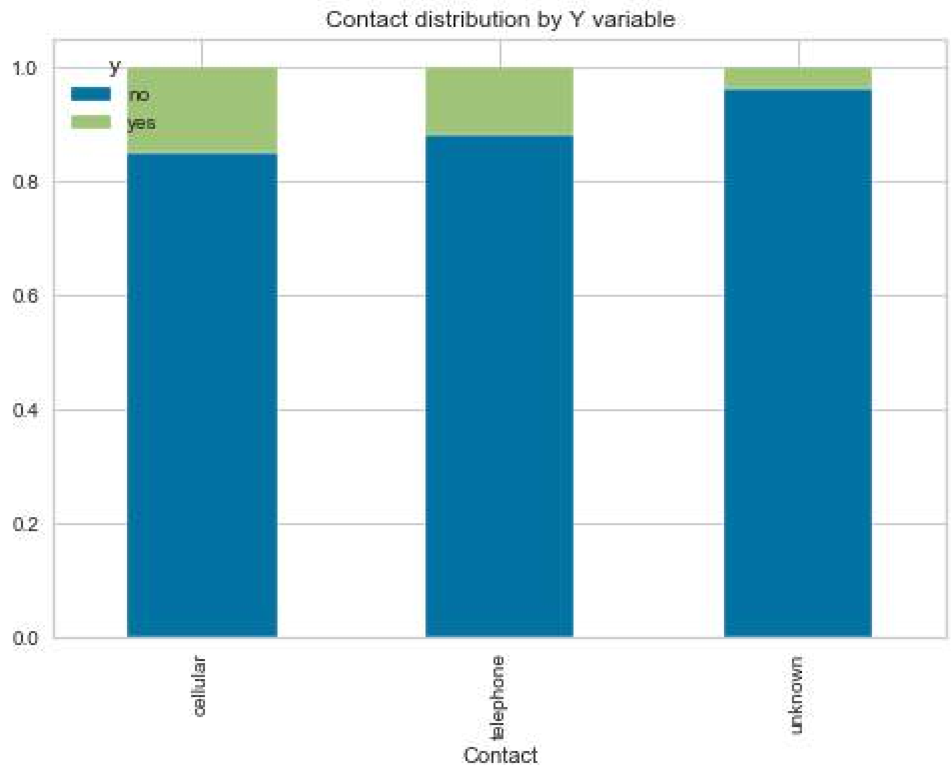
In [18]:
```python
# Loan distribution by Y variable
byloan.unstack().plot(kind='bar', stacked=True)
plt.title('Loan distribution by Y variable')
plt.xlabel('Loan')
```

Out[18]: Text(0.5, 0, 'Loan')



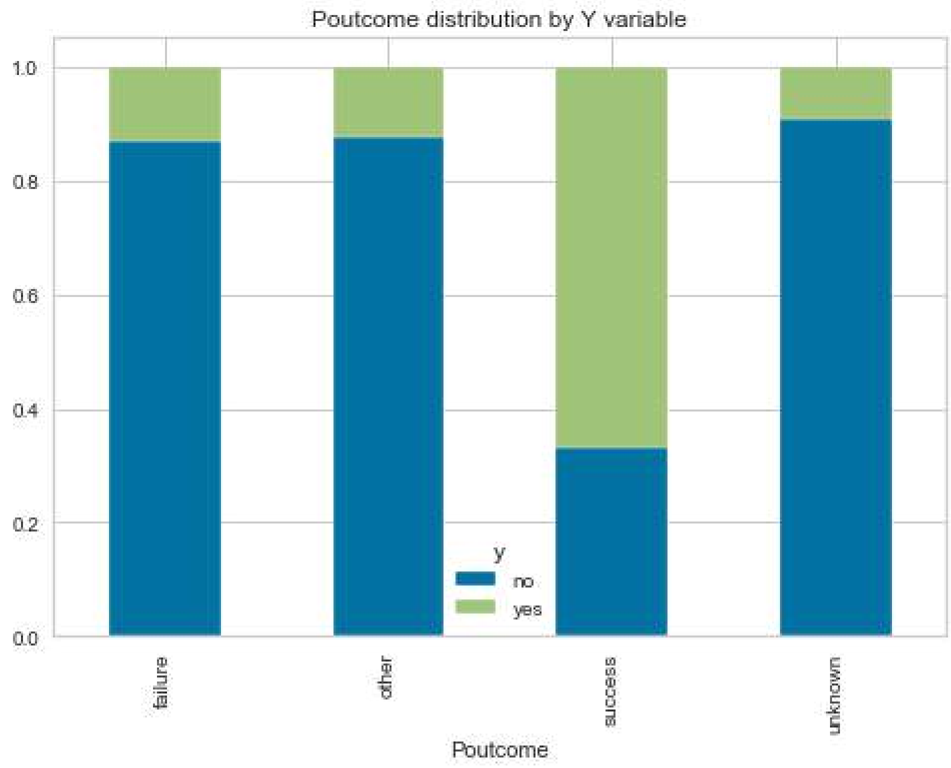
In [19]:
```python
# Contact distribution by Y variable
bycontact.unstack().plot(kind='bar', stacked=True)
plt.title('Contact distribution by Y variable')
plt.xlabel('Contact')
```

Out[19]: Text(0.5, 0, 'Contact')



In [20]:
```
# Poutcome distribution by Y variable
bypoutcome.unstack().plot(kind='bar', stacked=True)
plt.title('Poutcome distribution by Y variable')
plt.xlabel('Poutcome')
```

Out[20]: Text(0.5, 0, 'Poutcome')



Section B: Exploration

B-1. Data collation and data cleaning

In [5]:

```
df.y = df.y.replace('no',0)
df.y = df.y.replace('yes',1)

df.job = df.job.replace('retired',2)
df.job = df.job.replace('technician',3)
df.job = df.job.replace('self-employed',4)
df.job = df.job.replace('blue-collar',5)
df.job = df.job.replace('student',6)
df.job = df.job.replace('admin.',7)
df.job = df.job.replace('management',8)
df.job = df.job.replace('entrepreneur',9)
df.job = df.job.replace('housemaid',10)
df.job = df.job.replace('services',11)
df.job = df.job.replace('unemployed',12)
df.job = df.job.replace('unknown',13)

df.marital = df.marital.replace('married',14)
df.marital = df.marital.replace('single',15)
df.marital = df.marital.replace('divorced',16)

df.education = df.education.replace('unknown',17)
df.education = df.education.replace('tertiary',18)
df.education = df.education.replace('primary',19)
df.education = df.education.replace('secondary',20)

df.default = df.default.replace('yes',21)
df.default = df.default.replace('no',22)

df.housing = df.housing.replace('yes',23)
df.housing = df.housing.replace('no',24)

df.loan = df.loan.replace('yes',25)
df.loan = df.loan.replace('no',26)

df.contact = df.contact.replace('unknown',27)
df.contact = df.contact.replace('cellular',28)
df.contact = df.contact.replace('telephone',29)

df.poutcome = df.poutcome.replace('unknown',30)
df.poutcome = df.poutcome.replace('other',31)
df.poutcome = df.poutcome.replace('failure',32)
df.poutcome = df.poutcome.replace('success',33)
```

B-2. Finding important variables

(Through statistical analysis, default, housing, loan, duration and poutcome variable verified as a hghly relatied variables to variable y)

```
In [22]: '''
default variables: default, housing, loan, duration and poutcome
examine contact, day, campaign, pdays and previous variable by decision tree
whether it is useful variable or not

optimal : default, housing, loan, duration and poutcome
'''
```

```
Out[22]: '\ndefault variables: default, housing, loan, duration and poutcome\nexamine co
ntact, day, campaign, pdays and previous variable by decision tree \nwhether it
is useful variable or not\n\noptimal : default, housing, loan, duration and pou
tcome\n'
```

B2-1. Decision tree using default, housing, loan, duration, poutcome

```
In [6]: x = df[['default','housing','loan','duration','poutcome']]
y = df[['y']]
x_train, x_test, y_train, y_test = train_test_split(x,y,
                                                    test_size=0.25, random_stat
e=0, stratify=y)
tree = DecisionTreeClassifier(criterion='entropy', random_state=0)
tree.fit(x_train, y_train)
print('Train set score  : {:.3f}'.format(tree.score(x_train, y_train)))
print('Test set score  : {:.3f}'.format(tree.score(x_test, y_test)))
y_pred = tree.predict(x_test)
print(classification_report(y_pred, y_test))
```

```
Train set score  : 0.969
Test set score  : 0.868
              precision    recall  f1-score   support

           0       0.93      0.92      0.93      1121
           1       0.38      0.42      0.40       129

    accuracy                           0.87      1250
   macro avg       0.65      0.67      0.66      1250
weighted avg       0.87      0.87      0.87      1250
```

B2-2. Decision tree using default, housing, loan, duration, poutcome + contact

In [7]:
```
x2 = df[['default','housing','loan','duration','poutcome','contact']]
y2 = df[['y']]
x2_train, x2_test, y2_train, y2_test = train_test_split(x2,y2,
                                                     test_size=0.25, random_stat
e=0, stratify=y)
tree2 = DecisionTreeClassifier(criterion='entropy', random_state=0)
tree2.fit(x2_train, y2_train)
print('Train set score  : {:.3f}'.format(tree2.score(x2_train, y2_train)))
print('Test set score  : {:.3f}'.format(tree2.score(x2_test, y2_test)))
y_pred = tree2.predict(x2_test)
print(classification_report(y_pred, y2_test))
```

```
Train set score  : 0.976
Test set score  : 0.864
              precision    recall  f1-score   support

           0       0.93      0.92      0.92      1118
           1       0.37      0.40      0.38       132

    accuracy                           0.86      1250
   macro avg       0.65      0.66      0.65      1250
weighted avg       0.87      0.86      0.87      1250
```

B2-3. Decision tree using default, housing, loan, duration, poutcome + day

In [25]:
```
x3 = df[['default','housing','loan','duration','poutcome','day']]
y3 = df[['y']]
x3_train, x3_test, y3_train, y3_test = train_test_split(x3,y3,
                                                     test_size=0.25, random_stat
e=0, stratify=y)
tree3 = DecisionTreeClassifier(criterion='entropy', random_state=0)
tree3.fit(x3_train, y3_train)
print('Train set score 2 : {:.3f}'.format(tree3.score(x3_train, y3_train)))
print('Test set score 2 : {:.3f}'.format(tree3.score(x3_test, y3_test)))
y_pred = tree3.predict(x3_test)
print(classification_report(y_pred, y3_test))
```

```
Train set score 2 : 0.996
Test set score 2 : 0.850
              precision    recall  f1-score   support

           0       0.91      0.92      0.91      1096
           1       0.38      0.36      0.37       154

    accuracy                           0.85      1250
   macro avg       0.65      0.64      0.64      1250
weighted avg       0.85      0.85      0.85      1250
```

B2-4. Decision tree using default, housing, loan, duration, poutcome + campaign

```
In [8]: x4 = df[['default','housing','loan','duration','poutcome','campaign']]
y4 = df[['y']]
x4_train, x4_test, y4_train, y4_test = train_test_split(x4,y4,
                                                        test_size=0.25, random_stat
e=0, stratify=y)
tree4 = DecisionTreeClassifier(criterion='entropy', random_state=0)
tree4.fit(x4_train, y4_train)
print('Train set score 2 : {:.3f}'.format(tree4.score(x4_train, y4_train)))
print('Test set score 2 : {:.3f}'.format(tree4.score(x4_test, y4_test)))
y_pred = tree4.predict(x4_test)
print(classification_report(y_pred, y4_test))
```

```
Train set score 2 : 0.985
Test set score 2 : 0.854
              precision    recall  f1-score   support

           0       0.92      0.92      0.92      1101
           1       0.38      0.37      0.38       149

    accuracy                           0.85      1250
   macro avg       0.65      0.64      0.65      1250
weighted avg       0.85      0.85      0.85      1250
```

B2-5. Decision tree using default, housing, loan, duration, poutcome + pdays

```
In [9]: x5 = df[['default','housing','loan','duration','poutcome','pdays']]
y5 = df[['y']]
x5_train, x5_test, y5_train, y5_test = train_test_split(x5,y5,
                                                        test_size=0.25, random_stat
e=0, stratify=y)
tree5 = DecisionTreeClassifier(criterion='entropy', random_state=0)
tree5.fit(x5_train, y5_train)
print('Train set score 2 : {:.3f}'.format(tree5.score(x5_train, y5_train)))
print('Test set score 2 : {:.3f}'.format(tree5.score(x5_test, y5_test)))
y_pred = tree5.predict(x5_test)
print(classification_report(y_pred, y5_test))
```

```
Train set score 2 : 0.971
Test set score 2 : 0.868
              precision    recall  f1-score   support

           0       0.94      0.92      0.93      1131
           1       0.34      0.41      0.37       119

    accuracy                           0.87      1250
   macro avg       0.64      0.66      0.65      1250
weighted avg       0.88      0.87      0.87      1250
```

B2-6. Decision tree using default, housing, loan, duration, poutcome + previous

In [10]:
```
x6 = df[['default','housing','loan','duration','poutcome','previous']]
y6 = df[['y']]
x6_train, x6_test, y6_train, y6_test = train_test_split(x6,y6,
                                                    test_size=0.25, random_stat
e=0, stratify=y)
tree6 = DecisionTreeClassifier(criterion='entropy', random_state=0)
tree6.fit(x6_train, y6_train)
print('Train set score 2 : {:.3f}'.format(tree6.score(x6_train, y6_train)))
print('Test set score 2 : {:.3f}'.format(tree6.score(x6_test, y6_test)))
y_pred = tree6.predict(x6_test)
print(classification_report(y_pred, y6_test))
```

```
Train set score 2 : 0.971
Test set score 2 : 0.867
              precision    recall  f1-score   support

           0       0.93      0.92      0.93      1120
           1       0.38      0.42      0.39       130

    accuracy                           0.87      1250
   macro avg       0.65      0.67      0.66      1250
weighted avg       0.87      0.87      0.87      1250
```

B-3. Preparation for Decison Trees

In [62]:
```
df = pd.read_csv('lixcl68.csv')
```

In [63]:
```
df = df.drop(['age','job','marital','education', 'balance','contact','day','cam
paign','pdays','previous'],1)
```

In [64]:
```
df.head()
```

Out[64]:

	default	housing	loan	duration	poutcome	y
0	no	no	yes	249	unknown	no
1	no	yes	no	58	unknown	no
2	no	yes	no	504	unknown	yes
3	no	yes	no	179	other	no
4	no	yes	no	511	failure	yes

In [65]:
```
#creating labelEncoder
lb_make = LabelEncoder()
# Converting string labels into numbers
lb_make = LabelEncoder()
df["default"] = lb_make.fit_transform(df["default"])
df["housing"] = lb_make.fit_transform(df["housing"])
df["loan"] = lb_make.fit_transform(df["loan"])
df["poutcome"] = lb_make.fit_transform(df["poutcome"])

df['y'] = lb_make.fit_transform(df['y'])

label = df['y']

features = list(zip(df["default"],df["housing"],df["loan"],
                    df['duration'], df["poutcome"]))

x_train, x_test, y_train, y_test = train_test_split(features,label,
                                                    test_size=0.25, random_stat
e=0,stratify=label)
```

In [66]:
```
df.head()
```

Out[66]:

	default	housing	loan	duration	poutcome	y
0	0	0	1	249	3	0
1	0	1	0	58	3	0
2	0	1	0	504	3	1
3	0	1	0	179	1	0
4	0	1	0	511	0	1

In [16]:
```
feature_names = df.columns.tolist()
feature_names = feature_names[0:5]
target_name = np.array(['Y No', 'Y Yes'])
```

B-4. Decison Trees

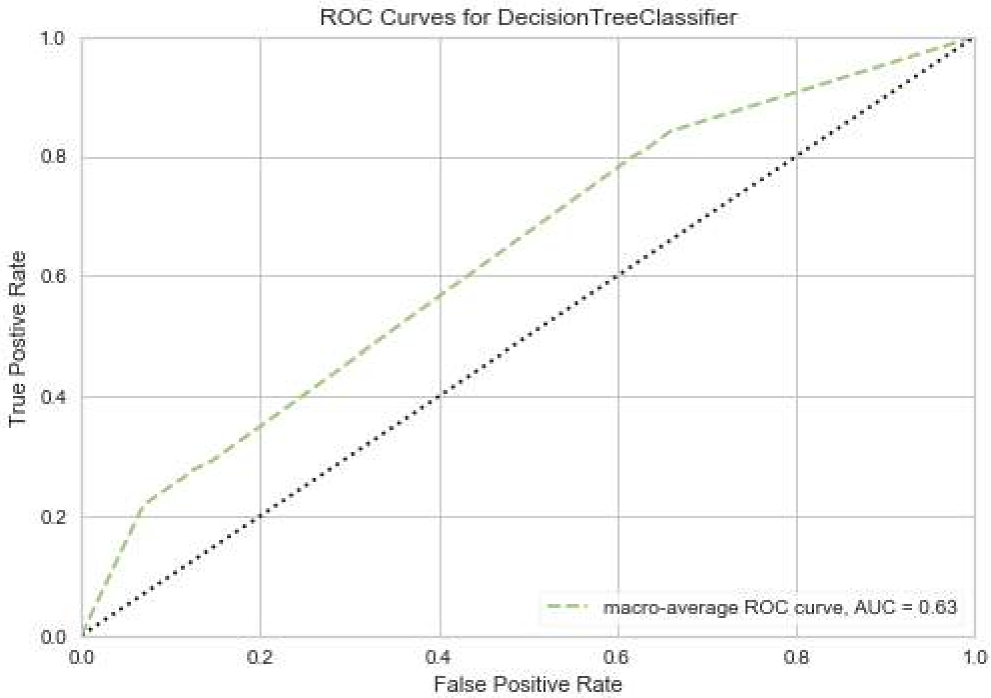
(variable: default, housing, loan, duration and poutcome)

In [89]:

```python
tree = DecisionTreeClassifier(criterion='entropy', random_state=0)
tree.fit(x_train, y_train)

print('Train set score 2 : {:.3f}'.format(tree.score(x_train, y_train)))
print('Test set score 2 : {:.3f}'.format(tree.score(x_test, y_test)))

y_pred = tree.predict(x_test)
print('=========================================================')
print('Confusion Matrix')
confusion = pd.DataFrame(metrics.confusion_matrix(y_test, y_pred, labels=[1,0
]),
                        index=['y_true Yes','y_ture No'],
                        columns=['y_predict Yes','y_predict No'])
print(confusion)
print('=========================================================')
print(classification_report(y_test, y_pred))
```

```
Train set score 2 : 0.969
Test set score 2 : 0.862
=========================================================
Confusion Matrix
            y_predict Yes  y_predict No
y_true Yes             49            95
y_ture No              77          1029
=========================================================
              precision    recall  f1-score   support

           0       0.92      0.93      0.92      1106
           1       0.39      0.34      0.36       144

    accuracy                           0.86      1250
   macro avg       0.65      0.64      0.64      1250
weighted avg       0.85      0.86      0.86      1250
```

```
In [68]: visualizer = ROCAUC(tree, classes=[0, 1], micro=False, macro=True, per_class=Fa
lse)
visualizer.fit(x_train, y_train)
visualizer.score(x_test, y_test)
visualizer.show()
print('roc_auc_score:', roc_auc_score(y_test, y_pred))
```




```
roc_auc_score: 0.6353287623066104
```

B-5. Decison Trees Optimization

Tuning parameters using RandomizedSearchCV

```
In [17]: max_features = ['auto', 'sqrt']
max_depth = [int(x) for x in np.linspace(10, 110, num = 22)]
min_samples_leaf = [int(x) for x in np.linspace(1, 10, num = 10)]
min_samples_split = [int(x) for x in np.linspace(2, 10, num = 9)]
criterion = ['gini', 'entropy']

param_dist = {'max_depth': max_depth,
              'max_features': max_features,
              'min_samples_leaf': min_samples_leaf,
              'min_samples_split': min_samples_split,
              'criterion': criterion}

print(param_dist)
```

```
{'max_depth': [10, 14, 19, 24, 29, 33, 38, 43, 48, 52, 57, 62, 67, 71, 76, 81,
86, 90, 95, 100, 105, 110], 'max_features': ['auto', 'sqrt'], 'min_samples_lea
f': [1, 2, 3, 4, 5, 6, 7, 8, 9, 10], 'min_samples_split': [2, 3, 4, 5, 6, 7, 8,
9, 10], 'criterion': ['gini', 'entropy']}
```

In [18]:
```
tree_random = DecisionTreeClassifier(random_state=0)
tree_cv = RandomizedSearchCV(estimator = tree_random, param_distributions = par
am_dist,
                              cv = 5, random_state=0)
tree_cv.fit(x_train, y_train)
```

Out[18]:
```
RandomizedSearchCV(cv=5, error_score='raise-deprecating',
                   estimator=DecisionTreeClassifier(class_weight=None,
                                                    criterion='gini',
                                                    max_depth=None,
                                                    max_features=None,
                                                    max_leaf_nodes=None,
                                                    min_impurity_decrease=0.0,
                                                    min_impurity_split=None,
                                                    min_samples_leaf=1,
                                                    min_samples_split=2,
                                                    min_weight_fraction_leaf=0.
0,
                                                    presort=False,
                                                    random_state=0,
                                                    splitter='best'),
                   iid='warn...e,
                   param_distributions={'criterion': ['gini', 'entropy'],
                                        'max_depth': [10, 14, 19, 24, 29, 33,
                                                      38, 43, 48, 52, 57, 62,
                                                      67, 71, 76, 81, 86, 90,
                                                      95, 100, 105, 110],
                                        'max_features': ['auto', 'sqrt'],
                                        'min_samples_leaf': [1, 2, 3, 4, 5, 6,
                                                             7, 8, 9, 10],
                                        'min_samples_split': [2, 3, 4, 5, 6, 7,
                                                              8, 9, 10]},
                   pre_dispatch='2*n_jobs', random_state=0, refit=True,
                   return_train_score=False, scoring=None, verbose=0)
```

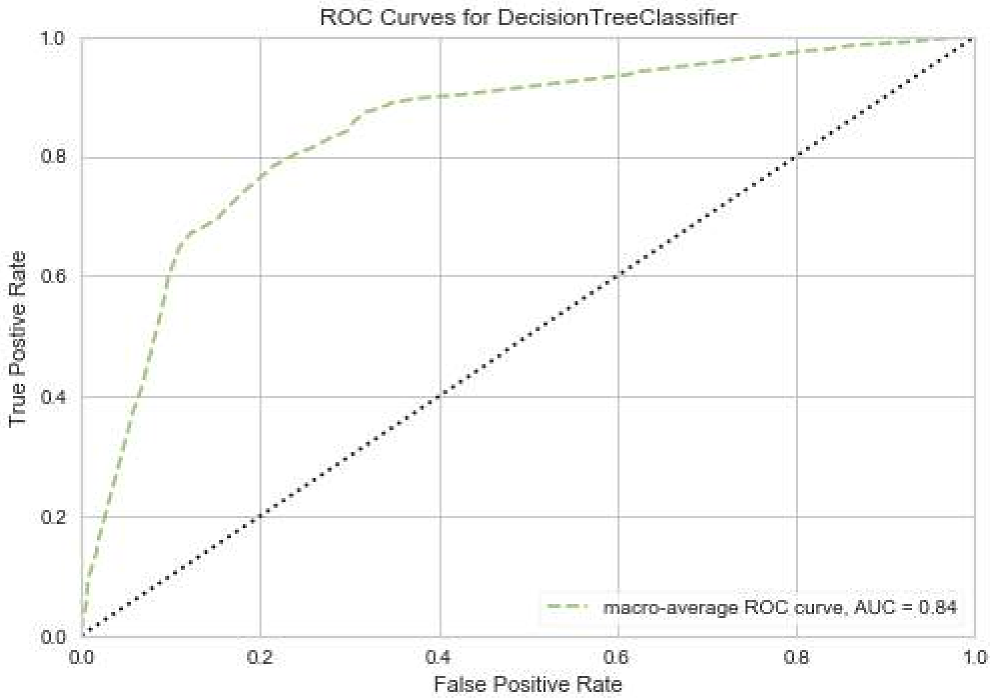
In [19]:
```
tree_cv.best_params_
```

Out[19]:
```
{'min_samples_split': 8,
 'min_samples_leaf': 10,
 'max_features': 'auto',
 'max_depth': 76,
 'criterion': 'gini'}
```

In [20]:
```
tree2 = DecisionTreeClassifier(criterion='gini', max_depth = 76,
                               max_features='auto', min_samples_leaf=10,
                               min_samples_split = 8, random_state=0)
tree2.fit(x_train, y_train)

print('Train set score 2 : {:.3f}'.format(tree2.score(x_train, y_train)))
print('Test set score 2 : {:.3f}'.format(tree2.score(x_test, y_test)))

y_pred = tree2.predict(x_test)
print('==========================================================')
print('Confusion Matrix')
confusion = pd.DataFrame(metrics.confusion_matrix(y_test, y_pred, labels=[1,0
]),
                        index=['y_true Yes','y_ture No'],
                        columns=['y_predict Yes','y_predict No'])
print(confusion)
print('==========================================================')
print(classification_report(y_test, y_pred))
```

```
Train set score 2 : 0.902
Test set score 2 : 0.889
=========================================================
Confusion Matrix
            y_predict Yes  y_predict No
y_true Yes             24           120
y_ture No              19          1087
=========================================================
              precision    recall  f1-score   support

           0       0.90      0.98      0.94      1106
           1       0.56      0.17      0.26       144

    accuracy                           0.89      1250
   macro avg       0.73      0.57      0.60      1250
weighted avg       0.86      0.89      0.86      1250
```

In [21]:
```
visualizer = ROCAUC(tree2, classes=[0, 1], micro=False, macro=True, per_class=F
alse)
visualizer.fit(x_train, y_train)
visualizer.score(x_test, y_test)
visualizer.show()
print('roc_auc_score:', roc_auc_score(y_test, y_pred))
```




```
roc_auc_score: 0.5747438215792646
```

B-6. Decison Tree Optimization 2

(Cannot make a garph from above code, too big to make,

downsize each parameters that available to visualize)

In [22]:
```
max_features = ['auto', 'sqrt']
max_depth = [int(x) for x in np.linspace(1, 5, num = 5)]
min_samples_leaf = [int(x) for x in np.linspace(1, 10, num = 10)]
min_samples_split = [int(x) for x in np.linspace(2, 10, num = 9)]
criterion = ['gini', 'entropy']

param_dist2 = {'max_depth': max_depth,
               'max_features': max_features,
               'min_samples_leaf': min_samples_leaf,
               'min_samples_split': min_samples_split,
               'criterion': criterion}

print(param_dist2)
```

```
{'max_depth': [1, 2, 3, 4, 5], 'max_features': ['auto', 'sqrt'], 'min_samples_l
eaf': [1, 2, 3, 4, 5, 6, 7, 8, 9, 10], 'min_samples_split': [2, 3, 4, 5, 6, 7,
8, 9, 10], 'criterion': ['gini', 'entropy']}
```

In [23]:
```
tree_random2 = DecisionTreeClassifier(random_state=0)
tree_cv2 = RandomizedSearchCV(estimator = tree_random, param_distributions = pa
ram_dist2,
                              cv = 5, random_state=0)
tree_cv2.fit(x_train, y_train)
```

Out[23]:
```
RandomizedSearchCV(cv=5, error_score='raise-deprecating',
                   estimator=DecisionTreeClassifier(class_weight=None,
                                                    criterion='gini',
                                                    max_depth=None,
                                                    max_features=None,
                                                    max_leaf_nodes=None,
                                                    min_impurity_decrease=0.0,
                                                    min_impurity_split=None,
                                                    min_samples_leaf=1,
                                                    min_samples_split=2,
                                                    min_weight_fraction_leaf=0.
0,
                                                    presort=False,
                                                    random_state=0,
                                                    splitter='best'),
                   iid='warn', n_iter=10, n_jobs=None,
                   param_distributions={'criterion': ['gini', 'entropy'],
                                        'max_depth': [1, 2, 3, 4, 5],
                                        'max_features': ['auto', 'sqrt'],
                                        'min_samples_leaf': [1, 2, 3, 4, 5, 6,
                                                             7, 8, 9, 10],
                                        'min_samples_split': [2, 3, 4, 5, 6, 7,
                                                              8, 9, 10]},
                   pre_dispatch='2*n_jobs', random_state=0, refit=True,
                   return_train_score=False, scoring=None, verbose=0)
```

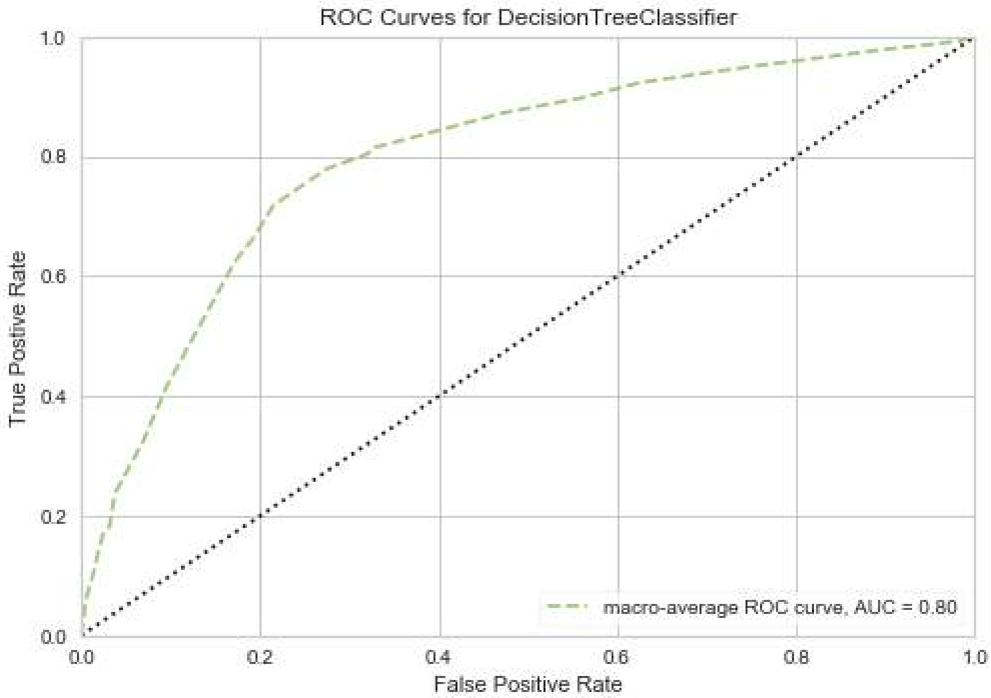
In [24]:
```
tree_cv2.best_params_
```

Out[24]:
```
{'min_samples_split': 2,
 'min_samples_leaf': 7,
 'max_features': 'sqrt',
 'max_depth': 4,
 'criterion': 'gini'}
```

In [25]:
```python
tree3 = DecisionTreeClassifier(criterion='gini', max_depth = 4,
                                max_features='sqrt', min_samples_leaf=7,
                                min_samples_split = 2, random_state=0)
tree3.fit(x_train, y_train)

print('Train set score 2 : {:.3f}'.format(tree3.score(x_train, y_train)))
print('Test set score 2 : {:.3f}'.format(tree3.score(x_test, y_test)))

y_pred = tree3.predict(x_test)
print('=========================================================')
print('Confusion Matrix')
confusion = pd.DataFrame(metrics.confusion_matrix(y_test, y_pred, labels=[1,0
]),
                        index=['y_true Yes','y_ture No'],
                        columns=['y_predict Yes','y_predict No'])
print(confusion)
print('=========================================================')
print(classification_report(y_test, y_pred))
```

```
Train set score 2 : 0.901
Test set score 2 : 0.896
=========================================================
Confusion Matrix
            y_predict Yes  y_predict No
y_true Yes             41           103
y_ture No              27          1079
=========================================================
              precision    recall  f1-score   support

           0       0.91      0.98      0.94      1106
           1       0.60      0.28      0.39       144

    accuracy                           0.90      1250
   macro avg       0.76      0.63      0.66      1250
weighted avg       0.88      0.90      0.88      1250
```

```
In [26]: visualizer = ROCAUC(tree3, classes=[0, 1], micro=False, macro=True, per_class=F
alse)
visualizer.fit(x_train, y_train)
visualizer.score(x_test, y_test)
visualizer.show()
print('roc_auc_score:', roc_auc_score(y_test, y_pred))
```

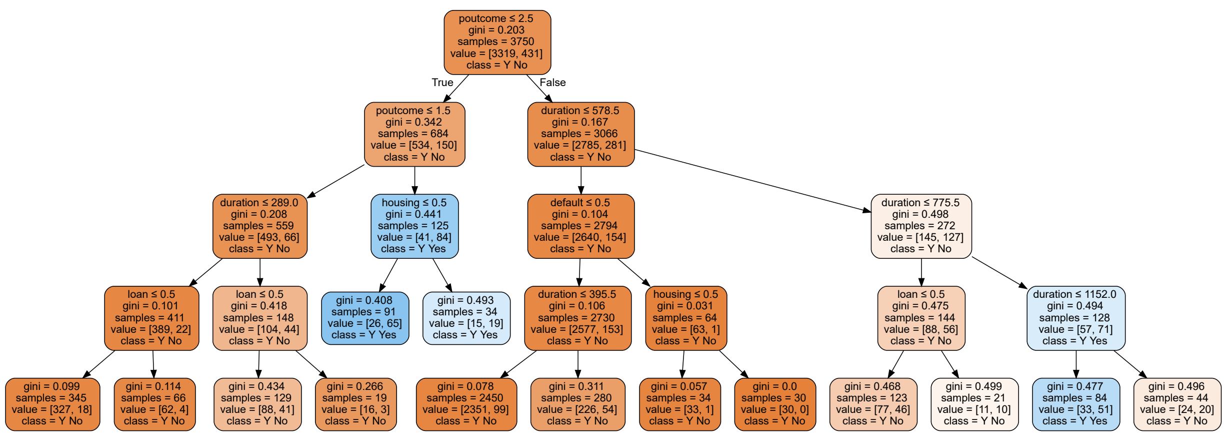



```
roc_auc_score: 0.6301549628290135
```

B-7. Decison Tree Visualization

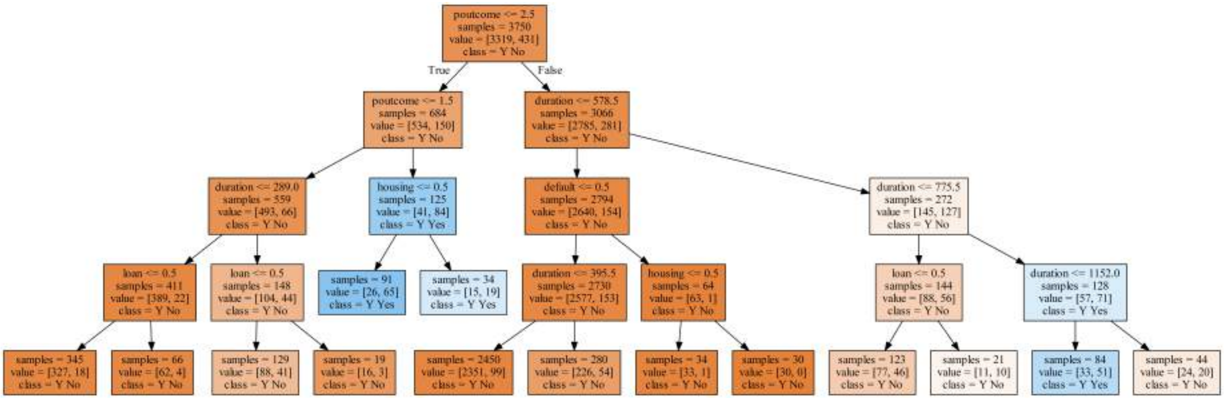
```
In [27]: # import tree one more time
from sklearn import tree
dot_data = tree.export_graphviz(tree3, out_file=None,
                                feature_names=feature_names,
                                class_names=target_name,
                                filled=True, rounded=True,
                                special_characters=True)
graph = graphviz.Source(dot_data)
graph
```

Out[27]:



In [81]:
```python
tree.export_graphviz(tree3, out_file = 'dtc.dot', class_names=target_name,
                     feature_names = feature_names, impurity=False, filled=True)
call(['dot', '-Tpng', 'dtc.dot', '-o', 'Decision_Tree.png', '-Gdpi=900'])
Image(filename = 'Decision_Tree.png')
```

Out[81]:



Section C: Model Evaluation

C-1. K-Nearest Neighbours

C1 - 1. Preparation for KNN

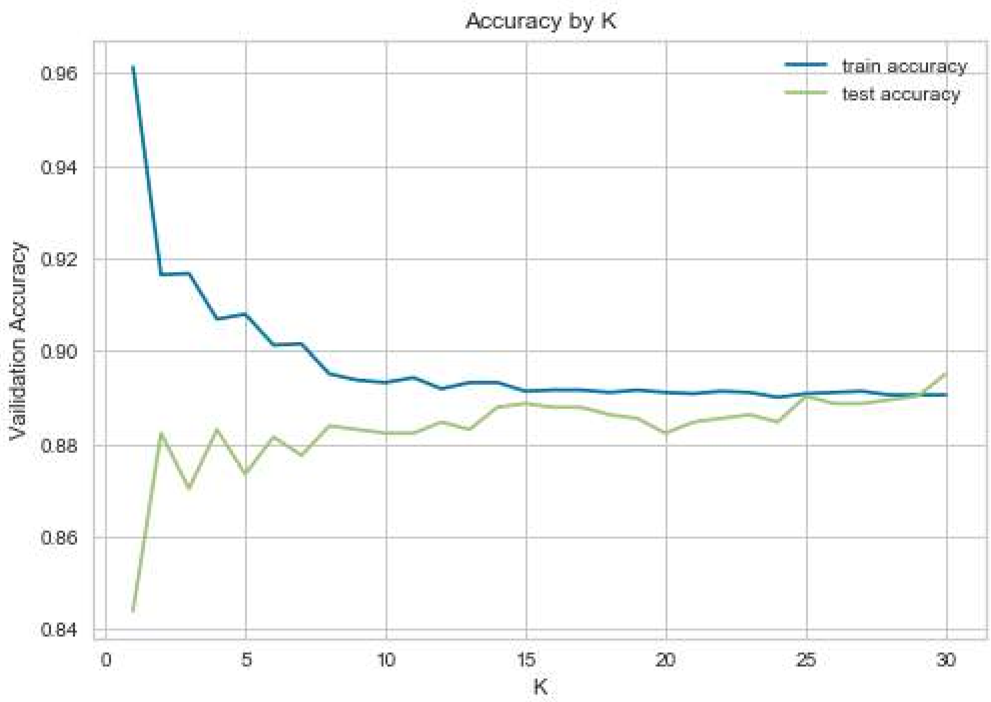
C1 - 2. KNN

In [91]:
```
knn = KNeighborsClassifier(algorithm='auto', n_jobs=-1, n_neighbors=1,
                           weights='uniform')
knn.fit(x_train, y_train)

print("train set accuracy: {:.3f}".format(knn.score(x_train, y_train)))
print("test set accuracy: {:.3f}".format(knn.score(x_test, y_test)))

y_pred = knn.predict(x_test)
print('=========================================================')
print('Confusion Matrix')
confusion = pd.DataFrame(metrics.confusion_matrix(y_test, y_pred, labels=[1,0
]),
                        index=['y_true Yes','y_ture No'],
                        columns=['y_predict Yes','y_predict No'])
print(confusion)
print('=========================================================')
print(classification_report(y_test, y_pred))
```

```
train set accuracy: 0.961
test set accuracy: 0.844
=========================================================
Confusion Matrix
            y_predict Yes  y_predict No
y_true Yes             41           103
y_ture No              92          1014
=========================================================
              precision    recall  f1-score   support

           0       0.91      0.92      0.91      1106
           1       0.31      0.28      0.30       144

    accuracy                           0.84      1250
   macro avg       0.61      0.60      0.60      1250
weighted avg       0.84      0.84      0.84      1250
```

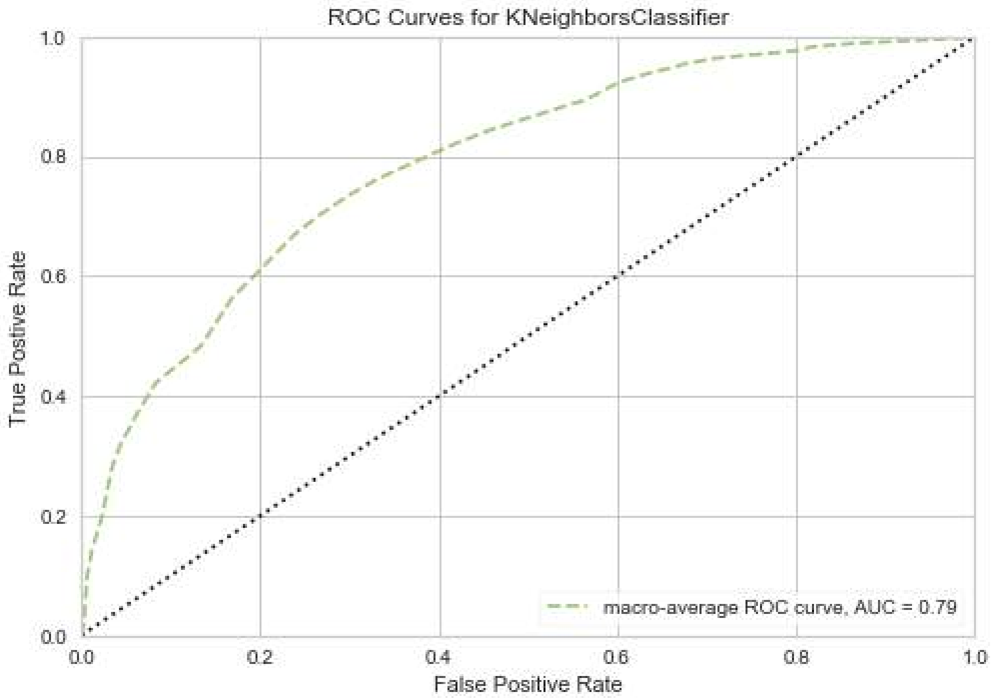
C1 - 3. Finding the optimal K in KNN

In [92]:
```
k_list = range(1, 31)
train_accuracies = []
test_accuracies = []

for k in k_list:
    knn = KNeighborsClassifier(algorithm='auto', leaf_size=30,
                n_jobs=-1, n_neighbors=k, weights='uniform')
    knn.fit(x_train, y_train)
    train_accuracies.append(knn.score(x_train, y_train))
    test_accuracies.append(knn.score(x_test, y_test))

plt.plot(k_list, train_accuracies, label='train accuracy')
plt.plot(k_list, test_accuracies, label='test accuracy')
plt.legend()
plt.xlabel('K')
plt.ylabel('Vailidation Accuracy')
plt.title('Accuracy by K')
plt.show()
```



C1 - 4. KNN ROC curve

In [93]:
```
visualizer = ROCAUC(knn, classes=[0, 1], micro=False, macro=True, per_class=False)
visualizer.fit(x_train, y_train)
visualizer.score(x_test, y_test)
visualizer.show()
print('roc_auc_score:', roc_auc_score(y_test, y_pred))
```




```
roc_auc_score: 0.6007697910387785
```

C1 - 5. KNN Optimization

Tuning parameters using GridSearchCV

In [28]:
```
n_neighbors = [int(x) for x in np.linspace(1,5, num=5)]
weights = ['uniform', 'distance']
leaf_size = [int(x) for x in np.linspace(1,10, num=10)]
algorithm = ['auto', 'kd_tree']

param_grid = { 'n_neighbors': n_neighbors,
               'weights' : weights,
               'leaf_size' : leaf_size,
               'algorithm' : algorithm}
```

In [29]:
```
knn_random = KNeighborsClassifier()
knn_cv = GridSearchCV(knn_random, param_grid,verbose=1, cv=3, n_jobs=-1)
knn_cv.fit(x_train, y_train)
```

```
Fitting 3 folds for each of 200 candidates, totalling 600 fits

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
[Parallel(n_jobs=-1)]: Done  34 tasks      | elapsed:   12.1s
[Parallel(n_jobs=-1)]: Done 184 tasks      | elapsed:   28.8s
[Parallel(n_jobs=-1)]: Done 434 tasks      | elapsed:   51.2s
[Parallel(n_jobs=-1)]: Done 600 out of 600 | elapsed:  1.1min finished
```

Out[29]:
```
GridSearchCV(cv=3, error_score='raise-deprecating',
             estimator=KNeighborsClassifier(algorithm='auto', leaf_size=30,
                                            metric='minkowski',
                                            metric_params=None, n_jobs=None,
                                            n_neighbors=5, p=2,
                                            weights='uniform'),
             iid='warn', n_jobs=-1,
             param_grid={'algorithm': ['auto', 'kd_tree'],
                         'leaf_size': [1, 2, 3, 4, 5, 6, 7, 8, 9, 10],
                         'n_neighbors': [1, 2, 3, 4, 5],
                         'weights': ['uniform', 'distance']},
             pre_dispatch='2*n_jobs', refit=True, return_train_score=False,
             scoring=None, verbose=1)
```

In [30]:
```
knn_cv.best_params_
```

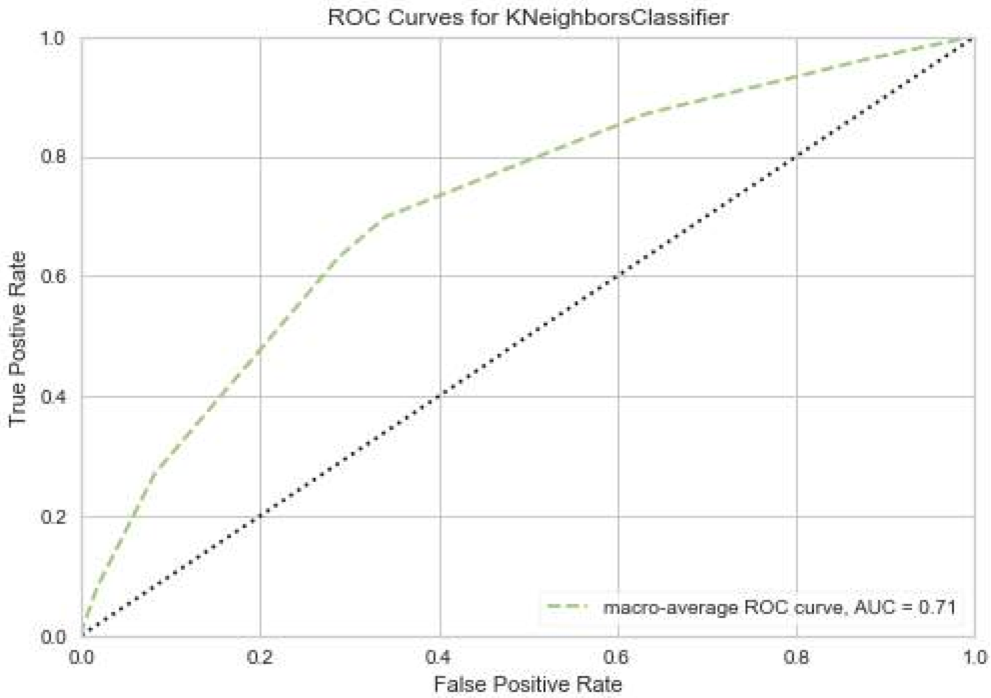
Out[30]:
```
{'algorithm': 'auto', 'leaf_size': 1, 'n_neighbors': 4, 'weights': 'uniform'}
```

In [73]:
```
knn2 = KNeighborsClassifier(algorithm = 'auto', leaf_size = 1,
                            n_jobs=-1, n_neighbors=4, weights='uniform')

knn2.fit(x_train, y_train)

print("train set accuracy: {:.3f}".format(knn2.score(x_train, y_train)))
print("test set accuracy: {:.3f}".format(knn2.score(x_test, y_test)))

y_pred = knn2.predict(x_test)
print('==========================================================')
print('Confusion Matrix')
confusion = pd.DataFrame(metrics.confusion_matrix(y_test, y_pred, labels=[1,0
]),
                        index=['y_true Yes','y_ture No'],
                        columns=['y_predict Yes','y_predict No'])
print(confusion)
print('==========================================================')
print(classification_report(y_test, y_pred))
```

```
train set accuracy: 0.907
test set accuracy: 0.883
==========================================================
Confusion Matrix
            y_predict Yes  y_predict No
y_true Yes             18           126
y_ture No              20          1086
==========================================================
              precision    recall  f1-score   support

           0       0.90      0.98      0.94      1106
           1       0.47      0.12      0.20       144

    accuracy                           0.88      1250
   macro avg       0.68      0.55      0.57      1250
weighted avg       0.85      0.88      0.85      1250
```

C1 - 6. Opimized KNN ROC curve

In [32]:
```
visualizer = ROCAUC(knn2, classes=[0, 1], micro=False, macro=True, per_class=False)
visualizer.fit(x_train, y_train)
visualizer.score(x_test, y_test)
visualizer.show()
print('roc_auc_score:', roc_auc_score(y_test, y_pred))
```




```
roc_auc_score: 0.5534584086799278
```

C-2. Logistic Regression

C2 - 1. Prepartion for Logistic Regression

C2 - 2. Logistic Regression

In [33]:
```
log = LogisticRegression(random_state=0)
log.fit(x_train, y_train)

x2 = sm.add_constant(features)
model = sm.OLS(label, x2)
result = model.fit()
print(result.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                      y   R-squared:                       0.179
Model:                            OLS   Adj. R-squared:                  0.179
Method:                 Least Squares   F-statistic:                     218.4
Date:                Sat, 11 Jan 2020   Prob (F-statistic):          2.32e-211
Time:                        12:58:59   Log-Likelihood:                -887.74
No. Observations:                5000   AIC:                             1787.
Df Residuals:                    4994   BIC:                             1827.
Df Model:                           5
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const          0.1404      0.013     10.588      0.000       0.114       0.166
x1            -0.0242      0.031     -0.784      0.433      -0.085       0.036
x2            -0.1038      0.008    -12.565      0.000      -0.120      -0.088
x3            -0.0609      0.011     -5.450      0.000      -0.083      -0.039
x4             0.0005   1.55e-05     29.588      0.000       0.000       0.000
x5            -0.0298      0.004     -7.143      0.000      -0.038      -0.022
==============================================================================
Omnibus:                     1687.302   Durbin-Watson:                   2.025
Prob(Omnibus):                  0.000   Jarque-Bera (JB):             5408.071
Skew:                           1.733   Prob(JB):                         0.00
Kurtosis:                       6.735   Cond. No.                     2.80e+03
==============================================================================

Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is correctl
y specified.
[2] The condition number is large, 2.8e+03. This might indicate that there are
strong multicollinearity or other numerical problems.

C:\Users\chanl\Anaconda3\lib\site-packages\sklearn\linear_model\logistic.py:43
2: FutureWarning: Default solver will be changed to 'lbfgs' in 0.22. Specify a
solver to silence this warning.
  FutureWarning)
```

In [81]:
```
print("train set accuracy: {:.3f}".format(log.score(x_train, y_train)))
print("test set accuracy: {:.3f}".format(log.score(x_test, y_test)))

y_pred = log.predict(x_test)
print('=========================================================')
print('Confusion Matrix')
confusion = pd.DataFrame(metrics.confusion_matrix(y_test, y_pred, labels=[1,0
]),
                        index=['y_true Yes','y_ture No'],
                        columns=['y_predict Yes','y_predict No'])
print(confusion)
print('=========================================================')
print(classification_report(y_test, y_pred))
```

```
train set accuracy: 0.888
test set accuracy: 0.897
=========================================================
Confusion Matrix
            y_predict Yes  y_predict No
y_true Yes             34           110
y_ture No              19          1087
=========================================================
              precision    recall  f1-score   support

           0       0.91      0.98      0.94      1106
           1       0.64      0.24      0.35       144

    accuracy                           0.90      1250
   macro avg       0.77      0.61      0.64      1250
weighted avg       0.88      0.90      0.88      1250
```

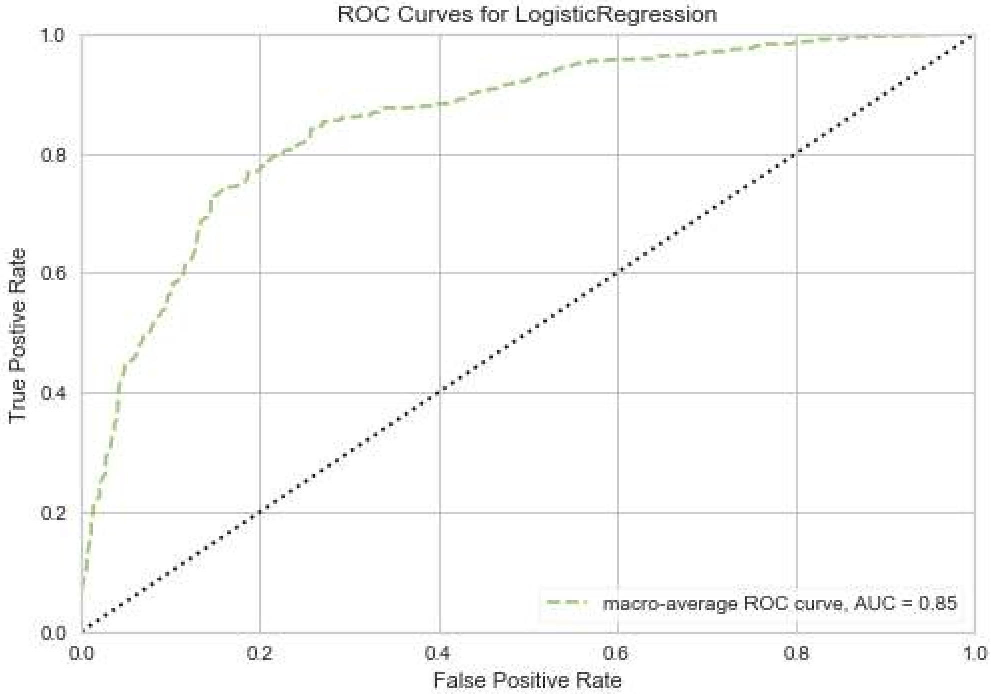
In [80]:
```
# Find odd in each variable
logit = sm.Logit(label,features)
result = logit.fit()
np.exp(result.params)
```

```
Optimization terminated successfully.
         Current function value: 0.301453
         Iterations 7
```

Out[80]:
```
x1    0.722882
x2    0.154148
x3    0.291117
x4    1.003090
x5    0.455043
dtype: float64
```

C2 - 3. Logistic Regression ROC curve

In [18]:
```
visualizer = ROCAUC(log, classes=[0, 1], micro=False, macro=True, per_class=False)
visualizer.fit(x_train, y_train)
visualizer.score(x_test, y_test)
visualizer.show()
print('roc_auc_score:', roc_auc_score(y_test, y_pred))
```




```
roc_auc_score: 0.609466043801487
```

C2 - 4. Logistic Regression Optimization

Tunning parameters using GridSearchCV

In [74]:
```
c_space = np.logspace(-10, 30, 20)


param_grid = {'C': c_space}
```

In [ ]:
```
log_random = LogisticRegression(random_state=0)
logreg_cv = GridSearchCV(log_random, param_grid, cv = 5)
logreg_cv.fit(x_train, y_train)
```

In [ ]:
```
'''
GridSearchCV(cv=5, error_score='raise-deprecating',
             estimator=LogisticRegression(C=1.0, class_weight=None, dual=False,
                                          fit_intercept=True,
                                          intercept_scaling=1, l1_ratio=None,
                                          max_iter=100, multi_class='warn',
                                          n_jobs=None, penalty='l2',
                                          random_state=0, solver='warn',
                                          tol=0.0001, verbose=0,
                                          warm_start=False),
             iid='warn', n_jobs=None,
             param_grid={'C': array([1.00000000e-...7674e-06, 2.06913808e-04,
       2.63665090e-02, 3.35981829e+00, 4.28133240e+02, 5.45559478e+04,
       6.95192796e+06, 8.85866790e+08, 1.12883789e+11, 1.43844989e+13,
       1.83298071e+15, 2.33572147e+17, 2.97635144e+19, 3.79269019e+21,
       4.83293024e+23, 6.15848211e+25, 7.84759970e+27, 1.00000000e+30])},
             pre_dispatch='2*n_jobs', refit=True, return_train_score=False,
             scoring=None, verbose=0)
'''
```

In [76]:
```
logreg_cv.best_params_
```

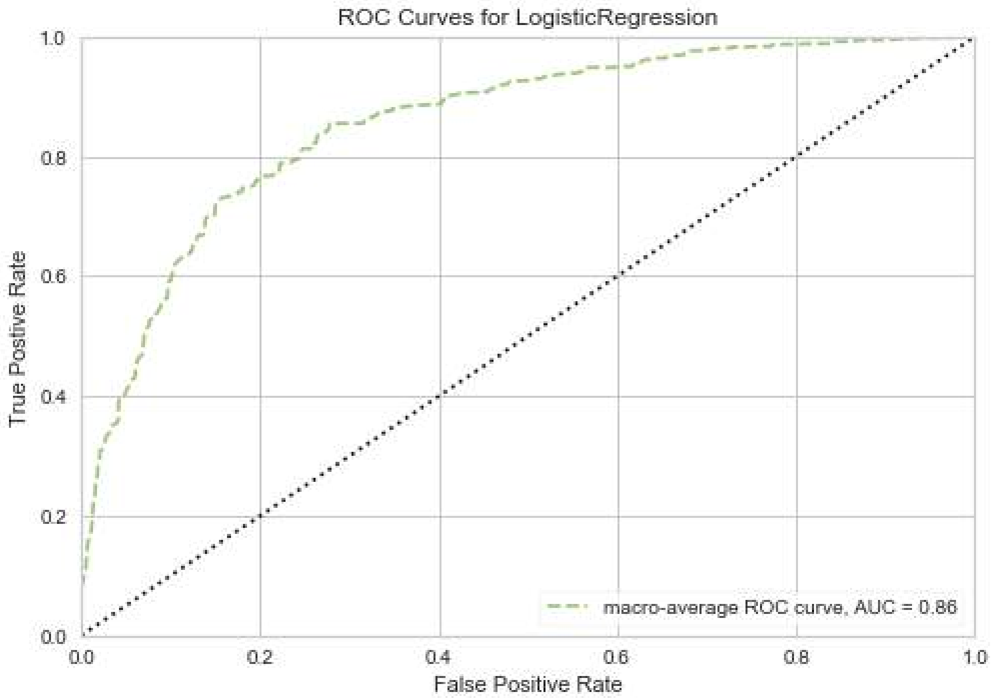
Out[76]:
```
{'C': 0.026366508987303555}
```

In [77]:
```
log2 = LogisticRegression(random_state=0, C=  0.026366508987303555 )
log2.fit(x_train, y_train)

print("train set accuracy: {:.3f}".format(log2.score(x_train, y_train)))
print("test set accuracy: {:.3f}".format(log2.score(x_test, y_test)))

y_pred = log2.predict(x_test)
print('=========================================================')
print('Confusion Matrix')
confusion = pd.DataFrame(metrics.confusion_matrix(y_test, y_pred, labels=[1,0
]),
                        index=['y_true Yes','y_ture No'],
                        columns=['y_predict Yes','y_predict No'])
print(confusion)
print('=========================================================')
print(classification_report(y_test, y_pred))
```

```
train set accuracy: 0.889
test set accuracy: 0.892
=========================================================
Confusion Matrix
            y_predict Yes  y_predict No
y_true Yes             27           117
y_ture No              18          1088
=========================================================
              precision    recall  f1-score   support

           0       0.90      0.98      0.94      1106
           1       0.60      0.19      0.29       144

    accuracy                           0.89      1250
   macro avg       0.75      0.59      0.61      1250
weighted avg       0.87      0.89      0.87      1250


C:\Users\chanl\Anaconda3\lib\site-packages\sklearn\linear_model\logistic.py:43
2: FutureWarning: Default solver will be changed to 'lbfgs' in 0.22. Specify a
solver to silence this warning.
  FutureWarning)
```

C2 - 5. Optimized Logistic Regression ROC curve

In [82]:
```
visualizer = ROCAUC(log2, classes=[0, 1], micro=False, macro=True, per_class=Fa
lse)
visualizer.fit(x_train, y_train)
visualizer.score(x_test, y_test)
visualizer.show()
print('roc_auc_score:', roc_auc_score(y_test, y_pred))
```




```
roc_auc_score: 0.609466043801487
```

C3. Random Forests

C3 - 1. Preparation for Random Forests

C3 - 2. Random Forests

In [83]:
```
forest = RandomForestClassifier(n_estimators=100, max_features=5,max_depth=20,
                               bootstrap=True, oob_score=True,n_jobs=-1, random_state=0
)
forest.fit(x_train, y_train)

print('train set accuracy: {:.3f}'.format(forest.score(x_train, y_train)))
print('test set accuracy: {:.3f}'.format(forest.score(x_test, y_test)))

y_pred = forest.predict(x_test)
print('Out-of-bag score estimate: {:.3f}'.format(forest.oob_score_))
print('==========================================================')
print('Confusion Matrix')
confusion = pd.DataFrame(metrics.confusion_matrix(y_test, y_pred, labels=[1,0
]),
                        index=['y_true Yes','y_ture No'],
                        columns=['y_predict Yes','y_predict No'])
print(confusion)
print('==========================================================')
print(classification_report(y_test, y_pred))
```

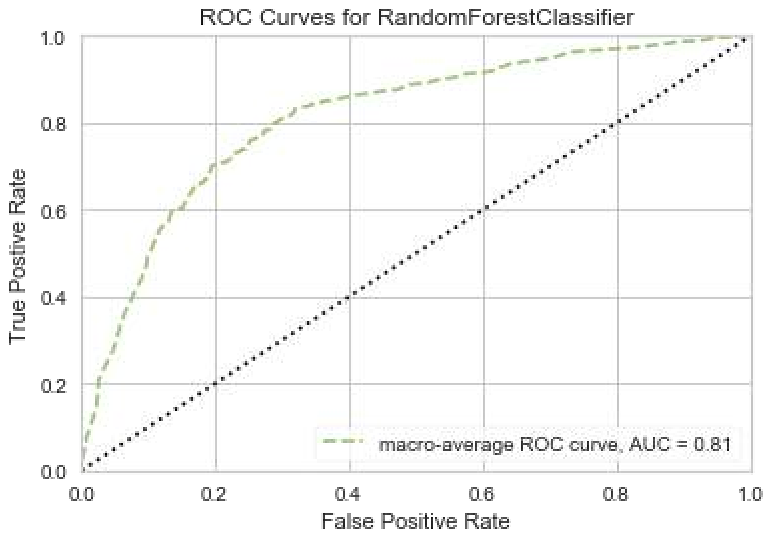
```
train set accuracy: 0.968
test set accuracy: 0.869
Out-of-bag score estimate: 0.868
==========================================================
Confusion Matrix
            y_predict Yes  y_predict No
y_true Yes             54            90
y_ture No              74          1032
==========================================================
              precision    recall  f1-score   support

           0       0.92      0.93      0.93      1106
           1       0.42      0.38      0.40       144

    accuracy                           0.87      1250
   macro avg       0.67      0.65      0.66      1250
weighted avg       0.86      0.87      0.87      1250
```

In [104]:
```
# Checking importatnce of each variable
for name, score in zip(x, forest.feature_importances_):
    print(name, score)
```

```
default 0.004111155029345727
housing 0.060054276878209335
loan 0.03982307569779734
duration 0.7549936186344837
poutcome 0.14101787376016395
```

C3 - 3. Random Forests ROC curve

In [105]:
```
visualizer = ROCAUC(forest, classes=[0, 1], micro=False, macro=True, per_class=False)
visualizer.fit(x_train, y_train)
visualizer.score(x_test, y_test)
visualizer.show()
print('roc_auc_score:', roc_auc_score(y_test, y_pred))
```




```
roc_auc_score: 0.6449856841470766
```

C3 - 4. Random Forests Opitmization

Tuning hyperparameters using RandomizedSearchCV

In [84]:
```
n_estimators = [int(x) for x in np.linspace(start = 150, stop = 250, num = 10)]
max_features = ['auto', 'sqrt']
max_depth = [int(x) for x in np.linspace(20, 40, num = 20)]
max_depth.append(None)
#min_samples_split = [2, 5, 10]
min_samples_split = [int(x) for x in np.linspace(5, 20, num = 10)]
#min_samples_leaf = [1, 2, 4]
min_samples_leaf = [int(x) for x in np.linspace(5, 10, num = 5)]
bootstrap = [True]

random_grid = {'n_estimators': n_estimators,
               'max_features': max_features,
               'max_depth': max_depth,
               'min_samples_split': min_samples_split,
               'min_samples_leaf': min_samples_leaf,
               'bootstrap': bootstrap}
print(random_grid)
```

```
{'n_estimators': [150, 161, 172, 183, 194, 205, 216, 227, 238, 250], 'max_featu
res': ['auto', 'sqrt'], 'max_depth': [20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 3
0, 31, 32, 33, 34, 35, 36, 37, 38, 40, None], 'min_samples_split': [5, 6, 8, 1
0, 11, 13, 15, 16, 18, 20], 'min_samples_leaf': [5, 6, 7, 8, 10], 'bootstrap':
[True]}
```

In [85]:
```
forest_random = RandomForestClassifier(random_state=0)
forest_cv = RandomizedSearchCV(estimator = forest, param_distributions = random
_grid,
                                  n_iter = 100, cv = 3, verbose=2, random_state
=0, n_jobs = -1)
forest_cv.fit(x_train, y_train)
```

```
Fitting 3 folds for each of 100 candidates, totalling 300 fits

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
[Parallel(n_jobs=-1)]: Done  25 tasks      | elapsed:   17.5s
[Parallel(n_jobs=-1)]: Done 146 tasks      | elapsed:   43.6s
[Parallel(n_jobs=-1)]: Done 300 out of 300 | elapsed:  1.3min finished
```

Out[85]:
```
RandomizedSearchCV(cv=3, error_score='raise-deprecating',
                   estimator=RandomForestClassifier(bootstrap=True,
                                                    class_weight=None,
                                                    criterion='gini',
                                                    max_depth=20,
                                                    max_features=5,
                                                    max_leaf_nodes=None,
                                                    min_impurity_decrease=0.0,
                                                    min_impurity_split=None,
                                                    min_samples_leaf=1,
                                                    min_samples_split=2,
                                                    min_weight_fraction_leaf=0.
0,
                                                    n_estimators=100, n_jobs=-
1,
                                                    oob_score=True,
                                                    ran...
                   param_distributions={'bootstrap': [True],
                                        'max_depth': [20, 21, 22, 23, 24, 25,
                                                      26, 27, 28, 29, 30, 31,
                                                      32, 33, 34, 35, 36, 37,
                                                      38, 40, None],
                                        'max_features': ['auto', 'sqrt'],
                                        'min_samples_leaf': [5, 6, 7, 8, 10],
                                        'min_samples_split': [5, 6, 8, 10, 11,
                                                              13, 15, 16, 18,
                                                              20],
                                        'n_estimators': [150, 161, 172, 183,
                                                         194, 205, 216, 227,
                                                         238, 250]},
                   pre_dispatch='2*n_jobs', random_state=0, refit=True,
                   return_train_score=False, scoring=None, verbose=2)
```

In [86]:
```
forest_cv.best_params_
```

Out[86]:
```
{'n_estimators': 216,
 'min_samples_split': 18,
 'min_samples_leaf': 6,
 'max_features': 'auto',
 'max_depth': 35,
 'bootstrap': True}
```

In [87]:
```
forest2 = RandomForestClassifier(n_estimators=216, max_features='auto',max_dept
h=35,
                                min_samples_split = 18,min_samples_leaf = 6,
                       bootstrap=True, oob_score=True,n_jobs=-1, random_state=0
)
forest2.fit(x_train, y_train)

print('train set accuracy: {:.3f}'.format(forest2.score(x_train, y_train)))
print('test set accuracy: {:.3f}'.format(forest2.score(x_test, y_test)))

y_pred = forest2.predict(x_test)
print('Out-of-bag score estimate: {:.3f}'.format(forest2.oob_score_))
print('=========================================================')
print('Confusion Matrix')
confusion = pd.DataFrame(metrics.confusion_matrix(y_test, y_pred, labels=[1,0
]),
                        index=['y_true Yes','y_ture No'],
                        columns=['y_predict Yes','y_predict No'])
print(confusion)
print('=========================================================')
print(classification_report(y_test, y_pred))
```

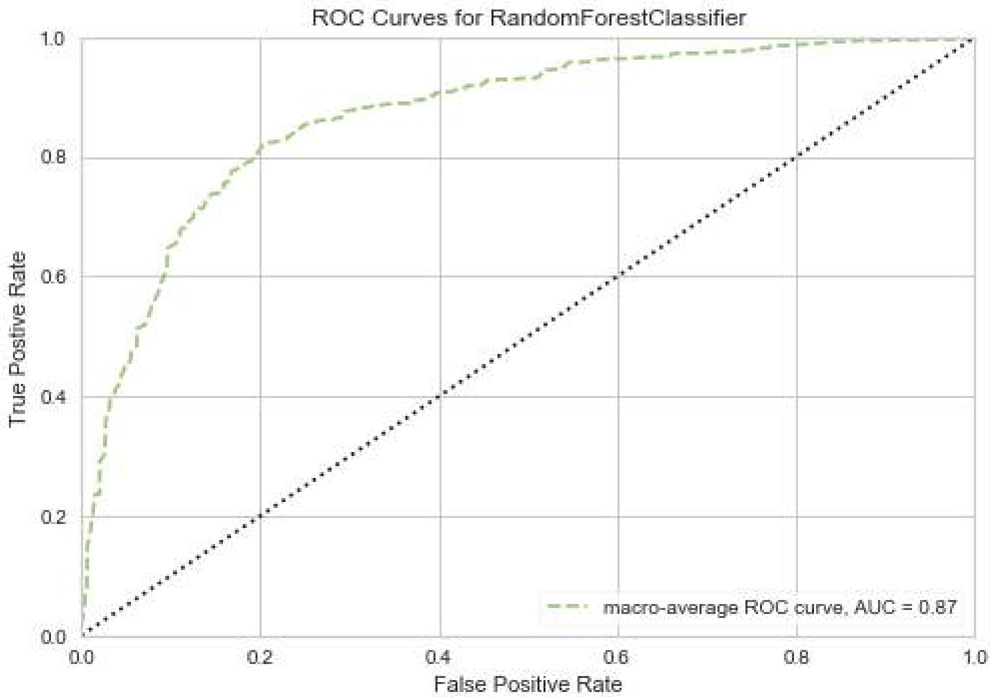
```
train set accuracy: 0.912
test set accuracy: 0.892
Out-of-bag score estimate: 0.897
=========================================================
Confusion Matrix
            y_predict Yes  y_predict No
y_true Yes             41           103
y_ture No              32          1074
=========================================================
              precision    recall  f1-score   support

           0       0.91      0.97      0.94      1106
           1       0.56      0.28      0.38       144

    accuracy                           0.89      1250
   macro avg       0.74      0.63      0.66      1250
weighted avg       0.87      0.89      0.88      1250
```

In [88]:
```
# Checking importatnce of each variable
for name, score in zip(features, forest2.feature_importances_):
    print(name, score)
```

```
(0, 0, 1, 249, 3) 0.0008022195034718320
(0, 1, 0, 58, 3) 0.09274743032615954
(0, 1, 0, 504, 3) 0.023076326910315935
(0, 1, 0, 179, 1) 0.6719789350599784
(0, 1, 0, 511, 0) 0.21139508820007424
```

C3 - 5. Opimized Random Forests ROC curve

In [89]:
```
visualizer = ROCAUC(forest2, classes=[0, 1], micro=False, macro=True, per_class
=False)
visualizer.fit(x_train, y_train)
visualizer.score(x_test, y_test)
visualizer.show()
print('roc_auc_score:', roc_auc_score(y_test, y_pred))
```




```
roc_auc_score: 0.6278945649989954
```

C3 - 6. Comparing Decision Trees and Random Forests

Using Folds and Cross validation

In [91]:
```
folds = KFold(n_splits=5, shuffle=True, random_state=0)
dc = dummy.DummyClassifier()

dt = DecisionTreeClassifier(criterion='gini', max_depth = 4,
                                max_features='sqrt', min_samples_leaf=7,
                                min_samples_split = 2, random_state=0)
rf = RandomForestClassifier(n_estimators=216, max_features='auto',max_depth=35,
                                min_samples_split = 18,min_samples_leaf = 6,
                           bootstrap=True, oob_score=True,n_jobs=-1, random_state=0
)

dt_scores = cross_val_score(dt, features, label, scoring='precision', cv=folds)
rf_scores = cross_val_score(rf, features, label, scoring='precision', cv=folds)

print("Mean DT Accuracy:", np.mean(dt_scores))
print("Mean RF Accuracy:", np.mean(rf_scores))
```

```
Mean DT Accuracy: 0.6025520795779556
Mean RF Accuracy: 0.5784353990457601
```

In [92]:

```
dt_pred = cross_val_predict(dt, features, label, cv=folds)
print(dt_pred)

dt_pred_matrix = confusion_matrix(label, dt_pred)
print(dt_pred_matrix)

rf_pred = cross_val_predict(rf, features, label, cv=folds)
print(rf_pred)

rf_pred_matrix = confusion_matrix(label, rf_pred)
print(rf_pred_matrix)
```

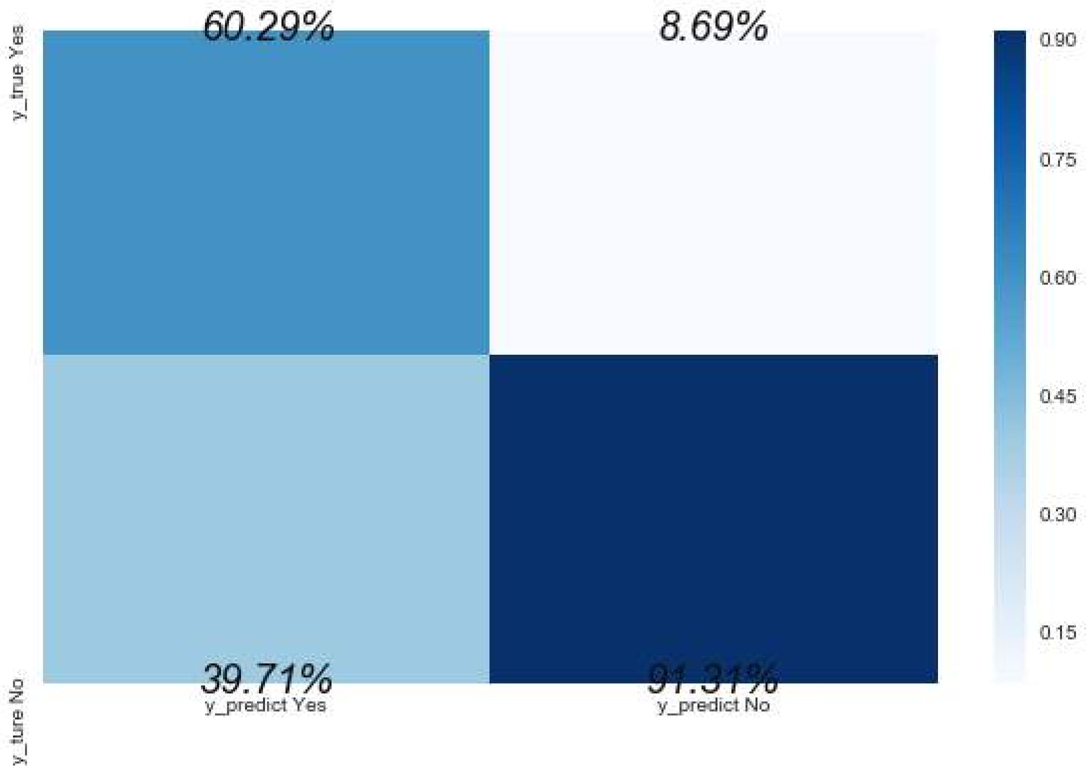
```
[0 0 0 ... 0 0 0]
[[4317  108]
 [ 411  164]]
[0 0 0 ... 0 0 0]
[[4310  115]
 [ 419  156]]
```

In [103]:
```
print('*** Decision Trees ***')
dt_confusion = pd.DataFrame(metrics.confusion_matrix(label, dt_pred, labels=[1,
0]),
                        index=['y_true Yes','y_ture No'],
                        columns=['y_predict Yes','y_predict No'])
print(dt_confusion)
print('=========================================================')
print(classification_report(label, dt_pred))

print('\n')
print('*** Random Forests ***')
rf_confusion = pd.DataFrame(metrics.confusion_matrix(label, rf_pred, labels=[1,
0]),
                        index=['y_true Yes','y_ture No'],
                        columns=['y_predict Yes','y_predict No'])
print(rf_confusion)
print('=========================================================')
print(classification_report(label, rf_pred))
```

```
*** Decision Trees ***
            y_predict Yes  y_predict No
y_true Yes            164           411
y_ture No             108          4317
=========================================================
              precision    recall  f1-score   support

           0       0.91      0.98      0.94      4425
           1       0.60      0.29      0.39       575

    accuracy                           0.90      5000
   macro avg       0.76      0.63      0.67      5000
weighted avg       0.88      0.90      0.88      5000



*** Random Forests ***
            y_predict Yes  y_predict No
y_true Yes            156           419
y_ture No             115          4310
=========================================================
              precision    recall  f1-score   support

           0       0.91      0.97      0.94      4425
           1       0.58      0.27      0.37       575

    accuracy                           0.89      5000
   macro avg       0.74      0.62      0.66      5000
weighted avg       0.87      0.89      0.88      5000
```

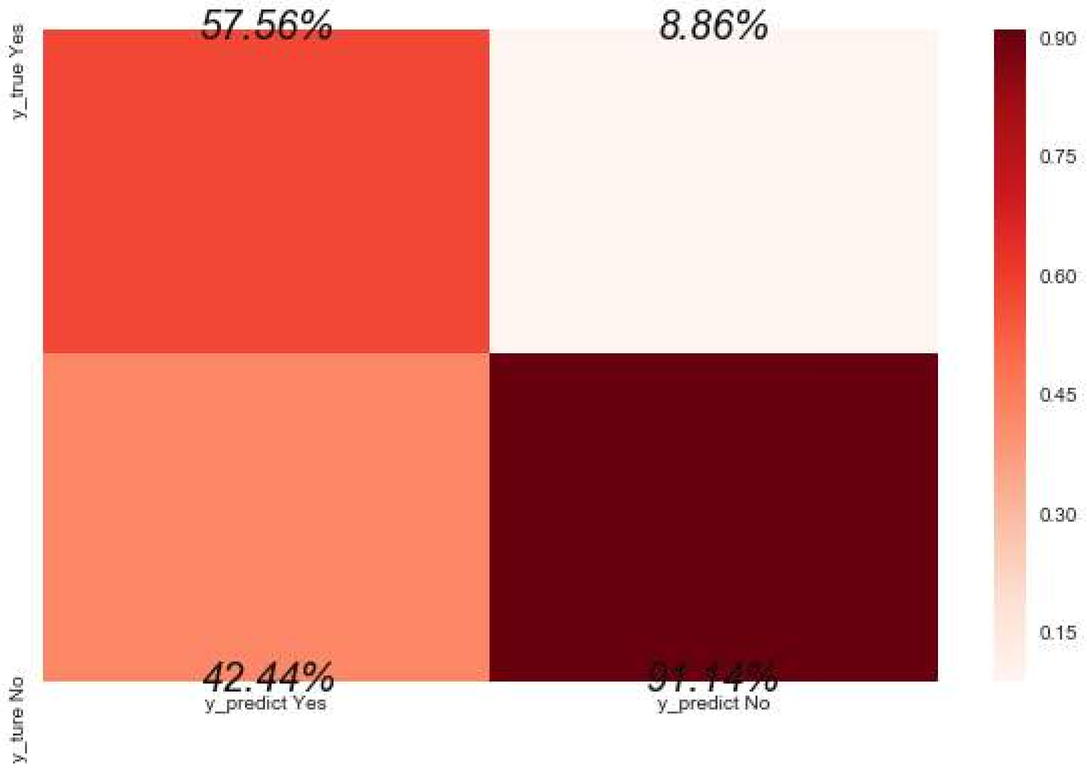
In [113]:
```
plt.figure(figsize=(10,6))
xticklables = ['y_predict Yes','y_predict No']
yticklables = ['y_true Yes','y_ture No']

annot_kws={'fontsize':20,
           'fontstyle':'italic',
           'color':"k",
           'alpha':1,
           'verticalalignment':'center'}

sns.heatmap(dt_confusion/np.sum(dt_confusion), annot=True,
            fmt='.2%', cmap='Blues',
           xticklabels = xticklables,
           yticklabels = yticklables,
           annot_kws = annot_kws)
```

Out[113]: <matplotlib.axes._subplots.AxesSubplot at 0x2457c65f400>



```
In [114]: plt.figure(figsize=(10,6))
xticklables = ['y_predict Yes','y_predict No']
yticklables = ['y_true Yes','y_ture No']

annot_kws={'fontsize':20,
           'fontstyle':'italic',
           'color':"k",
           'alpha':1,
           'verticalalignment':'center'}

sns.heatmap(rf_confusion/np.sum(rf_confusion), annot=True,
            fmt='.2%', cmap='Reds',
           xticklabels = xticklables,
           yticklabels = yticklables,
           annot_kws = annot_kws)
```

```
Out[114]: <matplotlib.axes._subplots.AxesSubplot at 0x245785db710>
```



Section E: Model Implementation

E1. Read file

```
In [1]: df = pd.read_csv('lixcl68.csv')
```

```
In [2]: df = df.drop(['age','job','marital','education', 'balance','contact','day','cam
paign','pdays','previous'],1)
```

In [3]:
```
df.head()
```

Out[3]:

	default	housing	loan	duration	poutcome	y
0	no	no	yes	249	unknown	no
1	no	yes	no	58	unknown	no
2	no	yes	no	504	unknown	yes
3	no	yes	no	179	other	no
4	no	yes	no	511	failure	yes

In [8]:
```
#creating labelEncoder
lb_make = LabelEncoder()

# Converting string labels into numbers
lb_make = LabelEncoder()
df["default"] = lb_make.fit_transform(df["default"])
df["housing"] = lb_make.fit_transform(df["housing"])
df["loan"] = lb_make.fit_transform(df["loan"])
df["poutcome"] = lb_make.fit_transform(df["poutcome"])

df['y'] = lb_make.fit_transform(df['y'])
```

E3. Setting festures and label

In [9]:
```
label = df['y']

features = list(zip(df["default"],df["housing"],df["loan"],
                    df['duration'], df["poutcome"]))
```

E4. Setting up Folds and Cross Validaion

In [10]:
```
folds = KFold(n_splits=5, shuffle=True, random_state=0)
dc = dummy.DummyClassifier()

dt = DecisionTreeClassifier(criterion='gini', max_depth = 4,
                            max_features='sqrt', min_samples_leaf=7,
                            min_samples_split = 2, random_state=0)

dt_scores = cross_val_score(dt, features, label, scoring='precision', cv=folds)

print("Mean DT Accuracy:", np.mean(dt_scores))
```

```
Mean DT Accuracy: 0.6025520795779556
```

E5. Model prediction

In [11]:
```
dt_pred = cross_val_predict(dt, features, label, cv=folds)
print(dt_pred)

dt_pred_matrix = confusion_matrix(label, dt_pred)
print(dt_pred_matrix)
```

```
[0 0 0 ... 0 0 0]
[[4317  108]
 [ 411  164]]
```

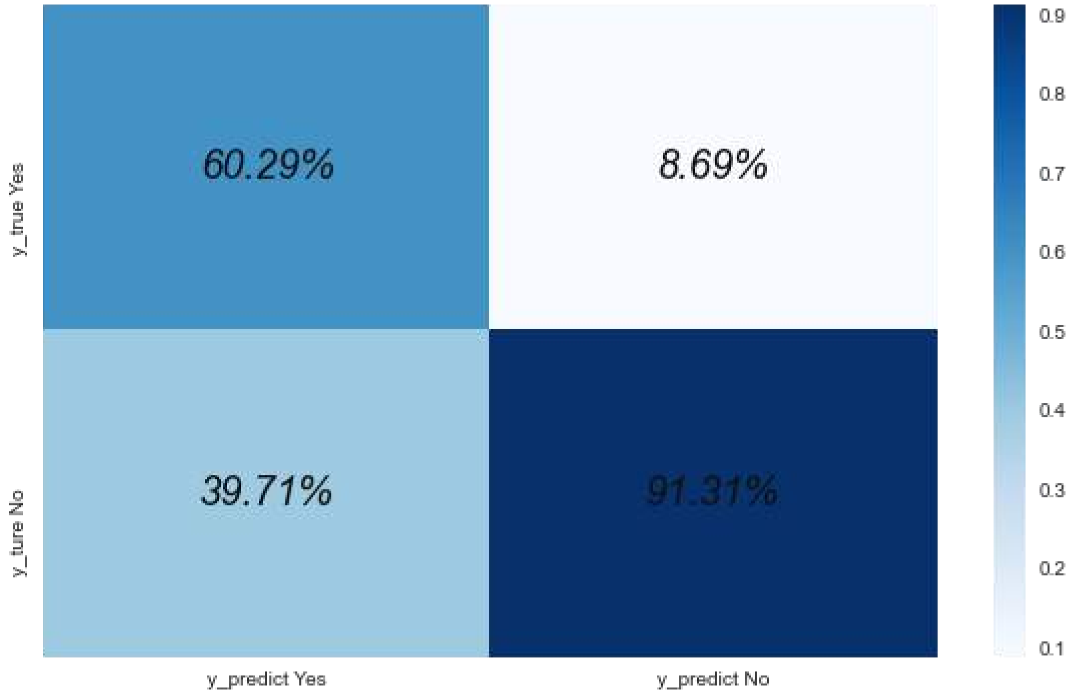
E6. Evaluation using the confusion matrix

In [12]:
```
print('*** Decision Trees ***')
dt_confusion = pd.DataFrame(metrics.confusion_matrix(label, dt_pred, labels=[1, 0]),
                        index=['y_true Yes','y_ture No'],
                        columns=['y_predict Yes','y_predict No'])
print(dt_confusion)
print('==========================================================')
print(classification_report(label, dt_pred))
```

```
*** Decision Trees ***
            y_predict Yes  y_predict No
y_true Yes            164           411
y_ture No             108          4317
==========================================================
              precision    recall  f1-score   support

           0       0.91      0.98      0.94      4425
           1       0.60      0.29      0.39       575

    accuracy                           0.90      5000
   macro avg       0.76      0.63      0.67      5000
weighted avg       0.88      0.90      0.88      5000
```

In [13]:
```
plt.figure(figsize=(10,6))
xticklables = ['y_predict Yes','y_predict No']
yticklables = ['y_true Yes','y_ture No']

annot_kws={'fontsize':20,
           'fontstyle':'italic',
           'color':"k",
           'alpha':1,
           'verticalalignment':'center'}

sns.heatmap(dt_confusion/np.sum(dt_confusion), annot=True,
            fmt='.2%', cmap='Blues',
           xticklabels = xticklables,
           yticklabels = yticklables,
           annot_kws = annot_kws)
```

Out[13]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x27bc6efc448>
```



E7. Deployment

In [14]:
```
Decision_Trees = DecisionTreeClassifier(criterion='gini', max_depth = 4,
                                        max_features='sqrt', min_samples_leaf=7,
                                        min_samples_split = 2, random_state=0)
Decision_Trees.fit(features, label)
```

Out[14]:
```
DecisionTreeClassifier(ccp_alpha=0.0, class_weight=None, criterion='gini',
                       max_depth=4, max_features='sqrt', max_leaf_nodes=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=7, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, presort='deprecated',
                       random_state=0, splitter='best')
```

Model Implementation

Six Steps of Model Implementation

1. Load in the new data

2. Encoding categorical data type

3. Setting features and label

4. Setting up Folds and Cross Validaion

5. Model prediction

```
In [ ]:
```

Coursework: Customer Analytics by KPIs Comparative Analysis

- University of Nottingham (UK), MSc Business Analytics
- Lecture: Data at Scale: Management, Processing, Visualization
- Year: 2019
- Language: PostgreSQL, Tableau

The Problem Denfiniton

A comparative analysis of the stores performance in terms of sales and profit relative to the size of the store. An comparative analysis of customer loyalty. Four store data collected over two years are given. The data consists of five SQL tables, with the table name as shown below.

- Customers (id, born, name)
- Products (code and details of product, department, category and sub category)
- Receipt lines (receipt id, product code, price, quantity)
- Receipts (receipt id, time, id, store code)
- Stores (informations about stores)

KPIs

KPI 1. Total sales vs Total sales in size

- The size of the store is determined by the number of product codes.
- Compare total sales based on store 0.
- Store 0 has the highest sales, and Store 3 has the highest sales considering the size of the store.

KPI 2. New customers

- Represents new customers on a monthly basis.

KPI 3. Active customers

- Indicates customers who visit more than three times a month.
- Few new customers, but more loyal stores can be found.
- You can find a store where sales can go up.

KPI 4. Monthly Sales

- Identify the changes in sales on a monthly basis.
- Find out when to introduce new marketing.

KPI 5. Top 3 departments

- Find the type of product that sells the most.
- Dairy > Grocery 2 > Fruits and vegetables

KPI 6. Top 3 category in dairy depart

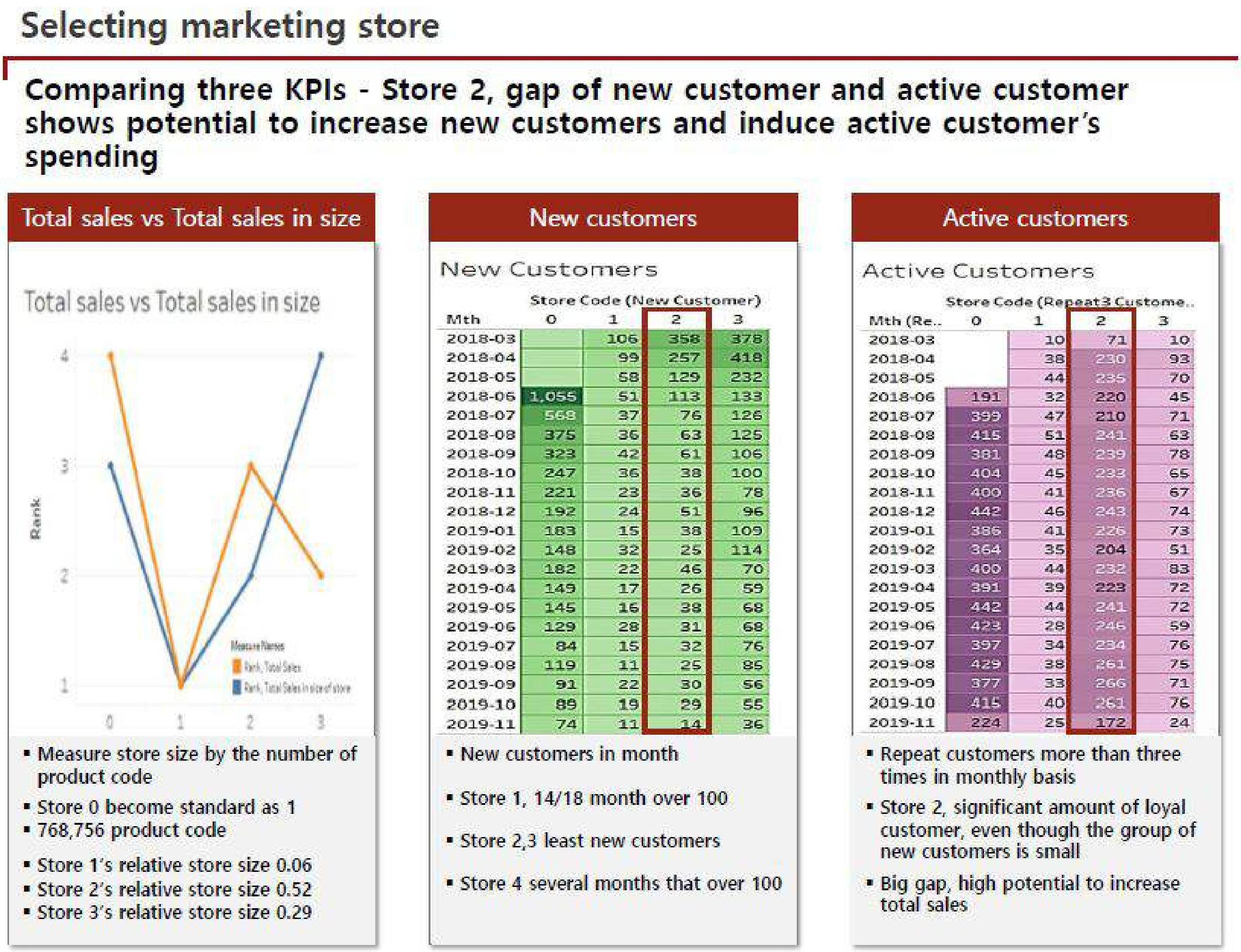
- Find a particular product in the most popular types of products.
- Milk in dairy products > yogurt and dessert > cheese

Report

https://github.com/Chan-Young/Coursework/blob/main/KPIs%20comparative%20analysis.pdf
(https://github.com/Chan-Young/Coursework/blob/main/KPIs%20comparative%20analysis.pdf)

Presentation ppt

https://github.com/Chan-Young/Coursework/blob/main/Presentation_SQL_coursework.pdf (https://github.com/Chan-Young/Coursework/blob/main/Presentation_SQL_coursework.pdf)



	Store Code (New Customer)			
Mth	0	1	2	3
2018-03		106	358	378
2018-04		99	257	418
2018-05		58	129	232
2018-06	1,055	51	113	133
2018-07	568	37	76	126
2018-08	375	36	63	125
2018-09	323	42	61	106
2018-10	247	36	38	100
2018-11	221	23	36	78
2018-12	192	24	51	96
2019-01	183	15	38	109
2019-02	148	32	25	114
2019-03	182	22	46	70
2019-04	149	17	26	59
2019-05	145	16	38	68
2019-06	129	28	31	68
2019-07	84	15	32	76
2019-08	119	11	25	85
2019-09	91	22	30	56
2019-10	89	19	29	55
2019-11	74	11	14	36

	Store Code (Repeat3 Custome..			
Mth (Re..	0	1	2	3
2018-03		10	71	10
2018-04		38	230	93
2018-05		44	235	70
2018-06	191	32	220	45
2018-07	399	47	210	71
2018-08	415	51	241	63
2018-09	381	48	239	78
2018-10	404	45	233	65
2018-11	400	41	236	67
2018-12	442	46	243	74
2019-01	386	41	226	73
2019-02	364	35	204	51
2019-03	400	44	232	83
2019-04	391	39	223	72
2019-05	442	44	241	72
2019-06	423	28	246	59
2019-07	397	34	234	76
2019-08	429	38	261	75
2019-09	377	33	266	71
2019-10	415	40	261	76
2019-11	224	25	172	24

Specifying marketing strategy

Comparing three KPIs – Store 2, marketing starts from March 2020 on milk category

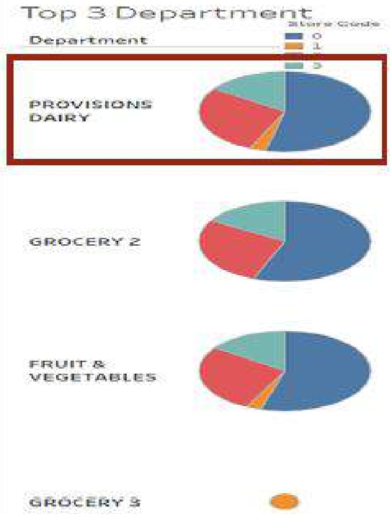
Monthly Sales

Monthly Sales

Month	Store Code 0	1	2	3
2018-04		165.1%	171.8%	166.4%
2018-05		10.8%	7.2%	-6.3%
2018-06		-17.3%	1.0%	-25.6%
2018-07	82.2%	17.9%	-6.0%	17.5%
2018-08	1.8%	1.6%	9.4%	-2.5%
2018-09	-2.2%	6.6%	-1.2%	4.6%
2018-10	-1.2%	-2.3%	-8.0%	8.3%
2018-11	0.9%	4.4%	5.1%	-5.3%
2018-12	6.1%	-0.8%	10.2%	0.5%
2019-01	-8.8%	-9.0%	-8.4%	9.3%
2019-02	-6.0%	-10.7%	-16.6%	-15.0%
2019-03	10.5%	17.3%	21.1%	13.6%
2019-04	-3.9%	6.9%	-3.2%	-0.9%
2019-05	15.2%	-3.2%	8.9%	-3.9%
2019-06	-1.5%	-15.2%	3.0%	-24.6%
2019-07	-11.2%	-4.9%	-7.7%	26.2%
2019-08	9.7%	7.7%	8.7%	-2.0%
2019-09	-8.7%	3.4%	-1.3%	-4.3%
2019-10	6.2%	11.1%	2.0%	1.2%
2019-11	-37.6%	-35.3%	-39.6%	-38.2%

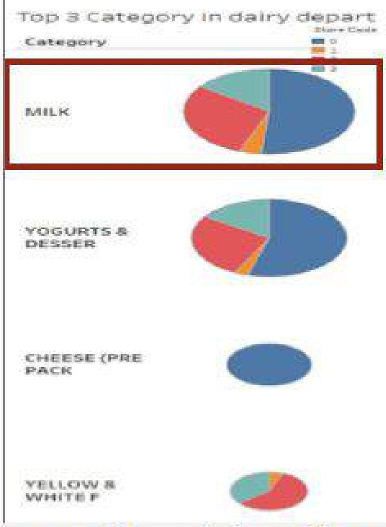
- Rate of monthly sales difference compare to previous month
- March to August shows almost continuous positive rate
- Last month, Nov 2019 was poor, start marketing on March 2020

Top 3 Department



- 'Provisions dairy', the largest sales in all department in four stores
- 61,174 in 'provisions dairy'
- 60,736 in 'grocery 2'
- 59,927 in 'fruit & vegetables'

Top 3 category in dairy depart



- Most sold categories in 'provision dairy department' in each store
- 'milk' category shows the most sold category in 'provision dairy'.
- store code 2 sold 20,503 quantities
- marketing should focus on 'milk category' in store code 2